Aprende a programar con Minecraft

## **9** **GOLPEAR COSAS CON LISTAS Y DICCIONARIOS**



Utilizamos las listas, como las listas de la compra o las listas de instrucciones, para recordar un grupo de elementos o para seguir unos pasos en un orden determinado. Las listas en Python son muy similares: se utilizan para almacenar una colección de datos dentro de una secuencia. Una *lista* puede almacenar varios tipos de datos, como cadenas, números, booleanos e incluso otras listas.

Normalmente, las variables sólo pueden contener un valor. Las listas son útiles porque te permiten almacenar varios valores en una sola variable, como los números del 1 al 100 o los nombres de pila de tus amigos. En otros lenguajes de programación, las listas se denominan a veces *matrices*.

Puedes utilizar listas de IDs de bloques, coordenadas, o una variedad de otras cosas para ganar mucho poder sobre tu mundo Minecraft. Como las listas pueden almacenar varios tipos de valores en una sola variable, te dan una flexibilidad que una variable normal no puede ofrecer.

En este capítulo, aprenderás a utilizar listas con la API Python de Minecraft para crear un minijuego para registrar la altura, hacer una barra de progreso y escribir un programa que deslice aleatoriamente al jugador por el juego.

### **Utilizar listas**

Hacer una lista con Python es sencillo. Para definir una lista, pon entre corchetes cualquier número de valores, o ningún valor, lo que se denomina lista *vacía*. Cada elemento de una lista debe ir separado por una coma.

Por ejemplo, una lista de ingredientes para una sopa de fideos podría tener este aspecto:

>>> noodleSoup = ["water", "soy sauce", "spring onions", "noodles", "beef"]

La lista sopa de fideos contiene varios elementos y todos ellos son cadenas. Puedes crear una lista vacía como ésta:

>>> emptyList = []

Utiliza una lista vacía cuando quieras añadir valores más adelante en tu programa. Puedes almacenar cualquier tipo de datos en tu lista e incluso mezclar distintos tipos de datos. Por ejemplo, puedes tener una lista que contenga enteros y cadenas:

>>> wackyList = ["cardigan", 33, "goofballs"]

A veces tus listas serán muy largas, lo que dificultará su lectura por los humanos. Pero puedes formatear listas largas en varias líneas en Python para que los programadores puedan leerlas fácilmente. Utilizar varias líneas para los elementos no afecta a la ejecución del código Python. Por ejemplo, el siguiente formato para los ingredientes de la sopa funciona igual que la lista anterior de sopa de fideos:

>>> noodleSoup = ["water",  
"soy sauce",  
"spring onions"  
"noodles",  
"beef"]

A continuación veremos cómo puedes acceder a los elementos de una lista y modificarlos.

#### **Acceder a un elemento de la lista**

Para acceder a un valor de una lista, haz referencia a la posición del elemento en la lista, lo que se conoce como su *índice*. Utilizando el ejemplo de la sopa de fideos, puedes acceder al primer elemento de la lista así:

>>> print(noodleSoup[0])  
water

Es importante tener en cuenta que el primer índice de una lista es 0. El segundo elemento es el índice 1, el tercero es el índice 2, y así sucesivamente. La razón es que los ordenadores cuentan desde cero cuando utilizan listas.

Contar desde cero puede parecer una tontería, pero hay una buena razón para ello. Los primeros ordenadores eran muy lentos y tenían muy poca memoria. Era más rápido y eficaz empezar a contar los índices desde cero. Aunque hoy en día los ordenadores son mucho más rápidos, siguen contando desde cero.

También es importante tener en cuenta que si intentas acceder a un índice de lista que sea mayor que el número de elementos de la lista, recibirás un mensaje de error. La siguiente línea intenta imprimir el elemento en la posición 5 del índice:

>>> print(noodleSoup[5])

Aquí tienes parte del mensaje de error:

IndexError: list index out of range

El IndexError me dice que no hay datos en el índice al que quiero acceder. La posición 5 del índice de la lista no tiene datos porque está fuera de la longitud de la lista. ¡Python no puede devolver un valor que no existe!

#### **Cambiar un elemento de la lista**

Al igual que puedes cambiar el valor de las variables, también puedes cambiar elementos individuales de las listas. Esto se debe a que las listas son *mutables*, lo que significa que pueden modificarse. Para cambiar un elemento de una lista, utiliza la posición del índice del elemento y establece su valor del mismo modo que establecerías el valor de una variable (utilizando un signo igual).

Cambiemos el elemento ternera de la sopa de fideos por pollo. La ternera es el quinto elemento de la lista, por lo que tiene un índice de 4 (recuerda que en las listas se cuenta desde cero). Podemos cambiar fácilmente el elemento 4 por pollo, así:

>>> noodleSoup[4] = "chicken"

Ahora vamos a hacer algo guay con las listas en Minecraft.

#### **Misión nº 47: Alto y Bajo**

Cuando exploro el mundo de Minecraft, es interesante repasar mi viaje. Desde las montañas más altas hasta las cuevas más bajas, la exploración es una de mis actividades favoritas en el juego. A veces, cuando juego con amigos, hacemos carreras para ver quién llega más rápido al punto más alto o más bajo del juego. Para que nadie haga trampas, escribí un programa que almacena las coordenadas y más bajas y más altas que alcanza el jugador en 60 segundos.

Cuando ejecuto el programa, me dice los lugares más alto y más bajo del juego a los que he viajado durante un minuto. [El listado 9-1](ch09.xhtml#ch9ex1) contiene el código que he iniciado para ti. Cópialo en un nuevo archivo y guárdalo como *altoYbajo.py* en una nueva carpeta llamada *listas*.

*highAndLow.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
  
➊ heights = [100, 0]  
count = 0  
  
while count < 60:  
pos = mc.player.getTilePos()  
  
if pos.y < heights[0]:  
➋ # Set the lowest height to the y variable  
elif pos.y > heights[1]:  
➌ # Set the highest height to the y variable  
  
count += 1  
time.sleep(1)  
  
➍ mc.postToChat("Lowest: ") # Output lowest height  
➎ mc.postToChat("Highest: ") # Output highest height

*Listado 9-1: Inicio del código para obtener las posiciones más baja y más alta que visita el jugador*

El programa almacenará las coordenadas y más bajas y más altas a las que ha viajado en una lista llamada alturas ➊. El primer elemento de la lista (posición índice 0) almacena la coordenada más baja y el segundo (posición índice 1) almacena la más alta. Tenemos que empezar con un valor "más bajo" alto y un valor "más alto" bajo para que la primera vez que ejecutemos el programa, la posición del jugador sea el nuevo valor más bajo o más alto y se muestre en el chat. Aquí he utilizado por defecto un valor mínimo de 100 y un valor máximo de 0.

