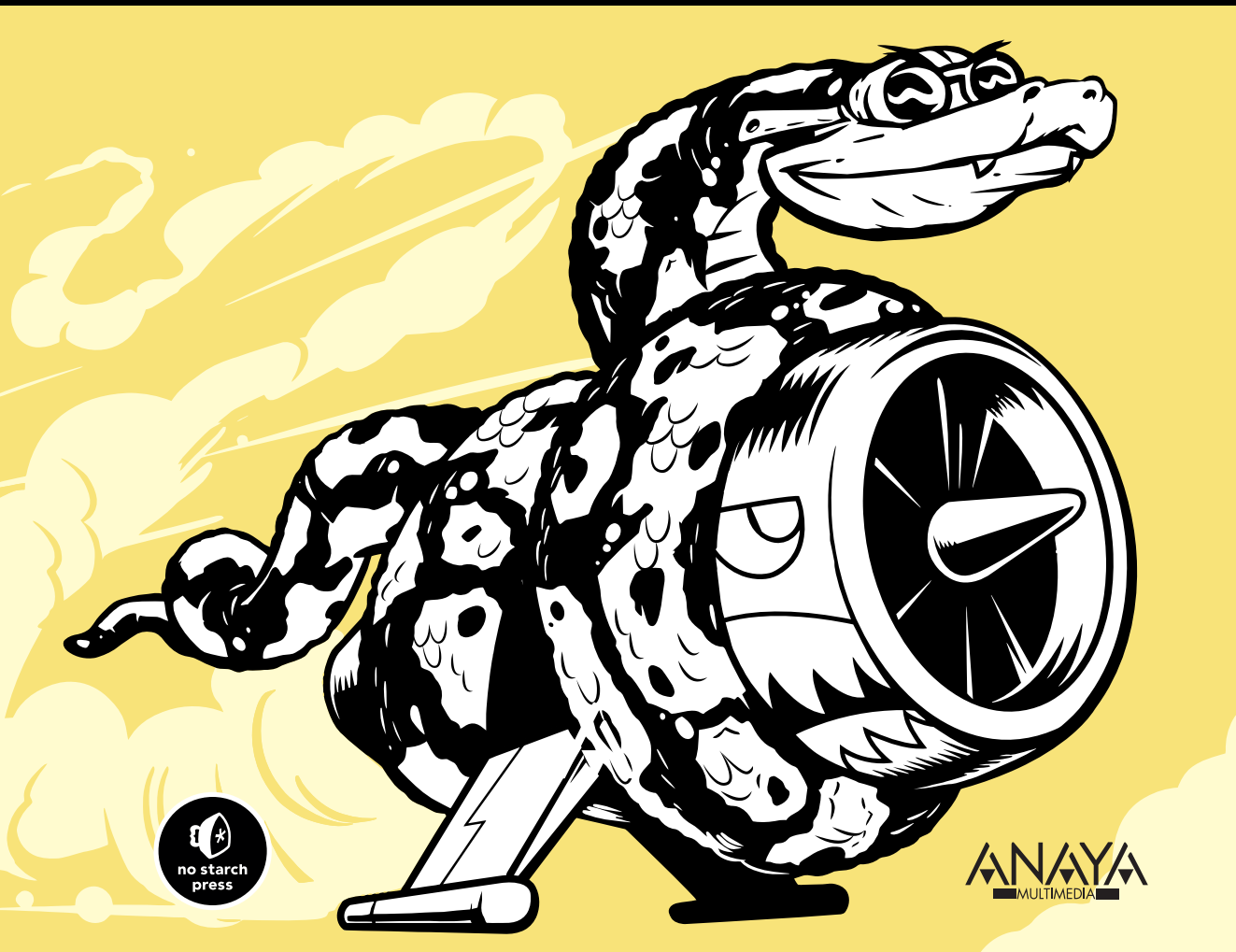
2ª EDICIÓN

CURSO INTENSIVO DE PYTHON

INTRODUCCIÓN PRÁCTICA A LA PROGRAMACIÓN BASADA EN PROYECTOS

ERIC MATTHES



no starch press

ANAYA MULTIMEDIA

ÍNDICE DE CONTENIDOS

Figura 1.1. Asegúrese de seleccionar la casilla de verificación Add Python to PATH.

