Aprende a programar con Minecraft

## **7** **FIESTAS DE BAILE Y DESFILES DE FLORES CON BUCLES WHILE**



*Los* bucles facilitan la repetición del código una y otra vez. En lugar de copiar y pegar el mismo código, puedes utilizar un bucle para repetir el código tantas veces como quieras. En este capítulo utilizarás bucles para hacer que tus programas se repitan sin tener que volver a ejecutarlos. Nos centraremos en un tipo de bucle de Python conocido como bucle while.

### **Un bucle while sencillo**

Los bucles while se utilizan para repetir bloques de código. Similar a las sentencias if, un bucle while ejecutará el código que contiene mientras una condición sea Verdadera. Es decir, debe cumplirse una condición para que se ejecute el cuerpo de la sentencia.

La diferencia entre un bucle while y una sentencia if es que el código de la sentencia if sólo se ejecuta una vez como máximo, mientras que el código del bucle while puede repetirse muchas veces. Los programadores llaman *iteración* a la repetición del código *.* Cuando un bucle se repite, se dice que *itera*.

Por ejemplo, este código utiliza un bucle while para imprimir los números del 1 al 5:

count = 1  
while count <= 5:  
print(count)  
count += 1  
print("Loop finished")

La variable count registra el número de veces que se ha repetido el bucle. Comienza con el valor 1. La condición del bucle while comprueba si la cuenta es menor o igual que 5.

**NOTA**

*En* el [*Capítulo 3*](ch03.xhtml#ch03) *aprendiste que* += *es un operador abreviado. Podrías utilizar el operador estándar de suma* cuenta = cuenta + 1 *para hacer* lo mismo.

La primera vez que se ejecuta el bucle, el valor de cuenta es 1, que es menor que 5. La condición del bucle es Verdadero, y el cuerpo del bucle se ejecuta. A continuación, el programa imprime el valor de la cuenta en el intérprete de comandos de Python, y luego añade 1 al valor de la cuenta. El bucle while vuelve a empezar y comprueba de nuevo la condición, pasando por cada paso hasta que la variable count sea mayor que 5.

Fuera del bucle hay una última línea, que imprime "Bucle finalizado".

Guarda este programa y ejecútalo; deberías ver la siguiente salida:

1  
2  
3  
4  
5  
Loop finished

Prueba a experimentar un poco con el código. Cambia las condiciones para que aparezcan más de 5 números o cambia la cantidad en que aumenta la variable de recuento. Aquí tienes un repaso de cómo funciona el código. La sentencia while sigue estos pasos

1. Comprueba si la condición es Verdadera.
2. Si la condición es Verdadera:
   1. Ejecuta el cuerpo del código.
   2. Repite el paso 1.
3. Si la condición es Falsa:
   1. Ignora el cuerpo de código.
4. Continúa en la línea siguiente al bloque del bucle while.

¡Intentemos utilizar un bucle while en Minecraft para teletransportarnos a un montón de sitios nuevos!

#### **Misión nº 33: Un viaje de teletransporte aleatorio**

En la Misión nº 3[(página 44](ch02.xhtml#page_44)), teletransportaste al jugador a distintas posiciones del juego. Vamos a reescribir ese programa utilizando un bucle while para que puedas repetir el teletransporte una y otra vez.

Mediante un bucle de código que teletransportará al jugador a un lugar aleatorio, puedes hacer que el programa sea más potente *y* mucho más fácil de leer. Mola, ¿eh?

El siguiente código teletransportará al jugador a un lugar aleatorio una vez, eligiendo valores aleatorios en el mundo del juego para las variables x, y y z. Luego fijará la posición del jugador utilizando esas variables.

import random  
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ # Add the count variable here  
➋ # Start the while loop here  
➌ x = random.randint(-127, 127) # Indent the code from this line  
y = random.randint(0, 64)  
z = random.randint(-127, 127)  
  
mc.player.setTilePos(x, y, z)  
➍ # Add 1 to the value of the count variable here

Ahora mismo, sin embargo, el código sólo teletransportará al jugador una vez. Aunque eso está muy bien, puedes hacerlo totalmente genial. Escribamos un bucle para que el código se repita cinco veces, convirtiéndolo en todo un viaje relámpago.

Para cambiar el código y utilizar un bucle, sigue estos cuatro pasos:

1. Crea una variable de recuento para controlar el bucle ➊.
2. Añade un bucle while con una condición basada en el recuento ➋.
3. Sangrar el cuerpo de la sentencia while ➌.
4. Incrementa el valor de count con cada bucle ➍.

La finalidad de la variable count y del incremento de count es llevar la cuenta del número de veces que se ha repetido el bucle. Hablaré más sobre ellos en la siguiente sección. Por ahora, todo lo que necesitas saber es que count nos permite controlar cuántas veces se repite este código.

[El listado 7-1](ch07.xhtml#ch7ex1) muestra el código con los cambios añadidos.

