3ª EDICIÓN

CURSO INTENSIVO DE PYTHON

INTRODUCCIÓN PRÁCTICA A LA PROGRAMACIÓN BASADA EN PROYECTOS

ERIC MATTHES

MÁS DE
1 500 000
DE COPIAS VENDIDAS
EN TODO EL MUNDO

no starch press

ANAYA MULTIMEDIA

ÍNDICE DE CONTENIDOS

PREFACIO A LA TERCERA EDICIÓN 19

INTRODUCCIÓN 21

PARTE I. LO BÁSICO 25

1. PRIMEROS PASOS 27

Configurar un entorno de programación....27
Versiones de Python....27
Ejecutar *snippets* de código en Python....28
Sobre el editor VS Code....28
Python en distintos sistemas operativos....29
Python en Windows....29
Python en macOS....30
Python en Linux....32
Ejecutar un programa Hello World....33
Instalar la extensión Python para VS Code....33
Ejecutar hello_world.py....34
Solución de problemas....34
Ejecutar programas de Python desde un terminal....35
En Windows....36
En macOS y Linux....36
Resumen....37

2. VARIABLES Y TIPOS DE DATOS SIMPLES 39

Lo que pasa en realidad cuando ejecutamos hello_world.py....39
Variables....40
Nombrar y usar variables....40
Evitar errores con los nombres al usar variables....41
Las variables son etiquetas....42
Cadenas....43
Cambiar mayúsculas y minúsculas en una cadena con métodos....44
Uso de variables en cadenas....45
Añadir espacios en blanco a cadenas con tabulaciones o nuevas líneas....46
Eliminar espacios en blanco....46
Eliminar prefijos....48
Evitar errores de sintaxis con cadenas....48
Números....50
Enteros....50
Flotantes....51
Enteros y flotantes....52
Guiones en números....52
Asignación múltiple....52
Constantes....53
Comentarios....53
¿Cómo se escriben los comentarios?....54
¿Qué tipo de comentarios debería escribir?....54
El Zen de Python....55
Resumen....56

3. INTRODUCCIÓN A LAS LISTAS 57

¿Qué es una lista?....57
Acceder a los elementos de una lista....58
Las posiciones de índice empiezan en 0, no en 1....58
Usar valores individuales de una lista....59
Modificar, añadir y eliminar elementos....60
Modificar elementos en una lista....60
Añadir elementos a una lista....61
Eliminar elementos de una lista....62
Organizar una lista....66
Ordenar una lista de manera permanente con el método sort()....67
Ordenar una lista temporalmente con la función sorted()....67
Imprimir una lista en orden inverso....68
Descubrir la longitud de una lista....69
Evitar errores de índice al trabajar con listas....70
Resumen....71

4. TRABAJO CON LISTAS 73

Pasar en bucle por una lista completa....73
Los bucles en detalle....74
Sacar más partido a un bucle for....75
Hacer algo después de un bucle for....76
Evitar errores de sangrado....77
Olvidar la sangría....77
Olvidar sangrar líneas adicionales....78
Sangrados innecesarios....78
Sangrado innecesario después de un bucle....79
Olvidar los dos puntos....80
Hacer listas numéricas....81
Utilizar la función range()....81
Usar range() para hacer una lista de números....82
Estadística sencilla con una lista de números....83
Listas por comprensión....84
Trabajar con parte de una lista....85
Partir una lista....85
Pasar en bucle por un trozo....87
Copiar una lista....87
Tuplas....90
Definir una tupla....90
Pasar en bucle por todos los valores de una tupla....91
Sobrescribir una tupla....92

Dar estilo a nuestro código 93
La guía de estilo 93
Sangrado 93
Longitud de línea 94
Líneas en blanco 94
Otras directrices de estilo 94
Resumen 95

5. SENTENCIAS IF 97

Un ejemplo sencillo 97
Pruebas condicionales 98
Comprobar la igualdad 98
Ignorar mayúsculas y minúsculas al comprobar la igualdad 99
Comprobar la desigualdad 100
Comparaciones numéricas 100
Comprobar varias condiciones 101
Comprobar si hay un valor en una lista 102
Comprobar si un valor no está en una lista 103
Expresiones booleanas 103
Sentencias if 104
Sentencias if simples 104
Sentencias if-else 105
La cadena if-elif-else 106
Utilizar múltiples bloques elif 107
Omitir el bloque else 108
Probar múltiples condiciones 109
Utilizar sentencias if con listas 111
Detectar elementos especiales 111
Comprobar que una lista no está vacía 113
Usar múltiples listas 113
Dar estilo a las sentencias if 115
Resumen 116

6. DICCIONARIOS 117

Un diccionario sencillo 117
Trabajar con diccionarios 118
Acceder a los valores de un diccionario 118
Añadir nuevos pares clave-valor 119
Empezar con un diccionario vacío 120
Modificar valores en un diccionario 120
Eliminar pares clave-valor 122
Un diccionario de objetos similares 122
Usar get() para acceder a valores 124
Pasar en bucle por un diccionario 125
Pasar en bucle por todos los pares clave-valor 125
Pasar en bucle por todas las claves del diccionario 127
Pasar en bucle por las claves de un diccionario en un orden particular 129
Pasar en bucle por todos los valores de un diccionario 129

Anidación 131
Una lista de diccionarios 132
Una lista en un diccionario 134
Un diccionario en un diccionario 136
Resumen 138

7. ENTRADA DEL USUARIO Y BUCLES WHILE 139

Cómo funciona la función input() 140
Escribir indicaciones claras 140
Usar int() para aceptar entrada numérica 141
El operador módulo 142
Introducción a los bucles while 143
El bucle while en acción 144
Dejar que el usuario elija cuándo salir 144
Usar una bandera 146
Usar break para salir de un bucle 147
Usar continue en un bucle 148
Evitar bucles infinitos 148
Usar un bucle while con listas y diccionarios 150
Pasar elementos de una lista a otra 150
Eliminar todos los casos de valores específicos de una lista 151
Rellenar un diccionario con entrada del usuario 152
Resumen 154

8. FUNCIONES 155

Definir una función 155
Pasar información a una función 156
Argumentos y parámetros 157
Pasar argumentos 157
Argumentos posicionales 157
Argumentos de palabra clave 159
Valores predeterminados 160
Llamadas a funciones equivalentes 161
Evitar errores con argumentos 162
Valores de retorno 163
Devolver un solo valor 163
Hacer un argumento opcional 164
Devolver un diccionario 165
Usar una función con un bucle while 166
Pasar una lista 168
Modificar una lista en una función 169
Evitar que una función modifique una lista 171
Pasar un número arbitrario de argumentos 172
Mezclar argumentos posicionales y arbitrarios 174
Usar argumentos de palabra clave arbitrarios 174
Guardar las funciones en módulos 176
Importar un módulo completo 176
Importar funciones específicas 178

