Aprende a programar con Minecraft

### **Uso de sentencias if y bucles while en funciones**

En los [Capítulos 6](ch06.xhtml#ch06) y [7](ch07.xhtml#ch07), aprendiste a poner sentencias if dentro de otras sentencias if y bucles while dentro de otros bucles while. Aprendiste que incluso puedes poner sentencias if dentro de bucles while y viceversa. En esta sección aprenderás a colocar sentencias if y bucles dentro de funciones. Esto hace que tus funciones sean muy flexibles, porque puedes utilizarlas para tomar decisiones y repetir código.

#### **Sentencias if**

Cuando escribes una sentencia if dentro de una función, la sintaxis es idéntica a la de una sentencia if normal. Sólo tienes que acordarte de sangrar la sentencia if cuatro espacios más al principio de cada línea para que Python sepa que forma parte de la función.

El siguiente código toma un número escrito como una cadena y devuelve el número como un entero. Por ejemplo, el argumento "cuatro " devuelve el valor 4:

def wordToNumber(numToConvert):  
""" Converts a number written as a word to an integer """  
if numToConvert == "one":  
numAsInt = 1  
elif numToConvert == "two":  
numAsInt = 2  
elif numToConvert == "three":  
numAsInt = 3  
elif numToConvert == "four":  
numAsInt = 4  
elif numToConver == "five":  
numAsInt = 5  
  
return numAsInt

Veamos otro ejemplo. La siguiente función comprueba si has conocido antes a una persona y utiliza un saludo adecuado en función del resultado:

➊ def chooseGreeting(metBefore):  
""" Chooses a greeting depending on whether you've met someone before.  
metBefore argument should be a Boolean value """  
if metBefore:  
➋ print("Nice to see you again")  
else:  
➌ print("Nice to meet you")  
  
chooseGreeting(True)  
chooseGreeting(False)

La función elegirSaludo( ) toma un argumento booleano, llamado metAntes ➊. La sentencia if que hay dentro de la función imprime una salida basada en el valor del argumento. Si el valor es True, la salida será "Encantado de volver a verte" ➋, y si es False ➌, la salida será "Encantado de conocerte".

#### **Misión nº 44: Ayudante del color de la lana**

Has utilizado los métodos setBlock() y setBlocks() con argumentos para establecer las coordenadas y el tipo de bloque, pero estos métodos también tienen un argumento extra opcional que establecerá el *estado* del bloque.

Cada bloque en Minecraft tiene 16 estados, de 0 a 15. La lana, por ejemplo, tiene un color diferente para cada estado. El TNT (ID del bloque 46) no explotará cuando lo aplastes en su estado por defecto (estado 0), pero es explosivo cuando lo aplastas en el estado 1 del bloque. Aunque cada bloque tiene 16 estados, no todos tienen comportamientos diferentes.

Para establecer el estado de un bloque, proporciona a la función setblock() o setblocks() un argumento extra. El siguiente código crea un bloque rosa de lana:

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
block = 35  
state = 6  
# Creates a single block of pink wool  
mc.setBlock(10, 3, -4, block, state)  
  
# Creates a cuboid of pink wool  
mc.setBlocks(11, 3, -4, 20, 6, -8, block, state)

La lana (ID de bloque 35) tiene muchos usos en Minecraft debido a sus diferentes colores, pero es difícil recordar los diferentes estados de los bloques. Afortunadamente, no necesitas memorizar los diferentes estados de los bloques cuando puedes utilizar un programa que te lo recuerde.

Hagamos un programa que contenga los estados del bloque lana. El programa contendrá una función con un argumento que toma el color que quieras escrito como una cadena. A continuación, la función devuelve el estado del bloque para el color de la lana como un número entero. La función contendrá la mayor parte del código del programa. Sin embargo, añadirás un par de líneas de código para recibir la entrada de un usuario y colocar el bloque en el juego, y utilizarás tu nueva y elegante función para establecer el color.

En primer lugar, tendrás que averiguar los estados de bloque para los distintos colores de lana. Puedes encontrarlos en la "[Hoja de trucos de ID de bloque](app03.xhtml#app03)" en [la página 299](app03.xhtml#page_299). Aquí tienes algo de código para empezar (el rosa es el estado de bloque 6):

*woolColors.py*

def getWoolState(color):  
""" Takes a color as a string and returns the wool block state for  
that color """  
➊ if color == "pink":  
blockState = 6  
elif # Add elif statements for the other colors  
# Return the blockState here  
  
➋ colorString = input("Enter a block color: ")  
state = getWoolState(colorString)  
  
➌ pos = mc.player.getTilePos()  
mc.setBlock(pos.x, pos,y, pos.z, 35, state)

De momento, el programa sólo tiene el principio de la función getWoolState(). Sólo tiene una sentencia if para el color rosa ➊. También se incluye código al final del programa para tomar la entrada del usuario para el color del bloque ➋ y código para colocar el bloque de lana en la posición del jugador ➌.

