

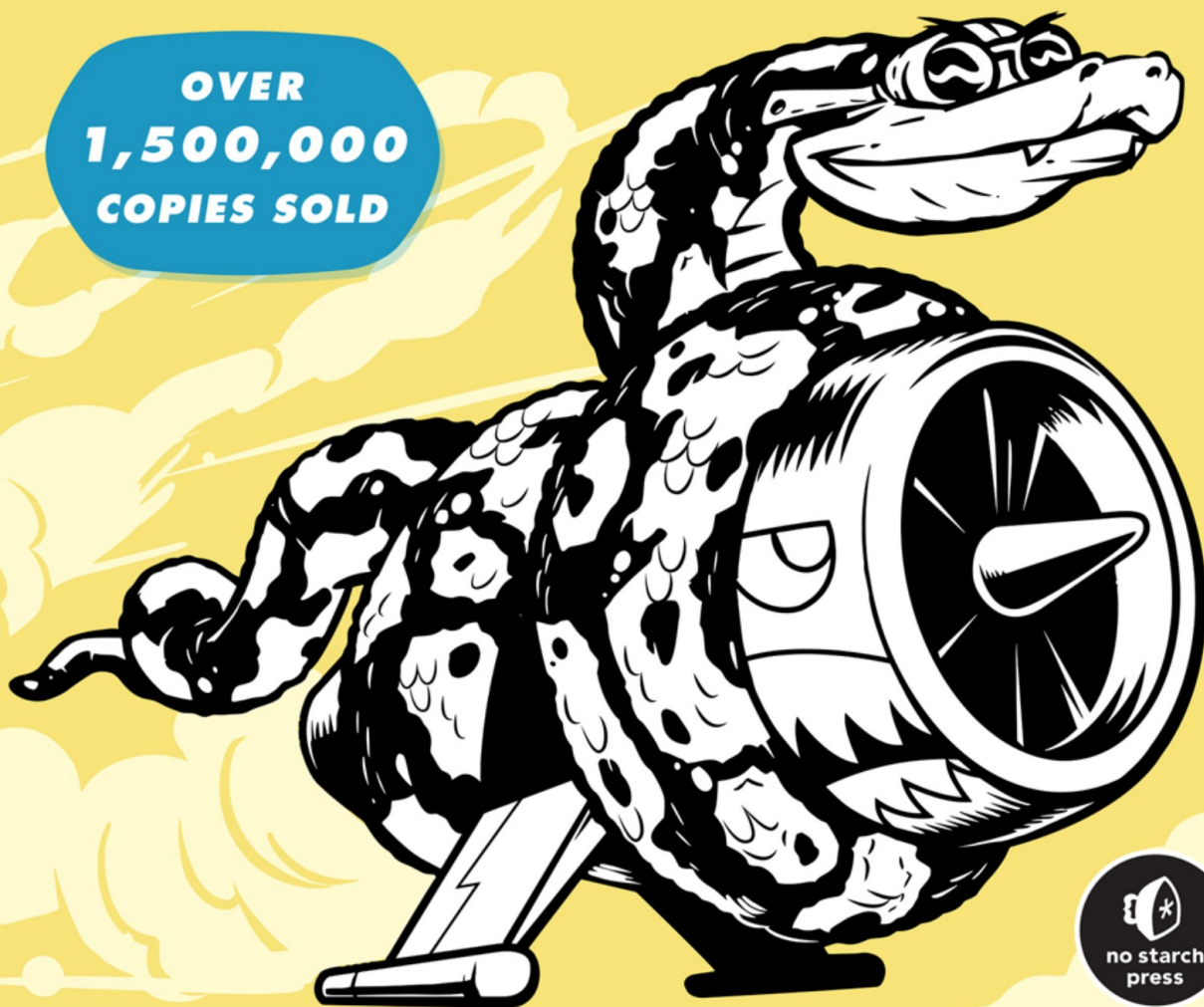
THIRD EDITION

PYTHON CRASH COURSE

**A HANDS-ON, PROJECT-BASED
INTRODUCTION TO PROGRAMMING**

ERIC MATTHES

**OVER
1,500,000
COPIES SOLD**



ELOGIOS PARA CURSO CRASH DE PYTHON

“Ha sido interesante ver a No Starch Press producir futuros clásicos que deberían estar junto a los libros de programación más tradicionales. Python Crash Course es uno de esos libros”.

—Greg Laden, Blogs de ciencia

"Aborda algunos proyectos bastante complejos y los presenta de una manera coherente, lógica y agradable que atrae al lector hacia el tema".

—Revista Círculo Completo

“Bien presentado con buenas explicaciones de los fragmentos de código. El libro trabaja contigo, un pequeño paso a la vez, creando un código más complejo y explicando lo que está sucediendo en todo momento”.

—Reseñas de hojear

“¡Aprender Python con el curso intensivo de Python fue una experiencia extremadamente positiva! Una gran elección si eres nuevo en Python”.

—Mikke se dedica a codificar

“Hace lo que dice y lo hace realmente bien. . . . Presenta una gran cantidad de ejercicios útiles, así como tres proyectos desafiantes y entretenidos”.

—RealPython.com

"Una introducción rápida pero completa a la programación con Python, Python Crash Course es otro excelente libro para agregar a tu biblioteca y ayudarte a dominar finalmente Python".

—TutorialEdge.net

“Una opción brillante para principiantes sin experiencia en codificación. Si está buscando una introducción sólida y sencilla a este lenguaje tan profundo, debo recomendar este libro”.

—WhatPixel.com

"Contiene literalmente todo lo que necesitas saber sobre Python y aún más".

—FireBearStudio.com

"Si bien el curso intensivo de Python utiliza Python para enseñarle a codificar, también enseña habilidades de programación limpias que se aplican a la mayoría de los demás lenguajes".

—Geek de los Grandes Lagos

PYTHONCRASH CURSO

3ª EDICIÓN

AH ys - En , Proyecto basado
Introducción a la programación

por Eric Matthes



**no starch
press**

San Francisco

CURSO CRASH DE PYTHON, 3ª EDICIÓN. Copyright © 2023 por Eric Matthes.

Reservados todos los derechos. Ninguna parte de este trabajo puede reproducirse o transmitirse de ninguna forma ni por ningún medio, electrónico o mecánico, incluidas fotocopias, grabaciones o cualquier sistema de almacenamiento o recuperación de información, sin el permiso previo por escrito del propietario de los derechos de autor y del editor.

Primera impresión

26 25 24 23 22 1 2 3 4 5

ISBN-13: 978-1-7185-0270-3 (impreso)

ISBN-13: 978-1-7185-0271-0 (libro electrónico)

Editorial: William Pollock

Editor jefe: Jill Franklin

Editor de producción: Jennifer Kepler

Editora de desarrollo: Eva Morrow

Ilustrador de portada: Josh Ellingson

Diseño de interiores: Estudios Octopod

Revisor técnico: Kenneth Love

Editor: Doug McNair

Compositor: Jeff Lytle, Happenstance Type-O-Rama

Corrector: Scout Festa

Para obtener información sobre distribución, ventas al por mayor, ventas corporativas o traducciones, comuníquese con No Starch Press, Inc. directamente en info@nostarch.com o:

Sin almidón Press, Inc.

245 Calle 8, San Francisco, CA 94103

teléfono: 1.415.863.9900

www.nostarch.com

La Biblioteca del Congreso ha catalogado la primera edición de la siguiente manera:

Matthes, Eric, 1972-

Curso intensivo de Python: una introducción práctica a la programación basada en proyectos / por Eric Matthes.

páginas cm

Incluye índice.

Resumen: "Una introducción a la programación en Python basada en proyectos, con ejercicios. Cubre conceptos generales de programación, fundamentos de Python y resolución de problemas. Incluye tres proyectos: cómo crear un videojuego simple, usar técnicas de visualización de datos para crear gráficos y tablas, y crear una aplicación web interactiva"--

Proporcionado por el editor.

ISBN 978-1-59327-603-4 - ISBN 1-59327-603-6

1. Python (Lenguaje de programación informática) I. Título.

QA76.73.P98M38 2015

005.13'3--dc23

2015018135

No Starch Press y el logotipo de No Starch Press son marcas comerciales registradas de No Starch Press, Inc. Otros nombres de productos y empresas mencionados aquí pueden ser marcas comerciales de sus respectivos propietarios. En lugar de utilizar un símbolo de marca registrada cada vez que aparece un nombre de marca registrada, utilizamos los nombres solo de manera editorial y para beneficio del propietario de la marca, sin intención de infringir la marca.

La información contenida en este libro se distribuye "tal cual", sin garantía. Si bien se han tomado todas las precauciones en la preparación de este trabajo, ni el autor ni No Starch Press, Inc. tendrán responsabilidad alguna ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o presuntamente causado directa o indirectamente por la información contenida en el mismo.

Para mi padre, que siempre se tomaba el tiempo para responder mis preguntas sobre programación, y para Ever, que recién comienza a hacerme sus preguntas.

Sobre el Autor

Eric Matthes fue profesor de matemáticas y ciencias en una escuela secundaria durante 25 años, e impartía clases de introducción a Python siempre que encontraba una manera de incluirlas en el plan de estudios. Eric ahora es escritor y programador a tiempo completo y participa en varios proyectos de código abierto. Sus proyectos tienen una amplia gama de objetivos, desde ayudar a predecir la actividad de deslizamientos de tierra en regiones montañosas hasta simplificar el proceso de implementación de proyectos Django. Cuando no está escribiendo o programando, le gusta escalar montañas y pasar tiempo con su familia.

Acerca del revisor técnico

Kenneth Love vive en el noroeste del Pacífico con su familia y sus gatos. Kenneth es un veterano programador de Python, colaborador de código abierto, profesor y conferencista.

CONTENIDOS BREVES

Prefacio a la Tercera Edición.	xxvii
Agradecimientos.	xxxi
Introducción	xxxiii
PARTE I: FUNDAMENTOS	1
Capítulo 1: Primeros pasos.	3
Capítulo 2: Variables y tipos de datos simples.	15
Capítulo 3: Presentación de listas.	33
Capítulo 4: Trabajar con listas.	49
Capítulo 5: Declaraciones if.	71
Capítulo 6: Diccionarios.	91
Capítulo 7: Entrada del usuario y bucles while.	113
Capítulo 8: Funciones.	129
Capítulo 9: Clases.	157
Capítulo 10: Archivos y excepciones.	183
Capítulo 11: Probando su código.	209
PARTE II: PROYECTOS	225
Capítulo 12: Un barco que dispara balas.	227
Capítulo 13: ¡Extraterrestres!	255
Capítulo 14: Puntuación.	277
Capítulo 15: Generación de datos.	301
Capítulo 16: Descarga de datos.....	329
Capítulo 17: Trabajar con API.	355

Capítulo 18: Primeros pasos con Django. 373

Capítulo 19: Cuentas de usuario. 403

Capítulo 20: Diseño e implementación de una aplicación. 433

Apéndice A: Instalación y solución de problemas. 463

Apéndice B: Editores de texto e IDE. 469

Apéndice C: Obtener ayuda. 477

Apéndice D: Uso de Git para el control de versiones. 483

Apéndice E: Solución de problemas de implementaciones. 493

Índice. 503

CONTENIDOS EN DETALLE

PREFACIO A LA TERCERA EDICIÓN	xxvii
EXPRESIONES DE GRATITUD	xxxi
INTRODUCCIÓN	xxxiii
¿Para quién es este libro?	xxxiii
¿Qué puedes esperar aprender?	xxxiii
Recursos en línea	xxxv
¿Por qué Python?	xxxvi
 PARTE I: FUNDAMENTOS	 1
1	
EMPEZANDO	3
Configurando su entorno de programación.	3
Versiones de Python.	4
Ejecución de fragmentos de código	4
Python Acerca del editor de	4
código VS Python en diferentes sistemas operativos.	5
Pitón en Windows.	5
Python en macOS.	7
Pitón en Linux.	8
Ejecución de un programa Hola Mundo.	9
Instalación de la extensión Python para VS	9
Code ejecutando hello_world.py.	10
Solución de problemas	10
Ejecutar programas Python desde una terminal.	11
En Windows..... 12	
En macOS y Linux 12	
Ejercicio 1-1: python.org.	13
Ejercicio 1-2: Errores tipográficos de Hola mundo.	13
Ejercicio 1-3: Habilidades Infinitas.	13
Resumen	13
 2	
VARIABLES Y TIPOS DE DATOS SIMPLES	15
Qué sucede realmente cuando ejecutas hello_world.py.	15
Variables.	dieciséis
Nombrar y usar variables.	17
Evitar errores de nombre al utilizar variables.	17
Las variables son etiquetas.	18
Ejercicio 2-1: Mensaje sencillo.	19
Ejercicio 2-2: Mensajes sencillos.	19

Cuerdas. . . 19

 Cambiar mayúsculas y minúsculas en una cadena con métodos. . . 20

 Uso de variables en cadenas. . . 20

 Agregar espacios en blanco a cadenas con tabulaciones o nuevas líneas. 21

 Eliminación de espacios en blanco. 22

 Eliminación de prefijos. . . 23

 Evitar errores de sintaxis con cadenas. 24

 Ejercicio 2-3: Mensaje personal. 25

 Ejercicio 2-4: Nombre de casos. 25

 Ejercicio 2-5: Cita célebre. 25

 Ejercicio 2-6: Cita célebre 2. 25

 Ejercicio 2-7: Eliminación de nombres. 25

 Ejercicio 2-8: Extensiones de archivos. 25

Números. . . 26

 Enteros. . . 26

 Flotadores. . . 26

 Enteros y flotantes. . . 27

 Guiones bajos en números. . . 28

 Asignación múltiple. . . 28

 Constantes. . . 28

 Ejercicio 2-9: Número ocho. 29

 Ejercicio 2-10: Número favorito. 29

Comentarios. 29

 ¿Cómo se escriben comentarios? . . . 29

 ¿Qué tipo de comentarios debería escribir? . . . 29

 Ejercicio 2-11: Agregar comentarios. 30

El Zen de Python. . . 30

 Ejercicio 2-12: Zen de Python. 31

Resumen . . . 32

3

PRESENTANDO LISTAS

33

¿Qué es una lista? 33

 Accediendo a elementos de una lista. . . 34

 Las posiciones del índice comienzan en 0, no en 1. . . 34

 Usar valores individuales de una lista. . . 35

 Ejercicio 3-1: Nombres. 36

 Ejercicio 3-2: Saludos. 36

 Ejercicio 3-3: Su propia lista. 36

Modificar, agregar y eliminar elementos. . . 36

 Modificar elementos en una lista. . . 36

 Agregar elementos a una lista. . . 37

 Eliminar elementos de una lista. . . 38

 Ejercicio 3-4: Lista de invitados. 41

 Ejercicio 3-5: cambiar la lista de invitados. 42

 Ejercicio 3-6: Más invitados. 42

 Ejercicio 3-7: Lista de invitados reducida. 42

Organizar una lista. . . 42

 Ordenar una lista de forma permanente con el método sort(). . . 43

 Ordenar una lista temporalmente con la función sorted(). . . 43

 Imprimir una lista en orden inverso. 44

Encontrar la longitud de una lista.	44
Ejercicio 3-8: Ver el mundo.	45
Ejercicio 3-9: Invitados a cenar.	45
Ejercicio 3-10: cada función.	45
Evitar errores de índice al trabajar con listas.	46
Ejercicio 3-11: Error intencional.	47
Resumen	47
4	
TRABAJAR CON LISTAS	49
Recorrer una lista completa.	49
Una mirada más cercana al bucle.	50
Hacer más trabajo dentro de un bucle for.	51
Hacer algo después de un bucle for.	52
Evitar errores de sangría.	53
Olvidarse de sangrar.	53
Olvidar aplicar sangría a líneas adicionales.	54
Sangrar innecesariamente.	54
Sangrar innecesariamente después del bucle.	55
Olvidando el Colón.	55
Ejercicio 4-1: Pizzas.	56
Ejercicio 4-2: Animales.	56
Hacer listas numéricas.	56
Usando la función range().	57
Usando range() para hacer una lista de números.	58
Estadísticas simples con una lista de números.	59
Lista de comprensiones.	59
Ejercicio 4-3: Contar hasta veinte.	60
Ejercicio 4-4: Un millón.	60
Ejercicio 4-5: Sumar un millón.	60
Ejercicio 4-6: Números impares.	60
Ejercicio 4-7: Tres.	60
Ejercicio 4-8: Cubos.	60
Ejercicio 4-9: Comprensión de cubos.	60
Trabajar con parte de una lista.	61
Cortar una lista.	61
Recorriendo un corte.	62
Copiar una lista.	63
Ejercicio 4-10: Rebanadas.	sesenta y cinco
Ejercicio 4-11: Mis pizzas, tus pizzas.	sesenta y cinco
Ejercicio 4-12: Más bucles.	sesenta y cinco
Tuplas.	sesenta y cinco
Definiendo una tupla.	sesenta y cinco
Recorriendo todos los valores de una tupla.	66
Escribir sobre una tupla.	67
Ejercicio 4-13: Buffet.	67
Diseñar su código.	68
La guía de estilo.	68
Sangría.	68
Longitud de la línea	69
Líneas en blanco	69

Otras pautas de estilo . . . 69	
Ejercicio 4-14: PEP 8	70
Ejercicio 4-15: Revisión de código	70
Resumen	70
 5	
DECLARACIONES SI	71
Un ejemplo sencillo. . . 72	
Pruebas condicionales. 72	
Comprobando la igualdad. . . 72	
Ignorar el caso al verificar la igualdad. . . 73	
Comprobando la desigualdad. . . 74	
Comparaciones numéricas. . . 74	
Comprobación de múltiples condiciones. . . 75	
Comprobar si un valor está en una lista. . . 76	
Comprobar si un valor no está en una lista..... 76	
Expresiones booleanas. . . 77	
Ejercicio 5-1: Pruebas condicionales.	77
Ejercicio 5-2: Más pruebas condicionales.	78
Si declaraciones..... 78	
Declaraciones if simples. . . 78	
Declaraciones if-else. 79	
La cadena if-elif-else. . . 80	
Usando múltiples bloques elif. . . 81	
Omitiendo el bloque else. . . 82	
Prueba de múltiples condiciones. . . 82	
Ejercicio 5-3: Colores alienígenas n.º 1.	84
Ejercicio 5-4: Colores alienígenas n.º 2.	84
Ejercicio 5-5: Colores alienígenas n.º 3.	84
Ejercicio 5-6: Etapas de la vida.	84
Ejercicio 5-7: Fruta favorita.	85
Uso de sentencias if con listas. . . 85	
Comprobación de artículos especiales. . . 85	
Comprobar que una lista no esté vacía..... 86	
Usando múltiples listas. . . 87	
Ejercicio 5-8: Hola administrador.	88
Ejercicio 5-9: Sin usuarios.	88
Ejercicio 5-10: Comprobación de nombres de usuario.	88
Ejercicio 5-11: Números ordinales.	88
Diseñar sus declaraciones if.	89
Ejercicio 5-12: Aplicar estilo a declaraciones if.	89
Ejercicio 5-13: Tus ideas.	89
Resumen	89
 6	
DICCIONARIOS	91
Un diccionario sencillo.	92
Trabajar con diccionarios.	92
Acceso a valores en un diccionario.	92
Agregar nuevos pares clave-valor.	93
Comenzando con un diccionario vacío.	94

Modificar valores en un diccionario.	94
Eliminación de pares clave-valor.	96
Un diccionario de objetos similares.	96
Usando get() para acceder a valores:	97
Ejercicio 6-1: Persona.	98
Ejercicio 6-2: Números favoritos.	98
Ejercicio 6-3: Glosario.	99
Recorriendo un diccionario.	99
Recorriendo todos los pares clave-valor.	99
Recorriendo todas las claves de un diccionario.....	101
Recorrer las claves de un diccionario en un orden particular.	102
Recorriendo todos los valores en un diccionario.	103
Ejercicio 6-4: Glosario 2.	104
Ejercicio 6-5: Ríos.	105
Ejercicio 6-6: sondeo.	105
Anidación.	105
Una lista de diccionarios.	105
Una lista en un diccionario.	108
Un diccionario en un diccionario.	110
Ejercicio 6-7: Personas.	111
Ejercicio 6-8: Mascotas.	111
Ejercicio 6-9: Lugares favoritos.	111
Ejercicio 6-10: Números favoritos.	111
Ejercicio 6-11: Ciudades.	111
Ejercicio 6-12: Extensiones.	111
Resumen	111
7	
ENTRADA DEL USUARIO Y BUCLES WHILE	113
Cómo funciona la función input().	114
Escribir indicaciones claras.	114
Usando int() para aceptar entradas numéricas.	115
El operador de módulo.	116
Ejercicio 7-1: Coche de alquiler.	117
Ejercicio 7-2: Asientos en un restaurante.	117
Ejercicio 7-3: Múltiplos de diez.	117
Presentamos los bucles while.	117
El bucle while en acción.	117
Permitir que el usuario elija cuándo salir.	118
Usando una bandera.	120
Usando break para salir de un bucle.	121
Usando continuar en un bucle.	122
Evitando bucles infinitos.	122
Ejercicio 7-4: Ingredientes para pizza.	123
Ejercicio 7-5: Entradas al cine.	123
Ejercicio 7-6: Tres salidas.	123
Ejercicio 7-7: Infinito.	123
Usando un bucle while con listas y diccionarios.	124
Mover elementos de una lista a otra.	124
Eliminación de todas las instancias de valores específicos de una lista.	125
Llenar un diccionario con entradas del usuario.	125
Ejercicio 7-8: Deli	127

Ejercicio 7-9: Sin pastrami.	127
Ejercicio 7-10: Vacaciones de ensueño.	127
Resumen	127
8	
FUNCIONES	129
Definición de una función.	130
Pasar información a una función.	130
Argumentos y parámetros.	131
Ejercicio 8-1: Mensaje.	131
Ejercicio 8-2: Libro favorito.	131
Pasar argumentos.	131
Argumentos posicionales.	132
Argumentos de palabras clave.	133
Valores predeterminados	134
Llamadas a funciones equivalentes.	135
Evitar errores de argumentación.	136
Ejercicio 8-3: Camiseta.	136
Ejercicio 8-4: Camisas grandes.	137
Ejercicio 8-5: Ciudades.	137
Valores de retorno.	137
Devolver un valor simple.	137
Hacer que un argumento sea opcional.	138
Devolver un diccionario.	139
Usando una función con un bucle while.	140
Ejercicio 8-6: Nombres de ciudades.	141
Ejercicio 8-7: Álbum.	142
Ejercicio 8-8: Álbumes de usuario.	142
Pasando una lista.	142
Modificar una lista en una función.	143
Evitar que una función modifique una lista.	145
Ejercicio 8-9: Mensajes.	146
Ejercicio 8-10: Envío de mensajes.	146
Ejercicio 8-11: Mensajes archivados.	146
Pasar un número arbitrario de argumentos.	146
Mezclando argumentos posicionales y arbitrarios.	147
Uso de argumentos de palabras clave arbitrarias.	148
Ejercicio 8-12: Sándwiches.	149
Ejercicio 8-13: Perfil de usuario.	149
Ejercicio 8-14: Coches.	149
Almacenamiento de funciones en módulos.	149
Importación de un módulo completo.	150
Importación de funciones específicas.	151
Usar as para darle un alias a una función.	151
Usar as para darle un alias a un módulo.	152
Importación de todas las funciones de un módulo.	152
Funciones de estilo.	153
Ejercicio 8-15: Modelos de impresión.	154
Ejercicio 8-16: Importaciones.	154
Ejercicio 8-17: Funciones de estilo.	154
Resumen	154

9

CLASES

157

Creación y uso de una clase.	158
Creando la clase de perro.	158
El método <code>__init__()</code>	159
Crear una instancia a partir de una clase.	159
Ejercicio 9-1: Restaurante.	162
Ejercicio 9-2: Tres restaurantes.	162
Ejercicio 9-3: Usuarios.	162
Trabajar con clases e instancias.	162
La clase de coche.	162
Establecer un valor predeterminado para un atributo.	163
Modificación de valores de atributos.	164
Ejercicio 9-4: Número atendido.	166
Ejercicio 9-5: intentos de inicio de sesión.	167
Herencia	167
El método <code>__init__()</code> para una clase secundaria.	167
Definición de atributos y métodos para la clase secundaria.	169
Anulación de métodos de la clase principal.	170
Instancias como atributos.	170
Modelado de objetos del mundo real.	172
Ejercicio 9-6: Puesto de helados.	173
Ejercicio 9-7: Administrador.	173
Ejercicio 9-8: Privilegios.	173
Ejercicio 9-9: Actualización de la batería.	173
Importación de clases.	173
Importando una sola clase.	174
Almacenamiento de varias clases en un módulo.	175
Importación de varias clases desde un módulo.	176
Importación de un módulo completo.	176
Importar todas las clases desde un módulo.	177
Importar un módulo a un módulo.	177
Usando alias.	178
Encontrar su propio flujo de trabajo.	179
Ejercicio 9-10: Restaurante importado.	179
Ejercicio 9-11: Administrador importado.	179
Ejercicio 9-12: Múltiples módulos.	179
La biblioteca estándar de Python.	179
Ejercicio 9-13: Datos.	180
Ejercicio 9-14: Lotería.	180
Ejercicio 9-15: Análisis de lotería.	180
Ejercicio 9-16: Módulo Python de la semana.	180
Clases de estilismo.	181
Resumen	181

10

ARCHIVOS Y EXCEPCIONES

183

Lectura de un archivo.	184
Leer el contenido de un archivo.	184
Rutas de archivo relativas y absolutas.	186
Accediendo a las líneas de un archivo.	186

Trabajar con el contenido de un archivo. . .	187
Archivos grandes: un millón de dígitos. . .	188
¿Tu cumpleaños está contenido en Pi? . .	189
Ejercicio 10-1: Aprender Python.	189
Ejercicio 10-2: Aprendizaje C	190
Ejercicio 10-3: Código más simple.	190
Escribir en un archivo. . . 190	
Escribir una sola línea. . . 190	
Escribir varias líneas. . . 191	
Ejercicio 10-4: Invitado.	192
Ejercicio 10-5: Libro de visitas.	192
Excepciones. . . 192	
Manejo de la excepción ZeroDivisionError. . .	192
Usando bloques try-except. . . 193	
Uso de excepciones para evitar fallas. . . 193	
El otro bloque. . . 194	
Manejo de la excepción FileNotFoundError. . .	195
Analizando texto. . . 196	
Trabajar con varios archivos. . . 197	
Fallando en silencio. . . 198	
Decidir qué errores informar. . . 199	
Ejercicio 10-6: Suma.	200
Ejercicio 10-7: Calculadora de sumas.	200
Ejercicio 10-8: Perros y gatos.	200
Ejercicio 10-9: Perros y gatos silenciosos.	200
Ejercicio 10-10: Palabras comunes.	200
Almacenamiento de datos . . 201	
Usando json.dumps() y json.loads(). . . 201	
Guardar y leer datos generados por el usuario. . .	202
Refactorización. . . 204	
Ejercicio 10-11: Número favorito.	206
Ejercicio 10-12: Número favorito recordado.	206
Ejercicio 10-13: Diccionario del usuario.	206
Ejercicio 10-14: verificar usuario.	206
Resumen . . 207	
 11	
PROBANDO SU CÓDIGO	209
Instalando pytest con pip. . 210	
Actualizando pip. . . 210	
Instalando pytest. 211	
Probar una función. . 211	
Pruebas unitarias y casos de prueba. 212.	
Una prueba de aprobación. . . 212	
Ejecutando una prueba. . 213	
Una prueba fallida. . 214	
Responder a una prueba fallida. . 215	
Agregar nuevas pruebas. . 216	
Ejercicio 11-1: Ciudad, País.	217
Ejercicio 11-2: Población.	217

Probar una clase. . : 217 ·

 Una variedad de afirmaciones: : 217 ·

 Una clase para probar. : 218 ·

 Probando la clase AnonymousSurvey. : 220 ·

 Usando accesorios. : 221 ·

 Ejercicio 11-3: Empleado. 223

Resumen . . 223 ·

PARTE II: PROYECTOS

225

12

UN BARCO QUE DISPARA BALAS 227

Planificando su proyecto. 228

Instalando Pygame. 228

Iniciando el proyecto del juego. 229

 Crear una ventana de Pygame y responder a la entrada del usuario. 229

 Controlar la velocidad de fotogramas. 230

 Configuración del color de fondo. 231

 Creando una clase de configuración. 232

Agregar la imagen del barco. 233

 Creando la clase de barco. 234

 Llevando el barco a la pantalla. 235

Refactorización: los métodos _check_events() y _update_screen()..... 237

 El método _check_events(). : 237 ·

 El método _update_screen(). : 237 ·

 Ejercicio 12-1: Cielo azul. 238

 Ejercicio 12-2: Personaje del juego. 238

Pilotando el barco. : 238 ·

 Responder a una pulsación de tecla. : 238 ·

 Permitiendo el movimiento continuo. . . 239 ·

 Moviéndose tanto hacia la izquierda como hacia la derecha. : 241 ·

 Ajuste de la velocidad del barco. : 242 ·

 Limitar el alcance del barco. : 243 ·

 Refactorizando _check_events(). : 244 ·

 Presionando Q para salir. 244 ·

 Ejecutando el juego en modo de pantalla completa: : 245 ·

Un resumen rápido.: 245 ·

 alien_invasion.py. : 246 ·

 configuración.py.: 246

 barco.py. . . 246 ·

 Ejercicio 12-3: Documentación de Pygame. 246

 Ejercicio 12-4: Cohete. 246

 Ejercicio 12-5: Claves. 246

Disparando balas. : 247 ·

 Agregar la configuración de viñetas: : 247 ·

 Creando la clase Bullet. : 247 ·

 Almacenamiento de viñetas en un grupo.: : 248 ·

 Disparando balas: : 249 ·

 Eliminación de viñetas antiguas.: 250 ·

Limitar el número de balas. . . 251	
Creando el método _update_bullets(). . 252	
Ejercicio 12-6: Tirador lateral.	253
Resumen . . . 253	
 13	
¡EXTRANJEROS!	255
Revisando el Proyecto.	256
Creando el primer alienígena.	256
Creando la clase alienígena.	257
Creando una instancia del alienígena.	257
Construyendo la flota alienígena.	259
Creando una fila de alienígenas.: 259	
Refactorizando _create_fleet(). . . 260	
Agregar filas. . . 261	
Ejercicio 13-1: Estrellas.	263
Ejercicio 13-2: Mejores estrellas.	263
Hacer que la flota se mueva.	263
Moviendo a los extraterrestres hacia la derecha. : 263	
Creación de configuraciones para la dirección de la flota.: 264	
Comprobar si un extraterrestre ha llegado al borde. . 265	
"Dejar caer la flota y cambiar de dirección". . 265	
Ejercicio 13-3: Gotas de lluvia.	266
Ejercicio 13-4: Lluvia constante.	266
Disparar a extraterrestres. : 266	
Detección de colisiones de balas. : 267	
"Hacer balas más grandes para realizar pruebas".: : 268	
Repoblación de la Flota. . 268	
Acelerando las balas. . . 269	
Refactorizando _update_bullets(). . . 269	
Ejercicio 13-5: Tirador lateral, parte 2.	270
Terminando el juego. . . 270	
Detección de colisiones de naves alienígenas. : 270	
Respondiendo a las colisiones de naves alienígenas. : 271	
Extraterrestres que llegan al final de la pantalla. . 273.	
¡Juego terminado! . . 274	
Identificar cuándo deben ejecutarse partes del juego. . . 275	
Ejercicio 13-6: Fin del juego.	275
Resumen	275
 14	
PUNTUACIÓN	277
Agregar el botón Reproducir.	278
Creando una clase de botón. . . 278	
Dibujando el botón a la pantalla. . 279	
Comenzando el juego.: 281	
Restablecer el juego. . 281	
Desactivar el botón Reproducir. . 282	
Ocultar el cursor del mouse. . 282	
Ejercicio 14-1: Presione P para reproducir.	283
Ejercicio 14-2: Práctica de tiro.	283

Subir de nivel . . . 283

 Modificación de la configuración de velocidad.: 283

 Restablecer la velocidad.:285

 Ejercicio 14-3: Práctica de tiro desafiante. 286

 Ejercicio 14-4: Niveles de dificultad. 286

Puntuación . . .286

 Visualización de la partitura..... 286

 Hacer un marcador. . 287

 Actualización de la puntuación a medida que los extraterrestres son derribados: .: 289

 Restablecer la puntuación: : 289

 Asegurándose de anotar todos los hits. .:290

 Valores de puntos crecientes. .:290

 Redondeando la puntuación: :291

 Puntajes altos..... 292

 Mostrando el nivel. . 294

 Visualización del número de barcos. . . 296

 Ejercicio 14-5: Puntuación más alta de todos los tiempos. 299

 Ejercicio 14-6: Refactorización. 299

 Ejercicio 14-7: Ampliando el juego. 299

 Ejercicio 14-8: Tirador lateral, versión final. 299

Resumen . . 299

15

GENERANDO DATOS

301

Instalando Matplotlib. . . 302

Trazar un gráfico lineal simple. . 302

 Cambiar el tipo de etiqueta y el grosor de la línea. . . 303

 Corrección de la trama.: 305

 Uso de estilos integrados.: 306

 Trazar y diseñar puntos individuales con scatter(). . 306

 Trazar una serie de puntos con scatter(). . 308

 Calcular datos automáticamente. . 308

 Personalización de etiquetas de ticks.: 309

 Definición de colores personalizados.: 310

 Usando un mapa de colores. : 310

 Guardar sus trazados automáticamente. .: 311

 Ejercicio 15-1: Cubos. 311

 Ejercicio 15-2: Cubos de colores. 311

Paseos aleatorios. 312

 Creando la clase RandomWalk. . . 312

 Elegir direcciones. . . 312

 Trazando el paseo aleatorio. . . 313

 Generando múltiples paseos aleatorios. . . 314

 Diseñando el paseo. .: 315

 Ejercicio 15-3: Movimiento molecular. 319

 Ejercicio 15-4: Paseos aleatorios modificados. 319

 Ejercicio 15-5: Refactorización. 319

Tirar dados con Plotly. 319

 Instalación de Plotly. 320

 Creando la clase de troquel. 320

 Tirando el dado. 320

 Analizando los resultados. 321

Hacer un histograma.....	322
Personalizando la trama. . .	323
Lanzar dos dados. . .	324
Más personalizaciones. . .	325
Dados rodantes de diferentes tamaños. . .	326
Ahorro de figuras. . .	327
Ejercicio 15-6: dos D8. . .	328
Ejercicio 15-7: Tres dados. . .	328
Ejercicio 15-8: Multiplicación. . .	328
Ejercicio 15-9: Comprensiones de dados. . .	328
Ejercicio 15-10: practicando con ambas bibliotecas. . .	328
Resumen . . .	328
decimals	
DESCARGAR DATOS	329
El formato de archivo CSV. . .	330
Analizando los encabezados del archivo CSV. . .	330
Impresión de los encabezados y sus posiciones. . .	331
Extracción y lectura de datos. . .	332
Trazar datos en un gráfico de temperatura. . .	332
El módulo de fecha y hora.	333
Trazar fechas.....	334
Trazar un período de tiempo más largo: .	336
Trazar una segunda serie de datos. . .	336
Sombrear un área en el gráfico. . .	337
Comprobación de errores. . .	338
Descarga de sus propios datos. . .	341
Ejercicio 16-1: Lluvias de Sitka. . .	342
Ejercicio 16-2: Comparación entre Sitka y el Valle de la Muerte. . .	342
Ejercicio 16-3: San Francisco. . .	342
Ejercicio 16-4: Índices automáticos. . .	342
Ejercicio 16-5: Explorar. . .	342
Mapeo de conjuntos de datos globales: formato GeoJSON. . .	342
Descarga de datos sobre terremotos. . .	343
Examinando datos GeoJSON. . .	343
Hacer una lista de todos los terremotos. . .	345
Extrayendo Magnitudes. . .	346
Extracción de datos de ubicación: .	346
Construyendo un mapa mundial: .	347
Representando Magnitudes. . .	348
Personalización de los colores de los marcadores. . .	349
Otras escalas de colores.....	350
Agregar texto flotante. . .	350
Ejercicio 16-6: Refactorización. . .	352
Ejercicio 16-7: Título automatizado. . .	352
Ejercicio 16-8: Terremotos recientes. . .	352
Ejercicio 16-9: Incendios mundiales. . .	352
Resumen . . .	352

17

TRABAJANDO CON APIS 355

Usando una	355
API Git y GitHub.	356
Solicitar datos mediante una llamada API.	356
Instalar solicitudes.	357
Procesando una respuesta API.	357
Trabajar con el diccionario de respuestas.	358
Resumiendo los principales repositorios.	361
Monitoreo de límites de tasa API.	362
Visualización de repositorios utilizando Plotly.	362
Aplicar estilo al gráfico.	364
Agregar información sobre herramientas personalizadas.	365
Agregar enlaces en los que se puede hacer clic.	366
Personalización de los colores de los marcadores.	367
Más información sobre Plotly y la API de GitHub	368
La API de noticias para hackers	368
Ejercicio 17-1: Otros idiomas.	371
Ejercicio 17-2: Discusiones activas.	371
Ejercicio 17-3: Prueba de python_repos.py.	372
Ejercicio 17-4: Exploración adicional.	372
Resumen	372

18

COMENZANDO CON DJANGO 373

Configuración de un proyecto.	374
Escribir una especificación.	374
Creando un entorno virtual.	374
Activando el Entorno Virtual.	375
Instalando Django.	375
Creando un proyecto en Django.	376
Creando la Base de Datos.	376
Visualizando el Proyecto.	377
Ejercicio 18-1: Nuevos proyectos.	378
Iniciar una aplicación.	379
Definición de modelos.	379
Activando Modelos.	380
El sitio de administración de Django.	381
Definición del modelo de entrada.	384
Migrando el modelo de entrada.	385
Registro de entrada en el sitio de administración.	385
El caparazón de Django.	386
Ejercicio 18-2: Entradas breves.	387
Ejercicio 18-3: La API de Django.	388
Ejercicio 18-4: Pizzería.	388
Creación de páginas: la página de inicio del registro de aprendizaje.	388
Mapeo de una URL.	388
Escribir una vista.	390
Escribir una plantilla.	390

Ejercicio 18-5: Planificador de comidas.	392
Ejercicio 18-6: Página de inicio de pizzería.	392
Construyendo páginas adicionales.	392
Herencia de plantilla.	392
La página de temas.	394
Páginas de temas individuales:	397
Ejercicio 18-7: Documentación de plantilla.	400
Ejercicio 18-8: Páginas de pizzería.	400
Resumen	400

19

CUENTAS DE USUARIO 403

Permitir a los usuarios ingresar datos:	404
Agregar nuevos temas.	404
Agregar nuevas entradas.	408
Edición de entradas.	412
Ejercicio 19-1: Blog.	415
Configuración de cuentas de usuario.	415
La App de cuentas.	415
La página de inicio de sesión:	416
Saliendo de tu cuenta:	419
La página de registro.	420
Ejercicio 19-2: Cuentas de blog.	423
Permitir que los usuarios sean propietarios de sus datos:	423
Restringir el acceso con @login_required.	423
Conexión de datos a determinados usuarios:	425
Restringir el acceso a los temas a los usuarios adecuados:	427
Proteger los temas de un usuario:	428
Protegiendo la página edit_entry.	429
Asociación de nuevos temas con el usuario actual.	429
Ejercicio 19-3: Refactorización.	430
Ejercicio 19-4: Protección de nueva_entrada.	430
Ejercicio 19-5: Blog protegido.	430
Resumen	430

20

DISEÑAR E IMPLEMENTAR UNA APLICACIÓN 433

Registro de aprendizaje de estilo:	434
La aplicación django-bootstrap5.	434
Uso de Bootstrap para diseñar el registro de aprendizaje:	434
Modificando base.html.	435
Diseñar la página de inicio usando un Jumbotron.	440
Aplicar estilo a la página de inicio de sesión:	441
Aplicar estilo a la página de temas.	442
Aplicar estilo a las entradas en la página del tema.	443
Ejercicio 20-1: Otras formas.	445
Ejercicio 20-2: Blog con estilo.	445
Implementación del registro de aprendizaje.	445
Crear una cuenta en Platform.sh.	445
Instalación de la CLI Platform.sh.	446
Instalando la plataformashconfig.	446

Creando un archivo de requisitos.txt.	446
Requisitos de implementación adicionales.	447
Agregar archivos de configuración.	447
Modificando settings.py para Platform.sh.	451
Usando Git para rastrear los archivos del proyecto.	451
Creando un proyecto en Platform.sh.	453
Empujando a Platform.sh.	455
Visualización del proyecto en vivo.	456
Refinando la implementación de Platform.sh.	456
Creación de páginas de error personalizadas.	459
Desarrollo en curso	460
Eliminar un proyecto en Platform.sh.	461
Ejercicio 20-3: Blog en vivo.	461
Ejercicio 20-4: Registro de aprendizaje ampliado.	462
Resumen	462

A

INSTALACIÓN Y SOLUCIÓN DE PROBLEMAS Python en	463
Windows.	463
Usando py en lugar de python.	463
Volviendo a ejecutar el instalador.	464
Python en macOS.	464
Instalar accidentalmente la versión de Python de Apple.	464
Python 2 en versiones anteriores de macOS.	465
Pitón en Linux.	465
Usando la instalación predeterminada de Python.	465
Instalación de la última versión de Python.	465
Comprobar qué versión de Python estás utilizando.	466
Palabras clave de Python y funciones integradas.	466
Palabras clave de Python.	466
Funciones integradas de Python.	467

B

EDITORES DE TEXTO E IDES que	469
funcionan de manera eficiente con VS Code:	470
Configurando el código VS.	470
Atajos de código VS.	473
Otros editores de texto e IDE.	474
INACTIVO 474	
Geany.	474
Texto sublime	474
Emacs y Vim.	475
PyCharm.	475
Cuadernos Jupyter.	475

C

OBTENER AYUDA	477
Primeros pasos.....	477
Pruébalo otra vez :	478
Tomar un descanso	478
Consulte los recursos de este libro.	478

Buscando en línea.	479
Desbordamiento de pila	479
La documentación oficial de Python.	479
Documentación Oficial de la	480
Biblioteca. r/learnpython:	480
Publicaciones de blog.	480
Discordia.	480
Flojo	481
 D	
USANDO GIT PARA EL CONTROL DE VERSIONES	483
Instalando Git.	484
Configurando Git.	484
Elaboración de un proyecto.....	484
Ignorar archivos.:484	
Inicializando un repositorio. . 485	
Comprobando el estado. . : 485	
Agregar archivos al repositorio. . . 486	
Hacer un compromiso. . : 486	
Comprobando el registro. . : 487	
El segundo compromiso.487	
Abandonar los cambios. . . 488	
Consultando confirmaciones anteriores. . :489	
Eliminando el repositorio. . . 491	
 mi	
SOLUCIÓN DE PROBLEMAS DE IMPLEMENTACIONES	493
Comprensión de las implementaciones.	494
Solución de problemas básicos.	494
Siga las sugerencias en pantalla.	495
Lea la salida del registro.	496
Solución de problemas específicos del sistema operativo.	497
Implementación desde Windows.	497
Implementación desde macOS.	499
Implementación desde Linux.	499
Otros enfoques de implementación.	500
 ÍNDICE	503

PREFACIO A LA TERCERA EDICIÓN

La respuesta a la primera y segunda edición del Python Crash Course ha sido abrumadoramente positiva. Se han impreso más de un millón de ejemplares, incluidas traducciones a más de 10 idiomas. He recibido cartas y correos electrónicos de lectores de tan sólo 10 años, así como de jubilados que quieren aprender a programar en su tiempo libre. Python Crash Course se utiliza en escuelas intermedias y secundarias, y también en clases universitarias. Los estudiantes a los que se les asignan libros de texto más avanzados utilizan Python Crash Course como texto complementario para sus clases y lo consideran un complemento valioso. La gente lo está utilizando para mejorar sus habilidades en el trabajo, cambiar de carrera y comenzar a trabajar en sus propios proyectos paralelos. En resumen, la gente está utilizando el libro para toda la gama de propósitos que esperaba, y mucho más.

Ha sido un placer tener la oportunidad de escribir una tercera edición del curso intensivo de Python . Aunque Python es un lenguaje maduro, continúa evolucionando como lo hacen todos los lenguajes. Mi objetivo principal al revisar el libro es mantenerlo como un curso introductorio de Python bien diseñado. Al leer este libro, aprenderá todo lo que necesita para comenzar a trabajar en sus propios proyectos y también construirá una base sólida para todo su aprendizaje futuro. Actualicé algunas secciones para reflejar formas más nuevas y simples de hacer las cosas en Python. También aclaré algunas secciones en las que ciertos detalles del lenguaje no se presentaban con la precisión que podrían haber sido. Todos los proyectos se han actualizado completamente utilizando bibliotecas populares y bien mantenidas que puede utilizar con confianza para crear sus propios proyectos.

El siguiente es un resumen de los cambios específicos que se han realizado en la tercera edición:

- El Capítulo 1 ahora presenta el editor de texto VS Code, que es popular entre programadores principiantes y profesionales y funciona bien en todos los sistemas operativos.
- El Capítulo 2 incluye los nuevos métodos `removeprefix()` y `removesuffix()`, que son útiles cuando se trabaja con archivos y URL. Este capítulo también presenta los mensajes de error recientemente mejorados de Python, que brindan información mucho más específica para ayudarlo a solucionar problemas de su código cuando algo sale mal.
- El Capítulo 10 utiliza el módulo `pathlib` para trabajar con archivos. Este es un enfoque mucho más sencillo para leer y escribir en archivos.
- El Capítulo 11 utiliza `pytest` para escribir pruebas automatizadas para el código que escribe. La biblioteca `pytest` se ha convertido en la herramienta estándar de la industria para escribir pruebas en Python. Es lo suficientemente amigable como para usarlo en tus primeras pruebas y, si sigues una carrera como programador de Python, también lo usarás en entornos profesionales.
- El proyecto Alien Invasion en los Capítulos 12-14 incluye un escenario para controlar la velocidad de cuadros, lo que hace que el juego se ejecute de manera más consistente en diferentes sistemas operativos. Se utiliza un enfoque más simple para construir la flota de extraterrestres y también se ha limpiado la organización general del proyecto.
- Los proyectos de visualización de datos de los Capítulos 15 a 17 utilizan las funciones más recientes de `Matplotlib` y `Plotly`. Las visualizaciones de `Matplotlib` presentan configuraciones de estilo actualizadas. El proyecto de caminata aleatoria tiene una pequeña mejora que aumenta la precisión de los gráficos, lo que significa que verá surgir una variedad más amplia de patrones cada vez que genere una nueva caminata. Todos los proyectos que presentan `Plotly` ahora utilizan el módulo `Plotly Express`, que le permite generar sus visualizaciones iniciales con solo unas pocas líneas de código. Puede explorar fácilmente una variedad de visualizaciones antes de comprometerse con un tipo de trama y luego concentrarse en refinar los elementos individuales de esa trama.
- El proyecto Registro de aprendizaje de los capítulos 18 a 20 se creó utilizando la última versión. Versión de Django y diseñada con la última versión de Bootstrap. Se ha cambiado el nombre de algunas partes del proyecto para que sea más fácil seguir la organización general del proyecto. El proyecto ahora está implementado en `Platform.sh`, un moderno servicio de alojamiento para proyectos de Django. El proceso de implementación está controlado por archivos de configuración YAML, que le brindan un gran control sobre cómo se implementa su proyecto. Este enfoque es consistente con la forma en que los programadores profesionales implementan proyectos modernos de Django.
- El Apéndice A se ha actualizado completamente para recomendar las mejores prácticas actuales para instalar Python en los principales sistemas operativos. El Apéndice B incluye instrucciones detalladas para configurar VS Code y breves descripciones de la mayoría de los principales editores de texto e IDE que se utilizan actualmente. Apéndice C dirige a los lectores a varios de los recursos en línea más populares para obtener

ayuda. El Apéndice D continúa ofreciendo un mini curso intensivo sobre el uso de Git para el control de versiones. El Apéndice E es completamente nuevo para la tercera edición. Incluso con un buen conjunto de instrucciones para implementar las aplicaciones que crea, hay muchas cosas que pueden salir mal. Este apéndice ofrece una guía detallada de solución de problemas que puede utilizar cuando el proceso de implementación no funciona en el primer intento.

- El índice se ha actualizado completamente para permitirle utilizar el Curso intensivo de Python como referencia para todos sus proyectos futuros de Python.

¡Gracias por leer el curso intensivo de Python! Si tienes algún comentario o preguntas, no dude en ponerse en contacto; Soy @ehmatthes en Twitter.

EXPRESIONES DE GRATITUD

Este libro no habría sido posible sin el maravilloso y extremadamente profesional personal de No Starch Press. Bill Pollock me invitó a escribir un libro introductorio y aprecio profundamente esa oferta original. Liz Chadwick ha trabajado en las tres ediciones y el libro es mejor gracias a su participación continua. Eva Morrow aportó nuevos ojos a esta nueva edición y sus ideas también han mejorado el libro. Aprecio la guía de Doug McNair sobre el uso de la gramática adecuada, sin volverse demasiado formal. Jennifer Kepler ha supervisado el trabajo de producción, que convierte mis numerosos archivos en un producto final pulido.

Hay muchas personas en No Starch Press que han ayudado a que este libro sea un éxito pero con quienes no he tenido la oportunidad de trabajar directamente. No Starch cuenta con un equipo de marketing fantástico, que va más allá de la simple venta de libros; se aseguran de que los lectores encuentren los libros que probablemente les funcionen bien y les ayuden a alcanzar sus objetivos. No Starch también tiene un sólido departamento de derechos extranjeros. El curso intensivo de Python ha llegado a lectores de todo el mundo, en muchos idiomas, gracias a la diligencia de este equipo. A todas estas personas con las que no he trabajado individualmente, gracias por ayudar a Python Crash Course a encontrar su audiencia.

Me gustaría agradecer a Kenneth Love, el revisor técnico de los tres ediciones del curso intensivo de Python. Conocí a Kenneth en PyCon un año y su entusiasmo por el lenguaje y la comunidad Python ha sido una fuente constante de inspiración profesional desde entonces. Kenneth, como siempre, fue más allá de la simple verificación de hechos y revisó el libro con el objetivo de

Ayudar a los programadores más nuevos a desarrollar una comprensión sólida del lenguaje Python y la programación en general. También estuvieron atentos a las áreas que funcionaron bastante bien en ediciones anteriores pero que podrían mejorarse si se les diera la oportunidad de realizar una reescritura completa. Dicho esto, cualquier inexactitud que quede es completamente mía.

También me gustaría expresar mi agradecimiento a todos los lectores que han compartido su experiencia trabajando a través del Python Crash Course. Aprender los conceptos básicos de la programación puede cambiar tu perspectiva sobre el mundo y, a veces, esto tiene un profundo impacto en las personas. Es una profunda lección de humildad escuchar estas historias y aprecio a todos los que han compartido sus experiencias tan abiertamente.

Me gustaría agradecer a mi padre por iniciarme en la programación desde muy joven y por no tener miedo de que rompiera su equipo. Me gustaría agradecer a mi esposa, Erin, por apoyarme y alentarme durante la escritura de este libro y durante todo el trabajo necesario para mantenerlo en múltiples ediciones. También me gustaría agradecer a mi hijo Ever, cuya curiosidad continúa inspirándome.

INTRODUCCIÓN



Cada programador tiene una historia sobre cómo aprendió a escribir su primer programa. Comencé a programar cuando era niño, cuando mi padre trabajaba para Digital Equipment

Corporation, una de las empresas pioneras de la era informática moderna. Escribí mi primer programa en un kit de computadora que mi papá había ensamblado en nuestro sótano. La computadora no consistía más que en una placa base conectada a un teclado sin carcasa, y su monitor era un tubo de rayos catódicos desnudo. Mi programa inicial era un simple juego de adivinanzas de números, que se parecía a esto:

¡Estoy pensando en un número! Intenta adivinar el número en el que estoy pensando: 25

¡Demasiado baja! Adivina de nuevo: 50

¡Demasiado alto! Adivina de nuevo: 42

¡Eso es todo! ¿Te gustaría volver a jugar? (sí/no) no

¡Gracias por jugar!

Siempre recordaré lo satisfecho que me sentí al ver a mi familia jugar un juego que creé y que funcionó como esperaba.

Esa primera experiencia tuvo un impacto duradero. Hay una verdadera satisfacción en construir algo con un propósito, que resuelva un problema. El software que escribo ahora satisface necesidades más importantes que los esfuerzos de mi infancia, pero la sensación de satisfacción que obtengo al crear un programa que funciona sigue siendo en gran medida la misma.

¿Para quién es este libro?

El objetivo de este libro es ponerlo al día con Python lo más rápido posible para que pueda crear programas que funcionen (juegos, visualizaciones de datos y aplicaciones web) mientras desarrolla una base en programación que le será útil para el resto de su vida. su vida. El curso intensivo de Python está escrito para personas de cualquier edad que nunca han programado en Python o nunca han programado nada. Este libro es para aquellos que desean aprender rápidamente los conceptos básicos de la programación para poder concentrarse en proyectos interesantes y para aquellos a quienes les gusta poner a prueba su comprensión de nuevos conceptos resolviendo problemas significativos. Python Crash Course también es perfecto para profesores de todos los niveles que quieran ofrecer a sus alumnos una introducción a la programación basada en proyectos.

Si estás tomando una clase universitaria y quieres una introducción a Python más amigable que el texto que te han asignado, este libro también puede hacer que tu clase sea más fácil. Si está buscando cambiar de carrera, Python Crash Course puede ayudarlo a realizar la transición hacia una carrera profesional más satisfactoria. Ha funcionado bien para una amplia variedad de lectores, con una amplia gama de objetivos.

¿Qué puedes esperar aprender?

El propósito de este libro es convertirte en un buen programador en general y en un buen programador de Python en particular. Aprenderá de manera eficiente y adoptará buenos hábitos a medida que obtenga una base sólida en conceptos generales de programación. Después de avanzar en el curso intensivo de Python, debería estar listo para pasar a técnicas de Python más avanzadas y su próximo lenguaje de programación será aún más fácil de comprender.

En la Parte I de este libro, aprenderá los conceptos básicos de programación que necesita para escribir programas en Python. Estos conceptos son los mismos que aprendería al comenzar en casi cualquier lenguaje de programación. Aprenderá sobre diferentes tipos de datos y las formas en que puede almacenar datos en sus programas. Creará colecciones de datos, como listas y diccionarios, y trabajará con esas colecciones de manera eficiente. Aprenderá a usar bucles while y sentencias if para probar ciertas condiciones, de modo que pueda ejecutar secciones específicas de código mientras esas condiciones son verdaderas y ejecutar otras secciones cuando no lo son, una técnica que le ayuda a automatizar muchos procesos.

Aprenderá a aceptar aportaciones de los usuarios para que sus programas sean interactivos y a mantenerlos ejecutándose todo el tiempo que el usuario desee. Explorarás cómo escribir funciones que hagan que partes de tu programa sean reutilizables.

por lo que solo tienes que escribir bloques de código que realicen ciertas acciones una vez, mientras usas ese código tantas veces como necesites. Luego ampliará este concepto a comportamientos más complicados con clases, haciendo que programas bastante simples respondan a una variedad de situaciones. Aprenderá a escribir programas que manejen correctamente los errores comunes. Después de trabajar en cada uno de estos conceptos básicos, escribirá una serie de programas cada vez más complejos utilizando lo que ha aprendido. Finalmente, darás el primer paso hacia la programación intermedia aprendiendo a escribir pruebas para tu código, de modo que puedas desarrollar más tus programas sin preocuparte por introducir errores.

Toda la información de la Parte I lo preparará para asumir proyectos más grandes y complejos.

En la Parte II, aplicará lo aprendido en la Parte I a tres proyectos. Puede realizar cualquiera o todos estos proyectos, en el orden que mejor le convenga. En el primer proyecto, en los capítulos 12 a 14, crearás un juego de disparos al estilo de Space Invaders llamado Alien Invasion, que incluye varios niveles de juego cada vez más difíciles. Una vez que hayas completado este proyecto, deberías estar en camino de poder desarrollar tus propios juegos 2D. Incluso si no aspiras a convertirte en programador de juegos, trabajar en este proyecto es una forma divertida de unir gran parte de lo que aprenderás en la Parte I.

El segundo proyecto, en los capítulos 15 a 17, le presenta la visualización de datos. Los científicos de datos utilizan una variedad de técnicas de visualización para ayudar a dar sentido a la gran cantidad de información disponible. Trabaja con conjuntos de datos que genere mediante código, conjuntos de datos que descargue de fuentes en línea y conjuntos de datos que sus programas descarguen automáticamente. Una vez que haya completado este proyecto, podrá escribir programas que analicen grandes conjuntos de datos y creen representaciones visuales de muchos tipos diferentes de información.

En el tercer proyecto, en los Capítulos 18 y 20, creará una pequeña aplicación web llamada Registro de aprendizaje. Este proyecto le permite llevar un diario organizado de la información que ha aprendido sobre un tema específico. Podrá mantener registros separados para diferentes temas y permitir que otros creen una cuenta y comiencen sus propios diarios. También aprenderá cómo implementar su proyecto para que cualquiera pueda acceder a él en línea, desde cualquier parte del mundo.

Recursos en línea

No Starch Press tiene más información sobre este libro disponible en línea en <https://nostarch.com/python-crash-course-3rd-edition>.

También mantengo un amplio conjunto de recursos complementarios en https://ehmatthes.github.io/pcc_3e. Estos recursos incluyen lo siguiente:

Instrucciones de configuración Las instrucciones de configuración en línea son idénticas a las del libro, pero incluyen enlaces activos en los que puede hacer clic para seguir los diferentes pasos. Si tiene algún problema de configuración, consulte este recurso.

Actualizaciones Python, como todos los lenguajes, está en constante evolución.

Mantengo un conjunto completo de actualizaciones, por lo que si algo no funciona, consulte aquí para ver si las instrucciones han cambiado.

Soluciones a los ejercicios Debe dedicar mucho tiempo a realizar los ejercicios de las secciones “Pruébelo usted mismo”. Sin embargo, si está estancado y no puede progresar, las soluciones para la mayoría de los ejercicios están en línea.

Hojas de referencia También está disponible en línea un conjunto completo de hojas de referencia descargables para una referencia rápida a los conceptos principales.

¿Por qué Python?

Cada año, considero si debo seguir usando Python o pasar a un lenguaje diferente, quizás uno que sea más nuevo en el mundo de la programación.

Pero sigo centrándome en Python por muchas razones. Python es un lenguaje increíblemente eficiente: sus programas harán más en menos líneas de código de lo que requerirían muchos otros lenguajes. La sintaxis de Python también te ayudará a escribir código "limpio". Su código será más fácil de leer, más fácil de depurar y más fácil de ampliar y desarrollar, en comparación con otros lenguajes.

La gente usa Python para muchos propósitos: crear juegos, crear aplicaciones web, resolver problemas comerciales y desarrollar herramientas internas en todo tipo de empresas interesantes. Python también se utiliza mucho en campos científicos, para investigación académica y trabajos aplicados.

Una de las razones más importantes por las que sigo usando Python es porque de la comunidad Python, que incluye un grupo de personas increíblemente diverso y acogedor. La comunidad es esencial para los programadores porque la programación no es una actividad solitaria. La mayoría de nosotros, incluso los programadores más experimentados, necesitamos pedir consejo a otras personas que ya han resuelto problemas similares. Tener una comunidad bien conectada y de apoyo es fundamental para ayudarlo a resolver problemas, y la comunidad de Python apoya plenamente a las personas que están aprendiendo Python como su primer lenguaje de programación o que llegan a Python con experiencia en otros lenguajes.

Python es un gran lenguaje para aprender, ¡así que comencemos!

PARTE I

LO ESENCIAL

La Parte I de este libro le enseña los conceptos básicos que necesitará para escribir programas en Python. Muchos de estos conceptos son comunes a todos los lenguajes de programación, por lo que te serán útiles a lo largo de tu vida como programador.

En el Capítulo 1 instalarás Python en tu computadora y ejecutarás tu primer programa, que imprime el mensaje ¡Hola mundo! a la pantalla.

En el Capítulo 2 aprenderá a asignar información a variables y a trabajar con texto y valores numéricos.

Los capítulos 3 y 4 presentan listas. Las listas pueden almacenar tanta información como desee en un solo lugar, lo que le permitirá trabajar con esos datos de manera eficiente. Podrás trabajar con cientos, miles e incluso millones de valores en tan solo unas pocas líneas de código.

En el Capítulo 5 usará sentencias if para escribir código que responda de una manera si ciertas condiciones son verdaderas y responda de otra manera si esas condiciones no son verdaderas.

El Capítulo 6 le muestra cómo utilizar los diccionarios de Python, que le permiten establecer conexiones entre diferentes piezas de información. Al igual que las listas, los diccionarios pueden contener tanta información como necesite almacenar.

En el Capítulo 7 aprenderá cómo aceptar entradas de los usuarios para que sus programas sean interactivos. También aprenderá sobre los bucles while, que ejecutan bloques de código repetidamente siempre que ciertas condiciones sigan siendo verdaderas.

En el Capítulo 8 escribirás funciones, que se denominan bloques de código que realizan una tarea específica y se pueden ejecutar cuando los necesite.

El Capítulo 9 presenta clases que le permiten modelar objetos del mundo real. Escribirás código que represente perros, gatos, personas, automóviles, cohetes y más.

El Capítulo 10 le muestra cómo trabajar con archivos y manejar errores para que sus programas no colapsen inesperadamente. Almacenará datos antes de que se cierre el programa y los volverá a leer cuando el programa se ejecute nuevamente. Aprenderá sobre las excepciones de Python, que le permiten anticipar errores y hacer que sus programas los manejen correctamente.

En el Capítulo 11 aprenderá a escribir pruebas para su código, para comprobar que sus programas funcionan de la forma prevista. Como resultado, podrá ampliar sus programas sin preocuparse por introducir nuevos errores. Probar tu código es una de las primeras habilidades que te ayudarán a pasar de programador principiante a programador intermedio.

1

EMPEZANDO



En este capítulo, ejecutará su primer programa Python, `hello_world.py`. Primero, deberá verificar si hay una versión reciente de Python instalada en su computadora; si no es así, lo instalará. También instalará un editor de texto para trabajar con sus programas Python. Los editores de texto reconocen el código Python y resaltan secciones mientras escribe, lo que facilita la comprensión de la estructura de su código.

Configurando su entorno de programación

Python difiere ligeramente en diferentes sistemas operativos, por lo que deberá tener en cuenta algunas consideraciones. En las siguientes secciones, nos aseguraremos de que Python esté configurado correctamente en su sistema.

Versiones de Python

Cada lenguaje de programación evoluciona a medida que surgen nuevas ideas y tecnologías, y los desarrolladores de Python continuamente han hecho que el lenguaje sea más versátil y poderoso. Al momento de escribir este artículo, la última versión es Python 3.11, pero todo lo contenido en este libro debe ejecutarse en Python 3.9 o posterior. En esta sección, descubriremos si Python ya está instalado en su sistema y si necesita instalar una versión más reciente. El Apéndice A también contiene detalles adicionales sobre la instalación de la última versión de Python en cada sistema operativo principal.

Ejecutar fragmentos de código Python

Puede ejecutar el intérprete de Python en una ventana de terminal, lo que le permitirá probar fragmentos de código Python sin tener que guardar y ejecutar un programa completo.

A lo largo de este libro, verá fragmentos de código similares a este:

```
>>> print("¡Hola intérprete de Python!")
¡Hola intérprete de Python!
```

El mensaje de tres corchetes angulares (`>>>`), al que nos referiremos como mensaje de Python, indica que debería utilizar la ventana de terminal. El texto en negrita es el código que debe escribir y luego ejecutar presionando ENTER.

La mayoría de los ejemplos de este libro son programas pequeños e independientes que ejecutará desde su editor de texto en lugar de desde la terminal, porque escribirá la mayor parte de su código en el editor de texto. Pero a veces, los conceptos básicos se mostrarán en una serie de fragmentos ejecutados en una sesión de terminal Python para demostrar conceptos particulares de manera más eficiente. Cuando ve tres corchetes angulares en una lista de códigos, está viendo el código y la salida de una sesión de terminal.

Intentaremos codificar en el intérprete de su sistema en un momento.

También usaremos un editor de texto para crear un programa simple llamado ¡Hola Mundo! que se ha convertido en un elemento básico para aprender a programar. Existe una larga tradición en el mundo de la programación que imprimir el mensaje ¡Hola mundo! a la pantalla como tu primer programa en un nuevo idioma te traerá buena suerte.

Un programa tan simple tiene un propósito muy real. Si se ejecuta correctamente en su sistema, cualquier programa Python que escriba debería funcionar también.

Acerca del editor de código VS

VS Code es un potente editor de texto de calidad profesional, gratuito y apto para principiantes. VS Code es ideal tanto para proyectos simples como complejos, por lo que si te sientes cómodo usándolo mientras aprendes Python, puedes continuar usándolo a medida que avanzas hacia proyectos más grandes y complicados. VS Code se puede instalar en todos los sistemas operativos modernos y es compatible con la mayoría de los lenguajes de programación, incluido Python.

El Apéndice B proporciona información sobre otros editores de texto. Si tiene curiosidad acerca de las otras opciones, tal vez desee leer el apéndice en este

punto. Si desea comenzar a programar rápidamente, puede usar VS Code para comenzar. Luego podrás considerar otros editores, una vez que hayas adquirido algo de experiencia como programador. En este capítulo, lo guiaré en la instalación de VS Code en su sistema operativo.

NOTA Si ya tiene instalado un editor de texto y sabe cómo configurarlo para ejecutar programas Python, puede utilizar ese editor en su lugar.

Python en diferentes sistemas operativos

Python es un lenguaje de programación multiplataforma, lo que significa que se ejecuta en todos los principales sistemas operativos. Cualquier programa Python que escriba debe ejecutarse en cualquier computadora moderna que tenga Python instalado. Sin embargo, los métodos para configurar Python en diferentes sistemas operativos varían ligeramente.

En esta sección, aprenderá cómo configurar Python en su sistema. Primero comprobará si hay una versión reciente de Python instalada en su sistema y la instalará si no es así. Luego instalará VS Code. Estos son los únicos dos pasos que son diferentes para cada sistema operativo.

En las secciones siguientes, ejecutará `hello_world.py` y solucionará cualquier problema que no funcione. Lo guiaré a través de este proceso para cada sistema operativo, para que tenga un entorno de programación Python en el que pueda confiar.

Python en Windows

Windows generalmente no viene con Python, por lo que probablemente necesitarás instalarlo y luego instalar VS Code.

Instalación de Python

Primero, verifique si Python está instalado en su sistema. Abra una ventana de comando ingresando el comando en el menú Inicio y haciendo clic en la aplicación Símbolo del sistema. En la ventana de la terminal, ingrese `python` en minúsculas. Si recibe un mensaje de Python (`>>>`) como respuesta, Python está instalado en su sistema. Si ve un mensaje de error que le indica que Python no es un comando reconocido, o si se abre la tienda de Microsoft, Python no está instalado. Cierra la tienda de Microsoft si se abre; Es mejor descargar un instalador oficial que utilizar la versión de Microsoft.

Si Python no está instalado en su sistema, o si ve una versión anterior a Python 3.9, necesita descargar un instalador de Python para Windows. Vaya a <https://python.org> y coloque el cursor sobre el enlace Descargas.

Debería ver un botón para descargar la última versión de Python.

Haga clic en el botón, que debería comenzar a descargar automáticamente el instalador correcto para su sistema. Una vez que haya descargado el archivo, ejecute el instalador. Asegúrese de seleccionar la opción Agregar Python a PATH, lo que facilitará la configuración correcta de su sistema. La Figura 1-1 muestra esta opción seleccionada.



Figura 1-1: Asegúrese de seleccionar la casilla de verificación denominada Agregar Python a la RUTA.

Ejecutar Python en una sesión de terminal

Abra una nueva ventana de comando e ingrese python en minúsculas. Debería ver un mensaje de Python (>>>), lo que significa que Windows ha encontrado la versión de Python que acaba de instalar.

```
C:\> python
Python 3.xx (principal, junio..., 13:29:14) [MSC v.1932 64 bits (AMD64)] en win32
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

NOTA no ve este resultado o algo similar, consulte las instrucciones de configuración más detalladas. ciones en el Apéndice A.

Ingresa la siguiente línea en su sesión de Python:

```
>>> print("¡Hola intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

Deberías ver el resultado ¡Hola intérprete de Python! Cada vez que desee ejecutar un fragmento de código Python, abra una ventana de comandos e inicie una sesión de terminal Python. Para cerrar la sesión de terminal, presione CTRL-Z y luego presione ENTER, o ingrese el comando exit().

Instalación de código VS

Puede descargar un instalador de VS Code en <https://code.visualstudio.com>. Haga clic en el botón Descargar para Windows y ejecute el instalador. Omita las siguientes secciones sobre macOS y Linux y siga los pasos en “Ejecución de un programa Hello World” en la página 9.

Python en MacOS

Python no está instalado de forma predeterminada en las últimas versiones de macOS, por lo que deberás instalarlo si aún no lo has hecho. En esta sección, instalará la última versión de Python y luego instalará VS Code y se asegurará de que esté configurado correctamente.

NOTA Python 2 se incluyó en versiones anteriores de macOS, pero es una versión desactualizada que no debes usar.

Comprobar si Python 3 está instalado

Abra una ventana de terminal yendo a Aplicaciones > Utilidades > Terminal.

También puede presionar `⌘+Espacio`, escribir terminal y luego presionar ENTRAR. Para ver si tiene instalada una versión suficientemente reciente de Python, ingrese `python3`. Lo más probable es que vea un mensaje sobre la instalación de las herramientas de desarrollo de la línea de comandos. Es mejor instalar estas herramientas después de instalar Python, por lo que si aparece este mensaje, cancele la ventana emergente.

Si el resultado muestra que tiene instalado Python 3.9 o una versión posterior, puede omitir la siguiente sección e ir a "Ejecutar Python en una sesión de terminal". Si ve alguna versión anterior a Python 3.9, siga las instrucciones de la siguiente sección para instalar la última versión.

Tenga en cuenta que en macOS, siempre que vea el comando `python` en este libro, deberá usar el comando `python3` para asegurarse de que está usando Python 3. En la mayoría de los sistemas macOS, el comando `python` apunta a una versión obsoleta de Python que Sólo debe ser utilizado por herramientas internas del sistema, o no apunta a nada y genera un mensaje de error.

Instalación de la última versión de Python

Puede encontrar un instalador de Python para su sistema en <https://python.org>. Pase el cursor sobre el enlace Descargar y debería ver un botón para descargar la última versión de Python. Haga clic en el botón, que debería comenzar a descargar automáticamente el instalador correcto para su sistema. Después de que se descargue el archivo, ejecute el instalador.

Después de que se ejecute el instalador, debería aparecer una ventana del Finder. Haga doble clic en el archivo `Install Certificates.command`. La ejecución de este archivo le permitirá instalar más fácilmente bibliotecas adicionales que necesitará para proyectos del mundo real, incluidos los proyectos de la segunda mitad de este libro.

Ejecutar Python en una sesión de terminal

Ahora puede intentar ejecutar fragmentos de código Python abriendo una nueva ventana de terminal y escribiendo `python3`:

```
$ python3
Python 3.xx (v3.11.0:eb0004c271, junio..., 10:03:01)
[Clang 13.0.0 (clang-1300.0.29.30)] en darwin
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

Este comando inicia una sesión de terminal Python. Deberías ver un mensaje de Python (>>>), lo que significa que macOS ha encontrado la versión de Python que acabas de instalar.

Ingresa la siguiente línea en la sesión del terminal:

```
>>> print("¡Hola intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

Debería ver el mensaje ¡Hola intérprete de Python!, que debería imprimirse directamente en la ventana de terminal actual. Puede cerrar el intérprete de Python presionando CTRL-D o ingresando el comando exit().

NOTA En los sistemas macOS más nuevos, verá un signo de porcentaje (%) como mensaje de terminal en lugar de un signo de dólar (\$).

Instalación de código VS

Para instalar el editor de VS Code, debe descargar el instalador en <https://código.visualstudio.com>. Haga clic en el botón Descargar y luego abra un Finder ventana y vaya a la carpeta Descargas . Arrastre el código de Visual Studio instalador en su carpeta Aplicaciones, luego haga doble clic en el instalador para ejecutarlo.

Omita la siguiente sección sobre Python en Linux y siga los pasos en "Ejecución de un programa Hello World" en la página 9.

Pitón en Linux

Los sistemas Linux están diseñados para la programación, por lo que Python ya está instalado en la mayoría de las computadoras Linux. Las personas que escriben y mantienen Linux esperan que usted haga su propia programación en algún momento y lo alientan a hacerlo. Por esta razón, hay muy poco que instalar y sólo algunas configuraciones que cambiar para comenzar a programar.

Comprobando su versión de Python

Abra una ventana de terminal ejecutando la aplicación Terminal en su sistema (en Ubuntu, puede presionar CTRL-ALT-T). Para saber qué versión de Python está instalada, ingrese python3 con una p minúscula. Cuando se instala Python, este comando inicia el intérprete de Python. Debería ver un resultado que indique qué versión de Python está instalada. También debería ver un mensaje de Python (>>>) donde puede comenzar a ingresar comandos de Python:

```
$ python3
Python 3.10.4 (principal, abril..., 09:04:19) [GCC 11.2.0] en Linux
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

Este resultado indica que Python 3.10.4 es actualmente la versión predeterminada de Python instalada en esta computadora. Cuando haya visto este resultado, presione CTRL-D o ingrese exit() para salir del indicador de Python y regresar a un

mensaje de terminal. Siempre que vea el comando python en este libro, ingrese python3 en su lugar.

Necesitará Python 3.9 o posterior para ejecutar el código de este libro. Si la versión de Python instalada en su sistema es anterior a Python 3.9, o si desea actualizar a la última versión disponible actualmente, consulte las instrucciones en el Apéndice A.

Ejecutar Python en una sesión de terminal

Puede intentar ejecutar fragmentos de código Python abriendo una terminal e ingresando python3, como lo hizo al verificar su versión. Haga esto nuevamente y cuando tenga Python ejecutándose, ingrese la siguiente línea en la sesión de terminal:

```
>>> print("¡Hola intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

El mensaje debe imprimirse directamente en la ventana de terminal actual.

Recuerde que puede cerrar el intérprete de Python presionando CTRL-D o ingresando el comando `exit()`.

Instalación de código VS

En Ubuntu Linux, puede instalar VS Code desde el Centro de software de Ubuntu. Haga clic en el ícono del software Ubuntu en su menú y busque `vscode`.

Haga clic en la aplicación llamada Visual Studio Code (a veces llamada código) y luego haga clic en Instalar. Una vez instalado, busque VS Code en su sistema e inicie la aplicación.

Ejecución de un programa Hola Mundo

Con una versión reciente de Python y VS Code instalada, está casi listo para ejecutar su primer programa Python escrito en un editor de texto. Pero antes de hacerlo, debes instalar la extensión Python para VS Code.

Instalación de la extensión Python para VS Code

VS Code funciona con muchos lenguajes de programación diferentes; Para aprovecharlo al máximo como programador de Python, deberá instalar la extensión de Python. Esta extensión agrega soporte para escribir, editar y ejecutar programas Python.

Para instalar la extensión Python, haga clic en el ícono Administrar, que parece un engranaje en la esquina inferior izquierda de la aplicación VS Code. En el menú que aparece, haga clic en Extensiones. Ingrese Python en el cuadro de búsqueda y haga clic en la extensión Python . (Si ve más de una extensión llamada Python, elija la proporcionada por Microsoft). Haga clic en Instalar e instale las herramientas adicionales que su sistema necesite para completar la instalación. Si ve un mensaje que indica que necesita instalar Python y ya lo ha hecho, puede ignorarlo.

NOTA está utilizando macOS y una ventana emergente le solicita que instale las herramientas de desarrollo de la línea de comandos, haga clic en Instalar. Es posible que vea un mensaje que indica que la instalación llevará demasiado tiempo, pero sólo debería llevar unos 10 o 20 minutos con una conexión a Internet razonable.

Ejecutando hola_mundo.py

Antes de escribir su primer programa, cree una carpeta llamada `python_work` en su escritorio para sus proyectos. Es mejor usar letras minúsculas y guiones bajos para los espacios en los nombres de archivos y carpetas, porque Python usa estas convenciones de nomenclatura. Puede crear esta carpeta en otro lugar que no sea el escritorio, pero será más fácil seguir algunos pasos posteriores si guarda la carpeta `python_work` directamente en su escritorio.

Abra VS Code y cierre la pestaña Comenzar si aún está abierta. Cree un archivo nuevo haciendo clic en Archivo4Nuevo archivo o presionando CTRL-N (-N en macOS). Guarde el archivo como `hello_world.py` en su carpeta `python_work` . La extensión .py le dice a VS Code que su archivo está escrito en Python y le indica cómo ejecutar el programa y resaltar el texto de una manera útil.

Después de haber guardado su archivo, ingrese la siguiente línea en el editor:

```
hello_world.py print("¡Hola mundo Python!")
```

Para ejecutar su programa, seleccione Run4Run sin depurar o presione CTRL-F5. Debería aparecer una pantalla de terminal en la parte inferior de la ventana de VS Code, mostrando el resultado de su programa:

```
¡Hola mundo Python!
```

Probablemente verá algún resultado adicional que muestre el intérprete de Python que se utilizó para ejecutar su programa. Si desea simplificar la información que se muestra para que solo vea el resultado de su programa, consulte el Apéndice B. También puede encontrar sugerencias útiles sobre cómo usar VS Code de manera más eficiente en el Apéndice B.

Si no ve este resultado, es posible que algo haya salido mal en el programa. Verifique cada carácter en la línea que ingresó. ¿Accidentalmente pusiste mayúscula en letra impresa? ¿Olvidó una o ambas comillas o paréntesis? Los lenguajes de programación esperan una sintaxis muy específica y, si no la proporciona, obtendrá errores. Si no puede ejecutar el programa, consulte las sugerencias en la siguiente sección.

Solución de problemas

Si no puede ejecutar `hello_world.py` , aquí hay algunos remedios que puede probar y que también son buenas soluciones generales para cualquier problema de programación:

- Cuando un programa contiene un error importante, Python muestra un rastreo, que es un informe de error. Python revisa el archivo e intenta identificar el problema. Verifique el rastreo; podría darle una pista sobre qué problema impide que se ejecute el programa.

- Aléjese de su computadora, tómese un breve descanso y luego vuelva a intentarlo.
Recuerde que la sintaxis es muy importante en la programación, por lo que algo tan simple como comillas o paréntesis que no coinciden pueden impedir que un programa se ejecute correctamente. Vuelva a leer las partes relevantes de este capítulo, revise su código e intente encontrar el error.
- Empezar de nuevo. Probablemente no necesites desinstalar ningún software, pero podría tener sentido eliminar tu archivo `hello_world.py` y volver a crearlo desde cero.
- Pídele a otra persona que siga los pasos de este capítulo, en tu computadora o en otra diferente, y observa atentamente lo que hace. Es posible que hayas omitido un pequeño paso que alguien más ha captado.
- Consulte las instrucciones de instalación adicionales en el Apéndice A; Algunos de los detalles incluidos en el Apéndice pueden ayudarle a resolver su problema.
- Busque a alguien que conozca Python y pídale que le ayude a configurarlo.
Si preguntas, es posible que descubras que inesperadamente conoces a alguien que usa Python.
- Las instrucciones de configuración de este capítulo también están disponibles a través del sitio web complementario de este libro en https://ehmatthes.github.io/pcc_3e.
La versión en línea de estas instrucciones podría funcionar mejor porque simplemente puede cortar y pegar código y hacer clic en los enlaces a los recursos que necesita.
- Pide ayuda en línea. El Apéndice C proporciona una serie de recursos, como foros y sitios de chat en vivo, donde puede solicitar soluciones a personas que ya han solucionado el problema al que se enfrenta actualmente.

No te preocupes si estás molestando a programadores experimentados. Todo programador se ha quedado atascado en algún momento y la mayoría de los programadores estarán felices de ayudarte a configurar su sistema correctamente. Siempre que pueda indicar claramente lo que está tratando de hacer, lo que ya ha intentado y los resultados que está obteniendo, es muy probable que alguien pueda ayudarte. Como se mencionó en la introducción, la comunidad Python es muy amigable y acogedora con los principiantes.

Python debería funcionar bien en cualquier computadora moderna. Los problemas de configuración temprana pueden resultar frustrantes, pero vale la pena solucionarlos. Una vez que recibas `hola_world.py` en ejecución, podrá comenzar a aprender Python y su trabajo de programación será más interesante y satisfactorio.

Ejecutar programas Python desde una terminal

Ejecutarás la mayoría de tus programas directamente en tu editor de texto. Sin embargo, a veces es útil ejecutar programas desde una terminal. Por ejemplo, es posible que desee ejecutar un programa existente sin abrirlo para editarlo.

Puede hacer esto en cualquier sistema con Python instalado si sabe cómo acceder al directorio donde está almacenado el archivo del programa. Para probar esto, asegúrese de haber guardado el archivo `hello_world.py` en la carpeta `python_work` de su escritorio.

En Windows

Puede usar el comando de terminal `cd`, para cambiar directorio, para navegar a través de su sistema de archivos en una ventana de comandos. El comando `dir`, para directorio, le muestra todos los archivos que existen en el directorio actual.

Abra una nueva ventana de terminal e ingrese los siguientes comandos para ejecutar `hello_world.py`:

```
C:\> cd Escritorio\python_work
C:\Escritorio\python_work> dir
hola_mundo.py
C:\Escritorio\python_work> python hola_mundo.py
¡Hola mundo Python!
```

Primero, use el comando `cd` para navegar a la carpeta `python_work`, que se encuentra en la carpeta `Escritorio`. A continuación, use el comando `dir` para asegurarse de que `hello_world.py` está en esta carpeta. Luego ejecute el archivo usando el comando `python hello_world.py`.

La mayoría de sus programas funcionarán bien directamente desde su editor. Sin embargo, A medida que su trabajo se vuelve más complejo, querrá ejecutar algunos de sus programas desde una terminal.

En macOS y Linux

Ejecutar un programa Python desde una sesión de terminal es lo mismo en Linux y macOS. Puede usar el comando de terminal `cd`, para cambiar directorio, para navegar a través de su sistema de archivos en una sesión de terminal. El comando `ls`, para lista, le muestra todos los archivos no ocultos que existen en el directorio actual.

Abra una nueva ventana de terminal e ingrese los siguientes comandos para ejecutar `hello_world.py`:

```
~$ cd Escritorio/python_work/
~/Escritorio/python_work$ ls
hola_mundo.py
~/Desktop/python_work$ python3 hola_mundo.py
¡Hola mundo Python!
```

Primero, use el comando `cd` para navegar a la carpeta `python_work`, que se encuentra en la carpeta `Escritorio`. A continuación, use el comando `ls` para asegurarse de que `hello_world.py` esté en esta carpeta. Luego ejecute el archivo usando el comando `python3 hello_world.py`.

La mayoría de sus programas funcionarán bien directamente desde su editor. Pero como Si su trabajo se vuelve más complejo, querrá ejecutar algunos de sus programas desde una terminal.

INTÉNTALO TÚ MISMO

Los ejercicios de este capítulo son de naturaleza exploratoria. A partir del Capítulo 2, los desafíos que resolverás se basarán en lo que hayas aprendido.

1-1. python.org: explore la página de inicio de Python (<https://python.org>) para encontrar temas destacados que le interesen. A medida que se familiarice con Python, diferentes partes del sitio le resultarán más útiles.

1-2. Errores tipográficos de Hello World: abra el archivo `hello_world.py` que acaba de crear. Haga un error tipográfico en algún lugar de la línea y ejecute el programa nuevamente.

¿Puedes cometer un error tipográfico que genere un error? ¿Puedes entender el mensaje de error? ¿Puedes cometer un error tipográfico que no genere un error? ¿Por qué crees que no cometió un error?

1-3. Habilidades infinitas: si tuvieras infinitas habilidades de programación, ¿qué construirías? Estás a punto de aprender a programar. Si tiene un objetivo final en mente, tendrá un uso inmediato para sus nuevas habilidades; Ahora es un buen momento para escribir breves descripciones de lo que quieres crear. Es un buen hábito tener un cuaderno de "ideas" al que puedas consultar siempre que quieras comenzar un nuevo proyecto. Tómame unos minutos para describir tres programas que deseas crear.

Resumen

En este capítulo, aprendió un poco sobre Python en general e instaló Python en su sistema si aún no estaba allí. También instaló un editor de texto para facilitar la escritura de código Python. Ejecutó fragmentos de código Python en una sesión de terminal y ejecutó su primer programa, `hello_world.py`. Probablemente también hayas aprendido un poco sobre la resolución de problemas.

En el próximo capítulo, aprenderá sobre los diferentes tipos de datos que puede trabajar en sus programas Python y comenzará a utilizar variables también.

2

VARIABLES TIPOS DE DATOS SIMPLES



En este capítulo aprenderá sobre los diferentes tipos de datos con los que puede trabajar en sus programas Python. También aprenderá a utilizar variables para representar datos en su programas.

Qué sucede realmente cuando ejecutas `hello_world.py`

Echemos un vistazo más de cerca a lo que hace Python cuando ejecuta `hello_world.py`. Resulta que Python hace bastante trabajo, incluso cuando ejecuta un programa simple:

```
hello_world.py print("¡Hola mundo Python!")
```

Cuando ejecute este código, debería ver el siguiente resultado:

```
¡Hola mundo Python!
```

Cuando ejecuta el archivo `hello_world.py`, la terminación `.py` indica que el archivo es un programa Python. Luego, su editor ejecuta el archivo a través del intérprete de Python, que lee el programa y determina qué significa cada palabra en el programa. Por ejemplo, cuando el intérprete ve la palabra imprimir seguida de paréntesis, imprime en la pantalla lo que esté dentro de los paréntesis.

Mientras escribe sus programas, su editor resalta diferentes partes de su programa de diferentes maneras. Por ejemplo, reconoce que `print()` es el nombre de una función y muestra esa palabra en un color. Reconoce que `"¡Hola mundo Python!"` no es código Python y muestra esa frase en un color diferente. Esta característica se llama resaltado de sintaxis y es bastante útil cuando comienzas a escribir tus propios programas.

variables

Intentemos usar una variable en `hello_world.py`. Agregue una nueva línea al comienzo del archivo y modifique la segunda línea:

```
hello_world.py mensaje = "¡Hola mundo Python!"
imprimir (mensaje)
```

Ejecute este programa para ver qué sucede. Deberías ver el mismo resultado que viste anteriormente:

```
¡Hola mundo Python!
```

Hemos agregado una variable llamada `mensaje`. Cada variable está conectada a un valor, que es la información asociada con esa variable. En este caso, el valor es `"¡Hola mundo Python!"` texto.

Agregar una variable hace un poco más de trabajo para el intérprete de Python. Cuando procesa la primera línea, asocia el mensaje variable con el mensaje `"¡Hola mundo Python!"` texto. Cuando llega a la segunda línea, imprime el valor asociado con el mensaje en la pantalla.

Amplíemos este programa modificando `hello_world.py` para imprimir un segundo mensaje. Agregue una línea en blanco a `hello_world.py` y luego agregue dos nuevas líneas de código:

```
mensaje = "¡Hola mundo Python!"
imprimir (mensaje)

mensaje = "¡Hola mundo del curso intensivo de Python!"
imprimir (mensaje)
```

Ahora, cuando ejecute `hello_world.py`, debería ver dos líneas de resultado:

```
¡Hola mundo Python!
¡Hola mundo del curso intensivo de Python!
```

Puede cambiar el valor de una variable en su programa en cualquier momento y Python siempre realizará un seguimiento de su valor actual.

Nombrar y usar variables

Cuando utiliza variables en Python, debe cumplir con algunas reglas y pautas. Romper algunas de estas reglas provocará errores; otras pautas simplemente lo ayudan a escribir código que sea más fácil de leer y comprender. Asegúrese de tener en cuenta las siguientes reglas cuando trabaje con variables:

- Los nombres de variables sólo pueden contener letras, números y guiones bajos.
Pueden empezar con una letra o un guión bajo, pero no con un número.
Por ejemplo, puede llamar a una variable `mensaje_1` pero no a `1_mensaje`.
- No se permiten espacios en los nombres de las variables, pero se pueden utilizar guiones bajos para separar palabras en nombres de variables. Por ejemplo, `mensaje_saludo` funciona pero el `mensaje de saludo` provocará errores.
- Evite el uso de palabras clave de Python y nombres de funciones como nombres de variables.
Por ejemplo, no utilice la palabra `imprimir` como nombre de variable; Python lo ha reservado para un propósito programático particular. (Consulte “Palabras clave de Python y funciones integradas” en la página 466).
- Los nombres de las variables deben ser breves pero descriptivos. Por ejemplo, `nombre` es mejor que `n`, `nombre_estudiante` es mejor que `s_n` y `longitud_nombre` es mejor que `longitud_nombre_personas`.
- Tenga cuidado al utilizar la letra minúscula `l` y la letra mayúscula `O` porque podrían confundirse con los números 1 y 0.

Puede requerir algo de práctica aprender a crear buenos nombres de variables, especialmente a medida que sus programas se vuelven más interesantes y complicados.
A medida que escribas más programas y empieces a leer el código de otras personas, mejorarás a la hora de encontrar nombres significativos.

NOTA Las variables de Python que está utilizando en este momento deben estar en minúsculas. No obtendrá errores si usa letras mayúsculas, pero las letras mayúsculas en los nombres de variables tienen significados especiales que analizaremos en capítulos posteriores.

Evitar errores de nombre al utilizar variables

Todo programador comete errores y la mayoría comete errores todos los días.

Aunque los buenos programadores pueden cometer errores, también saben cómo responder a esos errores de manera eficiente. Veamos un error que probablemente cometa desde el principio y aprendamos cómo solucionarlo.

Escribiremos un código que genere un error a propósito. Ingrese el siguiente código, incluido el mensaje de la palabra mal escrita, que se muestra en negrita:

```
mensaje = "¡Hola lector del curso intensivo de Python!"
imprimir (mensaje)
```

Cuando ocurre un error en su programa, el intérprete de Python hace todo lo posible para ayudarlo a descubrir dónde está el problema. El intérprete proporciona un rastreo cuando un programa no se puede ejecutar correctamente. Un rastreo es un registro de dónde tuvo problemas el intérprete al intentar ejecutar su código.

A continuación se muestra un ejemplo del rastreo que proporciona Python después de haber escrito mal accidentalmente el nombre de una variable:

Rastreo (llamadas recientes más última):

1 Archivo "hello_world.py", línea 2, en <módulo>

2 imprimir (mensaje)
 KAAAAA

3 NameError: el nombre 'mensaje' no está definido. ¿Quiso decir: 'mensaje'?

El resultado informa que se produce un error en la línea 2 del archivo hello_world.py 1.

El intérprete muestra esta línea 2 para ayudarnos a detectar el error rápidamente y nos dice qué tipo de error encontró 3. En este caso encontró un error de nombre

e informa que la variable que se está imprimiendo, mensaje, no ha sido definida.

Python no puede identificar el nombre de la variable proporcionada. Un error de nombre generalmente significa que olvidamos establecer el valor de una variable antes de usarla o que cometimos un error ortográfico al ingresar el nombre de la variable. Si Python encuentra un nombre de variable similar al que no reconoce, le preguntará si ese es el nombre que deseaba utilizar.

En este ejemplo omitimos la letra s en el mensaje del nombre de la variable en la segunda línea. El intérprete de Python no revisa la ortografía de su código, pero sí garantiza que los nombres de las variables estén escritos de manera consistente. Por ejemplo, observe lo que sucede cuando escribimos mensaje incorrectamente en la línea que define la variable:

```
mensaje = "¡Hola lector del curso intensivo de Python!"
```

```
imprimir (mensaje)
```

En este caso, ¡el programa se ejecuta correctamente!

```
¡Hola, lector del curso intensivo de Python!
```

Los nombres de las variables coinciden, por lo que Python no ve ningún problema. Los lenguajes de programación son estrictos, pero ignoran la buena y la mala ortografía. Como resultado, no necesita considerar las reglas ortográficas y gramaticales del inglés cuando intenta crear nombres de variables y escribir código.

Muchos errores de programación son simples errores tipográficos de un solo carácter en una línea de un programa. Si pasa mucho tiempo buscando uno de estos errores, sepa que está en buena compañía. Muchos programadores experimentados y talentosos pasan horas buscando este tipo de pequeños errores. Intente reírse de ello y siga adelante, sabiendo que sucederá con frecuencia a lo largo de su vida como programador.

Las variables son etiquetas

Las variables a menudo se describen como cuadros en los que puede almacenar valores. Esta idea puede ser útil las primeras veces que usa una variable, pero no es una forma precisa de describir cómo se representan las variables internamente en Python. Es mucho mejor pensar en las variables como etiquetas que puedes asignar a valores. También se puede decir que una variable hace referencia a un determinado valor.

Esta distinción probablemente no importará mucho en sus programas iniciales, pero vale la pena aprenderlo más temprano que tarde. En algún momento, verá un comportamiento inesperado de una variable y una comprensión precisa de cómo funcionan las variables le ayudará a identificar lo que sucede en su código.

NOTA La mejor manera de comprender nuevos conceptos de programación es intentar usarlos en sus programas. Si te quedas atascado mientras trabajas en un ejercicio de este libro, intenta hacer otra cosa por un tiempo. Si todavía estás atascado, revisa la parte relevante de ese capítulo. Si aún necesita ayuda, consulte las sugerencias en el Apéndice C.

INTÉNTALO TÚ MISMO

Escriba un programa separado para realizar cada uno de estos ejercicios. Guarde cada programa con un nombre de archivo que siga las convenciones estándar de Python, usando letras minúsculas y guiones bajos, como `simple_message.py` y `simple_messages.py`.

2-1. Mensaje simple: asigne un mensaje a una variable y luego imprímalo.

2-2. Mensajes simples: asigne un mensaje a una variable e imprima ese mensaje. Luego cambie el valor de la variable a un nuevo mensaje e imprima el nuevo mensaje.

Instrumentos de cuerda

Debido a que la mayoría de los programas definen y recopilan algún tipo de datos y luego hacen algo útil con ellos, es útil clasificar diferentes tipos de datos. El primer tipo de datos que veremos es la cadena. Las cuerdas son bastante simples a primera vista, pero puedes usarlas de muchas maneras diferentes.

Una cadena es una serie de caracteres. Todo lo que esté entre comillas se considera una cadena en Python, y puedes usar comillas simples o dobles alrededor de tus cadenas de esta manera:

```
"Esto es una cuerda".
"Esto también es una cuerda".
```

Esta flexibilidad le permite utilizar comillas y apóstrofes dentro de sus cadenas:

```
'Le dije a mi amigo: "¡Python es mi lenguaje favorito!"'
"El lenguaje 'Python' lleva el nombre de Monty Python, no de la serpiente".
"Una de las fortalezas de Python es su comunidad diversa y solidaria".
```

Exploremos algunas de las formas en que puede usar cadenas.

Cambiar caso en una cadena con métodos

Una de las tareas más simples que puedes hacer con cadenas es cambiar el caso de las palabras en una cadena. Mire el siguiente código e intente determinar qué está sucediendo:

```
nombre.py nombre = "ada lovelace"  
imprimir(nombre.title())
```

Guarde este archivo como `name.py` y luego ejecútelo. Deberías ver este resultado:

Ada Lovelace

En este ejemplo, el nombre de la variable hace referencia a la cadena minúscula "ada lovelace". El método `title()` aparece después de la variable en la llamada `print()`. Un método es una acción que Python puede realizar sobre un dato. El punto (.) después del nombre en `nombre.title()` le dice a Python que haga que el método `title()` actúe sobre el nombre de la variable. Cada método va seguido de un conjunto de paréntesis, porque los métodos a menudo necesitan información adicional para realizar su trabajo. Esa información se proporciona entre paréntesis. La función `title()` no necesita ninguna información adicional, por lo que sus paréntesis están vacíos.

El método `title()` cambia cada palabra a mayúsculas y minúsculas, donde cada palabra comienza con una letra mayúscula. Esto es útil porque a menudo querrás pensar en un nombre como una pieza de información. Por ejemplo, es posible que desee que su programa reconozca los valores de entrada Ada, ADA y ada como el mismo nombre y los muestre todos como Ada.

También hay varios otros métodos útiles disponibles para tratar el caso. Por ejemplo, puede cambiar una cadena a letras mayúsculas o minúsculas de esta manera:

```
nombre = "Ada Lovelace"  
imprimir(nombre.upper())  
imprimir(nombre.lower())
```

Esto mostrará lo siguiente:

Ada Lovelace
ada lovelace

El método `lower()` es particularmente útil para almacenar datos. Normalmente no querrás confiar en las mayúsculas que proporcionan tus usuarios, por lo que convertirás las cadenas a minúsculas antes de almacenarlas. Luego, cuando desee mostrar la información, utilizará el caso que tenga más sentido para cada cadena.

Usar variables en cadenas

En algunas situaciones, querrás utilizar el valor de una variable dentro de una cadena. Por ejemplo, es posible que desee utilizar dos variables para representar un nombre y

un apellido, respectivamente, y luego combine esos valores para mostrar el nombre completo de alguien:

```
nombre_completo.py nombre_completo = "ada"
apellido = "lovelace"
1 nombre_completo = f'{primer_nombre} {apellido}'
imprimir(nombre_completo)
```

Para insertar el valor de una variable en una cadena, coloque la letra f inmediatamente antes de las comillas de apertura 1. Coloque llaves alrededor del nombre o nombres de cualquier variable que desee usar dentro de la cadena. Python reemplazará cada variable con su valor cuando se muestre la cadena.

Estas cadenas se llaman cadenas f. La f es para formato, porque Python formatea la cadena reemplazando el nombre de cualquier variable entre llaves con su valor. El resultado del código anterior es:

```
ada lovelace
```

Puedes hacer mucho con las cuerdas f. Por ejemplo, puede utilizar cadenas f para redactar mensajes completos utilizando la información asociada con una variable, como se muestra aquí:

```
nombre_primer = "ada"
apellido = "lovelace"
nombre_completo = f'{primer_nombre} {apellido}'
1 print(f'Hola, {nombre_completo.titulo()}!')
```

El nombre completo se usa en una oración que saluda al usuario 1, y el método title() cambia el nombre al título en mayúsculas y minúsculas. Este código devuelve un saludo simple pero con un formato agradable:

```
¡Hola Ada Lovelace!
```

También puedes usar f-strings para redactar un mensaje y luego asignar el mensaje completo a una variable:

```
nombre_primer = "ada"
apellido = "lovelace"
nombre_completo = f'{primer_nombre} {apellido}'
1 mensaje = f'¡Hola, {nombre_completo.titulo()}!'
2 imprimir (mensaje)
```

Este código muestra el mensaje ¡Hola, Ada Lovelace! también, pero al asignar el mensaje a una variable 1 hacemos que la llamada final a print() sea mucho más simple 2.

Agregar espacios en blanco a cadenas con tabulaciones o nuevas líneas

En programación, los espacios en blanco se refieren a cualquier carácter que no se pueda imprimir, como espacios, tabulaciones y símbolos de fin de línea. Puede utilizar espacios en blanco para organizar su salida para que sea más fácil de leer para los usuarios.

Para agregar una pestaña a su texto, use la combinación de caracteres `\t`:

```
>>> imprimir("Python")
Pitón
>>> imprimir("\tPython")
    Pitón
```

Para agregar una nueva línea en una cadena, use la combinación de caracteres `\n`:

```
>>> print("Idiomas:\nPython\nC\nJavaScript")
Idiomas:
Pitón
C
javascript
```

También puedes combinar tabulaciones y nuevas líneas en una sola cadena. La cadena `"\n\t"` le dice a Python que se mueva a una nueva línea y comience la siguiente línea con una tabulación. El siguiente ejemplo muestra cómo puede utilizar una cadena de una línea para generar cuatro líneas de salida:

```
>>> print("Idiomas:\n\tPython\n\tC\n\tJavaScript")
Idiomas:
    Pitón
    C
    javascript
```

Las nuevas líneas y las tabulaciones serán muy útiles en los próximos dos capítulos, cuando comienzas a producir muchas líneas de resultados a partir de unas pocas líneas de código.

Eliminación de espacios en blanco

Los espacios en blanco adicionales pueden resultar confusos en sus programas. Para los programadores, `'python'` y `'python '` parecen más o menos iguales. Pero para un programa, son dos cadenas diferentes. Python detecta el espacio extra en `'python '` y lo considera significativo a menos que indique lo contrario.

Es importante pensar en los espacios en blanco, porque a menudo querrás comparar dos cadenas para determinar si son iguales. Por ejemplo, un caso importante podría implicar verificar los nombres de usuario de las personas cuando inician sesión en un sitio web. Los espacios en blanco adicionales también pueden resultar confusos en situaciones mucho más simples. Afortunadamente, Python facilita la eliminación de espacios en blanco adicionales de los datos que ingresan las personas.

Python puede buscar espacios en blanco adicionales en los lados derecho e izquierdo de una cadena. Para asegurarse de que no exista ningún espacio en blanco en el lado derecho de una cadena, utilice el método `rstrip()` :

```
1 >>> idioma_favorito = 'python '
2 >>> idioma_favorito
'pitón '
3 >>> idioma_favorito.rstrip()
'pitón'
4 >>> idioma_favorito
'pitón '
```

El valor asociado con el idioma_favorito 1 contiene espacios en blanco adicionales. al final de la cuerda. Cuando le pides a Python este valor en una sesión terminal, puedes ver el espacio al final del valor 2. Cuando el método `rstrip()` actúa sobre la variable favorito_idioma 3, este espacio extra se elimina. Sin embargo, sólo se elimina temporalmente. Si vuelve a solicitar el valor de idioma_favorito , la cadena tendrá el mismo aspecto que cuando se ingresó, incluido el espacio en blanco adicional 4.

Para eliminar el espacio en blanco de la cadena de forma permanente, debe asociar el valor eliminado con el nombre de la variable:

```
>>> idioma_favorito = 'python '
1 >>> idioma_favorito = idioma_favorito.rstrip()
>>> idioma_favorito
'pitón'
```

Para eliminar el espacio en blanco de la cadena, elimine el espacio en blanco del lado derecho de la cadena y luego asocie este nuevo valor con la variable original 1. El cambio del valor de una variable se realiza a menudo en la programación. Así es como se puede actualizar el valor de una variable a medida que se ejecuta un programa o en respuesta a la entrada del usuario.

También puedes eliminar espacios en blanco del lado izquierdo de una cadena usando el método `lstrip()` , o desde ambos lados a la vez usando `strip()`:

```
1 >>> idioma_favorito = python 2 >>>
idioma_favorito.rstrip()
'pitón'
3 >>> idioma_favorito.lstrip()
'pitón '
4 >>> idioma_favorito.strip()
'pitón'
```

En este ejemplo, comenzamos con un valor que tiene espacios en blanco al principio y al final 1. Luego eliminamos el espacio extra del lado derecho 2, del lado izquierdo 3 y de ambos lados 4. Experimentando con estas tiras Las funciones -ping pueden ayudarle a familiarizarse con la manipulación de cadenas. En el mundo real, estas funciones de eliminación se utilizan con mayor frecuencia para limpiar la entrada del usuario antes de almacenarla en un programa.

Eliminación de prefijos

Cuando se trabaja con cadenas, otra tarea común es eliminar un prefijo.

Considere una URL con el prefijo común `https://`. Queremos eliminar este prefijo, para poder centrarnos solo en la parte de la URL que los usuarios deben ingresar en la barra de direcciones. Aquí se explica cómo hacerlo:

```
>>> nostarch_url = 'https://nostarch.com'
>>> nostarch_url.removeprefix('https://')
'nostarch.com'
```

Ingrese el nombre de la variable seguido de un punto y luego el método `removeprefix()`. Dentro del paréntesis, ingrese el prefijo que desea eliminar de la cadena original.

Al igual que los métodos para eliminar espacios en blanco, `removeprefix()` deja la cadena original sin cambios. Si desea mantener el nuevo valor sin el prefijo, reasígnelo a la variable original o asígnelo a una nueva variable:

```
>>> simple_url = nostarch_url.removeprefix("https://")
```

Cuando ve una URL en una barra de direcciones y la parte `https://` no se muestra, el navegador probablemente esté usando un método como `removeprefix()` detrás del escenas.

Evitar errores de sintaxis con cadenas

Un tipo de error que puede ver con cierta regularidad es un error de sintaxis.

Se produce un error de sintaxis cuando Python no reconoce una sección de su programa como código Python válido. Por ejemplo, si utiliza un apóstrofe entre comillas simples, producirá un error. Esto sucede porque Python interpreta todo lo que se encuentra entre la primera comilla simple y el apóstrofo como una cadena. Luego intenta interpretar el resto del texto como código Python, lo que provoca errores.

A continuación se explica cómo utilizar correctamente las comillas simples y dobles. Guarde este programa como `apostrophe.py` y luego ejecútelo:

```
apostrophe.py message = "Uno de los puntos fuertes de Python es su comunidad diversa."
imprimir (mensaje)
```

El apóstrofe aparece dentro de un conjunto de comillas dobles, por lo que el intérprete de Python no tiene problemas para leer la cadena correctamente:

```
Una de las fortalezas de Python es su comunidad diversa.
```

Sin embargo, si usa comillas simples, Python no puede identificar dónde está la cadena. debería terminar:

```
message = 'Una de las fortalezas de Python es su comunidad diversa.'
imprimir (mensaje)
```

Verá el siguiente resultado:

```
Archivo "apostrophe.py", línea 1
    message = 'Una de las fortalezas de Python es su comunidad diversa.'
```

1 ^

```
SyntaxError: literal de cadena no terminada (detectado en la línea 1)
```

En el resultado puedes ver que el error ocurre justo después del último single. cita 1. Este error de sintaxis indica que el intérprete no reconoce

algo en el código es código Python válido y cree que el problema podría ser una cadena que no está entrecomillada correctamente. Los errores pueden provenir de diversas fuentes y señalaré algunas comunes a medida que surjan. Es posible que vea errores de sintaxis con frecuencia a medida que aprende a escribir código Python adecuado. Los errores de sintaxis también son el tipo de error menos específico, por lo que puede resultar difícil y frustrante identificarlos y corregirlos. Si se queda atascado en un error particularmente persistente, consulte las sugerencias en el Apéndice C.

NOTA La función de resaltado de sintaxis de su editor debería ayudarle a detectar algunos errores de sintaxis rápidamente mientras escribe sus programas. Si ve el código Python resaltado como si fuera inglés o el inglés resaltado como si fuera código Python, probablemente tenga comillas que no coinciden en algún lugar de su archivo.

INTÉNTALO TÚ MISMO

Guarde cada uno de los siguientes ejercicios como un archivo separado, con un nombre como nombre_casos.py. Si te quedas atascado, tómate un descanso o consulta las sugerencias en el Apéndice C.

2-3. Mensaje personal: utilice una variable para representar el nombre de una persona e imprima un mensaje para esa persona. Tu mensaje debe ser simple, como "Hola Eric, ¿te gustaría aprender algo de Python hoy?"

2-4. Casos de nombre: use una variable para representar el nombre de una persona y luego imprima el nombre de esa persona en minúsculas, mayúsculas y título.

2-5. Cita famosa: encuentre una cita de una persona famosa que admire. Imprima la cita y el nombre de su autor. Su resultado debería verse similar al siguiente, incluidas las comillas:

Albert Einstein dijo una vez: "Una persona que nunca cometió un error nunca intentó nada nuevo".

2-6. Cita famosa 2: repita el ejercicio 2-5, pero esta vez, represente el nombre de la persona famosa usando una variable llamada persona_famosa. Luego redacte su mensaje y represéntelo con una nueva variable llamada mensaje. Imprime tu mensaje.

2-7. Eliminación de nombres: utilice una variable para representar el nombre de una persona e incluya algunos caracteres de espacio en blanco al principio y al final del nombre.

Asegúrese de utilizar cada combinación de caracteres, "\t" y "\n", al menos una vez.

Imprima el nombre una vez, para que se muestre el espacio en blanco alrededor del nombre.

Luego imprima el nombre usando cada una de las tres funciones de eliminación, lstrip(), rstrip() y strip().

2-8. Extensiones de archivo: Python tiene un método removesuffix() que funciona exactamente como removeprefix(). Asigne el valor 'python_notes.txt' a una variable llamada nombre de archivo. Luego use el método removesuffix() para mostrar el nombre del archivo sin la extensión del archivo, como lo hacen algunos exploradores de archivos.

Números

Los números se utilizan con bastante frecuencia en programación para llevar la puntuación en los juegos, representar datos en visualizaciones, almacenar información en aplicaciones web, etc. Python trata los números de varias maneras diferentes, dependiendo de cómo se utilicen. Primero veamos cómo Python gestiona los números enteros, porque son los más sencillos para trabajar.

Enteros

Puede sumar (+), restar (-), multiplicar (*) y dividir (/) números enteros en Python.

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 6
12
>>> 3 / 2
1.5
```

En una sesión de terminal, Python simplemente devuelve el resultado de la operación. Python usa dos símbolos de multiplicación para representar exponentes:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python también admite el orden de las operaciones, por lo que puedes utilizar varias operaciones en una expresión. También puede usar paréntesis para modificar el orden de las operaciones para que Python pueda evaluar su expresión en el orden que especifique. Por ejemplo:

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
```

El espaciado en estos ejemplos no tiene ningún efecto sobre cómo Python evalúa las expresiones; simplemente le ayuda a detectar más rápidamente las operaciones que tienen prioridad cuando lee el código.

flotadores

Python llama flotante a cualquier número con punto decimal. Este término se utiliza en la mayoría de los lenguajes de programación y se refiere al hecho de que un punto decimal

puede aparecer en cualquier posición de un número. Cada lenguaje de programación debe diseñarse cuidadosamente para gestionar adecuadamente los números decimales, de modo que los números se comporten de manera adecuada, sin importar dónde aparezca el punto decimal.

En su mayor parte, puedes utilizar flotadores sin preocuparte por cómo se comportan. Simplemente ingrese los números que desea usar y Python probablemente hará lo que espera:

```
>>> 0,1 + 0,1
0,2
>>> 0,2 + 0,2
0,4
>>> 2 * 0,1
0,2
>>> 2 * 0,2
0,4
```

Sin embargo, tenga en cuenta que a veces puede obtener un número arbitrario de decimales en su respuesta:

```
>>> 0,2 + 0,1
0.30000000000000004
>>> 3 * 0,1
0.30000000000000004
```

Esto sucede en todos los idiomas y no es motivo de preocupación. Python intenta encontrar una manera de representar el resultado con la mayor precisión posible, lo que a veces resulta difícil dada la forma en que las computadoras tienen que representar los números internamente. Simplemente ignore los decimales adicionales por ahora; Aprenderá formas de ocuparse de los lugares adicionales cuando sea necesario en los proyectos de la Parte II.

Enteros y flotantes

Cuando divides dos números cualesquiera, incluso si son números enteros que dan como resultado un número entero, siempre obtendrás un flotante:

```
>>> 4/2
2.0
```

Si mezclas un número entero y un flotante en cualquier otra operación, también obtendrás un flotante:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python utiliza por defecto un flotante en cualquier operación que utilice un flotante, incluso si el resultado es un número entero.

Guiones bajos en números

Cuando escribe números largos, puede agrupar dígitos usando guiones bajos para que los números grandes sean más legibles:

```
>>> edad_universo = 14_000_000_000
```

Cuando imprime un número que se definió usando guiones bajos, Python imprime solo los dígitos:

```
>>> imprimir(edad_universo)
14000000000
```

Python ignora los guiones bajos al almacenar este tipo de valores. Incluso si no agrupa los dígitos de tres en tres, el valor no se verá afectado. Para Python, 1000 es lo mismo que 1_000, que es lo mismo que 10_00. Esta característica funciona tanto para números enteros como flotantes.

Asignación múltiple

Puede asignar valores a más de una variable usando una sola línea de código. Esto puede ayudar a acortar sus programas y hacerlos más fáciles de leer; Utilizará esta técnica con mayor frecuencia al inicializar un conjunto de números.

Por ejemplo, así es como puede inicializar las variables x, y y z a cero:

```
>>> x, y, z = 0, 0, 0
```

Debes separar los nombres de las variables con comas y hacer lo mismo con los valores, y Python asignará cada valor a su respectiva variable. Siempre que la cantidad de valores coincida con la cantidad de variables, Python los hará coincidir correctamente.

Constantes

Una constante es una variable cuyo valor permanece igual durante toda la vida de un programa. Python no tiene tipos constantes integrados, pero los programadores de Python usan todas las letras mayúsculas para indicar que una variable debe tratarse como una constante y nunca cambiarse:

```
MAX_CONEXIONES = 5000
```

Cuando desee tratar una variable como una constante en su código, escriba el nombre de la variable en letras mayúsculas.

INTÉNTALO TÚ MISMO

2-9. Número ocho: escriba operaciones de suma, resta, multiplicación y división que den como resultado el número 8. Asegúrese de encerrar sus operaciones en llamadas `print()` para ver los resultados. Deberías crear cuatro líneas que se vean así:

```
imprimir(5+3)
```

Su resultado debe ser de cuatro líneas, con el número 8 apareciendo una vez en cada línea.

2-10. Número favorito: utilice una variable para representar su número favorito. Luego, usando esa variable, crea un mensaje que revele tu número favorito. Imprime ese mensaje.

Comentarios

Los comentarios son una característica extremadamente útil en la mayoría de los lenguajes de programación. Todo lo que has escrito en tus programas hasta ahora es código Python. A medida que sus programas se vuelven más largos y complicados, debe agregar notas dentro de sus programas que describan su enfoque general del problema que está resolviendo. Un comentario le permite escribir notas en su idioma hablado, dentro de sus programas.

¿Cómo se escriben comentarios?

En Python, la almohadilla (`#`) indica un comentario. El intérprete de Python ignora todo lo que sigue a una marca hash en su código. Por ejemplo:

```
comment.py # Saluda a todos.  
print("¡Hola gente de Python!")
```

Python ignora la primera línea y ejecuta la segunda.

```
¡Hola gente de Python!
```

¿Qué tipo de comentarios debería escribir?

La razón principal para escribir comentarios es explicar qué se supone que debe hacer su código y cómo lo está haciendo funcionar. Cuando estás trabajando en un proyecto, comprendes cómo encajan todas las piezas.

Pero cuando regresa a un proyecto después de un tiempo fuera, probablemente tendrá

Olvidé algunos de los detalles. Siempre puedes estudiar tu código por un tiempo y descubrir cómo se supone que funcionan los segmentos, pero escribir buenos comentarios puede ahorrarte tiempo al resumir claramente tu enfoque general.

Si desea convertirse en un programador profesional o colaborar con otros programadores, debe escribir comentarios significativos. Hoy en día, la mayor parte del software se escribe de forma colaborativa, ya sea por un grupo de empleados de una empresa o por un grupo de personas que trabajan juntas en un proyecto de código abierto. Los programadores expertos esperan ver comentarios en el código, por lo que es mejor empezar a agregar comentarios descriptivos a sus programas ahora. Escribir comentarios claros y concisos en su código es uno de los hábitos más beneficiosos que puede formar como nuevo programador.

Cuando decida si escribir un comentario, pregúntese si tuvo que considerar varios enfoques antes de encontrar una forma razonable de hacer que algo funcione; Si es así, escribe un comentario sobre tu solución. Es mucho más fácil eliminar comentarios adicionales más tarde que volver atrás y escribir comentarios para un programa escasamente comentado. De ahora en adelante, usaré comentarios en ejemplos a lo largo de este libro para ayudar a explicar secciones de código.

INTÉNTALO TÚ MISMO

2-11. Agregar comentarios: elija dos de los programas que ha escrito y agregue al menos un comentario a cada uno. Si no tiene nada específico que escribir porque sus programas son demasiado simples en este momento, simplemente agregue su nombre y la fecha actual en la parte superior de cada archivo de programa. Luego escribe una oración que describa lo que hace el programa.

El zen de Python

Los programadores experimentados de Python lo alentarán a evitar la complejidad y buscar la simplicidad siempre que sea posible. La filosofía de la comunidad Python está contenida en “El Zen de Python” de Tim Peters. Puede acceder a este breve conjunto de principios para escribir un buen código Python ingresando `import this` en su intérprete. No reproduciré todo el “Zen de Python” aquí, pero compartiré algunas líneas para ayudarte a comprender por qué deberían ser importantes para ti como programador principiante de Python.

```
>>> importar esto
El Zen de Python, de Tim Peters
Lo bello es mejor que lo feo.
```

Los programadores de Python adoptan la noción de que el código puede ser bello y elegante. En programación, la gente resuelve problemas. Los programadores siempre han respetado las soluciones a los problemas bien diseñadas, eficientes e incluso hermosas. A medida que aprenda más sobre Python y lo utilice para escribir más código,

Alguien podría mirar por encima del hombro un día y decir: "¡Guau, qué código tan hermoso!".

Lo simple es mejor que lo complejo.

Si puede elegir entre una solución simple y una compleja, y ambas funcionan, utilice la solución simple. Su código será más fácil de mantener y será más fácil para usted y para otros desarrollarlo más adelante.

Complejo es mejor que complicado.

La vida real es confusa y, a veces, una solución sencilla a un problema es inalcanzable. En ese caso, utilice la solución más sencilla que funcione.

La legibilidad cuenta.

Incluso cuando su código sea complejo, intente hacerlo legible. Cuando estas Si trabaja en un proyecto que implica codificación compleja, concéntrese en escribir comentarios informativos para ese código.

Debería haber una (y preferiblemente sólo una) forma obvia de hacerlo.

Si se pide a dos programadores de Python que resuelvan el mismo problema, deberían encontrar soluciones bastante compatibles. Esto no quiere decir que no haya lugar para la creatividad en la programación. Al contrario, ¡hay mucho espacio para la creatividad! Sin embargo, gran parte de la programación consiste en utilizar enfoques pequeños y comunes para situaciones simples dentro de un proyecto más grande y creativo. Los aspectos prácticos de sus programas deberían tener sentido para otros programadores de Python.

Ahora es mejor que nunca.

Podrías pasar el resto de tu vida aprendiendo todas las complejidades de Python y de la programación en general, pero nunca completarías ningún proyecto. No intentes escribir código perfecto; escriba código que funcione y luego decida si mejorar su código para ese proyecto o pasar a algo nuevo.

A medida que continúe con el siguiente capítulo y comience a profundizar en temas más complicados, intente tener en mente esta filosofía de simplicidad y claridad. Los programadores experimentados respetarán más su código y estarán felices de brindarle comentarios y colaborar con usted en proyectos interesantes.

INTÉNTALO TÚ MISMO

2-12. Zen de Python: ingrese , importe esto en una sesión de terminal de Python y lea los principios adicionales.

Resumen

En este capítulo aprendiste cómo trabajar con variables. Aprendió a utilizar nombres de variables descriptivos y a resolver errores de nombre y de sintaxis cuando surgen. Aprendió qué son las cadenas y cómo mostrarlas usando minúsculas, mayúsculas y títulos. Comenzó a utilizar espacios en blanco para organizar la salida de forma ordenada y aprendió a eliminar elementos innecesarios de una cadena. Comenzó a trabajar con números enteros y flotantes y aprendió algunas de las formas en que puede trabajar con datos numéricos. También aprendió a escribir comentarios explicativos para que su código sea más fácil de leer para usted y otros. Finalmente, lees sobre la filosofía de mantener tu código lo más simple posible, siempre que sea posible.

En el Capítulo 3, aprenderá cómo almacenar colecciones de información en estructuras de datos llamadas listas. También aprenderá cómo trabajar con una lista, manipulando cualquier información en esa lista.

3

PRESENTANDO LISTAS



En este capítulo y el siguiente aprenderá qué son las listas y cómo empezar a trabajar con los elementos de una lista. Las listas le permiten almacenar conjuntos de información en un solo lugar, ya sea que tenga solo unos pocos elementos o millones de elementos. Las listas son una de las características más poderosas de Python, a las que los nuevos programadores pueden acceder fácilmente y reúnen muchos conceptos importantes en programación.

¿Qué es una lista?

Una lista es una colección de elementos en un orden particular. Puedes hacer una lista que incluya las letras del alfabeto, los dígitos del 0 al 9 o los nombres de todas las personas de tu familia. Puede poner lo que quiera en una lista y los elementos de su lista no tienen que estar relacionados de ninguna manera en particular. Porque

Una lista normalmente contiene más de un elemento, es una buena idea hacer que el nombre de la lista sea plural, como letras, dígitos o nombres.

En Python, los corchetes ([]) indican una lista y los elementos individuales de la lista están separados por comas. A continuación se muestra un ejemplo sencillo de una lista que contiene algunos tipos de bicicletas:

```
bicicletas.py bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
imprimir(bicicletas)
```

Si le pide a Python que imprima una lista, Python devuelve su representación de la lista, incluidos los corchetes:

```
['trek', 'cannondale', 'redline', 'especializado']
```

Debido a que este no es el resultado que desea que vean sus usuarios, aprendamos cómo para acceder a los elementos individuales de una lista.

Acceder a elementos en una lista

Las listas son colecciones ordenadas, por lo que puedes acceder a cualquier elemento de una lista diciéndole a Python la posición o índice del elemento deseado. Para acceder a un elemento de una lista, escriba el nombre de la lista seguido del índice del elemento entre corchetes.

Por ejemplo, saquemos la primera bicicleta de la lista de bicicletas:

```
bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
imprimir(bicicletas[0])
```

Cuando solicitamos un solo elemento de una lista, Python devuelve solo ese elemento sin corchetes:

```
emigrar
```

Este es el resultado que desea que vean sus usuarios: resultados limpios y bien formateados.

También puede utilizar los métodos de cadena del Capítulo 2 en cualquier elemento de esta lista. Por ejemplo, puedes formatear el elemento 'trek' para que parezca más presentable usando el método `title()`:

```
bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
imprimir(bicicletas[0].title())
```

Este ejemplo produce el mismo resultado que el ejemplo anterior, excepto que 'Trek' está en mayúscula.

Las posiciones del índice comienzan en 0, no en 1

Python considera que el primer elemento de una lista está en la posición 0, no en la posición 1. Esto es cierto para la mayoría de los lenguajes de programación, y la razón tiene que ver con

cómo se implementan las operaciones de lista en un nivel inferior. Si recibe resultados inesperados, pregúntese si está cometiendo un error simple pero común.

El segundo elemento de una lista tiene un índice de 1. Con este sistema de conteo, puede obtener cualquier elemento que desee de una lista restando uno de su posición en la lista. Por ejemplo, para acceder al cuarto elemento de una lista, solicita el elemento en el índice 3.

Lo siguiente solicita las bicicletas del índice 1 y del índice 3:

```
bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
imprimir(bicicletas[1])
imprimir(bicicletas[3])
```

Este código devuelve la segunda y cuarta bicicletas de la lista:

```
canondale
especializado
```

Python tiene una sintaxis especial para acceder al último elemento de una lista. Si usted solicita el elemento en el índice -1, Python siempre devuelve el último elemento de la lista:

```
bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
imprimir(bicicletas[-1])
```

Este código devuelve el valor 'especializado'. Esta sintaxis es bastante útil porque a menudo querrás acceder a los últimos elementos de una lista sin saber exactamente qué tan larga es la lista. Esta convención se extiende también a otros valores de índice negativos. El índice -2 devuelve el segundo elemento desde el final de la lista, el índice -3 devuelve el tercer elemento desde el final, y así sucesivamente.

Usar valores individuales de una lista

Puede utilizar valores individuales de una lista tal como lo haría con cualquier otra variable. Por ejemplo, puede utilizar cadenas f para crear un mensaje basado en un valor de una lista.

Intentemos sacar la primera bicicleta de la lista y redactar un mensaje usando ese valor:

```
bicicletas = ['trek', 'cannondale', 'redline', 'especializado']
mensaje = f"Mi primera bicicleta fue una {bicicletas[0].title()}."

imprimir (mensaje)
```

Construimos una oración usando el valor en `bicicletas[0]` y la asignamos a la variable `mensaje`. El resultado es una oración simple sobre la primera bicicleta de la lista:

```
Mi primera bicicleta fue una Trek.
```

INTÉNTALO TÚ MISMO

Pruebe estos programas breves para obtener experiencia de primera mano con las listas de Python.

Es posible que desees crear una nueva carpeta para los ejercicios de cada capítulo, para mantenerlos organizados.

3-1. Nombres: almacene los nombres de algunos de sus amigos en una lista llamada nombres. Imprima el nombre de cada persona accediendo a cada elemento de la lista, uno a la vez.

3-2. Saludos: comience con la lista que utilizó en el ejercicio 3-1, pero en lugar de simplemente imprimir el nombre de cada persona, imprima un mensaje para ellos. El texto de cada mensaje debe ser el mismo, pero cada mensaje debe estar personalizado con el nombre de la persona.

3-3. Tu propia lista: piensa en tu medio de transporte favorito, como una motocicleta o un automóvil, y haz una lista que contenga varios ejemplos. Utilice su lista para imprimir una serie de declaraciones sobre estos elementos, como "Me gustaría tener una motocicleta Honda".

Modificar, agregar y eliminar elementos

La mayoría de las listas que cree serán dinámicas, lo que significa que creará una lista y luego agregará y eliminará elementos a medida que su programa siga su curso. Por ejemplo, podrías crear un juego en el que un jugador tenga que disparar a extraterrestres desde el cielo. Podrías almacenar el conjunto inicial de extraterrestres en una lista y luego eliminar un extraterrestre de la lista cada vez que uno sea derribado. Cada vez que aparece un nuevo alienígena en la pantalla, lo agregas a la lista. Tu lista de alienígenas aumentará y disminuirá a lo largo del juego.

Modificar elementos en una lista

La sintaxis para modificar un elemento es similar a la sintaxis para acceder a un elemento en una lista. Para cambiar un elemento, use el nombre de la lista seguido del índice del elemento que desea cambiar y luego proporcione el nuevo valor que desea que tenga ese elemento.

Por ejemplo, digamos que tenemos una lista de motocicletas y el primer elemento de la lista es "honda". Podemos cambiar el valor de este primer elemento una vez creada la lista:

```
motos.py motos = ['honda', 'yamaha', 'suzuki']
            imprimir (motocicletas)

motos[0] = 'ducati'
imprimir (motocicletas)
```

Aquí definimos la lista de motocicletas, con 'honda' como primer elemento. Luego cambiamos el valor del primer elemento a 'ducati'. El resultado muestra que el primer elemento ha sido cambiado, mientras que el resto de la lista permanece igual:

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

Puede cambiar el valor de cualquier elemento de una lista, no solo el primer elemento.

Agregar elementos a una lista

Es posible que desee agregar un nuevo elemento a una lista por muchas razones. Por ejemplo, es posible que desee hacer que aparezcan nuevos extraterrestres en un juego, agregar nuevos datos a una visualización o agregar nuevos usuarios registrados a un sitio web que hayas creado. Python proporciona varias formas de agregar nuevos datos a listas existentes.

Agregar elementos al final de una lista

La forma más sencilla de agregar un nuevo elemento a una lista es agregar el elemento a la lista. Cuando agrega un elemento a una lista, el nuevo elemento se agrega al final de la lista. Usando la misma lista que teníamos en el ejemplo anterior, agregaremos el nuevo elemento 'ducati' al final de la lista:

```
motocicletas = ['honda', 'yamaha', 'suzuki']
imprimir (motocicletas)

motocicletas.append('ducati')
imprimir (motocicletas)
```

Aquí el método `append()` agrega 'ducati' al final de la lista, sin afectar ninguno de los otros elementos de la lista:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

El método `append()` facilita la creación de listas dinámicamente. Por ejemplo, puede comenzar con una lista vacía y luego agregar elementos a la lista usando una serie de llamadas `append()`. Usando una lista vacía, agreguemos los elementos 'honda', 'yamaha' y 'suzuki' a la lista:

```
motocicletas = []

motocicletas.append('honda')
motocicletas.append('yamaha')
motocicletas.append('suzuki')

imprimir (motocicletas)
```

La lista resultante tiene exactamente el mismo aspecto que las listas de los ejemplos anteriores:

```
['honda', 'yamaha', 'suzuki']
```

Crear listas de esta manera es muy común, porque a menudo no sabrá los datos que sus usuarios desean almacenar en un programa hasta que el programa se esté ejecutando. Para que sus usuarios tengan el control, comience definiendo una lista vacía que contendrá los valores de los usuarios. Luego agregue cada nuevo valor proporcionado a la lista que acaba de crear.

Insertar elementos en una lista

Puede agregar un nuevo elemento en cualquier posición de su lista usando `insert()` método. Para ello, especifique el índice del nuevo elemento y el valor del nuevo elemento:

```
motocicletas = ['honda', 'yamaha', 'suzuki']
```

```
motos.insert(0, 'ducati')
```

```
imprimir (motocicletas)
```

En este ejemplo, insertamos el valor 'ducati' al principio de la lista. El método `insert()` abre un espacio en la posición 0 y almacena el valor 'ducati' en esa ubicación:

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

Esta operación desplaza cada dos valores de la lista una posición hacia la derecha.

Eliminar elementos de una lista

A menudo, querrás eliminar un elemento o un conjunto de elementos de una lista. Por ejemplo, cuando un jugador derriba a un alienígena del cielo, lo más probable es que quieras eliminarlo de la lista de alienígenas activos. O cuando un usuario decide cancelar su cuenta en una aplicación web que usted creó, querrá eliminar a ese usuario de la lista de usuarios activos. Puede eliminar un elemento según su posición en la lista o según su valor.

Eliminar un artículo usando la declaración del

Si conoce la posición del elemento que desea eliminar de una lista, puede utilizar la declaración del :

```
motocicletas = ['honda', 'yamaha', 'suzuki']
```

```
imprimir (motocicletas)
```

```
de motos[0]
```

```
imprimir (motocicletas)
```

Aquí usamos la declaración `del` para eliminar el primer elemento, 'honda', de la lista de motocicletas:

```
['honda', 'yamaha', 'suzuki']  
['yamaha', 'suzuki']
```

Puede eliminar un elemento de cualquier posición en una lista usando la declaración del si conoce su índice. Por ejemplo, aquí se explica cómo eliminar el segundo elemento, "yamaha", de la lista:

```
motocicletas = ['honda', 'yamaha', 'suzuki']
imprimir (motocicletas)
```

```
de motos[1]
imprimir (motocicletas)
```

Se elimina de la lista la segunda motocicleta:

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

En ambos ejemplos, ya no puede acceder al valor que se eliminó de la lista después de usar la declaración del .

Eliminar un elemento utilizando el método pop()

A veces querrás utilizar el valor de un elemento después de eliminarlo de una lista. Por ejemplo, es posible que desees obtener la posición xey de un extraterrestre que acaba de ser derribado, para poder dibujar una explosión en esa posición. En una aplicación web, es posible que desee eliminar un usuario de una lista de miembros activos y luego agregarlo a una lista de miembros inactivos.

El método pop() elimina el último elemento de una lista, pero le permite trabajar con ese elemento después de eliminarlo. El término pop surge de pensar en una lista como una pila de elementos y sacar un elemento de la parte superior de la pila. En esta analogía, la parte superior de una pila corresponde al final de una lista.

Saquemos una motocicleta de la lista de motocicletas:

```
1 motocicletas = ['honda', 'yamaha', 'suzuki']
  imprimir (motocicletas)
```

```
2 popped_motorcycle = motocicletas.pop()
3 imprimir (motocicletas)
4 imprimir (motocicleta estallada)
```

Comenzamos definiendo e imprimiendo la lista de motocicletas 1. Luego sacamos un valor de la lista y asignamos ese valor a la variable popped_motorcycle 2. Imprimimos la lista 3 para mostrar que se ha eliminado un valor de la lista. Luego imprimimos el valor 4 extraído para demostrar que todavía tenemos acceso al valor que se eliminó.

El resultado muestra que el valor 'suzuki' se eliminó del final de la lista y ahora está asignada a la variable popped_motorcycle:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
Suzuki
```

¿Cómo podría ser útil este método `pop()` ? Imaginemos que las motos de la lista se almacenan en orden cronológico, según cuándo las tuvimos. Si este es el caso, podemos usar el método `pop()` para imprimir una declaración sobre la última motocicleta que compramos:

```
motocicletas = ['honda', 'yamaha', 'suzuki']
```

```
last_owned = motocicletas.pop()
print(f"La última motocicleta que tuve fue una {last_owned.title()}")
```

El resultado es una frase simple sobre la motocicleta más reciente que tuvimos:

La última moto que tuve fue una Suzuki.

Extraer elementos de cualquier posición en una lista

Puedes usar `pop()` para eliminar un elemento de cualquier posición en una lista incluyendo el índice del elemento que deseas eliminar entre paréntesis:

```
motocicletas = ['honda', 'yamaha', 'suzuki']
```

```
primero_propiedad = motocicletas.pop(0)
print(f"La primera motocicleta que tuve fue una {first_owned.title()}")
```

Comenzamos abriendo la primera motocicleta de la lista y luego imprimimos un mensaje sobre esa motocicleta. El resultado es una frase sencilla que describe la primera motocicleta que tuve:

La primera motocicleta que tuve fue una Honda.

Recuerda que cada vez que usas `pop()`, el elemento con el que trabajas no es ya no se almacenan en la lista.

Si no estás seguro de si utilizar la instrucción `del` o el método `pop()` , aquí tiene una forma sencilla de decidir: cuando desee eliminar un elemento de una lista y no utilizar ese elemento de ninguna manera, utilice la instrucción `del` ; Si desea utilizar un elemento mientras lo elimina, utilice el método `pop()` .

Eliminar un artículo por valor

A veces no sabrás la posición del valor que deseas eliminar de una lista. Si solo conoce el valor del elemento que desea eliminar, puede utilizar el método `remove()` .

Por ejemplo, digamos que queremos eliminar el valor 'ducati' de la lista de motocicletas:

```
motocicletas = ['honda', 'yamaha', 'suzuki', 'ducati']
imprimir (motocicletas)
```

```
motocicletas.remove('ducati')
imprimir (motocicletas)
```

Aquí el método `remove()` le dice a Python que averigüe dónde está 'ducati' aparece en la lista y elimina ese elemento:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

También puede utilizar el método `remove()` para trabajar con un valor que se elimina de una lista. Eliminemos el valor 'ducati' e imprimamos un motivo para eliminarlo de la lista:

```
1 motos = ['honda', 'yamaha', 'suzuki', 'ducati']
  imprimir (motocicletas)

2 demasiado_caro = 'ducati'
3 motocicletas.eliminar(demasiado_caras)
  imprimir (motocicletas)
4 print(f"\nA {too_expensive.title()} es demasiado caro para mí.")
```

Después de definir la lista 1, asignamos el valor 'ducati' a una variable llamada `too_expensive` 2. Luego usamos esta variable para decirle a Python qué valor eliminar de la lista 3. El valor 'ducati' se eliminó de la lista 4 pero todavía está accesible a través de la variable `too_expensive`, lo que nos permite imprimir una declaración sobre por qué eliminamos 'ducati' de la lista de motocicletas:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

Una Ducati me sale demasiado cara.

NOTA El método `remove()` elimina solo la primera aparición del valor que especifica. Si existe la posibilidad de que el valor aparezca más de una vez en la lista, deberá utilizar un bucle para asegurarse de que se eliminen todas las apariciones del valor. Aprenderá cómo hacer esto en el Capítulo 7.

INTÉNTALO TÚ MISMO

Los siguientes ejercicios son un poco más complejos que los del Capítulo 2, pero le brindan la oportunidad de utilizar listas en todas las formas descritas.

3-4. Lista de invitados: Si pudieras invitar a cenar a alguien, vivo o fallecido, ¿a quién invitarías? Haz una lista que incluya al menos tres personas a las que te gustaría invitar a cenar. Luego use su lista para imprimir un mensaje para cada persona, invitándolas a cenar.

(continuado)

3-5. Cambio de lista de invitados: Acabas de enterarte de que uno de tus invitados no puede asistir a la cena, por lo que debes enviar un nuevo conjunto de invitaciones. Tendrás que pensar en alguien más a quien invitar.

- Comience con su programa del Ejercicio 3-4. Agregue una llamada `print()` al final de su programa, indicando el nombre del invitado que no puede asistir.
- Modifica tu lista, reemplazando el nombre del huésped que no puede asistir con el nombre de la nueva persona que estás invitando.
- Imprima un segundo conjunto de mensajes de invitación, uno para cada persona que aún esté en tu lista.

3-6. Más invitados: acaba de encontrar una mesa de comedor más grande, por lo que ahora hay más espacio disponible. Piensa en tres invitados más para invitar a cenar.

- Comience con su programa del Ejercicio 3-4 o 3-5. Agregue una llamada `print()` al final de su programa, informando a las personas que encontró una tabla más grande.
- Utilice `insert()` para agregar un nuevo invitado al principio de su lista.
- Utilice `insert()` para agregar un nuevo invitado al medio de su lista.
- Utilice `append()` para agregar un nuevo invitado al final de su lista.
- Imprima un nuevo conjunto de mensajes de invitación, uno para cada persona de su lista.

3-7. Lista de invitados cada vez más reducida: acaba de descubrir que su nueva mesa no llegará a tiempo para la cena y ahora tiene espacio para solo dos invitados.

- Comience con su programa del Ejercicio 3-6. Agregue una nueva línea que imprima un mensaje que indique que solo puede invitar a dos personas a cenar.
- Utilice `pop()` para eliminar invitados de su lista uno a la vez hasta que solo queden dos nombres en su lista. Cada vez que saque un nombre de su lista, imprima un mensaje para esa persona haciéndole saber que lamenta no poder invitarla a cenar.
- Imprima un mensaje para cada una de las dos personas que aún están en su lista, informándoles que todavía están invitadas.
- Utilice `del` para eliminar los dos últimos nombres de su lista, de modo que tenga una lista vacía. Imprima su lista para asegurarse de tener una lista vacía al final de su programa.

Organizar una lista

A menudo, sus listas se crearán en un orden impredecible, porque no siempre puede controlar el orden en el que sus usuarios proporcionan sus datos. Aunque esto es inevitable en la mayoría de las circunstancias, con frecuencia querrás presentar tu información en un orden particular. A veces querrás

para preservar el orden original de su lista, y otras veces querrá cambiar el orden original. Python proporciona varias formas diferentes de organizar sus listas, según la situación.

Ordenar una lista de forma permanente con el método sort()

El método sort() de Python hace que sea relativamente fácil ordenar una lista. Imaginemos que tenemos una lista de automóviles y queremos cambiar el orden de la lista para almacenarlos alfabéticamente. Para simplificar la tarea, supongamos que todos los valores de la lista están en minúsculas:

```
coches.py coches = ['bmw', 'audi', 'toyota', 'subaru']
autos.sort()
imprimir (coches)
```

El método sort() cambia el orden de la lista de forma permanente. Los coches ahora están en orden alfabético y nunca podremos volver al orden original:

```
['audi', 'bmw', 'subaru', 'toyota']
```

También puede ordenar esta lista en orden alfabético inverso pasando el argumento reverse=True al método sort(). El siguiente ejemplo ordena la lista de automóviles en orden alfabético inverso:

```
coches = ['bmw', 'audi', 'toyota', 'subaru']
autos.sort(reverse=Verdadero)
imprimir (coches)
```

Nuevamente, el orden de la lista cambia permanentemente:

```
['toyota', 'subaru', 'bmw', 'audi']
```

Ordenar una lista temporalmente con la función sorted()

Para mantener el orden original de una lista pero presentarla ordenada, puede utilizar la función sorted(). La función sorted() le permite mostrar su lista en un orden particular, pero no afecta el orden real de la lista.

Probemos esta función en la lista de coches.

```
coches = ['bmw', 'audi', 'toyota', 'subaru']

1 print("Aquí está la lista original:")
  imprimir (coches)

2 print("\nAquí está la lista ordenada:")
  imprimir (ordenado (autos))

3 print("\nAquí está nuevamente la lista original:")
  imprimir (coches)
```

Primero imprimimos la lista en su orden original 1 y luego en orden alfabético 2. Después de que la lista se muestra en el nuevo orden, mostramos que la lista todavía está almacenada en su orden original 3:

Aquí está la lista original:

```
['bmw', 'audi', 'toyota', 'subaru']
```

Aquí está la lista ordenada:

```
['audi', 'bmw', 'subaru', 'toyota']
```

1 Aquí está nuevamente la lista original:

```
['bmw', 'audi', 'toyota', 'subaru']
```

Observe que la lista todavía existe en su orden original 1 después de `sorted()`

Se ha utilizado la función. La función `sorted()` también puede aceptar un valor `inverso=True` argumento si desea mostrar una lista en orden alfabético inverso.

NOTA Ordenar una lista alfabéticamente es un poco más complicado cuando no todos los valores están en minúsculas. Hay varias formas de interpretar las letras mayúsculas al determinar un orden de clasificación, y especificar el orden exacto puede ser más complejo de lo que queremos abordar en este momento. Sin embargo, la mayoría de los métodos de clasificación se basarán directamente en lo que aprendió en esta sección.

Imprimir una lista en orden inverso

Para invertir el orden original de una lista, puede utilizar el método `reverse()`. Si originalmente almacenábamos la lista de automóviles en orden cronológico según cuándo los teníamos, podríamos fácilmente reorganizar la lista en orden cronológico inverso:

```
coches = ['bmw', 'audi', 'toyota', 'subaru']
```

```
imprimir (coches)
```

```
coches.reverse()
```

```
imprimir (coches)
```

Observe que `reverse()` no ordena alfabéticamente hacia atrás; simplemente invierte el orden de la lista:

```
['bmw', 'audi', 'toyota', 'subaru']
```

```
['subaru', 'toyota', 'audi', 'bmw']
```

El método `inverso()` cambia el orden de una lista de forma permanente, pero puedes volver al orden original en cualquier momento aplicando `inverso()` a la misma lista por segunda vez.

Encontrar la longitud de una lista

Puede encontrar rápidamente la longitud de una lista utilizando la función `len()`. La lista de este ejemplo tiene cuatro elementos, por lo que su longitud es 4:

```
>>> coches = ['bmw', 'audi', 'toyota', 'subaru']
```

```
>>> len(coches)
```

```
4
```

Encontrarás que `len()` es útil cuando necesites identificar la cantidad de extraterrestres que aún faltan por derribar en un juego, determinar la cantidad de datos que tienes que gestionar en una visualización, o calcular el número de usuarios registrados en un sitio web, entre otras tareas.

NOTA Python cuenta los elementos en una lista que comienza con uno, por lo que no debería encontrarse con ningún error uno por uno al determinar la longitud de una lista.

INTÉNTALO TÚ MISMO

3-8. Ver el mundo: piensa en al menos cinco lugares del mundo que te gustaría visitar.

- Almacenar las ubicaciones en una lista. Asegúrese de que la lista no esté en orden alfabético.
- Imprima su lista en su orden original. No se preocupe por imprimir la lista de forma ordenada; simplemente imprímalo como una lista de Python sin formato.
- Utilice `sorted()` para imprimir su lista en orden alfabético sin modificar el lista real.
- Demuestre que su lista todavía está en su orden original imprimiéndola.
- Utilice `sorted()` para imprimir su lista en orden alfabético inverso sin cambiar el orden de la lista original.
- Demuestre que su lista todavía está en su orden original imprimiéndola nuevamente.
- Utilice `inversa()` para cambiar el orden de su lista. Imprima la lista para mostrar que su orden ha cambiado.
- Utilice `inversa()` para cambiar el orden de su lista nuevamente. Imprima la lista para mostrar que ha vuelto a su orden original.
- Utilice `sort()` para cambiar su lista para que se almacene en orden alfabético. Imprima la lista para mostrar que se ha cambiado su orden.
- Utilice `sort()` para cambiar su lista para que se almacene en orden alfabético inverso. Imprima la lista para mostrar que su orden ha cambiado.

3-9. Invitados a cenar: trabajando con uno de los programas de los ejercicios 3-4 al 3-7 (páginas 41–42), use `len()` para imprimir un mensaje que indique la cantidad de personas que está invitando a cenar.

3-10. Cada función: piensa en cosas que podrías almacenar en una lista. Por ejemplo, puedes hacer una lista de montañas, ríos, países, ciudades, idiomas o cualquier otra cosa que desees. Escriba un programa que cree una lista que contenga estos elementos y luego use cada función presentada en este capítulo al menos una vez.

Evitar errores de índice al trabajar con listas

Hay un tipo de error que es común ver cuando trabajas con listas por primera vez. Supongamos que tiene una lista con tres elementos y solicita el cuarto elemento:

```
motos.py motos = ['honda', 'yamaha', 'suzuki']
imprimir(motocicletas[3])
```

Este ejemplo da como resultado un error de índice:

Rastreo (llamadas recientes más última):
 Archivo "motorcycles.py", línea 2, en <módulo>
 imprimir(motocicletas[3])
 ~~~~~^~~~~~

IndexError: índice de lista fuera de rango

---

Python intenta proporcionarle el elemento en el índice 3. Pero cuando busca en la lista, ningún elemento en motocicletas tiene un índice de 3. Debido a la naturaleza de la indexación uno por uno en las listas, este error es típico. La gente piensa que el tercer elemento es el número 3 porque empiezan a contar en 1. Pero en Python el tercer elemento es el número 2 porque empieza a indexar en 0.

Un error de índice significa que Python no puede encontrar un elemento en el índice que solicitó. Si ocurre un error de índice en su programa, intente ajustar el índice que está solicitando en uno. Luego ejecute el programa nuevamente para ver si los resultados son correctos.

