Aprende a programar con Minecraft

## **11** **GUARDAR Y CARGAR EDIFICIOS CON ARCHIVOS Y MÓDULOS**



Los archivos son una parte importante de la informática. Te permiten guardar datos para almacenarlos a largo plazo y cargar datos para utilizarlos en tus programas. Hasta ahora, los programas con los que has estado trabajando sólo han almacenado datos en variables. O bien has utilizado datos codificados en tus programas, o bien has tomado datos de la entrada del usuario. Aunque puedes hacer cosas increíbles con estos datos, estás limitado a sesiones individuales y a tu propio mundo Minecraft. Una vez que aprendas a almacenar y recuperar datos, podrás guardar y cargar tus creaciones de Minecraft en cualquier mundo de Minecraft, ¡incluso en los juegos de tus amigos!

En este capítulo aprenderás a obtener datos de archivos y a enviar datos a archivos. Utilizarás algunas funciones incorporadas en Python y aprenderás a utilizar dos *módulos* de Python, los módulos pickle y shelve, para almacenar creaciones enteras de Minecraft.

Los módulos pueden ampliar lo que es posible lograr con Python. Utilizando módulos, puedes dibujar imágenes en la pantalla o ejecutar sitios web. Los módulos también proporcionan funciones para tareas comunes para que no tengas que escribir tus propias soluciones.

También aprenderás sobre pip, que es un programa muy útil para instalar nuevos módulos. Lo probarás utilizando el módulo Flask para hacer un sitio web sencillo que se conecte a Minecraft y muestre la posición del jugador.

### **Utilizar archivos**

Cuando trabajas con ordenadores, trabajas con archivos todo el tiempo. Cada vez que escribes un documento de texto o algo de código Python y lo guardas, estás trabajando con archivos. El texto, las imágenes, los vídeos y la música son archivos. Incluso este libro se guardó como un archivo de texto mientras lo escribía. Las funciones de manejo de archivos de Python son fáciles de aprender y te permitirán crear archivos, guardar archivos y leer información de archivos para hacer cosas interesantes en Minecraft. Empecemos por lo básico y aprendamos a leer y escribir en un archivo de texto en Python.

#### **Abrir un archivo**

Abrir un archivo es el primer paso cuando trabajas con archivos. Para abrir un archivo en Python, utilizas la función open(), que toma dos argumentos: la ubicación del archivo y sus permisos. La *ubicación* de un archivo es dónde está almacenado en tu ordenador. La pasarás a la función open () en forma de cadena. *Los permisos* del archivo controlan si Python puede o no leer o modificar el archivo.

Para abrir (o crear) un archivo de texto llamado archivosecreto *.txt* en Python, utilizarías el argumento "archivosecreto.txt":

secretFile = open("secretFile.txt", "w")

El segundo argumento, "w", es el argumento de permisos, que especifica lo que el programa puede hacer con el archivo. En este caso, w significa que el programa puede escribir datos en el archivo archivoarchivo\_secreto *.*txt.

Cuando un programa llama a la función open() con un nombre de archivo, Python comprueba primero si ya existe un archivo con ese nombre. Si el archivo existe, Python utilizará su contenido en el programa. Si no existe, Python creará un nuevo archivo con ese nombre.

Si no especificas un *directorio* con el nombre del archivo (una carpeta y un directorio son lo mismo), Python buscará el archivo en el directorio donde se encuentre el programa. Si el archivo está almacenado en un directorio diferente, debes especificarlo en el argumento. Por ejemplo, si secretFile *.txt* estuviera en el directorio *secrets*, el primer argumento sería "/secrets/secretFile.txt":

secretFile = open("/secrets/secretFile.txt", "w")

Si proporcionas un directorio en el argumento y el directorio no existe, o si el archivo no existe, recibirás un mensaje de error.

Hay cuatro opciones para el argumento permisos:

w Significa sólo *escritura*. Los permisos de sólo escritura permiten al programa escribir nuevos datos en el archivo y sobrescribir el contenido que ya está en el archivo, pero el programa no puede leer el contenido del archivo. Si no existe un archivo con el nombre que proporcionas como primer argumento, el programa creará uno nuevo.

r Significa *sólo* lectura. Los permisos de sólo lectura permiten al programa leer el contenido del archivo, pero no pueden modificarlo. Este permiso no puede utilizarse para crear un nuevo archivo.

r+ Significa *lectura y* escritura. Los permisos de lectura y escritura permiten al programa leer y modificar el contenido del archivo. El programa también puede escribir sobre cualquier contenido que ya esté en el archivo. Sin embargo, si el archivo no existe, no se creará uno nuevo; en su lugar, obtendrás un error.

a Significa *añadir*. Los permisos para añadir permiten al programa escribir nuevos datos sólo al final del archivo, dejando intacto el resto del contenido. El programa tampoco puede leer el contenido del archivo. Este permiso puede utilizarse para crear un nuevo archivo.

Hay distintas circunstancias en las que utilizarías cada tipo de permiso. Digamos que escribes unas instrucciones para llegar a una impresionante mina de diamantes que has encontrado, y quieres cargar las instrucciones en Minecraft sin cambiarlas accidentalmente. En ese caso, querrías utilizar el permiso de sólo lectura para asegurarte de que no cambia nada del archivo. Alternativamente, si quieres que alguien pueda añadir datos a un archivo pero no quieres que vea los demás datos almacenados en el archivo, utilizarías el permiso append. Por ejemplo, podrías utilizar append si quieres que tus amigos añadan notas a un diario de viaje compartido, ¡sin dejar que se metan dentro y lean todos tus tesoros secretos!

A continuación, aprenderás a escribir datos en un archivo abierto, y aprenderás a cerrar ese archivo para utilizar esos datos más tarde.

#### **Escribir y guardar un archivo**

La función write( ) escribe datos en un archivo que el programa ha abierto. Es el pan de cada día del trabajo con archivos, porque te permite guardar todo tipo de datos. Proporcionas los datos que quieres escribir en el archivo como argumento de la función write( ).

Por ejemplo, vamos a abrir un fichero y escribir en él una simple cadena:

secretFile = open("secretFile.txt", "w")  
secretFile.write("This is a secret file. Shhh! Don't tell anyone.")  
➊ secretFile.close()

En primer lugar, debes abrir el fichero utilizando la función open( ). A continuación, utiliza la notación por puntos para llamar a la función escribir( ) y escribir una cadena en *archivo secreto.txt*. A continuación, debes llamar a la función close( ) ➊, que guarda y cierra el archivo. Es importante que recuerdes incluir la función close( ); de lo contrario, los datos no se guardarán en el archivo.

Ejecuta el programa y luego abre *secretFile.txt* en un editor de texto para ver si se ha guardado tu mensaje secreto. Intenta cambiar la cadena para escribir algo diferente en el archivo y vuelve a ejecutar el programa. ¿Qué ocurre? ¡El mensaje antiguo debería haber sido sustituido por el mensaje nuevo! Prueba a cambiar de nuevo el mensaje, pero en lugar de pasar "w " pasa en su lugar "a". ¿Qué ocurre ahora? Muy chulo, ¿eh?

#### **Leer un archivo**

La función read( ) lee todo el contenido de un archivo que un programa ha abierto. Puede que quieras utilizar los datos en tu programa, modificarlos y luego devolverlos al archivo, o dar salida a los datos para facilitar su consulta. Sea cual sea el motivo, utilizarás la función read() para leer archivos.

