Aprende a programar con Minecraft

## **5** **AVERIGUAR LO QUE ES VERDADERO Y FALSO CON BOOLEANOS**



Siempre haces preguntas de sí o no: ¿Está lloviendo? ¿Tengo el pelo demasiado largo? Una vez que sabes si la respuesta es sí o no, puedes decidir qué hacer a continuación: traer un paraguas, o no; recortarte el pelo, o no. En todas estas situaciones, lo que hagas dependerá de si la respuesta a la pregunta es sí o no. Decidir qué hacer en función de la respuesta a una pregunta también es importante en programación. En este capítulo aprenderás a hacer preguntas en Python.

En programación, las preguntas que haces suelen consistir en comparar valores. ¿Es un valor igual a otro? ¿Es un valor mayor o menor que otro? La pregunta de sí o no se llama *condición*, y la respuesta no es *sí* o *no*, sino Verdadero o Falso. Digamos que haces la pregunta "¿Tengo más bloques de oro que mi amigo?" o, en otras palabras, "¿Es mi alijo de oro mayor que el alijo de oro de mi amigo?". Para convertir esa pregunta en una condición comprensible para Python, tenemos que formularla como una afirmación (como "Mi alijo de oro es mayor que el de mi amigo") que puede ser verdadera o falsa.

Comprobar si una condición es verdadera o falsa es tan útil en Python que existe un tipo de datos especial para almacenar los valores Verdadero y Falso. Hasta ahora has visto otros tipos de datos: enteros, flotantes y cadenas. El tipo de datos que almacena los valores Verdadero y Falso es el tipo de datos *Booleano*. Los booleanos sólo pueden ser Verdadero o Falso. Cuando utilizas Python para hacer preguntas, el resultado es Verdadero o Falso. Cuando una condición es verdadera o falsa, los programadores dicen que *se evalúa a* Verdadero o se *evalúa a* Falso.

En este capítulo utilizarás booleanos, operadores de comparación y operadores lógicos para probar distintas condiciones que implican valores. Entonces estarás preparado para el [Capítulo 6](ch06.xhtml#ch06), donde utilizarás las respuestas a las preguntas para tomar decisiones sobre qué hacer a continuación en un programa.

### **Conceptos básicos de los booleanos**

Un booleano es un poco como un interruptor de la luz: es Verdadero (encendido) o Falso (apagado). En Python, puedes declarar una variable booleana como ésta para representar que la luz está encendida:

light = True

Aquí asignas el valor Verdadero a la variable luz. Para apagar la luz, podrías asignar el valor Falso a luz:

light = False

Escribe siempre en mayúsculas la primera letra de Verdadero y Falso. Si no lo haces, Python no reconocerá el valor como booleano y lanzará una excepción en lugar de evaluar tu cálculo.

En la siguiente misión, utilizarás booleanos para impedir que el jugador rompa bloques en el mundo del juego.

#### **Misión nº 17: ¡Deja de romper bloques!**

En Minecraft, es fácil aplastar bloques, lo que está muy bien cuando quieres minar en busca de recursos. Pero es muy molesto pasarse años construyendo una estructura genial y luego romperla y destruirla accidentalmente. En esta misión, harás que tu mundo Minecraft sea indestructible.

Usando setting ("mundo\_inmutable", True) puedes hacer que los bloques sean *inmutables*, lo que significa que no se pueden cambiar. La línea de código setting( ) es una función como las funciones setTilePos() y setPos() que has visto. [El listado 5-1](ch05.xhtml#ch5ex1) muestra cómo hacer que el mundo sea inmutable.

*immutableOn.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
mc.setting("world\_immutable", True)

*Listado 5-1: Código que impide que se rompan los bloques*

La función setting () tiene opciones que puedes poner a True para activarlas. Una de las opciones es " mundo\_inmutable". Para activar una opción de setting (), escribe True después del nombre de la opción dentro de los paréntesis.

Escribe [el Listado 5-1](ch05.xhtml#ch5ex1) en IDLE y guárdalo como *inmutableOn.py* en una nueva carpeta llamada *booleans*. Cuando lo ejecutes, no deberías poder romper la mayoría de los bloques, como se muestra en la [Figura 5-1](ch05.xhtml#ch5fig1). Pero, ¿qué ocurre cuando *quieres* volver a romper bloques? Copia tu programa en un archivo nuevo y cámbialo para que el jugador pueda romper bloques. (Sugerencia: ¡utiliza un booleano!) Guarda el nuevo archivo como *inmutableOff.py* en la carpeta *booleanos*.



*Figura 5-1: ¡Por mucho que lo intento, el bloque no se rompe!*

### **Concatenar booleanos**

Al igual que los enteros y los flotantes, los booleanos deben convertirse en cadenas antes de poder concatenarlos. Por ejemplo, concatenas booleanos y cadenas cuando quieres imprimir booleanos con la función print(). Para ello, utiliza la función str():

>>> agree = True  
>>> print("I agree: " + str(agree))  
I agree: True

La variable agree almacena un booleano. Se convierte en una cadena en la segunda línea con str(agree), se concatena con la cadena "Estoy de acuerdo: " y se imprime.