Tenga en cuenta que siempre que desee acceder al último elemento de una lista, debería usar el índice -1. Esto siempre funcionará, incluso si su lista ha cambiado de tamaño desde la última vez que accedió a ella:

---

```
motocicletas = ['honda', 'yamaha', 'suzuki']
imprimir(motocicletas[-1])
```

---

El índice -1 siempre devuelve el último elemento de una lista, en este caso el valor 'suzuki':

---

Suzuki

---

La única vez que este enfoque causará un error es cuando solicite el último elemento de una lista vacía:

---

```
motocicletas = []
imprimir(motocicletas[-1])
```

---

No hay artículos en motocicletas, por lo que Python devuelve otro error de índice:

---

Rastreo (llamadas recientes más última):  
 Archivo "motorcycles.py", línea 3, en <módulo>  
 imprimir(motocicletas[-1])  
 ~~~~~^~~~~~

IndexError: índice de lista fuera de rango

Si ocurre un error de índice y no sabe cómo resolverlo, intente imprimir su lista o simplemente imprima la longitud de su lista. Su lista puede verse muy diferente de lo que pensaba, especialmente si su programa la ha administrado dinámicamente. Ver la lista real, o el número exacto de elementos de su lista, puede ayudarle a solucionar estos errores lógicos.

INTÉNTALO TÚ MISMO

3-11. Error intencional: si aún no ha recibido un error de índice en uno de sus programas, intente que suceda. Cambie un índice en uno de sus programas para producir un error de índice. Asegúrese de corregir el error antes de cerrar el programa.

Resumen

En este capítulo, aprendió qué son las listas y cómo trabajar con los elementos individuales de una lista. Aprendiste cómo definir una lista y cómo agregar y eliminar elementos. Aprendió a ordenar listas de forma permanente y temporal para fines de visualización. También aprendió cómo encontrar la longitud de una lista y cómo evitar errores de índice cuando trabaja con listas.

En el Capítulo 4 aprenderá cómo trabajar con elementos de una lista de manera más eficiente. Al recorrer cada elemento de una lista utilizando solo unas pocas líneas de código, podrá trabajar de manera eficiente, incluso cuando su lista contenga miles o millones de elementos.

4

TRABAJANDO CON LISTAS



En el Capítulo 3 aprendiste cómo hacer una lista simple y aprendiste a trabajar con los elementos individuales de una lista. En este capítulo aprenderá cómo recorrer una lista completa usando solo unas pocas líneas de código, sin importar cuán larga sea la lista. El bucle le permite realizar la misma acción, o conjunto de acciones, con cada elemento de una lista. Como resultado, podrá trabajar de manera eficiente con listas de cualquier longitud, incluidas aquellas con miles o incluso millones de elementos.

Recorriendo una lista completa

A menudo querrás revisar todas las entradas de una lista y realizar la misma tarea con cada elemento. Por ejemplo, en un juego es posible que desees mover todos los elementos de la pantalla en la misma cantidad. En una lista de números, es posible que desees realizar la misma operación estadística en cada elemento.

O quizás quieras mostrar cada título de una lista de artículos en un sitio web. Cuando desee realizar la misma acción con cada elemento de una lista, puede utilizar el bucle for de Python .

Digamos que tenemos una lista de nombres de magos y queremos imprimir cada nombre de la lista. Podríamos hacer esto recuperando cada nombre de la lista individualmente, pero este enfoque podría causar varios problemas. Por un lado, sería repetitivo hacer esto con una larga lista de nombres. Además, tendríamos que cambiar nuestro código cada vez que cambiara la longitud de la lista. El uso de un bucle for evita ambos problemas al permitir que Python los administre internamente.

Usemos un bucle for para imprimir cada nombre en una lista de magos:

```
magos.py magos = ['alicia', 'david', 'carolina']
para mago en magos:
    imprimir (mago)
```

Comenzamos definiendo una lista, tal como lo hicimos en el Capítulo 3. Luego definimos un bucle for . Esta línea le dice a Python que extraiga un nombre de la lista de magos y lo asocie con la variable mago. A continuación, le decimos a Python que imprima el nombre que acaba de asignarse al mago. Luego, Python repite estas dos últimas líneas, una vez para cada nombre de la lista. Podría ser útil leer este código como "Para cada mago en la lista de magos, imprima el nombre del mago".

El resultado es una impresión simple de cada nombre en la lista:

```
Alicia
David
carolina
```

Una mirada más cercana al bucle

El bucle es importante porque es una de las formas más comunes en que una computadora automatiza tareas repetitivas. Por ejemplo, en un bucle simple como el que usamos en magicians.py, Python inicialmente lee la primera línea del bucle:

```
para mago en magos:
```

Esta línea le dice a Python que recupere el primer valor de la lista de magos . y asociarlo con la variable mago. Este primer valor es 'alicia'. Luego, Python lee la siguiente línea:

```
    imprimir (mago)
```

Python imprime el valor actual de mago, que sigue siendo "alicia". Porque la lista contiene más valores, Python regresa a la primera línea del bucle:

```
para mago en magos:
```

Python recupera el siguiente nombre de la lista, 'david', y lo asocia valor con la variable mago. Python luego ejecuta la línea:

```
    imprimir (mago)
```

Python vuelve a imprimir el valor actual de mago , que ahora es 'david'. Python repite todo el ciclo una vez más con el último valor de la lista, 'carolina'. Como no hay más valores en la lista, Python pasa a la siguiente línea del programa. En este caso, nada viene después del bucle for , por lo que el programa finaliza.

Cuando utilice bucles por primera vez, tenga en cuenta que el conjunto de pasos se repite una vez para cada elemento de la lista, sin importar cuántos elementos haya en la lista. Si tiene un millón de elementos en su lista, Python repite estos pasos un millón de veces y, por lo general, muy rápidamente.

También tenga en cuenta al escribir sus propios bucles for que puede elegir cualquier nombre que desee para la variable temporal que se asociará con cada valor de la lista. Sin embargo, resulta útil elegir un nombre significativo que represente un único elemento de la lista. Por ejemplo, esta es una buena manera de iniciar un bucle for para una lista de gatos, una lista de perros y una lista general de elementos:

```
para gato en gatos:
para perro en perros:
para el artículo en list_of_items:
```

Estas convenciones de nomenclatura pueden ayudarle a seguir la acción que se realiza en cada elemento dentro de un bucle for . El uso de nombres en singular y plural puede ayudarle a identificar si una sección de código funciona con un solo elemento de la lista o con la lista completa.

Hacer más trabajo dentro de un bucle for

Puedes hacer casi cualquier cosa con cada elemento en un bucle for . Apoyémonos en el ejemplo anterior imprimiendo un mensaje a cada mago, diciéndoles que realizaron un gran truco:

```
magos.py magos = ['alicia', 'david', 'carolina']
para mago en magos:
    print(f"{magician.title()}, ¡ese fue un gran truco!")
```

La única diferencia en este código es que redactamos un mensaje para cada mago, comenzando con el nombre de ese mago. La primera vez que se realiza el ciclo, el valor del mago es 'alicia', por lo que Python inicia el primer mensaje con el nombre 'Alice'. La segunda vez, el mensaje comenzará con 'David', y la tercera vez, el mensaje comenzará con 'Carolina'.

El resultado muestra un mensaje personalizado para cada mago de la lista:

```
Alice, ¡ese fue un gran truco!
¡David, ese fue un gran truco!
Carolina, ¡ese fue un gran truco!
```

También puedes escribir tantas líneas de código como quieras en el bucle for . Cada línea sangrada que sigue a la línea para mago en magos se considera dentro del bucle, y cada línea sangrada se ejecuta una vez para cada valor de la lista. Por lo tanto, puedes hacer todo el trabajo que quieras con cada valor de la lista.

Agreguemos una segunda línea a nuestro mensaje, diciéndole a cada mago que estamos esperando su próximo truco:

```
magos = ['alicia', 'david', 'carolina']
para mago en magos:
    print(f"{magician.title()}, ¡ese fue un gran truco!")
    print(f"No puedo esperar a ver tu próximo truco, {magician.title()}\n")
```

Debido a que hemos sangrado ambas llamadas a print(), cada línea se ejecutará una vez por cada mago de la lista. La nueva línea ("\n") en la segunda llamada print() inserta una línea en blanco después de cada paso por el bucle. Esto crea un conjunto de mensajes que están cuidadosamente agrupados para cada persona en la lista:

Alice, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡David, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, David.

Carolina, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.

Puedes usar tantas líneas como quieras en tus bucles for . En la práctica, a menudo le resultará útil realizar varias operaciones diferentes con cada elemento de una lista cuando utilice un bucle for .

Hacer algo después de un bucle for

¿Qué sucede una vez que un bucle for ha terminado de ejecutarse? Por lo general, querrás resumir un bloque de resultados o pasar a otro trabajo que tu programa debe realizar.

Cualquier línea de código después del bucle for que no tenga sangría se ejecuta una vez sin repetición. Escribamos un agradecimiento al grupo de magos en su conjunto, agradeciéndoles por ofrecer un excelente espectáculo. Para mostrar este mensaje grupal después de que se hayan impreso todos los mensajes individuales, colocamos el mensaje de agradecimiento después del bucle for , sin sangría:

```
magos = ['alicia', 'david', 'carolina']
para mago en magos:
    print(f"{magician.title()}, ¡ese fue un gran truco!")
    print(f"No puedo esperar a ver tu próximo truco, {magician.title()}\n")

print("Gracias a todos. ¡Fue un gran espectáculo de magia!")
```

Las dos primeras llamadas a print() se repiten una vez para cada mago en el lista, como viste antes. Sin embargo, como la última línea no tiene sangría, se imprime sólo una vez:

Alice, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡David, ese fue un gran truco!

No puedo esperar a ver tu próximo truco, David.

Carolina, ¡ese fue un gran truco!

No puedo esperar a ver tu próximo truco, Carolina.

Gracias a todos. ¡Ese fue un gran espectáculo de magia!

Cuando procesa datos usando un bucle `for`, encontrará que esta es una buena manera de resumir una operación que se realizó en un conjunto de datos completo. Por ejemplo, puedes usar un bucle `for` para inicializar un juego ejecutando una lista de personajes y mostrando cada personaje en la pantalla.

Luego, podrías escribir algún código adicional después de este bucle que muestre un botón Reproducir ahora después de que todos los personajes hayan sido dibujados en la pantalla.

Evitar errores de sangría

Python usa sangría para determinar cómo se relaciona una línea, o un grupo de líneas, con el resto del programa. En los ejemplos anteriores, las líneas que imprimían mensajes a magos individuales formaban parte del bucle `for` porque tenían sangría. El uso de sangría por parte de Python hace que el código sea muy fácil de leer.

Básicamente, utiliza espacios en blanco para obligarte a escribir código perfectamente formateado con una estructura visual clara. En programas Python más largos, notará bloques de código sangrados en algunos niveles diferentes. Estos niveles de sangría le ayudan a tener una idea general de la organización general del programa.

A medida que comience a escribir código que se base en una sangría adecuada, deberá estar atento a algunos errores de sangría comunes. Por ejemplo, a veces las personas aplican sangría a líneas de código que no necesitan sangría u olvidan sangrar líneas que sí necesitan sangría. Ver ejemplos de estos errores ahora le ayudará a evitarlos en el futuro y corregirlos cuando aparezcan en sus propios programas.

Examinemos algunos de los errores de sangría más comunes.

Olvidarse de sangrar

Siempre sangra la línea después de la declaración `for` en un bucle. Si lo olvidas, Python te lo recordará:

```
magos.py magos = ['alicia', 'david', 'carolina']
          para mago en magos:
            1 impresión (mago)
```

La llamada a `print()` 1 debería tener sangría, pero no lo es. Cuando Python espera un bloque sangrado y no lo encuentra, le permite saber con qué línea tuvo un problema:

```
Archivo "magos.py", línea 3
    imprimir (mago)
```

`IndentationError`: se esperaba un bloque con sangría después de la declaración `'for'` en la línea 2

Generalmente puedes resolver este tipo de error de sangría aplicando una sangría al línea o líneas inmediatamente después de la declaración `for`.

Olvidar sangrar líneas adicionales

A veces, su bucle se ejecutará sin errores pero no producirá el resultado esperado. Esto puede suceder cuando intentas realizar varias tareas en un bucle y olvidas sangrar algunas de sus líneas.

Por ejemplo, esto es lo que sucede cuando olvidamos sangrar la segunda línea en el bucle que le dice a cada mago que estamos esperando su próximo truco:

```
magos = ['alicia', 'david', 'carolina']
para mago en magos:
    print(f"{magician.title()}, ¡ese fue un gran truco!")
1 print(f"No puedo esperar a ver tu próximo truco, {magician.title()}.n")
```

Se supone que la segunda llamada a `print()` 1 tiene sangría, pero debido a que Python encuentra al menos una línea con sangría después de la declaración `for`, no informa un error. Como resultado, la primera llamada `print()` se ejecuta una vez para cada nombre en la lista porque tiene sangría. La segunda llamada a `print()` no tiene sangría, por lo que se ejecuta solo una vez después de que el ciclo ha terminado de ejecutarse. Debido a que el valor final asociado con el mago es 'carolina', ella es la única que recibe el mensaje "esperando el próximo truco":

```
Alice, ¡ese fue un gran truco!
¡David, ese fue un gran truco!
Carolina, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.
```

Este es un error lógico. La sintaxis es código Python válido, pero el código no produce el resultado deseado porque ocurre un problema en su lógica. Si espera ver una determinada acción repetida una vez para cada elemento de una lista y se ejecuta solo una vez, determine si necesita simplemente sangrar una línea o un grupo de líneas.

Sangrar innecesariamente

Si accidentalmente sangra una línea que no necesita sangría, Python le informa sobre la sangría inesperada:

```
hello_world.py mensaje = "¡Hola mundo Python!"
    imprimir (mensaje)
```

No necesitamos sangrar la llamada `print()`, porque no es parte de un bucle; por lo tanto, Python informa ese error:

```
Archivo "hello_world.py", línea 2
    imprimir (mensaje)
```

IndentationError: sangría inesperada

Puede evitar errores de sangría inesperados aplicando sangría sólo cuando tenga un motivo específico para hacerlo. En los programas que estás escribiendo en este momento, las únicas líneas que debes sangrar son las acciones que deseas repetir para cada elemento en un bucle `for`.

Sangrar innecesariamente después del bucle

Si accidentalmente sangra el código que debería ejecutarse después de que finalice un bucle, ese código se repetirá una vez para cada elemento de la lista. A veces esto hace que Python informe un error, pero a menudo esto dará como resultado una respuesta lógica.

error.

Por ejemplo, veamos qué sucede cuando accidentalmente sangramos la línea que agradeció a los magos como grupo por ofrecer un buen espectáculo:

```
magos.py magos = ['alicia', 'david', 'carolina']
    para mago en magos:
        print(f"{magician.title()}, ¡ese fue un gran truco!")
        print(f"No puedo esperar a ver tu próximo truco, {magician.title()}.")
1 print("¡Gracias a todos, fue un gran espectáculo de magia!")
```

Debido a que la última línea 1 tiene sangría, se imprime una vez para cada persona de la lista:

```
Alice, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡Gracias a todos, fue un gran espectáculo de magia!
¡David, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, David.

¡Gracias a todos, fue un gran espectáculo de magia!
Carolina, ¡ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.

¡Gracias a todos, fue un gran espectáculo de magia!
```

Este es otro error lógico, similar al de "Olvidar la sangría
Líneas adicionales" en la página 54. Como Python no sabe lo que intenta lograr con su código, ejecutará todo el código escrito con una sintaxis válida. Si una acción se repite muchas veces cuando debería ejecutarse solo una vez, probablemente necesite quitar la sangría del código para esa acción.

Olvidando el Colón

Los dos puntos al final de una declaración `for` le dicen a Python que interprete la siguiente línea como el comienzo de un bucle.

```
magos = ['alicia', 'david', 'carolina']
1 para mago en magos
    imprimir (mago)
```

Si olvida accidentalmente los dos puntos 1, obtendrá un error de sintaxis porque Python no sabe exactamente lo que estás intentando hacer:

```
Archivo "magos.py", línea 2
para mago en magos
```

^

Error de sintaxis: esperado ':'

Python no sabe si simplemente olvidó los dos puntos o si pretendía escribir código adicional para configurar un bucle más complejo. Si el intérprete puede identificar una posible solución, sugerirá una, como agregar dos puntos al final de una línea, como lo hace aquí con la respuesta esperada ':'. Algunos errores tienen soluciones fáciles y obvias, gracias a las sugerencias en los rastreos de Python. Algunos errores son mucho más difíciles de resolver, incluso cuando la solución final solo involucra un carácter. No se sienta mal si lleva mucho tiempo encontrar una pequeña solución; No estás en absoluto solo en esta experiencia.

INTÉNTALO TÚ MISMO

4-1. Pizzas: piensa en al menos tres tipos de tu pizza favorita. Guarde estos nombres de pizza en una lista y luego use un bucle for para imprimir el nombre de cada pizza.

- Modifica tu bucle for para imprimir una oración usando el nombre de la pizza, en lugar de imprimir solo el nombre de la pizza. Para cada pizza, debe tener una línea de salida que contenga una declaración simple como Me gusta la pizza de pepperoni.
- Añade una línea al final de tu programa, fuera del bucle for, que indique cuánto te gusta la pizza. El resultado debe consistir en tres o más líneas sobre los tipos de pizza que te gustan y luego una oración adicional, como ¡Realmente amo la pizza!

4-2. Animales: Piensa en al menos tres animales diferentes que tengan una característica común. Guarde los nombres de estos animales en una lista y luego use un bucle for para imprimir el nombre de cada animal.

- Modifique su programa para imprimir una declaración sobre cada animal, como por ejemplo Un perro sería una excelente mascota.
- Añade una línea al final de tu programa, indicando qué tienen estos animales en común. Podrías imprimir una oración como: ¡Cualquiera de estos animales sería una excelente mascota!

Hacer listas numéricas

Existen muchas razones para almacenar un conjunto de números. Por ejemplo, necesitarás realizar un seguimiento de las posiciones de cada personaje en un juego y quizás quieras

para realizar un seguimiento de las puntuaciones más altas de un jugador también. En las visualizaciones de datos, casi siempre trabajará con conjuntos de números, como temperaturas, distancias, tamaños de población o valores de latitud y longitud, entre otros tipos de conjuntos numéricos.

Las listas son ideales para almacenar conjuntos de números y Python proporciona una variedad de herramientas para ayudarlo a trabajar de manera eficiente con listas de números. Una vez que comprenda cómo utilizar estas herramientas de manera eficaz, su código funcionará bien incluso cuando sus listas contengan millones de elementos.

Usando la función range()

La función range() de Python facilita la generación de una serie de números.

Por ejemplo, puedes usar la función range() para imprimir una serie de números como este:

first_numbers.py para el valor en el rango (1, 5):

```
imprimir (valor)
```

Aunque parece que este código debería imprimir los números del 1 al 5, no imprime el número 5:

```
1
2
3
4
```

En este ejemplo, range() imprime solo los números del 1 al 4. Este es otro resultado del comportamiento uno por uno que verá a menudo en los lenguajes de programación. La función range() hace que Python comience a contar en el primer valor que le proporciona y se detiene cuando alcanza el segundo valor que le proporciona. Debido a que se detiene en ese segundo valor, la salida nunca contiene el valor final, que habría sido 5 en este caso.

Para imprimir los números del 1 al 5, usarías range(1, 6):

para valor en el rango (1, 6):

```
imprimir (valor)
```

Esta vez la salida comienza en 1 y termina en 5:

```
1
2
3
4
5
```

Si su resultado es diferente de lo que espera cuando usa range(), intente ajustar su valor final en 1.

También puede pasar a range() solo un argumento, y comenzará la secuencia de números en 0. Por ejemplo, range(6) devolvería los números del 0 al 5.

Usando range() para hacer una lista de números

Si desea hacer una lista de números, puede convertir los resultados de range() directamente en una lista usando la función list(). Cuando envuelve list() alrededor de una llamada a la función range(), la salida será una lista de números.

En el ejemplo de la sección anterior, simplemente imprimimos una serie de números. Podemos usar list() para convertir ese mismo conjunto de números en una lista:

```
números = lista(rango(1, 6))
imprimir (números)
```

Este es el resultado:

```
[1, 2, 3, 4, 5]
```

También podemos usar la función range() para decirle a Python que omita números en un rango determinado. Si pasa un tercer argumento a range(), Python usa ese valor como tamaño de paso al generar números.

Por ejemplo, aquí se explica cómo enumerar los números pares entre 1 y 10:

```
números_pares.py números_pares = lista(rango(2, 11, 2))
imprimir (números pares)
```

En este ejemplo, la función range() comienza con el valor 2 y luego suma 2 a ese valor. Agrega 2 repetidamente hasta que alcanza o pasa el valor final, 11, y produce este resultado:

```
[2, 4, 6, 8, 10]
```

Puedes crear casi cualquier conjunto de números que quieras usando el rango() función. Por ejemplo, considere cómo podría hacer una lista de los primeros 10 números cuadrados (es decir, el cuadrado de cada número entero del 1 al 10). En Python, dos asteriscos (**) representan exponentes. Así es como puedes poner los primeros 10 números cuadrados en una lista:

```
cuadrado
_números.py cuadrados = []
para valor en el rango (1, 11):
1 cuadrado = valor ** 2
2 cuadrados.append(cuadrado)

imprimir (cuadrados)
```

Comenzamos con una lista vacía llamada cuadrados. Luego, le decimos a Python que recorra cada valor del 1 al 10 usando la función range(). Dentro del ciclo, el valor actual se eleva a la segunda potencia y se asigna a la variable cuadrado 1. Luego, cada nuevo valor de cuadrado se agrega a la lista de cuadrados 2.

Finalmente, cuando el bucle ha terminado de ejecutarse, se imprime la lista de cuadrados:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Para escribir este código de manera más concisa, omita la variable temporal `cuadrado` y agregue cada nuevo valor directamente a la lista:

```
cuadrados = []
para valor en el rango (1,11):
    cuadrados.append(valor**2)
```

```
imprimir (cuadrados)
```

Esta línea hace el mismo trabajo que las líneas dentro del bucle `for` del listado anterior. Cada valor en el bucle se eleva a la segunda potencia y luego se agrega inmediatamente a la lista de `cuadrados`.

Puede utilizar cualquiera de estos enfoques cuando esté creando listas más complejas. A veces, el uso de una variable temporal hace que el código sea más fácil de leer; otras veces hace que el código sea innecesariamente largo. Concéntrese primero en escribir código que entienda claramente y que haga lo que usted quiere que haga. Luego busque enfoques más eficientes mientras revisa su código.

Estadísticas simples con una lista de números

Algunas funciones de Python son útiles cuando se trabaja con listas de números. Por ejemplo, puedes encontrar fácilmente el mínimo, el máximo y la suma de una lista de números:

```
>>> dígitos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(dígitos)
0
>>> máx(dígitos)
9
>>> suma(dígitos)
45
```

NOTA Los ejemplos de esta sección utilizan listas breves de números que caben fácilmente en la página. Funcionarían igual de bien si su lista contuviera un millón o más de números.

Lista de comprensiones

El enfoque descrito anteriormente para generar la lista de cuadrados consistió en utilizar tres o cuatro líneas de código. Una lista por comprensión le permite generar esta misma lista en una sola línea de código. Una lista por comprensión combina el bucle `for` y la creación de nuevos elementos en una línea, y agrega automáticamente cada elemento nuevo. Las listas por comprensión no siempre se presentan a los principiantes, pero las he incluido aquí porque lo más probable es que las veas tan pronto como empieces a mirar el código de otras personas.

El siguiente ejemplo crea la misma lista de números cuadrados que vio anteriormente pero usa una lista de comprensión:

```
squares.py squares = [valor**2 para valor en el rango(1, 11)]
imprimir (cuadrados)
```

Para utilizar esta sintaxis, comience con un nombre descriptivo para la lista, como cuadrados. A continuación, abra un conjunto de corchetes y defina la expresión para los valores que desea almacenar en la nueva lista. En este ejemplo la expresión es `valor**2`, lo que eleva el valor a la segunda potencia. Luego, escriba un bucle `for` para generar los números que desea introducir en la expresión y cierre los corchetes. El bucle `for` en este ejemplo es para el valor en el rango `(1, 11)`, que introduce los valores del 1 al 10 en la expresión `valor**2`. Tenga en cuenta que no se utilizan dos puntos al final de la declaración `for`.

El resultado es la misma lista de números cuadrados que viste antes:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Se necesita práctica para escribir sus propias listas por comprensión, pero encontrará que valen la pena una vez que se sienta cómodo creando listas ordinarias. Cuando esté escribiendo tres o cuatro líneas de código para generar listas y comience a parecer repetitivo, considere escribir sus propias listas por comprensión.

INTÉNTALO TÚ MISMO

4-3. Contando hasta veinte: utilice un bucle `for` para imprimir los números del 1 al 20, inclusive.

4-4. Un millón: haga una lista de los números del uno al un millón y luego use un bucle `for` para imprimir los números. (Si la salida tarda demasiado, deténgala presionando `CTRL-C` o cerrando la ventana de salida).

4-5. Sumar un millón: haga una lista de los números del uno al un millón y luego use `min()` y `max()` para asegurarse de que su lista realmente comience en uno y termine en un millón. Además, utilice la función `sum()` para ver qué tan rápido Python puede sumar un millón de números.

4-6. Números impares: utilice el tercer argumento de la función `range()` para hacer una lista de los números impares del 1 al 20. Utilice un bucle `for` para imprimir cada número.

4-7. Tres: haz una lista de los múltiplos de 3, del 3 al 30. Usa un bucle `for` para imprimir los números de tu lista.

4-8. Cubos: Un número elevado a la tercera potencia se llama cubo. Por ejemplo, el cubo de 2 se escribe como `2**3` en Python. Haga una lista de los primeros 10 cubos (es decir, el cubo de cada número entero del 1 al 10) y use un bucle `for` para imprimir el valor de cada cubo.

4-9. Comprensión de cubos: utilice una lista de comprensión para generar una lista de los primeros 10 cubos.

Trabajar con parte de una lista

En el Capítulo 3 aprendiste cómo acceder a elementos individuales en una lista, y en este capítulo has aprendido cómo trabajar con todos los elementos de una lista. También puede trabajar con un grupo específico de elementos en una lista, llamado segmento en Python.

Cortar una lista

Para hacer un corte, especifica el índice del primer y último elemento con el que desea trabajar. Al igual que con la función `range()`, Python detiene un elemento antes del segundo índice que especifique. Para generar los primeros tres elementos de una lista, solicitaría los índices del 0 al 3, que devolverían los elementos 0, 1 y 2.

El siguiente ejemplo involucra una lista de jugadores de un equipo:

```
jugadores.py jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir(jugadores[0:3])
```

Este código imprime una porción de la lista. El resultado conserva la estructura de la lista e incluye los primeros tres jugadores de la lista:

```
['charles', 'martina', 'michael']
```

Puede generar cualquier subconjunto de una lista. Por ejemplo, si desea los elementos segundo, tercero y cuarto de una lista, debe comenzar el segmento en el índice 1 y finalizarlo en el índice 4:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir(jugadores[1:4])
```

Esta vez el segmento comienza con 'martina' y termina con 'florencia':

```
['martina', 'michael', 'florencia']
```

Si omite el primer índice en un segmento, Python inicia automáticamente su segmento al principio de la lista:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir(jugadores[:4])
```

Sin un índice inicial, Python comienza al principio de la lista:

```
['charles', 'martina', 'michael', 'florencia']
```

Una sintaxis similar funciona si desea un segmento que incluya el final de una lista. Por ejemplo, si desea todos los elementos desde el tercer hasta el último, puede comenzar con el índice 2 y omitir el segundo índice:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir(jugadores[2:])
```

Python devuelve todos los elementos desde el tercer elemento hasta el final de la lista:

```
['michael', 'florencia', 'eli']
```

Esta sintaxis le permite generar todos los elementos desde cualquier punto de su lista hasta el final, independientemente de la longitud de la lista. Recuerde que un índice negativo devuelve un elemento a cierta distancia del final de una lista; por lo tanto, puede generar cualquier segmento desde el final de una lista. Por ejemplo, si queremos generar los últimos tres jugadores de la lista, podemos usar el segmento `jugadores[-3:]`:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir(jugadores[-3:])
```

Esto imprime los nombres de los últimos tres jugadores y seguirá funcionando. a medida que la lista de jugadores cambia de tamaño.

NOTA puede incluir un tercer valor entre paréntesis que indique un sector. Si se incluye un tercer valor, esto le dice a Python cuántos elementos omitir entre elementos en el rango especificado.

Recorriendo un segmento

Puede utilizar un segmento en un bucle for si desea recorrer un subconjunto de elementos de una lista. En el siguiente ejemplo, recorreremos los primeros tres jugadores e imprimimos sus nombres como parte de una lista simple:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']

print("Aquí están los primeros tres jugadores de mi equipo:")
1 para jugador in jugadores[:3]:
    imprimir(jugador.título())
```

En lugar de recorrer toda la lista de jugadores, Python recorre solo los primeros tres nombres 1:

```
Aquí están los tres primeros jugadores de mi equipo:
Charles
martina
Miguel
```

Los cortes son muy útiles en varias situaciones. Por ejemplo, cuando estás creando un juego, puedes agregar la puntuación final de un jugador a una lista cada vez que ese jugador termina de jugar. Luego, podrías obtener las tres mejores puntuaciones de un jugador ordenando la lista en orden decreciente y tomando una porción que incluya solo las tres primeras puntuaciones. Cuando trabaja con datos, puede utilizar sectores para procesar sus datos en fragmentos de un tamaño específico. O, cuando esté creando una aplicación web, podría usar sectores para mostrar información en una serie de páginas con una cantidad adecuada de información en cada página.

Copiar una lista

A menudo, querrás comenzar con una lista existente y crear una lista completamente nueva basada en la primera. Exploremos cómo funciona la copia de una lista y examinemos una situación en la que copiar una lista es útil.

Para copiar una lista, puede crear un segmento que incluya toda la lista original omitiendo el primer índice y el segundo índice ([:]). Esto le dice a Python que haga un segmento que comience en el primer elemento y termine con el último elemento, produciendo una copia de la lista completa.

Por ejemplo, imagina que tenemos una lista de nuestras comidas favoritas y queremos hacer una lista separada de las comidas que le gustan a un amigo. A este amigo le gusta todo lo que hay en nuestra lista hasta el momento, así que podemos crear su lista copiando la nuestra:

```
alimentos.py mis_alimentos = ['pizza', 'falafel', 'pastel de zanahoria']
1 amigo_comidas = mis_comidas[:]

print("Mis comidas favoritas son:")
imprimir(mis_alimentos)

print("\nLas comidas favoritas de mi amigo son:")
imprimir (comidas_amigas)
```

Primero, hacemos una lista de los alimentos que nos gustan llamada `my_foods`. Luego hacemos una nueva lista llamada `friend_foods`. Hacemos una copia de `my_foods` solicitando una porción de `my_foods` sin especificar ningún índice 1 y asignamos la copia a `friend_foods`. Cuando imprimimos cada lista, vemos que ambas contienen los mismos alimentos:

```
Mis comidas favoritas son:
['pizza', 'falafel', 'pastel de zanahoria']

Las comidas favoritas de mi amigo son:
['pizza', 'falafel', 'pastel de zanahoria']
```

Para demostrar que en realidad tenemos dos listas separadas, agregaremos un nuevo alimento a cada lista y mostraremos que cada lista realiza un seguimiento de los alimentos favoritos de la persona adecuada:

```
mis_alimentos = ['pizza', 'falafel', 'pastel de zanahoria']
1 amigo_comidas = mis_comidas[:]

2 my_foods.append('cannoli')
3 friend_foods.append('helado')

print("Mis comidas favoritas son:")
imprimir(mis_alimentos)

print("\nLas comidas favoritas de mi amigo son:")
imprimir (comidas_amigas)
```

Copiamos los elementos originales en `my_foods` a la nueva lista `friend_foods`, como hicimos en el ejemplo anterior 1. A continuación, agregamos un nuevo alimento a cada lista: agregamos 'cannoli' a `my_foods` 2, y agregamos 'helado' a `friend_foods` 3. Luego imprimimos las dos listas para ver si cada uno de estos alimentos está en la lista apropiada:

Mis comidas favoritas son:

```
['pizza', 'falafel', 'pastel de zanahoria', 'cannoli']
```

Las comidas favoritas de mi amigo son:

```
['pizza', 'falafel', 'pastel de zanahoria', 'helado']
```

El resultado muestra que los 'cannoli' ahora aparecen en nuestra lista de comidas favoritas. pero el 'helado' no. Podemos ver que ahora aparece 'helado' en la lista de nuestros amigos pero 'cannoli' no. Si simplemente hubiéramos establecido `amigos_alimentos` igual a `mis_alimentos`, no produciríamos dos listas separadas. Por ejemplo, esto es lo que sucede cuando intentas copiar una lista sin usar un segmento:

```
mis_alimentos = ['pizza', 'falafel', 'pastel de zanahoria']
```

Esto no funciona:

```
alimentos_amigos = mis_alimentos
```

```
my_foods.append('cannoli')
friend_foods.append('helado')
```

```
print("Mis comidas favoritas son:")
imprimir(mis_alimentos)
```

```
print("\nLas comidas favoritas de mi amigo son:")
imprimir (comidas_amigas)
```

En lugar de asignar una copia de `my_foods` a `friend_foods`, configuramos `friend_foods` igual a `mis_alimentos`. Esta sintaxis en realidad le dice a Python que asocie la nueva variable `alimentos_amigos` con la lista que ya está asociada con `mis_alimentos`, por lo que ahora ambas variables apuntan a la misma lista. Como resultado, cuando agregamos 'cannoli' a `my_foods`, también aparecerá en `friend_foods`. Del mismo modo, aparecerá 'helado' en ambas listas, aunque parezca estar agregado sólo a `friend_foods`.

El resultado muestra que ambas listas son iguales ahora, que no es lo que queríamos:

Mis comidas favoritas son:

```
['pizza', 'falafel', 'pastel de zanahoria', 'cannoli', 'helado']
```

Las comidas favoritas de mi amigo son:

```
['pizza', 'falafel', 'pastel de zanahoria', 'cannoli', 'helado']
```

NOTA No se preocupe por los detalles de este ejemplo por ahora. Si está intentando trabajar con una copia de una lista y ve un comportamiento inesperado, asegúrese de copiar la lista usando un segmento, como hicimos en el primer ejemplo.

INTÉNTALO TÚ MISMO

4-10. Slices: utilizando uno de los programas que escribió en este capítulo, agregue varias líneas al final del programa que hagan lo siguiente:

- Imprimir el mensaje Los primeros tres elementos de la lista son:. Luego use una rebanada para imprima los primeros tres elementos de la lista de ese programa.
- Imprimir el mensaje Tres elementos del medio de la lista son:. Luego use un segmento para imprimir tres elementos del medio de la lista.
- Imprimir el mensaje Los últimos tres elementos de la lista son:. Luego use un segmento para imprimir los últimos tres elementos de la lista.

4-11. Mis pizzas, tus pizzas: comienza con tu programa del ejercicio 4-1 (página 56).

Haga una copia de la lista de pizzas y llámela `friend_pizzas`. Luego, haz lo siguiente:

- Agregar una nueva pizza a la lista original.
- Añade una pizza diferente a la lista `friend_pizzas`.
- Demuestre que tiene dos listas separadas. Imprimir el mensaje Mi pizza favorita `zas son:`, y luego usa un bucle `for` para imprimir la primera lista. Imprima el mensaje Las pizzas favoritas de mi amigo son: y luego use un bucle `for` para imprimir la segunda lista. Asegúrese de que cada pizza nueva esté almacenada en la lista adecuada.

4-12. Más bucles: todas las versiones de `alimentos.py` en esta sección han evitado el uso de bucles al imprimir, para ahorrar espacio. Elija una versión de `food.py` y escriba dos bucles `for` para imprimir cada lista de alimentos.

tuplas

Las listas funcionan bien para almacenar colecciones de elementos que pueden cambiar a lo largo de la vida de un programa. La capacidad de modificar listas es particularmente importante cuando trabajas con una lista de usuarios en un sitio web o una lista de personajes en un juego. Sin embargo, a veces querrás crear una lista de elementos que no pueden cambiar. Las tuplas te permiten hacer precisamente eso. Python se refiere a los valores que no pueden cambiar como inmutables, y una lista inmutable se llama tupla.

Definiendo una tupla

Una tupla se parece a una lista, excepto que se utilizan paréntesis en lugar de corchetes. Una vez que define una tupla, puede acceder a elementos individuales utilizando el índice de cada elemento, tal como lo haría con una lista.

Por ejemplo, si tenemos un rectángulo que siempre debe tener un tamaño determinado, podemos asegurarnos de que su tamaño no cambie poniendo las dimensiones en una tupla:

```
dimensiones.py dimensiones = (200, 50)
imprimir(dimensiones[0])
imprimir(dimensiones[1])
```

Definimos las dimensiones de la tupla usando paréntesis en lugar de corchetes. Luego imprimimos cada elemento de la tupla individualmente, usando la misma sintaxis que hemos estado usando para acceder a los elementos de una lista:

```
200
50
```

Veamos qué sucede si intentamos cambiar uno de los elementos en las dimensiones de la tupla:

```
dimensiones = (200, 50)
dimensiones[0] = 250
```

Este código intenta cambiar el valor de la primera dimensión, pero Python devuelve un error de tipo. Debido a que estamos intentando alterar una tupla, lo cual no se puede hacer con ese tipo de objeto, Python nos dice que no podemos asignar un nuevo valor a un elemento en una tupla:

```
Rastreo (llamadas recientes más última):
  Archivo "dimensions.py", línea 2, en <módulo>
    dimensiones[0] = 250
TypeError: el objeto 'tupla' no admite la asignación de elementos
```

Esto es beneficioso porque queremos que Python genere un error cuando una línea de código intente cambiar las dimensiones del rectángulo.

NOTA Las tuplas se definen técnicamente por la presencia de una coma; los paréntesis los hacen parecer más claros y legibles. Si desea definir una tupla con un elemento, debe incluir una coma al final:

```
mi_t = (3,)
```

Generalmente no tiene sentido construir una tupla con un elemento, pero esto puede suceder cuando las tuplas se generan automáticamente.

Recorriendo todos los valores de una tupla

Puedes recorrer todos los valores de una tupla usando un bucle for, tal como lo hiciste con una lista:

```
dimensiones = (200, 50)
para dimensión en dimensiones:
    imprimir (dimensión)
```

Python devuelve todos los elementos de la tupla, tal como lo haría con una lista:

```
200
50
```

Escribir sobre una tupla

Aunque no puedes modificar una tupla, puedes asignar un nuevo valor a una variable que represente una tupla. Por ejemplo, si quisiéramos cambiar las dimensiones de este rectángulo, podríamos redefinir toda la tupla:

```
dimensiones = (200, 50)
imprimir("Dimensiones originales:")
para dimensión en dimensiones:
    imprimir (dimensión)

dimensiones = (400, 100)
print("\nDimensiones modificadas:")
para dimensión en dimensiones:
    imprimir (dimensión)
```

Las primeras cuatro líneas definen la tupla original e imprimen las dimensiones iniciales. Luego asociamos una nueva tupla con las dimensiones variables e imprimimos los nuevos valores. Python no genera ningún error esta vez, porque reasignar una variable es válido:

```
Dimensiones originales:
200
50

Dimensiones modificadas:
400
100
```

En comparación con las listas, las tuplas son estructuras de datos simples. Úselos cuando desee almacenar un conjunto de valores que no deben cambiarse durante la vida de un programa.

INTÉNTALO TÚ MISMO

4-13. Buffet: Un restaurante estilo buffet ofrece sólo cinco alimentos básicos. Piensa en cinco alimentos sencillos y guárdalos en una tupla.

- Utilice un bucle for para imprimir cada comida que ofrece el restaurante.
- Intente modificar uno de los elementos y asegúrese de que Python rechace el cambiar.
- El restaurante cambia su menú, reemplazando dos de los artículos con alimentos diferentes. Agregue una línea que reescriba la tupla y luego use un bucle for para imprimir cada uno de los elementos en el menú revisado.

Diseñar su código

Ahora que está escribiendo programas más largos, es una buena idea aprender a diseñar su código de manera consistente. Tómese el tiempo para hacer que su código sea lo más fácil de leer posible. Escribir código fácil de leer le ayuda a realizar un seguimiento de lo que hacen sus programas y también ayuda a otros a comprender su código.

Los programadores de Python han acordado una serie de convenciones de estilo para garantizar que el código de todos esté estructurado aproximadamente de la misma manera. Una vez que haya aprendido a escribir código Python limpio, debería poder comprender la estructura general del código Python de cualquier otra persona, siempre que sigan las mismas pautas. Si esperas convertirte en un programador profesional en algún momento, debes comenzar a seguir estas pautas lo antes posible para desarrollar buenos hábitos.

La guía de estilo

Cuando alguien quiere realizar un cambio en el lenguaje Python, escribe una propuesta de mejora de Python (PEP). Uno de los PEP más antiguos es el PEP 8, que instruye a los programadores de Python sobre cómo diseñar su código. PEP 8 es bastante extenso, pero gran parte se relaciona con estructuras de codificación más complejas que las que has visto hasta ahora.

La guía de estilo de Python se escribió teniendo en cuenta que el código se lee con más frecuencia de lo que se escribe. Escribirá su código una vez y luego comenzará a leerlo a medida que comience a depurar. Cuando agrega funciones a un programa, dedicará más tiempo a leer su código. Cuando comparte su código con otros programadores, ellos también leerán su código.

Dada la opción entre escribir código que sea más fácil de escribir o código que sea más fácil de leer, los programadores de Python casi siempre lo alentarán a escribir código que sea más fácil de leer. Las siguientes pautas le ayudarán a escribir código claro desde el principio.

Sangría

PEP 8 recomienda utilizar cuatro espacios por nivel de sangría. El uso de cuatro espacios mejora la legibilidad y deja espacio para múltiples niveles de sangría en cada línea.

En un documento de procesamiento de textos, la gente suele utilizar tabulaciones en lugar de espacios para aplicar sangría. Esto funciona bien para documentos de procesamiento de textos, pero el intérprete de Python se confunde cuando las tabulaciones se mezclan con espacios. Cada editor de texto proporciona una configuración que le permite usar la tecla TAB pero luego convierte cada pestaña en una cantidad determinada de espacios. Definitivamente deberías usar la tecla TAB, pero también asegurarte de que tu editor esté configurado para insertar espacios en lugar de tabulaciones en tu documento.

Mezclar tabulaciones y espacios en su archivo puede causar problemas que son muy difíciles de diagnosticar. Si cree que tiene una combinación de pestañas y espacios, puede convertir todas las pestañas de un archivo en espacios en la mayoría de los editores.

Longitud de la línea

Muchos programadores de Python recomiendan que cada línea tenga menos de 80 caracteres. Históricamente, esta directriz se desarrolló porque la mayoría de las computadoras sólo podían contener 79 caracteres en una sola línea en una ventana de terminal.

Actualmente, las personas pueden colocar líneas mucho más largas en sus pantallas, pero existen otras razones para adherirse a la longitud de línea estándar de 79 caracteres.

Los programadores profesionales suelen tener varios archivos abiertos en la misma pantalla y el uso de la longitud de línea estándar les permite ver líneas completas en dos o tres archivos abiertos uno al lado del otro en la pantalla. PEP 8 también recomienda que limite todos sus comentarios a 72 caracteres por línea, porque algunas de las herramientas que generan documentación automática para proyectos más grandes agregan caracteres de formato al comienzo de cada línea comentada.

Las pautas de PEP 8 para la longitud de la línea no están escritas en piedra y algunos equipos prefieren un límite de 99 caracteres. No se preocupe demasiado por la longitud de las líneas de su código mientras aprende, pero tenga en cuenta que las personas que trabajan en colaboración casi siempre siguen las pautas de PEP 8. La mayoría de los editores le permiten configurar una señal visual, generalmente una línea vertical en su pantalla, que le muestra dónde están estos límites.

NOTA

Apéndice B le muestra cómo configurar su editor de texto para que siempre inserte cuatro espacios cada vez que presiona la tecla TAB y muestra una guía vertical para ayudarlo a seguir el límite de 79 caracteres.

Líneas en blanco

Para agrupar visualmente partes de su programa, utilice líneas en blanco. Debes utilizar líneas en blanco para organizar tus archivos, pero no lo hagas en exceso. Siguiendo los ejemplos proporcionados en este libro, debería lograr el equilibrio adecuado.

Por ejemplo, si tiene cinco líneas de código que crean una lista y luego otras tres líneas que hacen algo con esa lista, es apropiado colocar una línea en blanco entre las dos secciones. Sin embargo, no debes colocar tres o cuatro líneas en blanco entre las dos secciones.

Las líneas en blanco no afectarán la forma en que se ejecuta su código, pero sí afectarán la legibilidad de su código. El intérprete de Python utiliza sangría horizontal para interpretar el significado de su código, pero ignora el espaciado vertical.

Otras pautas de estilo

PEP 8 tiene muchas recomendaciones de estilo adicionales, pero la mayoría de las pautas se refieren a programas más complejos que los que estás escribiendo en este momento. A medida que aprenda estructuras de Python más complejas, compartiré las partes relevantes de las pautas de PEP 8.

INTÉNTALO TÚ MISMO

4-14. PEP 8: consulte la guía de estilo original de PEP 8 en <https://python.org/dev/peps/pep-0008>. No lo usará mucho ahora, pero puede ser interesante hojearlo.

4-15. Revisión de código: elija tres de los programas que ha escrito en este capítulo y modifique cada uno para cumplir con PEP 8.

- Utilice cuatro espacios para cada nivel de sangría. Configure su editor de texto para insertar cuatro espacios cada vez que presione la tecla TAB, si aún no lo ha hecho (consulte el Apéndice B para obtener instrucciones sobre cómo hacerlo).
- Utilice menos de 80 caracteres en cada línea y configure su editor para que muestre una guía vertical en la posición del carácter 80.
- No utilice excesivamente líneas en blanco en sus archivos de programa.

Resumen

En este capítulo, aprendió cómo trabajar eficientemente con los elementos de una lista. Aprendiste cómo trabajar con una lista usando un bucle for, cómo Python usa la sangría para estructurar un programa y cómo evitar algunos errores de sangría comunes. Aprendiste a hacer listas numéricas simples, así como algunas operaciones que puedes realizar en listas numéricas. Aprendió a dividir una lista para trabajar con un subconjunto de elementos y a copiar listas correctamente usando una división. También aprendió sobre las tuplas, que brindan un grado de protección a un conjunto de valores que no deberían cambiar, y cómo diseñar su código cada vez más complejo para que sea fácil de leer.

En el Capítulo 5, aprenderá a responder apropiadamente a diferentes condiciones usando declaraciones if. Aprenderá a encadenar conjuntos relativamente complejos de pruebas condicionales para responder apropiadamente exactamente al tipo de situación o información que está buscando. También aprenderá a usar declaraciones if mientras recorre una lista para realizar acciones específicas con elementos seleccionados de una lista.

5

DECLARACIONES SI



La programación a menudo implica examinar un conjunto de condiciones y decidir qué acción tomar en función de esas condiciones.

La declaración `if` de Python le permite examinar el estado actual de un programa y responder adecuadamente a ese estado.

En este capítulo, aprenderá a escribir pruebas condicionales, que le permitirán verificar cualquier condición de interés. Aprenderá a escribir declaraciones `if` simples y aprenderá a crear una serie más compleja de declaraciones `if` para identificar cuándo están presentes las condiciones exactas que desea. Luego aplicará este concepto a las listas, de modo que podrá escribir un bucle `for` que maneje la mayoría de los elementos de una lista de una manera, pero que maneje ciertos elementos con valores específicos de una manera diferente.

Un ejemplo sencillo

El siguiente ejemplo muestra cómo las pruebas le permiten responder correctamente a situaciones especiales. Imagine que tiene una lista de automóviles y desea imprimir el nombre de cada automóvil. Los nombres de los automóviles son nombres propios, por lo que los nombres de la mayoría de los automóviles deben imprimirse en mayúsculas y minúsculas. Sin embargo, el valor "bmw" debe imprimirse completamente en mayúsculas. El siguiente código recorre una lista de nombres de automóviles y busca el valor "bmw". Siempre que el valor es 'bmw', se imprime en mayúsculas en lugar del título:

```
coches.py coches = ['audi', 'bmw', 'subaru', 'toyota']

    para coche en coches:
1 si coche == 'bmw':
        imprimir(coche.superior())
    demás:
        imprimir(coche.titulo())
```

El bucle de este ejemplo comprueba primero si el valor actual del coche es 'bmw' 1. Si es así, el valor se imprime en mayúsculas. Si el valor del automóvil no es "bmw", se imprime en mayúsculas y minúsculas:

```
Audi
BMW
subaru
toyota
```

Este ejemplo combina varios de los conceptos que aprenderá en este capítulo. Comencemos viendo los tipos de pruebas que puede utilizar para examinar las condiciones de su programa.

Pruebas condicionales

En el corazón de cada declaración if hay una expresión que puede evaluarse como Verdadera o Falso y se denomina prueba condicional. Python usa los valores Verdadero y Falso para decidir si se debe ejecutar el código en una declaración if . Si una prueba condicional se evalúa como Verdadera, Python ejecuta el código siguiendo el if declaración. Si la prueba se evalúa como False, Python ignora el código que sigue a la declaración if .

Comprobando la igualdad

La mayoría de las pruebas condicionales comparan el valor actual de una variable con un valor de interés específico. La prueba condicional más simple comprueba si el valor de una variable es igual al valor de interés:

```
>>> coche = 'bmw'
>>> coche == 'bmw'

Verdadero
```

La primera línea establece el valor del automóvil en 'bmw' usando un único signo igual, como ya ha visto muchas veces. La siguiente línea comprueba si el valor del coche es 'bmw' utilizando un doble signo igual (==). Este operador de igualdad devuelve Verdadero si los valores en el lado izquierdo y derecho del operador coinciden, y Falso si no coinciden. Los valores de este ejemplo coinciden, por lo que Python devuelve True.

Cuando el valor del automóvil no es "bmw", esta prueba devuelve Falso:

```
>>> coche = 'audi'
>>> coche == 'bmw'
FALSO
```

Un único signo igual es en realidad una declaración; puede leer la primera línea de código aquí como "Establezca el valor del automóvil igual a 'audi'". Por otro lado, un doble signo igual plantea una pregunta: "¿El valor del coche es igual a 'bmw'?" La mayoría de los lenguajes de programación utilizan signos iguales de esta manera.

Ignorar el caso al verificar la igualdad

Las pruebas de igualdad distinguen entre mayúsculas y minúsculas en Python. Por ejemplo, dos valores con distinta capitalización no se consideran iguales:

```
>>> coche = 'Audi'
>>> coche == 'audi'
FALSO
```

Si el caso importa, este comportamiento es ventajoso. Pero si las mayúsculas y minúsculas no importan y solo desea probar el valor de una variable, puede convertir el valor de la variable a minúsculas antes de realizar la comparación:

```
>>> coche = 'Audi'
>>> coche.lower() == 'audi'
Verdadero
```

Esta prueba devolverá Verdadero sin importar cómo esté formateado el valor 'Audi' porque la prueba ahora no distingue entre mayúsculas y minúsculas. El método lower() no cambia el valor que se almacenó originalmente en car, por lo que puedes hacer este tipo de comparación sin afectar la variable original:

```
>>> coche = 'Audi'
>>> coche.lower() == 'audi'
Verdadero
>>> coche
'Audi'
```

Primero asignamos la cadena en mayúscula 'Audi' a la variable automóvil. Luego, convertimos el valor del automóvil a minúsculas y comparamos el valor en minúsculas con la cadena 'audi'. Las dos cadenas coinciden, por lo que Python devuelve True. Podemos ver que el valor almacenado en el automóvil no se ha visto afectado por el método lower() .

Los sitios web imponen ciertas reglas para los datos que los usuarios ingresan de manera similar a esta. Por ejemplo, un sitio podría utilizar una prueba condicional como esta para

asegúrese de que cada usuario tenga un nombre de usuario verdaderamente único, no solo una variación de las mayúsculas del nombre de usuario de otra persona. Cuando alguien envía un nuevo nombre de usuario, ese nuevo nombre de usuario se convierte a minúsculas y se compara con las versiones en minúsculas de todos los nombres de usuario existentes. Durante esta verificación, un nombre de usuario como 'John' será rechazado si ya se está utilizando alguna variación de 'john'.

Comprobando la desigualdad

Cuando desee determinar si dos valores no son iguales, puede utilizar el operador de desigualdad (!=). Usemos otra declaración if para examinar cómo usar el operador de desigualdad. Almacenaremos un ingrediente de pizza solicitado en una variable y luego imprimiremos un mensaje si la persona no pidió anchoas:

```
toppings.py request_topping = 'champiñones'
```

```
si se solicita_topping != 'anchoas':
    print("¡Detén las anchoas!")
```

Este código compara el valor de request_topping con el valor 'anchoas'. Si estos dos valores no coinciden, Python devuelve True y ejecuta el código que sigue a la declaración if. Si los dos valores coinciden, Python devuelve False y no ejecuta el código que sigue a la declaración if.

Debido a que el valor de request_topping no es 'anchoas', la función print() se ejecuta la función:

```
¡Agarra las anchoas!
```

La mayoría de las expresiones condicionales que escriba probarán la igualdad, pero a veces le resultará más eficaz comprobar la desigualdad.

Comparaciones numéricas

Probar valores numéricos es bastante sencillo. Por ejemplo, el siguiente código comprueba si una persona tiene 18 años:

```
>>> edad = 18
>>> edad == 18
```

Verdadero

También puedes probar para ver si dos números no son iguales. Por ejemplo, el siguiente código imprime un mensaje si la respuesta dada no es correcta:

```
magia
_número.py respuesta = 17
si respuesta!= 42:
    print("Esa no es la respuesta correcta. ¡Inténtelo de nuevo!")
```

La prueba condicional pasa, porque el valor de la respuesta (17) no es igual a 42. Debido a que la prueba pasa, se ejecuta el bloque de código sangrado:

```
Esa no es la respuesta correcta. ¡Inténtalo de nuevo!
```

También puede incluir varias comparaciones matemáticas en sus declaraciones condicionales, como menor que, menor o igual que, mayor que y mayor o igual que:

```
>>> edad = 19
>>> edad < 21
Verdadero

>>> edad <= 21
Verdadero

>>> edad > 21
FALSO

>>> edad >= 21
FALSO
```

Cada comparación matemática se puede utilizar como parte de una declaración if , que puede ayudarle a detectar las condiciones exactas de interés.

Comprobación de múltiples condiciones

Es posible que desee comprobar varias condiciones al mismo tiempo. Por ejemplo, a veces es posible que necesites que dos condiciones sean verdaderas para realizar una acción. Otras veces, es posible que se sienta satisfecho con que una sola condición sea Verdadera. Las palabras clave y y o pueden ayudarle en estas situaciones.

Uso y verificación de múltiples condiciones

Para comprobar si dos condiciones son verdaderas simultáneamente, utilice la palabra clave y para combinar las dos pruebas condicionales; si cada prueba pasa, la expresión general se evalúa como Verdadera. Si cualquiera de las pruebas falla o si ambas pruebas fallan, la expresión se evalúa como Falsa.

Por ejemplo, puedes comprobar si dos personas tienen más de 21 años mediante la siguiente prueba:

```
>>> edad_0 = 22
>>> edad_1 = 18
1 >>> edad_0 >= 21 y edad_1 >= 21
FALSO
2 >>> edad_1 = 22
>>> edad_0 >= 21 y edad_1 >= 21
Verdadero
```

Primero, definimos dos edades, edad_0 y edad_1. Luego comprobamos si ambas edades tienen 21 años o más 1. La prueba de la izquierda pasa, pero la prueba de la derecha falla, por lo que la expresión condicional general se evalúa como Falsa. Luego cambiamos edad_1 a 22 2. El valor de edad_1 ahora es mayor que 21, por lo que ambas pruebas individuales pasan, lo que hace que la expresión condicional general se evalúe como Verdadera.

Para mejorar la legibilidad, puede utilizar paréntesis alrededor de las pruebas individuales, pero no son obligatorios. Si usa paréntesis, su prueba se vería así:

```
(edad_0 >= 21) y (edad_1 >= 21)
```

Uso de `and` para verificar múltiples condiciones

La palabra clave `and` también le permite verificar múltiples condiciones, pero pasa cuando una o ambas pruebas individuales pasan. Una expresión o falla sólo cuando fallan ambas pruebas individuales.

Consideremos nuevamente dos edades, pero esta vez buscaremos que solo una persona tenga más de 21 años:

```
>>> edad_0 = 22
>>> edad_1 = 18
1 >>> edad_0 >= 21 o edad_1 >= 21
Verdadero
2 >>> edad_0 = 18
>>> edad_0 >= 21 o edad_1 >= 21
FALSO
```

Empezamos nuevamente con dos variables de edad. Debido a que la prueba para `edad_0` 1 pasa, la expresión general se evalúa como Verdadera. Luego bajamos `edad_0` a 18. En la prueba final 2, ambas pruebas ahora fallan y la expresión general se evalúa como Falsa.

Comprobar si un valor está en una lista

A veces es importante comprobar si una lista contiene un valor determinado antes de realizar una acción. Por ejemplo, es posible que desee comprobar si ya existe un nuevo nombre de usuario en una lista de nombres de usuario actuales antes de completar el registro de alguien en un sitio web. En un proyecto de mapeo, es posible que desee verificar si una ubicación enviada ya existe en una lista de ubicaciones conocidas.

Para saber si un valor en particular ya está en una lista, use la palabra clave `in`. Consideremos un código que podría escribir para una pizzería. Haremos una lista de ingredientes que un cliente ha solicitado para una pizza y luego verificaremos si ciertos ingredientes están en la lista.

```
>>> coberturas_solicitadas = ['champiñones', 'cebollas', 'piña']
>>> 'hongos' in request_toppings
Verdadero
>>> 'pepperoni' in los_ingredientes_solicitados
FALSO
```

La palabra clave `in` le dice a Python que verifique la existencia de `'hongos'` y `'pepperoni'` en la lista de `ingredientes_solicitados`. Esta técnica es bastante poderosa porque puede crear una lista de valores esenciales y luego verificar fácilmente si el valor que está probando coincide con uno de los valores de la lista.

Comprobar si un valor no está en una lista

Otras veces, es importante saber si un valor no aparece en una lista. Puede utilizar la palabra clave `not` en esta situación. Por ejemplo, considere una lista de usuarios a los que se les prohíbe comentar en un foro. Puedes comprobar si un usuario ha sido baneado antes de permitirle enviar un comentario:

```
usuarios_prohibidos.py usuarios_prohibidos = ['andrew', 'carolina', 'david']
usuario = 'marie'
```

si el usuario no está en usuarios_prohibidos:

```
print(f"{user.title()}, puedes publicar una respuesta si lo deseas.")
```

La declaración if aquí se lee con bastante claridad. Si el valor de usuario no está en la lista usuarios_prohibidos, Python devuelve Verdadero y ejecuta la línea sangrada.

La usuaria 'marie' no está en la lista usuarios_prohibidos, por eso ve un mensaje invitándola a publicar una respuesta:

Marie, puedes publicar una respuesta si lo deseas.

Expresiones booleanas

A medida que aprenda más sobre programación, escuchará el término expresión booleana. en algún momento. Una expresión booleana es sólo otro nombre para una prueba condicional. Un valor booleano es Verdadero o Falso, al igual que el valor de una expresión condicional después de haber sido evaluada.

Los valores booleanos se utilizan a menudo para realizar un seguimiento de ciertas condiciones, como si un juego se está ejecutando o si un usuario puede editar cierto contenido en un sitio web:

```
game_active = Verdadero
```

```
can_edit = Falso
```

Los valores booleanos proporcionan una manera eficiente de rastrear el estado de un programa. o una condición particular que sea importante en su programa.

INTÉNTALO TÚ MISMO

5-1. Pruebas condicionales: escriba una serie de pruebas condicionales. Imprima una declaración que describa cada prueba y su predicción para los resultados de cada prueba. Su código debería verse así:

```
coche = 'subaru'
print("¿Es el auto == 'subaru'? Predigo que es Verdadero.")
imprimir(coche == 'subaru')

print("\n¿Es auto == 'audi'? Predigo Falso.")
imprimir(coche == 'audi')
```

- Mire de cerca sus resultados y asegúrese de comprender por qué cada línea se evalúa como Verdadero o Falso.
- Crear al menos 10 pruebas. Haga que al menos 5 pruebas se evalúen como Verdadero y otra 5 pruebas se evalúan como Falso.

(continuado)

5-2. Más pruebas condicionales: no es necesario limitar el número de pruebas que crea a 10. Si desea probar más comparaciones, escriba más pruebas y agréguelas a `conditional_tests.py`. Obtenga al menos un resultado Verdadero y uno Falso para cada uno de los siguientes:

- Pruebas de igualdad y desigualdad con cadenas
- Pruebas utilizando el método `lower()`
- Pruebas numéricas que involucran igualdad y desigualdad, mayor que y menor que, mayor que o igual que, y menor que o igual que
- Pruebas utilizando la palabra clave `and` y la palabra clave `or`
- Probar si un elemento está en una lista
- Probar si un elemento no está en una lista

si declaraciones

Cuando comprenda las pruebas condicionales, podrá comenzar a escribir declaraciones `if`. Existen varios tipos diferentes de declaraciones `if`, y su elección de cuál usar depende de la cantidad de condiciones que necesita probar. Viste varios ejemplos de declaraciones `if` en la discusión sobre pruebas condicionales, pero ahora profundicemos en el tema.

Declaraciones `if` simples

El tipo más simple de declaración `if` tiene una prueba y una acción:

```
si prueba_condicional:
    hacer algo
```

Puede colocar cualquier prueba condicional en la primera línea y casi cualquier acción en el bloque sangrado que sigue a la prueba. Si la prueba condicional se evalúa como Verdadera, Python ejecuta el código que sigue a la declaración `if`. Si la prueba se evalúa como False, Python ignora el código que sigue a la declaración `if`.

Digamos que tenemos una variable que representa la edad de una persona y queremos saber si esa persona tiene edad suficiente para votar. El siguiente código prueba si la persona puede votar:

```
votación.py edad = 19
si edad >= 18:
    print("¡Tienes edad suficiente para votar!")
```

Python comprueba si el valor de `edad` es mayor o igual a 18. Lo es, por lo que Python ejecuta la llamada `print()` con sangría:

```
¡Tienes edad suficiente para votar!
```

La sangría juega el mismo papel en las declaraciones if que en los bucles for . Todas las líneas sangradas después de una instrucción if se ejecutarán si la prueba pasa, y todo el bloque de líneas sangradas se ignorará si la prueba no pasa.

Puede tener tantas líneas de código como desee en el bloque que sigue a la declaración if . Agreguemos otra línea de salida si la persona tiene edad suficiente para votar, preguntando si ya se ha registrado para votar:

```
edad = 19
si edad >= 18:
    print("¡Tienes edad suficiente para votar!")
    print("¿Ya te has registrado para votar?")
```

La prueba condicional pasa y ambas llamadas a print() tienen sangría, por lo que se imprimen ambas líneas:

```
¡Tienes edad suficiente para votar!
¿Ya te registraste para votar?
```

Si el valor de la edad es inferior a 18 años, este programa no producirá ningún resultado.

Declaraciones if-else

A menudo, querrás realizar una acción cuando pasa una prueba condicional y una acción diferente en todos los demás casos. La sintaxis if-else de Python hace esto posible. Un bloque if-else es similar a una declaración if simple , pero la declaración else le permite definir una acción o un conjunto de acciones que se ejecutan cuando falla la prueba condicional.

Mostraremos el mismo mensaje que teníamos anteriormente si la persona tiene edad suficiente para votar, pero esta vez agregaremos un mensaje para cualquiera que no tenga edad suficiente para votar:

```
edad = 17
1 si edad >= 18:
    print("¡Tienes edad suficiente para votar!")
    print("¿Ya te has registrado para votar?")
2 más:
    print("Lo siento, eres demasiado joven para votar.")
    print ("¡Regístrate para votar tan pronto como cumpla 18 años!")
```

Si pasa la prueba condicional 1, el primer bloque de impresión con sangría() se ejecutan las llamadas. Si la prueba se evalúa como False, se ejecuta el bloque else 2. Debido a que esta vez la edad es menor de 18 años, la prueba condicional falla y se ejecuta el código en el bloque else :

```
Lo sentimos, eres demasiado joven para votar.
¡Regístrate para votar tan pronto como cumpla 18 años!
```

Este código funciona porque sólo tiene dos situaciones posibles para evaluar: una persona tiene edad suficiente para votar o no tiene edad suficiente para votar. El si no

La estructura funciona bien en situaciones en las que desea que Python siempre ejecute una de dos acciones posibles. En una cadena simple como esta, siempre se ejecutará una de las dos acciones.

La cadena if-elif-else

A menudo, necesitará probar más de dos situaciones posibles y, para evaluarlas, puede utilizar la sintaxis if-elif-else de Python. Python ejecuta solo un bloque en una cadena if-elif-else. Ejecuta cada prueba condicional en orden, hasta que una pasa. Cuando se pasa una prueba, el código que sigue a esa prueba se ejecuta y Python omite el resto de las pruebas.

Muchas situaciones del mundo real implican más de dos condiciones posibles.

Por ejemplo, considere un parque de diversiones que cobra tarifas diferentes para diferentes grupos de edad:

- La entrada para menores de 4 años es gratuita.
- La entrada para cualquier persona entre 4 y 18 años cuesta \$25.
- La entrada para cualquier persona mayor de 18 años cuesta \$40.

¿Cómo podemos usar una declaración if para determinar la tasa de admisión de una persona?

El siguiente código prueba el grupo de edad de una persona y luego imprime un mensaje de precio de entrada:

```
diversión
_parque.py  edad = 12
            1 si edad < 4:
                print("Su costo de entrada es $0.")
            2 elif edad < 18:
                print("Su costo de entrada es $25.")
            3 más:
                print("Su costo de entrada es $40.")
```

La prueba if 1 comprueba si una persona tiene menos de 4 años. Cuando se supera la prueba, se imprime un mensaje apropiado y Python se salta el resto de las pruebas. La línea elif 2 es en realidad otra prueba if, que se ejecuta sólo si la prueba anterior falló. En este punto de la cadena, sabemos que la persona tiene al menos 4 años porque la primera prueba falló. Si la persona es menor de 18 años, se imprime un mensaje apropiado y Python omite el bloque else. Si tanto el si

y las pruebas elif fallan, Python ejecuta el código en el bloque else 3.

En este ejemplo, la prueba if 1 se evalúa como False, por lo que su bloque de código no se ejecuta. Sin embargo, la prueba elif se evalúa como Verdadera (12 es menor que 18), por lo que se ejecuta su código. El resultado es una frase que informa al usuario del coste de la entrada:

Su costo de entrada es de \$25.

Cualquier edad mayor a 17 años haría que las dos primeras pruebas fallaran. En estas situaciones, se ejecutaría el bloque else y el precio de entrada sería de 40\$.

En lugar de imprimir el precio de entrada dentro del bloque if-elif-else, Sería más conciso establecer sólo el precio dentro de la cadena if-elif-else.

y luego tener una única llamada `print()` que se ejecuta después de que se haya evaluado la cadena:

```
edad = 12

si edad < 4:
    precio = 0
edad elif < 18:
    precio = 25
demás:
    precio = 40

print(f"El costo de tu entrada es ${price}.")
```

Las líneas sangradas establecen el valor del precio según la edad de la persona, como en el ejemplo anterior. Después de que la cadena `if-elif-else` establece el precio, una llamada `print()` independiente sin sangría utiliza este valor para mostrar un mensaje que informa el precio de entrada de la persona.

Este código produce el mismo resultado que el ejemplo anterior, pero el propósito de la cadena `if-elif-else` es más limitado. En lugar de determinar un precio y mostrar un mensaje, simplemente determina el precio de la entrada. Además de ser más eficiente, este código revisado es más fácil de modificar que el enfoque original. Para cambiar el texto del mensaje de salida, necesitará cambiar solo una llamada `print()` en lugar de tres llamadas `print()` separadas.

Usando múltiples bloques `elif`

Puedes usar tantos bloques `elif` en tu código como quieras. Por ejemplo, si el parque de diversiones implementara un descuento para personas mayores, podría agregar una prueba condicional más al código para determinar si alguien califica para el descuento para personas mayores. Digamos que cualquier persona de 65 años o más paga la mitad de la entrada regular, o \$20:

```
edad = 12

si edad < 4:
    precio = 0
edad elif < 18:
    precio = 25
elif edad < 65:
    precio = 40
demás:
    precio = 20

print(f"El costo de tu entrada es ${price}.")
```

La mayor parte de este código no ha cambiado. El segundo bloque `elif` ahora verifica que una persona tenga menos de 65 años antes de asignarle la tarifa de admisión completa de \$40. Tenga en cuenta que el valor asignado en el bloque `else` debe cambiarse a \$20, porque las únicas edades que llegan a este bloque son las personas de 65 años o más.

Omitiendo el bloque else

Python no requiere un bloque else al final de una cadena if-elif .

A veces, un bloque else es útil. Otras veces, es más claro utilizar una declaración elif adicional que capte la condición específica de interés:

```
edad = 12

si edad < 4:
    precio = 0
edad elif < 18:
    precio = 25
elif edad < 65:
    precio = 40
edad elif >= 65:
    precio = 20

print(f"El costo de tu entrada es ${price}.")
```

El bloque elif final asigna un precio de \$20 cuando la persona tiene 65 años o más, lo cual es un poco más claro que el bloque else general . Con este cambio, cada bloque de código debe pasar una prueba específica para poder ser ejecutado.

El bloque else es una declaración general. Coincide con cualquier condición que no coincida con una prueba if o elif específica y que a veces puede incluir datos no válidos o incluso maliciosos. Si está probando una condición final específica, considere usar un bloque elif final y omita el bloque else . Como resultado, tendrá más confianza en que su código se ejecutará sólo en las condiciones correctas.

Prueba de múltiples condiciones

La cadena if-elif-else es poderosa, pero solo es apropiada para usarla cuando solo necesitas pasar una prueba. Tan pronto como Python encuentra una prueba que pasa, se salta el resto de las pruebas. Este comportamiento es beneficioso porque es eficiente y le permite realizar pruebas para una condición específica.

Sin embargo, a veces es importante comprobar todas las condiciones de interés. En este caso, debería utilizar una serie de declaraciones simples if sin elif o else bloques. Esta técnica tiene sentido cuando más de una condición podría ser Verdadera y desea actuar en cada condición que sea Verdadera.

Reconsideremos el ejemplo de la pizzería. Si alguien pide dos ingredientes pizza, deberás asegurarte de incluir ambos ingredientes en su pizza:

```
toppings.py request_toppings = ['champiñones', 'queso extra']
```

```
si hay 'champiñones' en los ingredientes solicitados:
    print("Añadiendo champiñones.")
1 si 'pepperoni' en los ingredientes solicitados:
    print("Agregando pepperoni.")
```

```

si hay 'queso extra' en los ingredientes solicitados:
    print("Agregando queso extra.")

```

```

print("\n¡Terminaste de hacer tu pizza!")

```

Comenzamos con una lista que contiene los ingredientes solicitados. el primero si la declaración verifica si la persona solicitó champiñones en su pizza. Si es así, se imprime un mensaje confirmando ese aderezo. La prueba para pep-peroni 1 es otra declaración if simple , no una declaración elif o else , por lo que esta prueba se ejecuta independientemente de si la prueba anterior pasó o no. La última declaración if verifica si se solicitó queso adicional, independientemente de los resultados de las dos primeras pruebas. Estas tres pruebas independientes se ejecutan cada vez que se ejecuta este programa.

Debido a que se evalúan todas las condiciones en este ejemplo, se agregan champiñones y queso adicional a la pizza:

```

Añadiendo champiñones.
Agregar queso extra.

```

```

¡Terminaste de hacer tu pizza!

```

Este código no funcionaría correctamente si usáramos un bloque if-elif-else , porque el código dejaría de ejecutarse después de pasar solo una prueba. Así es como se vería:

```

request_toppings = ['champiñones', 'queso extra']

si hay 'champiñones' en los ingredientes solicitados:
    print("Añadiendo champiñones.")
elif 'pepperoni' en los ingredientes solicitados:
    print("Agregando pepperoni.")
elif 'queso extra' en request_toppings:
    print("Agregando queso extra.")

print("\n¡Terminaste de hacer tu pizza!")

```

La prueba de los "champiñones" es la primera que se pasa, por lo que se añaden champiñones a la pizza. Sin embargo, los valores 'queso extra' y 'pepperoni' nunca se verifican, porque Python no ejecuta ninguna prueba más allá de la primera prueba que pasa en una cadena if-elif-else . Se agregará el primer ingrediente del cliente, pero se omitirán todos los demás ingredientes:

```

Añadiendo champiñones.

```

```

¡Terminaste de hacer tu pizza!

```

En resumen, si desea que solo se ejecute un bloque de código, use if-elif-else cadena. Si es necesario ejecutar más de un bloque de código, utilice una serie de declaraciones if independientes .

INTÉNTALO TÚ MISMO

5-3. Alien Colors #1: Imagina que acaban de derribar a un extraterrestre en un juego. Cree una variable llamada `alien_color` y asígnele un valor de "verde", "amarillo" o "rojo".

- Escriba una declaración `if` para comprobar si el color del extraterrestre es verde. Si es así, imprima un mensaje indicando que el jugador acaba de ganar 5 puntos.
- Escriba una versión de este programa que pase la prueba `if` y otra que falle. (La versión que falla no tendrá resultados).

5-4. Colores alienígenas #2: Elige un color para un alienígena como lo hiciste en el ejercicio 5-3 y escribe una cadena `si-si no`.

- Si el color del extraterrestre es verde, imprima una declaración de que el jugador acaba de ganar 5 puntos por dispararle al extraterrestre.
- Si el color del alienígena no es verde, imprime una declaración que el jugador acaba de ganar 10 puntos.
- Escriba una versión de este programa que ejecute el bloque `if` y otra que ejecute el bloque `else`.

5-5. Alien Colors #3: Convierte tu cadena `if-else` del Ejercicio 5-4 en una cadena `if-elif-else`.

- Si el alienígena es verde, imprime un mensaje de que el jugador obtuvo 5 puntos.
- Si el alienígena es amarillo, imprime un mensaje de que el jugador obtuvo 10 puntos.
- Si el alienígena es rojo, imprime un mensaje de que el jugador obtuvo 15 puntos.
- Escriba tres versiones de este programa, asegurándose de que cada mensaje se imprima con el color alienígena apropiado.

5-6. Etapas de la vida: escriba una cadena `if-elif-else` que determine la etapa de la vida de una persona. Establezca un valor para la variable `edad` y luego:

- Si la persona tiene menos de 2 años, imprima un mensaje de que la persona es un bebé.
- Si la persona tiene al menos 2 años pero menos de 4, imprima un mensaje que indique que la persona es un niño pequeño.
- Si la persona tiene al menos 4 años pero menos de 13, imprima un mensaje que indique que la persona es un niño.
- Si la persona tiene al menos 13 años pero menos de 20, imprima un mensaje que la persona es un adolescente.
- Si la persona tiene al menos 20 años pero menos de 65, imprima un mensaje de que la persona es un adulto.
- Si la persona tiene 65 años o más, imprima un mensaje que indique que la persona es una persona mayor.

5-7. Fruta favorita: haga una lista de sus frutas favoritas y luego escriba una serie de declaraciones independientes que verifiquen ciertas frutas en su lista.

- Haz una lista de tus tres frutas favoritas y llámala `frutas_favoritas`.
- Escribe cinco declaraciones `if`. Cada uno debe comprobar si un determinado tipo de fruta está en su lista. Si la fruta está en tu lista, el bloque `if` debería imprimir una declaración como:
¡Realmente te gustan los plátanos!

Usar declaraciones `if` con listas

Puedes hacer un trabajo interesante combinando listas y declaraciones `if`. Puede buscar valores especiales que deban tratarse de manera diferente a otros valores de la lista. Puede gestionar de manera eficiente las condiciones cambiantes, como la disponibilidad de ciertos artículos en un restaurante durante un turno. También puede comenzar a demostrar que su código funciona como espera en todas las situaciones posibles.

Comprobando artículos especiales

Este capítulo comenzó con un ejemplo sencillo que mostraba cómo manejar un valor especial como 'bmw', que debía imprimirse en un formato diferente al de otros valores de la lista. Ahora que tiene una comprensión básica de las pruebas condicionales y las declaraciones `if`, echemos un vistazo más de cerca a cómo puede buscar valores especiales en una lista y manejar esos valores apropiadamente.

Sigamos con el ejemplo de la pizzería. La pizzería muestra un mensaje cada vez que se agrega un aderezo a la pizza, mientras se prepara. El código para esta acción se puede escribir de manera muy eficiente haciendo una lista de ingredientes que el cliente ha solicitado y usando un bucle para anunciar cada ingrediente a medida que se agrega a la pizza:

```
toppings.py request_toppings = ['champiñones', 'pimientos verdes', 'queso extra']
```

```
para cobertura_solicitada in coberturas_solicitadas:
    print(f"Agregando {requested_topping}.")
```

```
print("\n¡Terminaste de hacer tu pizza!")
```

El resultado es sencillo porque este código es simplemente un bucle `for` simple :

Añadiendo champiñones.

Añadiendo pimientos verdes.

Agregar queso extra.

```
¡Terminaste de hacer tu pizza!
```

¿Pero qué pasa si la pizzería se queda sin pimientos verdes? Una declaración si dentro del bucle for puede manejar esta situación apropiadamente:

```
request_toppings = ['champiñones', 'pimientos verdes', 'queso extra']

para cobertura_solicitada en coberturas_solicitadas:
    if request_topping == 'pimientos verdes':
        print("Lo sentimos, se nos acabaron los pimientos verdes en este momento.")
    demás:
        print(f"Agregando {requested_topping}.")

print("\n¡Terminaste de hacer tu pizza!")
```

Esta vez, revisamos cada elemento solicitado antes de agregarlo a la pizza. La declaración if verifica si la persona solicitó pimientos verdes. Si es así, mostramos un mensaje informándoles por qué no pueden comer pimientos verdes. El bloque else garantiza que todos los demás ingredientes se agregarán a la pizza. El resultado muestra que cada ingrediente solicitado se maneja adecuadamente.

```
Añadiendo champiñones.
Lo sentimos, se nos acabaron los pimientos verdes en este momento.
Agregar queso extra.

¡Terminaste de hacer tu pizza!
```

Comprobar que una lista no esté vacía

Hemos hecho una suposición simple sobre cada lista con la que hemos trabajado hasta ahora: hemos asumido que cada lista tiene al menos un elemento. Pronto permitiremos que los usuarios proporcionen la información almacenada en una lista, por lo que no podremos asumir que una lista contiene elementos cada vez que se ejecuta un bucle. En esta situación, es útil comprobar si una lista está vacía antes de ejecutar un bucle for .

Como ejemplo, verifiquemos si la lista de ingredientes solicitados está vacía antes de preparar la pizza. Si la lista está vacía, avisaremos al usuario y nos aseguraremos de que quiera una pizza sencilla. Si la lista no está vacía, construiremos la pizza tal como lo hicimos en los ejemplos anteriores:

```
ingredientes_solicitados = []

si se solicitan_toppings:
    para cobertura_solicitada en coberturas_solicitadas:
        print(f"Agregando {requested_topping}.")
    print("\n¡Terminaste de hacer tu pizza!")
demás:
    print("¿Estás seguro de que quieres una pizza simple?")
```

Esta vez comenzamos con una lista vacía de ingredientes solicitados. En lugar de saltar directamente a un bucle for , primero hacemos una comprobación rápida. Cuando se usa el nombre de una lista en una declaración if , Python devuelve True si la lista contiene al menos un elemento; una lista vacía se evalúa como False. Si request_toppings pasa la prueba condicional, ejecutamos el mismo bucle for que usamos en el anterior

ejemplo. Si la prueba condicional falla, imprimimos un mensaje preguntando al cliente si realmente quiere una pizza simple y sin aderezos.

La lista está vacía en este caso, por lo que el resultado pregunta si el usuario realmente quiere una pizza simple:

¿Estás seguro de que quieres una pizza sencilla?

Si la lista no está vacía, el resultado mostrará cada ingrediente solicitado.
se agrega a la pizza.

Usar múltiples listas

La gente pedirá casi cualquier cosa, especialmente cuando se trata de ingredientes para pizza.

¿Qué pasa si un cliente realmente quiere papas fritas en su pizza? Puede utilizar listas y declaraciones if para asegurarse de que sus comentarios tengan sentido antes de actuar en consecuencia.

Tengamos cuidado con las solicitudes de ingredientes inusuales antes de preparar una pizza. El siguiente ejemplo define dos listas. La primera es una lista de ingredientes disponibles en la pizzería y la segunda es la lista de ingredientes que el usuario ha solicitado. Esta vez, cada elemento de `request_toppings` se compara con la lista de ingredientes disponibles antes de agregarlo a la pizza:

```
available_toppings = ['champiñones', 'aceitunas', 'pimientos verdes',
                    'pepperoni', 'piña', 'queso extra']
```

```
1 coberturas solicitadas = ['champiñones', 'papas fritas', 'queso extra']
```

```
    para cobertura_solicitada en coberturas_solicitadas:
```

```
2 si se solicita_topping en toppings_disponibles:
```

```
    print(f"Agregando {requested_topping}.")
```

```
3 más:
```

```
    print(f"Lo sentimos, no tenemos {requested_topping}.")
```

```
print("\n¡Terminaste de hacer tu pizza!")
```

Primero, definimos una lista de ingredientes disponibles en esta pizzería. Tenga en cuenta que esto podría ser una tupla si la pizzería tiene una selección estable de ingredientes. Luego, hacemos una lista de ingredientes que un cliente ha solicitado. Hay una solicitud inusual para un aderezo en este ejemplo: 'papas fritas' 1. A continuación recorreremos la lista de aderezos solicitados. Dentro del ciclo, verificamos si cada ingrediente solicitado está realmente en la lista de ingredientes disponibles 2. Si es así, agregamos ese ingrediente a la pizza. Si el ingrediente solicitado no está en la lista de ingredientes disponibles, el bloque else se ejecutará 3. El bloque else imprime un mensaje que le indica al usuario qué ingredientes no están disponibles.

Esta sintaxis de código produce una salida limpia e informativa:

Añadiendo champiñones.

Lo siento, no tenemos papas fritas.

Agregar queso extra.

¡Terminaste de hacer tu pizza!

¡En solo unas pocas líneas de código, hemos manejado una situación del mundo real de manera bastante efectiva!

INTÉNTALO TÚ MISMO

5-8. Hola administrador: haga una lista de cinco o más nombres de usuario, incluido el nombre "admin". Imagine que está escribiendo un código que imprimirá un saludo para cada usuario después de iniciar sesión en un sitio web. Recorra la lista e imprima un saludo para cada usuario.

- Si el nombre de usuario es 'admin', imprima un saludo especial, como Hola administrador, ¿Le gustaría ver un informe de estado?
- De lo contrario, imprima un saludo genérico, como Hola Jaden, gracias por iniciar sesión nuevamente.

5-9. Sin usuarios: agregue una prueba if a `hello_admin.py` para asegurarse de que la lista de usuarios no esté vacía.

- Si la lista está vacía, imprima el mensaje ¡Necesitamos encontrar algunos usuarios!
- Elimine todos los nombres de usuario de su lista y asegúrese de que se imprima el mensaje correcto.

5-10. Verificación de nombres de usuario: haga lo siguiente para crear un programa que simule cómo los sitios web garantizan que todos tengan un nombre de usuario único.

- Haga una lista de cinco o más nombres de usuario llamados `usuarios_actuales`.
- Haga otra lista de cinco nombres de usuario llamada `new_users`. Asegúrese de que uno o dos de los nuevos nombres de usuario también estén en la lista de usuarios actuales .
- Recorra la lista `new_users` para ver si cada nuevo nombre de usuario ya se ha utilizado. Si es así, imprima un mensaje indicando que la persona deberá ingresar un nuevo nombre de usuario. Si no se ha utilizado un nombre de usuario, imprima un mensaje que indique que el nombre de usuario está disponible.
- Asegúrese de que su comparación no distinga entre mayúsculas y minúsculas. Si se ha utilizado 'John' , no se debe aceptar 'JOHN' . (Para hacer esto, deberá hacer una copia de `current_users` que contenga las versiones en minúsculas de todos los usuarios existentes).

5-11. Números ordinales: los números ordinales indican su posición en una lista, como 1º o 2º. La mayoría de los números ordinales terminan en th, excepto 1, 2 y 3.

- Almacene los números del 1 al 9 en una lista.
- Recorrer la lista.
- Utilice una cadena if-elif-else dentro del bucle para imprimir el final ordinal adecuado para cada número. Su salida debe leer "1.º 2.º 3.º 4.º 5.º 6.º 7.º 8.º 9.º", y cada resultado debe estar en una línea separada.

Diseñar tus declaraciones if

En cada ejemplo de este capítulo, has visto buenos hábitos de estilo. La única recomendación que PEP 8 proporciona para diseñar pruebas condicionales es utilizar un espacio único alrededor de los operadores de comparación, como `==`, `>=` y `<=`. Por ejemplo:

si edad < 4:

es mejor que:

si edad<4:

Dicho espaciado no afecta la forma en que Python interpreta su código; es solo hace que su código sea más fácil de leer para usted y otros.

INTÉNTALO TÚ MISMO

5-12. Diseñar declaraciones if : revise los programas que escribió en este capítulo y asegúrese de haber diseñado sus pruebas condicionales de manera adecuada.

5-13. Tus ideas: En este punto, eres un programador más capaz que cuando empezaste este libro. Ahora que tiene una mejor idea de cómo se modelan las situaciones del mundo real en los programas, es posible que esté pensando en algunos problemas que podría resolver con sus propios programas. Registre cualquier idea nueva que tenga sobre los problemas que desee resolver a medida que sus habilidades de programación sigan mejorando. Considere los juegos que quizás desee escribir, los conjuntos de datos que desee explorar y las aplicaciones web que desee crear.

Resumen

En este capítulo aprendió a escribir pruebas condicionales, que siempre se evalúan como Verdadero o Falso. Aprendiste a escribir declaraciones simples if , if-else cadenas y cadenas if-elif-else . Comenzó a utilizar estas estructuras para identificar condiciones particulares que necesita probar y saber cuándo se cumplieron esas condiciones en sus programas. Aprendió a manejar ciertos elementos en una lista de manera diferente que todos los demás elementos mientras continuaba utilizando la eficiencia de un bucle for . También revisó las recomendaciones de estilo de Python para asegurarse de que sus programas cada vez más complejos sigan siendo relativamente fáciles de leer y comprender.

En el Capítulo 6 aprenderá sobre los diccionarios de Python. Un diccionario es similar a una lista, pero le permite conectar piezas de información. Aprenderá a crear diccionarios, recorrerlos en bucle y utilizarlos en combinación con listas y declaraciones if . Aprender sobre diccionarios le permitirá modelar una variedad aún más amplia de situaciones del mundo real.

6

DICCIONARIOS



En este capítulo aprenderá a utilizar los diccionarios de Python, que le permiten conectar piezas de información relacionada. Aprenderás cómo acceder a la información una vez que está en un dic-

cionario y cómo modificar esa información. Debido a que los diccionarios pueden almacenar una cantidad casi ilimitada de información, le mostraré cómo recorrer los datos en un diccionario. Además, aprenderá a anidar diccionarios dentro de listas, listas dentro de diccionarios e incluso diccionarios dentro de otros diccionarios.

Comprender los diccionarios le permite modelar una variedad de objetos del mundo real con mayor precisión. Podrás crear un diccionario que represente a una persona y luego almacenar tanta información como quieras sobre esa persona. Puedes almacenar su nombre, edad, ubicación, profesión y cualquier otro aspecto de una persona que puedas describir. Podrás almacenar dos tipos de información que puedan coincidir, como una lista de palabras y sus significados, una lista de nombres de personas y sus números favoritos, una lista de montañas y sus elevaciones, etc. adelante.

Un diccionario sencillo

Considere un juego en el que aparecen extraterrestres que pueden tener diferentes colores y valores en puntos. Este sencillo diccionario almacena información sobre un extraterrestre en particular:

```
alien.py alien_0 = {'color': 'verde', 'puntos': 5}
```

```
imprimir(alienígena_0['color'])
imprimir(alien_0['puntos'])
```

El diccionario alien_0 almacena el color y el valor en puntos del alienígena. El último Dos líneas acceden y muestran esa información, como se muestra aquí:

```
verde
5
```

Como ocurre con la mayoría de los nuevos conceptos de programación, el uso de diccionarios requiere práctica. Una vez que haya trabajado un poco con diccionarios, verá con qué eficacia pueden modelar situaciones del mundo real.

Trabajar con diccionarios

Un diccionario en Python es una colección de pares clave-valor. Cada clave está conectada a un valor y puede usar una clave para acceder al valor asociado con esa clave. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario. De hecho, puedes usar cualquier objeto que puedas crear en Python como valor en un diccionario.

En Python, un diccionario está entre llaves ({}) con una serie de pares clave-valor dentro de las llaves, como se muestra en el ejemplo anterior:

```
alien_0 = {'color': 'verde', 'puntos': 5}
```

Un par clave-valor es un conjunto de valores asociados entre sí. Cuando proporciona una clave, Python devuelve el valor asociado con esa clave. Cada clave está conectada a su valor mediante dos puntos y los pares clave-valor individuales están separados por comas. Puede almacenar tantos pares clave-valor como desee en un diccionario.

El diccionario más simple tiene exactamente un par clave-valor, como se muestra en esta versión modificada del diccionario alien_0 :

```
alien_0 = {'color': 'verde'}
```

Este diccionario almacena una pieza de información sobre alien_0: el color del alienígena. La cadena 'color' es una clave en este diccionario y su valor asociado es 'verde'.

Acceder a valores en un diccionario

Para obtener el valor asociado con una clave, proporcione el nombre del diccionario y luego coloque la clave dentro de un conjunto de corchetes, como se muestra aquí:

```
alien.py alien_0 = {'color': 'verde'}
imprimir(alienígena_0['color'])
```

Esto devuelve el valor asociado con la clave 'color' del diccionario alien_0:

```
verde
```

Puede tener un número ilimitado de pares clave-valor en un diccionario. Por ejemplo, aquí está el diccionario alien_0 original con dos pares clave-valor:

```
alien_0 = {'color': 'verde', 'puntos': 5}
```

Ahora puedes acceder al color o al valor en puntos de alien_0. Si un jugador derriba a este alienígena, puedes buscar cuántos puntos debería ganar usando un código como este:

```
alien_0 = {'color': 'verde', 'puntos': 5}

nuevos_puntos = alien_0['puntos']
print(f"¡Acabas de ganar {nuevos_puntos} puntos!")
```

Una vez que se ha definido el diccionario, extraemos el valor asociado con los 'puntos' clave del diccionario. Luego, este valor se asigna a la variable new_points. La última línea imprime una declaración sobre cuántos puntos acaba de ganar el jugador:

```
¡Acabas de ganar 5 puntos!
```

Si ejecuta este código cada vez que derriban a un alienígena, se recuperará el valor en puntos del alienígena.

Agregar nuevos pares clave-valor

Los diccionarios son estructuras dinámicas y puede agregar nuevos pares clave-valor a un diccionario en cualquier momento. Para agregar un nuevo par clave-valor, debe proporcionar el nombre del diccionario seguido de la nueva clave entre corchetes, junto con el nuevo valor.

Agreguemos dos nuevos datos al diccionario alien_0 : el las coordenadas x e y del extraterrestre , que nos ayudarán a mostrar al extraterrestre en una posición particular en la pantalla. Coloquemos al extraterrestre en el borde izquierdo de la pantalla, a 25 píxeles desde la parte superior. Debido a que las coordenadas de la pantalla generalmente comienzan en la esquina superior izquierda de la pantalla, colocaremos al extraterrestre en el borde izquierdo de la pantalla configurando la coordenada x en 0 y 25 píxeles desde la parte superior configurando su coordenada y en positiva. 25, como se muestra aquí:

```
alien.py alien_0 = {'color': 'verde', 'puntos': 5}
        imprimir (alienígena_0)

alien_0['x_position'] = 0
alien_0['y_position'] = 25
imprimir (alienígena_0)
```

Comenzamos definiendo el mismo diccionario con el que hemos estado trabajando. Luego imprimimos este diccionario, mostrando una instantánea de su información. A continuación, agregamos un nuevo par clave-valor al diccionario: la clave 'x_position' y el valor 0. Hacemos lo mismo para la clave 'y_position'. Cuando imprimimos el diccionario modificado, vemos los dos pares clave-valor adicionales:

```
{'color': 'verde', 'puntos': 5}
{'color': 'verde', 'puntos': 5, 'posición_x': 0, 'posición_y': 25}
```

La versión final del diccionario contiene cuatro pares clave-valor. Los dos originales especifican el color y el valor en puntos, y dos más especifican la posición del extraterrestre.

Los diccionarios conservan el orden en el que fueron definidos. Cuando imprime un diccionario o recorre sus elementos, verá los elementos en el mismo orden en que se agregaron al diccionario.

Comenzando con un diccionario vacío

A veces es conveniente, o incluso necesario, comenzar con un diccionario vacío y luego agregarle cada elemento nuevo. Para comenzar a llenar un diccionario vacío, defina un diccionario con un conjunto de llaves vacío y luego agregue cada par clave-valor en su propia línea. Por ejemplo, aquí se explica cómo crear el diccionario `alien_0` usando este enfoque:

```
alien.py alien_0 = {}

alien_0['color'] = 'verde'
alien_0['puntos'] = 5
```

```
imprimir (alien_0)
```

Primero definimos un diccionario `alien_0` vacío y luego le agregamos colores y valores de puntos. El resultado es el diccionario que hemos estado usando en ejemplos anteriores:

```
{'color': 'verde', 'puntos': 5}
```

Normalmente, utilizará diccionarios vacíos al almacenar datos proporcionados por el usuario. en un diccionario o al escribir código que genera una gran cantidad de pares clave-valor automáticamente.

Modificar valores en un diccionario

Para modificar un valor en un diccionario, proporcione el nombre del diccionario con la clave entre corchetes y luego el nuevo valor que desea asociar con esa clave. Por ejemplo, considere un extraterrestre que cambia de verde a amarillo a medida que avanza el juego:

```
alien.py alien_0 = {'color': 'verde'}
print(f"El extraterrestre es {alien_0['color']}")
```

```
alien_0['color'] = 'amarillo'
print(f"El extraterrestre ahora es {alien_0['color']}.")
```

Primero definimos un diccionario para alien_0 que contiene sólo el color del alienígena; luego cambiamos el valor asociado con la clave 'color' a 'amarillo'.

El resultado muestra que el alienígena efectivamente ha cambiado de verde a amarillo:

```
El extraterrestre es verde.
El alienígena ahora es amarillo.
```

Para ver un ejemplo más interesante, rastreemos la posición de un extraterrestre que puede moverse a diferentes velocidades. Almacenaremos un valor que representa la velocidad actual del alienígena y luego lo usaremos para determinar qué tan hacia la derecha debe moverse el alienígena:

```
alien_0 = {'x_position': 0, 'y_position': 25, 'velocidad': 'media'}
print(f"Posición original: {alien_0['x_position']}")
```

```
# Mueve al alienígena hacia la derecha.
```

```
# Determina qué tan lejos mover al alienígena en función de su velocidad actual.
```

```
1 si alien_0['velocidad'] == 'lento':
```

```
    incremento_x = 1
```

```
elif alien_0['velocidad'] == 'medio':
```

```
    incremento_x = 2
```

```
demás:
```

```
    # Este debe ser un alienígena rápido.
```

```
    incremento_x = 3
```

```
# La nueva posición es la posición anterior más el incremento.
```

```
2 alien_0['x_position'] = alien_0['x_position'] + x_increment
```

```
print(f"Nueva posición: {alien_0['x_position']}")
```

Comenzamos definiendo un extraterrestre con una posición x e y iniciales, y una velocidad 'media'. Hemos omitido el color y los valores de puntos por razones de simplicidad, pero este ejemplo funcionaría de la misma manera si incluyera también esos pares clave-valor. También imprimimos el valor original de x_position para ver qué tan lejos se mueve el extraterrestre hacia la derecha.

Una cadena if-elif-else determina qué tan lejos debe moverse el alienígena hacia la derecha y asigna este valor a la variable x_increment. Si la velocidad del alienígena es "lenta", se mueve una unidad hacia la derecha; si la velocidad es 'media', se mueve dos unidades hacia la derecha; y si es 'rápido', se mueve tres unidades hacia la derecha. Una vez que se ha calculado el incremento, se suma al valor de x_position y el resultado se almacena en x_position del diccionario.

Debido a que se trata de un alienígena de velocidad media, su posición se desplaza dos unidades hacia la derecha:

```
Posición x original: 0
```

```
Nueva posición x: 2
```

Esta técnica es genial: al cambiar un valor en el diccionario del alienígena, puedes cambiar el comportamiento general del alienígena. Por ejemplo, para convertir a este alienígena de velocidad media en uno rápido, añadirías esta línea:

```
alien_0['velocidad'] = 'rápido'
```

El bloque if-elif-else asignaría un valor mayor a `x_increment` la próxima vez que se ejecute el código.

Eliminación de pares clave-valor

Cuando ya no necesite una información almacenada en un diccionario, puede usar la instrucción `del` para eliminar por completo un par clave-valor.

Todo lo que necesita es el nombre del diccionario y la clave que desea eliminar.

Por ejemplo, eliminemos la clave 'puntos' del diccionario `alien_0`, junto con su valor:

```
alien.py alien_0 = {'color': 'verde', 'puntos': 5}
        imprimir (alienígena_0)

1 del alien_0['puntos']
        imprimir (alienígena_0)
```

La declaración del 1 le dice a Python que elimine los 'puntos' clave del diccionario `alien_0` y que también elimine el valor asociado con esa clave.

El resultado muestra que la clave 'puntos' y su valor de 5 se eliminan del diccionario, pero el resto del diccionario no se ve afectado:

```
{'color': 'verde', 'puntos': 5}
{'color': 'verde'}
```

NOTA tenga en cuenta que el par clave-valor eliminado se elimina permanentemente.

Un diccionario de objetos similares

El ejemplo anterior implicó almacenar diferentes tipos de información sobre un objeto, un extraterrestre, en un juego. También puedes utilizar un diccionario para almacenar un tipo de información sobre muchos objetos. Por ejemplo, supongamos que desea encuestar a varias personas y preguntarles cuál es su lenguaje de programación favorito. Un diccionario es útil para almacenar los resultados de una encuesta simple, como esta:

```
favorito _idiomas.py idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}
```

Como puede ver, hemos dividido un diccionario más grande en varias líneas. Cada clave es el nombre de una persona que respondió a la encuesta y cada valor es su elección de idioma. Cuando sepa que necesitará más de una línea para definir un diccionario, presione ENTRAR después de la llave de apertura. Luego sangra la siguiente línea un nivel (cuatro espacios) y escribe el primer par clave-valor, seguido de una coma. A partir de este momento, cuando presione ENTRAR, su editor de texto debería sangrar automáticamente todos los pares clave-valor posteriores para que coincidan con el primer par clave-valor.

Una vez que haya terminado de definir el diccionario, agregue una llave de cierre en una nueva línea después del último par clave-valor y sangra un nivel para que se alinee con las claves del diccionario. También es una buena práctica incluir una coma después del último par clave-valor, de modo que esté listo para agregar un nuevo par clave-valor en la siguiente línea.

NOTA mayoría de los editores tienen alguna funcionalidad que le ayuda a formatear listas y diccionarios extendidos. cionarios de manera similar a este ejemplo. También hay disponibles otras formas aceptables de formatear diccionarios largos, por lo que es posible que veas un formato ligeramente diferente en tu editor o en otras fuentes.

Para utilizar este diccionario, dado el nombre de una persona que realizó la encuesta, puedes buscar fácilmente su idioma favorito:

favorito
_idiomas.py

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}
```

```
1 idioma = idiomas_favoritos['sarah'].título()
print(f"El idioma favorito de Sarah es {idioma}.")
```

Para ver qué idioma eligió Sarah, solicitamos el valor en:

```
idiomas_favoritos['sarah']
```

Usamos esta sintaxis para extraer el idioma favorito de Sarah del diccionario 1 y asignarlo a la variable idioma. Crear una nueva variable aquí hace que la llamada print() sea mucho más limpia. El resultado muestra el idioma favorito de Sarah:

El idioma favorito de Sarah es el C.

Podrías usar esta misma sintaxis con cualquier individuo representado en el diccionario.

Usando get() para acceder a valores

Usar claves entre corchetes para recuperar el valor que le interesa de un diccionario puede causar un problema potencial: si la clave que solicita no existe, obtendrá un error.

Veamos qué sucede cuando preguntas el valor en puntos de un extraterrestre que no tiene un valor en puntos establecido:

```
extraterrestre_no
_puntos.py
alien_0 = {'color': 'verde', 'velocidad': 'lento'}
imprimir(alien_0['puntos'])
```

Esto da como resultado un rastreo que muestra un `KeyError`:

Rastreo (llamadas recientes más última):

```
Archivo "alien_no_puntos.py", línea 2, en <módulo>
  imprimir(alien_0['puntos'])
  ~~~~~^~~~~~
```

`KeyError: 'puntos'`

Aprenderá más sobre cómo manejar errores como este en general en el Capítulo 10. Para diccionarios específicamente, puede usar el método `get()` para establecer un valor predeterminado que se devolverá si la clave solicitada no existe.

El método `get()` requiere una clave como primer argumento. Como segunda opción argumento, puede pasar el valor que se devolverá si la clave no existe:

```
alien_0 = {'color': 'verde', 'velocidad': 'lento'}

point_value = alien_0.get('puntos', 'No se ha asignado ningún valor de puntos.')
imprimir(valor_punto)
```

Si los 'puntos' clave existen en el diccionario, obtendrá el valor correspondiente. Si no es así, obtendrá el valor predeterminado. En este caso, los puntos no existen y obtenemos un mensaje limpio en lugar de un error:

```
No se asigna ningún valor de puntos.
```

Si existe la posibilidad de que la clave que estás solicitando no exista, considera usando el método `get()` en lugar de la notación entre corchetes.

NOTA omite el segundo argumento en la llamada a `get()` y la clave no existe, Python devolverá el valor `Ninguno`. El valor especial `Ninguno` significa "no existe ningún valor".

Esto no es un error: es un valor especial destinado a indicar la ausencia de un valor.

Verás más usos de `Ninguno` en el Capítulo 8.

INTÉNTALO TÚ MISMO

6-1. Persona: utilice un diccionario para almacenar información sobre una persona que conoce.

Almacene su nombre, apellido, edad y la ciudad en la que vive. Debe tener claves como nombre, apellido, edad y ciudad. Imprime cada pieza de información almacenada en tu diccionario.

6-2. Números favoritos: use un diccionario para almacenar los números favoritos de las personas.

Piensa en cinco nombres y úsalos como claves en tu diccionario. Piensa en un favorito

número para cada persona y almacene cada uno como un valor en su diccionario. Imprime el nombre de cada persona y su número favorito. Para divertirse aún más, encuesta a algunos amigos y obtén datos reales para tu programa.

6-3. Glosario: Se puede utilizar un diccionario de Python para modelar un diccionario real.

Sin embargo, para evitar confusiones, llamémoslo glosario.

- Piensa en cinco palabras de programación que has aprendido en los capítulos anteriores. Utilice estas palabras como claves en su glosario y almacene sus significados como valores.
- Imprima cada palabra y su significado como resultado cuidadosamente formateado. tu podrías imprimir la palabra seguida de dos puntos y luego su significado, o imprima la palabra en una línea y luego imprima su significado con sangría en una segunda línea. Utilice el carácter de nueva línea (\n) para insertar una línea en blanco entre cada par de palabra y significado en su salida.

Recorriendo un diccionario

Un único diccionario de Python puede contener sólo unos pocos pares clave-valor o millones de pares. Debido a que un diccionario puede contener grandes cantidades de datos, Python le permite recorrer un diccionario. Los diccionarios se pueden utilizar para almacenar información de diversas formas; por lo tanto, existen varias formas diferentes de recorrerlos. Puede recorrer todos los pares clave-valor de un diccionario, sus claves o sus valores.

Recorriendo todos los pares clave-valor

Antes de explorar los diferentes enfoques del bucle, consideremos un nuevo diccionario diseñado para almacenar información sobre un usuario en un sitio web. El siguiente diccionario almacenaría el nombre de usuario, el nombre y el apellido de una persona:

```
usuario.py usuario_0 = {
    'nombre de usuario': 'efermi',
    'primero': 'enrico',
    'último': 'fermi',
}
```

Puede acceder a cualquier información sobre usuario_0 basándose en lo que ya ha aprendido en este capítulo. Pero, ¿qué pasaría si quisieras ver todo lo almacenado en el diccionario de este usuario? Para hacerlo, puedes recorrer el diccionario usando un bucle for :

```
usuario_0 = {
    'nombre de usuario': 'efermi',
    'primero': 'enrico',
    'último': 'fermi',
}
```

```

para clave, valor en user_0.items():
    imprimir(f"\nClave: {clave}")
    imprimir(f"Valor: {valor}")

```

Para escribir un bucle for para un diccionario, debe crear nombres para las dos variables que contendrán la clave y el valor en cada par clave-valor. Puede elegir los nombres que desee para estas dos variables. Este código funcionaría igual de bien si hubiera usado abreviaturas para los nombres de las variables, como esta:

```

para k, v en user_0.items()

```

La segunda mitad de la declaración for incluye el nombre del diccionario seguido del método items(), que devuelve una secuencia de pares clave-valor. Luego, el bucle for asigna cada uno de estos pares a las dos variables proporcionadas. En el ejemplo anterior, usamos las variables para imprimir cada clave, seguidas del valor asociado. El "\n" en la primera llamada print() garantiza que se inserte una línea en blanco antes de cada par clave-valor en la salida:

```

Clave: nombre de usuario
Valor: efermi

```

```

Clave: primero
Valor: enrico

```

```

Clave: último
Valor: fermi

```

Recorrer todos los pares clave-valor funciona particularmente bien para diccionarios como el ejemplo de favourite_languages.py en la página 96, que almacena el mismo tipo de información para muchas claves diferentes. Si recorre el diccionario de idiomas_favoritos, obtendrá el nombre de cada persona en el diccionario y su lenguaje de programación favorito. Debido a que las claves siempre se refieren al nombre de una persona y el valor es siempre un idioma, usaremos las variables nombre e idioma en el bucle en lugar de clave y valor. Esto hará que sea más fácil seguir lo que sucede dentro del bucle:

favorito
_idiomas.py

```

idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}

```

```

para nombre, idioma en favourite_languages.items():
    print(f"{name.title()} el idioma favorito es {language.title()}")

```

Este código le dice a Python que recorra cada par clave-valor en el diccionario. A medida que funciona con cada par, la clave se asigna al nombre de la variable y el valor se asigna al idioma de la variable. Estos nombres descriptivos hacen que sea mucho más fácil ver qué está haciendo la llamada print().

Ahora, en tan sólo unas pocas líneas de código, podemos mostrar toda la información de la encuesta:

```
El lenguaje favorito de Jen es Python.
El idioma favorito de Sarah es el C.
El idioma favorito de Edward es Rust.
El lenguaje favorito de Phil es Python.
```

Este tipo de bucle funcionaría igual de bien si nuestro diccionario almacenara los resultados de una encuesta de mil o incluso un millón de personas.

Recorriendo todas las claves de un diccionario

El método llaves() es útil cuando no es necesario trabajar con todos los valores de un diccionario. Repasemos el diccionario de idiomas_favoritos e imprimamos los nombres de todos los que participaron en la encuesta:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}
```

```
para el nombre en favourite_languages.keys():
    imprimir(nombre.título())
```

Este bucle for le dice a Python que extraiga todas las claves del diccionario favorito _languages y asígneles uno a la vez al nombre de la variable. El resultado muestra los nombres de todos los que realizaron la encuesta:

```
Jen
sara
Eduardo
filo
```

Recorrer las claves es en realidad el comportamiento predeterminado cuando se recorre un diccionario, por lo que este código tendría exactamente el mismo resultado si escribiera:

```
para el nombre en idiomas_favoritos:
```

en vez de:

```
para el nombre en favourite_languages.keys():
```

Puede optar por utilizar el método claves() explícitamente si hace que su código sea más fácil de leer, o puede omitirlo si lo desea.

Puede acceder al valor asociado con cualquier clave que le interese dentro del bucle, utilizando la clave actual. Imprimamos un mensaje para un par de amigos sobre los idiomas que eligieron. Repasaremos los nombres en

el diccionario como lo hicimos anteriormente, pero cuando el nombre coincida con uno de nuestros amigos, mostraremos un mensaje sobre su idioma favorito:

```

idiomas_favoritos = {
    --recorte--
}

amigos = ['phil', 'sarah']
para el nombre en favourite_languages.keys():
    print(f"Hola {nombre.título()}")

1 si nombre en amigos:
2     idioma = idiomas_favoritos[nombre].título()
    print(f"\t{nombre.título()}, ¡veo que te encanta {idioma}!")

```

Primero, hacemos una lista de amigos a los que queremos imprimir un mensaje. Adentro En el bucle, imprimimos el nombre de cada persona. Luego comprobamos si el nombre con el que estamos trabajando está en la lista de amigos 1. Si es así, determinamos el idioma favorito de la persona usando el nombre del diccionario y el valor actual de nombre como clave 2. Luego imprimimos un saludo especial, que incluye una referencia a su idioma de elección.

El nombre de todos está impreso, pero nuestros amigos reciben un mensaje especial:

```

Hola Jen.
Hola Sarah.
    Sarah, veo que amas a C!
Hola Eduardo.
Hola Phil.
    Phil, ¡veo que te encanta Python!

```

También puede utilizar el método `claves()` para averiguar si una persona en particular fue encuestada. Esta vez, averigüemos si Erin participó en la encuesta:

```

idiomas_favoritos = {
    --recorte--
}

si 'erin' no está en favourite_languages.keys():
    print ("¡Erin, por favor participa en nuestra encuesta!")

```

El método `llaves()` no es sólo para bucles: en realidad devuelve una secuencia de todas las claves, y la declaración `if` simplemente verifica si 'erin' está en esta secuencia. Como no lo es, se imprime un mensaje invitándola a realizar la encuesta:

```

Erin, ¡responde nuestra encuesta!

```

Recorrer las claves de un diccionario en un orden particular

Al recorrer un diccionario, se devuelven los elementos en el mismo orden en que se insertaron. A veces, sin embargo, querrás recorrer un diccionario en un orden diferente.

Una forma de hacer esto es ordenar las claves a medida que se devuelven en el bucle for .
Puede utilizar la función sorted() para obtener una copia de las claves en orden:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}
```

```
para nombre en ordenado(favorite_languages.keys()):
    print(f"{name.title()}, gracias por realizar la encuesta.")
```

Esta declaración for es como otras declaraciones for , excepto que hemos incluido la función sorted() alrededor del método Dictionary.keys() . Esto le dice a Python que obtenga todas las claves del diccionario y las ordene antes de iniciar el ciclo. El resultado muestra a todos los que realizaron la encuesta, con los nombres mostrados en orden:

```
Edward, gracias por realizar la encuesta.
Jen, gracias por realizar la encuesta.
Phil, gracias por realizar la encuesta.
Sarah, gracias por realizar la encuesta.
```

Recorriendo todos los valores en un diccionario

Si está interesado principalmente en los valores que contiene un diccionario, puede utilizar el método value() para devolver una secuencia de valores sin claves. Por ejemplo, digamos que simplemente queremos una lista de todos los idiomas elegidos en nuestra encuesta de lenguajes de programación, sin el nombre de la persona que eligió cada idioma:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'edward': 'óxido',
    'phil': 'pitón',
}
```

```
print("Se han mencionado los siguientes idiomas:")
para el idioma en favourite_languages.values():
    imprimir(idioma.título())
```

La declaración for aquí extrae cada valor del diccionario y lo asigna a la variable idioma. Cuando se imprimen estos valores, obtenemos una lista de todos los idiomas elegidos:

```
Se han mencionado los siguientes idiomas:
Pitón
c
Óxido
Pitón
```

Este enfoque extrae todos los valores del diccionario sin comprobar si hay repeticiones. Esto podría funcionar bien con una pequeña cantidad de valores, pero en una encuesta con una gran cantidad de encuestados, daría como resultado una lista muy repetitiva. Para ver cada idioma elegido sin repetición, podemos utilizar un conjunto. Un conjunto es una colección en la que cada elemento debe ser único:

```
idiomas_favoritos = {
    --recorte--
}

print("Se han mencionado los siguientes idiomas:")
para el idioma en el conjunto (favorite_languages.values()):
    imprimir(idioma.título())
```

Cuando envuelve set() alrededor de una colección de valores que contiene elementos duplicados, Python identifica los elementos únicos en la colección y construye un conjunto a partir de esos elementos. Aquí usamos set() para extraer los idiomas únicos en favourite_languages.values().

El resultado es una lista no repetitiva de idiomas que se han mencionado.
por personas que participaron en la encuesta:

Se han mencionado los siguientes idiomas:

```
Pitón
C
Óxido
```

A medida que continúas aprendiendo sobre Python, a menudo encontrarás una característica incorporada. del lenguaje que le ayuda a hacer exactamente lo que quiere con sus datos.

NOTA Puedes construir un conjunto directamente usando llaves y separando los elementos con comas:

```
>>> idiomas = {'python', 'rust', 'python', 'c'}
>>> idiomas
{'óxido', 'pitón', 'c'}
```

Es fácil confundir conjuntos con diccionarios porque ambos están entre llaves. Cuando ve llaves pero no pares clave-valor, probablemente esté viendo un conjunto. A diferencia de las listas y los diccionarios, los conjuntos no conservan los elementos en ningún orden específico.

INTÉNTALO TÚ MISMO

6-4. Glosario 2: ahora que sabe cómo recorrer un diccionario en bucle, limpie el código del Ejercicio 6-3 (página 99) reemplazando su serie de print() llamadas con un bucle que recorre las claves y valores del diccionario. Cuando esté seguro de que su bucle funciona, agregue cinco términos más de Python a su glosario. Cuando vuelva a ejecutar su programa, estas nuevas palabras y significados deberían incluirse automáticamente en el resultado.

6-5. Ríos: haz un diccionario que contenga tres ríos principales y el país por el que pasa cada río. Un par clave-valor podría ser 'nilo': 'egipto'.

- Utilice un bucle para imprimir una oración sobre cada río, como El Nilo atraviesa Egipto.
- Utilice un bucle para imprimir el nombre de cada río incluido en el diccionario.
- Utilice un bucle para imprimir el nombre de cada país incluido en el diccionario.

6-6. Encuesta: use el código en `favourite_languages.py` (página 96).

- Haga una lista de personas que deberían realizar la encuesta de idiomas favoritos. Incluye algunos nombres que ya están en el diccionario y otros que no.
- Recorra la lista de personas que deberían realizar la encuesta. Si ya realizaron la encuesta, imprima un mensaje agradeciéndoles por responder. Si aún no han realizado la encuesta, imprima un mensaje invitándolos a realizar la encuesta.

Anidación

A veces querrás almacenar varios diccionarios en una lista o una lista de elementos como un valor en un diccionario. A esto se le llama anidamiento. Puede anidar diccionarios dentro de una lista, una lista de elementos dentro de un diccionario o incluso un diccionario dentro de otro diccionario. El anidamiento es una característica poderosa, como lo demostrarán los siguientes ejemplos.

Una lista de diccionarios

El diccionario `alien_0` contiene una variedad de información sobre un extraterrestre, pero no tiene espacio para almacenar información sobre un segundo extraterrestre, y mucho menos una pantalla llena de extraterrestres. ¿Cómo puedes gestionar una flota de extraterrestres? Una forma es hacer una lista de extraterrestres en la que cada extraterrestre sea un diccionario de información sobre ese extraterrestre. Por ejemplo, el siguiente código crea una lista de tres extraterrestres:

```
aliens.py alien_0 = {'color': 'verde', 'puntos': 5}
alien_1 = {'color': 'amarillo', 'puntos': 10}
alien_2 = {'color': 'rojo', 'puntos': 15}

1 alienígenas = [alien_0, alien_1, alien_2]
```

para extraterrestre en extraterrestres:

```
imprimir (extraterrestre)
```

Primero creamos tres diccionarios, cada uno de los cuales representa un extraterrestre diferente. Almacenamos cada uno de estos diccionarios en una lista llamada `aliens` 1. Finalmente, recorremos la lista e imprimimos cada alien:

```
{'color': 'verde', 'puntos': 5}
{'color': 'amarillo', 'puntos': 10}
{'color': 'rojo', 'puntos': 15}
```

Un ejemplo más realista implicaría más de tres extraterrestres con un código que genera automáticamente cada extraterrestre. En el siguiente ejemplo, usamos `range()` para crear una flota de 30 alienígenas:

```
# Haz una lista vacía para almacenar extraterrestres.
extraterrestres = []

# Haz 30 alienígenas verdes.
1 para alien_number en el rango (30):
2 new_alien = {'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
3 extraterrestres.append(new_alien)

# Muestra los primeros 5 extraterrestres.
4 para extraterrestre en extraterrestres[:5]:
    imprimir (extraterrestre)
    imprimir("...")

# Muestra cuántos extraterrestres se han creado.
print(f"Número total de alienígenas: {len(aliens)}")
```

Este ejemplo comienza con una lista vacía para contener todos los extraterrestres que se crearán. La función `range()` devuelve una serie de números, que simplemente le dice a Python cuántas veces queremos que se repita el ciclo. Cada vez que se ejecuta el ciclo, creamos un nuevo alienígena 2 y luego agregamos cada nuevo alienígena a la lista `alienígenas` 3. Usamos un segmento para imprimir los primeros cinco alienígenas 4 y, finalmente, imprimimos la longitud de la lista para demostrar que De hecho, hemos generado la flota completa de 30 alienígenas:

```
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
...
```

Número total de extranjeros: 30

Todos estos alienígenas tienen las mismas características, pero Python considera que cada uno es un objeto separado, lo que nos permite modificar cada alienígena individualmente.

¿Cómo podrías trabajar con un grupo de extraterrestres como este? Imagina que en un aspecto de un juego algunos extraterrestres cambian de color y se mueven más rápido a medida que avanza el juego. Cuando llega el momento de cambiar los colores, podemos usar un bucle `for` y una declaración `if` para cambiar el color de los alienígenas. Por ejemplo, para cambiar

los tres primeros alienígenas a amarillo, alienígenas de velocidad media que valen 10 puntos cada uno, podríamos hacer esto:

```
# Haz una lista vacía para almacenar extraterrestres.
extraterrestres = []

# Haz 30 alienígenas verdes.
para alien_number en el rango (30):
    new_alien = {'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
    extraterrestres.append(nuevo_alienígena)

para extraterrestre en extraterrestres[:3]:
    si alienígena['color'] == 'verde':
        alienígena['color'] = 'amarillo'
        alienígena['velocidad'] = 'media'
        alienígena['puntos'] = 10

# Muestra los primeros 5 extraterrestres.
para extraterrestre en extraterrestres[:5]:
    imprimir (extraterrestre)
imprimir("...")
```

Como queremos modificar los primeros tres alienígenas, recorremos un segmento que incluye solo los primeros tres alienígenas. Todos los alienígenas son verdes ahora, pero ese no siempre será el caso, por lo que escribimos una declaración if para asegurarnos de que solo estamos modificando alienígenas verdes. Si el alienígena es verde, cambiamos el color a "amarillo", la velocidad a "media" y el valor de puntos a 10, como se muestra en el siguiente resultado:

```
{'color': 'amarillo', 'puntos': 10, 'velocidad': 'media'}
{'color': 'amarillo', 'puntos': 10, 'velocidad': 'media'}
{'color': 'amarillo', 'puntos': 10, 'velocidad': 'media'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
{'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
...
```

Podrías expandir este ciclo agregando un bloque elif que convierta a los alienígenas amarillos en rojos que se mueven rápidamente y valen 15 puntos cada uno. Sin volver a mostrar el programa completo, ese bucle se vería así:

```
para extraterrestre en extraterrestres[0:3]:
    si alienígena['color'] == 'verde':
        alienígena['color'] = 'amarillo'
        alienígena['velocidad'] = 'media'
        alienígena['puntos'] = 10
    elif alienígena['color'] == 'amarillo':
        alienígena['color'] = 'rojo'
        alienígena['velocidad'] = 'rápido'
        alienígena['puntos'] = 15
```

Es común almacenar varios diccionarios en una lista cuando cada diccionario

El diccionario contiene muchos tipos de información sobre un objeto. Por ejemplo, podrías crear un diccionario para cada usuario en un sitio web, como hicimos en user.py.

en la página 99 y almacene los diccionarios individuales en una lista llamada usuarios. Todos los diccionarios de la lista deben tener una estructura idéntica, de modo que pueda recorrer la lista y trabajar con cada objeto del diccionario de la misma manera.

Una lista en un diccionario

En lugar de poner un diccionario dentro de una lista, a veces resulta útil poner una lista dentro de un diccionario. Por ejemplo, considere cómo describiría una pizza que alguien está pidiendo. Si usaras solo una lista, todo lo que realmente podrías almacenar es una lista de los ingredientes de la pizza. Con un diccionario, una lista de ingredientes puede ser sólo un aspecto de la pizza que estás describiendo.

En el siguiente ejemplo, se almacenan dos tipos de información para cada pizza: un tipo de corteza y una lista de ingredientes. La lista de ingredientes es un valor asociado con la clave 'ingredientes'. Para utilizar los elementos de la lista, damos el nombre del diccionario y la clave 'ingredientes', como lo haríamos con cualquier valor del diccionario. En lugar de devolver un valor único, obtenemos una lista de ingredientes:

```

pizza.py # Almacena información sobre una pizza que se pide.

pizza = {
    'corteza': 'gruesa',
    'ingredientes': ['champiñones', 'queso extra'],
}

# Resume el pedido.
1 print(f"Pediste una pizza con masa de {pizza['crust']} "con los siguientes
    ingredientes:")

2 para cubrir la pizza['ingredientes']:
    imprimir(f"\t{superior}")

```

Comenzamos con un diccionario que contiene información sobre una pizza que se ha pedido. Una clave en el diccionario es "corteza" y el valor asociado es la cadena "gruesa". La siguiente clave, 'ingredientes', tiene una lista como valor que almacena todos los ingredientes solicitados. Resumimos el orden antes de construir la pizza 1. Cuando necesite dividir una línea larga en una llamada print(), elija un punto apropiado para dividir la línea que se está imprimiendo y finalice la línea con una comilla. Sangra la siguiente línea, agrega una comilla de apertura y continúa la cadena. Python combinará automáticamente todas las cadenas que encuentre entre paréntesis. Para imprimir los toppings, escribimos un for bucle 2. Para acceder a la lista de ingredientes, usamos la clave 'ingredientes' y Python toma la lista de ingredientes del diccionario.

El siguiente resultado resume la pizza que planeamos construir:

```

Pediste una pizza de masa gruesa con los siguientes ingredientes:
    hongos
    extra queso

```

Puede anidar una lista dentro de un diccionario en cualquier momento que desee asociar más de un valor con una única clave en un diccionario. En el ejemplo anterior de los lenguajes de programación favoritos, si almacenáramos las respuestas de cada persona en una lista, la gente podría elegir más de un lenguaje favorito.

Cuando recorremos el diccionario, el valor asociado con cada persona sería una lista de idiomas en lugar de un solo idioma. Dentro del bucle for del diccionario, utilizamos otro bucle for para recorrer la lista de idiomas asociados con cada persona:

```

favorito
_idiomas.py    idiomas_favoritos = {
                'jen': ['python', 'óxido'],
                'sara': ['c'],
                'edward': ['óxido', 'ir'],
                'phil': ['python', 'haskell'],
                }

1 para nombre, idiomas en favourite_languages.items():
    print(f"\n{name.title()} los idiomas favoritos de {name.title()} son:")
2 para lenguaje en idiomas:
    imprimir(f"\t{idioma.título()}")

```

El valor asociado con cada nombre en `idiomas_favoritos` ahora es una lista. Tenga en cuenta que algunas personas tienen un idioma favorito y otras tienen varios favoritos. Cuando recorremos el diccionario 1, usamos los idiomas de nombres de variables para contener cada valor del diccionario, porque sabemos que cada valor será una lista. Dentro del bucle del diccionario principal, utilizamos otro bucle for 2 para recorrer la lista de idiomas favoritos de cada persona. Ahora cada persona puede enumerar tantos idiomas favoritos como quiera:

```

Los idiomas favoritos de Jen son:
    Pitón
    Óxido

Los idiomas favoritos de Sarah son:
    C

Los idiomas favoritos de Edward son:
    Óxido
    Ir

Los idiomas favoritos de Phil son:
    Pitón
    Haskell

```

Para refinar aún más este programa, podría incluir una declaración if al comienzo del bucle for del diccionario para ver si cada persona tiene más de un idioma favorito examinando el valor de `len(languages)`. Si una persona tiene más de un favorito, el resultado seguirá siendo el mismo. Si la persona tiene solo un idioma favorito, puedes cambiar la redacción para reflejarlo. Por ejemplo, podría decir: "El idioma favorito de Sarah es C".

NOTA No debe anidar listas y diccionarios demasiado profundamente. Si está anidando elementos mucho más profundamente que lo que ve en los ejemplos anteriores, o si está trabajando con el código de otra persona con niveles significativos de anidamiento, lo más probable es que exista una forma más sencilla de resolver el problema.

Un diccionario en un diccionario

Puedes anidar un diccionario dentro de otro diccionario, pero tu código puede complicarse rápidamente cuando lo haces. Por ejemplo, si tiene varios usuarios para un sitio web, cada uno con un nombre de usuario único, puede utilizar los nombres de usuario como claves en un diccionario. Luego puede almacenar información sobre cada usuario utilizando un diccionario como valor asociado con su nombre de usuario. En el siguiente listado, almacenamos tres datos sobre cada usuario: su nombre, apellido y ubicación. Accederemos a esta información recorriendo los nombres de usuario y el diccionario de información asociado con cada nombre de usuario:

```

muchos_usuarios.py usuarios = {
    'aeinstein': {
        'primero': 'alberto',
        'último': 'einstein',
        'ubicación': 'princeton',
    },

    'mccurie': {
        'primero': 'marie',
        'último': 'curie',
        'ubicación': 'París',
    },

}

1 para nombre de usuario, información_usuario en usuarios.items():
2 print(f"\nNombre de usuario: {nombre de usuario}")
3 nombre_completo = f"{info_usuario['primero']} {info_usuario['último']}"
  ubicación = información_usuario['ubicación']

4 print(f"\tNombre completo: {nombre_completo.título()}")
  print(f"\tUbicación: {ubicación.título()}")

```

Primero definimos un diccionario llamado usuarios con dos claves: una para cada nombres de usuario 'aeinstein' y 'mcurie'. El valor asociado con cada clave es un diccionario que incluye el nombre, apellido y ubicación de cada usuario. Luego, recorreremos el diccionario de usuarios 1. Python asigna cada clave a la variable nombre de usuario y el diccionario asociado con cada nombre de usuario se asigna a la variable información de usuario. Una vez dentro del bucle del diccionario principal, imprimimos el nombre de usuario 2.

Luego, comenzamos a acceder al diccionario interno 3. La variable user_info, que contiene el diccionario de información del usuario, tiene tres claves: 'primero', 'último' y 'ubicación'. Usamos cada clave para generar un nombre completo y una ubicación cuidadosamente formateados para cada persona, y luego imprimimos un resumen de lo que sabemos sobre cada usuario 4:

```

Nombre de usuario: aeinstein
Nombre completo: Albert Einstein
Ubicación: Princeton

```


Nombre de usuario: mcurie

Nombre completo: Marie Curie

Ubicación: París

Observe que la estructura del diccionario de cada usuario es idéntica. Aunque Python no la requiere, esta estructura hace que sea más fácil trabajar con diccionarios anidados. Si el diccionario de cada usuario tuviera claves diferentes, el código dentro del bucle `for` sería más complicado.

INTÉNTALO TÚ MISMO

6-7. Personas: Comience con el programa que escribió para el Ejercicio 6-1 (página 98). Cree dos diccionarios nuevos que representen a diferentes personas y almacene los tres diccionarios en una lista llamada `personas`. Recorre tu lista de personas. A medida que recorra la lista, imprima todo lo que sepa sobre cada persona.

6-8. Mascotas: cree varios diccionarios, donde cada diccionario represente una mascota diferente. En cada diccionario incluye la clase de animal y el nombre del dueño. Guarde estos diccionarios en una lista llamada `mascotas`. A continuación, recorra su lista y, mientras lo hace, imprima todo lo que sabe sobre cada mascota.

6-9. Lugares favoritos: crea un diccionario llamado `lugares_favoritos`. Piense en tres nombres para utilizarlos como claves en el diccionario y guarde de uno a tres lugares favoritos de cada persona. Para que este ejercicio sea un poco más interesante, pídele a algunos amigos que mencionen algunos de sus lugares favoritos. Recorre el diccionario e imprime el nombre de cada persona y sus lugares favoritos.

6-10. Números favoritos: Modifique su programa del Ejercicio 6-2 (página 98) para que cada persona pueda tener más de un número favorito. Luego imprima el nombre de cada persona junto con sus números favoritos.

6-11. Ciudades: Haz un diccionario llamado `ciudades`. Usa los nombres de tres ciudades como claves en tu diccionario. Cree un diccionario de información sobre cada ciudad e incluya el país en el que se encuentra la ciudad, su población aproximada y un dato sobre esa ciudad. Las claves del diccionario de cada ciudad deberían ser algo como `país`, `población` y `hecho`. Imprime el nombre de cada ciudad y toda la información que tengas almacenada sobre ella.

6-12. Extensiones: ahora estamos trabajando con ejemplos que son lo suficientemente complejos como para poder ampliarlos de varias maneras. Utilice uno de los programas de ejemplo de este capítulo y amplíelo agregando nuevas claves y valores, cambiando el contexto del programa o mejorando el formato de la salida.

Resumen

En este capítulo, aprendió cómo definir un diccionario y cómo trabajar con la información almacenada en un diccionario. Aprendiste cómo acceder y modificar elementos individuales en un diccionario, y cómo recorrer todos

de la información en un diccionario. Aprendiste a recorrer los pares clave-valor de un diccionario, sus claves y sus valores. También aprendió a anidar varios diccionarios en una lista, anidar listas en un diccionario y anidar un diccionario dentro de un diccionario.

En el próximo capítulo aprenderá sobre los bucles while y cómo aceptar entradas de personas que utilizan sus programas. Este será un capítulo emocionante, porque aprenderá a hacer que todos sus programas sean interactivos: podrán responder a las entradas del usuario.

7

ENTRADA DEL USUARIO Y BUCLES MIENTRAS



La mayoría de los programas están escritos para resolver el problema del usuario final. Para hacerlo, normalmente necesita obtener cierta información del usuario. Por ejemplo, supongamos que alguien quiere saber si tiene edad suficiente para votar. Si escribe un programa para responder esta pregunta, necesita saber la edad del usuario antes de poder dar una respuesta. El programa deberá pedirle al usuario que ingrese su edad; Una vez que el programa tiene esta entrada, puede compararla con la edad para votar para determinar si el usuario tiene la edad suficiente y luego informar el resultado.

En este capítulo aprenderá cómo aceptar la entrada del usuario para que su programa pueda trabajar con ella. Cuando su programa necesite un nombre, podrá solicitarle un nombre al usuario. Cuando su programa necesite una lista de nombres, podrá solicitarle al usuario una serie de nombres. Para hacer esto, usará la función `input()`.

También aprenderá cómo mantener los programas ejecutándose durante el tiempo que los usuarios quieran, para que puedan ingresar tanta información como necesiten; entonces tu

El programa puede trabajar con esa información. Utilizará el bucle while de Python para mantener los programas ejecutándose mientras ciertas condiciones sigan siendo verdaderas.

Con la capacidad de trabajar con la entrada del usuario y la capacidad de controlar cuánto tiempo se ejecutan sus programas, podrá escribir programas completamente interactivos.

Cómo funciona la función input()

La función input() pausa su programa y espera a que el usuario ingrese algún texto. Una vez que Python recibe la entrada del usuario, asigna esa entrada a una variable para que le resulte más cómodo trabajar con ella.

Por ejemplo, el siguiente programa le pide al usuario que ingrese algún texto y luego le muestra ese mensaje:

```
parrot.py mensaje = input("Dime algo y te lo repetiré: ")
imprimir (mensaje)
```

La función input() toma un argumento: el mensaje que queremos mostrarle al usuario, para que sepa qué tipo de información ingresar. En este ejemplo, cuando Python ejecuta la primera línea, el usuario ve el mensaje Dime algo y te lo repetiré: . El programa espera mientras el usuario ingresa su respuesta y continúa después de que el usuario presiona ENTER. La respuesta se asigna a la variable mensaje, luego print(message) muestra la entrada al usuario:

```
Dime algo y te lo repetiré: ¡Hola a todos!
¡Hola a todos!
```

NOTA Algunos editores de texto no ejecutan programas que solicitan información al usuario. Puede utilizar estos editores para escribir programas que soliciten entradas, pero necesitará ejecutar estos programas desde una terminal. Consulte "Ejecución de programas Python desde una terminal" en la página 11.

Escribir indicaciones claras

Cada vez que utilice la función input() , debe incluir un mensaje claro y fácil de seguir que le indique al usuario exactamente qué tipo de información está buscando. Cualquier declaración que le diga al usuario qué ingresar debería funcionar. Por ejemplo:

```
greeter.py nombre = input("Ingrese su nombre: ")
print(f"\nHola, {nombre}!")
```

Agregue un espacio al final de sus mensajes (después de los dos puntos en el ejemplo anterior) para separar el mensaje de la respuesta del usuario y dejarle claro al usuario dónde ingresar su texto. Por ejemplo:

```
Por favor introduce tu nombre: Eric
¡Hola Eric!
```

A veces querrás escribir un mensaje de más de una línea.

Por ejemplo, es posible que desee decirle al usuario por qué solicita determinada información. Puede asignar su mensaje a una variable y pasar esa variable a la función `input()`. Esto le permite crear su mensaje en varias líneas y luego escribir una declaración `input()` limpia.

```
greeter.py Prompt = "Si comparte su nombre, podemos personalizar los mensajes que ve."
```

```
mensaje += "\n¿Cuál es tu nombre?"
```

```
nombre = entrada (mensaje)
```

```
print(f"\nHola, {nombre}!")
```

Este ejemplo muestra una forma de crear una cadena multilínea. La primera línea asigna la primera parte del mensaje al mensaje variable. En la segunda línea, el operador `+=` toma la cadena asignada al mensaje y agrega la nueva cadena al final.

El mensaje ahora abarca dos líneas, nuevamente con un espacio después del signo de interrogación para mayor claridad:

```
Si comparte su nombre, podemos personalizar los mensajes que ve.
```

```
¿Cuál es tu primer nombre? eric
```

```
¡Hola Eric!
```

Usando `int()` para aceptar entradas numéricas

Cuando usas la función `input()`, Python interpreta todo lo que el usuario ingresa como una cadena. Considere la siguiente sesión de intérprete, que pregunta la edad del usuario:

```
>>> edad = entrada("¿Cuántos años tienes? ")
```

```
¿Cuántos años tiene? 21
```

```
>>> edad
```

```
'21'
```

El usuario ingresa el número 21, pero cuando le preguntamos a Python el valor de edad, devuelve '21', la representación en cadena del valor numérico ingresado. Sabemos que Python interpretó la entrada como una cadena porque el número ahora está entre comillas. Si todo lo que quieres hacer es imprimir la entrada, esto funciona bien. Pero si intentas utilizar la entrada como un número, obtendrás un error:

```
>>> edad = entrada("¿Cuántos años tienes? ")
```

```
¿Cuántos años tiene? 21
```

```
1 >>> edad >= 18
```

```
Rastreo (llamadas recientes más última):
```

```
Archivo "<stdin>", línea 1, en <módulo>
```

```
2 TypeError: '>=' no se admite entre instancias de 'str' e 'int'
```

Cuando intentas usar la entrada para hacer una comparación numérica 1, Python produce un error porque no puede comparar una cadena con un número entero: la cadena '21' que está asignada a la edad no se puede comparar con el valor numérico 18 2.

Podemos resolver este problema usando la función `int()`, que convierte la cadena de entrada en un valor numérico. Esto permite que la comparación se ejecute correctamente:

```
>>> edad = entrada("¿Cuántos años tienes? ")
¿Cuántos años tiene? 21
1 >>> edad = int(edad)
>>> edad >= 18
Verdadero
```

En este ejemplo, cuando ingresamos 21 en el mensaje, Python interpreta el número como una cadena, pero luego `int()` 1 convierte el valor a una representación numérica. Ahora Python puede ejecutar la prueba condicional: compara la edad (que ahora representa el valor numérico 21) y 18 para ver si la edad es mayor o igual a 18. Esta prueba se evalúa como Verdadera.

¿ Cómo se utiliza la función `int()` en un programa real? Considere un programa que determine si las personas son lo suficientemente altas para andar en patineta. portavastos:

```
montaña rusa.py altura = entrada ("¿Qué altura tienes, en pulgadas?")
altura = int(altura)

si altura >= 48:
    print("\n¡Eres lo suficientemente alto para montar!")
demás:
    print("\nPodrás montar en bicicleta cuando seas un poco mayor.")
```

El programa puede comparar la altura con 48 porque `altura = int(altura)` convierte el valor de entrada en una representación numérica antes de realizar la comparación. Si el número ingresado es mayor o igual a 48, le decimos al usuario que es lo suficientemente alto:

```
¿Qué altura tienes, en pulgadas? 71
¡Eres lo suficientemente alto para montar!
```

Cuando utilice entradas numéricas para realizar cálculos y comparaciones, tenga en cuenta asegúrese de convertir primero el valor de entrada a una representación numérica.

El operador de módulo

Una herramienta útil para trabajar con información numérica es el operador de módulo (%), que divide un número entre otro número y devuelve el resto:

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> 7 % 3
1
```

El operador de módulo no le dice cuántas veces cabe un número dentro de otro; sólo te dice cuál es el resto.

Cuando un número es divisible por otro número, el resto es 0, por lo que el operador de módulo siempre devuelve 0. Puedes utilizar este hecho para determinar si un número es par o impar:

```
par_o_impar.py número = input("Ingresa un número y te diré si es par o impar: ")
número = int(número)

si número % 2 == 0:
    print(f"\nEl número {número} es par.")
demás:
    print(f"\nEl número {número} es impar.")
```

Los números pares siempre son divisibles por dos, por lo que si el módulo de un número y dos es cero (aquí, si `número % 2 == 0`) el número es par. De lo contrario, es extraño.

Introduce un número y te diré si es par o impar: 42

El número 42 es par.

INTÉNTALO TÚ MISMO

7-1. Coche de alquiler: escriba un programa que pregunte al usuario qué tipo de coche de alquiler le gustaría. Imprime un mensaje sobre ese automóvil, como "Déjame ver si puedo encontrarte un Subaru".

7-2. Asientos en restaurante: escriba un programa que pregunte al usuario cuántas personas hay en su grupo para cenar. Si la respuesta es más de ocho, imprima un mensaje diciendo que tendrán que esperar por una mesa. En caso contrario, informe que su mesa está lista.

7-3. Múltiplos de diez: solicite al usuario un número y luego informe si el número es múltiplo de 10 o no.

Presentando bucles while

El bucle for toma una colección de elementos y ejecuta un bloque de código una vez para cada elemento de la colección. Por el contrario, el bucle while se ejecuta mientras una determinada condición sea verdadera.

El bucle while en acción

Puede utilizar un bucle while para contar una serie de números. Por ejemplo, el siguiente bucle while cuenta del 1 al 5:

```
contando.py número_actual = 1
mientras número_actual <= 5:
```

```

imprimir(número_actual)
número_actual += 1

```

En la primera línea, comenzamos a contar desde 1 asignando `número_actual` al valor 1. El bucle `while` se configura para seguir ejecutándose siempre que el valor de `número_actual` sea menor o igual a 5. El código dentro del bucle imprime el valor de `número_actual` y luego suma 1 a ese valor con `número_actual += 1`.

(El operador `+=` es una abreviatura de `número_actual = número_actual + 1`.)

Python repite el ciclo siempre que la condición `número_actual <= 5` es verdad. Como 1 es menor que 5, Python imprime 1 y luego suma 1, haciendo que el número actual sea 2. Como 2 es menor que 5, Python imprime 2 y suma 1 nuevamente, haciendo que el número actual sea 3, y así sucesivamente. Una vez que el valor de la corriente `_number` es mayor que 5, el bucle deja de ejecutarse y el programa finaliza:

```

1
2
3
4
5

```

Los programas que utiliza todos los días probablemente contengan bucles `while`. Por ejemplo, un juego necesita un bucle `while` para seguir ejecutándose mientras quieras seguir jugando, por lo que puede dejar de ejecutarse tan pronto como le pidas que salga.

Los programas no serían divertidos de usar si dejaran de ejecutarse antes de que se lo indiquemos o si siguieran ejecutándose incluso después de que quisiéramos salir, por lo que los bucles `while` son bastante útiles.

Permitir que el usuario elija cuándo salir

Podemos hacer que el programa `parrot.py` se ejecute todo el tiempo que el usuario quiera poniendo la mayor parte del programa dentro de un bucle `while`. Definiremos un valor de salida y luego mantendremos el programa ejecutándose mientras el usuario no haya ingresado el valor de salida:

```

parrot.py Prompt = "\nDime algo y te lo repetiré."
Prompt += "\nIngrese 'salir' para finalizar el programa."

"""
mensaje =
mientras mensaje != 'salir':
    mensaje = entrada(mensaje)
    imprimir(mensaje)

```

Primero definimos un mensaje que le indica al usuario sus dos opciones: ingresar un mensaje o ingresar el valor de salida (en este caso, "salir"). Luego configuramos un mensaje variable para realizar un seguimiento de cualquier valor que ingrese el usuario. Definimos `mensaje` como una cadena vacía, "", por lo que Python tiene algo que verificar la primera vez que llega a la línea `while`. La primera vez que se ejecuta el programa y Python llega a la declaración `while`, necesita comparar el valor del `mensaje` con "salir", pero aún no se ha ingresado ninguna entrada del usuario. Si Python no tiene nada con qué comparar, no podrá continuar ejecutando el programa. para resolver esto

problema, nos aseguramos de darle al mensaje un valor inicial. Aunque es sólo una cadena vacía, tendrá sentido para Python y le permitirá realizar la comparación que hace que el ciclo while funcione. Este bucle while se ejecuta siempre que el valor del mensaje no sea "salir".

La primera vez que recorre el bucle, el mensaje es solo una cadena vacía, por lo que Python entra en el bucle. En mensaje = entrada (prompt), Python muestra el mensaje y espera a que el usuario ingrese su entrada. Todo lo que ingresan se asigna a un mensaje y se imprime; luego, Python reevalúa la condición mientras tanto declaración. Mientras el usuario no haya ingresado la palabra "salir", el mensaje se muestra nuevamente y Python espera más entradas. Cuando el usuario finalmente ingresa "salir", Python deja de ejecutar el ciclo while y el programa finaliza:

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. ¡Hola a todos!
 ¡Hola a todos!

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. Hola de nuevo.
 Hola de nuevo.

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. abandonar

abandonar

Este programa funciona bien, excepto que imprime la palabra "salir" como si fuera un mensaje real. Una simple prueba if soluciona este problema:

```
Prompt = "\nDime algo y te lo repetiré:"
Prompt += "\nIngrese 'salir' para finalizar el programa."

mensaje = ""
while mensaje != 'salir':
    mensaje = entrada (mensaje)

    if mensaje != 'salir':
        imprimir (mensaje)
```

Ahora el programa hace una comprobación rápida antes de mostrar el mensaje. y solo imprime el mensaje si no coincide con el valor de salida:

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. ¡Hola a todos!
 ¡Hola a todos!

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. Hola de nuevo.
 Hola de nuevo.

Dime algo y te lo repetiré:
 Ingrese 'salir' para finalizar el programa. abandonar

Usando una bandera

En el ejemplo anterior, hicimos que el programa realizara ciertas tareas mientras una condición determinada era verdadera. Pero ¿qué pasa con los programas más complicados en los que muchos eventos diferentes podrían provocar que el programa deje de ejecutarse?

Por ejemplo, en un juego, varios eventos diferentes pueden finalizar el juego. Cuando el jugador se queda sin barcos, se le acaba el tiempo o las ciudades que se suponía que debían proteger son destruidas, el juego debería terminar. Debe terminar si ocurre alguno de estos eventos. Si pueden ocurrir muchos eventos posibles para detener el programa, intente probar todas estas condiciones de una vez. La declaración se vuelve complicada y difícil.

Para un programa que debe ejecutarse sólo mientras se cumplan muchas condiciones, puede definir una variable que determine si todo el programa está activo o no. Esta variable, llamada bandera, actúa como una señal para el programa. Podemos escribir nuestros programas para que se ejecuten mientras el indicador esté establecido en Verdadero. y dejará de ejecutarse cuando cualquiera de varios eventos establezca el valor de la bandera en Falso. Como resultado, nuestra declaración while general necesita verificar solo una condición: si la bandera es actualmente Verdadera. Luego, todas nuestras otras pruebas (para ver si ocurrió un evento que debería establecer el indicador en Falso) se pueden organizar claramente en el resto del programa.

Agreguemos una bandera a parrot.py de la sección anterior. Este indicador, que llamaremos activo (aunque puedes llamarlo como quieras), controlará si el programa debe continuar ejecutándose o no:

```
Prompt = "\nDime algo y te lo repetiré:"
Prompt += "\nIngresa 'salir' para finalizar el programa.
```

```
activo = Verdadero
1 mientras está activo:
    mensaje = entrada (mensaje)

    si mensaje == 'salir':
        activo = falso
    demás:
        imprimir (mensaje)
```

Establecemos la variable active en True para que el programa comience en un estado activo. Hacerlo simplifica la declaración while porque no se realiza ninguna comparación en la declaración while en sí; la lógica se cuida en otras partes del programa. Mientras la variable activa siga siendo Verdadera, el ciclo continuará ejecutando 1.