Para leer un fichero, primero debes abrirlo y luego acordarte de cerrarlo cuando hayas terminado. Es importante que aprendas este hábito cuando trabajes con archivos en tus programas, ¡para evitar errores!

Vamos a leer un archivo y luego mostrar su contenido para que podamos ver lo que dice. Este programa, *showSecretFile.py*, muestra el contenido de un fichero utilizando las funciones read() e print():

*showSecretFile.py*

secretFile = open("secretFile.txt", "r")  
  
➊ print(secretFile.read())  
secretFile.close()

En primer lugar, abrimos el archivo y pasamos "r" como argumento de permiso para que nuestro programa pueda leer el archivo. También puedes pasar "r+", pero en este caso no vamos a escribir en el archivo, así que es mejor "r ". Para imprimir el contenido de *archivo\_secreto.txt*, pasamos archivo\_secreto.read() a una sentencia print. Por último, aunque no hayamos escrito ningún dato en el archivo, conviene cerrarlo con la función close().

Ejecuta el programa para ver qué ocurre. El contenido de *archivo\_secreto.txt* debería imprimirse en la pantalla. ¡Ahora puedes leer el archivo sin tener que abrirlo en un editor de texto como harías normalmente!

#### **Leer una línea de un archivo**

Supongamos que tienes un documento de texto largo y quieres ver sólo una parte. Aquí es donde resulta útil la función readline( ). A diferencia de la función read (), que obtiene todo el contenido del archivo, la función readline () obtiene una sola línea del archivo cada vez.

Para probar la función readline (), primero añade un montón de texto a *secretFile.txt*. Puedes hacerlo utilizando un editor de texto o usando tus nuevas habilidades de Python para escribir un montón de información en él. Si utilizas Python para escribir en tu archivo, añade \n a tus cadenas cada vez que quieras una nueva línea. Por ejemplo, si escribes " Guay\nBaile\nFiesta " en un archivo, Python coloca "Guay" en una línea, "Baile" en la siguiente y "Fiesta" en la última, así:

Cool  
Dance  
Party

Cuando hayas añadido el texto a *secretFile.txt*, escribe este código en un archivo Python y guárdalo como *showSecretLines.py* en una nueva carpeta llamada *files*:

*showSecretLines.py*

secretFile = open("secretFile.txt", "r")  
  
print(secretFile.readline())  
print(secretFile.readline())  
print(secretFile.readline())  
  
secretFile.close()

Una vez más, debes abrir secretFile *.txt* antes de poder leer de él utilizando la función readline(). Como quieres que tu programa showSecretLines *.* py lea datos del archivo, debes volver a introducir r (o r+). A continuación, incluye tres sentencias print para imprimir las tres primeras líneas de archivosecreto *.txt*. Por último, vuelve a cerrar el archivo utilizando close().

La función readline () comienza con la primera línea de tu archivo. Cada vez que se utiliza la función readline (), lee automáticamente la línea siguiente. Esta función es muy práctica para imprimir un par de líneas del principio de archivos de texto.

**NOTA**

*La* función readline () *convierte el archivo en una lista de cadenas, donde cada elemento de la lista representa una sola línea. Si quieres imprimir una línea desde la mitad de un documento de texto, ¡podrías escribir un bucle para encontrar e imprimir una cadena concreta de la lista!*

#### **Misión nº 64: Lista de tareas pendientes**

A veces puede que no tengas mucho tiempo libre para jugar a Minecraft. Puede que construyas estructuras complejas a lo largo de varios días en sesiones cortas. A medida que añadas programas para abrir puertas o teletransportar al jugador a algún lugar, tus construcciones se volverán más complejas y podrías tardar más en terminarlas. Trabajar en proyectos a lo largo de varios días puede hacer que olvides lo que estabas haciendo y lo que tienes que hacer a continuación. Esto me ocurre a menudo. Afortunadamente, ¡puedes crear un programa que te ayude a recordar!

Los programas de esta misión crean una lista de tareas pendientes y la muestran en el chat de Minecraft. Puedes utilizar este programa para hacer un seguimiento de tus objetivos en Minecraft, de modo que cuando tengas que dejar de jugar, puedas retomarlo fácilmente donde lo dejaste.

Para hacer la lista de tareas, escribirás dos programas distintos: uno para escribir la lista y otro para mostrarla. Empecemos creando el programa que escribe la lista.

#### **Parte 1: Escribir la lista de tareas pendientes**

En primer lugar, necesitas un programa para crear los elementos de la lista de tareas. [El listado 11-1](ch11.xhtml#ch11ex1) te inicia, utilizando un bucle while y la función input() para añadir elementos a la lista de tareas pendientes. Cópialo en un archivo en IDLE y guárdalo como *inputToDoList.py* en la carpeta de *archivos*.

*inputToDoList.py*

➊ toDoFile =  
  
➋ toDoList = ""  
  
➌ toDoItem = input("Enter a to-do list item: ")  
  
➍ while toDoItem != "exit":  
➎ toDoList = toDoList + toDoItem + "\n"  
toDoItem = input("Enter a to-do list item: ")  
  
➏ # Write the to-do list to the file  
➐ # Close the file

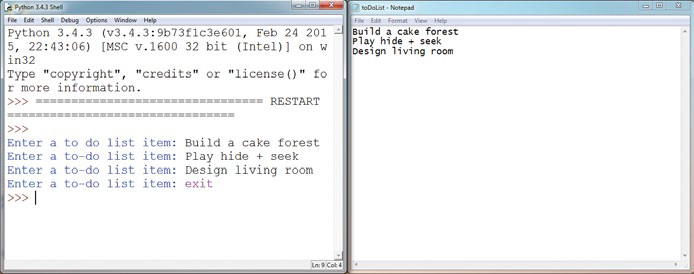
*Lista 11-1: Inicio del programa para escribir elementos en tu lista de tareas pendientes*

El programa crea una cadena vacía llamada toDoList ➋, que almacenará todos los elementos de tu lista de tareas pendientes cuando los introduzcas. Utilizando la función input(), el programa te pide que introduzcas un elemento en la lista de tareas pendientes ➌. A continuación, el bucle while comprueba si la entrada no es igual a "exit " ➍; si no lo es, el programa añade tu elemento a la lista de tareas pendientes con una nueva línea al final utilizando "\n" ➎. Sin embargo, si introduces "exit", el bucle deja de ejecutarse y no podrás añadir más elementos a la lista de tareas pendientes.

Tu misión es terminar el programa. Para ello, necesitas escribir el código que abra el archivo, escriba toDoList en el archivo y, a continuación, cierre el archivo. Utiliza la función open() para abrir el archivo al inicio del programa ➊. Debes abrirlo con permisos de escritura. Nombra el archivo que abre la función *comoHacerLista.txt*. Tu programa creará el archivo si no existe ya en el directorio.

Al final del programa, escribe el contenido de la lista de tareas en el archivo para poder acceder a él más tarde. Utiliza la función write() para escribir la variable toDoList en el archivo toDoFile ➏. Después de escribir en el archivo, asegúrate de cerrarlo con la función close () en la última línea ➐.

[La Figura 11-1](ch11.xhtml#ch11fig1) me muestra escribiendo una lista de tareas con el programa. Cuando termino, escribo exit.