*randomTeleport.py*

import random  
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
count = 0  
while count < 5:  
x = random.randint(-127, 127)  
y = random.randint(0, 64)  
z = random.randint(-127, 127)  
mc.player.setTilePos(x, y, z)  
count += 1

*Listado 7-1: Código para teletransportar aleatoriamente al jugador por el mundo del juego*

Copia [el Listado 7-1](ch07.xhtml#ch7ex1) en un nuevo archivo, guárdalo como *randomTeleport.py* en una nueva carpeta llamada *whileLoops*, y ejecuta el código. Deberías ver cómo el jugador se desplaza por el mundo de Minecraft. ¡Pero el código se ejecuta demasiado rápido! Todo el viaje termina en menos de un segundo. Vamos a arreglarlo juntos.

Utilizarás el módulo de tiempo para ralentizar el código. Sigue estos pasos:

1. En la primera línea del programa, añade la sentencia import time. Esto importa el módulo tiempo de Python, que contiene un conjunto de prácticas funciones relacionadas con el tiempo y otras cosas.
2. Añade la línea time.sleep(10) al final del cuerpo de tu bucle while para añadir un retardo de 10 segundos a tu programa. ¡Asegúrate de sangrar esta nueva línea final de tu programa para que quede dentro del bucle while!

Guarda el programa y ejecútalo. Ahora el jugador debería teletransportarse a un nuevo lugar aleatorio cada 10 segundos. [La Figura 7-1](ch07.xhtml#ch7fig1) muestra mi programa en ejecución.



*Figura 7-1: Cada 10 segundos, el programa me teletransporta a un nuevo lugar.*

**OBJETIVO EXTRA: DUERME TRANQUILO**

De momento, el programa esperará 10 segundos al final de cada bucle. ¿Qué ocurre si mueves la sentencia time.sleep(10) al inicio del bucle?

### **Controlar bucles con una variable de recuento**

Las variables de recuento son una forma habitual de almacenar el número de veces que se ha repetido un programa. Ya has visto estas variables en acción unas cuantas veces. Veamos otro ejemplo:

count = 0  
while count < 5:  
print(count)  
count += 1

La condición del bucle while comprueba que el valor de la variable de recuento es menor que 5. En el cuerpo del bucle, he cambiado el valor de la variable cuenta para registrar el número de veces que se ha repetido la cuenta. Añadir al valor de una variable de recuento se llama *incrementar*.

La última línea de este código incrementa en 1 el valor de la variable de recuento. Cada vez que se repita el código, comprobará el nuevo valor de la variable de recuento para ver si es menor que 5. Cuando sea igual o mayor que 5, el bucle se detendrá.

Si te olvidas de incrementar la variable count, acabarás con un *bucle infinito*, que repetirá el bucle eternamente, como se muestra en el siguiente ejemplo:

count = 0  
while count < 5:  
print(count)

El valor de count es siempre 0 porque nunca se incrementa. Por lo tanto, la condición del bucle siempre será Verdadero, y el bucle se repetirá *eternamente*. Si no me crees, ¡prueba a ejecutar el código!

0  
0  
0  
0  
0  
--snip--

Para interrumpir la ejecución de este programa infinito, pulsa CTRL-C. Para corregir el código, sólo tienes que añadir la cuenta de líneas += 1 al cuerpo del bucle. Ahora no quedarás atrapado en un bucle infinito. ¡Uf!

Los recuentos no siempre tienen que incrementarse en 1. En algunas situaciones puedes querer incrementar el recuento en un valor diferente. En el siguiente ejemplo, la cuenta se incrementa en 2 cada vez; el resultado es que el código imprime todos los números pares entre 0 y 100:

count = 0  
while count < 100:  
print(count)  
count += 2

También puedes contar hacia atrás utilizando un número negativo para *disminuir* el valor de la cuenta. El siguiente código cuenta *hacia* atrás desde 100 hasta 1:

count = 100  
while count > 0:  
print(count)  
count -= 1

La única diferencia entre este ejemplo y los anteriores es la condición. Aquí he utilizado un comparador mayor que(>). Mientras la cuenta sea mayor que 0, el bucle continúa; cuando la cuenta llega a 0, el bucle se detiene.

**NOTA**

*La variable utilizada para controlar un bucle no siempre se llama* recuento*.* *Podrías llamarla* repeticiones *o cualquier otra cosa que desees. Si miras el código de otras personas, verás una gran variedad de nombres diferentes.*

#### **Misión 34: La maldición acuosa**

Intentemos algo un poco desagradable y escribamos una maldición para el jugador que dure poco tiempo. Las maldiciones en los videojuegos pueden *debilitar* al personaje de alguna manera, como ralentizándolo o debilitándolo, a menudo sólo durante un rato.

Crearemos un programa de maldición que coloque un bloque de agua que fluya en la posición del jugador una vez por segundo durante 30 segundos. Esto hará que al jugador le resulte difícil moverse sin ser empujado por el agua que fluye.

El siguiente código coloca un bloque de agua que fluye en la posición del jugador:

*waterCurse.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
pos = mc.player.getPos()  
mc.setBlock(pos.x, pos.y, pos.z, 8)

Este código colocará un bloque de agua en la posición actual del jugador sólo una vez. Tu tarea es hacer que se repita. El código final debe repetirse 30 veces, y cada iteración del bucle debe durar 1 segundo.