En la declaración if dentro del bucle while , verificamos el valor del mensaje una vez que el usuario ingresa su entrada. Si el usuario ingresa "salir", configuramos active en False y el ciclo while se detiene. Si el usuario ingresa algo que no sea "salir", imprimimos su entrada como un mensaje.

Este programa tiene el mismo resultado que el ejemplo anterior donde colocamos la prueba condicional directamente en la declaración while . Pero ahora que nosotros

Si tiene una bandera para indicar si el programa general está en un estado activo, sería fácil agregar más pruebas (como declaraciones elif) para eventos que deberían hacer que active se vuelva falso. Esto es útil en programas complicados como juegos, en los que puede haber muchos eventos que deberían hacer que el programa deje de ejecutarse. Cuando cualquiera de estos eventos hace que la bandera activa se vuelva Falsa, se cerrará el bucle principal del juego, se mostrará un mensaje de Game Over y se le podrá dar al jugador la opción de jugar nuevamente.

Usar break para salir de un bucle

Para salir de un bucle while inmediatamente sin ejecutar ningún código restante en el bucle, independientemente de los resultados de cualquier prueba condicional, utilice la instrucción break . La declaración break dirige el flujo de su programa; puede usarlo para controlar qué líneas de código se ejecutan y cuáles no, de modo que el programa solo ejecute el código que usted desee, cuando así lo desee.

Por ejemplo, considere un programa que pregunta al usuario sobre los lugares que ha visitado. Podemos detener el ciclo while en este programa llamando a break tan pronto como el usuario ingresa el valor 'salir' :

```
ciudades.py Prompt = "\nIngrese el nombre de una ciudad que haya visitado:"
mensaje += "\n(Ingrese 'salir' cuando haya terminado.) "
```

1 mientras es Verdadero:

```
    ciudad = entrada (mensaje)

    si ciudad == 'salir':
        romper
    demás:
        print(f"¡Me encantaría ir a {ciudad.titulo()}!")
```

Un bucle que comienza con while True 1 se ejecutará para siempre a menos que llegue a una declaración de interrupción . El bucle en este programa continúa pidiendo al usuario que ingrese los nombres de las ciudades en las que ha estado hasta que ingresa "salir". Cuando ingresan 'salir', se ejecuta la declaración break , lo que hace que Python salga del ciclo:

```
Por favor ingrese el nombre de una ciudad que haya visitado:
(Ingrese 'salir' cuando haya terminado). Nueva York
¡Me encantaría ir a Nueva York!
```

```
Por favor ingrese el nombre de una ciudad que haya visitado:
(Ingrese 'salir' cuando haya terminado). San Francisco
¡Me encantaría ir a San Francisco!
```

```
Por favor ingrese el nombre de una ciudad que haya visitado:
(Ingrese 'salir' cuando haya terminado) .
```

NOTA Puede utilizar la instrucción break en cualquiera de los bucles de Python. Por ejemplo, podrías usar break para salir de un bucle for que funciona a través de una lista o un diccionario.

Usando continuar en un bucle

En lugar de salir completamente de un bucle sin ejecutar el resto de su código, puede utilizar la instrucción `continue` para volver al principio del bucle, según el resultado de una prueba condicional. Por ejemplo, considere un bucle que cuenta del 1 al 10 pero imprime solo los números impares en ese rango:

```
contando.py número_actual = 0
    mientras número_actual < 10:
1 número_actual += 1
    si número_actual % 2 == 0:
        continuar

    imprimir(número_actual)
```

Primero, configuramos `número_actual` en 0. Como es menor que 10, Python ingresa al ciclo `while`. Una vez dentro del bucle, incrementamos el recuento en 1, por lo que `número_actual` es 1. La instrucción `if` luego verifica el módulo de `número_actual` y 2. Si el módulo es 0 (lo que significa que `número_actual` es divisible por 2), la instrucción `continue` le dice a Python que ignore el resto del ciclo y regrese al principio. Si el número actual no es divisible por 2, se ejecuta el resto del ciclo y Python imprime el número actual:

```
1
3
5
7
9
```

Evitar bucles infinitos

Cada bucle `while` necesita una forma de detener su ejecución para que no continúe ejecutándose para siempre. Por ejemplo, este ciclo de conteo debería contar del 1 al 5:

```
contando.py x = 1
    mientras x <= 5:
        imprimir(x)
        x += 1
```

Sin embargo, si accidentalmente omite la línea `x += 1`, el bucle se ejecutará para siempre:

```
# ¡Este bucle dura para siempre!
x = 1
mientras x <= 5:
    imprimir(x)
```

Ahora el valor de x comenzará en 1 pero nunca cambiará. Como resultado, la prueba condicional $x \leq 5$ siempre se evaluará como Verdadero y el ciclo while se ejecutará para siempre, imprimiendo una serie de unos, como este:

```
1
1
1
1
--recorte--
```

Cada programador escribe accidentalmente un bucle while infinito de vez en cuando, especialmente cuando los bucles de un programa tienen condiciones de salida sutiles. Si su programa se atasca en un bucle infinito, presione CTRL-C o simplemente cierre la ventana de terminal que muestra la salida de su programa.

Para evitar escribir bucles infinitos, pruebe cada bucle while y asegúrese de que el bucle se detiene cuando lo esperas. Si desea que su programa finalice cuando el usuario ingrese un determinado valor de entrada, ejecute el programa e ingrese ese valor. Si el programa no finaliza, examine la forma en que su programa maneja el valor que debería provocar la salida del bucle. Asegúrese de que al menos una parte del programa pueda hacer que la condición del bucle sea Falsa o hacer que llegue a una declaración de interrupción.

NOTA S Code, como muchos editores, muestra el resultado en una ventana de terminal integrada. Para cancelar un bucle infinito, asegúrese de hacer clic en el área de salida del editor antes de presionar CTRL-C.

INTÉNTALO TÚ MISMO

7-4. Ingredientes para pizza: escriba un bucle que solicite al usuario que ingrese una serie de ingredientes para pizza hasta que ingrese un valor para "salir". A medida que ingresan cada ingrediente, imprima un mensaje que indique que agregará ese ingrediente a su pizza.

7-5. Entradas de cine: una sala de cine cobra diferentes precios de entradas según la edad de la persona. Si es menor de 3 años, la entrada es gratuita; si tienen entre 3 y 12 años, el boleto cuesta \$10; y si son mayores de 12 años la entrada cuesta \$15. Escriba un bucle en el que pregunte a los usuarios su edad y luego les diga el coste de su entrada al cine.

7-6. Tres salidas: escriba diferentes versiones del Ejercicio 7-4 o 7-5 que hagan cada uno de los siguientes al menos una vez:

- Utilice una prueba condicional en la sentencia while para detener el ciclo.
- Utilice una variable activa para controlar cuánto tiempo se ejecuta el bucle.
- Utilice una instrucción break para salir del bucle cuando el usuario introduzca un valor de 'salir'.

7-7. Infinito: escriba un bucle que nunca termine y ejecútalo. (Para finalizar el bucle, presione CTRL-C o cierre la ventana que muestra el resultado).

Usando un bucle while con listas y diccionarios

Hasta ahora, hemos trabajado solo con una información del usuario a la vez. Recibimos la entrada del usuario y luego imprimimos la entrada o una respuesta.

La próxima vez que pasemos por el bucle while , recibiremos otro valor de entrada y responderemos a él. Pero para realizar un seguimiento de muchos usuarios y piezas de información, necesitaremos usar listas y diccionarios con nuestros bucles while .

Un bucle for es eficaz para recorrer una lista, pero no debería modificar una lista dentro de un bucle for porque Python tendrá problemas para realizar un seguimiento de los elementos de la lista. Para modificar una lista a medida que trabaja en ella, utilice un bucle while .

El uso de bucles while con listas y diccionarios le permite recopilar, almacenar y organizar una gran cantidad de entradas para examinarlas e informarlas más adelante.

Mover elementos de una lista a otra

Considere una lista de usuarios recién registrados pero no verificados de un sitio web. Después de verificar a estos usuarios, ¿cómo podemos moverlos a una lista separada de usuarios confirmados? Una forma sería utilizar un bucle while para extraer usuarios de la lista de usuarios no confirmados a medida que los verificamos y luego agregarlos a una lista separada de usuarios confirmados. Así es como podría verse ese código:

```
confirmado # Comience con los usuarios que necesitan ser verificados,
_usuario.py # y una lista vacía para contener usuarios confirmados.
1 usuarios_no confirmados = ['alice', 'brian', 'candace']
  usuarios_confirmados = []

  # Verificar cada usuario hasta que no queden más usuarios sin confirmar.
  # Mueva cada usuario verificado a la lista de usuarios confirmados.

2 mientras usuarios_no confirmados:
3 usuario_actual = usuarios_no confirmados.pop()

    print(f"Verificando usuario: {current_user.title()}")
4 usuarios_confirmados.append(usuario_actual)

# Mostrar todos los usuarios confirmados.
print("\nLos siguientes usuarios han sido confirmados:")
para usuario_confirmado en usuarios_confirmados:
    imprimir (usuario_confirmado.título())
```

Comenzamos con una lista de usuarios no confirmados 1 (Alice, Brian y Candace) y una lista vacía para contener usuarios confirmados. El bucle while se ejecuta siempre que la lista usuarios_no confirmados no esté vacía 2. Dentro de este bucle, el método pop() elimina los usuarios no verificados uno a la vez desde el final de usuarios_no confirmados 3. Debido a que Candace es la última en la lista de usuarios_no confirmados , su nombre será el primero en ser eliminado, asignado al usuario_actual y agregado a los usuarios_confirmados . lista 4. El siguiente es Brian, luego Alice.

Simulamos la confirmación de cada usuario imprimiendo un mensaje de verificación y luego agregándolos a la lista de usuarios confirmados. A medida que la lista de usuarios no confirmados se reduce, la lista de usuarios confirmados crece. Cuando la lista de

Los usuarios no confirmados están vacíos, el ciclo se detiene y se imprime la lista de usuarios confirmados:

```
Verificando usuario: Candace
Verificando usuario: Brian
Usuario verificando: Alicia
```

```
Se han confirmado los siguientes usuarios:
Candace
brian
Alicia
```

Eliminar todas las instancias de valores específicos de una lista

En el Capítulo 3, usamos `remove()` para eliminar un valor específico de una lista. La función `remove()` funcionó porque el valor que nos interesaba apareció solo una vez en la lista. Pero, ¿qué sucede si desea eliminar todas las instancias de un valor de una lista?

Supongamos que tiene una lista de mascotas con el valor "gato" repetido varias veces. Para eliminar todas las instancias de ese valor, puede ejecutar un bucle `while` hasta que 'cat' ya no esté en la lista, como se muestra aquí:

```
mascotas.py mascotas = ['perro', 'gato', 'perro', 'pez dorado', 'gato', 'conejo', 'gato']
imprimir (mascotas)

mientras que 'gato' en mascotas:
    mascotas.remove('gato')

imprimir (mascotas)
```

Comenzamos con una lista que contiene múltiples instancias de 'gato'. Después de imprimir la lista, Python ingresa al ciclo `while` porque encuentra el valor 'gato' en la lista al menos una vez. Una vez dentro del bucle, Python elimina la primera instancia de 'cat', regresa a la línea `while` y luego vuelve a entrar en el bucle cuando descubre que 'cat' todavía está en la lista. Elimina cada instancia de 'cat' hasta que el valor ya no está en la lista, momento en el que Python sale del bucle e imprime la lista nuevamente:

```
['perro', 'gato', 'perro', 'pez dorado', 'gato', 'conejo', 'gato']
['perro', 'perro', 'pez dorado', 'conejo']
```

Llenar un diccionario con entradas del usuario

Puede solicitar toda la información que necesite en cada paso durante un tiempo bucle. Hagamos un programa de sondeo en el que cada paso por el bucle solicite el nombre y la respuesta del participante. Almacenaremos los datos que recopilamos en un diccionario, porque queremos conectar cada respuesta con un usuario en particular:

```
respuestas mountain_poll.py = {}
# Establecer una bandera para indicar que el sondeo está activo.
polling_active = Verdadero
```

```

while polling_active: #
    Solicita el nombre de la persona y su respuesta.
1 nombre = entrada("\n¿Cuál es tu nombre? ")
    respuesta = entrada("¿Qué montaña te gustaría escalar algún día? ")

    # Almacenar la respuesta en el diccionario. 2
respuestas[nombre] = respuesta

    # Descubra si alguien más va a realizar la encuesta. 3 repetir
= input("¿Le gustaría dejar que otra persona responda? (sí/no) ") if repetir == 'no': polling_active
    = False

# El sondeo está completo. Muestra los
resultados. print("\n--- Resultados de
la encuesta ---") 4 para el nombre, respuesta en
respuestas.items(): print(f"{nombre} quisiera subir a {respuesta}.")

```

El programa primero define un diccionario vacío (respuestas) y establece una bandera (polling_active) para indicar que el sondeo está activo. Mientras polling_active sea True, Python ejecutará el código en el bucle while .

Dentro del bucle, se le solicita al usuario que ingrese su nombre y una montaña que le gustaría escalar 1. Esa información se almacena en el diccionario de respuestas 2, y se le pregunta al usuario si desea mantener o no la encuesta. 3. Si ingresan sí, el programa ingresa nuevamente al ciclo while . Si ingresan no, el indicador polling_active se establece en False, el ciclo while deja de ejecutarse y el bloque de código final 4 muestra los resultados de la encuesta.

Si ejecuta este programa e ingresa respuestas de muestra, debería ver un resultado como este:

```

¿Cómo te llamas? Eric
¿Qué montaña te gustaría escalar algún día? Denali ¿ Te gustaría
dejar que otra persona responda? (sí/no) sí

¿Cómo te llamas? Lynn
¿Qué montaña te gustaría escalar algún día? Pulgar del Diablo ¿Te
gustaría dejar que otra persona responda? (sí/no) no

--- Resultados de la encuesta
--- A Eric le gustaría escalar Denali.
A Lynn le gustaría escalar el Pulgar del Diablo.

```

INTÉNTALO TÚ MISMO

7-8. Deli: haga una lista llamada `sandwich_orders` y rellénela con los nombres de varios sándwiches. Luego haga una lista vacía llamada `sándwiches_terminados`. Recorra la lista de pedidos de sándwiches e imprima un mensaje para cada pedido, como "Hice tu sándwich de atún". A medida que se prepara cada sándwich, muévelo a la lista de sándwiches terminados. Una vez que se hayan preparado todos los sándwiches, imprima un mensaje enumerando cada sándwich que se preparó.

7-9. Sin pastrami: utilizando la lista `sándwich_orders` del ejercicio 7-8, asegúrese de que el sándwich 'pastrami' aparezca en la lista al menos tres veces. Agregue código cerca del comienzo de su programa para imprimir un mensaje que indique que la tienda de delicatessen se ha quedado sin pastrami y luego use un bucle `while` para eliminar todas las apariciones de 'pastrami' de `sandwich_orders`. Asegúrese de que ningún sándwich de pastrami termine en `sándwiches_terminados`.

7-10. Vacaciones de ensueño: escriba un programa que encueste a los usuarios sobre las vacaciones de sus sueños. Escriba un mensaje similar a Si pudieras visitar un lugar en el mundo, ¿adónde irías? Incluya un bloque de código que imprima los resultados de la encuesta.

Resumen

En este capítulo, aprendió a usar `input()` para permitir a los usuarios proporcionar su propia información en sus programas. Aprendió a trabajar con entradas de texto y numéricas y a utilizar bucles `while` para que sus programas se ejecuten durante el tiempo que sus usuarios quieran. Viste varias formas de controlar el flujo de un bucle `while` estableciendo un indicador activo, usando la instrucción `break` y usando la instrucción `continue`. Aprendiste a usar un bucle `while` para mover elementos de una lista a otra y a eliminar todas las instancias de un valor de una lista. También aprendió cómo se pueden usar los bucles `while` con diccionarios.

En el Capítulo 8 aprenderá sobre las funciones. Las funciones te permiten romper Divida sus programas en pequeñas partes, cada una de las cuales realiza un trabajo específico. Puede llamar a una función tantas veces como desee y puede almacenar sus funciones en archivos separados. Al utilizar funciones, podrá escribir código más eficiente que sea más fácil de solucionar y mantener y que pueda reutilizarse en muchos programas diferentes.

8

FUNCIONES



En este capítulo aprenderá a escribir funciones, que son bloques de código con nombre diseñados para realizar un trabajo específico.

Cuando desea realizar una tarea particular que ha definido en una función, llama a la función responsable de ello. Si necesita realizar esa tarea varias veces a lo largo de su programa, no necesita escribir todo el código para la misma tarea una y otra vez; simplemente llamas a la función dedicada a manejar esa tarea, y la llamada le dice a Python que ejecute el código dentro de la función. Descubrirá que el uso de funciones hace que sus programas sean más fáciles de escribir, leer, probar

En este capítulo también aprenderá una variedad de formas de pasar información a funciones. Aprenderá a escribir ciertas funciones cuyo trabajo principal es mostrar información y otras funciones diseñadas para procesar datos y devolver un valor o conjunto de valores. Finalmente, aprenderá a almacenar funciones en archivos separados llamados módulos para ayudar a organizar los archivos de su programa principal.

Definiendo una función

Aquí hay una función simple llamada `greet_user()` que imprime un saludo:

```
saludador.py def saludar_usuario():
    """Muestra un saludo sencillo."""
    imprimir ("¡Hola!")

saludar_usuario()
```

Este ejemplo muestra la estructura más simple de una función. La primera línea usa la palabra clave `def` para informar a Python que estás definiendo una función. Esta es la definición de la función, que le dice a Python el nombre de la función y, si corresponde, qué tipo de información necesita la función para realizar su trabajo. Los paréntesis contienen esa información. En este caso, el nombre de la función es `greet_user()` y no necesita información para realizar su trabajo, por lo que sus paréntesis están vacíos. (Aun así, los paréntesis son obligatorios). Finalmente, la definición termina en dos puntos.

Cualquier línea sangrada que siga a `def greet_user()`: constituye el cuerpo de la función. El texto de la segunda línea es un comentario llamado cadena de documentación, que describe lo que hace la función. Cuando Python genera documentación para las funciones de sus programas, busca una cadena inmediatamente después de la definición de la función. Estas cadenas suelen estar entre comillas triples, lo que le permite escribir varias líneas.

La línea `print("Hello!")` es la única línea de código real en el cuerpo de esta función, por lo que `greet_user()` tiene solo un trabajo: `print("¡Hola!")`.

Cuando quieras utilizar esta función, debes llamarla. Una llamada de función le dice a Python que ejecute el código en la función. Para llamar a una función, escribe el nombre de la función, seguido de cualquier información necesaria entre paréntesis. Como aquí no se necesita información, llamar a nuestra función es tan simple como ingresar `greet_user()`. Como era de esperar, imprime ¡Hola!:

```
¡Hola!
```

Pasar información a una función

Si modifica ligeramente la función `greet_user()`, puede saludar al usuario por su nombre. Para que la función haga esto, ingrese el nombre de usuario entre paréntesis de la definición de la función en `def greet_user()`. Al agregar nombre de usuario aquí, permite que la función acepte cualquier valor de nombre de usuario que especifique. La función ahora espera que proporciones un valor para el nombre de usuario cada vez que la llames. Cuando llamas a `greet_user()`, puedes pasarle un nombre, como 'jesse', dentro del paréntesis:

```
def saludar_usuario(nombre de usuario):
    """Muestra un saludo sencillo."""
    print(f"¡Hola, {nombre de usuario.titulo()}!")

saludar_usuario('jesse')
```

Al ingresar a `greet_user('jesse')`, se llama a `greet_user()` y se le da a la función la información que necesita para ejecutar la llamada `print()`. La función acepta el nombre que le pasó y muestra el saludo para ese nombre:

```
¡Hola Jessie!
```

Del mismo modo, al ingresar `greet_user('sarah')` se llama a `greet_user()`, se le pasa 'sarah', y imprime ¡Hola, Sarah! Puede llamar a `greet_user()` tantas veces como desee y pasarle el nombre que desee para producir un resultado predecible en todo momento.

Argumentos y parámetros

En la función `greet_user()` anterior, definimos `greet_user()` para requerir un valor para la variable nombre de usuario. Una vez que llamamos a la función y le dimos la información (el nombre de una persona), imprimió el saludo correcto.

La variable nombre de usuario en la definición de `greet_user()` es un ejemplo de parámetro, una información que la función necesita para hacer su trabajo. El valor 'jesse' en `greet_user('jesse')` es un ejemplo de argumento. Un argumento es una pieza de información que se pasa de una llamada de función a una función.

Cuando llamamos a la función, colocamos entre paréntesis el valor con el que queremos que funcione la función. En este caso, el argumento 'jesse' se pasó a la función `greet_user()` y el valor se asignó al parámetro nombre de usuario.

NOTA A veces la gente habla de argumentos y parámetros indistintamente. No se sorprenda si ve las variables en una definición de función denominadas argumentos o las variables en una llamada de función denominadas parámetros.

INTÉNTALO TÚ MISMO

8-1. Mensaje: escriba una función llamada `display_message()` que imprima una frase que cuente a todos lo que está aprendiendo en este capítulo. Llame a la función y asegúrese de que el mensaje se muestre correctamente.

8-2. Libro favorito: escriba una función llamada `libro_favorito()` que acepte un parámetro, título. La función debería imprimir un mensaje, como Uno de mis libros favoritos es Alicia en el país de las maravillas. Llame a la función, asegurándose de incluir el título de un libro como argumento en la llamada a la función.

Pasar argumentos

Debido a que la definición de una función puede tener múltiples parámetros, una llamada a una función puede necesitar múltiples argumentos. Puede pasar argumentos a sus funciones de varias maneras. Puede utilizar argumentos posicionales, que deben estar en el mismo orden en que se escribieron los parámetros; argumentos de palabras clave, donde cada argumento consta de un nombre de variable y un valor; y listas y diccionarios de valores. Veamos cada uno de estos por turno.

Argumentos posicionales

Cuando llama a una función, Python debe hacer coincidir cada argumento en la llamada a la función con un parámetro en la definición de la función. La forma más sencilla de hacerlo se basa en el orden de los argumentos proporcionados. Los valores emparejados de esta manera se denominan argumentos posicionales.

Para ver cómo funciona esto, considere una función que muestra información sobre mascotas. La función nos dice qué tipo de animal es cada mascota y el nombre de la mascota, como se muestra aquí:

```

mascotas.py 1 def describe_pet(tipo_animal, nombre_mascota):
               """Mostrar información sobre una mascota."""
               print(f"\nTengo un {tipo_animal}.")
               print(f"El nombre de mi {animal_type} es {pet_name.title()}")

2 describe_pet('hámster', 'harry')

```

La definición muestra que esta función necesita un tipo de animal y el nombre del animal 1. Cuando llamamos a `describe_pet()`, necesitamos proporcionar un tipo de animal y un nombre, en ese orden. Por ejemplo, en la llamada a la función, el argumento 'hámster' se asigna al parámetro `tipo_animal` y el argumento 'harry' se asigna al parámetro `nombre_mascota` 2. En el cuerpo de la función, estos dos parámetros se utilizan para mostrar información sobre el mascota que se describe.

El resultado describe un hámster llamado Harry:

```

Tengo un hámster.
El nombre de mi hámster es Harry.

```

Llamadas a funciones múltiples

Puede llamar a una función tantas veces como sea necesario. Describir una segunda mascota diferente requiere sólo una llamada más a `describe_pet()`:

```

def describir_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {tipo_animal}.")
    print(f"El nombre de mi {animal_type} es {pet_name.title()}")

describe_pet('hámster', 'harry')
describe_pet('perro', 'willie')

```

En esta segunda llamada a función, pasamos `describe_pet()` los argumentos 'perro' y 'willie'. Al igual que con el conjunto de argumentos anterior que utilizamos, Python relaciona 'perro' con el parámetro `tipo_animal` y 'willie' con el parámetro `nombre_mascota`. Como antes, la función hace su trabajo, pero esta vez imprime valores para un perro llamado Willie. Ahora tenemos un hámster llamado Harry y un perro llamado Willie:

```

Tengo un hámster.
El nombre de mi hámster es Harry.

```

Tengo un perro.
El nombre de mi perro es Willie.

Llamar a una función varias veces es una forma muy eficiente de trabajar. El código que describe una mascota se escribe una vez en la función. Luego, cada vez que desee describir una nueva mascota, llame a la función con la información de la nueva mascota. Incluso si el código para describir una mascota se expandiera a 10 líneas, aún podría describir una nueva mascota en una sola línea llamando a la función nuevamente.

El orden importa en los argumentos posicionales

Puede obtener resultados inesperados si mezcla el orden de los argumentos en una llamada de función cuando usa argumentos posicionales:

```
def describir_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {tipo_animal}.")
    print(f"El nombre de mi {animal_type} es {pet_name.title()}")
```

```
describe_pet('harry', 'hámster')
```

En esta llamada a función, enumeramos primero el nombre y después el tipo de animal. Debido a que esta vez el argumento 'harry' aparece primero, ese valor se asigna al parámetro tipo_animal. Asimismo, 'hámster' se asigna a pet_name. Ahora tenemos un "harry" llamado "Hamster":

Tengo un harry.
El nombre de mi Harry es Hamster.

Si obtiene resultados divertidos como este, verifique que el orden de los argumentos en su llamada a función coincida con el orden de los parámetros en la definición de la función.

Argumentos de palabras clave

Un argumento de palabra clave es un par nombre-valor que se pasa a una función. Usted asocia directamente el nombre y el valor dentro del argumento, de modo que cuando pasa el argumento a la función, no hay confusión (no terminará con un Harry llamado Hamster). Los argumentos de palabras clave le liberan de tener que preocuparse por ordenar correctamente sus argumentos en la llamada a la función y aclaran la función de cada valor en la llamada a la función.

Reescribamos dogs.py usando argumentos de palabras clave para llamar a describe_pet():

```
def describir_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {tipo_animal}.")
    print(f"El nombre de mi {animal_type} es {pet_name.title()}")
```

```
describe_pet(tipo_animal='hámster', nombre_mascota='harry')
```

La función `describe_pet()` no ha cambiado. Pero cuando llamamos a la función, le decimos explícitamente a Python con qué parámetro debe coincidir cada argumento. Cuando Python lee la llamada a la función, sabe asignar el argumento 'hámster' al parámetro `tipo_animal` y el argumento 'harry' al `nombre_mascota`. El resultado muestra correctamente que tenemos un hámster llamado Harry.

El orden de los argumentos de las palabras clave no importa porque Python sabe dónde debe ir cada valor. Las siguientes dos llamadas a funciones son equivalentes:

```
describe_pet(tipo_animal='hámster', nombre_mascota='harry')
describe_pet(nombre_mascota='harry', tipo_animal='hámster')
```

NOTA Cuando utilice argumentos de palabras clave, asegúrese de utilizar los nombres exactos de los parámetros en la definición de la función.

Valores predeterminados

Al escribir una función, puede definir un valor predeterminado para cada parámetro.

Si se proporciona un argumento para un parámetro en la llamada a la función, Python usa el valor del argumento. De lo contrario, utiliza el valor predeterminado del parámetro. Entonces, cuando defines un valor predeterminado para un parámetro, puedes excluir el argumento correspondiente que normalmente escribirías en la llamada a la función. El uso de valores predeterminados puede simplificar las llamadas a funciones y aclarar las formas en que se utilizan normalmente las funciones.

Por ejemplo, si observa que la mayoría de las llamadas a `describe_pet()` se utilizan para describir perros, puede establecer el valor predeterminado de `animal_type` en 'perro'. Ahora cualquiera que llame a `describe_pet()` para un perro puede omitir esa información:

```
def describe_pet(nombre_mascota, tipo_animal='perro'):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {tipo_animal}.")
    print(f"El nombre de mi {animal_type} es {pet_name.title()}")

describe_pet(nombre_mascota='willie')
```

Cambiamos la definición de `describe_pet()` para incluir un valor predeterminado, 'perro', para `animal_type`. Ahora, cuando se llama a la función sin `animal_type` especificado, Python sabe utilizar el valor 'perro' para este parámetro:

```
Tengo un perro.
El nombre de mi perro es Willie.
```

Tenga en cuenta que se tuvo que cambiar el orden de los parámetros en la definición de la función. Debido a que el valor predeterminado hace innecesario especificar un tipo de animal como argumento, el único argumento que queda en la llamada a la función es el nombre de la mascota. Python todavía interpreta esto como un argumento posicional, por lo que si la función se llama solo con el nombre de una mascota, ese argumento coincidirá con el primer parámetro enumerado en la definición de la función. Esta es la razón por la que el primer parámetro debe ser `pet_name`.

La forma más sencilla de utilizar esta función ahora es proporcionar solo la imagen de un perro.
nombre en la llamada a la función:

```
describir_pet('willie')
```

Esta llamada a función tendría el mismo resultado que el ejemplo anterior.
El único argumento proporcionado es 'willie', por lo que coincide con el primer parámetro de la definición, pet_name. Como no se proporciona ningún argumento para animal_type, Python usa el valor predeterminado "perro".

Para describir un animal que no sea un perro, puedes usar una llamada de función como esta:

```
describe_pet(nombre_mascota='harry', tipo_animal='hámster')
```

Debido a que se proporciona un argumento explícito para tipo_animal, Python ignora el valor predeterminado del parámetro.

NOTA Cuando utiliza valores predeterminados, cualquier parámetro con un valor predeterminado debe aparecer después de todos los parámetros que no tienen valores predeterminados. Esto permite a Python continuar interpretando argumentos posicionales correctamente.

Llamadas a funciones equivalentes

Debido a que los argumentos posicionales, los argumentos de palabras clave y los valores predeterminados se pueden usar juntos, a menudo tendrás varias formas equivalentes de llamar a una función. Considere la siguiente definición de describe_pet() con un valor predeterminado proporcionado:

```
def describe_pet(nombre_mascota, tipo_animal='perro'):
```

Con esta definición, siempre es necesario proporcionar un argumento para pet_name, y este valor se puede proporcionar utilizando el formato posicional o de palabra clave. Si el animal que se describe no es un perro, un argumento para tipo_animal debe incluirse en la llamada y este argumento también se puede especificar utilizando el formato posicional o de palabra clave.

Todas las siguientes llamadas funcionarían para esta función:

```
# Un perro llamado Willie.
describir_pet('willie')
describe_pet(nombre_mascota='willie')

# Un hámster llamado Harry.
describe_pet('harry', 'hámster')
describe_pet(nombre_mascota='harry', tipo_animal='hámster')
describe_pet(tipo_animal='hámster', nombre_mascota='harry')
```

Cada una de estas llamadas a funciones tendría el mismo resultado que los ejemplos anteriores.

Realmente no importa qué estilo de llamada uses. Siempre que sus llamadas a funciones produzcan el resultado que desea, simplemente use el estilo que le resulte más fácil de entender.

Evitar errores de argumentación

Cuando empiece a utilizar funciones, no se sorprenda si encuentra errores sobre argumentos no coincidentes. Los argumentos no coincidentes ocurren cuando proporciona menos o más argumentos de los que una función necesita para realizar su trabajo. Por ejemplo, esto es lo que sucede si intentamos llamar a `describe_pet()` sin argumentos:

```
def describir_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {tipo_animal}.")
    print(f"El nombre de mi {animal_type} es {pet_name.title()}")
```

```
describir_mascota()
```

Python reconoce que falta cierta información en la función llame, y el rastreo nos dice que:

```
Rastreo (llamadas recientes más última):
1 Archivo "pets.py", línea 6, en <módulo>
2 describir_pet()
  ^^^^^^^^^^^^^^^^^
3 TypeError: a describe_pet() le faltan 2 argumentos posicionales requeridos: 'animal_type'
  y 'pet_name'
```

El rastreo primero nos dice la ubicación del problema 1, lo que nos permite mirar hacia atrás y ver que algo salió mal en nuestra llamada a función. A continuación, se escribe la llamada a la función infractora para que la veamos 2. Por último, el rastreo nos dice que a la llamada le faltan dos argumentos e informa los nombres de los argumentos que faltan 3. Si esta función estuviera en un archivo separado, podríamos Probablemente reescriba la llamada correctamente sin tener que abrir ese archivo y leer el código de función.

Python es útil porque lee el código de la función por nosotros y nos dice los nombres de los argumentos que debemos proporcionar. Esta es otra motivación para dar nombres descriptivos a las variables y funciones. Si lo hace, los mensajes de error de Python serán más útiles para usted y para cualquier otra persona que pueda usar su código.

Si proporciona demasiados argumentos, debería obtener un rastreo similar que pueda ayudarle a hacer coincidir correctamente su llamada de función con la definición de función.

INTÉNTALO TÚ MISMO

8-3. Camiseta: Escribe una función llamada `make_shirt()` que acepte una talla y el texto de un mensaje que debe imprimirse en la camiseta. La función debe imprimir una oración que resuma el tamaño de la camiseta y el mensaje impreso en ella.

Llame a la función una vez usando argumentos posicionales para hacer una camisa. Llame a la función por segunda vez utilizando argumentos de palabras clave.

8-4. Camisas grandes: modifique la función `make_shirt()` para que las camisas sean grandes de forma predeterminada con un mensaje que diga Me encanta Python. Haz una camiseta grande y una camiseta mediana con el mensaje predeterminado, y una camiseta de cualquier talla con un mensaje diferente.

8-5. Ciudades: Escribe una función llamada `describe_city()` que acepte el nombre de una ciudad y su país. La función debería imprimir una oración simple, como Reykjavik está en Islandia. Asigne al parámetro del país un valor predeterminado.

Llame a su función para tres ciudades diferentes, al menos una de las cuales no esté en el país predeterminado.

Valores de retorno

Una función no siempre tiene que mostrar su salida directamente. En cambio, puede procesar algunos datos y luego devolver un valor o un conjunto de valores. El valor que devuelve la función se denomina valor de retorno. La declaración de retorno toma un valor desde dentro de una función y lo envía de regreso a la línea que llamó a la función. Los valores de retorno le permiten mover gran parte del trabajo duro de su programa a funciones, lo que puede simplificar el cuerpo de su programa.

Devolver un valor simple

Veamos una función que toma un nombre y apellido y devuelve un nombre completo perfectamente formateado:

```

formateado
_nombre.py def get_formatted_name(primer_nombre, apellido):
            """Devuelve un nombre completo, cuidadosamente formateado."""
            1 nombre_completo = f'{primer_nombre} {apellido}'
            2 devolver nombre_completo.título()

            3 músico = get_formatted_name('jimi', 'hendrix')
            imprimir (músico)

```

La definición de `get_formatted_name()` toma como parámetros un nombre y apellido. La función combina estos dos nombres, agrega un espacio entre ellos y asigna el resultado a `nombre_completo` 1. El valor de `nombre_completo` se convierte a título en mayúsculas y minúsculas y luego se devuelve a la línea de llamada 2.

Cuando llama a una función que devuelve un valor, debe proporcionar una variable a la que se pueda asignar el valor de retorno. En este caso, el valor devuelto se asigna a la variable `músico` 3. El resultado muestra un nombre perfectamente formateado compuesto por las partes del nombre de una persona:

```
Jimi Hendrix
```

Puede parecer mucho trabajo conseguir un nombre bien formateado cuando podríamos haber escrito simplemente:

```
imprimir("Jimi Hendrix")
```

Sin embargo, cuando considera trabajar con un programa grande que necesita almacenar muchos nombres y apellidos por separado, funciones como `get_formatted_name()` se vuelve muy útil. Almacena los nombres y apellidos por separado y luego llama a esta función cuando desee mostrar un nombre completo.

Hacer que un argumento sea opcional

A veces tiene sentido hacer que un argumento sea opcional, de modo que las personas que usan la función puedan optar por proporcionar información adicional solo si así lo desean. Puede utilizar valores predeterminados para hacer que un argumento sea opcional.

Por ejemplo, digamos que queremos expandir `get_formatted_name()` para manejar también los segundos nombres. Un primer intento de incluir segundos nombres podría verse así:

```
def get_formatted_name(primer_nombre, segundo_nombre, apellido):
    """Devuelve un nombre completo, cuidadosamente formateado."""
    nombre_completo = f'{primer_nombre} {segundo_nombre} {apellido}'
    devolver nombre_completo.titulo()

músico = get_formatted_name('john', 'lee', 'prostituta')
imprimir(músico)
```

Esta función funciona cuando se le da un nombre, segundo nombre y apellido. La función toma las tres partes de un nombre y luego construye una cadena a partir de ellas. La función agrega espacios cuando corresponda y convierte el nombre completo en mayúsculas y minúsculas:

John Lee Hooker

Pero los segundos nombres no siempre son necesarios, y esta función tal como está escrita no funcionaría si intentara llamarla solo con un nombre y un apellido. Para que el segundo nombre sea opcional, podemos darle al argumento `middle_name` un valor predeterminado vacío e ignorar el argumento a menos que el usuario proporcione un valor. Para que `get_formatted_name()` funcione sin un segundo nombre, configuramos el valor predeterminado de `middle_name` en una cadena vacía y lo movemos al final de la lista de parámetros:

```
def get_formatted_name(primer_nombre, apellido, segundo_nombre=""):
    """Devuelve un nombre completo, cuidadosamente formateado."""
    1 si segundo nombre:
        nombre_completo = f'{primer_nombre} {segundo_nombre} {apellido}'
    2 más:
        nombre_completo = f'{primer_nombre} {apellido}'
    devolver nombre_completo.titulo()

músico = get_formatted_name('jimi', 'hendrix')
imprimir(músico)

3 músico = get_formatted_name('john', 'hooker', 'lee')
imprimir(músico)
```

En este ejemplo, el nombre se construye a partir de tres partes posibles. Como siempre hay un nombre y un apellido, estos parámetros aparecen primero en la definición de la función. El segundo nombre es opcional, por lo que aparece al final de la definición y su valor predeterminado es una cadena vacía.

En el cuerpo de la función, verificamos si se ha incluido un segundo nombre. proporcionó. Python interpreta las cadenas no vacías como True, por lo que la prueba condicional `if middle_name` se evalúa como True si un argumento de segundo nombre está en la llamada a la función 1. Si se proporciona un segundo nombre, el nombre, el segundo nombre y el apellido se combinan para formar un nombre completo. Luego, este nombre se cambia a mayúsculas y minúsculas y se devuelve a la línea de llamada de función, donde se asigna a la variable `músico` y se imprime. Si no se proporciona ningún segundo nombre, la cadena vacía no pasa la prueba `if` y el bloque `else` ejecuta 2. El nombre completo se crea solo con un nombre y apellido, y el nombre formateado se devuelve a la línea de llamada donde se asigna al `músico` y impreso.

Llamar a esta función con un nombre y apellido es sencillo. Si Estamos usando un segundo nombre, sin embargo, tenemos que asegurarnos de que el segundo nombre sea el último argumento pasado para que Python haga coincidir los argumentos posicionales correctamente 3.

Esta versión modificada de nuestra función funciona para personas con solo una primera y apellido, y funciona para personas que también tienen un segundo nombre:

Jimi Hendrix
John Lee Hooker

Los valores opcionales permiten que las funciones manejen una amplia gama de casos de uso y, al mismo tiempo, permiten que las llamadas a funciones sigan siendo lo más simples posible.

Devolver un diccionario

Una función puede devolver cualquier tipo de valor que necesite, incluidas estructuras de datos más complicadas como listas y diccionarios. Por ejemplo, la siguiente función toma partes de un nombre y devuelve un diccionario que representa a una persona:

```
persona.py def build_person(nombre, apellido):
    """Devuelve un diccionario de información sobre una persona."""
    1 persona = {'primero': primer_nombre, 'apellido': apellido}
    2 persona de regreso

    músico = build_person('jimi', 'hendrix')
    3 impresión (músico)
```

La función `build_person()` toma un nombre y un apellido y coloca estos valores en un diccionario 1. El valor de `first_name` se almacena con la clave `'first'` y el valor de `last_name` se almacena con la clave `'last'`. Luego, se devuelve el diccionario completo que representa a la persona 2. El valor devuelto se imprime 3 con las dos piezas originales de información textual ahora almacenadas en un diccionario:

```
{'primero': 'jimi', 'último': 'hendrix'}
```

Esta función toma información textual simple y la coloca en una estructura de datos más significativa que le permite trabajar con la información más allá de simplemente imprimirla. Las cadenas 'jimi' y 'hendrix' ahora están etiquetadas como nombre y apellido. Puede ampliar fácilmente esta función para aceptar valores opcionales como un segundo nombre, una edad, una ocupación o cualquier otra información que desee almacenar sobre una persona. Por ejemplo, el siguiente cambio también le permite almacenar la edad de una persona:

```
def build_person(nombre, apellido, edad=Ninguno):
    """Devuelve un diccionario de información sobre una persona."""
    persona = {'primero': primer_nombre, 'apellido': apellido}
    si edad:
        persona['edad'] = edad
    persona que regresa

músico = build_person('jimi', 'hendrix', edad=27)
imprimir (músico)
```

Agregamos un nuevo parámetro opcional edad a la definición de la función y asignamos al parámetro el valor especial Ninguno, que se usa cuando una variable no tiene un valor específico asignado. Puedes pensar en Ninguno como un valor de marcador de posición. En las pruebas condicionales, Ninguno se evalúa como Falso. Si la llamada a la función incluye un valor para la edad, ese valor se almacena en el diccionario. Esta función siempre almacena el nombre de una persona, pero también se puede modificar para almacenar cualquier otra información que desee sobre una persona.

Usando una función con un bucle while

Puedes usar funciones con todas las estructuras de Python que has aprendido hasta ahora. Por ejemplo, usemos la función `get_formatted_name()` con un tiempo bucle para saludar a los usuarios de manera más formal. Aquí hay un primer intento de saludar a las personas usando su nombre y apellido:

```
greeter.py def get_formatted_name(primer_nombre, apellido):
    """Devuelve un nombre completo, cuidadosamente formateado."""
    nombre_completo = f"{primer_nombre} {apellido}"
    devolver nombre_completo.titulo()

# ¡Este es un bucle infinito!
mientras que Verdadero:
    1 print("\nPor favor dime tu nombre:")
        f_name = input("Nombre: ")
        l_name = input("Apellido: ")

    nombre_formateado = obtener_nombre_formateado(f_name, l_name)
    print(f"\nHola, {nombre_formateado}!")
```

Para este ejemplo, utilizamos una versión simple de `get_formatted_name()` que no incluye segundos nombres. El bucle `while` le pide al usuario que ingrese su nombre y le solicitamos su nombre y apellido por separado 1.

Pero hay un problema con este bucle `while` : no hemos definido una condición para salir. ¿Dónde pones una condición para dejar de fumar cuando pides una serie de

entradas? Queremos que el usuario pueda salir lo más fácilmente posible, por lo que cada mensaje debe ofrecer una forma de salir. La declaración `break` ofrece una forma sencilla de salir del bucle en cualquiera de los dos mensajes:

```
def get_formatted_name(first_name, last_name): """Devuelve
    un nombre completo, cuidadosamente formateado."""
    full_name = f'{first_name} {apellido}'
    return full_name.title()
```

mientras que Verdadero:

```
print("\nPor favor dígame su nombre:")
print("(ingrese 'q' en cualquier momento para salir)")
```

```
f_name = input("Nombre: ")
if f_name == 'q':
    descanso
```

```
l_name = input("Apellido: ")
if l_name == 'q':
    romper
```

```
nombre_formateado = get_nombre_formateado(nombre_f,
nombre_l)
print(f"\nHola, {nombre_formateado}!")
```

Agregamos un mensaje que informa al usuario cómo salir y luego salimos del bucle si el usuario ingresa el valor de salida en cualquiera de las indicaciones. Ahora el programa continuará saludando a las personas hasta que alguien ingrese `q` para cualquiera de los nombres:

Por favor dime tu nombre:

(ingresa 'q' en cualquier momento para salir)

Nombre: Eric

Apellido: mates

¡Hola Eric Matthes!

Por favor dime tu nombre:

(ingresa 'q' en cualquier momento para salir)

Nombre: q

INTÉNTALO TÚ MISMO

8-6. Nombres de ciudades: escriba una función llamada `city_country()` que tome el nombre de una ciudad y su país. La función debería devolver una cadena con el formato siguiente:

"Santiago, Chile"

Llame a su función con al menos tres pares de ciudad-país e imprima los valores que se devuelven.

(continuado)

8-7. Álbum: escriba una función llamada `make_album()` que cree un diccionario que describa un álbum de música. La función debe incluir el nombre de un artista y el título de un álbum, y debe devolver un diccionario que contenga estos dos datos. Utilice la función para crear tres diccionarios que representen diferentes álbumes. Imprima cada valor de retorno para mostrar que los diccionarios almacenan la información del álbum correctamente.

Utilice Ninguno para agregar un parámetro opcional a `make_album()` que le permita almacenar la cantidad de canciones en un álbum. Si la línea de llamada incluye un valor para la cantidad de canciones, agregue ese valor al diccionario del álbum. Realice al menos una nueva llamada de función que incluya la cantidad de canciones de un álbum.

8-8. Álbumes de usuario: comience con su programa del Ejercicio 8-7. escriba un rato bucle que permite a los usuarios ingresar el artista y el título de un álbum. Una vez que tenga esa información, llame a `make_album()` con la entrada del usuario e imprima el diccionario creado. Asegúrese de incluir un valor de salida en el ciclo `while`.

Pasar una lista

A menudo le resultará útil pasar una lista a una función, ya sea una lista de nombres, números u objetos más complejos, como diccionarios. Cuando pasa una lista a una función, la función obtiene acceso directo al contenido de la lista. Usemos funciones para hacer que trabajar con listas sea más eficiente.

Digamos que tenemos una lista de usuarios y queremos imprimir un saludo para cada uno. El siguiente ejemplo envía una lista de nombres a una función llamada `greet_users()`, que saluda a cada persona de la lista individualmente:

```
saludar_usuarios.py def saludar_usuarios(nombres):
    """Imprime un saludo sencillo para cada usuario de la lista."""
    para nombre en nombres:
        msg = f"Hola, {nombre.titulo()}!"
        imprimir (mensaje)

nombres de usuario = ['hannah', 'ty', 'margot']
saludar_usuarios(nombres de usuario)
```

Definimos `greet_users()` para que espere una lista de nombres, que asigna a los nombres de los parámetros. La función recorre la lista que recibe e imprime un saludo para cada usuario. Fuera de la función, definimos una lista de usuarios y luego pasamos la lista de nombres de usuarios a `greet_users()` en la llamada a la función:

```
¡Hola Hannah!
¡Hola Ty!
¡Hola Margot!
```

Este es el resultado que queríamos. Cada usuario ve un saludo personalizado, y puede llamar a la función en cualquier momento que desee para saludar a un conjunto específico de usuarios.

Modificar una lista en una función

Cuando pasa una lista a una función, la función puede modificar la lista. Cualquier cambio realizado en la lista dentro del cuerpo de la función es permanente, lo que le permite trabajar de manera eficiente incluso cuando maneja grandes cantidades de datos.

Considere una empresa que crea modelos impresos en 3D de diseños que envían los usuarios. Los diseños que deben imprimirse se almacenan en una lista y, una vez impresos, se mueven a una lista separada. El siguiente código hace esto sin usar funciones:

```
impresión
_modelos.py # Comience con algunos diseños que deben imprimirse.
unprinted_designs = ['funda de teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []

# Simular la impresión de cada diseño, hasta que no quede ninguno.
# Mueva cada diseño a modelos_completados después de imprimir.
mientras que diseños_no impresos:
    diseño_actual = diseños_sin imprimir.pop()
    print(f"Modelo de impresión: {current_design}")
    modelos_completados.append(diseño_actual)

# Mostrar todos los modelos completados.
print("\nSe han impreso los siguientes modelos:")
para modelo_completado en modelos_completados:
    imprimir (modelo_completado)
```

Este programa comienza con una lista de diseños que deben imprimirse y una lista vacía llamada `modelos_completados` a la que se moverá cada diseño una vez impreso. Mientras los diseños permanezcan en `unprinted_designs`, mientras loop simula la impresión de cada diseño eliminando un diseño del final de la lista, almacenándolo en `current_design` y mostrando un mensaje que indica que el diseño actual se está imprimiendo. Luego agrega el diseño a la lista de modelos completados. Cuando el ciclo termina de ejecutarse, se muestra una lista de los diseños que se han impreso:

```
Modelo de impresión: dodecaedro
Modelo de impresión: colgante de robot
Modelo de impresión: funda de teléfono.
```

```
Se han impreso los siguientes modelos:
dodecaedro
colgante de robot
carcasa de telefono
```

Podemos reorganizar este código escribiendo dos funciones, cada una de las cuales realiza un trabajo específico. La mayor parte del código no cambiará; simplemente lo estamos estructurando con más cuidado. La primera función se encargará de imprimir los diseños y la segunda resumirá las impresiones que se han realizado:

```
1 def print_models(diseños_no impresos, modelos_completados):
```

```
    Simula la impresión de cada diseño, hasta que no quede ninguno.
```

Mueva cada diseño a `modelos_completados` después de imprimir.

```
mientras que diseños_no impresos:
    diseño_actual = diseños_sin imprimir.pop()
    print(f"Modelo de impresión: {current_design}")
    modelos_completados.append(diseño_actual)
```

2 definición `show_completed_models(modelos_completados)`:

```
"""Mostrar todos los modelos que se imprimieron."""
print("\nSe han impreso los siguientes modelos:")
para modelo_completado en modelos_completados:
    imprimir (modelo_completado)
```

```
unprinted_designs = ['funda de teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []
```

```
print_models(diseños_no impresos, modelos_completados)
show_completed_models(modelos_completados)
```

Definimos la función `print_models()` con dos parámetros: una lista de diseños que deben imprimirse y una lista de modelos completados 1. Dadas estas dos listas, la función simula la impresión de cada diseño vaciando la lista de diseños no impresos y llenando el lista de modelos terminados. Luego definimos la función `show_completed_models()` con un parámetro: la lista de modelos completados 2. Dada esta lista, `show_completed_models()` muestra el nombre de cada modelo que se imprimió.

Este programa tiene el mismo resultado que la versión sin funciones, pero el código está mucho más organizado. El código que realiza la mayor parte del trabajo se ha trasladado a dos funciones separadas, lo que hace que la parte principal del programa sea más fácil de entender. Mire el cuerpo del programa y observe con qué facilidad puede seguir lo que está sucediendo:

```
unprinted_designs = ['funda de teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []
```

```
print_models(diseños_no impresos, modelos_completados)
show_completed_models(modelos_completados)
```

Configuramos una lista de diseños no impresos y una lista vacía que contendrá los modelos completos. Luego, como ya hemos definido nuestras dos funciones, todo lo que tenemos que hacer es llamarlas y pasarles los argumentos correctos. Llamamos a `print_models()` y le pasamos las dos listas que necesita; como se esperaba, `print_models()` Simula la impresión de los diseños. Luego llamamos a `show_completed_models()` y le pasamos la lista de modelos completados para que pueda informar los modelos que se han impreso. Los nombres descriptivos de las funciones permiten que otros lean este código y lo comprendan, incluso sin comentarios.

Este programa es más fácil de ampliar y mantener que la versión sin funciones. Si necesitamos imprimir más diseños más adelante, simplemente podemos llamar

`print_models()` nuevamente. Si nos damos cuenta de que es necesario modificar el código de impresión, podemos cambiar el código una vez y nuestros cambios se realizarán en todos los lugares donde se llame a la función. Esta técnica es más eficaz que tener que actualizar el código por separado en varios lugares del programa.

Este ejemplo también demuestra la idea de que cada función debe tener un trabajo específico. La primera función imprime cada diseño y la segunda muestra los modelos completos. Esto es más beneficioso que utilizar una función para realizar ambos trabajos. Si estás escribiendo una función y notas que la función realiza demasiadas tareas diferentes, intenta dividir el código en dos funciones.

Recuerde que siempre puede llamar a una función desde otra función, lo que puede resultar útil al dividir una tarea compleja en una serie de pasos.

Evitar que una función modifique una lista

A veces querrás evitar que una función modifique una lista. Por ejemplo, digamos que comienza con una lista de diseños no impresos y escribe una función para moverlos a una lista de modelos completados, como en el ejemplo anterior. Puede decidir que, aunque haya impreso todos los diseños, desea conservar la lista original de diseños no impresos para sus registros. Pero debido a que sacó todos los nombres de diseños de `unprinted_designs`, la lista ahora está vacía y la lista vacía es la única versión que tiene; el original se ha ido. En este caso, puede solucionar este problema pasando a la función una copia de la lista, no la original. Cualquier cambio que la función realice en la lista afectará solo a la copia, dejando intacta la lista original.

Puede enviar una copia de una lista a una función como esta:

```
nombre_función(nombre_lista[:])
```

La notación de segmento `[:]` hace una copia de la lista para enviarla a la función. Si No queríamos vaciar la lista de diseños no impresos en `Printing_models.py`, podríamos llamar a `print_models()` de esta manera:

```
print_models(diseños_no impresos[:], modelos_completados)
```

La función `print_models()` puede hacer su trabajo porque aún recibe los nombres de todos los diseños no impresos. Pero esta vez utiliza una copia de la lista original de diseños no impresos, no la lista real de `diseños_no impresos`. La lista completa `_models` se completará con los nombres de los modelos impresos como lo hacía antes, pero la lista original de diseños no impresos no se verá afectada por la función.

Aunque puede conservar el contenido de una lista pasando una copia a sus funciones, debe pasar la lista original a las funciones a menos que tenga una razón específica para pasar una copia. Es más eficiente que una función funcione con una lista existente, porque esto evita usar el tiempo y la memoria necesarios para hacer una copia por separado. Esto es especialmente cierto cuando se trabaja con listas grandes.

INTÉNTALO TÚ MISMO

8-9. Mensajes: haga una lista que contenga una serie de mensajes de texto cortos. Pase la lista a una función llamada `show_messages()`, que imprime cada mensaje de texto.

8-10. Envío de mensajes: comience con una copia de su programa del Ejercicio 8-9. Escriba una función llamada `send_messages()` que imprima cada mensaje de texto y mueva cada mensaje a una nueva lista llamada `sent_messages` a medida que se imprime. Después de llamar a la función, imprima ambas listas para asegurarse de que los mensajes se movieron correctamente.

8-11. Mensajes archivados: comience con su trabajo del Ejercicio 8-10. Llame a la función `send_messages()` con una copia de la lista de mensajes. Después de llamar a la función, imprima ambas listas para mostrar que la lista original ha conservado sus mensajes.

Pasar un número arbitrario de argumentos

A veces no sabrá de antemano cuántos argumentos debe aceptar una función. Afortunadamente, Python permite que una función recopile una cantidad arbitraria de argumentos de la declaración de llamada.

Por ejemplo, considere una función que construye una pizza. Debe aceptar una cantidad de ingredientes, pero no se puede saber de antemano cuántos ingredientes querrá una persona. La función en el siguiente ejemplo tiene un parámetro, `*toppings`, pero este parámetro recopila tantos argumentos como proporciona la línea de llamada:

```
pizza.py def make_pizza(*ingredientes):
    """Imprimir la lista de ingredientes que se han solicitado."""
    imprimir (ingredientes)

hacer_pizza('peperoni')
make_pizza('champiñones', 'pimientos verdes', 'queso extra')
```

El asterisco en el nombre del parámetro `*toppings` le dice a Python que cree una tupla llamada `toppings`, que contenga todos los valores que recibe esta función. La llamada `print()` en el cuerpo de la función produce una salida que muestra que Python puede manejar una llamada a una función con un valor y una llamada con tres valores.

Trata las diferentes llamadas de manera similar. Tenga en cuenta que Python empaqueta los argumentos en una tupla, incluso si la función recibe solo un valor:

```
('peperoni',)
('champiñones', 'pimientos verdes', 'queso extra')
```

Ahora podemos reemplazar la llamada `print()` con un bucle que recorre el lista de ingredientes y describe la pizza que se pide:

```
def hacer_pizza(*ingredientes):
    """Resume la pizza que estamos a punto de hacer."""
```

```
print("\nPreparando una pizza con los siguientes ingredientes:")
para cubrir los aderezos:
    imprimir(f"- {cobertura}")
```

```
hacer_pizza('peperoni')
make_pizza('champiñones', 'pimientos verdes', 'queso extra')
```

La función responde apropiadamente, ya sea que reciba un valor o tres valores:

Hacer una pizza con los siguientes ingredientes:
- peperoni

Hacer una pizza con los siguientes ingredientes:
- hongos
- pimientos verdes
- extra queso

Esta sintaxis funciona sin importar cuántos argumentos reciba la función.

Mezclando argumentos posicionales y arbitrarios

Si desea que una función acepte varios tipos diferentes de argumentos, el parámetro que acepta un número arbitrario de argumentos debe colocarse al final en la definición de la función. Python primero compara los argumentos posicionales y de palabras clave y luego recopila los argumentos restantes en el parámetro final.

Por ejemplo, si la función necesita tomar un tamaño para la pizza, ese parámetro debe ir antes del parámetro *toppings:

```
def hacer_pizza(tamaño, *ingredientes):
    """Resume la pizza que estamos a punto de hacer."""
    print(f"\nHacer una pizza de {tamaño} pulgadas con los siguientes ingredientes:")
    para cubrir los aderezos:
        imprimir(f"- {cobertura}")

hacer_pizza(16, 'peperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

En la definición de la función, Python asigna el primer valor que recibe al tamaño del parámetro. Todos los demás valores que vienen después se almacenan en los ingredientes de la tupla. Las llamadas a funciones incluyen primero un argumento para el tamaño, seguido de tantos ingredientes como sean necesarios.

Ahora cada pizza tiene un tamaño y una cantidad de ingredientes, y cada información está impresa en el lugar correcto, mostrando primero el tamaño y los ingredientes.
después:

Hacer una pizza de 16 pulgadas con los siguientes ingredientes:
- peperoni

Hacer una pizza de 12 pulgadas con los siguientes ingredientes:

- hongos
- pimientos verdes
- extra queso

NOTA menudo verá el nombre del parámetro genérico `*args`, que recopila parámetros posicionales arbitrarios. argumentos como este.

Uso de argumentos de palabras clave arbitrarios

A veces querrás aceptar una cantidad arbitraria de argumentos, pero no sabrás de antemano qué tipo de información se pasará a la función. En este caso, puede escribir funciones que acepten tantos pares clave-valor como proporcione la declaración de llamada. Un ejemplo implica la creación de perfiles de usuario: sabes que obtendrás información sobre un usuario, pero no estás seguro de qué tipo de información recibirás. La función `build_profile()` en el siguiente ejemplo siempre toma un nombre y apellido, pero también acepta una cantidad arbitraria de argumentos de palabras clave:

```
perfil_usuario.py def perfil_compilación (primero, último, **info_usuario):
    """Construir un diccionario que contenga todo lo que sabemos sobre un usuario."""
    1 user_info['first_name'] = primero
      información_usuario['apellido'] = último
      devolver información_usuario

perfil_usuario = build_profile('albert', 'einstein',
                              ubicación='princeton',
                              campo = 'física')

imprimir (perfil_usuario)
```

La definición de `build_profile()` espera un nombre y apellido, y luego permite al usuario pasar tantos pares de nombre-valor como desee. Los asteriscos dobles antes del parámetro `**user_info` hacen que Python cree un diccionario llamado `user_info` que contiene todos los pares nombre-valor adicionales que recibe la función. Dentro de la función, puede acceder a los pares clave-valor en `user_info` tal como lo haría con cualquier diccionario.

En el cuerpo de `build_profile()`, agregamos el nombre y apellido al diccionario `user_info` porque siempre recibiremos estas dos piezas de información del usuario 1 y aún no se han colocado en el diccionario. Luego devolvemos el diccionario `user_info` a la línea de llamada de función.

Llamamos a `build_profile()`, pasándole el nombre 'albert', el apellido 'einstein' y los dos pares clave-valor `ubicación='princeton'` y `campo='física'`. Asignamos el perfil devuelto a `perfil_usuario` e imprimimos `perfil_usuario`:

```
{'ubicación': 'princeton', 'campo': 'física',
 'primer_nombre': 'alberto', 'apellido': 'einstein'}
```

El diccionario devuelto contiene el nombre y apellido del usuario y, en este caso, también la ubicación y el campo de estudio. La función funcionará sin importar cuántos pares clave-valor adicionales se proporcionen en la llamada a la función.

Puede mezclar valores posicionales, de palabras clave y arbitrarios de muchas maneras diferentes al escribir sus propias funciones. Es útil saber que todos estos tipos de argumentos existen porque los verás con frecuencia cuando empieces a leer el código de otras personas. Se necesita práctica para utilizar los diferentes tipos correctamente y saber cuándo utilizar cada tipo. Por ahora, recuerde utilizar el método más sencillo que haga el trabajo. A medida que avance, aprenderá a utilizar el enfoque más eficiente cada vez.

NOTA A menudo verá el nombre del parámetro `**kwargs` utilizado para recopilar argumentos de palabras clave no específicos.

INTÉNTALO TÚ MISMO

8-12. Sándwiches: escriba una función que acepte una lista de elementos que una persona quiere en un sándwich. La función debe tener un parámetro que recopile tantos elementos como proporcione la llamada a la función y debe imprimir un resumen del sándwich que se está pidiendo. Llame a la función tres veces, utilizando un número diferente de argumentos cada vez.

8-13. Perfil de usuario: comience con una copia de `user_profile.py` de la página 148. Cree un perfil suyo llamando a `build_profile()`, usando su nombre y apellido y otros tres pares clave-valor que lo describan.

8-14. Automóviles: escriba una función que almacene información sobre un automóvil en un diccionario. La función siempre debe recibir un nombre de fabricante y modelo. Luego debería aceptar un número arbitrario de argumentos de palabras clave. Llame a la función con la información requerida y otros dos pares de nombre-valor, como un color o una característica opcional. Su función debería funcionar para una llamada como esta:

```
auto = make_car('subaru', 'interior', color='azul', tow_package=True)
```

Imprima el diccionario que se devuelve para asegurarse de que toda la información se almacenó correctamente.

Almacenamiento de funciones en módulos

Una ventaja de las funciones es la forma en que separan bloques de código de su programa principal. Cuando utiliza nombres descriptivos para sus funciones, sus programas se vuelven mucho más fáciles de seguir. Puede ir un paso más allá almacenando sus funciones en un archivo separado llamado módulo y luego importando

ese módulo en su programa principal. Una declaración de importación le dice a Python que haga que el código de un módulo esté disponible en el archivo de programa que se está ejecutando actualmente.

Almacenar sus funciones en un archivo separado le permite ocultar los detalles del código de su programa y concentrarse en su lógica de nivel superior. También le permite reutilizar funciones en muchos programas diferentes. Cuando almacena sus funciones en archivos separados, puede compartir esos archivos con otros programadores sin

tener que compartir todo su programa. Saber cómo importar funciones también le permite utilizar bibliotecas de funciones que otros programadores han escrito.

Hay varias formas de importar un módulo y te mostraré cada una de ellas brevemente.

Importar un módulo completo

Para comenzar a importar funciones, primero necesitamos crear un módulo. Un módulo es un archivo que termina en .py y que contiene el código que desea importar a su programa. Hagamos un módulo que contenga la función `make_pizza()`. Para crear este módulo, eliminaremos todo del archivo `pizza.py` excepto la función `make_pizza()`:

```
pizza.py def make_pizza(tamaño, *ingredientes):
    """Resume la pizza que estamos a punto de hacer."""
    print(f"\nHacer una pizza de {tamaño} pulgadas con los siguientes ingredientes:")
    para cubrir los aderezos:
        imprimir(f"- {cobertura}")
```

Ahora crearemos un archivo separado llamado `making_pizzas.py` en el mismo directorio que `pizza.py`. Este archivo importa el módulo que acabamos de crear y luego realiza dos llamadas a `make_pizza()`:

```
making_pizzas.py
1 import pizza
  pizza.make_pizza(16, 'pepperoni')
  pizza.make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

Cuando Python lee este archivo, la línea `import pizza` le indica a Python que abra el archivo `pizza.py` y copie todas sus funciones en este programa. En realidad, no ve el código que se copia entre archivos porque Python copia el código detrás de escena, justo antes de que se ejecute el programa. Todo lo que necesitas saber es que cualquier función definida en `pizza.py` ahora estará disponible en `making_pizzas.py`.

Para llamar a una función desde un módulo importado, ingrese el nombre del módulo que importó, `pizza`, seguido del nombre de la función, `make_pizza()`, separados por un punto 1. Este código produce el mismo resultado que el programa original que no importó un módulo:

Hacer una pizza de 16 pulgadas con los siguientes ingredientes:
- pepperoni

Hacer una pizza de 12 pulgadas con los siguientes ingredientes:
- hongos
- pimientos verdes
- extra queso

Este primer método de importación, en el que simplemente escribe `import` seguido del nombre del módulo, hace que todas las funciones del módulo

disponible en su programa. Si utiliza este tipo de declaración de importación para importar un módulo completo llamado `module_name.py`, cada función en el módulo está disponible a través de la siguiente sintaxis:

```
nombre_módulo.nombre_función()
```

Importación de funciones específicas

También puede importar una función específica desde un módulo. Aquí está la sintaxis general de este enfoque:

```
desde nombre_módulo importar nombre_función
```

Puede importar tantas funciones como desee desde un módulo separando el nombre de cada función con una coma:

```
desde nombre_módulo importar función_0, función_1, función_2
```

El ejemplo de `making_pizzas.py` se vería así si queremos importar solo la función que vamos a usar:

```
de importación de pizza make_pizza
```

```
hacer_pizza(16, 'peperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

Con esta sintaxis, no necesita utilizar la notación de puntos cuando llama a una función. Debido a que hemos importado explícitamente la función `make_pizza()` en la declaración de importación, podemos llamarla por su nombre cuando usamos la función.

Usar `as` para darle un alias a una función

Si el nombre de una función que está importando puede entrar en conflicto con un nombre existente en su programa, o si el nombre de la función es largo, puede usar un alias corto y único, un nombre alternativo similar a un apodo para la función. Le darás a la función este apodo especial cuando la importes.

Aquí le damos a la función `make_pizza()` un alias, `mp()`, importando `make_pizza` como `mp`. La palabra clave `as` cambia el nombre de una función usando el alias que usted proporciona:

```
desde pizza importar make_pizza como mp
```

```
mp(16, 'peperoni')
mp(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

La declaración de importación que se muestra aquí cambia el nombre de la función `make_pizza()` a `mp()` en este programa. Cada vez que queramos llamar a `make_pizza()` podemos simplemente escribir `mp()` en su lugar, y Python ejecutará el código en `make_pizza()` evitando cualquier confusión con otra función `make_pizza()` que haya escrito en este archivo de programa.

La sintaxis general para proporcionar un alias es:

```
desde nombre_módulo importar nombre_función como fn
```

Usar as para darle un alias a un módulo

También puede proporcionar un alias para el nombre de un módulo. Darle a un módulo un alias corto, como `p` para `pizza`, le permite llamar a las funciones del módulo más rápidamente. Llamar a `p.make_pizza()` es más conciso que llamar a `pizza.make_pizza()`:

```
importar pizza como p
```

```
p.make_pizza(16, 'peperoni')
p.make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

Al módulo `pizza` se le asigna el alias `p` en la declaración de importación, pero todas las funciones del módulo conservan sus nombres originales. Llamar a las funciones escribiendo `p.make_pizza()` no sólo es más conciso que `pizza.make_pizza()`, sino que también redirige su atención desde el nombre del módulo y le permite centrarse en los nombres descriptivos de sus funciones. Estos nombres de funciones, que le indican claramente qué hace cada función, son más importantes para la legibilidad de su código que usar el nombre completo del módulo.

La sintaxis general de este enfoque es:

```
importar nombre_módulo como mn
```

Importar todas las funciones en un módulo

Puedes decirle a Python que importe cada función en un módulo usando el operador asterisk (*):

```
de importación de pizza *
```

```
hacer_pizza(16, 'peperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

El asterisco en la declaración de importación le dice a Python que copie cada función del módulo `pizza` en este archivo de programa. Debido a que cada función se importa, puede llamar a cada función por su nombre sin usar la notación de puntos. Sin embargo, es mejor no utilizar este enfoque cuando trabaja con módulos más grandes que usted no escribió: si el módulo tiene un nombre de función que coincide con un nombre existente en su proyecto, puede obtener resultados inesperados. Python puede ver varias funciones o variables con el mismo nombre y, en lugar de importar todas las funciones por separado, las sobrescribirá.

El mejor enfoque es importar la función o funciones que desee, o importar el módulo completo y usar la notación de puntos. Esto conduce a un código claro que es fácil de leer y comprender. Incluyo esta sección para que reconozcas declaraciones de importación como las siguientes cuando las veas en el código de otras personas:

```
desde la importación de nombre_módulo *
```

Funciones de estilo

Debes tener en cuenta algunos detalles al diseñar funciones.

Las funciones deben tener nombres descriptivos y estos nombres deben usar letras minúsculas y guiones bajos. Los nombres descriptivos le ayudan a usted y a otros a comprender lo que intenta hacer su código. Los nombres de los módulos también deben utilizar estas convenciones.

Cada función debe tener un comentario que explique de manera concisa qué hace la función. Este comentario debería aparecer inmediatamente después de la definición de la función y utilizar el formato de cadena de documentación. En una función bien documentada, otros programadores pueden usar la función leyendo solo la descripción en la cadena de documentación. Deberían poder confiar en que el código funciona como se describe y, siempre que sepan el nombre de la función, los argumentos que necesita y el tipo de valor que devuelve, deberían poder utilizarlo en sus programas.

Si especifica un valor predeterminado para un parámetro, no se deben utilizar espacios a cada lado del signo igual:

```
def nombre_función(parámetro_0, parámetro_1='valor predeterminado')
```

Se debe utilizar la misma convención para los argumentos de palabras clave en las llamadas a funciones:

```
nombre_función(valor_0, parámetro_1='valor')
```

PEP 8 (<https://www.python.org/dev/peps/pep-0008>) recomienda limitar las líneas de código a 79 caracteres para que cada línea sea visible en una ventana del editor de tamaño razonable. Si un conjunto de parámetros hace que la definición de una función tenga más de 79 caracteres, presione ENTRAR después del paréntesis inicial en la línea de definición. En la siguiente línea, presione la tecla TAB dos veces para separar la lista de argumentos del cuerpo de la función, que solo tendrá una sangría de un nivel.

La mayoría de los editores alinean automáticamente cualquier línea adicional de argumentos para coincidir con la sangría que has establecido en la primera línea:

```
def nombre_función(
    parámetro_0, parámetro_1, parámetro_2,
    parámetro_3, parámetro_4, parámetro_5):
    cuerpo funcional...
```

Si su programa o módulo tiene más de una función, puede separarlas. califique cada uno con dos líneas en blanco para que sea más fácil ver dónde termina una función y comienza la siguiente.

Todas las declaraciones de importación deben escribirse al principio de un archivo. El La única excepción es si utiliza comentarios al principio de su archivo para describir el programa general.

INTÉNTALO TÚ MISMO

8-15. Modelos de impresión: coloque las funciones para el ejemplo `Printing_models.py` en un archivo separado llamado `Printing_functions.py`. Escriba una declaración de importación en la parte superior de `Printing_models.py` y modifique el archivo para usar las funciones importadas.

8-16. Importaciones: utilizando un programa que usted escribió y que tiene una función, almacene esa función en un archivo separado. Importe la función a su archivo de programa principal y llame a la función utilizando cada uno de estos métodos:

```
importar nombre_módulo
desde nombre_módulo importar nombre_función
desde nombre_módulo importar nombre_función como fn
importar nombre_módulo como mn_*
desde la importación de nombre_módulo
```

8-17. Funciones de estilo: elija cualquiera de los tres programas que escribió para este capítulo y asegúrese de que sigan las pautas de estilo descritas en esta sección.

Resumen

En este capítulo, aprendió a escribir funciones y a pasar argumentos para que sus funciones tengan acceso a la información que necesitan para realizar su trabajo. Aprendió a utilizar argumentos posicionales y de palabras clave, y también a aceptar una cantidad arbitraria de argumentos. Viste funciones que muestran resultados y funciones que devuelven valores. Aprendiste a usar funciones con listas, diccionarios, sentencias `if` y bucles `while`. También vio cómo almacenar sus funciones en archivos separados llamados módulos, por lo que sus archivos de programa serán más simples y fáciles de entender. Finalmente, aprendió a diseñar sus funciones para que sus programas sigan estando bien estructurados y lo más fáciles de leer posible para usted y otros.

Uno de tus objetivos como programador debería ser escribir código simple que haga lo que quieres, y las funciones te ayudarán a lograrlo. Le permiten escribir bloques de código y dejarlos en paz una vez que sepa que funcionan. Cuando sabes que una función hace su trabajo correctamente, puedes confiar en que seguirá funcionando y pasará a tu siguiente tarea de codificación.

Las funciones le permiten escribir código una vez y luego reutilizarlo tantas veces como desee. Cuando necesitas ejecutar el código en una función, todo lo que necesitas hacer es escribir una llamada de una línea y la función hará su trabajo. Cuando necesita modificar el comportamiento de una función, solo tiene que modificar un bloque de código y su cambio tendrá efecto en todos los lugares donde haya realizado una llamada a esa función.

El uso de funciones hace que sus programas sean más fáciles de leer y los buenos nombres de las funciones resumen lo que hace cada parte de un programa. Leer una serie de llamadas a funciones le brinda una idea mucho más rápida de lo que hace un programa que leer una larga serie de bloques de código.

Las funciones también hacen que su código sea más fácil de probar y depurar. cuando la mayor parte Dado que el trabajo de su programa se realiza mediante un conjunto de funciones, cada una de las cuales tiene un trabajo específico, es mucho más fácil probar y mantener el código que ha escrito. Puede escribir un programa separado que llame a cada función y pruebe si cada función funciona en todas las situaciones que pueda encontrar. Cuando hace esto, puede estar seguro de que sus funciones funcionarán correctamente cada vez que las llame.

En el Capítulo 9, aprenderá a escribir clases. Las clases combinan funciones y datos en un paquete ordenado que se puede utilizar de manera flexible y eficiente.

9

CLASES



La programación orientada a objetos (POO) es uno de los enfoques más eficaces para escribir software. En la programación orientada a objetos, se escriben clases que representan cosas y situaciones del mundo real y se crean objetos basados en estas clases. Cuando escribes una clase, defines el comportamiento general que puede tener toda una categoría de objetos.

Cuando creas objetos individuales de la clase, cada objeto se crea automáticamente. Equipado automáticamente con el comportamiento general; Luego puedes darle a cada objeto los rasgos únicos que desees. Te sorprenderá lo bien que se pueden modelar situaciones del mundo real con programación orientada a objetos.

Crear un objeto a partir de una clase se llama creación de instancias y se trabaja con instancias de una clase. En este capítulo escribirás clases y crearás instancias de esas clases. Especificará el tipo de información que se puede almacenar en las instancias y definirá las acciones que se pueden realizar con estas instancias.

También escribirás clases que amplíen la funcionalidad de las clases existentes, de modo que clases similares puedan compartir una funcionalidad común y puedas hacer más con

menos código. Almacenará sus clases en módulos e importará clases escritas por otros programadores en sus propios archivos de programa.

Aprender sobre programación orientada a objetos te ayudará a ver el mundo como lo hace un programador. Le ayudará a comprender su código, no solo lo que sucede línea por línea, sino también los conceptos más amplios detrás de él. Conocer la lógica detrás de las clases lo capacitará para pensar de manera lógica, de modo que pueda escribir programas que aborden de manera efectiva casi cualquier problema que encuentre.

Las clases también le facilitan la vida a usted y a los demás programadores con los que trabajará a medida que asume desafíos cada vez más complejos. Cuando usted y otros programadores escriban código basado en el mismo tipo de lógica, podrán comprender el trabajo de los demás. Sus programas tendrán sentido para las personas con las que trabaja y permitirán que todos logren más.

Crear y usar una clase

Puedes modelar casi cualquier cosa usando clases. Comencemos escribiendo una clase simple, Perro, que represente un perro, no un perro en particular, sino cualquier perro. ¿Qué sabemos sobre la mayoría de los perros domésticos? Pues todos tienen un nombre y una edad. También sabemos que la mayoría de los perros se sientan y se dan vuelta. Esos dos datos (nombre y edad) y esos dos comportamientos (sentarse y darse vuelta) irán en nuestro Perro. clase porque son comunes a la mayoría de los perros. Esta clase le dirá a Python cómo crear un objeto que represente a un perro. Una vez escrita nuestra clase, la usaremos para crear instancias individuales, cada una de las cuales representa un perro específico.

Creando la clase de perro

Cada instancia creada a partir de la clase Perro almacenará un nombre y una edad, y le daremos a cada perro la capacidad de sentarse() y rodar():

dog.py 1 clase Perro:

```

    """Un simple intento de modelar un perro."""

2 def __init__(yo, nombre, edad):
    """Inicializar atributos de nombre y edad."""
3     self.nombre = nombre
    self.age = edad

4 definitivamente sentarse (yo):
    """Simular un perro sentado en respuesta a una orden."""
    print(f'{self.name} ahora está sentado.')

    def roll_over(yo):
        """Simular darse vuelta en respuesta a una orden."""
        print(f'{self.name} volteado!")
```

Hay mucho que notar aquí, pero no te preocupes. Verá esta estructura a lo largo de este capítulo y tendrá mucho tiempo para acostumbrarse a ella. Primero definimos una clase llamada Perro 1. Por convención, los nombres en mayúscula se refieren a clases en Python. No hay paréntesis en la definición de clase porque estamos creando esta clase desde cero. Luego escribimos una cadena de documentación que describe lo que hace esta clase.

El método `__init__()`

Una función que es parte de una clase es un método. Todo lo que aprendiste sobre funciones se aplica también a los métodos; La única diferencia práctica por ahora es la forma en que llamaremos a los métodos. El método `__init__()` ² es un método especial que Python ejecuta automáticamente cada vez que creamos una nueva instancia basada en la clase `Dog`. Este método tiene dos guiones bajos iniciales y dos guiones bajos finales, una convención que ayuda a evitar que los nombres de los métodos predeterminados de Python entren en conflicto con los nombres de sus métodos. Asegúrese de utilizar dos guiones bajos a cada lado de `__init__()`. Si usa solo uno en cada lado, el método no se llamará automáticamente cuando use su clase, lo que puede resultar en errores difíciles de identificar.

Definimos el método `__init__()` para que tenga tres parámetros: `self`, nombre y edad. El parámetro `self` es obligatorio en la definición del método y debe aparecer primero, antes que los demás parámetros. Debe incluirse en la definición porque cuando Python llame a este método más adelante (para crear una instancia de `Dog`), la llamada al método pasará automáticamente el argumento `self`.

Cada llamada a un método asociado con una instancia pasa automáticamente a `self`, que es una referencia a la instancia misma; le da a la instancia individual acceso a los atributos y métodos de la clase. Cuando creamos una instancia de `Dog`, Python llamará al método `__init__()` de la clase `Dog`.

Le pasaremos a `Dog()` un nombre y una edad como argumentos; `self` se pasa automáticamente, por lo que no es necesario que lo pasemos. Siempre que queramos crear una instancia de la clase `Perro`, proporcionaremos valores solo para los dos últimos parámetros, nombre y edad.

Las dos variables definidas en el cuerpo del método `__init__()` tienen cada una el prefijo `self` ³. Cualquier variable con el prefijo `self` está disponible para todos los métodos de la clase, y también podremos acceder a estas variables a través de cualquier instancia creada desde la clase. La línea `self.name = name` toma el valor asociado con el nombre del parámetro y lo asigna al nombre de la variable, que luego se adjunta a la instancia que se está creando. El mismo proceso ocurre con `self.age = edad`. Las variables a las que se puede acceder a través de instancias como esta se denominan atributos.

La clase `Dog` tiene otros dos métodos definidos: `sit()` y `roll_over()` ⁴.

Debido a que estos métodos no necesitan información adicional para ejecutarse, simplemente los definimos para que tengan un parámetro, `self`. Las instancias que creamos más adelante tendrán acceso a estos métodos. En otras palabras, podrán sentarse y darse la vuelta. Por ahora, `sit()` y `roll_over()` no hacen mucho. Simplemente imprimen un mensaje que dice que el perro está sentado o dándose vuelta. Pero el concepto puede extenderse a situaciones realistas: si esta clase fuera parte de un juego de computadora, estos métodos contendrían código para hacer que un perro animado se sentara y se volteara. Si esta clase se escribiera para controlar un robot, estos métodos dirigirían los movimientos que hacen que un perro robótico se siente y se dé vuelta.

Crear una instancia a partir de una clase

Piense en una clase como un conjunto de instrucciones sobre cómo crear una instancia. La clase `Perro` es un conjunto de instrucciones que le dicen a Python cómo crear instancias individuales que representen perros específicos.

Hagamos una instancia que represente un perro específico:

```
perro de clase:
```

```
--recorte--
```

```
1 mi_perro = Perro('Willie', 6)

2 print(f"El nombre de mi perro es {my_dog.name}.")
3 print(f"Mi perro tiene {my_dog.age} años.")
```

La clase Perro que estamos usando aquí es la que acabamos de escribir en el ejemplo anterior. Aquí le decimos a Python que cree un perro cuyo nombre sea 'Willie' y cuya edad es 6 1. Cuando Python lee esta línea, llama a `__init__()` método en Perro con los argumentos 'Willie' y 6. El método `__init__()` crea una instancia que representa a este perro en particular y establece los atributos de nombre y edad utilizando los valores que proporcionamos. Luego, Python devuelve una instancia que representa a este perro. Asignamos esa instancia a la variable `my_dog`. La convención de nomenclatura resulta útil en este caso; Generalmente podemos asumir que un nombre en mayúscula como Dog se refiere a una clase, y un nombre en minúscula como `my_dog` se refiere a una única instancia creada a partir de una clase.

Accediendo a los atributos

Para acceder a los atributos de una instancia, se utiliza la notación de puntos. Accedemos al valor del nombre del atributo 2 de `my_dog` escribiendo:

```
mi_perro.nombre
```

La notación de puntos se usa con frecuencia en Python. Esta sintaxis demuestra cómo Python encuentra el valor de un atributo. Aquí, Python mira la instancia `my_dog` y luego busca el nombre del atributo asociado con `my_dog`. Este es el mismo atributo al que se hace referencia como `self.name` en la clase Perro. Usamos el mismo enfoque para trabajar con el atributo edad 3.

El resultado es un resumen de lo que sabemos sobre `my_dog`:

```
El nombre de mi perro es Willie.
Mi perro tiene 6 años.
```

Métodos de llamada

Después de crear una instancia de la clase Dog, podemos usar la notación de puntos para llamar a cualquier método definido en Dog. Hagamos que nuestro perro se siente y se dé vuelta:

```
perro de clase:
```

```
--recorte--
```

```
mi_perro = Perro('Willie', 6)
mi_perro.sit()
mi_perro.roll_over()
```

Para llamar a un método, proporcione el nombre de la instancia (en este caso, `my_dog`) y el método que desea llamar, separados por un punto. Cuando Python lee `my_dog.sit()`, busca el método `sit()` en la clase `Dog` y ejecuta ese código. Python interpreta la línea `my_dog.roll_over()` de la misma manera.

Ahora Willie hace lo que le decimos:

```
Willie ahora está sentado.
¡Willie se dio la vuelta!
```

Esta sintaxis es bastante útil. Cuando a los atributos y métodos se les han dado nombres descriptivos apropiados como nombre, edad, `sit()` y `roll_over()`, podemos inferir fácilmente qué se supone que debe hacer un bloque de código, incluso uno que nunca hayamos visto antes.

Crear múltiples instancias

Puede crear tantas instancias de una clase como necesite. Creemos un segundo perro llamado `tu_perro`:

```
perro de clase:
--recorte--

mi_perro = Perro('Willie', 6)
tu_perro = Perro('Lucy', 3)

print(f"El nombre de mi perro es {my_dog.name}.")
print(f"Mi perro tiene {my_dog.age} años.")
mi_perro.sit()

print(f"\nEl nombre de tu perro es {tu_perro.nombre}.")
print(f"Tu perro tiene {your_dog.age} años.")
tu_perro.sit()
```

En este ejemplo creamos un perro llamado Willie y un perro llamado Lucy.

Cada perro es una instancia separada con su propio conjunto de atributos, capaz de realizar el mismo conjunto de acciones:

```
El nombre de mi perro es Willie.
Mi perro tiene 6 años.
Willie ahora está sentado.
```

```
El nombre de tu perro es Lucy.
Tu perro tiene 3 años.
Lucy ahora está sentada.
```

Incluso si usáramos el mismo nombre y edad para el segundo perro, Python aún crearía una instancia separada de la clase `Perro`. Puede crear tantas instancias de una clase como necesite, siempre que le dé a cada instancia un nombre de variable único o ocupe un lugar único en una lista o diccionario.

INTÉNTALO TÚ MISMO

9-1. Restaurante: Crea una clase llamada Restaurante. El método `__init__()` para Restaurante debe almacenar dos atributos: un `nombre_restaurante` y un `tipo_cocina`. Cree un método llamado `describe_restaurant()` que imprima estos dos datos y un método llamado `open_restaurant()` que imprima un mensaje que indique que el restaurante está abierto.

Crea una instancia llamada `restaurante` de tu clase. Imprima los dos atributos individualmente y luego llame a ambos métodos.

9-2. Tres Restaurantes: Comience con su clase del Ejercicio 9-1. Cree tres instancias diferentes de la clase y llame a `describe_restaurant()` para cada instancia.

9-3. Usuarios: cree una clase llamada Usuario. Crea dos atributos llamados `first_name` y `apellido`, y luego cree varios otros atributos que normalmente se almacenan en un perfil de usuario. Cree un método llamado `describe_user()` que imprima un resumen de la información del usuario. Cree otro método llamado `greet_user()` que imprima un saludo personalizado para el usuario.

Cree varias instancias que representen a diferentes usuarios y llame a ambos métodos para cada usuario.

Trabajar con clases e instancias

Puede utilizar clases para representar muchas situaciones del mundo real. Una vez que escriba una clase, pasará la mayor parte de su tiempo trabajando con instancias creadas a partir de esa clase. Una de las primeras tareas que querrás realizar es modificar los atributos asociados con una instancia en particular. Puede modificar los atributos de una instancia directamente o escribir métodos que actualicen los atributos de formas específicas.

La clase de coche

Escribamos una nueva clase que represente un automóvil. Nuestra clase almacenará información sobre el tipo de automóvil con el que estamos trabajando y tendrá un método que resuma esta información:

coche de clase `car.py`:

```
"""Un simple intento de representar un coche."""

1 def __init__(propio, marca, modelo, año):
    """Inicializar atributos para describir un automóvil."""
    self.make = hacer
    self.modelo = modelo
    self.año = año

2 def get_descriptive_name(self):
    """Devuelve un nombre descriptivo perfectamente formateado."""
    nombre_largo = f'{self.año} {self.make} {self.modelo}'
```

```
devolver nombre_largo.titulo()
```

```
3 mi_coche_nuevo = Coche('audi', 'a4', 2024)
imprimir(mi_auto_nuevo.get_nombre_descriptivo())
```

En la clase Car , definimos el método `__init__()` con el parámetro `self`

El primero, tal como lo hicimos con la clase Perro . También le damos otros tres parámetros: marca, modelo y año. El método `__init__()` toma estos parámetros y los asigna a los atributos que se asociarán con las instancias creadas a partir de esta clase. Cuando creamos una nueva instancia de Car , necesitaremos especificar una marca, modelo y año para nuestra instancia.

Definimos un método llamado `get_descriptive_name()` 2 que coloca el año , la marca y el modelo de un automóvil en una cadena que describe claramente el automóvil. Esto nos evitará tener que imprimir el valor de cada atributo individualmente. Para trabajar con los valores de los atributos en este método, utilizamos `self.make`, `self.model` y `self.year`.

Fuera de la clase, creamos una instancia de la clase Car y la asignamos a la variable `my_new_car` 3. Luego llamamos a `get_descriptive_name()` para mostrar qué tipo de auto tenemos:

```
2024 Audi A4
```

Para hacer la clase más interesante, agreguemos un atributo que cambie con el tiempo. Agregaremos un atributo que almacena el kilometraje total del automóvil.

Establecer un valor predeterminado para un atributo

Cuando se crea una instancia, los atributos se pueden definir sin pasarlos como parámetros. Estos atributos se pueden definir en `__init__()` método, donde se les asigna un valor predeterminado.

Agreguemos un atributo llamado `odometer_reading` que siempre comienza con un valor de 0. También agregaremos un método `read_odometer()` que nos ayuda a leer el odómetro de cada automóvil:

coche de clase:

```
def __init__(propio, marca, modelo, año):
    """Inicializar atributos para describir un automóvil."""
    self.make = hacer
    self.modelo = modelo
    self.año = año
1    self.odometer_reading = 0

    def get_descriptive_name(self):
        --recorte--

2 def read_odometer(yo):
    """Imprima un estado de cuenta que muestre el kilometraje del automóvil."""
    print(f"Este auto tiene {self.odometer_reading} millas.")

my_new_car = Coche('audi', 'a4', 2024)
imprimir(mi_auto_nuevo.get_nombre_descriptivo())
my_new_car.read_odometer()
```

Esta vez, cuando Python llama al método `__init__()` para crear una nueva instancia, almacena los valores de marca, modelo y año como atributos, como lo hizo en el ejemplo anterior. Luego Python crea un nuevo atributo llamado `odometer_reading` y establece su valor inicial en 0. También tenemos un nuevo método llamado `read_odometer()` que facilita la lectura del kilometraje de un automóvil.

Nuestro coche arranca con un kilometraje de 0:

```
2024 Audi A4
Este auto tiene 0 millas.
```

No se venden muchos autos con exactamente 0 millas en el odómetro, por lo que Necesito una forma de cambiar el valor de este atributo.

Modificar valores de atributos

Puede cambiar el valor de un atributo de tres maneras: puede cambiar el valor directamente a través de una instancia, establecer el valor a través de un método o incrementar el valor (agregarle una cierta cantidad) a través de un método. Veamos cada uno de estos enfoques.

Modificar el valor de un atributo directamente

La forma más sencilla de modificar el valor de un atributo es acceder al atributo directamente a través de una instancia. Aquí configuramos la lectura del odómetro directamente en 23:

```
coche de clase:
--recorte--

my_new_car = Coche('audi', 'a4', 2024)
imprimir(mi_auto_nuevo.get_nombre_descriptivo())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

Usamos notación de puntos para acceder al atributo `odometer_reading` del automóvil y establecemos su valor directamente. Esta línea le dice a Python que tome la instancia `my_new_car`, busque el atributo `odometer_reading` asociado con ella y establezca el valor de ese atributo en 23:

```
2024 Audi A4
Este auto tiene 23 millas.
```

A veces querrás acceder a atributos directamente como este, pero otras A veces querrás escribir un método que actualice el valor por ti.

Modificar el valor de un atributo mediante un método

Puede resultar útil disponer de métodos que actualicen determinados atributos. En lugar de acceder al atributo directamente, pasa el nuevo valor a un método que maneja la actualización internamente.

Aquí hay un ejemplo que muestra un método llamado `update_odometer()`:

coche de clase:

--recorte--

```
def update_odometer(self, kilometraje):
    """Ajuste la lectura del odómetro al valor indicado."""
    self.odometer_reading = kilometraje
```

```
my_new_car = Coche('audi', 'a4', 2024)
imprimir(mi_auto_nuevo.get_nombre_descriptivo())
```

```
1 my_new_car.update_odometer(23)
  my_new_car.read_odometer()
```

La única modificación de Car es la adición de `update_odometer()`. Este método toma un valor de kilometraje y lo asigna a `self.odometer_reading`.

Usando la instancia `my_new_car`, llamamos a `update_odometer()` con 23 como argumento 1. Esto establece la lectura del odómetro en 23 y `read_odometer()` imprime la lectura:

2024 Audi A4
Este auto tiene 23 millas.

Podemos extender el método `update_odometer()` para realizar trabajo adicional cada vez que se modifica la lectura del odómetro. Agreguemos un poco de lógica para asegurarnos de que nadie intente revertir la lectura del odómetro:

coche de clase:

--recorte--

```
def update_odometer(self, kilometraje):
    """
    Establezca la lectura del odómetro al valor indicado.
    Rechace el cambio si intenta hacer retroceder el odómetro.
    """
```

```
1     si kilometraje >= self.odometer_reading:
        self.odometer_reading = kilometraje
    demás:
2         print("¡No puedes hacer retroceder un odómetro!")
```

Ahora `update_odometer()` comprueba que la nueva lectura tenga sentido antes modificando el atributo. Si el valor proporcionado para el kilometraje es mayor o igual al kilometraje existente, `self.odometer_reading`, puede actualizar la lectura del odómetro al nuevo kilometraje 1. Si el nuevo kilometraje es menor que el kilometraje existente, recibirá una advertencia que No puedes retroceder un odómetro 2.

Incrementar el valor de un atributo mediante un método

A veces querrás incrementar el valor de un atributo en una cantidad determinada, en lugar de establecer un valor completamente nuevo. Digamos que compramos un auto usado y recorrimos 100 millas.

en él entre el momento en que lo compramos y el momento en que lo registramos. Aquí hay un método que nos permite pasar esta cantidad incremental y agregar ese valor a la lectura del odómetro:

coche de clase:

--recorte--

```
def update_odometer(self, kilometraje):
```

--recorte--

```
def incremento_odómetro(yo, millas):
```

"""Suma la cantidad indicada a la lectura del odómetro."""

self.odometer_reading += millas

```
1 my_used_car = Coche('subaru', 'interior', 2019)
  print(mi_coche_usado.get_nombre_descriptivo())
```

```
2 mi_coche_usado.update_odometer(23_500)
  mi_coche_usado.read_odometer()
```

```
mi_coche_usado.increment_odometer(100)
```

```
mi_coche_usado.read_odometer()
```

El nuevo método `increment_odometer()` toma una cantidad de millas y agrega este valor a `self.odometer_reading`. Primero, creamos un auto usado, `my_used_car` 1.

Configuramos su odómetro en 23,500 llamando a `update_odometer()` y pasándolo 23_500 2.

Finalmente, llamamos a `increment_odometer()` y lo pasamos 100 para sumar las 100 millas que recorrimos entre comprar el auto y registrarlo:

Subaru Outback 2019

Este auto tiene 23500 millas.

Este auto tiene 23600 millas.

Puede modificar este método para rechazar incrementos negativos, de modo que nadie use esta función para retroceder también un odómetro.

NOTA Puede utilizar métodos como este para controlar cómo los usuarios de su programa actualizan los valores.

como la lectura del odómetro, pero cualquier persona con acceso al programa puede configurar la lectura del odómetro en cualquier valor accediendo directamente al atributo. La seguridad eficaz requiere extrema atención a los detalles, además de comprobaciones básicas como las que se muestran aquí.

INTÉNTALO TÚ MISMO

9-4. Número atendido: Comience con su programa del ejercicio 9-1 (página 162).

Agregue un atributo llamado `number_served` con un valor predeterminado de 0. Cree una instancia llamada `restaurant` a partir de esta clase. Imprima la cantidad de clientes que ha atendido el restaurante, luego cambie este valor e imprímalo nuevamente.

Agregue un método llamado `set_number_served()` que le permita establecer la cantidad de clientes que han sido atendidos. Llame a este método con un nuevo número e imprima el valor nuevamente.

Agregue un método llamado `increment_number_served()` que le permita incrementar la cantidad de clientes que han sido atendidos. Llame a este método con cualquier número que desee que pueda representar cuántos clientes fueron atendidos en, digamos, un día hábil.

9-5. Intentos de inicio de sesión: agregue un atributo llamado `login_attempts` a su clase de Usuario del Ejercicio 9-3 (página 162). Escriba un método llamado `increment_login_attempts()` que incremente el valor de `login_attempts` en 1. Escriba otro método llamado `reset_login_attempts()` que restablezca el valor de `login_attempts` a 0.

Cree una instancia de la clase Usuario y llame a `increment_login_attempts()` varias veces. Imprima el valor de `login_attempts` para asegurarse de que se incrementó correctamente y luego llame a `reset_login_attempts()`. Imprima `login_attempts` nuevamente para asegurarse de que se haya restablecido a 0.

Herencia

No siempre es necesario empezar desde cero al escribir una clase. Si la clase que estás escribiendo es una versión especializada de otra clase que escribiste, puedes usar la herencia. Cuando una clase hereda de otra, adopta los atributos y métodos de la primera clase. La clase original se llama clase principal y la nueva clase es clase secundaria. La clase hija puede heredar cualquiera o todos los atributos y métodos de su clase padre, pero también es libre de definir nuevos atributos y métodos propios.

El método `__init__()` para una clase secundaria

Cuando escribes una nueva clase basada en una clase existente, a menudo querrás llamar al método `__init__()` desde la clase principal. Esto inicializará cualquier atributo que se haya definido en el método principal `__init__()` y los pondrá a disposición en la clase secundaria.

Como ejemplo, modelemos un coche eléctrico. Un automóvil eléctrico es solo un tipo específico de automóvil, por lo que podemos basar nuestra nueva clase `ElectricCar` en la clase `Car` que escribimos anteriormente. Entonces sólo tendremos que escribir código para los atributos y comportamientos específicos de los coches eléctricos.

Comencemos haciendo una versión simple de la clase `ElectricCar`, que hace todo lo que hace la clase `Car`:

`electric_car.py` Coche de 1 clase:

```
"""Un simple intento de representar un coche."""

def __init__(propio, marca, modelo, año):
    """Inicializar atributos para describir un automóvil."""
    self.make = hacer
    self.modelo = modelo
```

```

        self.año = año
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Devuelve un nombre descriptivo perfectamente formateado."""
        nombre_largo = f'{self.año} {self.make} {self.modelo}'
        devolver nombre_largo.titulo()

    def read_odometer(self):
        """Imprima un estado de cuenta que muestre el kilometraje del automóvil."""
        print(f"Este auto tiene {self.odometer_reading} millas.")

    def update_odometer(self, kilometraje):
        """Ajuste la lectura del odómetro al valor indicado."""
        si kilometraje >= self.odometer_reading:
            self.odometer_reading = kilometraje
        demás:
            print(";¡No puedes hacer retroceder un odómetro!")

    def incremento_odómetro(yo, millas):
        """Suma la cantidad indicada a la lectura del odómetro."""
        self.odometer_reading += millas

Coche eléctrico de 2 clases (coche):
    """Representan aspectos de un automóvil, específicos de los vehículos eléctricos."""

3 def __init__(propio, marca, modelo, año):
    """Inicializar atributos de la clase padre."""
4     super().__init__(marca, modelo, año)

5 mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
    imprimir(mi_hoja.get_nombre_descriptivo())

```

Comenzamos con el Coche 1. Cuando crea una clase secundaria, la clase principal debe ser parte del archivo actual y debe aparecer antes de la clase secundaria en el archivo. Luego definimos la clase secundaria, ElectricCar 2. El nombre de la clase principal debe incluirse entre paréntesis en la definición de una clase secundaria. El método `__init__()` toma la información necesaria para crear un automóvil . instancia 3.

La función `super()` 4 es una función especial que le permite llamar a un método desde la clase principal. Esta línea le dice a Python que llame a `__init__()` método de Car, que proporciona a una instancia de ElectricCar todos los atributos definidos en ese método. El nombre `super` proviene de una convención de llamar a la clase principal superclase y a la clase secundaria subclase.

Probamos si la herencia funciona correctamente intentando crear un coche eléctrico con el mismo tipo de información que proporcionaríamos al fabricar un coche normal. Creamos una instancia de la clase ElectricCar y la asignamos a `my_leaf` 5. Esta línea llama al método `__init__()` definido en ElectricCar, que a su vez le dice a Python que llame al método `__init__()` definido en la clase principal Car. Proporcionamos los argumentos 'nissan', 'leaf' y 2024.

Aparte de `__init__()`, todavía no existen atributos o métodos específicos de un automóvil eléctrico. En este punto sólo nos estamos asegurando de que el coche eléctrico tenga los comportamientos de coche adecuados:

2024 Nissan Leaf

La instancia de `ElectricCar` funciona igual que una instancia de `Car`, por lo que ahora podemos empezar a definir atributos y métodos específicos de los coches eléctricos.

Definición de atributos y métodos para la clase secundaria

Una vez que tenga una clase secundaria que herede de una clase principal, puede agregar todos los atributos y métodos nuevos necesarios para diferenciar la clase secundaria de la clase principal.

Agreguemos un atributo específico de los automóviles eléctricos (una batería, por ejemplo) y un método para informar sobre este atributo. Almacenaremos el tamaño de la batería y escribiremos un método que imprima una descripción de la batería:

coche de clase:

--recorte--

clase `ElectricCar(Coche)`:

"""Representan aspectos de un automóvil, específicos de los vehículos eléctricos."""

def `__init__`(propio, marca, modelo, año):

 Inicializa los atributos de la clase padre.

 Luego inicialice los atributos específicos de un automóvil eléctrico.

```
1     super().__init__(marca, modelo, año)
       self.battery_size = 40
```

```
2 def describe_batería (auto):
```

 """Imprima una declaración que describa el tamaño de la batería."""

 print(f"Este auto tiene una batería de {self.battery_size} kWh.")

`mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)`

`imprimir(mi_hoja.get_nombre_descriptivo())`

`my_leaf.describe_battery()`

Agregamos un nuevo atributo `self.battery_size` y establecemos su valor inicial en 40

1. Este atributo se asociará con todas las instancias creadas a partir de la clase `ElectricCar`, pero no se asociará con ninguna instancia de `Car`. También agregamos un método llamado `describe_battery()` que imprime información sobre la batería
2. Cuando llamamos a este método, obtenemos una descripción que es claramente específica de un automóvil eléctrico:

2024 Nissan Leaf

Este coche tiene una batería de 40 kWh.

No hay límite en cuanto a cuánto puedes especializar la clase `ElectricCar`. Puede agregar tantos atributos y métodos como necesite para modelar un

coche eléctrico con el grado de precisión que necesites. Un atributo o método que podría pertenecer a cualquier automóvil, en lugar de uno que sea específico de un automóvil eléctrico, debe agregarse a la clase Car en lugar de a la clase ElectricCar . Entonces, cualquiera que use la clase Car también tendrá esa funcionalidad disponible, y la clase ElectricCar solo contendrá código para la información y el comportamiento específicos de los vehículos eléctricos.

Anulación de métodos de la clase principal

Puede anular cualquier método de la clase principal que no se ajuste a lo que intenta modelar con la clase secundaria. Para hacer esto, define un método en la clase secundaria con el mismo nombre que el método que desea anular en la clase principal. Python ignorará el método de la clase principal y solo prestará atención al método que usted defina en la clase secundaria.

Digamos que la clase Car tenía un método llamado `fill_gas_tank()`. Este método no tiene sentido para un vehículo totalmente eléctrico, por lo que es posible que desees anularlo. Aquí hay una forma de hacerlo:

```

clase ElectricCar(Coche):
    --recorte--

    def fill_gas_tank(yo):
        """Los coches eléctricos no tienen depósito de gasolina."""
        print("¡Este auto no tiene tanque de gasolina!")

```

Ahora, si alguien intenta llamar a `fill_gas_tank()` con un coche eléctrico, Python ignorará el método `fill_gas_tank()` en Car y ejecutará este código en su lugar. Cuando usas la herencia, puedes hacer que tus clases secundarias conserven lo que necesitas y anulen todo lo que no necesitas de la clase principal.

Instancias como atributos

Al modelar algo del mundo real en código, es posible que agregues más y más detalles a una clase. Descubrirá que tiene una lista cada vez mayor de atributos y métodos y que sus archivos se están volviendo largos. En estas situaciones, es posible que reconozca que parte de una clase se puede escribir como una clase separada. Puede dividir su clase grande en clases más pequeñas que trabajen juntas; este enfoque se llama composición.

Por ejemplo, si continuamos agregando detalles a la clase ElectricCar , podríamos notar que estamos agregando muchos atributos y métodos específicos de la batería del automóvil. Cuando veamos que esto sucede, podemos detener y mover esos atributos y métodos a una clase separada llamada Batería. Luego podemos usar una instancia de Batería como atributo en la clase ElectricCar :

```

coche de clase:
    --recorte--

batería de clase:
    """Un sencillo intento de modelar una batería para un coche eléctrico."""

1 def __init__(self, tamaño_batería=40):

```

```

        """Inicializar los atributos de la batería."""
        self.battery_size = tamaño_batería

2 def describe_batería (auto):
    """Imprima una declaración que describa el tamaño de la batería."""
    print(f"Este auto tiene una batería de {self.battery_size}-kWh.")

class ElectricCar(Coche):
    """Representan aspectos de un automóvil, específicos de los vehículos eléctricos."""

    def __init__(propio, marca, modelo, año):

        Inicializa los atributos de la clase padre.
        Luego inicialice los atributos específicos de un automóvil eléctrico.

        super().__init__(marca, modelo, año)
3         self.battery = Batería()

mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
imprimir(mi_hoja.get_nombre_descriptivo())
my_leaf.battery.describe_battery()

```

Definimos una nueva clase llamada Batería que no hereda de ninguna otra clase. El método `__init__()` 1 tiene un parámetro, `tamaño_batería`, además de `self`. Este es un parámetro opcional que establece el tamaño de la batería en 40 si no se proporciona ningún valor. El método `describe_battery()` también se ha movido a esta clase 2.

En la clase `ElectricCar`, ahora agregamos un atributo llamado `self.battery` 3. Esta línea le dice a Python que cree una nueva instancia de `Battery` (con un tamaño predeterminado de 40, porque no especificamos un valor) y asigne esa instancia al atributo `self.battery`. Esto sucederá cada vez que se llame al método `__init__()`; cualquier instancia de `ElectricCar` ahora tendrá una instancia de `Batería` creada automáticamente.

Creamos un coche eléctrico y lo asignamos a la variable `my_leaf`. Cuando queremos describir la batería, necesitamos trabajar con la batería del automóvil. atributo:

```
my_leaf.battery.describe_battery()
```

Esta línea le dice a Python que mire la instancia `my_leaf` y encuentre su batería. atributo y llame al método `describe_battery()` que está asociado con la instancia de batería asignada al atributo.

El resultado es idéntico al que vimos anteriormente:

```
2024 Nissan Leaf
```

```
Este coche tiene una batería de 40 kWh.
```

Esto parece mucho trabajo extra, pero ahora podemos describir la batería. con tanto detalle como queramos sin saturar la clase `ElectricCar`. vamos

agregue otro método a Batería que informe la autonomía del automóvil según el tamaño de la batería:

coche de clase:

--recorte--

batería de clase:

--recorte--

```
def get_range(yo):
```

```
    """Imprima una declaración sobre el alcance que proporciona esta batería."""
```

```
    si self.battery_size == 40:
```

```
        rango = 150
```

```
    elif self.battery_size == 65:
```

```
        rango = 225
```

```
    print(f"Este automóvil puede recorrer aproximadamente {autonomía} millas con una carga completa.")
```

```
clase ElectricCar(Coche):
```

--recorte--

```
mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
```

```
imprimir(mi_hoja.get_nombre_descriptivo())
```

```
my_leaf.battery.describe_battery()
```

```
1 mi_hoja.bateria.get_range()
```

El nuevo método `get_range()` realiza un análisis simple. Si la capacidad de la batería es de 40 kWh, `get_range()` establece el alcance en 150 millas, y si la capacidad es de 65 kWh, establece el alcance en 225 millas. Luego informa este valor.

Cuando queramos utilizar este método, nuevamente tenemos que llamarlo a través del atributo 1 de la batería del automóvil .

El resultado nos indica la autonomía del automóvil en función del tamaño de su batería:

2024 Nissan Leaf

Este coche tiene una batería de 40 kWh.

Este coche puede recorrer unos 150 kilómetros con una carga completa.

Modelado de objetos del mundo real

A medida que empieces a modelar cosas más complicadas, como los coches eléctricos, tendrás que lidiar con preguntas interesantes. ¿La autonomía de un coche eléctrico es propiedad de la batería o del coche? Si solo describimos un automóvil, probablemente esté bien mantener la asociación del método `get_range()` con la batería.

clase. Pero si estamos describiendo la línea completa de automóviles de un fabricante, probablemente queramos mover `get_range()` a la clase `ElectricCar` . El método `get_range()` aún verificaría el tamaño de la batería antes de determinar el alcance, pero informaría un alcance específico para el tipo de automóvil con el que está asociado. Alternativamente, podríamos mantener la asociación del método `get_range()` con la batería pero pasarle un parámetro como `car_model`. El método `get_range()` luego informaría un rango basado en el tamaño de la batería y el modelo del automóvil.

Esto lo lleva a un punto interesante en su crecimiento como programador. Cuando luchas con preguntas como éstas, estás pensando en un nivel superior.

nivel lógico en lugar de un nivel centrado en la sintaxis. No estás pensando en Python, sino en cómo representar el mundo real en código. Cuando llegue a este punto, se dará cuenta de que a menudo no existen enfoques correctos o incorrectos para modelar situaciones del mundo real. Algunos enfoques son más eficientes que otros, pero se necesita práctica para encontrar las representaciones más eficientes. Si tu código funciona como deseas, ¡lo estás haciendo bien! No se desanime si descubre que está destrozando sus clases y reescribiéndolas varias veces utilizando diferentes enfoques. En la búsqueda de escribir código preciso y eficiente, todos pasan por este proceso.

INTÉNTALO TÚ MISMO

9-6. Puesto de helados: Un puesto de helados es un tipo específico de restaurante. Escriba una clase llamada `IceCreamStand` que herede de la clase `Restaurante` que escribió en el Ejercicio 9-1 (página 162) o en el Ejercicio 9-4 (página 166). Cualquiera de las versiones de la clase funcionará; simplemente elige el que más te guste. Agregue un atributo llamado `sabores` que almacene una lista de sabores de helado. Escriba un método que muestre estos sabores. Cree una instancia de `IceCreamStand` y llame a este método.

9-7. Administrador: un administrador es un tipo especial de usuario. Escriba una clase llamada `Admin` que herede de la clase `Usuario` que escribió en el Ejercicio 9-3 (página 162) o en el Ejercicio 9-5 (página 167). Agregue un atributo, `privilegios`, que almacene una lista de cadenas como "puede agregar publicación", "puede eliminar publicación", "puede prohibir al usuario", etc. Escriba un método llamado `show_privileges()` que enumere el conjunto de privilegios del administrador. Cree una instancia de `Admin` y llame a su método.

9-8. Privilegios: escriba una clase de privilegios separada. La clase debe tener un atributo, `privilegios`, que almacene una lista de cadenas como se describe en el ejercicio 9-7. Mueva el método `show_privileges()` a esta clase. Cree una instancia de `Privilegios` como un atributo en la clase `Admin`. Cree una nueva instancia de `Admin` y use su método para mostrar sus privilegios.

9-9. Actualización de batería: utilice la versión final de `electric_car.py` de esta sección. Agregue un método a la clase `Batería` llamado `Upgrade_battery()`. Este método debería verificar el tamaño de la batería y establecer la capacidad en 65 si aún no lo está. Cree un automóvil eléctrico con un tamaño de batería predeterminado, llame a `get_range()` una vez y luego llame a `get_range()` una segunda vez después de actualizar la batería. Deberías ver un aumento en la autonomía del coche.

Importar clases

A medida que agrega más funciones a sus clases, sus archivos pueden alargarse, incluso cuando usa la herencia y la composición correctamente. De acuerdo con la filosofía general de Python, querrás mantener tus archivos lo más ordenados posible. Para ayudar, Python te permite almacenar clases en módulos y luego importar las clases que necesitas a tu programa principal.

Importando una clase única

Creemos un módulo que contenga solo la clase `Car`. Esto plantea un problema sutil de nomenclatura: ya tenemos un archivo llamado `car.py` en este capítulo, pero este módulo debería llamarse `car.py` porque contiene código que representa un automóvil.

Resolveremos este problema de nombres almacenando la clase `Car` en un módulo llamado `car.py`, reemplazando el archivo `car.py` que estábamos usando anteriormente. De ahora en adelante, cualquier programa que utilice este módulo necesitará un nombre de archivo más específico, como `my_car.py`. Aquí está `car.py` con solo el código de la clase `Car`:

`car.py` 1 `"""Una clase que se puede utilizar para representar un automóvil."""`

coche de clase:

```
"""Un simple intento de representar un coche."""

def __init__(self, marca, modelo, año): """Inicializar
    atributos para describir un automóvil.""" self.make = make

    self.model = modelo
    self.year = año
    self.odometer_reading = 0

def get_descriptive_name(self): """Devuelve
    un nombre descriptivo perfectamente formateado.""" long_name =
    f'{self.year} {self.make} {self.model}' return long_name.title()

def read_odometer(self):
    """Imprime un estado de cuenta que muestre el kilometraje del
    auto.""" print(f"Este auto tiene {self.odometer_reading} millas.")

def update_odometer(self, kilometraje):

    Establezca la lectura del odómetro al valor indicado.
    Rechace el cambio si intenta hacer retroceder el odómetro.

    if kilometraje >= self.odometer_reading:
        self.odometer_reading = kilometraje else:

    print("¡No puedes retroceder un odómetro!")

def increment_odometer(self, millas): """Agregue
    la cantidad dada a la lectura del odómetro.""" self.odometer_reading
    += millas
```

Incluimos una cadena de documentación a nivel de módulo que describe brevemente el contenido de este módulo 1. Debe escribir una cadena de documentación para cada módulo que cree.

Ahora creamos un archivo separado llamado `my_car.py`. Este archivo importará la clase `Car` y luego creará una instancia a partir de esa clase:

`my_car.py` 1 de importación de automóviles

```
mi_coche_nuevo = Coche('audi', 'a4', 2024)
print(mi_coche_nuevo.get_descriptive_name())
```



```
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

La declaración de importación 1 le dice a Python que abra el módulo car e importe la clase Car. Ahora podemos usar la clase Car como si estuviera definida en este archivo. El resultado es el mismo que vimos antes:

```
2024 Audi A4
Este auto tiene 23 millas.
```

Importar clases es una forma eficaz de programar. Imagínese cuánto duraría este archivo de programa si se incluyera toda la clase Car. Cuando, en cambio, mueve la clase a un módulo e importa el módulo, aún obtiene la misma funcionalidad, pero mantiene el archivo principal del programa limpio y fácil de leer. También almacena la mayor parte de la lógica en archivos separados; Una vez que sus clases funcionen como usted desea, puede dejar esos archivos en paz y concentrarse en la lógica de nivel superior de su programa principal.

Almacenamiento de varias clases en un módulo

Puedes almacenar tantas clases como necesites en un solo módulo, aunque cada clase de un módulo debe estar relacionada de alguna manera. Las clases Battery y ElectricCar ayudan a representar automóviles, así que agreguémoslas al módulo car.py.

car.py """Un conjunto de clases utilizadas para representar automóviles eléctricos y de gasolina."""

coche de clase:

```
--recorte--
```

batería de clase:

```
"""Un sencillo intento de modelar una batería para un coche eléctrico."""
```

```
def __init__(self, tamaño_batería=40):
    """Inicializar los atributos de la batería."""
    self.battery_size = tamaño_batería
```

```
def describir_batería(yo):
    """Imprima una declaración que describa el tamaño de la batería."""
    print(f"Este auto tiene una batería de {self.battery_size}-kWh.")
```

```
def get_range(yo):
    """Imprima una declaración sobre el alcance que proporciona esta batería."""
    si self.battery_size == 40:
        rango = 150
    elif self.battery_size == 65:
        rango = 225
```

```
print(f"Este automóvil puede recorrer aproximadamente {autonomía} millas con una carga completa.")
```

clase ElectricCar(Coche):

```
"""Modelos aspectos de un automóvil, específicos de los vehículos eléctricos."""
```

```
def __init__(propio, marca, modelo, año):
```

```

Inicializa los atributos de la clase padre.
Luego inicialice los atributos específicos de un automóvil eléctrico.

super().__init__(marca, modelo, año)
self.battery = Bateria()

```

Ahora podemos crear un nuevo archivo llamado `my_electric_car.py`, importar el `ElectricCar` clase y hacer un coche eléctrico:

```

mi_electrico
_car.py
de importación de automóviles ElectricCar

mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
imprimir(mi_hoja.get_nombre_descriptivo())
my_leaf.battery.describe_battery()
mi_hoja.bateria.get_range()

```

Esto tiene el mismo resultado que vimos antes, aunque la mayor parte de la lógica está oculta en un módulo:

2024 Nissan Leaf

Este coche tiene una batería de 40 kWh.

Este coche puede recorrer unos 150 kilómetros con una carga completa.

Importar varias clases desde un módulo

Puede importar tantas clases como necesite en un archivo de programa. Si queremos hacer un coche normal y un coche eléctrico en el mismo archivo, necesitamos importar ambas clases, `Car` y `ElectricCar`:

`my_cars.py` 1 de importación de automóviles `Auto`, `ElectricCar`

```

2 mi_mustang = Coche('ford', 'mustang', 2024)
  print(mi_mustang.get_descriptive_name())
3 mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
  imprimir(mi_hoja.get_nombre_descriptivo())

```

Importas varias clases desde un módulo separando cada clase con una coma 1. Una vez que hayas importado las clases necesarias, podrás crear tantas instancias de cada clase como necesites.

En este ejemplo fabricamos un Ford Mustang 2 de gasolina y luego un Nissan Leaf 3 eléctrico:

Ford Mustang 2024
2024 Nissan Leaf

Importar un módulo completo

También puede importar un módulo completo y luego acceder a las clases que necesita usando la notación de puntos. Este enfoque es simple y da como resultado un código fácil de leer.

Debido a que cada llamada que crea una instancia de una clase incluye el nombre del módulo, no tendrá conflictos de nombres con ningún nombre utilizado en el archivo actual.

Así es como se ve importar todo el módulo del automóvil y luego crear un automóvil normal y un automóvil eléctrico:

```
my_cars.py 1 auto importado
```

```
2 mi_mustang = coche.Coche('ford', 'mustang', 2024)
  print(mi_mustang.get_descriptive_name())

3 mi_hoja = coche.CocheEléctrico('nissan', 'hoja', 2024)
  imprimir(mi_hoja.get_nombre_descriptivo())
```

Primero importamos todo el módulo 1 del automóvil . Luego accedemos a las clases que necesitamos a través de la sintaxis `module_name.ClassName` . Volvemos a crear un Ford Mustang 2 y un Nissan Leaf 3.

Importar todas las clases desde un módulo

Puede importar todas las clases desde un módulo usando la siguiente sintaxis:

```
desde la importación de nombre_módulo *
```

Este método no se recomienda por dos razones. Primero, es útil poder leer las declaraciones de importación en la parte superior de un archivo y tener una idea clara de qué clases utiliza un programa. Con este enfoque, no está claro qué clases estás usando del módulo. Este enfoque también puede generar confusión con los nombres del archivo. Si importa accidentalmente una clase con el mismo nombre que otra cosa en su archivo de programa, puede crear errores que son difíciles de diagnosticar. Muestro esto aquí porque, aunque no es un enfoque recomendado, es probable que lo veas en el código de otras personas en algún momento.

Si necesita importar muchas clases desde un módulo, es mejor importar el módulo completo y usar la sintaxis `module_name.ClassName` . No verá todas las clases utilizadas en la parte superior del archivo, pero verá claramente dónde se usa el módulo en el programa. También evitará los posibles conflictos de nombres que pueden surgir al importar cada clase en un módulo.

Importar un módulo a un módulo

A veces querrás distribuir tus clases en varios módulos para evitar que un archivo crezca demasiado y evitar almacenar clases no relacionadas en el mismo módulo. Cuando almacena sus clases en varios módulos, puede encontrar que una clase en un módulo depende de una clase en otro módulo.

Cuando esto sucede, puede importar la clase requerida al primer módulo.

Por ejemplo, almacenemos la clase `Car` en un módulo y la clase `ElectricCar` y clases de batería en un módulo separado. Crearemos un nuevo módulo llamado `electric_car.py`, que reemplazará el archivo `electric_car.py` que creamos anteriormente, y copiaremos solo las clases `Battery` y `ElectricCar` en este archivo:

```
electric_car.py """Un conjunto de clases que se pueden utilizar para representar coches eléctricos."""
```

```
de coche de importación
```

batería de clase:

--recorte--

clase ElectricCar(Coche):

--recorte--

La clase ElectricCar necesita acceso a su clase principal Car, por lo que importamos Car directamente al módulo. Si olvidamos esta línea, Python generará un error cuando intentemos importar el módulo electric_car. También necesitamos actualizar el módulo Car para que contenga solo la clase Car :

car.py """Una clase que se puede utilizar para representar un automóvil."""

coche de clase:

--recorte--

Ahora podemos importar desde cada módulo por separado y crear cualquier tipo de coche que necesitemos:

my_cars.py de importación de automóviles
de electric_car importar ElectricCar

```
my_mustang = Coche('ford', 'mustang', 2024)
print(mi_mustang.get_descriptive_name())
```

```
mi_hoja = Coche Eléctrico('nissan', 'hoja', 2024)
imprimir(mi_hoja.get_nombre_descriptivo())
```

Importamos Car desde su módulo y ElectricCar desde su módulo. Luego creamos un coche normal y un coche eléctrico. Ambos coches están creados correctamente:

```
Ford Mustang 2024
2024 Nissan Leaf
```

Usando alias

Como vio en el Capítulo 8, los alias pueden ser muy útiles cuando se utilizan módulos para organizar el código de sus proyectos. También puede utilizar alias al importar clases.

Como ejemplo, considere un programa en el que desea fabricar varios coches eléctricos. Puede resultar tedioso escribir (y leer) ElectricCar una y otra vez. Puede darle a ElectricCar un alias en la declaración de importación:

de electric_car importar ElectricCar como CE

Ahora puedes utilizar este alias siempre que quieras fabricar un coche eléctrico:

```
mi_hoja = EC('nissan', 'hoja', 2024)
```

También puedes darle un alias a un módulo. A continuación se explica cómo importar el archivo completo. módulo `electric_car` usando un alias:

```
importar electric_car como ec
```

Ahora puedes usar este alias de módulo con el nombre de clase completo:

```
mi_hoja = ec.ElectricCar('nissan', 'hoja', 2024)
```

Encontrar su propio flujo de trabajo

Como puede ver, Python le ofrece muchas opciones sobre cómo estructurar el código en un proyecto grande. Es importante conocer todas estas posibilidades para que pueda determinar las mejores formas de organizar sus proyectos y comprender los proyectos de otras personas.

Cuando estés empezando, mantén la estructura de tu código simple. Intente hacer todo en un solo archivo y mueva sus clases a módulos separados una vez que todo esté funcionando. Si le gusta cómo interactúan los módulos y los archivos, intente almacenar sus clases en módulos cuando inicie un proyecto. Encuentre un enfoque que le permita escribir código que funcione y continúe desde allí.

INTÉNTALO TÚ MISMO

9-10. Restaurante importado: utilizando su última clase de restaurante, guárdela en un módulo. Cree un archivo separado que importe `Restaurante`. Cree una instancia de `Restaurant` y llame a uno de los métodos de `Restaurant` para mostrar que la declaración de importación está funcionando correctamente.

9-11. Administrador importado: comience con su trabajo del Ejercicio 9-8 (página 173). Almacene las clases `Usuario`, `Privilegios` y `Administrador` en un módulo. Cree un archivo separado, cree una instancia de `administrador` y llame a `show_privileges()` para mostrar que todo está funcionando correctamente.

9-12. Múltiples módulos: almacene la clase `Usuario` en un módulo y almacene las clases `Privilegios` y `Administrador` en un módulo separado. En un archivo separado, cree una instancia de `administrador` y llame a `show_privileges()` para mostrar que todo sigue funcionando correctamente.

La biblioteca estándar de Python

La biblioteca estándar de Python es un conjunto de módulos incluidos con cada instalación de Python. Ahora que tiene un conocimiento básico de cómo funcionan las funciones y clases, puede comenzar a utilizar módulos como estos que han escrito otros programadores. Puede utilizar cualquier función o clase en la biblioteca estándar incluyendo una declaración de importación simple en la parte superior de su archivo. Veamos un módulo, `aleatorio`, que puede resultar útil para modelar muchas situaciones del mundo real.

Una función interesante del módulo aleatorio es `randint()`. Esta función toma dos argumentos enteros y devuelve un entero seleccionado aleatoriamente entre (e incluyendo) esos números.

A continuación se explica cómo generar un número aleatorio entre 1 y 6:

```
>>> de randint de importación aleatoria
>>> randint(1, 6)
3
```

Otra función útil es la elección(). Esta función toma una lista o tupla y devuelve un elemento elegido aleatoriamente:

```
>>> de elección de importación aleatoria
>>> jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
>>> first_up = elección(jugadores)
>>> primero_arriba
'florencia'
```

El módulo aleatorio no debe usarse cuando se crean elementos relacionados con la seguridad. aplicaciones, pero funciona bien para muchos proyectos divertidos e interesantes.

NOTA También puede descargar módulos de fuentes externas. Verá varios de estos ejemplos en la Parte II, donde necesitaremos módulos externos para completar cada proyecto.

INTÉNTALO TÚ MISMO

9-13. Dados: crea una clase `Die` con un atributo llamado `lados`, que tiene un valor predeterminado de 6. Escribe un método llamado `roll_die()` que imprima un número aleatorio entre 1 y el número de lados que tiene el dado. Haz un dado de 6 caras y tíralo 10 veces.

Haz un dado de 10 caras y otro de 20 caras. Lanza cada dado 10 veces.

9-14. Lotería: Haz una lista o tupla que contenga una serie de 10 números y 5 letras. Seleccione aleatoriamente 4 números o letras de la lista e imprima un mensaje que diga que cualquier boleto que coincida con estos 4 números o letras gana un premio.

9-15. Análisis de lotería: puede utilizar un bucle para ver qué tan difícil podría ser ganar el tipo de lotería que acaba de modelar. Haz una lista o tupla llamada `my_ticket`. Escribe un bucle que siga extrayendo números hasta que gane tu boleto. Imprima un mensaje informando cuántas veces tuvo que ejecutarse el ciclo para obtener un boleto ganador.

9-16. Módulo Python de la semana: un recurso excelente para explorar la biblioteca estándar de Python es un sitio llamado Módulo Python de la semana. Vaya a <https://pymotw.com> y mire la tabla de contenido. Encuentre un módulo que le parezca interesante y lea sobre él, tal vez comenzando con el módulo aleatorio .

Clases de estilismo

Vale la pena aclarar algunas cuestiones de estilo relacionadas con las clases, especialmente a medida que sus programas se vuelven más complicados.

Los nombres de las clases deben escribirse en CamelCase. Para hacer esto, escriba en mayúscula la primera letra de cada palabra en el nombre y no use guiones bajos. Los nombres de instancias y módulos deben escribirse en minúsculas, con guiones bajos entre las palabras.

Cada clase debe tener una cadena de documentación inmediatamente después de la definición de clase. La cadena de documentación debe ser una breve descripción de lo que hace la clase y debes seguir las mismas convenciones de formato que utilizaste para escribir cadenas de documentación en funciones. Cada módulo también debe tener una cadena de documentación que describa para qué se pueden usar las clases de un módulo.

Puedes usar líneas en blanco para organizar el código, pero no las uses en exceso. Dentro de una clase puedes usar una línea en blanco entre métodos y dentro de un módulo puedes usar dos líneas en blanco para separar clases.

Si necesita importar un módulo de la biblioteca estándar y un módulo que usted escribió, coloque primero la declaración de importación para el módulo de la biblioteca estándar. Luego agregue una línea en blanco y la declaración de importación para el módulo que escribió. En programas con múltiples declaraciones de importación, esta convención facilita ver de dónde provienen los diferentes módulos utilizados en el programa.

Resumen

En este capítulo, aprendió a escribir sus propias clases. Aprendiste cómo almacenar información en una clase usando atributos y cómo escribir métodos que le den a tus clases el comportamiento que necesitan. Aprendiste a escribir métodos `__init__()` que crean instancias de tus clases con exactamente los atributos que deseas. Viste cómo modificar los atributos de una instancia directamente y mediante métodos. Aprendió que la herencia puede simplificar la creación de clases relacionadas entre sí y aprendió a usar instancias de una clase como atributos en otra clase para mantener cada clase simple.

Viste cómo almacenar clases en módulos e importar las clases que necesitas en los archivos donde se usarán puede mantener tus proyectos organizados. Comenzó a aprender sobre la biblioteca estándar de Python y vio un ejemplo basado en el módulo aleatorio. Finalmente, aprendiste a diseñar tus clases usando las convenciones de Python.

En el Capítulo 10, aprenderá a trabajar con archivos para poder guardar el trabajo que ha realizado en un programa y el trabajo que ha permitido que realicen los usuarios. También aprenderá sobre las excepciones, una clase especial de Python diseñada para ayudarlo a responder a los errores cuando surjan.

10

ARCHIVOS Y EXCEPCIONES



Ahora que domina las habilidades básicas que necesita para escribir programas organizados que sean fáciles de usar, es hora de pensar en hacer que sus programas sean aún más relevantes y utilizables. En este capítulo, aprenderá a trabajar con archivos para que sus programas puedan analizar rápidamente muchos datos.

Aprenderá a manejar los errores para que sus programas no fallen cuando se encuentren con situaciones inesperadas. Aprenderá sobre las excepciones, que son objetos especiales que Python crea para gestionar los errores que surgen mientras se ejecuta un programa. También aprenderá sobre el módulo json, que le permite guardar datos de usuario para que no se pierdan cuando su programa deje de ejecutarse.

Aprender a trabajar con archivos y guardar datos hará que sus programas sean más fáciles de usar para las personas. Los usuarios podrán elegir qué datos ingresar y cuándo ingresarlos. Las personas podrán ejecutar su programa, trabajar un poco y luego cerrar el programa y continuar donde lo dejaron. Aprender a manejar excepciones le ayudará a afrontar situaciones en las que los archivos no existen y a solucionar otros problemas que pueden provocar que sus programas colapsen. Esto hará que sus programas sean más sólidos cuando encuentren datos incorrectos, ya sea que provengan de

errores inocentes o de intentos maliciosos de romper sus programas. Con las habilidades que aprenderá en este capítulo, hará que sus programas sean más aplicables, utilizables y estables.

Lectura de un archivo

Una cantidad increíble de datos está disponible en archivos de texto. Los archivos de texto pueden contener datos meteorológicos, datos de tráfico, datos socioeconómicos, obras literarias y más. La lectura de un archivo es particularmente útil en aplicaciones de análisis de datos, pero también es aplicable a cualquier situación en la que desee analizar o modificar información almacenada en un archivo. Por ejemplo, puede escribir un programa que lea el contenido de un archivo de texto y reescriba el archivo con un formato que permita que un navegador lo muestre.

Cuando desee trabajar con la información de un archivo de texto, el primer paso es leer el archivo en la memoria. Luego puede revisar todo el contenido del archivo a la vez o revisar el contenido línea por línea.

Leer el contenido de un archivo

Para comenzar, necesitamos un archivo con algunas líneas de texto. Comencemos con un archivo que contiene pi con 30 decimales, con 10 decimales por línea:

```
pi_dígitos.txt 3.1415926535
                8979323846
                2643383279
```

Para probar los siguientes ejemplos usted mismo, puede ingresar estas líneas en un editor y guardar el archivo como pi_digits.txt, o puede descargar el archivo desde los recursos del libro a través de https://ehmatthes.github.io/pcc_3e. Guarde el archivo en el mismo directorio donde almacenará los programas de este capítulo.

Aquí hay un programa que abre este archivo, lo lee e imprime el contenido del archivo a la pantalla:

file_reader.py de ruta de importación pathlib

```
1 ruta = Ruta('pi_digits.txt')
2 contenidos = ruta.read_text()
  imprimir (contenido)
```

Para trabajar con el contenido de un archivo, debemos indicarle a Python la ruta al archivo. Una ruta es la ubicación exacta de un archivo o carpeta en un sistema. Python proporciona un módulo llamado pathlib que facilita el trabajo con archivos y directorios, sin importar con qué sistema operativo esté trabajando usted o los usuarios de su programa. Un módulo que proporciona una funcionalidad específica como esta a menudo se denomina biblioteca, de ahí el nombre pathlib.

Comenzamos importando la clase Path desde pathlib. Hay muchas cosas que puedes hacer con un objeto Ruta que apunta a un archivo. Por ejemplo, puede comprobar que el archivo existe antes de trabajar con él, leer el contenido del archivo o escribir datos nuevos en el archivo. Aquí, construimos un objeto Ruta que representa el archivo pi_digits.txt, que asignamos a la ruta variable 1. Dado que este archivo se guarda en el mismo

directorio que el archivo .py que estamos escribiendo, el nombre del archivo es todo lo que Path necesita para acceder al archivo.

NOTA S Code busca archivos en la carpeta que se abrió más recientemente. Si está utilizando VS Code, comience abriendo la carpeta donde almacena los programas de este capítulo. Por ejemplo, si está guardando los archivos de su programa en una carpeta llamada capítulo_10, presione CTRL-O (⌘-O en macOS) y abra esa carpeta.

Una vez que tenemos un objeto Path que representa pi_digits.txt, usamos read_text() método para leer todo el contenido del archivo 2. El contenido del archivo se devuelve como una sola cadena, que asignamos al contenido de la variable. Cuando imprimimos el valor del contenido, vemos el contenido completo del archivo de texto:

```
3.1415926535
8979323846
2643383279
```

La única diferencia entre este resultado y el archivo original es la línea en blanco adicional al final del resultado. La línea en blanco aparece porque read_text() devuelve una cadena vacía cuando llega al final del archivo; esta cadena vacía aparece como una línea en blanco.

Podemos eliminar la línea en blanco adicional usando rstrip() en el contenido cadena:

```
desde pathlib importar ruta

ruta = Ruta('pi_digits.txt')
contenido = ruta.read_text()
contenido = contenido.rstrip()
imprimir (contenido)
```

Recuerde del Capítulo 2 que el método rstrip() de Python elimina, o elimina, cualquier carácter de espacio en blanco del lado derecho de una cadena. Ahora el resultado coincide exactamente con el contenido del archivo original:

```
3.1415926535
8979323846
2643383279
```

Podemos eliminar el carácter de nueva línea final cuando leemos el contenido del archivo, aplicando el método rstrip() inmediatamente después de llamar a read_text():

```
contenido = ruta.read_text().rstrip()
```

Esta línea le dice a Python que llame al método read_text() en el archivo con el que estamos trabajando. Luego aplica el método rstrip() a la cadena que lee _text() devuelve. Luego, la cadena limpia se asigna al contenido de la variable. Este enfoque se llama encadenamiento de métodos y verá que se utiliza con frecuencia en programación.

Rutas de archivos relativas y absolutas

Cuando pasas un nombre de archivo simple como `pi_digits.txt` a `Path`, Python busca en el directorio donde se está ejecutando el archivo que se está ejecutando actualmente (es decir, tu `.py` archivo de programa) se almacena.

A veces, dependiendo de cómo organices tu trabajo, el archivo que desees para abrir no estará en el mismo directorio que su archivo de programa. Por ejemplo, puedes almacenar los archivos de tu programa en una carpeta llamada `python_work`; Dentro de `python_work`, es posible que tenga otra carpeta llamada `text_files` para distinguir los archivos de su programa de los archivos de texto que están manipulando. Aunque `text_files` está en `python_work`, simplemente pasando `Path` el nombre de un archivo en `text_files` no funcionará, porque Python sólo buscará en `python_work` y se detendrá allí; No continuará ni buscará en archivos de texto. Para que Python abra archivos desde un directorio distinto de aquel donde está almacenado el archivo de su programa, debe proporcionar la ruta correcta.

Hay dos formas principales de especificar rutas en programación. Una ruta de archivo relativa le dice a Python que busque una ubicación determinada relativa al directorio donde está almacenado el archivo del programa que se está ejecutando actualmente. Dado que `text_files` está dentro de `python_work`, necesitamos crear una ruta que comience con el directorio `text_files` y termine con el nombre del archivo. A continuación se explica cómo construir este camino:

```
ruta = Ruta('archivos_de_texto/nombredearchivo.txt')
```

También puedes decirle a Python exactamente dónde está el archivo en tu computadora, independientemente de dónde esté almacenado el programa que se está ejecutando. Esto se llama ruta de archivo absoluta. Puede utilizar una ruta absoluta si una ruta relativa no funciona. Por ejemplo, si ha colocado `archivos_de_texto` en alguna carpeta que no sea `python_work`, simplemente pasando `Ruta` la ruta `'archivos_de_texto/ nombre_archivo.txt'` no funcionará porque Python solo buscará esa ubicación dentro de `python_work`. Deberá escribir una ruta absoluta para aclarar dónde desea que busque Python.

Las rutas absolutas suelen ser más largas que las relativas porque comienzan en la carpeta raíz de su sistema:

```
ruta = Ruta('/home/eric/data_files/text_files/filename.txt')
```

Usando rutas absolutas, puede leer archivos desde cualquier ubicación de su sistema. Por ahora, es más fácil almacenar archivos en el mismo directorio que los archivos de su programa, o en una carpeta como `text_files` dentro del directorio que almacena sus archivos de programa.

NOTA Los sistemas Windows utilizan una barra invertida (`\`) en lugar de una barra diagonal (`/`) cuando muestran las rutas de los archivos, pero usted debe utilizar barras diagonales en su código, incluso en Windows. La biblioteca `pathlib` utilizará automáticamente la representación correcta de la ruta cuando interactúe con su sistema o el sistema de cualquier usuario.

Accediendo a las líneas de un archivo

Cuando trabaja con un archivo, a menudo querrá examinar cada línea del archivo. Es posible que esté buscando cierta información en el archivo, o

es posible que desee modificar el texto del archivo de alguna manera. Por ejemplo, es posible que desee leer un archivo de datos meteorológicos y trabajar con cualquier línea que incluya la palabra soleado en la descripción del tiempo de ese día. En un informe de noticias, puede buscar cualquier línea con la etiqueta <título> y reescribir esa línea con un tipo de formato específico.

Puedes usar el método `splitlines()` para convertir una cadena larga en un conjunto de líneas y luego usar un bucle `for` para examinar cada línea de un archivo, una a la vez:

`file_reader.py` de ruta de importación `pathlib`

```
ruta = Ruta('pi_digits.txt')
1 contenido = ruta.read_text()

2 líneas = contenido.splitlines()
para línea en líneas:
    imprimir (línea)
```

Comenzamos leyendo todo el contenido del archivo, como lo hicimos anteriormente 1. Si planea trabajar con las líneas individuales de un archivo, no necesita eliminar ningún espacio en blanco al leer el archivo. Las líneas divididas() El método devuelve una lista de todas las líneas del archivo y asignamos esta lista a la variable líneas 2. Luego recorremos estas líneas e imprimimos cada una:

```
3.1415926535
8979323846
2643383279
```

Como no hemos modificado ninguna de las líneas, el resultado coincide con el archivo de texto original exactamente.

Trabajar con el contenido de un archivo

Después de haber leído el contenido de un archivo en la memoria, puedes hacer lo que quieras con esos datos, así que exploremos brevemente los dígitos de pi. Primero, intentaremos crear una única cadena que contenga todos los dígitos del archivo sin espacios en blanco:

`pi_string.py` de ruta de importación `pathlib`

```
ruta = Ruta('pi_digits.txt')
contenido = ruta.read_text()

líneas = contenido.líneas divididas()
pi_string = ''
para línea en líneas:
    pi_string += línea

imprimir (cadena_pi)
imprimir(len(pi_string))
```

Comenzamos leyendo el archivo y almacenando cada línea de dígitos en una lista, tal como lo hicimos en el ejemplo anterior. Luego creamos una variable, `pi_string`,

para contener los dígitos de pi. Escribimos un bucle que agrega cada línea de dígitos a pi_string 1. Imprimimos esta cadena y también mostramos cuánto mide la cadena:

```
3.1415926535 8979323846 2643383279
36
```

La variable pi_string contiene el espacio en blanco que estaba en el lado izquierdo de los dígitos en cada línea, pero podemos deshacernos de eso usando lstrip() en cada línea:

```
--recorte--
para línea en líneas:
    pi_string += línea.lstrip()

imprimir (cadena_pi)
imprimir(len(pi_string))
```

Ahora tenemos una cadena que contiene pi con 30 decimales. La cadena tiene 32 caracteres porque también incluye el 3 inicial y un punto decimal:

```
3.141592653589793238462643383279
32
```

NOTA Cuando Python lee un archivo de texto, interpreta todo el texto del archivo como una cadena. Si lees un número y quieres trabajar con ese valor en un contexto numérico, tendrás que convertirlo a un número entero usando la función int() o a un valor flotante usando float() función.

Archivos grandes: un millón de dígitos

Hasta ahora, nos hemos centrado en analizar un archivo de texto que contiene sólo tres líneas, pero el código de estos ejemplos funcionaría igual de bien en archivos mucho más grandes. Si comenzamos con un archivo de texto que contiene pi con 1.000.000 de decimales, en lugar de solo 30, podemos crear una única cadena que contenga todos estos dígitos. No necesitamos cambiar nuestro programa en absoluto, excepto pasarle un archivo diferente. También imprimiremos solo los primeros 50 decimales, para que no tengamos que ver pasar un millón de dígitos en la terminal:

pi_string.py de ruta de importación pathlib

```
ruta = Ruta('pi_million_digits.txt')
contenido = ruta.read_text()

líneas = contenido.líneas divididas()
pi_string =
para línea en líneas:
    pi_string += línea.lstrip()

imprimir(f'{pi_string[:52]}...')
imprimir(len(pi_string))
```

El resultado muestra que efectivamente tenemos una cadena que contiene pi con 1.000.000 de decimales:

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

Python no tiene un límite inherente a la cantidad de datos con los que puedes trabajar; tú puede trabajar con tantos datos como la memoria de su sistema pueda manejar.

NOTA Para ejecutar este programa (y muchos de los ejemplos que siguen), necesitará descargar los recursos disponibles en https://ehmatthes.github.io/pcc_3e.

¿Tu cumpleaños está contenido en Pi?

Siempre he tenido curiosidad por saber si mi cumpleaños aparece en algún lugar de los dígitos de pi. Usemos el programa que acabamos de escribir para averiguar si el cumpleaños de alguien aparece en algún lugar del primer millón de dígitos de pi. Podemos hacer esto expresando cada cumpleaños como una cadena de dígitos y viendo si esa cadena aparece en algún lugar de pi_string:

```
pi_cumpleaños.py --snip--
para línea en líneas:
    pi_string += línea.strip()

cumpleaños = input("Ingrese su fecha de cumpleaños, en el formato mmddyy: ")
si cumpleaños en pi_string:
    print("¡Tu cumpleaños aparece en el primer millón de dígitos de pi!")
demás:
    print("Tu cumpleaños no aparece en el primer millón de dígitos de pi.")
```

Primero solicitamos el cumpleaños del usuario y luego verificamos si esa cadena está en pi_string. Vamos a intentarlo:

```
Ingresa tu fecha de nacimiento, en el formato mmddyy: 120372
¡Tu cumpleaños aparece en el primer millón de dígitos de pi!
```

¡Mi cumpleaños aparece en los dígitos de pi! Una vez que hayas leído de un archivo, puede analizar su contenido de casi cualquier forma que pueda imaginar.

INTÉNTALO TÚ MISMO

10-1. Aprender Python: abra un archivo en blanco en su editor de texto y escriba algunas líneas que resuma lo que ha aprendido sobre Python hasta ahora. Comience cada línea con la frase En Python puedes. . . . Guarde el archivo como learning_python.txt en el mismo directorio que sus ejercicios de este capítulo. Escriba un programa que lea el archivo e imprima lo que escribió dos veces: imprima el contenido una vez leyendo el archivo completo y otra vez almacenando las líneas en una lista y luego recorriendo cada línea.

(continuado)

10-2. Aprendizaje de C: puede utilizar el método `reemplazar()` para reemplazar cualquier palabra en una cadena con una palabra diferente. Aquí hay un ejemplo rápido que muestra cómo reemplazar 'perro' por 'gato' en una oración:

```
>>> mensaje = "Me gustan mucho los perros."
>>> mensaje.replace('perro', 'gato')
"Me gustan mucho los gatos."
```

Lea cada línea del archivo que acaba de crear, `learning_python.txt`, y reemplace la palabra Python con el nombre de otro idioma, como C. Imprima cada línea modificada en la pantalla.

10-3. Código más simple: el programa `file_reader.py` en esta sección usa una variable temporal, `líneas`, para mostrar cómo funciona `splitlines()`. Puede omitir la variable temporal y recorrer directamente la lista que devuelve `splitlines()`:

para la línea en `contenidos.splitlines()`:

Elimine la variable temporal de cada uno de los programas de esta sección, para hacerlos más concisos.

Escribir en un archivo

Una de las formas más sencillas de guardar datos es escribirlos en un archivo. Cuando escribe texto en un archivo, la salida seguirá estando disponible después de cerrar la terminal que contiene la salida de su programa. Puede examinar el resultado después de que un programa termine de ejecutarse y también puede compartir los archivos de resultado con otras personas. También puedes escribir programas que lean el texto en la memoria y vuelvan a trabajar con él más tarde.

Escribir una sola línea

Una vez que haya definido una ruta, puede escribir en un archivo usando `write_text()` método. Para ver cómo funciona esto, escribamos un mensaje simple y guárdelo en un archivo en lugar de imprimirlo en la pantalla:

```
escribir desde pathlib importar ruta
_mensaje.py
ruta = Ruta('programación.txt')
path.write_text("Me encanta programar.")
```

El método `write_text()` toma un único argumento: la cadena que desea escribir en el archivo. Este programa no tiene salida de terminal, pero si abre el archivo `programming.txt`, verá una línea:

```
programación.txt Me encanta programar.
```

Este archivo se comporta como cualquier otro archivo en su computadora. Puede abrirlo, escribir texto nuevo en él, copiarlo, pegarlo, etc.

NOTA Python solo puede escribir cadenas en un archivo de texto. Si desea almacenar datos numéricos en un archivo de texto, primero deberá convertir los datos al formato de cadena usando `str()` función.