Usar as para dar un alias a una función 178
Usar as para dar un alias a un módulo 179
Importar todas las funciones de un módulo 179
Dar estilo a las funciones 180
Resumen 181

9. CLASES 183

Crear y usar una clase 184
Creación de la clase Dog 184
El método __init__() 185
Hacer una instancia de una clase 185
Trabajar con clases e instancias 188
La clase Car 188
Establecer un valor predeterminado para un atributo 189
Modificar el valor de un atributo 190
Herencia 193
El método __init__() para una clase derivada 193
Definir atributos y métodos para la clase derivada 195
Anular métodos de la clase base 196
Instancias como atributos 196
Modelar objetos del mundo real 198
Importar clases 200
Importar una sola clase 200
Almacenar varias clases en un módulo 201
Importar varias clases desde un módulo 202
Importar un módulo entero 203
Importar todas las clases de un módulo 203
Importar un módulo en otro módulo 204
Usar alias 205
Encontrar su propio flujo de trabajo 205
La biblioteca estándar de Python 206
Dar estilo a las clases 207
Resumen 208

10. ARCHIVOS Y EXCEPCIONES 209

Leer de un archivo 209
Leer los contenidos de un archivo 210
Rutas de archivo relativas y absolutas 212
Acceder a las líneas de un archivo 213
Trabajar con el contenido de un archivo 213
Archivos grandes: un millón de números 214
¿Está su cumpleaños contenido en pi? 215
Escribir a un archivo 216
Escribir una línea 217
Escribir múltiples líneas 217
Excepciones 218
Manejar la excepción ZeroDivisionError 219
Usar bloques try-except 219

Usar excepciones para evitar fallos 220
El bloque else 221
Manejar la excepción FileNotFoundError 222
Analizar texto 223
Trabajar con múltiples archivos 224
Fallos silenciosos 225
Decidir qué errores informar 226
Almacenar datos 227
Utilizar json.dumps() y json.loads() 228
Guardar y leer datos generados por usuarios 229
Refactorización 231
Resumen 234

11. PROBAR EL CÓDIGO 235

Instalar pytest con pip 236
Actualizar pip 236
Instalar pytest 237
Probar una función 237
Pruebas unitarias y casos de prueba 238
Una prueba que pasa 239
Ejecutar una prueba 240
Una prueba que falla 241
Responder a una prueba fallida 242
Añadir pruebas nuevas 243
Probar una clase 244
Varios métodos assert 244
Una clase para probar 245
Probar la clase AnonymousSurvey 247
Configuración de pruebas 249
Resumen 251

PARTE II. PROYECTOS 253

Alien Invasion: Hacer un juego con Python 253
Visualización de datos 253
Aplicaciones web 254

12. UNA NAVE QUE DISPARA BALAS 255

Planificación del proyecto 256
Instalar Pygame 256
Iniciar el proyecto del juego 256
Crear una ventana de Pygame y responder a entrada de usuario 257
Controlar la tasa de *frames* 258
Configurar el color de fondo 259
Crear una clase Settings 260
Añadir la imagen de la nave 261
Crear la clase Ship 262
Dibujar la nave en la pantalla 264

Refactorización: Los métodos _check_events() y _update_screen() 265
El método _check_events() 265
El método _update_screen() 266
Pilotar la nave 267
Responder a pulsaciones de teclas 267
Permitir un movimiento continuo 268
Movimiento hacia la izquierda y hacia la derecha 269
Ajustar la velocidad de la nave 271
Limitar el alcance de la nave 272
Refactorización de _check_events() 273
Pulsar Q para salir 274
Ejecutar el juego en modo pantalla completa 274
Un resumen rápido 275
alien_invasion.py 275
settings.py 275
ship.py 275
Disparar balas 276
Añadir la configuración de las balas 276
Crear la clase Bullet 276
Agrupar balas 278
Disparar balas 279
Borrar las balas viejas 280
Limitar el número de balas 281
Crear el método _update_bullets() 282
Resumen 283

13. ¡ALIENÍGENAS! 285

Revisión del proyecto 285
Crear el primer alien 286
Crear la clase Alien 287
Crear una instancia de Alien 287
Crear la flota extraterrestre 289
Crear una fila de alienígenas 289
Refactorización de _create_fleet() 291
Añadir filas 292
Hacer que se mueva la flota 294
Mover los aliens hacia la derecha 294
Crear configuraciones para la dirección de la flota 296
Comprobar si un alien ha llegado al borde 296
Descenso de la flota y cambio de dirección 297
Disparar a los aliens 298
Detectar colisiones de balas 298
Hacer balas más grandes para pruebas 299
Repoblar la flota 300
Acelerar las balas 301
Refactorización de _update_bullets() 301
Fin del juego 302
Detectar colisiones entre un alien y la nave 302
Responder a colisiones entre aliens y la nave 303

Aliens que llegan al fondo de la pantalla 306
Game Over 307
Identificar cuándo deberían ejecutarse partes del juego 307
Resumen 308

14. PUNTUACIÓN 309

Añadir el botón Play 309
Crear una clase Button 310
Dibujar el botón en la pantalla 311
Iniciar el juego 313
Reiniciar el juego 313
Desactivar el botón Play 314
Ocultar el cursor del ratón 314
Subir de nivel 316
Modificar las configuraciones de velocidad 316
Restablecer la velocidad 317
Puntuaciones 318
Mostrar la puntuación 319
Crear un marcador 320
Actualizar la puntuación a medida que se abaten aliens 321
Restablecer la puntuación 322
Asegurarse de contabilizar todos los aciertos 323
Aumentar los valores en puntos 323
Redondear la puntuación 324
Puntuaciones altas 325
Mostrar el nivel 327
Mostrar el número de naves 330
Resumen 334

15. GENERAR DATOS 335

Instalar Matplotlib 336
Trazar un gráfico de líneas sencillo 336
Cambiar el tipo de etiqueta y el grosor de la línea 337
Corregir el trazado 338
Utilizar estilos integrados 339
Trazar puntos individuales y darles estilo con scatter() 341
Trazar una serie de puntos con scatter() 342
Calcular datos automáticamente 343
Personalizar las etiquetas de los puntos de los ejes 345
Definir colores personalizados 345
Utilizar un mapa de color 345
Guardar los trazados automáticamente 346
Caminatas aleatorias 347
Crear la clase RandomWalk() 347
Elegir direcciones 348
Trazar una caminata aleatoria 349
Generar múltiples caminatas aleatorias 350
Dar estilo a la caminata 351

Tirar dados con Plotly 355
Instalar Plotly 356
Crear la clase Die 356
Tirar el dado 357
Analizar los resultados 357
Hacer un histograma 358
Personalizar el gráfico 359
Tirar dos dados 361
Más personalizaciones 362
Tirar dados de distinto tamaño 363
Guardar figuras 364
Resumen 365