Guarda este código como *waterCurse.py* en la carpeta *whileLoops* y ejecútalo una vez para asegurarte de que funciona. Deberías ver aparecer un único bloque de agua en la posición del jugador antes de que el programa se detenga.

Vamos a ver qué hay que añadir a continuación para que esta maldición dure. Utiliza lo que has aprendido sobre los bucles while y las variables de recuento para hacer lo siguiente:

1. Añade una variable de recuento al programa.
2. Añade un bucle al programa para repetir las dos últimas líneas de código. El bucle debe repetirse 30 veces.
3. Incrementa la variable cuenta al final del bucle.
4. Importa el módulo de tiempo (en la primera línea de tu programa) y añade una suspensión de 1 segundo en la última línea del bucle while.

Guarda el programa y pruébalo. Mientras caminas por el mundo del juego, el programa debería crear un bloque de agua cada segundo durante 30 segundos. Si te quedas atascado, vuelve a los pasos de la Misión nº 33[(página 129](ch07.xhtml#page_129)) para obtener ayuda.

La[Figura 7-2](ch07.xhtml#ch7fig2) muestra la maldición en acción.



*Figura 7-2: ¡Oh, no! Me persigue una pequeña inundación.*

**OBJETIVO EXTRA: UNA INUNDACIÓN MÁS RÁPIDA**

¿Cómo harías que el bucle se repitiera el doble de rápido (cada medio segundo) sin dejar de durar 30 segundos?

#### **Bucles while infinitos**

En la mayoría de los casos, es muy importante que la condición booleana de tu bucle while se convierta finalmente en Falso; de lo contrario, el bucle iterará eternamente y tu ordenador podría bloquearse.

Pero hay ocasiones en las que puedes querer programar un bucle infinito. Por ejemplo, los videojuegos suelen utilizar un bucle infinito para comprobar la entrada del usuario y gestionar el movimiento del jugador. Por supuesto, estos videojuegos incluyen un botón de Salir para que puedas pausar o detener los bucles infinitos cuando necesites tomarte un descanso.

Una forma sencilla de crear un bucle infinito es utilizar una condición Verdadero al definir un bucle while, como se muestra aquí:

while True:  
print("Hello")

Este código se repetirá eternamente, imprimiendo la cadena "Hola" una y otra vez. Tanto si pretendías crear un bucle infinito como si no, pulsar CTRL-C en el intérprete de comandos de Python es una forma habitual de detenerlo. En IDLE puedes seleccionar **Shell** image **Reiniciar Shell** para detener también el bucle.

Ten en cuenta que cualquier código que se coloque *después de* un bucle while infinito nunca se ejecutará. En el siguiente ejemplo, la última línea de código es inalcanzable debido al bucle while infinito que le precede:

while True:  
print("Hello")  
print("This line is never reached")

Aunque los bucles infinitos a veces pueden ser complicados, también puedes crearlos para hacer un montón de cosas chulas. ¡Intentémoslo a continuación!

#### **Misión 35: Camino de flores**

El programa que escribirás en esta misión es como el de la Misión 34, pero en lugar de colocar bloques de agua, crearás un rastro de flores detrás del jugador. ¡Las flores son mucho más bonitas que las inundaciones!

Abre el archivo *waterCurse.py* en la carpeta *whileLoops* y guárdalo como *flowerTrail.py*.

Para que aparezca un rastro infinito de flores mientras el jugador camina por el juego, haz los siguientes cambios en el programa:

1. Cambia la condición del bucle while a True.
2. Elimina la variable cuenta y el incremento.
3. Cambia el argumento del tipo de bloque en la función setBlock() de 8 a 38.
4. Reduce el valor del argumento en la función sleep () a 0,2 para que aparezcan cinco flores cada segundo.
5. Guarda el programa y ejecútalo. [La Figura 7-3](ch07.xhtml#ch7fig3) muestra lo que deberías ver.



*Figura 7-3: ¡Mira qué flores más bonitas!*

**OBJETIVO EXTRA: UN RASTRO DE DESTRUCCIÓN**

El programa *flowerTrail.py* es muy flexible. Prueba a cambiar el tipo de bloque que coloca el programa. Un tipo de bloque divertido es el TNT explosivo(setBlock(x, y, z, 46, 1)). Fíjate en el argumento extra 1 después de 46, que es el tipo de bloque TNT. El 1 establece el estado del TNT para que detone con sólo golpearlo, sin necesidad de pedernal y acero. ¡Sólo tienes que pulsar el botón izquierdo del ratón unas cuantas veces cuando apuntes a la TNT para hacerla explotar!

### **Condiciones extravagantes**

Como los bucles while esperan un valor booleano para su condición, puedes utilizar cualquiera de los comparadores y operadores booleanos que has aprendido hasta ahora. Por ejemplo, ya has visto que los operadores mayor que y menor que funcionan igual que en capítulos anteriores.

Pero también puedes controlar los bucles while con comparadores y operadores booleanos de otras formas. ¡Echemos un vistazo!