El bucle while se ejecuta una vez por segundo durante 60 segundos para actualizar constantemente los valores de las alturas. La sentencia if comprueba si la altura actual del jugador es inferior al valor más bajo almacenado en la lista ➋. A continuación, la sentencia elif comprueba si la altura actual es mayor que la posición más alta almacenada en la lista ➌.

Para completar el código, tienes que establecer el valor de la altura más baja, altura[0], al valor de pos.y en ➋. Recuerda que estableces los valores en listas como lo harías con una variable, por lo que la línea de código debería tener este aspecto: altura[0 ] = pos.y. También tienes que establecer la altura más alta, altura[1], al valor de pos.y ➌.

Por último, tienes que mostrar el valor de las alturas ➍ más baja y ➎ más alta en las dos últimas líneas del programa. Para ello, tendrás que acceder a las posiciones de los índices de las alturas más baja y más alta de la lista de alturas (de nuevo, el índice 0 es la altura más baja y el índice 1 es la altura más alta).

Ejecuta el programa y empieza a correr por el juego. Comprueba lo alto y lo bajo que puedes llegar. Después de 60 segundos, el bucle se detendrá y el programa mostrará tus alturas más alta y más baja. Ejecuta el programa varias veces y comprueba si puedes batir tu récord.

[La Figura 9-1](ch09.xhtml#ch9fig1) muestra uno de mis intentos.



*Figura 9-1: La coordenada y más baja que visité fue 15 y la más alta 102.*

**OBJETIVO EXTRA: UN ERROR INESPERADO**

En *highAndLow.py*, los valores por defecto para las posiciones más baja y más alta se establecen en 100 y 0. Esto no es un problema siempre que vayas más abajo de 100 y más arriba de 0. Sin embargo, si no vas más abajo de 100 y más arriba de 0, el valor no cambiará, lo que puede hacer que el programa sea impreciso. ¿Puedes solucionar esto?

### **Manipulación de listas**

Las listas tienen un conjunto de funciones incorporadas que te permiten manipularlas. Estas funciones incluyen operaciones comunes como añadir un elemento a una lista, insertar un elemento o borrar un elemento.

#### **Añadir un elemento**

Puedes añadir un elemento al final de una lista utilizando la función append(): sólo tienes que incluir como argumento el valor del elemento que quieres añadir.

La sopa de fideos estaría mejor si añadiéramos algunas verduras. Para ello, utiliza la función append():

>>> noodleSoup.append("vegetables")

Ahora la lista sopaDeFideos contiene una cadena "verduras " como último elemento de la lista.

Añadir elementos es muy útil cuando empiezas con una lista vacía. Utilizando la función append (), puedes añadir el primer elemento a una lista vacía:

>>> food = []  
>>> food.append("cake")

#### **Insertar un elemento**

También es posible insertar un elemento en medio de una lista. La función insertar () coloca un elemento entre dos elementos existentes y cambia las posiciones de índice de todos los elementos situados después del elemento recién insertado.

Esta función toma dos argumentos, la posición del índice donde quieres insertar el elemento y el valor que quieres insertar.

Por ejemplo, aquí está nuestra lista actual de sopa de fideos:

>>> noodleSoup = ["water", "soy sauce", "spring onions", "noodles", "beef",  
"vegetables"]

Añadamos "pimienta" a la lista en la tercera posición del índice:

>>> noodleSoup.insert(3, "pepper")

La lista actualizada contiene los siguientes valores después de la inserción:

["water", "soy sauce", "spring onions", "pepper", "noodles", "beef", "vegetables"]

Si intentas insertar un elemento en una posición de índice mayor que la longitud de la lista, el elemento se añadirá después del último elemento. Por ejemplo, si tu lista tiene siete elementos, pero intentas insertar en la posición 10, el elemento se añadirá al final de la lista.

>>> noodleSoup.insert(10, "salt")

Después de ejecutar este código, el último elemento de la lista será "sal":

["water", "soy sauce", "spring onions", "pepper", "noodles", "beef",  
"vegetables", "salt"]

Observa que la sal no está en la posición 10 del índice, sino en la posición 7.

#### **Borrar un elemento**

A veces querrás deshacerte de un elemento de una lista. Para ello, utiliza la palabra clave del. La palabra clave va delante del nombre de la lista, con la posición del índice del elemento que quieres borrar entre corchetes.

Por ejemplo, para eliminar el elemento "ternera ", que ahora está en la posición 5 del índice de la lista sopa de fideos, haz esto:

>>> del noodleSoup[5]

También puedes utilizar la palabra clave del en combinación con la función index() si quieres encontrar la posición de índice de un valor y luego borrarlo:

>>> beefPosition = noodleSoup.index("beef")  
>>> del noodleSoup[beefPosition]

Después de eliminar un elemento, las posiciones del índice de una lista cambiarán. Este es el aspecto que tendrá la lista después de eliminar "ternera" en la posición 5 del índice:

["water", "soy sauce", "spring onions", "pepper", "noodles", "vegetables", "salt"]

La posición del índice " verduras " cambia de 6 a 5, y la posición del índice "sal " cambia de 7 a 6. Ten en cuenta que sólo se verán afectados los índices posteriores al elemento eliminado; los índices anteriores al elemento eliminado no cambiarán. Tenlo en cuenta cuando borres elementos de tus listas.

#### **Misión nº 48: Barra de progreso**

Vamos a utilizar algunas de las funciones de lista para crear una barra de progreso en Minecraft. Se parecerá a la que ves en pantalla cuando estás descargando un archivo de Internet o cuando llevas la cuenta de tu siguiente subida de nivel en un juego de rol.

El programa utilizará la barra de progreso para contar hasta 10 segundos. Cuando se inicie el programa, la barra de progreso estará formada por bloques de cristal. Por cada segundo que pase, la barra de progreso sustituirá un bloque de cristal por un bloque de lapislázuli. La [Figura 9-2](ch09.xhtml#ch9fig2) muestra los cinco primeros pasos de la barra de progreso.



*Figura 9-2: La barra de progreso muestra el progreso al 50% (5 de cada 10 bloques son de lapislázuli).*

Abre IDLE y crea un nuevo archivo. Guárdalo como *progressBar.py* en la carpeta *de listas*. En [el Listado 9-2](ch09.xhtml#ch9ex2) encontrarás una versión incompleta del programa. Cópiala en tu editor de texto.