### **Comparadores**

Se te da muy bien comparar valores. Sabes que 5 es mayor que 2, que 8 y 8 son el mismo número y que 6 y 12 no son el mismo número. Un ordenador también es bueno comparando valores; sólo tienes que decirle exactamente qué tipo de comparación quieres escribiendo un símbolo llamado *comparador*. Por ejemplo, ¿quieres que compruebe si un valor es mayor que el otro, o quieres que compruebe si es menor?

Los comparadores (u *operadores de comparación*) de Python te permiten comparar datos. Python utiliza seis comparadores:

- Igual a(==)

- No igual a (!=)

- Menor que (<)

- Menor o igual que (<=)

- Mayor que(>)

- Mayor o igual que(>=)

Cada comparador devuelve un valor booleano(Verdadero o Falso) que indica si se ha cumplido la condición. Veamos estos comparadores y exploremos cómo utilizarlos.

#### **Igual a**

Cuando quieras averiguar si dos valores son iguales, puedes utilizar el comparador igual a(==). Cuando los valores son iguales, la comparación devuelve el valor booleano Verdadero. Cuando los valores son diferentes, la comparación devuelve Falso.

Por ejemplo, asignemos el valor de dos variables y utilicemos el operador igual a para compararlas:

>>> length = 2  
>>> width = 2  
>>> length == width  
True

El resultado es Verdadero porque los valores de las variables longitud y anchura son iguales.

Pero si son diferentes, el resultado es Falso:

>>> length = 4  
>>> width = 1  
>>> length == width  
False

Puedes utilizar el comparador igual a en todos los tipos de variables: cadenas, enteros, flotantes y booleanos.

Fíjate en que he utilizado == para comparar la longitud y la anchura en lugar de =, que se utiliza para establecer una variable. Python utiliza el operador == para diferenciar entre una comparación (preguntar si dos valores son iguales) y establecer una variable (hacer que una variable sea igual a algún valor). Intenta recordar esta diferencia para evitar errores en tu código. No te preocupes; ¡incluso yo cometo el error de utilizar = en lugar de == de vez en cuando!

#### **Misión nº 18: ¿Estoy nadando?**

Ahora utilizarás comparadores para crear un programa que indique si estás o no en el agua. Los resultados se publicarán en el chat de Minecraft.

Para averiguar el tipo de bloque en determinadas coordenadas, utilizarás la función getBlock(). Esta función toma las coordenadas como tres argumentos y devuelve el tipo de bloque como un número entero. Por ejemplo:

blockType = mc.getBlock(10, 18, 13)

Aquí he almacenado el resultado de mc.getBlock(10, 18, 13) en una variable llamada tipoBloque. Si el tipo de bloque en las coordenadas (10, 18, 13) es melón (valor de bloque 103), la variable blockType tendrá el valor 103.

Pongamos a trabajar la función getBlock(). El listado [5-2](ch05.xhtml#ch5ex2) comprueba si el jugador está en tierra firme.

*swimming.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
pos = mc.player.getPos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
blockType = mc.getBlock(x, y, z)  
mc.postToChat(blockType == 0)

*Listado 5-2: Este código comprueba el tipo de bloque donde están las piernas del jugador.*

Aquí, obtengo las tres coordenadas de la posición del jugador y paso esas coordenadas como argumentos a getBlock (). Almaceno el resultado de mc.getBlock(x, y, z) en blockType. La expresión blockType == 0 comprueba si el bloque es aire; si es aire, sabes que estás parado en algún lugar de tu mundo Minecraft, la expresión es True, y True se publica en el chat. Si no es aire, se envía False al chat, ¡así que debes estar bajo el agua o quizás ahogándote en la arena!

Copia [el Listado 5-2](ch05.xhtml#ch5ex2) y guárdalo como *nadando.py* en el directorio [*del Capítulo 5*](ch05.xhtml#ch05). Luego cambia el código para que compruebe si el jugador está de pie en el agua (bloque tipo 9) y ejecútalo.

Prueba a ponerte de pie en el agua y a ejecutar el programa. Asegúrate de que cuando el jugador está en el agua, el chat muestra Verdadero. Cuando el jugador no esté en el agua, el chat debe mostrar Falso.

La salida del programa debe parecerse a [la Figura 5-2](ch05.xhtml#ch5fig2).

**NOTA**

*En este punto, no podrás ejecutar este programa de forma continua. Deberás ejecutar el programa cada vez que quieras comprobar el bloque situado debajo del jugador. Esto se aplica también a todas las demás misiones de este capítulo.*



*Figura 5-2: Aunque veo que estoy en el agua, Python me lo confirma amablemente.*

**OBJETIVO EXTRA: ¡ESTOY VOLANDO!**

Con un par de cambios en el código, puedes comprobar si el bloque que *hay debajo de* ti es aire. Esto te indicaría que estás volando o saltando. ¿Cómo lo harías?