16. DESCARGAR DATOS 367

El formato de archivo CSV 368
Analizar los encabezados de archivo CSV 368
Imprimir los encabezados y sus posiciones 369
Extraer y leer datos 370
Trazar datos en un gráfico de temperatura 370
El módulo datetime 372
Trazar fechas 373
Trazar un periodo de tiempo más largo 373
Trazar una segunda serie de datos 374
Sombrear un área del gráfico 375
Comprobación de errores 376
Descargar sus propios datos 380
Mapear conjuntos de datos globales: formato JSON 381
Descargar datos de terremotos 381
Examinar datos JSON 382
Hacer una lista con todos los terremotos 384
Extraer magnitudes 385
Extraer datos de ubicación 385
Crear un mapa del mundo 386
Representar magnitudes 387
Personalizar los colores de los marcadores 388
Otras escalas de colores 390
Añadir texto emergente 390
Resumen 392

17. TRABAJAR CON API 393

Usar una API 393
Git y GitHub 393
Solicitar datos usando una llamada a la API 394
Instalar solicitudes 395
Procesar una respuesta de la API 395
Trabajar con el diccionario de respuesta 396
Resumir los principales repositorios 399
Monitorizar los límites de cuota de la API 400

Visualizar repositorios con Plotly 400
Dar estilo al gráfico 402
Añadir mensajes emergentes personalizados 403
Añadir enlaces activos a nuestro gráfico 405
Personalizar los colores de los marcadores 405
Más sobre Plotly y la API de GitHub 406
La API de Hacker News 406
Resumen 410

18. PRIMEROS PASOS CON DJANGO 411

Configurar un proyecto 411
Escribir una especificación 412
Crear un entorno virtual 412
Activar el entorno virtual 413
Instalar Django 413
Crear un proyecto en Django 414
Crear la base de datos 414
Visionar el proyecto 415
Iniciar una aplicación 417
Definir modelos 417
Activar modelos 418
El sitio admin de Django 420
Definir el modelo Entry 422
Migrar el modelo Entry 423
Registrar Entry con el sitio Admin 424
El intérprete de Django 425
Crear páginas: La página de inicio de Learning Log 427
Asignar una URL 427
Escribir una vista 429
Escribir una plantilla 429
Crear páginas adicionales 431
Herencia de plantillas 431
La página topics 433
Página de temas individuales 436
Resumen 440

19. CUENTAS DE USUARIO 441

Permitir que los usuarios introduzcan datos 441
Añadir temas nuevos 442
Añadir nuevas entradas 446
Editar entradas 450
Configurar cuentas de usuario 453
La aplicación accounts 454
La página de inicio de sesión 455
Cerrar sesión 458
La página de registro 459

Permitir que los usuarios controlen sus datos 462
Restringir el acceso con @login_required 462
Conectar datos con determinados usuarios 464
Restringir el acceso a temas a los usuarios adecuados 467
Proteger los temas de un usuario 467
Proteger la página edit_entry 468
Asociar temas nuevos con el usuario actual....469
Resumen 470

20. ESTILO Y DESPLIEGUE DE UNA *APP* ***471***

Dar estilo a Learning Log 471
La aplicación django-bootstrap5....472
Usar Bootstrap para dar estilo a Learning Log 472
Modificar base.html 473
Dar estilo a la página de inicio con un *jumbotron*....479
Dar estilo a la página de inicio de sesión 480
Dar estilo a la página de temas....481
Dar estilo a las entradas en la página de un tema 482
Desplegar Learning Log 484
Crear una cuenta en Platform.sh 485
Instalar la CLI de Platform.sh 485
Instalar platformshconfig 485
Crear un archivo requirements.txt 485
Requisitos de despliegue adicionales....486
Añadir archivos de configuración 487
Modificar settings.py en Platform.sh 490
Usar Git para hacer un seguimiento de los archivos del proyecto 491
Crear un proyecto en Platform.sh 493
Pasar a Platform.sh 494
Ver el proyecto en vivo 495
Refinar el despliegue de Platform.sh 496
Crear páginas de error personalizadas 498
Desarrollo continuo 500
Borrar un proyecto en Platform.sh 500
Resumen 502

PARTE III. APÉNDICES **503**

APÉNDICE A. INSTALACIÓN Y SOLUCIÓN DE PROBLEMAS **505**

APÉNDICE B. EDITORES DE TEXTO E IDE **511**

APÉNDICE C. CONSEGUIR AYUDA **519**

APÉNDICE D. USAR GIT PARA EL CONTROL DE VERSIONES **523**

APÉNDICE E. SOLUCIONAR PROBLEMAS DE IMPLEMENTACIÓN **533**

ÍNDICE ALFABÉTICO **543**

¿Para quién es este libro?

El objetivo de este libro es introducir al lector en Python con la mayor rapidez posible para que pueda empezar a crear programas operativos, incluidos juegos, visualizaciones de datos y aplicaciones web, al tiempo que desarrolla una base de programación que le servirá para el resto de su vida. Este libro está escrito para personas de cualquier edad que nunca han programado con Python o que nunca han programado en general. Este libro es apto para todo aquel que quiera aprender rápidamente lo más básico de la programación y concentrarse en proyectos interesantes, y para los que quieran poner a prueba su comprensión de nuevos conceptos resolviendo problemas significativos. Esta guía también es perfecta para profesores de enseñanza media y superior que desean ofrecer a sus alumnos una introducción a la programación basada en proyectos. Si va a cursar una asignatura universitaria y necesita una introducción a Python más sencilla que su libro de texto, esta obra le facilitará el seguimiento de las clases. Si tiene pensado cambiar de trabajo, este libro le permitirá reorientar su carrera profesional. Ha funcionado para muchos lectores con objetivos muy diversos.

¿Qué puede esperar aprender?

Este libro tiene por objeto convertirle en un buen programador en general y en un buen programador de Python en particular. Al adquirir una buena base en conceptos generales de programación, aprenderá de manera eficiente y adoptará buenos hábitos. Cuando termine con este libro, debería estar listo para pasar a técnicas de Python más avanzadas y tendrá más facilidad para aprender nuevos lenguajes de programación.

En la primera parte del libro, descubrirá los conceptos de programación básicos que necesita conocer para escribir programas con Python. Estos conceptos son los mismos que aprendería al iniciarse en prácticamente cualquier lenguaje de programación. Aprenderá a crear diferentes tipos de datos y a almacenarlos en sus programas. Aprenderá a construir colecciones de datos, como listas y diccionarios, y a manejar estas colecciones de forma eficaz. Aprenderá a usar bucles `while` y sentencias `if` para comprobar determinadas condiciones, con el fin de poder ejecutar secciones de código específico si se cumplen esas condiciones o ejecutar otras secciones cuando no sea así, una técnica de gran ayuda a la hora de automatizar numerosos procesos.