*progressBar.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
  
pos = mc.player.getTilePos()  
x = pos.x + 1  
y = pos.y  
z = pos.z  
  
# Add 10 glass blocks (ID 20) to this empty list  
➊ blocks = [ ]  
barBlock = 22 # Lapis lazuli  
  
count = 0  
while count <= len(blocks):  
  
mc.setBlock(x, y, z, blocks[0])  
mc.setBlock(x, y + 1, z, blocks[1])  
mc.setBlock(x, y + 2, z, blocks[2])  
➋ # Add setBlock() for the remaining blocks in the list  
  
count += 1  
  
➌ # Delete the last block in the list  
  
➍ # Insert a lapis lazuli block at the first position in the list  
  
time.sleep(2)

*Listado 9-2: Código incompleto para hacer una barra de progreso*

Para completar el programa del Listado [9-2](ch09.xhtml#ch9ex2), tendrás que hacer lo siguiente:

1. Añadir 10 bloques de vidrio (ID 20) a la lista de bloques vacíos en ➊.
2. Utiliza la función setBlock() para colocar los 10 bloques ➋ de la lista en el mundo del juego. Los tres primeros bloques ya están definidos.
3. Escribe una sentencia que elimine el último bloque de la lista (posición índice 9) ➌. Recuerda que para borrar un elemento de una lista se utiliza la palabra clave del.
4. Inserta un nuevo bloque lapislázuli al principio de la lista ➍. Utiliza la función insert() con la variable barBlock para insertar un nuevo bloque lapislázuli en la posición 0 del índice.

Se incluyen comentarios en el código para ayudarte a encontrar dónde tienes que realizar estas tareas.

**OBJETIVO EXTRA: ARRIBA Y ABAJO DE NUEVO**

Actualmente, la barra de progreso de *progressBar.* py sólo cuenta hacia arriba y se detiene cuando está llena. ¿Puedes averiguar cómo hacer que la barra de progreso cuente hacia abajo en sentido contrario?

### **Tratar las cadenas como listas**

Las cadenas pueden tratarse como listas, porque una cadena también es una *secuencia* de datos. Puedes acceder a caracteres individuales de una cadena utilizando su índice; sin embargo, no puedes cambiar los caracteres de cada posición del índice utilizando las funciones añadir o insertar, porque las cadenas son *inmutables*. Esto significa que no se pueden modificar.

El siguiente código imprimirá la segunda letra de la cadena "Uva":

>>> flavor = "Grape"  
>>> print(flavor[1])  
r

Esto demuestra que puedes acceder a partes de una cadena como si fueran elementos de una lista. Por ejemplo, podrías acceder a las primeras letras del nombre y apellidos de alguien para imprimir sus iniciales:

>>> firstName = "Lyra"  
>>> lastName = "Jones"  
>>> initials = firstName[0] + " " + lastName[0]  
>>> print(initials)  
L J

La nueva cadena "L J " que obtienes accediendo a partes de una cadena mediante posiciones de índice se denomina *subcadena*. Ten en cuenta que el índice de una cadena también cuenta desde cero.

### **Tuplas**

*Las tuplas* son un tipo de lista inmutable. Pero, al igual que otras listas, son una secuencia de elementos de cualquier tipo variable. Las tuplas utilizan paréntesis en lugar de corchetes, y usan comas para separar los elementos.

Por ejemplo, supongamos que el único atleta olímpico de una nación, procedente de un programa de entrenamiento con escasa financiación, registra una serie de distancias para sus saltos de longitud en metros:

>>> distance = (5.17, 5.20, 4.56, 53.64, 9.58, 6.41, 2.20)

Si el atleta saltara sólo una vez, también podrías crear una tupla con un único valor. Para escribir una tupla con un único valor, debes incluir una coma:

>>> distance = (5.17,)

Cuando defines una tupla, los paréntesis son opcionales, así que puedes definir una tupla simplemente colocando comas entre los valores, de esta forma:

>>> distance = 5.17, 5.20, 4.56, 53.64, 9.58, 6.41, 2.20

Para acceder a los valores de las tuplas, utiliza la notación de corchetes que utilizas con las listas normales. Asignemos el valor del índice 1 de la tupla distancia a la variable salto:

>>> jump = distance[1]  
>>> print(jump)  
5.20

La principal diferencia entre las listas y las tuplas es que las tuplas son inmutables: no puedes cambiar su contenido. No puedes añadir elementos al final de la tupla, insertar elementos, borrar elementos ni actualizar ningún valor. Utiliza tuplas en lugar de listas cuando tu programa no necesite cambiar los valores de los elementos de la tupla.

#### **Establecer variables con tuplas**

Una característica útil de las tuplas es que puedes utilizarlas para establecer más de una variable al mismo tiempo. Esto ahorra espacio y puede mantener agrupadas las variables relacionadas.

Normalmente, te referirías a una tupla como lo harías a una lista, utilizando un único nombre de variable:

measurements = 6, 30

Sin embargo, supongamos que queremos almacenar los valores en dos variables en lugar de en una. La sintaxis para hacerlo no es compleja. Separas los nombres de las variables con comas, utilizas un signo igual y escribes las tuplas al otro lado del signo igual. Cada valor de tupla se asignará a la variable en la posición correspondiente. Echemos un vistazo.

En este ejemplo, dos variables, anchura y altura, se establecen en los valores 6 y 30, respectivamente:

width, height = 6, 30

Ahora tenemos dos variables. Una se llama anchura y tiene el valor 6, y la otra se llama altura y tiene el valor 30. ¡Y lo hemos hecho utilizando una sola línea de código!

#### **Misión nº 49: Deslizamiento**

Establecer variables con tuplas es una forma rápida y sencilla de ahorrar espacio en tus programas. También es útil para establecer variables relacionadas entre sí en un mismo lugar de tu programa. Por ejemplo, a lo largo del libro has utilizado código como éste para establecer los valores de las variables x, y y z:

x = 10  
y = 11  
z = 12

En cambio, puedes utilizar una tupla para establecer todos estos valores en una sola línea:

x, y, z = 10, 11, 12

A continuación, pondrás en práctica tus nuevas habilidades para escribir código. Tu misión es crear un programa que mueva al jugador aleatoriamente por el mundo del juego en pequeños pasos, haciendo que parezca que patinas sobre hielo. He empezado el programa por ti en el Listado [9-3](ch09.xhtml#ch9ex3); faltan algunas partes y tienes que completarlas.