Escribir varias líneas

El método `write_text()` hace algunas cosas detrás de escena. Si el archivo al que apunta esa ruta no existe, crea ese archivo. Además, después de escribir la cadena en el archivo, se asegura de que el archivo esté cerrado correctamente. Los archivos que no se cierran correctamente pueden provocar que se pierdan datos o se dañen.

Para escribir más de una línea en un archivo, necesita crear una cadena que contenga todo el contenido del archivo y luego llamar a `write_text()` con esa cadena. Escribamos varias líneas en el archivo `programming.txt` :

```
desde pathlib importar ruta

contenidos = "Me encanta programar.\n"
contenidos += "Me encanta crear juegos nuevos.\n"
contenidos += "También me encanta trabajar con datos.\n"

ruta = Ruta('programación.txt')
ruta.write_text(contenido)
```

Definimos una variable llamada `contenidos` que contendrá todo el contenido del archivo. En la siguiente línea, usamos el operador `+=` para agregar a esta cadena. Puedes hacer esto tantas veces como necesites para crear cadenas de cualquier longitud. En este caso incluimos caracteres de nueva línea al final de cada línea, para asegurarnos de que cada declaración aparezca en su propia línea.

Si ejecuta esto y luego abre `programming.txt`, verá cada una de estas líneas en el archivo de texto:

```
Me encanta la programación.
Me encanta crear nuevos juegos.
También me encanta trabajar con datos.
```

También puede utilizar espacios, caracteres de tabulación y líneas en blanco para formatear su salida, tal como lo ha estado haciendo con la salida basada en terminal. No hay límite para la longitud de sus cadenas, y esta es la cantidad de documentos generados por computadora que se crean.

NOTA tenga cuidado al llamar a `write_text()` en un objeto de ruta. Si el archivo ya existe, `write_text()` borrará el contenido actual del archivo y escribirá contenido nuevo en el archivo. Más adelante en este capítulo, aprenderá a verificar si un archivo existe usando `pathlib`.

INTÉNTALO TÚ MISMO

10-4. Invitado: escriba un programa que solicite al usuario su nombre. Cuando respondan, escriba su nombre en un archivo llamado `guest.txt`.

10-5. Libro de visitas: escriba un bucle `while` que solicite a los usuarios su nombre. Recopile todos los nombres ingresados y luego escríbalos en un archivo llamado `guest_book.txt`. Asegúrese de que cada entrada aparezca en una nueva línea del archivo.

Excepciones

Python utiliza objetos especiales llamados excepciones para gestionar los errores que surgen durante la ejecución de un programa. Cada vez que ocurre un error que hace que Python no esté seguro de qué hacer a continuación, crea un objeto de excepción. Si escribe código que maneje la excepción, el programa continuará ejecutándose. Si no maneja la excepción, el programa se detendrá y mostrará un rastreo, que incluye un informe de la excepción que se generó.

Las excepciones se manejan con bloques `try-except`. Un bloque `try-except` pregunta Python para hacer algo, pero también le dice a Python qué hacer si se genera una excepción. Cuando utiliza bloques `try-except`, sus programas continuarán ejecutándose incluso si las cosas empiezan a salir mal. En lugar de rastreos, que pueden resultar confusos de leer para los usuarios, los usuarios verán mensajes de error amigables que usted haya escrito.

Manejo de la excepción `ZeroDivisionError`

Veamos un error simple que hace que Python genere una excepción. Probablemente sepas que es imposible dividir un número entre cero, pero pidámosle a Python que lo haga de todos modos:

división
_calculadora.py

```
imprimir(5/0)
```

Python no puede hacer esto, por lo que obtenemos un rastreo:

Rastreo (llamadas recientes más última):

```
Archivo "division_calculator.py", línea 1, en <módulo>
    imprimir(5/0)
```

1 `ZeroDivisionError`: división por cero

El error informado en el rastreo, `ZeroDivisionError`, es un objeto de excepción 1. Python crea este tipo de objeto en respuesta a una situación en la que no puede hacer lo que le pedimos. Cuando esto sucede, Python detiene el programa y nos dice el tipo de excepción que se generó. Podemos utilizar esta información para modificar nuestro programa. Le diremos a Python qué hacer cuando ocurra este tipo de excepción; De esa manera, si vuelve a suceder, estaremos preparados.

Usando bloques try-except

Cuando crea que puede ocurrir un error, puede escribir un bloque try-except para manejar la excepción que podría surgir. Le dice a Python que intente ejecutar algún código y le dice qué hacer si el código genera un tipo particular de excepción.

Así es como se ve un bloque try-except para manejar la excepción ZeroDivisionError :

```
intentar:
    imprimir(5/0)
excepto ZeroDivisionError:
    print("¡No se puede dividir por cero!")
```

Ponemos print(5/0), la línea que causó el error, dentro de un bloque try . Si el código de un bloque try funciona, Python omite el bloque except . Si el código en el bloque try causa un error, Python busca un bloque excepto cuyo error coincida con el que se generó y ejecuta el código en ese bloque.

En este ejemplo, el código en el bloque try produce un ZeroDivisionError, por lo que Python busca un bloque excepto que le indica cómo responder. Luego, Python ejecuta el código en ese bloque y el usuario ve un mensaje de error amigable en lugar de un rastreo:

```
¡No puedes dividir por cero!
```

Si hubiera más código seguido del bloque try-except , el programa continuaría ejecutándose porque le dijimos a Python cómo manejar el error. Veamos un ejemplo en el que detectar un error puede permitir que un programa continúe ejecutándose.

Uso de excepciones para evitar fallos

Manejar los errores correctamente es especialmente importante cuando el programa tiene más trabajo que hacer después de que ocurre el error. Esto sucede a menudo en programas que solicitan información a los usuarios. Si el programa responde adecuadamente a una entrada no válida, puede solicitar más entradas válidas en lugar de fallar.

Creemos una calculadora simple que solo haga divisiones:

```
división
_calculadora.py

print("Dame dos números y los dividiré.")
print("Ingresa 'q' para salir.")

mientras que Verdadero:
1 primer_número = entrada("\nPrimer número: ")
    si primer_número == 'q':
        romper
2 segundo_número = entrada("Segundo número: ")
    si segundo_número == 'q':
        romper
3 respuesta = int(primer_número) / int(segundo_número)
    imprimir (respuesta)
```

Este programa solicita al usuario que ingrese un primer número 1 y, si el usuario no ingresa q para salir, un segundo número 2. Luego dividimos estos dos números para obtener una respuesta 3. Este programa no hace nada para manejar los errores, por lo que le pide que dividir por cero hace que falle:

Dame dos números y los dividiré.

Ingresa 'q' para salir.

Primer número: 5

Segundo número: 0

Rastreo (llamadas recientes más última):

Archivo "division_calculator.py", línea 11, en <módulo>

respuesta = int(primer_número) / int(segundo_número)

ZeroDivisionError: división por cero

Es malo que el programa haya fallado, pero tampoco es una buena idea permitir que los usuarios vean los rastreos. Los usuarios no técnicos se sentirán confundidos y, en un entorno malicioso, los atacantes aprenderán más de lo que usted desea. Por ejemplo, sabrán el nombre de su archivo de programa y verán una parte de su código que no funciona correctamente. En ocasiones, un atacante experto puede utilizar esta información para determinar qué tipo de ataques utilizar contra su código.

El bloque más

Podemos hacer que este programa sea más resistente a errores ajustando la línea que podría producir errores en un bloque try-except . El error ocurre en la línea que realiza la división, así que ahí es donde colocaremos el bloque try-except .

Este ejemplo también incluye un bloque else . Cualquier código que dependa del intento.

El bloque que se ejecuta con éxito va en el bloque else :

```
--recorte--
```

```
mientras que Verdadero:
```

```
--recorte--
```

```
si segundo_número == 'q':
```

```
    romper
```

1 intento:

```
    respuesta = int(primer_número) / int(segundo_número)
```

2 excepto ZeroDivisionError:

```
    print("¡No se puede dividir entre 0!")
```

3 más:

```
    imprimir (respuesta)
```

Le pedimos a Python que intente completar la operación de división en un bloque de prueba 1, que incluye solo el código que podría causar un error. Cualquier código que dependa del éxito del bloque try se agrega al bloque else . En este caso, si la operación de división es exitosa, usamos el bloque else para imprimir el resultado 3.

El bloque excepto le dice a Python cómo responder cuando se produce un error ZeroDivisionError. surge 2. Si el bloque try no tiene éxito debido a un error de división por cero,

imprimimos un mensaje amigable diciéndole al usuario cómo evitar este tipo de error. El programa continúa ejecutándose y el usuario nunca ve un rastreo:

Dame dos números y los dividiré.
Ingrese 'q' para salir.

Primer número: 5
Segundo número: 0
¡No puedes dividir por 0!

Primer número: 5
Segundo número: 2
2.5

Primer número: q

El único código que debería ir en un bloque try es el código que podría provocar que se genere una excepción. A veces tendrás código adicional que debería ejecutarse sólo si el bloque de prueba fue exitoso; este código va en el bloque else .

El bloque `except` le dice a Python qué hacer en caso de que surja una determinada excepción cuando intenta ejecutar el código en el bloque `try`.

Al anticipar las posibles fuentes de errores, puede escribir programas sólidos que continúan ejecutándose incluso cuando encuentran datos no válidos y recursos faltantes. Su código será resistente a errores inocentes de usuarios y ataques maliciosos.

Manejo de la excepción FileNotFoundError

Un problema común al trabajar con archivos es el manejo de archivos faltantes. Es posible que el archivo que está buscando esté en una ubicación diferente, que el nombre del archivo esté mal escrito o que el archivo no exista en absoluto. Puedes manejar todas estas situaciones con un bloque try-except .

Intentemos leer un archivo que no existe. El siguiente programa intenta leer el contenido de Alicia en el país de las maravillas, pero no he guardado el archivo `alice.txt` en el mismo directorio que `alice.py`:

alice.py de ruta de importación pathlib

```
ruta = Ruta('alice.txt')
contenido = ruta.read_text(codificación='utf-8')
```

Tenga en cuenta que aquí estamos usando `read_text()` de una manera ligeramente diferente a la que vio anteriormente. El argumento de codificación es necesario cuando la codificación predeterminada de su sistema no coincide con la codificación del archivo que se está leyendo. Es más probable que esto suceda al leer un archivo que no se creó en su sistema.

Python no puede leer un archivo faltante, por lo que genera una excepción:

Rastreo (llamadas recientes más última):

- 1 Archivo "alice.py", línea 4, en <módulo>
- 2 contenidos = ruta.read_text(codificación='utf-8')

Archivo ".../pathlib.py", línea 1056, en read_text
 con self.open(modos='r', codificación=codificación, errores=errores) como f:

Archivo ".../pathlib.py", línea 1042, en abierto
 devolver io.open(self, modos, almacenamiento en búfer, codificación, errores, nueva línea)

3 FileNotFoundError: [Errno 2] No existe tal archivo o directorio: 'alice.txt'

Este es un rastreo más largo que los que hemos visto anteriormente, así que veamos cómo puedes entender rastreos más complejos. A menudo es mejor comenzar desde el final del rastreo. En la última línea, podemos ver que se generó una excepción FileNotFoundError 3. Esto es importante porque nos dice qué tipo de excepción usar en el bloque de excepción que escribiremos.

Mirando hacia atrás cerca del comienzo del rastreo 1, podemos ver que el error ocurrió en la línea 4 del archivo alice.py. La siguiente línea muestra la línea de código que causó el error 2. El resto del rastreo muestra parte del código de las bibliotecas que participan en la apertura y lectura de archivos. Por lo general, no es necesario leer ni comprender todas estas líneas en un rastreo.

Para manejar el error que se genera, el bloque de prueba comenzará con la línea que se identificó como problemática en el rastreo. En nuestro ejemplo, esta es la línea que contiene read_text():

```
desde pathlib importar ruta
```

```
ruta = Ruta('alice.txt')
```

```
intentar:
```

```
    contenido = ruta.read_text(codificación='utf-8')
```

```
1 excepto FileNotFoundError:
```

```
    print(f"Lo sentimos, el archivo {ruta} no existe.")
```

En este ejemplo, el código en el bloque try produce un FileNotFoundError, por lo que escribimos un bloque excepto que coincida con ese error 1. Luego, Python ejecuta el código en ese bloque cuando no se puede encontrar el archivo y el resultado es un error amigable. mensaje en lugar de un rastreo:

```
Lo sentimos, el archivo alice.txt no existe.
```

El programa no tiene nada más que hacer si el archivo no existe, por lo que este es todo el resultado que vemos. Analicemos este ejemplo y veamos cómo el manejo de excepciones puede ayudar cuando se trabaja con más de un archivo.

Analizando texto

Puede analizar archivos de texto que contengan libros completos. Muchas obras literarias clásicas están disponibles como archivos de texto simples porque son de dominio público. Los textos utilizados en esta sección provienen del Proyecto Gutenberg (<https://gutenberg.org>). El Proyecto Gutenberg mantiene una colección de obras literarias que están disponibles en el dominio público y es un gran recurso si está interesado en trabajar con textos literarios en sus proyectos de programación.

Busquemos el texto de Alicia en el país de las maravillas e intentemos contar el número de palabras en el texto. Para hacer esto, usaremos el método de cadena `split()`, que de forma predeterminada divide una cadena dondequiera que encuentre espacios en blanco:

```

desde pathlib importar ruta

ruta = Ruta('alice.txt')

intentar:
    contenido = ruta.read_text(codificación='utf-8')
excepto FileNotFoundError:
    print(f"Lo sentimos, el archivo {ruta} no existe.")
demás:

# Cuente el número aproximado de palabras en el archivo:
1 palabras = contenido.split()
2 núm_palabras = len(palabras)
print(f"El archivo {ruta} tiene aproximadamente {num_words} palabras.")

```

Moví el archivo `alice.txt` al directorio correcto, por lo que el bloque `try` funcionará esta vez. Tomamos el contenido de la cadena, que ahora contiene todo el texto de Alicia en el País de las Maravillas como una cadena larga, y usamos `split()` para producir una lista de todas las palabras del libro 1. Usar `len()` en esta lista 2 nos da una buena aproximación del número de palabras del texto original. Por último, imprimimos una declaración que informa cuántas palabras se encontraron en el archivo. Este código se coloca en el bloque `else` porque solo funciona si el código del bloque `try` se ejecutó correctamente.

El resultado nos dice cuántas palabras hay en `alice.txt`:

El archivo `alice.txt` tiene aproximadamente 29594 palabras.

El recuento es un poco alto porque el editor proporciona información adicional en el archivo de texto utilizado aquí, pero es una buena aproximación de la duración de Alicia en el país de las maravillas.

Trabajar con múltiples archivos

Agreguemos más libros para analizar, pero antes de hacerlo, traslademos la mayor parte de este programa a una función llamada `count_words()`. Esto facilitará la ejecución del análisis de varios libros:

`word_count.py` de `ruta` de importación `pathlib`

```

def contar_palabras(ruta):
1 """Cuente el número aproximado de palabras en un archivo."""
    intentar:
        contenido = ruta.read_text(codificación='utf-8')
    excepto FileNotFoundError:
        print(f"Lo sentimos, el archivo {ruta} no existe.")
    demás:

        # Cuente el número aproximado de palabras en el archivo:
        palabras = contenido.split()
        num_palabras = len(palabras)
        print(f"El archivo {ruta} tiene aproximadamente {num_words} palabras.")

```

```
ruta = Ruta('alice.txt')
contar_palabras(ruta)
```

La mayor parte de este código no ha cambiado. Solo se le ha sangrado y se ha movido al cuerpo de `count_words()`. Es un buen hábito mantener los comentarios actualizados cuando modificas un programa, por lo que el comentario también se cambió a una cadena de documentación y se reformuló ligeramente 1.

Ahora podemos escribir un bucle corto para contar las palabras de cualquier texto que queramos analizar. Hacemos esto almacenando los nombres de los archivos que queremos analizar en una lista y luego llamamos a `count_words()` para cada archivo de la lista. Intentaremos contar las palabras de Alicia en el país de las maravillas, Siddhartha, Moby Dick y Mujercitas, todas las cuales están disponibles en el dominio público. He dejado intencionalmente `siddhartha.txt` fuera del directorio que contiene `word_count.py`, para que podamos ver qué tan bien nuestro programa maneja un archivo faltante:

```
desde pathlib importar ruta
```

```
def contar_palabras(nombre de archivo):
    --recorte--
```

```
nombres de archivo = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt',
    'pequeñas_mujeres.txt']
```

```
para nombre de archivo en nombres de archivo:
```

```
1 ruta = Ruta (nombre de archivo)
    contar_palabras (ruta)
```

Los nombres de los archivos se almacenan como cadenas simples. Luego, cada cadena se convierte en un objeto `Path` 1, antes de la llamada a `count_words()`. El archivo `siddhartha.txt` que falta no tiene ningún efecto sobre el resto de la ejecución del programa:

El archivo `alice.txt` tiene aproximadamente 29594 palabras.

Lo sentimos, el archivo `siddhartha.txt` no existe.

El archivo `moby_dick.txt` tiene aproximadamente 215864 palabras.

El archivo `little_women.txt` tiene aproximadamente 189142 palabras.

El uso del bloque `try-except` en este ejemplo proporciona dos ventajas importantes. Evitamos que nuestros usuarios vean un rastreo y permitimos que el programa continúe analizando los textos que puede encontrar. Si no detectamos el `FileNotFoundError` que genera `siddhartha.txt`, el usuario verá un rastreo completo y el programa dejará de ejecutarse después de intentar analizar Siddhartha. Nunca analizaría Moby Dick o Mujercitas.

Fallando silenciosamente

En el ejemplo anterior, informamos a nuestros usuarios que uno de los archivos no estaba disponible. Pero no es necesario que informe todas las excepciones que detecte. A veces querrás que el programa falle silenciosamente cuando se produzca una excepción y continúe como si nada hubiera pasado. Para hacer que un programa falle silenciosamente, escribe un bloque `try` como de costumbre, pero le dices explícitamente a Python que lo haga.

nada en el bloque excepto . Python tiene una declaración de paso que le indica que no haga nada en un bloque:

```
def contar_palabras(ruta):
    """Cuenta el número aproximado de palabras en un archivo."""
    intentar:
        --recorte--
    excepto FileNotFoundError:
        aprobar
    demás:
        --recorte--
```

La única diferencia entre este listado y el anterior es el pase. declaración en el bloque excepto . Ahora, cuando se genera un `FileNotFoundError` , el código del bloque except se ejecuta, pero no sucede nada. No se produce ningún rastreo y no hay resultados en respuesta al error que se generó. Los usuarios ven el recuento de palabras de cada archivo que existe, pero no ven ninguna indicación de que no se encontró un archivo:

El archivo `alice.txt` tiene aproximadamente 29594 palabras.
 El archivo `moby_dick.txt` tiene aproximadamente 215864 palabras.
 El archivo `little_women.txt` tiene aproximadamente 189142 palabras.

La declaración de pase también actúa como marcador de posición. Es un recordatorio de que elige no hacer nada en un punto específico de la ejecución de su programa y que es posible que desee hacer algo allí más adelante. Por ejemplo, en este programa podríamos decidir escribir los nombres de archivos que faltan en un archivo llamado `miss_files.txt`. Nuestros usuarios no verían este archivo, pero podríamos leerlo y solucionar los textos faltantes.

Decidir qué errores informar

¿Cómo sabes cuándo informar un error a tus usuarios y cuándo dejar que tu programa falle silenciosamente? Si los usuarios saben qué textos se deben analizar, podrían agradecer un mensaje que les informe por qué algunos textos no fueron analizados. Si los usuarios esperan ver algunos resultados pero no saben qué libros se deben analizar, es posible que no necesiten saber que algunos textos no estaban disponibles. Proporcionar a los usuarios información que no buscan puede disminuir la usabilidad de su programa. Las estructuras de manejo de errores de Python le brindan un control detallado sobre cuánto compartir con los usuarios cuando algo sale mal; Depende de usted decidir cuánta información compartir.

El código bien escrito y probado adecuadamente no es muy propenso a errores internos, como errores de sintaxis o lógicos. Pero cada vez que su programa depende de algo externo, como la entrada del usuario, la existencia de un archivo o la disponibilidad de una conexión de red, existe la posibilidad de que se genere una excepción. Un poco de experiencia le ayudará a saber dónde incluir bloques de manejo de excepciones en su programa y cuánto informar a los usuarios sobre los errores que surjan.

INTÉNTALO TÚ MISMO

10-6. Además: un problema común cuando se solicita entrada numérica ocurre cuando las personas proporcionan texto en lugar de números. Cuando intentas convertir la entrada a un `int`, obtendrás un `ValueError`. Escribe un programa que solicite dos números. Súmalos e imprime el resultado. Capte el `ValueError` si alguno de los valores de entrada no es un número e imprima un mensaje de error amigable. Pruebe su programa ingresando dos números y luego ingresando algún texto en lugar de un número.

10-7. Calculadora de sumas: incluya el código del ejercicio 10-5 en un bucle `while` para que el usuario pueda continuar ingresando números, incluso si comete un error e ingresa texto en lugar de un número.

10-8. Perros y gatos: cree dos archivos, `cats.txt` y `dogs.txt`. Guarde al menos tres nombres de gatos en el primer archivo y tres nombres de perros en el segundo archivo. Escribe un programa que intente leer estos archivos e imprimir el contenido del archivo en la pantalla. Envuelva su código en un bloque `try-except` para detectar el error `FileNotFound` e imprima un mensaje amigable si falta un archivo. Mueva uno de los archivos a una ubicación diferente en su sistema y asegúrese de que el código en el bloque `except` se ejecute correctamente.

10-9. Perros y gatos silenciosos: modifique su bloque `except` en el ejercicio 10-7 para que falle silenciosamente si falta alguno de los archivos.

10-10. Palabras comunes: visite el Proyecto Gutenberg (<https://gutenberg.org>) y encuentre algunos textos que le gustaría analizar. Descargue los archivos de texto de estos trabajos o copie el texto sin formato de su navegador a un archivo de texto en su computadora.

Puedes utilizar el método `count()` para saber cuántas veces aparece una palabra o frase en una cadena. Por ejemplo, el siguiente código cuenta el número de veces que aparece 'fila' en una cadena:

```
>>> line = "Rema, rema, rema tu bote"
>>> línea.count('fila')
2
>>> línea.inferior().count('fila')
3
```

Tenga en cuenta que al convertir la cadena a minúsculas usando `lower()` se capturan todas las apariencias de la palabra que está buscando, independientemente de su formato.

Escribe un programa que lea los archivos que encuentre en el Proyecto Gutenberg y determine cuántas veces aparece la palabra "el" en cada texto. Esta será una aproximación porque también contará palabras como "entonces" y "allí".

Intente contar 'el ', con un espacio en la cadena, y vea cuánto baja su contar es.

Almacenamiento de datos

Muchos de sus programas pedirán a los usuarios que introduzcan ciertos tipos de información. Podría permitir a los usuarios almacenar preferencias en un juego o proporcionar datos para una visualización. Cualquiera que sea el enfoque de su programa, almacenará la información que los usuarios proporcionen en estructuras de datos como listas y diccionarios. Cuando los usuarios cierran un programa, casi siempre querrás guardar la información que ingresaron. Una forma sencilla de hacerlo consiste en almacenar sus datos utilizando el módulo `json`.

El módulo `json` le permite convertir estructuras de datos simples de Python en cadenas con formato JSON y luego cargar los datos de ese archivo la próxima vez que se ejecute el programa. También puedes usar `json` para compartir datos entre diferentes programas Python. Aún mejor, el formato de datos JSON no es específico de Python, por lo que puede compartir los datos que almacene en formato JSON con personas que trabajan en muchos otros lenguajes de programación. Es un formato útil y portátil, y es fácil de aprender.

NOTA El formato JSON (JavaScript Object Notation) se desarrolló originalmente para JavaScript. Sin embargo, desde entonces se ha convertido en un formato común utilizado por muchos lenguajes, incluido Python.

Usando `json.dumps()` y `json.loads()`

Escribamos un programa corto que almacene un conjunto de números y otro programa que lea estos números en la memoria. El primer programa usará `json.dumps()` para almacenar el conjunto de números y el segundo programa usará `json.loads()`.

La función `json.dumps()` toma un argumento: un dato que debe convertirse al formato JSON. La función devuelve una cadena, que luego podemos escribir en un archivo de datos:

```
número
_escritor.py
desde pathlib importar ruta
importar json

números = [2, 3, 5, 7, 11, 13]

1 ruta = Ruta('números.json')
2 contenidos = json.dumps(números)
  ruta.write_text(contenido)
```

Primero importamos el módulo `json` y luego creamos una lista de números para trabajar con. Luego elegimos un nombre de archivo en el que almacenar la lista de números 1. Es habitual utilizar la extensión de archivo `.json` para indicar que los datos del archivo se almacenan en formato JSON. A continuación, usamos `json.dumps()` 2 Función para generar una cadena que contiene la representación JSON de los datos con los que estamos trabajando. Una vez que tenemos esta cadena, la escribimos en el archivo usando el mismo método `write_text()` que usamos antes.

Este programa no tiene salida, pero abramos el archivo `números.json` y Míralo. Los datos se almacenan en un formato similar al de Python:

```
[2, 3, 5, 7, 11, 13]
```

Ahora escribiremos un programa separado que use `json.loads()` para leer la lista nuevamente en la memoria:

```
número
_lector.py  desde pathlib importar ruta
            importar json

1 ruta = Ruta('números.json')
2 contenidos = ruta.read_text()
3 números = json.loads(contenido)

imprimir (números)
```

Nos aseguramos de leer desde el mismo archivo que escribimos en 1. Dado que el archivo de datos es solo un archivo de texto con un formato específico, podemos leerlo con `read_text()` método 2. Luego pasamos el contenido del archivo a `json.loads()` 3. Esta función toma una cadena con formato JSON y devuelve un objeto Python (en este caso, una lista), que asignamos a `números`. Finalmente, imprimimos la lista de números recuperada y vemos que es la misma lista creada en `number_writer.py`:

```
[2, 3, 5, 7, 11, 13]
```

Esta es una forma sencilla de compartir datos entre dos programas.

Guardar y leer datos generados por el usuario

Guardar datos con json es útil cuando trabajas con datos generados por el usuario, porque si no almacenas la información del usuario de alguna manera, la perderás cuando el programa deje de ejecutarse. Veamos un ejemplo en el que le pedimos al usuario su nombre la primera vez que ejecuta un programa y luego recordamos su nombre cuando vuelve a ejecutar el programa.

Comencemos almacenando el nombre del usuario:

```
recordar
_yo.py  desde pathlib importar ruta
        importar json

1 nombre de usuario = entrada("¿Cuál es tu nombre? ")

2 ruta = Ruta('nombredeusuario.json')
  contenido = json.dumps(nombre de usuario)
  ruta.write_text(contenido)

3 print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
```

Primero solicitamos un nombre de usuario para almacenar 1. Luego, escribimos los datos que acabamos de recopilar en un archivo llamado `nombre de usuario.json` 2. Luego imprimimos un mensaje informando al usuario que hemos almacenado su información 3:

```
¿Cómo te llamas? eric
¡Te recordaremos cuando regreses, Eric!
```

Ahora escribamos un nuevo programa que salude a un usuario cuyo nombre ya ha sido almacenado:

greet_user.py de la ruta de importación de pathlib
importar json

```
1 ruta = Ruta('nombredeusuario.json')
  contenido = ruta.read_text()
2 nombre de usuario = json.loads (contenido)
```

```
print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
```

Leemos el contenido del archivo de datos 1 y luego usamos json.loads() para asignar los datos recuperados a la variable nombre de usuario 2. Como hemos recuperado el nombre de usuario, podemos darle la bienvenida al usuario con un saludo personalizado:

```
¡Bienvenido de nuevo, Eric!
```

Necesitamos combinar estos dos programas en un solo archivo. Cuando alguien ejecuta Remember_me.py, queremos recuperar su nombre de usuario de la memoria si es posible; de lo contrario, le solicitaremos un nombre de usuario y lo almacenaremos en nombre de usuario.json para la próxima vez. Podríamos escribir un bloque try-except aquí para responder adecuadamente si nombre de usuario.json no existe, pero en su lugar usaremos un método útil del módulo pathlib :

recordar
_yo.py desde pathlib importar ruta
importar json

```
ruta = Ruta('nombre de usuario.json')
1 si la ruta.exists():
  contenido = ruta.read_text()
  nombre de usuario = json.loads(contenido)
  print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
2 más:
  nombre de usuario = entrada("¿Cuál es tu nombre? ")
  contenido = json.dumps (nombre de usuario)
  ruta.write_text(contenido)
  print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
```

Hay muchos métodos útiles que puedes usar con objetos Path . El método existe() devuelve Verdadero si un archivo o carpeta existe y Falso si no existe.

Aquí usamos path.exists() para averiguar si ya se ha almacenado un nombre de usuario 1.

Si nombre de usuario.json existe, cargamos el nombre de usuario e imprimimos un saludo personalizado para el usuario.

Si el archivo nombre de usuario.json no existe 2, solicitamos un nombre de usuario y almacenamos el valor que ingresa el usuario. También imprimimos el mensaje familiar de que los recordaremos cuando regresen.

Cualquiera que sea el bloque que se ejecute, el resultado es un nombre de usuario y un nombre apropiado. saludo. Si es la primera vez que se ejecuta el programa, este es el resultado:

```
¿Cómo te llamas? eric
¡Te recordaremos cuando regreses, Eric!
```

De lo contrario:

¡Bienvenido de nuevo, Eric!

Este es el resultado que ve si el programa ya se ejecutó al menos una vez.

Aunque los datos de esta sección son solo una cadena, el programa funcionaría igual de bien con cualquier dato que pueda convertirse a una cadena con formato JSON.

Refactorización

A menudo, llegará a un punto en el que su código funcionará, pero reconocerá que podría mejorarlo dividiéndolo en una serie de funciones que tienen tareas específicas. Este proceso se llama refactorización. La refactorización hace que su código sea más limpio, más fácil de entender y de ampliar.

Podemos refactorizar `Remember_me.py` moviendo la mayor parte de su lógica a una o más funciones. El objetivo de `Remember_me.py` es saludar al usuario, así que muevamos todo nuestro código existente a una función llamada `greet_user()`:

recordar
_yo.py

```

desde pathlib importar ruta
importar json

def saludar_usuario():
1 """Saluda al usuario por su nombre."""
    ruta = Ruta('nombre de usuario.json')
    si la ruta existe():
        contenido = ruta.read_text()
        nombre de usuario = json.loads(contenido)
        print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
    demás:
        nombre de usuario = entrada("¿Cuál es tu nombre? ")
        contenido = json.dumps(nombre de usuario)
        ruta.write_text(contenido)
        print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")

saludar_usuario()

```

Debido a que ahora estamos usando una función, reescribimos los comentarios como una cadena de documentación que refleja cómo funciona actualmente el programa. 1. Este archivo es un poco más limpio, pero la función `greet_user()` hace más que solo saludar al usuario: es también recuperar un nombre de usuario almacenado si existe y solicitar un nuevo nombre de usuario si no existe.

Refactoricemos `greet_user()` para que no realice tantas tareas diferentes. Bien Comience moviendo el código para recuperar un nombre de usuario almacenado a una función separada:

```

desde pathlib importar ruta
importar json

def get_stored_username(ruta):
1 """Obtener nombre de usuario almacenado si está disponible."""

```

```

si path.exists(): contenido
    = path.read_text() nombre de usuario =
    json.loads(contenido) devuelve nombre de
    usuario más:

```

2 regresar Ninguno

```

def greet_user():
    """Saluda al usuario por su nombre.""" ruta
    = Ruta('nombredeusuario.json') nombre
    de usuario = get_stored_username(ruta)
3 si nombre de usuario:
    print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
else:
    nombre de usuario = input("¿Cuál es tu nombre? ")
    contenidos = json.dumps(nombre de usuario)
    path.write_text(contenidos) print(f"Te
    recordaremos cuando vuelvas, {nombre de usuario}!")

```

saludar_usuario()

La nueva función `get_stored_username()` 1 tiene un propósito claro, como se indica en la cadena de documentación. Esta función recupera un nombre de usuario almacenado y devuelve el nombre de usuario si encuentra uno. Si la ruta pasada a `get_stored_username()` no existe, la función devuelve Ninguno 2. Esta es una buena práctica: una función debe devolver el valor esperado o debe devolver Ninguno.

Esto nos permite realizar una prueba sencilla con el valor de retorno de la función. Imprimimos un mensaje de bienvenida para el usuario si el intento de recuperar un nombre de usuario tiene éxito 3 y, si no es así, solicitamos un nuevo nombre de usuario.

Deberíamos factorizar un bloque más de código de `greet_user()`. Si el nombre de usuario no existe, debemos mover el código que solicita un nuevo nombre de usuario a una función dedicada a ese propósito:

```

desde pathlib importar ruta importar
json

def get_stored_username(ruta): """Obtener
    nombre de usuario almacenado si está disponible. """ --
    snip--

def get_new_username(ruta): """Solicitar
    un nuevo nombre de usuario. """ nombre de
    usuario = input("¿Cuál es tu nombre? ") contenidos =
    json.dumps(nombre de usuario)
    ruta.write_text(contenido) devolver
    nombre de usuario

def greet_user():
    """Saluda al usuario por su nombre.""" ruta
    = Ruta('nombredeusuario.json') 1 nombre
    de usuario = get_stored_nombredeusuario(ruta) if nombre de
    usuario:
        print(f"¡Bienvenido de nuevo, {nombre de usuario}!")

```

demás:

```
2 nombre de usuario = get_new_username (ruta)
   print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
```

```
saludar_usuario()
```

Cada función en esta versión final de `Remember_me.py` tiene un propósito único y claro. Llamamos a `greet_user()`, y esa función imprime un mensaje apropiado: da la bienvenida a un usuario existente o saluda a un nuevo usuario. Lo hace llamando a `get_stored_username()` 1, que es responsable sólo de recuperar un nombre de usuario almacenado, si existe. Finalmente, si es necesario, `greet_user()` llama a `get_new_username()` 2, que es responsable únicamente de obtener un nuevo nombre de usuario y almacenarlo. Esta compartimentación del trabajo es una parte esencial para escribir un código claro que sea fácil de mantener y ampliar.

INTÉNTALO TÚ MISMO

10-11. Número favorito: escriba un programa que solicite el número favorito del usuario. Utilice `json.dumps()` para almacenar este número en un archivo. Escriba un programa separado que lea este valor e imprima el mensaje "¡Conozco tu número favorito! Es ____."

10-12. Número favorito recordado: combine los dos programas que escribió en el ejercicio 10-11 en un solo archivo. Si el número ya está almacenado, informe el número favorito al usuario. De lo contrario, solicite el número favorito del usuario y guárdelo en un archivo. Ejecute el programa dos veces para ver que funciona.

10-13. Diccionario de usuario: el ejemplo de `Remember_me.py` solo almacena una información: el nombre de usuario. Amplíe este ejemplo solicitando dos datos más sobre el usuario y luego almacene toda la información que recopile en un diccionario. Escriba este diccionario en un archivo usando `json.dumps()` y léalo nuevamente usando `json.loads()`. Imprima un resumen que muestre exactamente lo que su programa recuerda sobre el usuario.

10-14. Verificar usuario: el listado final de `Remember_me.py` supone que el usuario ya ingresó su nombre de usuario o que el programa se está ejecutando por primera vez. Deberíamos modificarlo en caso de que el usuario actual no sea la persona que utilizó el programa por última vez.

Antes de imprimir un mensaje de bienvenida en `greet_user()`, pregúntele al usuario si este es el nombre de usuario correcto. Si no es así, llame a `get_new_username()` para obtener la información correcta. nombre de usuario.

Resumen

En este capítulo, aprendió a trabajar con archivos. Aprendió a leer el contenido completo de un archivo y luego a revisar el contenido línea por línea si es necesario. Aprendiste a escribir todo el texto que quieras en un archivo.

También leerá sobre excepciones y cómo manejar las excepciones que probablemente verá en sus programas. Finalmente, aprendió a almacenar estructuras de datos de Python para poder guardar la información que brindan sus usuarios, evitando que tengan que comenzar de nuevo cada vez que ejecutan un programa.

En el Capítulo 11, aprenderá formas eficientes de probar su código. Esto ayudará a confiar en que el código que desarrolla es correcto y le ayudará a identificar los errores que se introducen a medida que continúa desarrollando los programas que ha escrito.

11

PROBANDO SU CÓDIGO



Cuando escribes una función o una clase, también puedes escribir pruebas para ese código. Las pruebas demuestran que su código funciona como se supone

en respuesta a todos los tipos de información para la que está diseñado. Cuando escribe pruebas, puede estar seguro de que su código funcionará correctamente a medida que más personas comiencen a utilizar sus programas. También podrá probar código nuevo a medida que lo agregue, para asegurarse de que sus cambios no rompan el comportamiento existente de su programa. Todo programador comete errores, por lo que cada programador debe probar su código con frecuencia para detectar problemas antes de que los usuarios los encuentren.

En este capítulo, aprenderá a probar su código usando `pytest`. El `pytest` La biblioteca es una colección de herramientas que lo ayudarán a escribir sus primeras pruebas de manera rápida y sencilla, mientras respaldan sus pruebas a medida que crecen en complejidad junto con sus proyectos. Python no incluye `pytest` de forma predeterminada, por lo que aprenderá a instalar bibliotecas externas. Saber cómo instalar bibliotecas externas pondrá a su disposición una amplia variedad de código bien diseñado. Estas bibliotecas ampliarán enormemente los tipos de proyectos en los que puede trabajar.

Aprenderá a crear una serie de pruebas y comprobará que cada conjunto de entradas dé como resultado el resultado que desea. Verá cómo se ve una prueba aprobada y una prueba fallida, y aprenderá cómo una prueba fallida puede ayudarlo a mejorar su código. Aprenderá a probar funciones y clases y comenzará a comprender cuántas pruebas escribir para un proyecto.

Instalando pytest con pip

Si bien Python incluye muchas funciones en la biblioteca estándar, los desarrolladores de Python también dependen en gran medida de paquetes de terceros. Un paquete de terceros es una biblioteca desarrollada fuera del lenguaje principal Python. Algunas bibliotecas populares de terceros finalmente se adoptan en la biblioteca estándar y terminan incluyéndose en la mayoría de las instalaciones de Python a partir de ese momento. Esto sucede con mayor frecuencia con bibliotecas que es poco probable que cambien mucho una vez que se hayan resuelto sus errores iniciales. Este tipo de bibliotecas pueden evolucionar al mismo ritmo que el lenguaje en general.

Sin embargo, muchos paquetes se mantienen fuera de la biblioteca estándar para que puedan desarrollarse en una línea de tiempo independiente del lenguaje mismo. Estos paquetes tienden a actualizarse con más frecuencia de lo que lo harían si estuvieran vinculados al cronograma de desarrollo de Python. Esto es cierto para pytest y para la mayoría de las bibliotecas que usaremos en la segunda mitad de este libro. No debe confiar ciegamente en todos los paquetes de terceros, pero tampoco debe desanimarse por el hecho de que muchas funciones importantes se implementan a través de dichos paquetes.

Actualizando pip

Python incluye una herramienta llamada pip que se utiliza para instalar paquetes de terceros. Debido a que pip ayuda a instalar paquetes desde recursos externos, se actualiza con frecuencia para abordar posibles problemas de seguridad. Entonces, comenzaremos actualizando pip.

Abra una nueva ventana de terminal y emita el siguiente comando:

```
$ python -m pip instalar --actualizar pip
1 Requisito ya satisfecho: pip en /.../python3.11/site-packages (22.0.4)
--recorte--
2 pip-22.1.2 instalado correctamente
```

La primera parte de este comando, `python -m pip`, le dice a Python que ejecute el módulo pip. La segunda parte, `install --upgrade`, le dice a pip que actualice un paquete que ya ha sido instalado. La última parte, `pip`, especifica qué paquete de terceros debe actualizarse. El resultado muestra que mi versión actual de pip, versión 22.0.4 1, fue reemplazada por la última versión en el momento de escribir este artículo, 22.1.2 2.

Puede utilizar este comando para actualizar cualquier paquete de terceros instalado en su sistema:

```
$ python -m pip install --actualizar nombre_paquete
```

NOTA

está utilizando Linux, es posible que pip no esté incluido con su instalación de Python. Si recibe un error al intentar actualizar pip, consulte las instrucciones en el Apéndice A.

Instalando pytest

Ahora que pip está actualizado, podemos instalar pytest:

```
$ python -m pip instalar --usuario pytest
Recogiendo pytest
--recorte--
Attrs-21.4.0 iniconfig-1.1.1 ...pytest-7.xx se instaló correctamente
```

Todavía estamos usando el comando principal pip install, sin --upgrade bandera esta vez. En su lugar, estamos usando el indicador --user , que le indica a Python que instale este paquete solo para el usuario actual. El resultado muestra que la última versión de pytest se instaló correctamente, junto con otros paquetes de los que depende pytest .

Puede utilizar este comando para instalar muchos paquetes de terceros:

```
$ python -m pip install --user nombre_paquete
```

NOTA Si tiene alguna dificultad para ejecutar este comando, intente ejecutar el mismo comando sin el indicador --user .

Probar una función

Para aprender sobre las pruebas, necesitamos código para probar. Aquí hay una función simple que toma un nombre y apellido, y devuelve un nombre completo perfectamente formateado:

```
nombre
_función.py def get_formatted_name(primer, último):
    """Genere un nombre completo con un formato ordenado."""
    nombre_completo = f"{primer} {último}"
    devolver nombre_completo.título()
```

La función get_formatted_name() combina el nombre y apellido con un espacio entre ellos para completar un nombre completo, y luego escribe en mayúscula y devuelve el nombre completo. Para comprobar que get_formatted_name() funciona, creemos un programa que utilice esta función. El programa nombres.py permite a los usuarios ingresar un nombre y apellido, y ver un nombre completo perfectamente formateado:

```
nombres.py de name_function importar get_formatted_name

print("Ingrese 'q' en cualquier momento para salir.")
mientras que Verdadero:
    primero = entrada("\nPor favor, dame un nombre: ")
    si primero == 'q':
        romper

    apellido = entrada("Por favor, dame un apellido: ")
    si último == 'q':
        romper

    nombre_formateado = obtener_nombre_formateado(primer, apellido)
    print(f"\tNombre bien formateado: {nombre_formateado}.")
```

Este programa importa `get_formatted_name()` desde `name_function.py`. El usuario puede ingresar una serie de nombres y apellidos y ver los nombres completos formateados que se generan:

Ingrese 'q' en cualquier momento para salir.

Por favor dame un nombre: janis

Por favor dame un apellido: joplin

Nombre cuidadosamente formateado: Janis Joplin.

Por favor dame un nombre: bob

Por favor dame un apellido: dylan

Nombre cuidadosamente formateado: Bob Dylan.

Por favor dame un nombre: q

Podemos ver que los nombres generados aquí son correctos. Pero digamos que queremos modificar `get_formatted_name()` para que también pueda manejar segundos nombres. Mientras lo hacemos, queremos asegurarnos de no alterar la forma en que la función maneja los nombres que solo tienen un nombre y apellido. Podríamos probar nuestro código ejecutando `nombres.py` e ingresando un nombre como Janis Joplin cada vez que modificamos `get_formatted_name()`, pero eso sería tedioso. Afortunadamente, `pytest` proporciona una manera eficiente de automatizar las pruebas de la salida de una función. Si automatizamos las pruebas de `get_formatted_name()`, siempre podemos estar seguros de que la función funcionará cuando se le den los tipos de nombres para los que hemos escrito las pruebas.

Pruebas unitarias y casos de prueba

Existe una amplia variedad de enfoques para probar software. Uno de los tipos de prueba más simples es la prueba unitaria. Una prueba unitaria verifica que un aspecto específico del comportamiento de una función es correcto. Un caso de prueba es una colección de pruebas unitarias que juntas demuestran que una función se comporta como se supone que debe hacerlo, dentro de la gama completa de situaciones que espera que maneje.

Un buen caso de prueba considera todos los tipos posibles de entrada que una función podría recibir e incluye pruebas para representar cada una de estas situaciones. Un caso de prueba con cobertura completa incluye una gama completa de pruebas unitarias que cubren todas las formas posibles en que se puede utilizar una función. Lograr una cobertura total en un proyecto grande puede resultar abrumador. A menudo es suficiente escribir pruebas para los comportamientos críticos de su código y luego aspirar a una cobertura completa sólo si el proyecto comienza a tener un uso generalizado.

Una prueba de aprobación

Con `pytest`, escribir tu primera prueba unitaria es bastante sencillo. Escribiremos una única función de prueba. La función de prueba llamará a la función que estamos probando y haremos una afirmación sobre el valor devuelto. Si nuestra afirmación es correcta, la prueba pasará; si la afirmación es incorrecta, la prueba fallará.

Aquí está la primera prueba de la función `get_formatted_name()`:

```

nombre de la prueba desde nombre_función importar get_formatted_name

_función.py
1 definición test_first_last_name():
    """¿Funcionan nombres como 'Janis Joplin'?"""
2 nombre_formateado = get_nombre_formateado('janis', 'joplin')
3 afirmar nombre_formatado == 'Janis Joplin'

```

Antes de ejecutar la prueba, echemos un vistazo más de cerca a esta función. El nombre de un archivo de prueba es importante; debe comenzar con `test_`. Cuando le preguntamos a `pytest` Para ejecutar las pruebas que hemos escrito, buscará cualquier archivo que comience con `test_` y ejecutará todas las pruebas que encuentre en ese archivo.

En el archivo de prueba, primero importamos la función que queremos probar: `get_nombre_formateado()`. Luego definimos una función de prueba: en este caso, `test_first_last_name()` 1. Este es un nombre de función más largo que el que hemos estado usando, por una buena razón. Primero, las funciones de prueba deben comenzar con la palabra prueba, seguida de un guión bajo. `pytest` descubrirá cualquier función que comience con `test_` y se ejecutará como parte del proceso de prueba.

Además, los nombres de las pruebas deben ser más largos y descriptivos que el nombre de una función típica. Nunca llamarás a la función tú mismo; `pytest` encontrará la función y la ejecutará por usted. Los nombres de las funciones de prueba deben ser lo suficientemente largos para que, si ve el nombre de la función en un informe de prueba, tenga una buena idea del comportamiento que se estaba probando.

A continuación, llamamos a la función que estamos probando 2. Aquí llamamos a `get_formatted_name()` con los argumentos 'janis' y 'joplin', tal como usamos cuando ejecutamos `nombres.py`. Asignamos el valor de retorno de esta función a `formatted_name`.

Finalmente, hacemos una afirmación 3. Una afirmación es una afirmación sobre una condición. Aquí afirmamos que el valor de `formatted_name` debería ser 'Janis Joplin'.

Ejecutar una prueba

Si ejecuta el archivo `test_name_function.py` directamente, no obtendrá ningún resultado porque nunca llamamos a la función de prueba. En su lugar, haremos que `pytest` ejecute el archivo de prueba por nosotros.

Para hacer esto, abra una ventana de terminal y navegue hasta la carpeta que contiene el archivo de prueba. Si está utilizando VS Code, puede abrir la carpeta que contiene el archivo de prueba y usar el terminal que está integrado en la ventana del editor. En la ventana de la terminal, ingrese el comando `pytest`. Esto es lo que deberías ver:

```

$ pytest
===== comienza la sesión de prueba =====

1 plataforma darwin: Python 3.xx, pytest-7.xx, pluggy-1.xx
2 directorio raíz: /.../python_work/chapter_11
3 recogidos 1 artículo

4 test_name_function.py. [100%]
===== 1 pasó en 0.00s =====

```

Intentemos darle sentido a este resultado. En primer lugar, vemos información sobre el sistema en el que se ejecuta la prueba 1. Estoy probando esto en un sistema macOS, por lo que es posible que vea algún resultado diferente aquí. Lo más importante es que podemos ver qué versiones de Python, pytest y otros paquetes se están utilizando para ejecutar la prueba.

A continuación, vemos el directorio donde se ejecuta la prueba desde 2: en mi caso, `python_work/chapter_11`. Podemos ver que pytest encontró una prueba para ejecutar 3, y podemos ver el archivo de prueba que se está ejecutando 4. El único punto después del nombre del archivo nos indica que se pasó una única prueba, y el 100% deja claro que todas de las pruebas se han realizado. Un proyecto grande puede tener cientos o miles de pruebas, y los puntos y el indicador de porcentaje completo pueden ser útiles para monitorear el progreso general de la ejecución de la prueba.

La última línea nos dice que se pasó una prueba y que tomó menos de 0,01 segundos ejecutarla.

Este resultado indica que la función `get_formatted_name()` siempre funcionará para nombres que tienen un nombre y apellido, a menos que modifiquemos la función. Cuando modificamos `get_formatted_name()`, podemos ejecutar esta prueba nuevamente. Si la prueba pasa, sabemos que la función seguirá funcionando para nombres como Janis Joplin.

NOTA Si no está seguro de cómo navegar hasta la ubicación correcta en la terminal, consulte "Ejecución de programas Python desde una terminal" en la página 11. Además, si ve un mensaje que indica que no se encontró el comando `pytest`, use el comando `python -m pytest` en su lugar.

Una prueba fallida

¿Cómo se ve una prueba fallida? Modifiquemos `get_formatted_name()` para que pueda manejar segundos nombres, pero hagámoslo de una manera que rompa la función para nombres con solo un nombre y apellido, como Janis Joplin.

Aquí hay una nueva versión de `get_formatted_name()` que requiere un argumento de segundo nombre:

```

nombre
def get_formatted_name(primer_nombre, segundo_nombre, apellido):
    """Genere un nombre completo con un formato ordenado."""
    nombre_completo = f'{primer_nombre} {segundo_nombre} {apellido}'
    devolver nombre_completo.titulo()
función.py

```

Esta versión debería funcionar para personas con segundos nombres, pero cuando la probamos, vemos que hemos roto la función para personas con solo un nombre y apellido.

Esta vez, ejecutar `pytest` produce el siguiente resultado:

```

$ pytest
===== comienza la sesión de prueba =====

--recorte--

1 test_name_function.py F [100%]
2 ===== FALLAS =====
3 _____ prueba_primer_apellido _____
   def prueba_primer_apellido():
       """¿Funcionan nombres como 'Janis Joplin'?"
4 >         nombre_formateado = get_nombre_formateado('janis', 'joplin')
5 mi         TypeError: get_formatted_name() falta 1 posicional requerido
           argumento: 'último'

```



```
test_name_function.py:5: Error de tipo
===== información breve del resumen de la prueba =====
FALLADO test_name_function.py::test_first_last_name - TypeError:
    get_formatted_name() falta 1 argumento posicional requerido: 'último'
===== 1 falló en 0,04 s =====
```

Hay mucha información aquí porque es posible que necesite saber muchas cosas cuando falla una prueba. El primer elemento a destacar en el resultado es una única F 1, que nos indica que una prueba falló. Luego vemos una sección que se centra en FALLOS 2, porque las pruebas fallidas suelen ser lo más importante en lo que centrarse en una ejecución de prueba. A continuación, vemos que `test_first_last_name()` fue la función de prueba que falló 3. Un corchete angular 4 indica la línea de código que provocó que la prueba fallara. La E en la siguiente línea 5 muestra el error real que causó la falla: un `TypeError` debido a la falta de un argumento posicional requerido, último. La información más importante se repite en un resumen más breve al final, de modo que cuando ejecute muchas pruebas, pueda tener una idea rápida de qué pruebas fallaron y por qué.

Responder a una prueba fallida

¿Qué haces cuando una prueba falla? Suponiendo que está verificando las condiciones correctas, una prueba aprobada significa que la función se está comportando correctamente y una prueba fallida significa que hay un error en el nuevo código que escribió. Entonces, cuando una prueba falla, no la cambie. Si lo hace, sus pruebas podrían pasar, pero cualquier código que llame a su función como lo hace la prueba dejará de funcionar repentinamente. En su lugar, corrija el código que está provocando que la prueba falle. Examine los cambios que acaba de realizar en la función y descubra cómo esos cambios rompieron el comportamiento deseado.

En este caso, `get_formatted_name()` solía requerir solo dos parámetros: un nombre y un apellido. Ahora requiere un nombre, segundo nombre y apellido. La adición de ese parámetro obligatorio de segundo nombre rompió el comportamiento original de `get_formatted_name()`. La mejor opción aquí es hacer que el segundo nombre sea opcional. Una vez que lo hagamos, nuestra prueba para nombres como Janis Joplin debería pasar nuevamente y también deberíamos poder aceptar segundos nombres. Modifiquemos `get_formatted_name()` para que los segundos nombres sean opcionales y luego ejecutemos el caso de prueba nuevamente. Si se aprueba, pasaremos a asegurarnos de que la función maneje los segundos nombres correctamente.

Para que los segundos nombres sean opcionales, movemos el parámetro `middle` al final de la lista de parámetros en la definición de función y asígnele un valor predeterminado vacío. También agregamos una prueba `if` que genera el nombre completo correctamente, dependiendo de si se proporciona un segundo nombre:

```
nombre
_función.py
def get_formatted_name(primer_nombre, último_nombre, medio=""):
    """Genere un nombre completo con un formato ordenado."""
    si_medio:
        nombre_completo = f'{primer_nombre} {segundo_nombre} {apellido}'
    demás:
        nombre_completo = f'{primer_nombre} {último_nombre}'
    devolver nombre_completo.título()
```

En esta nueva versión de `get_formatted_name()`, el segundo nombre es opcional. Si se pasa un segundo nombre a la función, el nombre completo contendrá un nombre, un segundo nombre y un apellido. De lo contrario, el nombre completo constará únicamente de nombre y apellido. Ahora la función debería funcionar para ambos tipos de nombres. Para saber si la función todavía funciona para nombres como Janis Joplin, ejecutemos la prueba nuevamente:

```
$ pytest
===== comienza la sesión de prueba =====

--recorte--
test_name_function.py . [100%]
===== 1 pasó en 0.00s =====
```

La prueba pasa ahora. Esto es ideal; significa que la función vuelve a funcionar para nombres como Janis Joplin, sin que tengamos que probar la función manualmente. Arreglar nuestra función fue más fácil porque la prueba fallida nos ayudó a identificar cómo el nuevo código rompía el comportamiento existente.

Agregar nuevas pruebas

Ahora que sabemos que `get_formatted_name()` vuelve a funcionar para nombres simples, escribamos una segunda prueba para personas que incluyen un segundo nombre. Hacemos esto agregando otra función de prueba al archivo `test_name_function.py`:

```
nombre de la prueba
_función.py

desde nombre_función importar get_formatted_name

def prueba_primer_apellido():
    --recorte--

def prueba_primer_apellido_segundo_nombre():
    """¿Funcionan nombres como 'Wolfgang Amadeus Mozart'?"""
    1 nombre_formateado = obtener_nombre_formateado(
        'wolfgang', 'mozart', 'amadeus')
    2 afirmar nombre_formato == 'Wolfgang Amadeus Mozart'
```

A esta nueva función la llamamos `test_first_last_middle_name()`. El nombre de la función debe comenzar con `test_` para que la función se ejecute automáticamente cuando ejecutamos `pytest`. Nombramos la función para dejar claro cuál es el comportamiento de `get_formatted_name()` estamos probando. Como resultado, si la prueba falla, sabremos de inmediato qué tipos de nombres se ven afectados.

Para probar la función, llamamos a `get_formatted_name()` con un nombre, apellido y segundo nombre 1, y luego hacemos una afirmación 2 de que el nombre completo devuelto coincide con el nombre completo (primero, segundo y apellido) que esperamos. Cuando ejecutamos `pytest` nuevamente, ambas pruebas pasan:

```
$pytest
===== comienza la sesión de prueba =====

--recorte--
recogió 2 artículos

1 test_name_function.py .. [100%]
===== 2 pasado en 0.01s =====
```

Los dos puntos 1 indican que se pasaron dos pruebas, lo que también queda claro en la última línea de resultado. ¡Esto es genial! Ahora sabemos que la función todavía funciona para nombres como Janis Joplin y podemos estar seguros de que también funcionará para nombres como Wolfgang Amadeus Mozart .

INTÉNTALO TÚ MISMO

11-1. Ciudad, País: escriba una función que acepte dos parámetros: un nombre de ciudad y un nombre de país. La función debe devolver una única cadena con el formato Ciudad, País, como Santiago, Chile. Guarde la función en un módulo llamado `city_functions.py` y guarde este archivo en una carpeta nueva para que `pytest` no intente ejecutar las pruebas que ya hemos escrito.

Cree un archivo llamado `test_cities.py` que pruebe la función que acaba de escribir. Escriba una función llamada `test_city_country()` para verificar que llamar a su función con valores como 'santiago' y 'chile' dé como resultado la cadena correcta. Ejecute la prueba y asegúrese de que `test_city_country()` pase.

11-2. Población: Modifique su función para que requiera un tercer parámetro, población. Ahora debería devolver una sola cadena con el formato Ciudad, País – población xxx, como Santiago, Chile – población 5000000. Ejecute la prueba nuevamente y asegúrese de que `test_city_country()` falle esta vez.

Modifique la función para que el parámetro de población sea opcional. Ejecute la prueba, y asegúrese de que `test_city_country()` pase nuevamente.

Escribe una segunda prueba llamada `test_city_country_population()` que verifique que puedes llamar a tu función con los valores 'santiago', 'chile' y 'población'. =5000000'. Ejecute las pruebas una vez más y asegúrese de que esta nueva prueba pase.

Probar una clase

En la primera parte de este capítulo, escribiste pruebas para una sola función. Ahora escribirás pruebas para una clase. Utilizará clases en muchos de sus propios programas, por lo que es útil poder demostrar que sus clases funcionan correctamente. Si ha aprobado pruebas para una clase en la que está trabajando, puede estar seguro de que las mejoras que realice en la clase no alterarán accidentalmente su comportamiento actual.

Una variedad de afirmaciones

Hasta ahora, has visto sólo un tipo de afirmación: una afirmación de que una cadena tiene un valor específico. Al escribir una prueba, puede hacer cualquier afirmación que pueda expresarse como una declaración condicional. Si la condición es Verdadera como se esperaba, se confirmará su suposición sobre cómo se comporta esa parte de su programa; puede estar seguro de que no existen errores. Si la condición que usted

suponga que es Verdadero y en realidad es Falso, la prueba fallará y sabrá que hay un problema que resolver. La tabla 11-1 muestra algunos de los tipos de afirmaciones más útiles que puede incluir en sus pruebas iniciales.

Tabla 11-1: Declaraciones de aserción comúnmente utilizadas en las pruebas

| Afirmación | Afirmar |
|--|--|
| <code>afirmar a == b</code> | Afirma que dos valores son iguales. |
| <code>afirmar a != b</code> | Afirma que dos valores no son iguales. |
| <code>afirmar un</code> | Afirme que a se evalúa como Verdadero. |
| <code>afirmar no un</code> | Afirme que a se evalúa como Falso. |
| <code>afirmar elemento en la lista</code> | Afirmar que un elemento está en una lista. |
| <code>afirmar elemento no en la lista</code> | Afirma que un elemento no está en una lista. |

Estos son solo algunos ejemplos; cualquier cosa que pueda expresarse como una declaración condicional puede incluirse en una prueba.

Una clase para probar

Probar una clase es similar a probar una función, porque gran parte del trabajo implica probar el comportamiento de los métodos de la clase. Sin embargo, existen algunas diferencias, así que escribamos una clase para probar. Considere una clase que ayude a administrar encuestas anónimas:

`clase encuesta.py` Encuesta anónima:

```

"""Recopilar respuestas anónimas a una pregunta de encuesta."""

1 def __init__(yo, pregunta):
    """Guarde una pregunta y prepárese para almacenar respuestas."""
    auto.pregunta = pregunta
    auto.respuestas = []

2 def show_question(yo):
    """Mostrar la pregunta de la encuesta."""
    imprimir (autopregunta)

3 def store_response(self, nueva_respuesta):
    """Almacenar una única respuesta a la encuesta."""
    self.responses.append(nueva_respuesta)

4 definición show_results(yo):
    """Mostrar todas las respuestas que se han dado."""
    imprimir("Resultados de la encuesta:")
    para la respuesta en respuestas propias:
        imprimir(f"- {respuesta}")

```

Esta clase comienza con una pregunta de encuesta que usted proporciona 1 e incluye una lista vacía para almacenar respuestas. La clase tiene métodos para imprimir la pregunta 2 de la encuesta, agregar una nueva respuesta a la lista de respuestas 3 e imprimir todas las respuestas almacenadas en la lista 4. Para crear una instancia de esta clase, todos sus

tenemos que proporcionar es una pregunta. Una vez que tenga una instancia que represente una encuesta en particular, muestre la pregunta de la encuesta con `show_question()`, almacene una respuesta usando `store_response()` y muestre los resultados con `show_results()`.

Para demostrar que la clase `AnonymousSurvey` funciona, escribamos un programa que use la clase:

```
idioma
_encuesta.py

desde la importación de encuestas AnonymousSurvey

# Defina una pregunta y realice una encuesta.
question = "¿Qué idioma aprendiste a hablar por primera vez?"
language_survey = Encuesta Anónima(pregunta)

# Mostrar la pregunta y almacenar las respuestas a la pregunta.
language_survey.show_question()
print("Ingrese 'q' en cualquier momento para salir.\n")
mientras que Verdadero:
    respuesta = entrada ("Idioma: ")
    si respuesta == 'q':
        romper
    language_survey.store_response(respuesta)

# Mostrar los resultados de la encuesta.
print("\n¡Gracias a todos los que participaron en la encuesta!")
language_survey.show_results()
```

Este programa define una pregunta ("¿Qué idioma aprendiste a hablar por primera vez?") y crea un objeto `AnonymousSurvey` con esa pregunta. El programa llama a `show_question()` para mostrar la pregunta y luego solicita respuestas. Cada respuesta se almacena a medida que se recibe. Cuando se han ingresado todas las respuestas (el usuario ingresa q para salir), `show_results()` imprime los resultados de la encuesta:

¿Qué idioma aprendiste a hablar por primera vez?

Ingrese 'q' en cualquier momento para salir.

Idioma: inglés
Idioma: español
Idioma: inglés
Idioma: mandarín
Idioma: q

¡Gracias a todos los que participaron en la encuesta!

Resultados de la encuesta:

- Inglés
- Español
- Inglés
- mandarín

Esta clase funciona para una encuesta anónima simple, pero decimos que queremos mejorar `AnonymousSurvey` y el módulo en el que se encuentra, `encuesta`. Podríamos permitir que cada usuario ingrese más de una respuesta, podríamos escribir un método para enumerar solo respuestas únicas e informar cuántas veces se dio cada respuesta, o incluso podríamos escribir otra clase para administrar encuestas no anónimas.

La implementación de tales cambios correría el riesgo de afectar el comportamiento actual de la clase `AnonymousSurvey`. Por ejemplo, es posible que al intentar permitir que cada usuario ingrese múltiples respuestas, accidentalmente podamos cambiar la forma en que se manejan las respuestas individuales. Para asegurarnos de no alterar el comportamiento existente a medida que desarrollamos este módulo, podemos escribir pruebas para la clase.

Probando la clase `AnonymousSurvey`

Escribamos una prueba que verifique un aspecto de la forma en que se comporta `AnonymousSurvey`. Escribiremos una prueba para verificar que una única respuesta a la pregunta de la encuesta se almacene correctamente:

`test_survey.py` de la importación de encuestas `AnonymousSurvey`

```
1 definición test_store_single_response():
    """Pruebe que una única respuesta esté almacenada correctamente."""
    question = "¿Qué idioma aprendiste a hablar por primera vez?"
2 language_survey = Encuesta Anónima(pregunta)
    language_survey.store_response('inglés')
3 afirmar 'inglés' en language_survey.responses
```

Empezamos importando la clase que queremos probar, `AnonymousSurvey`. La primera función de prueba verificará que cuando almacenemos una respuesta a la pregunta de la encuesta, la respuesta terminará en la lista de respuestas de la encuesta. Un buen nombre descriptivo para esta función es `test_store_single_response()` 1. Si esta prueba falla, sabremos por el nombre de la función en el resumen de la prueba que hubo un problema al almacenar una única respuesta a la encuesta.

Para probar el comportamiento de una clase, necesitamos crear una instancia de la clase. Creamos una instancia llamada `language_survey` 2 con la pregunta "¿Qué idioma aprendiste a hablar por primera vez?" Almacenamos una única respuesta, en inglés, utilizando el método `store_response()`. Luego verificamos que la respuesta se almacenó correctamente afirmando que el inglés está en la lista `language_survey.responses` 3.

De forma predeterminada, ejecutar el comando `pytest` sin argumentos ejecutará todas las pruebas que `pytest` descubra en el directorio actual. Para centrarse en las pruebas de un archivo, pase el nombre del archivo de prueba que desea ejecutar. Aquí ejecutaremos solo la prueba que escribimos para `AnonymousSurvey`:

```
$ pytest prueba_encuesta.py
===== comienza la sesión de prueba =====
--recorte--
test_survey.py. [100%]
===== 1 pasó en 0.01s =====
```

Éste es un buen comienzo, pero una encuesta sólo es útil si genera más de una respuesta. Verifiquemos que tres respuestas se puedan almacenar correctamente. Para hacer esto, agregamos otro método a `TestAnonymousSurvey`:

desde la importación de encuestas `AnonymousSurvey`

```
def test_store_single_response():
```

--recorte--

```
def test_store_tres_respuestas():
    """Pruebe que tres respuestas individuales estén almacenadas correctamente."""
    question = "¿Qué idioma aprendiste a hablar por primera vez?"
    language_survey = Encuesta Anónima(pregunta)
1 respuestas = ['inglés', 'español', 'mandarín']
    para respuesta en respuestas:
        language_survey.store_response(respuesta)

2 para respuesta en respuestas:
    afirmar respuesta en language_survey.responses
```

Llamamos a la nueva función `test_store_tres_respuestas()`. Creamos un objeto de encuesta tal como lo hicimos en `test_store_single_response()`. Definimos una lista que contiene tres respuestas diferentes 1 y luego llamamos a `store_response()` para cada una de estas respuestas. Una vez que se han almacenado las respuestas, escribimos otro bucle y afirmamos que cada respuesta ahora está en `language_survey .respuestas 2`.

Cuando volvemos a ejecutar el archivo de prueba, ambas pruebas (para una única respuesta y para tres respuestas) pase:

```
$ pytest prueba_encuesta.py
===== comienza la sesión de prueba =====
```

--recorte--

```
test_survey.py .. [100%]
===== 2 aprobado en 0.01s =====
```

Esto funciona perfectamente. Sin embargo, estas pruebas son un poco repetitivas, por lo que utilice otra característica de pytest para hacerlos más eficientes.

Usando accesorios

En `test_survey.py`, creamos una nueva instancia de `AnonymousSurvey` en cada función de prueba. Esto está bien en el breve ejemplo con el que estamos trabajando, pero en un proyecto del mundo real con decenas o cientos de pruebas, sería problemático.

Durante las pruebas, un dispositivo ayuda a configurar un entorno de prueba. A menudo, esto significa crear un recurso que sea utilizado por más de una prueba. Creamos un dispositivo en pytest escribiendo una función con el decorador `@pytest.fixture`. Un decorador es una directiva colocada justo antes de la definición de una función; Python aplica esta directiva a la función antes de ejecutarla, para alterar el comportamiento del código de la función.

No te preocupes si esto suena complicado; puedes empezar a utilizar decoradores de paquetes de terceros antes de aprender a escribirlos tú mismo.

Usemos un dispositivo para crear una única instancia de encuesta que se pueda usar en ambas funciones de prueba en `test_survey.py`:

```
importar prueba de Python
desde la importación de encuestas AnonymousSurvey
```

1 `@pytest.accesorio`

2 definición de `encuesta_idioma()`:

```
"""Una encuesta que estará disponible para todas las funciones de prueba."""
```

```
question = "¿Qué idioma aprendiste a hablar por primera vez?"
language_survey = Encuesta Anónima(pregunta)
devolver encuesta_idioma
```

3 definición `test_store_single_response(language_survey)`:

```
"""Pruebe que una única respuesta esté almacenada correctamente."""
```

4 `language_survey.store_response('inglés')`

```
afirmar 'inglés' en language_survey.responses
```

5 definición `test_store_tres_respuestas(encuesta_idioma)`:

```
"""Pruebe que tres respuestas individuales estén almacenadas correctamente."""
```

```
respuestas = ['inglés', 'español', 'mandarín']
```

```
para respuesta en respuestas:
```

6 `language_survey.store_response(respuesta)`

```
para respuesta en respuestas:
```

```
afirmar respuesta en language_survey.responses
```

Necesitamos importar `pytest` ahora, porque estamos usando un decorador definido en `pytest`. Aplicamos el decorador 1 `@pytest.fixture` a la nueva función `language_survey()` 2. Esta función construye un objeto `AnonymousSurvey` y devuelve la nueva encuesta.

Observe que las definiciones de ambas funciones de prueba han cambiado 3 5; Cada función de prueba ahora tiene un parámetro llamado `language_survey`. Cuando un parámetro en una función de prueba coincide con el nombre de una función con el decorador `@pytest.fixture`, el dispositivo se ejecutará automáticamente y el valor de retorno se pasará a la función de prueba. En este ejemplo, la función `language_survey()` proporciona a `test_store_single_response()` y `test_store_tres_responses()` una instancia de `language_survey`.

No hay código nuevo en ninguna de las funciones de prueba, pero observe que se han eliminado dos líneas de cada función 4 6: la línea que definió una pregunta y la línea que creó un objeto `AnonymousSurvey`.

Cuando volvemos a ejecutar el archivo de prueba, ambas pruebas aún pasan. Estas pruebas resultar particularmente útil cuando se intenta expandir `AnonymousSurvey` para manejar múltiples respuestas para cada persona. Después de modificar el código para aceptar múltiples respuestas, puede ejecutar estas pruebas y asegurarse de que no haya afectado la capacidad de almacenar una sola respuesta o una serie de respuestas individuales.

Es casi seguro que la estructura anterior parecerá complicada; Contiene algunos de los códigos más abstractos que hayas visto hasta ahora. No es necesario utilizar accesorios de inmediato; Es mejor escribir pruebas que tengan mucho código repetitivo que no escribir ninguna prueba. Solo sepa que cuando haya escrito suficientes pruebas como para que la repetición se interponga en su camino, existe una manera bien establecida de lidiar con la repetición. Además, los elementos fijos en ejemplos simples como este realmente no hacen que el código sea más corto ni más sencillo de seguir. Pero en proyectos con muchas pruebas, o en situaciones en las que se necesitan muchas líneas para crear un recurso que se utiliza en múltiples pruebas, los accesorios pueden mejorar drásticamente su código de prueba.

Cuando desee escribir un dispositivo, escriba una función que genere el recurso que utilizan múltiples funciones de prueba. Agregue el accesorio `@pytest`.

decorador a la nueva función y agregue el nombre de esta función como parámetro para cada función de prueba que utilice este recurso. Sus pruebas serán más cortas y más fáciles de escribir y mantener a partir de ese momento.

INTÉNTALO TÚ MISMO

11-3. Empleado: escriba una clase llamada Empleado. El método `__init__()` debe tomar un nombre, un apellido y un salario anual, y almacenar cada uno de ellos como atributos. Escriba un método llamado `Give_raise()` que agregue \$5,000 al salario anual de forma predeterminada pero que también acepte un monto de aumento diferente.

Escriba un archivo de prueba para Empleado con dos funciones de prueba, `test_give_default_raise()` y `test_give_custom_raise()`. Escriba sus pruebas una vez sin utilizar un dispositivo y asegúrese de que ambas pasen. Luego, escriba un dispositivo para no tener que crear una nueva instancia de empleado en cada función de prueba. Ejecute las pruebas nuevamente y asegúrese de que ambas se aprueben.

Resumen

En este capítulo, aprendió a escribir pruebas para funciones y clases usando herramientas en el módulo `pytest`. Aprendió a escribir funciones de prueba que verifican comportamientos específicos que deben exhibir sus funciones y clases. Viste cómo se pueden usar dispositivos para crear eficientemente recursos que se pueden usar en múltiples funciones de prueba en un archivo de prueba.

Las pruebas son un tema importante al que muchos programadores nuevos no están expuestos. No es necesario que escriba pruebas para todos los proyectos simples que prueba como nuevo programador. Pero tan pronto como empiece a trabajar en proyectos que impliquen un esfuerzo de desarrollo significativo, deberá probar los comportamientos críticos de sus funciones y clases. Estará más seguro de que el nuevo trabajo en su proyecto no dañará las partes que funcionan, y esto le dará la libertad de realizar mejoras en su código. Si accidentalmente interrumpes una funcionalidad existente, lo sabrás de inmediato, por lo que aún podrás solucionar el problema fácilmente.

Responder a una prueba fallida que ejecutó es mucho más fácil que responder a un informe de error de un usuario descontento.

Otros programadores respetarán más tus proyectos si incluyes algunas pruebas iniciales. Se sentirán más cómodos experimentando con su código y estarán más dispuestos a trabajar con usted en proyectos. Si desea contribuir a un proyecto en el que están trabajando otros programadores, se espera que demuestre que su código pasa las pruebas existentes y, por lo general, se espera que escriba pruebas para cualquier comportamiento nuevo que introduzca en el proyecto.

Experimente con las pruebas para familiarizarse con el proceso de prueba de su código. Escriba pruebas para los comportamientos más críticos de sus funciones y clases, pero no busque una cobertura completa en los primeros proyectos a menos que tenga una razón específica para hacerlo.

PARTE II

PROYECTOS

¡Felicidades! Ahora sabes lo suficiente sobre Python para comenzar a crear proyectos interactivos y significativos. Crear sus propios proyectos le enseñará nuevas habilidades y solidificará su comprensión de los conceptos presentados en la Parte I.

La Parte II contiene tres tipos de proyectos y usted puede elegir realizar cualquiera o todos estos proyectos en el orden que desee. A continuación se incluye una breve descripción de cada proyecto para ayudarle a decidir en cuál profundizar primero.

Invasión alienígena: hacer un juego con Python

En el proyecto Alien Invasion (capítulos 12, 13 y 14), utilizarás el paquete Pygame para desarrollar un juego 2D. El objetivo del juego es derribar una flota de alienígenas a medida que descienden por la pantalla, en niveles que aumentan en velocidad y dificultad. Al final del proyecto, habrás aprendido habilidades que te permitirán desarrollar tus propios juegos 2D en Pygame.

Visualización de datos

Los proyectos de visualización de datos comienzan en el Capítulo 15, donde aprenderá a generar datos y crear una serie de visualizaciones hermosas y funcionales de esos datos usando Matplotlib y Plotly. El Capítulo 16 le enseña a acceder a datos de fuentes en línea y a introducirlos en un paquete de visualización para crear gráficos de datos meteorológicos y un mapa de la actividad sísmica global. Finalmente, el Capítulo 17 le muestra cómo escribir un programa para descargar automáticamente

y visualizar datos. Aprender a realizar visualizaciones le permite explorar el campo de la ciencia de datos, que es una de las áreas de programación de mayor demanda en la actualidad.

Aplicaciones web

En el proyecto de aplicación web (capítulos 18, 19 y 20), utilizará el paquete Django para crear una aplicación web sencilla que permita a los usuarios llevar un diario sobre diferentes temas que han estado aprendiendo. Los usuarios crearán una cuenta con un nombre de usuario y contraseña, ingresarán un tema y luego harán entradas sobre lo que están aprendiendo. También implementará su aplicación en un servidor remoto para que cualquier persona en el mundo pueda acceder a ella.

Después de completar este proyecto, podrá comenzar a crear sus propias aplicaciones web simples y estará listo para profundizar en recursos más completos sobre la creación de aplicaciones con Django.

12

UN BARCO QUE DISPARA BALAS



¡Creemos un juego llamado Alien Invasion! Usaremos Pygame, una colección de divertidos y potentes módulos de Python que administran gráficos, animación e incluso sonido, lo que le facilitará la creación de juegos sofisticados. Con Pygame manejando tareas como dibujar imágenes en la pantalla, puedes concentrarte en la lógica de nivel superior de la dinámica del juego.

En este capítulo, configurarás Pygame y luego crearás un cohete que se mueve hacia la derecha y hacia la izquierda y dispara balas en respuesta a las acciones del jugador. En los próximos dos capítulos, crearás una flota de alienígenas para destruir y luego continuarás perfeccionando el juego estableciendo límites en la cantidad de naves que puedes usar y agregando un marcador.

Mientras creas este juego, también aprenderás a gestionar grandes proyectos que abarcan varios archivos. Refactorizaremos una gran cantidad de código y administraremos el contenido de los archivos para organizar el proyecto y hacer que el código sea eficiente.

Hacer juegos es una forma ideal de divertirse mientras aprende un idioma. Es profundamente satisfactorio jugar un juego que tú escribiste, y escribir un juego simple te enseñará mucho sobre cómo los profesionales desarrollan juegos. A medida que avanza en este capítulo, ingrese y ejecute el código para identificar cómo cada bloque de código contribuye al juego general. Experimenta con diferentes valores y configuraciones para comprender mejor cómo refinar las interacciones en tus juegos.

NOTA Alien Invasion abarca varios archivos diferentes, así que cree un nuevo `alien_invasion` carpeta en su sistema. Asegúrese de guardar todos los archivos del proyecto en esta carpeta para que sus declaraciones de importación funcionen correctamente.

Además, si se siente cómodo utilizando el control de versiones, es posible que desee utilizarlo para este proyecto. Si no ha utilizado el control de versiones antes, consulte el Apéndice D para obtener una descripción general.

Planificando tu proyecto

Cuando estás creando un proyecto grande, es importante preparar un plan antes de comenzar a escribir código. Su plan lo mantendrá concentrado y aumentará las probabilidades de que complete el proyecto.

Escribamos una descripción del juego general. Aunque la siguiente descripción no cubre todos los detalles de Alien Invasion, proporciona una idea clara de cómo empezar a construir el juego:

En Alien Invasion, el jugador controla un cohete que aparece en la parte inferior central de la pantalla. El jugador puede mover la nave hacia la derecha y hacia la izquierda usando las teclas de flecha y disparar balas usando la barra espaciadora. Cuando comienza el juego, una flota de extraterrestres llena el cielo y se mueve a lo largo y ancho de la pantalla. El jugador dispara y destruye a los alienígenas. Si el jugador destruye a todos los alienígenas, aparece una nueva flota que se mueve más rápido que la flota anterior. Si algún extraterrestre golpea la nave del jugador o llega al final de la pantalla, el jugador pierde una nave. Si el jugador pierde tres barcos, el juego termina.

Para la primera fase de desarrollo, crearemos una nave que pueda moverse hacia la derecha y hacia la izquierda cuando el jugador presione las teclas de flecha y disparar balas cuando el jugador presione la barra espaciadora. Después de configurar este comportamiento, podemos crear alienígenas y perfeccionar la jugabilidad.

Instalación de Pygame

Antes de comenzar a codificar, instale Pygame. Haremos esto de la misma manera que instalamos `pytest` en el Capítulo 11: con `pip`. Si se saltó el Capítulo 11 o necesita un repaso sobre `pip`, consulte “Instalación de `pytest` con `pip`” en la página 210.

Para instalar Pygame, ingrese el siguiente comando en el símbolo del terminal:

```
$ python -m pip instalar --usuario pygame
```

Si utiliza un comando que no sea Python para ejecutar programas o iniciar un `terminal` sesión, como `python3`, asegúrese de usar ese comando en su lugar.

Comenzando el proyecto del juego

Comenzaremos a construir el juego creando una ventana de Pygame vacía. Más adelante dibujaremos en esta ventana los elementos del juego, como la nave y los alienígenas. También haremos que nuestro juego responda a las entradas del usuario, estableceremos el color de fondo y cargaremos una imagen de barco.

Crear una ventana de Pygame y responder a la entrada del usuario

Crearemos una ventana de Pygame vacía creando una clase para representar el juego. En su editor de texto, cree un nuevo archivo y guárdelo como `alien_invasion.py`; luego ingresa lo siguiente:

```

extraterrestre
sistema de importación

_invasion.py

importar pygame

clase invasión alienígena:
    """Clase general para gestionar los activos y el comportamiento del juego."""

    def __init__(yo):
        """Inicializa el juego y crea recursos para el juego."""
1         pygame.init()

2         self.screen = pygame.display.set_mode((1200, 800))
        pygame.display.set_caption("Invasión alienígena")

    def run_game(yo):
        """Inicia el bucle principal del juego."""
3         mientras que Verdadero:
            # Esté atento a los eventos del teclado y el mouse.
4             para evento en pygame.event.get():
5                 si evento.tipo == pygame.SALIR:
                    sys.salir()

            # Hacer visible la pantalla dibujada más recientemente.
6             pygame.display.flip()

si __nombre__ == '__principal__':
    # Crea una instancia de juego y ejecuta el juego.
    ai = invasión alienígena()
    ai.run_game()

```

Primero, importamos los módulos `sys` y `pygame`. El módulo `pygame` contiene la funcionalidad que necesitamos para crear un juego. Usaremos herramientas en el sistema. módulo para salir del juego cuando el jugador sale.

`Alien Invasion` comienza como una clase llamada `AlienInvasion`. En el `__init__()` método, la función `pygame.init()` inicializa la configuración de fondo que Pygame necesita para funcionar correctamente 1. Luego llamamos a `pygame.display.set_mode()` para crear una ventana de visualización 2, en la que dibujaremos todos los elementos gráficos del juego. El argumento `(1200, 800)` es una tupla que define las dimensiones de la ventana del juego, que será de 1.200 píxeles de ancho por 800 píxeles de alto. (Puede ajustar estos valores según el tamaño de su pantalla). Asignamos esto

ventana de visualización al atributo `self.screen`, por lo que estará disponible en todos los métodos de la clase.

El objeto que asignamos a `self.screen` se llama superficie. Una superficie en Pygame es una parte de la pantalla donde se puede mostrar un elemento del juego. Cada elemento del juego, como un extraterrestre o una nave, es su propia superficie. La superficie devuelta por `display.set_mode()` representa toda la ventana del juego. Cuando activamos el bucle de animación del juego, esta superficie se volverá a dibujar en cada paso por el bucle, para que pueda actualizarse con cualquier cambio provocado por la entrada del usuario.

El juego está controlado por el método `run_game()`. Este método contiene un bucle `while 3` que se ejecuta continuamente. El bucle `while` contiene un bucle de eventos y un código que gestiona las actualizaciones de la pantalla. Un evento es una acción que el usuario realiza mientras juega, como presionar una tecla o mover el mouse. Para que nuestro programa responda a eventos, escribimos un bucle de eventos para escuchar eventos y realizar tareas apropiadas según los tipos de eventos que ocurren.

El bucle `for 4` anidado dentro del bucle `while` es un bucle de eventos.

Para acceder a los eventos que detecta Pygame, usaremos `pygame.event.get()` función. Esta función devuelve una lista de eventos que han tenido lugar desde la última vez que se llamó a esta función. Cualquier evento de teclado o mouse hará que se ejecute este bucle `for`. Dentro del ciclo, escribiremos una serie de declaraciones `if` para detectar y responder a eventos específicos. Por ejemplo, cuando el jugador hace clic en el botón de cerrar la ventana del juego, se detecta un evento `pygame.QUIT` y llamamos a `sys.exit()` para salir del juego 5.

La llamada a `pygame.display.flip()` 6 le dice a Pygame que haga visible la pantalla dibujada más recientemente. En este caso, simplemente dibuja una pantalla vacía en cada paso por el bucle `while`, borrando la pantalla anterior para que solo sea visible la nueva. Cuando movemos los elementos del juego, `pygame.display.flip()` actualiza continuamente la pantalla para mostrar las nuevas posiciones de los elementos del juego y ocultar las antiguas, creando la ilusión de un movimiento suave.

Al final del archivo, creamos una instancia del juego y luego llamamos `ejecutar_juego()`. Colocamos `run_game()` en un bloque `if` que solo se ejecuta si el archivo se llama directamente. Cuando ejecuta este archivo `alien_invasion.py`, debería ver una ventana de Pygame vacía.

Controlar la velocidad de fotogramas

Idealmente, los juegos deberían ejecutarse a la misma velocidad o velocidad de fotogramas en todos los sistemas. Controlar la velocidad de fotogramas de un juego que puede ejecutarse en múltiples sistemas es una cuestión compleja, pero Pygame ofrece una forma relativamente sencilla de lograr este objetivo. Haremos un reloj y nos aseguraremos de que el reloj marque una vez en cada paso por el bucle principal. Cada vez que el bucle se procese más rápido que la velocidad que definimos, Pygame calculará la cantidad de tiempo correcta para pausar para que el juego se ejecute a una velocidad constante.

Definiremos el reloj en el método `__init__()`:

`invasión_alienígena.py`

```
def __init__(yo):
    """Inicializa el juego y crea recursos para el juego."""
    pygame.init()
```



```
self.reloj = pygame.time.Reloj()
--recorte--
```

Después de inicializar pygame, creamos una instancia de la clase Clock, desde el módulo pygame.time . Luego haremos que el reloj marque el final del ciclo while en run_game():

```
def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:
    --recorte--
    pygame.display.flip()
    reloj.automático.tick(60)
```

El método tick() toma un argumento: la velocidad de fotogramas del juego. Aquí estoy usando un valor de 60, por lo que Pygame hará todo lo posible para que el bucle se ejecute exactamente 60 veces por segundo.

NOTA El reloj de Pygame debería ayudar a que el juego se ejecute de manera consistente en la mayoría de los sistemas. Si esto hace que el juego se ejecute de manera menos consistente en tu sistema, puedes probar diferentes valores para la velocidad de fotogramas. Si no puedes encontrar una buena velocidad de fotogramas en tu sistema, puedes omitir el reloj por completo y ajustar la configuración del juego para que funcione bien en tu sistema.

Configurar el color de fondo

Pygame crea una pantalla negra por defecto, pero eso es aburrido. Establezcamos un color de fondo diferente. Haremos esto al final del método __init__() .

```
extraterrestre
_invasion.py

def __init__(yo):
    --recorte--
    pygame.display.set_caption("Invasión alienígena")

    # Establecer el color de fondo.
    1 self.bg_color = (230, 230, 230)

    def run_game(yo):
        --recorte--
        para evento en pygame.event.get():
            si evento.tipo == pygame.SALIR:
                sys.salir()

        2 # Vuelva a dibujar la pantalla durante cada paso por el bucle.
          self.screen.fill(self.bg_color)

        # Hacer visible la pantalla dibujada más recientemente.
        pygame.display.flip()
        reloj.automático.tick(60)
```

Los colores en Pygame se especifican como colores RGB: una mezcla de rojo, verde y azul. Cada valor de color puede variar de 0 a 255. El valor de color (255, 0, 0) es rojo, (0, 255, 0) es verde y (0, 0, 255) es azul. Puedes mezclar diferentes valores RGB para crear hasta 16 millones de colores. El valor del color (230, 230, 230)

Mezcla cantidades iguales de rojo, azul y verde, lo que produce un color de fondo gris claro. Asignamos este color a `self.bg_color` 1.

Rellenamos la pantalla con el color de fondo usando el método `fill()` 2, que actúa sobre una superficie y toma un solo argumento: un color.

Crear una clase de configuración

Cada vez que introducimos una nueva funcionalidad en el juego, generalmente también creamos algunas configuraciones nuevas. En lugar de agregar configuraciones a lo largo del código, escribamos un módulo llamado configuración que contenga una clase llamada Configuración para almacenar todos estos valores en un solo lugar. Este enfoque nos permite trabajar con un solo objeto de configuración cada vez que necesitemos acceder a una configuración individual. Esto también facilita la modificación de la apariencia y el comportamiento del juego a medida que nuestro proyecto crece. Para modificar el juego, cambiaremos los valores relevantes en `settings.py`, que crearemos a continuación, en lugar de buscar diferentes configuraciones a lo largo del proyecto.

Cree un nuevo archivo llamado `settings.py` dentro de su carpeta `alien_invasion` y agregue esta clase de Configuración inicial :

Configuración de la clase `settings.py`:

```
"""Una clase para almacenar todas las configuraciones de Alien Invasion."""

def __init__(yo):
    """Inicializa la configuración del juego."""
    # Ajustes de pantalla
    self.ancha_pantalla = 1200
    self.screen_height = 800
    self.bg_color = (230, 230, 230)
```

Para crear una instancia de Configuración en el proyecto y usarla para acceder a nuestra configuración, necesitamos modificar `alien_invasion.py` de la siguiente manera:

```
estraterrestre
_invasion.py  --recorte--
import pygame

desde la configuración importar configuración

clase invasión alienígena:
    """Clase general para gestionar los activos y el comportamiento del juego."""

    def __init__(yo):
        """Inicializa el juego y crea recursos para el juego."""
        pygame.init()
        self.reloj = pygame.time.Realoj()
1         self.settings = Configuración()

2         self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Invasión alienígena")
```

```

def run_game(yo):
    --recorte--
    # Vuelva a dibujar la pantalla durante cada paso por el bucle.
    self.screen.fill(self.settings.bg_color)

    # Hacer visible la pantalla dibujada más recientemente.
    pygame.display.flip()
    reloj.automático.tick(60)
--recorte--

```

Importamos la configuración al archivo principal del programa. Luego creamos una instancia de Configuración y la asignamos a `self.settings` 1, luego de realizar la llamada a `pygame.init()`. Cuando creamos una pantalla 2, usamos los atributos `screen_width` y `screen_height` de `self.settings`, y luego usamos `self.settings` para acceder al color de fondo al llenar también la pantalla 3.

Cuando ejecutes `alien_invasion.py` ahora, todavía no verás ningún cambio, porque todo lo que hemos hecho es mover la configuración que ya estábamos usando en otro lugar. Ahora estamos listos para comenzar a agregar nuevos elementos a la pantalla.

Agregar la imagen del barco

Agreguemos el barco a nuestro juego. Para dibujar la nave del jugador en la pantalla, cargaremos una imagen y luego usaremos el método `blit()` de Pygame para dibujar la imagen.

Cuando elijas el arte para tus juegos, asegúrate de prestar atención. ción a la concesión de licencias. La forma más segura y económica de comenzar es utilizar gráficos con licencia gratuita que puede usar y modificar desde un sitio web como <https://opengameart.org>.

Puedes usar casi cualquier tipo de archivo de imagen en tu juego, pero es más fácil cuando usas un archivo de mapa de bits (.bmp) porque Pygame carga mapas de bits de forma predeterminada. Aunque puede configurar Pygame para usar otros tipos de archivos, algunos tipos de archivos dependen de ciertas bibliotecas de imágenes que deben estar instaladas en su computadora. La mayoría de las imágenes que encontrarás están en formatos .jpg o .png, pero puedes convertirlas a mapas de bits usando herramientas como Photoshop, GIMP y Paint.

Preste especial atención al color de fondo de la imagen elegida. Intente encontrar un archivo con un fondo transparente o sólido que pueda reemplazar con cualquier color de fondo utilizando un editor de imágenes. Tus juegos se verán mejor si el color de fondo de la imagen coincide con el color de fondo de tu juego. Alternativamente, puedes hacer coincidir el fondo de tu juego con el fondo de la imagen.

Para Alien Invasion, puede utilizar el archivo `ship.bmp` (Figura 12-1), que es disponible en los recursos de este libro en https://ehmatthes.github.io/pcc_3e. El color de fondo del archivo coincide con la configuración que estamos usando en este proyecto. Crea una carpeta llamada imágenes dentro de la carpeta principal de tu proyecto `alien_invasion`. Guarde el archivo `ship.bmp` en la carpeta de imágenes.

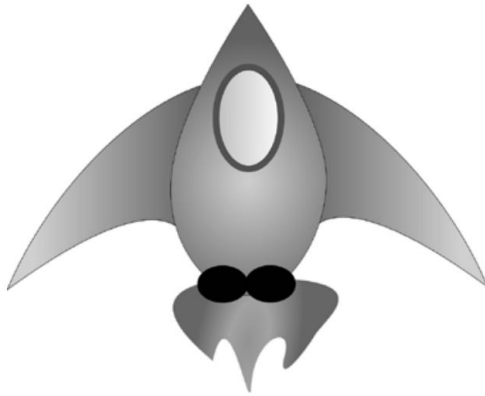


Figura 12-1: La nave de Alien Invasion

Creando la clase de barco

Después de elegir una imagen para el barco, debemos mostrarla en la pantalla. Para usar nuestro barco, crearemos un nuevo módulo de barco que contendrá la clase Barco. Esta clase gestionará la mayor parte del comportamiento de la nave del jugador:

ship.py importar pygame

Clase de barco:

```

    """Una clase para gestionar el barco."""

    def __init__(self, ai_game):
        """Inicializa el barco y establece su posición inicial."""
        1     self.pantalla = ai_game.pantalla
        2     self.screen_rect = ai_game.screen.get_rect()

        # Cargue la imagen del barco y obtenga su corrección.
        3     self.image = pygame.image.load('imágenes/ship.bmp')
        self.rect = self.image.get_rect()

        # Inicie cada barco nuevo en la parte inferior central de la pantalla.
        4     self.rect.midbottom = self.screen_rect.midbottom

    5 def blitme(yo):
        """Dibuja el barco en su ubicación actual."""
        self.screen.blit(self.image, self.rect)

```

Pygame es eficiente porque te permite tratar todos los elementos del juego como rectángulos (rectángulos), incluso si no tienen exactamente la forma de rectángulos. Tratar un elemento como un rectángulo es eficaz porque los rectángulos son formas geométricas simples. Cuando Pygame necesita determinar si dos elementos del juego han chocado, por ejemplo, puede hacerlo más rápidamente si trata cada objeto como un rectángulo. Este enfoque suele funcionar lo suficientemente bien como para que nadie que interprete el papel

El juego notará que no estamos trabajando con la forma exacta de cada elemento del juego. En esta clase trataremos la nave y la pantalla como rectángulos.

Importamos el módulo pygame antes de definir la clase. El `__init__()`

El método de Ship toma dos parámetros: la autorreferencia y una referencia a la instancia actual de la clase AlienInvasion. Esto le dará a Ship acceso a todos los recursos del juego definidos en AlienInvasion. Luego asignamos la pantalla a un atributo del Barco 1, para que podamos acceder a ella fácilmente en todos los métodos de esta clase. Accedemos al atributo `rect` de la pantalla usando el método `get_rect()` y lo asignamos a `self.screen_rect` 2. Hacerlo nos permite colocar la nave en la ubicación correcta en la pantalla.

Para cargar la imagen, llamamos a `pygame.image.load()` 3 y le damos la ubicación de la imagen de nuestra nave. Esta función devuelve una superficie que representa el barco, que asignamos a `self.image`. Cuando se carga la imagen, llamamos a `get_rect()` para acceder al atributo `rect` de la superficie del barco para que luego podamos usarlo para colocar el barco.

Cuando trabajas con un objeto recto, puedes usar las coordenadas `x` y `y` de los bordes superior, inferior, izquierdo y derecho del rectángulo, así como el centro, para colocar el objeto. Puede establecer cualquiera de estos valores para establecer la posición actual del rect. Cuando estés centrando un elemento del juego, trabaja con los atributos `center`, `centerx` o `centery` de un rect. Cuando estés trabajando en un borde de la pantalla, trabaja con los atributos superior, inferior, izquierdo o derecho. También hay atributos que combinan estas propiedades, como `midbottom`, `midtop`, `midleft` y `midright`. Cuando estés ajustando la ubicación horizontal o vertical del recto, puedes usar las teclas `x` e `y`

atributos, que son las coordenadas `x` y `y` de su esquina superior izquierda. Estos atributos le ahorran tener que hacer cálculos que antes los desarrolladores de juegos tenían que hacer manualmente, y los utilizará con frecuencia.

NOTA En Pygame, el origen (0, 0) está en la esquina superior izquierda de la pantalla y las coordenadas aumentan a medida que desciende y hacia la derecha. En una pantalla de 1200×800, el origen está en la esquina superior izquierda y la esquina inferior derecha tiene las coordenadas (1200, 800). Estas coordenadas se refieren a la ventana del juego, no a la pantalla física.

Colocaremos el barco en la parte inferior central de la pantalla. Para hacerlo, haga que el valor de `self.rect.midbottom` coincida con el atributo `midbottom` del rect 4 de la pantalla. Pygame usa estos atributos `rect` para posicionar la imagen de la nave de modo que esté centrada horizontalmente y alineada con la parte inferior de la pantalla.

Finalmente, definimos el método 5 `blitme()`, que dibuja la imagen en la pantalla en la posición especificada por `self.rect`.

Llevando el barco a la pantalla

Ahora actualicemos `alien_invasion.py` para que cree una nave y llame al método `blitme()` de la nave:

```
extraterrestre
_invasion.py

--recorte--
desde la configuración importar configuración
desde barco importación Barco
```

```

clase invasión alienígena:
    """Clase general para gestionar los activos y el comportamiento del juego."""

    def __init__(yo):
        --recorte--
        pygame.display.set_caption("Invasión alienígena")

1         self.ship = Enviar (yo)

    def run_game(yo):
        --recorte--
        # Vuelva a dibujar la pantalla durante cada paso por el bucle.
        self.screen.fill(self.settings.bg_color)
2         auto.envío.blitme()

        # Hacer visible la pantalla dibujada más recientemente.
        pygame.display.flip()
        reloj.automático.tick(60)

--recorte--

```

Importamos Ship y luego creamos una instancia de Ship después de que se haya creado la pantalla 1. La llamada a Ship() requiere un argumento: una instancia de AlienInvasion. El argumento propio aquí se refiere a la instancia actual de AlienInvasion. Este es el parámetro que le da a Ship acceso a los recursos del juego, como el objeto de pantalla. Asignamos esta instancia de Ship a self.ship.

Después de llenar el fondo, dibujamos el barco en la pantalla llamando ship.blitme(), para que el barco aparezca encima del fondo 2.

Cuando ejecutes alien_invasion.py ahora, deberías ver un juego vacío. pantalla con el cohete sentado en la parte inferior central, como se muestra en la Figura 12-2.

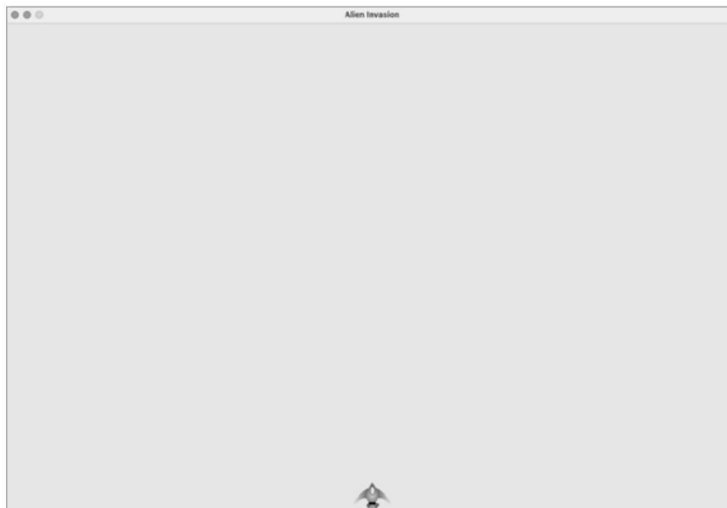


Figura 12-2: Invasión alienígena con la nave en la parte inferior central de la pantalla

Refactorización: los métodos `_check_events()` y `_update_screen()`

En proyectos grandes, a menudo refactorizarás el código que hayas escrito antes de agregar más código. La refactorización simplifica la estructura del código que ya ha escrito, lo que facilita su desarrollo. En esta sección, romperemos el `run_game()` método, que se está volviendo largo, en dos métodos auxiliares. Un método de ayuda funciona dentro de una clase pero no debe ser utilizado por código fuera de la clase. En Python, un único guión bajo inicial indica un método auxiliar.

El método `_check_events()`

Moveremos el código que administra eventos a un método separado llamado `_check_events()`. Esto simplificará `run_game()` y aislará el ciclo de gestión de eventos. Aislar el bucle de eventos te permite gestionar los eventos por separado de otros aspectos del juego, como la actualización de la pantalla.

Aquí está la clase `AlienInvasion` con el nuevo método `_check_events()`, que sólo afecta al código en `run_game()`:

extraterrestre
_invasion.py

```
def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:
1        self._check_events()

        # Vuelva a dibujar la pantalla durante cada paso por el bucle.
        --recorte--

2 def _check_events(yo):
    """Responder a pulsaciones de teclas y eventos del mouse."""
    para evento en pygame.event.get():
        si evento.tipo == pygame.SALIR:
            sys.salir()
```

Creamos un nuevo método `_check_events()` 2 y movemos las líneas que verifican si el jugador ha hecho clic para cerrar la ventana en este nuevo método.

Para llamar a un método desde dentro de una clase, use la notación de puntos con la variable `self` y el nombre del método 1. Llamamos al método desde dentro del bucle `while` en `run_game()`.

El método `_update_screen()`

Para simplificar aún más `run_game()`, moveremos el código para actualizar la pantalla a un método separado llamado `_update_screen()`:

invasión_alienígena.py

```
def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:
        self._check_events()
        self._update_screen()
        reloj.automático.tick(60)

def _check_events(yo):
    --recorte--
```

```
def _update_screen(auto):
    """Actualiza imágenes en la pantalla y cambia a la nueva pantalla."""
    self.screen.fill(self.settings.bg_color)
    auto.envío.blitme()

    pygame.display.flip()
```

Movimos el código que dibuja el fondo y el barco y volteamos la pantalla a `_update_screen()`. Ahora el cuerpo del bucle principal en `run_game()` es mucho más sencillo. Es fácil ver que estamos buscando nuevos eventos, actualizando la pantalla y marcando el reloj en cada paso por el bucle.

Si ya has creado varios juegos, probablemente comenzarás dividiendo tu código en métodos como estos. Pero si nunca ha abordado un proyecto como este, probablemente al principio no sabrá exactamente cómo estructurar su código. Este enfoque le da una idea de un proceso de desarrollo realista: comienza escribiendo su código de la manera más simple posible y luego lo refactoriza a medida que su proyecto se vuelve más complejo.

Ahora que hemos reestructurado el código para que sea más fácil agregarlo, podemos ¡Trabaja en los aspectos dinámicos del juego!

INTÉNTALO TÚ MISMO

12-1. Cielo azul: crea una ventana de Pygame con un fondo azul.

12-2. Personaje del juego: busque una imagen de mapa de bits de un personaje del juego que le guste o convierta una imagen a un mapa de bits. Cree una clase que dibuje el personaje en el centro de la pantalla, luego haga coincidir el color de fondo de la imagen con el color de fondo de la pantalla o viceversa.

Pilotando el barco

A continuación, le daremos al jugador la posibilidad de mover el barco hacia la derecha y hacia la izquierda. Escribiremos un código que responda cuando el jugador presione la tecla de flecha derecha o izquierda. Nos centraremos primero en el movimiento hacia la derecha y luego aplicaremos los mismos principios para controlar el movimiento hacia la izquierda. A medida que agreguemos este código, aprenderá cómo controlar el movimiento de las imágenes en la pantalla y responder a las entradas del usuario.

Responder a una pulsación de tecla

Cada vez que el jugador presiona una tecla, esa pulsación se registra en Pygame como un evento. Cada evento es recogido por el método `pygame.event.get()`. Necesitamos especificar en nuestro método `_check_events()` qué tipo de eventos queremos que el juego verifique. Cada pulsación de tecla se registra como un evento `KEYDOWN`.

Cuando Pygame detecta un evento KEYDOWN , debemos verificar si la tecla que se presionó es una que desencadena una determinada acción. Por ejemplo, si el jugador presiona la tecla de flecha derecha, queremos aumentar el valor rect.x del barco para mover el barco hacia la derecha:

```

extraterrestre
_invasion.py

def _check_events(yo):
    """Responder a pulsaciones de teclas y eventos del mouse."""
    para evento en pygame.event.get():
        si evento.tipo == pygame.SALIR:
            sys.salir()
1         elif evento.tipo == pygame.KEYDOWN:
2             si evento.key == pygame.K_RIGHT:
                    # Mueve el barco hacia la derecha.
3                 auto.envio.rect.x += 1

```

Dentro de `_check_events()` agregamos un bloque `elif` al bucle de eventos, para responder cuando Pygame detecta un evento KEYDOWN 1. Comprobamos si la tecla presionada, `event.key`, es la tecla de flecha derecha 2. La tecla de flecha derecha está representada por `pygame.K_RIGHT`. Si se presionó la tecla de flecha derecha, movemos el barco hacia la derecha aumentando el valor de `self.ship.rect.x` en 1 3.

Cuando ejecutas `alien_invasion.py` ahora, la nave debería moverse un píxel hacia la derecha cada vez que presionas la tecla de flecha derecha. Eso es un comienzo, pero no es una manera eficiente de controlar la nave. Mejoremos este control permitiendo el movimiento continuo.

Permitir el movimiento continuo

Cuando el jugador mantiene presionada la tecla de flecha derecha, queremos que el barco continúe moviéndose hacia la derecha hasta que el jugador suelte la tecla. Haremos que el juego detecte un evento `pygame.KEYUP` para saber cuándo se suelta la tecla de flecha derecha; luego usaremos los eventos KEYDOWN y KEYUP junto con una bandera llamada `moving_right` para implementar el movimiento continuo.

Cuando la bandera `moving_right` sea `False`, el barco estará inmóvil. Cuando el jugador presiona la tecla de flecha derecha, configuraremos la bandera en `Verdadero`, y cuando el jugador suelte la tecla, configuraremos la bandera en `Falso` nuevamente.

La clase `Ship` controla todos los atributos del barco, por lo que le daremos un atributo llamado `moving_right` y un método `update()` para verificar el estado de la bandera `moving_right`. El método `update()` cambiará la posición del barco si la bandera está configurada en `Verdadero`. Llamaremos a este método una vez en cada paso por el bucle `while` para actualizar la posición del barco.

Estos son los cambios en `Enviar`:

```

clase ship.py Envío:
    """Una clase para gestionar el barco."""

    def __init__(self, ai_game):
        --recorte--
        # Inicie cada barco nuevo en la parte inferior central de la pantalla.
        self.rect.midbottom = self.screen_rect.midbottom

```

```

1      # Bandera de movimiento; Comience con un barco que no se mueve.
      self.moving_right = Falso

2 actualización de definición (auto):
      """Actualiza la posición del barco según la bandera de movimiento."""
      si self.moving_right:
          auto.rect.x += 1

```

```

def blitme(yo):
    --recorte--

```

Agregamos un atributo `self.moving_right` en el método `__init__()` y lo configuramos en `False` inicialmente. Luego agregamos `update()`, que mueve el barco hacia la derecha si la bandera es `True`. El método `update()` será llamado desde afuera. la clase, por lo que no se considera un método auxiliar.

Ahora necesitamos modificar `_check_events()` para que `moving_right` esté establecido en `True` cuando se presiona la tecla de flecha derecha y `False` cuando se suelta la tecla:

```

estraterrestre
_invasion.py

def _check_events(yo):
    """Responder a pulsaciones de teclas y eventos del mouse."""
    para evento en pygame.event.get():
        --recorte--
        elif evento.tipo == pygame.KEYDOWN:
            si evento.key == pygame.K_RIGHT:
1                self.ship.moving_right = Verdadero
2            elif evento.tipo == pygame.KEYUP:
                si evento.key == pygame.K_RIGHT:
                    self.ship.moving_right = Falso

```

Aquí, modificamos cómo responde el juego cuando el jugador presiona la tecla de flecha derecha: en lugar de cambiar la posición del barco directamente, simplemente configuramos `moving_right` en `True`. Luego agregamos un nuevo bloque `elif`, que responde a los eventos `KEYUP`. 2. Cuando el jugador suelta la tecla de flecha derecha (`K_RIGHT`), configuramos `moving_right` en `False`.

A continuación, modificamos el bucle `while` en `run_game()` para que llame a la actualización del barco(). método en cada paso a través del bucle:

```

invasión_alienígena.py

def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:
        self._check_events()
        auto.envío.actualización()
        self._update_screen()
        reloj.automático.tick(60)

```

La posición del barco se actualizará después de que hayamos verificado los eventos del teclado y antes de actualizar la pantalla. Esto permite que la posición del barco se actualice en respuesta a la entrada del jugador y garantiza que la posición actualizada se utilizará al dibujar el barco en la pantalla.

Cuando ejecuta `alien_invasion.py` y mantiene presionada la tecla de flecha derecha, el La nave debe moverse continuamente hacia la derecha hasta que sueltes la tecla.

Moviéndose tanto hacia la izquierda como hacia la derecha

Ahora que el barco puede moverse continuamente hacia la derecha, agregar movimiento hacia la izquierda es sencillo. Nuevamente, modificaremos la clase Ship y el método `_check_events()`. Estos son los cambios relevantes a `__init__()` y `update()` en Ship:

```
nave.py
def __init__(self, ai_game):
    --recorte--
    # Banderas de movimiento; Comience con un barco que no se mueve.
    self.moving_right = Falso
    self.moving_left = Falso

    actualización def (auto):
        """Actualiza la posición del barco según las banderas de movimiento."""
        si self.moving_right:
            auto.rect.x += 1

        si self.moving_left:
            auto.rect.x -= 1
```

En `__init__()`, agregamos una bandera `self.moving_left`. En `update()`, usamos dos bloques if separados, en lugar de un elif, para permitir que el valor `rect.x` del barco aumente y luego disminuya cuando se mantienen presionadas ambas teclas de flecha. Esto hace que el barco se detenga. Si usáramos elif para movernos hacia la izquierda, la tecla de flecha derecha siempre tendría prioridad. El uso de dos bloques if hace que los movimientos sean más precisos cuando el jugador puede mantener presionadas momentáneamente ambas teclas al cambiar de dirección.

Tenemos que hacer dos adiciones a `_check_events()`:

```
invasión_alienígena.py
def _check_events(yo):
    """Responder a pulsaciones de teclas y eventos del mouse."""
    para evento en pygame.event.get():
        --recorte--
        elif evento.tipo == pygame.KEYDOWN:
            si evento.key == pygame.K_RIGHT:
                self.ship.moving_right = Verdadero
            elif evento.key == pygame.K_LEFT:
                self.ship.moving_left = Verdadero

        elif evento.tipo == pygame.KEYUP:
            si evento.key == pygame.K_RIGHT:
                self.ship.moving_right = Falso
            elif evento.key == pygame.K_LEFT:
                self.ship.moving_left = Falso
```

Si ocurre un evento KEYDOWN para la tecla `K_LEFT`, configuramos `moving_left` en True. Si ocurre un evento KEYUP para la tecla `K_LEFT`, configuramos `moving_left` en False. Podemos usar bloques elif aquí porque cada evento está conectado a una sola clave. Si el jugador presiona ambas teclas a la vez, se detectarán dos eventos separados.

Cuando ejecutes `alien_invasion.py` ahora, deberías poder mover la nave continuamente hacia la derecha y hacia la izquierda. Si mantienes presionadas ambas teclas, el barco debería dejar de moverse.

A continuación, perfeccionaremos aún más el movimiento del barco. Ajustemos la velocidad del barco y limitemos qué tan lejos puede moverse para que no desaparezca de los lados de la pantalla.

Ajustar la velocidad del barco

Actualmente, la nave se mueve un píxel por ciclo a través del bucle while , pero podemos tomar un control más preciso de la velocidad de la nave agregando un atributo ship_speed a la clase Configuración . Usaremos este atributo para determinar qué tan lejos mover el barco en cada paso por el circuito. Aquí está el nuevo atributo en settings.py:

Configuración de la clase settings.py :

```
"""Una clase para almacenar todas las configuraciones de Alien Invasion."""

def __init__(yo):
    --recorte--

    # Configuración del barco
    self.ship_speed = 1,5
```

Establecemos el valor inicial de ship_speed en 1,5. Cuando el barco se mueve ahora, su posición se ajusta en 1,5 píxeles (en lugar de 1 píxel) en cada paso por el bucle.

Estamos usando un flotador para configurar la velocidad para darnos un control más preciso de la velocidad del barco cuando aumentemos el ritmo del juego más adelante. Sin embargo, los atributos rect como x almacenan solo valores enteros, por lo que debemos realizar algunas modificaciones en Ship:

clase ship.py Envío:

```
"""Una clase para gestionar el barco."""

def __init__(self, ai_game):
    """Inicializa el barco y establece su posición inicial."""
    self.pantalla = ai_game.pantalla
    self.settings = ai_game.settings
    --recorte--

    # Inicie cada barco nuevo en la parte inferior central de la pantalla.
    self.rect.midbottom = self.screen_rect.midbottom

    # Almacene un flotador para la posición horizontal exacta del barco.
    self.x = flotante(self.rect.x)

    # Banderas de movimiento; Comience con un barco que no se mueve.
    self.moving_right = Falso
    self.moving_left = Falso

    actualización def (auto):
        """Actualiza la posición del barco según las banderas de movimiento."""
        # Actualiza el valor x del barco, no el rect.
        si self.moving_right:
            3         self.x += self.settings.ship_speed
```

```

        si self.moving_left:
            self.x -= self.settings.ship_speed

4         # Actualizar objeto rect desde self.x.
        self.rect.x = self.x

def blitme(yo):
    --recorte--

```

Creemos un atributo de configuración para Ship, para que podamos usarlo en update() 1. Debido a que estamos ajustando la posición del barco por fracciones de píxel, necesitamos asignar la posición a una variable a la que se le puede asignar un flotador.

Puedes usar un flotante para establecer un atributo de un rect, pero rect solo mantendrá la parte entera de ese valor. Para realizar un seguimiento preciso de la posición del barco, definimos un nuevo self.x 2. Usamos la función float() para convertir el valor de self.rect.x en un float y asignamos este valor a self.x.

Ahora, cuando cambiamos la posición del barco en update(), el valor de self.x se ajusta según la cantidad almacenada en settings.ship_speed 3. Después de self.x se ha actualizado, utilizamos el nuevo valor para actualizar self.rect.x, que controla la posición de la nave 4. Sólo la parte entera de self.x se asignará a self.rect.x, pero eso está bien para mostrando el barco.

Ahora podemos cambiar el valor de ship_speed y cualquier valor mayor que 1 hará que el barco se mueva más rápido. Esto ayudará a que la nave responda lo suficientemente rápido como para derribar alienígenas y nos permitirá cambiar el ritmo del juego a medida que el jugador avanza en el juego.

Limitar el alcance del barco

En este punto, el barco desaparecerá de cualquier borde de la pantalla si mantienes presionada la tecla de flecha el tiempo suficiente. Corrijamos esto para que la nave deje de moverse cuando llegue al borde de la pantalla. Hacemos esto modificando el método update() en Ship:

```

nave.py      actualización def (auto):
              """Actualiza la posición del barco según las banderas de movimiento."""
              # Actualiza el valor x del barco, no el rect.
1             si self.moving_right y self.rect.right < self.screen_rect.right:
                self.x += self.settings.ship_speed
2             si self.moving_left y self.rect.left > 0:
                self.x -= self.settings.ship_speed

              # Actualizar objeto rect desde self.x.
              self.rect.x = self.x

```

Este código verifica la posición del barco antes de cambiar el valor de self.x. El código self.rect.right devuelve la coordenada x del borde derecho del recto del barco. Si este valor es menor que el valor devuelto por self.screen

_rect.right, el barco no ha llegado al borde derecho de la pantalla 1. Lo mismo ocurre con el borde izquierdo: si el valor del lado izquierdo del rect es mayor que 0, el barco no ha llegado al borde izquierdo de la pantalla 2. Esto asegura que el barco esté dentro de estos límites antes de ajustar el valor de self.x.

Cuando ejecutas `alien_invasion.py` ahora, la nave debería dejar de moverse en cualquier borde de la pantalla. Esto está muy bien; Todo lo que hemos hecho es agregar una prueba condicional en una declaración `if`, ¡pero se siente como si la nave golpeará una pared o un campo de fuerza en cualquier borde de la pantalla!

Refactorización `_check_events()`

El método `_check_events()` aumentará en longitud a medida que avancemos en el desarrollo del juego, así que dividamos `_check_events()` en dos métodos separados: uno que maneja eventos `KEYDOWN` y otro que maneja eventos `KEYUP`:

```

invasión_alienígena.py    def _check_events(yo):
                           """Responder a pulsaciones de teclas y eventos del mouse."""
                           para evento en pygame.event.get():
                               si evento.tipo == pygame.SALIR:
                                   sys.salir()
                               elif evento.tipo == pygame.KEYDOWN:
                                   self._check_keydown_events(evento)
                               elif evento.tipo == pygame.KEYUP:
                                   self._check_keyup_events(evento)

                           def _check_keydown_events(yo, evento):
                               """Responder a las pulsaciones de teclas."""
                               si evento.key == pygame.K_RIGHT:
                                   self.ship.moving_right = Verdadero
                               elif evento.key == pygame.K_LEFT:
                                   self.ship.moving_left = Verdadero

                           def _check_keyup_events(yo, evento):
                               """Responder a comunicados clave."""
                               si evento.key == pygame.K_RIGHT:
                                   self.ship.moving_right = Falso
                               elif evento.key == pygame.K_LEFT:
                                   self.ship.moving_left = Falso

```

Creemos dos nuevos métodos auxiliares: `_check_keydown_events()` y `_check_keyup_events()`. Cada uno necesita un parámetro propio y un parámetro de evento. Los cuerpos de estos dos métodos se copiaron de `_check_events()` y reemplazamos el código antiguo con llamadas a los nuevos métodos. Los `_check_events()` El método es más simple ahora con esta estructura de código más limpia, lo que facilitará el desarrollo de respuestas adicionales a las entradas de los jugadores.

Presionando Q para salir

Ahora que respondemos a las pulsaciones de teclas de manera eficiente, podemos agregar otra forma de salir del juego. Se vuelve tedioso hacer clic en la X en la parte superior de la ventana del juego para finalizar el juego cada vez que pruebas una nueva función, por lo que agregaremos un atajo de teclado para finalizar el juego cuando el jugador presiona Q:

```

invasión_alienígena.py    def _check_keydown_events(yo, evento):
                           --recorte--
                           elif evento.key == pygame.K_LEFT:

```

```

        self.ship.moving_left = Verdadero
    elif evento.key == pygame.K_q:
        sys.salir()

```

En `_check_keydown_events()`, agregamos un nuevo bloque que finaliza el juego cuando el jugador presiona Q. Ahora, durante la prueba, puedes presionar Q para cerrar el juego en lugar de usar el cursor para cerrar la ventana.

Ejecutar el juego en modo de pantalla completa

Pygame tiene un modo de pantalla completa que quizás te guste más que ejecutar el juego en una ventana normal. Algunos juegos se ven mejor en modo de pantalla completa y, en algunos sistemas, el juego puede funcionar mejor en general en modo de pantalla completa.

Para ejecutar el juego en modo de pantalla completa, realiza los siguientes cambios en `__init__()`:

```

extraterrestre
_invasion.py

def __init__(yo):
    """Inicializa el juego y crea recursos para el juego."""
    pygame.init()
    self.settings = Configuración()

1     self.screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
2     self.settings.screen_width = self.screen.get_rect().ancho
    self.settings.screen_height = self.screen.get_rect().altura
    pygame.display.set_caption("Invasión alienígena")

```

Al crear la superficie de la pantalla, pasamos un tamaño de (0, 0) y el parámetro `pygame.FULLSCREEN` 1. Esto le dice a Pygame que determine un tamaño de ventana que llenará la pantalla. Debido a que no conocemos el ancho y el alto de la pantalla de antemano, actualizamos estas configuraciones después de crear la pantalla. 2. Usamos los atributos de ancho y alto del rect de la pantalla para actualizar el objeto de configuración .

Si te gusta cómo se ve o se comporta el juego en modo de pantalla completa, mantén esta configuración. Si te gustó más el juego en su propia ventana, puedes volver al enfoque original donde establecimos un tamaño de pantalla específico para el juego.

NOTA asegúrate de poder salir presionando Q antes de ejecutar el juego en modo de pantalla completa; Pygame no ofrece una forma predeterminada de salir de un juego en modo de pantalla completa.

Un resumen rápido

En la siguiente sección, agregaremos la capacidad de disparar balas, lo que implica agregar un nuevo archivo llamado `bala.py` y realizar algunas modificaciones a algunos de los archivos que ya estamos usando. En este momento tenemos tres archivos que contienen varias clases y métodos. Para tener claro cómo está organizado el proyecto, revisemos cada uno de estos archivos antes de agregar más funciones.

invasión_alienígena.py

El archivo principal, alien_invasion.py, contiene la clase AlienInvasion . Esta clase crea una serie de atributos importantes que se utilizan a lo largo del juego: las configuraciones se asignan a la configuración, la superficie de visualización principal se asigna a la pantalla y también se crea una instancia de barco en este archivo. El bucle principal del juego, un bucle while , también se almacena en este módulo. El bucle while llama a _check_events(), ship.update() y _update_screen(). También marca el reloj en cada paso por el circuito.

El método _check_events() detecta eventos relevantes, como pulsaciones y liberaciones de teclas, y procesa cada uno de estos tipos de eventos a través de los métodos _check_keydown_events() y _check_keyup_events(). Por ahora, estos métodos gestionan el movimiento del barco. La clase AlienInvasion también contiene _update_screen(), que vuelve a dibujar la pantalla en cada paso por el bucle principal.

El archivo alien_invasion.py es el único archivo que necesita ejecutar cuando lo desee para jugar invasión alienígena. Los otros archivos, settings.py y ship.py, contienen código que se importa a este archivo.

configuración.py

El archivo settings.py contiene la clase Configuración . Esta clase sólo tiene un __init__() método, que inicializa los atributos que controlan la apariencia del juego y la velocidad del barco.

nave.py

El archivo ship.py contiene la clase Ship . La clase Ship tiene un método __init__() , un método update() para gestionar la posición del barco y un método blitme() para dibujar el barco en la pantalla. La imagen del barco se almacena en ship.bmp, que se encuentra en la carpeta de imágenes .

INTÉNTALO TÚ MISMO

12-3. Documentación de Pygame: ya hemos avanzado lo suficiente en el juego como para que desees consultar parte de la documentación de Pygame. La página de inicio de Pygame está en <https://pygame.org> y la página de inicio de la documentación está en <https://pygame.org/docs>. Simplemente lea la documentación por ahora. No lo necesitarás para completar este proyecto, pero te ayudará si luego quieres modificar Alien Invasion o crear tu propio juego.

12-4. Cohete: crea un juego que comience con un cohete en el centro de la pantalla. Permite que el jugador mueva el cohete hacia arriba, abajo, izquierda o derecha usando las cuatro teclas de flecha. Asegúrate de que el cohete nunca se mueva más allá de ningún borde de la pantalla.

12-5. Claves: crea un archivo Pygame que crea una pantalla vacía. En el bucle de eventos, imprima el atributo event.key cada vez que se detecte un evento pygame.KEYDOWN . Ejecute el programa y presione varias teclas para ver cómo responde Pygame.

Disparar balas

Ahora agreguemos la capacidad de disparar balas. Escribiremos un código que dispare una bala, representada por un pequeño rectángulo, cuando el jugador presione la barra espaciadora. Luego, las balas viajarán hacia arriba en la pantalla hasta que desaparezcan de la parte superior de la pantalla.

Agregar la configuración de viñetas

Al final del método `__init__()`, actualizaremos `settings.py` para incluir los valores que necesitaremos para una nueva clase `Bullet`:

```
configuración.py def __init__(yo):
    --recorte--
    # Configuración de viñetas
    self.bullet_speed = 2.0
    self.bullet_width = 3
    self.bullet_height = 15
    self.bullet_color = (60, 60, 60)
```

Estas configuraciones crean viñetas de color gris oscuro con un ancho de 3 píxeles y un altura de 15 píxeles. Las balas viajarán un poco más rápido que el barco.

Creando la clase Bullet

Ahora cree un archivo `bala.py` para almacenar nuestra clase `Bullet`. Aquí está la primera parte de `Bullet.py`:

```
bala.py importar pygame
desde pygame.sprite importar Sprite

clase Bala (Sprite):
    """Una clase para gestionar las balas disparadas desde el barco."""

    def __init__(self, ai_game):
        """Crea un objeto bala en la posición actual del barco."""
        super().__init__()
        self.pantalla = ai_game.pantalla
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # Cree una viñeta recta en (0, 0) y luego establezca la posición correcta.
1         self.rect = pygame.Rect(0, 0, self.settings.bullet_width,
2             self.settings.bullet_height)
3         self.rect.midtop = ai_game.ship.rect.midtop

        # Almacena la posición de la bala como un flotador.
        self.y = flotante(self.rect.y)
```

La clase `Bullet` hereda de `Sprite`, que importamos del `pygame`. módulo `sprite`. Cuando usas sprites, puedes agrupar elementos relacionados en tu juego y actuar sobre todos los elementos agrupados a la vez. Para crear una instancia de viñeta, `__init__()` necesita la instancia actual de `AlienInvasion`, y llamamos

`super()` para heredar correctamente de `Sprite`. También configuramos atributos para los objetos de pantalla y configuración, y para el color de la viñeta.

A continuación creamos el atributo `rect` 1 de la viñeta. La viñeta no se basa en un imagen, por lo que tenemos que construir un `rect` desde cero usando la clase `pygame.Rect()`. Esta clase requiere las coordenadas `x` e `y` de la esquina superior izquierda del rectángulo, y el ancho y alto del rectángulo. Inicializamos el `rect` en `(0, 0)`, pero lo moveremos a la ubicación correcta en la siguiente línea, porque la posición de la bala depende de la posición del barco. Obtenemos el ancho y el alto de la viñeta a partir de los valores almacenados en `self.settings`.

Configuramos el atributo `midtop` de la bala para que coincida con el atributo `midtop` 2 del barco. Esto hará que la bala emerja desde la parte superior del barco, haciendo que parezca que la bala fue disparada desde el barco. Usamos un flotador para la coordenada `y` de la bala para poder hacer ajustes finos a la velocidad de la bala 3.

Aquí está la segunda parte de `bullet.py`, `update()` y `draw_bullet()`:

```
bala.py      actualización def (auto):
              """Mueve la bala hacia arriba en la pantalla."""
              # Actualiza la posición exacta de la bala.
1             self.y -= self.settings.bullet_speed
              # Actualiza la posición recta.
2             self.rect.y = self.y

              def draw_bullet(yo):
              """Dibuja la bala hacia la pantalla."""
3             pygame.draw.rect(self.pantalla, self.color, self.rect)
```

El método `update()` gestiona la posición de la bala. Cuando se dispara una bala, sube en la pantalla, lo que corresponde a un valor decreciente de la coordenada `y`. Para actualizar la posición, restamos la cantidad almacenada en la configuración.

`.bullet_speed` de `self.y` 1. Luego usamos el valor de `self.y` para establecer el valor de `self.rect.y` 2.

La configuración `Bullet_speed` nos permite aumentar la velocidad de las balas a medida que avanza el juego o según sea necesario para refinar el comportamiento del juego. Una vez que se dispara una bala, nunca cambiamos el valor de su coordenada `x`, por lo que viajará verticalmente en línea recta incluso si el barco se mueve.

Cuando queremos dibujar una viñeta, llamamos a `draw_bullet()`. El sorteo `rect()` La función llena la parte de la pantalla definida por el `rect` de la viñeta con el color almacenado en `self.color` 3.

Almacenamiento de viñetas en un grupo

Ahora que tenemos una clase `Bullet` y las configuraciones necesarias definidas, podemos escribir código para disparar una bala cada vez que el jugador presiona la barra espaciadora. Crearemos un grupo en `AlienInvasion` para almacenar todas las balas activas para que podamos administrar las balas que ya se han disparado. Este grupo será una instancia de la clase `pygame.sprite.Group`, que se comporta como una lista con algunas funciones adicionales que son útiles al crear juegos. Usaremos este grupo para dibujar viñetas en la pantalla en cada paso por el bucle principal y para actualizar la posición de cada viñeta.

Primero, importaremos la nueva clase Bullet :

```

invasión_alienígena.py --snip--
desde barco importación Barco
de viñeta importar viñeta

```

A continuación crearemos el grupo que contiene las viñetas en `__init__()`:

```

invasión_alienígena.py

def __init__(yo):
    --recorte--
    self.ship = Enviar (yo)
    self.bullets = pygame.sprite.Group()

```

Luego necesitamos actualizar la posición de las viñetas en cada paso por el bucle `while` :

```

invasión_alienígena.py

def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:
        self._check_events()
        auto.envío.actualización()
        self.bullets.actualización()
        self._update_screen()
        reloj.automático.tick(60)

```

Cuando llamas a `update()` en un grupo, el grupo llama automáticamente a `update()` para cada objeto del grupo. La línea `self.bullets.update()` llama a `bullet.update()` para cada viñeta que colocamos en el grupo de viñetas.

Disparando balas

En `AlienInvasion`, necesitamos modificar `_check_keydown_events()` para disparar una bala cuando el jugador presiona la barra espaciadora. No necesitamos cambiar `_check_keyup_events()` porque no pasa nada cuando se suelta la barra espaciadora. También necesitamos modificar `_update_screen()` para asegurarnos de que cada viñeta se dibuje en la pantalla antes de llamar a `flip()`.

Habrá un poco de trabajo que hacer cuando disparemos una bala, así que escribamos un nuevo método, `_fire_bullet()`, para manejar este trabajo:

```

extraterrestre
_invasion.py

def _check_keydown_events(yo, evento):
    --recorte--
    elif evento.key == pygame.K_q:
        sys.salir()
1    elif event.key == pygame.K_SPACE:
        self._fire_bullet()

def _check_keyup_events(yo, evento):
    --recorte--

def _fire_bullet(yo):
2    """Cree una nueva viñeta y agréguela al grupo de viñetas."""
3    new_bullet = Viñeta(yo)
    self.bullets.add(new_bullet)

```

```

def _update_screen(auto):
    """Actualiza imágenes en la pantalla y cambia a la nueva pantalla."""
    self.screen.fill(self.settings.bg_color)
4   para viñeta en self.bullets.sprites():
        bala.draw_bullet()
    auto.envío.blitme()

    pygame.display.flip()
--recorte--

```

Llamamos a `_fire_bullet()` cuando se presiona la barra espaciadora 1. En `_fire_bullet()`, creamos una instancia de `Bullet` y la llamamos `new_bullet` 2. Luego lo agregamos al grupo de viñetas usando el método `add()` 3. El método `add()` es similar a `append()`, pero está escrito específicamente para grupos de Pygame.

El método `bullets.sprites()` devuelve una lista de todos los sprites del grupo de viñetas. Para dibujar todas las balas disparadas en la pantalla, recorremos los sprites en balas y llamamos a `draw_bullet()` en cada una 4. Colocamos este bucle antes de la línea que dibuja el barco, para que las balas no comiencen encima de la barco.

Cuando ejecutes `alien_invasion.py` ahora, deberías poder mover la nave hacia la derecha y hacia la izquierda y disparar tantas balas como quieras. Las balas suben por la pantalla y desaparecen cuando llegan a la cima, como se muestra en la Figura 12-3. Puede modificar el tamaño, el color y la velocidad de las viñetas en `settings.py`.

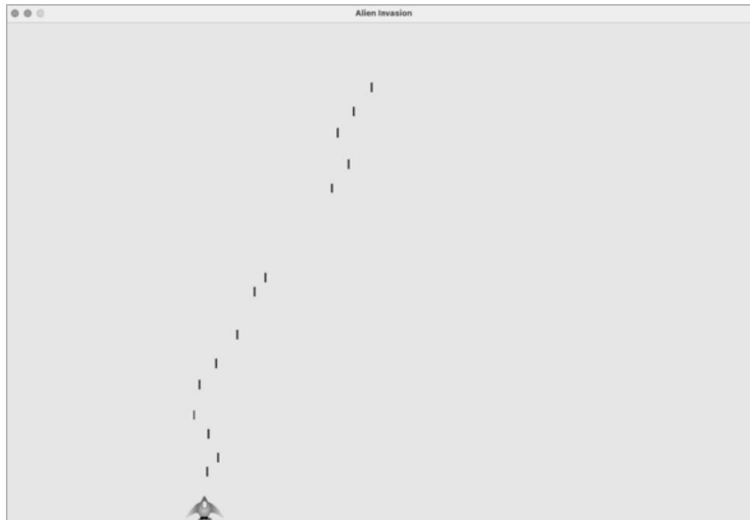


Figura 12-3: El barco después de disparar una serie de balas.

Eliminar viñetas antiguas

Por el momento, las balas desaparecen cuando llegan a la parte superior, pero sólo porque Pygame no puede dibujarlas por encima de la parte superior de la pantalla. En realidad, las balas siguen existiendo; sus valores de coordenadas y se vuelven cada vez más negativos. Esto es un problema porque continúan consumiendo memoria y potencia de procesamiento.

Necesitamos deshacernos de estas viejas balas, o el juego se ralentizará por hacer tanto trabajo innecesario. Para hacer esto, necesitamos detectar cuando el valor inferior del rect de una viñeta tiene un valor de 0, lo que indica que la viñeta ha pasado por la parte superior de la pantalla:

```

extraterrestre
_invasion.py

def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:

        self._check_events()
        auto.envío.actualización()
        self.bullets.actualización()

        # Deshazte de las balas que han desaparecido.
1      para viñeta en self.bullets.copy():
2          si bala.rect.bottom <= 0:
3              auto.balas.eliminar(bala)
4          imprimir(len(self.bullets))

        self._update_screen()
        reloj.automático.tick(60)

```

Cuando usa un bucle for con una lista (o un grupo en Pygame), Python espera que la lista mantenga la misma longitud mientras se ejecute el bucle. Eso significa que no puedes eliminar elementos de una lista o grupo dentro de un bucle for, por lo que tenemos que recorrer una copia del grupo. Usamos el método copy() para configurar el bucle for 1, lo que nos deja libres para modificar el grupo de viñetas original dentro del bucle. Verificamos cada viñeta para ver si ha desaparecido de la parte superior de la pantalla 2. Si es así, la eliminamos de las viñetas 3. Insertamos una llamada print() para mostrar cuántas viñetas existen actualmente en el juego y verificar que Se eliminan cuando llegan a la parte superior de la pantalla 4.

Si este código funciona correctamente, podemos observar la salida del terminal mientras dispara balas y ver que el número de balas disminuye a cero después de que cada serie de balas haya pasado la parte superior de la pantalla. Después de ejecutar el juego y verificar que las viñetas se eliminan correctamente, elimine la llamada print(). Si lo dejas activado, el juego se ralentizará significativamente porque lleva más tiempo escribir la salida en el terminal que dibujar gráficos en la ventana del juego.

Limitar el número de balas

Muchos juegos de disparos limitan la cantidad de balas que un jugador puede tener en la pantalla al mismo tiempo; hacerlo anima a los jugadores a disparar con precisión. Haremos lo mismo en Alien Invasion.

Primero, almacene la cantidad de viñetas permitidas en settings.py:

```

configuración.py

# Configuración de viñetas
--recorte--
self.bullet_color = (60, 60, 60)
self.bullets_allowed = 3

```

Esto limita al jugador a tres balas a la vez. Usaremos esta configuración en AlienInvasion para comprobar cuántas balas existen antes de crear una nueva bala en `_fire_bullet()`:

invasión_alienígena.py

```
def _fire_bullet(yo):
    """Cree una nueva viñeta y agréguela al grupo de viñetas."""
    si len(self.bullets) < self.settings.bullets_allowed:
        new_bullet = Viñeta(yo)
        self.bullets.add(new_bullet)
```

Cuando el jugador presiona la barra espaciadora, verificamos la longitud de las balas. Si `len(self.bullets)` es menor que tres, creamos una nueva viñeta. Pero si ya hay tres balas activas, no pasa nada cuando se presiona la barra espaciadora. Cuando ejecutes el juego ahora, solo deberías poder disparar balas en grupos de tres.

Creando el método `_update_bullets()`

Queremos mantener la clase AlienInvasion razonablemente bien organizada, por lo que ahora que hemos escrito y verificado el código de administración de viñetas, podemos moverlo a un método separado. Crearemos un nuevo método llamado `_update_bullets()` y agréguelo justo antes de `_update_screen()`:

invasión_alienígena.py

```
def _update_bullets(yo):
    """Actualiza la posición de las balas y elimina las balas viejas."""
    # Actualizar las posiciones de las viñetas.
    self.bullets.actualización()

    # Deshazte de las balas que han desaparecido.
    para viñeta en self.bullets.copy():
        si bala.rect.bottom <= 0:
            auto.balas.eliminar(bala)
```

El código para `_update_bullets()` se corta y pega desde `run_game()`; Todo lo que hemos hecho aquí es aclarar los comentarios.

El bucle `while` en `run_game()` vuelve a parecer simple:

invasión_alienígena.py

```
mientras que Verdadero:
    self._check_events()
    auto.envío.actualización()
    self._update_bullets()
    self._update_screen()
    reloj.automático.tick(60)
```

Ahora nuestro bucle principal contiene sólo un código mínimo, por lo que podemos leer rápidamente los nombres de los métodos y comprender lo que está sucediendo en el juego. El bucle principal verifica la entrada del jugador y luego actualiza la posición del barco y las balas que se han disparado. Luego usamos las posiciones actualizadas para dibujar una nueva pantalla y marcar el reloj al final de cada paso por el bucle.

Ejecute `alien_invasion.py` una vez más y asegúrese de poder disparar balas sin errores.

INTÉNTALO TÚ MISMO

12-6. Sideways Shooter: escribe un juego que coloque un barco en el lado izquierdo de la pantalla y permita al jugador mover el barco hacia arriba y hacia abajo. Haz que la nave dispare una bala que recorra la pantalla cuando el jugador presione la barra espaciadora. Asegúrese de que las viñetas se eliminen una vez que desaparezcan de la pantalla.

Resumen

En este capítulo, aprendiste a hacer un plan para un juego y aprendiste la estructura básica de un juego escrito en Pygame. Aprendió a establecer un color de fondo y a almacenar la configuración en una clase separada donde puede ajustarla más fácilmente. Viste cómo dibujar una imagen en la pantalla y darle al jugador control sobre el movimiento de los elementos del juego. Creaste elementos que se mueven por sí solos, como balas que vuelan por una pantalla, y eliminaste objetos que ya no son necesarios. También aprendió a refactorizar el código de un proyecto de forma regular para facilitar el desarrollo continuo.

En el Capítulo 13, agregaremos extraterrestres a Alien Invasion. Al final del capítulo, podrás derribar alienígenas, ¡con suerte antes de que lleguen a tu nave!

13

¡EXTRANJEROS!



En este capítulo, agregaremos extraterrestres a Alien Invasion. Agregaremos un alienígena cerca de la parte superior de la pantalla y luego generaremos una flota completa de alienígenas. Haremos que la flota avance hacia los lados y hacia abajo, y nos desharemos de los alienígenas alcanzados por una bala. Finalmente, limitaremos la cantidad de barcos que tiene un jugador y finalizaremos el juego cuando el jugador se quede sin barcos.

A medida que avance en este capítulo, aprenderá más sobre Pygame y sobre la gestión de un proyecto grande. También aprenderás a detectar colisiones entre objetos del juego, como balas y extraterrestres. Detectar colisiones te ayuda a definir las interacciones entre elementos de tus juegos. Por ejemplo, puedes confinar a un personaje dentro de las paredes de un laberinto o pasar una pelota entre dos personajes. Continuaremos trabajando a partir de un plan que revisamos ocasionalmente para mantener el enfoque de nuestras sesiones de escritura de códigos.

Antes de comenzar a escribir código nuevo para agregar una flota de extraterrestres a la pantalla, veamos el proyecto y actualicemos nuestro plan.

Revisando el proyecto

Cuando comienzas una nueva fase de desarrollo en un proyecto grande, siempre es una buena idea revisar tu plan y aclarar lo que quieres lograr con el código que estás a punto de escribir. En este capítulo, haremos lo siguiente:

- Agregue un solo alienígena a la esquina superior izquierda de la pantalla, con el espacio apropiado alrededor.
- Llena la parte superior de la pantalla con tantos alienígenas como podamos que quepan horizontalmente. Luego crearemos filas adicionales de alienígenas hasta que tengamos una flota completa.
- Haz que la flota se mueva hacia los lados y hacia abajo hasta que toda la flota sea derribada, un alienígena golpee la nave o un alienígena llegue al suelo. Si derriban toda la flota, crearemos una nueva flota. Si un extraterrestre golpea la nave o el suelo, destruiremos la nave y crearemos una nueva flota.
- Limita el número de barcos que el jugador puede usar y finaliza el juego cuando el jugador haya agotado el número de barcos asignado.

Refinaremos este plan a medida que implementemos funciones, pero esto es lo suficientemente específico como para comenzar a escribir código.

También debes revisar tu código existente cuando comiences a trabajar en una nueva serie de funciones en un proyecto. Debido a que cada nueva fase normalmente hace que un proyecto sea más complejo, es mejor limpiar cualquier código desordenado o ineficiente. Hemos estado refactorizando a medida que avanzamos, por lo que no hay ningún código que debamos refactorizar en este momento.

Creando el primer extraterrestre

Colocar un extraterrestre en la pantalla es como colocar un barco en la pantalla. El comportamiento de cada alienígena está controlado por una clase llamada Alien, que estructuraremos como la clase Ship . Continuaremos usando imágenes de mapa de bits para simplificar. Puedes encontrar tu propia imagen de un extraterrestre o usar la que se muestra en la Figura 13-1, que está disponible en los recursos del libro en https://ehmatthes.github.io/pcc_3e. Esta imagen tiene un fondo gris, que coincide con el color de fondo de la pantalla. Asegúrese de guardar el archivo de imagen que elija en la carpeta de imágenes .



Figura 13-1: El alienígena que usaremos para construir la flota

Creando la clase alienígena

Ahora escribiremos la clase Alien y la guardaremos como alien.py:

```
alien.py importar pygame
desde pygame.sprite importar Sprite

clase alienígena (Sprite):
    """Una clase para representar a un solo alienígena en la flota."""

    def __init__(self, ai_game):
        """Inicializa el alienígena y establece su posición inicial."""
        super().__init__()
        self.pantalla = ai_game.pantalla

        # Cargue la imagen alienígena y establezca su atributo rect.
        self.image = pygame.image.load("imágenes/alien.bmp")
        self.rect = self.image.get_rect()

        # Inicie cada nuevo alienígena cerca de la parte superior izquierda de la pantalla.
1        self.rect.x = self.rect.ancho
        self.rect.y = self.rect.altura

        # Almacene la posición horizontal exacta del extraterrestre.
2        self.x = flotante(self.rect.x)
```

La mayor parte de esta clase es como la clase Barco, excepto por la ubicación del extraterrestre en la pantalla. Inicialmente colocamos a cada alienígena cerca de la esquina superior izquierda de la pantalla; Agregamos un espacio a la izquierda que es igual al ancho del alienígena y un espacio encima igual a su altura 1, para que sea fácil de ver. Nos preocupa principalmente la velocidad horizontal de los alienígenas, por lo que rastreamos la posición horizontal de cada alienígena precisamente 2.

Esta clase Alien no necesita un método para dibujarla en la pantalla; en su lugar, usaremos un método de grupo de Pygame que dibuja automáticamente todos los elementos de un grupo en la pantalla.

Creando una instancia del alienígena

Queremos crear una instancia de Alien para poder ver el primer alienígena en la pantalla. Debido a que es parte de nuestro trabajo de configuración, agregaremos el código para esta instancia al final del método `__init__()` en `AlienInvasion`. Con el tiempo, crearemos una flota completa de alienígenas, lo que supondrá bastante trabajo, por lo que crearemos un nuevo método auxiliar llamado `_create_fleet()`.

El orden de los métodos en una clase no importa, siempre y cuando haya cierta coherencia en su ubicación. Colocaré `_create_fleet()` justo antes del método `_update_screen()`, pero cualquier lugar de `AlienInvasion` funcionará. Primero, importaremos la clase Alien.

Aquí están las declaraciones de importación actualizadas para `alien_invasion.py`:

```
invasión_alienígena.py --snip--
de viñeta importar viñeta
de importación extranjera Alien
```

Y aquí está el método `__init__()` actualizado:

```
invación_alienígena.py    def __init__(yo):
                           --recorte--
                           self.ship = Enviar (yo)
                           self.bullets = pygame.sprite.Group()
                           self.aliens = pygame.sprite.Group()

                           self._create_fleet()
```

Creamos un grupo para contener la flota de alienígenas y llamamos `_create_fleet()`, que estamos a punto de escribir.

Aquí está el nuevo método `_create_fleet()` :

```
invación_alienígena.py    def _create_fleet(yo):
                           """Crea la flota de alienígenas."""
                           # Haz un extraterrestre.

                           extraterrestre = extraterrestre (yo)
                           self.aliens.add(alienígena)
```

En este método, creamos una instancia de Alien y luego la agregamos al grupo que contendrá la flota. El extraterrestre se colocará en el área superior izquierda predeterminada de la pantalla.

Para que aparezca el alienígena, necesitamos llamar al método `draw()` del grupo en `_update_screen()`:

```
invación_alienígena.py    def _update_screen(auto):
                           --recorte--
                           auto.envío.blitme()
                           self.aliens.draw(auto.pantalla)

                           pygame.display.flip()
```

Cuando llamas a `draw()` en un grupo, Pygame dibuja cada elemento del grupo en la posición definida por su atributo `rect`. El método `draw()` requiere un argumento: una superficie sobre la cual dibujar los elementos del grupo. La Figura 13-2 muestra el primer extraterrestre en la pantalla.

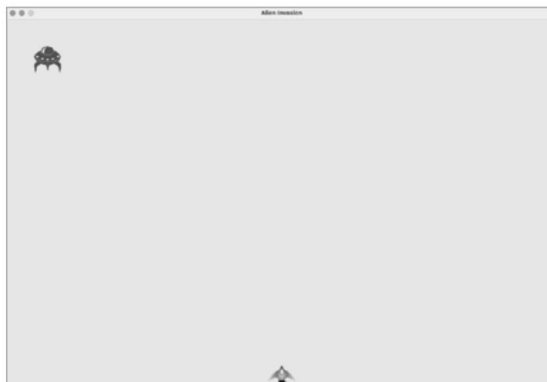


Figura 13-2: Aparece el primer extraterrestre.