#### **No igual a**

El comparador no igual a es lo contrario del comparador igual a. En lugar de comprobar si dos valores son iguales, comprueba si son diferentes. Cuando los dos valores son diferentes, la comparación se evalúa como Verdadero. Si son iguales, se evalúa como Falso.

Supongamos que quieres asegurarte de que un objeto es un rectángulo pero no un cuadrado. Como un rectángulo no cuadrado tiene una longitud y una anchura diferentes, podrías escribir una comparación para comprobar que la longitud y la anchura no son iguales:

>>> width = 3  
>>> length = 2  
>>> width != length  
True

La expresión anchura != longitud pregunta si los valores de anchura y longitud son diferentes.

El resultado de esta comparación es Verdadero porque la variable anchura y la variable longitud tienen valores diferentes.

Pero si estos valores son iguales, la comparación devuelve Falso:

>>> width = 3  
>>> length = 3  
>>> width != length  
False

El comparador no igual a también funciona con cadenas, enteros, flotantes y booleanos, igual que el comparador igual a.

#### **Misión 19: ¿Estoy parado en algo más que aire?**

Supongamos que quieres comprobar si estás sobre algo distinto del aire, como agua, lava, tierra, grava o cualquier otro tipo de bloque. En la Misión nº 18, comprobaste si el bloque en tu posición actual era aire, y resolviste cómo comprobar si estabas en el agua. Podrías copiar y pegar el programa muchas veces, cambiándolo ligeramente cada vez para comprobar si hay lava, tierra, grava, etc., uno por uno. Pero eso sería muy aburrido. En lugar de eso, utiliza el comparador no igual a para comprobar si estás bajo tierra, atrapado en arena, en el fondo del océano, ¡o incluso ahogándote en lava!

Abre el programa de la Misión 18*(nadando.py*) y guárdalo como *notAir.py* en la carpeta de *booleanos*. Elimina la última línea del programa y sustitúyela por el Listado [5-3](ch05.xhtml#ch5ex3).

*notAir.py*

➊ notAir = blockType == 0  
mc.postToChat("The player is not standing in air: " + str(notAir))

*Listado 5-3: Cambios en el programa de natación*

La última línea de este código imprimirá si no estás en el aire. El resultado de la comparación se almacena en la variable noAire ➊. Cuando la comparación se evalúe como Verdadero, el valor de la variable noAire será Verdadero, y cuando la comparación se evalúe como Falso, el valor de la variable noAire será Falso.

Pero la comparación de la primera línea no es del todo correcta ➊. Actualmente comprueba si el bloque Tipo es igual a aire utilizando el comparador igual a(==). En lugar de eso, debería comprobar si la variable bloqueTipo no es igual a aire utilizando el comparador no igual a (!=). Cambia la primera línea para utilizar el comparador no igual a en lugar del comparador igual a. Esto comprobará si el bloque en la posición actual del jugador no es igual a aire.

Cuando ejecutes el programa, asegúrate de que funciona cuando estás en el aire y cuando estás bajo el agua, en la lava, en la grava, en la arena o teletransportado al suelo. El mensaje enviado al chat cuando la condición es Verdadera se muestra en la [Figura 5-3](ch05.xhtml#ch5fig3).



*Figura 5-3: Dándome un agradable y relajante baño en el agua, que no es aire.*

#### **Mayor que y menor que**

Cuando necesites averiguar si un valor es mayor que otro, utiliza el comparador mayor que. El comparador mayor que devolverá Verdadero cuando el valor de la izquierda sea mayor que el de la derecha. Si el valor de la izquierda es menor o igual que el de la derecha, la comparación devolverá Falso.

Supongamos que tenemos una vagoneta minera que no puede levantar más de 99 bloques de obsidiana. Mientras el límite de elevación de nuestro minicarro sea mayor que el número de bloques de obsidiana que está intentando levantar, los bloques se podrán levantar:

>>> limit = 100  
>>> obsidian = 99  
>>> limit > obsidian  
True

¡Brillante! Nuestro minicarro puede transportar cualquier número de bloques de obsidiana que sea menor que 100, y 99 es menor que 100, por lo que límite > obsidiana se evalúa como Verdadero. Pero, ¿y si alguien añade otro bloque de obsidiana a la pila?

>>> limit = 100  
>>> obsidian = 100  
>>> canLift = limit > obsidian  
False

Oh, no, ¡ahora se ha alcanzado el límite! El resultado ahora es Falso: 100 no es mayor que 100; es lo mismo. Nuestro minicarro no puede levantar la obsidiana.

El comparador menor que funciona de la misma manera.

