Aprende a programar con Minecraft

## **3** **CONSTRUYENDO RÁPIDO Y VIAJANDO LEJOS CON LAS MATEMÁTICAS**



En [el Capítulo 2](ch02.xhtml#ch02), aprendiste a crear una variable y a cambiar su valor. En este capítulo, aprenderás a utilizar las matemáticas en Python para generar cualquier bloque que desees y construir rápidamente estructuras complejas en tu mundo de Minecraft. Incluso te darás superpoderes para hacer que el jugador ¡supersalte!

### **Expresiones y sentencias**

Cuando mantienes una conversación con alguien, quieres que entienda lo que le estás diciendo. Utilizas frases cortas, como "tres diamantes" o "detrás de un árbol", para dar información a tu interlocutor. Sin embargo, las frases no tienen sentido por sí solas a menos que se combinen en oraciones, como "Encontré tres diamantes detrás de un árbol".

La programación en Python tiene conceptos similares a las frases y oraciones, que se denominan expresiones y sentencias.

Puedes combinar valores, variables y operadores para crear pequeños trozos de código llamados *expresiones*, como 2 + 2. Las expresiones pueden combinarse en *sentencias*, sobre las que aprendiste en [el Capítulo 2](ch02.xhtml#ch02). Las sentencias son líneas individuales o bloques cortos de código que hacen algo en un programa, como zombies = 2 + 2. En este ejemplo, 2 + 2 es una expresión y forma parte de la sentencia zombies = 2 + 2.

Para programas más largos que utilicen un editor de texto en lugar del intérprete de comandos de Python, asegúrate de escribir expresiones completas. Por ejemplo, el shell de Python y un programa escrito en un editor de texto tratarán la expresión 2 + 2 de forma totalmente diferente. Cuando utilices el intérprete de comandos de Python en IDLE, Python mostrará 4 como resultado de 2 + 2, como se muestra aquí:

>>> 2 + 2  
4

Sin embargo, cuando utilices un editor de texto, Python no hará nada con la expresión porque no forma parte de una sentencia completa. Para convertir esta expresión en una sentencia completa, podrías asignar su valor a una variable, así:

zombies = 2 + 2

Luego imprime esa variable para ver su valor:

print(zombies)

Cuando ejecutes este código, imprimirá 4.

De nuevo, cuando escribas programas en el editor de texto, es muy importante que utilices sentencias completas, no sólo expresiones.

### **Operadores**

En matemáticas, los *operadores* se utilizan para alterar y combinar números. Por ejemplo, el operador suma te permite sumar dos (o más) números, y el operador resta se utiliza para restar un número de otro.

Python utiliza todos los operadores matemáticos básicos que ya conoces -suma, resta, multiplicación y división-, así como operadores más avanzados, como los exponentes. Empecemos por la suma.

#### **Suma**

En Python, la suma se escribe normalmente con el signo más(+). Por ejemplo, si tienes dos flores y coges dos más, podrías describirlo con una expresión que utilice la suma:

>>> flowers = 2 + 2

Python calcula el resultado de la expresión a la derecha del signo igual y lo asigna a la variable de la izquierda. En este caso, el resultado de la expresión de la derecha es 4. El resto del tiempo que se utilice este código concreto, la variable flores tendrá el valor 4.

Puedes utilizar la suma en Minecraft para construir cosas en un abrir y cerrar de ojos. ¿Estás preparado para tu próxima misión? ¡Empecemos!

#### **Misión nº 5: Apilar bloques**

Puedes utilizar la función setBlock( ) para crear y colocar un bloque en Minecraft. Al igual que setPos() y setTilePos(), setBlock () toma como argumentos las coordenadas x, y y z, pero además necesita un cuarto valor: el tipo de bloque. Este valor identifica el tipo de bloque que quieres colocar en el juego.

Ya sea hierba, lava, melón o cualquier otro bloque, cada tipo está representado por un número entero específico. Por ejemplo, la hierba es 2, el aire vacío es 0, el agua es 8 y el melón es 103. Para ver una lista completa de bloques y sus valores enteros, consulta "[Hoja de trucos de ID de bloque](app03.xhtml#app03)" en la [página 299](app03.xhtml#page_299).

Para utilizar setBlock(), pasa a la función los valores de las coordenadas x, y y z y el número entero que representa el tipo de bloque, separados por comas. Por ejemplo, vamos a colocar un bloque melón (tipo 103) en las coordenadas (6, 5, 28):

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
mc.setBlock(6, 5, 28, 103)

Después de las dos primeras líneas familiares que verás en todos los programas Python de Minecraft, sólo tienes que llamar a setBlock() con todos los valores que quieras utilizar. También puedes utilizar variables en lugar de números para conseguir el mismo efecto, como se muestra en el Listado [3-1](ch03.xhtml#ch3ex1).