Ejecutar Python en una sesión de terminal

Abra una ventana de comandos y escriba **python** en minúscula. Debería aparecer un indicador de Python (>>>), lo que significa que Windows ha encontrado la versión de Python que acabamos de instalar.

```
C:\> python
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

NOTA: Si no ve esta salida o algo similar, consulte las instrucciones de instalación detalladas del apéndice A.

Escriba la siguiente línea en su sesión de Python y compruebe que la salida es `¡Hola, intérprete de Python!`.

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

Siempre que desee ejecutar un *snippet* de código Python, abra una ventana de comandos e inicie una sesión de terminal. Para terminar la sesión de Python, pulse **Control-D** y, a continuación, **Intro**, o escriba el comando `exit()`.

Instalación de Sublime Text

Puede descargar un instalador de Sublime Text en `https://sublimetext.com/`. Haga clic en el enlace de descarga y busque un instalador para Windows. Una vez descargado el instalador, ejecútelo y acepte todas las opciones predeterminadas.

Python en macOS

Python viene instalado en la mayoría de los sistemas macOS, pero seguramente se trate de una versión desfasada con la que no le conviene aprender. En esta sección, instalaremos la versión más reciente de Python y, después, instalaremos Sublime Text y nos aseguraremos de configurarlo bien.

Comprobar si está instalado Python 3

Abra una ventana de terminal yendo a Aplicaciones>Utilidades>Terminal. También puede pulsar **Comando-Barra espaciadora**, escriba **terminal** y pulse **Intro**. Para ver qué versión de Python está instalada, escriba **python**, con "p" minúscula; así también se inicia el intérprete de Python dentro del terminal para poder escribir comandos.

La salida debería indicarle qué versión de Python está instalada en su sistema y podrá empezar a introducir comandos junto al indicador >>>, así:

```
$ python
Python 2.7.15 (default, Aug 17 2018, 22:39:05)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits", or "license" for more information.
>>>
```

Esta salida indica que Python 2.7.15 es la versión instalada por defecto en este ordenador. Una vez vista la salida, pulse **Control-D** o escriba `exit()` para salir del intérprete de Python y volver al del terminal. Para comprobar si tiene Python 3 instalado, escriba el comando `python3`. Seguramente recibirá un mensaje de error, lo que significa que no tiene ninguna versión de Python 3 instalada. Si la salida muestra que tiene Python 3.6 o una versión posterior instalada, puede saltar al apartado "Ejecutar Python en una sesión de terminal". Si Python 3 no está instalado por defecto, tendrá que instalarlo manualmente. Observe que siempre que vea el comando `python` en este libro, tendrá que usar `python3` en su lugar para asegurarse de usar Python 3 y no Python 2; son tan diferentes que tendrá problemas si intenta ejecutar el código de este libro usando Python 2. Si detecta cualquier versión anterior a Python 3.6, siga las instrucciones del siguiente apartado para instalar la versión más reciente.

Instalación de la última versión de Python

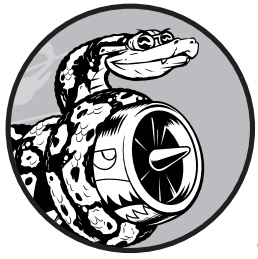
Encontrará un instalador de Python para su sistema en `https://python.org/`. Pase el ratón por encima de Download para que aparezca un botón para descargar la versión más reciente de Python. Haga clic en ese botón para iniciar la descarga automática del instalador correcto para su sistema. Una vez descargado el archivo, ejecute el instalador. Cuando haya terminado, escriba lo siguiente en un terminal:

```
$ python3 --version
Python 3.7.2
```

Debería ver una salida similar a esta, en cuyo caso, ya puede probar Python. Recuerde: cuando vea `python` aquí, asegúrese de usar `python3`.

3

INTRODUCCIÓN A LAS LISTAS



En este capítulo y el siguiente, descubrirá qué son las listas y cómo empezar a trabajar con los elementos que contienen. Las listas permiten almacenar conjuntos de información en un lugar, da igual si se trata de unos pocos elementos o de millones. Las listas son una de las características más potentes de Python fácilmente accesibles para los nuevos programadores y aúnan muchos conceptos de programación importantes.

¿Qué es una lista?

Una lista es una colección de elementos en un orden particular. Podemos hacer una lista que incluya las letras del abecedario, los números del 0 al 9 o los nombres de todos nuestros familiares. Podemos poner todo lo que queramos en una lista y esos elementos no tienen por qué estar relacionados de una forma concreta. Como una lista suele contener más de un elemento, conviene ponerle un nombre en plural, como `letras`, `números` o `nombres`.

En Python, los corchetes ([]) indican una lista y, dentro, los elementos individuales se separan por comas. Aquí tiene un sencillo ejemplo de lista con unos pocos tipos de bicicleta:

bicycles.py

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

En resumen, si solo quiere que se ejecute un bloque de código, use una cadena `if-elif-else`. Si necesita ejecutar más de un bloque de código, utilice una serie de sentencias `if` independientes.

PRUÉBELO

- **5-3. Colores de aliens #1:** Imagine que se acaba de disparar a un alien en un juego. Cree una variable llamada `color_alien` y asígnele como valor `'verde'`, `'amarillo'` o `'rojo'`.
  - Escriba una sentencia `if` para comprobar si el color del alien es verde. Si lo es, imprima un mensaje informando al jugador de que ha ganado 5 puntos.
  - Escriba una versión de este programa que pase la prueba `if` y otra que no. (La versión que no supera la prueba no tendrá salida).
- **5-4. Colores de aliens #2:** Elija un color para un alien igual que en el ejercicio 5-3 y escriba una cadena `if-else`.
  - Si el color del alien es verde, imprima una mensaje informando al jugador de que ha ganado 5 puntos por disparar al alien.
  - Si el color del extraterrestre no es verde, imprima una frase informando al jugador de que 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. Colores de aliens #3:** Convierta la cadena `if-else` del ejercicio 5-4 en una cadena `if-elif-else`.
  - Si el alien es verde, imprima un mensaje diciendo al jugador que ha ganado 5 puntos.
  - Si el alien es amarillo, imprima un mensaje diciendo al jugador que ha ganado 10 puntos.
  - Si el alien es rojo, imprima un mensaje diciendo al jugador que ha ganado 15 puntos.
  - Escriba tres versiones de este programa, asegurándose de que se imprime cada mensaje para el color de alien adecuado.
- **5-6. Etapas vitales:** Escriba una cadena `if-elif-else` para determinar la etapa vital de una persona. Atribuya un valor a la variable `edad` y:
  - Si la persona tiene menos de 2 años, imprima un mensaje diciendo que es un bebé.
  - Si la persona tiene entre 2 y 4 años, imprima un mensaje diciendo que es un infante.
  - Si la persona tiene como mínimo 4 años, pero menos de 13, imprima un mensaje diciendo que es un niño.
  - Si la persona tiene como mínimo 13 años, pero menos de 20, imprima un mensaje diciendo que es un adolescente.
  - Si la persona tiene al menos 20 años, pero no llega a 65, imprima un mensaje diciendo que es un adulto.
  - Si la persona tiene 65 años o más, imprima un mensaje diciendo que es un anciano.
- **5-7. Fruta favorita:** Haga una lista de sus frutas favoritas y escriba una serie de sentencias `if` independientes que comprueben ciertas frutas en su lista.
  - Haga una lista de sus frutas favoritas y llámela `frutas_favoritas`.
  - Escriba cinco sentencias `if`. Cada una debería comprobar si una fruta concreta está en su lista. Si lo está, el bloque `if` debería imprimir un mensaje como "¡Pues sí que te gustan los plátanos!".

Utilizar sentencias if con listas

Podemos hacer un trabajo interesante combinando listas y sentencias `if`. Por ejemplo, podemos detectar valores especiales que requieren un tratamiento distinto al resto de valores de la lista. También podemos gestionar con eficiencia condiciones cambiantes, como la disponibilidad de algunos elementos en un restaurante durante un turno, o empezar a probar que nuestro código funciona como queremos en todas las situaciones posibles.

Detectar elementos especiales

Este capítulo empezó con un sencillo ejemplo que mostraba cómo manejar un valor especial `'bmw'`, que tenía que imprimirse en un formato diferente al del resto de los valores de la lista. Ahora que tiene unos conocimientos básicos de las pruebas condicionales y las sentencias `if`, vamos a concentrarnos en cómo puede detectar valores especiales en una lista para manejarlos adecuadamente.

Vamos a seguir con el ejemplo de la pizzería. La pizzería muestra un mensaje cada vez que se añade un ingrediente a una pizza mientras se está preparando. El código para esta acción puede escribirse de una forma muy eficiente haciendo una lista de los ingredientes que ha pedido el cliente y usando un bucle para ir anunciándolos según se añaden a la pizza:

toppings.py

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
   print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")
```

Empezamos con una lista en la que `'cat'` aparece varias veces. Tras imprimirla, Python entra en el bucle `while` porque encuentra el valor `'cat'` en la lista al menos una vez. Ya dentro del bucle, Python elimina el primer caso de `'cat'`, regresa a la línea `while` y vuelve a entrar en el bucle al descubrir que `'cat'` sigue en la lista. Elimina cada aparición de `'cat'` hasta que ese valor ya no está en la lista. Entonces, Python sale del bucle y vuelve a imprimir la lista:

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

Rellenar un diccionario con entrada del usuario

Podemos pedir a los usuarios toda la información que necesitemos en cada paso por un bucle `while`. Vamos a hacer un programa de sondeo en el que cada paso por el bucle pida el nombre del participante y una respuesta. Guardaremos los datos recogidos en un diccionario para poder conectar cada respuesta con el correspondiente usuario:

mountain_poll.py

```
responses = {}

# Configura una bandera para indicar que la encuesta está activa.
polling_active = True

while polling_active:
    # Pide el nombre y la respuesta de la persona.
❶   name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb someday? ")

    # Guarda la respuesta en el diccionario.
❷   responses[name] = response

    # Averigua si alguien más va a hacer la encuesta.
❸   repeat = input("Would you like to let another person respond? (yes/ no) ")
    if repeat == 'no':
        polling_active = False

# La encuesta está completa. Muestra los resultados.
print("\n--- Poll Results ---")
❹ for name, response in responses.items():
    print(f"{name} would like to climb {response}.")
```

El programa define primero un diccionario vacío (`responses`) y configura una bandera (`polling_active`) para indicar que la encuesta está activa. Mientras `polling_active` sea `True`, Python ejecutará el código del bucle `while`.

Dentro del bucle, se pide al usuario que escriba su nombre y el de una montaña que le gustaría escalar ❶. Esa información se guarda en el diccionario de respuestas, ❷, y se pregunta al usuario si quiere que la encuesta siga ejecutándose ❸. Si escribe `yes`, el programa vuelve a entrar en el bucle `while`. Si escribe `no`, la bandera `polling_active` se pone en `False`, el bucle `while` deja de ejecutarse y el último bloque de código de ❹ muestra el resultado de la encuesta.