Aprenderá a aceptar entradas de usuarios para que sus programas sean interactivos y a hacer que se ejecuten mientras el usuario quiera. Explorará cómo escribir funciones que conviertan en reutilizables partes de sus programas; así, solo tendrá que escribir bloques de código que realicen determinadas acciones una vez mientras utiliza el código tantas veces como lo necesite. Más adelante, hará extensivo este concepto a comportamientos más complicados con clases, haciendo que programas bastante sencillos puedan responder a diversas situaciones. Aprenderá a escribir programas para gestionar con gracia errores frecuentes. Después de trabajar con estos conceptos básicos, escribirá unos cuantos programas de complejidad creciente que le permitirán poner en práctica lo aprendido. Por último, dará un primer paso hacia la programación intermedia aprendiendo a escribir pruebas para su código, para poder desarrollar más sus programas sin miedo a generar errores. Toda la información de la primera parte le preparará para emprender proyectos más complejos y ambiciosos.

En la segunda parte, aplicaremos lo aprendido en la primera a tres proyectos. Puede trabajar en todos o solo en algunos, en el orden que crea conveniente. En el primer proyecto (capítulos 12-14), creará un juego de disparar al estilo de Space Invaders, llamado Alien Invasion, que tendrá varios niveles de dificultad. Una vez completado este proyecto, debería estar ya encaminado hacia la creación de sus propios juegos en 2D. Incluso si no aspira a convertirse en programador de juegos, este proyecto es una manera muy satisfactoria de trabajar los contenidos aprendidos en la primera parte.

El segundo proyecto (capítulos 15-17) es una introducción a la visualización de datos. Los científicos de datos utilizan diferentes técnicas de visualización para dar sentido a la enorme cantidad de información de la que disponen. Trabajaremos con conjuntos de datos generados con código, conjuntos de datos descargados de recursos en línea y conjuntos de datos descargados automáticamente por nuestros programas. Una vez completado este proyecto, será capaz de escribir programas que filtren grandes conjuntos de datos y creen representaciones visuales de diferentes tipos de información.

En el tercer proyecto (capítulos 18-20), crearemos una pequeña aplicación web llamada Learning Log. Este proyecto permite mantener un registro organizado de la información aprendida sobre un tema particular. Podrá mantener registros separados para temas diferentes y permitir que otros creen una cuenta para iniciar sus propios registros. También aprenderá a desplegar su proyecto para que cualquiera pueda acceder en línea a él desde cualquier parte.

Recursos en línea

Puede descargarse los recursos del libro (en inglés) en la página web de Anaya Multimedia: `http://www.anayamultimedia.es`, en la opción Selecciona Complemento que encontrará en la ficha correspondiente a este libro. Además, dispone de estos mismos recursos y algunos adicionales en la página web del libro original en `https://ehmatthes.github.io/pcc_3e/`. Estos recursos (en inglés) incluyen:

- **Instrucciones de instalación:** Las instrucciones de instalación en línea son idénticas a las del libro, pero incluyen enlaces activos para cada uno de los pasos. Si tiene problemas de configuración, utilice este recurso.
- **Actualizaciones:** Como todos los lenguajes de programación, Python está en constante evolución. Tengo una lista exhaustiva de actualizaciones, así que, si algo no le funciona, compruebe aquí si han cambiado las instrucciones.

- Imprimir el mensaje "Estos tres elementos están en el medio de la lista:". A continuación, use un trozo para imprimir los tres elementos centrales de la lista.
- Imprimir el mensaje "Estos son los tres últimos elementos de la lista:". A continuación, use un trozo para imprimir los tres últimos elementos de la lista.

- **4-11. Mis pizzas, sus pizzas:** Empiece con el programa del ejercicio 4-1. Haga una copia de la lista de pizzas y llámela `friend_pizzas`. A continuación, haga lo siguiente:
  - Añada una pizza nueva a la lista original.
  - Añada una pizza diferente a la lista `friend_pizzas`.
  - Compruebe que tiene dos listas separadas. Imprima el mensaje "Mis pizzas favoritas son:" y luego use un bucle `for` para imprimir la primera lista. Imprima el mensaje "Las pizzas favoritas de mi amigo son:" y después utilice un bucle `for` para imprimir la segunda lista. Asegúrese de que cada pizza se guarda en la lista adecuada.
- **4-12. Más bucles:** Todas las versiones de `foods.py` de esta sección han evitado usar bucles `for` al imprimir para ahorrar espacio. Elija una versión de `foods.py` y escriba dos bucles para imprimir cada lista de comida.

Tuplas

Las listas funcionan bien para almacenar colecciones de elementos que pueden cambiar a lo largo de la vida de un programa. La capacidad para modificar listas es de especial importancia cuando se trabaja con una lista de usuarios de un sitio web o de personajes de un juego. Sin embargo, a veces necesitamos crear listas de elementos que no se puedan alterar. Eso es justo lo que podemos hacer con las tuplas. Python se refiere a los valores que no pueden cambiar como inmutables, y una lista inmutable se denomina "tupla".

Definir una tupla

Una tupla es muy parecida a una lista, solo que emplea paréntesis en lugar de corchetes. Una vez definida la tupla, podemos acceder a los elementos individuales usando sus índices, como haríamos con una lista. Por ejemplo, si tenemos un rectángulo que siempre debería tener el mismo tamaño, podemos asegurarnos de que no cambia incluyendo sus dimensiones en una tupla:

dimensions.py

```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```


Definimos la tupla `dimensions`, usando paréntesis en vez de corchetes. A continuación, imprimimos cada elemento de la tupla de manera individual con la misma sintaxis que hemos estado usando para acceder a los elementos de una lista:

```
200
50
```

Veamos qué pasa si intentamos cambiar uno de los elementos de la tupla `dimensions`:

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

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

```
Traceback (most recent call last):
  File "dimensions.py", line 2, in <module>
    dimensions[0] = 250
TypeError: 'tuple' object does not support item assignment
```

Esto es bueno porque queremos que Python dé 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 las hacen parecer más claras y legibles. Si queremos definir una tupla con un solo elemento, tendremos que incluir una coma:

```
my_t = (3,)
```

No suele tener mucho sentido crear una tupla con un solo elemento, pero puede ocurrir cuando las tuplas se generan automáticamente.

Pasar en bucle por todos los valores de una tupla

Podemos pasar en bucle por todos los valores de una tupla con un bucle `for`, igual que hemos hecho con las listas:

```
dimensions = (200, 50)
for dimension in dimensions:
  print(dimension)
```

Python devuelve todos los elementos de la tupla, igual que haría con una lista:

```
200
50
```

Sobrescribir una tupla

Aunque no se puede modificar una tupla, sí se puede asignar un nuevo valor a una variable que representa una tupla. Por ejemplo, si quisiéramos cambiar las dimensiones de este rectángulo, podríamos redefinir la tupla entera:

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
   print(dimension)

dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
   print(dimension)
```

Las cuatro primeras líneas definen la tupla original e imprimen las dimensiones iniciales. A continuación, asociamos la nueva tupla con la variable `dimensions` e imprimimos las nuevas dimensiones. Python no da error esta vez porque reasignar una variable es válido:

```
Original dimensions:
200
50

Modified dimensions:
400
100
```

En comparación con las listas, las tuplas son estructuras de datos simples. Utilícelas cuando quiera guardar un conjunto de valores que no deberían cambiar durante la vida de un programa.