Empezaremos escribiendo una condición más interactiva. El siguiente código crea la variable continueAnswer antes de que se inicie el bucle y comprueba que el valor es igual a "Y". Ten en cuenta que no podemos utilizar la palabra continuar como nombre de variable porque es una palabra reservada en Python.

continueAnswer = "Y"  
coins = 0  
while continueAnswer == "Y":  
coins = coins + 1  
continueAnswer = input("Continue? Y/N")  
print("You have " + str(coins) + " coins")

En la última línea del bucle while, el programa pide una entrada al usuario. Si el usuario pulsa algo más que "Y " como respuesta, el bucle saldrá. El usuario puede pulsar repetidamente Y y Y y Y, y cada vez el valor de la variable monedas aumentará en 1.

Observa que la variable que se comprueba, continuarRespuesta, se crea antes de que se inicie el bucle. Si no fuera así, el programa mostraría un error. Por eso, la variable que utilizamos para comprobar la condición debe existir antes de que intentemos utilizarla, y debe ser Verdadera cuando el programa llegue al bucle while la primera vez; de lo contrario, la condición no se cumplirá, y la sentencia del cuerpo del bucle while nunca se ejecutará.

#### **Misión 36: Concurso de inmersión**

Vamos a divertirnos un poco con los bucles while y el comparador igual a(==). En esta misión, crearás un minijuego en el que el jugador se sumerja bajo el agua todo el tiempo que pueda. El programa registrará cuántos segundos permanece bajo el agua y mostrará su puntuación al final del programa. Para felicitar al jugador, el programa le regalará flores si permanece bajo el agua más de 6 segundos.

Aquí tienes algo de código para empezar:

*divingContest.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
import time  
  
score = 0  
pos = mc.player.getPos()  
➊ blockAbove = mc.getBlock(pos.x, pos.y + 2, pos.z)  
  
➋ # Add a while loop here  
time.sleep(1)  
pos = mc.player.getPos()  
➌ blockAbove = mc.getBlock(pos.x, pos.y + 2, pos.z)  
➍ score = score + 1  
mc.postToChat("Current score: " + str(score))  
  
mc.postToChat("Final score: " + str(score))  
  
➎ if score > 6:  
finalPos = mc.player.getTilePos()  
mc.setBlocks(finalPos.x - 5, finalPos.y + 10, finalPos.z - 5,  
finalPos.x + 5, finalPos.y + 10, finalPos.z + 5, 38)

Guarda el programa como *divingContest.py* en tu carpeta *whileLoops*. La variable puntuación lleva la cuenta de cuántos segundos permanece el jugador bajo el agua.

Ejecuta el código para ver qué ocurre. De momento, el programa no está completo: sólo comprueba una vez si el jugador está bajo el agua y luego termina.

Antes de arreglar esto, veamos qué hace el resto del código. La variable bloqueEncima almacena el tipo de bloque situado en la cabeza del jugador ➊. Por ejemplo, si la cabeza del jugador está bajo el agua, esta variable almacenará un valor de 8 (lo que significa que el bloque es agua). Más adelante en el código, volverás a establecer blockAbove para que almacene el valor del bloque situado encima de la cabeza del jugador ➌, de modo que cuando crees tu bucle while, actualizará blockAbove con el bloque actual situado encima de la cabeza del jugador. En ➍, el programa añade 1 punto al total por cada segundo que el jugador esté bajo el agua, y en ➎, utiliza una sentencia if para crear una lluvia de flores sobre el jugador si la puntuación es superior a 6.

Te toca añadir un bucle al programa que utilice la variable bloqueSobre como condición en ➋. Haz que el bucle while compruebe si bloqueSobre es igual a agua (bloque de tipo 8) o igual a agua corriente (bloque de tipo 9). Puedes utilizar la siguiente condición en el bucle while para comprobarlo: while bloqueEncima == 8 o bloqueEncima == 9. Esto comprueba si el jugador está actualmente bajo el agua y seguirá comprobando si el jugador está bajo el agua cada vez que se repita el bucle.

Para probar tu programa, busca un poco de agua que tenga al menos tres bloques de profundidad y sumérgete en ella. El programa sólo se ejecutará si ya estás bajo el agua. Cuando ejecutes el programa, debería empezar a mostrar cuántos segundos has estado bajo el agua. Al cabo de un rato, nada hasta la superficie. El programa mostrará tu puntuación y te colmará de flores si has estado bajo el agua 6 segundos o más. La [Figura 7-4](ch07.xhtml#ch7fig4) muestra al jugador bajo el agua y la puntuación que se muestra. La Figura [7-5](ch07.xhtml#ch7fig5) muestra las flores que aparecen cuando ganas.



*Figura 7-4: Estoy aguantando la respiración bajo el agua, y se muestra el número de segundos que he estado bajo el agua.*



*Figura 7-5: ¡He ganado mi propia celebración florida!*

**OBJETIVO EXTRA: UN GANADOR ERES TÚ**

Prueba a añadir premios extra escribiendo más código en la sentencia if al final del programa. Si el jugador obtiene una puntuación alta, podrías darle un bloque de oro. Prueba a añadir varios niveles de dificultad con premios diferentes para cada uno.