Una furgoneta que pasa por debajo de un puente necesita saber si es lo bastante pequeña para caber debajo de él:

>>> vanHeight = 8  
>>> bridgeHeight = 12  
>>> vanHeight < bridgeHeight  
True

En este caso, la furgoneta cabrá porque es más pequeña que la altura del puente: 8 es menor que 12. Más adelante, la misma furgoneta puede encontrarse con otro puente demasiado bajo para pasar por debajo:

>>> vanHeight = 8  
>>> bridgeHeight = 7  
>>> vanHeight < bridgeHeight  
False

Como 8 no es menor que 7, el resultado es Falso.

#### **Mayor o igual que y Menor o igual que**

Al igual que el comparador mayor que, el comparador mayor o igual que determina si un valor es mayor que otro. A diferencia del comparador mayor que, también se evaluará como Verdadero si los valores son iguales.

Supongamos que voy a dar pegatinas a todas las personas que vinieron a ver la presentación de mi increíble programa. Necesito comprobar si tengo suficientes pegatinas para todos:

>>> stickers = 30  
>>> people = 30  
>>> stickers >= people  
True

Tengo suficientes pegatinas: 30 es lo mismo que 30, por lo que pegatinas >= personas evalúa a Verdadero. Pero supongamos que a uno de mis amigos le parecen chulas las pegatinas y quiere una. Ahora, 31 personas quieren pegatinas:

>>> stickers = 30  
>>> people = 31  
>>> stickers >= people  
False

No tengo suficientes pegatinas: 30 no es mayor o igual que 31. Parece que mi amigo no puede tener una pegatina.

A estas alturas, ya estás preparado para enfrentarte a casi cualquier comparación. Mientras estás en IDLE, prueba también el comparador menor o igual que(<=) para ver cómo funciona.

**NOTA**

*Los comparadores mayor que, mayor o igual que, menor que y menor o igual que no funcionan con cadenas, aunque sí con enteros, flotantes y booleanos.*

#### **Misión nº 20: ¿Estoy por encima del suelo?**

La coordenada y de un jugador en Minecraft muestra la altura a la que se encuentra en el juego. Los bloques también se almacenan utilizando coordenadas, lo que te permite obtener los tipos de bloque en coordenadas concretas utilizando getBlock () y crear bloques en coordenadas concretas utilizando setBlocks().

Para obtener el bloque más alto de Minecraft, puedes utilizar la función getHeight(). La función toma una coordenada x y z y devuelve la coordenada y del bloque más alto en esa posición, como se muestra en el Listado [5-4](ch05.xhtml#ch5ex4).

*aboveGround.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
highestBlockY = mc.getHeight(x, z)  
mc.postToChat(highestBlockY)

*Listado 5-4: Código para encontrar la coordenada y del bloque más alto en la posición actual del jugador*

Este programa obtiene la posición actual del jugador, obtiene la coordenada y del bloque más alto en la posición del jugador, y luego publica este valor en el chat de Minecraft.

Combinando este programa con un comparador mayor que o igual a, puedes comprobar si el jugador está o no por encima del suelo. Hagámoslo ahora.

Copia el programa del [Listado 5-4](ch05.xhtml#ch5ex4) y guárdalo como *aboveGround.py*. Cambia el programa para comprobar si la coordenada y del jugador es mayor que la variable bloqueYmayor. Después, añade código para enviar el resultado al chat con el formato "El jugador está por encima del suelo: Verdadero/Falso".

**CONSEJO**

*Recuerda que puedes almacenar el resultado de una comparación en una variable. Por ejemplo, si quisiera comprobar si* y *es mayor o igual que 10 y almacenar la respuesta en una variable llamada* altoSuficiente*, utilizaría la siguiente sentencia*: altoSuficiente = y >= 10.

Ejecuta el programa cuando hayas hecho estos cambios. La salida de los resultados Falso del programa se muestran en la [Figura 5-4](ch05.xhtml#ch5fig4).



*Figura 5-4: Ahora estoy en una cueva, así que Python tiene mucha razón en que no estoy en la superficie.*

#### **Misión nº 21: ¿Estoy cerca de casa?**

Mientras vagas por el mundo de Minecraft, podrías perderte y olvidar dónde está tu casa. Podrías vagar durante horas sólo para descubrir que estabas cerca de casa cuando te perdiste por primera vez.

Con una sola línea de código, puedes comprobar a qué distancia estás de cualquier coordenada del juego. Por ejemplo, podrías utilizar las coordenadas de tu casa y tu posición actual para calcular a qué distancia estás. Añadiendo un comparador, también puedes comprobar si estás o no a un determinado número de manzanas de tu casa. Diremos que estás cerca de casa si estás a sólo 40 manzanas.

¡Escribamos un programa Python que compruebe esto por ti! El código de esta misión debe comprobar a qué distancia estás de tu casa, como se muestra en el Listado [5-5](ch05.xhtml#ch5ex5).

