Aprende a programar con Minecraft

### **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.*

[anterior](ch05_4.html)[Subtema 5 de 6: (Ver todo)](ch05.html)[siguiente](ch05_6.html)