*Figura 11-1: Introduciendo cosas que hacer, como construir un bosque de pasteles y jugar al escondite*

#### **Parte 2: Visualización de la lista de cosas por hacer**

Ahora que tienes un programa para escribir una lista de cosas por hacer en un archivo, necesitas mostrar la lista de cosas por hacer en el chat de Minecraft, línea a línea. El listado [11-2](ch11.xhtml#ch11ex2) inicia el programa por ti. Copia el listado en un nuevo archivo y guárdalo como *outputToDoList.py* en la carpeta de *archivos*.

*outputToDoList.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ toDoList =  
  
for line in toDoList:  
➋ # Output "line" to the chat

*Listado 11-2: Programa para enviar el archivo de la lista de tareas al chat de Minecraft*

[El listado 11-2](ch11.xhtml#ch11ex2) utiliza un bucle for para enviar cada línea del archivo *toDoList.txt* al chat de Minecraft, de una en una. De momento, el programa está incompleto. Para terminar el programa, añade la función open() para abrir el archivo *toDoList.t* xt que creaste con *inputToDoList.py* ➊. Asegúrate de que el archivo tiene permisos de lectura. Después de abrir el archivo, añade código dentro del bucle for para enviar la cadena almacenada en la variable line al chat de Minecraft ➋. Para ello tendrás que utilizar las funciones readline() y postToChat().

[La Figura 11](ch11.xhtml#ch11fig2) -2 muestra mi lista de tareas pendientes en el chat de Minecraft.



*Figura 11-2: Ahora, cuando vuelva a construir, podré ver lo que tengo que hacer.*

### **Uso de módulos**

*Los* módulos son colecciones de funciones que puedes importar a Python para no tener que escribir esas funciones en tus programas. Un módulo suele tener una finalidad específica, como realizar cálculos científicos o crear juegos, y hay una gran variedad de módulos disponibles para Python. ¡Puede que te sorprenda saber que has estado utilizando módulos a lo largo de este libro! La API Python de Minecraft es un módulo: cada vez que has escrito from mcpi.minecraft import Minecraft, has estado utilizando un módulo. El módulo API Python de Minecraft te permite conectar tus programas Python a Minecraft. Como está preescrito por otra persona, puedes utilizar las funciones del módulo sin tener que escribir el código tú mismo.

Python viene con un montón de módulos que puedes utilizar en tus programas. Estos módulos, junto con todo el Python que has aprendido en este libro hasta ahora, se denominan *biblioteca estándar de Python*. También puedes instalar módulos que no formen parte de la biblioteca estándar; lo haremos en "[Instalar nuevos módulos con pip](ch11.xhtml#ch11lev1sec04)" en [la página 256](ch11.xhtml#page_256).

En esta sección, aprenderás todas las formas en que puedes configurar tus programas para que utilicen módulos. Como ejemplo, utilizaremos el módulo pickle, que proporciona formas más avanzadas de guardar y cargar datos con archivos que simplemente escribir y leer datos de ellos. Veamos ahora el módulo pickle.

#### **El módulo pickle**

El módulo pickle es muy útil cuando escribes datos complicados en un archivo. Por ejemplo, los diccionarios y las listas multidimensionales son difíciles de almacenar y recuperar utilizando las funciones estándar que hemos utilizado anteriormente en el capítulo. Aquí es donde el módulo pickle resulta útil.

El módulo pickle puede ahorrarte horas de escribir y depurar tus propias soluciones para almacenar datos complejos. También puedes utilizar el módulo pickle con datos sencillos: por ejemplo, puedes utilizarlo para almacenar números sin convertirlos a y desde cadenas, lo que es necesario para la entrada y salida estándar de archivos.

En otras palabras, puedes utilizar el módulo pickle para guardar el valor de una variable en un archivo y luego leer el valor de la variable directamente en otro programa sin ningún procesamiento adicional. El tipo de datos sigue siendo el mismo que cuando guardaste el valor, aunque el tipo de datos sea una cadena, un entero, un flotante o un booleano.

A continuación, aprenderás a importar módulos utilizando pickle como ejemplo. Luego utilizarás pickle para guardar algunos datos complejos: ¡un edificio entero de Minecraft!

#### **Importar pickle**

Para utilizar las funciones de cualquier módulo, tienes que importarlo utilizando la palabra clave import. En realidad, ya has utilizado la palabra clave importar para importar módulos, como el módulo del tiempo, así como funciones de la API Minecraft de Python.

Una vez que hayas importado el módulo en tu programa, puedes utilizar las funciones del módulo utilizando la notación punto. Incluye el nombre del módulo, un punto y la función que quieras utilizar. Vamos a importar el módulo pickle y a utilizar un par de sus funciones:

➊ import pickle  
  
locations = {'John': 'Forest', 'Phillipa': 'Mountains', 'Pete': 'City'}  
  
➋ secretFile= open("secretFile.txt", "wb")  
➌ pickle.dump(locations, secretFile)

Importamos el módulo pickle en ➊. A continuación abrimos *secretFile.txt* con un permiso de archivo especial, "wb" ➋. Cuando abres un archivo con pickle, debes añadir b al permiso del archivo. En este caso, " wb" escribe datos en el archivo utilizando un formato especial que requiere el módulo pickle.

La función dump () escribe en el archivo en ➌. La función dump () del módulo pickle almacena una variable en un archivo. Toma dos argumentos: los datos que se escribirán en el archivo y el archivo abierto en el que escribirá. Este ejemplo almacena las ubicaciones de los agentes secretos en un diccionario llamado ubicaciones y luego vuelca ese diccionario en un archivo llamado archivoSecreto. Como dump() pertenece al módulo pickle, debes utilizar la notación con puntos para especificar tanto el módulo como la función con pickle.dump(). A diferencia de las funciones de archivo estándar de Python, la función dump () guarda los datos en el archivo automáticamente: no necesitas cerrar el archivo con la función close().

El módulo pickle también te permite leer datos almacenados. Puedes utilizar la función load() de picklepara leer el contenido de un archivo. Toma un argumento, el archivo que quieres cargar, y devuelve el contenido del archivo. El siguiente ejemplo carga el diccionario de localizaciones que almacenamos anteriormente. Añade este código al programa:

import pickle  
  
➊ secretFile= open("secretFile.txt", "rb")  
locations = pickle.load(secretFile)

Primero abrimos el archivo con el permiso "rb" ➊, que permite a tu programa leer un formato de datos especial que utiliza pickle. Luego cargamos el diccionario.

Una vez cargado el diccionario, puedes tratarlo como cualquier otro diccionario. Por ejemplo, puedes acceder al valor de una de las claves. Sólo tienes que añadir este código después de la función pickle .load():

print(locations['Phillipa'])

Esto imprimirá 'Montañas', el valor de la clave 'Phillipa'. Esto se debe a que el diccionario del archivo no cambia cuando se carga en el programa con pickle :sigue siendo un diccionario, por lo que podemos acceder a sus claves y valores y utilizarlo como cualquier otro diccionario de Python. También podrías hacer lo mismo con una lista o una variable.

#### **Importar una función con la cláusula from**

Importar un módulo significa que tienes acceso a todas las funciones de ese módulo, pero a veces sólo necesitas una función de un módulo. Si quieres importar sólo una función, utiliza la cláusula from al importar el módulo. Esta cláusula te permite acceder a la función sin incluir el nombre del módulo y la notación de punto cada vez que llames a una función. Sólo tendrías que escribir function() en lugar de module.function().