Ahora que el primer alienígena aparece correctamente, escribiremos el código para dibujar una flota completa.

Construyendo la flota alienígena

Para dibujar una flota, necesitamos descubrir cómo llenar la parte superior de la pantalla con alienígenas, sin saturar la ventana del juego. Hay varias maneras de lograr este objetivo. Lo abordaremos agregando extraterrestres en la parte superior de la pantalla, hasta que no quede espacio para un nuevo extraterrestre. Luego repetiremos este proceso, siempre que tengamos suficiente espacio vertical para agregar una nueva fila.

Creando una fila de extraterrestres

Ahora estamos listos para generar una fila completa de alienígenas. Para hacer una fila completa, primero crearemos un solo alienígena para tener acceso al ancho del alienígena. Colocaremos un extraterrestre en el lado izquierdo de la pantalla y luego seguiremos agregando extraterrestres hasta que nos quedemos sin espacio:

```

extraterrestre
_invasion.py

def _create_fleet(yo):
    """Crea la flota de alienígenas."""
    # Crea un extraterrestre y sigue agregando extraterrestres hasta que no quede espacio.
    # El espacio entre alienígenas es un ancho de alienígena.
    extraterrestre = extraterrestre(yo)
    alien_width = alien.rect.ancho

1    actual_x = ancho_extraterrestre
2    mientras que current_x < (self.settings.screen_width - 2 * alien_width):
3        new_alien = alienígena(yo)
4        nuevo_alien.x = actual_x
        nuevo_alien.rect.x = actual_x
        self.aliens.add(nuevo_alienígena)
5        actual_x += 2 * ancho_extraterrestre

```

Obtenemos el ancho del alienígena del primer alienígena que creamos y luego definimos una variable llamada `current_x` 1. Esto hace referencia a la posición horizontal del siguiente extraterrestre que pretendemos colocar en la pantalla. Inicialmente configuramos esto en un ancho alienígena, para desplazar al primer alienígena de la flota desde el borde izquierdo de la pantalla.

A continuación, comenzamos el ciclo `while` 2; Seguiremos agregando extraterrestres mientras haya suficiente espacio para colocar uno. Para determinar si hay espacio para colocar otro alienígena, compararemos `current_x` con algún valor máximo. Un primer intento de definir este bucle podría verse así:

```
mientras que current_x < self.settings.screen_width:
```

Parece que esto podría funcionar, pero colocaría al último extraterrestre en la fila en el extremo derecho de la pantalla. Entonces agregamos un pequeño margen en el lado derecho de la pantalla. Siempre que haya al menos dos anchos de espacio alienígena en el borde derecho de la pantalla, ingresamos al bucle y agregamos otro alienígena a la flota.

Siempre que haya suficiente espacio horizontal para continuar el bucle, queremos hacer dos cosas: crear un extraterrestre en la posición correcta y definir la posición horizontal del siguiente extraterrestre en la fila. Creamos un alienígena y lo asignamos a `new_alien` 3. Luego establecemos la posición horizontal precisa en el valor actual de `current_x` 4. También posicionamos el recto del alienígena en este mismo valor `x` y agregamos el nuevo alienígena al grupo `self.aliens` .

Finalmente, incrementamos el valor de `current_x` 5. Agregamos dos anchos de alienígenas a la posición horizontal, para pasar el alienígena que acabamos de agregar y dejar algo de espacio entre los alienígenas también. Python reevaluará la condición al inicio del ciclo `while` y decidirá si hay espacio para otro extraterrestre. Cuando no quede espacio, el bucle terminará y deberíamos tener una fila completa de alienígenas.

Cuando ejecute Alien Invasion ahora, debería ver la primera fila de alienígenas. aparecen, como en la Figura 13-3.



Figura 13-3: La primera fila de extraterrestres

NOTA No siempre es obvio exactamente cómo construir un bucle como el que se muestra en esta sección. Una cosa buena de la programación es que sus ideas iniciales sobre cómo abordar un problema como este no tienen por qué ser correctas. Es perfectamente razonable iniciar un bucle como este con los alienígenas colocados demasiado a la derecha y luego modificar el bucle hasta que tengas una cantidad adecuada de espacio en la pantalla.

Refactorización `_create_fleet()`

Si el código que hemos escrito hasta ahora fuera todo lo que necesitábamos para crear una flota, probablemente dejaríamos `_create_fleet()` como está. Pero tenemos más trabajo por hacer, así que limpiemos un poco el método. Agregaremos un nuevo método auxiliar, `_create_alien()`, y lo llamaremos desde `_create_fleet()`:

```
extraterrestre
_invasion.py

def _create_fleet(yo):
    --recorte--
    mientras que current_x < (self.settings.screen_width - 2 * alien_width):
        self._create_alien(actual_x)
        actual_x += 2 * ancho_extraterrestre
```

```

1 def _create_alien(yo, posición_x):
    """Crea un extraterrestre y colócalo en la fila."""
    new_alien = alienígena(yo)
    nuevo_alienígena.x = posición_x
    new_alien.rect.x = posición_x
    self.aliens.add(nuevo_alienígena)

```

El método `_create_alien()` requiere un parámetro además de `self`: el valor `x` que especifica dónde se debe colocar el extraterrestre 1. El código en el cuerpo de `_create_alien()` es el mismo código que estaba en `_create_fleet()`, excepto que usamos el nombre del parámetro `x_position` en lugar de `current_x`. Esta refactorización facilitará la adición de nuevas filas y la creación de una flota completa.

Agregar filas

Para terminar la flota, seguiremos agregando más filas hasta que nos quedemos sin espacio. Usaremos un bucle anidado; envolveremos otro bucle `while` alrededor del actual. El bucle interno colocará a los extraterrestres horizontalmente en una fila enfocándose en los valores `x` de los extraterrestres. El bucle exterior colocará a los extraterrestres verticalmente centrándose en los valores de `y`. Dejaremos de agregar filas cuando nos acerquemos a la parte inferior de la pantalla, dejando suficiente espacio para la nave y algo de espacio para comenzar a disparar a los alienígenas.

Aquí se explica cómo anidar los dos bucles `while` en `_create_fleet()`:

```

def _create_fleet(yo):
    """Crea la flota de alienígenas."""
    # Crea un extraterrestre y sigue agregando extraterrestres hasta que no quede espacio.
    # El espacio entre alienígenas es un ancho alienígena y una altura alienígena.
    extraterrestre = extraterrestre(yo)
1    alien_width, alien_height = alien.rect.size
2
3    current_x, current_y = ancho_extraterrestre, altura_extraterrestre
4    mientras que current_y < (self.settings.screen_height - 3 * alien_height):
        mientras que current_x < (self.settings.screen_width - 2 * alien_width):
            self._create_alien(actual_x, actual_y)
            actual_x += 2 * ancho_extraterrestre
5
        # Terminó una fila; restablecer el valor x e incrementar el valor y.
        actual_x = ancho_extraterrestre
        actual_y += 2 * altura_alienígena

```

Necesitaremos saber la altura del alienígena para poder colocar filas, así que tomamos el ancho y alto del alienígena usando el atributo de tamaño de un rectángulo alienígena 1. El atributo de tamaño de un rectángulo es una tupla que contiene su ancho y alto.

A continuación, establecemos los valores `x` e `y` iniciales para la ubicación del primer alienígena. en la flota 2. Lo colocamos a un ancho alienígena desde la izquierda y a una altura alienígena hacia abajo desde la parte superior. Luego definimos el bucle `while` que controla cuántas filas se colocan en la pantalla 3. Mientras el valor `y` para la siguiente fila sea menor que la altura de la pantalla, menos tres alturas alienígenas, seguiremos agregando filas. (Si esto no deja la cantidad adecuada de espacio, podemos ajustarlo más tarde).

Llamamos a `_create_alien()` y le pasamos el valor y así como su posición x 4.

Modificaremos `_create_alien()` en un momento.

Observe la sangría de las dos últimas líneas del código 5. Están dentro del bucle `while` externo, pero fuera del bucle `while` interno. Este bloque se ejecuta una vez finalizado el bucle interno; se ejecuta una vez después de crear cada fila. Después de agregar cada fila, restablecemos el valor de `current_x` para que el primer alienígena de la siguiente fila se coloque en la misma posición que el primer alienígena de las filas anteriores. Luego agregamos dos alturas alienígenas al valor actual de `current_y`, por lo que la siguiente fila se colocará más abajo en la pantalla. La sangría es realmente importante aquí; si no ves la flota correcta cuando ejecutas `alien_invasion.py`

Al final de esta sección, verifique la sangría de todas las líneas en estos bucles anidados.

Necesitamos modificar `_create_alien()` para establecer correctamente la posición vertical del extraterrestre:

```
def _create_alien(self, posición_x, posición_y):
    """Crea un alienígena y colócalo en la flota."""
    new_alien = alienígena(yo)
    nuevo_alienígena.x = posición_x
    new_alien.rect.x = posición_x
    new_alien.rect.y = posición_y
    self.aliens.add(nuevo_alienígena)
```

Modificamos la definición del método para aceptar el valor de y para el nuevo alienígena, y establecemos la posición vertical del rect en el cuerpo del método.

Cuando ejecute el juego ahora, deberías ver una flota completa de alienígenas, como se muestra en la Figura 13-4.

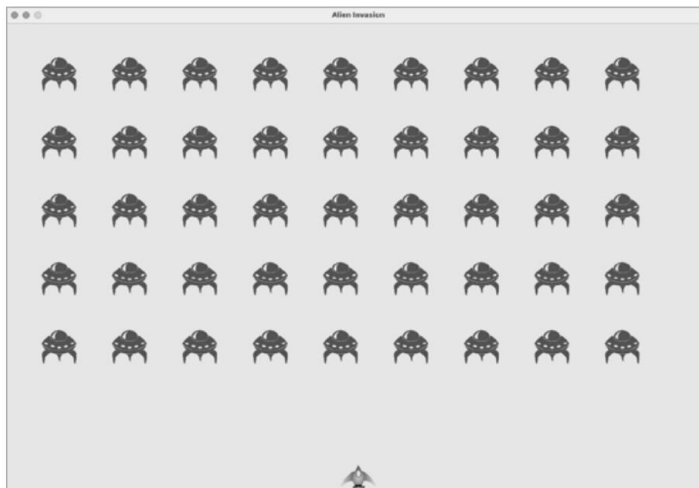


Figura 13-4: Aparece la flota completa.

¡En la siguiente sección, haremos que la flota se mueva!

INTÉNTALO TÚ MISMO

13-1. Estrellas: Encuentra una imagen de una estrella. Haz que aparezca una cuadrícula de estrellas en la pantalla.

13-2. Mejores estrellas: puedes crear un patrón de estrellas más realista introduciendo aleatoriedad al colocar cada estrella. Recuerde del Capítulo 9 que puede obtener un número aleatorio como este:

```
de randint de importación aleatoria
número_aleatorio = randint(-10, 10)
```

Este código devuelve un número entero aleatorio entre -10 y 10. Usando su código
En el Ejercicio 13-1, ajuste la posición de cada estrella en una cantidad aleatoria.

Hacer que la flota se mueva

Ahora hagamos que la flota de alienígenas se mueva hacia la derecha a través de la pantalla hasta que llegue al borde, y luego hagamos que caiga una cantidad determinada y se mueva en la otra dirección. Continuaremos este movimiento hasta que todos los alienígenas hayan sido derribados, uno choque con la nave o llegue al final de la pantalla.
Comencemos haciendo que la flota se mueva hacia la derecha.

Moviendo a los extraterrestres hacia la derecha

Para mover a los alienígenas, usaremos un método `update()` en `alien.py`, que llamaremos para cada alienígena del grupo de alienígenas. Primero, agrega una configuración para controlar la velocidad de cada alienígena:

```
configuración.py    def __init__(yo):
                    --recorte--
                    # Configuración alienígena
                    self.alien_speed = 1.0
```

Luego use esta configuración para implementar `update()` en `alien.py`:

```
extraterrestre.py  def __init__(self, ai_game):
                    """Inicializa el alienígena y establece su posición inicial."""
                    super().__init__()
                    self.pantalla = ai_game.pantalla
                    self.settings = ai_game.settings
                    --recorte--

                    actualización def (auto):
                        """Mueve al alienígena hacia la derecha."""
                        1 self.x += self.settings.alien_speed
                        2 self.rect.x = self.x
```

Creemos un parámetro de configuración en `__init__()` para que podamos acceder a la velocidad del extraterrestre en `update()`. Cada vez que actualizamos la posición de un alienígena, lo movemos a la

justo por la cantidad almacenada en `alien_speed`. Realizamos un seguimiento de la posición exacta del alienígena con el atributo `self.x`, que puede contener valores flotantes 1. Luego usamos el valor de `self.x` para actualizar la posición del rect 2 del alienígena.

En el ciclo `while` principal, ya tenemos llamadas para actualizar las posiciones del barco y de las balas. Ahora también agregaremos una llamada para actualizar la posición de cada alienígena:

`invasión_alienígena.py`

```
mientras que Verdadero:
    self._check_events()
    auto.envío.actualización()
    self._update_bullets()
    self._update_alienígena()
    self._update_screen()
    reloj.automático.tick(60)
```

Estamos a punto de escribir un código para gestionar el movimiento de la flota, por lo que creamos un nuevo método llamado `_update_alienígena()`. Actualizamos las posiciones de los alienígenas después de que se hayan actualizado las balas, porque pronto comprobaremos si alguna bala alcanzó a algún alienígena.

El lugar donde coloque este método en el módulo no es crítico. Pero para mantener el código organizado, lo colocaré justo después de `_update_bullets()` para que coincida con el orden de las llamadas a métodos en el bucle `while`. Aquí está la primera versión de `_update_alienígena()`:

`invasión_alienígena.py`

```
def _update_alienígena(yo):
    """Actualiza las posiciones de todos los alienígenas de la flota."""
    self.alienígena.actualización()
```

Usamos el método `update()` en el grupo de extraterrestres, que llama al método `update()` de cada extraterrestre. Cuando ejecutes Alien Invasion ahora, deberías ver que la flota se mueve hacia la derecha y desaparece del costado de la pantalla.

Crear configuraciones para la dirección de la flota

Ahora crearemos los ajustes que harán que la flota se mueva hacia abajo en la pantalla y hacia la izquierda cuando llegue al borde derecho de la pantalla. A continuación se explica cómo implementar este comportamiento:

`configuración.py`

```
# Configuración alienígena
self.alien_speed = 1.0
self.fleet_drop_speed = 10
# Fleet_direction de 1 representa la derecha; -1 representa la izquierda.
self.fleet_direction = 1
```

La configuración `Fleet_drop_speed` controla la rapidez con la que la flota desciende de la pantalla cada vez que un extraterrestre llega a cualquiera de los bordes. Es útil separar esta velocidad de la velocidad horizontal de los alienígenas para que puedas ajustar las dos velocidades de forma independiente.

Para implementar la configuración de dirección `_flota`, podríamos usar un valor de texto como `'izquierda'` o `'derecha'`, pero terminaríamos con declaraciones `if-elif` que prueban la dirección de la flota. En cambio, debido a que solo tenemos dos direcciones con las que lidiar, usemos los valores 1 y -1 y cambiemos entre ellos cada vez que la flota

cambia de dirección. (Usar números también tiene sentido porque moverse hacia la derecha implica sumar al valor de la coordenada x de cada extraterrestre, y moverse hacia la izquierda implica restar del valor de la coordenada x de cada extraterrestre).

Comprobar si un extraterrestre ha llegado al borde

Necesitamos un método para verificar si un extraterrestre está en cualquiera de los bordes y debemos modificar `update()` para permitir que cada extraterrestre se mueva en la dirección apropiada. Este código es parte de la clase `Alien`:

```

extraterrestre.py    def check_edges(yo):
                    """Devuelve True si el extraterrestre está en el borde de la pantalla."""
                    screen_rect = self.screen.get_rect()
1                   retorno (self.rect.right >= screen_rect.right) o (self.rect.left <= 0)

                    actualización def (auto):
                    """Mueve al alienígena hacia la derecha o hacia la izquierda."""
2                   self.x += self.settings.alien_speed * self.settings.fleet_direction
                    self.rect.x = self.x

```

Podemos llamar al nuevo método `check_edges()` en cualquier alienígena para ver si está en el borde izquierdo o derecho. El alienígena está en el borde derecho si el atributo `recto` de su recto es mayor o igual que el atributo `recto` del recto de la pantalla. Está en el borde izquierdo si su valor izquierdo es menor o igual a 0. En lugar de colocar esta prueba condicional en un bloque `if`, colocamos la prueba directamente en la declaración de retorno. Este método devolverá Verdadero si el alienígena está en el borde derecho o izquierdo, y Falso si no está en ninguno de los bordes.

Modificamos el método `update()` para permitir el movimiento hacia la izquierda o hacia la derecha multiplicando la velocidad del alienígena por el valor de `Fleet_direction`. Si `Fleet_direction` es 1, el valor de `alien_speed` se agregará a la posición actual del alienígena, moviéndolo hacia la derecha; si `Fleet_direction` es -1, el valor se restará de la posición del alienígena, moviéndolo hacia la izquierda.

Abandonar la flota y cambiar de dirección

Cuando un alienígena llega al borde, toda la flota debe descender y cambiar de dirección. Por lo tanto, necesitamos agregar algo de código a `AlienInvasion`.

porque ahí es donde comprobaremos si hay extraterrestres en el borde izquierdo o derecho. Haremos que esto suceda escribiendo los métodos `_check_fleet_edges()` y `_change_fleet_direction()`, y luego modificando `_update_aliens()`. Pondré estos nuevos métodos después de `_create_alien()`, pero nuevamente, la ubicación de estos métodos en la clase no es crítica.

```

extraterrestre
_invasion.py    def _check_fleet_edges(yo):
                """Responda apropiadamente si algún extraterrestre ha llegado a un límite."""
                para extraterrestre en self.aliens.sprites():
                si alienígena.check_edges():
2                 self._change_fleet_direction()
                romper

                def _change_fleet_direction(auto):

```

3

```

"""Soltar toda la flota y cambiar la dirección de la flota."""
para extraterrestre en self.aliens.sprites():
    alien.rect.y += self.settings.fleet_drop_speed
self.settings.fleet_direction *= -1

```

En `_check_fleet_edges()`, recorremos la flota y llamamos a `check_edges()` en cada alienígena. 1. Si `check_edges()` devuelve `True`, sabemos que un alienígena está en un borde y toda la flota necesita cambiar de dirección; entonces llamamos `_change_fleet_direction()` y salimos del bucle 2. En `_change_fleet_direction()`, recorremos todos los alienígenas y soltamos a cada uno usando la configuración `fleet_drop_speed` 3; luego cambiamos el valor de `Fleet_direction` multiplicando su valor actual por `-1`.

La línea que cambia la dirección de la flota no forma parte del bucle `for`. Queremos cambiar la posición vertical de cada alienígena, pero sólo queremos cambiar la dirección de la flota una vez.

Estos son los cambios a `_update.aliens()`:

invasión_alienígena.py

```

def _update.aliens(yo):
    """Comprueba si la flota está en un límite y luego actualiza las posiciones."""
    self._check_fleet_edges()
    self.aliens.actualización()

```

Hemos modificado el método llamando a `_check_fleet_edges()` antes actualizando la posición de cada alienígena.

Cuando ejecutas el juego ahora, la flota debería moverse hacia adelante y hacia atrás entre los bordes de la pantalla y descender cada vez que llegue a un borde.

Ahora podemos empezar a derribar alienígenas y estar atentos a cualquier alienígena que golpee la nave o llegue al final de la pantalla.

INTÉNTALO TÚ MISMO

13-3. Gotas de lluvia: busque una imagen de una gota de lluvia y cree una cuadrícula de gotas de lluvia. Haz que las gotas de lluvia caigan hacia la parte inferior de la pantalla hasta que desaparezcan.

13-4. Lluvia constante: Modifique su código en el Ejercicio 13-3 para que cuando una fila de gotas de lluvia desaparezca de la parte inferior de la pantalla, aparezca una nueva fila en la parte superior de la pantalla y comience a caer.

Disparar extraterrestres

Hemos construido nuestra nave y una flota de extraterrestres, pero cuando las balas alcanzan a los extraterrestres, simplemente pasan porque no estamos comprobando si hay colisiones. En la programación de juegos, las colisiones ocurren cuando los elementos del juego se superponen. Para hacer que las balas derriben a los extraterrestres, usaremos la función `sprite.groupcollide()` para buscar colisiones entre miembros de dos grupos.

Detección de colisiones de balas

Queremos saber de inmediato cuándo una bala alcanza a un extraterrestre para poder hacer que un extraterrestre desaparezca tan pronto como lo alcance. Para hacer esto, buscaremos colisiones inmediatamente después de actualizar la posición de todas las viñetas.

La función `sprite.groupcollide()` compara las rectas de cada elemento en un grupo con las rectas de cada elemento en otro grupo. En este caso, compara el recto de cada bala con el recto de cada alienígena y devuelve un diccionario que contiene las balas y los alienígenas que han chocado. Cada clave en el diccionario será una bala y el valor correspondiente será el alienígena que fue alcanzado. (También usaremos este diccionario cuando implementemos un sistema de puntuación en el Capítulo 14).

Agregue el siguiente código al final de `_update_bullets()` para comprobar si hay colisiones. siones entre balas y extraterrestres:

invasión_alienígena.py

```
def _update_bullets(yo):
    """Actualiza la posición de las balas y elimina las balas viejas."""
    --recorte--

    # Compruebe si hay balas que hayan alcanzado a los extraterrestres.
    # Si es así, deshazte de la bala y del extraterrestre.
    colisiones = pygame.sprite.groupcollide(
        auto.balas, auto.alienígenas, Verdadero, Verdadero)
```

El nuevo código que agregamos compara las posiciones de todas las balas en `self.bullets` y de todos los alienígenas en `self.aliens`, e identifica aquellas que se superponen.

Siempre que los rectos de una bala y un alienígena se superponen, `groupcollide()` agrega un par clave-valor al diccionario que devuelve. Los dos argumentos Verdaderos le dicen a Pygame que borre las balas y los alienígenas que han chocado. (Para crear una bala de alta potencia que pueda viajar hasta la parte superior de la pantalla, destruyendo a todos los alienígenas a su paso, puedes establecer el primer argumento booleano en Falso y mantener el segundo argumento booleano en Verdadero. Los alienígenas impactados desaparecerían, pero todas las balas permanecerían activas hasta que desaparecieran de la parte superior de la pantalla).

Cuando ejecute Alien Invasion ahora, los alienígenas que golpees deberían desaparecer. La Figura 13-5 muestra una flota que ha sido parcialmente derribada.

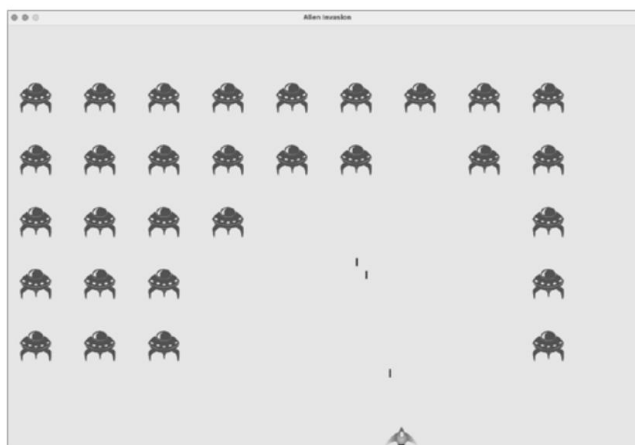


Figura 13-5: ¡Podemos disparar a los extraterrestres!

Hacer balas más grandes para realizar pruebas

Puedes probar muchas funciones de Alien Invasion simplemente ejecutando el juego, pero algunas funciones son tediosas de probar en la versión normal del juego. Por ejemplo, es mucho trabajo derribar a cada alienígena en la pantalla varias veces para probar si su código responde correctamente a una flota vacía.

Para probar funciones particulares, puedes cambiar ciertas configuraciones del juego para enfocarte en un área particular. Por ejemplo, puedes reducir la pantalla para que haya menos alienígenas a los que derribar o aumentar la velocidad de las balas y recibir muchas balas a la vez.

Mi cambio favorito para probar Alien Invasion es usar balas muy anchas que permanecen activas incluso después de haber golpeado a un alienígena (ver Figura 13-6). ¡Intenta configurar `Bullet_width` en 300, o incluso en 3000, para ver qué tan rápido puedes derribar la flota!

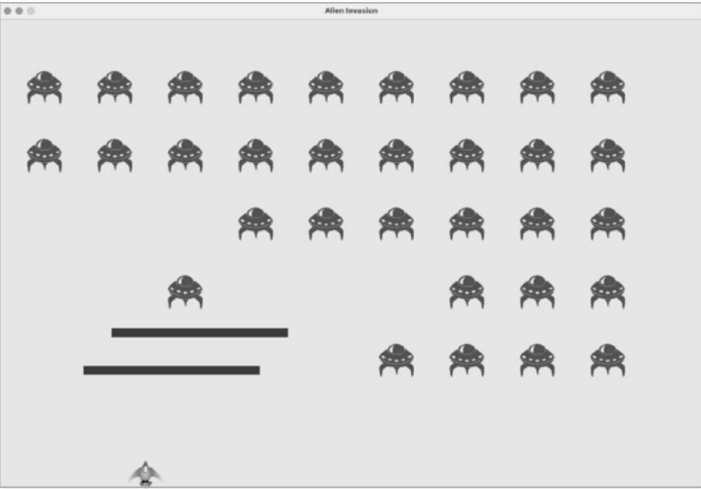


Figura 13-6: Las balas extrapoderosas hacen algunos aspectos del juego más fácil de probar.

Cambios como estos te ayudarán a probar el juego de manera más eficiente y posiblemente generarán ideas para otorgar poderes adicionales a los jugadores. Sólo recuerde restaurar la configuración a la normalidad cuando haya terminado de probar una función.

Repoblación de la flota

Una característica clave de Alien Invasion es que los alienígenas son implacables: cada vez que se destruye la flota, debería aparecer una nueva flota.

Para hacer que aparezca una nueva flota de alienígenas después de que una flota haya sido destruida, Primero comprobamos si el grupo de extraterrestres está vacío. Si es así, hacemos una llamada a `_create_fleet()`. Realizaremos esta verificación al final de `_update_bullets()`, porque ahí es donde se destruyen los extraterrestres individuales.

```
extraterrestre
_invasion.py 1
def _update_bullets(yo):
    --recorte--
    si no es uno mismo.extraterrestres:
```

2

```
# Destruye las balas existentes y crea una nueva flota.
self.balas.vacio()
self._create_fleet()
```

Comprobamos si el grupo de extraterrestres está vacío 1. Un grupo vacío se evalúa como Falso, por lo que esta es una forma sencilla de comprobar si el grupo está vacío. Si es así, nos deshacemos de las viñetas existentes usando el método `vacio()`, que elimina todos los sprites restantes de un grupo 2. También llamamos a `_create_fleet()`, que llena la pantalla con extraterrestres nuevamente.

Ahora aparece una nueva flota tan pronto como destruyes la flota actual.

Acelerando las balas

Si has intentado disparar a los alienígenas en el estado actual del juego, es posible que descubras que las balas no viajan a la mejor velocidad para el juego. Puede que sean demasiado lentos o demasiado rápidos. En este punto, puedes modificar la configuración para hacer que el juego sea más interesante. Ten en cuenta que el juego se volverá progresivamente más rápido, así que no lo hagas demasiado rápido al principio.

Modificamos la velocidad de las balas ajustando el valor de `bullet_speed` en `configuración.py`. En mi sistema, ajustaré el valor de `bullet_speed` a 2,5, para que las balas viajen un poco más rápido:

```
configuración.py      # Configuración de viñetas
                      self.bullet_speed = 2,5
                      self.bullet_width = 3
                      --recorte--
```

El mejor valor para esta configuración depende de tu experiencia en el juego, así que encuentre un valor que funcione para usted. También puedes ajustar otras configuraciones.

Refactorización `_update_bullets()`

Refactoricemos `_update_bullets()` para que no realice tantas tareas diferentes.

Moveremos el código para lidiar con colisiones entre balas y extraterrestres a un método separado:

```
invasión_alienígena.py def _update_bullets(yo):
                        --recorte--
                        # Deshazte de las balas que han desaparecido.
                        para viñeta en self.bullets.copy():
                            si bala.rect.bottom <= 0:
                                auto.balas.eliminar(bala)

                        self._check_bullet_alien_collisions()

def _check_bullet_alien_collisions(yo):
    """Responder a colisiones bala-alienígenas."""
    # Elimina las balas y los extraterrestres que hayan chocado.
    colisiones = pygame.sprite.groupcollide(
        auto.balas, auto.alienígenas, Verdadero, Verdadero)
```

```
si no es uno mismo.extraterrestres:
```

```
# Destruye las balas existentes y crea una nueva flota.
self.balas.vacio()
self._create_fleet()
```

Hemos creado un nuevo método, `_check_bullet_alien_collisions()`, para buscar colisiones entre balas y alienígenas y responder adecuadamente si toda la flota ha sido destruida. Hacerlo evita que `_update_bullets()` crezca demasiado y simplifica el desarrollo posterior.

INTÉNTALO TÚ MISMO

13-5. Tirador lateral Parte 2: Hemos recorrido un largo camino desde el Ejercicio 12-6, Tirador lateral. Para este ejercicio, intenta desarrollar Sideways Shooter hasta el mismo punto al que llevamos Alien Invasion. Añade una flota de alienígenas y haz que se muevan de lado hacia la nave. O escribe un código que coloque extraterrestres en posiciones aleatorias a lo largo del lado derecho de la pantalla y luego los envíe hacia la nave. Además, escribe un código que haga que los alienígenas desaparezcan cuando sean golpeados.

Terminando el juego

¿Cuál es la diversión y el desafío de jugar un juego que no puedes perder? Si el jugador no derriba la flota lo suficientemente rápido, haremos que los alienígenas destruyan la nave cuando hagan contacto. Al mismo tiempo, limitaremos la cantidad de naves que un jugador puede usar y destruiremos la nave cuando un extraterrestre llegue al final de la pantalla. El juego terminará cuando el jugador haya agotado todos sus barcos.

Detección de colisiones de naves alienígenas

Comenzaremos verificando colisiones entre los extraterrestres y la nave para que podamos responder adecuadamente cuando un extraterrestre la golpee. Comprobaremos si hay colisiones de naves alienígenas inmediatamente después de actualizar la posición de cada alienígena en `AlienInvasion`:

```

extraterrestre
_invasion.py

def _update_alien(yo):
    --recorte--
    self.aliens.actualización()

    # Busque colisiones de naves alienígenas.
    si pygame.sprite.spritecollideany(self.ship, self.aliens):
1
2        print("¡¡¡Barco impactado!!!")
```

La función `spritecollideany()` toma dos argumentos: un `sprite` y un grupo. La función busca cualquier miembro del grupo que haya chocado con el `sprite` y deja de recorrer el grupo tan pronto como encuentra un miembro que ha chocado con el `sprite`. Aquí, recorre el grupo de alienígenas y devuelve al primer alienígena que encuentra que ha chocado con la nave.

Si no ocurren colisiones, `spritecollideany()` devuelve Ninguno y el bloque if no ejecutará 1. Si encuentra un alienígena que ha chocado con la nave, devuelve ese alienígena y el bloque if se ejecuta: imprime ¡¡¡ Nave golpeada!!!

2. Cuando un alienígena golpee la nave, tendremos que realizar una serie de tareas: eliminar todos los alienígenas y balas restantes, volver a centrar la nave y crear una nueva flota. Antes de escribir código para hacer todo esto, queremos saber si nuestro enfoque para detectar colisiones de naves alienígenas funciona correctamente. Escribir una llamada `print()` es una forma sencilla de garantizar que estamos detectando estas colisiones correctamente.

Ahora, cuando ejecutas Alien Invasion, aparece el mensaje Ship hit!!! Debería aparecer en la terminal cada vez que un extraterrestre entra en la nave. Cuando estés probando esta función, configura `Fleet_drop_speed` en un valor más alto, como 50 o 100, para que los alienígenas lleguen a tu nave más rápido.

Respondiendo a las colisiones de naves alienígenas

Ahora necesitamos descubrir exactamente qué sucederá cuando un extraterrestre choque con la nave. En lugar de destruir la instancia del barco y crear una nueva, contaremos cuántas veces el barco ha sido golpeado mediante las estadísticas de seguimiento del juego. Las estadísticas de seguimiento también serán útiles para puntuar.

Escribamos una nueva clase, `GameStats`, para realizar un seguimiento de las estadísticas del juego y guárdela como `game_stats.py`:

`game_stats.py` clase `GameStats`:

```
"""Seguimiento de estadísticas de invasión alienígena."""
```

```
def __init__(self, ai_game):
    """Inicializar estadísticas."""
```

1

```
    self.settings = ai_game.settings
    self.reset_stats()
```

```
def reset_stats(yo):
    """Inicializa estadísticas que pueden cambiar durante el juego."""
    self.ships_left = self.settings.ship_limit
```

Crearemos una instancia de `GameStats` durante todo el tiempo que se esté ejecutando Alien Invasion , pero necesitaremos restablecer algunas estadísticas cada vez que el jugador comience un nuevo juego. Para hacer esto, inicializaremos la mayoría de las estadísticas en `reset_stats()` método, en lugar de directamente en `__init__()`. Llamaremos a este método desde `__init__()` por lo que las estadísticas se configuran correctamente cuando se crea por primera vez la instancia de `GameStats` 1. Pero también podremos llamar a `reset_stats()` cada vez que el jugador comience un nuevo juego. En este momento solo tenemos una estadística, `barcos_izquierdos`, cuyo valor cambiará a lo largo del juego.

La cantidad de barcos con los que comienza el jugador debe almacenarse en `settings.py` como `limite_envio`:

`configuración.py`

```
# Configuración del barco
self.ship_speed = 1,5
self.ship_limit = 3
```

También necesitamos realizar algunos cambios en `alien_invasion.py` para crear una instancia de `GameStats`. Primero, actualizaremos las declaraciones de importación en la parte superior del archivo:

```
alien_invasion.py importar sistema
                    desde el momento importar dormir

                    importar pygame

                    desde la configuración importar configuración
                    desde game_stats importar GameStats
                    desde barco importación Barco
                    --recorte--
```

Importamos la función `sleep()` del módulo de tiempo en la biblioteca estándar de Python, para que podamos pausar el juego por un momento cuando el barco es golpeado. También importamos `GameStats`.

Crearemos una instancia de `GameStats` en `__init__()`:

```
invasión_alienígena.py def __init__(yo):
                        --recorte--
                        self.screen = pygame.display.set_mode(
                            (self.settings.screen_width, self.settings.screen_height))
                        pygame.display.set_caption("Invasión alienígena")

                        # Crea una instancia para almacenar estadísticas del juego.
                        self.stats = GameStats(yo)

                        self.ship = Enviar (yo)
                        --recorte--
```

Creamos la instancia después de crear la ventana del juego pero antes de definir otros elementos del juego, como el barco.

Cuando un extraterrestre golpee la nave, restaremos 1 del número de naves que quedan, destruiremos todos los extraterrestres y las balas existentes, crearemos una nueva flota y reposicionaremos la nave en el medio de la pantalla. También pausaremos el juego por un momento para que el jugador pueda notar la colisión y reagruparse antes de que aparezca una nueva flota.

Pongamos la mayor parte de este código en un nuevo método llamado `_ship_hit()`. llamaremos este método de `_update_aliens()` cuando un extraterrestre golpea la nave:

```
extraterrestre
_invasion.py def _ship_hit(yo):
                """Responder a la nave siendo golpeada por un extraterrestre."""
                # Decremento de barcos_izquierdos.
                1 self.stats.ships_left -= 1

                # Deshazte de las balas y los extraterrestres restantes.
                2 self.balas.vacio()
                auto.alienígenas.vacio()

                # Crea una nueva flota y centra el barco.
                3 self._create_fleet()
                self.ship.center_ship()
```

4 # Pausa.
dormir(0.5)

El nuevo método `_ship_hit()` coordina la respuesta cuando un extraterrestre golpea una nave. Dentro de `_ship_hit()`, el número de naves que quedan se reduce en 1, después de lo cual vaciamos los grupos de balas y alienígenas.

A continuación, creamos una nueva flota y centramos el barco. (Agregaremos el método `center_ship()` a `Ship` en un momento). Luego agregamos una pausa después de que se hayan realizado las actualizaciones en todos los elementos del juego, pero antes de cualquier cambio. han sido dibujados en la pantalla, para que el jugador pueda ver que su nave ha sido alcanzada. 4. La llamada `sleep()` detiene la ejecución del programa durante medio segundo, el tiempo suficiente para que el jugador vea que el extraterrestre ha golpeado la nave. Cuando finaliza la función `sleep()`, la ejecución del código pasa a `_update_screen()` método, que dibuja la nueva flota en la pantalla.

En `_update_alien()`, reemplazamos la llamada `print()` con una llamada a `_ship_hit()` cuando un extraterrestre choca contra la nave:

invasión_alienígena.py

```
def _update_alien(yo):
    --recorte--
    pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()
```

Aquí está el nuevo método `center_ship()`, que pertenece a `ship.py`:

nave.py

```
def centro_nave(yo):
    """Centrar el barco en la pantalla."""
    self.rect.midbottom = self.screen_rect.midbottom
    self.x = flotante(self.rect.x)
```

Centramos la nave de la misma manera que lo hicimos en `__init__()`. Después de centrarlo, restablecemos el atributo `self.x`, que nos permite rastrear la posición exacta del barco.

NOTA tenga en cuenta que nunca fabricamos más de un barco; Creamos solo una instancia de barco para todo el juego y la volvemos a centrar cada vez que el barco ha sido impactado. La estadística se envía_izquierda nos avisará cuando el jugador se haya quedado sin naves.

Ejecuta el juego, dispara a algunos extraterrestres y deja que un extraterrestre golpee la nave. El juego debería pausarse y debería aparecer una nueva flota con el barco centrado en la parte inferior de la pantalla nuevamente.

Aliens que llegan al final de la pantalla

Si un extraterrestre llega al final de la pantalla, el juego responderá de la misma manera que cuando un extraterrestre golpea la nave. Para comprobar cuándo sucede esto, agregue un nuevo método en `alien_invasion.py`:

extraterrestre

_invasion.py

```
def _check_alien_bottom(yo):
    """Comprueba si algún extraterrestre ha llegado al final de la pantalla."""
    para extraterrestre en self.aliens.sprites():
```

1

```

si alien.rect.bottom >= self.settings.screen_height:
    # Trate esto igual que si el barco hubiera sido alcanzado.
    self._ship_hit()
    romper

```

El método `_check.aliens.bottom()` comprueba si algún extraterrestre ha llegado la parte inferior de la pantalla. Un extraterrestre llega al fondo cuando está en posición recta.

El valor es mayor o igual a la altura de la pantalla 1. Si un extraterrestre llega al fondo, llamamos `_ship_hit()`. Si un alienígena toca el fondo, no hay necesidad de comprobar el resto, por lo que salimos del bucle después de llamar a `_ship_hit()`.

Llamaremos a este método desde `_update.aliens()`:

invasión_alienígena.py

```

def _update.aliens(yo):
    --recorte--
    # Busque colisiones de naves alienígenas.
    si pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()

    # Busque extraterrestres que lleguen a la parte inferior de la pantalla.
    self._check.aliens.bottom()

```

Llamamos a `_check.aliens.bottom()` después de actualizar las posiciones de todos los extraterrestres y después de buscar colisiones de naves extraterrestres. Ahora aparecerá una nueva flota cada vez que la nave sea golpeada por un extraterrestre o un extraterrestre llegue al final de la pantalla.

¡Juego terminado!

Alien Invasion se siente más completo ahora, pero el juego nunca termina. El valor de los barcos dejados se vuelve cada vez más negativo. Agreguemos una bandera `game_active`, para que podamos finalizar el juego cuando el jugador se quede sin barcos. Estableceremos esta bandera al final del método `__init__()` en `AlienInvasion`:

invasión_alienígena.py

```

def __init__(yo):
    --recorte--
    # Inicie Alien Invasion en un estado activo.
    self.game_active = Verdadero

```

Ahora agregamos código a `_ship_hit()` que establece `game_active` en `False` cuando el El jugador ha agotado todos sus barcos:

invasión_alienígena.py

```

def _ship_hit(yo):
    """Responder al barco que es golpeado por un extraterrestre."""
    si self.stats.ships_left > 0:
        # Decremento de barcos_izquierdos.
        self.stats.ships_left -= 1
        --recorte--
        # Pausa.
        dormir(0.5)
    demás:
        self.game_active = Falso

```

La mayor parte de `_ship_hit()` no ha cambiado. Hemos movido todo el código existente a un bloque `if`, que prueba para asegurarse de que al jugador le quede al menos un barco. Si es así, creamos una nueva flota, hacemos una pausa y seguimos adelante. Si al jugador no le quedan barcos, configuramos `game_active` en `False`.

Identificar cuándo deben ejecutarse partes del juego

Necesitamos identificar las partes del juego que siempre deben ejecutarse y las partes que deben ejecutarse solo cuando el juego está activo:

`invasión_alienígena.py`

```
def run_game(yo):
    """Inicia el bucle principal del juego."""
    mientras que Verdadero:

        self._check_events()

        si self.game_active:
            auto.envío.actualización()
            self._update_bullets()
            self._update.aliens()

        self._update_screen()
    reloj.automático.tick(60)
```

En el bucle principal, siempre necesitamos llamar a `_check_events()`, incluso si el juego está inactivo. Por ejemplo, todavía necesitamos saber si el usuario presiona `Q` para salir del juego o hace clic en el botón para cerrar la ventana. También continuamos actualizando la pantalla para que podamos realizar cambios en la pantalla mientras esperamos ver si el jugador elige comenzar un nuevo juego. El resto de las llamadas a funciones deben realizarse solo cuando el juego está activo, porque cuando el juego está inactivo, no necesitamos actualizar las posiciones de los elementos del juego.

Ahora, cuando juegues *Alien Invasion*, el juego debería congelarse cuando hayas agotado todas tus naves.

INTÉNTALO TÚ MISMO

13-6. Fin del juego: en *Sideways Shooter*, lleva la cuenta del número de veces que el barco es golpeado y el número de veces que el barco golpea a un alienígena. Decida una condición apropiada para terminar el juego y deténgalo cuando ocurra esta situación.

Resumen

En este capítulo, aprendiste cómo agregar una gran cantidad de elementos idénticos a un juego creando una flota de alienígenas. Usaste bucles anidados para crear una cuadrícula de elementos e hiciste que un gran conjunto de elementos del juego se movieran llamando al método `update()` de cada elemento. Aprendiste a controlar la dirección de

objetos en la pantalla y para responder a situaciones específicas, como cuando la flota llega al borde de la pantalla. Detectaste y respondiste a colisiones cuando las balas alcanzaron a los extraterrestres y los extraterrestres impactaron en la nave. También aprendió a realizar un seguimiento de las estadísticas de un juego y a utilizar el indicador `game_active` para determinar cuándo termina el juego.

En el próximo y último capítulo de este proyecto, agregaremos un botón Jugar para que el jugador pueda elegir cuándo comenzar su primer juego y si desea volver a jugar cuando finalice el juego. Aceleraremos el juego cada vez que el jugador derribe a toda la flota y agregaremos un sistema de puntuación. ¡El resultado final será un juego totalmente jugable!

14

PUNTUACIÓN



En este capítulo, terminaremos de construir Alien Invasion. Agregaremos un botón Jugar para iniciar el juego a pedido y reiniciarlo una vez que finalice. También cambiaremos el juego para que se acelere cuando el jugador suba de nivel e implementaremos un sistema de puntuación. Al final del capítulo, sabrá lo suficiente como para empezar a escribir juegos que aumenten en dificultad a medida que el jugador progresa y que incluyan sistemas de puntuación completos.

Agregar el botón Reproducir

En esta sección, agregaremos un botón Jugar que aparece antes de que comience un juego y reaparece cuando finaliza para que el jugador pueda volver a jugar.

En este momento, el juego comienza tan pronto como ejecutas `alien_invasion.py`. Iniciemos el juego en un estado inactivo y luego solicitemos al jugador que haga clic en el botón Reproducir para comenzar. Para hacer esto, modifique el método `__init__()` de `AlienInvasion`:

```
invasión_alienígena.py    def __init__(yo):
                        """Inicializa el juego y crea recursos para el juego."""
                        pygame.init()
                        --recorte--

                        # Inicie Alien Invasion en un estado inactivo.
                        self.game_active = Falso
```

Ahora el juego debería comenzar en un estado inactivo, sin posibilidad de que el jugador lo inicie hasta que hagamos un botón Reproducir.

Crear una clase de botón

Debido a que Pygame no tiene un método incorporado para crear botones, escribiremos una clase Botón para crear un rectángulo relleno con una etiqueta. Puedes usar este código para crear cualquier botón en un juego. Aquí está la primera parte de la clase Botón ; guárdelo como `botón.py`:

```
botón.py importar pygame.font
```

Botón de clase:

```
"""Una clase para construir botones para el juego."""

1 def __init__(self, ai_game, mensaje):
    """Inicializar atributos del botón."""
    self.pantalla = ai_game.pantalla
    self.screen_rect = self.screen.get_rect()

    # Establecer las dimensiones y propiedades del botón.
2     ancho.propio, alto.propio = 200, 50
    self.button_color = (0, 135, 0)
    self.text_color = (255, 255, 255)
3     self.font = pygame.font.SysFont(Ninguno, 48)

    # Construye el objeto recto del botón y céntralo.
4     self.rect = pygame.Rect(0, 0, self.ancho, self.alto)
    self.rect.center = self.screen_rect.center

    # El mensaje del botón debe prepararse solo una vez.
5     self._prep_msg(msg)
```

Primero, importamos el módulo `pygame.font`, que permite a Pygame representar texto en la pantalla. El método `__init__()` toma los parámetros `self`, el `ai_game` object y `msg`, que contiene el texto del botón 1. Configuramos las dimensiones del botón 2, configuramos `button_color` para colorear el objeto recto del botón de color verde oscuro y configuramos `text_color` para representar el texto en blanco.

A continuación, preparamos un atributo de fuente para representar el texto 3. El argumento Ninguno le dice a Pygame que use la fuente predeterminada y 48 especifica el tamaño del texto. Para centrar el botón en la pantalla, creamos un rect para el botón 4 y establezca su atributo central para que coincida con el de la pantalla.

Pygame trabaja con texto representando la cadena que desea mostrar una imagen. Finalmente, llamamos a `_prep_msg()` para manejar esta representación 5.

Aquí está el código para `_prep_msg()`:

```
botón.py def _prep_msg(self, mensaje):
1         """Convierte el mensaje en una imagen renderizada y centra el texto en el botón."""
2         self.msg_image = self.font.render(msg, Verdadero, self.text_color,
            self.botón_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center
```

El método `_prep_msg()` necesita un parámetro propio y el texto que se va a representar como una imagen (`msg`). La llamada a `font.render()` convierte el texto almacenado en `msg` en una imagen, que luego almacenamos en `self.msg_image` 1. El `font.render()` El método también toma un valor booleano para activar o desactivar el antialiasing (antialiasing suaviza los bordes del texto). Los argumentos restantes son el color de fuente y el color de fondo especificados. Configuramos el antialiasing en Verdadero y configuramos el fondo del texto en el mismo color que el botón. (Si no incluye un color de fondo, Pygame intentará representar la fuente con un fondo transparente).

Centramos la imagen del texto en el botón creando un rect a partir del imagen y configurando su atributo central para que coincida con el del botón 2.

Finalmente, creamos un método `draw_button()` al que podemos llamar para mostrar el botón en pantalla:

```
botón.py def botón_dibujar(yo):
        """Dibuja un botón en blanco y luego dibuja un mensaje."""
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

Llamamos `screen.fill()` para dibujar la parte rectangular del botón. Luego llamamos a `screen.blit()` para dibujar la imagen de texto en la pantalla, pasándole una imagen y el objeto rect asociado con la imagen. Esto completa la clase Botón .

Dibujar el botón en la pantalla

Usaremos la clase Botón para crear un botón Reproducir en AlienInvasion. Primero, actualizaremos las declaraciones de importación :

```
invasión_alienígena.py --snip--
desde game_stats importar GameStats
desde el botón importar botón
```

Como solo necesitamos un botón Reproducir, crearemos el botón en el método `__init__()` de `AlienInvasion`. Podemos colocar este código al final de `__init__()`:

```
invasión_alienígena.py    def __init__(yo):
                           --recorte--
                           self.game_active = Falso

                           # Haz el botón Reproducir.
                           self.play_button = Botón(self, "Reproducir")
```

Este código crea una instancia de Botón con la etiqueta Reproducir, pero no dibuja el botón en la pantalla. Para hacer esto, llamaremos al botón `draw_button()` método en `_update_screen()`:

```
invasión_alienígena.py    def _update_screen(auto):
                           --recorte--
                           self.aliens.draw(auto.pantalla)

                           # Dibuja el botón de reproducción si el juego está inactivo.
                           si no es self.game_active:
                               self.play_button.draw_button()

                           pygame.display.flip()
```

Para que el botón Reproducir sea visible sobre todos los demás elementos de la pantalla, lo dibujamos después de que se hayan dibujado todos los demás elementos, pero antes de pasar a una nueva pantalla. Lo incluimos en un bloque `if`, por lo que el botón solo aparece cuando el juego está inactivo.

Ahora, cuando ejecutes Alien Invasion, deberías ver un botón Reproducir en el centro de la pantalla, como se muestra en la Figura 14-1.

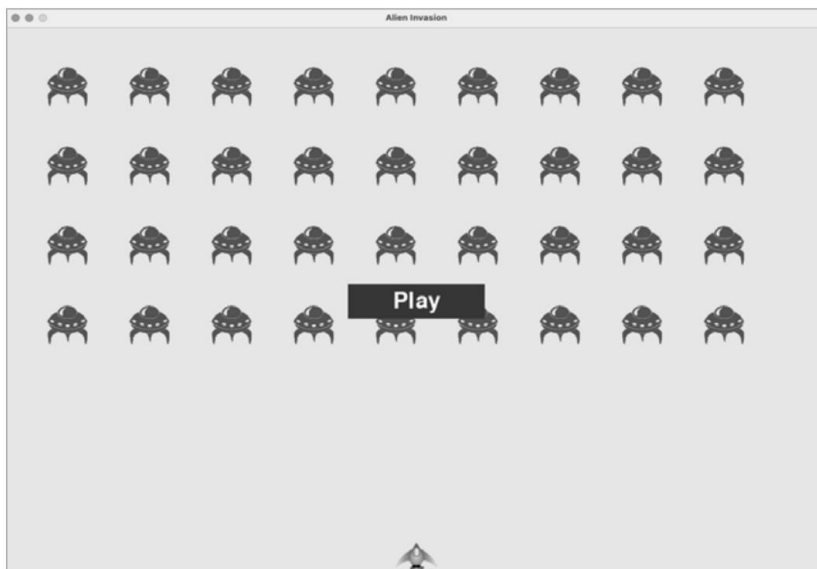


Figura 14-1: Aparece un botón Jugar cuando el juego está inactivo.

Comenzando el juego

Para comenzar un nuevo juego cuando el jugador hace clic en Jugar, agregue el siguiente elif bloque hasta el final de `_check_events()` para monitorear los eventos del mouse sobre el botón:

```

extraterrestre
_invasion.py

def _check_events(yo):
    """Responder a pulsaciones de teclas y eventos del mouse."""
    para evento en pygame.event.get():
        si evento.tipo == pygame.SALIR:
            --recorte--
1         elif event.type == pygame.MOUSEBUTTONDOWN:
2             mouse_pos = pygame.mouse.get_pos()
3             self._check_play_button(mouse_pos)

```

Pygame detecta un evento `MOUSEBUTTONDOWN` cuando el jugador hace clic en cualquier lugar de la pantalla 1, pero queremos restringir nuestro juego para que responda a los clics del mouse solo en el botón Reproducir. Para lograr esto, usamos `pygame.mouse.get_pos()`, que devuelve una tupla que contiene las coordenadas x y del cursor del mouse cuando se hace clic en el botón del mouse 2. Enviamos estos valores al nuevo método `_check_play_button()` 3.

Aquí está `_check_play_button()`, que elegí colocar después de `_check_events()`:

```

extraterrestre
_invasion.py

def _check_play_button(self, mouse_pos):
    """Comienza un nuevo juego cuando el jugador hace clic en Jugar."""
1     si self.play_button.rect.collidepoint(mouse_pos):
        self.game_active = Verdadero

```

Usamos el método `rect.collidepoint()` para verificar si el punto del clic del mouse se superpone a la región definida por el `rect 1` del botón Reproducir. Si es así, configuramos `game_active` en `True` y ¡el juego comienza!

En este punto, deberías poder comenzar y jugar un juego completo. Cuando el juego termina, el valor de `game_active` debería volverse `False` y el botón Reproducir debería reaparecer.

Reiniciar el juego

El código del botón Reproducir que acabamos de escribir funciona la primera vez que el jugador hace clic en Reproducir. Pero no funciona después de que termina el primer juego, porque las condiciones que causaron que el juego terminara no se han restablecido.

Para restablecer el juego cada vez que el jugador hace clic en Jugar, necesitamos restablecer las estadísticas del juego, eliminar los viejos alienígenas y las balas, construir una nueva flota y centrar la nave, como se muestra aquí:

```

extraterrestre
_invasion.py

def _check_play_button(self, mouse_pos):
    """Comienza un nuevo juego cuando el jugador hace clic en Jugar."""
    si self.play_button.rect.collidepoint(mouse_pos):
        # Restablecer las estadísticas del juego.
1         self.stats.reset_stats()
        self.game_active = Verdadero

```

```

2          # Deshazte de las balas y los extraterrestres restantes.
          self.balas.vacio()
          auto.alienigenas.vacio()

3          # Crea una nueva flota y centra el barco.
          self._create_fleet()
          self.ship.center_ship()

```

Restablecemos las estadísticas del juego 1, lo que le da al jugador tres barcos nuevos. Luego configuramos `game_active` en `True` para que el juego comience tan pronto como termine de ejecutarse el código de esta función. Vaciamos los alienígenas y los grupos de balas 2, y luego creamos una nueva flota y centramos la nave 3.

Ahora el juego se reiniciará correctamente cada vez que hagas clic en Jugar, permitiéndote ¡Para jugarlo tantas veces como quieras!

Desactivar el botón Reproducir

Un problema con nuestro botón Reproducir es que la región del botón en la pantalla seguirá respondiendo a los clics incluso cuando el botón Reproducir no esté visible. Si haces clic en el área del botón Jugar por accidente después de que comience un juego, ¡el juego se reiniciará!

Para solucionar este problema, configura el juego para que se inicie solo cuando `game_active` sea `False`:

```

estraterrestre
_invasion.py
1      def _check_play_button(self, mouse_pos):
2          """Comienza un nuevo juego cuando el jugador hace clic en Jugar."""
          button_clicked = self.play_button.rect.collidepoint(mouse_pos)
          si se hace clic en el botón y no en self.game_active:
              # Restablecer las estadísticas del juego.
              self.stats.reset_stats()
              --recorte--

```

La bandera `button_clicked` almacena un valor Verdadero o Falso 1, y el juego se reiniciará solo si se hace clic en Reproducir y el juego no está activo actualmente 2. Para probar este comportamiento, inicie un nuevo juego y haga clic repetidamente donde debería estar el botón Reproducir. Si todo funciona como se esperaba, hacer clic en el área del botón Reproducir no debería tener ningún efecto en el juego.

Ocultar el cursor del mouse

Queremos que el cursor del mouse esté visible cuando el juego esté inactivo, pero una vez que comienza el juego, simplemente estorba. Para solucionar este problema, lo haremos invisible cuando el juego se active. Podemos hacer esto al final del bloque `if` en `_check_play_button()`:

```

invasión_alienigena.py
      def _check_play_button(self, mouse_pos):
          """Comienza un nuevo juego cuando el jugador hace clic en Jugar."""
          button_clicked = self.play_button.rect.collidepoint(mouse_pos)
          si se hace clic en el botón y no en self.game_active:
              --recorte--
              # Ocultar el cursor del mouse.
              pygame.mouse.set_visible(False)

```

Pasar `False` a `set_visible()` le dice a Pygame que oculte el cursor cuando el mouse está sobre la ventana del juego.

Haremos que el cursor reaparezca una vez que termine el juego para que el jugador pueda Haz clic en Jugar nuevamente para comenzar un nuevo juego. Aquí está el código para hacer eso:

```
invasión_alienígena.py
def _ship_hit(yo):
    """Responder al barco que es golpeado por un extraterrestre."""
    si self.stats.ships_left > 0:
        --recorte--
    demás:
        self.game_active = Falso
        pygame.mouse.set_visible(Verdadero)
```

Hacemos que el cursor vuelva a ser visible tan pronto como el juego se vuelve inactivo, lo que sucede en `_ship_hit()`. La atención a detalles como este hace que tu juego tenga un aspecto más profesional y permite al jugador concentrarse en jugar, en lugar de descubrir la interfaz de usuario.

INTÉNTALO TÚ MISMO

14-1. Presiona P para jugar: debido a que Alien Invasion usa la entrada del teclado para controlar la nave, sería útil comenzar el juego presionando una tecla. Agregue código que le permita al jugador presionar P para comenzar. Podría ser útil mover algún código de `_check_play_button()` a un método `_start_game()` que se puede llamar desde `_check_play_button()` y `_check_keydown_events()`.

14-2. Práctica de tiro: cree un rectángulo en el borde derecho de la pantalla que se mueva hacia arriba y hacia abajo a un ritmo constante. Luego, en el lado izquierdo de la pantalla, crea un barco que el jugador puede mover hacia arriba y hacia abajo mientras dispara balas al objetivo rectangular. Agrega un botón Jugar que inicia el juego, y cuando el jugador falla el objetivo tres veces, finaliza el juego y haz que el botón Jugar vuelva a aparecer. Deje que el jugador reinicie el juego con este botón Jugar.

Subir de nivel

En nuestro juego actual, una vez que un jugador derriba a toda la flota alienígena, el jugador alcanza un nuevo nivel, pero la dificultad del juego no cambia. Animemos un poco las cosas y hagamos que el juego sea más desafiante aumentando la velocidad del juego cada vez que un jugador limpia la pantalla.

Modificar la configuración de velocidad

Primero reorganizaremos la clase Configuración para agrupar las configuraciones del juego en estáticas y dinámicas. También nos aseguraremos de que cualquier configuración que cambie

durante el reinicio del juego cuando comenzamos un nuevo juego. Aquí está el `__init__()` método para `settings.py`:

```
configuración.py    def __init__(yo):
                    """Inicializa la configuración estática del juego."""
                    # Ajustes de pantalla
                    self.ancha_pantalla = 1200
                    self.screen_height = 800
                    self.bg_color = (230, 230, 230)

                    # Configuración del barco
                    self.ship_limit = 3

                    # Configuración de viñetas
                    self.bullet_width = 3
                    self.bullet_height = 15
                    self.bullet_color = 60, 60, 60
                    self.bullets_allowed = 3

                    # Configuración alienígena
                    self.fleet_drop_speed = 10

                    # Qué tan rápido se acelera el juego
1                   self.speedup_scale = 1.1

2                   self.initialize_dynamic_settings()
```

Continuamos inicializando configuraciones que permanecen constantes en `__init__()` método. Agregamos una configuración `speedup_scale` 1 para controlar qué tan rápido se acelera el juego: un valor de 2 duplicará la velocidad del juego cada vez que el jugador alcance un nuevo nivel; un valor de 1 mantendrá la velocidad constante. Un valor como 1,1 debería aumentar la velocidad lo suficiente como para que el juego sea desafiante pero no imposible. Finalmente, llamamos al método `initialize_dynamic_settings()` para inicializar los valores de los atributos que deben cambiar a lo largo del juego 2.

Aquí está el código para `initialize_dynamic_settings()`:

```
configuración.py    def inicializar_configuración_dinámica(auto):
                    """Inicializa configuraciones que cambian a lo largo del juego."""
                    self.ship_speed = 1,5
                    self.bullet_speed = 2,5
                    self.alien_speed = 1.0

                    # Fleet_direction de 1 representa la derecha; -1 representa la izquierda.
                    self.fleet_direction = 1
```

Este método establece los valores iniciales para las velocidades del barco, la bala y el alienígena. Aumentaremos estas velocidades a medida que el jugador avance en el juego y las restableceremos cada vez que el jugador comience un nuevo juego. Incluimos `flota_dirección` en este método para que los alienígenas siempre se muevan justo al comienzo de un nuevo juego. No necesitamos aumentar el valor de `Fleet_drop_speed`,

porque cuando los alienígenas se mueven más rápido por la pantalla, también bajarán más rápido por la pantalla.

Para aumentar las velocidades de la nave, las balas y los alienígenas cada vez que el jugador alcanza un nuevo nivel, escribiremos un nuevo método llamado `aumentar_velocidad()`:

```
configuración.py    def aumentar_velocidad(yo):
                    """Aumentar la configuración de velocidad."""
                    self.ship_speed *= self.speedup_scale
                    self.bullet_speed *= self.speedup_scale
                    self.alien_speed *= self.speedup_scale
```

Para aumentar la velocidad de estos elementos del juego, multiplicamos cada velocidad. configuración por el valor de `speedup_scale`.

Aumentamos el tempo del juego llamando a `aumentar_velocidad()` en `_check_bullet_alien_collisions()` cuando el último alienígena de una flota ha sido derribado:

```
invasión_alienígena.py def _check_bullet_alien_collisions(yo):
                        --recorte--
                        si no es uno mismo.extraterrestres:
                            # Destruye las balas existentes y crea una nueva flota.
                            self.balas.vacio()
                            self._create_fleet()
                            self.settings.increase_speed()
```

¡Cambiar los valores de las configuraciones de velocidad `ship_speed`, `alien_speed` y `bullet_speed` es suficiente para acelerar todo el juego!

Restablecer la velocidad

Ahora necesitamos devolver cualquier configuración modificada a sus valores iniciales cada vez que el jugador comience un nuevo juego; de lo contrario, cada nuevo juego comenzaría con la configuración de velocidad aumentada del juego anterior:

```
invasión_alienígena.py def _check_play_button(self, mouse_pos):
                        """Comienza un nuevo juego cuando el jugador hace clic en Jugar."""
                        button_clicked = self.play_button.rect.collidepoint(mouse_pos)
                        si se hace clic en el botón y no en self.game_active:
                            # Restablecer la configuración del juego.
                            self.settings.initialize_dynamic_settings()
                        --recorte--
```

Jugar Alien Invasion debería ser más divertido y desafiante ahora. Cada vez que limpias la pantalla, el juego debería acelerarse y volverse un poco más difícil. Si el juego se vuelve demasiado difícil y demasiado rápido, disminuya el valor de `settings.speedup_scale`. O si el juego no es lo suficientemente desafiante, aumenta ligeramente el valor. Encuentre un punto óptimo aumentando la dificultad en un período de tiempo razonable. Las primeras dos pantallas deberían ser fáciles, las siguientes deberían ser desafiantes pero factibles, y las pantallas siguientes deberían ser casi imposiblemente difíciles.

INTÉNTALO TÚ MISMO

14-3. Práctica de tiro desafiante: comience con su trabajo del ejercicio 14-2 (página 283). Haga que el objetivo se mueva más rápido a medida que avanza el juego y reinicie el objetivo a la velocidad original cuando el jugador haga clic en Jugar.

14-4. Niveles de dificultad: crea un conjunto de botones para Alien Invasion que permitan al jugador seleccionar un nivel de dificultad inicial apropiado para el juego. Cada botón debe asignar los valores apropiados para los atributos en Configuración necesarios para crear diferentes niveles de dificultad.

Puntuación

Implementemos un sistema de puntuación para realizar un seguimiento de la puntuación del juego en tiempo real y mostrar la puntuación más alta, el nivel y la cantidad de barcos restantes.

La puntuación es una estadística del juego, por lo que agregaremos un atributo de puntuación a GameStats:

```
game_stats.py class GameStats:
    --recorte--
    def reset_stats(yo):
        """Inicializa estadísticas que pueden cambiar durante el juego."""
        self.ships_left = self.ai_settings.ship_limit
        puntuación propia = 0
```

Para restablecer la puntuación cada vez que comienza un nuevo juego, inicializamos la puntuación en `reset_stats()` en lugar de `__init__()`.

Mostrando la puntuación

Para mostrar la puntuación en la pantalla, primero creamos una nueva clase, Marcador. Por ahora, esta clase solo mostrará la puntuación actual. Con el tiempo, también lo usaremos para informar la puntuación más alta, el nivel y la cantidad de barcos restantes. Aquí está la primera parte de la clase; guárdelo como `marcador.py`:

```
marcador.py importar pygame.font

Marcador de clase:
    """Una clase para reportar información de puntuación."""

1 definición __init__(self, ai_game):
    """Inicializar atributos de puntuación."""
    self.pantalla = ai_game.pantalla
    self.screen_rect = self.screen.get_rect()
    self.settings = ai_game.settings
    self.stats = ai_game.stats

    # Configuración de fuente para información de puntuación.
2     self.text_color = (30, 30, 30)
3     self.font = pygame.font.SysFont(Ninguno, 48)
```



```

4          # Prepare la imagen de partitura inicial.
          self.prep_score()

```

Debido a que Scoreboard escribe texto en la pantalla, comenzamos importando el módulo `pygame.font`. A continuación, le damos a `__init__()` el parámetro `ai_game` para que pueda acceder a los objetos de configuración, pantalla y estadísticas, que necesitará para informar los valores que estamos rastreando 1. Luego configuramos un color de texto 2 y creamos una instancia de un objeto de fuente 3.

Para convertir el texto que se mostrará en una imagen, llamamos a `prep_score()` 4, que definimos aquí:

```

marcador.py      def puntuación_prep(yo):
                  """Convierte la partitura en una imagen renderizada."""
1                 puntuación_str = str(self.stats.score)
2                 self.score_image = self.font.render(score_str, Verdadero,
                  self.text_color, self.settings.bg_color)

                  # Muestra la puntuación en la parte superior derecha de la pantalla.
3                 self.score_rect = self.score_image.get_rect()
4                 self.score_rect.right = self.screen_rect.right - 20
5                 self.score_rect.top = 20

```

En `prep_score()`, convertimos el valor numérico `stats.score` en una cadena 1 y luego pasamos esta cadena a `render()`, lo que crea la imagen 2. Para mostrar la partitura claramente en pantalla, pasamos el color de fondo de la pantalla y el color del texto a `render()`.

Colocaremos la puntuación en la esquina superior derecha de la pantalla y haremos que se expanda hacia la izquierda a medida que aumenta la puntuación y crece el ancho del número. Para asegurarnos de que la puntuación siempre se alinee con el lado derecho de la pantalla, creamos un rect llamado `score_rect` 3 y configuramos su borde derecho a 20 píxeles del borde derecho de la pantalla 4. Luego colocamos el borde superior a 20 píxeles hacia abajo desde el parte superior de la pantalla 5.

Luego creamos un método `show_score()` para mostrar la imagen de partitura renderizada:

```

marcador.py      def show_score(yo):
                  """Dibuja la puntuación en la pantalla."""
                  self.screen.blit(self.score_image, self.score_rect)

```

Este método dibuja la imagen de la partitura en la pantalla en la ubicación `score_rect` específica.

Hacer un marcador

Para mostrar la puntuación, crearemos una instancia de `Scoreboard` en `AlienInvasion`. Primero, actualicemos las declaraciones de importación :

```

invasión_alienígena.py --snip--
    desde game_stats importar GameStats
    desde el marcador importar el marcador
--recorte--

```

A continuación, creamos una instancia de Scoreboard en `__init__()`:

```
invasión_alienígena.py def __init__(self): --snip--

    pygame.display.set_caption("Invasión alienígena")

    # Crea una instancia para almacenar estadísticas del juego, #
    # y crea un marcador.
    self.stats = GameStats(self) self.sb =
    Marcador(self) --snip--
```

Luego dibujamos el marcador en pantalla en `_update_screen()`:

```
invasión_alienígena.py def _update_screen(self): --snip--

    self.aliens.draw(self.pantalla)

    # Dibujar la información de la puntuación.
    self.sb.show_score()

    # Dibuja el botón de reproducción si el juego está inactivo. --
    recorte--
```

Llamamos a `show_score()` justo antes de dibujar el botón Reproducir.

Cuando ejecutas Alien Invasion ahora, debería aparecer un 0 en la parte superior derecha de la pantalla. (En este punto, sólo queremos asegurarnos de que la puntuación aparezca en el lugar correcto antes de desarrollar más el sistema de puntuación). La figura 14-2 muestra la puntuación tal como aparece antes de que comience el juego.

¡A continuación, asignaremos valores de puntos a cada alienígena!

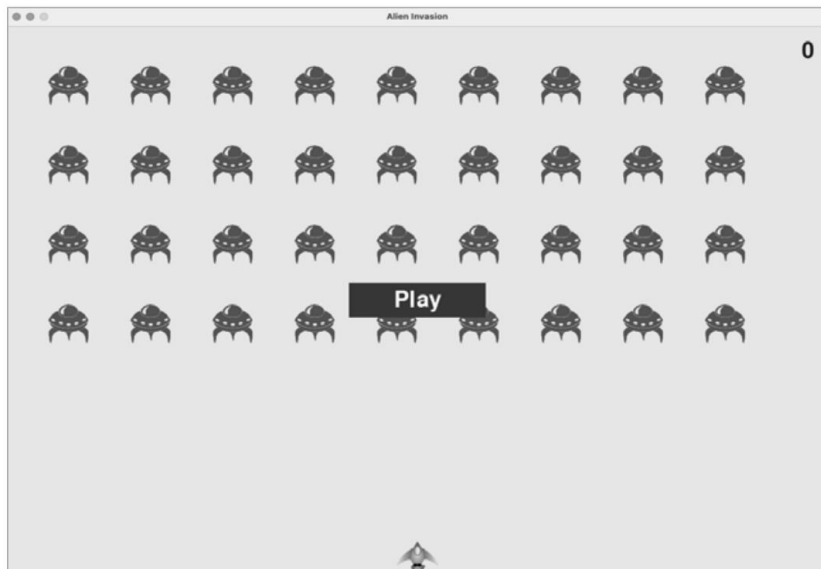


Figura 14-2: La puntuación aparece en la esquina superior derecha de la pantalla.

Actualización de la puntuación a medida que los extraterrestres son derribados

Para escribir una puntuación en vivo en pantalla, actualizamos el valor de `stats.score` cada vez que golpeamos a un extraterrestre y luego llamamos a `prep_score()` para actualizar la imagen de la puntuación. Pero primero, determinemos cuántos puntos obtiene un jugador cada vez que derriba a un alienígena:

```
configuración.py    def inicializar_configuración_dinámica(auto):
                    --recorte--

                    # Configuración de puntuación
                    self.alien_points = 50
```

Aumentaremos el valor de puntos de cada alienígena a medida que avance el juego. Para asegurarnos de que este valor de puntos se restablezca cada vez que comienza un nuevo juego, configuramos el valor en `initialize_dynamic_settings()`.

Actualicemos la puntuación en `_check_bullet_alien_collisions()` cada vez que derriben a un extraterrestre:

```
invasión_alienígena.py def _check_bullet_alien_collisions(yo):
                        """Responder a colisiones bala-alienígenas."""
                        # Elimina las balas y los extraterrestres que hayan chocado.
                        colisiones = pygame.sprite.groupcollide(
                            auto.balas, auto.alienígenas, Verdadero, Verdadero)

                        si colisiones:

                            self.stats.score += self.settings.alien_points
                            self.sb.prep_score()

                        --recorte--
```

Cuando una bala alcanza a un extraterrestre, Pygame devuelve un diccionario de colisiones . Comprobamos si el diccionario existe y, si existe, el valor del extraterrestre se suma a la puntuación. Luego llamamos a `prep_score()` para crear una nueva imagen para la puntuación actualizada.

¡Ahora, cuando juegues Alien Invasion, deberías poder acumular puntos!

Restablecer la puntuación

En este momento, solo estamos preparando una nueva puntuación después de que un alienígena haya sido golpeado, lo cual funciona durante la mayor parte del juego. Pero cuando comenzamos un nuevo juego, seguiremos viendo nuestra puntuación del juego anterior hasta que golpeemos al primer alienígena.

Podemos solucionar este problema preparando la puntuación al iniciar un nuevo juego:

```
invasión_alienígena.py def _check_play_button(self, mouse_pos):
                        --recorte--

                        si se hace clic en el botón y no en self.game_active:
                            --recorte--

                            # Restablecer las estadísticas del juego.
                            self.stats.reset_stats()
                            self.sb.prep_score()

                        --recorte--
```

Llamamos a `prep_score()` después de restablecer las estadísticas del juego al iniciar un juego nuevo. Esto prepara el marcador con una puntuación de 0.

Asegurarse de anotar todos los golpes

Tal como está escrito actualmente, nuestro código podría no puntuar para algunos extraterrestres. Por ejemplo, si dos balas chocan con alienígenas durante el mismo paso por el bucle o si hacemos una bala extra ancha para impactar a varios alienígenas, el jugador solo recibirá puntos por impactar a uno de los alienígenas. Para solucionar este problema, refinemos la forma en que se detectan las colisiones entre balas y extraterrestres.

En `_check_bullet_alien_collisions()`, cualquier bala que colisione con un extraterrestre se convierte en una clave en el diccionario de colisiones. El valor asociado con cada bala es una lista de alienígenas con los que ha chocado. Revisamos los valores en el diccionario de colisiones para asegurarnos de otorgar puntos por cada impacto alienígena:

```
invasión_alienígena.py    def _check_bullet_alien_collisions(yo):
                           --recorte--
                           si colisiones:
                               para extraterrestres en colisiones.valores():
                                   self.stats.score += self.settings.alien_points * len(alienígenas)
                                   self.sb.prep_score()
                           --recorte--
```

Si se ha definido el diccionario de colisiones, recorreremos todos los valores del diccionario. Recuerda que cada valor es una lista de alienígenas alcanzados por una sola bala. Multiplicamos el valor de cada extranjero por el número de extranjeros en cada lista y sumamos esta cantidad a la puntuación actual. Para probar esto, cambia el ancho de una bala a 300 píxeles y verifica que recibes puntos por cada alienígena que golpees con tus balas extra anchas; luego devuelva el ancho de la viñeta a su valor normal.

Valores de puntos crecientes

Debido a que el juego se vuelve más difícil cada vez que un jugador alcanza un nuevo nivel, los alienígenas en niveles posteriores deberían valer más puntos. Para implementar esta funcionalidad, agregaremos código para aumentar el valor de los puntos cuando aumente la velocidad del juego:

Configuración de la clase `settings.py` :

```
"""Una clase para almacenar todas las configuraciones de Alien Invasion."""

def __init__(yo):
    --recorte--
    # Qué tan rápido se acelera el juego
    self.speedup_scale = 1.1
    # Qué tan rápido aumentan los valores de puntos alienígenas
    self.score_scale = 1,5

    self.initialize_dynamic_settings()

def inicializar_configuración_dinámica(auto):
    --recorte--
```

1

```
def aumentar_velocidad(yo):
    """Aumentar la configuración de velocidad y los valores de puntos alienígenas."""
    self.ship_speed *= self.speedup_scale
    self.bullet_speed *= self.speedup_scale
    self.alien_speed *= self.speedup_scale
```

```
2 self.alien_points = int(self.alien_points * self.score_scale)
```

Definimos una tasa a la que aumentan los puntos, a la que llamamos puntuación_escalas 1. Un pequeño aumento en la velocidad (1.1) hace que el juego sea más desafiante rápidamente. Pero para ver una diferencia más notable en la puntuación, necesitamos cambiar el valor de los puntos alienígenas en una cantidad mayor (1,5). Ahora, cuando aumentamos la velocidad del juego, también aumentamos el valor en puntos de cada golpe 2. Usamos la función int() para aumentar el valor en puntos en números enteros.

Para ver el valor de cada alienígena, agregue una llamada print() al incremento_speed() método en Configuración:

```
configuración.py def aumentar_velocidad(yo):
    --recorte--
    self.alien_points = int(self.alien_points * self.score_scale)
    imprimir(self.alien_points)
```

El nuevo valor de puntos debería aparecer en la terminal cada vez que alcances un nuevo nivel.

NOTA asegúrese de eliminar la llamada print() después de verificar que el valor de los puntos está aumentando, o podría afectar el rendimiento del juego y distraer al jugador.

Redondeando la puntuación

La mayoría de los juegos de disparos estilo arcade reportan puntuaciones como múltiplos de 10, así que sigamos ese ejemplo con nuestras puntuaciones. Además, formateemos la partitura para incluir separadores de coma en números grandes. Haremos este cambio en el Marcador:

```
marcador.py def puntuación_prep(yo):
    """Convierte la partitura en una imagen renderizada."""
    puntuación_redondeada = ronda(self.stats.score, -1)
    puntuación_str = f"{puntuación_redondeada:}"
    self.score_image = self.font.render(score_str, Verdadero,
                                        self.text_color, self.settings.bg_color)
    --recorte--
```

La función round() normalmente redondea un flotante a un número determinado de decimales dado como segundo argumento. Sin embargo, cuando pasa un número negativo como segundo argumento, round() redondeará el valor al 10, 100, 1000, etc. más cercano. Este código le dice a Python que redondee el valor de stats.score al 10 más cercano y lo asigne a rounded_score.

Luego usamos un especificador de formato en la cadena f para la partitura. un formato El especificador es una secuencia especial de caracteres que modifica la forma en que se presenta el valor de una variable. En este caso la secuencia :, le dice a Python que inserte

comas en los lugares apropiados del valor numérico proporcionado. Esto da como resultado cadenas como 1.000.000 en lugar de 1.000.000.

Ahora, cuando ejecute el juego, debería ver un icono redondeado y perfectamente formateado. Puntúe incluso cuando acumule muchos puntos, como se muestra en la Figura 14-3.



Figura 14-3: Puntuación redondeada con separadores de coma

Puntuaciones altas

Todos los jugadores quieren superar la puntuación más alta de un juego, así que hagamos un seguimiento e informemos las puntuaciones altas para darles a los jugadores algo por lo que trabajar. Almacenaremos puntuaciones altas en GameStats:

juego_stats.py

```
def __init__(self, ai_game):
    --recorte--
    # La puntuación alta nunca debe restablecerse.
    self.puntuación_alta = 0
```

Debido a que la puntuación más alta nunca debe restablecerse, inicializamos high_score en __init__() en lugar de reset_stats().

A continuación, modificaremos el marcador para mostrar la puntuación más alta. Comencemos con el método __init__() :

marcador.py

```
def __init__(self, ai_game):
    --recorte--
    # Prepare las imágenes de partitura inicial.
    self.prep_score()
    self.prep_high_score()
```

La puntuación más alta se mostrará por separado de la puntuación, por lo que necesitamos una Nuevo método, prep_high_score(), para preparar la imagen de puntuación más alta 1.

Aquí está el método `prep_high_score()` :

```

marcador.py def prep_high_score(yo):
            """Convierta la puntuación más alta en una imagen renderizada."""
1           puntuación_alta = ronda(self.stats.puntuación_alta, -1)
            cadena_puntuación_alta = f"{puntuación_alta:}"
2           self.high_score_image = self.font.render(high_score_str, Verdadero,
                self.text_color, self.settings.bg_color)

            # Centrar la puntuación más alta en la parte superior de la pantalla.
            self.high_score_rect = self.high_score_image.get_rect()
3           self.high_score_rect.centerx = self.screen_rect.centerx
4           self.high_score_rect.top = self.score_rect.top

```

Redondeamos la puntuación más alta al 10 más cercano y le damos formato con comas 1. Luego generamos una imagen a partir de la puntuación más alta 2, centramos la puntuación más alta en sentido horizontal 3 y configuramos su atributo superior para que coincida con la parte superior de la imagen de la puntuación. 4.

El método `show_score()` ahora dibuja el puntaje actual en la parte superior derecha y el puntaje más alto en la parte superior central de la pantalla:

```

marcador.py def show_score(yo):
            """Dibuja la puntuación en la pantalla."""
            self.screen.blit(self.score_image, self.score_rect)
            self.screen.blit(self.high_score_image, self.high_score_rect)

```

Para comprobar puntuaciones altas, escribiremos un nuevo método, `check_high_score()`, en `Scoreboard`:

```

marcador.py def check_high_score(yo):
            """Comprueba si hay una nueva puntuación más alta."""
            si self.stats.score > self.stats.high_score:
                self.stats.high_score = self.stats.puntuación
                self.prep_high_score()

```

El método `check_high_score()` compara la puntuación actual con la puntuación más alta. Si la puntuación actual es mayor, actualizamos el valor de `high_score` y llame a `prep_high_score()` para actualizar la imagen de la puntuación más alta.

Necesitamos llamar a `check_high_score()` cada vez que un alienígena es golpeado después de la actualización. ing la puntuación en `_check_bullet_alien_collisions()`:

```

invasión_alienígena.py def _check_bullet_alien_collisions(yo):
            --recorte--
            si colisiones:
                para extraterrestres en colisiones.valores():
                    self.stats.score += self.settings.alien_points * len(alienígenas)
                    self.sb.prep_score()
                    self.sb.check_high_score()
            --recorte--

```

Llamamos a `check_high_score()` cuando el diccionario de colisiones está presente, y Lo hacemos después de actualizar la puntuación de todos los alienígenas que han sido alcanzados.

La primera vez que juegues Alien Invasion, tu puntuación será la puntuación más alta, por lo que se mostrará como la puntuación actual y la puntuación más alta. Pero cuando comienzas un segundo juego, tu puntuación más alta debería aparecer en el medio y tu puntuación actual debería aparecer a la derecha, como se muestra en la Figura 14-4.

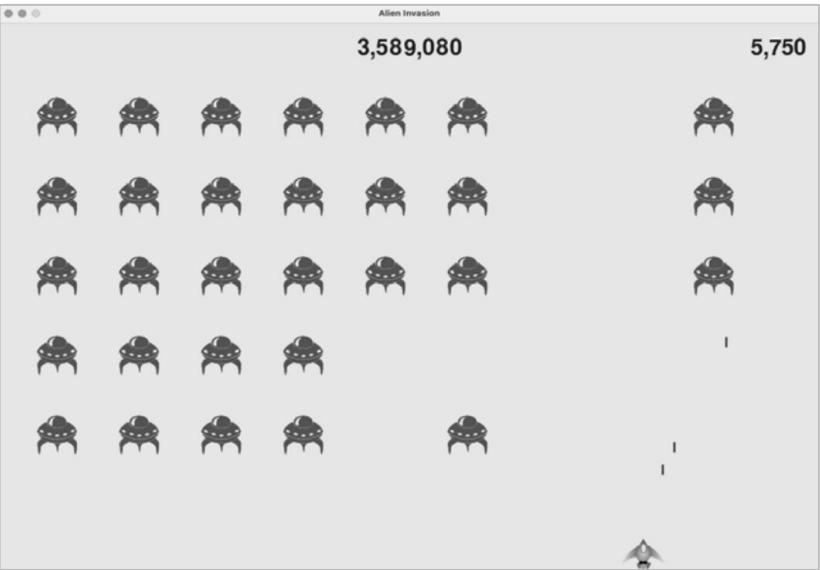


Figura 14-4: La puntuación más alta se muestra en la parte superior central de la pantalla.

Mostrando el nivel

Para mostrar el nivel del jugador en el juego, primero necesitamos un atributo en GameStats que represente el nivel actual. Para restablecer el nivel al comienzo de cada nuevo juego, inicialízalo en reset_stats():

juego_stats.py

```
def reset_stats(yo):
    """Inicializa estadísticas que pueden cambiar durante el juego."""
    self.ships_left = self.settings.ship_limit
    puntuación propia = 0
    auto.nivel = 1
```

Para que Scoreboard muestre el nivel actual, llamamos a un nuevo método, prep_level(), desde __init__():

marcador.py

```
def __init__(self, ai_game):
    --recorte--
    self.prep_high_score()
    self.prep_level()
```

Aquí está prep_level():

marcador.py

```
def nivel_preparación(yo):
    """Convierte el nivel en una imagen renderizada."""
    nivel_str = str(self.stats.level)
```



```

1         self.level_image = self.font.render(level_str, Verdadero,
            self.text_color, self.settings.bg_color)

        # Coloque el nivel debajo de la puntuación.
        self.level_rect = self.level_image.get_rect()
2         self.level_rect.right = self.score_rect.right
3         self.level_rect.top = self.score_rect.bottom + 10

```

El método `prep_level()` crea una imagen a partir del valor almacenado en `stats.level` 1 y establece el atributo correcto de la imagen para que coincida con el correcto de la puntuación. atributo 2. Luego establece el atributo superior 10 píxeles debajo de la parte inferior de la imagen de la partitura para dejar espacio entre la partitura y el nivel 3.

También necesitamos actualizar `show_score()`:

marcador.py

```

def show_score(yo):
    """Dibuja puntuaciones y niveles en la pantalla."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)

```

Esta nueva línea dibuja la imagen del nivel en la pantalla.

Incrementaremos `stats.level` y actualizaremos la imagen del nivel en `_check_bullet_colisiones_alienígenas()`:

invasión_alienígena.py

```

def _check_bullet_alien_collisions(yo):
    --recorte--
    si no es uno mismo.extraterrestres:

        # Destruye las balas existentes y crea una nueva flota.
        self.balas.vacio()
        self._create_fleet()
        self.settings.increase_speed()

        # Aumentar el nivel.
        self.stats.level += 1
        self.sb.prep_level()

```

Si una flota es destruida, incrementamos el valor de `stats.level` y llamamos

`prep_level()` para asegurarse de que el nuevo nivel se muestre correctamente.

Para garantizar que la imagen del nivel se actualice correctamente al comienzo de un nuevo juego, También llamamos a `prep_level()` cuando el jugador hace clic en el botón Reproducir:

invasión_alienígena.py

```

def _check_play_button(self, mouse_pos):
    --recorte--
    si se hace clic en el botón y no en self.game_active:
        --recorte--
        self.sb.prep_score()
        self.sb.prep_level()
        --recorte--

```

Llamamos a `prep_level()` justo después de llamar a `prep_score()`.

Ahora verá cuántos niveles ha completado, como se muestra en la Figura 14-5.

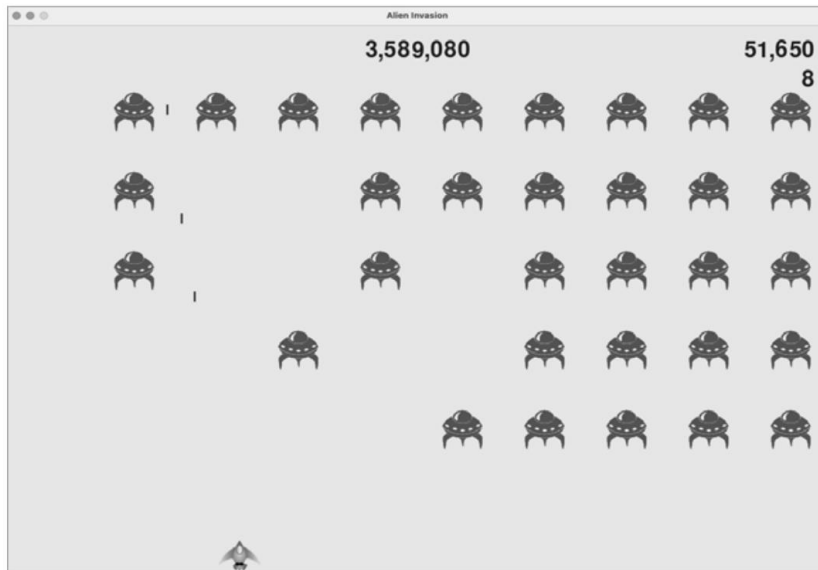


Figura 14-5: El nivel actual aparece justo debajo de la puntuación actual.

NOTA En algunos juegos clásicos, las puntuaciones tienen etiquetas, como Puntuación, Puntuación alta y Nivel.

Hemos omitido estas etiquetas porque el significado de cada número queda claro una vez que has jugado. Para incluir estas etiquetas, agréguelas a las cadenas de puntuación justo antes de las llamadas a `font.render()` en `Scoreboard`.

Visualización del número de barcos

Finalmente, mostremos la cantidad de barcos que le quedan al jugador, pero esta vez usemos un gráfico. Para hacerlo, dibujaremos barcos en la esquina superior izquierda de la pantalla para representar cuántos barcos quedan, tal como lo hacen muchos juegos arcade clásicos.

Primero, necesitamos hacer que `Ship` herede de `Sprite` para poder crear un grupo de barcos:

```
ship.py importar pygame
       desde pygame.sprite importar Sprite
```

Barco de 1 clase (`Sprite`):

```
"""Una clase para gestionar el barco."""

def __init__(self, ai_game):
    """Inicializa el barco y establece su posición inicial."""
2   super().__init__()
    --recorte--
```

Aquí importamos `Sprite`, nos aseguramos de que `Ship` herede de `Sprite 1` y llamamos a `super()` al comienzo de `__init__()` 2.

A continuación, debemos modificar el marcador para crear un grupo de barcos que podamos mostrar. Aquí están las declaraciones de importación para Scoreboard:

```
marcador.py importar pygame.font
del grupo de importación pygame.sprite

desde barco importación Barco
```

Debido a que estamos creando un grupo de barcos, importamos el grupo y el barco. clases.

Aquí está `__init__()`:

```
marcador.py def __init__(self, ai_game):
    """Inicializar atributos de puntuación."""
    self.ai_game = ai_juego
    self.pantalla = ai_game.pantalla
    --recorte--
    self.prep_level()
    self.prep_ships()
```

Asignamos la instancia del juego a un atributo, porque la necesitaremos para crear algunos barcos. Llamamos a `prep_ships()` después de la llamada a `prep_level()`.

Aquí está `prep_ships()`:

```
marcador.py def prep_ships(yo):
    """Muestra cuántos barcos quedan."""
1     self.ships = Grupo()
2     para número_barco en el rango(self.stats.ships_left):
        barco = Barco(self.ai_game)
        barco.rect.x = 10 + número_barco * ancho.rect.barco
        envío.rect.y = 10
5     self.ships.add(enviar)
```

El método `prep_ships()` crea un grupo vacío, `self.ships`, para contener las instancias de barco

1. Para llenar este grupo, se ejecuta un bucle una vez por cada barco que el jugador ha dejado
2. Dentro del bucle, creamos un nuevo barco y configuramos el valor de la coordenada x de cada barco para que los barcos aparezcan uno al lado del otro con un margen de 10 píxeles en el lado izquierdo del grupo de barcos
3. Establecemos el valor de la coordenada y 10 píxeles hacia abajo desde la parte superior de la pantalla para que los barcos Aparecen en la esquina superior izquierda de la pantalla
4. Luego agregamos cada barco nuevo al grupo de barcos
- 5.

Ahora necesitamos dibujar los barcos en la pantalla:

```
marcador.py def show_score(yo):
    """Dibuja puntuaciones, niveles y barcos en la pantalla."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
    self.ships.draw (auto.pantalla)
```

Para mostrar los barcos en la pantalla, llamamos a `draw()` en el grupo, y Pygame dibuja cada barco.

Para mostrarle al jugador con cuántos barcos tiene para comenzar, llamamos a `prep_ships()` cuando comienza un nuevo juego. Hacemos esto en `_check_play_button()` en *Invasión alienígena*:

```
invasión_alienígena.py    def _check_play_button(self, mouse_pos):
                           --recorte--
                           si se hace clic en el botón y no en self.game_active:
                               --recorte--
                               self.sb.prep_level()
                               self.sb.prep_ships()
                               --recorte--
```

También llamamos a `prep_ships()` cuando un barco es impactado, para actualizar la visualización del barco. Imágenes cuando el jugador pierde un barco:

```
invasión_alienígena.py    def _ship_hit(yo):
                           """Responder al barco que es golpeado por un extraterrestre."""
                           si self.stats.ships_left > 0:
                               # Disminuir barcos_izquierdos y actualizar el marcador.
                               self.stats.ships_left -= 1
                               self.sb.prep_ships()
                               --recorte--
```

Llamamos a `prep_ships()` después de disminuir el valor de `ships_left`, por lo que el número correcto de barcos restantes se muestra cada vez que se destruye un barco.

La Figura 14-6 muestra el sistema de puntuación completo, con los barcos restantes mostrados en la parte superior izquierda de la pantalla.

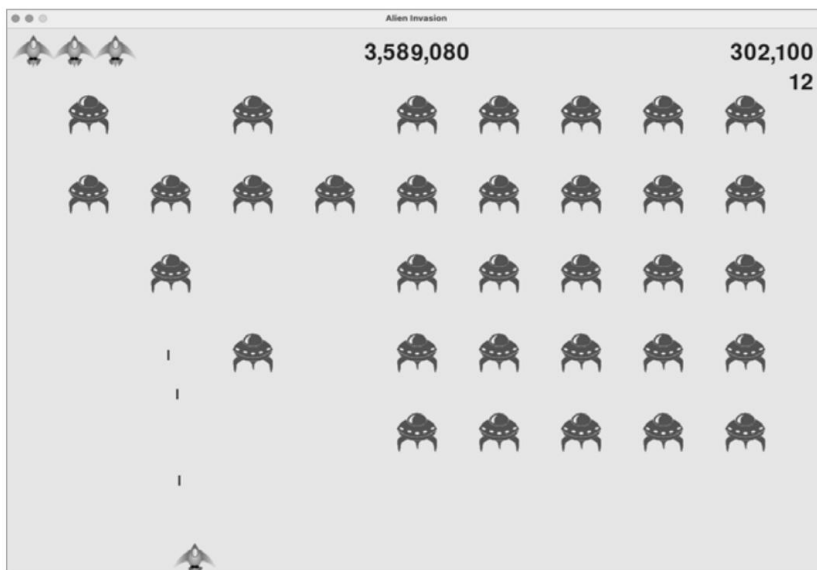


Figura 14-6: El sistema de puntuación completo de Alien Invasion

INTÉNTALO TÚ MISMO

14-5. Puntuación más alta de todos los tiempos: la puntuación más alta se restablece cada vez que un jugador cierra y reinicia Alien Invasion. Solucione este problema escribiendo la puntuación más alta en un archivo antes de llamar a `sys.exit()` y leyendo la puntuación más alta al inicializar su valor en `GameStats`.

14-6. Refactorización: busque métodos que realicen más de una tarea y refactorícelos para organizar su código y hacerlo eficiente. Por ejemplo, mueva parte del código en `_check_bullet_alien_collisions()`, que inicia un nuevo nivel cuando la flota alienígena ha sido destruida, a una función llamada `inicio_nuevo_nivel()`. Además, mueva las cuatro llamadas a métodos separados en el método `__init__()` en `Scoreboard` a un método llamado `prep_images()` para acortar `__init__()`. la preparación. El método `_images()` también podría ayudar a simplificar `_check_play_button()` o `start_game()` si ya has refactorizado `_check_play_button()`.

NOTA Antes de intentar refactorizar el proyecto, consulte el Apéndice D para aprender cómo restaurar el proyecto a un estado de funcionamiento si introduce errores durante la refactorización.

14-7. Ampliando el juego: piensa en una manera de expandir Alien Invasion. Por ejemplo, puedes programar a los extraterrestres para que disparen balas a tu nave. También puedes agregar escudos para que tu nave se esconda detrás, los cuales pueden ser destruidos por balas desde cualquier lado. O puedes usar algo como el módulo `pygame.mixer` para agregar efectos de sonido, como explosiones y sonidos de disparos.

14-8. Sideways Shooter, Versión Final: Continuar desarrollando Sideways Shooter, utilizando todo lo que hemos hecho en este proyecto. Agregue un botón Jugar, haga que el juego se acelere en los puntos apropiados y desarrolle un sistema de puntuación. Asegúrate de refactorizar tu código mientras trabajas y busca oportunidades para personalizar el juego más allá de lo que se ha mostrado en este capítulo.

Resumen

En este capítulo, aprendiste cómo implementar un botón Jugar para iniciar un nuevo juego. También aprendiste cómo detectar eventos del mouse y ocultar el cursor en juegos activos. Puedes usar lo que has aprendido para crear otros botones, como un botón de Ayuda para mostrar instrucciones sobre cómo jugar a tus juegos. También aprendió a modificar la velocidad de un juego a medida que avanza, implementar un sistema de puntuación progresivo y mostrar información en forma textual y no textual.

15

GENERACIÓN DE DATOS



La visualización de datos es el uso de representaciones visuales para explorar y presentar patrones en conjuntos de datos. Está estrechamente asociado con el análisis de datos, que utiliza código para explorar los patrones y conexiones en un conjunto de datos. Un conjunto de datos puede ser una pequeña lista de números que caben en una sola línea de código, o pueden ser terabytes de datos que incluyen muchos tipos diferentes de información.

Crear visualizaciones de datos efectivas es algo más que simplemente hacer que la información se vea bien. Cuando una representación de un conjunto de datos es simple y visualmente atractiva, su significado queda claro para los espectadores. Las personas verán patrones y significados en sus conjuntos de datos que nunca supieron que existían.

Afortunadamente, no se necesita una supercomputadora para visualizar datos complejos. Python es tan eficiente que con solo una computadora portátil, puedes explorar rápidamente conjuntos de datos que contienen millones de puntos de datos individuales. Estos puntos de datos no

tienen que ser números; Con los conceptos básicos que aprendió en la primera parte de este libro, también puede analizar datos no numéricos.

La gente usa Python para trabajos con uso intensivo de datos en genética, investigación climática, análisis político y económico, y mucho más. Los científicos de datos han escrito una impresionante variedad de herramientas de visualización y análisis en Python, muchas de las cuales también están disponibles para usted. Una de las herramientas más populares es Matplotlib, una biblioteca de trazado matemático. En este capítulo, usaremos Matplotlib para crear gráficos simples, como gráficos de líneas y diagramas de dispersión. Luego crearemos un conjunto de datos más interesante basado en el concepto de paseo aleatorio: una visualización generada a partir de una serie de decisiones aleatorias.

También usaremos un paquete llamado Plotly, que crea visualizaciones que funcionan bien en dispositivos digitales, para analizar los resultados de tirar los dados. Plotly genera visualizaciones que cambian de tamaño automáticamente para adaptarse a una variedad de dispositivos de visualización. Estas visualizaciones también pueden incluir una serie de características interactivas, como enfatizar aspectos particulares del conjunto de datos cuando los usuarios pasan el cursor sobre diferentes partes de la visualización. Aprender a usar Matplotlib y Plotly le ayudará a empezar a visualizar los tipos de datos que más le interesan.

Instalación de Matplotlib

Para usar Matplotlib para su conjunto inicial de visualizaciones, necesitará instalarlo usando pip, tal como lo hicimos con pytest en el Capítulo 11 (consulte “Instalación de pytest con pip” en la página 210).

Para instalar Matplotlib, ingrese el siguiente comando en el símbolo del terminal:

```
$ python -m pip instalar --user matplotlib
```

Si utiliza un comando que no sea Python para ejecutar programas o iniciar un terminal, como python3, su comando se verá así:

```
$ python3 -m pip instalar --user matplotlib
```

Para ver los tipos de visualizaciones que puede realizar con Matplotlib, visite la página de inicio de Matplotlib en <https://matplotlib.org> y haga clic en Tipos de trazado. Al hacer clic en una visualización en la galería, verá el código utilizado para generar el gráfico.

Trazar un gráfico lineal simple

Tracemos un gráfico de líneas simple usando Matplotlib y luego personalicémoslo para crear una visualización de datos más informativa. Usaremos la secuencia de números cuadrados 1, 4, 9, 16 y 25 como datos para la gráfica.

Para hacer un gráfico lineal simple, especifique los números con los que desea trabajar con y deje que Matplotlib haga el resto:

```
mpl_squares.py importar matplotlib.pyplot como plt
```

```
cuadrados = [1, 4, 9, 16, 25]
```



```
1 fig, ax = plt.subplots()
  ax.plot(cuadrados)

plt.show()
```

Primero importamos el módulo pyplot usando el alias `plt` para no tener que escribir `pyplot` repetidamente. (Verá esta convención con frecuencia en ejemplos en línea, por lo que la usaremos aquí). El módulo `pyplot` contiene una serie de funciones que ayudan a generar gráficos y diagramas.

Creamos una lista llamada `cuadrados` para contener los datos que trazaremos. Entonces nosotros siga otra convención común de Matplotlib llamando a `subplots()` función 1. Esta función puede generar uno o más gráficos en la misma figura. La variable `fig` representa la figura completa, que es la colección de gráficos que se generan. La variable `ax` representa un único gráfico en la figura; esta es la variable que usaremos la mayor parte del tiempo al definir y personalizar un gráfico único.

Luego usamos el método `plot()`, que intenta trazar los datos que se proporcionan de una manera significativa. La función `plt.show()` abre el visor de Matplotlib y muestra el gráfico, como se muestra en la Figura 15-1. El visor le permite hacer zoom y navegar por la trama, y puede guardar cualquier imagen de la trama que desee haciendo clic en el icono del disco.

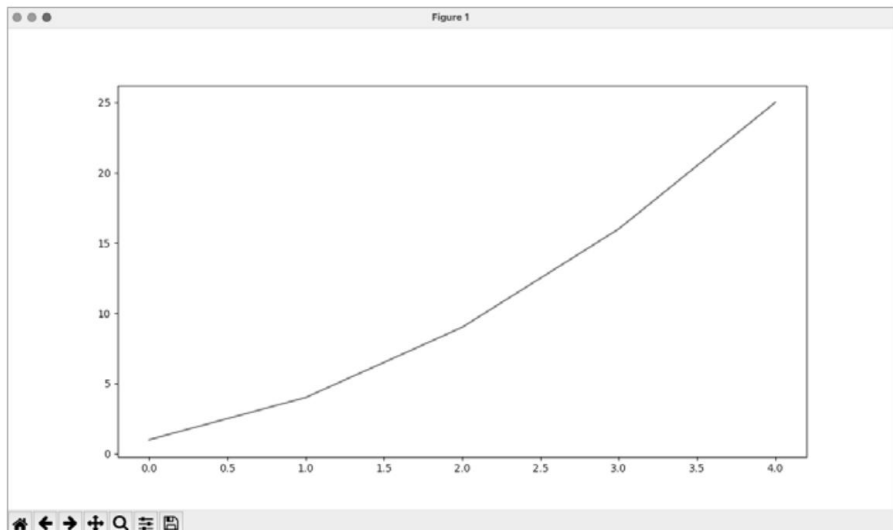


Figura 15-1: Uno de los gráficos más simples que puedes hacer en Matplotlib

Cambiar el tipo de etiqueta y el grosor de la línea

Aunque el gráfico de la Figura 15-1 muestra que los números están aumentando, el tipo de etiqueta es demasiado pequeño y la línea un poco delgada para leerla fácilmente. Afortunadamente, Matplotlib te permite ajustar cada característica de una visualización.

Usaremos algunas de las personalizaciones disponibles para mejorar la trama de esta trama. legibilidad. Comencemos agregando un título y etiquetando los ejes:

```
mpl_squares.py importar matplotlib.pyplot como plt

cuadrados = [1, 4, 9, 16, 25]

higo, hacha = plt.subplots()
1 diagrama de hacha (cuadrados, ancho de línea = 3)

# Establecer el título del gráfico y los ejes de etiquetas.
2 ax.set_title("Números cuadrados", tamaño de fuente=24)
3 ax.set_xlabel("Valor", tamaño de fuente=14)
  ax.set_ylabel("Cuadrado de valor", tamaño de fuente=14)

# Establecer el tamaño de las etiquetas de marca.
4 ax.tick_params (tamaño de etiqueta = 14)

plt.mostrar()
```

El parámetro ancho de línea controla el grosor de la línea que traza() genera 1. Una vez que se ha generado un gráfico, hay muchos métodos disponibles para modificarlo antes de presentarlo. El método `set_title()` establece un título general para el gráfico 2. Los parámetros de tamaño de fuente , que aparecen repetidamente en todo el código, controlan el tamaño del texto en varios elementos del gráfico.

Los métodos `set_xlabel()` y `set_ylabel()` le permiten establecer un título para cada uno de los ejes 3, y el método `tick_params()` estiliza las marcas de graduación 4. Aquí `tick_params()` establece el tamaño de fuente de las etiquetas de las marcas de graduación en 14 en ambos ejes.

Como puede ver en la Figura 15-2, el gráfico resultante es mucho más fácil de leer. El tipo de etiqueta es más grande y el gráfico de líneas es más grueso. A menudo vale la pena experimentar con estos valores para ver qué funciona mejor en el gráfico resultante.

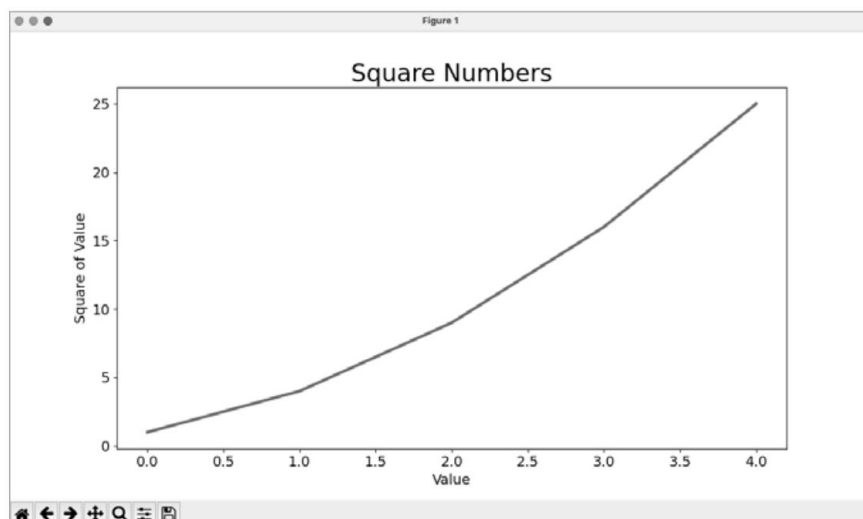


Figura 15-2: El gráfico es mucho más fácil de leer ahora.

Corrigiendo la trama

Ahora que podemos leer mejor el gráfico, podemos ver que los datos no están trazados correctamente. ¡Observe que al final del gráfico el cuadrado de 4,0 se muestra como 25! Arreglemos eso.

Cuando le das a `plot()` una única secuencia de números, se supone que el primer punto de datos corresponde a un valor x de 0, pero nuestro primer punto corresponde a un valor x de 1. Podemos anular el comportamiento predeterminado dando `plot()` los valores de entrada y salida utilizados para calcular los cuadrados:

```
mpl_squares.py importar matplotlib.pyplot como plt
```

```
valores_entrada = [1, 2, 3, 4, 5]
cuadrados = [1, 4, 9, 16, 25]

higo, hacha = plt.subplots()
ax.plot(valores_entrada, cuadrados, ancho de línea=3)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

Ahora `plot()` no tiene que hacer ninguna suposición sobre cómo se generaron los números de salida. El gráfico resultante, que se muestra en la Figura 15-3, es correcto.

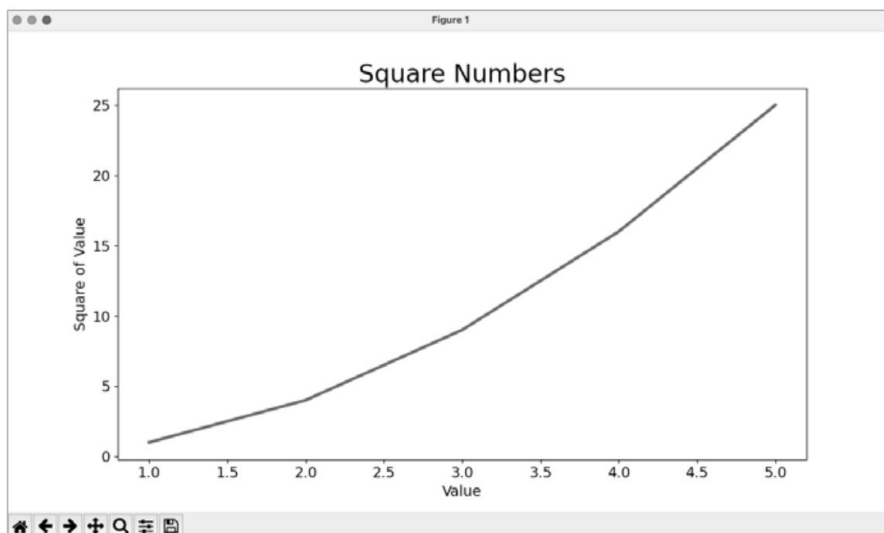


Figura 15-3: Los datos ahora están trazados correctamente.

Puede especificar una cantidad de argumentos al llamar a `plot()` y utilizar varios métodos para personalizar sus gráficos después de generarlos. Continuaremos explorando estos enfoques de personalización a medida que trabajemos con conjuntos de datos más interesantes a lo largo de este capítulo.

Usar estilos integrados

Matplotlib tiene varios estilos predefinidos disponibles. Estos estilos contienen una variedad de configuraciones predeterminadas para colores de fondo, líneas de cuadrícula, anchos de línea, fuentes, tamaños de fuente y más. Pueden hacer que sus visualizaciones sean atractivas sin requerir mucha personalización. Para ver la lista completa de estilos disponibles, ejecute las siguientes líneas en una sesión de terminal:

```
>>> importar matplotlib.pyplot como plt
>>> plt.estilo.disponible
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery',
--recorte--
```

Para usar cualquiera de estos estilos, agregue una línea de código antes de llamar a `subplots()`:

```
mpl_squares.py importar matplotlib.pyplot como plt
```

```
valores_entrada = [1, 2, 3, 4, 5]
cuadrados = [1, 4, 9, 16, 25]
```

```
plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
--recorte--
```

Este código genera el gráfico que se muestra en la Figura 15-4. Una amplia variedad de los estilos están disponibles; Juega con estos estilos para encontrar alguno que te guste.

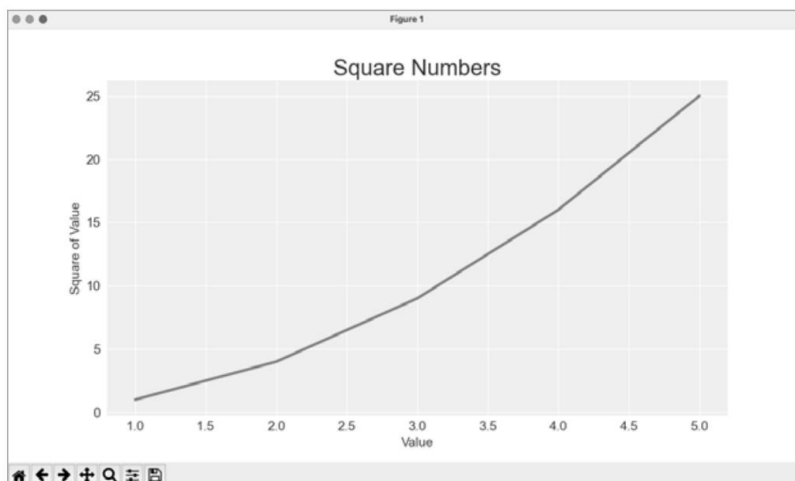


Figura 15-4: El estilo Seaborn incorporado

Trazar y diseñar puntos individuales con `scatter()`

A veces, resulta útil trazar y diseñar puntos individuales en función de determinadas características. Por ejemplo, puede trazar valores pequeños en un color y valores más grandes en un color diferente. También puede trazar un conjunto de datos grande con un conjunto de opciones de estilo y luego enfatizar puntos individuales volviendo a trazarlos con diferentes opciones.

Para trazar un solo punto, pase los valores únicos x e y del punto a scatter():

```
dispersión
_cuadrados.py

importar matplotlib.pyplot como plt

plt.style.use('seaborn') fig, ax =
plt.subplots() ax.scatter(2, 4)
```

```
plt.mostrar()
```

Diseñemos el resultado para hacerlo más interesante. Agregaremos un título, etiqueta de los ejes y asegúrese de que todo el texto sea lo suficientemente grande como para leer:

```
importar matplotlib.pyplot como plt

plt.style.use('seaborn') fig, ax =
plt.subplots() 1 ax.scatter(2, 4,
s=200)

# Establecer el título del gráfico y los ejes de etiquetas.
ax.set_title("Números cuadrados", tamaño de fuente=24)
ax.set_xlabel("Valor", tamaño de fuente=14)
ax.set_ylabel("Cuadrado de valor", tamaño de fuente=14)

# Establecer el tamaño de las etiquetas de marca.
ax.tick_params(tamaño de etiqueta=14)

plt.mostrar()
```

Llamamos a scatter() y usamos el argumento s para establecer el tamaño de los puntos utilizados para dibujar el gráfico 1. Cuando ejecute scatter_squares.py ahora, debería ver un solo punto en el medio del gráfico, como se muestra en la Figura 15. -5.

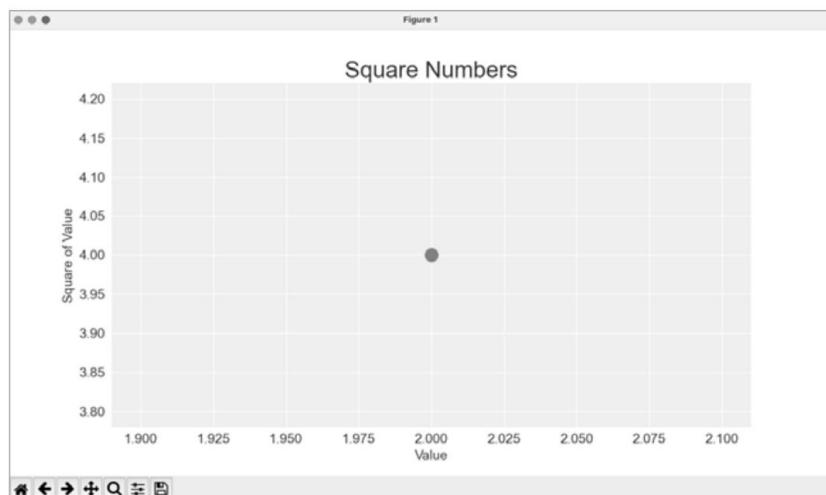


Figura 15-5: Trazar un solo punto

Trazar una serie de puntos con dispersión()

Para trazar una serie de puntos, podemos pasar `scatter()` listas separadas de valores de `x` e `y`, como esta:

```
dispersión
_cuadrados.py

importar matplotlib.pyplot como plt

valores_x = [1, 2, 3, 4, 5]
valores_y = [1, 4, 9, 16, 25]

plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
ax.scatter(valores_x, valores_y, s=100)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

La lista `x_values` contiene los números que se van a elevar al cuadrado y `y_values` contiene el cuadrado de cada número. Cuando estas listas se pasan a `scatter()`, Matplotlib lee un valor de cada lista mientras traza cada punto. Los puntos a trazar son (1, 1), (2, 4), (3, 9), (4, 16) y (5, 25); La figura 15-6 muestra el resultado.

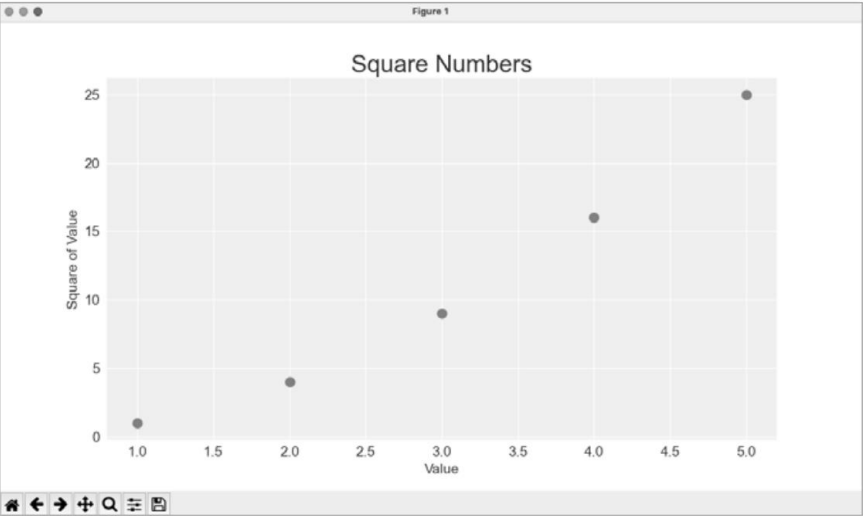


Figura 15-6: Un diagrama de dispersión con múltiples puntos

Calcular datos automáticamente

Escribir listas a mano puede resultar ineficiente, especialmente cuando tenemos muchos puntos. En lugar de escribir cada valor, usemos un bucle para hacer los cálculos por nosotros.

Así es como se vería esto con 1000 puntos:

```
dispersión
_cuadrados.py

1 valores_x = rango(1, 1001)
  valores_y = [x**2 para x en valores_x]
```

```

plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
2 ax.scatter(valores_x, valores_y, s=10)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--

# Establezca el rango para cada eje.
3 eje.eje([0, 1100, 0, 1_100_000])

plt.mostrar()

```

Comenzamos con un rango de valores de x que contienen los números del 1 al 1000
 1. A continuación, una lista por comprensión genera los valores de y recorriendo los valores de x (para x en valores_x), elevando al cuadrado cada número (x^2), y asignando los resultados a y_values. Luego pasamos las listas de entrada y salida a scatter(). 2. Debido a que se trata de un conjunto de datos grande, utilizamos un tamaño de puntos más pequeño.

Antes de mostrar el gráfico, utilizamos el método axis() para especificar el rango de cada eje 3. El método axis() requiere cuatro valores: los valores mínimo y máximo para el eje x y el eje y. Aquí, corremos el eje x de 0 a 1100 y el eje y de 0 a 1100000. La figura 15-7 muestra el resultado.

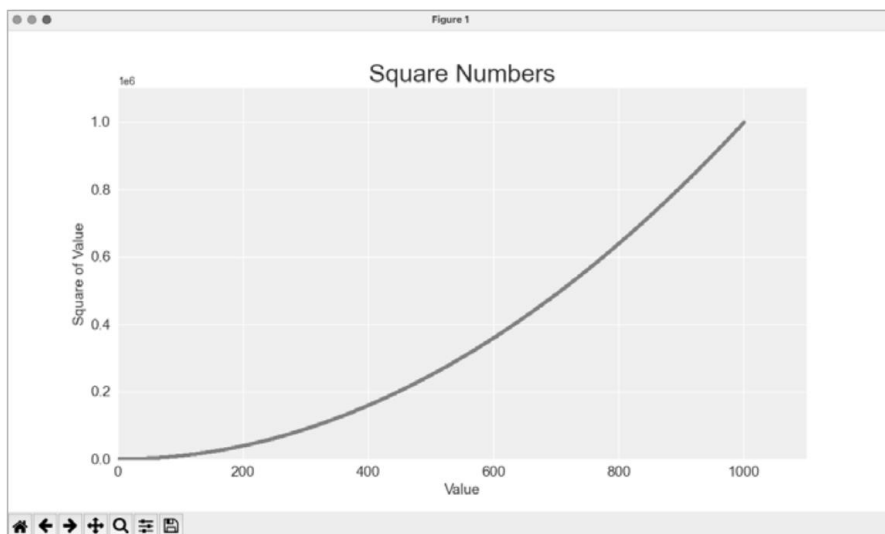


Figura 15-7: Python puede trazar 1000 puntos con la misma facilidad con la que traza 5 puntos.

Personalización de etiquetas de ticks

Cuando los números en un eje crecen lo suficiente, Matplotlib utiliza de forma predeterminada la notación científica para las etiquetas de marcas. Esto suele ser algo bueno, porque los números más grandes en notación simple ocupan mucho espacio innecesario en una visualización.

Casi todos los elementos de un gráfico son personalizables, por lo que puedes decirle a Matplotlib que siga usando notación simple si lo prefieres:

```
--recorte--
# Establezca el rango para cada eje.
eje.eje([0, 1100, 0, 1_100_000])
ax.ticklabel_format(estilo='simple')

plt.mostrar()
```

El método `ticklabel_format()` le permite anular el estilo de etiqueta de marca predeterminado para cualquier gráfico.

Definición de colores personalizados

Para cambiar el color de los puntos, pase el argumento `color` a `scatter()` con el nombre de un color a usar entre comillas, como se muestra aquí:

```
ax.scatter(valores_x, valores_y, color='rojo', s=10)
```

También puede definir colores personalizados utilizando el modelo de color RGB. Para definir un color, pase el argumento de color una tupla con tres valores flotantes (uno para rojo, uno para verde y otro para azul, en ese orden), usando valores entre 0 y 1.

Por ejemplo, la siguiente línea crea un gráfico con puntos de color verde claro:

```
ax.scatter(valores_x, valores_y, color=(0, 0.8, 0), s=10)
```

Los valores más cercanos a 0 producen colores más oscuros y los valores más cercanos a 1 producen colores más claros.

Usando un mapa de colores

Un mapa de colores es una secuencia de colores en un degradado que se mueve desde un color inicial hasta un color final. En las visualizaciones, los mapas de colores se utilizan para enfatizar patrones en los datos. Por ejemplo, puede convertir los valores bajos en un color claro y los valores altos en un color más oscuro. El uso de un mapa de colores garantiza que todos los puntos de la visualización varíen de manera suave y precisa a lo largo de una escala de colores bien diseñada.

El módulo `pyplot` incluye un conjunto de mapas de colores integrados. Para utilizar uno de estos mapas de colores, debe especificar cómo `pyplot` debe asignar un color a cada punto del conjunto de datos. A continuación se explica cómo asignar un color a cada punto, según su valor `y`:

```
dispersión
_cuadrados.py
--recorte--
plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
ax.scatter(valores_x, valores_y, c=valores_y, cmap=plt.cm.Blues, s=10)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

El argumento `c` es similar al `color`, pero se usa para asociar una secuencia de valores con un mapeo de colores. Pasamos la lista de valores de `y` a `c`, y luego

Indíquele a pyplot qué mapa de colores usar con el argumento `cmap`. Este código colorea los puntos con valores de y más bajos en azul claro y los puntos con valores de y más altos en azul oscuro. La Figura 15-8 muestra el gráfico resultante.

NOTA Puede ver todos los mapas de colores disponibles en pyplot en <https://matplotlib.org>. Vaya a Tutoriales, desplácese hacia abajo hasta Colores y haga clic en Elegir mapas de colores en Matplotlib.

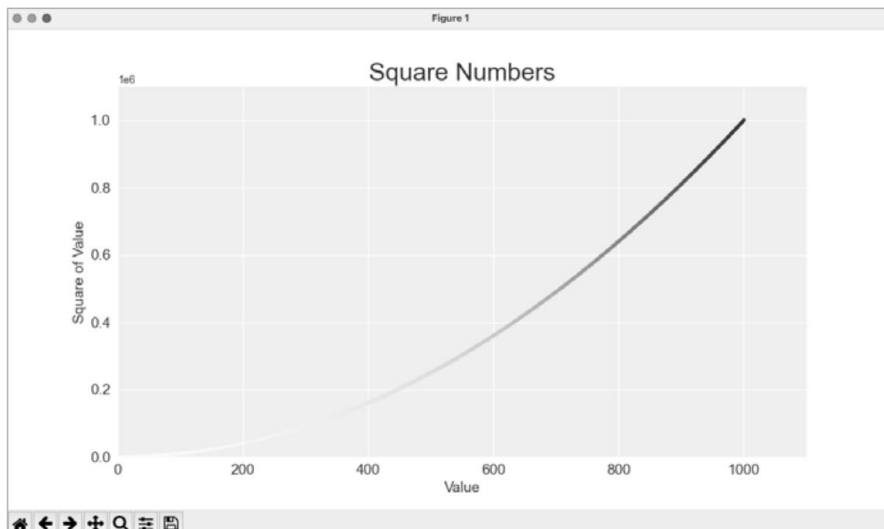


Figura 15-8: Un gráfico utilizando el mapa de colores Blues

Guardar sus parcelas automáticamente

Si desea guardar el gráfico en un archivo en lugar de mostrarlo en el visor Matplotlib, puede usar `plt.savefig()` en lugar de `plt.show()`:

```
plt.savefig('squares_plot.png', bbox_inches='apretado')
```

El primer argumento es un nombre de archivo para la imagen del trazado, que se guardará en el mismo directorio que `scatter_squares.py`. El segundo argumento recorta los espacios en blanco adicionales del gráfico. Si desea espacios en blanco adicionales alrededor del gráfico, puede omitir este argumento. También puede llamar a `savefig()` con un objeto `Path` y escribir el archivo de salida en cualquier lugar que desee de su sistema.

INTÉNTALO TÚ MISMO

15-1. Cubos: Un número elevado a la tercera potencia es un cubo. Traza los primeros cinco números cúbicos y luego traza los primeros 5000 números cúbicos.

15-2. Cubos de colores: aplique un mapa de colores al gráfico de sus cubos.

Paseos aleatorios

En esta sección, usaremos Python para generar datos para un recorrido aleatorio y luego usaremos Matplotlib para crear una representación visualmente atractiva de esos datos. Un paseo aleatorio es un camino determinado por una serie de decisiones simples, cada una de las cuales se deja enteramente al azar. Podríamos imaginar un paseo aleatorio como el camino que tomaría una hormiga confundida si diera cada paso en una dirección aleatoria.

Los paseos aleatorios tienen aplicaciones prácticas en la naturaleza, la física, la biología, la química y la economía. Por ejemplo, un grano de polen que flota en una gota de agua se mueve a través de la superficie del agua porque las moléculas de agua lo empujan constantemente. El movimiento molecular en una gota de agua es aleatorio, por lo que el camino que traza un grano de polen en la superficie es un paseo aleatorio. El código que escribiremos a continuación modela muchas situaciones del mundo real.

Creando la clase RandomWalk

Para crear una caminata aleatoria, crearemos una clase RandomWalk , que tomará decisiones aleatorias sobre qué dirección debe tomar la caminata. La clase necesita tres atributos: una variable para realizar un seguimiento del número de puntos del recorrido y dos listas para almacenar las coordenadas x e y de cada punto del recorrido.

Sólo necesitaremos dos métodos para la clase RandomWalk : el `__init__()` método y `fill_walk()`, que calculará los puntos en la caminata. Comencemos con el método `__init__()` :

```
aleatorio 1 de elección de importación aleatoria
_walk.py

class Paseo aleatorio:
    """Una clase para generar paseos aleatorios."""

    2 def __init__(self, num_puntos=5000):
        """Inicializar atributos de un paseo."""
        self.num_puntos = num_puntos

        # Todos los paseos comienzan en (0, 0).
    3         self.x_valores = [0]
            self.y_valores = [0]
```

Para tomar decisiones aleatorias, almacenaremos posibles movimientos en una lista y usaremos la función `Choice()` (del módulo `aleatorio`) para decidir qué movimiento hacer cada vez que se da un paso 1. Establecemos el número predeterminado de puntos en una caminata en 5000, que es lo suficientemente grande como para generar algunos patrones interesantes pero lo suficientemente pequeño para generar caminatas rápidamente 2. Luego hacemos dos listas para contener los valores de x e y, y comenzamos cada caminata en el punto (0, 0) 3.

Elegir direcciones

Usaremos el método `fill_walk()` para determinar la secuencia completa de puntos en la caminata. Agregue este método a `random_walk.py`:

```
aleatorio
_walk.py

def fill_walk(yo):
    """Calcular todos los puntos del recorrido."""
```

```

1      # Siga dando pasos hasta que la caminata alcance la longitud deseada.
      mientras que len(self.x_values) < self.num_points:

2          # Decide en qué dirección ir y hasta dónde ir.
          dirección_x = elección([1, -1])
          distancia_x = elección([0, 1, 2, 3, 4])
3          x_paso = x_dirección * x_distancia

          dirección_y = elección([1, -1])
          distancia_y = elección([0, 1, 2, 3, 4])
4          paso_y = dirección_y * distancia_y

5          # Rechaza movimientos que no van a ninguna parte.
          si paso_x == 0 y paso_y == 0:
              continuar

          # Calcular la nueva posición.
6          x = self.x_valores[-1] + x_paso
          y = self.y_valores[-1] + y_paso

          self.x_values.append(x)
          self.y_values.append(y)

```

Primero configuramos un bucle que se ejecuta hasta que la caminata se llena con el número correcto de puntos 1. La parte principal de `fill_walk()` le dice a Python cómo simular cuatro decisiones aleatorias: ¿La caminata irá hacia la derecha o hacia la izquierda? ¿Hasta dónde llegará en esa dirección? ¿Subirá o bajará? ¿Hasta dónde llegará en esa dirección?

Usamos `elección([1, -1])` para elegir un valor para `x_direction`, que devuelve 1 para el movimiento hacia la derecha o -1 para el movimiento hacia la izquierda 2. A continuación, `elección([0, 1, 2, 3, 4])` selecciona aleatoriamente una distancia para moverse en esa dirección. Asignamos este valor a `x_distancia`. La inclusión de un 0 permite la posibilidad de pasos que tengan movimiento a lo largo de un solo eje.

Determinamos la longitud de cada paso en las direcciones x e y multiplicando la dirección del movimiento por la distancia elegida 3 4. Un resultado positivo para `x_step` significa moverse hacia la derecha, un resultado negativo significa moverse hacia la izquierda, y 0 significa moverse verticalmente. Un resultado positivo para `y_step` significa moverse hacia arriba, negativo significa moverse hacia abajo y 0 significa moverse horizontalmente. Si los valores de `x_step` e `y_step` son 0, la caminata no llega a ninguna parte; cuando esto sucede, continuamos el bucle 5.

Para obtener el siguiente valor de x para la caminata, sumamos el valor en `x_step` al último valor almacenado en `x_values` 6 y hacemos lo mismo con los valores de y. Cuando tenemos las coordenadas del nuevo punto, las agregamos a `valores_x` e `valores_y`.

Trazar el paseo aleatorio

Aquí está el código para trazar todos los puntos de la caminata:

```
rw_visual.py importar matplotlib.pyplot como plt
```

```
desde random_walk importar RandomWalk
```

```
# Haz una caminata aleatoria.
```

```

1 rw = Paseo aleatorio()
  rw.fill_walk()

# Traza los puntos en el recorrido.
plt.style.use('clásico')
higo, hacha = plt.subplots()
2 ax.scatter(rw.x_values, rw.y_values, s=15)
3 ax.set_aspect('igual')
  plt.mostrar()

```

Comenzamos importando pyplot y RandomWalk. Luego creamos un paseo aleatorio y lo asignamos a `rw` 1, asegurándonos de llamar a `fill_walk()`. Para visualizar la caminata, alimentamos los valores `x` e `y` de la caminata a `scatter()` y elegimos un tamaño de punto apropiado 2. De forma predeterminada, Matplotlib escala cada eje de forma independiente. Pero ese enfoque extendería la mayoría de las caminatas horizontal o verticalmente. Aquí usamos el método `set_aspect('igual')` para especificar que ambos ejes deben tener el mismo espacio entre las marcas 3.

La Figura 15-9 muestra el gráfico resultante con 5000 puntos. Las imágenes de esta sección omiten el visor de Matplotlib, pero seguirás viéndolo cuando ejecutes `rw_visual.py`.

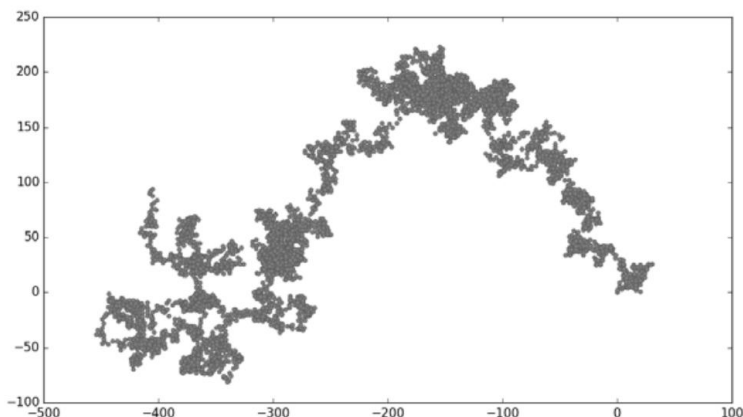


Figura 15-9: Un paseo aleatorio con 5000 puntos

Generando múltiples paseos aleatorios

Cada paseo aleatorio es diferente y es divertido explorar los distintos patrones que se pueden generar. Una forma de utilizar el código anterior para realizar múltiples recorridos sin tener que ejecutar el programa varias veces es envolverlo en un bucle `while`, como este:

```
rw_visual.py importar matplotlib.pyplot como plt
```

```
desde random_walk importar RandomWalk
```

```
# Sigue realizando nuevas caminatas, mientras el programa esté activo.
```

mientras que Verdadero:

```
# Haz una caminata aleatoria.

--recorte--

plt.mostrar()

keep_running = input("¿Hacer otro paseo? (s/n): ")
si keep_running == 'n':
    romper
```

Este código genera un recorrido aleatorio, lo muestra en el visor de Matplotlib y se detiene con el visor abierto. Cuando cierres el visor, se te preguntará si deseas generar otro recorrido. Si genera algunas caminatas, debería ver algunas que permanecen cerca del punto de inicio, algunas que se desvían principalmente en una dirección, algunas que tienen secciones delgadas que conectan grupos más grandes de puntos y muchos otros tipos de caminatas. Cuando desee finalizar el programa, presione N.

Diseñar la caminata

En esta sección, personalizaremos nuestras tramas para enfatizar las características importantes de cada caminata y restar importancia a los elementos que distraen. Para ello, identificamos las características que queremos resaltar, como dónde empezó la caminata, dónde terminó y el camino recorrido. A continuación, identificamos las características a restar importancia, como marcas y etiquetas. El resultado debe ser una representación visual sencilla que comunique claramente el camino seguido en cada paseo aleatorio.

Coloreando los puntos

Usaremos un mapa de colores para mostrar el orden de los puntos en el recorrido y eliminaremos el contorno negro de cada punto para que el color de los puntos sea más claro. Para colorear los puntos según su posición en el recorrido, le pasamos al argumento `c` una lista que contiene la posición de cada punto. Debido a que los puntos están trazados en orden, esta lista solo contiene los números del 0 al 4999:

`rw_visual.py` --recorte--

mientras que Verdadero:

```
# Haz una caminata aleatoria.

rw = Paseo aleatorio()
rw.fill_walk()

# Traza los puntos en el recorrido.
plt.style.use('clásico')
higo, hacha = plt.subplots(
1 números_puntos = rango(rw.num_puntos)
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
               colores de borde='ninguno', s=15)
    ax.set_aspect('igual')
    plt.mostrar()
--recorte--
```

Usamos `range()` para generar una lista de números igual a la cantidad de puntos en la caminata 1. Asignamos esta lista a `point_numbers`, que usaremos para establecer el color de cada punto en la caminata. Pasamos `point_numbers` al argumento `c`, usamos el mapa de colores `Blues` y luego pasamos `edgecolors='none'` para eliminar el contorno negro alrededor de cada punto. El resultado es un gráfico que varía del azul claro al azul oscuro, que muestra exactamente cómo se mueve la caminata desde su punto inicial hasta su punto final. Esto se muestra en la Figura 15-10.

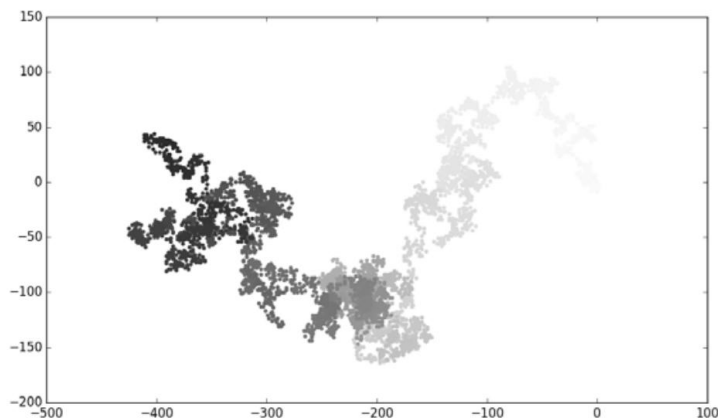


Figura 15-10: Un paseo aleatorio coloreado con el mapa de colores Azules

Trazar los puntos inicial y final

Además de colorear los puntos para mostrar su posición a lo largo del recorrido, sería útil ver exactamente dónde comienza y termina cada recorrido. Para hacerlo, podemos trazar el primer y el último punto individualmente después de trazar la serie principal. Agrandaremos los puntos finales y los colorearemos de manera diferente para que se destaquen:

```
rw_visual.py --recorte--
mientras que Verdadero:
    --recorte--
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
               colores de borde='ninguno', s=15)
    ax.set_aspect('igual')

    # Enfatice el primer y el último punto.
    ax.scatter(0, 0, c='verde', colores de borde='ninguno', s=100)
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='rojo', edgecolors='ninguno',
               s=100)

    plt.mostrar()
    --recorte--
```

Para mostrar el punto inicial, trazamos el punto (0, 0) en verde y en un tamaño mayor (`s=100`) que el resto de puntos. Para marcar el punto final,

Trace también los últimos valores de x e y en rojo con un tamaño de 100. Asegúrese de insertar este código justo antes de llamar a `plt.show()` para que los puntos inicial y final se dibujen encima de todos los demás puntos.

Cuando ejecute este código, debería poder detectar exactamente dónde comienza y termina cada recorrido. Si estos puntos finales no se destacan con suficiente claridad, ajuste su color y tamaño hasta que lo hagan.

Limpiar las hachas

Quitemos los ejes de este gráfico para que no distraigan el camino de cada paseo. Aquí se explica cómo ocultar los ejes:

```
rw_visual.py --recorte--
    mientras que Verdadero:
        --recorte--
        ax.scatter(rw.x_values[-1], rw.y_values[-1], c='rojo', edgecolors='ninguno',
                    s=100)

        # Retire los ejes.
        ax.get_xaxis().set_visible(Falso)
        ax.get_yaxis().set_visible(Falso)

    plt.mostrar()
--recorte--
```

Para modificar los ejes, utilizamos los métodos `ax.get_xaxis()` y `ax.get_yaxis()` para obtener cada eje, y luego encadenamos el método `set_visible()` para hacer que cada eje sea invisible. A medida que continúe trabajando con visualizaciones, verá con frecuencia este encadenamiento de métodos para personalizar diferentes aspectos de una visualización.

Ejecute `rw_visual.py` ahora; Deberías ver una serie de gráficos sin ejes.

Agregar puntos de trama

Aumentemos el número de puntos, para tener más datos con los que trabajar. Para ello, aumentamos el valor de `num_points` cuando realizamos un `RandomWalk`. instancia y ajuste el tamaño de cada punto al dibujar el gráfico:

```
rw_visual.py --recorte--
    mientras que Verdadero:
        # Haz una caminata aleatoria.
        rw = Paseo aleatorio(50_000)
        rw.fill_walk()

        # Traza los puntos en el recorrido.
        plt.style.use('clásico')
        higo, hacha = plt.subplots()
        números_puntos = rango(rw.num_puntos)
        ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
                    colores de borde='ninguno', s=1)

    --recorte--
```

Este ejemplo crea un paseo aleatorio con 50.000 puntos y traza cada uno apunte al tamaño $s=1$. La caminata resultante es tenue y parecida a una nube, como se muestra en la Figura 15-11. ¡Hemos creado una obra de arte a partir de un simple diagrama de dispersión!

Experimente con este código para ver cuánto puede aumentar el número de puntos en una caminata antes de que su sistema comience a ralentizarse significativamente o la trama pierda su atractivo visual.

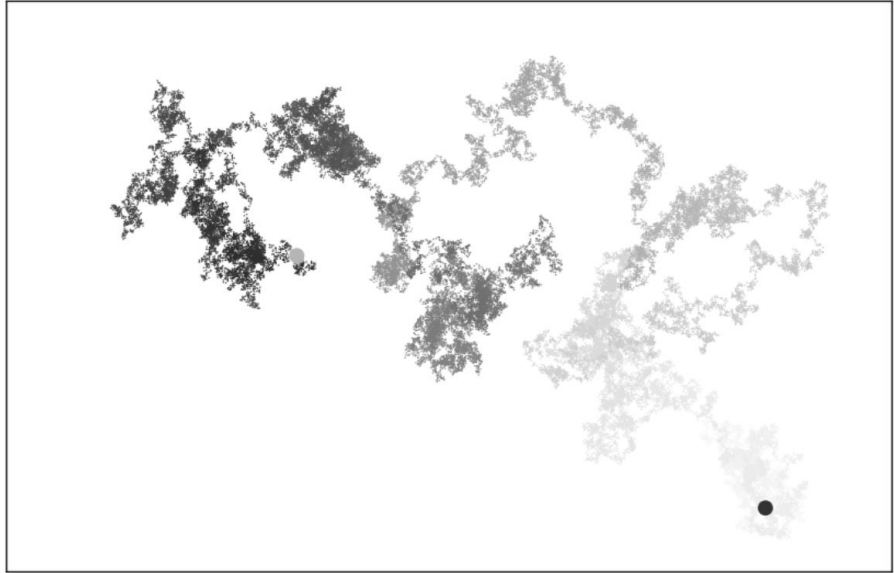


Figura 15-11: Un paseo con 50.000 puntos

Modificar el tamaño para llenar la pantalla

Una visualización es mucho más eficaz a la hora de comunicar patrones en los datos si encaja bien en la pantalla. Para que la ventana de trazado se ajuste mejor a su pantalla, puede ajustar el tamaño de la salida de Matplotlib. Esto se hace en la llamada a `subplots()` :

```
higo, hacha = plt.subplots(tamaño de higo=(15, 9))
```

Al crear una gráfica, puede pasar a `subplots()` un argumento `figsize` , que establece el tamaño de la figura. El parámetro `figsize` toma una tupla que le indica a Matplotlib las dimensiones de la ventana de trazado en pulgadas.

Matplotlib asume que la resolución de su pantalla es de 100 píxeles por pulgada; Si este código no le proporciona un tamaño de trazado preciso, ajuste los números según sea necesario. O, si conoce la resolución de su sistema, puede pasarle a `subplots()` la resolución usando el parámetro `dpi` :

```
higo, hacha = plt.subplots(tamaño de higo=(10, 6), ppp=128)
```

Esto debería ayudar a hacer el uso más eficiente del espacio disponible en tu pantalla.

INTÉNTALO TÚ MISMO

15-3. Movimiento molecular: modifique `rw_visual.py` reemplazando `ax.scatter()` con `ax.plot()`. Para simular la trayectoria de un grano de polen en la superficie de una gota de agua, pase `rw.x_values` y `rw.y_values` e incluya un argumento de ancho de línea. Utilice 5.000 en lugar de 50.000 puntos para evitar que la trama esté demasiado ocupada.

15-4. Paseos aleatorios modificados: en la clase `RandomWalk`, `x_step` e `y_step` se generan a partir del mismo conjunto de condiciones. La dirección se elige aleatoriamente de la lista `[1, -1]` y la distancia de la lista `[0, 1, 2, 3, 4]`. Modifique los valores en estas listas para ver qué sucede con la forma general de sus caminatas. Pruebe con una lista más larga de opciones para la distancia, como del 0 al 8, o elimine el `-1` de la lista de direcciones `x` o `y`.

15-5. Refactorización: el método `fill_walk()` es largo. Cree un nuevo método llamado `get_step()` para determinar la dirección y la distancia de cada paso y luego calcule el paso. Deberías terminar con dos llamadas a `get_step()` en `fill_walk()`:

```
x_paso = self.get_step()
y_paso = self.get_step()
```

Esta refactorización debería reducir el tamaño de `fill_walk()` y hacer que el método sea más fácil de leer y comprender.

Tirar dados con Plotly

En esta sección, usaremos Plotly para producir visualizaciones interactivas. Plotly es particularmente útil cuando creas visualizaciones que se mostrarán en un navegador, porque las visualizaciones se escalarán automáticamente para adaptarse a la pantalla del espectador. Estas visualizaciones también son interactivas; cuando el usuario pasa el cursor sobre ciertos elementos en la pantalla, se resalta la información sobre esos elementos. Construiremos nuestra visualización inicial en solo un par de líneas de código usando Plotly Express, un subconjunto de Plotly que se enfoca en generar gráficos con la menor cantidad de código posible. Una vez que sepamos que nuestro gráfico es correcto, personalizaremos la salida tal como lo hicimos con Matplotlib.

En este proyecto, analizaremos los resultados de tirar dados. Cuando lanzas un dado normal de seis caras, tienes las mismas posibilidades de obtener cualquiera de los números del 1 al 6. Sin embargo, cuando usas dos dados, es más probable que obtengas ciertos números que otros. Intentaremos determinar qué números tienen más probabilidades de ocurrir generando un conjunto de datos que represente el lanzamiento de dados. Luego trazaremos los resultados de un gran número de tiradas para determinar qué resultados son más probables que otros.

Este trabajo ayuda a modelar juegos que involucran dados, pero las ideas centrales también se aplican a juegos que involucran azar de cualquier tipo, como los juegos de cartas. También se relaciona con muchas situaciones del mundo real donde la aleatoriedad juega un factor importante.

Instalación de trama

Instale Plotly usando pip, tal como lo hizo con Matplotlib:

```
$ python -m pip install --usuario trazado
$ python -m instalación de pip --usuario pandas
```

Plotly Express depende de pandas, que es una biblioteca para trabajar de manera eficiente con datos, por lo que también debemos instalarla. Si usó python3 u otra cosa al instalar Matplotlib, asegúrese de usar el mismo comando aquí.

Para ver qué tipo de visualizaciones son posibles con Plotly, visite la galería de tipos de gráficos en <https://plotly.com/python>. Cada ejemplo incluye código fuente, para que puedas ver cómo Plotly genera las visualizaciones.

Creando la clase de troquel

Crearemos la siguiente clase Die para simular la tirada de un dado:

die.py de randint de importación aleatoria

```
class morir:
    """Una clase que representa un solo dado."""

1 def __init__(self, num_sides=6):
    """Supongamos un dado de seis caras."""
    self.num_sides = num_sides

    tirada de definición (yo):
    """Devuelve un valor aleatorio entre 1 y el número de lados."""
2     devolver randint(1, self.num_sides)
```

El método `__init__()` toma un argumento opcional 1. Con el dado clase, cuando se crea una instancia de nuestro dado, el número de lados será seis si no se incluye ningún argumento. Si se incluye un argumento, ese valor establecerá el número de lados del dado. (Los dados reciben el nombre de su número de caras: un dado de seis caras es un D6, un dado de ocho caras es un D8, y así sucesivamente).