#### **Operadores booleanos y bucles while**

Puedes utilizar operadores booleanos como *y*, *o*, y *no* con un bucle while cuando quieras que el bucle utilice más de una condición. Por ejemplo, el siguiente bucle iterará mientras el usuario no haya introducido la contraseña correcta y haya hecho tres intentos o menos:

password = "cats"  
passwordInput = input("Please enter the password: ")  
attempts = 0  
➊ while password != passwordInput and attempts < 3:  
➋ attempts += 1  
➌ passwordInput = input("Incorrect. Please enter the password: ")  
➍ if password == passwordInput:  
print("Password accepted.")

La condición del bucle while ➊ realiza dos tareas: comprueba si la contraseña es diferente de la introducida por el usuario(contraseña != contraseñaEntrada) y comprueba si el usuario ha intentado introducir la contraseña tres veces o menos(intentos < 3). El operador and permite que el bucle while compruebe ambas condiciones al mismo tiempo. Si la condición es False, el bucle incrementa la variable intent os ➋ y pide al usuario que vuelva a introducir la contraseña ➌. El bucle finalizará si el usuario introduce la contraseña correcta o la variable intentos es mayor que 3. Una vez finalizado el bucle, el programa mostrará Contraseña aceptada sólo si el usuario ha introducido la contraseña correcta ➍.

#### **Comprobar un rango de valores en bucles while**

También puedes comprobar valores en un rango determinado utilizando un bucle while. Por ejemplo, el siguiente código comprueba si el valor que ha introducido el usuario está entre 0 y 10. Si no lo está, el bucle saldrá.

position = 0  
➊ while 0 <= position <= 10:  
position = int(input("Enter your position 0-10: "))  
print(position)

Si la variable de posición es mayor que 10, el bucle no se repetirá ➊. Lo mismo ocurrirá si el valor es menor que 0. Esto es útil en Minecraft cuando compruebas si la posición del jugador se encuentra en una determinada zona del juego, como verás en la siguiente misión.

#### **Misión 37: Hacer una pista de baile**

¡Es hora de bailar! Pero antes de que puedas hacer unos buenos movimientos, necesitarás una pista de baile. El programa de esta misión generará una pista de baile que parpadeará con colores diferentes cada medio segundo mientras el jugador permanezca en la pista.

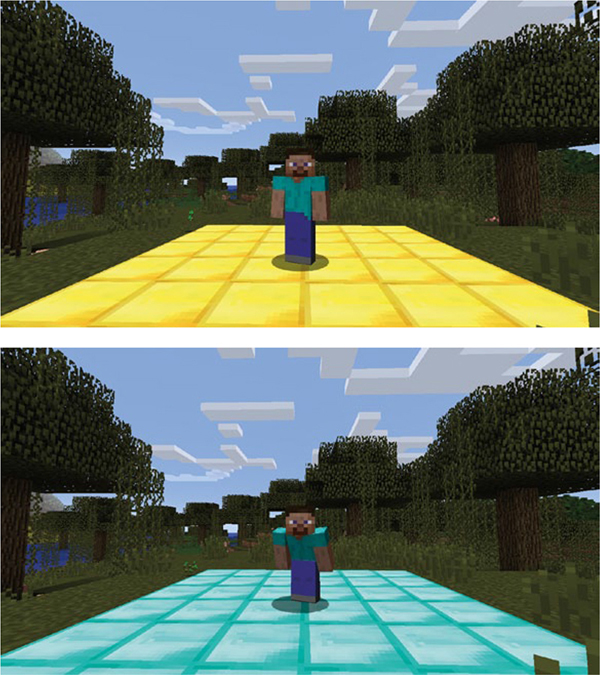
A continuación se muestra el inicio del código. Crea una pista de baile en la posición actual del jugador y utiliza una sentencia if para cambiar los colores. Pero el código no está completo.

*danceFloor.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
import time  
  
pos = mc.player.getTilePos()  
floorX = pos.x – 2  
floorY = pos.y - 1  
floorZ = pos.z – 2  
width = 5  
length = 5  
block = 41  
➊ mc.setBlocks(floorX, floorY, floorZ,  
floorX + width, floorY, floorZ + length, block)  
  
➋ while floorX <= pos.x <= floorX + width and # Check z is within the floor  
➌ if block == 41:  
block = 57  
else:  
block = 41  
  
mc.setBlocks(floorX, floorY, floorZ,  
floorX + width, floorY, floorZ + length, block)  
pos = mc.player.getTilePos()  
time.sleep(0.5)

Abre IDLE, crea un nuevo archivo y guarda el programa como pista de *baile.py* en la carpeta *whileLoops*. El código construye la pista de baile basándose en la posición actual del jugador ➊ y almacena la ubicación y el tamaño de la pista de baile en las variables floorX, floorY, floorZ, width y length. Dentro del bucle while, el código utiliza una sentencia if para alternar los bloques de los que está hecha la pista de baile ➌, haciendo que la pista de baile parezca que parpadea.