PRUÉBELO

- **4-13. Bufé:** Un restaurante de tipo bufé ofrece solo cinco comidas básicas. Piense en cinco platos básicos y guárdelos en una tupla.
  - Use un bucle `for` para imprimir cada comida que ofrece el restaurante.
  - Intente modificar alguno de los elementos y asegúrese de que Python rechaza el cambio.
  - El restaurante cambia su menú, sustituyendo dos elementos por comidas diferentes. Añada una línea que rehaga la tupla y use un bucle `for` para imprimir cada uno de los elementos del menú modificado.

Dar estilo a nuestro código

Ahora que empezamos a escribir programas más largos, es una buena idea aprender a dar a nuestro código un estilo consistente. Tómese su tiempo para hacer que su código sea tan fácil de leer como sea posible. Escribir código fácilmente legible nos ayuda a saber lo que hacen nuestros programas y también ayuda a otros a entender nuestro código.

Los programadores de Python han acordado una serie de convenciones de estilo para asegurarse de que el código de todo el mundo se estructura más o menos de la misma forma. Cuando ya sepa escribir código Python limpio, debería ser capaz de entender la estructura general del de cualquier otro programador, siempre que sigan las mismas directrices. Si aspira a convertirse en programador profesional, debería empezar a seguir esas directrices lo antes posible para desarrollar buenos hábitos.

La guía de estilo

Cuando alguien quiere hacer un cambio en el lenguaje Python, escriben una PEP (*Python Enhancement Proposal*, propuesta de mejora de Python). Una de las más antiguas es la PEP 8, que instruye a los programadores de Python en la estilización de código. La PEP 8 es bastante larga, pero buena parte de ella está relacionada con estructuras de código más complejas que las que hemos visto hasta ahora.

La guía de estilo de Python se escribió entendiendo que es más frecuente leer código que escribir código. Escribimos el código una vez y luego empezamos a leerlo para depurarlo. Cuando añadimos funciones a un programa, pasamos más tiempo leyendo nuestro código. Cuando compartimos código con otros programadores, también leerán nuestro código.

Si tuviera que elegir entre escribir código que sea fácil de escribir o fácil de leer, los programadores de Python casi siempre le animarán a escribir código fácil de leer. Las siguientes directrices le ayudarán a escribir código claro desde el principio.

Sangrado

La PEP 8 recomienda usar cuatro espacios por nivel de sangrado. Usar cuatro espacios mejora la legibilidad y deja sitio para varios niveles de sangrado en cada línea.

En un documento elaborado con un procesador de texto, suelen usarse tabulaciones en vez de espacios para crear sangrías, pero el intérprete de Python se confunde cuando se mezclan tabulaciones y espacios. Todos los editores de texto ofrecen alguna configuración que permite usar el tabulador, pero luego convierte esa tabulación en un número de espacios. Debería usar el tabulador, pero asegúrese de que su editor está configurado para insertar espacios en vez de tabulaciones en el documento.

Mezclar tabuladores y espacios en un archivo puede causar problemas difíciles de diagnosticar. Si cree que tiene una mezcla de tabulaciones y espacios, puede convertir todas las tabulaciones de un archivo en espacios en la mayoría de editores.

```
        print(f"This car has {self.odometer_reading} miles on it.")

my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

Esta vez, cuando Python llama al método `__init__()` para crear una nueva instancia, guarda la marca, el modelo y el año como atributos, igual que en el ejemplo anterior. Luego crea un atributo llamado `odometer_reading` y establece su valor inicial en `0` ❶. También tenemos un nuevo método llamado `read_odometer()` en ❷ que facilita la lectura del kilometraje del coche. Nuestro coche empieza con 0 millas:

```
2024 Audi A4
This car has 0 miles on it.
```

No hay muchos coches que se vendan con 0 millas en el cuentakilómetros, así que necesitamos una forma de cambiar el valor de este atributo.

Modificar el valor de un atributo

Podemos cambiar el valor de un atributo de tres maneras: directamente a través de una instancia, estableciendo el valor con un método o incrementando el valor (sumándole una cantidad dada) a través de un método. Veamos cada una de estas técnicas.

Modificar el valor de un atributo directamente

La forma más fácil de cambiar el valor de un atributo consiste en acceder directamente al atributo a través de una instancia. Aquí vamos a poner el cuentakilómetros a 23 directamente:

```
class Car:
   --fragmento omitido--

my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

Usamos la notación de punto para acceder al atributo `odometer_reading` del coche y establecer su valor directamente. Esta línea dice a Python que coja la instancia `my_new_car`, busque el atributo `odometer_reading` asociado con ella y configure el valor de ese atributo como 23:

```
2024 Audi A4
This car has 23 miles on it.
```

A veces, nos interesa acceder directamente a los atributos así, pero otras nos conviene escribir un método que actualice el valor por nosotros.

Modificar el valor de un atributo a través de un método

Puede ser útil tener métodos que actualicen ciertos atributos por nosotros. En lugar de acceder directamente al atributo, pasamos el nuevo valor a un método que gestiona la actualización internamente.

Veamos un ejemplo con un método llamado `update_odometer()`:

```
class Car:
   --fragmento omitido--

   def update_odometer(self, mileage):
      """Configura el kilometraje con el valor dado."""
      self.odometer_reading = mileage

my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())

❶ my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

La única modificación de `Car` es la adición de `update_odometer()`. Este método coge un valor de kilometraje y se lo asigna a `self.odometer_reading`. Usando la instancia `my_new_car`, llamamos a `update_odometer()` con `23` como argumento ❶. Con ello se establece el kilometraje en 23 y `read_odometer()` lo imprime:

```
2024 Audi A4
This car has 23 miles on it.
```

Podemos ampliar el método `update_odometer()` para que trabaje más cada vez que se modifique el cuentakilómetros. Vamos a añadir una lógica para asegurarnos de que nadie intenta manipular la lectura del cuentakilómetros:

```
class Car:
   --fragmento omitido--

   def update_odometer(self, mileage):
      """
      Configura el cuentakilómetros con el valor dado.
      Rechaza el cambio si se intenta hacer retroceder el cuentakilómetros.
      """