El método `roll()` utiliza la función `randint()` para devolver un número aleatorio entre 1 y el número de lados 2. Esta función puede devolver el valor inicial (1), el valor final (`num_sides`) o cualquier número entero entre los dos.

Tirar el dado

Antes de crear una visualización basada en la clase Die, lancemos un D6, imprimamos los resultados y verifiquemos que los resultados parezcan razonables:

die_visual.py de morir importar morir

```
# Crea un D6.
1 dado = morir()
```

```
# Haga algunos rollos y almacene los resultados en una lista.
resultados = []
2 para roll_num en el rango (100):
    resultado = morir.roll()
    resultados.añadir(resultado)
```

```
imprimir (resultados)
```

Creamos una instancia de Die con los seis lados predeterminados 1. Luego tiramos el dado 100 veces 2 y almacenamos el resultado de cada tirada en la lista de resultados.

A continuación se muestra un conjunto de resultados de muestra:

```
[4, 6, 5, 6, 1, 5, 6, 3, 5, 3, 5, 3, 2, 2, 1, 3, 1, 5, 3, 6, 3, 6, 5, 4,
1, 1, 4, 2, 3, 6, 4, 2, 6, 4, 1, 3, 2, 5, 6, 3, 6, 2, 1, 1, 3, 4, 1, 4,
3, 5, 1, 4, 5, 5, 2, 3, 3, 1, 2, 3, 5, 6, 2, 5, 6, 1, 3, 2, 1, 1, 1, 6,
5, 5, 2, 2, 6, 4, 1, 4, 5, 1, 1, 1, 4, 5, 3, 3, 1, 3, 5, 4, 5, 6, 5, 4,
1, 5, 1, 2]
```

Un análisis rápido de estos resultados muestra que la clase Die parece estar funcionando. Vemos los valores 1 y 6, por lo que sabemos que se devuelven los valores más pequeños y más grandes posibles y, como no vemos 0 o 7, sabemos que todos los resultados están en el rango apropiado. También vemos cada número del 1 al 6, lo que indica que todos los resultados posibles están representados. Determinemos exactamente cuántas veces aparece cada número.

Analizando los resultados

Analizaremos los resultados de tirar un D6 contando cuántas veces tiramos cada número:

```
die_visual.py --snip--
# Haga algunos rollos y almacene los resultados en una lista.
resultados = []
1 para roll_num en el rango (1000):
    resultado = morir.roll()
    resultados.añadir(resultado)

# Analizar los resultados.
frecuencias = []
2 poss_results = rango(1, die.num_sides+1)
    por valor en poss_results:
3 frecuencia = resultados.recuento(valor)
4 frecuencias.append(frecuencia)

imprimir (frecuencias)
```

Como ya no imprimimos los resultados, podemos aumentar el número de tiradas simuladas a 1000 1. Para analizar las tiradas, creamos la lista vacía de frecuencias para almacenar el número de veces que se lanza cada valor. Luego generamos todos los resultados posibles que pudimos obtener; en este ejemplo, son todos los números desde 1 hasta el número de lados que tenga el dado 2. Recorremos los valores posibles, contamos cuántas veces aparece cada número en los resultados 3 y

luego agregue este valor a las frecuencias 4. Imprimimos esta lista antes de realizar una visualización:

```
[155, 167, 168, 170, 159, 181]
```

Estos resultados parecen razonables: vemos seis frecuencias, una para cada número posible cuando tiras un D6. También vemos que ninguna frecuencia es significativamente mayor que otra. Ahora visualicemos estos resultados.

Hacer un histograma

Ahora que tenemos los datos que queremos, podemos generar una visualización en solo un par de líneas de código usando Plotly Express:

die_visual.py importar plotly.express como px

```
de troquel importado troquel
--recorte--

por valor en poss_results:
    frecuencia = resultados.recuento(valor)
    frecuencias.append(frecuencia)

# Visualiza los resultados.
fig = px.bar(x=poss_results, y=frecuencias)
figura.mostrar()
```

Primero importamos el módulo plotly.express , usando el alias convencional px. Luego usamos la función px.bar() para crear un gráfico de barras. En el uso más simple de esta función, solo necesitamos pasar un conjunto de valores de x y un conjunto de valores de y. Aquí los valores de x son los posibles resultados de lanzar un solo dado, y los valores de y son las frecuencias de cada resultado posible.

La última línea llama a fig.show(), lo que le indica a Plotly que represente el gráfico resultante como un archivo HTML y abra ese archivo en una nueva pestaña del navegador. El resultado se muestra en la Figura 15-12.

Este es un gráfico realmente simple y ciertamente no está completo. Pero así es exactamente como debe usarse Plotly Express; escribes un par de líneas de código, miras el gráfico y te aseguras de que represente los datos de la manera que deseas. Si le gusta lo que ve, puede pasar a personalizar elementos del gráfico, como etiquetas y estilos. Pero si desea explorar otros tipos de gráficos posibles, puede hacerlo ahora, sin tener que dedicar más tiempo al trabajo de personalización. Siéntete libre de probar esto ahora cambiando px.bar() a algo como px.scatter() o px.line(). Puede encontrar una lista completa de los tipos de gráficos disponibles en <https://plotly.com/python/plotly-express>.

Este gráfico es dinámico e interactivo. Si cambia el tamaño de la ventana de su navegador, el gráfico cambiará de tamaño para coincidir con el espacio disponible. Si pasa el cursor sobre cualquiera de las barras, verá una ventana emergente que resalta los datos específicos relacionados con esa barra.

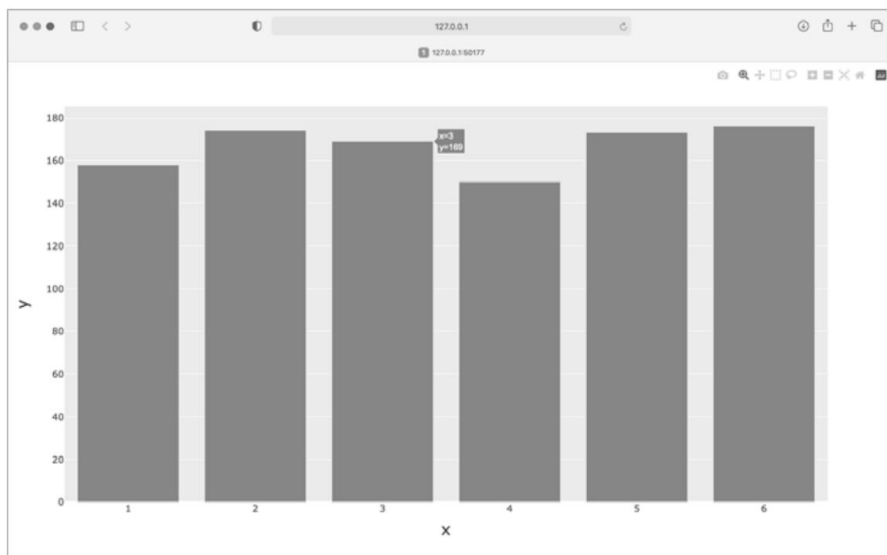


Figura 15-12: El gráfico inicial producido por Plotly Express

Personalizando la trama

Ahora que sabemos que tenemos el tipo correcto de gráfico y que nuestros datos se representan con precisión, podemos concentrarnos en agregar las etiquetas y estilos apropiados para el gráfico.

La primera forma de personalizar un gráfico con Plotly es utilizar algunos parámetros opcionales en la llamada inicial que genera el gráfico, en este caso, `px.bar()`. A continuación se explica cómo agregar un título general y una etiqueta para cada eje:

```
die_visual.py --snip--
# Visualiza los resultados.
1 título = "Resultados de lanzar un D6 1000 veces"
2 etiquetas = {'x': 'Resultado', 'y': 'Frecuencia del resultado'}
fig = px.bar(x=pos_results, y=frecuencias, título=título, etiquetas=etiquetas)
figura.mostrar()
```

Primero definimos el título que queremos, aquí asignado al título 1. Para definir las etiquetas de los ejes, escribimos un diccionario 2. Las claves en el diccionario se refieren a las etiquetas que queremos personalizar y los valores son las etiquetas personalizadas que queremos usar. Aquí le damos al eje x la etiqueta Resultado y al eje y la etiqueta Frecuencia del resultado. La llamada a `px.bar()` ahora incluye el título y las etiquetas de los argumentos opcionales.

Ahora, cuando se genera el gráfico, incluye un título apropiado y una etiqueta para cada eje, como se muestra en la Figura 15-13.

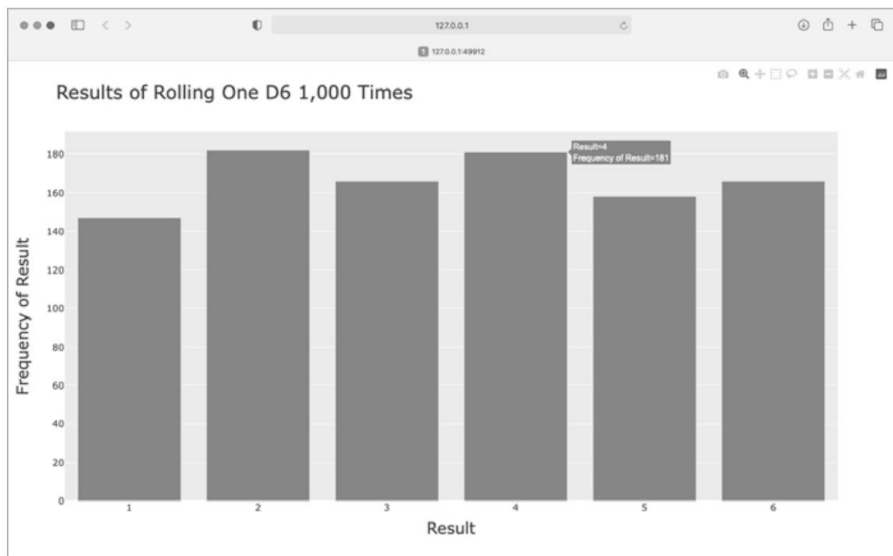


Figura 15-13: Un gráfico de barras simple creado con Plotly

Tirar dos dados Tirar

dos dados da como resultado números más grandes y una distribución diferente de los resultados. Modifiquemos nuestro código para crear dos dados D6 para simular la forma en que tiramos un par de dados. Cada vez que saquemos el par, sumaremos los dos números (uno de cada dado) y almacenaremos la suma en los resultados. Guarde una copia de `die_visual.py` como `dice_visual.py` y realice los siguientes cambios:

`dice_visual.py` importar `plotly.express` como `px`

```

de troquel importado troquel

# Crea dos dados D6. morir_1
= morir() morir_2
= morir()

# Haga algunos rollos y almacene los resultados en una lista.
resultados = []
para roll_num en el rango (1000): 1
    resultado = die_1.roll() + die_2.roll() resultados.append(resultado)

# Analizar los resultados.
frecuencias = []
2 max_result = die_1.num_sides + die_2.num_sides 3 poss_results =
rango(2, max_result+1) para el valor en poss_results:
    frecuencia = resultados.count(valor)
    frecuencias.append(frecuencia)

```

```
# Visualiza los resultados.
```

```
title = "Resultados de lanzar dos dados D6 1000 veces"
```

```
etiquetas = {'x': 'Resultado', 'y': 'Frecuencia del resultado'}
```

```
fig = px.bar(x=poss_results, y=frecuencias, título=título, etiquetas=etiquetas)
```

```
figura.mostrar()
```

Después de crear dos instancias de Die, tiramos los dados y calculamos la suma de los dos dados para cada tirada 1. El resultado más pequeño posible (2) es la suma del número más pequeño en cada dado. El mayor resultado posible (12) es la suma del número más grande en cada dado, que asignamos a max_result 2. La variable max_result hace que el código para generar poss_results sea mucho más fácil de leer 3. Podríamos haber escrito range(2, 13), pero esto sólo funcionaría con dos dados D6. Al modelar situaciones del mundo real, es mejor escribir código que pueda modelar fácilmente una variedad de situaciones. Este código nos permite simular el lanzamiento de un par de dados con cualquier número de caras.

Después de ejecutar este código, debería ver un gráfico similar al de la Figura 15-14.

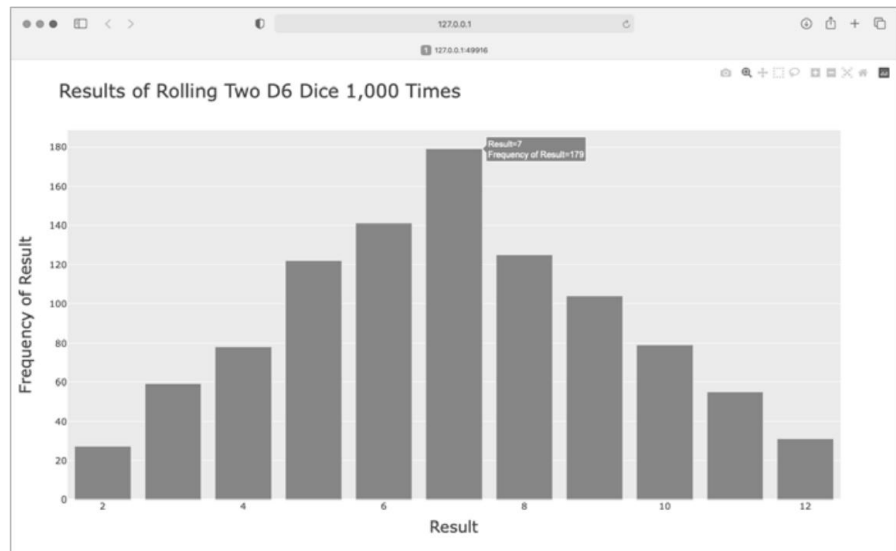


Figura 15-14: Resultados simulados de lanzar dos dados de seis caras 1000 veces

Este gráfico muestra la distribución aproximada de los resultados que probablemente obtendrás cuando lanzas un par de dados D6. Como puede ver, es menos probable que obtenga un 2 o un 12 y lo más probable es que obtenga un 7. Esto sucede porque hay seis formas de obtener un 7: 1 y 6, 2 y 5, 3 y 4, 4 y 3, 5 y 2, y 6 y 1.

Más personalizaciones

Hay un problema que deberíamos abordar con la trama que acabamos de generar.

Ahora que hay 11 barras, la configuración de diseño predeterminada para el eje x deja algunas de las barras sin etiquetar. Si bien la configuración predeterminada funciona bien para la mayoría de las visualizaciones, este gráfico se vería mejor con todas las barras etiquetadas.

Plotly tiene un método `update_layout()` que se puede utilizar para realizar una amplia variedad de actualizaciones a una figura después de haberla creado. Aquí se explica cómo decirle a Plotly que le dé a cada barra su propia etiqueta:

```
dice_visual.py --snip--
fig = px.bar(x=poss_results, y=frecuencias, título=título, etiquetas=etiquetas)

# Personalizar aún más el gráfico.
figura.update_layout(xaxis_dtick=1)

figura.mostrar()
```

El método `update_layout()` actúa sobre el objeto `fig`, que representa el gráfico general. Aquí usamos el argumento `xaxis_dtick`, que especifica la distancia entre las marcas en el eje x. Establecemos ese espacio en 1, para que cada barra esté etiquetada. Cuando vuelvas a ejecutar `dice_visual.py`, deberías ver una etiqueta en cada barra.

Dados rodantes de diferentes tamaños

Creemos un dado de seis caras y un dado de diez caras, y veamos qué sucede cuando los lanzamos 50.000 veces:

```
dados_visual
_d6d10.py

importar plotly.express como px

de troquel importado troquel

# Crea un D6 y un D10.
morir_1 = morir()
1 dado_2 = Dado(10)

# Haga algunos rollos y almacene los resultados en una lista.
resultados = []
para roll_num en el rango (50_000):
    resultado = dado_1.roll() + dado_2.roll()
    resultados.añadir(resultado)

# Analizar los resultados.
--recorte--

# Visualiza los resultados.
2 title = "Resultados de tirar un D6 y un D10 50.000 veces"
etiquetas = {'x': 'Resultado', 'y': 'Frecuencia del resultado'}
--recorte--
```

Para hacer un D10, pasamos el argumento 10 al crear el segundo dado. instancia 1 y cambie el primer bucle para simular 50.000 rollos en lugar de 1.000. Cambiamos el título del gráfico también 2.

La Figura 15-15 muestra el gráfico resultante. En lugar de un resultado más probable, hay cinco resultados de este tipo. Esto sucede porque todavía hay una sola forma de sacar el valor más pequeño (1 y 1) y el valor más grande (6 y 10), pero el dado más pequeño limita la cantidad de formas en que puedes generar los números del medio. Hay seis formas de sacar un 7, 8, 9, 10 u 11; estos son los resultados más comunes y es igualmente probable que saque cualquiera de ellos.

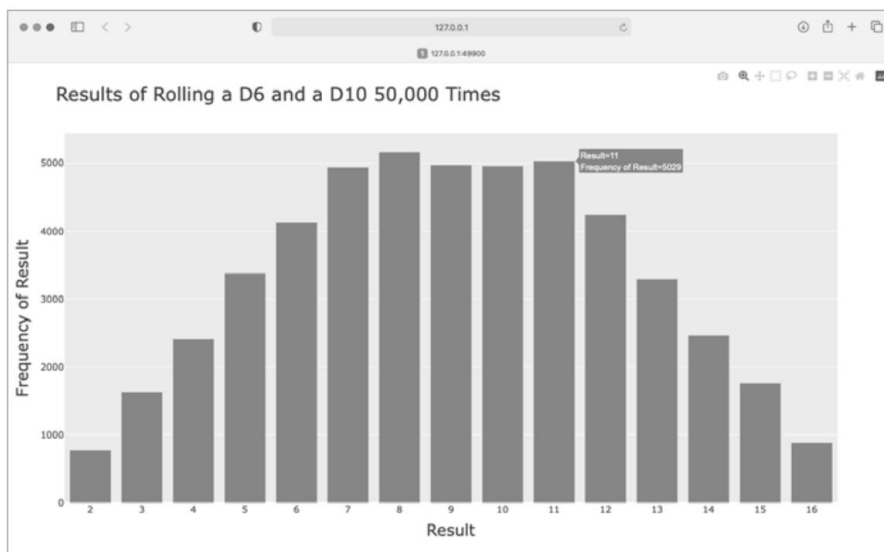


Figura 15-15: Los resultados de lanzar un dado de seis caras y uno de diez caras 50.000 veces

Nuestra capacidad de usar Plotly para modelar el lanzamiento de dados nos brinda una considerable libertad para explorar este fenómeno. En sólo unos minutos, podrás simular una enorme cantidad de tiradas utilizando una gran variedad de dados.

Ahorro de cifras

Cuando tengas una figura que te guste, siempre puedes guardar el gráfico como un archivo HTML a través de tu navegador. Pero también puedes hacerlo mediante programación. Para guardar su gráfico como un archivo HTML, reemplaza la llamada a `fig.show()` con una llamada a `fig.write_html()`:

```
.write_html():
```

```
fig.write_html('dice_visual_d6d10.html')
```

El método `write_html()` requiere un argumento: el nombre del archivo en el que escribir. Si solo proporciona un nombre de archivo, el archivo se guardará en el mismo directorio que el archivo `.py`. También puede llamar a `write_html()` con un objeto `Path` y escribir el archivo de salida en cualquier lugar que desee de su sistema.

INTÉNTALO TÚ MISMO

15-6. Dos D8: crea una simulación que muestre lo que sucede cuando lanzas dos dados de ocho caras 1000 veces. Intente imaginarse cómo cree que se verá la visualización antes de ejecutar la simulación, luego vea si su intuición fue correcta.

Aumente gradualmente la cantidad de tiradas hasta que comience a ver los límites de las capacidades de su sistema.

15-7. Tres dados: cuando tiras tres dados D6, el número más pequeño que puedes tirar es 3 y el número más grande es 18. Crea una visualización que muestre lo que sucede cuando tiras tres dados D6.

15-8. Multiplicación: cuando tiras dos dados, normalmente sumas los dos números para obtener el resultado. Crea una visualización que muestre lo que sucede si multiplicas estos números entre sí.

15-9. Comprensiones: Para mayor claridad, los listados de esta sección utilizan la forma larga de bucles `for`. Si se siente cómodo usando listas por comprensión, intente escribir una comprensión para uno o ambos bucles en cada uno de estos programas.

15-10. Practicando con ambas bibliotecas: intente usar Matplotlib para hacer una visualización de lanzamiento de dados y use Plotly para hacer la visualización de un recorrido aleatorio. (Deberá consultar la documentación de cada biblioteca para completar este ejercicio).

Resumen

En este capítulo, aprendió a generar conjuntos de datos y crear visualizaciones de esos datos. Creaste gráficos simples con Matplotlib y usaste un diagrama de dispersión para explorar paseos aleatorios. También creaste un histograma con Plotly y lo usaste para explorar los resultados de lanzar dados de diferentes tamaños.

Generar sus propios conjuntos de datos con código es una forma interesante y poderosa de modelar y explorar una amplia variedad de situaciones del mundo real. A medida que continúe trabajando en los proyectos de visualización de datos que siguen, esté atento a las situaciones que podría modelar con código. Mire las visualizaciones que ve en los medios de comunicación y vea si puede identificar aquellas que se generaron utilizando métodos similares a los que está aprendiendo en estos proyectos.

En el Capítulo 16, descargará datos de fuentes en línea y continuará utilizar Matplotlib y Plotly para explorar esos datos.

dieciséis

DESCARGAR DATOS



En este capítulo, descargará conjuntos de datos de fuentes en línea y creará visualizaciones funcionales de esos datos. Puedes encontrar

una increíble variedad de datos en línea, muchos de los cuales no han sido examinados en profundidad. La capacidad de analizar estos datos le permite descubrir patrones y conexiones que nadie más ha encontrado.

Accederemos y visualizaremos datos almacenados en dos formatos de datos comunes: CSV y JSON. Usaremos el módulo csv de Python para procesar datos meteorológicos almacenados en formato CSV y analizar temperaturas altas y bajas a lo largo del tiempo en dos ubicaciones diferentes. Luego usaremos Matplotlib para generar un gráfico basado en nuestros datos descargados para mostrar variaciones de temperatura en dos entornos diferentes: Sitka, Alaska y Death Valley, California. Más adelante en el capítulo, usaremos el módulo json para acceder a los datos de terremotos almacenados en el formato GeoJSON y usaremos Plotly para dibujar un mapa mundial que muestre las ubicaciones y magnitudes de los terremotos recientes.

Al final de este capítulo, estará preparado para trabajar con varios tipos de conjuntos de datos en diferentes formatos y tendrá una comprensión más profunda de cómo crear visualizaciones complejas. Poder acceder y visualizar datos en línea es esencial para trabajar con una amplia variedad de conjuntos de datos del mundo real.

El formato de archivo CSV

Una forma sencilla de almacenar datos en un archivo de texto es escribir los datos como una serie de valores separados por comas, llamados valores separados por comas. Los archivos resultantes son archivos CSV. Por ejemplo, aquí hay una gran cantidad de datos meteorológicos en formato CSV:

```
"USW00025333","AEROPUERTO DE SITKA, AK EE.UU.,"2021-01-01","44","40"
```

Este es un extracto de datos meteorológicos del 1 de enero de 2021 en Sitka, Alaska. Incluye las temperaturas máximas y mínimas del día, así como otras mediciones de ese día. Los archivos CSV pueden resultar tediosos de leer para los humanos, pero los programas pueden procesar y extraer información de ellos de forma rápida y precisa.

Comenzaremos con un pequeño conjunto de datos meteorológicos en formato CSV registrados en Sitka; está disponible en los recursos de este libro en https://ehmatthes.github.io/pcc_3e. Cree una carpeta llamada `Weather_data` dentro de la carpeta donde está guardando los programas de este capítulo. Copie el archivo `sitka_weather_07-2021_simple.csv` en esta nueva carpeta. (Después de descargar los recursos de este libro, tendrá todos los archivos que necesita para este proyecto).

NOTA Los datos meteorológicos de este proyecto se descargaron originalmente de <https://ncdc.noaa.gov/cdo-web>.

Analizando los encabezados del archivo CSV

El módulo `csv` de Python en la biblioteca estándar analiza las líneas de un archivo CSV y nos permite extraer rápidamente los valores que nos interesan. Comencemos examinando la primera línea del archivo, que contiene una serie de encabezados para los datos. Estos encabezados nos dicen qué tipo de información contienen los datos:

`sitka_highs.py` de ruta de importación de `pathlib`

```
importar archivos csv
```

```
1 ruta = Ruta('weather_data/sitka_weather_07-2021_simple.csv')
   líneas = ruta.read_text().splitlines()
```

```
2 lector = csv.reader(líneas)
```

```
3 header_row = siguiente (lector)
   imprimir (encabezado_fila)
```

Primero importamos `Path` y el módulo `csv`. Luego construimos un objeto `Path` que busca en la carpeta `Weather_data` y apunta al archivo de datos meteorológicos específico con el que queremos trabajar. Leemos el archivo y encadenamos las líneas divididas(). método para obtener una lista de todas las líneas del archivo, que asignamos a las `líneas`.

A continuación, creamos un objeto lector 2. Este es un objeto que se puede usar para analizar cada línea del archivo. Para crear un objeto lector, llame a la función `csv.reader()` y pásale la lista de líneas del archivo CSV.

Cuando se le proporciona un objeto lector, la función `next()` devuelve la siguiente línea del archivo, comenzando desde el principio del archivo. Aquí llamamos a `next()` solo una vez, por lo que obtenemos la primera línea del archivo, que contiene los encabezados del archivo 3. Asignamos los datos que se devuelven a `header_row`. Como puedes ver, `header_row` contiene encabezados significativos relacionados con el clima que nos dicen qué información contiene cada línea de datos:

```
['ESTACIÓN', 'NOMBRE', 'FECHA', 'TAVG', 'TMAX', 'TMIN']
```

El objeto lector procesa la primera línea de valores separados por comas en el archivo y almacena cada valor como un elemento en una lista. El encabezado ESTACIÓN representa el código de la estación meteorológica que registró estos datos. La posición de este encabezado nos dice que el primer valor de cada línea será el código de la estación meteorológica. El encabezado NOMBRE indica que el segundo valor en cada línea es el nombre de la estación meteorológica que realizó la grabación. El resto de encabezados especifican qué tipo de información se registró en cada lectura. Los datos que más nos interesan por ahora son la fecha (DATE), la temperatura máxima (TMAX) y la temperatura mínima (TMIN). Este es un conjunto de datos simple que contiene solo datos relacionados con la temperatura. Cuando descarga sus propios datos meteorológicos, puede optar por incluir otras mediciones relacionadas con la velocidad y la dirección del viento y los datos de precipitación.

Imprimir los encabezados y sus posiciones

Para que sea más fácil comprender los datos del encabezado del archivo, imprimamos cada encabezado y su posición en la lista:

```
sitka_highs.py --snip--
lector = csv.reader(líneas)
header_row = siguiente (lector)

para índice, encabezado de columna en enumerar (encabezado_fila):
    imprimir (índice, encabezado_columna)
```

La función `enumerate()` devuelve tanto el índice de cada elemento como el valor de cada elemento a medida que recorre una lista. (Tenga en cuenta que hemos eliminado la línea `print(header_row)` en favor de esta versión más detallada).

Aquí está el resultado que muestra el índice de cada encabezado:

```
0 ESTACIÓN
1 NOMBRE
2 FECHA
3 TAVG
4 TMAX
5 TMIN
```

Podemos ver que las fechas y sus temperaturas máximas se almacenan en las columnas 2 y 4. Para explorar estos datos, procesaremos cada fila de datos en sitka. `_weather_07-2021_simple.csv` y extrae los valores con los índices 2 y 4.

Extracción y lectura de datos

Ahora que sabemos qué columnas de datos necesitamos, leamos algunos de esos datos.

Primero, leeremos la temperatura alta de cada día:

```
sitka_highs.py --snip--
lector = csv.reader(líneas)
header_row = siguiente(lector)

# Extraer a altas temperaturas.
1 máximos = []
2 para la fila en el lector:
3 alto = int(fila[4])
    altos.append(alto)

imprimir (altos)
```

Creemos una lista vacía llamada `máximos` 1 y luego recorremos las filas restantes en el archivo 2. El objeto lector continúa desde donde lo dejó en el archivo CSV y devuelve automáticamente cada línea siguiendo su posición actual. Como ya leímos la fila del encabezado, el bucle comenzará en la segunda línea donde comienzan los datos reales. En cada paso por el bucle extraemos los datos del índice 4, correspondiente al encabezado TMAX, y los asignamos a la variable `high` 3. Usamos la función `int()` para convertir los datos, que se almacenan como una cadena, en un formato numérico para que podamos usarlo. Luego agregamos este valor a los `máximos`.

El siguiente listado muestra los datos ahora almacenados en `máximos`:

```
[61, 60, 66, 60, 65, 59, 58, 58, 57, 60, 60, 60, 57, 58, 60, 61, 63, 63, 70,
64, 59, 63, 61, 58, 59, 64, 62, 70, 70, 73, 66]
```

Hemos extraído la temperatura alta para cada fecha y almacenado cada valor en una lista. Ahora creemos una visualización de estos datos.

Trazar datos en un gráfico de temperatura

Para visualizar los datos de temperatura que tenemos, primero crearemos un gráfico simple de los `máximos` diarios usando Matplotlib, como se muestra aquí:

```
sitka_highs.py de ruta de importación de pathlib
importar archivos csv

importar matplotlib.pyplot como plt

ruta = Ruta('weather_data/sitka_weather_07-2021_simple.csv')
líneas = ruta.read_text().splitlines()
--recorte--
```

```

# Trazar las altas temperaturas.
plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
1 ax.plot(alto, color='rojo')

# Dar formato al gráfico.
2 ax.set_title("Temperaturas máximas diarias, julio de 2021", tamaño de fuente=24)
3 ax.set_xlabel("", tamaño de fuente=16)
  ax.set_ylabel("Temperatura (F)", tamaño de fuente=16)
  ax.tick_params(tamaño de etiqueta=16)

plt.mostrar()

```

Pasamos la lista de máximos a `plot()` y pasamos `color='red'` para trazar los puntos en rojo 1. (Trazaremos los máximos en rojo y los mínimos en azul). Luego especificamos algunos otros detalles de formato, como el título, el tamaño de fuente y las etiquetas 2, tal como lo hicimos en el Capítulo 15. Debido a que todavía tenemos que agregar las fechas, no etiquetaremos el eje x, pero `ax.set_xlabel()` modifica el tamaño de fuente. para que las etiquetas predeterminadas sean más legibles 3. La Figura 16-1 muestra el gráfico resultante: un gráfico lineal simple de las altas temperaturas de julio de 2021 en Sitka, Alaska.

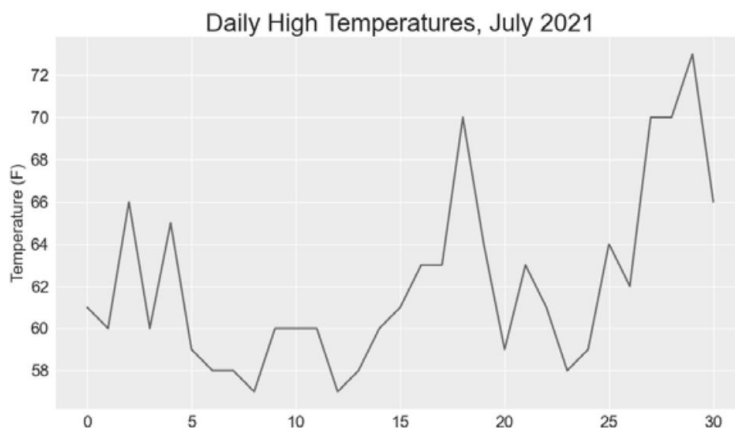


Figura 16-1: Gráfico lineal que muestra las temperaturas máximas diarias para julio de 2021 en Sitka, Alaska

El módulo de fecha y hora

Agreguemos fechas a nuestro gráfico para hacerlo más útil. La primera fecha del archivo de datos meteorológicos se encuentra en la segunda fila del archivo:

```
"USW00025333","AEROPUERTO DE SITKA, AK EE.UU.,"2021-07-01","61","53"
```

Los datos se leerán como una cadena, por lo que necesitamos una forma de convertir los cadena "2021-07-01" a un objeto que representa esta fecha. podemos construir

un objeto que representa el 1 de julio de 2021, utilizando el método `strptime()` del módulo `datetime` . Veamos cómo funciona `strptime()` en una sesión de terminal:

```
>>> desde fecha y hora importar fecha y hora
>>> primera_fecha = datetime.strptime('2021-07-01', '%Y-%m-%d')
>>> imprimir(primera_fecha)
2021-07-01 00:00:00
```

Primero importamos la clase `datetime` del módulo `datetime` . Luego llamamos al método `strptime()` con la cadena que contiene la fecha que queremos procesar como primer argumento. El segundo argumento le dice a Python cómo se formatea la fecha. En este ejemplo, `'%Y-'` le dice a Python que busque un año de cuatro dígitos antes del primer guión; `'%m-'` indica un mes de dos dígitos antes del segundo guión; y `'%d'` significa que la última parte de la cadena es el día del mes, del 1 al 31.

El método `strptime()` puede tomar una variedad de argumentos para determinar cómo interpretar la fecha. La tabla 16-1 muestra algunos de estos argumentos.

Tabla 16-1: Argumentos de formato de fecha y hora del módulo `datetime`

| Argumento Significado | |
|-----------------------|---|
| <code>%A</code> | Nombre del día de la semana, como lunes |
| <code>%B</code> | Nombre del mes, como enero |
| <code>%m</code> | Mes, como número (01 a 12) |
| <code>%d</code> | Día del mes, como número (01 a 31) |
| <code>%Y</code> | Año de cuatro dígitos, como 2019 |
| <code>%a</code> | Año de dos dígitos, como 19 |
| <code>%H</code> | Hora, en formato de 24 horas (00 a 23) |
| <code>%I</code> | Hora, en formato de 12 horas (01 a 12) |
| <code>%p</code> | Am o PM |
| <code>%M</code> | Minutos (00 a 59) |
| <code>%S</code> | Segundos (00 a 61) |

Trazar fechas

Podemos mejorar nuestro gráfico extrayendo fechas para las lecturas diarias de temperatura alta y usando estas fechas en el eje x:

```
sitka_highs.py de ruta de importación de pathlib
importar archivos csv
desde fecha y hora importar fecha y hora

importar matplotlib.pyplot como plt

ruta = Ruta('weather_data/sitka_weather_07-2021_simple.csv')
líneas = ruta.read_text().splitlines()
```



```

lector = csv.reader(líneas)
header_row = siguiente(lector)

# Extraer dátiles y altas temperaturas. 1 fechas,
máximos = [], [] para la fila en
el lector:
2 fecha_actual = datetime.strptime(fila[2], '%Y-%m-%d') alto = int(fila[4])

fechas.append(fecha_actual)
altos.append(alto)

# Trazar las altas temperaturas.
plt.style.use('seaborn') fig, ax
= plt.subplots() 3 ax.plot(fechas,
máximos, color='rojo')

# Dar formato al
gráfico. ax.set_title("Temperaturas máximas diarias, julio de 2021", fontsize=24)
ax.set_xlabel("", fontsize=16) 4
fig.autofmt_xdate()
ax.set_ylabel("Temperatura (F)", tamaño de fuente=16)
ax.tick_params(labelsize=16)

plt.mostrar()

```

Creamos dos listas vacías para almacenar las fechas y las altas temperaturas del archivo 1. Luego convertimos los datos que contienen la información de la fecha (fila [2]) a un objeto de fecha y hora 2 y los agregamos a las fechas. Pasamos las fechas y los valores de temperatura alta a plot() 3. La llamada a fig.autofmt_xdate() 4 dibuja las etiquetas de fecha en diagonal para evitar que se superpongan. La Figura 16-2 muestra el gráfico mejorado.

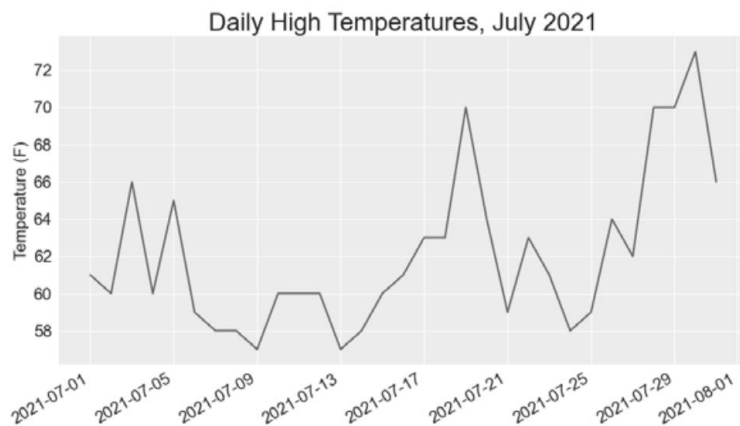


Figura 16-2: El gráfico es más significativo ahora que tiene fechas en el eje x.

Trazar un período de tiempo más

largo Con nuestro gráfico configurado, incluyamos datos adicionales para obtener una imagen más completa del clima en Sitka. Copie el archivo `sitka_weather_2021_simple.csv`, que contiene datos meteorológicos de un año completo para Sitka, en la carpeta donde almacena los datos para los programas de este capítulo.

Ahora podemos generar un gráfico para el clima de todo el año:

```
sitka_highs.py --snip--
ruta = Ruta('weather_data/sitka_weather_2021_simple.csv') líneas =
ruta.read_text().splitlines() --snip-- #
Formatear
gráfico.
ax.set_title("Temperaturas máximas diarias, 2021", fontsize=24)
ax.set_xlabel("", fontsize=16) --snip--
```

Modificamos el nombre del archivo para usar el nuevo archivo de datos `sitka_weather_2021_simple.csv` y actualizamos el título de nuestro gráfico para reflejar el cambio en su contenido. La Figura 16-3 muestra el gráfico resultante.

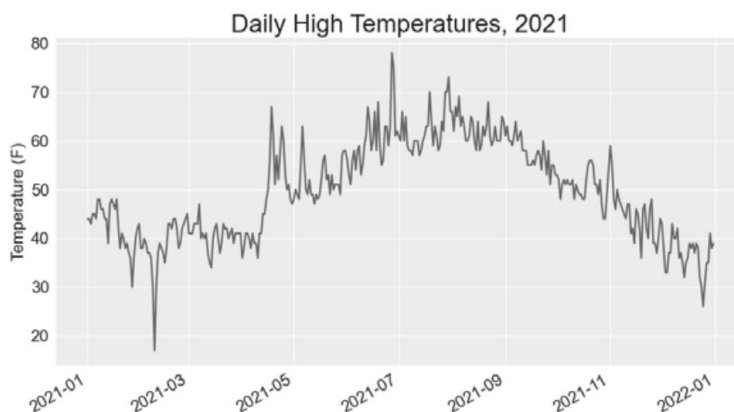


Figura 16-3: Datos de un año

Trazar una segunda serie de datos

Podemos hacer que nuestro gráfico sea aún más útil si incluimos las bajas temperaturas. Necesitamos extraer las bajas temperaturas del archivo de datos y luego agregarlas a nuestro gráfico, como se muestra aquí:

```
sitka
_highs_lows.py --snip--
lector = csv.reader(líneas)
header_row = siguiente(lector)

# Extraer dátiles y temperaturas altas y bajas. 1 fechas,
máximos, mínimos = [], [], [] para la fila
en el lector:
```

```

        fecha_actual = fechahora.strptime(fila[2], '%Y-%m-%d')
        alto = int(fila[4])
2 bajo = int(fila[5])
        fechas.append(fecha_actual)
        altos.append(alto)
        bajos.append(bajo)

# Trazar las temperaturas máximas y mínimas.
plt.style.use('nacadado en el mar')
higo, hacha = plt.subplots()
ax.plot(fechas, máximos, color='rojo')
3 ax.plot(fechas, mínimos, color='azul')

# Dar formato al gráfico.
4 ax.set_title("Temperaturas máximas y mínimas diarias, 2021", tamaño de fuente=24)
--recorte--

```

Agregamos la lista vacía de temperaturas bajas para mantener las temperaturas bajas 1, y luego extraemos y almacenamos la temperatura baja para cada fecha desde la sexta posición en cada fila (fila [5]) 2. Agregamos una llamada a plot() para las bajas temperaturas y coloreamos estos valores de azul 3. Finalmente, actualizamos el título 4. La Figura 16-4 muestra el gráfico resultante.

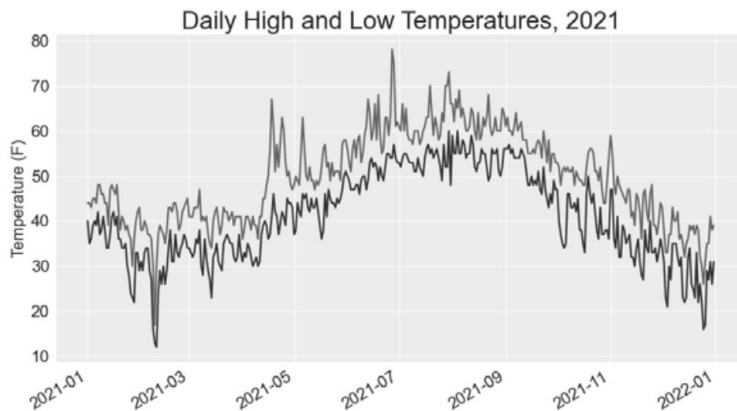


Figura 16-4: Dos series de datos en el mismo gráfico

Sombrear un área en el gráfico

Habiendo agregado dos series de datos, ahora podemos examinar el rango de temperaturas para cada día. Agreguemos un toque final al gráfico usando sombreado para mostrar el rango entre las temperaturas máximas y mínimas de cada día. Para hacerlo, usaremos el método fill_between(), que toma una serie de valores de x y dos series de valores de y y llena el espacio entre las dos series de valores de y:

```

sitka
_alto_bajos.py
--recorte--
# Trazar las temperaturas máximas y mínimas.
plt.style.use('nacadado en el mar')

```

```

higo, hacha = plt.subplots()
1 ax.plot(fechas, máximos, color='rojo', alfa=0.5)
  ax.plot(fechas, mínimos, color='azul', alfa=0.5)
2 ax.fill_between(fechas, máximos, mínimos, color de cara='azul', alfa=0.1)
--recorte--

```

El argumento `alfa` controla la transparencia 1 de un color. Un valor `alfa` de 0 es completamente transparente y un valor de 1 (el valor predeterminado) es completamente opaco. Al establecer `alfa` en 0,5, hacemos que las líneas gráficas rojas y azules parezcan más claras.

Pasamos `fill_between()` la lista de fechas para los valores de `x` y luego las dos series de valores de `y` máximos y mínimos 2. El argumento `facecolor` determina el color de la región sombreada; Le damos un valor `alfa` bajo de 0,1 para que la región rellena conecte las dos series de datos sin distraer la atención de la información que representan. La Figura 16-5 muestra el gráfico con la región sombreada entre los máximos y mínimos.

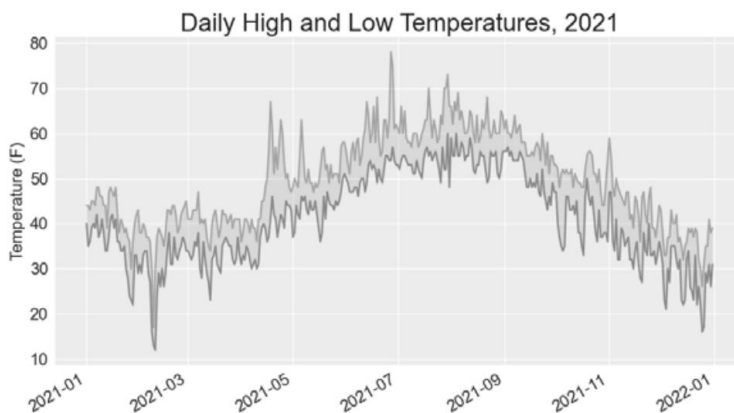


Figura 16-5: La región entre los dos conjuntos de datos está sombreada.

El sombreado ayuda a que el rango entre los dos conjuntos de datos sea inmediatamente evidente.

Comprobación de errores

Deberíamos poder ejecutar el código `sitka_highs_lows.py` utilizando datos de cualquier ubicación. Pero algunas estaciones meteorológicas recopilan datos diferentes que otras, y algunas ocasionalmente funcionan mal y no recopilan algunos de los datos que se supone que deben recopilar. Los datos faltantes pueden provocar excepciones que bloqueen nuestros programas, a menos que los manejemos adecuadamente.

Por ejemplo, veamos qué sucede cuando intentamos generar una plantilla.

Parcela de temperatura para Death Valley, California. Copia el archivo `death_valley_2021_simple.csv` a la carpeta donde almacena los datos de los programas de este capítulo.

Primero, ejecutemos el código para ver los encabezados incluidos en este archivo de datos:

```

valle_muerte
_altos_bajos.py

desde pathlib importar ruta
importar csv

ruta = Ruta('weather_data/death_valley_2021_simple.csv') líneas =
ruta.read_text().splitlines()

lector = csv.reader(líneas)
header_row = siguiente(lector)

para índice, encabezado de columna en enumerar (fila de encabezado):
    imprimir (índice, encabezado de columna)

```

Aquí está el resultado:

```

0 ESTACIÓN
1 NOMBRE
2 FECHA
3 TMAX
4 TMIN
5 TOBAS

```

La fecha está en la misma posición, en el índice 2. Pero las temperaturas alta y baja están en los índices 3 y 4, por lo que necesitaremos cambiar los índices en nuestro código para reflejar estas nuevas posiciones. En lugar de incluir una lectura de temperatura promedio para el día, esta estación incluye TOBS, una lectura para un tiempo de observación específico.

Cambie sitka_highs_lows.py para generar un gráfico para el Valle de la Muerte usando los índices que acabamos de anotar y veamos qué sucede:

```

valle_muerte
_altos_bajos.py

--snip--
path = Path('weather_data/death_valley_2021_simple.csv') líneas =
path.read_text().splitlines() --snip-- # Extrae fechas
y
temperaturas altas y bajas. fechas, máximos, mínimos = [], [], []
para la fila en el lector: fecha_actual =
datetime.strptime(fila[2],
    '%Y-%m-%d') alto = int(fila[3]) bajo = int(fila[4]) fechas.append(fecha_actual)
--snip--

```

Actualizamos el programa para leer el archivo de datos del Valle de la Muerte y cambie los índices para que correspondan a las posiciones TMAX y TMIN de este archivo .

Cuando ejecutamos el programa, obtenemos un error:

```

Rastreo (última llamada más reciente): Archivo
"death_valley_highs_lows.py", línea 17, en <módulo> alto = int(fila[3])

```

1 ValueError: literal no válido para int() con base 10:

El rastreo nos dice que Python no puede procesar la temperatura alta para una de las fechas porque no puede convertir una cadena vacía ("") en un número entero 1.

En lugar de revisar los datos para descubrir qué lectura falta, simplemente manejaremos los casos de datos faltantes directamente.

Ejecutaremos un código de verificación de errores cuando los valores se lean desde el Archivo CSV para manejar excepciones que puedan surgir. Aquí se explica cómo hacer esto:

```

Valle de la Muerte
_altos_bajos.py

--recorte--
para fila en lector:
    fecha_actual = fechahora.strptime(fila[2], '%Y-%m-%d')
1 intento:
    alto = int(fila[3])
    bajo = int(fila[4])
    excepto ValueError:
2         print(f"Faltan datos para {fecha_actual}")
3 más:
        fechas.append(fecha_actual)
        altos.append(alto)
        bajos.append(bajo)

# Trazar las temperaturas máximas y mínimas.
--recorte--

# Dar formato al gráfico.
4 título = "Temperaturas máximas y mínimas diarias, 2021\nDeath Valley, CA"
ax.set_title(título, tamaño de fuente=20)
ax.set_xlabel("", tamaño de fuente=16)
--recorte--

```

Cada vez que examinamos una fila, intentamos extraer la fecha y la temperatura máxima y mínima 1. Si falta algún dato, Python generará un ValueError y lo manejamos imprimiendo un mensaje de error que incluye la fecha de los datos faltantes 2. Después de imprimir el error, el bucle continuará procesando la siguiente fila. Si todos los datos de una fecha se recuperan sin errores, se ejecutará el bloque else y los datos se agregarán a las listas apropiadas. 3. Debido a que estamos trazando información para una nueva ubicación, actualizamos el título para incluir la ubicación en el gráfico. , y utilizamos un tamaño de fuente más pequeño para dar cabida al título más largo 4.

Cuando ejecutes death_valley_highs_lows.py ahora, verás que solo faltaban datos en una fecha:

Faltan datos para 2021-05-04 00:00:00

Debido a que el error se maneja adecuadamente, nuestro código puede generar un gráfico que omite los datos faltantes. La Figura 16-6 muestra el gráfico resultante.

Comparando este gráfico con el gráfico de Sitka, podemos ver que el Valle de la Muerte Es más cálido en general que el sureste de Alaska, como esperamos. Además, la amplitud de temperaturas cada día es mayor en el desierto. La altura de la región sombreada lo deja claro.

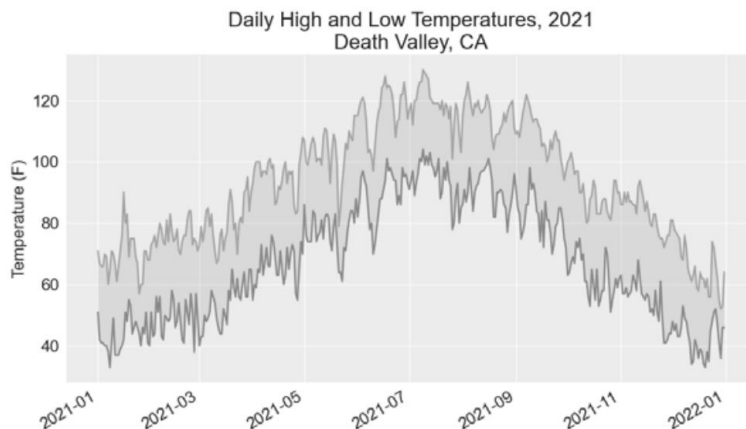


Figura 16-6: Temperaturas máximas y mínimas diarias para el Valle de la Muerte

A muchos conjuntos de datos con los que trabaja les faltarán datos, estarán formateados incorrectamente o serán incorrectos. Puede utilizar las herramientas que aprendió en la primera mitad de este libro para manejar estas situaciones. Aquí usamos un bloque try-except-else para manejar los datos faltantes. A veces usará `continue` para omitir algunos datos, o usar `remove()` o `del` para eliminar algunos datos después de haberlos extraído. Utilice cualquier enfoque que funcione, siempre que el resultado sea una visualización significativa y precisa.

Descargar tus propios datos

Para descargar sus propios datos meteorológicos, siga estos pasos:

1. Visite el sitio en línea de datos climáticos de la NOAA en <https://www.ncdc.noaa.gov/cdo-web>. En la sección Descubrir datos por, haga clic en Herramienta de búsqueda. En el cuadro Seleccionar un conjunto de datos, elija Resúmenes diarios.
2. Seleccione un rango de fechas y, en la sección Buscar, elija Códigos postales. Ingrese el código postal que le interesa y haga clic en Buscar.
3. En la página siguiente, verá un mapa y cierta información sobre el área en la que se está enfocando. Debajo del nombre de la ubicación, haga clic en Ver detalles completos, o haga clic en el mapa y luego haga clic en Detalles completos.
4. Desplácese hacia abajo y haga clic en Lista de estaciones para ver las estaciones meteorológicas que están disponibles en esta área. Haga clic en uno de los nombres de las estaciones y luego haga clic en Agregar al carrito. Estos datos son gratuitos, aunque el sitio utiliza un icono de carrito de compras. En la esquina superior derecha, haz clic en el carrito.
5. En Seleccionar el formato de salida, elija CSV diario de GHCN personalizado. Hacer Asegúrese de que el rango de fechas sea correcto y haga clic en Continuar.

6. En la página siguiente, puede seleccionar los tipos de datos que desea. Puede descargar un tipo de datos (por ejemplo, centrándose en la temperatura del aire) o puede descargar todos los datos disponibles desde esta estación. Haga sus elecciones y luego haga clic en Continuar.
7. En la última página, verá un resumen de su pedido. Ingrese su dirección de correo electrónico y haga clic en Enviar pedido. Recibirás una confirmación de que tu pedido fue recibido y, en unos minutos, deberías recibir otro correo electrónico con un enlace para descargar tus datos.

Los datos que descargue deben estar estructurados igual que los datos con los que trabajamos en esta sección. Es posible que tenga encabezados diferentes a los que vio en esta sección, pero si sigue los mismos pasos que utilizamos aquí, debería poder generar visualizaciones de los datos que le interesan.

INTÉNTALO TÚ MISMO

16-1. Precipitaciones de Sitka: Sitka se encuentra en una selva tropical templada, por lo que recibe una buena cantidad de lluvia. En el archivo de datos `sitka_weather_2021_full.csv` hay un encabezado llamado PRCP, que representa las cantidades de lluvia diarias. Haga una visualización centrándose en los datos de esta columna. Puedes repetir el ejercicio para el Valle de la Muerte si tienes curiosidad sobre la poca lluvia que ocurre en un desierto.

16-2. Comparación de Sitka y el Valle de la Muerte: Las escalas de temperatura en los gráficos de Sitka y el Valle de la Muerte reflejan los diferentes rangos de datos. Para comparar con precisión el rango de temperatura en Sitka con el del Valle de la Muerte, se necesitan escalas idénticas en el eje y. Cambie la configuración para el eje y en uno o ambos gráficos en las Figuras 16-5 y 16-6. Luego haga una comparación directa entre los rangos de temperatura en Sitka y el Valle de la Muerte (o dos lugares cualesquiera que desee comparar).

16-3. San Francisco: ¿ Las temperaturas en San Francisco se parecen más a las temperaturas de Sitka o a las del Valle de la Muerte? Descargue algunos datos de San Francisco y genere un gráfico de temperaturas altas y bajas para San Francisco para hacer una comparación.

16-4. Índices automáticos: en esta sección, codificamos los índices correspondientes a las columnas TMIN y TMAX . Utilice la fila del encabezado para determinar los índices de estos valores, de modo que su programa pueda funcionar para Sitka o Death Valley. Utilice también el nombre de la estación para generar automáticamente un título apropiado para su gráfico.

16-5. Explorar: genere algunas visualizaciones más que examinen cualquier otro aspecto meteorológico que le interese para cualquier ubicación que le interese.

Mapeo de conjuntos de datos globales: formato GeoJSON

En esta sección, descargará un conjunto de datos que representa todos los terremotos que ocurrieron en el mundo durante el mes anterior. Luego hará un mapa que muestre la ubicación de estos terremotos y su importancia.

cada uno lo era. Debido a que los datos se almacenan en formato GeoJSON, trabajaremos con ellos usando el módulo `json`. Usando el diagrama `scatter_geo()` de `Plotly`, creará visualizaciones que muestren claramente la distribución global de los terremotos.

Descarga de datos sobre terremotos

Cree una carpeta llamada `eq_data` dentro de la carpeta donde guarda los programas de este capítulo. Copie el archivo `eq_1_day_m1.geojson` en esta nueva carpeta.

Los terremotos se clasifican según su magnitud en la escala de Richter. Este archivo incluye datos de todos los terremotos con una magnitud M1 o mayor que tuvieron lugar en las últimas 24 horas (al momento de escribir este artículo). Estos datos provienen de una de las fuentes de datos de terremotos del Servicio Geológico de los Estados Unidos, en <https://earthquake.usgs.gov/earthquakes/feed>.

Examinando datos GeoJSON

Cuando abres `eq_1_day_m1.geojson`, verás que es muy denso y difícil de leer:

```
{
  "type": "FeatureCollection",
  "metadata": {
    "generated": 1649052296000,
    ...
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "mag": 1.6,
        "place": "63 km al SE de Peda..."
      },
      ...
    },
    {
      "type": "Feature",
      "properties": {
        "mag": 2.2,
        "place": "27 km al SSE de Ca..."
      },
      ...
    },
    {
      "type": "Feature",
      "properties": {
        "mag": 3.7,
        "place": "102 km al SSE de S..."
      },
      ...
    },
    {
      "type": "Feature",
      "properties": {
        "mag": 2.92000008,
        "place": "49 km al SE..."
      },
      ...
    },
    {
      "type": "Feature",
      "properties": {
        "mag": 1.4,
        "place": "44 km al NE de Sus..."
      },
      ...
    }
  ]
}
```

Este archivo está formateado más para máquinas que para humanos. Pero podemos ver que el archivo contiene algunos diccionarios, así como información que nos interesa, como magnitudes y ubicaciones de los terremotos.

El módulo `json` proporciona una variedad de herramientas para explorar y trabajar con datos JSON. Algunas de estas herramientas nos ayudarán a reformatear el archivo para que podamos ver los datos sin procesar más fácilmente antes de trabajar con ellos mediante programación.

Comencemos cargando los datos y mostrándolos en un formato que sea más fácil de leer. Este es un archivo de datos largo, por lo que en lugar de imprimirlo, reescribiremos los datos en un archivo nuevo. Luego podremos abrir ese archivo y desplazarnos hacia adelante y hacia atrás por los datos más fácilmente:

```
eq_explorar desde pathlib importar ruta
_datos.py   importar json

# Leer datos como una cadena y convertirlos en un objeto Python.
ruta = Ruta('eq_data/eq_data_1_day_m1.geojson')
contenido = ruta.read_text()
1 all_eq_data = json.loads(contenido)

# Cree una versión más legible del archivo de datos.
2 ruta = Ruta('eq_data/readable_eq_data.geojson')
3 contenido_legible = json.dumps(all_eq_data, sangría=4)
  ruta.write_text(contenido_legible)
```

Leemos el archivo de datos como una cadena y usamos `json.loads()` para convertir la representación de cadena del archivo en un objeto Python 1. Este es el mismo enfoque que usamos en el Capítulo 10. En este caso, se convierte todo el conjunto de datos. a un único diccionario, que asignamos a `all_eq_data`. Luego definimos una nueva ruta donde podemos escribir estos mismos datos en un formato más legible. 2. La función `json.dumps()` que vio en el Capítulo 10 puede tomar una sangría opcional. argumento 3, que le indica cuánto sangrar los elementos anidados en los datos estructura.

Cuando buscas en tu directorio `eq_data` y abres el archivo `readable_eq_data.json`, aquí está la primera parte de lo que verá:

```
legible_eq      {
  _datos.json    "tipo": "Colección de características",
                  1 "metadatos": {
                      "generado": 1649052296000,
                      "url": "https://earthquake.usgs.gov/earthquakes/.../1.0_day.geojson",
                      "title": "Terremotos de magnitud 1.0+ del USGS, el día anterior",
                      "estado": 200,
                      "api": "1.10.3",
                      "cuenta": 160
                    },
                  2 "características": [
                      --recorte--
```

La primera parte del archivo incluye una sección con la clave "metadatos" 1. Esto nos indica cuándo se generó el archivo de datos y dónde podemos encontrarlos en línea. También nos proporciona un título legible por humanos y la cantidad de terremotos incluidos en este archivo. En este período de 24 horas se registraron 160 terremotos.

Este archivo GeoJSON tiene una estructura que resulta útil para datos basados en la ubicación. La información se almacena en una lista asociada con las "características" clave 2. Debido a que este archivo contiene datos de terremotos, los datos están en forma de lista donde cada elemento de la lista corresponde a un solo terremoto. Esta estructura puede parecer confusa, pero es bastante poderosa. Permite a los geólogos almacenar tanta información como necesiten en un diccionario sobre cada terremoto y luego agrupar todos esos diccionarios en una gran lista.

Veamos un diccionario que representa un solo terremoto:

```
legible_eq      --recorte--
  _datos.json    {
                  "tipo": "Característica",
                  1  "propiedades": {
                      "mag": 1,6,
                      --recorte--
                  2  "title": "M 1,6 - 27 km al NO de Susitna, Alaska"
                    },
                  3  "geometría": {
                      "tipo": "Punto",
                      "coordenadas": [
                  4    -150.7585,
                  5    61.7591,
                      56.3
```

```
    ],
    },
    "id": "ak0224bju1jx"
  },

```

Las "propiedades" clave contienen mucha información sobre cada terremoto 1. Nos interesa principalmente la magnitud de cada terremoto, asociada con la "mag" clave. También nos interesa el "título" de cada evento, que proporciona un buen resumen de su magnitud y ubicación 2.

La "geometría" clave nos ayuda a comprender dónde ocurrió el terremoto 3. Necesitaremos esta información para mapear cada evento. Podemos encontrar la longitud 4. y la latitud 5 para cada terremoto en una lista asociada con la clave "coordenadas".

Este archivo contiene mucho más anidamiento del que usaríamos en el código que escribimos. Así que si parece confuso, no te preocupes: Python manejará la mayor parte de la complejidad. Solo trabajaremos con uno o dos niveles de anidación a la vez. Comenzaremos sacando un diccionario para cada terremoto registrado en el período de 24 horas.

NOTA Cuando hablamos de ubicaciones, a menudo decimos primero la latitud de la ubicación, seguida de su longitud. Esta convención probablemente surgió porque los humanos descubrieron la latitud mucho antes de que desarrollemos el concepto de longitud. Sin embargo, muchos marcos geoespaciales enumeran primero la longitud y luego la latitud, porque esto corresponde a la convención (x, y) que usamos en las representaciones matemáticas. El formato GeoJSON sigue la convención (longitud, latitud). Si utiliza un marco diferente, es importante saber qué convención sigue ese marco.

Hacer una lista de todos los terremotos

Primero, haremos una lista que contenga toda la información sobre cada terremoto ocurrido.

```
eq_explorar desde pathlib importar ruta
_datos.py   importar json

# Leer datos como una cadena y convertirlos en un objeto Python.
ruta = Ruta('eq_data/eq_data_1_day_m1.geojson')
contenido = ruta.read_text()
all_eq_data = json.loads(contenido)

# Examina todos los terremotos en el conjunto de datos.
all_eq_dicts = all_eq_data['características']
imprimir(len(all_eq_dicts))
```

Tomamos los datos asociados con las 'características' clave en all_eq_data diccionario y asígnalo a all_eq_dicts. Sabemos que este archivo contiene registros de 160 terremotos y el resultado verifica que hemos capturado todos los terremotos en el archivo:

160

Observe lo corto que es este código. El archivo cuidadosamente formateado `readable_eq_data.json` tiene más de 6000 líneas. Pero en solo unas pocas líneas, podemos leer todos esos datos y almacenarlos en una lista de Python. A continuación, extraeremos las magnitudes de cada terremoto.

Extrayendo magnitudes

Podemos recorrer la lista que contiene datos sobre cada terremoto y extraer cualquier información que queramos. Saquemos la magnitud de cada terremoto:

```
eq_explorar      --recorte--
_datos.py        all_eq_dicts = all_eq_data['características']

1 revistas = []
  para eq_dict en all_eq_dicts:
2 mag = eq_dict['properties']['mag']
    revistas.append(mag)

imprimir(revistas[:10])
```

Hacemos una lista vacía para almacenar las magnitudes y luego recorremos la lista `all_eq_dicts`

1. Dentro de este bucle, cada terremoto está representado por el diccionario `eq_dict`. La magnitud de cada terremoto se almacena en las 'propiedades'

sección de este diccionario, bajo la clave 'mag'

2. Almacenamos cada magnitud en la variable `mag` y luego la agregamos a la lista `mags`.

Imprimimos las primeras 10 magnitudes, para que podamos ver si estamos obteniendo los datos correctos:

```
[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]
```

A continuación, extraeremos los datos de ubicación de cada terremoto y luego podremos hacer un mapa de los terremotos.

Extracción de datos de ubicación

Los datos de ubicación de cada terremoto se almacenan bajo la clave "geometría".

Dentro del diccionario de geometría hay una clave de "coordenadas" y los dos primeros valores de esta lista son la longitud y la latitud. Así es como extraeremos estos datos:

```
eq_explorar      --recorte--
_datos.py        all_eq_dicts = all_eq_data['características']

mags, lons, lats = [], [], []
para eq_dict en all_eq_dicts:
    mag = eq_dict['properties']['mag']
1 lon = eq_dict['geometría']['coordenadas'][0]
    lat = eq_dict['geometría']['coordenadas'][1]
    revistas.append(mag)
    lons.append(lon)
```

```
lats.append(lat)
```

```
imprimir(revistas[:10])
```

```
imprimir(lons[:5])
```

```
imprimir(latitud[:5])
```

Hacemos listas vacías para las longitudes y latitudes. El código `eq_dict['geometría']` accede al diccionario que representa el elemento geométrico del terremoto 1. La segunda clave, 'coordenadas', extrae la lista de valores asociados con las 'coordenadas'. Finalmente, el índice 0 solicita el primer valor de la lista de coordenadas, que corresponde a la longitud de un terremoto.

Cuando imprimimos las primeras 5 longitudes y latitudes, el resultado muestra que estamos obteniendo los datos correctos:

```
[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]
[-150.7585, -153.4716, -148.7531, -159.6267, -155.248336791992]
[61.7591, 59.3152, 63.1633, 54.5612, 18.7551670074463]
```

Con estos datos, podemos pasar al mapeo de cada terremoto.

Construyendo un mapa mundial

Usando la información que hemos obtenido hasta ahora, podemos construir un mapa mundial simple. Aunque todavía no se verá presentable, queremos asegurarnos de que la información se muestre correctamente antes de centrarnos en cuestiones de estilo y presentación.

Aquí está el mapa inicial:

```
eq_mundo
_mapa.py

desde pathlib importar ruta
importar json

importar plotly.express como px

--recorte--
para eq_dict en all_eq_dicts:
    --recorte--

título = 'Terremotos globales'
1 figura = px.scatter_geo(lat=lats, lon=lons, título=título)
figura.mostrar()
```

Importamos `plotly.express` con el alias `px`, tal como lo hicimos en el Capítulo 15. La función `scatter_geo()` 1 le permite superponer un diagrama de dispersión de datos geográficos en un mapa. En el uso más simple de este tipo de gráfico, solo necesita proporcionar una lista de latitudes y una lista de longitudes. Pasamos la lista `lats` al argumento `lat` y `lons` al argumento `lon`.

Cuando ejecute este archivo, debería ver un mapa similar al de la Figura 16-7. Esto muestra nuevamente el poder de la biblioteca `Plotly Express`; En sólo tres líneas de código, tenemos un mapa de la actividad sísmica global.

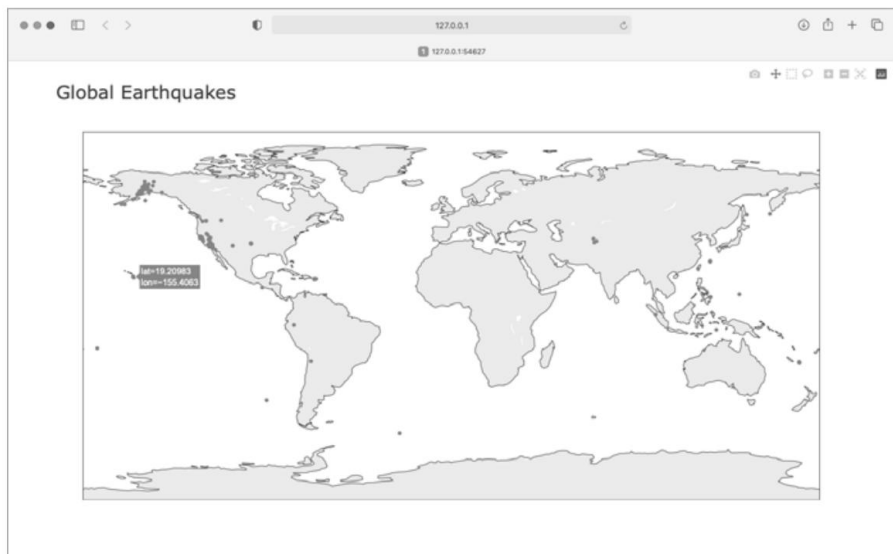


Figura 16-7: Un mapa simple que muestra dónde ocurrieron todos los terremotos en las últimas 24 horas

Ahora que sabemos que la información de nuestro conjunto de datos se está trazando correctamente, podemos realizar algunos cambios para que el mapa sea más significativo y más fácil de leer.

Representando Magnitudes

Un mapa de la actividad sísmica debería mostrar la magnitud de cada terremoto. También podemos incluir más datos, ahora que sabemos que los datos se están trazando correctamente.

```
--recorte--
# Leer datos como una cadena y convertirlos en un objeto Python.
ruta = Ruta('eq_data/eq_data_30_day_m1.geojson')
contenido = ruta.read_text()
--recorte--

título = 'Terremotos globales'
fig = px.scatter_geo(lat=lats, lon=lons, tamaño=mags, título=título)
figura.mostrar()
```

Cargamos el archivo `eq_data_30_day_m1.geojson`, para incluir el valor completo de 30 días de actividad sísmica. También usamos el argumento de `tamaño` en `px.scatter_geo()` llamada, que especifica cómo se dimensionarán los puntos en el mapa. Pasamos la lista de revistas al `tamaño`, por lo que los terremotos con una magnitud mayor aparecerán como puntos más grandes en el mapa.

El mapa resultante se muestra en la Figura 16-8. Los terremotos generalmente ocurren cerca de los límites de las placas tectónicas, y el período más largo de actividad sísmica incluido en este mapa revela las ubicaciones exactas de estos límites.

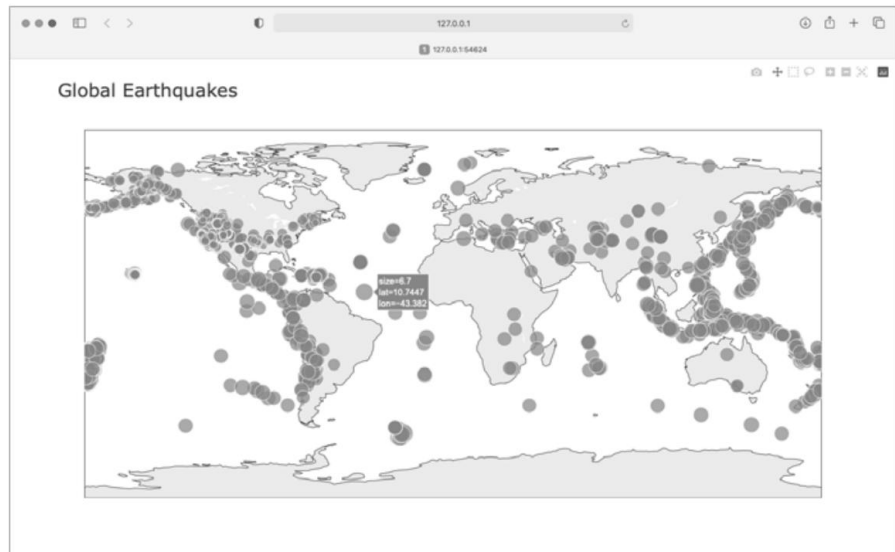


Figura 16-8: El mapa ahora muestra la magnitud de todos los terremotos en los últimos 30 días.

Este mapa es mejor, pero todavía es difícil distinguir qué puntos representan los terremotos más importantes. También podemos mejorar esto aún más usando colores para representar magnitudes.

Personalización de los colores de los marcadores

Podemos utilizar las escalas de color de Plotly para personalizar el color de cada marcador, según la gravedad del terremoto correspondiente. También usaremos una proyección diferente para el mapa base.

```
eq_mundo
_mapa.py  --recorte--
fig = px.scatter_geo(lat=lats, lon=lons, size=mags, title=título,
1 color = revistas,
2 color_continuous_scale='Viridis',
3 etiquetas = {'color': 'Magnitud'},
4 proyección = 'tierra natural',
)
figura.mostrar()
```

Todos los cambios significativos aquí ocurren en la llamada a la función `px.scatter_geo()`. El argumento `color` le dice a Plotly qué valores debe usar para determinar dónde cae cada marcador en la escala de color 1. Usamos la lista de revistas para determinar el color de cada punto, tal como lo hicimos con el argumento tamaño.

El argumento `color_continuous_scale` le dice a Plotly qué escala de color usar 2. `Viridis` es una escala de color que va del azul oscuro al amarillo brillante y funciona bien para este conjunto de datos. De forma predeterminada, la escala de colores a la derecha del mapa está etiquetada como `color`; esto no es representativo de lo que realmente significan los colores. El argumento de etiquetas, que se muestra en el Capítulo 15, toma un diccionario como valor 3. Solo necesitamos establecer una etiqueta personalizada en este gráfico, asegurándonos de que la escala de colores esté etiquetada como `Magnitud` en lugar de `color`.

Agregamos un argumento más, para modificar el mapa base sobre el cual se trazan los terremotos. El argumento de la proyección acepta una serie de proyecciones cartográficas comunes 4. Aquí utilizamos la proyección de la 'tierra natural', que redondea los extremos del mapa. Además, tenga en cuenta la coma final después de este último argumento. Cuando una llamada a función tiene una larga lista de argumentos que abarcan varias líneas como esta, es una práctica común agregar una coma al final para que siempre esté listo para agregar otro argumento en la siguiente línea.

Cuando ejecute el programa ahora, verá un mapa mucho más atractivo.

En la Figura 16-9, la escala de colores muestra la gravedad de los terremotos individuales; los terremotos más severos se destacan como puntos de color amarillo claro, en contraste con muchos puntos más oscuros. También puedes saber qué regiones del mundo tienen una actividad sísmica más significativa.

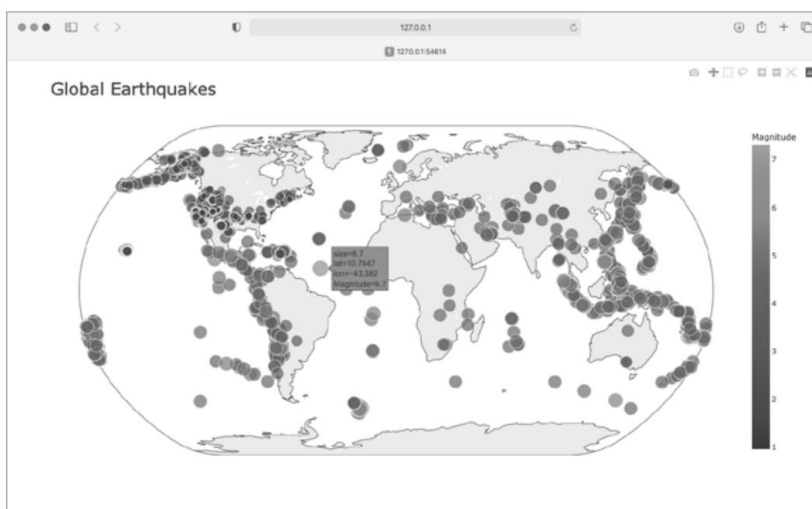


Figura 16-9: En terremotos de 30 días, el color y el tamaño se utilizan para representar la magnitud de cada terremoto.

Otras escalas de colores

Puede elegir entre otras escalas de colores. Para ver las escalas de colores disponibles, ingrese las dos líneas siguientes en una sesión de terminal Python:

```
>>> importar plotly.express como px
>>> px.colores.named_colorscales()
['aggrrnyl', 'agsunset', 'blackbody', ..., 'mygbm']
```

Siéntase libre de probar estas escalas de colores en el mapa de terremotos o con cualquier conjunto de datos donde los colores que varían continuamente puedan ayudar a mostrar patrones en los datos.

Agregar texto flotante

Para finalizar este mapa, agregaremos un texto informativo que aparece cuando pasas el cursor sobre el marcador que representa un terremoto. Además de mostrar la longitud y la latitud, que aparecen de forma predeterminada, mostraremos la magnitud y también proporcionaremos una descripción de la ubicación aproximada.

Para realizar este cambio, necesitamos extraer un poco más de datos del archivo:

```
eq_mundo --recorte--
_mapa.py 1 mags, lons, lats, eq_titles = [], [], [], []
          mag = eq_dict['properties']['mag']
          lon = eq_dict['geometría']['coordenadas'][0]
          lat = eq_dict['geometría']['coordenadas'][1]
2 eq_title = eq_dict['properties']['título']
          revistas.append(mag)
          lons.append(lon)
          lats.append(lat)
          eq_titles.append(eq_título)

          título = 'Terremotos globales'
          fig = px.scatter_geo(lat=lats, lon=lons, size=mags, title=título,
                               --recorte--
                               proyección='tierra natural',
3                               hover_name=eq_titles,
                               )
          figura.mostrar()
```

Primero hacemos una lista llamada `eq_titles` para almacenar el título de cada terremoto 1. La sección 'título' de los datos contiene un nombre descriptivo de la magnitud y ubicación de cada terremoto, además de su longitud y latitud. Extraemos esta información y la asignamos a la variable `eq_title 2`, y luego la agregamos a la lista `eq_titles`.

En la llamada a `px.scatter_geo()`, pasamos `eq_titles` al argumento 3 de `hover_name`. Plotly ahora agregará la información del título de cada terremoto al texto flotante en cada punto. Cuando ejecute este programa, debería poder pasar el cursor sobre cualquier marcador, ver una descripción de dónde ocurrió ese terremoto y leer su magnitud exacta. Un ejemplo de esta información se muestra en la Figura 16-10.

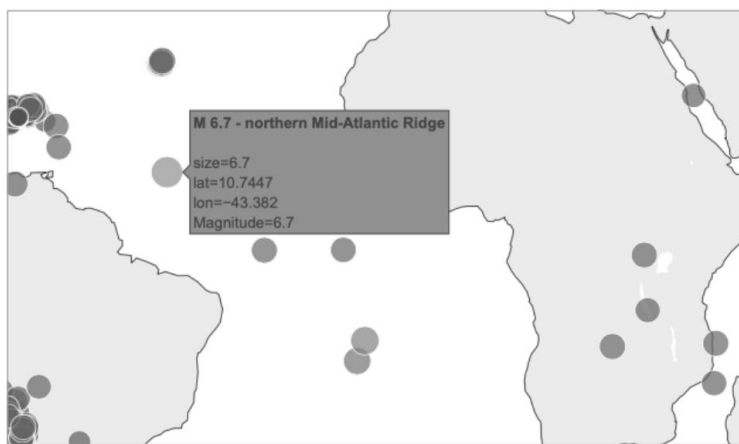


Figura 16-10: El texto flotante ahora incluye un resumen de cada terremoto.

¡Esto es impresionante! En menos de 30 líneas de código, hemos creado un mapa visualmente atractivo y significativo de la actividad sísmica global que también

Ilustra la estructura geológica del planeta. Plotly ofrece una amplia gama de formas de personalizar la apariencia y el comportamiento de sus visualizaciones. Usando las muchas opciones de Plotly, puedes crear gráficos y mapas que muestren exactamente lo que deseas.

INTÉNTALO TÚ MISMO

16-6. Refactorización: el bucle que extrae datos de `all_eq_dicts` utiliza variables para la magnitud, longitud, latitud y título de cada terremoto antes de agregar estos valores a sus listas apropiadas. Este enfoque se eligió para aclarar cómo extraer datos de un archivo GeoJSON, pero no es necesario en su código.

En lugar de utilizar estas variables temporales, extraiga cada valor de `eq_dict` y agréguelo a la lista adecuada en una línea. Hacerlo debería acortar el cuerpo de este bucle a solo cuatro líneas.

16-7. Título automatizado: en esta sección utilizamos el título genérico Global Earthquakes. En su lugar, puede utilizar el título del conjunto de datos en la parte de metadatos del archivo GeoJSON. Extraiga este valor y asígnelo al título de la variable.

16-8. Terremotos recientes: puede encontrar archivos de datos en línea que contienen información sobre los terremotos más recientes durante períodos de 1 hora, 1 día, 7 días y 30 días. Vaya a <https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php> y verá una lista de enlaces a conjuntos de datos para varios períodos de tiempo, centrándose en terremotos de diferentes magnitudes. Descargue uno de estos conjuntos de datos y cree una visualización de la actividad sísmica más reciente.

16-9. Incendios mundiales: en los recursos de este capítulo, encontrará un archivo llamado `world_fires_1_day.csv`. Este archivo contiene información sobre incendios que arden en diferentes lugares del mundo, incluida la latitud, longitud y brillo de cada incendio. Utilizando el trabajo de procesamiento de datos de la primera parte de este capítulo y el trabajo de mapeo de esta sección, haga un mapa que muestre qué partes del mundo se ven afectadas por los incendios.

Puede descargar versiones más recientes de estos datos en <https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data>. Puede encontrar enlaces a los datos en formato CSV en la sección Archivos SHP, KML y TXT.

Resumen

En este capítulo, aprendió a trabajar con conjuntos de datos del mundo real. Procesó archivos CSV y GeoJSON y extrajo los datos en los que desea centrarse. Utilizando datos meteorológicos históricos, aprendió más sobre cómo trabajar con Matplotlib, incluido cómo usar el módulo de fecha y hora y cómo trazar varias series de datos en un gráfico. Trazó datos geográficos en un mapa mundial en Plotly y aprendió a personalizar el estilo del mapa.

A medida que adquieras experiencia trabajando con archivos CSV y JSON, podrás para procesar casi cualquier dato que desee analizar. Puede descargar la mayoría de los conjuntos de datos en línea en uno o ambos formatos. Al trabajar con estos

formatos, también podrá aprender a trabajar con otros formatos de datos más fácilmente.

En el próximo capítulo, escribirá programas que recopilan automáticamente sus propios datos de fuentes en línea y luego creará visualizaciones de esos datos. Estas son habilidades divertidas si quieres programar como pasatiempo y son habilidades críticas si estás interesado en programar profesionalmente.

17

TRABAJANDO CON APIS



En este capítulo, aprenderá a escribir un programa autónomo que genere una visualización basada en los datos que recupera. Su

El programa utilizará una interfaz de programación de aplicaciones (API) para solicitar automáticamente información específica de un sitio web y luego usará esa información para generar una visualización. Debido a que los programas escritos como este siempre usarán datos actuales para generar una visualización, incluso cuando esos datos puedan estar cambiando rápidamente, la visualización siempre estará actualizada.

Usando una API

Una API es parte de un sitio web diseñado para interactuar con programas. Esos programas utilizan URL muy específicas para solicitar cierta información. Este tipo de solicitud se denomina llamada API. Los datos solicitados serán devueltos en un

formato de fácil procesamiento, como JSON o CSV. La mayoría de las aplicaciones que utilizan fuentes de datos externas, como las aplicaciones que se integran con sitios de redes sociales, dependen de llamadas API.

Git y GitHub

Basaremos nuestra visualización en información de GitHub (<https://github.com>), un sitio que permite a los programadores colaborar en proyectos de codificación. Usaremos la API de GitHub para solicitar información sobre proyectos de Python en el sitio y luego generaremos una visualización interactiva de la popularidad relativa de estos proyectos usando Plotly.

GitHub toma su nombre de Git, un sistema de control de versiones distribuido.

Git ayuda a las personas a administrar su trabajo en un proyecto de una manera que evita que los cambios realizados por una persona interfieran con los cambios que realizan otras personas. Cuando implementas una nueva característica en un proyecto, Git rastrea los cambios que realizas en cada archivo. Cuando tu nuevo código funcione, te comprometes los cambios que has realizado y Git registra el nuevo estado de tu proyecto. Si comete un error y desea revertir los cambios, puede volver fácilmente a cualquier estado de funcionamiento anterior. (Para obtener más información sobre el control de versiones usando Git, consulte el Apéndice D.) Los proyectos en GitHub se almacenan en repositorios, que contienen todo lo asociado con el proyecto: su código, información sobre sus colaboradores, cualquier problema o informe de error, etc.

Cuando a los usuarios de GitHub les gusta un proyecto, pueden destacarlo para mostrar su apoyo y realizar un seguimiento de los proyectos que podrían querer utilizar. En este capítulo, escribiremos un programa para descargar automáticamente información sobre los proyectos Python más destacados en GitHub y luego crearemos una visualización informativa de estos proyectos.

Solicitar datos mediante una llamada API

La API de GitHub te permite solicitar una amplia gama de información a través de llamadas API. Para ver cómo se ve una llamada API, ingrese lo siguiente en la barra de direcciones de su navegador y presione ENTER:

<https://api.github.com/search/repositories?q=language:python+sort:stars>

Esta llamada devuelve la cantidad de proyectos de Python alojados actualmente en GitHub, así como información sobre los repositorios de Python más populares. Examinemos la llamada. La primera parte, <https://api.github.com/>, dirige la solicitud a la parte de GitHub que responde a las llamadas API. La siguiente parte, `búsqueda/repositorios`, le dice a la API que realice una búsqueda en todos los repositorios en GitHub.

El signo de interrogación después de los repositorios indica que estamos a punto de aprobar un argumento. La `q` significa consulta y el signo igual (`=`) nos permite comenzar a especificar una consulta (`q=`). Al usar `lenguaje:python`, indicamos que queremos información solo sobre repositorios que tienen Python como idioma principal.

La parte final, `+sort:stars`, clasifica los proyectos por el número de estrellas que han recibido.

El siguiente fragmento muestra las primeras líneas de la respuesta:

```
{
1 "recuento_total": 8961993,
2 "resultados_incompletos": verdadero,
3 "elementos": [
    {
      "identificación": 54346799,
      "node_id": "MDEwOlJlcG9zaXRvcnk1NDM0Njc5OQ==",
      "nombre": "apis-públicas",
      "full_name": "apis-públicas/apis-públicas",
      --recorte--
    }
  ]
}
```

Puede ver en la respuesta que esta URL no está destinada principalmente a ser ingresada por humanos, porque está en un formato destinado a ser procesado por un programa. GitHub encontró poco menos de nueve millones de proyectos Python al momento de escribir este artículo 1. El valor de "incomplete_results" es verdadero, lo que nos indica que GitHub no procesó completamente la consulta 2. GitHub limita cuánto tiempo puede ejecutarse cada consulta, para mantener la API responde a todos los usuarios. En este caso encontró algunos de los repositorios de Python más populares, pero no tuvo tiempo de encontrarlos todos; Lo arreglaremos en un momento. Los "elementos" devueltos se muestran en la lista siguiente, que contiene detalles sobre los proyectos Python más populares en GitHub 3.

Instalar solicitudes

El paquete Solicitudes permite que un programa Python solicite fácilmente información de un sitio web y examine la respuesta. Utilice pip para instalar Solicitudes:

```
$ python -m pip install --solicitudes de usuario
```

Si utiliza un comando que no sea Python para ejecutar programas o iniciar un sesión terminal, como python3, su comando se verá así:

```
$ python3 -m pip install --solicitudes de usuario
```

Procesando una respuesta API

Ahora escribiremos un programa para emitir automáticamente una llamada API y procesar los resultados:

```
pitón      solicitudes de importación
_repos.py

# Realice una llamada API y verifique la respuesta.
1 URL = "https://api.github.com/search/repositories"
  url += "?q=idioma:python+ordenar:estrellas+estrellas:>10000"

2 encabezados = {"Acceptar": "application/vnd.github.v3+json"}
3 r = solicitudes.get(url, encabezados=encabezados)
4 print(f"Código de estado: {r.status_code}")
```

```
# Convertir el objeto de respuesta en un diccionario.
5 respuesta_dict = r.json()

# Resultados del proceso.
imprimir(respuesta_dict.claves())
```

Primero importamos el módulo de solicitudes . Luego asignamos la URL de la llamada API a la variable URL 1. Esta es una URL larga, por lo que la dividimos en dos líneas. La primera línea es la parte principal de la URL y la segunda línea es la cadena de consulta. Hemos incluido una condición más en la cadena de consulta original: estrellas:>10000, que le indica a GitHub que solo busque repositorios de Python que tengan más de 10,000 estrellas. Esto debería permitir a GitHub devolver un conjunto de resultados completo y consistente.

GitHub se encuentra actualmente en la tercera versión de su API, por lo que definimos encabezados para la llamada a la API que solicitan explícitamente usar esta versión de la API y devuelven los resultados en formato JSON 2. Luego usamos solicitudes para realizar la llamada al API 3. Llamamos a get() y le pasamos la URL y el encabezado que definimos, y asignamos el objeto de respuesta a la variable r.

El objeto de respuesta tiene un atributo llamado status_code, que nos dice si la solicitud fue exitosa. (Un código de estado de 200 indica una respuesta exitosa). Imprimimos el valor de status_code para asegurarnos de que la llamada se realizó correctamente. 4. Le pedimos a la API que devuelva la información en formato JSON, por lo que usamos json () método para convertir la información a un diccionario de Python 5. Asignamos el diccionario resultante a Response_dict.

Finalmente, imprimimos las claves de Response_dict y vemos el siguiente resultado:

```
Código de estado: 200
dict_keys(['total_count', 'incomplete_results', 'items'])
```

Como el código de estado es 200, sabemos que la solicitud se realizó correctamente. El diccionario de respuestas contiene sólo tres claves: 'total_count', 'incomplete_results' y 'elementos'. Echemos un vistazo al interior del diccionario de respuestas.

Trabajar con el diccionario de respuestas

Con la información de la llamada API representada como un diccionario, podemos trabajar con los datos almacenados allí. Generemos algún resultado que resuma la información. Esta es una buena manera de asegurarnos de que recibimos la información que esperábamos y de comenzar a examinar la información que nos interesa:

```
pitón
_repos.py
```

```
solicitudes de importación

# Realizar una llamada API y almacenar la respuesta.
--recorte--

# Convertir el objeto de respuesta en un diccionario.
respuesta_dict = r.json()
```



```

1 print(f"Repositorios totales: {response_dict['total_count']}")
  print(f"Resultados completos: {no_dictamen_respuesta['resultados_incompletos']}")

  # Explorar información sobre los repositorios.
2 repo_dicts = respuesta_dict['elementos']
  print(f"Repositorios devueltos: {len(repo_dicts)}")

  # Examina el primer repositorio.
3 repo_dict = repositorio_dicts[0]
4 print(f"\nClaves: {len(repo_dict)}")
5 para clave en ordenado (repo_dict.keys()):
    imprimir (clave)

```

Comenzamos a explorar el diccionario de respuestas imprimiendo el valor asociado con 'total_count', que representa el número total de repositorios de Python devueltos por esta llamada API 1. También usamos el valor asociado con 'incomplete_results', así sabremos si GitHub pudo procesar completamente la consulta. En lugar de imprimir este valor directamente, imprimimos su opuesto: un valor de Verdadero indicará que recibimos un conjunto completo de resultados.

El valor asociado con 'elementos' es una lista que contiene varios diccionarios, cada uno de los cuales contiene datos sobre un repositorio de Python individual. Asignamos esta lista de diccionarios a repo_dicts 2. Luego imprimimos la longitud de repo_dicts para ver para cuántos repositorios tenemos información.

Para ver más de cerca la información devuelta sobre cada repositorio, sacamos el primer elemento de repo_dicts y lo asignamos a repo_dict 3. Luego imprimimos el número de claves en el diccionario para ver cuánta información tenemos 4. Finalmente, imprimimos toda Las claves del diccionario para ver qué tipo de información incluye 5.

Los resultados nos dan una imagen más clara de los datos reales:

```

Código de estado: 200
1 Total de repositorios: 248
2 Resultados completos: Verdadero
  Repositorios devueltos: 30

3 claves: 78
  permitir_forking
  URL_archivo
  archivado
  --recorte--
  URL
  visibilidad
  observadores
  cuenta_observadores

```

Al momento de escribir este artículo, solo hay 248 repositorios de Python con más de 10,000 estrellas 1. Podemos ver que GitHub pudo procesar completamente la llamada API 2. En esta respuesta, GitHub devolvió información sobre los primeros 30 repositorios que coincidan con las condiciones de nuestra consulta. Si queremos más repositorios, podemos solicitar páginas adicionales de datos.

La API de GitHub devuelve mucha información sobre cada repositorio: hay 78 claves en `repo_dict` 3. Cuando revises estas claves, tendrás una idea del tipo de información que puedes extraer sobre un proyecto. (La única forma de saber qué información está disponible a través de una API es leer la documentación o examinar la información a través del código, como lo estamos haciendo aquí).

Saquemos los valores de algunas de las claves en `repo_dict`:

```
pitón
_repos.py

--recorte--
# Examina el primer repositorio.
repo_dict = repositorio_dicts[0]

print("\nInformación seleccionada sobre el primer repositorio:")
1 print(f"Nombre: {repo_dict['nombre']}")
2 print(f"Propietario: {repo_dict['propietario']}['iniciar sesión']}")
3 print(f"Estrellas: {repo_dict['stargazers_count']}")
  print(f"Repositorio: {repo_dict['html_url']}")
4 print(f"Creado: {repo_dict['created_at']}")
5 print(f"Actualizado: {repo_dict['updated_at']}")
  print(f"Descripción: {repo_dict['descripcion']}")
```

Aquí, imprimimos los valores de varias claves del diccionario del primer repositorio. Comenzamos con el nombre del proyecto 1. Un diccionario completo representa al propietario del proyecto, por lo que usamos el propietario clave para acceder al diccionario que representa al propietario y luego usamos la clave de inicio de sesión para obtener el nombre de inicio de sesión del propietario 2. A continuación, Imprimimos cuántas estrellas ha obtenido el proyecto 3 y la URL del repositorio GitHub del proyecto. Luego mostramos cuándo se creó 4 y cuándo se actualizó por última vez 5. Finalmente, imprimimos la descripción del repositorio.

La salida debería verse así:

```
Código de estado: 200
Repositorios totales: 248
Resultados completos: Verdadero
Repositorios devueltos: 30

Información seleccionada sobre el primer repositorio:
Nombre: apis públicas
Propietario: apis públicas
Estrellas: 191493
Repositorio: https://github.com/public-apis/public-apis
Creado: 2016-03-20T23:49:42Z
Actualizado: 2022-05-12T06:37:11Z
Descripción: una lista colectiva de API gratuitas
```

Podemos ver que el proyecto Python más destacado en GitHub al momento de escribir este artículo es public-apis. Su propietario es una organización con el mismo nombre y ha sido protagonizada por casi 200.000 usuarios de GitHub. Podemos ver la URL del repositorio del proyecto, su fecha de creación de marzo de 2016 y que fue actualizado recientemente. Además, la descripción nos dice que public-apis contiene una lista de API gratuitas que podrían interesar a los programadores.

Resumiendo los principales repositorios

Cuando hagamos una visualización de estos datos, queremos incluir más de un repositorio. Escribamos un bucle para imprimir información seleccionada sobre cada repositorio que devuelve la llamada API para que podamos incluirlos todos en la visualización:

```

pitón      --snip--
_repos.py  # Explora información sobre los repositorios. repo_dicts
           = respuesta_dict['items'] print(f"Repositorios
           devueltos: {len(repo_dicts)}")

1 print("\nInformación seleccionada sobre cada repositorio:") 2 para
repo_dict en repo_dicts:
    print(f"\nNombre: {repo_dict['name']}")
    print(f"Propietario: {repo_dict['owner']['login']}")
    print(f"Estrellas: {repo_dict['stargazers_count']}")
    print(f"Repositorio: {repo_dict['html_url']}")
    print(f"Descripción: {repo_dict['descripcion ']}")

```

Primero imprimimos un mensaje introductorio 1. Luego recorremos todos los diccionarios en `repo_dicts` 2. Dentro del bucle, imprimimos el nombre de cada proyecto, su propietario, cuántas estrellas tiene, su URL en GitHub y el nombre del proyecto. descripción:

```

Código de estado: 200
Repositorios totales: 248
Resultados completos: Verdadero
Repositorios devueltos: 30

```

Información seleccionada sobre cada repositorio:

```

Nombre: public-apis
Propietario: public-
apis Estrellas:
191494 Repositorio: https://github.com/public-apis/public-apis
Descripción: una lista colectiva de API gratuitas

```

```

Nombre: cartilla-de-diseño-de-sistemas
Propietario: donnemartin
Estrellas:
179952 Repositorio: https://github.com/donnemartin/system-design-primer
Descripción: Aprenda a diseñar sistemas a gran escala. Preparación para el sistema
entrevista de diseño. Incluye tarjetas de Anki. --
recorte--

```

```

Nombre: PayloadsAllTheThings
Propietario:
swisskyrepo
Estrellas: 37227 Repositorio: https://github.com/swisskyrepo/
PayloadsAllTheThings Descripción: Una lista de cargas útiles y omisiones para la seguridad de aplicaciones
y Pentest/CTF

```

En estos resultados aparecen algunos proyectos interesantes y podría valer la pena analizar algunos. Pero no dedique demasiado tiempo aquí, porque estamos a punto de crear una visualización que hará que los resultados sean mucho más fáciles de leer.

Monitoreo de límites de tasa de API

La mayoría de las API tienen límites de velocidad, lo que significa que hay un límite en la cantidad de solicitudes que puede realizar en un período de tiempo determinado. Para ver si te estás acercando a los límites de GitHub, ingresa a https://api.github.com/rate_limit en un navegador web. Deberías ver una respuesta que comienza así:

```
{
  "recursos": {
    --recorte--
1 "buscar": {
2   "límite": 10,
3   "restante": 9,
4   "restablecer": 1652338832,
    "usado": 1,
    "recurso": "buscar"
  },
  --recorte--
```

La información que nos interesa es el límite de velocidad para la API de búsqueda 1. Vemos que el límite es de 10 solicitudes por minuto 2 y que nos quedan 9 solicitudes para el minuto actual 3. El valor asociado a la clave "reset" representa el tiempo en Unix o época de tiempo (el número de segundos desde la medianoche del 1 de enero de 1970) en el que nuestra cuota se restablecerá a 4. Si alcanza su cuota, recibirá una breve respuesta que le permitirá saber que ha alcanzado la Límite de API. Si alcanzas el límite, solo espera hasta tu cuota.

se reinicia.

NOTA Muchas API requieren que usted se registre y obtenga una clave API o un token de acceso para realizar llamadas API. Al momento de escribir este artículo, GitHub no tiene tal requisito, pero si obtiene un token de acceso, sus límites serán mucho más altos.

Visualización de repositorios usando Plotly

Hagamos una visualización usando los datos que hemos recopilado para mostrar la popularidad relativa de los proyectos de Python en GitHub. Haremos un gráfico de barras interactivo: la altura de cada barra representará la cantidad de estrellas que ha adquirido el proyecto y podrá hacer clic en la etiqueta de la barra para ir a la página de inicio de ese proyecto en GitHub.

Guarde una copia del programa en el que hemos estado trabajando como `python_repos_visual.py`, luego modifíquelo para que diga lo siguiente:

```
python_repos
_visual.py

solicitudes de importación
importar plotly.express como px

# Realice una llamada API y verifique la respuesta.
URL = "https://api.github.com/search/repositories"
url += "?q=idioma:python+ordenar:estrellas+estrellas:>10000"

encabezados = {"Aceptar": "aplicación/vnd.github.v3+json"}
r = solicitudes.get(url, encabezados=encabezados)
1 print(f"Código de estado: {r.status_code}")

# Procesar resultados generales.
respuesta_dict = r.json()
2 print(f"Resultados completos: {no dictamen_respuesta['resultados_incompletos']}")

# Procesar información del repositorio.
repo_dicts = respuesta_dict['elementos']
3 nombres_repositorios, estrellas = [], []
para repo_dict en repo_dicts:
    repo_names.append(repo_dict['nombre'])
    estrellas.append(repo_dict['stargazers_count'])

# Hacer visualización.
4 fig = px.bar(x=repo_names, y=estrellas)
figura.mostrar()
```

Importamos Plotly Express y luego realizamos la llamada a la API como lo hemos estado haciendo. Continuamos imprimiendo el estado de la respuesta de la llamada API para saber si hay un problema 1. Cuando procesamos los resultados generales, continuamos imprimiendo el mensaje confirmando que obtuvimos un conjunto completo de resultados 2. Eliminamos el resto de las llamadas `print()` porque ya no estamos en la fase exploratoria; sabemos que tenemos los datos que queremos.

Luego creamos dos listas vacías 3 para almacenar los datos que incluiremos en el gráfico inicial. Necesitaremos el nombre de cada proyecto para etiquetar las barras (`repo_names`) y el número de estrellas para determinar la altura de las barras (`estrellas`). En el bucle, agregamos el nombre de cada proyecto y la cantidad de estrellas que tiene a estas listas.

Hacemos la visualización inicial con solo dos líneas de código 4. Esto es consistente con la filosofía de Plotly Express de que debería poder ver su visualización lo más rápido posible antes de refinar su apariencia.

Aquí usamos la función `px.bar()` para crear un gráfico de barras. Pasamos la lista `repo_names` como argumento `x` y `estrellas` como argumento `y`.

La Figura 17-1 muestra el gráfico resultante. Podemos ver que los primeros proyectos son significativamente más populares que el resto, pero todos son proyectos importantes en el ecosistema Python.

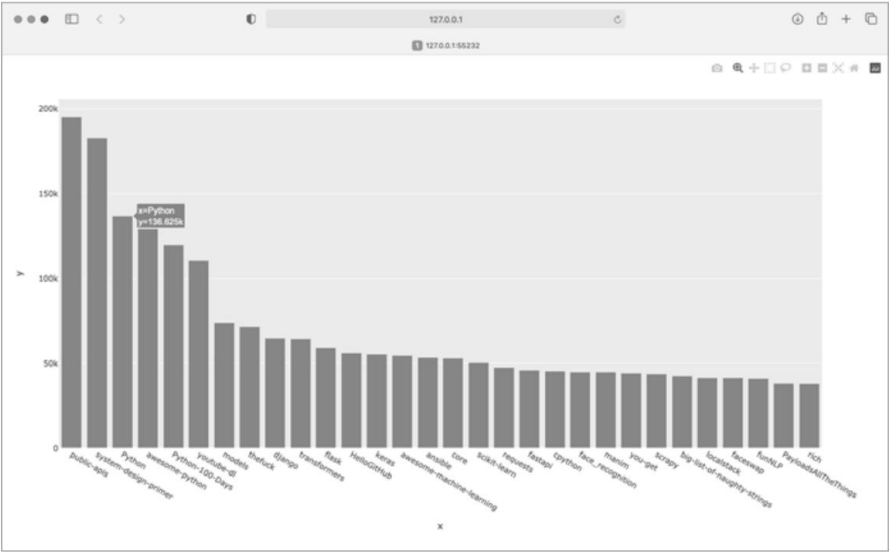


Figura 17-1: Los proyectos Python más destacados en GitHub

Aplicar estilo al gráfico

Plotly admite varias formas de diseñar y personalizar los gráficos, una vez que sepa que la información del gráfico es correcta. Haremos algunos cambios en la llamada inicial a `px.bar()` y luego haremos algunos ajustes adicionales en la figura. objeto después de haber sido creado.

Comenzaremos a diseñar el gráfico agregando un título y etiquetas para cada eje:

```
python_repos
_visual.py

--recorte--
# Hacer visualización.
title = "Proyectos Python más destacados en GitHub"
etiquetas = {'x': 'Repositorio', 'y': 'Estrellas'}
fig = px.bar(x=repo_names, y=estrellas, título=título, etiquetas=etiquetas)
```

```
1 fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                    yaxis_title_font_size=20)
```

```
figura.mostrar()
```

Primero agregamos un título y etiquetas para cada eje, como lo hicimos en los Capítulos 15 y 16. Luego usamos el método `fig.update_layout()` para modificar elementos específicos del gráfico 1. Plotly usa una convención donde los aspectos de un elemento del gráfico son conectados por guiones bajos. A medida que se familiarice con la documentación de Plotly, comenzará a ver patrones consistentes en cómo se nombran y modifican los diferentes elementos de un gráfico. Aquí configuramos el tamaño de fuente del título en 28 y el tamaño de fuente para cada título de eje en 20. El resultado se muestra en la Figura 17-2.

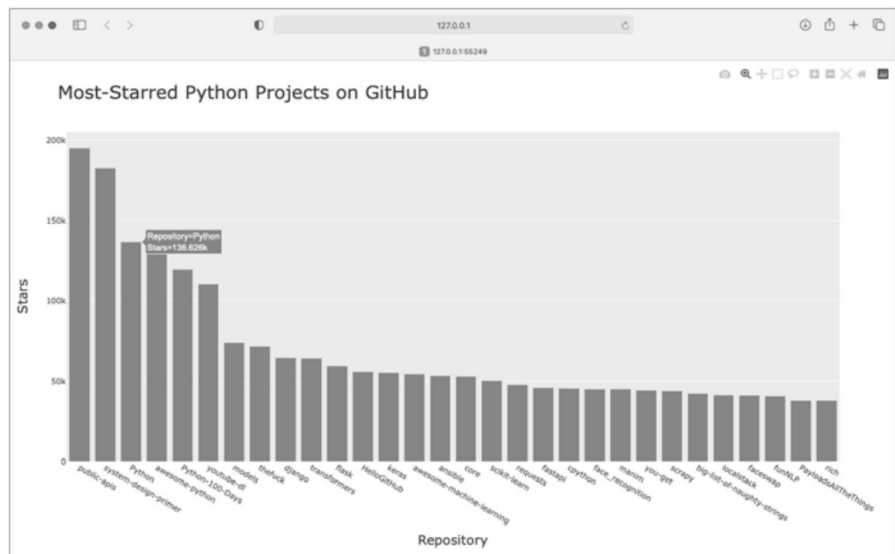


Figura 17-2: Se ha agregado un título al gráfico principal y también a cada eje.

Agregar información sobre

herramientas personalizadas En Plotly, puede colocar el cursor sobre una barra individual para mostrar la información que representa la barra. Esto comúnmente se denomina información sobre herramientas y, en este caso, muestra actualmente la cantidad de estrellas que tiene un proyecto. Creemos una información sobre herramientas personalizada para mostrar la descripción de cada proyecto, así como el propietario del proyecto.

Necesitamos obtener algunos datos adicionales para generar información sobre herramientas:

```
python_repos
_visual.py

--snip-- #
Procesar información del repositorio.
repo_dicts = Response_dict["items"] 1
repo_names, stars, hover_texts = [], [], [] para repo_dict en
repo_dicts:
    repo_names.append(repo_dict["name"])
    stars.append(repo_dict["stargazers_count"])

# Construir textos
flotantes. 2 propietario = repo_dict["propietario"] ["iniciar sesión"]
descripción = repo_dict["descripción"] 3 hover_text
= f"{propietario}<br />{descripción}" hover_texts.append(hover_text)

# Hacer visualización. title
= "Proyectos de Python más destacados en GitHub" etiquetas
= {'x': 'Repositorio', 'y': 'Estrellas'} 4 fig =
px.bar(x=repo_names, y=stars, title=title, etiquetas=etiquetas,
nombre_desplazamiento=textos_desplazamiento)
```

```
fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)
```

```
figura.mostrar()
```

Primero definimos una nueva lista vacía, `hover_texts`, para contener el texto que queremos mostrar para cada proyecto 1. En el bucle donde procesamos los datos, extraemos el propietario y la descripción de cada proyecto 2. Plotly le permite usar HTML código dentro de los elementos de texto, por lo que generamos una cadena para la etiqueta con un salto de línea (`<br />`) entre el nombre de usuario del propietario del proyecto y la descripción 3. Luego agregamos esta etiqueta a la lista `hover_texts`.

En la llamada a `px.bar()`, agregamos el argumento `hover_name` y lo pasamos `_texts` 4. Este es el mismo enfoque que utilizamos para personalizar la etiqueta de cada punto en el mapa de actividad sísmica global. A medida que Plotly crea cada barra, extraerá etiquetas de esta lista y solo las mostrará cuando el espectador pase el cursor sobre una barra. La Figura 17-3 muestra una de estas descripciones emergentes personalizadas.

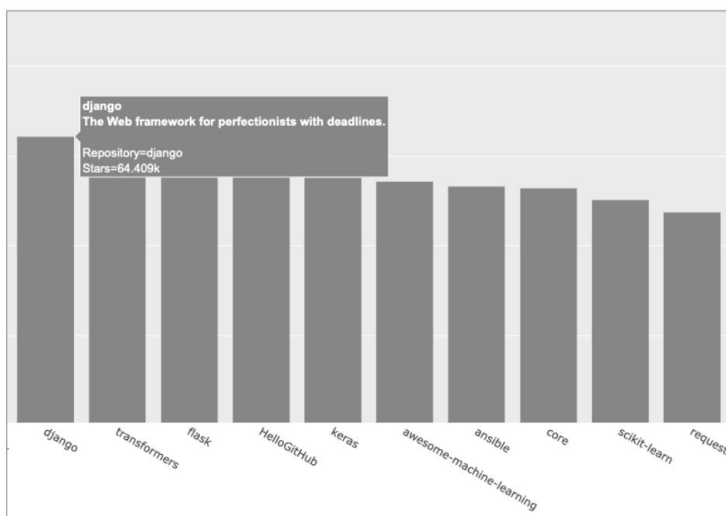


Figura 17-3: Al pasar el cursor sobre una barra se muestra el propietario y la descripción del proyecto.

Agregar enlaces en los que se puede hacer clic

Debido a que Plotly le permite usar HTML en elementos de texto, podemos agregar fácilmente enlaces a un gráfico. Usamos las etiquetas del eje x como una forma de permitir que el espectador visite la página de inicio de cualquier proyecto en GitHub. Necesitamos extraer las URL de los datos y usarlas al generar las etiquetas del eje x:

```
python_repos
_visual.py
--recorte--
# Procesar información del repositorio.
repo_dicts = respuesta_dict['elementos']
1 repo_links, estrellas, hover_texts = [], [], []
  para repo_dict en repo_dicts:
      # Convierta los nombres de los repositorios en enlaces activos.
      nombre_repo = dicta_repo['nombre']
      2 repositorio_url = repositorio_dict['html_url']
```



```

3 repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
  repo_links.append(repo_link)

  estrellas.append(repo_dict['stargazers_count'])
--recorte--

# Hacer visualización.
title = "Proyectos Python más destacados en GitHub"
etiquetas = {'x': 'Repositorio', 'y': 'Estrellas'}
fig = px.bar(x=repo_links, y=estrellas, título=título, etiquetas=etiquetas,
             hover_name=hover_textos)

fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

figura.mostrar()

```

Actualizamos el nombre de la lista que estamos creando de `repo_names` a `repo_links` para comunicar con mayor precisión el tipo de información que estamos reuniendo para el gráfico 1. Luego extraemos la URL del proyecto de `repo_dict` y la asignamos a la variable temporal `repo_url` 2. A continuación, generamos un enlace a el proyecto 3. Usamos la etiqueta de anclaje HTML, que tiene la forma `<a href='URL'>texto del enlace</a>`, para generar el enlace. Luego agregamos este enlace a `repo_links`.

Cuando llamamos a `px.bar()`, usamos `repo_links` para los valores de `x` en el gráfico. El resultado es el mismo que antes, pero ahora el espectador puede hacer clic en cualquiera de los nombres de los proyectos en la parte inferior del gráfico para visitar la página de inicio de ese proyecto en GitHub. ¡Ahora tenemos una visualización interactiva e informativa de los datos recuperados a través de una API!

Personalización de los colores de los marcadores

Una vez que se ha creado un gráfico, casi cualquier aspecto del mismo se puede personalizar mediante un método de actualización. Hemos utilizado el método `update_layout()` anteriormente. Se puede utilizar otro método, `update_traces()`, para personalizar los datos que se representan en un gráfico.

Cambiemos las barras a un azul más oscuro, con algo de transparencia:

```

--recorte--
fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

fig.update_traces(marker_color='AzulAcero', Marker_opacity=0.6)

figura.mostrar()

```

En Plotly, un seguimiento se refiere a una colección de datos en un gráfico. La actualización El método `_traces()` puede tomar varios argumentos diferentes; cualquier argumento que comience con `marcador_` afecta a los marcadores del gráfico. Aquí configuramos el color de cada marcador en 'SteelBlue'; cualquier color CSS con nombre funcionará aquí. También configuramos la opacidad de cada marcador en 0,6. Una opacidad de 1,0 será completamente opaca y una opacidad de 0 será completamente invisible.

Más información sobre Plotly y la API de GitHub

La documentación de Plotly es extensa y está bien organizada; sin embargo, puede resultar difícil saber por dónde empezar a leer. Un buen lugar para comenzar es con el artículo "Plotly Express en Python", en <https://plotly.com/python/plotly-express>. Esta es una descripción general de todos los gráficos que puede crear con Plotly Express y puede encontrar enlaces a artículos más extensos sobre cada tipo de gráfico individual.

Si desea comprender cómo personalizar mejor los gráficos de Plotly, el artículo "Diseñar figuras de Plotly Express en Python" ampliará lo que ha visto en los Capítulos 15 a 17. Puede encontrar este artículo en <https://plotly.com/python/estilo-plotly-express>.

Para obtener más información sobre la API de GitHub, consulte su documentación en <https://docs.github.com/en/rest>. Aquí aprenderá cómo extraer una amplia variedad de información de GitHub. Para ampliar lo que vio en este proyecto, busque la sección Buscar de la referencia en la barra lateral. Si tiene una cuenta de GitHub, puede trabajar con sus propios datos, así como con los datos disponibles públicamente de los repositorios de otros usuarios.

La API de noticias para hackers

Para explorar cómo usar llamadas API en otros sitios, echemos un vistazo rápido a Hacker News (<https://news.ycombinator.com>). En Hacker News, la gente comparte artículos sobre programación y tecnología y participa en animadas discusiones sobre esos artículos. La API de Hacker News brinda acceso a datos sobre todos los envíos y comentarios en el sitio, y puede usar la API sin tener que registrarse para obtener una clave.

La siguiente llamada devuelve información sobre el artículo principal actual al momento de escribir este artículo:

```
https://hacker-news.firebaseio.com/v0/item/31353677.json
```

Cuando ingresa esta URL en un navegador, verá que el texto de la página está entre llaves, lo que significa que es un diccionario. Pero la respuesta es difícil de examinar sin un mejor formato. Ejecutemos esta URL a través del método `json.dumps()`, como hicimos en el proyecto del terremoto en el Capítulo 16, para que podamos explorar el tipo de información que se devuelve sobre un artículo:

```
solicitudes de importación hn_article.py
importar json

# Realizar una llamada API y almacenar la respuesta.
URL = "https://hacker-news.firebaseio.com/v0/item/31353677.json"
r = solicitudes.get(url)
print(f"Código de estado: {r.status_code}")

# Explorar la estructura de los datos.
respuesta_dict = r.json()
cadena_respuesta = json.dumps(dict_respuesta, sangría = 4)
1 impresión (cadena_respuesta)
```

Todo en este programa debería resultarle familiar, porque lo hemos usado todo en los dos capítulos anteriores. La principal diferencia aquí es que podemos imprimir la cadena de respuesta formateada 1 en lugar de escribirla en un archivo, porque el resultado no es particularmente largo.

El resultado es un diccionario de información sobre el artículo con el ID 31353677:

```
{
  "por": "sohkamyung",
1 "descendientes": 302,
  "identificación": 31353677,
2 "niños": [
    31354987,
    31354235,
    --recorte--
  ],
  "puntuación": 785,
  "tiempo": 1652361401,
3 "título": "Los astrónomos revelan la primera imagen del agujero negro
  en el corazón de nuestra galaxia",
  "tipo": "historia",
4 "url": "https://public.nrao.edu/news/..."
}
```

El diccionario contiene una serie de claves con las que podemos trabajar. La clave "descendientes" nos dice el número de comentarios que ha recibido el artículo 1.

La clave "niños" proporciona los ID de todos los comentarios realizados directamente en respuesta a este envío 2. Cada uno de estos comentarios también puede tener sus propios comentarios, por lo que la cantidad de descendientes que tiene un envío suele ser mayor que su número de niños. Podemos ver el título del artículo que se está discutiendo 3 y también una URL para el artículo que se está discutiendo 4.

La siguiente URL devuelve una lista simple de todos los ID del top actual artículos sobre Hacker News:

<https://hacker-news.firebaseio.com/v0/topstories.json>

Podemos usar esta llamada para averiguar qué artículos están en la página de inicio en este momento y luego generar una serie de llamadas API similares a la que acabamos de examinar. Con este enfoque, podemos imprimir un resumen de todos los artículos que aparecen en la portada de Hacker News en este momento:

```
hn
_presentaciones.py

solicitudes de importación

# Realice una llamada API y verifique la respuesta.
1 URL = "https://hacker-news.firebaseio.com/v0/topstories.json"
  r = solicitudes.get(url)
  print(f"Código de estado: {r.status_code}")

# Procesar información sobre cada envío.
2 id_presentación = r.json()
```

```

3 dictámenes de envío = []
  para id_envío en id_envío[:5]:
    # Realice una nueva llamada API para cada envío.

4 URL = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
  r = solicitudes.get(url)
  print(f"id: {submission_id}\tstatus: {r.status_code}")
  respuesta_dict = r.json()

  # Construya un diccionario para cada artículo.
5 dict_envío = {
    'título': respuesta_dict['título'],
    'hn_link': f"https://news.ycombinator.com/item?id={submission_id}",
    'comentarios': respuesta_dict['descendientes'],
  }
6 sumisión_dicts.append(sumisión_dict)

7 dictámenes_envío = ordenados(dictamen_envío, clave=itemgetter('comentarios'),
                                reverso=Verdadero)

8 para sumisión_dict en sumisión_dicts:
  print(f"\nTítulo: {submission_dict['título']}")
  print(f"Enlace de discusión: {submission_dict['hn_link']}")
  print(f"Comentarios: {submission_dict['comentarios']}")

```

Primero, realizamos una llamada a la API e imprimimos el estado de la respuesta 1. Esta llamada a la API devuelve una lista que contiene los ID de hasta 500 de los artículos más populares de Hacker News en el momento en que se emite la llamada. Luego convertimos el objeto de respuesta a una lista 2 de Python, que asignamos a `submit_ids`. Usaremos estos ID para crear un conjunto de diccionarios, cada uno de los cuales contiene información sobre uno de los envíos actuales.

Configuramos una lista vacía llamada `submit_dicts` para almacenar estos diccionarios 3. Luego recorremos los ID de los 30 envíos principales. Realizamos una nueva llamada API para cada envío generando una URL que incluye el valor actual de `submit_id` 4. Imprimimos el estado de cada solicitud junto con su ID, para que podamos ver si se realizó correctamente.

A continuación, creamos un diccionario para el envío que se está procesando actualmente. 5. Almacenamos el título del envío, un enlace a la página de discusión para ese elemento y la cantidad de comentarios que el artículo ha recibido hasta el momento. Luego agregamos cada `submit_dict` a la lista `submit_dicts` 6.

Cada envío en Hacker News se clasifica según una puntuación general basada en una serie de factores, incluyendo cuántas veces se votó, cuántos comentarios recibió y qué tan reciente es el envío. Queremos ordenar la lista de diccionarios por el número de comentarios.

Para hacer esto, usamos una función llamada `itemgetter()` 7, que proviene del módulo `operator`. Le pasamos a esta función la clave 'comentarios' y extrae el valor asociado con esa clave de cada diccionario de la lista. El ordenado() La función luego usa este valor como base para ordenar la lista. Ordenamos la lista en orden inverso, para colocar primero las historias más comentadas.

Una vez ordenada la lista, recorremos la lista 8 e imprimimos tres piezas de información sobre cada una de las presentaciones principales: el título, un enlace

a la página de discusión y el número de comentarios que tiene actualmente el envío:

Código de estado: 200

identificación: 31390506 estado: 200

identificación: estado: 200

31389893 estado: 200

identificación: 31390742 --snip--

Título: Fly.io: El recuperador de la magia de Heroku

Enlace de discusión: <https://news.ycombinator.com/item?id=31390506>

Comentarios: 134

Título: La extraña opción de Hewlett Packard FreeDOS

Enlace de discusión: <https://news.ycombinator.com/item?id=31389893>

Comentarios: 64

Título: Tutorial de JavaScript moderno

Enlace de discusión: <https://news.ycombinator.com/item?id=31390742>

Comentarios: 20

--recorte--

Utilizaría un proceso similar para acceder y analizar información con cualquier API. Con estos datos, podría hacer una visualización que muestre qué envíos han inspirado las discusiones recientes más activas. Esta es también la base de aplicaciones que brindan una experiencia de lectura personalizada para sitios como Hacker News. Para obtener más información sobre a qué tipo de información puede acceder a través de la API de Hacker News, visite la página de documentación en <https://github.com/HackerNews/API>.

NOTA Hacker News a veces permite a las empresas a las que apoya realizar puestos de contratación especiales, y los comentarios están deshabilitados en estas publicaciones. Si ejecuta este programa mientras una de estas publicaciones está presente, obtendrá un `KeyError`. Si esto causa un problema, puede incluir el código que crea `submit_dict` en un bloque `try-except` y omitir estas publicaciones.

INTÉNTALO TÚ MISMO

17-1. Otros idiomas: modifique la llamada API en `python_repos.py` para que genere un gráfico que muestre los proyectos más populares en otros idiomas. Pruebe lenguajes como JavaScript, Ruby, C, Java, Perl, Haskell y Go.

17-2. Discusiones activas: utilizando los datos de `hn_submissions.py`, cree un gráfico de barras que muestre las discusiones más activas que tienen lugar actualmente en Hacker News. La altura de cada barra debe corresponder a la cantidad de comentarios que tiene cada envío. La etiqueta de cada barra debe incluir el título del envío y actuar como un enlace a la página de discusión de ese envío. Si obtiene un `KeyError` al crear un gráfico, use un bloque `try-except` para omitir las publicaciones promocionales.

(continuado)

17-3. Prueba de `python_repos.py`: en `python_repos.py`, imprimimos el valor de `status_code` para asegurarnos de que la llamada a la API fuera exitosa. Escriba un programa llamado `test_python_repos.py` que use `pytest` para afirmar que el valor de `status_code` es 200. Descubra algunas otras afirmaciones que puede hacer: por ejemplo, que se espera el número de elementos devueltos y que el número total de repositorios es mayor que una determinada cantidad.

17-4. Exploración adicional: visite la documentación de Plotly y la API de GitHub o la API de Hacker News. Utilice parte de la información que encuentre allí para personalizar el estilo de los gráficos que ya hemos creado o extraer información diferente y crear sus propias visualizaciones. Si tiene curiosidad por explorar otras API, eche un vistazo a las API mencionadas en el repositorio de GitHub en <https://github.com/public-apis>.

Resumen

En este capítulo, aprendió a usar API para escribir programas autónomos que recopilan automáticamente los datos que necesitan y los usan para crear una visualización. Utilizó la API de GitHub para explorar los proyectos de Python más destacados en GitHub y también analizó brevemente la API de Hacker News. Aprendiste cómo usar el paquete `Requests` para emitir automáticamente una llamada API y cómo procesar los resultados de esa llamada. También introdujimos algunas configuraciones de Plotly que personalizan aún más la apariencia de los gráficos que genera.

En el siguiente capítulo, utilizará Django para crear una aplicación web como su proyecto final.

18

COMENZAR CON DJANGO



A medida que Internet ha evolucionado, la línea entre los sitios web y las aplicaciones móviles se ha desdibujado.

Tanto los sitios web como las aplicaciones ayudan a los usuarios a interactuar con los datos de diversas formas.

Afortunadamente, puedes usar Django para crear un único proyecto que sirva tanto para un sitio web dinámico como para un conjunto de aplicaciones móviles. Django es el marco web más popular de Python , un conjunto de herramientas diseñadas para crear aplicaciones web interactivas. En este capítulo, aprenderá cómo usar Django para crear un proyecto llamado Learning Log, un sistema de diario en línea que le permite realizar un seguimiento de la información que ha aprendido sobre diferentes temas.

Escribiremos una especificación para este proyecto y luego definiremos modelos para los datos con los que funcionará la aplicación. Usaremos el sistema de administración de Django para ingresar algunos datos iniciales y luego escribiremos vistas y plantillas para que Django pueda crear las páginas del sitio.

Django puede responder a solicitudes de páginas y facilitar la lectura y escritura en una base de datos, administrar usuarios y mucho más. En los capítulos 19 y 20,

perfeccionará el proyecto de Registro de aprendizaje y luego lo implementará en un servidor activo para que usted (y todos los demás en el mundo) puedan usarlo.

Configurar un proyecto

Al comenzar a trabajar en algo tan importante como una aplicación web, primero debe describir los objetivos del proyecto en una especificación. Una vez que tenga un conjunto claro de objetivos, puede comenzar a identificar tareas manejables para lograr esos objetivos.

En esta sección, escribiremos una especificación para el Registro de aprendizaje y comenzaremos a trabajar en la primera fase del proyecto. Esto implicará configurar un entorno virtual y desarrollar los aspectos iniciales de un proyecto Django.

Escribir una especificación

Una especificación completa detalla los objetivos del proyecto, describe la funcionalidad del proyecto y analiza su apariencia e interfaz de usuario. Como cualquier buen proyecto o plan de negocios, una especificación debe mantenerse enfocado y ayudarte a mantener tu proyecto encaminado. No escribiremos aquí una especificación completa del proyecto, pero estableceremos algunos objetivos claros para mantener enfocado el proceso de desarrollo. Aquí está la especificación que usaremos:

Escribiremos una aplicación web llamada Learning Log que permita a los usuarios registrar los temas que les interesan y realizar anotaciones en un diario a medida que aprenden sobre cada tema. La página de inicio de Learning Log describirá el sitio e invitará a los usuarios a registrarse o iniciar sesión. Una vez que haya iniciado sesión, un usuario puede crear nuevos temas, agregar nuevas entradas y leer y editar entradas existentes.

Cuando investigas un tema nuevo, llevar un diario de lo que has aprendido puede ayudarte a realizar un seguimiento de la información nueva y de la que ya has encontrado. Esto es especialmente cierto cuando se estudian materias técnicas. Una buena aplicación, como la que crearemos, puede ayudar a que este proceso sea más eficiente.

Creando un entorno virtual

Para trabajar con Django, primero configuraremos un entorno virtual. Un entorno virtual es un lugar en su sistema donde puede instalar paquetes y aislarlos de todos los demás paquetes de Python. Separar las bibliotecas de un proyecto de otros proyectos es beneficioso y será necesario cuando implementemos Learning Log en un servidor en el Capítulo 20.

Cree un nuevo directorio para su proyecto llamado `learning_log`, cambie a ese directorio en una terminal e ingrese el siguiente código para crear un entorno virtual:

```
log_aprendizaje$ python -m venv ll_env
registro_aprendizaje$
```

Aquí estamos ejecutando el módulo de entorno virtual `venv` y usándolo para crear un entorno llamado `ll_env` (tenga en cuenta que este nombre comienza con

dos L minúsculas , no dos unidades). Si usa un comando como python3 Cuando ejecute programas o instale paquetes, asegúrese de utilizar ese comando aquí.

Activando el entorno virtual

Ahora necesitamos activar el entorno virtual, usando el siguiente comando:

```
learning_log$ fuente ll_env/bin/activate
(ll_env)learning_log$
```

Este comando ejecuta el script enable en ll_env/bin/. Cuando el entorno esté activo, verá el nombre del entorno entre paréntesis. Esto indica que puede instalar nuevos paquetes en el entorno y utilizar paquetes que ya se han instalado. Los paquetes que instale en ll_env no estarán disponibles cuando el entorno esté inactivo.

NOTA Si está utilizando Windows, utilice el comando ll_env\Scripts\activate (sin la palabra fuente) para activar el entorno virtual. Si está utilizando PowerShell, es posible que deba escribir Activar en mayúscula.

Para dejar de utilizar un entorno virtual, ingrese desactivar:

```
(ll_env)learning_log$ desactivar
registro_aprendizaje$
```

El entorno también quedará inactivo cuando cierre la terminal en la que se está ejecutando.

Instalando Django

Con el entorno virtual activado, ingresa lo siguiente para actualizar pip e instalar Django:

```
(ll_env)learning_log$ pip instalar --actualizar pip
(ll_env)learning_log$ pip instalar django
Coleccionando Django
--recorte--
Instalación de paquetes recopilados: sqlparse, asgiref, django
Instalado exitosamente asgiref-3.5.2 django-4.1 sqlparse-0.4.2
(ll_env)learning_log$
```

Debido a que descarga recursos de una variedad de fuentes, pip se actualiza con bastante frecuencia. Es una buena idea actualizar pip cada vez que crea un nuevo entorno virtual.

Estamos trabajando en un entorno virtual ahora, por lo que el comando para instalar Django es el mismo en todos los sistemas. No es necesario utilizar comandos más largos, como python -m pip install nombre_paquete, ni incluir el comando --user bandera. Tenga en cuenta que Django estará disponible sólo cuando el entorno ll_env esté activo.

NOTA Django lanza una nueva versión aproximadamente cada ocho meses, por lo que es posible que vea una versión más nueva cuando instale Django. Lo más probable es que este proyecto funcione tal como está escrito aquí, incluso en versiones más nuevas de Django. Si desea asegurarse de utilizar la misma versión de Django que ve aquí, utilice el comando `pip install django==4.1.*`. Esto instalará la última versión de Django 4.1. Si tiene algún problema relacionado con la versión que está utilizando, consulte los recursos en línea de este libro en https://ehmatthes.github.io/pcc_3e.

Creando un proyecto en Django

Sin salir del entorno virtual activo (recuerda buscar `ll_env` entre paréntesis en el símbolo del terminal), ingrese los siguientes comandos para crear un nuevo proyecto:

```
1 (ll_env)learning_log$ django-admin startproject ll_project.
2 (ll_env)learning_log$ ls
  ll_env ll_project administrar.py
3 (ll_env)learning_log$ ls ll_project
  __init__.py asgi.py configuraciones.py urls.py wsgi.py
```

El comando `startproject` 1 le dice a Django que configure un nuevo proyecto llamado `ll_project`. El punto (.) al final del comando crea el nuevo proyecto con una estructura de directorios que facilitará la implementación de la aplicación en un servidor cuando hayamos terminado de desarrollarla.

NOTA No olvide este punto o podría encontrarse con algunos problemas de configuración al implementar la aplicación. Si olvida el punto, elimine los archivos y carpetas que se crearon (excepto `ll_env`) y ejecute el comando nuevamente.

Al ejecutar el comando `ls` (dir en Windows) 2 se muestra que Django ha creado un nuevo directorio llamado `ll_project`. También creó un archivo `Manage.py`, que es un programa corto que recibe comandos y los envía a la parte relevante de Django. Usaremos estos comandos para administrar tareas, como trabajar con bases de datos y ejecutar servidores.

El directorio `ll_project` contiene cuatro archivos 3; los más importantes son `settings.py`, `urls.py` y `wsgi.py`. El archivo `settings.py` controla cómo Django interactúa con su sistema y administra su proyecto. Modificaremos algunas de estas configuraciones y agregaremos algunas configuraciones propias a medida que el proyecto evolucione. Las `URL.py` El archivo le dice a Django qué páginas construir en respuesta a las solicitudes del navegador. El archivo `wsgi.py` ayuda a Django a servir los archivos que crea. El nombre del archivo es un acrónimo de "interfaz de puerta de enlace del servidor web".

Creando la base de datos

Django almacena la mayor parte de la información de un proyecto en una base de datos, por lo que a continuación necesitamos crear una base de datos con la que Django pueda trabajar. Ingrese el siguiente comando (aún en un entorno activo):

```
(ll_env)learning_log$ python administrar.py migrar
1 Operaciones a realizar:
```

```

Aplicar todas las migraciones: administrador, autenticación, tipos de contenido, sesiones
Ejecutando migraciones:
  Aplicando contenttypes.0001_initial... OK
  Aplicando auth.0001_initial... OK
  --recorte--
  Aplicando sesiones.0001_initial... OK
2 (ll_env)learning_log$ ls
db.sqlite3 ll_env ll_project administrar.py

```

Cada vez que modificamos una base de datos, decimos que estamos migrando la base de datos. Emitir el comando de migración por primera vez le dice a Django que se asegure de que la base de datos coincida con el estado actual del proyecto. La primera vez que ejecutamos este comando en un nuevo proyecto usando SQLite (más sobre SQLite en un momento), Django creará una nueva base de datos para nosotros. Aquí, Django informa que preparará la base de datos para almacenar la información que necesita para manejar las tareas administrativas y de autenticación 1.

Al ejecutar el comando `ls` se muestra que Django creó otro archivo llamado `db.sqlite3`. SQLite es una base de datos que se ejecuta en un único archivo; Es ideal para escribir aplicaciones sencillas porque no tendrás que prestar mucha atención a la gestión de la base de datos.

NOTA En un entorno virtual activo, utilice el comando `python` para ejecutar el comando `Manage.py`. Comandos, incluso si usa algo diferente, como `python3`, para ejecutar otros programas. En un entorno virtual, el comando `python` se refiere a la versión de Python que se utilizó para crear el entorno virtual.

Ver el proyecto

Asegurémonos de que Django haya configurado el proyecto correctamente. Ingrese el comando `runserver` para ver el proyecto en su estado actual:

```

(ll_env)learning_log$ python administrar.py servidor de ejecución
Observando cambios de archivos con StatReloader
Realizando comprobaciones del sistema...

```

- 1 La verificación del sistema no identificó problemas (0 silenciado).
19 de mayo de 2022 - 21:52:35
 - 2 Django versión 4.1, usando la configuración 'll_project.settings'
 - 3 Iniciando el servidor de desarrollo en `http://127.0.0.1:8000/`
Salga del servidor con `CONTROL-C`.
-

Django debería iniciar un servidor llamado servidor de desarrollo, para que puedas ver el proyecto en tu sistema y ver qué tan bien funciona. Cuando solicita una página ingresando una URL en un navegador, el servidor Django responde a esa solicitud creando la página adecuada y enviándola al navegador.

Django primero verifica que el proyecto esté configurado correctamente 1; luego informa la versión de Django en uso y el nombre del archivo de configuración en uso 2. Finalmente, informa la URL donde se sirve el proyecto 3.

La URL `http://127.0.0.1:8000/` indica que el proyecto está escuchando solicitudes en el puerto 8000 de su computadora, que se denomina `localhost`. El

el término localhost se refiere a un servidor que solo procesa solicitudes en su sistema; no permite que nadie más vea las páginas que estás desarrollando.

Abra un navegador web e ingrese la URL `http://localhost:8000/` o `http://127.0.0.1:8000/` si el primero no funciona. Deberías ver algo como la Figura 18-1: una página que Django crea para hacerte saber que todo está funcionando correctamente hasta el momento. Mantenga el servidor en ejecución por ahora, pero cuando desee detenerlo, presione CTRL-C en la terminal donde se emitió el comando `runserver`.

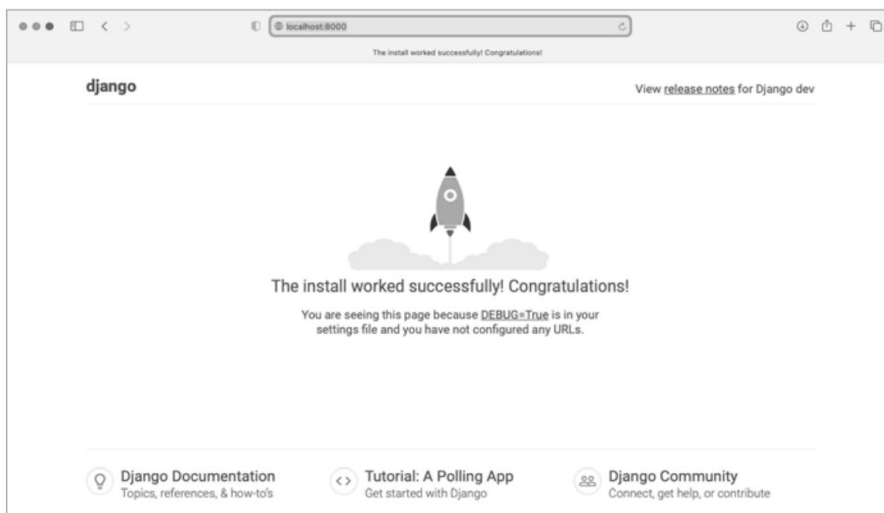


Figura 18-1: Todo está funcionando hasta ahora.

NOTA Si recibe el mensaje de error "Ese puerto ya está en uso", dígame a Django que use un puerto diferente ingresando `python Manage.py RunServer 8001` y luego pasando por números más altos hasta encontrar un puerto abierto.

INTÉNTALO TÚ MISMO

18-1. Nuevos proyectos: para tener una mejor idea de lo que hace Django, cree un par de proyectos vacíos y observe lo que crea Django. Cree una nueva carpeta con un nombre simple, como `tik_gram` o `insta_tok` (fuera de su directorio `learning_log`), navegue hasta esa carpeta en una terminal y cree un entorno virtual. Instale Django y ejecute el comando `django-admin.py startproject tg_project`.

(asegurándose de incluir el punto al final del comando).

Mire los archivos y carpetas que crea este comando y compárelos con el Registro de aprendizaje. Haga esto varias veces, hasta que esté familiarizado con lo que crea Django al iniciar un nuevo proyecto. Luego elimine los directorios del proyecto si lo desea.

Iniciar una aplicación

Un proyecto Django está organizado como un grupo de aplicaciones individuales que trabajan juntas para hacer que el proyecto funcione como un todo. Por ahora, crearemos una aplicación para realizar la mayor parte del trabajo de nuestro proyecto. Agregaremos otra aplicación en el Capítulo 19 para administrar cuentas de usuario.

Debe dejar el servidor de desarrollo ejecutándose en la ventana de terminal que abrió anteriormente. Abra una nueva ventana (o pestaña) de terminal y navegue hasta el directorio que contiene `Manage.py`. Active el entorno virtual y luego ejecute el comando `startapp` :

```
learning_log$ fuente ll_env/bin/activate
(ll_env)learning_log$ python administrar.py startapp learning_logs
1 (ll_env)learning_log$ ls
db.sqlite3 learning_logs ll_env ll_project administrar.py
2 (ll_env)learning_log$ ls aprendizaje_logs/
__init__.py admin.py apps.py migraciones modelos.py pruebas.py vistas.py
```

El comando `startapp` `apname` le dice a Django que cree la infraestructura necesaria para construir una aplicación. Cuando mires en el directorio del proyecto ahora, verás una nueva carpeta llamada `learning_logs` 1. Usa el comando `ls` para ver qué ha creado Django 2. Los archivos más importantes son `models.py`, `admin.py` y `views.py`. Usaremos `models.py` para definir los datos que queremos administrar en nuestra aplicación. Veremos `admin.py` y `views.py` un poco más adelante.

Definición de modelos

Pensemos en nuestros datos por un momento. Cada usuario deberá crear una serie de temas en su registro de aprendizaje. Cada entrada que realicen estará vinculada a un tema y estas entradas se mostrarán como texto. También necesitaremos almacenar la marca de tiempo de cada entrada para poder mostrar a los usuarios cuándo hicieron cada una.

Abra el archivo `models.py` y observe su contenido existente:

`models.py` de `django.db` importar `modelos`

Crea tus modelos aquí.

Se está importando un módulo llamado `modelos` y se nos invita a crear nuestros propios modelos. Un modelo le dice a Django cómo trabajar con los datos que se almacenarán en la aplicación. Un modelo es una clase; tiene atributos y métodos, como todas las clases que hemos analizado. Este es el modelo de los temas que almacenarán los usuarios:

desde `django.db` importar `modelos`

Tema de clase (`modelos.Modelo`):

"""Un tema sobre el que el usuario está aprendiendo."""

1 `texto = modelos.CharField(max_length=200)`

2 `date_added = modelos.DateTimeField(auto_now_add=True)`

```
3 def __str__(yo):
    """Devuelve una representación de cadena del modelo."""
    volver auto.texto
```

Hemos creado una clase llamada Tema, que hereda de Model, una clase principal incluida en Django que define la funcionalidad básica de un modelo. Agregamos dos atributos a la clase Tema : texto y fecha_añadida.

El atributo de texto es un CharField, un dato que se compone de caracteres o texto 1. Utiliza CharField cuando desea almacenar una pequeña cantidad de texto, como un nombre, un título o una ciudad. Cuando definimos un atributo CharField , tenemos que decirle a Django cuánto espacio debe reservar en la base de datos.

Aquí le damos una longitud máxima de 200 caracteres, que debería ser suficiente para contener la mayoría de los nombres de temas.

El atributo date_added es un DateTimeField, un dato que registrará una fecha y hora 2. Pasamos el argumento auto_now_add=True, que le dice a Django que establezca automáticamente este atributo en la fecha y hora actuales cada vez que el usuario cree un nuevo tema.

Es una buena idea decirle a Django cómo desea que represente una instancia de un modelo. Si un modelo tiene un método __str__() , Django llama a ese método cada vez que necesita generar una salida que haga referencia a una instancia de ese modelo.

Aquí hemos escrito un método __str__() que devuelve el valor asignado al atributo de texto 3.

Para ver los diferentes tipos de campos que puede usar en un modelo, consulte la página "Referencia de campos del modelo" en <https://docs.djangoproject.com/en/4.1/ref/models/fields/>. No necesitarás toda la información en este momento, pero será extremadamente útil cuando estés desarrollando tus propios proyectos Django.

Activando modelos

Para usar nuestros modelos, tenemos que decirle a Django que incluya nuestra aplicación en el proyecto general. Abra settings.py (en el directorio ll_project); Verás una sección que le dice a Django qué aplicaciones están instaladas en el proyecto:

```
configuración.py --snip--
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.archivos estáticos',
]
--recorte--
```

Agregue nuestra aplicación a esta lista modificando INSTALLED_APPS para que se vea así:

```
--recorte--
INSTALLED_APPS = [
    # Mis aplicaciones.
    'registros_aprendizaje',
```

```
# Aplicaciones Django predeterminadas.
'django.contrib.admin',
--recorte--
]
--recorte--
```

Agrupar aplicaciones en un proyecto ayuda a realizar un seguimiento de ellas a medida que el proyecto crece para incluir más aplicaciones. Aquí iniciamos una sección llamada Mis aplicaciones, que por ahora incluye sólo 'learning_logs' . Es importante colocar sus propias aplicaciones antes que las aplicaciones predeterminadas, en caso de que necesite anular cualquier comportamiento de las aplicaciones predeterminadas con su propio comportamiento personalizado.

A continuación, debemos decirle a Django que modifique la base de datos para que pueda almacenar información relacionada con el tema del modelo. Desde la terminal, ejecute el siguiente comando:

```
(ll_env)learning_log$ python administrar.py makemigrations learning_logs
Migraciones para 'learning_logs':
  learning_logs/migrations/0001_initial.py
  - Crear tema modelo
(ll_env)learning_log$
```

El comando makemigrations le dice a Django que descubra cómo modificar la base de datos para que pueda almacenar los datos asociados con cualquier modelo nuevo que hayamos definido. El resultado aquí muestra que Django ha creado un archivo de migración llamado 0001_initial.py. Esta migración creará una tabla para el modelo. Tema en la base de datos.

Ahora aplicaremos esta migración y haremos que Django modifique la base de datos por nosotros:

```
(ll_env)learning_log$ python administrar.py migrar
Operaciones a realizar:
  Aplicar todas las migraciones: admin, auth, contenttypes, learning_logs, sesiones
Ejecutando migraciones:
  Aplicando learning_logs.0001_initial... OK
```

La mayor parte del resultado de este comando es idéntico al resultado de la primera vez que emitimos el comando de migración . Necesitamos verificar la última línea de este resultado, donde Django confirma que la migración de learning_logs funcionó bien.

Siempre que queramos modificar los datos que administra Learning Log, seguiremos estos tres pasos: modificar models.py, llamar a makemigrations en learning_logs y decirle a Django que migre el proyecto.

El sitio de administración de Django

Django facilita el trabajo con tus modelos a través de su sitio de administración. El sitio de administración de Django sólo está destinado a ser utilizado por los administradores del sitio; No está destinado a usuarios habituales. En esta sección, configuraremos el sitio de administración y lo usaremos para agregar algunos temas a través del modelo de temas .

Configurar un superusuario

Django te permite crear un superusuario, un usuario que tiene todos los privilegios disponibles en el sitio. Los privilegios de un usuario controlan las acciones que puede realizar. La configuración de privilegios más restrictiva permite al usuario leer únicamente información pública en el sitio. Los usuarios registrados suelen tener el privilegio de leer sus propios datos privados y cierta información seleccionada que está disponible sólo para los miembros. Para administrar eficazmente un proyecto, el propietario del sitio normalmente necesita acceso a toda la información almacenada en el sitio. Un buen administrador tiene cuidado con la información confidencial de sus usuarios, porque los usuarios confían mucho en las aplicaciones a las que acceden.

Para crear un superusuario en Django, ingrese el siguiente comando y responde a las indicaciones:

```
(ll_env)learning_log$ python administrar.py crearsuperusuario
```

1 Nombre de usuario (deje en blanco para usar 'eric'): ll_admin

2 Dirección de correo electrónico:

3 Contraseña:

Contraseña de nuevo):

Superusuario creado exitosamente.

```
(ll_env)learning_log$
```

Cuando ejecuta el comando `createsuperuser`, Django le solicita que ingrese un nombre de usuario para el superusuario 1. Aquí estoy usando `ll_admin`, pero puede ingresar cualquier nombre de usuario que desee. Puede ingresar una dirección de correo electrónico o simplemente dejar este campo en blanco 2. Deberá ingresar su contraseña dos veces 3.

NOTA Parte de la información confidencial se puede ocultar a los administradores de un sitio. Por ejemplo, Django no almacena la contraseña que ingresas; en su lugar, almacena una cadena derivada de la contraseña, llamada hash. Cada vez que ingresa su contraseña, Django codifica su entrada y la compara con el hash almacenado. Si los dos hash coinciden, estás autenticado. Al exigir que los hashes coincidan, Django garantiza que si un atacante obtiene acceso a la base de datos de un sitio, podrá leer los hashes almacenados pero no las contraseñas. Cuando un sitio está configurado correctamente, es casi imposible obtener las contraseñas originales de los hashes.