A veces, cuando utilizas el módulo pickle, puedes querer utilizar sólo la función dump(), y no sus otras funciones. Para ello, cambia el código que escribiste antes para que tenga este aspecto:

➊ from pickle import dump  
  
locations = {'John': 'Forest', 'Phillipa': 'Mountains', 'Pete': 'City'}  
  
secretFile= open("secretFile", "wb")  
➋ dump(locations, secretFile)

La primera línea utiliza la cláusula from para importar sólo la función dump( ) del módulo pickle ➊. La última línea llama a la función dump () ➋. Fíjate en que no tiene notación de punto. Simplemente llamas al nombre de la función sin hacer referencia al nombre del módulo.

También puedes importar más de una función de un módulo utilizando from. Lo único que tienes que hacer es separar los nombres de las funciones con una coma. Por ejemplo, si quieres utilizar las funciones dump() y load() de pickle en el mismo archivo, puedes importarlas ambas:

➊ from pickle import dump, load  
locations = {'John': 'Forest', 'Phillipa': 'Mountains', 'Pete': 'City'}  
  
secretFile= open("secretFile", "wb")  
➋ dump(locations, secretFile)  
  
➌ locations = load(secretFile)  
print(locations['Phillipa'])

La primera línea utiliza la cláusula from con comas para importar las funciones dump( ) y load( ) ➊. Esto significa que más adelante en el programa, puedes utilizar estas funciones sin tener que incluir el nombre de la función y la notación de puntos, que puedes ver en ➋ y ➌.

#### **Importar todas las funciones con \***

También puedes importar todas las funciones de un módulo para no tener que incluir el nombre del módulo con la notación de puntos cada vez que lo utilices. Para ello, introduce un asterisco(\*) al final de la sentencia import, como se indica a continuación:

➊ from pickle import \*  
locations = {'John': 'Forest', 'Phillipa': 'Mountains', 'Pete': 'City'}  
  
secretFile= open("secretFile", "wb")  
➋ dump(locations, secretFile)  
  
➌ locations = load(secretFile)  
print(locations['Phillipa'])

Como este código importó todas las funciones del módulo utilizando un asterisco ➊, no necesitamos utilizar la notación con puntos cuando llamemos a las funciones dump() ➋ y load() ➌.

La opción \* es muy práctica, ¡pero conlleva un riesgo! Si trabajas con varios módulos, puede que dos módulos compartan los mismos nombres de función. Cuando esto ocurra, Python se confundirá y podrías obtener un error. Por eso, cuando trabajes con muchos módulos, es mejor que evites utilizar la opción \* y en su lugar importes sólo las funciones que necesites utilizar.

#### **Dar un apodo a un módulo**

A veces querrás cambiar el nombre de un módulo porque su nombre es demasiado largo y quieres utilizar un nombre más corto en tu programa. O quieres cambiar el nombre del módulo para que sea más fácil de recordar. O quizás quieras cambiar su nombre porque el módulo comparte el mismo nombre que otro módulo y quieres evitar conflictos.

Puedes utilizar la cláusula as con la sentencia import para dar a un módulo un *alias, un*apodo. Por ejemplo, este código importa el módulo pickle y lo renombra a p:

import pickle as p

Ahora, cada vez que quieras utilizar el módulo pickle, puedes escribir p en tu programa en lugar de pickle. Aquí tienes un ejemplo de esto en acción:

p.dump(locations, secretFile)

Observa que se utiliza p. dump() en lugar de pickle .dump(). Esto te ahorra tiempo, ¡porque no tienes que seguir escribiendo pickle!

#### **Misión nº 65: Salvar un edificio**

Construir cosas es mi parte favorita de Minecraft. Me he pasado horas construyendo casas, castillos, aldeas y muchas otras cosas. Pero cuando me traslado a otra parte del mapa o a un mundo diferente, tengo que dejar atrás mis creaciones. Seguro que tú también has tenido que abandonar algunas creaciones increíbles.

¿No sería genial que pudieras guardar tus edificios y llevártelos contigo cuando te trasladas a mundos diferentes? Pues con Pickle y la API de Python, ¡puedes hacerlo!

En esta misión, desarrollarás dos programas que guardarán y cargarán edificios en tu juego Minecraft. Un programa guardará el edificio, y el otro lo cargará. Ambos programas se basan en *duplicateArea.py* del [Capítulo 10](ch10.xhtml#ch10)[(página 229](ch10.xhtml#page_229)).

#### **Parte 1: Guardar el edificio**

El primer programa guardará un edificio en un archivo. [El listado 11-3](ch11.xhtml#ch11ex3) incluye el código para copiar el edificio. Copia el listado en un archivo en IDLE y guárdalo como *guardarEstructura.* py en la carpeta *archivos*.

*saveStructure.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import pickle  
  
  
def sortPair(val1, val2):  
if val1 > val2:  
return val2, val1  
else:  
return val1, val2  
  
  
➊ def copyStructure(x1, y1, z1, x2, y2, z2):  
x1, x2 = sortPair(x1, x2)  
y1, y2 = sortPair(y1, y2)  
z1, z2 = sortPair(z1, z2)  
  
width = x2 - x1  
height = y2 - y1  
length = z2 - z1  
  
structure = []  
  
print("Please wait..." )  
  
# Copy the structure  
for row in range(height):  
structure.append([])  
for column in range(width):  
structure[row].append([])  
for depth in range(length):  
➋ block = mc.getBlock(x1 + column, y1 + row, z1 + depth)  
structure[row][column].append(block)  
  
return structure  
  
➌ # Get the position of the first corner  
input("Move to the first position and press ENTER in this window")  
pos1 = mc.player.getTilePos()  
  
x1 = pos1.x  
y1 = pos1.y  
z1 = pos1.z  
  
➍ # Get the position of the second corner  
input("Move to the opposite corner and press ENTER in this window")  
pos2 = mc.player.getTilePos()  
  
x2 = pos2.x  
y2 = pos2.y  
z2 = pos2.z  
  
➎ structure = copyStructure( x1, y1, z1, x2, y2, z2)  
  
➏ # Store the structure in a file

*Listado 11-3: Código incompleto para guardar un edificio en un archivo*

La función copiarEstructura( ) copia un área del juego en un conjunto de listas tridimensionales ➊. Toma dos conjuntos de coordenadas como argumentos. He hecho un ligero cambio en la función copyStructure () en comparación con *duplicateArea.py*. He utilizado la función getBlockWithData () en lugar de la función getBlock() ➋. En lugar de obtener sólo el ID de bloque de un bloque en unas coordenadas determinadas, la función getBlockWithData () obtiene también el estado del bloque. Esto es útil para bloques como las escaleras, donde la dirección de las escaleras se almacena en el estado del bloque. Cuando se copie la estructura, las escaleras y otros bloques que estén orientados en una dirección determinada se construirán en la dirección correcta.

He incluido un código muy útil para que puedas utilizar la posición del jugador para establecer las coordenadas de un edificio que quieras copiar. Cuando ejecutes el programa, te pedirá que te desplaces a la primera esquina de la estructura y luego pulses INTRO en el intérprete de comandos de Python ➌. El programa utiliza la posición del jugador para obtener el primer conjunto de coordenadas del edificio. A continuación, te pide que te muevas a la esquina opuesta de la estructura y hagas lo mismo ➍. Como resultado, puedes situarte donde quieras para empezar a copiar tu edificio en lugar de escribir las coordenadas o codificarlas en tu programa.