Para que el programa funcione correctamente, tienes que cambiar la condición del bucle while para comprobar si la coordenada z del jugador está en la pista de baile ➋. En otras palabras, comprueba si pos.z es mayor o igual que floorZ y menor o igual que floorZ más la longitud. Como orientación, mira cómo he comprobado si pos. x está en la pista de baile utilizando(sueloX <= pos.x <= sueloX + anchura). ¡La [Figura 7-6](ch07.xhtml#ch7fig6) muestra la pista de baile en acción!



*Figura 7-6: Mostrando mis movimientos en la pista de baile.*

Cuando hayas terminado el programa, guárdalo y ejecútalo. Debería aparecer una pista de baile debajo del reproductor y cambiar cada medio segundo. Baila un poco en , ¡diviértete! Cuando hayas terminado, sal de la pista de baile y asegúrate de que deja de parpadear. No volverá a encenderse a menos que vuelvas a ejecutar el programa para crear una nueva pista de baile.

**OBJETIVO EXTRA: SE ACABÓ LA FIESTA**

Cuando el jugador haya terminado de bailar en la pista de baile, haz que ésta desaparezca. Para ello, cambia la pista de baile a aire cuando termine el bucle.

#### **Sentencias if anidadas y bucles while**

Puedes escribir programas más potentes utilizando sentencias if y sentencias if anidadas dentro de bucles while. Puede que hayas visto una sentencia if anidada en el código de la Misión 37[(página 139](ch07.xhtml#page_139)).

En el siguiente ejemplo, la sentencia if anidada comprueba la última palabra que se imprimió y decide si se imprimen las palabras "mine" y "craft". El bucle se repite 50 veces.

word = "mine"  
count = 0  
while count < 50:  
print(word)  
if word == "mine":  
word = "craft"  
else:  
word = "mine"

La variable palabra almacena la primera palabra que se imprimirá. La sentencia if del bucle comprueba si la palabra actual es "mina " y, si lo es, cambia la palabra a "artesanía " y la imprime en la siguiente iteración del bucle. Si la palabra no es "mía ", se cambiará a "artesanía ". Se trata de un bucle infinito, ¡así que asegúrate de utilizar CTRL-C para escapar!

También puedes anidar sentencias elif y otros bucles while dentro de bucles while.

El siguiente programa pregunta al usuario si desea imprimir todos los números comprendidos entre uno y un millón:

userAnswer = input("Print the numbers between 1 and 1000000? (yes/no): ")  
  
➊ if userAnswer = "yes":  
count = 1  
➋ while count <= 1000000:  
print(count)  
count += 1

La sentencia if comprueba si la entrada del usuario es sí ➊. Si lo es, el programa ejecuta el bucle anidado en la sentencia if ➋. Si la entrada es cualquier otra, el programa no ejecutará el bucle y terminará.

#### **Misión 38: El toque de Midas**

Midas es un rey de leyenda. Todo lo que tocaba se convertía en oro. Tu misión es escribir un programa que convierta en oro todos los bloques situados por debajo del jugador, excepto el aire y el agua, claro, ¡o estarías en un buen lío! Recuerda que el bloque dorado tiene un valor de 41, el agua es 9 y el aire es 0.

*midas.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
air = 0  
water = 9  
  
➊ # Add an infinite while loop here  
pos = mc.player.getTilePos()  
blockBelow = mc.getBlock(pos.x, pos.y - 1, pos.z)  
  
➋ # Add if statement here  
mc.setBlock(pos.x, pos.y - 1, pos.z, 41)

Abre IDLE y crea un nuevo archivo. Guarda el archivo como *midas.py* en la carpeta *whileLoops*. Tienes que añadir algo más al programa para que pueda hacer lo que necesitas. En primer lugar, añadirás un bucle while infinito ➊. Recuerda que un bucle while infinito tiene una condición que siempre es Verdadera. También tienes que añadir una sentencia if que compruebe si el bloque situado debajo del jugador no es igual a aire y no es igual a agua estancada ➋. El valor del bloque situado debajo del jugador se almacena en la variable bloqueDebajo, y los valores del aire y del agua se almacenan en las variables aire y agua.

Cuando hayas completado el programa, guárdalo y ejecútalo. El jugador debería dejar un rastro de oro tras de sí. Cuando salte al agua o vuele por el aire, los bloques que hay debajo no deberían cambiar. [La Figura 7-7](ch07.xhtml#ch7fig7) muestra el programa en acción.



*Figura 7-7: Cada bloque que piso se convierte en oro.*

Para salir del bucle infinito, ve a **Shell** image **Reiniciar Shell** en tu shell IDLE o haz clic en el shell y pulsa CTRL-C.

**OBJETIVO EXTRA: SOY UN LABRADOR**

Puedes cambiar *midas.* py para que sirva a diversos propósitos. ¿Qué tal si lo cambias para que cambie automáticamente los bloques de tierra por bloques de hierba? ¿Qué tal cambiar los bloques de tierra por bloques de hierba?