Registrar un modelo en el sitio de administración

Django incluye algunos modelos en el sitio de administración automáticamente, como Usuario y Grupo, pero los modelos que creamos deben agregarse manualmente.

Cuando iniciamos la aplicación `learning_logs`, Django creó un archivo `admin.py` en el mismo directorio que `models.py`. Abra el archivo `admin.py`:

```
admin.py de django.contrib importar administrador
```

```
# Registre sus modelos aquí.
```

Para registrar el tema en el sitio de administración, ingrese lo siguiente:

```
desde django.contrib administrador de importación
```

```
desde .models importar tema
```

```
admin.site.register(Tema)
```

Este código primero importa el modelo que queremos registrar, Tema. El punto delante de los modelos le dice a Django que busque models.py en el mismo directorio que admin.py. El código admin.site.register() le dice a Django que administre nuestro modelo a través del sitio de administración.

Ahora use la cuenta de superusuario para acceder al sitio de administración. Ir a <http://localhost:8000/admin/> e ingrese el nombre de usuario y la contraseña del superusuario que acaba de crear. Debería ver una pantalla similar a la que se muestra en la Figura 18-2. Esta página le permite agregar nuevos usuarios y grupos, y cambiar los existentes. También puedes trabajar con datos relacionados con el modelo de tema que acabamos de definir.

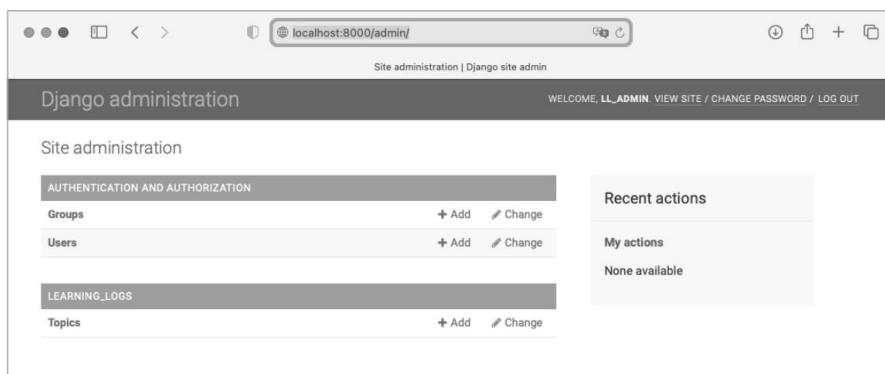


Figura 18-2: El sitio de administración con el tema incluido

NOTA ve un mensaje en su navegador que indica que la página web no está disponible, asegúrese de tener todavía el servidor Django ejecutándose en una ventana de terminal. Si no lo hace, active un entorno virtual y vuelva a emitir el comando `python Manage.py RunServer`. Si tiene problemas para ver su proyecto en cualquier punto del proceso de desarrollo, cerrar cualquier terminal abierta y volver a emitir el comando de ejecución es un buen primer paso para solucionar el problema.

Agregar temas

Ahora que el tema se ha registrado en el sitio de administración, agreguemos nuestro primer tema. Haga clic en Temas para ir a la página Temas, que está prácticamente vacía porque todavía no tenemos temas que administrar. Haga clic en Agregar tema y aparecerá un formulario para agregar un nuevo tema. Ingrese Chess en el primer cuadro y haga clic en Guardar. Se le enviará de regreso a la página de administración de Temas y verá el tema que acaba de crear.

Creemos un segundo tema para tener más datos con los que trabajar.

Haga clic en Agregar tema nuevamente e ingrese Escalada en roca. Haga clic en Guardar y volverá a la página principal de Temas. Ahora verá Ajedrez y Escalada en roca en la lista.

Definición del modelo de entrada

Para que un usuario registre lo que ha estado aprendiendo sobre ajedrez y escalada en roca, necesitamos definir un modelo para los tipos de entradas que los usuarios pueden realizar en sus registros de aprendizaje. Cada entrada debe estar asociada con un tema en particular. Esta relación se denomina relación de muchos a uno, lo que significa que se pueden asociar muchas entradas con un tema.

Aquí está el código para el modelo Entry . Colóquelo en su archivo models.py :

models.py de django.db importar modelos

Tema de clase (modelos.Modelo):

--recorte--

Entrada de 1 clase (modelos.Modelo):

"""Algo específico aprendido sobre un tema."""

2 tema = modelos.ForeignKey (Tema, on_delete = modelos.CASCADE)

3 texto = modelos.TextField()

date_added = modelos.DateTimeField(auto_now_add=True)

Meta de 4 clases:

verbose_name_plural = 'entradas'

def __str__(yo):

"""Devuelve una cadena simple que representa la entrada."""

5 devolver f"{self.text[:50]}..."

La clase Entry hereda de la clase Model base de Django , al igual que Topic hizo 1. El primer atributo, tema, es una instancia de ForeignKey 2. Una clave externa es un término de base de datos; es una referencia a otro registro en la base de datos. Este es el código que conecta cada entrada con un tema específico. A cada tema se le asigna una clave o ID cuando se crea. Cuando Django necesita establecer una conexión entre dos datos, utiliza las claves asociadas con cada dato. Usaremos estas conexiones en breve para recuperar todas las entradas asociadas con un tema determinado. El argumento on_delete=models.CASCADE le dice a Django que cuando se elimina un tema, todas las entradas asociadas con ese tema también deben eliminarse. Esto se conoce como eliminación en cascada.

El siguiente es un atributo llamado texto, que es una instancia de TextField 3. Este tipo de campo no necesita un límite de tamaño porque no queremos limitar el tamaño de las entradas individuales. El atributo date_added nos permite presentar las entradas en el orden en que fueron creadas y colocar una marca de tiempo al lado de cada entrada.

La clase Meta está anidada dentro de la clase Entry 4. La clase Meta contiene información adicional para gestionar un modelo; aquí, nos permite configurar un atributo especial que le indica a Django que use Entradas cuando necesite hacer referencia a más de una entrada. Sin esto, Django se referiría a múltiples entradas como Entradas.

El método __str__() le dice a Django qué información mostrar cuando se refiere a entradas individuales. Debido a que una entrada puede ser un cuerpo de texto largo, __str__() devuelve solo los primeros 50 caracteres del texto 5. También agregamos puntos suspensivos para aclarar que no siempre mostramos la entrada completa.

Migrar el modelo de entrada

Debido a que agregamos un nuevo modelo, necesitamos migrar la base de datos nuevamente. Este proceso le resultará bastante familiar: modifica `models.py`, ejecuta el comando `python Manage.py makemigrations nombre_aplicación` y luego ejecuta el comando `python Manage.py migrar`.

Migre la base de datos y verifique el resultado ingresando lo siguiente comandos:

```
(ll_env)learning_log$ python administrar.py makemigrations learning_logs
Migraciones para 'learning_logs':
1 learning_logs/migrations/0002_entry.py
  - Crear entrada de modelo
(ll_env)learning_log$ python administrar.py migrar
Operaciones a realizar:
--recorte--
2 Aplicando learning_logs.0002_entry... OK
```

Se genera una nueva migración llamada `0002_entry.py`, que le dice a Django cómo modificar la base de datos para almacenar información relacionada con la Entrada 1 del modelo. Cuando emitimos el comando de migración, vemos que Django aplicó esta migración y todo funcionó correctamente 2.

Registro de entrada en el sitio de administración

También necesitamos registrar el modelo `Entry`. Así es como debería verse `admin.py` ahora:

`admin.py` de `django.contrib` importar `administrador`

desde `.models` importar `tema, entrada`

```
admin.site.register(Tema)
admin.site.register(Entrada)
```

Vuelva a `http://localhost/admin/` y debería ver las Entradas enumeradas en `Learning_Logs`. Haga clic en el enlace `Agregar para Entradas`, o haga clic en `Entradas` y luego elija `Agregar entrada`. Debería ver una lista desplegable para seleccionar el tema para el que está creando una entrada y un cuadro de texto para agregar una entrada. Seleccionar `ajedrez` de la lista desplegable y agregue una entrada. Aquí está la primera entrada que hice:

La apertura es la primera parte del juego, aproximadamente los primeros diez movimientos. Al principio, es una buena idea hacer tres cosas: saca tus alfiles y caballos, intenta controlar el centro del tablero y enroca a tu rey.

Por supuesto, estas son sólo pautas. Será importante saber cuándo seguir estas pautas y cuándo ignorar estas sugerencias.

Al hacer clic en `Guardar`, volverá a la página de administración principal para realizar entradas. Aquí verá el beneficio de usar `texto[:50]` como representación de cadena para cada entrada; es mucho más fácil trabajar con múltiples entradas en

la interfaz de administración si ve solo la primera parte de una entrada, en lugar del texto completo de cada entrada.

Haga una segunda entrada para Ajedrez y una entrada para Escalada en roca para que podamos tener algunos datos iniciales. Aquí hay una segunda entrada para Ajedrez:

En la fase inicial del juego, es importante sacar a relucir tus alfiles y caballos. Estas piezas son lo suficientemente potentes y maniobrables como para desempeñar un papel importante en los movimientos iniciales de una partida.

Y aquí tenéis una primera entrada para Escalada en Roca:

Uno de los conceptos más importantes en la escalada es mantener el peso en los pies tanto como sea posible. Existe el mito de que los escaladores pueden colgarse de los brazos todo el día. En realidad, los buenos escaladores han practicado formas específicas de mantener el peso sobre los pies siempre que sea posible.

Estas tres entradas nos darán algo con qué trabajar a medida que continuamos para desarrollar el Registro de aprendizaje.

El caparazón de Django

Ahora que hemos ingresado algunos datos, podemos examinarlos mediante programación a través de una sesión de terminal interactiva. Este entorno interactivo se llama Django Shell y es un excelente entorno para probar y solucionar problemas de su proyecto. A continuación se muestra un ejemplo de una sesión de shell interactiva:

```
(ll_env)learning_log$ python administrar.py shell
1 >>> de learning_logs.models importar tema
>>> Tema.objetos.todos()
<QuerySet [<Tema: Ajedrez>, <Tema: Escalada en roca>]>
```

El comando `python enable.py shell`, ejecutado en un entorno virtual activo, inicia un intérprete de Python que puede utilizar para explorar los datos almacenados en la base de datos de su proyecto. Aquí, importamos el modelo `Topic` del módulo `1 learning_logs.models`. Luego usamos el método `Topic.objects`.

`.all()` para obtener todas las instancias del tema del modelo; la lista que se devuelve se llama conjunto de consultas.

Podemos recorrer un conjunto de consultas tal como lo haríamos con una lista. Así es como Puede ver el ID que se ha asignado a cada objeto de tema:

```
>>> temas = Tema.objetos.todos()
>>> para tema en temas:
...     imprimir(tema.id, tema)
...
1 ajedrez
2 escalada en roca
```

Asignamos el conjunto de consultas a los temas y luego imprimimos el atributo de identificación de cada tema y la representación de cadena de cada tema. Podemos ver que el Ajedrez tiene un ID de 1 y la Escalada en Roca tiene un ID de 2.

Si conoce el ID de un objeto en particular, puede utilizar el método `Tema.objects.get()` para recuperar ese objeto y examinar cualquier atributo que tenga. Veamos el texto y los valores `date_added` de Chess:

```
>>> t = Tema.objects.get(id=1)
>>> t.texto
'Ajedrez'
>>> t.fecha_añadida
fecha y hora. fecha y hora (2022, 5, 20, 3, 33, 36, 928759,
tzinfo=fechahora.zonahoraria.utc)
```

También podemos mirar las entradas relacionadas con un tema determinado. Anteriormente, definimos el atributo de tema para el modelo de entrada. Esta era una `ForeignKey`, una conexión entre cada entrada y un tema. Django puede usar esta conexión para obtener cada entrada relacionada con un tema determinado, como este:

```
1 >>> t.entry_set.all()
<QuerySet [<Entrada: La apertura es la primera parte del juego, más o menos...>, <Entrada:
En la fase inicial del juego, es importante...>]>
```

Para obtener datos a través de una relación de clave externa, se utiliza el nombre en minúscula del modelo relacionado seguido de un guión bajo y la palabra conjunto 1. Por ejemplo, digamos que tiene los modelos `Pizza` y `Topping`, y `Topping` está relacionado con `Pizza` a través de una clave externa. Si su objeto se llama `my_pizza` y representa una sola pizza, puede obtener todos los ingredientes de la pizza usando el código `my_pizza.topping_set.all()`.

Usaremos esta sintaxis cuando comencemos a codificar las páginas que los usuarios pueden solicitar. El shell es realmente útil para asegurarse de que su código recupere los datos que desea. Si su código funciona como espera en el shell, también debería funcionar correctamente en los archivos dentro de su proyecto. Si su código genera errores o no recupera los datos que espera, es mucho más fácil solucionar problemas de su código en el entorno de shell simple que dentro de los archivos que generan páginas web. No nos referiremos mucho al shell, pero debería continuar usándolo para practicar cómo trabajar con la sintaxis de Django para acceder a los datos almacenados en el proyecto.

Cada vez que modifique sus modelos, deberá reiniciar el shell para ver los efectos de esos cambios. Para salir de una sesión de shell, presione CTRL-D; en Windows, presione CTRL-Z y luego presione ENTRAR.

INTÉNTALO TÚ MISMO

18-2. Entradas cortas: El método `__str__()` en el modelo `Entry` actualmente agrega puntos suspensivos a cada instancia de `Entry` cuando Django la muestra en el sitio de administración o en el shell. Agregue una declaración `if` al método `__str__()` que agregue

puntos suspensivos (continuación)

sólo si la entrada tiene más de 50 caracteres. Utilice el sitio de administración para agregar una entrada que tenga menos de 50 caracteres y verifique que no tenga puntos suspensivos cuando se vea.

18-3. La API de Django: cuando escribes código para acceder a los datos de tu proyecto, estás escribiendo una consulta. Hojee la documentación para consultar sus datos en <https://docs.djangoproject.com/en/4.1/topics/db/queries>. Gran parte de lo que vea le parecerá nuevo, pero le resultará muy útil cuando empiece a trabajar en sus propios proyectos.

18-4. Pizzería: inicie un nuevo proyecto llamado `pizzeria_project` con una aplicación llamada `pizzas`. Defina un modelo de pizza con un campo llamado `nombre`, que contendrá valores de nombre, como `Hawaiian` y `Meat Lovers`. Defina un modelo llamado `Topping` con campos llamados `pizza` y `nombre`. El campo de `pizza` debe ser una clave externa para `Pizza` y el `nombre` debe poder contener valores como `piña`, `tocino canadiense` y `embutido`.

Registre ambos modelos en el sitio de administración y utilice el sitio para ingresar algunos nombres y aderezos de pizza. Utilice el shell para explorar los datos que ingresó.

Creación de páginas: la página de inicio del registro de aprendizaje

Crear páginas web con Django consta de tres etapas: definir URL, escribir vistas y escribir plantillas. Puedes hacerlo en cualquier orden, pero en este proyecto siempre comenzaremos definiendo el patrón de URL. Un patrón de URL

describe la forma en que se presenta la URL. También le dice a Django qué buscar cuando hace coincidir una solicitud del navegador con la URL de un sitio, para que sepa qué página regresar.

Luego, cada URL se asigna a una vista particular. La función de vista recupera y procesa los datos necesarios para esa página. La función de vista a menudo representa la página usando una plantilla, que contiene la estructura general de la página. Para ver cómo funciona esto, creemos la página de inicio del Registro de aprendizaje.

Definiremos la URL de la página de inicio, escribiremos su función de visualización y crearemos una plantilla simple.

Como solo queremos asegurarnos de que el Registro de aprendizaje funcione como se supone, crearemos una página simple por ahora. Es divertido diseñar una aplicación web que funcione cuando esté completa; una aplicación que se ve bien pero no funciona bien no tiene sentido. Por ahora, la página de inicio mostrará sólo un título y una breve descripción.

Mapeo de una URL

Los usuarios solicitan páginas ingresando URL en un navegador y haciendo clic en enlaces, por lo que tendremos que decidir qué URL se necesitan. La URL de la página de inicio es la primera: es la URL base que la gente usa para acceder al proyecto. Por el momento, la URL base, `http://localhost:8000/`, devuelve el sitio Django predeterminado que nos permite saber que el proyecto se configuró correctamente. Cambiaremos esto asignando la URL base a la página de inicio de Learning Log.

En la carpeta principal ll_project , abra el archivo urls.py. Deberías ver el siguiente código:

```
ll_proyecto/ 1 de django.contrib administrador de importación
URL.py      desde la ruta de importación django.urls

2 patrones de URL = [
3 ruta('admin/', admin.site.urls),
4 ]
```

Las dos primeras líneas importan el módulo de administración y una función para crear rutas URL 1. El cuerpo del archivo define la variable urlpatterns 2. En este urls.py archivo, que define las URL para el proyecto en su conjunto, la variable urlpatterns incluye conjuntos de URL de las aplicaciones del proyecto. La lista incluye el módulo admin.site.urls, que define todas las URL que se pueden solicitar desde el sitio de administración 3.

Necesitamos incluir las URL de learning_logs, así que agregue lo siguiente:

```
desde django.contrib administrador de importación
desde la ruta de importación django.urls, incluya

patrones de URL = [
    ruta('admin/', admin.site.urls),
    ruta("", incluir('learning_logs.urls')),
5 ]
```

Importamos la función include() y también agregamos una línea para incluir el módulo learning_logs.urls.

El URLs.py predeterminado está en la carpeta ll_project ; ahora necesitamos crear un segundo archivo urls.py en la carpeta learning_logs . Cree un nuevo archivo Python, guárdelo como urls.py en learning_logs e ingrese este código en él:

```
registros_de_aprendizaje/ 1 """Define patrones de URL para learning_logs."""
URL.py                    2 de la ruta de importación de django.urls

3 de                      . importar vistas

4 nombre_aplicación = 'registros_aprendizaje'
5 patrones de URL = [
    # Página de inicio
6 ruta("", vistas.index, nombre='índice'),
7 ]
```

Para dejar claro en qué urls.py estamos trabajando, agregamos una cadena de documentación en el comienzo del archivo 1. Luego importamos la función de ruta , que es necesaria al asignar URL a las vistas 2. También importamos el módulo de vistas 3; el punto le dice a Python que importe el módulo views.py desde el mismo directorio que el módulo urls.py actual . La variable app_name ayuda a Django a distinguir este archivo urls.py de archivos con el mismo nombre en otras aplicaciones dentro del proyecto 4. La variable urlpatterns en este módulo es una lista de páginas individuales que se pueden solicitar desde la aplicación learning_logs 5.

El patrón de URL real es una llamada a la función `path()`, que toma tres argumentos

6. El primer argumento es una cadena que ayuda a Django a enrutar la solicitud actual correctamente. Django recibe la URL solicitada e intenta enrutar la solicitud a una vista. Para ello, busca todos los patrones de URL que hemos definido para encontrar uno que coincida con la solicitud actual. Django ignora la URL base del proyecto (`http://localhost:8000/`), por lo que la cadena vacía (`"`) coincide con la URL base. Cualquier otra URL no coincidirá con este patrón y Django devolverá una página de error si la URL solicitada no coincide con ningún patrón de URL existente.

El segundo argumento en `path()` 6 especifica qué función llamar en `views.py`. Cuando una URL solicitada coincide con el patrón que estamos definiendo, Django llama a la función `index()` desde `views.py`. (Escribiremos esta función de vista en la siguiente sección). El tercer argumento proporciona el índice de nombres para este patrón de URL para que podamos hacer referencia a él más fácilmente en otros archivos a lo largo del proyecto. Siempre que queramos proporcionar un enlace a la página de inicio, usaremos este nombre en lugar de escribir una URL.

Escribir una vista

Una función de vista toma información de una solicitud, prepara los datos necesarios para generar una página y luego envía los datos de regreso al navegador. A menudo lo hace mediante el uso de una plantilla que define cómo se verá la página.

El archivo `views.py` en `learning_logs` se generó automáticamente cuando ejecutamos el comando `python Manage.py startapp`. Esto es lo que hay en `views.py` en este momento:

```
views.py de django.shortcuts importar render
```

```
# Crea tus vistas aquí.
```

Actualmente, este archivo solo importa la función `render()`, que genera la respuesta en función de los datos proporcionados por las vistas. Abra `views.py` y agregue el siguiente código para la página de inicio:

```
desde django.shortcuts importar render
```

```
índice def (solicitud):
```

```
    """La página de inicio del Registro de aprendizaje."""
    devolver render(solicitud, 'learning_logs/index.html')
```

Cuando una solicitud de URL coincide con el patrón que acabamos de definir, Django busca para una función llamada `index()` en el archivo `views.py`. Luego, Django pasa el objeto de solicitud a esta función de vista. En este caso, no necesitamos procesar ningún dato para la página, por lo que el único código en la función es una llamada a `render()`. La función `render()` aquí pasa dos argumentos: el objeto de solicitud original y una plantilla que puede usar para construir la página. Escribamos esta plantilla.

Escribir una plantilla

La plantilla define cómo debería verse la página y Django completa los datos relevantes cada vez que se solicita la página. Una plantilla le permite

acceder a cualquier dato proporcionado por la vista. Debido a que nuestra vista de la página de inicio no proporciona datos, esta plantilla es bastante simple.

Dentro de la carpeta `learning_logs`, cree una nueva carpeta llamada `plantillas`. Dentro de la carpeta de `plantillas`, cree otra carpeta llamada `learning_logs`. Esto puede parecer un poco redundante (tenemos una carpeta llamada `learning_logs` dentro de una carpeta llamada `templates` dentro de una carpeta llamada `learning_logs`), pero establece una estructura que Django puede interpretar sin ambigüedades, incluso en el contexto de un proyecto grande que contiene muchas aplicaciones individuales. Dentro de los registros de aprendizaje internos carpeta, cree un nuevo archivo llamado `index.html`. La ruta al archivo será `ll_project/learning_logs/templates/learning_logs/index.html`. Ingrese el siguiente código en ese archivo:

```
index.html <p>Registro de aprendizaje</p>
```

```
<p>Learning Log te ayuda a realizar un seguimiento de tu aprendizaje, para cualquier tema que estés interesado.</p>
```

Este es un archivo muy simple. Si no está familiarizado con HTML, el `<p></p>` Las etiquetas significan párrafos. La etiqueta `<p>` abre un párrafo y la etiqueta `</p>` lo cierra. Tenemos dos párrafos: el primero actúa como título y el segundo describe lo que los usuarios pueden hacer con Learning Log.

Ahora, cuando solicite la URL base del proyecto, `http://localhost:8000/`, debería ver la página que acabamos de crear en lugar de la página predeterminada de Django. Django tomará la URL solicitada y esa URL coincidirá con el patrón `''`; luego Django llamará a la función `views.index()`, que renderizará la página usando la plantilla contenida en `index.html`. La Figura 18-3 muestra la página resultante.

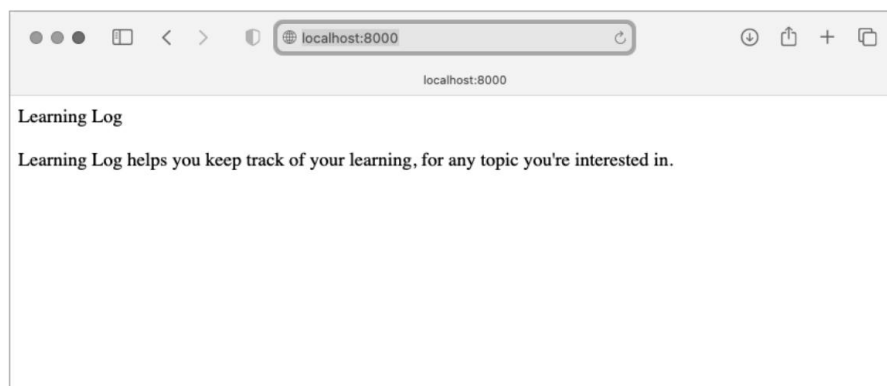


Figura 18-3: La página de inicio del Registro de aprendizaje

Aunque puede parecer un proceso complicado para crear una página, esta separación entre URL, vistas y plantillas funciona bastante bien. Le permite pensar en cada aspecto de un proyecto por separado. En proyectos más grandes, permite a las personas que trabajan en el proyecto centrarse en las áreas en las que son más fuertes. Por ejemplo, un especialista en bases de datos puede centrarse en los modelos, un programador puede centrarse en el código de vista y un especialista en frontend puede centrarse en las plantillas.

NOTA Es posible que vea el siguiente mensaje de error:

ModuleNotFoundError: ningún módulo llamado 'learning_logs.urls'

Si lo hace, detenga el servidor de desarrollo presionando CTRL-C en la ventana del terminal donde emitió el comando `runserver`. Luego vuelva a emitir el comando `python Manage.py RunServer`. Deberías poder ver la página de inicio. Cada vez que se encuentre con un error como este, intente detener y reiniciar el servidor.

INTÉNTALO TÚ MISMO

18-5. Planificador de comidas: considere una aplicación que ayude a las personas a planificar sus comidas durante la semana. Cree una nueva carpeta llamada `food_planner` e inicie un nuevo proyecto Django dentro de esta carpeta. Luego crea una nueva aplicación llamada `comida`. `_planes`. Haga una página de inicio sencilla para este proyecto.

18-6. Página de inicio de Pizzeria: agregue una página de inicio al proyecto de Pizzeria que inició en el Ejercicio 18-4 (página 388).

Construyendo páginas adicionales

Ahora que hemos establecido una rutina para crear una página, podemos comenzar a desarrollar el proyecto Registro de aprendizaje. Construiremos dos páginas que muestren datos: una página que enumere todos los temas y una página que muestre todas las entradas de un tema en particular. Para cada página, especificaremos un patrón de URL, escribiremos una función de vista y escribiremos una plantilla. Pero antes de hacer esto, crearemos una plantilla base de la que todas las plantillas del proyecto pueden heredar.

Herencia de plantilla

Al crear un sitio web, será necesario repetir algunos elementos en cada página. En lugar de escribir estos elementos directamente en cada página, puede escribir una plantilla base que contenga los elementos repetidos y luego hacer que cada página herede de la base. Este enfoque le permite concentrarse en desarrollar los aspectos únicos de cada página y hace que sea mucho más fácil cambiar la apariencia general del proyecto.

La plantilla principal

Crearemos una plantilla llamada `base.html` en el mismo directorio que `index.html`. Este archivo contendrá elementos comunes a todas las páginas; todas las demás plantillas heredarán de `base.html`. El único elemento que queremos repetir en cada página ahora es el título en la parte superior. Debido a que incluiremos esta plantilla en cada página, hagamos del título un enlace a la página de inicio:

`base.html` <p>

```
1 <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a>
```

</p>

2 {% contenido de bloque %}{% contenido de bloque final %}

La primera parte de este archivo crea un párrafo que contiene el nombre del proyecto, que también actúa como enlace a la página de inicio. Para generar un enlace, utilizamos una etiqueta de plantilla, que se indica mediante llaves y signos de porcentaje ({% %}). Una etiqueta de plantilla genera información que se mostrará en una página. La etiqueta de plantilla {% url 'learning_logs:index' %} que se muestra aquí genera una URL que coincide con el patrón de URL definido en `learning_logs/urls.py` con el nombre 'index' 1. En este ejemplo, `learning_logs` es el espacio de nombres y `index` es un patrón de URL con nombre en ese espacio de nombres. El espacio de nombres proviene del valor que asignamos a `app_name` en el archivo `learning_logs/urls.py`.

En una página HTML simple, un enlace está rodeado por la etiqueta de anclaje <a>:

 texto del enlace

Hacer que la etiqueta de plantilla genere la URL por nosotros hace que sea mucho más fácil mantener nuestros enlaces actualizados. Solo necesitamos cambiar el patrón de URL en `urls.py`, y Django insertará automáticamente la URL actualizada la próxima vez que se solicite la página. Cada página de nuestro proyecto heredará de `base.html`, por lo que de ahora en adelante, cada página tendrá un enlace a la página de inicio.

En la última línea, insertamos un par de etiquetas de bloque 2. Este bloque, llamado contenido, es un marcador de posición; la plantilla secundaria definirá el tipo de información que va en el bloque de contenido.

Una plantilla secundaria no tiene que definir todos los bloques de su principal, por lo que puede reservar espacio en las plantillas principales para tantos bloques como desee; la plantilla secundaria utiliza sólo tantos como necesita.

NOTA

el código Python, casi siempre usamos cuatro espacios cuando aplicamos sangría. Los archivos de plantilla tienden a tener más niveles de anidamiento que los archivos de Python, por lo que es común usar solo dos espacios para cada nivel de sangría.

La plantilla infantil

Ahora necesitamos reescribir `index.html` para heredar de `base.html`. Agregue el siguiente código a `index.html`:

`index.html` 1 {% extiende 'learning_logs/base.html' %}

2 {% bloquear contenido %}

```
<p>Learning Log te ayuda a realizar un seguimiento de tu aprendizaje, para cualquier tema que estés interesado.</p>
```

3 {% contenido del bloque final %}

Si compara esto con el `index.html` original, puede ver que hemos reemplazado el título del Registro de aprendizaje con el código para heredar de una plantilla principal 1. Una plantilla secundaria debe tener una etiqueta {% extends %} en la primera línea. para decirle a Django de qué plantilla principal heredar. El archivo `base.html`

es parte de `learning_logs`, por lo que incluimos `learning_logs` en la ruta a la plantilla principal. Esta línea extrae todo lo contenido en la plantilla `base.html` y permite que `index.html` defina lo que va en el espacio reservado por el contenido.

bloquear.

Definimos el bloque de contenido insertando una etiqueta `{% block %}` con el nombre contenido

2. Todo lo que no heredamos de la plantilla principal va dentro del bloque de contenido . Aquí, ese es el párrafo que describe el proyecto Registro de aprendizaje. Indicamos que hemos terminado de definir el contenido usando una etiqueta `{% endblock content %}` 3. La etiqueta `{% endblock %}` no requiere un nombre, pero si una plantilla crece hasta contener múltiples bloques, puede Será útil saber exactamente qué bloque está terminando.

Puede comenzar a ver los beneficios de la herencia de plantillas: en una plantilla secundaria, solo necesitamos incluir contenido que sea exclusivo de esa página. Esto no sólo simplifica cada plantilla, sino que también hace que sea mucho más fácil modificar el sitio. Para modificar un elemento común a muchas páginas, solo necesita modificar la plantilla principal. Luego, sus cambios se transfieren a cada página que hereda de esa plantilla. En un proyecto que incluye decenas o cientos de páginas, esta estructura puede hacer que sea mucho más fácil y rápido mejorar su sitio.

En un proyecto grande, es común tener una plantilla principal llamada `base.html` para todo el sitio y plantillas principales para cada sección principal del sitio. Todas las plantillas de sección heredan de `base.html` y cada página del sitio hereda de una plantilla de sección. De esta manera, puede modificar fácilmente la apariencia del sitio en su conjunto, cualquier sección del sitio o cualquier página individual. Esta configuración proporciona una forma muy eficiente de trabajar y le anima a actualizar constantemente su proyecto a lo largo del tiempo.

La página de temas

Ahora que tenemos un enfoque eficiente para crear páginas, podemos centrarnos en las dos páginas siguientes: la página de temas generales y la página para mostrar entradas para un solo tema. La página de temas mostrará todos los temas que los usuarios han creado y es la primera página que implicará trabajar con datos.

El patrón de URL de temas

Primero, definimos la URL de la página de temas. Es común elegir un fragmento de URL simple que refleje el tipo de información presentada en la página. Usaremos la palabra temas, por lo que la URL `http://localhost:8000/topics/` devolverá esta página. Así es como modificamos `learning_logs/urls.py`:

```
registros_de_aprendizaje/
URL.py

"""Define patrones de URL para learning_logs."""
--recorte--
patrones de URL = [
    # Página de inicio
    ruta("", vistas.index, nombre='índice'),
    # Página que muestra todos los temas.
    ruta('temas/', vistas.temas, nombre='temas'),
]
```

El nuevo patrón de URL es la palabra temas, seguida de una barra diagonal.

Cuando Django examina una URL solicitada, este patrón coincidirá con cualquier URL que tenga la URL base seguida de temas. Puedes incluir u omitir una barra diagonal al final, pero no puede haber nada más después de la palabra temas o el patrón no coincidirá. Cualquier solicitud con una URL que coincida con este patrón se pasará a la función temas() en views.py.

La vista de temas

La función topic() necesita recuperar algunos datos de la base de datos y enviarlos a la plantilla.

Agregue lo siguiente a views.py:

views.py de django.shortcuts importar render

1 del tema de importación .models

índice def (solicitud):

--recorte--

2 temas definidos (solicitud):

""Mostrar todos los temas.""

3 temas = Topic.objects.order_by('date_added')

4 contexto = {'temas': temas}

5 render de retorno (solicitud, 'learning_logs/topics.html', contexto)

Primero importamos el modelo asociado con los datos que necesitamos 1. La función issues() necesita un parámetro: el objeto de solicitud que Django recibió del servidor 2.

Consultamos la base de datos preguntando por los objetos Topic , ordenados por el atributo date_added 3. Asignamos el conjunto de consultas resultante a temas.

Luego definimos un contexto que enviaremos a la plantilla 4. Un contexto es un diccionario en el que las claves son nombres que usaremos en la plantilla para acceder a los datos que queremos y los valores son los datos que necesitamos enviar. a la plantilla. En este caso, hay un par clave-valor que contiene el conjunto de temas que mostraremos en la página. Al crear una página que usa datos, llamamos a render() con el objeto de solicitud , la plantilla que queremos usar y el diccionario de contexto 5.

La plantilla de temas

La plantilla para la página de temas recibe el diccionario de contexto , por lo que la plantilla puede usar los datos que proporciona issues() . Cree un archivo llamado topic.html en el mismo directorio que index.html. Así es como podemos mostrar los temas en la plantilla:

temas.html {% extiende 'learning_logs/base.html' %}

{% bloquear contenido %}

<p>Temas</p>

```

1 <ul>
2 {% para tema en temas %}
3     <li>{{ tema.texto }}</li>
4 {% vacío %}
5     <li>Aún no se han agregado temas.</li>
6 {% fin de %}
6</ul>

```

```
{% contenido del bloque final %}
```

Usamos la etiqueta `{% extends %}` para heredar de `base.html`, tal como lo hicimos en la página de inicio, y luego abrimos un bloque de contenido. El cuerpo de esta página contiene una lista con viñetas de los temas que se han ingresado. En HTML estándar, una lista con viñetas se denomina lista desordenada y se indica mediante las etiquetas `<ul></ul>`. La etiqueta de apertura `<ul>` comienza la lista con viñetas de temas 1.

A continuación usamos una etiqueta de plantilla que es equivalente a un bucle `for`, que recorre los temas de la lista del diccionario de contexto 2. El código utilizado en las plantillas difiere de Python en algunos aspectos importantes. Python usa sangría para indicar qué líneas de una declaración `for` son parte de un bucle. En una plantilla, cada bucle `for` necesita una etiqueta `{% endfor %}` explícita que indique dónde se produce el final del bucle. Entonces, en una plantilla, verás bucles escritos así:

```

{% para elemento en la lista %}
    hacer algo con cada elemento
{% final de %}

```

Dentro del bucle, queremos convertir cada tema en un elemento de la lista con viñetas. Para imprimir una variable en una plantilla, ajuste el nombre de la variable entre llaves dobles. Las llaves no aparecerán en la página; simplemente le indican a Django que estamos usando una variable de plantilla. Entonces, el código `{{ topic.text }}` 3 será reemplazado por el valor del atributo de texto del tema actual en cada paso por el bucle. La etiqueta HTML `<li></li>` indica un elemento de la lista. Todo lo que esté entre estas etiquetas, dentro de un par de etiquetas `<ul></ul>`, aparecerá como un elemento con viñetas en la lista.

También usamos la etiqueta de plantilla 4 `{% vacío %}`, que le dice a Django qué hacer si no hay elementos en la lista. En este caso, imprimimos un mensaje informando al usuario que aún no se han agregado temas. Las dos últimas líneas cierran el bucle `for` 5 y luego cierran la lista con viñetas 6.

Ahora necesitamos modificar la plantilla base para incluir un enlace a la página de temas. Agregue el siguiente código a `base.html`:

```

base.html <p>
1 <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a> -
2 <a href="{% url 'learning_logs:topics' %}">Temas</a>
</p>

{% bloquear contenido %}{% bloquear contenido %}

```

Agregamos un guión después del enlace a la página de inicio 1 y luego agregamos un enlace a la página de temas usando la etiqueta de plantilla `{% url %}` nuevamente 2. Esta línea indica

Django generará un enlace que coincida con el patrón de URL con el nombre 'temas' en `learning_logs/urls.py`.

Ahora, cuando actualice la página de inicio en su navegador, verá un enlace de Temas. Cuando hace clic en el enlace, verá una página similar a la Figura 18-4.



Figura 18-4: La página de temas

Páginas de temas individuales

A continuación, debemos crear una página que pueda centrarse en un solo tema, mostrando el nombre del tema y todas las entradas de ese tema. Definiremos un nuevo patrón de URL, escribiremos una vista y crearemos una plantilla. También modificaremos la página de temas para que cada elemento de la lista con viñetas se vincule a su página de tema correspondiente.

El patrón de URL del tema

El patrón de URL para la página del tema es un poco diferente de los patrones de URL anteriores porque utilizará el atributo de identificación del tema para indicar qué tema se solicitó. Por ejemplo, si el usuario desea ver la página de detalles del tema de Ajedrez (donde el ID es 1), la URL será `http://localhost:8000/temas/1/`. Aquí hay un patrón que coincide con esta URL, que debes colocar en `learning_logs/urls.py`:

```
registros_de_aprendizaje/
URL.py
--recorte--
patrones de URL = [
    --recorte--
    # Página de detalles para un solo tema.
    ruta('temas/<int:topic_id>/', vistas.tema, nombre='tema'),
]
```

Examinemos la cadena `'topics/<int:topic_id>/'` en este patrón de URL. La primera parte de la cadena le dice a Django que busque URL que tengan la palabra temas después de la URL base. La segunda parte de la cadena, `/<int:topic_id>/`, coincide con un número entero entre dos barras diagonales y asigna el valor entero a un argumento llamado `topic_id`.

Cuando Django encuentra una URL que coincide con este patrón, llama a la función de vista `topic()` con el valor asignado a `topic_id` como argumento. Usaremos el valor de `topic_id` para obtener el tema correcto dentro de la función.

La vista de tema

La función `topic()` necesita obtener el tema y todas las entradas asociadas de la base de datos, muy parecido a lo que hicimos anteriormente en el shell de Django:

```
vistas.py --snip--
1 tema de definición (solicitud, topic_id):
    """Mostrar un solo tema y todas sus entradas."""
2 tema = Tema.objetos.get(id=topic_id)
3 entradas = topic.entry_set.order_by('-date_added')
4 contexto = {'tema': tema, 'entradas': entradas}
5 render de retorno (solicitud, 'learning_logs/topic.html', contexto)
```

Esta es la primera función de vista que requiere un parámetro distinto del objeto de solicitud. La función acepta el valor capturado por la expresión `<int:topic_id>/` y lo asigna a `topic_id` 1. Luego usamos `get()` para recuperar el tema, tal como lo hicimos en Django Shell 2. Luego, obtenemos todas las entradas asociadas con este tema y ordénalas según `date_added` 3. El signo menos delante de `date_added` ordena los resultados en orden inverso, lo que mostrará primero las entradas más recientes. Almacenamos el tema y las entradas en el diccionario de contexto 4 y llamamos a `render()` con el objeto de solicitud, `topic.html`

plantilla y el diccionario de contexto 5.

NOTA Las frases de código en 2 y 3 se denominan consultas porque consultan la base de datos para obtener información específica. Cuando escribes consultas como estas en tus propios proyectos, es útil probarlas primero en el shell de Django. Obtendrá comentarios mucho más rápidos en el shell que si escribiera una vista y una plantilla y luego verificara los resultados en un navegador.

La plantilla de tema

La plantilla debe mostrar el nombre del tema y las entradas. También debemos informar al usuario si aún no se han realizado entradas para este tema.

```
topic.html {% extiende 'learning_logs/base.html' %}

{% bloquear contenido %}

1 <p>Tema: {{ topic.text }}</p>

    <p>Entradas:</p>
2 <ul>
3 {% para entrada en entradas %}
    <li>
4         <p>{{ entrada.date_added|fecha:'M d, YH:i' }}</p>
5         <p>{{ entrada.text|saltos de línea }}</p>
    </li>
6 {% vacío %}
```



```

        <li>Aún no hay entradas para este tema.</li>
    {% final de %}
</ul>

```

```
{% contenido del bloque final %}
```

Ampliamos `base.html`, como haremos con todas las páginas del proyecto. A continuación mostramos el atributo de texto del tema que se ha solicitado 1. El tema variable está disponible porque está incluido en el diccionario de contexto. Luego comenzamos una lista con viñetas 2 para mostrar cada una de las entradas y las recorreremos 3, como hicimos con los temas anteriores.

Cada viñeta enumera dos datos: la marca de tiempo y el texto completo de cada entrada. Para la marca de tiempo 4, mostramos el valor del atributo `date_added`. En las plantillas de Django, una línea vertical (`|`) representa un filtro de plantilla: una función que modifica el valor en una variable de plantilla durante el proceso de renderizado. La fecha del filtro: `'M d, YH:i'` muestra marcas de tiempo en el formato 1 de enero de 2022 23:00. La siguiente línea muestra el valor del atributo de texto de la entrada actual. El filtro saltos de línea 5 garantiza que las entradas de texto largas incluyan saltos de línea en un formato comprensible para los navegadores, en lugar de mostrar un bloque de texto ininterrumpido. Nuevamente usamos la etiqueta de plantilla `{% vacío %}` 6 para imprimir un mensaje informando al usuario que no se han realizado entradas.

Enlaces desde la página de temas

Antes de ver la página del tema en un navegador, debemos modificar la plantilla de temas para que cada tema se vincule a la página adecuada. Este es el cambio que debe realizar en `temas.html`:

```

temas.html --snip--
    {% para tema en temas %}
    <li>
        <a href="{% url 'learning_logs:topic' topic.id %}">
            {{ tema.text }}</a></li>
    </li>
    {% vacío %}
--recorte--

```

Usamos la etiqueta de plantilla de URL para generar el enlace adecuado, según el patrón de URL en `learning_logs` con el nombre "tema". Este patrón de URL requiere un argumento `topic_id`, por lo que agregamos el atributo `topic.id` a la etiqueta de la plantilla de URL. Ahora cada tema en la lista de temas es un enlace a una página de tema, como `http://localhost:8000/topics/1/`.

Cuando actualiza la página de temas y hace clic en un tema, debería ver un página que se parece a la Figura 18-5.

NOTA Hay una diferencia sutil pero importante entre `topic.id` y `topic_id`. La expresión `topic.id` examina un tema y recupera el valor del ID correspondiente. La variable `topic_id` es una referencia a ese ID en el código. Si encuentra errores al trabajar con ID, asegúrese de utilizar estas expresiones de la manera adecuada.

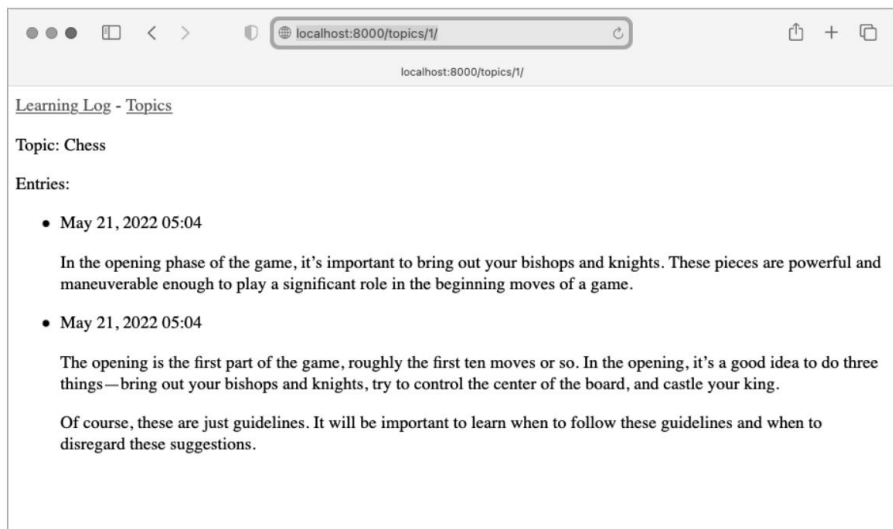


Figura 18-5: La página de detalles de un solo tema, que muestra todas las entradas de un tema

INTÉNTALO TÚ MISMO

18-7. Documentación de la plantilla: lea la documentación de la plantilla de Django en <https://docs.djangoproject.com/en/4.1/ref/templates>. Puede consultarlo cuando esté trabajando en sus propios proyectos.

18-8. Páginas de pizzería: agregue una página al proyecto Pizzería del ejercicio 18-6 (página 392) que muestre los nombres de las pizzas disponibles. Luego vincule el nombre de cada pizza a una página que muestre los ingredientes de la pizza. Asegúrese de utilizar la herencia de plantillas para crear sus páginas de manera eficiente.

Resumen

En este capítulo, aprendió cómo comenzar a crear una aplicación web simple utilizando el marco Django. Viste una breve especificación del proyecto, instalaste Django en un entorno virtual, configuraste un proyecto y verificaste que el proyecto estaba configurado correctamente. Usted configura una aplicación y define modelos para representar los datos de su aplicación. Aprendiste sobre las bases de datos y cómo Django te ayuda a migrar tu base de datos después de realizar un cambio en tus modelos. Creó un superusuario para el sitio de administración y utilizó el sitio de administración para ingresar algunos datos iniciales.

También exploraste el shell de Django, que te permite trabajar con tu los datos del proyecto en una sesión de terminal. Aprendiste a definir URL, crear funciones de visualización y escribir plantillas para crear páginas para tu sitio. También utilizó la herencia de plantillas para simplificar la estructura de las plantillas individuales y facilitar la modificación del sitio a medida que evoluciona el proyecto.

En el Capítulo 19, creará páginas intuitivas y fáciles de usar que permitirán a los usuarios agregar nuevos temas y entradas y editar las entradas existentes sin tener que pasar por el sitio de administración. También agregará un sistema de registro de usuarios, que permitirá a los usuarios crear una cuenta y crear su propio registro de aprendizaje. Este es el corazón de una aplicación web: la capacidad de crear algo con lo que cualquier número de usuarios pueda interactuar.

19

CUENTAS DE USUARIO



En el corazón de una aplicación web está la capacidad de cualquier usuario, en cualquier parte del mundo, de registrar una cuenta en su aplicación y comenzar a usarla. En este capítulo, creará formularios para que los usuarios puedan agregar sus propios temas y entradas, y editar las entradas existentes. También aprenderá cómo Django protege contra ataques comunes contra páginas basadas en formularios, por lo que no tendrá que dedicar mucho tiempo a pensar en proteger sus aplicaciones.

También implementará un sistema de autenticación de usuarios. Creará una página de registro para que los usuarios creen cuentas y luego restringirá el acceso a ciertas páginas solo a los usuarios que hayan iniciado sesión. Luego modificará algunas de las funciones de visualización para que los usuarios sólo puedan ver sus propios datos. Aprenderá a mantener los datos de sus usuarios seguros y protegidos.

Permitir a los usuarios ingresar datos

Antes de crear un sistema de autenticación para crear cuentas, primero agregaremos algunas páginas que permitan a los usuarios ingresar sus propios datos. Brindaremos a los usuarios la posibilidad de agregar un nuevo tema, agregar una nueva entrada y editar sus entradas anteriores.

Actualmente, sólo un superusuario puede ingresar datos a través del sitio de administración. No queremos que los usuarios interactúen con el sitio de administración, por lo que usaremos las herramientas de creación de formularios de Django para crear páginas que permitan a los usuarios ingresar datos.

Agregar nuevos temas

Comencemos permitiendo a los usuarios agregar un nuevo tema. Agregar una página basada en formulario funciona de manera muy similar a agregar las páginas que ya hemos creado: definimos una URL, escribimos una función de vista y escribimos una plantilla. La única diferencia significativa es la adición de un nuevo módulo llamado `form.py`, que contendrá los formularios.

El formulario del modelo de tema

Cualquier página que permita a un usuario ingresar y enviar información en una página web involucra un elemento HTML llamado formulario. Cuando los usuarios ingresan información, debemos validar que la información proporcionada sea el tipo correcto de datos y no sea malicioso, como un código diseñado para interrumpir nuestro servidor. Luego necesitamos procesar y guardar información válida en el lugar apropiado de la base de datos. Django automatiza gran parte de este trabajo.

La forma más sencilla de crear un formulario en Django es utilizar un `ModelForm`, que utiliza la información de los modelos que definimos en el Capítulo 18 para crear un formulario automáticamente. Escriba su primer formulario en el archivo `form.py`, que debe crearse en el mismo directorio que `models.py`:

`formularios.py` de formularios de importación de Django

desde `.models` importar tema

```
1 clase TopicForm(formularios.ModelForm):
    metaclass:
2     modelo = Tema
3     campos = ['texto']
4     etiquetas = {'texto': ''}
```

Primero importamos el módulo de formularios y el modelo con el que trabajaremos, `Tema`. Luego definimos una clase llamada `TopicForm`, que hereda de los formularios.
`.ModelForm` 1.

La versión más simple de un `ModelForm` consiste en una clase `Meta` anidada que le dice a Django en qué modelo basar el formulario y qué campos incluir en el formulario. Aquí especificamos que el formulario debe basarse en el modelo de tema 2 y que solo debe incluir el campo de texto 3. La cadena vacía en el diccionario de etiquetas le dice a Django que no genere una etiqueta para el campo de texto 4.

La URL nuevo_tema

La URL de una página nueva debe ser breve y descriptiva. Cuando el usuario quiera agregar un nuevo tema, lo enviaremos a `http://localhost:8000/nuevo_tema/`. Aquí está el patrón de URL para la página `new_topic`; agregue esto a `learning_logs/urls.py`:

registros_de_aprendizaje/

URL.py

```
--recorte--
patrones de URL = [
    --recorte--
    # Página para agregar un nuevo tema.
    ruta('nuevo_tema/', vistas.nuevo_tema, nombre='nuevo_tema'),
]
```

Este patrón de URL envía solicitudes a la función de vista `new_topic()`, que escribiremos a continuación.

La función de vista `new_topic()`

La función `new_topic()` necesita manejar dos situaciones diferentes: solicitudes iniciales para la página `new_topic`, en cuyo caso debería mostrar un formulario en blanco; y el procesamiento de cualquier dato enviado en el formulario. Una vez procesados los datos de un formulario enviado, es necesario redirigir al usuario a la página de temas :

`views.py` de `django.shortcuts` importa `renderizado`, `redirige`

```
desde .models importar tema
desde .forms importar TopicForm

--recorte--
def nuevo_tema(solicitud):
    """Agregar un nuevo tema."""
    1 si solicitud.método! = 'POST':
        # No se enviaron datos; crear un formulario en blanco.
    2     formulario = TemaForm()
        demás:
    3         # datos POST enviados; procesar datos.
        formulario = TemaForm(datos=solicitud.POST)
    4     si form.is_valid():
    5         formulario.guardar()
    6         devolver redirección ('learning_logs: temas')

    # Mostrar un formulario en blanco o no válido.
    7 contexto = {'formulario': formulario}
        devolver render(solicitud, 'learning_logs/new_topic.html', contexto)
```

Importamos la función de redirección, que usaremos para redirigir al usuario a la página de temas después de enviar su tema. También importamos el formulario que acabamos de escribir, `TopicForm`.

Solicitudes OBTENER y PUBLICAR

Los dos tipos principales de solicitudes que utilizará al crear aplicaciones son GET y POST. Utiliza solicitudes GET para páginas que solo leen datos del servidor.

Por lo general, utiliza solicitudes POST cuando el usuario necesita enviar información a través de un formulario. Especificaremos el método POST para procesar todos nuestros formularios. (Existen algunos otros tipos de solicitudes, pero no las usaremos en este proyecto).

La función `new_topic()` toma el objeto de solicitud como parámetro.

Cuando el usuario solicita inicialmente esta página, su navegador enviará una solicitud GET. Una vez que el usuario haya completado y enviado el formulario, su navegador enviará una solicitud POST. Dependiendo de la solicitud, sabremos si el usuario solicita un formulario en blanco (GET) o nos pide que procesemos un formulario completo (POST).

Usamos una prueba `if` para determinar si el método de solicitud es GET o POST 1. Si el método de solicitud no es POST, la solicitud probablemente sea GET, por lo que debemos devolver un formulario en blanco. (Si se trata de otro tipo de solicitud, aún es seguro devolver un formulario en blanco). Creamos una instancia de `TopicForm` 2, la asignamos al formulario variable y enviamos el formulario a la plantilla en el diccionario de contexto 7. Porque no incluye argumentos al crear una instancia de `TopicForm`, Django crea un formulario en blanco que el usuario puede completar.

Si el método de solicitud es POST, el bloque `else` se ejecuta y procesa los datos enviados en el formulario. Hacemos una instancia de `TopicForm` 3 y le pasamos los datos ingresados por el usuario, los cuales se asignan a `request.POST`. La forma El objeto devuelto contiene la información enviada por el usuario.

No podemos guardar la información enviada en la base de datos hasta que hayamos comprobado que es válida. 4. El método `is_valid()` verifica que se hayan completado todos los campos obligatorios (todos los campos de un formulario son obligatorios de forma predeterminada) y que los datos ingresado coincide con los tipos de campo esperados; por ejemplo, que la longitud del texto sea inferior a 200 caracteres, como especificamos en `models.py` en el Capítulo 18. Esta validación automática nos ahorra mucho trabajo. Si todo es válido, podemos llamar a `save()` 5, que escribe los datos del formulario en la base de datos.

Una vez que hayamos guardado los datos, podremos salir de esta página. La función de redirección() toma el nombre de una vista y redirige al usuario a la página asociada con esa vista. Aquí usamos `redirección()` para redirigir el navegador del usuario a la página de temas 6, donde el usuario debería ver el tema que acaba de ingresar en la lista de temas.

La variable de contexto se define al final de la función de vista y la página se representa usando la plantilla `new_topic.html`, que crearemos a continuación. Este código se coloca fuera de cualquier bloque `if`; se ejecutará si se creó un formulario en blanco y se ejecutará si se determina que un formulario enviado no es válido. Un formulario no válido incluirá algunos mensajes de error predeterminados para ayudar al usuario a enviar datos aceptables.

La plantilla `new_topic`

Ahora crearemos una nueva plantilla llamada `new_topic.html` para mostrar el formulario que acabamos de crear:

```
new_topic.html {% extiende "learning_logs/base.html" %}

{% bloquear contenido %}
<p>Agregar un nuevo tema:</p>

1 <formulario action="{% url 'learning_logs:new_topic' %}" método='post'>
2 {% csrf_token %}
3 {{ formulario.as_div }}
4 <button name="submit">Agregar tema</button>
   </formulario>

{% contenido del bloque final %}
```

Esta plantilla extiende `base.html`, por lo que tiene la misma estructura base que la resto de las páginas del Registro de aprendizaje. Usamos las etiquetas `<form></form>` para definir un formulario HTML 1. El argumento de acción le dice al navegador dónde enviar los datos enviados en el formulario; en este caso, lo enviamos de regreso a la función de vista `new_topic()`. El argumento del método le dice al navegador que envíe los datos como una solicitud POST.

Django usa la etiqueta de plantilla `{% csrf_token %}` 2 para evitar que los atacantes utilicen el formulario para obtener acceso no autorizado al servidor. (Este tipo de ataque se denomina falsificación de solicitud entre sitios). A continuación, mostramos el formulario; Aquí puedes ver lo simple que Django puede realizar ciertas tareas, como mostrar un formulario. Sólo necesitamos incluir la variable de plantilla `{{ form.as_div }}` para que Django cree todos los campos necesarios para mostrar el formulario automáticamente 3. El modificador `as_div` le dice a Django que represente todos los elementos del formulario como elementos HTML `<div></div>`; esta es una forma sencilla de mostrar el formulario de forma ordenada.

Django no crea un botón de envío para formularios, por lo que definimos uno antes de cerrar el formulario 4.

Vinculación a la página `new_topic`

A continuación, incluimos un enlace a la página `new_topic` en la página de temas :

```
temas.html {% extiende "learning_logs/base.html" %}

{% bloquear contenido %}

<p>Temas</p>

<ul>
  --recorte--
</ul>

<a href="{% url 'learning_logs:new_topic' %}">Agregar un nuevo tema</a>

{% contenido del bloque final %}
```

Coloque el enlace después de la lista de temas existentes. La figura 19-1 muestra la forma resultante; Intente utilizar el formulario para agregar algunos temas nuevos propios.



Figura 19-1: La página para agregar un nuevo tema

Agregar nuevas entradas

Ahora que el usuario puede agregar un tema nuevo, también querrá agregar nuevas entradas. Nuevamente definiremos una URL, escribiremos una función de vista y una plantilla, y vincularemos a la página. Pero primero, agregaremos otra clase a form.py.

El formulario del modelo de entrada

Necesitamos crear un formulario asociado al modelo Entry , pero esta vez, con un poco más de personalización que TopicForm:

formularios.py de formularios de importación de Django

desde .models importar tema, entrada

```
class TemaForm(formularios.ModelForm):
    --recorte--

class EntryForm(formularios.ModelForm):
    metaclass:
        modelo = Entrada
        campos = ['texto']
1         etiquetas = {'texto': ''}
2         widgets = {'text': formularios.Textarea(attrs={'cols': 80})}
```

Actualizamos la declaración de importación para incluir Entrada y Tema. Creamos una nueva clase llamada EntryForm que hereda de formularios.ModelForm. La clase EntryForm tiene una clase Meta anidada que enumera el modelo en el que se basa y el campo que se incluirá en el formulario. Nuevamente le damos al campo 'texto' una etiqueta en blanco 1.

Para EntryForm, incluimos el atributo 2 de widgets . Un widget es un HTML elemento de formulario, como un cuadro de texto de una sola línea, un área de texto de varias líneas o una lista desplegable. Al incluir el atributo widgets , puedes anular el atributo de Django.

opciones de widget predeterminadas. Aquí le estamos diciendo a Django que use un elemento form.Textarea con un ancho de 80 columnas, en lugar de las 40 columnas predeterminadas. Esto les da a los usuarios suficiente espacio para escribir una entrada significativa.

La URL nueva_entrada

Las nuevas entradas deben estar asociadas con un tema en particular, por lo que debemos incluir un argumento topic_id en la URL para agregar una nueva entrada. Aquí está la URL, que agregas a learning_logs/urls.py:

registros_de_aprendizaje/

URL.py

```
--recorte--
patrones de URL = [
    --recorte--
    # Página para agregar una nueva entrada.
    ruta('nueva_entrada/<int:topic_id>/', vistas.nueva_entrada, nombre='nueva_entrada'),
]
```

Este patrón de URL coincide con cualquier URL con el formato http://localhost:8000/new_entry/id/, donde id es un número que coincide con el ID del tema. El código <int:topic_id> captura un valor numérico y lo asigna a la variable topic_id. Cuando se solicita una URL que coincide con este patrón, Django envía la solicitud y el ID del tema a la función de vista new_entry().

La función de vista new_entry()

La función de visualización para new_entry es muy parecida a la función para agregar un nuevo tema. Agregue el siguiente código a su archivo views.py :

views.py de django.shortcuts importa renderizado, redirige

```
desde .models importar tema
desde .forms importar TopicForm, EntryForm

--recorte--
def nueva_entrada(solicitud, id_tema):
    """Agregar una nueva entrada para un tema en particular."""
    1 tema = Tema.objetos.get(id=topic_id)

    2 si solicitud.método! = 'POST':
        # No se enviaron datos; crear un formulario en blanco.
        3 formulario = Formulario de entrada()
        demás:
            # datos POST enviados; procesar datos.
            4 formulario = Formulario de entrada (datos = solicitud.POST)
            si form.is_valid():
                5 nueva_entrada = formulario.guardar(confirmar=False)
                6 new_entry.topic = tema
                nueva_entrada.guardar()
                7 devolver redirección('learning_logs:tema', topic_id=topic_id)

    # Mostrar un formulario en blanco o no válido.
    contexto = {'tema': tema, 'formulario': formulario}
    devolver render(solicitud, 'learning_logs/new_entry.html', contexto)
```

Actualizamos la declaración de importación para incluir el EntryForm que acabamos de crear. La definición de `new_entry()` tiene un parámetro `topic_id` para almacenar el valor que recibe de la URL. Necesitaremos el tema para representar la página y procesar los datos del formulario, por lo que usamos `topic_id` para obtener el objeto de tema 1 correcto.

A continuación, verificamos si el método de solicitud es POST o GET 2. El bloque `if` se ejecuta si es una solicitud GET y creamos una instancia en blanco de EntryForm 3.

Si el método de solicitud es POST, procesamos los datos creando una instancia de EntryForm, completado con los datos POST del objeto de solicitud 4. Luego verificamos si el formulario es válido. Si es así, debemos configurar el atributo de tema del objeto de entrada antes de guardarlo en la base de datos. Cuando llamamos a `save()`, incluimos el argumento `commit=False` 5 para indicarle a Django que cree un nuevo objeto de entrada y lo asigne a `new_entry`, sin guardarlo todavía en la base de datos. Nosotros fijamos el tema atributo de `new_entry` al tema que extrajimos de la base de datos al comienzo de la función 6. Luego llamamos a `save()` sin argumentos, guardando la entrada en la base de datos con el tema asociado correcto.

La llamada de `redirección()` requiere dos argumentos: el nombre de la vista a la que queremos redirigir y el argumento que requiere la función de vista. 7. Aquí, estamos redirigiendo a `tema()`, que necesita el argumento `topic_id`. Luego, esta vista muestra la página del tema para la que el usuario realizó una entrada y debería ver su nueva entrada en la lista de entradas.

Al final de la función, creamos un diccionario de contexto y representamos la página usando la plantilla `new_entry.html`. Este código se ejecutará para un formulario en blanco o para un formulario que se envió pero resulta no válido.

La plantilla `new_entry`

Como puede ver en el siguiente código, la plantilla para `new_entry` es similar a la plantilla para `new_topic`:

```
new_entry.html {% extiende "learning_logs/base.html" %}

{% bloquear contenido %}

1 <p><a href="{% url 'learning_logs:topic' topic.id %}">{{ tema }}</a></p>

    <p>Añadir una nueva entrada:</p>
2 <formulario action="{% url 'learning_logs:new_entry' topic.id %}" método='post'>
    {% csrf_token %}
    {{ formulario.as_div }}
    <button name='submit'>Agregar entrada</button>
    </formulario>

{% contenido del bloque final %}
```

Mostramos el tema en la parte superior de la página 1, para que el usuario pueda ver cuál tema al que están agregando una entrada. El tema también actúa como un enlace a la página principal de ese tema.

El argumento de acción del formulario incluye el valor topic.id en la URL, por lo que la función de vista puede asociar la nueva entrada con el tema 2 correcto. Aparte de eso, esta plantilla se parece a new_topic.html.

Vinculación a la página new_entry

A continuación, debemos incluir un enlace a la página new_entry de cada página de tema, en la plantilla de tema:

```
topic.html {% extiende "learning_logs/base.html" %}

{% bloquear contenido %}

<p>Tema: {{ tema }}</p>

<p>Entradas:</p>
<p>
  <a href="{% url 'learning_logs:new_entry' topic.id %}">Agregar nueva entrada</a>
</p>

<ul>
--recorte--
</ul>

{% contenido del bloque final %}
```

Colocamos el enlace para agregar entradas justo antes de mostrar las entradas, porque agregar una nueva entrada será la acción más común en esta página. La Figura 19-2 muestra la página new_entry . Ahora los usuarios pueden agregar nuevos temas y tantas entradas como quieran para cada tema. Pruebe la página new_entry agregando algunas entradas a algunos de los temas que ha creado.



Figura 19-2: La página new_entry

Editar entradas

Ahora crearemos una página para que los usuarios puedan editar las entradas que agregaron.

La URL `editar_entrada`

La URL de la página debe pasar el ID de la entrada que se va a editar. Aquí está `learning_logs/urls.py`:

```
URL.py --snip--
patrones de URL = [
    --recorte--
    # Página para editar una entrada.
    ruta('edit_entry/<int:entry_id>/', vistas.edit_entry, nombre='edit_entry'),
]
```

Este patrón de URL coincide con URL como `http://localhost:8000/edit_entry/id/`.

Aquí el valor de `id` se asigna al parámetro `Entry_ID`. Django envía solicitudes que coinciden con este formato a la función de vista `edit_entry()`.

La función de vista `edit_entry()`

Cuando la página `edit_entry` recibe una solicitud GET, la función `edit_entry()` devuelve un formulario para editar la entrada. Cuando la página recibe una solicitud POST con texto de entrada revisado, guarda el texto modificado en la base de datos:

`views.py` de `django.shortcuts` importa `renderizado`, `redirige`

```
desde .models importar tema, entrada
desde .forms importar TopicForm, EntryForm
--recorte--

def editar_entrada(solicitud, entrada_id):
    """Editar una entrada existente."""
    1 entrada = Entrada.objetos.get(id=entrada_id)
    tema = entrada.tema

    si solicitud.método! = 'POST':
        # Solicitud inicial; Complete previamente el formulario con la entrada actual.
        2 formulario = Formulario de entrada (instancia = entrada)
        demás:
            # datos POST enviados; procesar datos.
            3 formulario = Formulario de entrada (instancia = entrada, datos = solicitud.POST)
            si form.is_valid():
                4 formulario.guardar()
                5 return redirigir('learning_logs:tema', topic_id=topic.id)

    contexto = {'entrada': entrada, 'tema': tema, 'formulario': formulario}
    devolver render(solicitud, 'learning_logs/edit_entry.html', contexto)
```

Primero importamos el modelo `Entry`. Luego obtenemos el objeto de entrada que el usuario desea editar 1 y el tema asociado con esta entrada. en el si

bloque, que se ejecuta para una solicitud GET, creamos una instancia de EntryForm con el argumento instancia=entrada 2. Este argumento le dice a Django que cree el formulario, previamente completado con información del objeto de entrada existente. El usuario verá sus datos existentes y podrá editarlos.

Al procesar una solicitud POST, pasamos los argumentos instancia=entrada y datos=request.POST 3. Estos argumentos le dicen a Django que cree una instancia de formulario basada en la información asociada con el objeto de entrada existente, actualizada con cualquier dato relevante de la solicitud. .CORREO. Luego verificamos si el formulario es válido; si es así, llamamos a save() sin argumentos porque la entrada ya está asociada con el tema correcto 4. Luego redirigimos a la página del tema, donde el usuario debería ver la versión actualizada de la entrada que editó 5.

Si mostramos un formulario inicial para editar la entrada o si el formulario enviado no es válido, creamos el diccionario de contexto y representamos la página usando la plantilla edit_entry.html.

La plantilla edit_entry

A continuación, creamos una plantilla edit_entry.html, que es similar a new_entry.html:

```
edit_entry.html {% extiende "learning_logs/base.html" %}

{% bloquear contenido %}

<p><a href="{% url 'learning_logs:topic' topic.id %}">{{ tema }}</a></p>

<p>Editar entrada:</p>

1 <formulario action="{% url 'learning_logs:edit_entry' entrada.id %}" método='post'>
    {% csrf_token %}
    {{ formulario.as_div }}
2 <button name="submit">Guardar cambios</button>
</formulario>

{% contenido del bloque final %}
```

El argumento de acción envía el formulario de regreso a la función edit_entry() para procesar 1. Incluimos Entry.id como argumento en {% url %} etiqueta, para que la función de vista pueda modificar el objeto de entrada correcto. Etiquetamos el botón de enviar como Guardar cambios para recordarle al usuario que está guardando las ediciones, no creando una nueva entrada 2.

Vinculación a la página edit_entry

Ahora necesitamos incluir un enlace a la página edit_entry para cada entrada en la página del tema:

```
tema.html --recorte--
    {% para entrada en entradas %}
    <li>
```

```

<p>{{ entrada.date_added|fecha:'M d, YH:i' }}</p>
<p>{{ entrada.text|saltos de línea }}</p>
<p>
  <a href="{% url 'learning_logs:edit_entry' entrada.id %}">
    Editar entrada</a></p>
</li>
--recorte--

```

Incluimos el enlace de edición después de que se haya mostrado la fecha y el texto de cada entrada. Usamos la etiqueta de plantilla {% url %} para determinar la URL para el patrón de URL nombrado edit_entry, junto con el atributo ID de la entrada actual en el bucle (entry.id). El texto del enlace Editar entrada aparece después de cada entrada en la página. La Figura 19-3 muestra cómo se ve la página del tema con estos enlaces.

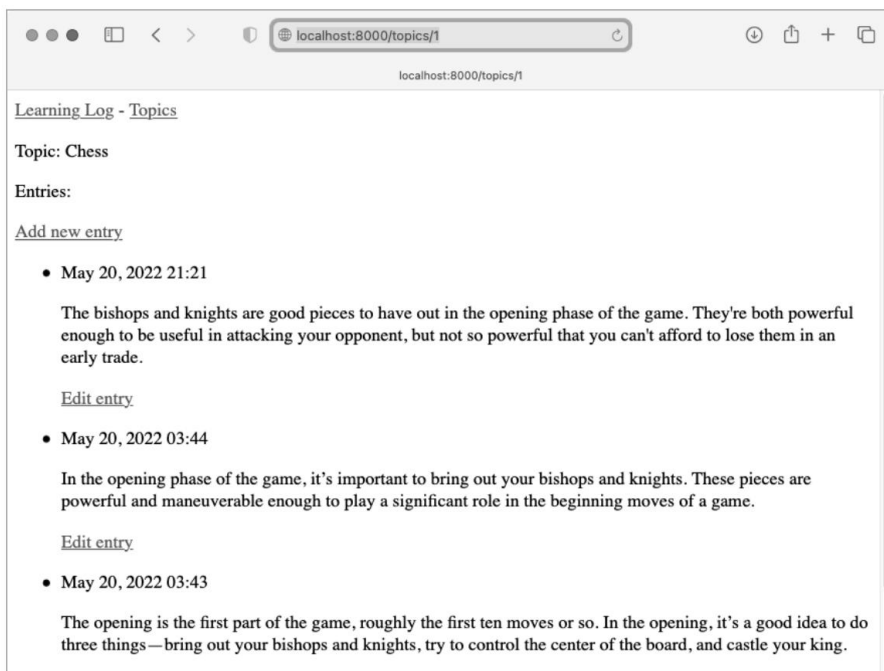


Figura 19-3: Cada entrada ahora tiene un enlace para editarla.

Learning Log ahora tiene la mayor parte de la funcionalidad que necesita. Los usuarios pueden agregar temas y entradas, y pueden leer cualquier conjunto de entradas que deseen. En la siguiente sección, implementaremos un sistema de registro de usuarios para que cualquiera pueda crear una cuenta en Learning Log y crear su propio conjunto de temas y entradas.

INTÉNTALO TÚ MISMO

19-1. Blog: inicie un nuevo proyecto de Django llamado Blog. Cree una aplicación llamada blogs, con un modelo que represente un blog general y un modelo que represente una publicación de blog individual. Proporcione a cada modelo un conjunto apropiado de campos.

Cree un superusuario para el proyecto y utilice el sitio de administración para crear un blog y un par de publicaciones breves. Cree una página de inicio que muestre todas las publicaciones en el orden apropiado.

Cree páginas para crear un blog, crear publicaciones nuevas y editar publicaciones existentes. Utilice sus páginas para asegurarse de que funcionen.

Configurar cuentas de usuario

En esta sección, configuraremos un sistema de autorización y registro de usuarios para que las personas puedan registrar una cuenta, iniciar sesión y cerrar sesión. Crearemos una nueva aplicación que contenga todas las funciones relacionadas con el trabajo con los usuarios. Usaremos el sistema de autenticación de usuario predeterminado incluido con Django para hacer la mayor parte del trabajo posible. También modificaremos ligeramente el modelo de tema para que cada tema pertenezca a un determinado usuario.

La aplicación de cuentas

Comenzaremos creando una nueva aplicación llamada cuentas, usando el comando startapp :

```
(ll_env)learning_log$ python administrar.py cuentas de inicio de aplicación
(ll_env)learning_log$ ls
1 cuentas db.sqlite3 learning_logs ll_env ll_project Manage.py
  (ll_env)learning_log$ ls cuentas
2 __init__.py admin.py apps.py migraciones modelos.py pruebas.py vistas.py
```

El sistema de autenticación predeterminado se basa en el concepto de cuentas de usuario, por lo que el uso de cuentas de nombres facilita la integración con el sistema predeterminado. El comando startapp que se muestra aquí crea un nuevo directorio llamado cuentas 1 con una estructura idéntica a la aplicación learning_logs 2.

Agregar cuentas a settings.py

Necesitamos agregar nuestra nueva aplicación a INSTALLED_APPS en settings.py, así:

```
configuración.py --snip--
INSTALLED_APPS = [
    # Mis aplicaciones
    'registros_aprendizaje',
    'cuentas',
```

```
# Aplicaciones Django predeterminadas.
--recorte--
]
--recorte--
```

Ahora Django incluirá la aplicación de cuentas en el proyecto general.

Incluyendo las URL de las cuentas

A continuación, debemos modificar la raíz `urls.py` para que incluya las URL que escribiremos para la aplicación de cuentas :

`ll_project/urls.py` de `django.contrib` administrador de importación
desde la ruta de importación `django.urls`, incluya

```
patrones de URL = [
    ruta('admin/', admin.sitio.urls),
    ruta('cuentas/', incluir('cuentas.urls')),
    ruta("", incluir('learning_logs.urls')),
]
```

Agregamos una línea para incluir el archivo `urls.py` de cuentas. Esta línea coincidirá con cualquier URL que comience con la palabra cuentas, como `http://localhost:8000/cuentas/iniciar sesión/`.

La página de inicio de sesión

Primero implementaremos una página de inicio de sesión. Usaremos la vista de inicio de sesión predeterminada que proporciona Django, por lo que el patrón de URL para esta aplicación se ve un poco diferente. Cree un nuevo archivo `urls.py` en el directorio `ll_project/accounts/` y agréguele lo siguiente:

`cuentas/urls.py` """Define patrones de URL para cuentas."""

desde la ruta de importación `django.urls`, incluya

```
nombre_aplicación = 'cuentas'
patrones de URL = [
    # Incluir URL de autenticación predeterminadas.
    ruta("", incluir('django.contrib.auth.urls')),
]
```

Importamos la función de ruta y luego importamos la función de inclusión para que podamos incluir algunas URL de autenticación predeterminadas que Django ha definido.

Estas URL predeterminadas incluyen patrones de URL con nombre, como "iniciar sesión" y "cerrar sesión". Configuramos la variable `nombre_aplicación` en 'cuentas' para que Django pueda distinguir estas URL de las URL que pertenecen a otras aplicaciones. Incluso las URL predeterminadas proporcionadas por Django, cuando se incluyen en el archivo `urls.py` de la aplicación de cuentas, serán accesibles a través del espacio de nombres de las cuentas.

El patrón de la página de inicio de sesión coincide con la URL `http://localhost:8000/accounts/acceso/`. Cuando Django lee esta URL, la palabra `cuentas` le dice a Django que busque en `cuentas/urls.py`, y el inicio de sesión le dice que envíe solicitudes a la vista de inicio de sesión predeterminada de Django.

La plantilla de inicio de sesión

Cuando el usuario solicita la página de inicio de sesión, Django usará una función de vista predeterminada, pero aún debemos proporcionar una plantilla para la página. Las vistas de autenticación predeterminadas buscan plantillas dentro de una carpeta llamada `registro`, por lo que necesitaremos crear esa carpeta. Dentro del directorio `ll_project/accounts/`, cree un directorio llamado `plantillas`; Dentro de eso, cree otro directorio llamado `registro`. Aquí está la plantilla `login.html`, que debe guardarse en `ll_project/accounts/plantillas/registro`:

```
login.html {% extiende 'learning_logs/base.html' %}
```

```
{% bloquear contenido %}
```

```
1 {% si formulario.erroses %}
```

```
    <p>Su nombre de usuario y contraseña no coinciden. Por favor, inténtalo de nuevo.</p>
```

```
{% terminara si %}
```

```
2 <formulario acción="{% url 'cuentas: iniciar sesión' %}" método='publicación'>
```

```
    {% csrf_token %}
```

```
3 {{ formulario.as_div }}
```

```
4 <button name="submit">Iniciar sesión</button>
```

```
</formulario>
```

```
{% contenido del bloque final %}
```

Esta plantilla amplía `base.html` para garantizar que la página de inicio de sesión tenga la misma apariencia que el resto del sitio. Tenga en cuenta que una plantilla en una aplicación puede heredar de una plantilla en otra aplicación.

Si el atributo de errores del formulario está configurado, mostramos un mensaje de error 1, informando que la combinación de nombre de usuario y contraseña no coincide con nada almacenado en la base de datos.

Queremos que la vista de inicio de sesión procese el formulario, por lo que configuramos el argumento de acción como la URL de la página de inicio de sesión 2. La vista de inicio de sesión envía un objeto de formulario a la plantilla, y depende de nosotros mostrar el formulario 3 y agregar un botón de enviar 4.

La configuración `LOGIN_REDIRECT_URL`

Una vez que un usuario inicia sesión exitosamente, Django necesita saber dónde enviar a ese usuario. Controlamos esto en el archivo de configuración.

Agregue el siguiente código al final de settings.py:

```
configuración.py --snip--
# Mi configuración.
LOGIN_REDIRECT_URL = 'registros_de_aprendizaje:índice'
```

Con todas las configuraciones predeterminadas en settings.py, es útil marcar la sección donde agregamos nuevas configuraciones. La primera configuración nueva que agregaremos es LOGIN_REDIRECT_URL, que le dice a Django a qué URL redirigir después de un intento de inicio de sesión exitoso.

Vinculación a la página de inicio de sesión

Agreguemos el enlace de inicio de sesión a base.html para que aparezca en cada página. No queremos que el enlace se muestre cuando el usuario ya haya iniciado sesión, por lo que lo anidamos dentro de una etiqueta {% if %} :

```
base.html <p>
    <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a> -
    <a href="{% url 'learning_logs:topics' %}">Temas</a> -
1 {% si usuario.is_authenticated %}
2 Hola, {{ usuario.nombredeusuario }}.
    {% demás %}
3 <a href="{% url 'accounts:login' %}">Iniciar sesión</a>
    {% terminara si %}
</p>

{% bloquear contenido %}{% bloquear contenido %}
```

En el sistema de autenticación de Django, cada plantilla tiene un objeto de usuario disponible que siempre tiene un atributo is_authenticated establecido: el atributo es Verdadero si el usuario ha iniciado sesión y Falso si no lo está. Este atributo le permite mostrar un mensaje a usuarios autenticados y otro a usuarios no autenticados.

Aquí mostramos un saludo a los usuarios actualmente conectados 1. Autenticado los usuarios tienen un conjunto de atributos de nombre de usuario adicional , que utilizamos para personalizar el saludo y recordarle al usuario que ha iniciado sesión 2. Para los usuarios que no han sido autenticados, mostramos un enlace a la página de inicio de sesión 3.

Usando la página de inicio de sesión

Ya hemos configurado una cuenta de usuario, así que iniciemos sesión para ver si la página funciona. Vaya a <http://localhost:8000/admin/>. Si todavía ha iniciado sesión como administrador, busque un enlace para cerrar sesión en el encabezado y haga clic en él.

Cuando cierre la sesión, vaya a <http://localhost:8000/accounts/login/>. Debería ver una página de inicio de sesión similar a la que se muestra en la Figura 19-4. Ingrese el nombre de usuario y la contraseña que configuró anteriormente y volverá a la página de inicio. El encabezado de la página de inicio debe mostrar un saludo personalizado con su nombre de usuario.

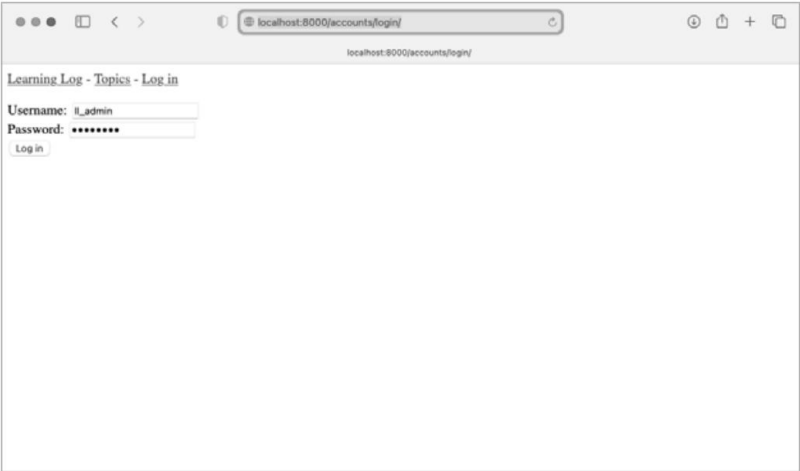


Figura 19-4: La página de inicio de sesión

Saliendo de tu cuenta

Ahora debemos proporcionar una forma para que los usuarios cierren sesión. Las solicitudes de cierre de sesión deben enviarse como solicitudes POST, por lo que agregaremos un pequeño formulario de cierre de sesión a base.html. Cuando los usuarios hacen clic en el botón de cerrar sesión, irán a una página que confirma que han cerrado sesión.

Agregar un formulario de cierre de sesión a base.html

Agregaremos el formulario para cerrar sesión en base.html para que esté disponible en todas las páginas. Lo incluiremos en otro bloque if , para que solo los usuarios que ya hayan iniciado sesión puedan verlo:

```
base.html --recorte--
{% bloquear contenido %}{% bloquear contenido %}

{% si usuario.is_authenticated %}
1 <hora />
2 <formulario action="{% url 'accounts:logout' %}" método='post'>
    {% csrf_token %}
    <button name='submit'>Cerrar sesión</button>
</formulario>
{% terminara si %}
```

El patrón de URL predeterminado para cerrar sesión es 'cuentas/cerrar sesión/'. Sin embargo, la solicitud debe enviarse como solicitud POST; de lo contrario, los atacantes pueden forzar fácilmente solicitudes de cierre de sesión. Para que la solicitud de cierre de sesión utilice POST, definimos un formulario simple.

Colocamos el formulario en la parte inferior de la página, debajo de un elemento de regla horizontal (<hr />)

1. Esta es una manera fácil de mantener siempre el botón de cerrar sesión en un

posición constante debajo de cualquier otro contenido de la página. El formulario en sí tiene la URL de cierre de sesión como argumento de acción y 'publicar' como método de solicitud 2.

Cada formulario en Django debe incluir `{% csrf_token %}`, incluso un formulario simple como este. Este formulario está vacío excepto por el botón de enviar.

La configuración de `LOGOUT_REDIRECT_URL`

Cuando el usuario hace clic en el botón de cerrar sesión, Django necesita saber dónde enviarlos.

Controlamos este comportamiento en `settings.py`:

`configuración.py` --snip--

```
# Mi configuración.
```

```
LOGIN_REDIRECT_URL = 'registros_de_aprendizaje:índice'
```

```
LOGOUT_REDIRECT_URL = 'registros_de_aprendizaje:índice'
```

La configuración `LOGOUT_REDIRECT_URL` que se muestra aquí le dice a Django que redirija a los usuarios desconectados a la página de inicio. Esta es una forma sencilla de confirmar que cerraron la sesión, porque ya no deberían ver su nombre de usuario después de cerrar la sesión.

La página de registro

A continuación, crearemos una página para que los nuevos usuarios puedan registrarse. Usaremos el `UserCreationForm` predeterminado de Django, pero escribiremos nuestra propia función de vista y plantilla.

La URL de registro

El siguiente código proporciona el patrón de URL para la página de registro, que debe colocarse en `cuentas/urls.py`:

`cuentas/urls.py` `"""Define patrones de URL para cuentas."""`

```
desde la ruta de importación django.urls, incluya
```

```
de . importar vistas
```

```
nombre_aplicación = cuentas
```

```
patrones de URL = [
```

```
    # Incluir URL de autenticación predeterminadas.
```

```
    ruta("", incluir("django.contrib.auth.urls")),
```

```
    # Página de registro.
```

```
    ruta('registro/', vistas.registro, nombre='registro'),
```

```
]
```

Importamos el módulo de vistas de las cuentas, que necesitamos porque estamos escribiendo nuestra propia vista para la página de registro. El patrón para la página de registro coincide con la URL `http://localhost:8000/accounts/register/` y envía solicitudes a la función `registro()` que estamos a punto de escribir.

La función de vista de registro()

La función de vista de registro() necesita mostrar un formulario de registro en blanco cuando se solicita por primera vez la página de registro y luego procesar los formularios de registro completos cuando se envían. Cuando un registro es exitoso, la función también necesita iniciar sesión con el nuevo usuario. Agregue el siguiente código a cuentas/views.py:

```

cuentas/ desde django.shortcuts importar render, redirigir
vistas.py desde django.contrib.auth importar inicio de sesión
desde django.contrib.auth.forms importar UserCreationForm

registro def (solicitud):
    """Registrar un nuevo usuario."""
    si solicitud.método! = 'POST':
        # Mostrar formulario de registro en blanco.
1         formulario = Formulario de creación de usuario()
        demás:
            # Procesar formulario completado.
2             formulario = UserCreationForm(datos=solicitud.POST)

3             si form.is_valid():
4                 nuevo_usuario = formulario.guardar()
                    # Inicie sesión como usuario y luego redirija a la página de inicio.
5                 iniciar sesión (solicitud, nuevo_usuario)
6                 devolver redirección('learning_logs:index')

        # Mostrar un formulario en blanco o no válido.
        contexto = {'formulario': formulario}
        return render(solicitud, 'registro/registro.html', contexto)

```

Importamos las funciones render() y redirigir() , y luego importamos la función login() para iniciar la sesión del usuario si su información de registro es correcta. También importamos el UserCreationForm predeterminado. En la función Register() comprobamos si estamos respondiendo a una solicitud POST. Si no es así, creamos una instancia de UserCreationForm sin datos iniciales 1.

Si respondemos a una solicitud POST, creamos una instancia de UserCreationForm en función de los datos enviados 2. Verificamos que los datos sean válidos 3; en este caso, que el nombre de usuario tenga los caracteres apropiados, que las contraseñas coincidan y que el El usuario no intenta hacer nada malicioso en su envío.

Si los datos enviados son válidos, llamamos al método save() del formulario para guardar el nombre de usuario y el hash de la contraseña en la base de datos. 4. Save () El método devuelve el objeto de usuario recién creado, que asignamos a new_user. Cuando se guarda la información del usuario, iniciamos sesión llamando al login() funciona con la solicitud y los objetos new_user 5, lo que crea una sesión válida para el nuevo usuario. Finalmente redirigimos al usuario a la página de inicio 6,

donde un saludo personalizado en el encabezado les indica que su registro fue exitoso.

Al final de la función, renderizamos la página, que será una formulario en blanco o un formulario enviado que no es válido.

La plantilla de registro

Ahora cree una plantilla para la página de registro, que será similar a la página de inicio de sesión. Asegúrese de guardarlo en el mismo directorio que login.html:

```
registrarse.html {% extiende "learning_logs/base.html" %}
```

```
{% bloquear contenido %}
```

```
<formulario acción="{% url 'cuentas:registro' %}" método='publicación'>
  {% csrf_token %}
  {{ formulario.as_div }}
```

```
<botón nombre="enviar">Registrarse</botón>
</formulario>
```

```
{% contenido del bloque final %}
```

Esto debería parecerse a las otras plantillas basadas en formularios que hemos estado escribiendo. Usamos el método `as_div` nuevamente para que Django muestre todos los campos del formulario de manera adecuada, incluido cualquier mensaje de error si el formulario no se completa correctamente.

Enlace a la página de registro

A continuación, agregaremos código para mostrar el enlace de la página de registro a cualquier usuario que no haya iniciado sesión actualmente:

```
base.html --recorte--
```

```
{% si usuario.is_authenticated %}
  Hola, {{ usuario.nombre de usuario }}.
{% demás %}

<a href="{% url 'accounts:register' %}">Registrarse</a> -
<a href="{% url 'accounts:login' %}">Iniciar sesión</a>

{% terminara si %}
```

```
--recorte--
```

Ahora los usuarios que han iniciado sesión ven un saludo personalizado y un botón para cerrar sesión. Los usuarios que no han iniciado sesión ven un enlace de registro y un enlace de inicio de sesión. Pruebe la página de registro creando varias cuentas de usuario con diferentes nombres de usuario.

En la siguiente sección, restringiremos algunas de las páginas para que estén disponibles sólo para usuarios registrados, y nos aseguraremos de que cada tema pertenezca a un usuario específico.

NOTA sistema de registro que hemos configurado permite que cualquiera pueda crear cualquier cantidad de cuentas para Learning Log. Algunos sistemas requieren que los usuarios confirmen su identidad enviando un correo electrónico de confirmación al que los usuarios deben responder. Al hacerlo, el sistema genera menos cuentas de spam que el sistema simple que estamos usando aquí. Sin embargo, cuando estás aprendiendo a crear aplicaciones, es perfectamente apropiado practicar con un sistema de registro de usuarios simple como el que estamos usando.

INTÉNTALO TÚ MISMO

19-2. Cuentas de blog: agregue un sistema de registro y autenticación de usuarios al proyecto de blog que inició en el ejercicio 19-1 (página 415). Asegúrese de que los usuarios que hayan iniciado sesión vean su nombre de usuario en algún lugar de la pantalla y que los usuarios no registrados vean un enlace a la página de registro.

Permitir que los usuarios sean propietarios de sus datos

Los usuarios deberían poder ingresar datos privados en sus registros de aprendizaje, por lo que crearemos un sistema para determinar qué datos pertenecen a qué usuario. Luego restringiremos el acceso a ciertas páginas para que los usuarios solo puedan trabajar con sus propios datos.

Modificaremos el modelo de tema para que cada tema pertenezca a un usuario específico. Esto también se encargará de las entradas, porque cada entrada pertenece a un tema específico. Empezaremos restringiendo el acceso a determinadas páginas.

Restringir el acceso con `@login_required`

Django facilita restringir el acceso a ciertas páginas a través del `@login_required`. Se requiere decorador. Recuerde del Capítulo 11 que un decorador es una directiva colocada justo antes de la definición de una función, que modifica cómo se comporta la función. Veamos un ejemplo.

Restringir el acceso a la página de temas

Cada tema será propiedad de un usuario, por lo que sólo los usuarios registrados podrán solicitar la página de temas. Agregue el siguiente código a `learning_logs/views.py`:

```
registros_de_aprendizaje/
vistas.py

desde django.shortcuts importar render, redirigir
desde django.contrib.auth.decorators importar login_required

desde .models importar tema, entrada
--recorte--

@Necesario iniciar sesión
temas def (solicitud):
    """Mostrar todos los temas."""
    --recorte--
```

Primero importamos la función `login_required()` . Aplicamos `login_required()` como decorador de la función de vista de temas() anteponiendo `login_required` con el símbolo `@` . Como resultado, Python sabe ejecutar el código al iniciar sesión. `_required()` antes del código en temas().

El código en `login_required()` comprueba si un usuario ha iniciado sesión y Django ejecuta el código en `issues()` sólo si lo está. Si el usuario no ha iniciado sesión, se le redirige a la página de inicio de sesión.

Para que esta redirección funcione, necesitamos modificar `settings.py` para que Django lo sepa. dónde encontrar la página de inicio de sesión. Agregue lo siguiente al final de `settings.py`:

configuración.py --snip--

```
# Mi configuración.
```

```
LOGIN_REDIRECT_URL = 'registros_de_aprendizaje:índice'
```

```
LOGOUT_REDIRECT_URL = 'registros_de_aprendizaje:índice'
```

```
LOGIN_URL = 'cuentas:iniciar sesión'
```

Ahora, cuando un usuario no autenticado solicita una página protegida por el decorador `@login_required` , Django enviará al usuario a la URL definida por `LOGIN_URL` en `settings.py`.

Puede probar esta configuración cerrando sesión en cualquier cuenta de usuario y accediendo a la página de inicio. Haga clic en el enlace Temas , que debería redireccionarlo a la página de inicio de sesión. Luego inicie sesión en cualquiera de sus cuentas y, desde la página de inicio, haga clic nuevamente en el enlace Temas . Debería poder acceder a la página de temas.

Restringir el acceso a todo el registro de aprendizaje

Django facilita la restricción del acceso a las páginas, pero usted debe decidir qué páginas proteger. Es mejor pensar primero qué páginas no deben estar restringidas y luego restringir todas las demás páginas del proyecto. Puede corregir fácilmente el acceso excesivamente restringido y es menos peligroso que dejar páginas confidenciales sin restricciones.

En Learning Log, mantendremos la página de inicio y la página de registro sin restricciones. Restringiremos el acceso a todas las demás páginas.

Aquí está `learning_logs/views.py` con los decoradores `@login_required` aplicados a todas las vistas excepto `index()`:

registros_de_aprendizaje/

vistas.py

```
--recorte--
```

```
@Necesario iniciar sesión
```

```
temas def (solicitud):
```

```
--recorte--
```

```
@Necesario iniciar sesión
```

```
def tema(solicitud, topic_id):
```

```
--recorte--
```

```
@Necesario iniciar sesión
```

```
def nuevo_tema(solicitud):
```

```
--recorte--
```

```
@Necesario iniciar sesión
def nueva_entrada(solicitud, id_tema):
    --recorte--
```

```
@Necesario iniciar sesión
def editar_entrada(solicitud, entrada_id):
    --recorte--
```

Intente acceder a cada una de estas páginas mientras está desconectado; Debería ser redirigido a la página de inicio de sesión. Tampoco podrá hacer clic en enlaces a páginas como new_topic. Pero si ingresa la URL http://localhost:8000/new_topic/, será redirigido a la página de inicio de sesión. Debe restringir el acceso a cualquier URL que sea de acceso público y esté relacionada con datos privados del usuario.

Conexión de datos a ciertos usuarios

A continuación, debemos conectar los datos con el usuario que los envió. Solo necesitamos conectar los datos más altos de la jerarquía a un usuario, y los datos de nivel inferior seguirán. En Learning Log, los temas son el nivel más alto de datos en la aplicación y todas las entradas están conectadas a un tema. Siempre que cada tema pertenezca a un usuario específico, podemos rastrear la propiedad de cada entrada en la base de datos.

Modificaremos el modelo de tema agregando una relación de clave externa a un usuario. Luego tendremos que migrar la base de datos. Finalmente, modificaremos algunas de las vistas para que solo muestren los datos asociados con el usuario actualmente conectado.

Modificar el modelo de tema

La modificación de models.py consta de solo dos líneas:

```
models.py de django.db importar modelos
desde django.contrib.auth.models importar usuario
```

```
Tema de clase (modelos.Modelo):
    """Un tema sobre el que el usuario está aprendiendo."""
    Texto = modelos.CharField(max_length=200)
    date_added = modelos.DateTimeField(auto_now_add=True)
    propietario = modelos.ForeignKey (Usuario, on_delete = modelos.CASCADE)

    def __str__(yo):
        """Devuelve una cadena que representa el tema."""
        Devolver texto propio
```

```
Entrada de clase (modelos.Modelo):
    --recorte--
```

Importamos el modelo de Usuario desde django.contrib.auth. Luego agregamos un campo de propietario al Tema, que establece una relación de clave externa con el Usuario. modelo. Si se elimina un usuario, también se eliminarán todos los temas asociados con ese usuario.

Identificar usuarios existentes

Cuando migramos la base de datos, Django modificará la base de datos para que pueda almacenar una conexión entre cada tema y un usuario. Para realizar la migración, Django necesita saber qué usuario asociar con cada tema existente. El enfoque más sencillo es comenzar asignando todos los temas existentes a un usuario (por ejemplo, el superusuario). Pero primero necesitamos saber el ID de ese usuario.

Veamos las identificaciones de todos los usuarios creados hasta ahora. Inicie una sesión de shell de Django y ejecute los siguientes comandos:

```
(ll_env)learning_log$ python administrar.py shell
1 >>> de django.contrib.auth.models importar usuario
2 >>> Usuario.objetos.todos()
<QuerySet [<Usuario: ll_admin>, <Usuario: eric>, <Usuario: willie>]>
3 >>> para usuario en User.objects.all():
...     imprimir (usuario.nombre de usuario, usuario.id)
...
ll_admin 1
eric 2
Willy 3
>>>
```

Primero importamos el modelo de usuario a la sesión de shell 1. Luego miramos todos los usuarios que se han creado hasta ahora 2. El resultado muestra tres usuarios para mi versión del proyecto: ll_admin, eric y willie.

A continuación, recorremos la lista de usuarios e imprimimos el nombre de usuario de cada usuario. e ID 3. Cuando Django pregunta con qué usuario asociar los temas existentes, usaremos uno de estos valores de ID.

Migrando la base de datos

Ahora que conocemos los ID, podemos migrar la base de datos. Cuando hacemos esto, Python nos pedirá que conectemos el modelo de tema a un propietario particular temporalmente o que agreguemos un valor predeterminado a nuestro archivo models.py para indicarle qué hacer. Elija la opción 1:

```
1 (ll_env)learning_log$ python administrar.py makemigrations learning_logs
2 Es imposible agregar un campo 'propietario' que no admita valores NULL al tema sin
  especificando un valor predeterminado. Esto es porque...
3 Seleccione una solución:
  1) Proporcione un valor predeterminado único ahora (se establecerá en todas las filas existentes con un
    valor nulo para esta columna)
  2) Salga y defina manualmente un valor predeterminado en models.py.
4 Seleccione una opción: 1
5 Ingrese el valor predeterminado ahora, como Python válido
  Los módulos datetime y django.utils.timezone están disponibles...
  Escriba 'salir' para salir de este mensaje
6 >>> 1

Migraciones para 'learning_logs':
  learning_logs/migrations/0003_topic_owner.py
- Agregar propietario del campo al tema
(ll_env)learning_log$
```

Comenzamos emitiendo el comando `makemigrations` 1. En el resultado, Django indica que estamos intentando agregar un campo obligatorio (no anulable) a un modelo (tema) existente sin ningún valor predeterminado especificado 2. Django nos da dos opciones: podemos proporcionar un valor predeterminado ahora mismo, o podemos salir y agregar un valor predeterminado en `models.py` 3. Aquí elegí la primera opción 4. Luego, Django nos pide que ingresemos el valor predeterminado 5.

Para asociar todos los temas existentes con el usuario administrador original, `ll_admin`, ingrese el ID de usuario 1 6. Puede usar el ID de cualquier usuario que haya creado; No es necesario que sea un superusuario. Luego, Django migra la base de datos usando este valor y genera el archivo de migración `0003_topic_owner.py`, que agrega el propietario del campo al modelo de tema .

Ahora podemos ejecutar la migración. Ingrese lo siguiente en un entorno virtual activo:

```
(ll_env)learning_log$ python administrar.py migrar
```

Operaciones a realizar:

Aplicar todas las migraciones: `admin`, `auth`, `contenttypes`, `learning_logs`, `sesiones`

Ejecutando migraciones:

1 Aplicando `learning_logs.0003_topic_owner...` OK

```
(ll_env)learning_log$
```

Django aplica la nueva migración y el resultado es OK 1.

Podemos verificar que la migración funcionó como se esperaba en una sesión de shell, así:

```
>>> desde learning_logs.models importar tema
```

```
>>> para el tema en Topic.objects.all():
```

```
...     imprimir(tema,tema.propietario)
```

```
...
```

```
Ajoutez ll_admin
```

```
Escalada en roca ll_admin
```

```
>>>
```

Importamos el tema desde `learning_logs.models` y luego recorremos todos los temas existentes, imprimiendo cada tema y el usuario al que pertenece. Puedes ver que cada tema ahora pertenece al usuario `ll_admin`. (Si recibe un error al ejecutar este código, intente salir del shell e iniciar uno nuevo).

NOTA Puede simplemente restablecer la base de datos en lugar de migrar, pero eso perderá todos los datos existentes. Es una buena práctica aprender a migrar una base de datos manteniendo la integridad de los datos de los usuarios. Si desea comenzar con una base de datos nueva, ejecute el comando `python Manage.py Flush` para reconstruir la estructura de la base de datos. Tendrás que crear un nuevo superusuario y todos tus datos desaparecerán.

Restringir el acceso a los temas a los usuarios adecuados

Actualmente, si ha iniciado sesión, podrá ver todos los temas, sin importar con qué usuario haya iniciado sesión. Cambiaremos eso mostrando a los usuarios solo los temas que les pertenecen.

Realice el siguiente cambio en la función topic() en views.py:

```
registros_de_aprendizaje/ --recorte--
vistas.py @Necesario iniciar sesión
temas def (solicitud):
    """Mostrar todos los temas."""
    temas = Tema.objetos.filtro(proprietario=solicitud.usuario).order_by('fecha_añadida')
    contexto = {'temas': temas}
    devolver render(solicitud, 'learning_logs/topics.html', contexto)
--recorte--
```

Cuando un usuario inicia sesión, el objeto de solicitud tiene un atributo request.user establecido, que contiene información sobre el usuario. La consulta Tema.objetos.filter(owner=request.user) le dice a Django que recupere solo los objetos de tema de la base de datos cuyo atributo de propietario coincida con el usuario actual. Como no vamos a cambiar la forma en que se muestran los temas, no necesitamos cambiar la plantilla de la página de temas en absoluto.

Para ver si esto funciona, inicie sesión como el usuario al que conectó todos los temas existentes y vaya a la página de temas. Deberías ver todos los temas. Ahora cierre sesión y vuelva a iniciar sesión como un usuario diferente. Deberías ver el mensaje "Aún no se han agregado temas".

Proteger los temas de un usuario

Aún no hemos restringido el acceso a las páginas de temas, por lo que cualquier usuario registrado podría probar varias URL (como <http://localhost:8000/topics/1/>) y recuperar páginas de temas que coincidan.

Inténtalo tú mismo. Mientras está conectado como el usuario propietario de todos los temas, copie la URL o anote el ID en la URL de un tema y luego cierre la sesión y vuelva a iniciarla como un usuario diferente. Ingrese la URL de ese tema. Debería poder leer las entradas, aunque haya iniciado sesión como un usuario diferente.

Arreglaremos esto ahora realizando una verificación antes de recuperar el pedido entradas en la función de vista topic() :

```
registros_de_aprendizaje/ desde django.shortcuts importar render, redirigir
vistas.py desde django.contrib.auth.decorators importar login_required
1 de django.http importar Http404

--recorte--
@Necesario iniciar sesión
def tema(solicitud, topic_id):
    """Mostrar un solo tema y todas sus entradas."""
    tema = Tema.objetos.get(id=topic_id)
    # Asegúrese de que el tema pertenezca al usuario actual.
    2 si tema.propietario != solicitud.usuario:
        levantar Http404

    entradas = topic.entry_set.order_by('-date_added')
    contexto = {'tema': tema, 'entradas': entradas}
    devolver render(solicitud, 'learning_logs/topic.html', contexto)
--recorte--
```

Una respuesta 404 es una respuesta de error estándar que se devuelve cuando un recurso solicitado no existe en un servidor. Aquí importamos el `Http404` excepción 1, que plantearemos si el usuario solicita un tema al que no debería tener acceso. Después de recibir una solicitud de tema, nos aseguramos de que el usuario del tema coincida con el usuario que ha iniciado sesión actualmente antes de mostrar la página. Si el propietario del tema solicitado no es el mismo que el usuario actual, generamos la excepción `Http404` 2 y Django devuelve una página de error 404.

Ahora, si intentas ver las entradas de temas de otro usuario, verás el mensaje "Página no encontrada" de Django. En el Capítulo 20, configuraremos el proyecto para que los usuarios vean una página de error adecuada en lugar de una página de depuración.

Protegiendo la página `edit_entry`

Las páginas `edit_entry` tienen URL del formato `http://localhost:8000/edit_entry/id_entrada/`, donde `id_entrada` es un número. Protejamos esta página para que nadie pueda usar la URL para obtener acceso a las entradas de otra persona:

```
registros_de_aprendizaje/  --recorte--
vistas.py                @Necesario iniciar sesión
                           def editar_entrada(solicitud, entrada_id):
                               """Editar una entrada existente."""
                               entrada = Entrada.objects.get(id=entrada_id)
                               tema = entrada.tema
                               si tema.propietario != solicitud.usuario:
                                   levantar Http404

                               si solicitud.método != 'POST':
                                   --recorte--
```

Recuperamos la entrada y el tema asociado a esta entrada. Luego comprobamos si el propietario del tema coincide con el usuario que ha iniciado sesión actualmente; si no coinciden, generamos una excepción `Http404`.

Asociar nuevos temas con el usuario actual

Actualmente, la página para agregar nuevos temas está rota porque no asocia nuevos temas con ningún usuario en particular. Si intenta agregar un tema nuevo, verá el mensaje `IntegrityError` junto con la restricción `NOT NULL` fallida: `learning_logs_topic.owner_id`. Django dice que no se puede crear un tema nuevo sin especificar un valor para el campo propietario del tema.

Hay una solución sencilla para este problema, porque tenemos acceso al usuario actual a través del objeto de solicitud. Agregue el siguiente código, que asocia el nuevo tema con el usuario actual:

```
registros_de_aprendizaje/  --recorte--
vistas.py                @Necesario iniciar sesión
                           def nuevo_tema(solicitud):
                               --recorte--
                               demás:
                                   # datos POST enviados; procesar datos.
                                   formulario = TemaForm(datos=solicitud.POST)
                                   si form.is_valid():
```

```

1         nuevo_tema = formulario.guardar(confirmar=False)
2         nuevo_tema.propietario = solicitud.usuario
3         nuevo_tema.guardar()

        devolver redirección ('learning_logs: temas')

# Mostrar un formulario en blanco o no válido.
contexto = {'formulario': formulario}
devolver render(solicitud, 'learning_logs/new_topic.html', contexto)
--recorte--

```

Cuando llamamos por primera vez a `form.save()`, pasamos el argumento `commit=False` porque necesitamos modificar el nuevo tema antes de guardarlo en la base de datos 1. Luego configuramos el atributo de propietario del nuevo tema para el usuario actual 2. Finalmente, llamar a `guardar()` en la instancia del tema que acabamos de definir 3. Ahora el tema tiene todos los datos necesarios y se guardará correctamente.

Debería poder agregar tantos temas nuevos como desee para tantos usuarios diferentes como desee. Cada usuario solo tendrá acceso a sus propios datos, ya sea que esté viendo datos, ingresando datos nuevos o modificando datos antiguos.

INTÉNTALO TÚ MISMO

19-3. Refactorización: hay dos lugares en `views.py` donde nos aseguramos de que el usuario asociado con un tema coincida con el usuario actualmente conectado. Coloque el código para esta verificación en una función llamada `check_topic_owner()` y llame a esta función cuando corresponda.

19-4. Protección de `new_entry`: actualmente, un usuario puede agregar una nueva entrada al registro de aprendizaje de otro usuario ingresando una URL con el ID de un tema que pertenece a otro usuario. Evite este ataque verificando que el usuario actual sea el propietario del tema de la entrada antes de guardar la nueva entrada.

19-5. Blog protegido: en su proyecto de blog, asegúrese de que cada publicación del blog esté conectada a un usuario en particular. Asegúrese de que todas las publicaciones sean de acceso público, pero que solo los usuarios registrados puedan agregar publicaciones y editar publicaciones existentes. En la vista que permite a los usuarios editar sus publicaciones, asegúrese de que el usuario esté editando su propia publicación antes de procesar el formulario.

Resumen

En este capítulo, aprendió cómo los formularios permiten a los usuarios agregar nuevos temas y entradas, y editar entradas existentes. Luego aprendió cómo implementar cuentas de usuario. Les di a los usuarios existentes la posibilidad de iniciar y cerrar sesión, y usaste el `UserCreationForm` predeterminado de Django para permitir que las personas crearan nuevas cuentas.

Después de crear un sistema simple de autenticación y registro de usuarios, restringió el acceso a los usuarios que iniciaron sesión para ciertas páginas usando `@login_required` . decorador. Luego asignó datos a usuarios específicos a través de una relación de clave externa. También aprendió a migrar la base de datos cuando la migración requiere que especifique algunos datos predeterminados.

Finalmente, aprendió cómo asegurarse de que un usuario solo pueda ver los datos que le pertenecen modificando las funciones de visualización. Recuperaste datos apropiados usando el método `filter()` y comparaste al propietario de los datos solicitados con el usuario actualmente conectado.

Puede que no siempre sea inmediatamente obvio qué datos debe poner a disposición y qué datos debe proteger, pero esta habilidad se adquirirá con la práctica. Las decisiones que hemos tomado en este capítulo para proteger los datos de nuestros usuarios también ilustran por qué trabajar con otras personas es una buena idea al crear un proyecto: hacer que otra persona revise su proyecto hace que sea más probable que detecte áreas vulnerables. .

Ahora tiene un proyecto en pleno funcionamiento ejecutándose en su máquina local. En el capítulo final, diseñará el Registro de aprendizaje para que sea visualmente atractivo e implementará el proyecto en un servidor para que cualquier persona con acceso a Internet pueda registrarse y crear una cuenta.

20

DISEÑAR E IMPLEMENTAR

UNA APLICACIÓN



Learning Log ahora es completamente funcional, pero no tiene estilo y solo se ejecuta en su máquina local. En este capítulo, diseñará el proyecto de una manera simple pero profesional y luego lo implementará en un servidor activo para que cualquier persona en el mundo pueda crear una cuenta y usarlo.

Para el estilo, usaremos la biblioteca Bootstrap , una colección de herramientas para diseñar aplicaciones web para que luzcan profesionales en todos los dispositivos modernos, desde un teléfono pequeño hasta un monitor de escritorio grande. Para hacer esto, usaremos la aplicación django-bootstrap5, que también te permitirá practicar el uso de aplicaciones creadas por otros desarrolladores de Django.

Implementaremos Learning Log utilizando Platform.sh, un sitio que le permite enviar su proyecto a uno de sus servidores, poniéndolo a disposición de cualquier persona con conexión a Internet. También comenzaremos a usar un sistema de control de versiones llamado Git para rastrear los cambios en el proyecto.

Cuando haya terminado con Learning Log, podrá desarrollar aplicaciones web simples, darles una apariencia profesional e implementarlas en un servidor en vivo. También podrá utilizar recursos de aprendizaje más avanzados a medida que desarrolle sus habilidades.

Registro de aprendizaje de estilo

Hemos ignorado deliberadamente el estilo hasta ahora para centrarnos primero en la funcionalidad de Learning Log. Esta es una buena manera de abordar el desarrollo, porque una aplicación sólo es útil si funciona. Una vez que una aplicación está funcionando, su apariencia es fundamental para que la gente quiera usarla.

En esta sección, instalaremos la aplicación django-bootstrap5 y la agregaremos al proyecto. Luego lo usaremos para diseñar las páginas individuales del proyecto, de modo que todas las páginas tengan una apariencia consistente.

La aplicación django-bootstrap5

Usaremos django-bootstrap5 para integrar Bootstrap en nuestro proyecto. Esta aplicación descarga los archivos Bootstrap necesarios, los coloca en una ubicación adecuada de su proyecto y hace que las directivas de estilo estén disponibles en las plantillas de su proyecto.

Para instalar django-bootstrap5, ejecute el siguiente comando en un entorno virtual activo:

```
(ll_env)learning_log$ pip instalar django-bootstrap5
--recorte--
Beautifulsoup4-4.11.1 instalado con éxito django-bootstrap5-21.3
tamiz de sopa-2.3.2.post1
```

A continuación, debemos agregar django-bootstrap5 a INSTALLED_APPS en settings.py:

```
configuración.py --snip--
INSTALLED_APPS = [
    # Mis aplicaciones.
    'registros_aprendizaje',
    'cuentas',

    # Aplicaciones de terceros.
    'django_bootstrap5',

    # Aplicaciones Django predeterminadas.
    'django.contrib.admin',
    --recorte--
```

Inicie una nueva sección llamada Aplicaciones de terceros, para aplicaciones creadas por otros desarrolladores, y agregue 'django_bootstrap5' a esta sección. Asegúrate de colocar esta sección después de Mis aplicaciones pero antes de la sección que contiene las aplicaciones predeterminadas de Django.

Uso de Bootstrap para diseñar el registro de aprendizaje

Bootstrap es una gran colección de herramientas de diseño. También tiene una serie de plantillas que puede aplicar a su proyecto para crear un estilo general. Es mucho más fácil utilizar estas plantillas que utilizar herramientas de diseño individuales. Ver el

plantillas que Bootstrap ofrece, ingresa a <https://getbootstrap.com> y haga clic en Ejemplos. Usaremos la plantilla estática Navbar , que proporciona una barra de navegación superior simple y un contenedor para el contenido de la página.

La Figura 20-1 muestra cómo se verá la página de inicio después de aplicar Plantilla de Bootstrap a base.html y modifique ligeramente index.html .

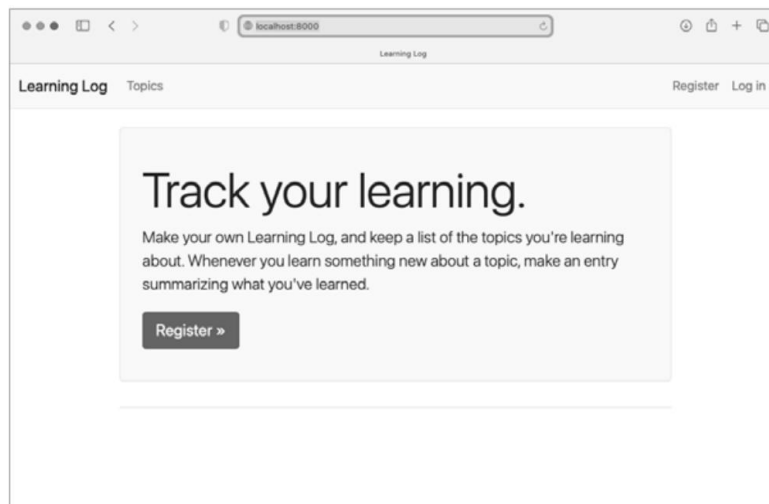


Figura 20-1: La página de inicio del Registro de aprendizaje usando Bootstrap

Modificando base.html

Necesitamos reescribir base.html usando la plantilla Bootstrap. Desarrollaremos el nuevo base.html en secciones. Este es un archivo grande; es posible que desee copiar este archivo de los recursos en línea, disponibles en https://ehmatthes.github.io/pcc_3e. Si copia el archivo, aún debe leer la siguiente sección para comprender los cambios que se realizaron.

Definición de los encabezados HTML

El primer cambio que haremos en base.html define los encabezados HTML del archivo. También agregaremos algunos requisitos para usar Bootstrap en nuestras plantillas y le daremos un título a la página. Elimina todo en base.html y reemplázalo con el siguiente código:

```
base.html 1 <!tipo de documento html>
           2 <html lang="es">
           3 <cabeza>
               <meta juego de caracteres="utf-8">
               <meta nombre="viewport" content="ancho=ancho-dispositivo, escala-inicial=1">
           4 <title>Registro de aprendizaje</title>

           5 {% carga django_bootstrap5 %}
               {% bootstrap_css %}
               {% bootstrap_javascript %}

           </cabeza>
```

Primero declaramos este archivo como un documento HTML 1 escrito en inglés 2.

Un archivo HTML se divide en dos partes principales: el encabezado y el cuerpo. El encabezado del archivo comienza con una etiqueta `<head>` de apertura 3. El encabezado de un archivo HTML no contiene nada del contenido de la página; simplemente le dice al navegador lo que necesita saber para mostrar la página correctamente. Incluimos un elemento `<title>` para la página, que se mostrará en la barra de título del navegador cada vez que se abra el Registro de aprendizaje 4.

Antes de cerrar la sección principal, cargamos la colección de etiquetas de plantilla disponibles en django-bootstrap5 5. La etiqueta de plantilla `{% bootstrap_css %}` es una etiqueta personalizada de django-bootstrap5; carga todos los archivos CSS necesarios para implementar estilos Bootstrap. La etiqueta que sigue habilita todo el comportamiento interactivo que podría utilizar en una página, como barras de navegación plegables. La etiqueta de cierre `</head>` aparece en la última línea.

Todas las opciones de estilo de Bootstrap ahora están disponibles en cualquier plantilla que herede de `base.html`. Si desea utilizar etiquetas de plantilla personalizadas de django-bootstrap5, cada plantilla deberá incluir `{% load django`
Etiqueta `_bootstrap5 %}` .

Definición de la barra de navegación

El código que define la barra de navegación en la parte superior de la página es bastante largo, porque tiene que funcionar igualmente bien en pantallas de teléfono estrechas y monitores de escritorio anchos. Trabajaremos en la barra de navegación en secciones.

Aquí está la primera parte de la barra de navegación:

```
base.html --recorte--
</cabeza>
<cuerpo>

1 <nav class="barra de navegación barra de navegación-expandir-md barra de navegación-luz bg-luz mb-4 borde">
  <div class="contenedor-fluido">
2    <a class="navbar-brand" href="{% url 'learning_logs:index' %}">
      Registro de aprendizaje</a>
3    <botón clase="navbar-toggler" tipo="botón" data-bs-toggle="colapso"
      data-bs-target="#navbarCollapse" aria-controls="navbarCollapse"
      aria-expanded="false" aria-label="Alternar navegación">
      <span class="navbar-toggler-icon">
</botón>
4    <div class="contraer barra de navegación-collapse" id="barra de navegaciónCollapse">
5      <ul class="navbar-nav me-auto mb-2 mb-md-0">
6        <li class="nav-item">
7          <a class="nav-link" href="{% url 'learning_logs:topics' %}">
            Temas</a></li>
          </ul> <!-- Fin de los enlaces en el lado izquierdo de la barra de navegación -->
        </div> <!-- Cierra las partes plegables de la barra de navegación -->
      </div> <!-- Cierra el contenedor de la barra de navegación -->
    </nav> <!-- Fin de la barra de navegación -->
```

8 {% contenido de bloque %}{% contenido de bloque final %}

```
</cuerpo>  
</html>
```

El primer elemento nuevo es la etiqueta `<body>` de apertura. El cuerpo de un archivo HTML contiene el contenido que los usuarios verán en una página. A continuación tenemos un elemento `<nav>`, que abre el código de la barra de navegación en la parte superior de la página 1. Todo lo contenido en este elemento tiene un estilo de acuerdo con las reglas de estilo Bootstrap definidas por los selectores `navbar`, `navbar-expand-md` y el resto que ves aquí. Un selector determina a qué elementos de una página se aplica una determinada regla de estilo. Los selectores `navbar-light` y `bg-light` diseñan la barra de navegación con un fondo de temática clara. El `mb` en `mb-4` es la abreviatura de `margin-bottom`; este selector asegura que aparezca un pequeño espacio entre la barra de navegación y el resto de la página. El selector de bordes proporciona un borde fino alrededor del fondo claro para diferenciarlo un poco del resto de la página.

La etiqueta `<div>` en la siguiente línea abre un contenedor de tamaño variable que contendrá la barra de navegación general. El término `div` es la abreviatura de división; una página web se construye dividiéndola en secciones y definiendo reglas de estilo y comportamiento que se aplican a esa sección. Cualquier regla de estilo o comportamiento definida en una etiqueta `<div>` de apertura afecta todo lo que ve hasta su etiqueta de cierre correspondiente, escrita como `</div>`.

A continuación configuramos el nombre del proyecto, Registro de aprendizaje, para que aparezca como el primer elemento en la barra de navegación 2. Esto también servirá como enlace a la página de inicio, tal como lo ha estado haciendo en la versión minimalista del proyecto que hemos creado. construido en los dos capítulos anteriores. El selector de marca de la barra de navegación le da estilo a este enlace para que se destaque del resto de los enlaces y ayude a agregar algo de marca al sitio.

La plantilla Bootstrap luego define un botón que aparece si el navegador La ventana es demasiado estrecha para mostrar toda la barra de navegación horizontalmente 3. Cuando el usuario hace clic en el botón, los elementos de navegación aparecen en una lista desplegable. La referencia de colapso hace que la barra de navegación se colapse cuando el usuario reduce la ventana del navegador o cuando el sitio se muestra en dispositivos con pantallas pequeñas.

A continuación, abrimos una nueva sección (`<div>`) de la barra de navegación 4. Esta es la parte de la barra de navegación que puede colapsar dependiendo del tamaño de la ventana del navegador.

Bootstrap define los elementos de navegación como elementos en una lista desordenada 5, con reglas de estilo que hacen que no parezca una lista. Cada enlace o elemento que necesite en la barra se puede incluir como un elemento en una lista desordenada 6. Aquí, el único elemento en la lista es nuestro enlace a la página de temas 7. Observe la etiqueta de cierre `</li>` al final del enlace; cada etiqueta de apertura necesita una etiqueta de cierre correspondiente.

El resto de las líneas que se muestran aquí cierran todas las etiquetas que se han abrió. En HTML, un comentario se escribe así:

```
<!-- Este es un comentario HTML. -->
```

Las etiquetas de cierre normalmente no tienen comentarios, pero si eres nuevo en HTML, puede ser muy útil etiquetar algunas de tus etiquetas de cierre. Una sola etiqueta faltante o una etiqueta adicional pueden alterar el diseño de una página completa. Incluimos el bloque de contenido 8 y también las etiquetas de cierre `</body>` y `</html>`.

No hemos terminado con la barra de navegación, pero ahora tenemos un documento HTML completo. Si el servidor de ejecución está actualmente activo, detenga el servidor actual y reinicielo. Vaya a la página de inicio del proyecto y debería ver una barra de navegación que tiene algunos de los elementos que se muestran en la Figura 20-1. Ahora agreguemos el resto de los elementos a la barra de navegación.

Agregar enlaces de cuentas de usuario

Todavía necesitamos agregar los enlaces asociados con las cuentas de usuario. Comenzaremos agregando todos los enlaces relacionados con la cuenta excepto el formulario de cierre de sesión.

Realice los siguientes cambios en `base.html`:

`base.html`

```
--recorte--
</ul> <!-- Fin de los enlaces en el lado izquierdo de la barra de navegación -->

<!-- Enlaces relacionados con la cuenta -->
1 <ul class="navbar-nav ms-auto mb-2 mb-md-0">
2     {% si usuario.is_authenticated %}
3         <li class="nav-item">
4             <span class="navbar-text me-2">Hola, {{ usuario.nombre de usuario }}.
                </li>
            {% demás %}
            <li class="nav-item">
                <a class="nav-link" href="{% url 'cuentas:registro' %}">
                    Regístrate</a></li>
            <li class="nav-item">
                <a class="nav-link" href="{% url 'cuentas: iniciar sesión' %}">
                    Iniciar sesión</a></li>
            {% terminara si %}

        </ul> <!-- Fin de los enlaces relacionados con la cuenta -->

</div> <!-- Cierra las partes plegables de la barra de navegación -->
--recorte--
```

Comenzamos un nuevo conjunto de enlaces usando otra etiqueta de apertura `<ul>` 1. Usted Puede tener tantos grupos de enlaces como necesites en una página. El selector `ms-auto` es la abreviatura de `margin-start-automatic`: este selector examina los otros elementos en la barra de navegación y elabora un margen izquierdo (inicio) que empuja este grupo de enlaces al lado derecho de la ventana del navegador.

El bloque `if` es el mismo bloque condicional que usamos anteriormente para mostrar mensajes apropiados a los usuarios, dependiendo de si han iniciado sesión 2.

El bloque ahora es un poco más largo porque hay algunas reglas de estilo dentro de las etiquetas condicionales. El saludo para usuarios autenticados está envuelto en un elemento `<span>` 3. Un elemento `span` da estilo a fragmentos de texto o elementos de una página que forman parte de una línea más larga. Mientras que los elementos `div` crean sus propias divisiones en una página, los elementos `span` son continuos dentro de una sección más grande. Esto puede

Puede resultar confuso al principio, porque muchas páginas tienen elementos div profundamente anidados. Aquí, utilizamos el elemento span para aplicar estilo al texto informativo en la barra de navegación: en este caso, el nombre del usuario que inició sesión.

En el bloque else , que se ejecuta para usuarios no autenticados, incluimos los enlaces para registrar una nueva cuenta e iniciar sesión 4. Estos deberían verse igual que el enlace a la página de temas.

Si quisiera agregar más enlaces a la barra de navegación, agregaría otro elemento a uno de los grupos que hemos definido, usando directivas de estilo como las que ha visto aquí.

Ahora agreguemos el formulario de cierre de sesión a la barra de navegación.

Agregar el formulario de cierre de sesión a la barra de navegación

Cuando escribimos por primera vez el formulario de cierre de sesión, lo agregamos al final de base.html. Ahora pongámoslo en un lugar mejor, en la barra de navegación:

base.html

```
--recorte--
</ul> <!-- Fin de los enlaces relacionados con la cuenta -->

{% si usuario.is_authenticated %}
  <formulario action="{% url 'cuentas: cerrar sesión' %}" método='publicar'>
    {% csrf_token %}
    <nombre del botón='enviar' clase='btn btn-outline-secundario btn-sm'>
      Cerrar sesión</botón>
    </formulario>
  {% terminara si %}

</div> <!-- Cierra las partes plegables de la barra de navegación -->
--recorte--
```

El formulario de cierre de sesión debe colocarse después del conjunto de enlaces relacionados con la cuenta, pero dentro de la sección plegable de la barra de navegación. El único cambio en el formulario es la adición de varias clases de estilo Bootstrap en el <botón> elemento, que aplica elementos de estilo Bootstrap al botón de cierre de sesión 1.

Vuelva a cargar la página de inicio y debería poder iniciar y cerrar sesión usando cualquiera de las cuentas que haya creado.

Todavía hay un poco más que debemos agregar a base.html. Necesitamos definir dos bloques que las páginas individuales pueden usar para colocar el contenido específico de esas páginas.

Definición de la parte principal de la página

El resto de base.html contiene la parte principal de la página:

base.html

```
--recorte--
</nav> <!-- Fin de la barra de navegación -->

1 <clase principal="contenedor">
2 <div class="pb-2 mb-2 borde-inferior">
  {% bloquear page_header %}{% endblock page_header %}
</div>
```

```

3 <div>
    {% bloquear contenido %}{% bloquear contenido %}
  </div>
</principal>

</cuerpo>
</html>

```

Primero abrimos una etiqueta `<main>` 1. El elemento principal se utiliza para la parte más importante del cuerpo de una página. Aquí asignamos el contenedor del selector de arranque, que es una forma sencilla de agrupar elementos en una página. Colocaremos dos elementos `div` en este contenedor.

El primer elemento `div` contiene un bloque `page_header` 2. Usaremos este bloque para titular la mayoría de las páginas. Para que esta sección se destaque del resto de la página, colocamos algo de relleno debajo del encabezado. El relleno se refiere al espacio entre el contenido de un elemento y su borde. El selector `pb-2` es una directiva de arranque que proporciona una cantidad moderada de relleno en la parte inferior del elemento con estilo. Un margen es el espacio entre el borde de un elemento y otros elementos de la página. El selector `mb-2` proporciona una cantidad moderada de margen en la parte inferior de este `div`. Queremos un borde en la parte inferior de este bloque, por lo que usamos el selector `border-bottom`, que proporciona un borde delgado en la parte inferior del bloque `page_header`.

Luego definimos un elemento `div` más que contiene el contenido del bloque 3.

No aplicamos ningún estilo específico a este bloque, por lo que podemos diseñar el contenido de cualquier página como mejor nos parezca. El final del archivo `base.html` tiene etiquetas de cierre para los elementos principal, cuerpo y `html`.

Cuando carga la página de inicio de Learning Log en un navegador, debería ver una barra de navegación de aspecto profesional que coincide con la que se muestra en la Figura 20-1. Intente cambiar el tamaño de la ventana para que sea muy estrecha; un botón debería reemplazar la barra de navegación. Haga clic en el botón y todos los enlaces deberían aparecer en una lista desplegable.

Diseñar la página de inicio usando un Jumbotron

Para actualizar la página de inicio, usaremos un elemento Bootstrap llamado `jumbotron`, un cuadro grande que se destaca del resto de la página. Normalmente, se utiliza en las páginas de inicio para contener una breve descripción del proyecto general y un llamado a la acción que invita al espectador a involucrarse.

Aquí está el archivo `index.html` revisado :

```
index.html {% extiende "learning_logs/base.html" %}
```

```

1 {% bloquear page_header %}
2 <div class="p-3 mb-4 bg-borde claro redondeado-3">
    <div class="contenedor-fluido py-4">
3       <h1 class="display-3">Haz un seguimiento de tu aprendizaje.</h1>
4       <p class="lead">Crea tu propio Registro de aprendizaje y mantén una lista de los
        temas sobre los que estás aprendiendo. Siempre que aprendes algo nuevo
        sobre un tema, haz una entrada que resuma lo que has aprendido.</p>

```

```

5      <a class="btn btn-primario btn-lg mt-1"
      href="{% url 'accounts:register' %}">Registrarse &raquo;</a>
    </div>
  </div>
{% bloque final page_header %}

```

Primero le decimos a Django que estamos a punto de definir lo que va en la página .
 _bloque de encabezado 1. Un jumbotron se implementa como un par de elementos div con un conjunto de directivas de estilo aplicadas a ellos 2. El div exterior tiene configuraciones de margen y relleno, un color de fondo claro y esquinas redondeadas. El div interno es un contenedor que cambia junto con el tamaño de la ventana y también tiene algo de relleno. El selector py-4 agrega relleno en la parte superior e inferior del elemento div. Siéntase libre de ajustar los números en estas configuraciones y ver cómo cambia la página de inicio.

Dentro del jumbotron hay tres elementos. El primero es un mensaje corto, Realice un seguimiento de su aprendizaje, que brinda a los nuevos visitantes una idea de lo que hace el Registro de aprendizaje 3. El elemento <h1> es un encabezado de primer nivel y el selector de visualización 3 le agrega una apariencia más delgada y alta. cabecera determinada. También incluimos un mensaje más largo que proporciona más información sobre lo que el usuario puede hacer con su registro de aprendizaje 4. Tiene el formato de un párrafo inicial , que debe destacarse de los párrafos normales.

En lugar de simplemente usar un enlace de texto, creamos un botón que invita a los usuarios a registrar una cuenta en Learning Log 5. Este es el mismo enlace que en el encabezado, pero el botón se destaca en la página y muestra al espectador lo que necesita. hacer para comenzar a utilizar el proyecto. Los selectores que ve aquí lo diseñan como un botón grande que representa una llamada a la acción. El código » es una entidad HTML que parece dos corchetes angulares combinados (>>). Finalmente, proporcionamos etiquetas div de cierre y cerramos el bloque page_header .
 Con sólo dos elementos div en este archivo, no es particularmente útil etiquetar las etiquetas div de cierre. No agregaremos nada más a esta página, por lo que no necesitamos definir el bloque de contenido en esta plantilla.

La página de inicio ahora se parece a la Figura 20-1. ¡Esta es una mejora significativa con respecto a la versión sin estilo del proyecto!

Aplicar estilo a la página de inicio de sesión

Hemos refinado la apariencia general de la página de inicio de sesión, pero el formulario de inicio de sesión en sí aún no tiene ningún estilo. Hagamos que el formulario parezca coherente con el resto de la página modificando login.html:

```

login.html {% extiende 'learning_logs/base.html' %}
1 {% carga django_bootstrap5 %}

2 {% bloquear page_header %}
  <h2>Inicie sesión en su cuenta.</h2>
  {% bloque final page_header %}

  {% bloquear contenido %}

```

```
<formulario action="{% url 'cuentas:iniciar sesión' %}" método='publicar'>
    {% csrf_token %}
3 {% formulario bootstrap_form %}
4 {% bootstrap_button button_type="enviar" contenido="Iniciar sesión" %}
</formulario>

{% contenido del bloque final %}
```

Primero cargamos las etiquetas de la plantilla bootstrap5 en esta plantilla 1. Luego definimos el bloque `page_header`, que le dice al usuario para qué es la página 2. Observe que hemos eliminado el bloque `{% if form.errors %}` de la plantilla; django-bootstrap5 gestiona los errores de formulario automáticamente.

Para mostrar el formulario, usamos la etiqueta de plantilla `{% bootstrap_form %}` 3; este reemplaza el elemento `{{ form.as_div }}` que estábamos usando en el Capítulo 19. La etiqueta de plantilla `{% bootstrap_form %}` inserta reglas de estilo Bootstrap en los elementos individuales del formulario a medida que se representa el formulario. Para generar el botón de envío, usamos la etiqueta `{% bootstrap_button %}` con argumentos que lo designan como un botón de envío y le damos la etiqueta Iniciar sesión 4.

La Figura 20-2 muestra el formulario de inicio de sesión ahora. La página es mucho más limpia, con un estilo consistente y un propósito claro. Intente iniciar sesión con un nombre de usuario o contraseña incorrectos; Verá que incluso los mensajes de error tienen un estilo coherente y se integran bien con el sitio en general.

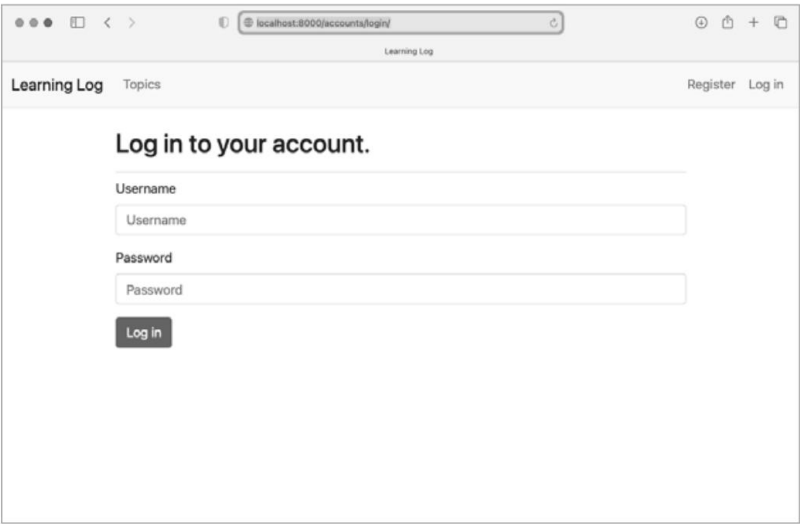


Figura 20-2: La página de inicio de sesión diseñada con Bootstrap

Aplicar estilo a la página de temas

Asegurémonos de que las páginas para ver información también tengan el estilo adecuado, comenzando con la página de temas:

```
temas.html {% extiende 'learning_logs/base.html' %}

{% bloquear page_header %}
```

```

1 <h1>Temas</h1>
  {% bloque final page_header %}

  {% bloquear contenido %}

2 <ul class="lista-grupo borde-fondo pb-2 mb-4">
  {% para tema en temas %}
3     <li class="lista-grupo-elemento-borde-0">
      <a href="{% url 'learning_logs:topic' topic.id %}">
        {{ tema.texto }}</a>
      </li>
  {% vacío %}
4     <li class="list-group-item border-0">Aún no se han agregado temas.</li>
  {% final de %}
</ul>

<a href="{% url 'learning_logs:new_topic' %}">Agregar un nuevo tema</a>

{% contenido del bloque final %}

```

No necesitamos la etiqueta `{% load bootstrap5 %}`, porque no estamos usando ninguna etiquetas de plantilla bootstrap5 personalizadas en este archivo. Movemos el encabezado Temas en el bloque `page_header` y conviértalo en un elemento `<h1>` en lugar de un simple párrafo 1.

El contenido principal de esta página es una lista de temas, por lo que utilizamos el componente de grupo de listas de Bootstrap para representar la página. Esto aplica un conjunto simple de directivas de estilo a la lista general y a cada elemento de la lista. Cuando abrimos la etiqueta `<ul>`, primero incluimos la clase `list-group` para aplicar las directivas de estilo predeterminadas a la lista 2. Personalizamos aún más la lista colocando un borde al final de la lista, un poco de relleno debajo la lista (`pb-2`) y un margen debajo del borde inferior (`mb-4`).

Cada elemento de la lista necesita la clase `list-group-item`, y personalizamos el estilo predeterminado eliminando el borde alrededor de los elementos individuales 3. El mensaje que se muestra cuando la lista está vacía necesita estas mismas clases 4.

Cuando visite la página de temas ahora, debería ver una página con un estilo que coincida con la página de inicio.

Aplicar estilo a las entradas en la página del tema

En la página del tema, usaremos el componente de tarjeta de Bootstrap para que cada entrada se destaque. Una tarjeta es un conjunto anidable de divs con estilos flexibles y predefinidos que son perfectos para mostrar las entradas de un tema:

```
topic.html {% extiende 'learning_logs/base.html' %}
```

```

1 {% bloquear page_header %}
  <h1>{{ tema.texto }}</h1>
  {% bloque final page_header %}

```

```

{% bloquear contenido %}
<p>
  <a href="{% url 'learning_logs:new_entry' topic.id %}">Agregar nueva entrada</a>
</p>

{% para entrada en entradas %}
2 <div class="tarjeta mb-3">
    <!-- Encabezado de tarjeta con marca de tiempo y enlace de edición -->
3    <h4 class="encabezado-de-tarjeta">
        {{ entrada.date_added|fecha:'M d, YH:i' }}
4    <small><a href="{% url 'learning_logs:edit_entry' entrada.id %}">
        editar entrada</a></small>
    </h4>
    <!-- Cuerpo de la tarjeta con texto de entrada -->
5    <div class="card-body">{{ entrada.text|saltos de línea }}</div>
  </div>
  {% vacío %}
6 <p>Aún no hay entradas para este tema.</p>
  {% final de %}

{% contenido del bloque final %}

```

Primero colocamos el tema en el bloque 1 de encabezado de página. Luego eliminamos la estructura de lista desordenada utilizada anteriormente en esta plantilla. En lugar de convertir cada entrada en un elemento de lista, abrimos un elemento div con la tarjeta selectora 2. Esta tarjeta tiene dos elementos anidados: uno para contener la marca de tiempo y el enlace para editar la entrada, y otro para contener el cuerpo de la entrada. El selector de tarjetas se encarga de la mayor parte del estilo que necesitamos para este div; Personalizamos la tarjeta añadiendo un pequeño margen en la parte inferior de cada tarjeta (mb-3).

El primer elemento de la tarjeta es un encabezado, que es un elemento `<h4>` con el encabezado de la tarjeta selectora 3. Este encabezado contiene la fecha en que se realizó la entrada y un enlace para editar la entrada. La etiqueta `<small>` alrededor del enlace `edit_entry` hace que parezca un poco más pequeño que la marca de tiempo 4. El segundo elemento es un div con el cuerpo de la tarjeta selectora 5, que coloca el texto de la entrada en un cuadro simple en la tarjeta. Observe que el código de Django para incluir la información en la página no ha cambiado; sólo tienen elementos que afectan la apariencia de la página. Como ya no tenemos una lista desordenada, reemplazamos las etiquetas de elementos de la lista alrededor del mensaje de la lista vacía con etiquetas de párrafo simples 6.

La Figura 20-3 muestra la página del tema con su nueva apariencia. La funcionalidad de Learning Log no ha cambiado, pero parece mucho más profesional y atractiva para los usuarios.

Si desea utilizar una plantilla Bootstrap diferente para un proyecto, siga un proceso similar al que hemos hecho hasta ahora en este capítulo. Copie la plantilla que desea utilizar en `base.html` y modifique los elementos que contienen contenido real para que la plantilla muestre la información de su proyecto. Luego use las herramientas de estilo individuales de Bootstrap para diseñar el contenido de cada página.

NOTA El proyecto Bootstrap tiene una documentación excelente. Visite la página de inicio en <https://getbootstrap.com> y haga clic en Documentos para obtener más información sobre lo que ofrece Bootstrap.

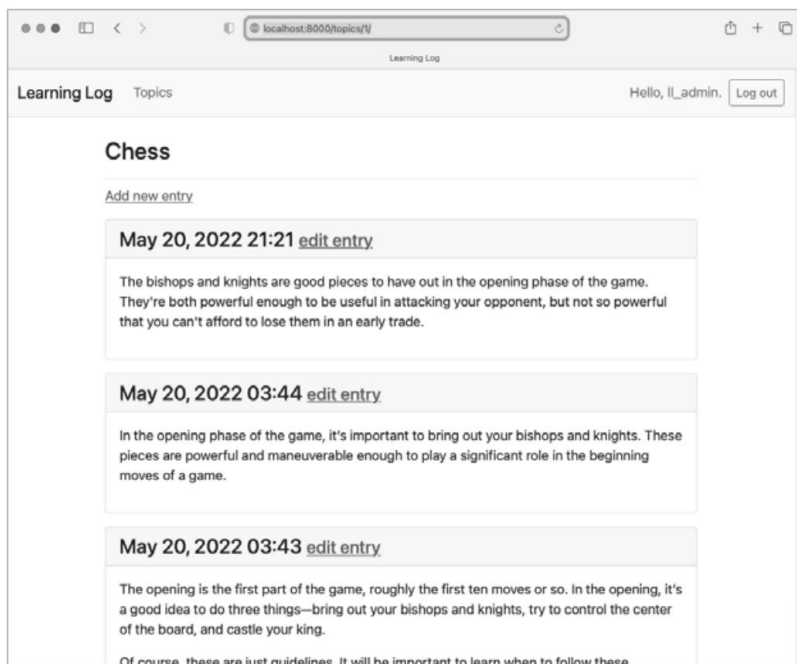


Figura 20-3: La página del tema con estilo Bootstrap

INTÉNTALO TÚ MISMO

20-1. Otras formas: aplicamos los estilos de Bootstrap a la página de inicio de sesión. Realice cambios similares en el resto de las páginas basadas en formularios, incluidas `new_topic`, `new_entry`, `edit_entry` y registre.

20-2. Blog elegante: use Bootstrap para diseñar el proyecto de blog que creó en el Capítulo 19.

Implementación del registro de aprendizaje

Ahora que tenemos un proyecto de aspecto profesional, implementémoslo en un servidor activo para que cualquiera con conexión a Internet pueda usarlo. Usaremos Platform.sh, una plataforma basada en web que le permite administrar la implementación de aplicaciones web. Pondremos Learning Log en funcionamiento en Platform.sh.

Crear una cuenta en Platform.sh

Para crear una cuenta, vaya a <https://platform.sh> y haga clic en el botón Prueba gratuita. Platform.sh tiene un nivel gratuito que, al momento de escribir este artículo, no requiere una tarjeta de crédito. El período de prueba le permite implementar una aplicación con recursos mínimos, lo que le permite probar su proyecto en una implementación en vivo antes de comprometerse con un plan de alojamiento pago.

NOTA Los límites específicos de los planes de prueba tienden a cambiar periódicamente, ya que las plataformas de alojamiento luchan contra el spam y el abuso de recursos. Puede ver los límites actuales de la prueba gratuita en <https://platform.sh/free-trial>.

Instalación de la CLI de Platform.sh

Para implementar y administrar un proyecto en Platform.sh, necesitará las herramientas disponibles en la interfaz de línea de comandos (CLI). Para instalar la última versión de la CLI, visite <https://docs.platform.sh/development/cli.html> y siga las instrucciones de su sistema operativo.

En la mayoría de los sistemas, puede instalar la CLI ejecutando el siguiente comando en una terminal:

```
$ curl -fsS https://platform.sh/cli/installer | PHP
```

Una vez que este comando haya terminado de ejecutarse, deberá abrir una nueva ventana de terminal antes de poder usar la CLI.

NOTA Este comando probablemente no funcionará en una terminal estándar de Windows. Puede utilizar el Subsistema de Windows para Linux (WSL) o una terminal Git Bash. Si necesita instalar PHP, puede utilizar el instalador XAMPP de <https://apacheFriends.org>. Si tiene alguna dificultad para instalar la CLI de Platform.sh, consulte las instrucciones de instalación más detalladas en el Apéndice E.

Instalación de platformashconfig

También necesitarás instalar un paquete adicional, platformashconfig. Este paquete ayuda a detectar si el proyecto se está ejecutando en su sistema local o en un servidor Platform.sh. En un entorno virtual activo, emita el siguiente comando:

```
(ll_env)learning_log$ pip instalar platformashconfig
```

Usaremos este paquete para modificar la configuración del proyecto cuando se ejecute en el servidor en vivo.

Creando un archivo de requisitos.txt

El servidor remoto necesita saber de qué paquetes depende el Registro de aprendizaje, por lo que usaremos pip para generar un archivo que los enumere. Nuevamente, desde un entorno virtual activo, ejecute el siguiente comando:

```
(ll_env)learning_log$ congelación de pips > requisitos.txt
```

El comando congelar le dice a pip que escriba los nombres de todos los paquetes actualmente instalados en el proyecto en el archivo requisitos.txt. Abra este archivo para ver los paquetes y números de versión instalados en su proyecto:

```
requisitos.txt asgiref==3.5.2
hermosasopa4==4.11.1
Django==4.1
```

```

django-bootstrap5==21.3
plataformashconfig==2.4.0
tamiz de sopa==2.3.2.post1
sqlparse==0.4.2

```

Learning Log ya depende de versiones específicas de siete paquetes diferentes, por lo que requiere un entorno coincidente para ejecutarse correctamente en un servidor remoto. (Instalamos tres de estos paquetes manualmente y cuatro de ellos se instalaron automáticamente como dependencias de estos paquetes).

Cuando implementemos Learning Log, Platform.sh instalará todos los paquetes enumerados en requisitos.txt, creando un entorno con los mismos paquetes que estamos usando localmente. Debido a esto, podemos estar seguros de que el proyecto implementado funcionará tal como lo ha hecho en nuestro sistema local. Este enfoque para gestionar un proyecto es fundamental a medida que comienza a crear y mantener varios proyectos en su sistema.

NOTA Si el número de versión de un paquete que aparece en su sistema difiere del que se muestra aquí, conserve la versión que tiene en su sistema.

Requisitos de implementación adicionales

El servidor en vivo requiere dos paquetes adicionales. Estos paquetes se utilizan para servir el proyecto en un entorno de producción, donde muchos usuarios pueden realizar solicitudes al mismo tiempo.