Si ejecuta este programa introduciendo respuestas de muestra, debería ver una salida como esta:

```
What is your name? Eric
Which mountain would you like to climb someday? Denali
Would you like to let another person respond? (yes/ no) yes

What is your name? Lynn
Which mountain would you like to climb someday? Devil's Thumb
Would you like to let another person respond? (yes/ no) no

--- Poll Results ---
Lynn would like to climb Devil's Thumb.
Eric would like to climb Denali.
```

PRUÉBELO

- **7-8. Bocatería:** Haga una lista llamada `pedidos_bocadillos` y rellénela con los nombres de varios bocadillos. Luego haga una lista vacía llamada `bocadillos_terminados`. Pase en bucle por la lista de pedidos de bocadillos e imprima un mensaje para cada pedido, como "Su bocadillo de atún está listo". A medida que se hagan los bocadillos, páselos a la lista de terminados. Cuando todos los bocadillos estén hechos, imprima un mensaje que los enumere todos.
- **7-9. Ya no hay pastrami:** Usando la lista `pedidos_bocadillos` del ejercicio 7-8, asegúrese de que el bocadillo de `'pastrami'` aparezca en la lista al menos tres veces. Añada código al principio del programa para imprimir un mensaje diciendo que no queda pastrami y use un bucle `while` para eliminar todas las apariciones de `'pastrami'` en `pedidos_bocadillos`. Asegúrese de que no pasa ningún bocadillo de pastrami a la lista `bocadillos_terminados`.
- **7-10. Vacaciones de ensueño:** Escriba un programa que pregunte a los usuarios por las vacaciones de sus sueños. Escriba unas instrucciones como "Si pudieras visitar cualquier lugar del mundo, ¿dónde irías?". Incluya un bloque de código que imprima el resultado de la encuesta.

Resumen

En este capítulo ha aprendido a usar `input()` para permitir que los usuarios introduzcan información en sus programas. Ahora sabe cómo trabajar con entrada textual y numérica y cómo usar bucles `while` para asegurarse de que el programa se ejecuta mientras los usuarios quieran que lo haga. Hemos visto varias formas de controlar el

9

CLASES

La programación orientada a objetos es uno de los enfoques más efectivos para escribir software. En esta aproximación, escribimos clases que representan cosas y situaciones del mundo real y creamos objetos basados en esas clases. Cuando escribimos una clase, definimos el comportamiento general que puede tener una categoría completa de objetos. Cuando creamos objetos individuales de la clase, cada uno se dota automáticamente del comportamiento general; después, podemos dar a cada objeto los rasgos únicos que queramos. Le sorprenderá lo bien que se puede modelar situaciones del mundo real con la programación orientada a objetos.

La creación de un objeto a partir de una clase recibe el nombre de "instanciación" y trabajamos con "instancias" de una clase. En este capítulo, escribiremos clases y crearemos instancias de esas clases. Especificaremos el tipo de información que puede albergar cada instancia y definiremos las acciones que se puede hacer con ellas. También escribiremos clases que amplíen la funcionalidad de clases existentes para que las que sean similares puedan compartir código de una manera eficiente. Guardaremos las clases en módulos e importaremos clases escritas por otros programadores a nuestros propios archivos de programa.

Entender la programación orientada a objetos le ayudará a ver el mundo como lo hacen los programadores. Le ayudará a conocer de verdad su código, no solo lo que pasa en cada línea, sino los conceptos generales subyacentes a todo. Conocer la lógica que hay detrás de las clases le entrenará para pensar lógicamente para escribir programas que resuelvan con eficacia casi cualquier problema que encuentre.

comportamiento del juego a medida que el proyecto crezca: para modificar el juego solo tendremos que cambiar algunos valores en `settings.py`, que es lo que vamos a crear ahora, en vez de buscar distintas configuraciones por todo el proyecto.

Cree un nuevo archivo llamado `settings.py` en su carpeta `alien_invasion` y añada esta clase `Settings` inicial:

settings.py

```
class Settings:
   """Una clase para guardar toda la configuración de Alien Invasion."""

   def __init__(self):
      """Inicializa la configuración del juego."""
      # Configuración de la pantalla
      self.screen_width = 1200
      self.screen_height = 800
      self.bg_color = (230, 230, 230)
```

Para crear una instancia de `Settings` en el proyecto y usarla para acceder a la configuración, tendremos que modificar `alien_invasion.py` así:

alien_invasion.py

```
--fragmento omitido--
import pygame

from settings import Settings

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

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

❷     self.screen = pygame.display.set_mode(
         (self.settings.screen_width, self.settings.screen_height))
      pygame.display.set_caption("Alien Invasion")

   def run_game(self):
      --fragmento omitido--
      # Redibuja la pantalla en cada paso por el bucle.
❸     self.screen.fill(self.settings.bg_color)

      # Hace visible la última pantalla dibujada.
      pygame.display.flip()
--fragmento omitido--
```

Importamos `Settings` al archivo de programa principal. Luego creamos una instancia de `Settings` y se la asignamos a `self.settings` ❶, después de llamar a `pygame.init()`. Cuando creamos una pantalla ❷, usamos los atributos `screen_width` y `screen_height` de `self.settings` y luego `self.settings` para acceder al color de fondo cuando rellenamos la pantalla en ❸.