*farFromHome.py*

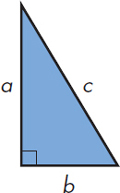
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
import math  
➊ homeX = 10  
homeZ = 10  
pos = mc.player.getTilePos()  
x = pos.x  
z = pos.z  
➋ distance = math.sqrt((homeX - x) \*\* 2 + (homeZ - z) \*\* 2)  
➌ mc.postToChat(distance)

*Listado 5-5: Código que calcula la distancia a tu casa*

Este código supone que tu casa está en las coordenadas x = 10 y z = 10, que se establecen con las variables homeX y homeZ ➊. En este caso, no necesitamos conocer la coordenada y. Utilizo la función getTilePos() para obtener la posición del jugador y establecer los valores x y z.

Para calcular la variable distancia, utilizamos una fórmula llamada *teorema de Pitágoras*. Calcula la longitud de un lado de un triángulo rectángulo, y puedes utilizarla en Minecraft para calcular la distancia entre dos puntos. Es posible que hayas visto esta fórmula escrita en clase de matemáticas como a2 + b2 = c2, donde *a* y *b* son los dos catetos de un triángulo rectángulo, y *c* es la hipotenusa del triángulo, como se muestra en [la Figura 5-5](ch05.xhtml#ch5fig5). En ➋, estamos resolviendo para *c*, que está representada por la variable distancia.

Guarda [el listado 5-5](ch05.xhtml#ch5ex5) como *distanciaFromHome.py* en la carpeta de *booleanos*.



*Figura 5-5: Un triángulo rectángulo*

Para terminar el programa, utiliza un comparador menor o igual que para comprobar si el valor de la variable distancia es menor o igual que 40 y envía el resultado al chat con el formato "Tu casa está cerca: Verdadero/Falso" ➌. Utiliza la concatenación para combinar la cadena con el resultado de la comparación. Actualiza el contenido de la función postToChat() ➌ para mostrar la cadena.

Prueba el programa. Cuando estés a menos de 40 manzanas de tu casa, deberías recibir un mensaje Verdadero; cuando no estés a menos de 40 manzanas, deberías ver un mensaje Falso. [La Figura 5-6](ch05.xhtml#ch5fig6) muestra el programa en funcionamiento.



*Figura 5-6: Definitivamente, estoy a menos de 40 manzanas de mi casa. De hecho, ¡ahí está la puerta principal!*

### **Operadores lógicos**

A menudo es necesario combinar dos o más comparadores en los programas. Puede que quieras determinar si dos condiciones son Verdaderas: por ejemplo, puede que quieras un coche que sea rojo *y* cueste menos de 10.000 $.

Para combinar dos o más comparadores, utilizas *operadores lóg*icos. Al igual que los comparadores, puedes utilizar operadores lógicos en cualquier lugar en el que utilizarías un valor booleano. Los operadores lógicos también se llaman *operadores booleanos*. Conocerás tres tipos de operadores lógicos: y, o, y no.

#### **y**

Utiliza el operador y cuando quieras comprobar si dos comparaciones son ambas Verdaderas. Para que una expresión con un operador y sea Verdadera, ambas comparaciones deben ser Verdaderas. Si alguna de las comparaciones es Falsa, la expresión devolverá Falso.

Supongamos que quiero averiguar si una persona tiene más de 18 años y es propietaria de un coche. Podría escribir el siguiente programa:

>>> age = 21  
>>> ownsCar = True  
>>> age > 18➊ and ownsCar == True➋  
True

Aquí estamos combinando dos comparadores en ➊ y ➋ con y. Como la edad de la persona es superior a 18(edad > 18 se evalúa como Verdadero) y posee un coche (poseeCoche== Verdadero), toda la expresión edad > 18 y poseeCoche == Verdadero se evalúa como Verdadero.

Si una de estas comparaciones fuera Falsa, la expresión se evaluaría como Falsa. Supongamos que la persona no tiene coche pero es mayor de 18 años:

>>> age = 25  
>>> ownsCar = False  
>>> age > 18 and ownsCar == True  
False

Aquí, edad > 18 se evalúa como Verdadero y cochepropio == Verdadero se evalúa como Falso, con lo que toda la expresión sería Falsa.

[La Tabla 5-1](ch05.xhtml#ch05tab01) resume los resultados de todas las combinaciones booleanas posibles y los resultados al utilizar el operador y.

**Tabla 5-1:** Las distintas combinaciones de Verdadero y Falso con el operador y

| **Comparación A** | **Comparación B** | **A y B** |
| --- | --- | --- |
| TRUE | TRUE | VERDADERO |
| VERDADERO | FALSO | FALSO |
| FALSO | TRUE | FALSE |
| FALSE | FALSE | FALSO |

#### **Misión nº 22: ¿Estoy totalmente sumergido?**

En la Misión nº 18[(página 89](ch05.xhtml#page_89)), comprobaste si el jugador estaba nadando. El programa devolvía Verdadero o Falso dependiendo de si el bloque en la posición actual del jugador era igual al agua. Eso te decía si las piernas del jugador estaban bajo el agua, pero daría el mismo resultado si la cabeza del jugador estaba o no bajo el agua. ¿Cómo comprobar si tanto las piernas como la cabeza del jugador estaban bajo el agua?