*sliding.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import random  
import time  
  
➊ # Get the player's position  
  
➋ # Set the x, y, and z variables on the same line using a tuple  
  
while True:  
➌ x += random.uniform(-0.2, 0.2)  
# Change the z variable by a random float  
➍ z +=  
y = mc.getHeight(x, z)  
  
mc.player.setPos(x, y, z)  
time.sleep(0.1)

*Listado 9-3: Inicio del código para hacer que el jugador se deslice por el mapa*

Copia [el Listado 9-3](ch09.xhtml#ch9ex3) en un nuevo archivo y guárdalo como *deslizamiento.py* en tu carpeta *de listas*. Para terminar el programa, necesitas obtener la posición inicial del jugador ➊ y establecer los valores de las variables x, y, z ➋. Utiliza una tupla para fijar estos valores. Este programa también utiliza la función uniforme () ➌, que es como la función randint() (ver "[Jugar con números aleatorios](ch03.xhtml#ch03lev2sec12)" en [la página 66](ch03.xhtml#page_66)) pero devuelve un valor flotante aleatorio en lugar de un valor entero. Utiliza la función uniform() para cambiar el valor de la variable z en el bucle ➍. Esto ya se ha hecho para la variable x ➌.

[La Figura 9-3](ch09.xhtml#ch9fig3) muestra a mi jugador deslizándose lentamente por mi juego.



*Figura 9-3: Deslizándose lentamente hacia atrás por mi jardín*

**OBJETIVO EXTRA: DESLIZAR BLOQUES**

El programa *deslizar.* py hace que el jugador se deslice aleatoriamente por el juego. ¿Puedes averiguar cómo cambiar el programa para que haga deslizar un bloque?

#### **Devolver una tupla**

Algunas de las funciones incorporadas en Python devuelven una tupla. Cuando defines tus propias funciones, también pueden devolver el resultado como una tupla. Para ello, coloca una tupla después de la palabra clave return. Por ejemplo, creemos una función para convertir una fecha en una tupla. Damos la fecha como argumento de cadena, y la función devolverá el año, el mes y el día en una tupla. Éste es el código:

def getDateTuple(dateString):  
year = int(dateString[0:4])  
month = int(dateString[5:7])  
day = int(dateString[8:10])  
return year, month, day

Cuando llamamos a la función y le damos una fecha como cadena, nos devuelve una tupla que contiene el año, el mes y el día en ese orden:

>>> getDateTuple("1997-09-27")  
(1997, 9, 27)

Cuando llamamos a la función, podemos almacenar la tupla devuelta como queramos. Este código almacena cada valor en una variable independiente:

year, month, day = getDateTuple("1997-09-27")

Ahora podemos convertir rápidamente cadenas de fechas en variables individuales. En mi trabajo como desarrollador de software, utilizo código muy similar a éste todo el tiempo.

### **Otras funciones útiles de las listas**

Puedes hacer muchas otras tareas con listas. En esta sección se explica cómo averiguar la longitud de una lista, cómo elegir aleatoriamente un elemento de una lista y cómo utilizar una sentencia if para comprobar si un valor está en una lista.

#### **Longitud de una lista**

La función len( ) es una forma rápida de averiguar la longitud de cualquier lista en Python. La función devuelve el número de elementos de una lista cuando se utiliza una lista como argumento. Veámosla en acción:

>>> noodleSoup = ["water", "soy sauce", "spring onions", "noodles", "beef",  
"vegetables"]  
>>> print(len(noodleSoup))  
6

Aunque Python empieza a contar los índices en cero, cuenta cuántos elementos hay en una lista en números de conteo regulares. El índice más alto de esta lista es 5, ¡pero Python sabe que hay 6 elementos en total!

#### **Misión nº 50: Golpes de bloque**

La API Python de Minecraft tiene una práctica función que devuelve una lista de lugares que has golpeado con tu espada. Puedes utilizar los elementos de la lista para obtener las coordenadas de los bloques que has golpeado. Verás lo útil que es esto en programas más adelante en este capítulo y más adelante en el libro.

También puedes hacer un juego corto y divertido que cuente el número de bloques que puedes golpear en un minuto. En esta misión, harás precisamente eso. Es un juego muy divertido: ¡juega contra un amigo e intentad superar las puntuaciones de los demás! También puedes ampliarlo, por ejemplo, manteniendo una puntuación alta.

[La Figura 9-4](ch09.xhtml#ch9fig4) muestra el programa en acción.



*Figura 9-4: En 60 segundos he conseguido 197 bloques.*

No se necesita mucho código para hacer este juego. He aquí un resumen de la estructura del código:

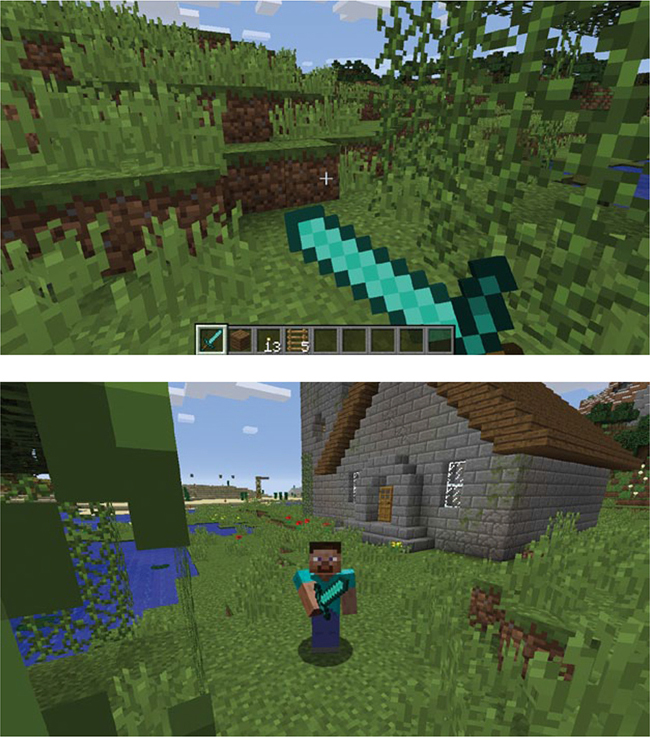
1. Conéctate al juego Minecraft.
2. Espera 60 segundos.
3. Obtener la lista de bloques golpeados.
4. Mostrar la longitud de la lista de éxitos de bloque al chat.

El siguiente código muestra la única parte que no has visto hasta ahora, que es el código que obtiene la lista de éxitos de bloque del juego:

blockHits = mc.events.pollBlockHits()

Este código utiliza la función pollBlockHits() para devolver una lista de golpes de bloque y almacena esa lista en una variable llamada blockHits. La variable blockHits actuará como cualquier otro tipo de lista, por lo que puedes acceder a los datos desde las posiciones de índice y obtener la longitud de la lista.