Cuando ejecute `alien_invasion.py` ahora no verá ningún cambio porque lo único que hemos hecho es mover la configuración que ya teníamos a otra parte. Ya estamos listos para empezar a añadir elementos a la pantalla.

Añadir la imagen de la nave

Vamos a añadir la nave al juego. Para dibujar la nave del jugador en la pantalla, cargaremos una imagen y usaremos el método `blit()` de Pygame para dibujar la imagen. Cuando escoja material gráfico para sus juegos, asegúrese de prestar atención a las licencias. La forma más segura y económica de empezar es usar gráficos con licencia gratuita para usar y modificar, como los de `https://pixabay.com/`.

Puede emplear prácticamente cualquier tipo de archivo de imagen en su juego, pero lo más fácil es usar un mapa de bits (`.bmp`) porque Pygame carga estos archivos por defecto. Aunque se puede configurar Pygame para que utilice otro tipo de archivos, algunos dependen de determinadas bibliotecas de imágenes que deberían estar instaladas en el ordenador. La mayoría de las imágenes que encontrará estarán en formato `.jpg` o `.png`, pero puede convertirlas en mapas de bits con herramientas como Photoshop, GIMP y Paint.

Preste especial atención al color de fondo de la imagen seleccionada. Busque un archivo con un fondo transparente o sólido que pueda reemplazar con cualquier color de fondo en un editor de imágenes. Sus juegos quedarán mejor si el color de fondo de la imagen coincide con el del juego. Otra opción es hacer que el color de fondo del juego coincida con el de la imagen.

Para *Alien Invasion*, puede usar el archivo `ship.bmp` (figura 12.1), disponible en los recursos del libro . El color de fondo del archivo coincide con la configuración que estamos usando en este proyecto. Haga una carpeta llamada `images` dentro de la carpeta del proyecto, `alien_invasion`. Guarde el archivo `ship.bmp` en `images`.

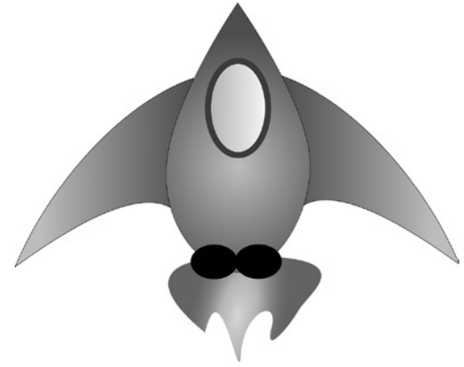


Figura 12.1. La nave de *Alien Invasion*.

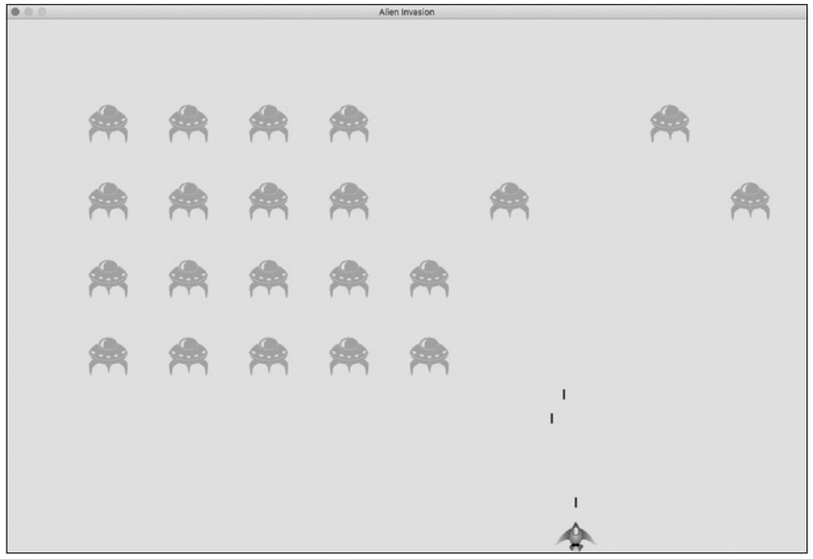


Figura 13.5. ¡Podemos disparar a los aliens!

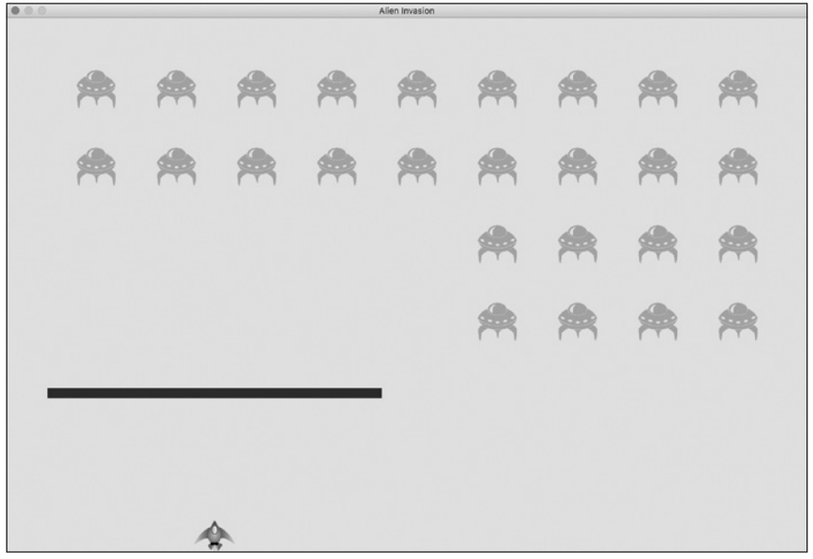


Figura 13.6. Las balas ultrapotentes hacen que sea más fácil probar algunos aspectos del juego.