❶     if mileage >= self.odometer_reading:
         self.odometer_reading = mileage
      else:
❷        print("You can't roll back an odometer!")
```

Ahora `update_odometer()` comprueba que la nueva lectura tenga sentido antes de modificar el atributo. Si el valor proporcionado para `mileage` es mayor o igual que el existente, `self.odometer_reading`, podemos actualizar el cuentakilómetros al nuevo kilometraje ❶. Si es inferior al existente, recibiremos un aviso de que no podemos hacer retroceder un cuentakilómetros ❷.

cómo escribir un programa que descarga y visualiza datos automáticamente. Aprender a hacer visualizaciones le permitirá explorar el campo de la minería de datos, una de las habilidades más demandadas hoy en día en el ámbito de la programación.

Aplicaciones web

En el proyecto de aplicaciones web (capítulos 18, 19 y 20), usaremos el paquete Django para crear una sencilla aplicación web que permita a los usuarios llevar un diario acerca de una serie de temas de estudio. Los usuarios crearán una cuenta con un nombre de usuario y una contraseña, introducirán un tema y harán entradas sobre aquello que estén aprendiendo. También veremos cómo desplegar la aplicación para que cualquier persona del mundo pueda acceder a ella.

Tras completar este proyecto, podrá empezar a crear sus propias aplicaciones web sencillas y estará listo para explorar recursos más exhaustivos sobre la creación de aplicaciones con Django.

12

UNA NAVE QUE DISPARA BALAS

¡Vamos a crear un juego llamado Alien Invasion! Usaremos Pygame, una colección de módulos divertidos y potentes de Python que gestionan gráficos, animación e incluso sonido, lo que nos facilita mucho la creación de juegos sofisticados. Con Pygame ocupándose de tareas como dibujar imágenes en la pantalla, podemos concentrarnos en la lógica de alto nivel de la dinámica del juego. En este capítulo, instalaremos Pygame y crearemos un cohete espacial que se mueva hacia la izquierda y hacia la derecha disparando balas en respuesta a la entrada del jugador. En los dos capítulos siguientes, crearemos una flota alienígena para destruir y seguiremos refinando el juego poniendo límites al número de naves que se puede usar y añadiendo un marcador.

Mientras crea este juego, también aprenderá a manejar proyectos grandes que se distribuyan en varios archivos. Refactorizaremos mucho código y administraremos el contenido de los archivos para organizar el proyecto y hacer que el código sea efectivo.

Crear juegos es una forma ideal de divertirse mientras se aprende un lenguaje de programación. Jugar a un juego creado por uno mismo es muy satisfactorio, y escribir un juego tan sencillo nos ayudará a entender cómo desarrollan sus juegos los profesionales. Mientras trabaja en este capítulo, escriba y ejecute el código para identificar de qué manera cada bloque va aportando algo a la experiencia global del juego. Experimente con distintos valores y configuraciones para entender mejor cómo ir refinando las interacciones en sus juegos.

Nota: Alien Invasion ocupa varios archivos, así que cree una nueva carpeta `alien_invasion` en su sistema. Asegúrese de guardar ahí todos los archivos de proyecto para que las sentencias `import` funcionen correctamente.

Si se siente cómodo con el control de versiones, podría interesarle usarlo para este proyecto. Si no lo ha hecho nunca, consulte el apéndice D para una visión general del tema.

Planificación del proyecto

Cuando creamos un proyecto grande, es importante trazar un plan antes de empezar a escribir código. Este plan le ayudará a mantenerse centrado y aumenta las probabilidades de completar el proyecto. Vamos a escribir una descripción de la mecánica del juego. Aunque la siguiente descripción no cubre todos los detalles de Alien Invasion, da una idea clara de cómo empezar a montar el juego:

En Alien Invasion, el jugador controla una nave que aparece en el centro de la pantalla, en la parte inferior. El jugador puede mover la nave hacia la izquierda y hacia la derecha con las teclas de dirección y disparar balas con la **Barra espaciadora**. Cuando comienza el juego, una flota de extraterrestres llena el cielo y se mueve hacia abajo por la pantalla. El jugador dispara a los aliens y los destruye. Cuando el jugador consiga acabar con todos los alienígenas, aparece una nueva flota que se mueve más rápido que la anterior. Si un alien toca la nave del jugador o llega al fondo de la pantalla, el jugador pierde una vida. El juego termina cuando el jugador pierde tres vidas.

Para la primera fase de desarrollo, haremos una nave que se pueda desplazar hacia la derecha y hacia la izquierda cuando el jugador pulse las flechas de dirección del teclado y que dispare cuando el jugador pulse la **Barra espaciadora**. Tras configurar este comportamiento, podemos crear los aliens y refinar la mecánica del juego.

Instalar Pygame

Antes de empezar a escribir código, instale Pygame. Lo haremos del mismo modo en que instalamos pytest en el capítulo 11: con pip. Si se ha saltado el capítulo 11 o si necesita refrescar sus conocimientos sobre pip, vuelva a consultar este capítulo. Para instalar Pygame, escriba el siguiente comando:

```
$ python -m pip install --user pygame
```

Si usa un comando distinto de `python` para ejecutar programas o iniciar una sesión de terminal, como `python3`, asegúrese de utilizar dicho comando.

Iniciar el proyecto del juego

Empezaremos a construir el juego creando una ventana de Pygame vacía. Más adelante dibujaremos los elementos del juego, como la nave y los extraterrestres, en esta ventana. También haremos que nuestro juego responda a entrada de usuario, configuraremos el color de fondo y cargaremos una imagen de una nave.

Crear una ventana de Pygame y responder a entrada de usuario

Haremos una ventana vacía de Pygame creando una clase que represente el juego. En su editor de texto, cree un nuevo archivo y guárdelo como `alien_invasion.py`; luego escriba lo siguiente:

alien_invasion.py

```
import sys

import pygame

class AlienInvasion:
    """Clase general para gestionar los recursos y el comportamiento del juego."""

    def __init__(self):
        """Inicializa el juego y crea recursos."""
❶       pygame.init()

❷       self.screen = pygame.display.set_mode((1200, 800))
        pygame.display.set_caption("Alien Invasion")

    def run_game(self):
        """Inicia el bucle principal para el juego."""
❸       while True:
            # Busca eventos de teclado y ratón.
❹           for event in pygame.event.get():
❺               if event.type == pygame.QUIT:
                    sys.exit()

