Aprende a programar con Minecraft

### **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.

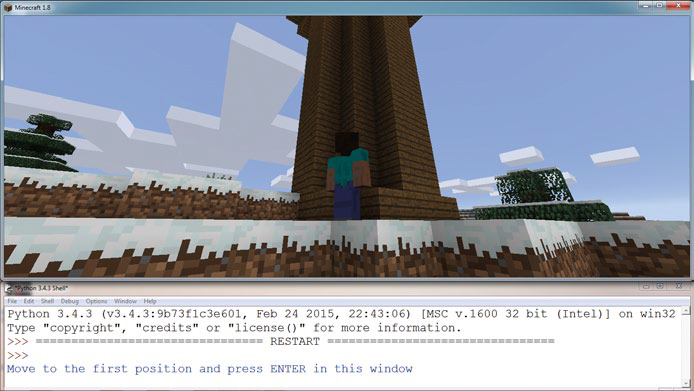
Para completar el código, necesitas abrir un nuevo archivo con pickle. Llama al nuevo archivo "pickleArchivo". A continuación, escribe el código para almacenar el edificio en el archivo. Hazlo utilizando el módulo pickle para escribir el valor de la variable estructura en un archivo ➏.

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



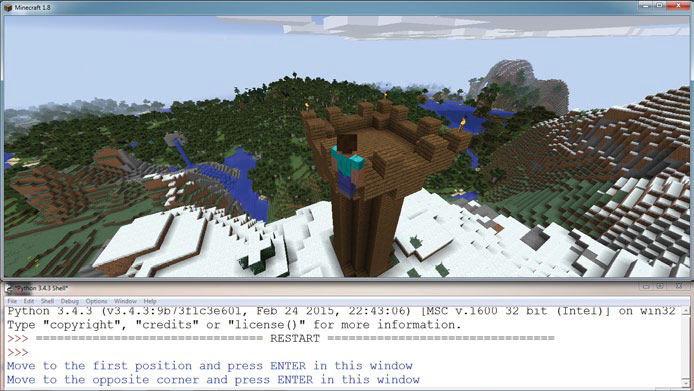
*Figura 11-3: Mi torre que quiero copiar*

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.

[anterior](ch11_2.html)[Subtema 3 de 7: (Ver todo)](ch11.html)[siguiente](ch11_4.html)