Los valores de estas variables de coordenadas se pasan a la función copiarEstructura() en ➎. El valor devuelto se almacena en una variable llamada estructura.

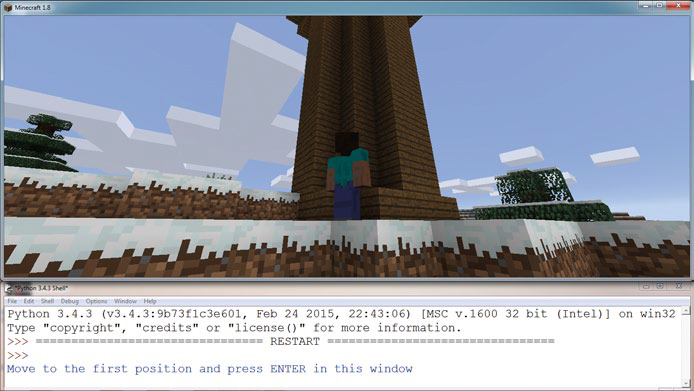
Para completar el código, necesitas abrir un nuevo archivo con pickle. Llama al nuevo archivo "pickleArchivo". A continuación, escribe el código para almacenar el edificio en el archivo. Hazlo utilizando el módulo pickle para escribir el valor de la variable estructura en un archivo ➏.

[La Figura 11-3](ch11.xhtml#ch11fig3) muestra una torre que construí en mi mundo Minecraft.



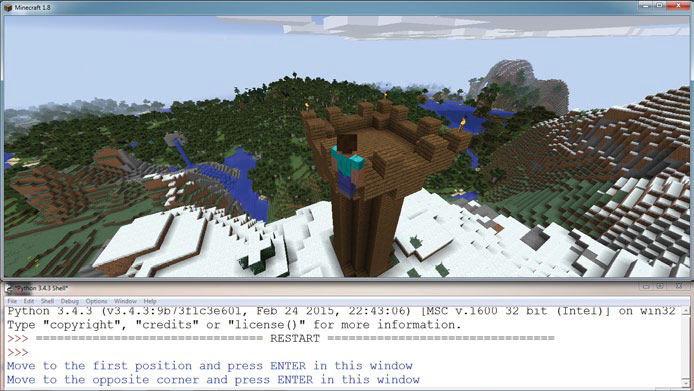
*Figura 11-3: Mi torre que quiero copiar*

Para copiar la torre utilizando *saveStructure.*py, me muevo a una esquina y pulso ENTER en IDLE[(Figura 11-4](ch11.xhtml#ch11fig4)).



*Figura 11-4: De pie junto a una esquina de la torre*

Luego me desplazo a la esquina opuesta de la torre y pulso ENTER en IDLE por segunda vez (Figura[11-5](ch11.xhtml#ch11fig5)).



*Figura 11-5: Volando hacia la esquina opuesta de la torre*

Sigue los mismos pasos para utilizar *saveStructure.* py para guardar uno de tus propios edificios. A continuación completaremos la otra mitad del proceso y cargaremos nuestros edificios guardados en el juego.

#### **Parte 2: Cargar el edificio**

El segundo programa necesita cargar el edificio en el juego desde el archivo (llamado *pickleFile*) creado por *saveStructure*.py. [El listado 11-4](ch11.xhtml#ch11ex4) incluye código del programa *duplicarArea.py*[(página 229](ch10.xhtml#page_229)) que coloca un edificio guardado en listas. Copia el listado en un archivo en IDLE y guárdalo como *loadStructure.* py en la carpeta de *archivos*.

*loadStructure.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import pickle  
  
  
➊ def buildStructure(x, y, z, structure):  
xStart = x  
zStart = z  
for row in structure:  
for column in row:  
for block in column:  
mc.setBlock(x, y, z, block.id, block.data)  
z += 1  
x += 1  
z = zStart  
y += 1  
x = xStart  
  
# Open and load the structure file  
➋ structure =  
  
➌ pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
➍ buildStructure(x, y, z, structure)

*Listado 11-4: Cuando esté completo, este programa construirá un edificio a partir de un archivo.*

La función buildStructure() ➊ hace la mayor parte del trabajo en este programa. Construye la estructura en el juego utilizando cuatro argumentos: coordenadas x, y y z y una estructura almacenada en una lista tridimensional.

Importa el módulo pickle para poder cargar la estructura en el programa, y luego almacénala en la variable estructura en ➋. Utilizando la función open(), abre el archivo *pickleFile* en el que guardaste la estructura. A continuación, cárgala en la variable estructura utilizando la función load() de pickle. Una vez cargada la estructura, cierra *pickleFile* utilizando la función close() de pickle.

También se incluye en [el Listado 11-4](ch11.xhtml#ch11ex4) un código que obtiene la posición del jugador para utilizarla como ubicación inicial de la estructura ➌.

Una vez cargada la estructura y fijadas las coordenadas, pasa la estructura a la función buildStructure( ) junto con una posición ➍, que construirá la estructura guardada.

[La Figura 11-6](ch11.xhtml#ch11fig6) muestra el programa en acción. El edificio que guardé antes se ha cargado en el juego y se ha reconstruido en una nueva posición. Pruébalo tú mismo: ¡ahora puedes llevarte tus creaciones dondequiera que vayas!



*Figura 11-6: ¡Mira, es una copia de mi torre!*

Pero, ¿y si creas un pueblo entero y quieres llevártelo contigo? Podrías guardar cada edificio en su propio archivo utilizando pickle, pero eso no es muy cómodo. El módulo pickle funciona muy bien para guardar un solo edificio, pero no es tan bueno para guardar un montón de edificios. Ahí es donde entra en juego el módulo shelve. Veámoslo a continuación.

### **Almacenar muchos datos con el módulo shelve**

El módulo pepinillo sólo puede almacenar un dato cada vez. En algunos programas, puede que quieras almacenar varias variables; si utilizas el módulo pickle, tendrás que crear varios archivos, lo que puede ser difícil de gestionar.

El módulo shelve de Python resuelve este problema. Puede almacenar varios datos en un único archivo. Funciona como un diccionario en el que cada valor de datos tiene una clave que puedes utilizar para almacenar y recuperar los datos. Piensa en shelve como en una estantería: cada compartimento de la estantería almacena distintos valores de datos.

#### **Abrir un archivo con shelve**

Tras importar el módulo shelve, utilizarás su función open() para abrir un archivo. Si el archivo no existe, se creará uno nuevo.

El siguiente código abre el archivo *locationsFile.db* y lo almacena en la variable shelveFile:

import shelve  
shelveFile = shelve.open("locationsFile.db")

La función open( ) sólo toma un argumento, el nombre del archivo. No necesitas especificar los permisos del archivo cuando utilices el módulo shelve, ya que éste concede automáticamente privilegios de lectura y escritura.

Al nombrar un archivo con el módulo shelve, debes incluir la extensión *.* db al final del nombre del archivo. Puedes ver el *.d* b al final de mi archivo *locationsFile.db*.