Cambios como este le ayudarán a probar el juego de una forma más eficiente y puede que le den alguna idea para dar a los jugadores poderes extra. Solo recuerde devolver la configuración a su estado normal cuando haya terminado de probar una característica.

Repoblar la flota

Una característica clave de *Alien Invasion* es que los aliens son implacables: cada vez que se destruya una flota, debería aparecer una nueva.

Para hacer que aparezca una nueva flota de aliens después de que se haya destruido otra, primero comprobaremos si el grupo `aliens` está vacío. Si lo está, haremos una llamada a `_create_fleet()`. Haremos esta comprobación al final de `_update_bullets()`, ya que es ahí donde se destruyen los extraterrestres individuales.

alien_invasion.py

```
def _update_bullets(self):
   --fragmento omitido--
❶  if not self.aliens:
      # Destruye las balas existentes y crea una flota nueva.
❷     self.bullets.empty()
      self._create_fleet()
```

En ❶, comprobamos si el grupo `aliens` está vacío. Un grupo vacío se evalúa como `False`, así que esta es una forma sencilla de comprobar si el grupo está vacío. Si lo está, nos deshacemos de cualquier bala existente con el método `empty()`, que elimina todos los *sprites* que quedan en un grupo ❷. También llamamos a `_create_fleet()`, que vuelve a llenar la pantalla de marcianitos.

Ahora aparecerá una flota nueva en cuanto se destruya la actual.

Acelerar las balas

Si ha intentado disparar a los aliens en el estado actual del juego, es probable que le haya parecido que las balas no van a la velocidad más adecuada para la mecánica del mismo. Puede que sean un poco lentas en su sistema o demasiado rápidas. En este punto, puede modificar la configuración para que el juego sea más interesante y divertido en su sistema. Modificamos la velocidad de las balas ajustando el valor de `bullet_speed` en `settings.py`. En mi sistema, lo ajustaré a 1,5 para que las balas vayan un poco más deprisa:

settings.py

```
# Configuraciones de las balas
self.bullet_speed = 1.5
self.bullet_width = 3
--fragmento omitido--
```

El mejor valor para esta configuración depende de la velocidad del sistema, así que busque el que le funcione. También puede ajustar otras configuraciones.

Cada eje puede configurarse de distintas maneras y cada opción de configuración se guarda como una entrada en un diccionario. En ❸, solo estamos estableciendo el título de cada eje. La clase `Layout()` devuelve un objeto que especifica la disposición y configuración del gráfico como un todo ❹. Aquí establecemos el título del gráfico y pasamos también los diccionarios de configuración de los ejes x e y.

Para generar el trazado, llamamos a la función `offline.plot()` ❺. Esta función necesita un diccionario que contiene los datos y objetos de diseño, y también acepta un nombre para el archivo donde se guardará el gráfico. Guardaremos la salida en un archivo llamado `d6.html`.

Al ejecutar el programa `die_visual.py`, es probable que se abra un navegador mostrando el archivo `d6.html`. Si no pasa esto automáticamente, abra una pestaña en cualquier navegador web y, a continuación, abra el archivo `d6.html` (en la carpeta donde guardó el archivo `die_visual.py`). Debería ver un gráfico parecido al de la figura 15.12. (He modificado un poco el gráfico para la impresión; por defecto, Plotly genera gráficos con el texto más pequeño que el que vemos aquí).

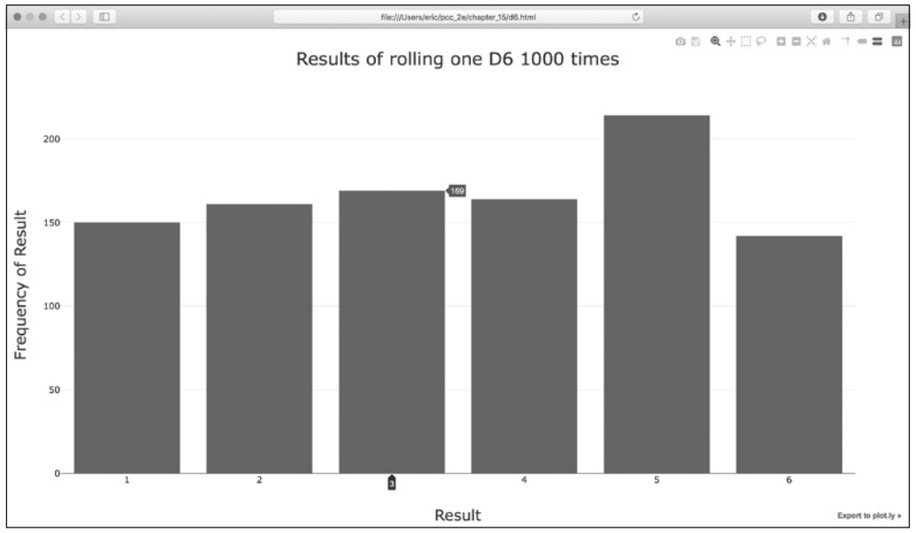


Figura 15.12. Un sencillo gráfico de barras creado con Plotly.