Cuando juegues a este juego, tendrás que hacer clic con el botón derecho en los bloques para llevar la cuenta de ellos. La razón es que la función pollBlockHits( ) registra todos los bloques que *pulsas* con el botón derecho del ratón. En la versión para PC de Minecraft, al hacer clic con el botón derecho con la espada parece más que te estés defendiendo que que estés golpeando algo, pero sigue registrando los bloques en los que has hecho clic. [La Figura 9-5](ch09.xhtml#ch9fig5) muestra este aspecto. Asegúrate de que sólo haces clic derecho con la espada: los clics izquierdos con la espada no se registrarán, ¡y tampoco los clics derechos con cualquier otra cosa que tengas en la mano! Pero puedes utilizar cualquier tipo de espada, incluidas las de hierro, oro y diamante.



*Figura 9-5: Cuando hago clic con el botón derecho, el jugador sujeta la espada así.*

Cuando imprimas la salida de la lista, debe tener un aspecto similar a éste, aunque los valores cambiarán cada vez dependiendo de dónde golpees:

[BlockEvent(BlockEvent.HIT, 76, -2, 144, 1, 452),  
BlockEvent(BlockEvent.HIT, 79, -2, 145, 1, 452),  
BlockEvent(BlockEvent.HIT, 80, -3, 147, 1, 452),  
BlockEvent(BlockEvent.HIT, 76, -3, 149, 1, 452)]

Esta salida de lista almacena los detalles de cuatro golpes en bloque. Cada elemento contiene las coordenadas del golpe. Aprenderás a acceder a estas coordenadas en la Misión nº 55[(página 200](ch10.xhtml#page_200)).

Para ayudarte a empezar con el programa, he escrito la estructura básica en el Listado [9-4](ch09.xhtml#ch9ex4).

*swordHits.py*

# Connect to the Minecraft game  
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
  
# Wait 60 seconds  
time.sleep(60)  
  
# Get the list of block hits  
➊ blockHits =  
  
# Display the length of the block hits list to chat  
➋ blockHitsLength =  
mc.postToChat("Your score is " + str(blockHitsLength))

*Listado 9-4: Inicio del juego de golpes de espada*

Para completar este programa, abre IDLE, crea un nuevo archivo y copia [el Listado 9-4](ch09.xhtml#ch9ex4) en él. Guarda este archivo como *swordHits.py* en la carpeta de *listas*. Establece la variable blockHits utilizando la función pollBlockHits() ➊ y establece la variable blockHitsLength obteniendo la longitud de la variable blockHits ➋.

#### **Elegir un elemento al azar**

A estas alturas ya te habrás dado cuenta de que me gusta mucho utilizar elementos generados aleatoriamente en mis programas. La aleatoriedad hace que un programa se comporte de forma impredecible cada vez que lo ejecutas.

Cuando utilices listas, querrás acceder a elementos aleatorios de la lista de vez en cuando. Por ejemplo, puede que quieras elegir un bloque al azar de una lista de bloques.

La función choice() del módulo random es la función a la que se puede recurrir para elegir un elemento de la lista al azar. La función toma un argumento, la lista que quieres utilizar, y devuelve un elemento aleatorio de dentro de la lista.

En [el Listado 9-5](ch09.xhtml#ch9ex5), la lista colores contiene los nombres de varios colores. Elige uno al azar utilizando la función elección() y luego imprímelo:

import random  
colors = ["red", "green", "blue", "yellow", "orange", "purple"]  
print(random.choice(colors))

*Listado 9-5: Impresión de un color aleatorio de una lista de colores*

Cuando ejecutes el código, el programa imprimirá un elemento de la lista al azar.

#### **Misión nº 51: Bloque aleatorio**

En Minecraft, seleccionar un ID de bloque aleatorio a partir de un rango de números puede causar problemas en el programa porque algunos ID de bloque no tienen bloques correspondientes. Una solución es utilizar una lista de bloques válidos para seleccionarlos al azar. Las listas te permiten crear un número limitado de elementos y luego seleccionar uno al azar utilizando la función choice().

Tu misión es crear una lista de IDs de bloques, seleccionar un bloque al azar de esa lista y, a continuación, colocar el bloque en la posición del jugador. Puedes utilizar [el Listado 9-5](ch09.xhtml#ch9ex5) como punto de partida.

Primero, crea una lista de IDs de bloques. Segundo, utiliza la función random.choice () para seleccionar un bloque de la lista. Tercero, utiliza la función setBlock () para colocar el bloque aleatorio en el juego Minecraft.

Guarda el programa como *randomBlock.py* en la carpeta de *listas*.

Incluye tantos bloques como quieras en tu lista. Para mi lista elegí cinco bloques, incluyendo melón, diamante y oro. Puedes ver el resultado de ejecutar el programa en la [Figura 9-6](ch09.xhtml#ch9fig6).



*Figura 9-6: El programa seleccionó al azar un bloque de oro.*

#### **Copiar una lista**

Copiar listas es bastante complicado en la mayoría de los lenguajes de programación. En realidad, las variables de lista no contienen valores, sino una referencia a una dirección de la memoria de tu ordenador que tiene más referencias a los valores contenidos en la lista. Aunque tu ordenador se encarga de esta función entre bastidores, merece la pena que entiendas cómo funciona, ¡porque te convertirá en un programador más inteligente! Puedes ver la dirección de memoria de una lista utilizando la función id():

>>> cake = ["Eggs",  
"Butter",  
"Sugar",  
"Milk",  
"Flour"]  
>>> print(id(cake))

Por ejemplo, la salida de este código en mi ordenador fue 3067456428. El valor 3067456428 es la posición de memoria donde se almacena pastel. Cuando ejecutes este código en tu ordenador, probablemente obtendrás un número diferente porque está almacenado en un lugar distinto de la memoria de tu ordenador.

No es necesario que entiendas completamente este comportamiento, pero sí que sepas que tiene consecuencias cuando quieres copiar una lista en otra variable. En lugar de copiarse los valores de la lista, como cabría esperar, se copia la posición de la lista en la memoria de la nueva variable. Esto significa que cuando cambies un valor de cualquiera de las dos listas, afectará a la otra.