#### **Añadir, modificar y acceder a elementos con shelve**

El módulo shelve funciona como un diccionario. Para añadir datos al archivo, utilizas corchetes con un nombre de clave para almacenar un valor. Por ejemplo, supongamos que una agente secreta llamada Beatriz está en un submarino, y queremos almacenar la ubicación de Beatriz en el diccionario shelveFichero:

import shelve  
shelveFile = shelve.open("locationsFile.db")  
shelveFile['Beatrice'] = 'Submarine'  
shelveFile.close()

Primero abrimos el archivo. A continuación, damos al diccionario shelveFile una clave "Beatriz" y el valor "Submarino". Esta línea crea un nuevo elemento en el diccionario shelveFile con la clave " Beatriz " y el valor "Submarino ".

A continuación, utilizamos la función close() de shelvepara añadir los nuevos datos al archivo y cerrarlo de forma segura.

Si una clave ya existe en un archivo shelve, este código actualizaría el valor antiguo por el nuevo. Supongamos que Beatrice termina su misión y vuelve al cuartel general. Podrías actualizar la ubicación de Beatrice de esta forma:

import shelve  
shelveFile = shelve.open('locationsFile.db')  
shelveFile['Beatrice'] = 'Headquarters'  
shelveFile.close()

Ahora el valor correspondiente de la clave Beatrice es "Cuartel General". Acceder a un valor desde la estantería también funciona como en un diccionario. Utilizas claves para acceder a valores concretos. Por ejemplo, para imprimir la ubicación de Beatriz, utilizaríamos el siguiente código:

import shelve  
shelveFile = shelve.open('locationsFile.db')  
print(shelveFile['Beatrice'])

Esto mostrará la ubicación de Beatriz, Sede Central.

Al igual que un diccionario estándar, el módulo estantería puede almacenar cualquier tipo de datos, incluidos flotantes, cadenas, booleanos, listas multidimensionales, otros diccionarios, etc. De hecho, en la siguiente misión, ¡almacenarás y accederás a listas multidimensionales para guardar y cargar múltiples estructuras!

#### **Misión nº 66: Guardar una colección de estructuras**

Los programas de esta misión almacenarán y cargarán todas tus estructuras guardadas utilizando un único archivo. Esta misión se divide de nuevo en dos programas: uno para guardar y otro para cargar.

Tendrás que convertir los programas de la Misión nº 65 para que utilicen el módulo shelve en lugar del módulo pickle. También añadirás código para tomar la información del usuario, de modo que éste pueda poner nombre a sus edificios. Abre los archivos *saveStructure.py* y *loadStructure.py* y guárdalos como *saveCollection.py* y *loadCollection.py*.

Como hicimos en la misión anterior, vamos a realizar los cambios en estos programas en dos partes.

#### **Parte 1: Guardar una estructura en una colección**

Aquí se incluye y anota parte del archivo original *saveStructure.* py para ayudarte a identificar dónde realizarás los cambios. Aquí tienes la primera línea y las últimas líneas de *saveCollection*.py:

*saveCollection.py*

➊ import pickle  
  
--snip--  
  
# Name the structure  
➋ structureName = input("What do you want to call the structure?")  
# Store the structure in a file  
➌ pickleFile = open("pickleFile", "wb")  
➍ pickleFile.dump(structure)

Se añade una línea extra al archivo para preguntar cómo quieres llamar a la estructura cuando la guardes con pickle ➋. Por ejemplo, mi versión del programa pregunta "¿Cómo quieres llamar a la estructura?" y puedo responder con algo como "Casa" o "Bosque de pasteles". Asegúrate de llamar a cada nueva estructura por un nombre distinto; si una nueva estructura tiene el mismo nombre que otra, la estructura antigua será sobrescrita por la nueva.

Para cambiar este programa para que utilice el módulo shelve en lugar de pickle, tienes que hacer dos cambios. Primero, cambia la importación del módulo pickle por shelve ➊. En segundo lugar, cambia las últimas líneas del código para utilizar shelve en lugar de pickle. Abre un archivo llamado *structuresFile.db* y guárdalo en una variable llamada shelveFile utilizando la función shelve.open( ) ➌. A continuación, almacena la variable de estructura en un diccionario shelve utilizando la variable structureName para el nombre de la clave del diccionario ➍. Debería quedar algo así: shelveFile [nombredelaestructura] = estructura. Por último, cierra shelveFile en la última línea utilizando close().

#### **Parte 2: Cargar una estructura desde una colección**

Ahora tienes que modificar el archivo *loadCollection.py*. He eliminado la parte central del archivo para ahorrar espacio y para que las partes que tienes que cambiar sean más fáciles de ver:

*loadCollection.py*

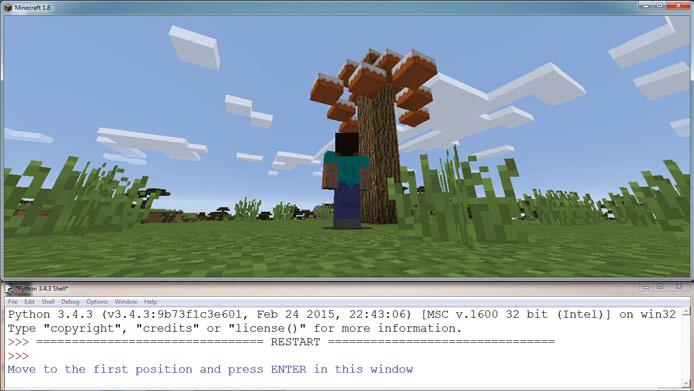
➊ import pickle  
  
--snip--  
➋ structure = pickle.load("pickleFile")  
➌ structureName = input("Enter the structure's name")  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➍ buildStructure(x, y, z, structureDictionary[structureName])

He añadido una línea extra al código que pide el nombre de la estructura que quieres construir ➌. Además, he añadido un poco de código a la última línea que obtiene la estructura del diccionario de estanterías y la pasa a la función buildStructure().

Tienes que hacer un par de cambios en este programa. Primero, al igual que en *saveCollection.py*, cambia import por shelve en lugar de pickle ➊. En segundo lugar, carga el archivo shelveFile que creaste en saveCollection *.* py utilizando shelve.open() ➋. Almacena los datos devueltos por la función shelve.open( ) en la variable structureDictionary ➍. El código debería ser algo parecido a lo siguiente: estructuraDiccionario = shelve.load("archivo shelve").

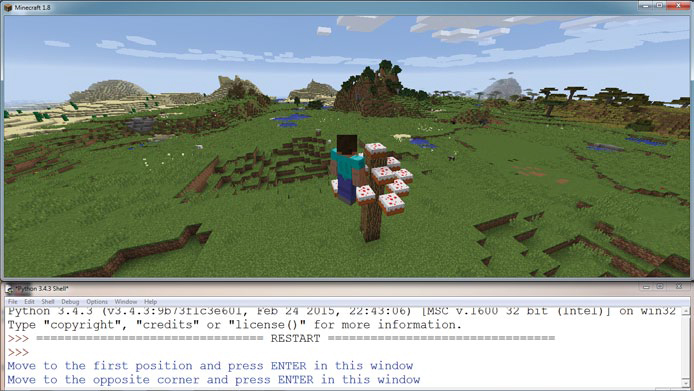
Todos los datos de las estructuras, incluidos sus nombres y bloques, se almacenan en el archivo *structuresFile.db*, lo que significa que no necesitas hacer ningún cambio en *loadCollection.* py antes de ejecutarlo. Todo lo que tienes que hacer es introducir el nombre de la estructura que quieres utilizar cuando ejecutes el programa.