### **Terminar un bucle while con break**

Con los bucles while, tienes control total sobre cómo y cuándo termina el bucle. Hasta ahora sólo has utilizado condiciones para terminar bucles, pero también puedes utilizar una sentencia break. La sentencia break permite que tu código salga inmediatamente de un bucle while. Veamos este concepto.

Una forma de utilizar las sentencias break es ponerlas en una sentencia if anidada en el bucle. Al hacerlo, el bucle se detiene inmediatamente cuando la condición de la sentencia if es Verdadera. El siguiente código pide continuamente la entrada del usuario hasta que escriba "exit":

➊ while True:  
➋ userInput = input("Enter a command: ")  
➌ if userInput == "exit":  
➍ break  
print(userInput)  
➎ print("Loop exited")

Se trata de un bucle infinito porque utiliza while True ➊. Cada vez que se repite el bucle, pide al usuario que introduzca un comando ➋. El programa comprueba si la entrada es " salida" ➌ utilizando una sentencia if. Si la entrada cumple la condición, la sentencia break detiene la repetición del bucle ➍, y el programa continúa en la línea inmediatamente posterior al cuerpo del bucle, imprimiendo " Bucle salido " en la shell de Python ➎.

#### **Misión nº 39: Crear una conversación persistente con un bucle**

En la Misión nº 13[(página 76](ch04.xhtml#page_76)), creaste un programa que publica el mensaje del usuario en el chat utilizando cadenas, entrada y salida. Aunque este programa era útil, estaba bastante limitado porque tenías que volver a ejecutar el programa cada vez que querías publicar un nuevo mensaje.

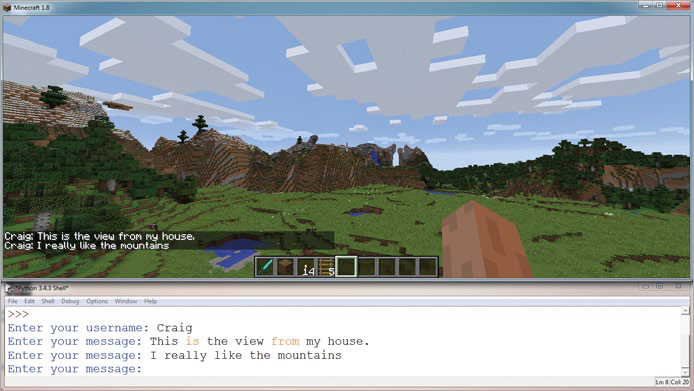
En esta misión, mejorarás tu programa de chat utilizando un bucle while para que los usuarios puedan publicar tantos mensajes como quieran sin reiniciar el programa.

Abre el archivo *userChat.py* en la carpeta *strings* y luego guárdalo como *chatLoop.py* en la carpeta *whileLoops*.

Para publicar un nuevo mensaje cada vez que quieras sin reiniciar el programa, añade lo siguiente a tu código:

1. Añade un bucle while infinito al programa.
2. Añade una sentencia if al bucle para comprobar si la entrada del usuario es "exit". Si la entrada es "exit ", el bucle debería romperse.
3. Asegúrate de que la variable userName está definida antes del inicio del bucle.

Cuando hayas añadido los cambios, guarda tu programa y ejecútalo. Un prompt en el shell de Python te pedirá que escribas un nombre de usuario. Hazlo y pulsa ENTER. A continuación, el programa te pedirá que introduzcas un mensaje. Escribe un mensaje y pulsa INTRO. El programa seguirá pidiéndote que introduzcas un mensaje hasta que escribas exit. [La Figura 7-8](ch07.xhtml#ch7fig8) muestra mi programa de chat en funcionamiento.



*Figura 7-8: Estoy chateando conmigo mismo.*

**OBJETIVO EXTRA: BLOQUEAR EL CHAT**

Amplía la función de chat para que los usuarios puedan crear bloques. Por ejemplo, si el usuario introduce "lana", el programa crea un bloque de lana. Puedes hacerlo añadiendo sentencias elif a tu sentencia if para comprobar la entrada del usuario.

#### **Sentencias while-else**

Al igual que las sentencias if, los bucles while pueden tener condiciones secundarias activadas por sentencias else.

La sentencia else se ejecuta cuando la condición de una sentencia while es False. A diferencia del cuerpo de una sentencia while, la sentencia else se ejecutará sólo una vez, como se muestra aquí:

message = input("Please enter a message.")  
  
while message != "exit":  
print(message)  
message = input("Please enter a message.")  
else:  
print("User has left the chat.")

Este bucle se repite mientras el mensaje introducido no sea igual a "exit". Si el mensaje es "exit ", el bucle dejará de repetirse, y el cuerpo de la sentencia else imprimirá "El usuario ha abandonado el chat".

Si utilizas una sentencia break en la sentencia while, la sentencia else no se ejecutará. El siguiente código es similar al ejemplo anterior, pero incluye una sentencia if anidada y una sentencia break. Cuando el usuario escriba abort en lugar de exit, el bucle de chat saldrá sin imprimir el mensaje "El usuario ha abandonado el chat. " en el chat.

message = input("Please enter a message.")  
  
while message != "exit":  
print(message)  
message = input("Please enter a message.")  
if message == "abort":  
break  
else:  
print("User has left the chat.")