Por ejemplo, el siguiente programa crea una lista llamada tarta y luego establece el valor de tartaChocolate para que sea el mismo que tarta. A continuación, se añade un elemento, "Chocolate", a la lista tartaChocolate:

>>> cake = ["Eggs",  
"Butter",  
"Sugar",  
"Milk",  
"Flour"]  
  
>>> # Store the list in a second variable  
>>> chocolateCake = cake  
>>> chocolateCake.append("Chocolate")

Por desgracia, "Chocolate " también se añade a la lista tarta, aunque no querías que lo hiciera. Puedes ver este error cuando se imprimen las listas:

>>> print(cake)  
['Eggs', 'Butter', 'Sugar', 'Milk', 'Flour', 'Chocolate']  
>>> print(chocolateCake)  
['Eggs', 'Butter', 'Sugar', 'Milk', 'Flour', 'Chocolate']

Este problema se produce porque las variables almacenan la posición de memoria de la lista, no los elementos de la lista.

Una forma sencilla de superar este problema es utilizar una *rebanada* de lista. Cuando rebanas un alimento con un cuchillo, lo estás cortando en diferentes partes. Un corte de lista en Python es similar. Cuando cortas una lista, tomas un trozo de la lista. Puedes utilizar una rebanada de lista para tomar sólo determinados elementos de una lista, pero en este caso, utilizarás una rebanada de lista para copiar todos los elementos de una lista. Para copiar la lista Tarta en la variable TartaDeChocolate, utiliza este código:

>>> chocolateCake = cake[:]

La variable tartaChocolate contendrá ahora los valores de la lista de tartas, pero con una dirección de memoria diferente.

El código de los ingredientes de la tarta se puede corregir utilizando la rebanada de lista:

>>> cake = ["Eggs",  
"Butter",  
"Sugar",  
"Milk",  
"Flour"]  
  
>>> # Store the list in a second variable  
➊ >>> chocolateCake = cake[:]  
>>> chocolateCake.append("Chocolate")

Puedes ver que los elementos de tarta se han copiado a tartaChocolate utilizando [:] en ➊.

Este es el resultado:

>>> print(cake)  
['Eggs', 'Butter', 'Sugar', 'Milk', 'Flour']  
>>> print(chocolateCake)  
['Eggs', 'Butter', 'Sugar', 'Milk', 'Flour', 'Chocolate']

Observa que los valores de ambas listas son ahora diferentes: sólo tartaChocolate contiene el valor "Chocolate".

#### **Elementos y sentencias if**

Para saber si un valor está en una lista, puedes utilizar el operador in. El operador in va entre un valor y la lista que quieres comprobar. Si el valor está en la lista, la expresión se evaluará como Verdadero; si el valor no está en la lista, la expresión se evaluará como Falso.

El siguiente ejemplo comprueba si el valor "Huevos" está en la lista de pasteles:

>>> cake = ["Eggs", "Butter", "Sugar", "Milk", "Flour"]  
>>> print("Eggs" in cake)

Se imprimirá el valor Verdadero, porque "Huevos " está en la lista.

Por supuesto, puedes utilizar el operador in como parte de una condición de la sentencia if. El código siguiente amplía y adapta este ejemplo para utilizar una sentencia if en lugar de imprimir el valor booleano. Comprueba si "Jamón" está en la lista de pasteles e imprime diferentes mensajes dependiendo de si está o no en la lista:

>>> cake = ["Eggs", "Butter", "Sugar", "Milk", "Flour"]  
>>> if "Ham" in cake:  
>>> print("That cake sounds disgusting.")  
>>> else:  
>>> print("Good. Ham in a cake is a terrible mistake.")

Puedes combinar el operador not con el operador in para producir el efecto contrario. En lugar de devolver Verdadero cuando un elemento está en una lista, el código devolverá Falso y viceversa. Así es como queda (observa que también se han intercambiado los cuerpos de las sentencias if y else ):

>>> cake = ["Eggs", "Butter", "Sugar", "Milk", "Flour"]  
>>> if "Ham" not in cake:  
>>> print("Good. Ham in a cake is a terrible mistake.")  
>>> else:  
>>> print("That cake sounds disgusting")

Puedes utilizar cualquiera de las dos técnicas en tus programas. Sólo tienes que elegir la que creas que tiene más sentido.

#### **Misión nº 52: Espada de visión nocturna**

¿Te olvidas de llevar suficientes antorchas cuando exploras cuevas en Minecraft? Yo lo hago siempre. A veces me olvido de llevar antorchas, y estoy demasiado dentro de la cueva para volver atrás. Así que voy tanteando en la oscuridad, sin estar muy seguro de si encuentro algo útil. Pero con tus conocimientos de Python, puedes hacer un programa que te ayude a encontrar diamantes con tu espada.

Escribamos un programa básico que utilice la función pollBlockHits() para comprobar si alguno de los bloques que has golpeado es mineral de diamante. Esto es útil para explorar cuevas sin luz o para jugar a "encontrar el mineral de diamante" en la oscuridad. El código está en [el Listado 9-6](ch09.xhtml#ch9ex6). Cópialo en un archivo nuevo y guárdalo como *nightVisionSword.py* en la carpeta de *listas*.

*nightVisionSword.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
  
blocks = []  
  
while True:  
hits = mc.events.pollBlockHits()  
if len(hits) > 0:  
hit = hits[0]  
➊ hitX, hitY, hitZ = hit.pos.x, hit.pos.y, hit.pos.z  
block = mc.getBlock(hitX, hitY, hitZ)  
blocks.append(block)  
  
➋ # Add the if statement here  
  
time.sleep(0.2)

*Listado 9-6: Este programa te ayudará a encontrar mineral de diamante en la oscuridad.*

Observa cómo se utilizan hit .pos.x, hit.pos.y y hit.pos.z ➊. Cada golpe almacena las coordenadas del bloque sobre el que se hizo clic mediante una tupla. Puedes acceder a estas coordenadas utilizando la notación de puntos. En este ejemplo, el nombre de la variable hit se utiliza para nombrar la lista que contiene cada bloque hit, por lo que accedo a las coordenadas utilizando hit.pos.x, hit.pos.y y hit.pos.z.

El código está casi completo. Sólo queda comprobar si has encontrado algún diamante. Añade una sentencia if ➋ para comprobar si el mineral de diam ante (bloque ID 56) está en la lista de bloques y envía un mensaje al chat diciendo "¡Has encontrado mineral de diamante!" si es así. Añade también una sentencia break dentro de la sentencia if para que el bucle deje de repetirse cuando encuentres el mineral.

[La Figura 9-7](ch09.xhtml#ch9fig7) muestra el programa en acción.