Observe que Plotly ha hecho el gráfico interactivo: al pasar el cursor por encima de cualquier barra, se ven los datos asociados. Esta función es especialmente útil cuando trazamos múltiples conjuntos de datos en un mismo gráfico. Fíjese también en los iconos de arriba a la derecha, que permiten ver una panorámica o acercar la vista y guardar la visualización como imagen.

Tirar dos dados

El resultado de tirar dos dados es números más altos y una distribución distinta de los resultados. Vamos a modificar nuestro código para crear dos D6 y simular la forma en que lanzamos un par de dados. En cada lanzamiento, sumaremos los dos números (uno de cada dado) y guardaremos la suma en `results`. Guarde una copia de `die_visual.py` como `dice_visual.py` e introduzca estos cambios:

dice_visual.py

```
from plotly.graph_objs import Bar, Layout
from plotly import offline

from die import Die

# Crea dos dados D6.
die_1 = Die()
die_2 = Die()

# Hace algunas tiradas y guarda los resultados en una lista.
results = []
for roll_num in range(1000):
❶    result = die_1.roll() + die_2.roll()
    results.append(result)

# Analiza los resultados.
frequencies = []
❷ max_result = die_1.num_sides + die_2.num_sides
❸ for value in range(2, max_result+1):
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualiza los resultados.
x_values = list(range(2, max_result+1))
data = [Bar(x=x_values, y=frequencies)]

❹ x_axis_config = {'title': 'Result', 'dtick': 1}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(title='Results of rolling two D6 dice 1000 times',
    xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6_d6.html')
```

Después de crear dos instancias de `Die`, tiramos los dados y calculamos la suma de cada tirada ❶. El resultado más alto posible (`12`) es la suma del número máximo de caras de cada dado, que almacenamos en `max_result` ❷. El resultado más pequeño posible (2) es la suma del número mínimo de caras de cada dado. Al analizar los resultados, contamos el número de resultados para cada valor entre `2` y `max_result` ❸. (Podríamos haber usado `range(2, 13)`, pero esto solo funcionaría para dos dados D6. Cuando modelamos situaciones del mundo real, es mejor escribir código que pueda ocuparse de distintas situaciones. Este nos permite simular el lanzamiento de dos dados con cualquier número de caras).

19

CUENTAS DE USUARIO

La gracia de una aplicación web es que pueda utilizarla cualquier usuario, en cualquier parte del mundo, para registrar una cuenta y empezar a usarla. En este capítulo, vamos a construir formularios para que el usuario puede añadir sus propios temas y entradas y editar las entradas que ya existan. También aprenderá cómo se protege Django de ataques habituales a páginas basadas en formularios para que no tenga que perder mucho tiempo pensando en la seguridad de sus aplicaciones.

Además, implementaremos un sistema de autenticación de usuarios. Crearemos una página de registro para que los usuarios creen cuentas y restringiremos el acceso a determinadas páginas solo para usuarios registrados. Después, modificaremos algunas de las funciones de vista para que los usuarios puedan ver solo sus propios datos. Aprenderá a mantener los datos de sus usuarios a salvo.

Permitir que los usuarios introduzcan datos

Antes de crear un sistema de autenticación para crear cuentas, vamos a añadir unas páginas para permitir que los usuarios introduzcan sus propios datos. Les daremos la capacidad de añadir temas y entradas nuevos y de editar sus entradas anteriores.

Ahora mismo, solo un superusuario puede introducir datos a través del sitio admin. No queremos que los usuarios interactúen con este sitio, así que usaremos las herramientas para construir formularios de Django para crear páginas que permitan a los usuarios introducir datos.

Instalar Python

Para instalar la versión más reciente de Python, introduzca el siguiente comando:

```
$ brew install python
```

Compruebe qué versión se ha instalado con este otro comando:

```
$ python3 --version
Python 3.7.2
$
```

Ya puede empezar una sesión de terminal de Python con el comando `python3`. También puede usar el comando `python3` en su editor de texto para que ejecute programas con la versión de Python que acaba de instalar y no con la que estuviera antes en el sistema. Si necesita ayuda para configurar Sublime Text para que use la versión que acaba de instalar, consulte las instrucciones del capítulo 1.

Python en Linux

Python está incluido por defecto en casi todos los sistemas Linux. Pero, si la versión predeterminada es anterior a Python 3.6, debería instalar la más reciente. Las siguientes instrucciones deberían funcionar para la mayoría de los sistemas basados en apt.