Veamos el programa en acción, utilizando una estructura de mi mundo Minecraft. Primero, copio la estructura utilizando *saveCollection.* py volando hasta una esquina de la estructura y pulsando ENTER[(Figura 11-7](ch11.xhtml#ch11fig7)).



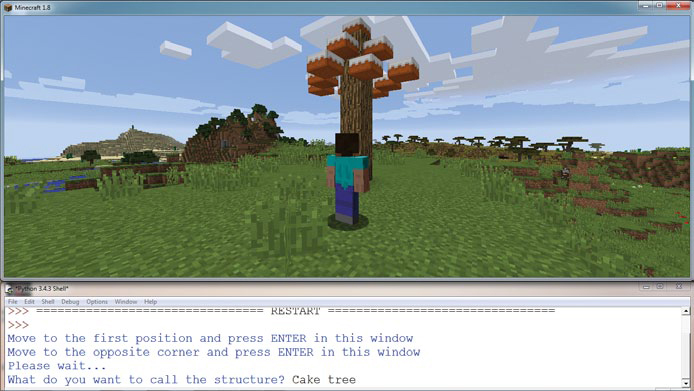
*Figura 11-7: Me desplazo a una esquina de la estructura que quiero guardar.*

Después, vuelo hasta la esquina opuesta de la estructura y pulso ENTER de nuevo[(](ch11.xhtml#ch11fig8)Figura 11-8).



*Figura 11-8: Me desplazo a la esquina opuesta.*

A continuación, el programa me pide que introduzca un nombre para mi estructura. La Figura [11-9](ch11.xhtml#ch11fig9) muestra que he llamado a mi estructura "Árbol de tartas".



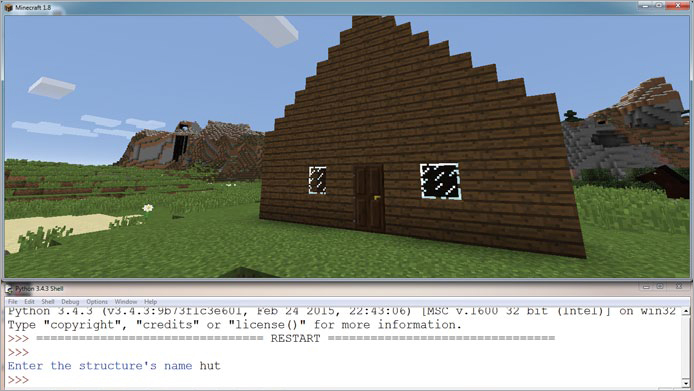
*Figura 11-9: Introduzco el nombre con el que quiero guardar la estructura.*

Por último, ejecuto *loadCollection.py*, vuelo a la ubicación donde quiero construir una copia de la estructura e introduzco el nombre de la estructura que quiero construir[(Figura 11](ch11.xhtml#ch11fig10)-10). El programa empieza a construir delante de mí, ¡como por arte de magia!



*Figura 11-10: Ahora, cuando quiero crear una copia, sólo tengo que introducir el nombre de la estructura y se construye.*

Puedes repetir este proceso con tantos edificios o estructuras como quieras; por ejemplo, yo he hecho una copia de una cabaña en la [Figura 11-11](ch11.xhtml#ch11fig11). Cuando hayas copiado una estructura una vez, podrás cargarla cuando quieras con sólo ejecutar *loadCollection.py* e introducir el nombre de la estructura.



*Figura 11-11: Puedes utilizar el programa para guardar varias estructuras. Aquí he copiado una cabaña.*

### **Instalar nuevos módulos con pip**

Además de pickle y shelve, puedes importar miles de módulos más para utilizarlos en tus programas Python. Con tantos módulos disponibles, instalarlos correctamente es muy importante. Para simplificar la instalación de módulos, Python proporciona un gestor de paquetes llamado pip. Un gestor de *paquetes* es un software que tiene una base de datos de otros programas que puedes instalar en tu ordenador. También incluye funciones que facilitan la instalación, actualización y desinstalación del otro software.

El gestor de paquetes pip puede instalar, actualizar y eliminar paquetes en Python. También tiene una gran colección de módulos que puedes utilizar en Python. Esta sección te muestra cómo instalar un paquete utilizando pip y muestra el módulo Flask, ¡que puedes utilizar para crear un sitio web!

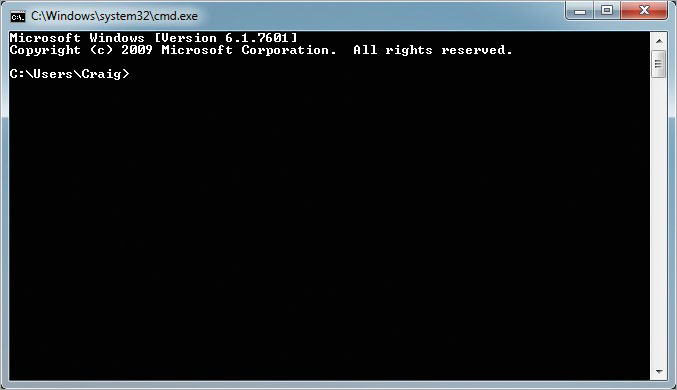
Si utilizas la versión más reciente de Python 3, pip está preinstalado. Si utilizas una versión anterior de Python, puede que pip no esté instalado. La forma más fácil de conseguir pip es instalar la última versión de Python. (Consulta "[Instalar Py](ch01.xhtml#ch01lev2sec02)thon" en la [página 3](ch01.xhtml#page_3) para Windows y en [la página 15](ch01.xhtml#page_15) para Mac).

Veamos cómo utilizar pip. Dependiendo del sistema operativo que utilices, puedes usar pip de un par de maneras. ¡Asegúrate de seguir las instrucciones que correspondan a tu ordenador!

#### **Utilizar pip en Windows**

Para utilizar pip en Windows, tienes que abrir el símbolo del sistema de Windows. El símbolo del sistema es similar al shell de Python. Te permite introducir un único comando en una línea, que se ejecuta cuando pulsas ENTER.

Para abrir el símbolo del sistema, pulsa la tecla Windows o abre el menú Inicio y busca cmd. Cuando abras el programa, verás una ventana negra[(Figura 11](ch11.xhtml#ch11fig12)-12).



*Figura 11-12: El símbolo del sistema de Windows*

Para utilizar pip en el símbolo del sistema, escribe pip seguido de la acción que quieres que realice. Por ejemplo, vamos a instalar el módulo Flask de Python, que puedes utilizar para crear sitios web con Python. Introduce el siguiente comando en el símbolo del sistema:

> pip install Flask

En el sitio web del índice de paquetes de Python en [*http://pypi.python.org/,*](http://pypi.python.org/) puedes encontrar muchos otros módulos de Python que puedes instalar.

#### **Utilizar pip en un Mac o Raspberry Pi**

Si utilizas pip en un Mac o en una Raspberry Pi, tendrás que incluir sudo al principio del comando para que funcione. Por ejemplo, introduce esta línea:

$ sudo pip install Flask

Si aparece un error, consulta el [Capítulo 1](ch01.xhtml#ch01) para volver a comprobar las instrucciones de instalación en Mac o Raspberry Pi.

En el sitio web del índice de paquetes de Python, [*http://pypi.python.org/,*](http://pypi.python.org/) puedes encontrar muchos otros módulos de Python que puedes instalar.