*Figura 9-7: Está oscuro, pero he encontrado mineral de diamante. ¡Qué bien!*

Si no eres tan olvidadizo como yo y te acuerdas de llevar antorchas a las cuevas, también puedes utilizar este código como juego. Crea una habitación subterránea sin luz y coloca un único mineral de diamante en algún lugar de la pared. Ejecuta el programa y comprueba cuánto tardas en encontrar el mineral de diamante en la oscuridad. ¡Recuerda hacer clic con el botón derecho del ratón con una espada! Sólo así la función pollBlockHits( ) podrá registrar los bloques que golpeas.

**OBJETIVO EXTRA: DESAFÍO DEL DIAMANTE**

Sería genial convertir el programa *nightVisionSword.* py en un minijuego completo. ¿Puedes generar automáticamente una habitación con un único bloque de diamante colocado al azar, poner al jugador en esa habitación y luego cronometrar cuánto tarda en encontrar el bloque en la oscuridad?

### **Diccionarios**

Los diccionarios son un tipo de lista que utiliza un enfoque diferente. En lugar de utilizar un índice para identificar los elementos, los diccionarios identifican los elementos utilizando un conjunto de claves definidas por el programador.

Por ejemplo, este diccionario raceTimes almacena los nombres de las personas que corrieron en una carrera y sus tiempos de carrera:

raceTimes = {'Katy': 26,  
'Alex': 30,  
'Richard': 19}

La clave identifica de forma única cada valor del diccionario. En este ejemplo, la clave es el nombre de la persona. La clave "Katy " tiene un valor asociado de 26.

Al igual que las listas, los diccionarios son mutables; su contenido puede modificarse.

#### **Definir un diccionario**

Para definir un diccionario, utiliza un par de llaves alrededor de un conjunto de pares clave-valor. Por ejemplo, puedes utilizar un diccionario para describir a una persona. Puedes utilizar claves como "nombre" y "animal favorito " para almacenar información sobre la persona, de esta forma

person = {'name': 'David',  
'age': 42,  
'favoriteAnimal': 'Snake',  
'favoritePlace': 'Inside a cardboard box'}

En este ejemplo, cada clave es una cadena. Cada clave se empareja con un valor mediante dos puntos. Por ejemplo, "edad" es una clave y 42 es su valor correspondiente. A continuación, los elementos del diccionario se separan mediante comas.

Te habrás dado cuenta de que utilizar diccionarios facilita al programador la comprensión de lo que representa cada elemento de la lista; por ejemplo, es fácil entender que la clave "nombre " almacena un nombre, no un número u otra información aleatoria.

También puedes utilizar enteros y flotantes como claves del diccionario. Utilizar flotantes o enteros en los diccionarios es muy útil cuando las claves que quieres emparejar con valores no siguen una secuencia estricta.

El siguiente ejemplo crea un diccionario de horas de tren. La hora del tren (que es un flotante) se almacena como clave, y el destino del tren se almacena como valor:

trainTimes = {1.00: 'Castle Town',  
2.30: 'Sheep Farm',  
3.15: 'Lake City',  
3.45: 'Castle Town',  
3.55: 'Storage Land'  
}

Como los diccionarios pueden almacenar dos datos que van juntos como un par, son ideales para una situación como ésta. Si utilizara una lista de destinos de trenes en lugar de un diccionario, no podría emparejar los tiempos con los destinos. Sólo podría utilizar las posiciones de índice de la lista, que serían 0, 1, 2, 3, 4, etc., en lugar de las horas.

#### **Acceder a los elementos de los diccionarios**

Para acceder al valor de un elemento de un diccionario, utiliza corchetes y una clave en lugar de un índice. La clave suele ser una cadena o un número entero. Cuando crees un diccionario que utilice cadenas como claves, asegúrate de entrecomillarlas.

Por ejemplo, para acceder al valor de la clave "nombre " en el diccionario de personas creado anteriormente, utilizarías esta sintaxis

person = {'name': 'David',  
'age': 42,  
'favoriteAnimal': 'Snake',  
'favoritePlace': 'Inside a cardboard box'}  
  
agentName = person['name']

La variable NombreAgente contendrá el valor "David " porque accede al valor de la clave " nombre ". Del mismo modo, si quisieras acceder a la edad del agente, utilizarías la clave ' edad ':

agentAge = person['age']

Esto almacenaría el valor 42 en la variable agentAge.

En el ejemplo de trainTimes, puedes acceder a los valores del diccionario (los destinos) utilizando sus valores clave (las horas de los trenes), que son flotantes:

trainTimes = {1.00: 'Castle Town',  
2.30: 'Sheep Farm',  
3.15: 'Lake City',  
3.45: 'Castle Town',  
3.55: 'Storage Land'  
}  
  
myTrain = trainTimes[3.15]

Al acceder a la clave 3.15 del diccionario trainTimes, la variable myTrain se establece en 'Lake City'.

#### **Misión nº 53: Guía turística**

Cuando utilizas diccionarios, puedes almacenar cualquier tipo de datos como valor, incluso listas y tuplas. Por ejemplo, puedes almacenar una tupla que contenga los valores de x, y y z. He aquí un ejemplo de código que hace exactamente eso:

places = {'Living room': (76, 1, -61), 'Bedroom': (61, 9, -61)}

El diccionario de lugares almacena dos elementos. La clave del diccionario es el nombre de una ubicación en mi juego Minecraft (como mi salón o mi dormitorio), y el valor es una tupla de las coordenadas. Si quisiera acceder a las coordenadas de mi salón, utilizaría el siguiente código:

location = places['Living room']  
x, y, z = location[0], location[1], location[2]

Tu misión es crear un programa que utilice un diccionario para almacenar las ubicaciones de los distintos lugares de tu juego Minecraft, de forma que puedas teletransportarte a ellos por su nombre. Incluye tantas localizaciones en el diccionario como quieras. Para teletransportarte a esos lugares, tienes que acceder a la tupla de coordenadas almacenada en el diccionario y luego establecer x, y y z a los valores almacenados en la tupla. Los comentarios del código muestran dónde hacerlo.

Copia [el Listado 9-7](ch09.xhtml#ch9ex7) en el editor de texto IDLE y guárdalo en la carpeta de *listas* como *guiaviajes.py*.

*sightseeingGuide.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
# Add locations to the dictionary  
places = {}  
  
choice = ""  
while choice != "exit":  
➊ choice = input("Enter a location ('exit' to close): ")  
➋ if choice in places:  
# Store the dictionary item's value using its key (choice)  
location =  
# Store the values stored in the tuple in the x, y, and z variables  
x, y, z =  
mc.player.setTilePos(x, y, z)

*Listado 9-7: Código para teletransportarse a distintos lugares*

He incluido una sentencia que te pide que introduzcas el nombre del lugar al que quieres ir. Esta información se almacena en la variable de elección ➊. A continuación, el programa utiliza una sentencia if para comprobar si el valor de elección está en el diccionario ➋. La última línea utiliza las variables x, y y z para teletransportar al jugador a la posición almacenada en el diccionario.