Utilizaremos un paquete llamado `deadsnakes`, que hace más fácil instalar varias versiones de Python. Introduzca los siguientes comandos:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt install python3.7
```

Estos comandos deberían instalar Python 3.7 en su sistema. Introduzca el siguiente comando para iniciar una sesión de terminal que ejecute Python 3.7:

```
$ python3.7
>>>
```

También le conviene usar este comando cuando configure su editor de texto y cuando ejecute programas desde el terminal.

Palabras clave y funciones integradas de Python

Python tiene su propio conjunto de palabras clave y funciones integradas. Es importante tenerlo en cuenta a la hora de poner nombre a las variables: los nombres que ponga no pueden coincidir con estas palabras clave ni deberían ser los mismos que los de las funciones, porque, si lo son, las sobrescribirá.

Esta sección recoge las palabras clave y las funciones integradas de Python para que sepa qué nombres debe evitar.

Palabras clave de Python

Cada una de las siguientes palabras clave tiene un significado específico y verá un error si intenta usar cualquiera de ellas como nombre de variable.

`False`	`await`	`else`	`import`	`pass`
`None`	`break`	`except`	`in`	`raise`
`True`	`class`	`finally`	`is`	`return`
`and`	`continue`	`for`	`lambda`	`try`
`as`	`def`	`from`	`nonlocal`	`while`
`assert`	`del`	`global`	`not`	`with`
`async`	`elif`	`if`	`or`	`yield`

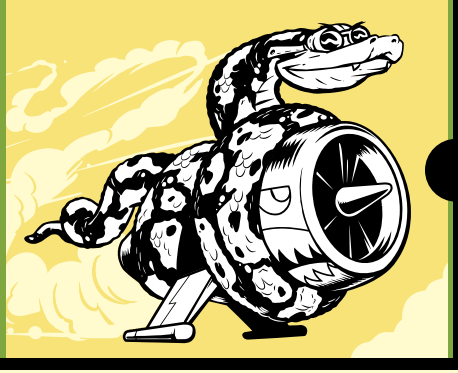
Funciones integradas de Python

No obtendrá un error si usa una de las siguientes funciones integradas como nombre de variable, pero anulará el comportamiento de esa función:

`abs()`	`delattr()`	`hash()`	`memoryview()`	`set()`
`all()`	`dict()`	`help()`	`min()`	`setattr()`
`any()`	`dir()`	`hex()`	`next()`	`slice()`
`ascii()`	`divmod()`	`id()`	`object()`	`sorted()`
`bin()`	`enumerate()`	`input()`	`oct()`	`staticmethod()`
`bool()`	`eval()`	`int()`	`open()`	`str()`
`breakpoint()`	`exec()`	`isinstance()`	`ord()`	`sum()`
`bytearray()`	`filter()`	`issubclass()`	`pow()`	`super()`
`bytes()`	`float()`	`iter()`	`print()`	`tuple()`
`callable()`	`format()`	`len()`	`property()`	`type()`
`chr()`	`frozenset()`	`list()`	`range()`	`vars()`
`classmethod()`	`getattr()`	`locals()`	`repr()`	`zip()`
`compile()`	`globals()`	`map()`	`reversed()`	`__import__()`
`complex()`	`hasattr()`	`max()`	`round()`	

SUPERVENTAS MUNDIAL

MÁS DE 500.000 COPIAS VENDIDAS



¡APRENDA PYTHON RÁPIDO!

Esta completa guía es una introducción rápida a la programación con Python con la que no tardará nada en empezar a escribir programas, resolver problemas y hacer cosas que funcionen. Esta segunda edición actualizada se ha revisado en profundidad para recoger lo último en código y prácticas de Python.

En la primera parte del libro, que incluye la cobertura mejorada de temas como las cadenas f, las constantes y la gestión de datos, conocerá conceptos de programación básicos, como "variables", "listas", "clases" y "bucles" y practicará la creación de código limpio con ejercicios sobre cada tema. También aprenderá a escribir sus propios programas interactivos y a probar su código con seguridad antes de añadirlo a un proyecto.

En la segunda mitad, se ha actualizado el código de los proyectos con una estructura mejor, una sintaxis más limpia y herramientas y librerías más populares y actualizadas, como Plotly y la versión más reciente de Django para poner sus nuevos conocimientos en práctica con tres proyectos sustanciosos: un juego arcade inspirado en *Space Invaders*, un conjunto de visualizaciones de datos con las útiles librerías de Python y una sencilla aplicación web que pueda 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.
- Hacer juegos en 2D que respondan a pulsaciones de teclado y clics de ratón y aumenten en dificultad.
- Usar datos para generar visualizaciones interactivas.
- Crear y personalizar aplicaciones web y desplegarlas con seguridad en línea.
- Vérselas con fallos y errores para poder resolver sus propios problemas de programación.

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

SOBRE EL AUTOR

Eric Matthes es profesor de ciencias, matemáticas y programación en un instituto en Alaska. Lleva escribiendo programas desde que tenía cinco años y es el autor de *Python Flash Cards*, de No Starch Press.

CUBRE PYTHON 3.X

ANAYA MULTIMEDIA
www.anayamultimedia.es
no starch press
ISBN 978-84-415-4334-8
2315160
9 788441 543348