            # Hace visible la última pantalla dibujada.
❻           pygame.display.flip()

if __name__ == '__main__':
    # Hace una instancia del juego y lo ejecuta.
    ai = AlienInvasion()
    ai.run_game()
```

Primero, importamos los módulos `sys` y `pygame`. El módulo `pygame` contiene la funcionalidad que necesitamos para crear un juego. Usaremos las herramientas del módulo `sys` para salir del juego cuando el jugador quiera.

Alien Invasion empieza como una clase llamada `AlienInvasion`. En el método `__init__()`, la función `pygame.init()` inicializa la configuración de fondo que necesita Pygame para funcionar correctamente ❶. A continuación llamamos a `pygame.display.set_mode()` para crear una ventana ❷ 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 tendrá 1.200 píxeles de ancho por 800 de alto. (Puede ajustar estos valores dependiendo del tamaño de su monitor). Asignamos esta ventana al atributo `self.screen` para que esté disponible en todos los métodos de la clase.

scatter_squares.py

```
import matplotlib.pyplot as plt

❶ x_values = range(1, 1001)
y_values = [x**2 for x in x_values]

plt.style.use('seaborn')
fig, ax = plt.subplots()
❷ ax.scatter(x_values, y_values, s=10)

# Establece el título del gráfico y las etiquetas de los ejes.
--fragmento omitido--

# Establece el rango para cada eje.
❸ ax.axis([0, 1100, 0, 1_100_000])

plt.show()
```

Empezamos con un rango de valores *x* que contiene los números del 1 al 1.000 ❶. A continuación, una comprensión de lista genera los valores *y* pasando en bucle por los valores *x* (`for x in x_values`), elevando cada número al cuadrado (`x**2`) y asignando los resultados en `y_values`. A continuación, pasamos las listas de entrada y salida a `scatter()` ❷. Como es un conjunto de datos grande, usamos un tamaño de punto más pequeño. Antes de mostrar el trazado, utilizamos el método `axis()` para especificar el rango de cada eje ❸. El método `axis()` requiere cuatro valores: los valores máximos y mínimos para los ejes *x* e *y*. Aquí, el eje *x* va de 0 a 1.100 y el y de 0 a 1.000.000. La figura 15.7 muestra el resultado.

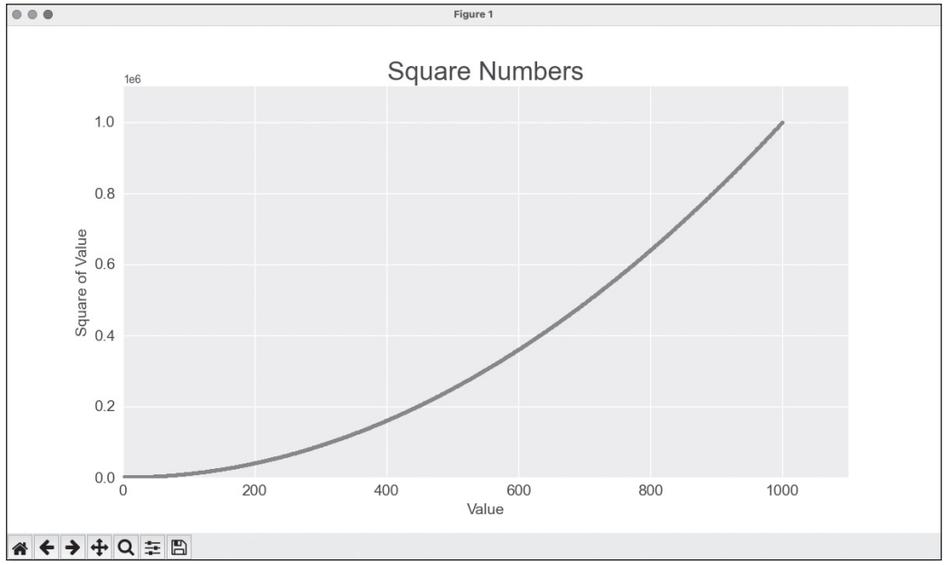


Figura 15.7. Python puede trazar 1.000 puntos con la misma facilidad con la que traza 5.

Personalizar las etiquetas de los puntos de los ejes

Cuando los números de un eje son lo suficientemente grandes, Matplotlib recurre por defecto a la notación científica para las etiquetas de los puntos de los ejes. Por lo general, esto es bueno, ya que los números grandes ocupan mucho espacio innecesario en una visualización.

Prácticamente todos los elementos de un gráfico son personalizables, por lo que podemos indicarle a Matplotlib que siga utilizando notación sencilla si así lo deseamos.

```
--fragmento omitido--
# Establece el rango de cada eje.
ax.axis([0, 1100, 0, 1_100_000])
ax.ticklabel_format(style='plain')

plt.show()
```

El método `ticklabel_format()` permite reemplazar el estilo predeterminado de las etiquetas de los puntos de los ejes en cualquier gráfico.

Definir colores personalizados

Para cambiar el color de los puntos, pase el argumento `color` a `scatter()` con el nombre del color que quiere usar entre comillas simples, como aquí:

```
ax.scatter(x_values, y_values, color='red', s=10)
```

También puede definir colores personalizados con el modelo de color RGB. Para definir un color, pase al argumento `color` a una tupla con tres valores flotantes (uno para rojo, otro para verde y otro para azul, en ese orden), usando valores entre 0 y 1. Por ejemplo, la siguiente línea crea un trazado con puntos de color verde claro:

```
ax.scatter(x_values, y_values, c=(0, 0.8, 0), s=10)
```

Los valores más próximos a 0 producen colores más oscuros y los más cercanos a 1, colores más claros.

Utilizar un mapa de color

Un mapa de color es una secuencia de colores en un gradiente que va de un color inicial a un color final. En las visualizaciones, se emplean mapas de color para enfatizar un patrón en los datos. Por ejemplo, podríamos hacer que los valores bajos tengan un color claro y los altos, uno más oscuro. El uso de un mapa de color nos garantiza que todos los puntos en una visualización varíen suavemente y con precisión de acuerdo con una escala de color correctamente diseñada.

El módulo `pyplot` incluye una serie de mapas de color integrados. Para usar uno de estos mapas, es preciso especificar la manera en que `pyplot` debería asignar un color a cada punto del conjunto de datos. Veamos cómo asignar un color a cada punto, basado en su valor *y*:

Generar múltiples caminatas aleatorias

Cada caminata aleatoria es diferente. Es divertido explorar los distintos patrones que pueden generarse. Una forma de usar el código anterior para hacer varios paseos sin tener que ejecutar el programa repetidamente consiste en meterlo en un bucle `while`, así:

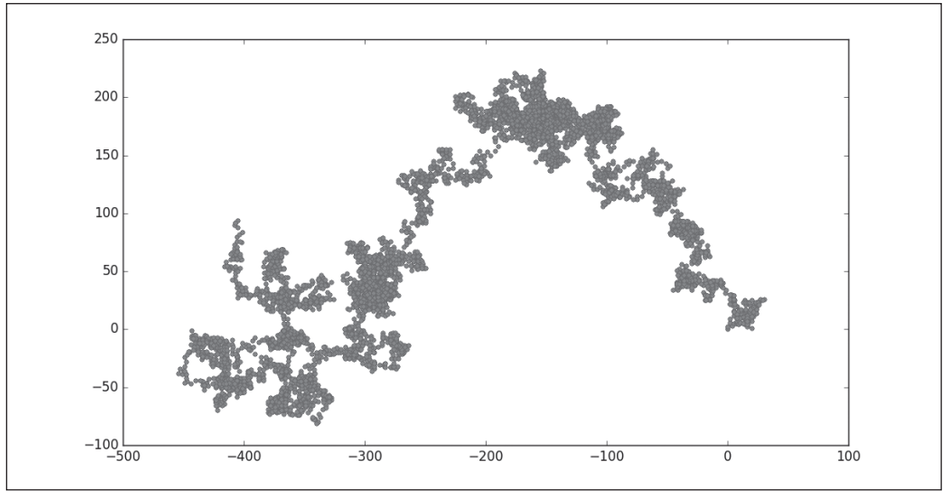


Figura 15.9. Una caminata aleatoria con 5.000 puntos.

rw_visual.py

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# Sigue generando caminatas nuevas mientras el programa esté activo.
while True:
   # Crea una caminata aleatoria.
   --fragmento omitido--
   plt.show()

   keep_running = input("Make another walk? (y/n): ")
   if keep_running == 'n':
      break
```