### **Utilizar un módulo de pip: Flask**

Flask es un módulo de Python que puedes utilizar para desarrollar sitios web. En esta sección, aprenderás a configurar un sitio web básico de Flask y luego a integrar el sitio web con Minecraft, ¡para que puedas mostrar la posición de tu jugador en un sitio web!

Con Flask, sólo necesitas unas pocas líneas de código para crear y gestionar un sitio web. Sólo tienes que escribir el código Python como lo harías normalmente y añadir a alguna información extra relacionada con Flask. A continuación, ejecutas tu código, y éste crea un sitio web al que tu ordenador puede acceder. Entonces podrás ver el sitio web en tu navegador.

[El listado 11-5](ch11.xhtml#ch11ex5) crea un sitio web Flask básico que incluye la información más importante sobre mí: mi nombre.

*namePage.py*

from flask import Flask  
➊ app = Flask(\_\_name\_\_)  
  
➋ @app.route("/")  
def showName():  
➌ return "Craig Richardson"  
  
➍ app.run()

*Listado 11-5: Un programa Python que utiliza* Flask *para crear un* sitio web

Para utilizar Flask, primero tienes que crear Flask utilizando la función Flask() ➊. El argumento \_\_name\_\_ indica a Flask que tu proyecto Flask está contenido en este archivo, y no necesita buscar en ningún otro sitio para encontrar otras partes del programa. Observa los dos guiones bajos, no uno, al principio y al final del argumento \_\_name\_\_.

La etiqueta @app.route( ) utiliza un decorador. *Los decor* adores proporcionan información adicional a Python sobre tus funciones. Por ejemplo, en este programa el decorador @app .route () indica a Flask en qué parte del sitio web se utilizará la función. En este caso, "/" le dice a Flask que la función showName () se utilizará en la página de inicio ➋. La declaración de retorno de la función indica a Flask lo que se mostrará en la página. En este ejemplo, devuelve mi nombre, por lo que mi nombre se mostrará en la página ➌. La última línea del programa indica a Python que inicie Flask cuando se ejecute este archivo ➍.

Guarda este archivo como *nombrePágina.py* en la carpeta de *archivos*. Añade tu propio texto para crear tu propio sitio web.

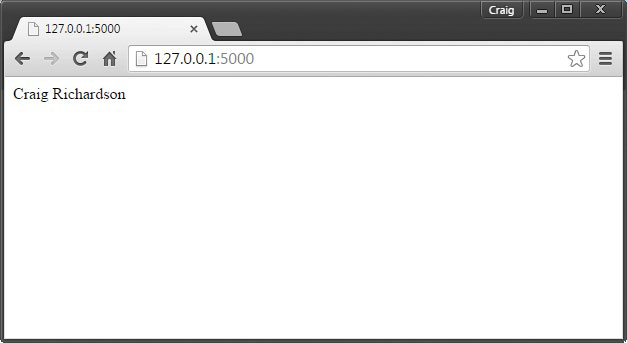
Para ejecutar el sitio web, haz clic en **Ejecutar** image **Ejecutar módulo** en IDLE. El programa se ejecuta y genera un archivo de sitio web que puedes abrir en un navegador web. Para encontrar la ubicación del sitio web, tienes que leer la línea de código que emite el programa cuando empiezas a ejecutarlo. Cuando ejecuto el programa, la salida tiene este aspecto:

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

A partir de esta línea, puedo saber que si introduzco *http://127.0.0.1:5000/* en un navegador web me llevará al sitio web de Flask que acabo de empezar a ejecutar. Ahora, cuando abro el navegador y entro en este sitio, puedo ver mi nombre en la página[(Figura 11-13](ch11.xhtml#ch11fig13)).

**NOTA**

*El sitio web que creas en este programa sólo está disponible en tu ordenador en este momento. Sólo tú puedes acceder a este sitio web, nadie más en Internet puede verlo.*



*Figura 11-13: ¡Mira mi sitio web! Puedes cambiar lo que se muestra aquí por lo que quieras. ¿Qué escribirás en tu sitio web?*

Para detener el programa, entra en IDLE y pulsa CTRL-C o haz clic en **Shell** image **Reiniciar Shell**.

**NOTA**

*Este capítulo sólo cubre una introducción muy básica a* Flask. Es *muy útil aprender Flask porque te permite construir sitios web interactivos rápidamente con Python. Si quieres saber más sobre* Flask*, consulta este tutorial en el* *sitio web* de Flask *en* [http://flask.pocoo.org/docs/0.10/tutorial/.](http://flask.pocoo.org/docs/0.10/tutorial/)

#### **Misión nº 67: Posicionar sitio web**

Una de las mejores cualidades de Python es que es fácil integrar las características de diferentes módulos en un único programa. Has estado utilizando el módulo API Python de Minecraft durante todo el libro y acabas de conocer el módulo Flask. Con unos pocos pasos puedes integrar ambos.

En esta misión, combinarás la API Python de Minecraft con Flask para mostrar la posición de un jugador en una página web.

Crea un nuevo archivo en IDLE y guárdalo como *positionPage.py* en la carpeta de *archivos*. Tienes que obtener la posición del jugador desde Minecraft y mostrarla en la página web utilizando una función con la etiqueta @app.route("/") de Flask. Puedes basar el código en el ejemplo del [Listado 11-5](ch11.xhtml#ch11ex5). Haz que el código muestre la posición en el formato "x 10, y 110, z 12".

Ejecuta el programa y comprueba tu página web. Muy chulo, ¿eh? Con Flask puedes crear páginas web con todo tipo de información. ¡Incluso puedes subir estas páginas a Internet para compartirlas con tus amigos!

### **Lo que has aprendido**

En este capítulo se te ha presentado el uso de archivos con Python. Aprendiste a leer y escribir en archivos utilizando la biblioteca estándar de Python, lo que te da control sobre los archivos cuando creas tus propios programas. También aprendiste a utilizar módulos, que amplían las capacidades de Python y lo que puedes hacer en Python.

Exploraste el módulo pickle, el módulo shelve y el gestor de paquetes pip. Los módulos pickle y shelve se utilizan con fines distintos. El módulo pickle guarda el valor de una única variable, especialmente cuando contiene una lista multidimensional o un diccionario, que sería difícil de almacenar y abrir utilizando la biblioteca estándar. El módulo shelve tiene los mismos puntos fuertes que el módulo pickle, pero te da más flexibilidad para almacenar varios valores a la vez en una estructura similar a un diccionario. Con pip aprendiste a instalar nuevos módulos. También conociste el módulo Flask, que es una forma rápida y flexible de crear sitios web con Python.

Con estos conocimientos, completaste cuatro misiones. La primera te permitió crear listas de tareas, que puedes mostrar en el juego Minecraft para recordar en qué estás trabajando. La segunda permitía guardar edificios y cargarlos en tu mundo actual y en otros mundos. La tercera misión modificó la segunda para que pudieras guardar todos tus edificios en un único archivo en lugar de un archivo para cada edificio. La última misión te mostró cómo utilizar el módulo Flask para crear una página web que muestre la posición actual del jugador.

¡Has hecho un gran trabajo hasta ahora! El próximo capítulo es el capítulo final. Aprenderás sobre las clases y la programación orientada a objetos, un popular estilo de programación que te permite reutilizar el código.