Aprende a programar con Minecraft

### **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:

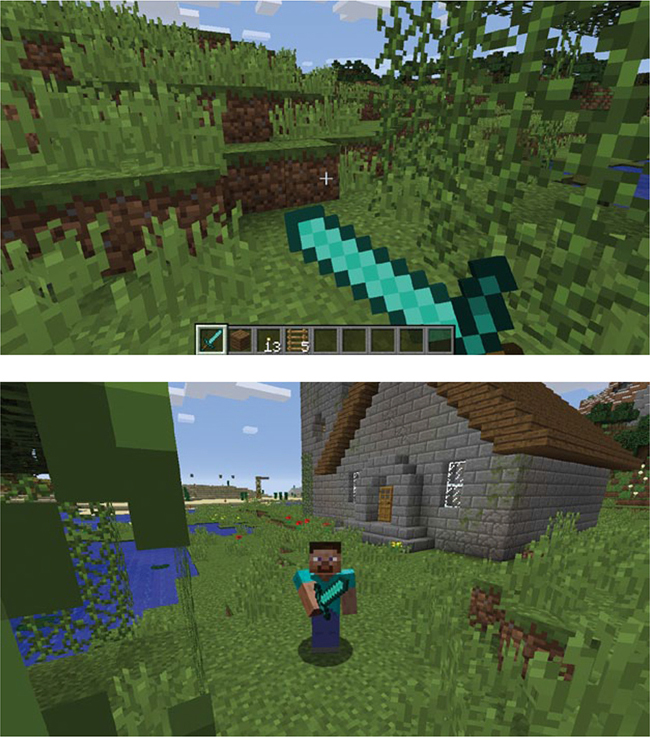
1. Conéctate al juego Minecraft.
2. Espera 60 segundos.
3. Obtener la lista de bloques golpeados.
4. Mostrar la longitud de la lista de éxitos de bloque al chat.

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?

[anterior](ch09_5.html)[Subtema 6 de 8: (Ver todo)](ch09.html)[siguiente](ch09_7.html)