Cuando se ejecute el programa, introduce el nombre del lugar al que quieres ir. [La Figura 9-8](ch09.xhtml#ch9fig8) muestra mi versión del programa teletransportándome a distintos lugares de mi juego.



*Figura 9-8: Me teletransporté a mi salón (arriba) y a mi dormitorio (abajo).*

#### **Cambiar o añadir un elemento en un diccionario**

No cuesta mucho trabajo cambiar el valor de un elemento de un diccionario. Utiliza corchetes con una tecla para acceder al elemento y establécelo como harías con una variable normal (con un signo igual). También puedes añadir un nuevo elemento utilizando este método.

Cambiemos el valor del elemento edad del diccionario persona de 42 a 43:

person['age'] = 43

Añadamos también un nuevo elemento llamado ubicación con el valor "USS Discovery":

person['location'] = 'USS Discovery'

Después de ejecutar este código, el diccionario tendrá una nueva clave llamada ubicación con el valor "USS Discovery".

#### **Borrar elementos de los diccionarios**

A veces querrás borrar un elemento de un diccionario. Al igual que con una lista, utiliza la palabra clave del para hacerlo. Por ejemplo, para borrar el ítem AnimalFavorito del diccionario Persona, harías lo siguiente:

del person['favoriteAnimal']

Como puedes ver, funciona igual que borrar elementos de una lista.

#### **Misión nº 54: Bloquear la puntuación de los aciertos**

En la Misión nº 50[(página 184](ch09.xhtml#page_184)), escribiste un programa que cuenta el número de veces que el jugador golpea un bloque con su espada en 60 segundos. Por muy divertido que sea el programa, sería aún más genial si pudieras registrar las puntuaciones de todos los que jugaron.

Para añadir un marcador al juego, utilizarás un diccionario. El diccionario almacenará el nombre del jugador y su puntuación, que luego podrá mostrarse junto a las puntuaciones de todos los demás.

Para empezar, abre *swordHits.py* y guárdalo como *swordHitsScore.py* en la carpeta de *listas*. Actualiza el código para que coincida con [el Listado 9-8](ch09.xhtml#ch9ex8), donde he hecho algunos cambios en el programa para que se repita, pregunte al jugador su nombre y luego imprima todas las puntuaciones. (También he incluido las soluciones al código que falta en *swordHits.py*.) Las secciones antiguas están en gris. (Recuerda sangrar todo dentro del bucle).

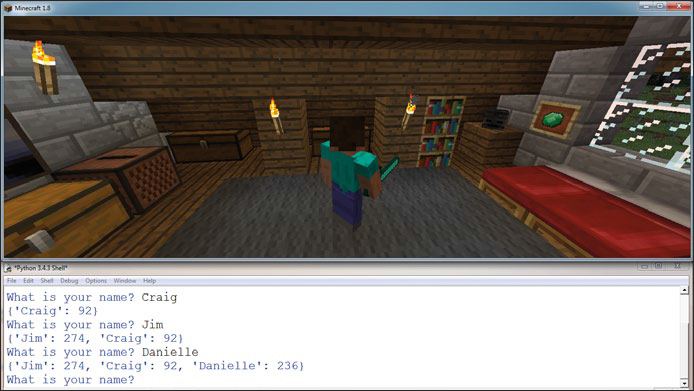
*swordHitsScore.py*

# Connect to the Minecraft game  
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
  
name = ""  
scoreboard = {}  
  
while True:  
# Get the player's name  
name = input("What is your name? ")  
# Break loop if name is exit  
if name == "exit":  
break  
mc.postToChat("Go!")  
  
# Wait 60 seconds  
time.sleep(60)  
  
# Get the list of block hits  
blockHits = mc.events.pollBlockHits()  
  
# Display the length of the block hits list to chat  
blockHitsLength = len(blockHits)  
mc.postToChat("Your score is " + str(blockHitsLength))  
  
➊ # Add the player to the scoreboard  
  
# Display the scoreboard  
print(scoreboard)

*Listado 9-8: Cuando el código esté completo, añadirá un marcador al juego de golpes de bloque.*

Para terminar el programa, necesitas almacenar el nombre y la puntuación de cada jugador que participe en el juego. Hazlo añadiendo un nuevo elemento de diccionario utilizando los datos del código en ➊. El diccionario se llama marcador, y el nombre del jugador se almacena en la variable nombre.

[La Figura 9-9](ch09.xhtml#ch9fig9) muestra la salida de mi marcador.



*Figura 9-9: Mis amigos y yo jugamos una partida, y Jim es el ganador con 274 aciertos de bloque.*

**NOTA**

*Te habrás dado cuenta de que cuando* se *imprime el diccionario del marcador, no es fácil de leer. Aprenderás a solucionarlo en la Misión 59 (*[*página 209*](ch10.xhtml#page_209)*).*

**OBJETIVO EXTRA: MEJOR PUNTUACIÓN**

Actualmente, si alguien juega al juego *swordHitsScore.* py dos o más veces (e introduce el mismo nombre de usuario), el programa sólo registrará su puntuación más reciente. ¿Puedes averiguar cómo utilizar una sentencia if para comprobar si el jugador ya ha introducido una puntuación y almacenar la nueva puntuación sólo si es mayor que la anterior? Aquí tienes el principio del código para ayudarte. Comprueba si el nombre de alguien ya está en el diccionario del marcador:

if name in scoreboard:

### **Lo que has aprendido**

¡Excelente trabajo! En este capítulo has aprendido sobre listas, tuplas y diccionarios. Has visto que pueden almacenar varios valores de datos en una sola variable. Son una forma muy útil de estructurar y almacenar datos en tus programas.

En las misiones, creaste varios programas divertidos que utilizan listas, diccionarios y tuplas. Con listas, creaste una barra de progreso utilizando lapislázuli y cristal. Con las tuplas, aprendiste una forma más rápida de establecer las variables x, y y z. Y los diccionarios te permitieron almacenar las coordenadas de las cosas que has construido y teletransportarte a ellas introduciendo sus nombres.

En [el Capítulo 10](ch10.xhtml#ch10), desarrollarás aún más tus conocimientos sobre listas aprendiendo sobre los bucles for. Crearás algunos programas muy chulos, incluido uno que puedes utilizar para duplicar objetos que hayas construido.