Añade a la función getWoolState( ) declaraciones elif para otros colores de lana y sus correspondientes estados de bloque. El programa debe tomar un argumento para el color del bloque y devolver el valor entero del estado del bloque. Por ejemplo, si proporcionas el argumento "rosa ", devolverá el valor 6. También tendrás que añadir una sentencia return al programa. Utiliza los comentarios como guía.

Guarda el archivo como *lanaColores.py* en la carpeta de *funciones*.

Si quieres que el programa sea más fácil de usar, puedes enviar un mensaje al chat si el argumento no es un color válido. [La Figura 8-4](ch08.xhtml#ch8fig4) muestra la entrada en el intérprete de comandos Python y el bloque de lana colocado en el juego.



*Figura 8-4: Ahora puedo crear un bloque de lana de cualquier color introduciendo el nombre del color que quiero.*

#### **Bucles while**

Al igual que las sentencias if, los bucles se pueden escribir dentro de funciones. La sintaxis de un bucle dentro de una función es la misma que la de un bucle normal. Sólo tienes que recordar que el bucle debe tener una sangría adicional de cuatro espacios en cada línea para indicar que pertenece a la función.

En el siguiente ejemplo, el bucle while dentro de la función imprimirá el argumento toPrint. El número de veces que se repite el bucle viene determinado por el argumento repeticiones.

def printMultiple(toPrint, repeats):  
""" Prints a string a number of times determined by the repeats variable """  
count = 0  
while count < repeats:  
print(toPrint)  
count += 1

También puedes utilizar sentencias return y bucles while en la misma función. En la mayoría de los casos, querrás que la sentencia de retorno esté fuera del bucle. (Si utilizas la sentencia return dentro de un bucle, romperá el bucle y terminará la función). Veamos un ejemplo:

def doubleUntilHundred(numberToDbl):  
""" Doubles a number until it is greater than 100. Returns the number of  
times the number was doubled """  
count = 0  
while numToDbl < 100:  
numberToDbl = numberToDbl \* 2  
count += 1  
➊ return count  
  
print(doubleUntilHundred(2))

Este programa duplica un número hasta que es mayor que 100. Luego devuelve el número de veces que se repitió el bucle ➊.

También puedes poner llamadas a funciones dentro de bucles, como hiciste en capítulos anteriores.

#### **Misión nº 45: Bloques, por todas partes**

Al utilizar bucles dentro de funciones, puedes utilizar un argumento para determinar el número de veces que se repite un bucle. Utilizando la función setBlock( ), también puedes determinar el tipo de bloque que se colocará dentro del bucle.

**ADVERTENCIA**

*El programa de esta misión podría ser destructivo, por lo que tal vez quieras probarlo en un mundo nuevo para preservar tus preciadas creaciones.*

En esta misión, crearás una función que coloca bloques aleatoriamente por el mapa. El número de bloques que coloca y el tipo de bloque que coloca vienen determinados por los argumentos de la función.

[El listado 8-4](ch08.xhtml#ch8ex4) genera un melón en un lugar aleatorio del mapa.

*blocksEverywhere.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
import random  
  
def randomBlockLocations(blockType, repeats):  
➊ count = 0  
➋ # Add the loop here  
x = random.randint(-127, 127)  
z = random.randint(-127, 127)  
➌ y = mc.getHeight(x, z)  
mc.setBlock(x, y, z, blockType)  
count += 1

*Listado 8-4: Al ser llamada, esta función colocará un bloque aleatoriamente en el juego.*

Copia [el Listado 8-4](ch08.xhtml#ch8ex4) en un nuevo archivo en IDLE y guárdalo como *blocksEverywhere.py* en la carpeta *functions*. En ➋, añade un bucle while dentro de la función para que el código se repita. La variable cuenta ➊ te facilita saber cuántas veces se ha repetido el bucle. Compara el argumento repeticiones con la variable cuenta en la condición del bucle para establecer cuántas veces debe repetirse el bucle. Pon sangría a todas las líneas dentro de la función después de ➋ para que también estén dentro del bucle. La función getHeight( ) se encarga de que el bloque se sitúe por encima del suelo ➌.

Por último, añade tres llamadas a funciones para crear bloques. La primera función debe crear 10 bloques, la segunda debe crear 37 bloques y la tercera debe crear 102 bloques. Elige los tipos de bloque que quieras.

Guarda el programa y ejecútalo. El programa debería crear bloques aleatoriamente alrededor del mapa. [La Figura 8-5](ch08.xhtml#ch8fig5) muestra un ejemplo.



*Figura 8-5: Puedes ver algunos de los bloques que el programa ha colocado aleatoriamente. He creado un mundo nuevo para demostrar este programa, de modo que no dañe ninguno de mis edificios.*

[anterior](ch08_2.html)[Subtema 3 de 5: (Ver todo)](ch08.html)[siguiente](ch08_4.html)