Con unos sencillos cambios para incluir un operador y, el programa nadando *.* py puede comprobar si las piernas y la cabeza del jugador están bajo el agua. Abre *nadando*.py y guárdalo como *bajoelagua.py*.

Haz los siguientes cambios para que el programa compruebe si el jugador está completamente bajo el agua:

1. Añade una segunda variable que compruebe el tipo de bloque en la posición y + 1 del jugador. Esta variable almacena el tipo de bloque en la cabeza del jugador. Llama a esta variable bloqueTipo2.
2. Comprueba si blockType es igual a agua y si blockType2 es igual a agua.
3. Publica el resultado de la comparación en el chat con este mensaje "El jugador está bajo el agua: Verdadero/Falso".

**CONSEJO**

*Para comprobar* si bloqueTipo *y* bloqueTipo2 son iguales a agua *, puedes utilizar el* *operador y. Primero, comprueba* si bloqueTipo es igual a agua *con la expresión* bloqueTipo == 9*.* *En segundo* lugar, comprueba si bloqueTipo2 es igual a agua con *la* expresión bloqueTipo2 == 9*.* *Para combinar las dos, pon un* *operador* y *en medio, así:* bloqueTipo == 9 y bloqueTipo2 == 9.

Cuando ejecutes el programa, asegúrate de que funciona en los tres casos (cuando el jugador está por encima del agua, cuando sólo las piernas del jugador están en el agua y cuando están completamente debajo del agua). [La Figura 5-7](ch05.xhtml#ch5fig7) muestra un ejemplo de funcionamiento del programa.



*Figura 5-7: El jugador está bajo el agua, corriendo por el fondo marino.*

**OBJETIVO EXTRA: ¿ESTOY EN UN TÚNEL?**

Comprueba si el jugador está o no en un túnel de tierra o de adoquines. Para ello, tendrás que comprobar los bloques situados por encima y por debajo del jugador.

#### **o**

El operador o funciona de forma diferente a y. Cuando una o ambas comparaciones son Verdaderas, la expresión o devolverá Verdadero. Mientras una de las comparaciones sea Verdadera, la expresión seguirá siendo Verdadera. Pero si ninguna de las comparaciones es Verdadera, la expresión se evaluará como Falsa.

Supongamos que quiero adoptar un gato de color negro o naranja. Podría utilizar el código siguiente para obtener la entrada del usuario, y luego ver si el valor de la cadena es "negro" o "naranja":

catColor = input("What color is the cat?")  
myCatNow = catColor == "black" or catColor == "orange"  
print("Adopt this cat: " + str(myCatNow))

Mientras el color del gato sea " negro" o "naranja", lo adoptaré. Pero si es de otro color, como "gris ", miColorGato sería Falso y no adoptaría al gato.

[La Tabla 5-2](ch05.xhtml#ch05tab02) contiene todas las combinaciones posibles y los resultados de utilizar el operador o con booleanos.

**Tabla 5-2:** Las distintas combinaciones de Verdadero y Falso con el operador o

| **Comparación A** | **Comparación B** | **A o B** |
| --- | --- | --- |
| TRUE | TRUE | VERDADERO |
| VERDADERO | FALSO | VERDADERO |
| FALSO | VERDADERO | TRUE |
| FALSO | FALSO | FALSO |

#### **Misión nº 23: ¿Estoy en un árbol?**

Los programas que has creado hasta ahora en este capítulo han mostrado Verdadero o Falso en función de si el jugador está sobre o en un tipo de bloque concreto. Pero, ¿y si quisieras comprobar si el jugador está en un árbol? ¿Cómo lo harías? Como los árboles están hechos de madera y hojas, tendrías que comprobar si el jugador está sobre madera *u* hojas.

Escribamos un programa. Abre de nuevo *nadando.py* y guárdalo como un nuevo programa llamado *enÁrbol.py*.

Cambia el programa para que compruebe el tipo de bloque que está un bloque por debajo del jugador. Querrás utilizar el operador o para comprobar si el bloque que está debajo del jugador es de hojas (tipo de bloque 18) o de madera (tipo de bloque 11), y luego enviar el resultado al chat.

Recuerda que puedes comprobar el bloque que hay debajo del jugador utilizando y = y - 1.

**NOTA**

*Aunque los árboles y las hojas tienen colores diferentes, todos los árboles comparten el mismo ID de bloque, y todas las hojas comparten el mismo ID de bloque. (Las únicas excepciones son la madera y las hojas de acacia y roble oscuro, que son un tipo de bloque diferente. Por ahora, ignoremos la Acacia y el Roble Oscuro). El color se establece utilizando un segundo valor, que conocerás en un capítulo posterior.*

Cuando ejecutes el programa, deberías ver la misma salida que en la [Figura 5](ch05.xhtml#ch5fig8)-8.