En el mismo directorio donde se guarda requisitos.txt, cree un nuevo archivo llamado requisitos_remoto.txt. Agregue los siguientes dos paquetes:

| | |
|---------------------------|--|
| requisitos
_remoto.txt | # Requisitos para proyecto en vivo.
gunicornio
psycpg2 |
|---------------------------|--|

El paquete gunicorn responde a las solicitudes a medida que llegan al servidor remoto; esto reemplaza el servidor de desarrollo que hemos estado usando localmente. El paquete psycpg2 es necesario para permitir que Django administre la base de datos Postgres que utiliza Platform.sh. Postgres es una base de datos de código abierto que se adapta muy bien a aplicaciones de producción.

Agregar archivos de configuración

Toda plataforma de hosting requiere cierta configuración para que un proyecto se ejecute correctamente en sus servidores. En esta sección, agregaremos tres archivos de configuración:

- .platform.app.yaml Este es el archivo de configuración principal del proyecto. Esto le dice a Platform.sh qué tipo de proyecto estamos intentando implementar y qué tipos de recursos necesita nuestro proyecto, e incluye comandos para construir el proyecto en el servidor.
- .platform/routes.yaml Este archivo define las rutas a nuestro proyecto. Cuando Platform.sh recibe una solicitud, esta es la configuración que ayuda a dirigir estas solicitudes a nuestro proyecto específico.
- .platform/services.yaml Este archivo define cualquier servicio adicional que nuestro proyecto necesite.

Todos estos son archivos YAML (YAML no es lenguaje de marcado). YAML es un lenguaje diseñado para escribir archivos de configuración; está hecho para que lo lean fácilmente tanto humanos como computadoras. Puede escribir o modificar un archivo YAML típico a mano, pero una computadora también puede leer e interpretar el archivo sin ambigüedades.

Los archivos YAML son excelentes para la configuración de implementación, porque brindan le da un gran control sobre lo que sucede durante el proceso de implementación.

Hacer visibles los archivos ocultos

La mayoría de los sistemas operativos ocultan archivos y carpetas que comienzan con un punto, como .platform. Cuando abre un explorador de archivos, no verá este tipo de archivos y carpetas de forma predeterminada. Pero como programador, necesitarás verlos. A continuación se explica cómo ver archivos ocultos, según su sistema operativo:

- En Windows, abra el Explorador de Windows y luego abra una carpeta como Escritorio. Haga clic en la pestaña Ver y asegúrese de que Extensiones de nombre de archivo y Elementos ocultos estén marcados.
- En macOS, puede presionar -SHIFT- (punto) en cualquier ventana del Finder para ver archivos y carpetas ocultos.
- En sistemas Linux como Ubuntu, puede presionar CTRL-H en cualquier explorador de archivos para mostrar archivos y carpetas ocultos. Para que esta configuración sea permanente, abra un explorador de archivos como Nautilus y haga clic en la pestaña de opciones (indicada por tres líneas). Seleccione la casilla de verificación Mostrar archivos ocultos .

El archivo de configuración .platform.app.yaml

El primer archivo de configuración es el más largo porque controla el proceso de implementación general. Lo mostraremos en partes; puede ingresarlo manualmente en su editor de texto o descargar una copia de los recursos en línea en https://ehmatthes.github.io/pcc_3e.

Aquí está la primera parte de .platform.app.yaml, que debe guardarse en el mismo directorio que Manage.py:

```
.plataforma 1 nombre: "Il_project"
.app.yaml   tipo: "python: 3.10"

2 relaciones:
    base de datos: "db:postgresql"

    # La configuración de la aplicación cuando está expuesta a la web.
3 redes:
    río arriba:
        familia_socket: unix
    comandos:
4        inicio: "gunicorn -w 4 -b unix:$SOCKET Il_project.wsgi:aplicación"
5 ubicaciones:
    "/":
        paso a través: verdadero
    "/estático":
```

raíz: "estática"

caduca: 1h

permitir: verdadero

El tamaño del disco persistente de la aplicación (en MB).
6 discos: 512

Cuando guarde este archivo, asegúrese de incluir el punto al principio del nombre del archivo. Si omite el punto, Platform.sh no encontrará el archivo y su proyecto no se implementará.

No es necesario que comprenda todo lo que hay en .platform.app.yaml en este momento; Destacaré las partes más importantes de la configuración. El archivo comienza especificando el nombre del proyecto, al que llamaremos 'Il_project' para ser coherente con el nombre que usamos al iniciar el proyecto 1. También debemos especificar la versión de Python que estamos usando (3.10 en el momento de escribir este artículo). Puede encontrar una lista de versiones compatibles en <https://docs.platform.sh/idiomas/python.html>.

La siguiente es una sección denominada relaciones que define otros servicios que el proyecto necesita 2. Aquí la única relación es con una base de datos de Postgres. Después de eso está la sección web 3. La sección comandos: inicio le dice a Platform.sh qué proceso usar para atender las solicitudes entrantes. Aquí especificamos que gunicorn manejará las solicitudes 4. Este comando reemplaza el comando python Manage.py RunServer que hemos estado usando localmente.

La sección de ubicaciones le indica a Platform.sh dónde enviar las solicitudes entrantes 5. La mayoría de las solicitudes deben pasarse a gunicorn; Nuestros archivos urls.py le dirán a gunicorn exactamente cómo manejar esas solicitudes. Las solicitudes de archivos estáticos se manejarán por separado y se actualizarán una vez cada hora. La última línea muestra que estamos solicitando 512 MB de espacio en disco en uno de los servidores 6 de Platform.sh.

El resto de .platform.app.yaml es como sigue:

```
--recorte--
disco: 512
```

```
# Establecer un montaje de lectura/escritura local para los registros.
1 monturas:
```

```
  "registros":
    fuente: locales
    source_path: registros
```

```
# Los ganchos se ejecutan en varios puntos del ciclo de vida de la aplicación.
2 ganchos:
```

```
  construir: |
3      instalación de pip --actualizar pip
      instalación de pip -r requisitos.txt
      instalación de pip -r requisitos_remoto.txt
```

```
  registros mkdir
4      python administrar.py recopila estática
      rm -rf registros
```

```
5 desplegar: |
      python administrar.py migrar
```

La sección de montajes 1 nos permite definir directorios donde podemos leer y escribir datos mientras el proyecto se está ejecutando. Esta sección define un directorio logs/ para el proyecto implementado.

La sección 2 de ganchos define acciones que se toman en varios puntos durante el proceso de implementación. En la sección de compilación , instalamos todos los paquetes necesarios para ejecutar el proyecto en el entorno en vivo 3. También ejecutamos Collectstatic 4, que recopila todos los archivos estáticos necesarios para el proyecto en un solo lugar para que puedan entregarse de manera eficiente. .

Finalmente, en la sección 5 de implementación , especificamos que las migraciones deben ejecutarse cada vez que se implementa el proyecto. En un proyecto simple esto no tendrá ningún efecto cuando no haya habido cambios.

Los otros dos archivos de configuración son mucho más cortos; escribámoslos ahora.

El archivo de configuración route.yaml

Una ruta es la ruta que toma una solicitud mientras la procesa el servidor. Cuando Platform.sh recibe una solicitud, necesita saber dónde enviarla.

Cree una nueva carpeta llamada .platform, en el mismo directorio que administrar.py. Asegúrate de incluir el punto al principio del nombre. Dentro de esa carpeta, cree un archivo llamado rutas.yaml e ingrese lo siguiente:

```
.platforma/ # Cada ruta describe cómo Platform.sh procesará una URL entrante.
rutas.yaml

"https://{predeterminado}"/:
  tipo: aguas arriba
  aguas arriba: "ll_project:http"

"https://www.{predeterminado}"/:
  tipo: redirigir
  a: "https://{predeterminado}"/
```

Este archivo garantiza que solicitudes como https://project_url.com y www.project_url.com todos se dirigen al mismo lugar.

El archivo de configuración services.yaml

Este último archivo de configuración especifica los servicios que nuestro proyecto necesita para ejecutarse. Guarde este archivo en el directorio .platform/ , junto con route.yaml:

```
.platforma/ # Cada servicio enumerado se implementará en su propio contenedor como parte de su
rutas.yaml # Proyecto Platform.sh.

# Base de datos
tipo: postgresql: 12
disco: 1024
```

Este archivo define un servicio, una base de datos Postgres.

Modificando settings.py para Platform.sh

Ahora necesitamos agregar una sección al final de settings.py para modificar algunas configuraciones para el entorno Platform.sh. Agregue este código al final de settings.py:

```
configuración.py --snip--
# Configuración de Platform.sh.
1 de la configuración de importación de platformshconfig

configuración = Configuración()
2 si config.is_valid_platform():
3 ALLOWED_HOSTS.append('.platformsh.site')

4 si config.appDir:
    STATIC_ROOT = Ruta(config.appDir) / 'estática'
5 si config.projectEntropy:
    SECRET_KEY = config.projectEntropy

si no es config.in_build():
6    db_settings = config.credentials('base de datos')
    BASES DE DATOS = {
        'por defecto': {
            'MOTOR': 'django.db.backends.postgresql',
            'NOMBRE': db_settings['ruta'],
            'USUARIO': db_settings['nombre de usuario'],
            'CONTRASEÑA': db_settings['contraseña'],
            'HOST': db_settings['host'],
            'PUERTO': db_settings['puerto'],
        },
    }
```

Normalmente colocamos declaraciones de importación al comienzo de un módulo, pero en este caso, es útil mantener todas las configuraciones remotas específicas en una sección. Aquí importamos la configuración de la plataforma shconfig 1, que ayuda a determinar la configuración en el servidor remoto. Sólo modificamos la configuración si el método config.is_valid_platform() devuelve True 2, lo que indica que la configuración se está utilizando en un servidor Platform.sh.

Modificamos ALLOWED_HOSTS para permitir que el proyecto sea servido por hosts que finalicen en .platformsh.site 3. Todos los proyectos implementados en el nivel gratuito se servirán utilizando este host. Si la configuración se carga en el directorio 4 de la aplicación implementada, configuramos STATIC_ROOT para que los archivos estáticos se entreguen correctamente. También configuramos una SECRET_KEY más segura en el servidor remoto 5.

Finalmente, configuramos la base de datos de producción 6. Esto solo se configura si el proceso de compilación ha terminado de ejecutarse y el proyecto está en servicio. Todo lo que ves aquí es necesario para permitir que Django se comuniquen con el servidor Postgres que Platform.sh configuró para el proyecto.

Usando Git para rastrear los archivos del proyecto

Como se analizó en el Capítulo 17, Git es un programa de control de versiones que le permite tomar una instantánea del código en su proyecto cada vez que implementa una nueva característica con éxito. Si algo sale mal, puedes volver fácilmente a

la última instantánea de trabajo de su proyecto; por ejemplo, si accidentalmente introduces un error mientras trabajas en una nueva característica. Cada instantánea se llama `commit`.

Con Git, puedes intentar implementar nuevas funciones sin preocuparte por arruinar tu proyecto. Cuando realiza la implementación en un servidor activo, debe asegurarse de implementar una versión funcional de su proyecto. Para leer más sobre Git y el control de versiones, consulte el Apéndice D.

Instalación de Git

Es posible que Git ya esté instalado en su sistema. Para averiguarlo, abra una nueva ventana de terminal y ejecute el comando `git --version`:

```
(ll_env)learning_log$ git --version
git versión 2.30.1 (Apple Git-130)
```

Si recibe un mensaje que indica que Git no está instalado, consulte la instalación. instrucciones de funcionamiento en el Apéndice D.

Configurando Git

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para hacer esto, Git necesita saber su nombre de usuario y correo electrónico. Debes proporcionar un nombre de usuario, pero puedes crear un correo electrónico para tus proyectos de práctica:

```
(ll_env)learning_log$ git config --global nombre.usuario "eric"
(ll_env)learning_log$ git config --global usuario.correo electrónico "eric@example.com"
```

Si olvida este paso, Git le solicitará esta información cuando realice su primera confirmación.

Ignorar archivos

No necesitamos que Git rastree todos los archivos del proyecto, por lo que le indicaremos que ignore algunos archivos. Cree un archivo llamado `.gitignore` en la carpeta que contiene `Manage.py`.

Observe que este nombre de archivo comienza con un punto y no tiene extensión de archivo. Aquí está el código que va en `.gitignore`:

```
.gitignore ll_env/
__pycache__ /
*.sqlite3
```

Le decimos a Git que ignore todo el directorio `ll_env`, porque podemos recrear automáticamente en cualquier momento. Tampoco realizamos un seguimiento del directorio `__pycache__`, que contiene los archivos `.pyc` que se crean automáticamente cuando se ejecutan los archivos `.py`. No realizamos un seguimiento de los cambios en la base de datos local, porque es un mal hábito: si alguna vez usa SQLite en un servidor, podría sobrescribir accidentalmente la base de datos activa con su base de datos de prueba local cuando envíe el proyecto al servidor. El asterisco en `*.sqlite3` le dice a Git que ignore cualquier archivo que termine con la extensión `.sqlite3`.

NOTA está utilizando macOS, agregue .DS_Store a su archivo .gitignore . Este es un archivo que almacena información sobre la configuración de carpetas en macOS y no tiene nada que ver con este proyecto.

Comprometer el proyecto

Necesitamos inicializar un repositorio Git para el Registro de aprendizaje, agregar todos los archivos necesarios al repositorio y confirmar el estado inicial del proyecto. Aquí se explica cómo hacerlo:

```
1 (ll_env)learning_log$ git init
  Repositorio Git vacío inicializado en /Users/eric/.../learning_log/.git/
2 (ll_env)learning_log$ git add.
3 (ll_env)learning_log$ git commit -am "Listo para la implementación en Platform.sh".
  [main (root-commit) c7ffaad] Listo para la implementación en Platform.sh.
  42 archivos modificados, 879 inserciones(+)
  crear modo 100644 .gitignore
  crear modo 100644 .platform.app.yaml
  --recorte--
  crear modo 100644 requisitos_remoto.txt
4 (ll_env)learning_log$ estado de git
  En la rama principal
  nada que comprometer, trabajando árbol limpio
(ll_env)learning_log$
```

Emitimos el comando git init para inicializar un repositorio vacío en el directorio que contiene el Registro de aprendizaje 1. Luego usamos git add. comando, que agrega todos los archivos que no se ignoran al repositorio 2. (No olvide el punto). A continuación, emitimos el comando git commit -am "mensaje de confirmación": el indicador -a le dice a Git que incluya todos los archivos modificados en esta confirmación, y el -m flag le dice a Git que registre un mensaje de registro 3.

Emitir el comando git status 4 indica que estamos en la página principal rama y que nuestro árbol de trabajo esté limpio. Este es el estado que querrás ver cada vez que envíes tu proyecto a un servidor remoto.

Creando un proyecto en Platform.sh

En este punto, el proyecto Learning Log todavía se ejecuta en nuestro sistema local y también está configurado para ejecutarse correctamente en un servidor remoto. Usaremos la CLI Platform.sh para crear un nuevo proyecto en el servidor y luego enviaremos nuestro proyecto al servidor remoto.

Asegúrate de estar en una terminal, en el directorio learning_log/ , y emite el siguiente comando:

```
(ll_env)learning_log$ inicio de sesión en la plataforma
URL abierta: http://127.0.0.1:5000
Utilice el navegador para iniciar sesión.
--recorte--
1 ¿Quieres crear un archivo de configuración SSH automáticamente? [S/n] Sí
```

Este comando abrirá una pestaña del navegador donde podrá iniciar sesión. Una vez que haya iniciado sesión, puede cerrar la pestaña del navegador y regresar a la terminal. Si se le solicita que cree un archivo de configuración SSH 1, ingrese Y para poder conectarse al servidor remoto más tarde.

Ahora crearemos un proyecto. Hay muchos resultados, así que veremos el proceso de creación en secciones. Comience emitiendo el comando crear :

```
(ll_env)learning_log$ creación de plataforma
  Título del proyecto (--título)
Predeterminado: Proyecto sin título
1 > ll_proyecto

*
  Región (--región)
La región donde se alojará el proyecto.
--recorte--
[us-3.platform.sh] Moses Lake, Estados Unidos (AZURE) [514 gC02eq/kWh]
2 > us-3.plataforma.sh

*Planificar (--planificar)
Predeterminado: desarrollo
Ingrese un número para elegir:
  [0] desarrollo
--recorte--
3 > 0

* Ambientes (--ambientes)
El número de ambientes
Predeterminado: 3
4 > 3

*
  Almacenamiento (--almacenamiento)
La cantidad de almacenamiento por entorno, en GiB
Predeterminado: 5
5 > 5
```

El primer mensaje solicita un nombre para el proyecto 1, por lo que usamos el nombre ll_project. El siguiente mensaje pregunta en qué región nos gustaría que esté el servidor. 2. Elija el servidor más cercano a usted; para mí, ese es us-3.platform.sh. Para el resto de las indicaciones, puede aceptar los valores predeterminados: un servidor en el plan de desarrollo más bajo 3, tres entornos para el proyecto 4 y 5 GB de almacenamiento para el proyecto general 5.

Hay tres indicaciones más a las que responder:

```
Rama predeterminada (--rama predeterminada)
El nombre de rama de Git predeterminado para el proyecto (el entorno de producción)
Predeterminado: principal
1 > principal

Repositorio Git detectado: /Users/eric/.../learning_log
2 ¿Establecer el nuevo proyecto ll_project como remoto para este repositorio? [S/n] Sí

El costo mensual estimado de este proyecto es: $10 USD
3 ¿Estás seguro de que quieres continuar? [S/n] Sí
```


El Bot Platform.sh está activando tu proyecto



¡El proyecto ya está listo!

Un repositorio Git puede tener varias sucursales; Platform.sh nos pregunta si la rama predeterminada para el proyecto debe ser principal 1. Luego nos pregunta si queremos conectar el repositorio del proyecto local al repositorio remoto 2. Finalmente, se nos informa que este proyecto costará alrededor de \$10 por mes si lo mantenemos funcionando más allá del período de prueba gratuito 3. Si aún no ha ingresado una tarjeta de crédito, no debería tener que preocuparse por este costo. Platform.sh simplemente suspenderá su proyecto si excede los límites de la prueba gratuita sin agregar una tarjeta de crédito.

Empujando a Platform.sh

El último paso antes de ver la versión en vivo del proyecto es enviar nuestro código al servidor remoto. Para hacer eso, emita el siguiente comando:

```
(ll_env)learning_log$ impulso de plataforma
1 ¿Está seguro de que desea pasar a la rama principal (producción)? [S/n] Sí
--recorte--
No se puede establecer la autenticidad del host 'git.us-3.platform.sh (...)'.
La huella digital de la clave RSA es SHA256: Tvn...7PM
2 ¿Está seguro de que desea continuar conectándose (sí/no/[huella digital])? Y
Empujando HEAD al entorno existente principal
--recorte--
A git.us-3.platform.sh:3pp3mqcexhlvy.git
* [nueva sucursal]          CABEZA -> principal
```

Cuando emita el comando `platform push`, se le pedirá una confirmación más de que desea impulsar el proyecto 1. También puede ver un mensaje sobre la autenticidad de Platform.sh, si es la primera vez que se conecta a el sitio 2. Ingrese Y para cada una de estas indicaciones y verá un montón de resultados desplazándose. Este resultado probablemente parezca confuso al principio, pero si algo sale mal, es muy útil tenerlo durante la resolución de problemas.

Si hojear el resultado, podrá ver dónde Platform.sh instala los paquetes necesarios, recopila archivos estáticos, aplica migraciones y configura las URL para el proyecto.

NOTA Es posible que vea un error debido a algo que pueda diagnosticar fácilmente, como un error tipográfico en uno de los archivos de configuración. Si esto sucede, corrija el error en su editor de texto, guarde el archivo y vuelva a emitir el comando `git commit`. Luego puedes ejecutar el empuje de plataforma nuevamente.

Ver el proyecto en vivo

Una vez que se completa el envío, puede abrir el proyecto:

```
(ll_env)learning_log$ URL de la plataforma
Ingrese un número para abrir una URL
[0] https://main-bvxhea6i-wmye2fx7wwqgu.us-3.platformsh.site/
--recorte--
> 0
```

El comando `platform url` enumera las URL asociadas con un proyecto implementado; Podrás elegir entre varias URL, todas válidas para tu proyecto. Elija uno y su proyecto debería abrirse en una nueva pestaña del navegador.

Esto se verá igual que el proyecto que hemos estado ejecutando localmente, pero puedes compartir esta URL con cualquier persona en el mundo, y ellos podrán acceder y utilizar tu proyecto.

NOTA Cuando implemente su proyecto usando una cuenta de prueba, no se sorprenda si a veces una página tarda más de lo habitual en cargarse. En la mayoría de las plataformas de alojamiento, los recursos gratuitos que están inactivos a menudo se suspenden y solo se reinician cuando llegan nuevas solicitudes. La mayoría de las plataformas responden mucho mejor a los planes de alojamiento pagos.

Refinando la implementación de Platform.sh

Ahora perfeccionaremos la implementación creando un superusuario, tal como lo hicimos localmente. También haremos que el proyecto sea más seguro cambiando la configuración `DEBUG` a `False`, de modo que los mensajes de error no muestren a los usuarios ninguna información adicional que puedan usar para atacar el servidor.

Creando un superusuario en Platform.sh

La base de datos para el proyecto en vivo se configuró, pero está completamente vacía. Todos los usuarios que creamos anteriormente solo existen en nuestra versión local del proyecto.

Para crear un superusuario en la versión en vivo del proyecto, iniciaremos una sesión SSH (secure socket shell) donde podemos ejecutar comandos de administración en el servidor remoto:

```
(ll_env)learning_log$ entorno de plataforma:ssh
```

```
--
_ \L_| | / _|___--_| |
| | / _'| | / '\'| '( < '\
| | \ , | | \ / | | | ( ) / | |
```

Bienvenido a Platform.sh.

```
1 web@ll_project.0:~$ ls
   cuentas learning_logs ll_project logs administrar.py requisitos.txt
   requisitos_remoto.txt estático
2 web@ll_project.0:~$ python administrar.py crearsuperusuario
3 Nombre de usuario (deje en blanco para usar 'web'): ll_admin live
```

Dirección de correo electrónico:

```

Contraseña:
Contraseña de nuevo):
Superusuario creado exitosamente.
4 web@ll_project.0:~$ salir
      cerrar sesión
Conexión a ssh.us-3.platform.sh cerrada.
5 (ll_env)learning_log$

```

Cuando ejecuta por primera vez el comando entorno de plataforma:ssh , es posible que reciba otro mensaje sobre la autenticidad de este host. Si ve este mensaje, ingrese Y y debería iniciar sesión en una sesión de terminal remota.

Después de ejecutar el comando ssh , su terminal actúa como una terminal en el servidor remoto. Tenga en cuenta que su mensaje ha cambiado para indicar que está en una sesión web asociada con el proyecto llamado ll_project 1. Si ejecuta el comando ls , verá los archivos que se han enviado a Platform.sh servidor.

Emita el mismo comando createsuperuser que utilizamos en el Capítulo 18 2. Este. Esta vez, ingrese un nombre de usuario de administrador, ll_admin_live, que es distinto del que usé localmente 3. Cuando haya terminado de trabajar en la sesión de terminal remota, ingrese el comando de salida 4. El mensaje le indicará que está trabajando en su local sistema nuevamente 5.

Ahora puede agregar /admin/ al final de la URL de la aplicación en vivo e iniciar sesión en el sitio de administración. Si otras personas ya han comenzado a utilizar tu proyecto, ¡ten en cuenta que tendrás acceso a todos sus datos! Tome esta responsabilidad en serio y los usuarios seguirán confiándole sus datos.

NOTA Los usuarios de Windows utilizarán los mismos comandos que se muestran aquí (como ls en lugar de dir), porque están ejecutando una terminal Linux a través de una conexión remota.

Asegurar el proyecto en vivo

Hay un problema de seguridad evidente en la forma en que se implementa actualmente nuestro proyecto: la configuración DEBUG = True en settings.py, que proporciona mensajes de depuración cuando ocurren errores. Las páginas de error de Django te brindan información de depuración vital cuando estás desarrollando un proyecto; sin embargo, brindan demasiada información a los atacantes si los dejas habilitados en un servidor activo.

Para ver qué tan grave es esto, vaya a la página de inicio de su proyecto implementado. Inicie sesión en la cuenta de un usuario y agregue /topics/999/ al final de la URL de la página de inicio. Suponiendo que no haya creado miles de temas, debería ver una página con el mensaje DoesNotExist en /topics/999/. Si se desplaza hacia abajo, debería ver una gran cantidad de información sobre el proyecto y el servidor.

No querrá que sus usuarios vean esto y ciertamente no querrá que esta información esté disponible para cualquiera interesado en atacar el sitio.

Podemos evitar que esta información se muestre en el sitio en vivo configurando DEBUG = False en la parte de settings.py que solo se aplica a la versión implementada del proyecto. De esta manera, seguirá viendo información de depuración localmente, cuando esa información sea útil, pero no aparecerá en el sitio en vivo.

Abra settings.py en su editor de texto y agregue una línea de código a la parte que modifica la configuración de Platform.sh:

```
configuración.py --snip--
si config.is_valid_platform():
    ALLOWED_HOSTS.append('.platformsh.site')
    DEPURACIÓN = Falso
--recorte--
```

Todo el trabajo realizado para establecer la configuración de la versión implementada del proyecto ha dado sus frutos. Cuando queremos ajustar la versión en vivo del proyecto, simplemente cambiamos la parte relevante de la configuración que configuramos anteriormente.

Comprometer e impulsar cambios

Ahora necesitamos confirmar los cambios realizados en settings.py y enviar los cambios a Platform.sh. Aquí hay una sesión de terminal que muestra la primera parte de este proceso:

```
(ll_env)learning_log$ git commit -am "Establecer DEBUG False en el sitio en vivo".
[main d2ad0f7] Establezca DEBUG False en el sitio en vivo.
    1 archivo modificado, 1 inserción(+)
2 (ll_env)learning_log$ estado de git
En la rama principal
nada que comprometer, trabajando árbol limpio
(ll_env)learning_log$
```

Emitimos el comando git commit con un mensaje de confirmación breve pero descriptivo

1. Recuerde que el indicador -am garantiza que Git confirme todos los archivos que han cambiado y registra el mensaje de registro. Git reconoce que un archivo ha cambiado y envía este cambio al repositorio.

Ejecutar git status muestra que estamos trabajando en la rama principal del repositorio y que ahora no hay nuevos cambios para confirmar 2. Es importante verificar el estado antes de enviarlo a un servidor remoto. Si no ve un estado limpio, entonces algunos cambios no se han confirmado y esos cambios no se enviarán al servidor. Puedes intentar emitir el comando de confirmación nuevamente; Si no está seguro de cómo resolver el problema, lea el Apéndice D.

para comprender mejor cómo trabajar con Git.

Ahora enviemos el repositorio actualizado a Platform.sh:

```
(ll_env)learning_log$ impulso de plataforma
¿Está seguro de que desea pasar a la rama principal (producción)? [S/n] Sí
Empujando HEAD al entorno existente principal
--recorte--
A git.us-3.platform.sh:wmye2fx7wwqgu.git
fce0206..d2ad0f7 CABEZA -> principal
(ll_env)learning_log$
```

Platform.sh reconoce que el repositorio se ha actualizado y reconstruye el proyecto para asegurarse de que se hayan tenido en cuenta todos los cambios. No reconstruye la base de datos, por lo que no hemos perdido ningún dato.

Para asegurarse de que este cambio haya entrado en vigor, visite `/topics/999/` URL nuevamente. Debería ver solo el mensaje Error del servidor (500), sin ninguna información confidencial sobre el proyecto.

Crear páginas de error personalizadas

En el Capítulo 19, configuramos el Registro de aprendizaje para que devuelva un error 404 si el usuario solicita un tema o entrada que no le pertenece. Ahora también has visto un error de servidor 500. Un error 404 generalmente significa que su código Django es correcto, pero el objeto solicitado no existe. Un error 500 generalmente significa que hay un error en el código que ha escrito, como un error en una función en `views.py`. Django actualmente devuelve la misma página de error genérica en ambas situaciones, pero podemos escribir nuestras propias plantillas de páginas de error 404 y 500 que coincidan con la apariencia general de Learning Log. Estas plantillas pertenecen al directorio raíz de plantillas.

Hacer plantillas personalizadas

En la carpeta `learning_log`, cree una nueva carpeta llamada `plantillas`. Luego cree un nuevo archivo llamado `404.html`; la ruta a este archivo debe ser `learning_log/plantillas/404.html`. Aquí está el código de este archivo:

```
404.html {% extiende "learning_logs/base.html" %}

{% bloquear page_header %}
<h2>El artículo que solicitaste no está disponible. (404)</h2>
{% bloque final page_header %}
```

Esta plantilla simple proporciona información genérica de la página de error 404, pero tiene un estilo que coincide con el resto del sitio.

Cree otro archivo llamado `500.html` usando el siguiente código:

```
500.html {% extiende "learning_logs/base.html" %}

{% bloquear page_header %}
<h2>Ha habido un error interno. (500)</h2>
{% bloque final page_header %}
```

Estos nuevos archivos requieren un ligero cambio en `settings.py`.

```
configuración.py --snip--
PLANTILLAS = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR/'plantillas'],
        'APP_DIRS': Verdadero,
        --recorte--
    },
]
--recorte--
```

Este cambio le dice a Django que busque en el directorio raíz de plantillas las plantillas de página de error y cualquier otra plantilla que no esté asociada con una aplicación en particular.

Impulsando los cambios en Platform.sh

Ahora necesitamos confirmar los cambios que acabamos de realizar y enviarlos a Platform.sh:

```
1 (ll_env)learning_log$ git add.
2 (ll_env)learning_log$ git commit -am "Se agregaron páginas de error 404 y 500 personalizadas".
   3 archivos modificados, 1 inserciones(+), 1 eliminación(-)
   crear modo 100644 plantillas/404.html
   crear modo 100644 plantillas/500.html
3 (ll_env)learning_log$ impulso de plataforma
--recorte--
   A git.us-3.platform.sh:wmye2fx7wwqgu.git
   d2ad0f7..9f042ef CABEZA -> principal
(ll_env)learning_log$
```

git add en el proyecto. . comando 1 porque creamos algunos archivos nuevos. Emitimos
Luego confirmamos los cambios 2 y enviamos el proyecto actualizado a Platform.sh 3.

Ahora, cuando aparece una página de error, debería tener el mismo estilo que la
resto del sitio, lo que hace que la experiencia del usuario sea más fluida cuando surgen errores.

Desarrollo en curso

Es posible que desee desarrollar aún más el Registro de aprendizaje después de su envío inicial a un servidor en vivo, o tal vez desee desarrollar sus propios proyectos para implementar. Al hacerlo, existe un proceso bastante consistente para actualizar sus proyectos.

Primero, realizará los cambios necesarios en su proyecto local. Si sus cambios dan como resultado archivos nuevos, agregue esos archivos al repositorio de Git usando el comando git add. (asegurándose de incluir el punto al final del comando). Cualquier cambio que requiera una migración de la base de datos necesitará este comando, porque cada migración genera un nuevo archivo de migración.

En segundo lugar, confirme los cambios en su repositorio usando git commit -am "mensaje de confirmación". Luego envíe sus cambios a Platform.sh, usando el comando platform push. Visite su proyecto en vivo y asegúrese de que los cambios que espera ver hayan surtido efecto.

Es fácil cometer errores durante este proceso, así que no se sorprenda si algo sale mal. Si el código no funciona, revise lo que ha hecho e intenta detectar el error. Si no puede encontrar el error o no sabe cómo deshacerlo, consulte las sugerencias para obtener ayuda en el Apéndice C. No dude en pedir ayuda: todos los demás aprendieron a crear proyectos preguntándole a las mismas preguntas que usted probablemente hará, por lo que alguien estará encantado de ayudarlo. Resolver cada problema que surja lo ayudará a desarrollar constantemente sus habilidades hasta que esté construyendo proyectos significativos y confiables y respondiendo también las preguntas de otras personas.

Eliminar un proyecto en Platform.sh

Es una buena práctica ejecutar el proceso de implementación varias veces con el mismo proyecto o con una serie de proyectos pequeños para dominar la implementación. Pero necesitarás saber cómo eliminar un proyecto que se haya implementado. Platform.sh también limita la cantidad de proyectos que puede alojar de forma gratuita y no desea saturar su cuenta con proyectos de práctica.

Puede eliminar un proyecto utilizando la CLI:

```
(ll_env)learning_log$ proyecto de plataforma:eliminar
```

Se le pedirá que confirme que desea realizar esta acción destructiva.

Responda a las indicaciones y su proyecto será eliminado.

El comando `platform create` también le dio al repositorio Git local una referencia al repositorio remoto en los servidores de Platform.sh. También puedes eliminar este control remoto desde la línea de comando:

```
(ll_env)learning_log$ git remoto
plataforma
(ll_env)learning_log$ git plataforma de eliminación remota
```

El comando `git remoto` enumera los nombres de todas las URL remotas asociadas con el repositorio actual. El comando `git remoto eliminar nombre_remoto` elimina estas URL remotas del repositorio local.

También puede eliminar los recursos de un proyecto iniciando sesión en el sitio web Platform.sh y visitando su panel en <https://console.platform.sh>. Esta página enumera todos sus proyectos activos. Haga clic en los tres puntos en el cuadro de un proyecto y haga clic en Editar plan. Esta es una página de precios para el proyecto; haga clic en Eliminar proyecto en la parte inferior de la página y se le mostrará una página de confirmación donde podrá continuar con la eliminación. Incluso si eliminó su proyecto usando la CLI, es una buena idea familiarizarse con el panel de cualquier proveedor de alojamiento en el que implemente.

NOTA Eliminar un proyecto en Platform.sh no afecta la versión local del proyecto.

Si nadie ha utilizado su proyecto implementado y solo está practicando el proceso de implementación, es perfectamente razonable eliminar su proyecto en Platform.sh y volver a implementarlo. Sólo tenga en cuenta que si algo deja de funcionar, es posible que se haya topado con las limitaciones del nivel gratuito del host.

INTÉNTALO TÚ MISMO

20-3. Blog en vivo: implemente el proyecto de blog en el que ha estado trabajando en Platform.sh. Asegúrese de configurar `DEBUG` en `False`, para que los usuarios no vean las páginas de error completas de Django cuando algo sale mal.

(continuado)

20-4. Registro de aprendizaje extendido: agregue una función al Registro de aprendizaje e impulse el cambio a su implementación en vivo. Pruebe un cambio simple, como escribir más sobre el proyecto en la página de inicio. Luego intente agregar una función más avanzada, como brindar a los usuarios la opción de hacer público un tema. Esto requeriría un atributo llamado público como parte del modelo de tema (esto debe establecerse en Falso de forma predeterminada) y un elemento de formulario en la página new_topic que permite al usuario cambiar un tema de privado a público. Luego deberá migrar el proyecto y revisar views.py para que cualquier tema público también sea visible para usuarios no autenticados.

Resumen

En este capítulo, aprendiste a darle a tus proyectos una apariencia simple pero profesional usando la biblioteca Bootstrap y la aplicación django-bootstrap5.

Con Bootstrap, los estilos que elijas funcionarán de manera consistente en casi cualquier dispositivo que la gente use para acceder a tu proyecto.

Aprendió sobre las plantillas de Bootstrap y utilizó la plantilla estática de la barra de navegación para crear una apariencia simple para Learning Log. Utilizó un jumbo-tron para resaltar el mensaje de una página de inicio y aprendió a diseñar todas las páginas de un sitio de manera consistente.

En la parte final del proyecto, aprendiste cómo implementar un proyecto en un servidor remoto para que cualquiera pueda acceder a él. Creó una cuenta de Platform.sh e instaló algunas herramientas que ayudan a administrar el proceso de implementación. Usó Git para enviar el proyecto de trabajo a un repositorio y luego envió el repositorio a un servidor remoto en Platform.sh. Finalmente, aprendió a comenzar a proteger su aplicación configurando `DEBUG = False` en el servidor en vivo. También creó páginas de error personalizadas, por lo que los errores inevitables que surjan se verán bien manejados.

Ahora que ha terminado el Registro de aprendizaje, puede comenzar a crear el suyo propio. proyectos. Comience de manera simple y asegúrese de que el proyecto funcione antes de agregar complejidad. ¡Disfruta de tu aprendizaje continuo y buena suerte con tus proyectos!

A

INSTALACIÓN Y SOLUCIÓN DE PROBLEMAS



Hay muchas versiones de Python disponibles y numerosas formas de configurarlo en cada sistema operativo. Si el método del Capítulo 1 no funcionó, o si desea instalar una versión de Python diferente a la que está instalada actualmente, las instrucciones de este apéndice pueden re

Python en Windows

Las instrucciones del Capítulo 1 le muestran cómo instalar Python utilizando el instalador oficial en <https://python.org>. Si no pudo ejecutar Python después de usar el instalador, las instrucciones de solución de problemas en esta sección le ayudarán a poner Python en funcionamiento.

Usando `py` en lugar de `python`

Si ejecuta un instalador reciente de Python y luego emite el comando `python` en una terminal, debería ver el mensaje de Python para una sesión de terminal (`>>>`).

Cuando Windows no reconoce el comando Python , abrirá Microsoft Store porque cree que Python no está instalado o recibirá un mensaje como "No se encontró Python". Si se abre Microsoft Store, ciérrela; es mejor usar el instalador oficial de Python de <https://python.org> que el que mantiene Microsoft.

La solución más sencilla, sin realizar ningún cambio en su sistema, es probar el comando `py` . Esta es una utilidad de Windows que encuentra la última versión de Python instalada en su sistema y ejecuta ese intérprete. Si este comando funciona y desea usarlo, simplemente use `py` en cualquier lugar donde vea el comando `python` o `python3` en este libro.

Volver a ejecutar el instalador

La razón más común por la que Python no funciona es que las personas olvidan seleccionar la opción Agregar Python a la RUTA cuando ejecutan el instalador; Este es un error fácil de cometer. La variable PATH es una configuración del sistema que le dice a Python dónde buscar programas de uso común. En este caso, Windows no sabe cómo encontrar el intérprete de Python.

La solución más sencilla en esta situación es ejecutar el instalador nuevamente. Si hay un instalador más nuevo disponible en <https://python.org>, descargue el nuevo instalador y ejecútelo, asegurándose de marcar la casilla Agregar Python a la RUTA .

Si ya tiene el instalador más reciente, ejecútelo nuevamente y seleccione Modificar opción. Verá una lista de funciones opcionales; mantenga las opciones predeterminadas seleccionadas en esta pantalla. Luego haga clic en Siguiente y marque la casilla Agregar Python a las variables de entorno . Finalmente, haga clic en Instalar. El instalador reconocerá que Python ya está instalado y agregará la ubicación del intérprete de Python a la variable PATH . Asegúrese de cerrar todas las terminales abiertas, porque seguirán usando la antigua variable PATH . Abra una nueva ventana de terminal y vuelva a ejecutar el comando `python` ; Deberías ver un mensaje de Python (`>>>`).

Python en MacOS

Las instrucciones de instalación del Capítulo 1 utilizan el instalador oficial de Python en <https://python.org>. El instalador oficial ha estado funcionando bien durante años, pero hay algunas cosas que pueden desviarlo. Esta sección le ayudará si algo no funciona de manera sencilla.

Instalar accidentalmente la versión de Python de Apple

Si ejecuta el comando `python3` y Python aún no está instalado en su sistema, lo más probable es que vea un mensaje indicando que las herramientas de desarrollo de la línea de comandos Es necesario instalarlo. El mejor enfoque en este punto es cerrar la ventana emergente que muestra este mensaje, descargar el instalador de Python desde <https://python.org>, y ejecute el instalador.

Si elige instalar las herramientas de desarrollador de línea de comandos en este punto, macOS instalará la versión de Python de Apple junto con las herramientas de desarrollador. El único problema con esto es que la versión de Python de Apple suele estar algo por detrás de la última versión oficial de Python. Sin embargo, aún puedes descargar y ejecutar el instalador oficial desde <https://python.org>. y `python3`

Luego apuntará a la versión más reciente. No se preocupe por tener instaladas las herramientas de desarrollo; Hay algunas herramientas útiles allí, incluido el sistema de control de versiones Git que se analiza en el Apéndice D.

Python 2 en versiones anteriores de macOS

En versiones anteriores de macOS, anteriores a Monterey (macOS 12), se instalaba de forma predeterminada una versión desactualizada de Python 2. En estos sistemas, el comando Python apunta al intérprete del sistema obsoleto. Si está utilizando una versión de macOS con Python 2 instalado, asegúrese de utilizar el comando python3 y siempre utilizará la versión de Python que instaló.

Pitón en Linux

Python está incluido de forma predeterminada en casi todos los sistemas Linux. Sin embargo, si la versión predeterminada de su sistema es anterior a Python 3.9, debe instalar la última versión. También puede instalar la última versión si desea las funciones más recientes, como los mensajes de error mejorados de Python. Las siguientes instrucciones deberían funcionar para la mayoría de los sistemas basados en apt.

Usando la instalación predeterminada de Python

Si desea utilizar la versión de Python a la que apunta python3 , asegúrese de tener instalados estos tres paquetes adicionales:

```
$ sudo apto instalar python3-dev python3-pip python3-venv
```

Estos paquetes incluyen herramientas que son útiles para desarrolladores y herramientas que le permiten instalar paquetes de terceros, como los que se utilizan en la sección de proyectos de este libro.

Instalación de la última versión de Python

Usaremos un paquete llamado deadsnakes, que facilita la instalación de múltiples versiones de Python. Ingrese los siguientes comandos:

```
$ sudo agregar-apt-repository ppa:deadsnakes/ppa
$ sudo actualización apta
$ sudo apto instalar python3.11
```

Estos comandos instalarán Python 3.11 en su sistema.

Ingrese el siguiente comando para iniciar una sesión de terminal que ejecute Python 3.11:

```
$ python3.11
>>>
```

En cualquier lugar donde vea el comando python en este libro, use python3.11 en cambio. También querrás utilizar este comando cuando ejecute programas desde la terminal.

Necesitará instalar dos paquetes más para aprovechar al máximo su instalación de Python:

```
$ sudo apto instalar python3.11-dev python3.11-venv
```

Estos paquetes incluyen módulos que necesitará al instalar y ejecutar paquetes de terceros, como los utilizados en los proyectos de la segunda mitad del libro.

NOTA paquete Deadsnakes se ha mantenido activamente durante mucho tiempo. cuando sea más nuevo salen versiones de Python, puedes usar estos mismos comandos, reemplazando python3.11 con la última versión disponible actualmente.

Comprobar qué versión de Python estás usando

Si tiene algún problema al ejecutar Python o al instalar paquetes adicionales, puede resultar útil saber exactamente qué versión de Python está utilizando. Es posible que tenga varias versiones de Python instaladas y no tenga claro qué versión se está utilizando actualmente.

Emita el siguiente comando en una terminal:

```
$ python --versión
Pitón 3.11.0
```

Esto le indica exactamente qué versión tiene actualmente el comando Python. apuntando a. El comando más corto python -V dará el mismo resultado.

Palabras clave de Python y funciones integradas

Python viene con su propio conjunto de palabras clave y funciones integradas. Es importante tener esto en cuenta cuando nombras cosas en Python: tus nombres no pueden ser los mismos que estas palabras clave y no deben ser los mismos que los nombres de las funciones, o sobrescribirás las funciones.

En esta sección, enumeraremos las palabras clave de Python y los nombres de funciones integradas. así sabrás qué nombres evitar.

Palabras clave de Python

Cada una de las siguientes palabras clave tiene un significado específico y verá un error si intenta utilizar cualquiera de ellas como nombre de variable.

| | | | | |
|-------------|----------------|---------------------------|-------------------|----------|
| FALSO | esperar | demás | importar | aprobar |
| Ninguno | descanso | excepto | en | aumentar |
| | que la | clase finalmente continúa | | devolver |
| Verdadero y | para lambda | | | intentar |
| como | | de | no local mientras | |
| afirmar | definición del | global si | no | con |
| asíncrono | elif | | o | producir |

Funciones integradas de Python

No obtendrá un error si utiliza una de las siguientes funciones integradas disponibles como nombre de variable, pero anulará el comportamiento de esa función:

abs() complejo() hash() min() segmento()
aiter() delattr() ayuda() siguiente() ordenado()
todo() dict() hexadecimal() objeto() método estático()
cualquiera() dir() id() oct() str()
anext() divmod() entrada() abrir() suma()
ascii() enumerar() int() ord() super()
bin() eval() isinstance() pow() tupla()
bool() ejecutivo() issubclass() print() tipo()
punto de interrupción() filtro() iter() propiedad() vars()
bytearray() float() len() rango() zip()
bytes() formato() lista() repr() __import__()
invocable() frozenset() locales() invertido()
chr() getattr() mapa() ronda()
método de clase() globals() max() conjunto()
compilar() hasattr() memoriaview() setattr()

B

EDITORES DE TEXTO E IDES



Los programadores dedican mucho tiempo a escribir, leer y editar código, y utilizar un editor de texto o un IDE (entorno de desarrollo integrado) para que este trabajo sea lo más eficiente posible es esencial. Un buen editor realizará tareas simples, como resaltar la estructura de su código para que pueda detectar errores comunes mientras trabaja. Pero no hará tanto que te distraiga de tu pensamiento. Los editores también tienen funciones útiles como sangría automática, marcadores para mostrar la longitud de línea adecuada y atajos de teclado para operaciones comunes.

Un IDE es un editor de texto con una serie de otras herramientas incluidas, como inter-Depuradores activos e introspección de código. Un IDE examina su código a medida que lo ingresa e intenta aprender sobre el proyecto que está creando. Por ejemplo, cuando empiezas a escribir el nombre de una función, un IDE puede mostrarte todos los argumentos que acepta esa función. Este comportamiento puede resultar muy útil cuando todo funciona y comprendes lo que estás viendo. Pero puede

También puede ser abrumador para un principiante y difícil de solucionar cuando no está seguro de por qué su código no funciona en el IDE.

Hoy en día, las líneas divisorias entre los editores de texto y los IDE se han desdibujado. Los editores más populares tienen algunas características que solían ser exclusivas de los IDE. Del mismo modo, la mayoría de los IDE se pueden configurar para ejecutarse en un modo más ligero que distrae menos mientras trabaja, pero le permite usar las funciones más avanzadas cuando las necesite.

Si ya tiene instalado un editor o IDE que le gusta y si ya está configurado para funcionar con una versión reciente de Python instalada en su sistema, le recomiendo que se quede con lo que ya sabe.

Explorar diferentes editores puede ser divertido, pero también es una forma de evitar el trabajo de aprender un nuevo idioma.

Si aún no tienes un editor o IDE instalado, te recomiendo VS Code por varias razones:

- Es gratis y se publica bajo una licencia de código abierto.
- Se puede instalar en todos los principales sistemas operativos.
- Es apto para principiantes pero también lo suficientemente potente como para que muchos profesionales los programadores lo utilizan como su editor principal.
- Encuentra las versiones de Python que tienes instaladas y normalmente no requiere ninguna configuración para ejecutar tus primeros programas.
- Tiene una terminal integrada, por lo que su salida aparece en la misma ventana que su código.
- Hay disponible una extensión de Python que hace que el editor sea altamente eficiente para escribir y mantener código Python.
- Es altamente personalizable, por lo que puedes ajustarlo para que coincida con tu forma de trabajar con código.

En este apéndice, aprenderá cómo comenzar a configurar VS Code para que funcione bien para usted. También aprenderá algunos atajos que le permitirán trabajar de manera más eficiente. Ser un mecanógrafo rápido no es tan importante como mucha gente piensa en programación, pero comprender a su editor y saber cómo usarlo de manera eficiente es muy útil.

Dicho todo esto, VS Code no funciona para todos. Si por alguna razón no funciona bien en su sistema, o si lo distrae mientras trabaja, existen otros editores que pueden resultarle más atractivos.

Este apéndice incluye una breve descripción de algunos de los otros editores e IDE que debería considerar.

Trabajar de manera eficiente con VS Code

En el Capítulo 1, instaló VS Code y también agregó la extensión Python. Esta sección le mostrará algunas configuraciones adicionales que puede realizar, además de atajos para trabajar eficientemente con su código.

Configurar el código VS

Hay algunas formas de cambiar los ajustes de configuración predeterminados para VS Code. Algunos cambios se pueden realizar a través de la interfaz y otros requerirán

cambios en los archivos de configuración. A veces, estos cambios tendrán efecto para todo lo que haga en VS Code, mientras que otros afectarán solo a los archivos dentro de la carpeta que contiene el archivo de configuración.

Por ejemplo, si tiene un archivo de configuración en su carpeta `python_work`, esa configuración solo afectará los archivos en esa carpeta (y sus subcarpetas).

Esta es una buena característica, porque significa que puede tener configuraciones específicas del proyecto que anulan su configuración global.

Usar tabulaciones y espacios

Si utiliza una combinación de tabulaciones y espacios en su código, puede causar problemas en sus programas que son difíciles de diagnosticar. Cuando se trabaja en un archivo `.py` con la extensión Python instalada, VS Code está configurado para insertar cuatro espacios cada vez que presiona la tecla `TAB`. Si está escribiendo solo su propio código y tiene instalada la extensión Python, probablemente nunca tendrá problemas con las tabulaciones y los espacios.

Sin embargo, es posible que su instalación de VS Code no esté configurada correctamente. Además, en algún momento, puedes terminar trabajando en un archivo que solo tiene pestañas o una combinación de pestañas y espacios. Si sospecha algún problema con las pestañas y los espacios, mire la barra de estado en la parte inferior de la ventana de VS Code y haga clic en Espacios o Tamaño de pestaña. Aparecerá un menú desplegable que le permitirá alternar entre el uso de pestañas y espacios. También puede cambiar el nivel de sangría predeterminado y convertir todas las sangrías del archivo en tabulaciones o espacios.

Si está viendo algún código y no está seguro de si la sangría consta de tabulaciones o espacios, resalte varias líneas de código. Esto hará visibles los espacios en blanco invisibles. Cada espacio se mostrará como un punto y cada pestaña se mostrará como una flecha.

NOTA programación, se prefieren los espacios a las tabulaciones porque todas las herramientas que trabajan con un archivo de código pueden interpretar los espacios sin ambigüedades. El ancho de las pestañas puede ser interpretado de forma diferente por diferentes herramientas, lo que genera errores que pueden ser extremadamente difíciles de diagnosticar.

Cambiar el tema del color

VS Code usa un tema oscuro de forma predeterminada. Si desea cambiar esto, haga clic en Archivo (Código en la barra de menú en macOS), luego haga clic en Preferencias y elija Tema de color. Aparecerá una lista desplegable que le permitirá elegir un tema que funcione bien para usted.

Configuración del indicador de longitud de línea

La mayoría de los editores le permiten configurar una señal visual, generalmente una línea vertical, para mostrar dónde deben terminar sus líneas. En la comunidad Python, la convención es limitar las líneas a 79 caracteres o menos.

Para configurar esta característica, haga clic en Código y luego en Preferencias, y luego elija Ajustes. En el cuadro de diálogo que aparece, ingrese reglas. Verás una configuración para

Editor: Gobernantes; haga clic en el enlace etiquetado Editar en settings.json. En el archivo que aparece, agregue lo siguiente a la configuración editor.rulers :

configuración.json

```
"editor.gobernantes": [
  80,
]
```

Esto agregará una línea vertical en la ventana de edición en la posición de 80 caracteres. Puedes tener más de una línea vertical; por ejemplo, si desea una línea adicional de 120 caracteres, el valor de su configuración sería [80, 120]. Si no ve las líneas verticales, asegúrese de haber guardado el archivo de configuración; Es posible que también deba salir y volver a abrir VS Code para que los cambios surtan efecto en algunos sistemas.

Simplificando la salida

De forma predeterminada, VS Code muestra el resultado de sus programas en una ventana de terminal integrada. Esta salida incluye los comandos que se utilizan para ejecutar el archivo. Para muchas situaciones, esto es ideal, pero puede distraerte más de lo que deseas cuando estás aprendiendo Python por primera vez.

Para simplificar el resultado, cierre todas las pestañas que están abiertas en VS Code y luego salga de VS Code. Inicie VS Code nuevamente y abra la carpeta que contiene los archivos Python en los que está trabajando; este podría ser simplemente el python_work carpeta donde se guarda hello_world.py .

Haga clic en el ícono Ejecutar/Depurar (que parece un triángulo con un pequeño error) y luego haga clic en Crear un archivo launch.json. Seleccione las opciones de Python en las indicaciones que aparecen. En el archivo launch.json que se abre, realice el siguiente cambio:

lanzamiento.json {

```
--recorte--
"configuraciones": [
  {
    --recorte--
    "consola": "Consola interna",
    "justMyCode": verdadero
  }
]
```

Aquí, estamos cambiando la configuración de la consola de terminal integrada a consola interna. Después de guardar el archivo de configuración, abra un archivo .py como hola_world.py y ejecútelo presionando CTRL-F5. En el panel de salida de VS Code, haga clic en Consola de depuración si aún no está seleccionado. Debería ver solo el resultado de su programa, y el resultado debe actualizarse cada vez que ejecute un programa.

NOTA La consola de depuración es de sólo lectura. No funcionará para archivos que usan la función input() , que comenzará a usar en el Capítulo 7. Cuando necesite ejecutar estos programas, puede cambiar la configuración de la consola nuevamente al Terminal integrado predeterminado, o puede ejecutar estos programas en una ventana de terminal separada como se describe en “Ejecución de programas Python desde una terminal” en la página 11.

Explorando más personalizaciones

Puede personalizar VS Code de muchas maneras para ayudarle a trabajar de manera más eficiente. Para comenzar a explorar las personalizaciones disponibles, haga clic en Código y luego en Preferencias, y luego elija Configuración. Verá una lista titulada De uso común; haga clic en cualquiera de los subtítulos para ver algunas formas comunes en las que puede modificar su instalación de VS Code. Tómese un tiempo para ver si hay alguno que haga que VS Code funcione mejor para usted, ¡pero no se pierda tanto en la configuración de su editor que posponga aprender a usar Python!

Atajos de código VS

Todos los editores e IDE ofrecen formas eficientes de realizar tareas comunes que todos deben realizar al escribir y mantener código. Por ejemplo, puede sangrar fácilmente una sola línea de código o un bloque completo de código; puedes mover con la misma facilidad un bloque de líneas hacia arriba o hacia abajo en un archivo.

Hay demasiados atajos para describirlos completamente aquí. Esta sección compartirá solo algunos que probablemente le resulten útiles mientras escribe sus primeros archivos Python. Si terminas usando un editor diferente a VS Code, asegúrate de aprender cómo realizar estas mismas tareas de manera eficiente en el editor que has elegido.

Bloques de código con sangría y sin sangría

Para sangrar un bloque completo de código, resáltelo y presione CTRL-] o -] en macOS. Para quitar la sangría de un bloque de código, resáltelo y presione CTRL-[o -[en macOS.

Comentar bloques de código

Para deshabilitar temporalmente un bloque de código, puede resaltar el bloque y comentarlo para que Python lo ignore. Resalte la sección de código que desea ignorar y presione CTRL-/ o -/ en macOS. Las líneas seleccionadas se comentarán con una almohadilla (#) con sangría al mismo nivel que la línea de código, para indicar que no son comentarios normales. Cuando desee descomentar el bloque de código, resalte el bloque y vuelva a emitir el mismo comando.

Mover líneas hacia arriba o hacia abajo

A medida que sus programas se vuelven más complejos, es posible que desee mover un bloque de código hacia arriba o hacia abajo dentro de un archivo. Para hacerlo, resalte el código que desea mover y presione ALT-flecha hacia arriba u Opción-flecha hacia arriba en macOS. La misma combinación de teclas con la flecha hacia abajo moverá el bloque hacia abajo en el archivo.

Si estás moviendo una sola línea hacia arriba o hacia abajo, puedes hacer clic en cualquier parte esa línea; No es necesario resaltar toda la línea para moverla.

Ocultar el Explorador de archivos

El explorador de archivos integrado en VS Code es realmente conveniente. Sin embargo, puede distraerte cuando escribes código y puede ocupar un espacio valioso en una pantalla más pequeña. El comando CTRL-B, o -B en macOS, alterna la visibilidad del panel del explorador de archivos.

Encontrar atajos adicionales

Trabajar de manera eficiente en un entorno de edición requiere práctica, pero también intención. Cuando esté aprendiendo a trabajar con código, intente notar las cosas que hace repetidamente. Cualquier acción que realices en tu editor probablemente tenga un atajo; Si hace clic en elementos del menú para realizar tareas de edición, busque los accesos directos para esas acciones. Si cambia con frecuencia entre el teclado y el mouse, busque los atajos de navegación que le impiden alcanzar el mouse con tanta frecuencia.

Puede ver todos los atajos de teclado en VS Code haciendo clic en Código y luego en Preferencias, y luego eligiendo Atajos de teclado. Puede usar la barra de búsqueda para encontrar un acceso directo en particular o puede desplazarse por la lista para encontrar accesos directos que puedan ayudarlo a trabajar de manera más eficiente.

Recuerde, es mejor concentrarse en el código en el que está trabajando y evitar dedicar demasiado tiempo a las herramientas que está utilizando.

Otros editores de texto e IDE

Escuchará y verá a personas que utilizan otros editores de texto. La mayoría de ellos se pueden configurar para ayudarlo de la misma manera que personalizó VS Code. Aquí hay una pequeña selección de editores de texto de los que quizás haya oído hablar.

INACTIVO

IDLE es un editor de texto incluido con Python. Es un poco menos intuitivo trabajar con él que con otros editores más modernos. Sin embargo, verás referencias a él en otros tutoriales dirigidos a principiantes, por lo que quizás quieras probarlo.

Geany

Geany es un editor de texto simple que muestra toda su salida en una ventana de terminal separada, lo que le ayuda a sentirse cómodo usando terminales. Geany tiene una interfaz muy minimalista, pero es lo suficientemente potente como para que un número significativo de programadores experimentados todavía la utilicen.

Si encuentra que VS Code le distrae demasiado y está lleno de demasiadas funciones, consulte Considere usar Geany en su lugar.

Texto sublime

Sublime Text es otro editor minimalista que deberías considerar usar si encuentras que VS Code está demasiado ocupado. Sublime Text tiene una interfaz realmente limpia y es conocido por funcionar bien incluso con archivos muy grandes. Es un editor que se apartará de tu camino y te permitirá concentrarte en el código que estás escribiendo.

Sublime Text tiene una prueba gratuita ilimitada, pero no es gratuita ni de código abierto. Si decide que le gusta y puede permitirse el lujo de comprar una licencia completa, debe hacerlo. La compra es una tarifa única; no es una suscripción de software.

Emacs y Vim

Emacs y Vim son dos editores populares preferidos por muchos programadores experimentados porque están diseñados para que puedas usarlos sin que tus manos tengan que dejar el teclado. Esto hace que escribir, leer y modificar código sea muy eficiente, una vez que aprenda cómo funciona el editor. También significa que ambos editores tienen una curva de aprendizaje bastante pronunciada. Vim está incluido en la mayoría de las máquinas Linux y macOS, y tanto Emacs como Vim se pueden ejecutar completamente dentro de una terminal. Por esta razón, se suelen utilizar para escribir código en servidores a través de sesiones de terminal remotas.

Los programadores a menudo le recomendarán que los pruebe, pero muchos Los programadores competentes olvidan cuánto están tratando de aprender los nuevos programadores. Es bueno conocer estos editores, pero debes posponer su uso hasta que te sientas cómodo trabajando con código en un editor más fácil de usar que te permita concentrarte en aprender a programar, en lugar de aprender a usar un editor.

PyCharm

PyCharm es un IDE popular entre los programadores de Python porque fue creado para funcionar específicamente con Python. La versión completa requiere una suscripción paga, pero también está disponible una versión gratuita llamada PyCharm Community Edition, y muchos desarrolladores la encuentran útil.

Si prueba PyCharm, tenga en cuenta que, de forma predeterminada, configura un entorno aislado para cada uno de sus proyectos. Esto suele ser algo bueno, pero puede provocar un comportamiento inesperado si no comprende lo que hace por usted.

Cuadernos Jupyter

Jupyter Notebook es un tipo de herramienta diferente a los editores de texto o IDE tradicionales, ya que es una aplicación web construida principalmente a partir de bloques; cada bloque es un bloque de código o un bloque de texto. Los bloques de texto se representan en Markdown, por lo que puedes incluir un formato simple en tus bloques de texto.

Los Jupyter Notebooks se desarrollaron para admitir el uso de Python en aplicaciones científicas, pero desde entonces se han ampliado para volverse útiles en una amplia variedad de situaciones. En lugar de simplemente escribir comentarios dentro de un .py archivo, puede escribir texto claro con formato simple, como encabezados, listas con viñetas e hipervínculos entre secciones de código. Cada bloque de código se puede ejecutar de forma independiente, lo que le permite probar pequeñas partes de su programa, o puede ejecutar todos los bloques de código a la vez. Cada bloque de código tiene su propia área de salida y puede activar o desactivar las áreas de salida según sea necesario.

Los Jupyter Notebooks pueden resultar confusos a veces debido a las interacciones entre diferentes celdas. Si define una función en una celda, esa función también estará disponible para otras celdas. Esto es beneficioso la mayor parte del tiempo, pero puede resultar confuso en portátiles más largos y si no comprende completamente cómo funciona el entorno de Notebook.

Si está realizando algún trabajo científico o centrado en datos en Python, podrá Es casi seguro que vea Jupyter Notebooks en algún momento.

C

OBTENIENDO AYUDA



Todo el mundo se queda estancado en algún momento cuando aprende a programar. Entonces, una de las habilidades más importantes que debe aprender como programador es cómo despegarse de manera eficiente. Este apéndice describe varias formas de ayudarlo a empezar de nuevo cuando la programación se vuelve confusa.

Primeros pasos

Cuando esté estancado, su primer paso debería ser evaluar su situación. Antes de pedir ayuda a otra persona, responda claramente las siguientes tres preguntas:

- ¿Qué estás tratando de hacer?
- ¿Qué has intentado hasta ahora?
- ¿Qué resultados has estado obteniendo?

Haga sus respuestas lo más específicas posible. Para la primera pregunta, declaraciones explícitas como "Estoy intentando instalar la última versión de Python en mi nueva computadora portátil con Windows" son lo suficientemente detalladas como para que otros miembros de la comunidad Python puedan ayudarlo. Declaraciones como "Estoy intentando instalar Python" no proporcionan suficiente información para que otros puedan ofrecer mucha ayuda.

Su respuesta a la segunda pregunta debe proporcionar suficientes detalles para que no le aconsejen que repita lo que ya intentó: "Fui a <https://python.org/descargas> y hice clic en el botón Descargar para mi sistema. Luego ejecuté el instalador" es más útil que "Fui al sitio web de Python y descargué algo".

Para la tercera pregunta, es útil saber los mensajes de error exactos que recibió, de modo que pueda usarlos para buscar una solución en línea o proporcionarlos cuando solicite ayuda.

A veces, simplemente responder estas tres preguntas antes de pedir ayuda a los demás te permite ver algo que te estás perdiendo y te ayuda a despegarte sin tener que ir más lejos. Los programadores incluso tienen un nombre para esto: depuración con patitos de goma. La idea es que si explicas claramente tu situación a un pato de goma (o cualquier objeto inanimado) y le haces una pregunta específica, a menudo podrás responder tu propia pregunta. Algunos equipos de programación incluso tienen a mano un pato de goma real para animar a la gente a "hablar con el pato".

Pruébalo otra vez

Simplemente volver al principio y volver a intentarlo puede ser suficiente para resolver muchos problemas. Digamos que estás intentando escribir un bucle for basado en un ejemplo de este libro. Es posible que solo se haya perdido algo simple, como dos puntos al final de la línea for . Repasar los pasos nuevamente podría ayudarle a evitar repetir el mismo error.

Tomar un descanso

Si llevas tiempo trabajando en el mismo problema, tomarte un descanso es una de las mejores tácticas que puedes probar. Cuando trabajamos en la misma tarea durante largos períodos de tiempo, nuestro cerebro comienza a concentrarse en una sola solución. Perdemos de vista las suposiciones que hemos hecho y tomar un descanso nos ayuda a tener una nueva perspectiva del problema. No es necesario que sea un descanso largo, sólo algo que te saque de tu forma de pensar actual. Si ha estado sentado durante mucho tiempo, haga algo físico: dé un paseo corto, salga un rato al aire libre o tal vez beba un vaso de agua o coma un refrigerio ligero.

Si te sientes frustrado, puede que valga la pena dejar tu trabajo a un lado por el día. Una buena noche de sueño casi siempre hace que el problema sea más abordable.

Consulte los recursos de este libro

Los recursos en línea para este libro, disponibles en https://ehmatthes.github.io/pcc_3e, incluya una serie de secciones útiles sobre cómo configurar su sistema y cómo trabajar en cada capítulo. Si aún no lo ha hecho, eche un vistazo a estos recursos y vea si hay algo que ayude en su situación.

Buscando en línea

Es muy probable que alguien más haya tenido el mismo problema que usted y haya escrito sobre ello en línea. Unas buenas habilidades de búsqueda y consultas específicas le ayudarán a encontrar recursos existentes para resolver el problema al que se enfrenta. Por ejemplo, si tiene dificultades para instalar la última versión de Python en un nuevo sistema Windows, buscar instalar Python en Windows y limitar los resultados a los recursos del último año puede conducirlo a una respuesta clara.

Buscar el mensaje de error exacto también puede resultar muy útil. Por ejemplo, supongamos que recibe el siguiente error cuando intenta ejecutar un programa Python desde una terminal en un nuevo sistema Windows:

```
> python hola_mundo.py
No se encontró Python; ejecutar sin argumentos para instalar desde Microsoft
Almacenar...
```

Al buscar la frase completa, “No se encontró Python; ejecutar sin argumentos para instalar desde Microsoft Store”, probablemente le dará buenos consejos.

Cuando comience a buscar temas relacionados con la programación, algunos sitios aparecerán repetidamente. Describiré algunos de estos sitios brevemente para que sepas lo útiles que pueden resultar.

Desbordamiento de pila

Desbordamiento de pila (<https://stackoverflow.com>) es uno de los sitios de preguntas y respuestas más populares para programadores y, a menudo, aparecerá en la primera página de resultados de búsquedas relacionadas con Python. Los miembros publican preguntas cuando están estancados y otros miembros intentan dar respuestas útiles. Los usuarios pueden votar por las respuestas que consideren más útiles, por lo que las mejores respuestas suelen ser las primeras que encontrará.

Muchas preguntas básicas de Python tienen respuestas muy claras en Stack Overflow, porque la comunidad los ha refinado con el tiempo. También se anima a los usuarios a publicar actualizaciones, por lo que las respuestas tienden a permanecer relativamente actualizadas. Al momento de escribir este artículo, casi dos millones de preguntas relacionadas con Python han sido respondidas en Stack Overflow.

Hay una expectativa que debes tener en cuenta antes de publicar en Stack Overflow. Las preguntas están destinadas a ser el ejemplo más breve del tipo de problema al que se enfrenta. Si publica entre 5 y 20 líneas de código que generan el error al que se enfrenta y si responde a las preguntas mencionadas en “Primeros pasos” en la página 477 anteriormente en este apéndice, probablemente alguien le ayudará. Si comparte un enlace a un proyecto con varios archivos grandes, es muy poco probable que la gente le ayude. Hay una excelente guía para redactar una buena pregunta en <https://stackoverflow.com/help/how-to-ask>. Las sugerencias de esta guía son aplicables para obtener ayuda en cualquier comunidad de programadores.

La documentación oficial de Python

La documentación oficial de Python (<https://docs.python.org>) es un poco más impredecible para los principiantes, porque su propósito es más documentar el

lenguaje que dar explicaciones. Los ejemplos de la documentación oficial deberían funcionar, pero es posible que no comprenda todo lo que se muestra. Aún así, es un buen recurso para comprobar cuando aparece en sus búsquedas, y le resultará más útil a medida que continúe desarrollando su comprensión de Python.

Documentación oficial de la biblioteca

Si está utilizando una biblioteca específica, como Pygame, Matplotlib o Django, los enlaces a la documentación oficial a menudo aparecerán en las búsquedas. Por ejemplo, <https://docs.djangoproject.com> Es muy útil cuando se trabaja con Django. Si planea trabajar con alguna de estas bibliotecas, es una buena idea familiarizarse con su documentación oficial.

r/aprendepython

Reddit se compone de varios subforos llamados subreddits. El r/learnpython subreddit (<https://reddit.com/r/learnpython>) es muy activo y solidario. Puedes leer las preguntas de otros y publicar las tuyas también. A menudo obtendrás múltiples perspectivas sobre las preguntas que plantees, lo que puede ser realmente útil para obtener una comprensión más profunda del tema en el que estás trabajando.

Publicaciones de blog

Muchos programadores mantienen blogs y comparten publicaciones sobre las partes del lenguaje con las que trabajan. Debe buscar una fecha en las publicaciones del blog que encuentre para ver qué tan aplicable es la información para la versión de Python que está utilizando.

Discordia

Discord es un entorno de chat en línea con una comunidad de Python donde puedes pedir ayuda y seguir discusiones relacionadas con Python.

Para comprobarlo, diríjase a <https://pythondiscord.com> y haga clic en Discord enlace en la parte superior derecha. Si ya tiene una cuenta de Discord, puede iniciar sesión con su cuenta existente. Si no tiene una cuenta, ingrese un nombre de usuario y siga las instrucciones para completar su registro en Discord.

Si es la primera vez que visitas Python Discord, deberás aceptar las reglas de la comunidad antes de participar plenamente. Una vez que hayas hecho eso, podrás unirte a cualquiera de los canales que te interesen. Si busca ayuda, asegúrese de publicarla en uno de los canales de ayuda de Python.

Flojo

Slack es otro entorno de chat en línea. A menudo se utiliza para comunicaciones internas de la empresa, pero también hay muchos grupos públicos a los que puede unirse.

Si desea consultar los grupos de Python Slack, comience con <https://pyslackers.com>.

Haga clic en el enlace de Slack en la parte superior de la página, luego ingrese su dirección de correo electrónico para recibir una invitación.

Una vez que esté en el espacio de trabajo de Python Developers, verá una lista de canales. Haga clic en Canales y luego elija los temas que le interesen. Quizás quieras comenzar con los canales `#help` y `#django`.

D

USANDO GIT FOR CONTROL DE VERSIONES



El software de control de versiones le permite tomar instantáneas de un proyecto siempre que esté en funcionamiento. Cuando realiza cambios en un proyecto (por ejemplo, cuando implementa una nueva característica), puede volver a un estado de trabajo anterior si el estado actual del proyecto no funciona b

El uso de software de control de versiones le brinda la libertad de trabajar en mejoras y cometer errores sin preocuparse por arruinar su proyecto. Esto es especialmente crítico en proyectos grandes, pero también puede ser útil en proyectos más pequeños, incluso cuando trabaja en programas contenidos en un solo archivo.

En este apéndice, aprenderá a instalar Git y usarlo para el control de versiones en los programas en los que está trabajando ahora. Git es el software de control de versiones más popular que se utiliza en la actualidad. Muchas de sus herramientas avanzadas ayudan a los equipos a colaborar en proyectos grandes, pero sus funciones más básicas también funcionan bien para desarrolladores individuales. Git implementa el control de versiones al rastrear los cambios realizados en cada archivo de un proyecto; Si comete un error, puede volver al estado guardado anteriormente.

Instalación de Git

Git se ejecuta en todos los sistemas operativos, pero existen diferentes enfoques para instalarlo en cada sistema. Las siguientes secciones proporcionan instrucciones específicas para cada sistema operativo.

Git se incluye en algunos sistemas de forma predeterminada y, a menudo, se incluye con otros paquetes que quizás ya haya instalado. Antes de intentar instalar Git, vea si ya está en su sistema. Abra una nueva ventana de terminal y ejecute el comando `git --version`. Si ve un resultado que enumera un número de versión específico, Git está instalado en su sistema. Si ve un mensaje que le solicita que instale o actualice Git, siga las instrucciones en pantalla.

Si no ve ninguna instrucción en pantalla y está usando Windows o macOS, puede descargar un instalador desde <https://git-scm.com>. Si es un usuario de Linux con un sistema compatible con apt, puede instalar Git con el comando `sudo apt install git`.

Configurando Git

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para hacer esto, Git necesita saber su nombre de usuario y correo electrónico. Debes proporcionar un nombre de usuario, pero puedes crear una dirección de correo electrónico falsa:

```
$ git config --global usuario.nombre "nombre de usuario"
$ git config --global usuario.correo electrónico "nombre de usuario@ejemplo.com"
```

Si olvida este paso, Git le solicitará esta información cuando realice su primera confirmación.

También es mejor establecer el nombre predeterminado para la rama principal de cada proyecto. Un buen nombre para esta rama es principal:

```
$ git config --global init.defaultBranch principal
```

Esta configuración significa que cada nuevo proyecto que utilices Git para administrar comenzará con una única rama de confirmaciones llamada main.

Hacer un proyecto

Hagamos un proyecto para trabajar. Cree una carpeta en algún lugar de su sistema llamada `git_practice`. Dentro de la carpeta, cree un programa Python simple:

```
hello_git.py print("¡Hola mundo Git!")
```

Usaremos este programa para explorar la funcionalidad básica de Git.

Ignorar archivos

Los archivos con la extensión `.pyc` se generan automáticamente a partir de archivos `.py`, por lo que no necesitamos Git para realizar un seguimiento de ellos. Estos archivos se almacenan en un directorio.

llamado `__pycache__`. Para decirle a Git que ignore este directorio, cree un archivo especial llamado `.gitignore` (con un punto al principio del nombre del archivo y sin extensión de archivo) y agréguele la siguiente línea:

```
.gitignore __pycache__/
```

Este archivo le dice a Git que ignore cualquier archivo en el directorio `__pycache__`. Usando un El archivo `.gitignore` mantendrá su proyecto ordenado y será más fácil trabajar con él.

Es posible que deba modificar la configuración de su explorador de archivos para que se muestren los archivos ocultos (archivos cuyos nombres comienzan con un punto). En el Explorador de Windows, marque la casilla en el menú Ver denominada Elementos ocultos. En macOS, presione `-SHIFT-` (punto). En Linux, busque una configuración denominada Mostrar archivos ocultos.

NOTA

Si estás en macOS, agrega una línea más a `.gitignore`. Agregue el nombre `.DS_Store`; Estos son archivos ocultos que contienen información sobre cada directorio en macOS y abarrotarán su proyecto si no los agrega a `.gitignore`.

Inicializando un repositorio

Ahora que tiene un directorio que contiene un archivo Python y un archivo `.gitignore`, puede inicializar un repositorio Git. Abra una terminal, navegue hasta `git_practice` carpeta y ejecute el siguiente comando:

```
git_practice$ git init
Repositorio Git vacío inicializado en git_practice/.git/
git_practice$
```

El resultado muestra que Git ha inicializado un repositorio vacío en `git_practice`. Un repositorio es el conjunto de archivos de un programa que Git rastrea activamente. Todos los archivos que utiliza Git para administrar el repositorio se encuentran en el directorio oculto `.git`, con el que no necesitarás trabajar en absoluto. Simplemente no elimines ese directorio o perderás el historial de tu proyecto.

Comprobando el estado

Antes de hacer nada más, veamos el estado del proyecto:

```
git_practice$ git status
1 en rama principal
Aún no hay compromisos
```

2 archivos sin seguimiento:

```
(use "git add <file>..." para incluir lo que se confirmará)
.gitignore
hola_git.py
```

3 no se agregó nada para confirmar, pero hay archivos sin seguimiento presentes (use "git add" para realizar un seguimiento)

```
git_practice$
```

En Git, una rama es una versión del proyecto en el que estás trabajando; aquí puedes ver que estamos en una rama llamada principal 1. Cada vez que verificas el estado de tu proyecto, debería mostrar que estás en la rama principal. Luego verá que estamos a punto de realizar el compromiso inicial. Una confirmación es una instantánea del proyecto en un momento determinado.

Git nos informa que los archivos sin seguimiento están en el proyecto 2, porque aún no le hemos dicho qué archivos rastrear. Luego se nos dice que no se ha agregado nada a la confirmación actual, pero que hay archivos sin seguimiento que quizás queramos agregar al repositorio 3.

Agregar archivos al repositorio

Agreguemos los dos archivos al repositorio y verifiquemos el estado nuevamente:

```
1 git_practice$ git agregar.
2 git_practice$ estado de git
  En la rama principal
  Aún no hay compromisos

Cambios a cometer:
  (use "git rm --cached <archivo>..." para cancelar el escenario)
3      nuevo archivo: .gitignore
      nuevo archivo: hola_git.py

git_practice$
```

El comando git add. agrega al repositorio todos los archivos dentro de un proyecto que aún no están siendo rastreados 1, siempre y cuando no estén listados en .gitignore. No confirma los archivos; simplemente le dice a Git que comience a prestarles atención. Cuando verificamos el estado del proyecto ahora, podemos ver que Git reconoce algunos cambios que deben confirmarse 2. La etiqueta nuevo archivo significa que estos archivos se agregaron recientemente al repositorio 3.

Hacer un compromiso

Hagamos el primer compromiso:

```
1 git_practice$ git commit -m "Proyecto iniciado".
2 [main (root-commit) cea13dd] Proyecto iniciado.
3 2 archivos modificados, 5 inserciones(+)
   crear modo 100644 .gitignore
   crear modo 100644 hello_git.py
4 git_practice$ estado de git
  En la rama principal
  nada que comprometer, trabajando árbol limpio
git_practice$
```

Emitimos el comando git commit -m "message" 1 para hacer una instantánea de el proyecto. El indicador -m le dice a Git que registre el mensaje que sigue (Iniciado

proyecto.) en el registro del proyecto. El resultado muestra que estamos en el camino principal. rama 2 y que dos archivos han cambiado 3.

Cuando verificamos el estado ahora, podemos ver que estamos en la rama principal, y tenemos un árbol de trabajo limpio 4. Este es el mensaje que debería ver cada vez que confirma un estado de trabajo de su proyecto. Si recibe un mensaje diferente, léalo atentamente; es probable que hayas olvidado agregar un archivo antes de realizar una confirmación.

Comprobando el registro

Git mantiene un registro de todas las confirmaciones realizadas en el proyecto. Revisemos el registro:

```
git_practice$ git registro
confirmar cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> principal)
Autor: eric <eric@example.com>
Fecha: lunes 6 de junio 19:37:26 2022 -0800
```

Proyecto iniciado.
git_practice\$

Cada vez que realiza una confirmación, Git genera una identificación de referencia única de 40 caracteres. Registra quién realizó la confirmación, cuándo se realizó y el mensaje grabado. No siempre necesitarás toda esta información, por lo que Git proporciona una opción para imprimir una versión más simple de las entradas del registro:

```
git_practice$ git log --pretty=una línea
cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Proyecto iniciado.
git_practice$
```

El indicador `--pretty=oneline` proporciona los dos datos más importantes: el ID de referencia de la confirmación y el mensaje grabado para la confirmación.

El segundo compromiso

Para ver el poder real del control de versiones, necesitamos realizar un cambio en el proyecto y confirmar ese cambio. Aquí simplemente agregaremos otra línea a `hello_git.py`:

```
hello_git.py print("¡Hola mundo Git!")
imprimir("Hola a todos.")
```

Cuando comprobamos el estado del proyecto, veremos que Git ha notado el archivo que cambió:

```
git_practice$ estado de git
1 en rama principal
```

Cambios no preparados para confirmación:

(use "git add <file>..." para actualizar lo que se confirmará)
(use "git restaurar <archivo>..." para descartar cambios en el directorio de trabajo)

2 modificado: hello_git.py

3 no se agregaron cambios al compromiso (use "git add" y/o "git commit -a")
git_practice\$

Vemos la rama en la que estamos trabajando 1, el nombre del archivo que se modificó 2 y que no se han confirmado cambios 3. Confirmemos el cambio y verifiquemos el estado nuevamente:

```
1 git_practice$ git commit -am "Saludo extendido".
   [principal 945fa13] Saludo extendido.
     1 archivo modificado, 1 inserción(+), 1 eliminación(-)
2 git_practice$ estado de git
   En la rama principal
     nada que comprometer, trabajando árbol limpio
3 git_practice$ git log --pretty=oneline
   945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> principal) Saludo extendido.
   cea13ddc51b885d05a410201a54faf20e0d2e246 Proyecto iniciado.
git_practice$
```

Hacemos una nueva confirmación, pasando las banderas -am cuando usamos el comando git commit 1. La bandera -a le dice a Git que agregue todos los archivos modificados en el repositorio a la confirmación actual. (Si crea archivos nuevos entre confirmaciones, vuelva a emitir el comando git add . para incluir los archivos nuevos en el repositorio). El indicador -m le indica a Git que registre un mensaje en el registro para esta confirmación.

Cuando comprobamos el estado del proyecto, vemos que nuevamente tenemos un árbol de trabajo limpio 2. Finalmente, vemos las dos confirmaciones en el registro 3.

Abandonando los cambios

Ahora veamos cómo abandonar un cambio y volver al estado de funcionamiento anterior.

Primero, agrega una nueva línea a hello_git.py:

```
hello_git.py print("¡Hola mundo Git!")
               imprimir("Hola a todos.")

               print("¡Oh no, rompí el proyecto!")
```

Guarde y ejecute este archivo.

Comprobamos el estado y vemos que Git nota este cambio:

```
git_practice$ estado de git
   En la rama principal

   Cambios no preparados para confirmación:
     (use "git add <file>..." para actualizar lo que se confirmará)
     (use "git restaurar <archivo>..." para descartar cambios en el directorio de trabajo)
```

1 modificado: hello_git.py

no se agregaron cambios al compromiso (use "git add" y/o "git commit -a")
git_practice\$

Git ve que modificamos `hello_git.py` 1 y podemos confirmar el cambio si queremos. Pero esta vez, en lugar de confirmar el cambio, volveremos a la última confirmación cuando sabíamos que nuestro proyecto estaba funcionando. No haremos nada con `hello_git.py`: no eliminaremos la línea ni usaremos la función `Deshacer` en el editor de texto. En su lugar, ingrese los siguientes comandos en su sesión de terminal:

```
git_practice$ git restaurar.
git_practice$ estado de git
En la rama principal
nada que comprometer, trabajando árbol limpio
git_practice$
```

El comando `git restaurar` nombre de archivo le permite abandonar todos los cambios desde la última confirmación en un archivo específico. El comando `git restaurar.` abandona todos los cambios realizados en todos los archivos desde la última confirmación; esta acción restaura el proyecto al último estado comprometido.

Cuando regreses a tu editor de texto, verás que `hello_git.py` ha vuelto a ser esto:

```
print("¡Hola mundo Git!")
imprimir("Hola a todos.")
```

Aunque volver a un estado anterior puede parecer trivial en este proyecto simple, si estuviéramos trabajando en un proyecto grande con docenas de archivos modificados, se restaurarían todos los archivos que habían cambiado desde la última confirmación. Esta característica es increíblemente útil: puedes realizar tantos cambios como quieras al implementar una nueva característica, y si no funcionan, puedes descartarlos sin afectar el proyecto. No es necesario recordar esos cambios y deshacerlos manualmente. Git hace todo eso por ti.

NOTA Es posible que tengas que actualizar el archivo en tu editor para ver la versión restaurada.

Comprobando confirmaciones anteriores

Puede volver a visitar cualquier confirmación en su registro mediante el comando de pago , utilizando los primeros seis caracteres de una ID de referencia. Después de comprobar y revisar una confirmación anterior, puede volver a la última confirmación o abandonar su trabajo reciente y retomar el desarrollo de la confirmación anterior:

```
git_practice$ git log --pretty=una línea
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> principal) Saludo extendido.
cea13ddc51b885d05a410201a54faf20e0d2e246 Proyecto iniciado.
git_practice$ git pago cea13d
Nota: cambiar a 'cea13d'.
```

- 1 Estás en el estado 'CABEZA separada'. Puedes mirar a tu alrededor, hacer experimentos. cambios y confirmarlos, y puede descartar cualquier confirmación que realice en este estado sin afectar ninguna rama al volver a una rama.

Si desea crear una nueva rama para conservar las confirmaciones que cree, puede hacerlo (ahora o más tarde) usando -c con el comando switch. Ejemplo:

```
git switch -c <nombre-nueva-rama>
```

2 O deshacer esta operación con:

```
interrupción de git -
```

Desactive este consejo estableciendo la variable de configuración asesamiento.detachedHead en falso

HEAD ahora está en cea13d Proyecto iniciado.

```
git_practice$
```

Cuando revisas una confirmación anterior, sales de la rama principal y ingrese lo que Git denomina estado HEAD separado 1. HEAD es el estado comprometido actual del proyecto; estás desconectado porque has dejado una rama con nombre (principal, en este caso).

Para volver a la rama principal , sigue la sugerencia 2 para deshacer la operación anterior:

```
git_practice$ git cambiar -
```

El puesto anterior de HEAD fue cea13d Proyecto iniciado.

Cambiado a la rama 'principal'

```
git_practice$
```

Este comando lo lleva de regreso a la rama principal . A menos que quieras Para trabajar con algunas características más avanzadas de Git, es mejor no realizar ningún cambio en su proyecto cuando haya verificado una confirmación anterior. Sin embargo, si es el único que trabaja en un proyecto y desea descartar todas las confirmaciones más recientes y volver a un estado anterior, puede restablecer el proyecto a una confirmación anterior. Trabajando desde la rama principal , ingrese lo siguiente:

```
1 git_practice$ estado de git
```

En la rama principal

nada que confirmar, directorio de trabajo limpio

```
2 git_practice$ git log --pretty=oneline
```

945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> principal) Saludo extendido.

cea13ddc51b885d05a410201a54faf20e0d2e246 Proyecto iniciado.

```
3 git_practice$ git reset --hard cea13d
```

HEAD está ahora en cea13dd Proyecto iniciado.

```
4 git_practice$ estado de git
```

En la rama principal

nada que confirmar, directorio de trabajo limpio

```
5 git_practice$ git log --pretty=oneline
```

cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Proyecto iniciado.

```
git_practice$
```

Primero verificamos el estado para asegurarnos de que estamos en la rama principal 1.

Cuando miramos el registro, vemos ambas confirmaciones 2. Luego emitimos el comando git reset --hard con los primeros seis caracteres del ID de referencia de

el compromiso al que queremos volver permanentemente 3. Verificamos el estado nuevamente y vemos que estamos en la rama principal sin nada que confirmar 4. Cuando miramos el registro nuevamente, vemos que estamos en el compromiso que queríamos para empezar de nuevo desde 5.

Eliminar el repositorio

A veces arruinarás el historial de tu repositorio y no sabrás cómo recuperarlo. Si esto sucede, primero considere pedir ayuda usando los enfoques discutidos en el Apéndice C. Si no puede solucionarlo y está trabajando en un proyecto en solitario, puede continuar trabajando con los archivos pero deshacerse del historial del proyecto. eliminando el directorio `.git`. Esto no afectará el estado actual de ninguno de los archivos, pero eliminará todas las confirmaciones, por lo que no podrá verificar ningún otro estado del proyecto.

Para hacer esto, abra un explorador de archivos y elimine el repositorio `.git` o elimínelo desde la línea de comando. Luego, deberá comenzar de nuevo con un repositorio nuevo para comenzar a rastrear sus cambios nuevamente. Así es como se ve todo este proceso en una sesión de terminal:

```
1 git_practice$ estado de git
   En la rama principal

   nada que confirmar, directorio de trabajo limpio
2 git_practice$ rm -rf .git/
3 git_practice$ estado de git
   fatal: No es un repositorio de git (ni ninguno de los directorios principales): .git
4 git_practice$ git inicio
   Repositorio Git vacío inicializado en git_practice/.git/
5 git_practice$ estado de git
   En la rama principal

   Aún no hay compromisos

Archivos sin seguimiento:
   (use "git add <file>..." para incluir lo que se confirmará)
   .gitignore
   hola_git.py

   no se agregó nada para confirmar, pero hay archivos sin seguimiento presentes (use "git add" para realizar un seguimiento)
6 git_practice$ git agregar.
   git_practice$ git commit -m "Empezar de nuevo".
   [principal (root-commit) 14ed9db] Comenzando de nuevo.
   2 archivos cambiados, 5 inserciones(+)
   crear modo 100644 .gitignore
   crear modo 100644 hola_git.py
7 git_practice$ estado de git
   En la rama principal

   nada que comprometer, trabajando árbol limpio
git_practice$
```

Primero verificamos el estado y vemos que tenemos un directorio de trabajo limpio 1. Luego usamos el comando `rm -rf .git/` para eliminar el directorio `.git` (del `.git` en Windows) 2. Cuando verificamos el estado después eliminando el `.git`

carpeta, se nos dice que este no es un repositorio de Git 3. Toda la información que Git usa para rastrear un repositorio se almacena en la carpeta .git , por lo que al eliminarla se elimina todo el repositorio.

Entonces somos libres de usar git init para iniciar un repositorio nuevo 4. Verificar el estado muestra que estamos de regreso en la etapa inicial, esperando la primera confirmación 5. Agregamos los archivos y realizamos la primera confirmación 6.

Comprobando el estado ahora nos muestra que estamos en la nueva rama principal sin nada que confirmar.

Usar el control de versiones requiere un poco de práctica, pero una vez que comiences a usarlo, no querrás volver a trabajar sin él.

mi

SOLUCIÓN DE PROBLEMAS DESPLIEGUES



Implementar una aplicación es tremendamente satisfactorio cuando funciona, especialmente si nunca lo has hecho antes. Sin embargo, pueden surgir muchos obstáculos en el proceso de implementación y, desafortunadamente, algunos de estos problemas pueden ser difíciles de identificar y abordar. Este apéndice le ayudará a comprender los enfoques modernos de implementación y le brindará formas específicas de solucionar problemas del proceso de implementación cuando las cosas no

Si la información adicional de este apéndice no es suficiente para ayudarle a realizar el proceso de implementación con éxito, consulte los recursos en línea en https://ehmatthes.github.io/pcc_3e; Es casi seguro que las actualizaciones allí le ayudarán a realizar una implementación exitosa.

Comprender las implementaciones

Cuando intenta solucionar problemas de un intento de implementación en particular, es útil tener una comprensión clara de cómo funciona una implementación típica.

La implementación se refiere al proceso de tomar un proyecto que funciona en su sistema local y copiarlo a un servidor remoto de una manera que le permita responder a las solicitudes de cualquier usuario en Internet. El entorno remoto difiere de un sistema local típico en varios aspectos importantes: probablemente no sea el mismo sistema operativo (SO) que el que está utilizando y lo más probable es que sea uno de muchos servidores virtuales en un único servidor físico. .

Cuando implementa un proyecto o lo envía al servidor remoto, se deben seguir los siguientes pasos:

- Crear un servidor virtual en una máquina física en un centro de datos.
- Establecer una conexión entre el sistema local y el servidor remoto.
- Copie el código del proyecto al servidor remoto.
- Identificar todas las dependencias del proyecto e instalarlas en el servidor remoto.
- Configurar una base de datos y ejecutar las migraciones existentes.
- Copie archivos estáticos (CSS, archivos JavaScript y archivos multimedia) a un lugar donde puedan servirse de manera eficiente.
- Iniciar un servidor para manejar las solicitudes entrantes.
- Comience a enrutar las solicitudes entrantes al proyecto, una vez que esté listo para manejarse. peticiones.

Cuando se considera todo lo que implica una implementación, no es de extrañar que las implementaciones fallen a menudo. Afortunadamente, una vez que comprenda lo que debería estar sucediendo, tendrá más posibilidades de identificar qué salió mal. Si puede identificar qué salió mal, es posible que pueda identificar una solución que hará que el próximo intento de implementación sea exitoso.

Puede desarrollar localmente en un tipo de sistema operativo y enviarlo a un servidor que ejecute un sistema operativo diferente. Es importante saber qué tipo de sistema está utilizando, porque eso puede informar parte de su trabajo de solución de problemas. Al momento de escribir este artículo, un servidor remoto básico en Platform.sh ejecuta Debian Linux; la mayoría de los servidores remotos son sistemas basados en Linux.

Solución de problemas básicos

Algunos pasos de solución de problemas son específicos de cada sistema operativo, pero llegaremos a eso en un momento. Primero, consideremos los pasos que todos deberían seguir al solucionar problemas de una implementación.

Su mejor recurso es el resultado generado durante el intento de envío. Este resultado puede parecer intimidante; Si eres nuevo en la implementación de aplicaciones, puede parecer muy técnico y, por lo general, hay mucho. La buena noticia es que no es necesario que comprenda todo lo que aparece en el resultado. Debe tener dos objetivos al hojear la salida del registro: identificar los pasos de implementación que funcionaron e identificar los pasos que no funcionaron. Si puedes hacer esto, es posible que estés

capaz de descubrir qué cambiar en su proyecto, o en su proceso de implementación, para que su próximo impulso sea exitoso.

Siga las sugerencias en pantalla

A veces, la plataforma a la que estás accediendo generará un mensaje que tiene una sugerencia clara sobre cómo abordar el problema. Por ejemplo, este es el mensaje que verá si crea un proyecto Platform.sh antes de inicializar un repositorio Git y luego intenta impulsar el proyecto:

```
$ empuje de plataforma
```

```
1 Ingrese un número para elegir un proyecto:
```

```
    [0] ll_project (votohz445ljyg)
    > 0
```

```
2 [Excepción raíz no encontrada]
```

```
    No se encontró la raíz del proyecto. Esto sólo se puede ejecutar desde dentro de un proyecto.
    directorio.
```

```
3 Para configurar el proyecto para este repositorio Git, ejecute:
```

```
    proyecto de plataforma: set-remote [id]
```

Estamos intentando impulsar un proyecto, pero el proyecto local aún no se ha asociado con un proyecto remoto. Entonces, la CLI de Platform.sh pregunta qué proyecto remoto queremos enviar a 1. Ingresamos 0 para seleccionar el único proyecto en la lista.

Pero a continuación, vemos una `RootNotFoundException` 2. Esto sucede porque Platform.sh busca un directorio `.git` cuando inspecciona el proyecto local, para descubrir cómo conectar el proyecto local con el proyecto remoto. En este caso, como no había ningún directorio `.git` cuando se creó el proyecto remoto, esa conexión nunca se estableció. La CLI sugiere una solución 3; nos dice que podemos especificar el proyecto remoto que debe asociarse con este proyecto local, usando el comando `proyecto:set-remote`.

Probemos esta sugerencia:

```
Proyecto de plataforma $ : set-remote votohz445ljyg
```

```
Configurar el proyecto remoto para este repositorio en: ll_project (votohz445ljyg)
```

```
El proyecto remoto para este repositorio es
```

```
    ahora configurado en: ll_project (votohz445ljyg)
```

En el resultado anterior, la CLI mostró el ID de este proyecto remoto, `votohz445ljyg`. Entonces ejecutamos el comando sugerido, usando esta ID, y la CLI puede establecer la conexión entre el proyecto local y el proyecto remoto.

Ahora intentemos impulsar el proyecto nuevamente:

```
$ empuje de plataforma
```

```
¿Está seguro de que desea pasar a la rama principal (producción)? [T/n] sí
```

```
Empujando HEAD al entorno existente principal
```

```
--recorte--
```

Este fue un impulso exitoso; Seguir la sugerencia en pantalla funcionó.

Debe tener cuidado al ejecutar comandos que no comprende completamente. Sin embargo, si tiene buenas razones para creer que un comando puede causar poco daño y si confía en la fuente de la recomendación, podría ser razonable probar las sugerencias que ofrecen las herramientas que está utilizando.

NOTA tenga en cuenta que hay personas que le indicarán que ejecute comandos que borrarán su sistema o lo expondrán a una explotación remota. Seguir las sugerencias de una herramienta proporcionada por una empresa u organización en la que confía es diferente a seguir las sugerencias de personas al azar en línea. Siempre que se trate de conexiones remotas, proceda con mucha precaución.

Leer la salida del registro

Como se mencionó anteriormente, el resultado del registro que ve cuando ejecuta un comando como `platform push` puede ser informativo e intimidante. Lea el siguiente fragmento de salida del registro, tomado de un intento diferente de usar la plataforma `push`, y vea si puede detectar el problema:

```
--recorte--
Recolectando tamiz de sopa==2.3.2.post1
Usando el sieve en caché-2.3.2.post1-py3-none-any.whl (37 kB)
Recopilando sqlparse==0.4.2
Usando sqlparse-0.4.2-py3-none-any.whl en caché (42 kB)
Instalación de paquetes recopilados: platformhconfig, sqlparse,...
Instalé exitosamente Django-4.1 asgiref-3.5.2 beautifulsoup4-4.11.1...
W: ERROR: No se pudo encontrar una versión que cumpla con el requisito gunicorn
W: ERROR: No se encontró ninguna distribución coincidente para gunicorn

130 archivos estáticos copiados en '/app/static'.

Ejecutando controles previos al vuelo...
--recorte--
```

Cuando falla un intento de implementación, una buena estrategia es revisar la salida del registro y ver si puede detectar algo que parezca advertencias o errores. Las advertencias son bastante comunes; a menudo son mensajes sobre próximos cambios en las dependencias de un proyecto, para ayudar a los desarrolladores a abordar los problemas antes de que causen fallas reales.

Un envío exitoso puede tener advertencias, pero no debería tener ningún error. En este caso, `Platform.sh` no pudo encontrar una manera de instalar el requisito `gunicorn`. Se trata de un error tipográfico en el archivo `requisitos_remote.txt`, que se suponía que incluía `gunicorn` (con una `r`). No siempre es fácil detectar la raíz del problema en la salida del registro, especialmente cuando el problema causa una serie de errores y advertencias en cascada. Al igual que cuando se lee un rastreo en su sistema local, es una buena idea observar detenidamente los primeros errores que se enumeran, y también los últimos errores. La mayoría de los errores intermedios tienden a ser paquetes internos que se quejan de que algo salió mal y pasan mensajes sobre el error a otros paquetes internos. El error real que podemos corregir suele ser uno de los primeros o últimos errores enumerados.

A veces podrá detectar el error y otras veces no tendrá idea de lo que significa el resultado. Ciertamente vale la pena intentarlo, y utilizar la salida del registro para diagnosticar con éxito un error es una sensación tremendamente satisfactoria. A medida que dedique más tiempo a revisar los resultados del registro, podrá identificar mejor la información que sea más significativa para usted.

Solución de problemas específicos del sistema operativo

Puede desarrollar en cualquier sistema operativo que desee y enviarlo a cualquier host que desee. Las herramientas para impulsar proyectos se han desarrollado lo suficiente como para modificar su proyecto según sea necesario para ejecutarse correctamente en el sistema remoto.

Sin embargo, pueden surgir algunos problemas específicos del sistema operativo.

En el proceso de implementación de Platform.sh, una de las fuentes más probables. Una de las dificultades es instalar la CLI. Aquí está el comando para hacerlo:

```
$ curl -fsS https://platform.sh/cli/installer | PHP
```

El comando comienza con curl, una herramienta que le permite solicitar recursos remotos, a los que se accede a través de una URL, dentro de una terminal. Aquí, se utiliza para descargar el instalador CLI desde un servidor Platform.sh. La sección -fsS del comando es un conjunto de indicadores que modifican la forma en que se ejecuta curl. La bandera f le dice a curl que suprima la mayoría de los mensajes de error, para que el instalador de CLI pueda manejarlos en lugar de informarle todos. La bandera s le dice a curl que se ejecute en silencio; permite al instalador de CLI decidir qué información mostrar en el terminal. La bandera S le dice a curl que muestre un mensaje de error si el comando general falla. El | php al final del comando le indica a su sistema que ejecute el archivo de instalación descargado usando un intérprete PHP, porque la CLI de Platform.sh está escrita en PHP.

Esto significa que su sistema necesita curl y PHP para instalar la CLI de Platform.sh. Para usar la CLI, también necesitará Git y una terminal que pueda ejecutar comandos Bash. Bash es un lenguaje que está disponible en la mayoría de los entornos de servidores. La mayoría de los sistemas modernos tienen mucho espacio para instalar múltiples herramientas como esta.

Las siguientes secciones le ayudarán a abordar estos requisitos para su sistema operativo. Si aún no tiene Git instalado, consulte las instrucciones para instalar Git en la página 484 en el Apéndice D y luego vaya a la sección aquí que corresponda a su sistema operativo.

NOTA Una herramienta excelente para comprender comandos de terminal como el que se muestra aquí es <https://explainshell.com>. Ingrese el comando que está tratando de comprender y el sitio le mostrará la documentación de todas las partes de su comando. Pruébelo con el comando utilizado para instalar la CLI Platform.sh.

Implementación desde Windows

Windows ha experimentado un resurgimiento en popularidad entre los programadores en los últimos años. Windows ha integrado muchos elementos diferentes de otros sistemas operativos, brindando a los usuarios una serie de opciones sobre cómo realizar trabajos de desarrollo local e interactuar con sistemas remotos.

Una de las dificultades más importantes al implementar desde Windows es que el sistema operativo principal de Windows no es el mismo que utiliza un servidor remoto basado en Linux. Un sistema Windows base tiene un conjunto diferente de herramientas y lenguajes que un sistema Linux base, por lo que para llevar a cabo el trabajo de implementación desde Windows, deberá elegir cómo integrar conjuntos de herramientas basadas en Linux en su entorno local.

Subsistema de Windows para Linux

Un enfoque popular es utilizar el Subsistema de Windows para Linux (WSL), un entorno que permite que Linux se ejecute directamente en Windows. Si tiene WSL configurado, usar la CLI de Platform.sh en Windows se vuelve tan fácil como usarlo en Linux. La CLI no sabrá que se está ejecutando en Windows; solo verá el entorno Linux en el que lo estás usando.

Configurar WSL es un proceso de dos pasos: primero instala WSL y luego elige una distribución de Linux para instalarla en el entorno WSL. Configurar un entorno WSL es más de lo que se puede describir aquí; Si está interesado en este enfoque y aún no lo tiene configurado, consulte la documentación en <https://docs.microsoft.com/en-us/windows/wsl/about>. Una vez que haya configurado WSL, puede seguir las instrucciones en la sección Linux de este apéndice para continuar con su trabajo de implementación.

Git bash

Otro enfoque para crear un entorno local que pueda implementar utiliza Git Bash, un entorno de terminal que es compatible con Bash pero se ejecuta en Windows. Git Bash se instala junto con Git cuando usa el instalador de <https://git-scm.com>. Este enfoque puede funcionar, pero no es tan sencillo como WSL. En este enfoque, tendrás que usar una terminal de Windows para algunos pasos y una terminal de Git Bash para otros.

Primero necesitarás instalar PHP. Puedes hacer esto con XAMPP, un paquete que combina PHP con algunas otras herramientas centradas en desarrolladores. Ir a <https://apachefriends.org> y haga clic en el botón para descargar XAMPP para Windows.

Abra el instalador y ejecútelo; Si ve una advertencia sobre las restricciones del Control de cuentas de usuario (UAC), haga clic en Aceptar. Acepte todos los valores predeterminados del instalador.

Cuando el instalador termine de ejecutarse, deberá agregar PHP a la ruta de su sistema; esto le indicará a Windows dónde buscar cuando desee ejecutar PHP. En el menú Inicio, ingrese la ruta y haga clic en Editar las variables de entorno del sistema; haga clic en el botón denominado Variables de entorno. Deberías ver la variable Ruta resaltada; haga clic en Editar debajo de este panel. Haga clic en Nuevo para agregar una nueva ruta a la lista actual de rutas. Suponiendo que mantuvo la configuración predeterminada al ejecutar el instalador de XAMPP, agregue C:\xampp\php en el cuadro que aparece, luego haga clic en Aceptar. Cuando haya terminado, cierre todos los cuadros de diálogo del sistema que aún estén abiertos.

Una vez cumplidos estos requisitos, puede instalar la CLI de Platform.sh. Necesitará utilizar una terminal de Windows con privilegios de administrador; ingrese el comando en el menú Inicio y en la aplicación Símbolo del sistema,

haga clic en Ejecutar como administrador. En la terminal que aparece, ingrese el siguiente comando:

```
> curl -fsS https://platform.sh/cli/installer | PHP
```

Esto instalará la CLI de Platform.sh, como se describió anteriormente.

Finalmente, trabajarás en Git Bash. Para abrir una terminal Git Bash, vaya al menú Inicio y busque git bash. Haga clic en la aplicación Git Bash que aparece; Deberías ver una ventana de terminal abierta. Puede utilizar comandos tradicionales basados en Linux como ls en esta terminal, así como comandos basados en Windows como dir. Para asegurarse de que la instalación se haya realizado correctamente, ejecute el comando de lista de plataformas . Debería ver una lista de todos los comandos en la CLI de Platform.sh. A partir de este momento, lleve a cabo todo su trabajo de implementación utilizando la CLI Platform.sh dentro de una ventana de terminal Git Bash.

Implementación desde macOS

El sistema operativo macOS no está basado en Linux, pero ambos se desarrollaron con principios similares. Lo que esto significa, en la práctica, es que muchos de los comandos y flujos de trabajo que utiliza en macOS también funcionarán en un entorno de servidor remoto. Es posible que necesites instalar algunos recursos centrados en desarrolladores para tener todas estas herramientas disponibles en tu entorno macOS local. Si recibe un mensaje para instalar las herramientas de desarrollador de línea de comandos en cualquier momento de su trabajo, haga clic en Instalar para aprobar la instalación.

La dificultad más probable al instalar la CLI de Platform.sh es asegurarse de que PHP esté instalado. Si ve un mensaje que indica que no se encuentra el comando php , deberá instalar PHP. Una de las formas más sencillas de instalar PHP es mediante el administrador de paquetes Homebrew , que facilita la instalación de una amplia variedad de paquetes de los que dependen los programadores. Si aún no tienes Homebrew instalado, visita <https://brew.sh> y siga las instrucciones para instalarlo.

Una vez que Homebrew esté instalado, use el siguiente comando para instalar PHP:

```
$ brew install php
```

Esto tardará un poco en ejecutarse, pero una vez que se haya completado, deberías estar capaz de instalar con éxito la CLI Platform.sh.

Implementación desde Linux

Debido a que la mayoría de los entornos de servidor están basados en Linux, no debería tener dificultades para instalar y utilizar la CLI de Platform.sh. Si intenta instalar la CLI en un sistema con una instalación nueva de Ubuntu, le dirá exactamente qué paquetes necesita:

```
$ curl -fsS https://platform.sh/cli/installer | PHP
El comando 'curl' no se encontró, pero se puede instalar con:
sudo apt install curl
El comando 'php' no se encuentra, pero se puede instalar con:
sudo apt install php-cli
```

El resultado real tendrá más información sobre algunos otros paquetes que funcionarían, además de información sobre la versión. El siguiente comando instalará curl y PHP:

```
$ sudo apto instalar curl php-cli
```

Después de ejecutar este comando, el comando de instalación de la CLI de Platform.sh debería ejecutarse correctamente. Dado que su entorno local es bastante similar a la mayoría de los entornos de alojamiento basados en Linux, gran parte de lo que aprenda sobre cómo trabajar en su terminal también se trasladará al trabajo en un entorno remoto.

Otros enfoques de implementación

Si Platform.sh no funciona para usted, o si desea probar un enfoque diferente, existen muchas plataformas de alojamiento para elegir. Algunos funcionan de manera similar al proceso descrito en el Capítulo 20 y otros tienen un enfoque muy diferente para llevar a cabo los pasos descritos al comienzo de este apéndice:

- Platform.sh le permite utilizar un navegador para llevar a cabo los pasos para los que utilizamos la CLI. Si le gustan más las interfaces basadas en navegador que los flujos de trabajo basados en terminales, es posible que prefiera este enfoque.
- Existen otros proveedores de alojamiento que ofrecen enfoques basados tanto en CLI como en navegador. Algunos de estos proveedores ofrecen terminales dentro de su navegador, por lo que no es necesario instalar nada en su sistema.
- Algunos proveedores le permiten enviar su proyecto a un sitio de alojamiento de código remoto como GitHub y luego conectar su repositorio de GitHub al sitio de alojamiento. Luego, el host extrae su código de GitHub, en lugar de solicitarle que envíe su código desde su sistema local directamente al host.
Platform.sh también admite este tipo de flujo de trabajo.
- Algunos proveedores ofrecen una variedad de servicios entre los que usted puede seleccionar, en para armar una infraestructura que funcione para su proyecto.
Por lo general, esto requiere que tenga una comprensión más profunda del proceso de implementación y de lo que necesita un servidor remoto para atender un proyecto. Estos hosts incluyen Amazon Web Services (AWS) y la plataforma Azure de Microsoft. Puede resultar mucho más difícil realizar un seguimiento de sus costos en este tipo de plataformas, porque cada servicio puede acumular cargos de forma independiente.
- Muchas personas alojan sus proyectos en un servidor privado virtual (VPS). En este enfoque, usted alquila un servidor virtual que actúa como una computadora remota, inicia sesión en el servidor, instala el software necesario para ejecutar su proyecto, copia su código, configura las conexiones correctas y permite que su servidor comience a aceptar solicitudes. .

Regularmente aparecen nuevas plataformas y enfoques de alojamiento; encuentre uno que le parezca atractivo e invierta tiempo en conocer el proceso de implementación de ese proveedor. Mantenga su proyecto el tiempo suficiente para llegar a

sepa qué funciona bien con el enfoque de su proveedor y qué no. Ninguna plataforma de alojamiento será perfecta; deberá tomar una decisión constante sobre si el proveedor que está utilizando actualmente es lo suficientemente bueno para su caso de uso.

Ofreceré una última advertencia sobre la elección de una plataforma de implementación y un enfoque general para la implementación. Algunas personas lo guiarán con entusiasmo hacia enfoques y servicios de implementación demasiado complejos que están destinados a hacer que su proyecto sea altamente confiable y capaz de atender a millones de usuarios simultáneamente. Muchos programadores dedican mucho tiempo, dinero y energía a desarrollar una estrategia de implementación compleja, sólo para descubrir que casi nadie está utilizando su proyecto. La mayoría de los proyectos de Django se pueden configurar en un plan de hosting pequeño y ajustarse para atender miles de solicitudes por minuto. Si su proyecto recibe menos de este nivel de tráfico, tómese el tiempo para configurar su implementación para que funcione bien en una plataforma mínima antes de invertir en infraestructura destinada a algunos de los sitios más grandes del mundo.

La implementación es increíblemente desafiante a veces, pero igualmente satisfactoria cuando su proyecto en vivo funciona bien. Disfruta el desafío y pide ayuda cuando la necesites.

ÍNDICE

Símbolos

+ (suma), 26 +=
(suma en su lugar), 122
* (argumentos arbitrarios), 146
** (argumentos de palabras clave arbitrarias), 148 {}
(llaves)
 diccionarios, 92
 conjuntos,
104 @ (decorador), 221, 424 /
(división), 26 ==
(igualdad), 72, 74
** (exponente), 26 >
(mayor que), 75 >= (mayor
o igual que), 75 # (marca de almohadilla),
para comentarios, 29 != (desigualdad), 74 <
(menor que), 75 <=
(menor o igual a), 75
[] (lista), 34 % (módulo), 116 +=
(cadenas
multilínea), 115
(multiplicación), 26 \n (nueva línea),
* 22 >>> (solicitud de Python),
4 - (resta), 26 \t
(pestaña), 22 (guión bajo) en
nombres de archivos y
carpetas, 10 en
_ números, 28 en
_ nombres de variables, 17

A

alias, 151–152, 178–179 alice.py,
195–197 Alien Invasion.
Véase también Pygame alienígenas, 256–
274 construyendo
 flota, 259–262 comprobando
 bordes, 265

colisiones, con balas, 267 colisiones,
con barco, 270–273 controlar la dirección
de la flota, 264–266 crear un
 alienígena,
256–258 dejar caer la flota, 265–266
llegar al final de la pantalla, 273–
274 reconstruir la flota, 268–269

balas, 247–253, 266–270
 colisiones, con extraterrestres, 267
 eliminar elementos antiguos,
250–251 disparar,
249–250 más
grandes, 268 limitar el número de, 251–
252
configuraciones, 247
acelerar, 269 clases
 extraterrestre, 257
Invasión alienígena, 229
Viñeta, 247–248
Botón, 278–279
Estadísticas del juego, 271
Cuadro de indicadores, 286–287
Configuración, 232
Barco, 234–235
finalizando el juego, 274–275
inicializando la configuración
 dinámica, 283–285
niveles
 modificar la configuración de
 velocidad, 283–285
restablecer la velocidad, 285
mostrar, 294–296 mover la
flota, 263–266 planificar, 228

Botón de reproducción, 278–283
Clase de botón , 278–279
desactivación, 282

Invasión alienígena (continuación)

- Botón de reproducción
 - (continuación) dibujo,
 - 279–280 ocultar el cursor del mouse, 282–
 - 283 restablecer el juego, 281–282
 - iniciar el juego, 281
- revisar el proyecto, 256 puntuación,
- 286–298 todos los
 - aciertos, 290
 - puntuación alta, 292–294
 - valores de puntos crecientes ,
 - 290–291
 - nivel, 294–296
 - número de barcos, 296–299
 - restablecer, 289–290
 - redondear y formatear, 291–292
 - atributo de
 - puntuación, 286 actualizar,
 - 289 configuraciones,
- almacenar, 232–233 barco, 233–
- 244 ajustar la
 - velocidad, 242–243 movimiento
 - continuo,
 - 239–242
 - encontrar una imagen, 233–234
 - rango límite, 243–244
- amusement_park.py, 80–82 y
- palabra clave, 75
- antialiasing, 279 API.

Ver programación de aplicaciones

- interfaz

apostrophe.py, 24–25

método append() , 37–38 interfaz

de programación de aplicaciones

- (API), 355 llamada API,
- 355–357 API GitHub,
- 368 API Hacker News,
- 368–371 procesamiento de una
- respuesta API, 357–362 límites de
- velocidad ,
- 362 solicitar datos,
- 356–357 visualizar resultados, 362–
- 368 argumentos, 131. Ver también

funciones como palabra clave, 151–152 afirmaciones,

213, 217–218 atributos, 159.

Ver también clases

B

banned_users.py, 76–77

bicicletas.py, 34–35

valores booleanos, 77

Bootstrap, 433. Véase también unxder Diccionarios de llaves

- Django ({}), 92
- conjuntos,
- 104 declaraciones de
- interrupción , 121 funciones integradas, 467

C

llamadas (funciones), 130, 132–135

car.py, 162–178

cars.py, 43–45, 72

ciudades.py,

121

- atributos de clases,
 - 159 acceso, 160
 - valores predeterminados, 163–
 - 164 modificación, 164–166
- creación , 158–161
- importación, 173–179
 - clases múltiples, 175–176 clases
 - individuales, 174–175 herencia,
 - 167–172 atributos y métodos,
 - 169 clases secundarias, 167–170
 - composición, 170 método
 - __init__() , 167–169
 - instancias como atributos, 170–172
 - métodos primordiales, 170 clases
 - principales, 170 subclases, 168
 - función super() , 168
 - superclases, 168
 - instancias, 157 métodos, 159
 - llamadas, 160
- encadenamiento,
- 185 método
 - __init__() , 159
 - modelado de
 - objetos del mundo real, 172-173

instancias múltiples, 161

convenciones de nomenclatura,

158 objetos,

157 pautas de estilo, 181

- archivos de valores separados por comas. Consulte los
- archivos CSV
- comment.py, 29
- comentarios, 29–30 pruebas condicionales, 72–77. Ver también si declaraciones
- confirmed_users.py, 124–125
- constantes, 28
- declaración de continuación , 122
- counting.py, 117–118, 122–123 archivos CSV, 330–341 función
 - csv.reader() , 330–333 verificación de errores, 338–341 encabezados de archivos, 330–332
- Análisis de datos D , 301
- bases de datos. Ver en Visualización
- de datos de Django, 301. Ver también
 - Matplotlib; Plotly módulo
- de fecha y hora , 333–335
- death_valley_highs_lows.py, 339–341 decoradores, 221–223, 423–425 atributos de clase de valores
 - predeterminados, 163–164
 - parámetros de función, 134–135 definición (funciones), 130 palabra clave def, 130 declaración del con
- diccionarios, 96
 - con listas, 38–40
 - dice_visual_d6d10.py, 326–327 dice_visual.py, 324–326
- diccionarios
 - definición, 92
 - vacío, 94
 - formato más grande, 96–97
 - KeyError, 98
 - pares clave-valor, 92
 - agregar, 93–94
 - eliminar, 96
 - recorrer claves, 101–
 - 102 claves en
 - orden, 102–103 pares clave-valor, 99–101 valores, 103–104
 - métodos get(), 97 –98
 - items(), 99–
 - 101
 - claves(), 101–103
 - valores(), 103–104 anidar
- diccionarios
 - en diccionarios, 110–111 diccionarios en listas, 105–108 listas en diccionarios, 108–109
 - ordenar, 94, 102–103 ordenar una lista
- de, 370 acceso a valores, 92–93, 97–98 modificación, 94–96
- die.py,
 - 320 die_visual.py, 320–321
 - dimensiones.py, 66–67
- div (HTML),
 - 437 division_calculator.py, 192–195 Django. Véase también Git; Aplicación
- de cuentas de proyecto Learning Log, 415–423 creación de aplicación, 415–416 cierre de sesión, 419–420 página de inicio de sesión, 416–419 página de registro, 420–423 sitio de administración, 381–386 asociación de datos con un usuario, 425–430 Bootstrap, 434–
- 445 tarjeta, 443 navegación plegable, 437
- contenedor, 440 aplicación
 - django-
 - bootstrap5, 434 documentación, 444 encabezados HTML, 435–436 jumbotron, 440–441 grupos de listas, 443 barra de navegación, 436–439 formularios de estilo, 441–442 comandos
 - crear superusuario, 382
 - vaciar, 427
 - realizar migraciones, 381, 385, 426 migrar, 377 ejecutar
 - servidor, 377–378, 383, 392 shell, 386
 - iniciar
 - aplicación, 379, 415
 - iniciar proyecto, 376
 - crear nuevos proyectos, 376

Eliminación en cascada de bases de

- datos de Django
- (continuación) ,
- 384 creación, 376 claves
- externas, 384, 425 relaciones de
- muchos a uno, 384 migración, 377,
- 381, 385, 426 campo no anulable, 427
- Postgres, 447
- consultas, 398, 428
- conjuntos de consultas, 386–387,
- 395, 398, 426–428
- restablecer, 427
- SQLite, 377
- implementación, 445–461, 493–501
 - confirmar el proyecto, 453 archivos
 - de configuración, 447–450 crear el
 - proyecto Platform.sh, 453–455 crear
 - superusuario, 456–457 páginas de
 - error personalizadas, 459–460
 - eliminar proyectos , 461
 - límites de prueba
 - gratuitos, 446
 - unicorn, 447 ignorar archivos,
 - 452–453 instalar Platform.sh CLI,
 - 446, 497–500
 - instalar platform
 - shconfig, 446
 - otros enfoques de
 - implementación, 500
 - Platform.sh, 445
 - Base de datos Postgres, 447,
 - 450–451
 - psycpg2, 447
 - impulsar un proyecto, 455
 - impulsar cambios, 458, 460
 - requisitos.txt, 446
 - proteger el proyecto, 457–460
 - configuraciones,
 - 451 sesiones SSH, 456–457
 - solución de problemas, 494–501
 - usar Git, 451 ver
 - el proyecto, 456 servidor
- de desarrollo, 377–378, 383, 392

documentación

- campos modelo, 380
- consultas, 388
- plantillas, 400
- formularios, 404–423, 429–430
 - csrf_token, 407
 - Solicitudes GET y POST, 406
 - ModelForm, 404, 408
 - procesamiento de formularios, 405–
 - 406, 409–410, 412–
 - 413, 421–422, 429–
 - 430 método save() , 405–406,
 - 409–410, 430
 - plantillas, 407, 410–411, 413 , 417,
 - 419, 422
 - validación, 404–406
 - widgets, 408
- HTML
 - etiqueta de anclaje (<a>),
 - 393 elemento <body> ,
 - 437 comentarios,
 - 437 elementos <div> ,
 - 437 elemento <main> ,
 - 440 márgenes,
 - 440 relleno, 440
 - elementos <p> , 391
 - elementos , 438
- HTTP 404 error, 428–429, 459–460
- INSTALLED_APPS, 380
- instalación, 375–376
- localhost, 378
- cerrar sesión, 419–420
- @login_required decorador, 423–424 plantilla
- de inicio de sesión, 417
- URL de mapeo, 388–390, 397–398 migrar la
- base de datos , 426–427 modelos, 379
- activación, 380–
 - 381 definición, 379, 384
 - claves externas, 384,
 - 425 registro con
 - administrador, 382–383, 385–
 - 386 Método __str__() ,
 - 380, 384 proyectos (vs. aplicaciones),
- 379 función de redirección
- () , 405–406 ciclo de liberación, 376
- restringir el acceso a
- los datos, 427–430

| | |
|--|--|
| configuraciones | números_pares.py, 58 |
| ALLOWED_HOSTS, | pares_o_odd.py, 117 |
| 451 DEBUG, 457–458 | excepciones, 183, 192–199 |
| INSTALLED_APPS, | decidir qué errores informar, 199 bloquear |
| 380–381, 415–416, 434 | más , 194–195 fallar |
| | silenciosamente, 198–199 |
| LOGIN_REDIRECT_URL, 417– | error FileNotFound , 195–196 |
| 418 LOGIN_URL, | manejar excepciones, 192–196 prevenir |
| 424 LOGOUT_REDIRECT_URL, | fallas , 193–195 bloques try-except , |
| 420 SECRET_KEY, 451 | 193 ZeroDivisionError, 192– |
| shell, 386–387, 426–427 iniciando una | 195 exponentes (**), 26 |
| aplicación, 379 estilos. Ver | |
| Django: | F |
| superusuarios de | favorito_languages.py, 96–97, 100–104, 109 |
| Bootstrap, 382, 456–457 | file_reader.py, 184–187 |
| etiquetas de bloque de | |
| plantillas, 393 | argumento de codificación de archivos , |
| plantilla | 195–196 error FileNotFound , 195– |
| secundaria, 393–394 | 196 rutas de |
| diccionario de contexto, 395 | archivo, 186 |
| filtros, 399 | absoluto, 186 método existe() , |
| formularios, 407 sangrías, | 203–204 módulo pathlib , |
| 393 herencia, 392–394 | 184 Objetos de ruta , 184–186, |
| saltos de línea , 399 enlaces, | 330 relativo, |
| 392– 393, 399 bucles, | 186 de cadenas , 198 |
| 395–397 plantilla principal, 392– | en Windows, 186 |
| 393 etiquetas de | método read_text() , 185, 195–196 método |
| plantilla, 393 marcas | splittlines(), 186–187 método write_text() , |
| de vista, 398–399 objetos de usuario , 418 | 190–191 first_numbers.py , 57 partidos, |
| escritura, 390–392 URL. Consulte | 221–223 banderas, 120– |
| Django: mapeo de URL | 121 flotadores, 26–28 |
| UserCreationForm, | alimentos.py, 63– |
| 421–422 valores de ID de | 64 bucles for , |
| usuario, 426 versiones, 376 funciones | 49–56, 99–104. |
| de vista, 388, 390 entornos | Ver también diccionarios; enumera |
| virtuales, 374–375 | formatted_name.py, |
| cadenas de documentos, 130, 153, 181 dog.py, 158–162 | notación de puntos, 150, 160 |
| | formato |
| | de cadenas f, 291–292 usando |
| E terremotos. Consulte mapeo de terremotos | variables en, 20–21 full_name.py, |
| electric_car.py, 167–173 | 21 funciones, 129– |
| módulo, 177–179 | 155 |
| argumento de codificación , 195–196 | argumentos |
| función enumerate() , 331 | arbitrarios, 146–149 |
| eq_explore_data.py, 343–347 | valores predeterminados, 134– |
| operador de igualdad (==), 72, 74 | 135 errores, |
| eq_world_map.py, 347– 352 | 136 palabras clave, 133–134 |

- funciones (continuación)
 - argumentos (continuación)
 - listas como, 142–
 - 145 opcionales, 138–
 - 139 posicionales, 131–
 - 133 cuerpo,
 - 130 integradas,
 - 467 funciones de llamada, 130, 132–
 - 135 definitorias,
 - 130 importadoras, 149–
 - 153 alias, 151–152
 - módulos completos, 150–151
 - funciones específicas, 151
 - modificar una lista en una función, 142–145
 - módulos, 149–153
 - parámetros, 131
 - valores de retorno, 137–
 - 141 pautas de estilo, 153

G

Archivos GeoJSON, 342–347, 350–351

solicitudes GET, 406. Consulte Django:

formularios

- para obtener
- ayuda Discord, 480 documentación oficial de
- Python, 479–480 recursos en línea, xxxv, 478 r/
- learnpython, 480 depuración del patito de goma, 478
- búsqueda
- en línea , 479 Slack,
- 481 Stack Overflow, 479 tres preguntas principales, 477–478 Git, 356, 451–453, 483–492. Véase también
- Django: implementación que abandona cambios, 488–489
- agregando
- archivos, 486 ramas, 486 verificando confirmaciones anteriores, 489–491
- confirmaciones, 486–
- 488 configuración, 452,
- 484 eliminando un repositorio, 491–492 .gitignore,
- 484 HEAD,
- 490 ignorando
- archivos, 484 inicializando un repositorio, 485

- instalación, 484
- registro,
 - 487 repositorios,
 - 356 estado, 485–
- 486 GitHub,
- 356 greeter.py, 114–115, 130–
- 131 greet_users.py, 142

H

Hacker News API, 368–371

marca de almohadilla (#), para comentarios, 29

hello_git.py, 484–491

hello_world.py, 10–12, 15–

19 archivos ocultos, 448,

485 hn_article.py, 368–369 hn_submissions .py, 369–371

I

IDE (entorno de desarrollo integrado), 469–470

si declaraciones

- y palabra clave, 75
- expresiones booleanas, 77
- comprobación
 - de igualdad (==),
 - 72 desigualdad (!=),
 - 74 elemento en
 - la lista, 76 elemento
 - no en la lista, 76 lista no
- vacía, 86–87 declaración
- elif , 80–83 declaración
- else , 79–80 declaraciones y listas
- if , 85–88 ignorando
- mayúsculas y minúsculas, 73–74
- comparaciones
- numéricas,
- 74–76 o palabra clave,
- 76 simple, 78 pautas de estilo, 89 prueba

de múltiples

condiciones, 82–83

inmutable, 65

importación *, 152, 177 importan

esto, 30 a 31 errores de

sangría, 53 a 56 errores de índice, 46 a 47

herencia, 167 a 173. Véase también la función de clases input() ,

- entrada numérica, 115–116
- indicaciones de escritura, 114–115

método `insert()` , 38
 función `itemgetter()` , método
 370 `items()` , 99–101

J Archivos

JSON Archivos GeoJSON, 342–347, 350–
 351 Formato de datos JSON,
 201 Función `json.dumps()` , 201–204, 343–
 344, 368 Función
`json.loads()` , 201–204, 343–344

Método `K claves()` , 101–103

pares clave-valor, 92. Consulte también diccionarios
 argumentos de palabras clave, 133–
 134 palabras clave, 466

L `language_survey.py`, 219

proyecto de registro de aprendizaje,
 373

archivos, 392
 404.html, 459
 500.html, 459 `cuentas/urls.py`,
 416, 420 `cuentas/views.py`, 421–
 422 `admin.py`, 382–383
`base.html`, 392–393, 396,
 418–419, 422, 435–440
`edit_entry.html`, 413
`form.py`, 404, 408–
 409 `.gitignore`, 452–453
`index.html`, 390–394, 440–441
`learning_logs/urls.py`, 389–390, 394–
 395, 397–398, 405, 409, 412

`learning_logs/views.py`, 390,
 395, 398, 405–406, 409–410,
 412–413, 423–425, 428–430

`ll_project/urls.py`, 388–389, 416
`login.html`, 417, 441–442
`models.py`, 379–380, 384
`new_entry.html`, 410
`new_topic.html`,
 407 `.platform.app.yaml`, 448–450

`registrarse.html`, 422
`requisitos.txt`, 446–447
 `rutas.yaml`, 450
`servicios.yaml`, 450
`configuraciones.py`, 380–381, 415–
 418, 420, 424, 434, 451,
 457–460

`topic.html`, 398–399, 443–444
`temas.html`, 395–396, 442–443

desarrollo en curso, 460 páginas,

391 editar

entrada, 412–414 página
 de inicio, 388–394 página
 de inicio de sesión, 416–
 419 nueva entrada, 408–
 411 tema nuevo, 404–
 408 registro, 420–423 tema,
 397–400 temas,
 394–397 escribir una

especificación (spec), 374 función `len()` , 44–

45 biblioteca, 184 Linux

Python

verificando la

versión instalada, 8 configuración, 8–
 12, 465–466 terminales que

ejecutan

programas, 12 inicio de sesión de
 Python, 9 solución de problemas

de instalación, 10 Código VS,
 instalación,

9 listas, 33 como argumentos,

142–145

comprensiones, 59–60 copia,
 63– 64 elementos accediendo,

34 accediendo por
 último, 35

agregando con

`append()`, 37–38

agregando con `insert()`, 38 identificando
 único, 104 modificando, 36–37

eliminando con `del`, 38–39

eliminando con `pop()`,

39–40 eliminar con `remove()`, 40–

41 vacío, 37–38 función `enumerate()` ,
 331

| | |
|---|---|
| listas (continuación) | mapeo de terremotos, 342–352. Consulte |
| errores | también |
| sangría, 53–56 índice, 46 | Descarga de datos de Plotly, 343, |
| bucles for , | 352 archivos GeoJSON, 342–347, 350–351 |
| 49–56 anidado, 108– | ordenamiento de latitud y longitud, 345 |
| 109, 261–262 índices, 34–35 | datos de ubicación, 346–347 |
| índice negativo, 35 | magnitudes, 346 mapa |
| índice cero, 34–35 | mundial, 347–348 Matplotlib |
| función len() , 44–45 | |
| denominación, 33– 34 | ejes |
| diccionarios | método set_aspect() , 313–314 |
| anidados | eliminación, 317 |
| en listas, 105–108 listas en | objetos ax , 303 |
| diccionarios, 108–109 listas numéricas, | mapas de colores, 310– |
| 56–60 función max() , 59 | 311 objetos fig , 303 |
| función min() , 59 | argumento figsize , 318 |
| función range() , 58–59 | argumento alfa de |
| función sum() , 59 eliminar todo | trazados de formato , 337–338 |
| apariciones de un valor, | estilos integrados, 306 |
| 125 sectores, 61–62 método de | colores personalizados, |
| clasificación | 310 etiquetas, 303– |
| inversa() , 44 | 304 grosor de línea, 303–304 |
| función | tamaño de |
| sorted() , 43–44 método | trazado, 318 sombreado, |
| sort() , 43 corchetes, 34 errores | 337–338 etiquetas de |
| lógicos, 54 método | marca, 309– |
| lstrip() , 22–23 | 310 galería, 302 |
| | instalación, 302 método plot() , |
| | 303–306 módulo pyplot , 302– |
| | 303 método savefig() , 311 |
| | guardar trazados , |
| | 311 método scatter() , 306–311 |
| | gráfico de líneas simples, 302–306 |
| | función subplots() , 303 métodos, |
| | 20 métodos |
| | auxiliares, 237 módulos, |
| | 149–152, 173–179. Ver también clases: |
| | importar; funciones: importar operador |
| | de módulo |
| | (%) , 116–117 motocicletas.py, 36–41 |
| | mountain_poll.py, 125– |
| | 126 mpl_squares.py, 302–306 |
| | my_car.py, 174–175 my_cars.py, |
| | 176–179 |
| | my_electric_car.py , 176 |

N

errores de nombre, 17–
 18 `name_function.py`, 211–217
`name.py`, 20
`nombres.py`, 211–212
 anidamiento. Ver diccionarios: anidamiento;
 listas: for
 bucles nueva línea (`\n`),
 21–22 función `next()` , 330–331
 Ninguno, 98,
 140 `number_reader.py`,
 202 números, 26–
 28 aritmética, 26
 constantes, 28
 exponentes, 26
 flotantes, 26–
 27 formateo, 291–292
 enteros, 26
 mezcla de enteros y flotantes, 27–28
 orden de operaciones, 26
 función `round()` , 291–292
 guiones bajos, 28
`number_writer.py`, 201

O

programación orientada a objetos
 (OOP), 157. Ver también clases
 o palabras clave, 76. Ver también declaraciones if

P

`pandas`, 320
 parámetros, 131
`parrot.py`, 114, 118–121
 declaración de paso , 198–
 199 rutas. Ver archivos: rutas
 de archivo PEP
 8, 68–69 `person.py`,
 139–140 `pets.py` , 125,
 132–136 `pip`,
 210–211 instalando Django, 374–
 376 instalando Matplotlib, 302
 instalando Plotly, 320
 instalando Pygame, 228
 instalación de `pytest`, 211
 instalación de solicitudes, 357
 Linux, instalación de `pip`, 465–466
 actualización, 210

`pi_string.py`, 187–189

`pizza.py`, 146–148

`Plataforma.sh`. Véase Django: implementación
 de jugadores.py,

61–62 `Plotly`, 302, 319. Véase también

 mapeo de terremotos; tipos de

 gráficos de dados

 rodantes, 322 personalización de gráficos,
 323, 325–326, 364

 documentación, 368 método

`fig.show()` , 322 método `fig.write_html()` ,

 327 formato de

 etiquetas de ejes

 de gráficos, 323 escalas

 de color, 349–350 texto flotante, 350–

 351, 365–366 enlaces en

 gráficos, 366–367 colores de

 marcador, 349–350, 367

 marcas de

 graduación, 325–326

 títulos, 323 información sobre
 herramientas,

 365–366 método `update_layout()` ,

 325–326, 364

 método

`update_traces()` ,

 367 galería, 320 histogramas, 322

 instalación,

 320 módulo

`plotly.express` , 322, 347, 368 alias de `px` ,

 322 función `px.bar()` ,

 322–323, 363–367 guardar figuras, 327

función `scatter_geo()` ,

347– 352 método `pop()` , 39–40 argumentos

 posicionales, 131–133. Ver

también funciones: argumentos

 Solicitudes POST,

406. Ver también Django:

 formularios `Printing_models.py`,

 143–145 Proyecto

 Gutenberg, 196–197

indicaciones, 114–115 Extensión de archivo `.py` , 15–16 `Pygame`.

 colores de fondo, 231–232 método

`clock.tick()` , 230–231 colisiones, 266–

 267, 270–271, 289–290 crear una
 ventana

 vacía, 229–230 cursor, ocultar, 282–283

Pygame (continuación)

- mostrando texto, 278–280
- finalizando juegos, 274–275
- bucles de eventos, 229–230
- velocidades de fotogramas, 230–231
- modo de pantalla completa, 245
- grupos
 - agregar elementos, 249–250
 - definir, 248–249
 - dibujar todos los elementos, 249–250, 257–258
 - vaciar, 268–269
 - recorrer, 249–251
 - eliminar elementos de, 250–251
 - actualizar todos los elementos, 248–249

- imágenes, 234–236
 - instalación, 228
 - niveles, 283–285
 - Botón de reproducción, 278–283
 - llamadas de print() , 251
 - salir, 244–245
 - rectificar objetos, 234–235
 - crear desde cero, 247–248
 - método get_rect() , posicionamiento 234–235, 234–235, 238–243, 247–248, 256–262, 278, 286–298
 - atributo de tamaño , 261
 - respondiendo a la entrada, 230
 - eventos, 230
 - pulsaciones de teclas, 238–242
 - clics del mouse, 281–283
 - coordenadas de pantalla, 235
 - superficies, 230
 - juegos de prueba, 268
- pytest. Consulte el código de prueba
- Python >>> , 4
 - funciones integradas, 467
 - verificando la versión instalada, 466
 - instalando
 - en Linux, 465–466
 - en macOS, 7–11, 464–465
 - en Windows, 5–6, 463–464

- intérprete, 15–16
- palabras clave, 466
- propuestas de mejora de Python (PEP), 68
- biblioteca
 - estándar, 179–180
 - sesiones de terminal, 4
 - en Linux, 9
 - en macOS, 7–8
 - en Windows, 6
 - versiones, 4
 - por qué usar Python, xxxvi
 - python_repos.

- py, 357–362
- python_repos_visual.py, 362–367

Valores de salida de Q , 118

R

- random_walk.py, 312–313
- paseos aleatorios, 312–318
 - función Choice() , 313
 - puntos para colorear, 315–316
 - método fill_walk() , 312–313
 - generación de recorridos múltiples, 314–315
 - trazado, 313–314
- Clase RandomWalk , 312–313
 - puntos iniciales y finales, 316–317
 - función range() , 58–59
 - método read_text() , 185, 195–196
 - refactorización, 204–206, 237–238, 260, 269–270
- recordar_me.py, 202–206
- método removeprefix() , 24
- método removesuffix() , 25
- Paquete de solicitudes, instalación, 357
- valores de retorno, 137–141
- rollercoaster.py, 116
- tirar dados, 319–327. Véase también Análisis
 - gráfico de resultados, 321–322
 - Clase de dado , 320
 - dados de diferentes tamaños, 326–327
 - función randint() , 320
 - lanzar dos dados, 324–326
- depuración del pato de goma, 478
- método rstrip() , 22–23
- rw_visual.py , 313–318

- S scatter_squares.py, 306–311
- conjuntos, 103–
- 104 sitka_highs_lows.py, 336–338
- sitka_highs.py, 330–336
- función sleep() , 272 cortes,
- 61–64 función
- sorted() , 43–44, 102–103 método sort() ,
- 43 método splitlines() ,
- 186–187 método split() , 196–197 Base
- de datos SQLite, 376–377
- square_numbers.py, 58–59
- squares.py, 59–60
- Desbordamiento de
- pila, 479 almacenamiento
- de datos, 201 –204. Consulte también archivos
- JSON que guardan y leen datos, 202–204
- cadena, 19–25
 - cambio de mayúsculas
 - y minúsculas, 20 f-strings, 20–
 - 21, 291–
 - 292 métodos
 - lower(), 20 lstrip(),
 - 22–23 removeprefix(), 23 –
 - 24 removesuffix(), 25
 - rstrip(), 22–23
 - split(), 196–197
 - líneas divididas(), 186–187
 - strip(), 22–23 título
 - () , 20
 - superior(), 20
 - líneas múltiples,
 - 115 nuevas líneas en ,
 - 21–22 comillas simples y dobles, 19, 24–
 - 25 tabulaciones,
 - 21–22 variables, 20–21
 - espacios en blanco, 21–
- 23 método strip() , 22–23
- método strptime() , 333–335 pautas
- de estilo , 68–69 líneas en
 - blanco, 69
 - CamelCase, 181
 - clases, 181
 - diccionarios, 96–97
 - funciones, 153
 - declaraciones if , 89
 - sangría, 68

- longitud de línea,
- 69 PEP 8,
- 68 encuesta.py,
- 218 errores de
 - sintaxis, 24 evitar con cadenas, 24–
- 25 resaltado de sintaxis, 16

t

- pestaña (\t), 21–
- 22 plantillas. Consulte el código
- de prueba de Django, 209–
 - 223 afirmaciones, 213, 217–
 - 218 pruebas fallidas, 214–
 - 216 cobertura
 - completa, 212 pruebas
 - de nombres, 213 pruebas
 - aprobadas, 212–
 - 214 pytest, 209–223
 - accesorios, 221–223
 - instalación, 210– 211
- pruebas en
 - ejecución, 213–214 casos de
 - prueba, 212 clases de prueba,
 - 217–223
- funciones de prueba, 211–217
- pruebas unitarias, 212
- test_name_function.py, 212–217 test_survey.py,
- 220–223 editores de texto
- e IDE. Véase
- también VS
- Code Emacs y Vim, 475
- Geany, 474 IDLE,
- 474 Jupyter Notebooks,
- 475 PyCharm, 475 Sublime
- Text, 474 paquete de
- terceros, 210 toppings.py, 74, 82–83
- tracebacks, 10, 17–18, 192 , 195–196
- bloques de
 - prueba
 - excepto . Ver tuplas
 - de excepciones,
- 65–67 definición, 65 bucle for , 66–67 escritura sobre, 67 errores de

U

- guión bajo (_) en
 - nombres de archivos y carpetas,
 - 10 en números, 28
 - en nombres de variables, 17

- pruebas unitarias,
- 212 `user_profile.py`, 148–149
- Método `V valores()` , 103–104
- variables, 16–19, 28
 - constantes, 28
 - como etiquetas, 18–
 - 19 asignación múltiple, 28
 - errores de nombre, 17–
 - 18 convenciones de nomenclatura,
 - 17 valores, 16 módulo `venv` ,
 - 374–375 control de versiones .
- Consulte Entornos virtuales Git, 374–
- 375 `vote.py`, 78–80
- Código VS, 4–5
 - configuración, 470–473
 - características, 469–
470
 - instalación en
 - Linux, 9 en
 - macOS, 8 en
 - Windows, 6 Extensión de
 - Python, 9–10 abrir archivos con Python,
 - 185 extensión de Python,
 - 9 archivos en
 - ejecución, 10 accesos
 - directos, 473–474 tabulaciones y espacios, 471

W.

- datos meteorológicos, 330–341. Ver también
 - archivos CSV;
- Matplotlib bucles `while` ,
 - 117–126 indicador activo ,
 - 120–121 declaración de
 - interrupción , 121 declaración
 - de continuación , 122 bucles
 - infinitos, 122–123 mover elementos entre
 - listas, 124 valores
 - de salida, 118 eliminar todos los elementos de la lista, 125
- espacios en blanco, 21–23. Consulte también
 - cadenas Rutas de
 - archivos
 - de Windows, 186 Configuración de
 - Python, 5–6, 9–12, 463–464 instalación
 - de solución
 - de problemas, 10 terminales que
 - ejecutan programas, 12 inicio de
 - sesión de Python, 6 Código
- VS, instalación, 6
- `word_count.py`, 197– 199
- `write_message.py`, 190–191 método `write_text()` , 190–191

Z

- Zen de Python, 30–31
- Error de división cero, 192–195

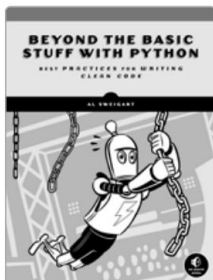
RECURSOS

Visite <https://nostarch.com/python-crash-course-3rd-edition> para erratas y más información.

Más libros sensatos de



SIN PRENSA DE ALMIDÓN



MÁS ALLÁ DE LO BÁSICO

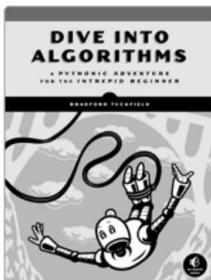
CON PITÓN

Mejores prácticas para escribir código limpio

por al sweigart

384 págs., \$34.95

isbn 978-1-59327-966-0



SUMERGIRSE EN ALGORITMOS

Una aventura pitónica para el

Principiante intrépido

por Bradford Tuckfield

248 págs., \$39.95

isbn 978-1-7185-0068-6



PITÓN ORIENTADO A OBJETOS

Domine la programación orientada a objetos mediante la creación de juegos y GUI

por irv kalb

416 págs., \$44.99

isbn 978-1-7185-0206-2



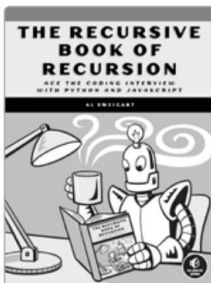
TARJETAS FLASH DE PITÓN

Sintaxis, conceptos y ejemplos

por eric matthes

101 tarjetas, \$27.95 isbn

978-1-59327-896-0



EL LIBRO RECURSIVO

DE RECURSIÓN

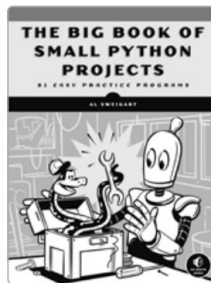
Entrevista con Ace the Coding con

Pitón y JavaScript

por al sweigart

328 págs., \$39.99

isbn 978-1-7185-0202-4



EL LIBRO GRANDE DEL PEQUEÑO

PROYECTOS DE PITÓN

81 programas de práctica fáciles

por al sweigart

432 págs., \$39.99

isbn 978-1-7185-0124-9

teléfono:

800.420.7240 o

415.863.9900

correo electrónico:

ventas@nostarch.com

web:

www.nostarch.com

**WORLDWIDE
BEST SELLER**



**LEARN
PYTHON—
FAST!**

Python Crash Course is the world's best-selling guide to the Python programming language. This fast-paced, thorough introduction will have you writing programs, solving problems, and developing functioning applications in no time.

You'll start by learning basic programming concepts, such as variables, lists, classes, and loops, and practice writing clean code with exercises for each topic. You'll also learn how to make your programs interactive and test your code safely before adding it to a project. You'll put your new knowledge into practice by creating a *Space Invaders*-inspired arcade game, building a set of data visualizations with Python's handy libraries, and deploying a simple application online.

As you work through the book, you'll learn how to:

- Use powerful Python libraries and tools, including pytest, Pygame, Matplotlib, Plotly, and Django
- Make increasingly complex 2D games that respond to keypresses and mouse clicks
- Generate interactive data visualizations using a variety of datasets

- Build apps that allow users to create accounts and manage their own data, and deploy your apps online
- Troubleshoot coding errors and solve common programming problems

New to this edition: This third edition is completely revised to reflect the latest in Python code. New and updated coverage includes VS Code for text editing, the pathlib module for file handling, pytest for testing your code, as well as the latest features of Matplotlib, Plotly, and Django.

If you've been thinking about digging into programming, *Python Crash Course* will provide you with the skills to write real programs fast. Why wait any longer? Start your engines and code!

ABOUT THE AUTHOR

Eric Matthes taught high school math and science for 25 years, as well as introductory Python classes. Matthes is involved with a number of open source projects and is a full-time writer and programmer.

Covers Python 3.x



THE FINEST IN GEEK ENTERTAINMENT™
nostarch.com

\$44.99 (\$59.99 CDN)

ISBN 978-1-7185-0270-3



9 781718 502703