*blockStack.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
x = 6  
y = 5  
z = 28  
blockType = 103  
mc.setBlock(x, y, z, blockType)

*Listado 3-1: Un programa para crear un bloque melón*

Primero, crea variables para representar las coordenadas del bloque(x, y, z) y el tipo(tipoBloque). Luego, pasa todas las variables a la función setBlock(), y la API Python de Minecraft hará su magia. Ahora puedes volver a utilizar esas variables en cualquier parte de tu programa, y si decides cambiar sus valores más adelante, sólo tendrás que cambiarlos en un lugar.

Cuando combinas este código con operadores matemáticos, puedes hacer cosas muy chulas. Vamos a crear una pila de bloques.

Crea una nueva carpeta llamada *math* dentro de la carpeta *Python de Minecraft*. Abre IDLE y crea un programa en blanco utilizando el editor de texto de IDLE. Guarda este archivo como *blockStack.py* en la carpeta *math*. Copia el código [del Listado 3-1](ch03.xhtml#ch3ex1) en tu editor y añade las dos líneas del [Listado 3-2](ch03.xhtml#ch3ex2) para apilar otro bloque melón encima del que acabas de colocar.

*blockStack.py*

➊ y = y + 1  
➋ mc.setBlock(x, y, z, blockType)

*Listado 3-2: Código adicional para apilar un segundo bloque melón sobre el primer melón*

Estás añadiendo 1 al valor de y ➊, y estás utilizando la función setBlock() para crear otro nuevo bloque ➋. Al aumentar en 1 el valor de y, el segundo bloque se coloca más alto en el eje y que el primero, por lo que el segundo bloque se apila sobre el primero.

A partir de aquí, tu misión es añadir dos bloques más a la pila. Prueba a modificar tu programa *blockStack.py* para que apile cuatro bloques en lugar de dos. Cuando ejecutes tu programa, debería aparecer una pila de cuatro bloques de melón, como se muestra en la [Figura 3-1](ch03.xhtml#ch3fig1).



*Figura 3-1: He hecho una pila de bloques melón.*

**CONSEJO**

*Para añadir un segundo bloque encima del primero, aumentamos la* *variable* y *en 1 y luego utilizamos* *de nuevo la función* setBlock() *. ¿Qué crees que pasaría si reutilizaras estas dos sentencias al final de tu programa? ¿Y si las utilizases tres veces? ¿Sería ésta una solución para crear una pila de cuatro bloques?*

**OBJETIVO EXTRA: CREAR UN ARCO IRIS**

Podrías escribir muchas variaciones del programa *blockStack.py*. Modificando los tipos de bloques, ¡puedes crear un arco iris o una torre de lava! Prueba a cambiar los tipos de bloque para ver qué puedes crear.

#### **Misión nº 6: Súper Salto**

En [el Capítulo 2](ch02.xhtml#ch02), aprendiste a cambiar la ubicación del jugador. Llevemos esa habilidad un paso más allá y lancemos al jugador por los aires utilizando el poder de la suma. Primero, averigua dónde está el jugador llamando a getTilePos(), como se muestra en el Listado [3-3](ch03.xhtml#ch3ex3).

*superJump.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
position = mc.player.getTilePos()  
x = position.x  
y = position.y  
z = position.z

*Listado 3-3: Código para averiguar la posición del jugador*

El punto entre la variable posición y las x, y y z se llama *notación* de punto. La notación de punto se utiliza en determinadas variables y funciones, como todas las funciones que utilizas en la API Python de Minecraft (por ejemplo, mc.setTilePos()). Aprenderás más sobre la notación de puntos en [los Capítulos 11](ch11.xhtml#ch11) y [12](ch12.xhtml#ch12).

Una vez que tengas la posición del jugador, puedes establecer las variables x, y y z a las coordenadas actuales del jugador, que están representadas por posicion.x, posicion.y y posicion.z. A continuación, puedes teletransportar al jugador a donde quieras en relación con las coordenadas actuales, como se muestra en [el](ch03.xhtml#ch3ex4) Listado 3-4.

*superJump.py*

x = x + 5  
mc.player.setTilePos(x, y, z)

*Listado 3-4: Código para mover la posición x del jugador 5 bloques hacia arriba*

Aquí he transportado al jugador 5 bloques a lo largo del eje x, pero esto no tiene nada de especial: puedes mover al jugador horizontalmente siempre que quieras en Minecraft. En lugar de eso, ¡démosle al jugador un super salto!

Tu misión es hacer que el jugador salte 10 bloques en el aire por encima de su posición actual. Deberías poder hacerlo utilizando el código de los Listados [3-3](ch03.xhtml#ch3ex3) y [3-4](ch03.xhtml#ch3ex4), pero con algunas ligeras diferencias. Copia el código de los Listados [3-3](ch03.xhtml#ch3ex3) y [3-4](ch03.xhtml#ch3ex4) en IDLE, guárdalo como *superSalto.py*, y cambia la variable y de forma similar a como yo cambié la variable x. Cuando ejecutes el programa, el jugador debería saltar en el aire, como en la [Figura 3-2](ch03.xhtml#ch3fig2).