*Figura 5-8: Estoy en un árbol.*

#### **not**

El operador not es bastante diferente de los operadores and y or. Se utiliza sobre un único valor booleano o comparación y simplemente cambia su valor por el opuesto.

En otras palabras, not cambia Verdadero a Falso y Falso a Verdadero:

>>> not True  
False  
>>> not False  
True

El operador no es útil cuando empiezas a combinarlo con otros operadores lógicos. Asignemos el valor de tiempoParaCama si no tienes hambre y tienes sueño.

>>> hungry = False  
>>> sleepy = True  
>>> timeForBed = not hungry and sleepy  
>>> print(timeForBed)  
True

El operador no sólo se aplica al booleano al que precede. Aquí invierte el valor de la variable hambriento y deja solo el de la variable somnoliento. Como antes pusimos hambriento en Falso, escribir no hambriento cambia ahora el valor a Verdadero. El valor de somnoliento es Verdadero. Ambos valores son ahora Verdadero, por lo que tiempoParaCama es Verdadero.

#### **Misión 24: ¿Este bloque no es un melón?**

Tienes hambre y quieres saber si tienes comida en casa. Tu comida favorita es el melón, que siempre guardas en el mismo espacio de tu casa. Pero no recuerdas si te queda melón, y tienes que decidir si vas a por comida de camino a casa.

Por suerte, ¡estás aprendiendo Python! Con un poco de cerebro, puedes escribir un programa Python para comprobar si tienes un melón en casa.

En esta misión, crearás un programa que diga si necesitas o no encontrar comida antes de volver a tu casa de Minecraft. El programa comprobará si hay un melón en determinadas coordenadas. Las coordenadas que comprobarás dependen de ti: pueden estar en tu casa, en tu granja o en cualquier lugar donde decidas guardar algún melón. Colocar un melón en esas coordenadas también depende de ti.

Copia [el Listado 5-6](ch05.xhtml#ch5ex6) y guárdalo como *noAMelón.py*.

*notAMelon.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
x = 10  
y = 11  
z = 12  
➊ melon = 103  
➋ block = mc.getBlock(x, y, z)  
  
➌ noMelon = # Check the block is not a melon  
  
➍ mc.postToChat("You need to get some food: " + str(noMelon))

*Listado 5-6: El inicio del código para comprobar si hay un melón en un lugar concreto*

El código sirve para comprobar si el bloque que hay en una posición concreta es un bloque melón. He incluido una variable llamada melón que almacena el ID de bloque de un melón (103) ➊, y he llamado al método getBlock() y almacenado el resultado en una variable llamada bloque ➋. Para completar este programa, tienes que terminar la línea en ➌ que comprueba si la variable melón no es igual a la variable bloque. El resultado debe almacenarse en la variable noMelon para que pueda enviarse al chat de Minecraft en la última línea ➍.

Puedes escribir la comprobación ➌ para ver si las variables melón y bloque no son iguales de dos formas: puedes utilizar el comparador no igual a o el operador lógico no. Aunque el programa funcionará de cualquier forma, prueba a utilizar el operador no lógico para este programa.

Ejecuta el programa cuando hayas hecho los cambios. El resultado debería parecerse a [la Figura 5-9](ch05.xhtml#ch5fig9).



*Figura 5-9: Hay un melón en mi granja, así que no necesito encontrar otro alimento.*

**OBJETIVO EXTRA: UNA DESPENSA BIEN SURTIDA**

Cambia el tipo de bloque que comprueba el programa. Podrías comprobar si crece maíz en tu granja o si alguien ha robado la puerta de tu casa.

#### **Orden de los operadores lógicos**

Puedes combinar tantos operadores lógicos como quieras en una sola sentencia. Por ejemplo, aquí tienes una combinación bastante elegante utilizando y, o, y no:

>>> True and not False or False  
True

Este código se evalúa como Verdadero. ¿Te sorprende? En este ejemplo, la parte no Falsa de la sentencia se evalúa primero a Verdadero. Esto equivale a

>>> True and True or False  
True

A continuación se evalúa la y, y Verdadero y Verdad ero se evalúa a Verdadero, lo que equivale a:

>>> True or False  
True

Por último, se evalúa la o, por lo que Verdadero o Falso se convierte en Verdadero.

Cuando Python evalúa operadores lógicos, utiliza un orden determinado. Si te equivocas en el orden, ¡puedes obtener un resultado que no esperabas! Esto es lo que Python evalúa primero, segundo y tercero:

1. no
2. y
3. o

Practica creando sentencias con operadores lógicos en IDLE y comprueba si puedes adivinar el resultado de cada una.