Este código genera una caminata aleatoria, la muestra en el visor de Matplotlib y hace una pausa con el visor abierto. Cuando cierre el visor, se le preguntará si quiere generar otra. Si genera varias caminatas, verá que algunas se quedan cerca del punto de inicio, mientras otras se desvían en una dirección, otras contienen secciones finas que conectan grupos de puntos de mayor tamaño, y muchas categorías más. Cuando quiera finalizar el programa, pulse **N**.

Dar estilo a la caminata

En este apartado, personalizaremos nuestros trazados para enfatizar las características importantes de cada paseo o caminata y restar importancia a los elementos distractores. Para ello, identificamos las características que queremos enfatizar, como dónde empezó la caminata, dónde terminó y qué ruta siguió. A continuación, identificamos las características a las que queremos restar énfasis, como las marcas y las etiquetas. El resultado debería ser una representación visual sencilla que comunique claramente la ruta seguida en cada caminata aleatoria.

Colorear los puntos

Usaremos un mapa de color para mostrar el orden de los puntos en la caminata y luego eliminaremos el contorno negro de cada punto para que destaque el color de relleno de los puntos. Para colorear los puntos según su posición en el camino, pasamos al argumento `c` una lista con la posición de cada punto. Como los puntos se trazan en orden, esta lista solo contiene los números del 0 al 4.999, como se muestra aquí:

rw_visual.py

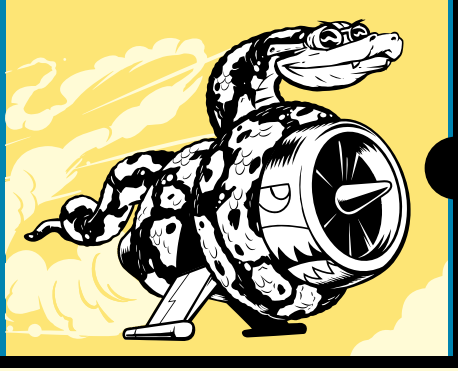
```
--fragmento omitido--
while True:
   # Crea una caminata aleatoria.
   rw = RandomWalk()
   rw.fill_walk()

   # Traza los puntos de la caminata.
   plt.style.use('classic')
   fig, ax = plt.subplots()
❶  point_numbers = range(rw.num_points)
   ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
      edgecolors='none', s=15)
   ax.set_aspect('equal')
   plt.show()
   --fragmento omitido--
```

Usamos `range()` para generar una lista de números igual a la cantidad de puntos de la caminata ❶. Asignamos esta lista a `point_numbers`, que usaremos para establecer el color de cada punto de la caminata. Pasamos `point_numbers` al argumento `c`, usamos el mapa de color `Blues` y pasamos `edgecolors='none'` para deshacernos del contorno negro de cada punto. El resultado es un trazado que varía de azul claro a azul oscuro en un gradiente, mostrando exactamente cómo la caminata se mueve desde el punto de partida hasta su destino final. Así se ve en la figura 15.10.

Trazar los puntos de inicio y de fin

Además de colorear puntos para mostrar su posición, sería útil ver dónde empieza y acaba cada caminata. Para ello, podemos trazar el primer punto y el último individualmente después de trazar la serie principal. Haremos que estos dos puntos sean más grandes y los pintaremos de un color diferente para que destaquen, como se muestra aquí:

SUPERVENTAS MUNDIAL



¡APRENDA PYTHON RÁPIDO!

Este superventas mundial es una guía al lenguaje de programación Python. Gracias a esta trepidante y completa introducción a Python, no tardará en empezar a escribir programas, resolver problemas y desarrollar aplicaciones que funcionen.

Comenzará aprendiendo conceptos básicos de programación, como "variables", "listas", "clases" y "bucles" y practicará la creación de código limpio con ejercicios sobre cada tema. Además, aprenderá a escribir programas interactivos y a probar su código con seguridad antes de añadirlo a un proyecto. En la segunda parte del libro, pondrá sus nuevos conocimientos en práctica con tres proyectos: un juego arcade inspirado en *Space Invaders*, un conjunto de visualización de datos con las útiles librerías de Python y una sencilla aplicación web que podrá desplegar en línea.

A medida que trabaje con el libro, aprenderá a:

- Usar librerías y herramientas de Python potentes, como Pygame, Matplotlib, Plotly y Django.
- Crear juegos en 2D de complejidad creciente que respondan a pulsaciones de teclado y clics de ratón.
- Generar visualizaciones interactivas a partir de diferentes conjuntos de datos.
- Crear y personalizar aplicaciones web y desplegarlas en línea.
- Solucionar fallos de código y resolver problemas frecuentes de programación.

Esta tercera edición actualizada se ha revisado en profundidad con el fin de reflejar las últimas novedades en Python. Entre las novedades se incluye la incorporación de VS Code para la edición de texto, el módulo pathlib para la gestión de archivos y pytest para probar el código, así como las últimas novedades en Matplotlib, Plotly y Django.

Si tiene ganas de profundizar en la programación, este libro le ayudará a escribir programas de verdad rápidamente. ¿Por qué esperar más? ¡Empiece ya a programar!

SOBRE EL AUTOR

Eric Matthes fue profesor de matemáticas y ciencias en Secundaria, así como de clases de introducción a Python, durante 25 años. En la actualidad Matthes trabaja en diversos proyectos de código abierto, y como escritor y programador a tiempo completo.

CUBRE PYTHON 3.X

ANAYA MULTIMEDIA
www.anayamultimedia.es
no starch press
ISBN 978-84-415-4924-1
2315255