La sentencia if comprueba si el mensaje introducido es "abortar ". Si es True, se ejecuta la sentencia break y el bucle saldrá. Como se ha utilizado la sentencia break, el cuerpo de la sentencia else no se ejecutará y no se imprimirá "El usuario ha abandonado el chat".

#### **Misión 40: Frío y caliente**

En esta misión, crearemos una partida de Frío y Caliente en Minecraft. Si nunca has jugado, la idea es que tu amigo esconde un objeto y tú tienes que encontrarlo. Tu amigo te da pistas en función de lo lejos que estés del objeto. Si estás cerca, tu amigo dirá "Caliente", y si estás lejos, dirá "Frío". Cuando estés justo al lado del objeto, dirá "¡Estás ardiendo!" y si estás muy lejos, dirá "¡Congelación!".

El objetivo del juego es encontrar y colocarte sobre el bloque de diamante que se ha colocado aleatoriamente en el mundo del juego. En esta versión del juego, jugarás tú solo, y el programa Python te dirá a qué distancia del bloque oculto te encuentras. El juego termina cuando te sitúas sobre el bloque diamante.

El listado[7-2](ch07.xhtml#ch7ex2) coloca un bloque en un lugar aleatorio.

*blockHunter.py*

from mcpi.minecraft import Minecraft  
import math  
import time  
import random  
mc = Minecraft.create()  
  
destX = random.randint(-127, 127)  
destZ = random.randint(-127, 127)  
➊ destY = mc.getHeight(destX, destZ)  
  
block = 57  
➋ mc.setBlock(destX, destY, destZ, block)  
mc.postToChat("Block set")  
  
while True:  
pos = mc.player.getPos()  
➌ distance = math.sqrt((pos.x - destX) \*\* 2 + (pos.z - destZ) \*\* 2)  
  
➍ if distance > 100:  
mc.postToChat("Freezing")  
elif distance > 50:  
mc.postToChat("Cold")  
elif distance > 25:  
mc.postToChat("Warm")  
elif distance > 12:  
mc.postToChat("Boiling")  
elif distance > 6:  
mc.postToChat("On fire!")  
elif distance == 0:  
➎ mc.postToChat("Found it")

*Listado 7-2: Inicio del programa Frío y Caliente*

Antes de colocar un bloque al azar, el programa se asegura de que el bloque no se colocará bajo tierra. Para ello, utiliza la función getHeight() ➊, que encuentra el bloque que tiene la coordenada y más alta (es decir, en la superficie) para cualquier posición del juego. Luego coloca un bloque de diamante en una posición aleatoria ➋.

El código en ➌ calcula la distancia al bloque de diamante. Utiliza la función sqrt (), que está en el módulo matemático; por eso es necesario importar matemática al principio del programa. La función sqrt () calcula la raíz cuadrada de un número.

**NOTA**

[*El listado 7-2*](ch07.xhtml#ch7ex2) *utiliza una fórmula llamada* teorema de Pitágoras*. La fórmula utiliza dos lados de un triángulo para calcular la longitud del tercero. En este caso, utilizo la distancia del jugador al bloque oculto en el eje x y en el eje z para calcular la distancia al bloque oculto en línea recta.*

El mensaje que muestra el programa depende de la distancia a la que te encuentres del bloque, que puedes averiguar utilizando una sentencia if y la variable de distancia ➍. El programa muestra "Congelación" si estás muy lejos y " ¡En llamas!" si estás muy cerca.

Copia [el Listado 7-2](ch07.xhtml#ch7ex2) en un nuevo archivo en IDLE y guarda el programa como *blockHunter.py* en la carpeta *whileLoops*.

De momento el programa funciona, pero no termina cuando encuentras el bloque. Para terminar el código, tienes que añadir una sentencia break cuando la distancia del jugador al bloque sea 0 ➎.

Cuando hayas terminado el programa, guárdalo y ejecútalo. Se generará un bloque aleatorio y tendrás que encontrarlo. El programa debería detenerse cuando encuentres el bloque y te pares sobre él. [La Figura 7-9](ch07.xhtml#ch7fig9) muestra que acabo de encontrar el bloque.



*Figura 7-9: He encontrado el bloque y ahora sólo tengo que ponerme encima.*

**OBJETIVO EXTRA: TIEMPO AL TIEMPO**

El programa *blockHunter.* py te da todo el tiempo que necesites para encontrar el bloque. ¿Se te ocurre alguna forma de mostrar el tiempo que tarda el jugador en encontrar el bloque o incluso de limitar el tiempo que tiene para jugar?

### **Lo que has aprendido**

¡Bien hecho! Has aprendido mucho sobre los bucles while. Puedes crear bucles while y bucles while infinitos, y puedes utilizar bucles con condiciones y operadores booleanos. Usando bucles, ahora puedes escribir programas que repiten código, lo que te ahorrará mucho tiempo para que puedas centrarte en dominar Minecraft. En [el Capítulo 8](ch08.xhtml#ch08), aprenderás otra forma de hacer código reutilizable utilizando funciones.