#### **¿Está mi número entre otros dos?**

A menudo, querrás comprobar si un valor es menor que otro y mayor que otro. Imaginemos que quieres asegurarte de que tienes entre 10 y 20 lobos, porque te encantan los lobos y quieres más de 10, pero 20 o más podrían causar problemas, ya que te quedarías sin comida. Podrías comprobar esta condición utilizando un operador y:

wolves = input("Enter the number of wolves: ")  
enoughWolves = wolves > 10 and wolves < 20  
print("Enough wolves: " + str(enoughWolves))

Pero también podrías hacerlo de otra forma. En lugar de utilizar el operador y, escribe la variable una vez en medio de dos comparaciones:

wolves = input("Enter the number of wolves: ")  
enoughWolves = 10 < wolves < 20  
print("Enough wolves: " + str(enoughWolves))

Si ejecutas cualquiera de estos programas e introduces un número entre 10 y 20, pero que no sea igual a ninguno de los dos, entonces SuficienteLobos será Verdadero. Puedes hacer lo mismo utilizando los operadores mayor que o igual a(>=) y menor que o igual a(<=):

wolves = input("Enter the number of wolves: ")  
enoughWolves = 10 <= wolves <= 20  
print("Enough wolves: " + str(enoughWolves))

En este caso, introducir 10 ó 20 también daría a suficientesLobos un valor de Verdadero.

#### **Misión nº 25: ¿Estoy en casa?**

Con código Python, puedes hacer que ocurran acciones interesantes cuando entres en una determinada zona del mapa. Puedes hacer que se abra una puerta secreta cuando el jugador entre en un bloque concreto, o puedes atraparlo en una caja cuando pase por encima de una trampa. En esta misión, te mostraré cómo detectar si hay alguien en tu casa de Minecraft.

En la Misión 8[(página 59](ch03.xhtml#page_59)), creaste un programa que construye automáticamente las paredes, el techo y el suelo de un edificio. Guardaste el programa como *edificio.py* en la carpeta *matemáticas*. Abre este programa.

Lee el código del programa edificio *.* py y anota los valores de las variables anchura, altura y longitud (por defecto los valores eran 10, 5 y 6, respectivamente). Anota también las coordenadas en las que te encuentras actualmente. Ejecuta el programa de construcción para construir una casa.

Ahora que has creado un edificio, podemos escribir un programa como el Listado [5-7](ch05.xhtml#ch5ex7) que compruebe si el jugador está dentro de él.

*insideHouse.py*

from mcpi.minecraft import Minecraft  
mc = minecraft.create()  
  
➊ buildX =  
buildY =  
buildZ =  
➋ width = 10  
height = 5  
length = 6  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➌ inside = buildX < x < buildX + width and

*Listado 5-7: El inicio del programa para comprobar si el jugador está dentro de su casa*

Se supone que el Listado[5-7](ch05.xhtml#ch5ex7) comprueba que la coordenada x del jugador está dentro del edificio creado por *edificio.py, ¡pero* el programa no está terminado! Tu trabajo consiste en asegurarte de que el programa comprueba también las coordenadas y y z con las coordenadas de la casa que has construido con el programa edificio *.* py.

Copia [el Listado 5-7](ch05.xhtml#ch5ex7) en un nuevo archivo y guárdalo como *casainterior.py*. Completarás el programa para que compruebe si el jugador está dentro del edificio.

Para completar el programa, haz lo siguiente

1. Añade las coordenadas del edificio (son las coordenadas en las que estabas cuando ejecutaste el programa *edificio.* py) ➊.
2. Corrige las variables anchura, altura y longitud si son diferentes de las utilizadas en tu programa edificio *.* py ➋.
3. Completa la comparación de la variable inside para que compruebe si las coordenadas del jugador están dentro del edificio. La primera parte, comprobar si la posición x está en la casa, ya la has hecho ➌. Tienes que añadir las comparaciones para las posiciones y y z. Las expresiones son similares a la que he incluido para la posición x(construirX < x < construirX + anchura).
4. Publica en el chat el valor de la variable interior.
5. Cuando hayas hecho los cambios, guarda y ejecuta el programa. Deberías ver una salida similar a la de [la Figura 5-10](ch05.xhtml#ch5fig10).

* 
* *Figura 5-10: Estoy en mi dormitorio, que efectivamente está dentro de mi casa.*

### **Lo que has aprendido**

En este capítulo has utilizado booleanos, comparadores y operadores lógicos para responder a preguntas en tus programas. En [el Capítulo 6](ch06.xhtml#ch06), escribirás programas que tomen decisiones basadas en las respuestas a estas preguntas. Comprobarás si una condición es verdadera o no, y le dirás al programa que ejecute cierto código si la condición es verdadera o que ejecute un código diferente si es falsa. En el [Capítulo 7](ch07.xhtml#ch07), tus programas seguirán ejecutando un trozo de código mientras una condición sea verdadera y se detendrán si la condición pasa a ser falsa. Éste es el verdadero poder de los booleanos y los comparadores. Te ayudan a controlar qué código se ejecuta en tu programa y exactamente cuándo se ejecuta el código.