*Figura 3-2: ¡El súper salto en acción!*

#### **Resta**

Python maneja la resta de forma similar a como maneja la suma. Supongamos que estás explorando una cueva, te ataca una araña y pierdes algo de salud:

health = 20  
health = health - 2

El valor de la salud en la declaración es ahora 18. Al igual que con la operación de suma, Python calcula el resultado de la operación a la derecha del signo igual y asigna a la variable ese valor.

¡Vamos a divertirnos con la resta en Minecraft!

#### **Misión nº 7: Cambia los bloques que tienes debajo**

¿Alguna vez has querido tender una trampa a alguien en Minecraft? Imagina que el suelo bajo el jugador cambia repentinamente a lava cuando menos se lo espera. Puedes utilizar Python para hacer realidad tu deseo. Utilizando la sustracción, puedes colocar bloques por debajo de la posición actual del jugador. De hecho, ¡sólo hacen falta unas pocas líneas de código para colocar cualquier bloque que quieras directamente debajo del jugador!

En esta misión, cambiarás el bloque que hay debajo del jugador por lava utilizando getTilePos() y setBlock(). Pero ésta es una misión peligrosa, así que ten cuidado cuando la pruebes: si no mueves al jugador a una nueva posición lo suficientemente rápido, ¡podría caer en la lava!

El programa del [Listado 3-5](ch03.xhtml#ch3ex5) crea un bloque en la posición actual del jugador. Copia este código en un nuevo archivo en IDLE y guárdalo como *bloqueBajo.py*. Luego, utilizando tus conocimientos sobre el operador de resta, cambia el código para que coloque un bloque de lava justo debajo de los pies del jugador, como se muestra en la [Figura 3-3](ch03.xhtml#ch3fig3).

*blockBelow.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
blockType = 10  
mc.setBlock(x, y, z, blockType)

*Listado 3-5: Este código coloca un bloque en la posición actual del jugador.*

Observa que he llamado pos a la variable que almacena la posición del jugador. Elegí este nombre porque utilizo mucho esta variable, es fácil de entender lo que significa el nombre y es más corto y rápido de escribir que posición.

La coordenada y determina lo alto o bajo que está un bloque. Tu misión es averiguar cómo cambiar la variable y para colocar un bloque debajo del jugador.



*Figura 3-3: Después de cambiar el bloque que estaba debajo de mí, caí en la lava.*

**OBJETIVO EXTRA: BLOQUES A TU ALREDEDOR**

Has aprendido a colocar bloques debajo del jugador. ¿Puedes averiguar cómo colocar un bloque por encima del jugador? Cuando hayas descubierto cómo hacerlo, intenta colocar varios bloques a la vez alrededor del jugador. ¡Entonces podrás empezar a crear edificios alrededor del jugador!

Intenta combinar este programa con el de la Misión nº 6[(página 55](ch03.xhtml#page_55)). ¿Puedes averiguar cómo hacer que el jugador salte por los aires y luego colocar un bloque inmediatamente debajo para que no se caiga? ¿Te sientes malvado? Podrías escribir un programa para que el jugador caiga desde una gran altura en un charco de lava.

#### **Uso de operadores matemáticos en argumentos**

Cuando utilizas una función, como setBlock() o setTilePos(), le das argumentos a la función, que especifican los valores que quieres que utilice la función cuando se ejecute.

Hasta ahora, has conocido los operadores de suma y resta. Puedes utilizar estos operadores dentro de los paréntesis de una función para establecer los valores de los argumentos. Volvamos a los bloques de apilamiento de la Misión nº 5[(página 53](ch03.xhtml#page_53)). Podemos utilizar el operador de suma dentro de los paréntesis de la función setBlock(), como se muestra en el Listado [3-6](ch03.xhtml#ch3ex6), y sumará dos valores dentro de los paréntesis sin necesidad de una declaración adicional.

*blockStack1.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
x = 6  
y = 5  
z = 28  
blockType = 103  
mc.setBlock(x, y, z, blockType)  
➊ mc.setBlock(x, y + 1, z, blockType)

*Listado 3-6: El programa de apilamiento de bloques con un operador en los argumentos*

El Listado[3-6](ch03.xhtml#ch3ex6) es prácticamente igual que el programa de apilamiento de bloques. Sin embargo, utiliza el operador de suma en el paréntesis de la función setBlock( ) en lugar de en una sentencia aparte. La última línea utiliza y + 1 como argumento en la función ➊. Aunque el valor de este argumento es 6(5 + 1), el valor de la variable y sigue siendo 5. El argumento te permite sumar a la variable y sin cambiar realmente su valor, lo que resulta útil si quieres volver a utilizar y en algún otro lugar de tu código.

También puedes sumar dos variables y utilizarlas como un único argumento. [El listado 3-7](ch03.xhtml#ch3ex7) es igual que [el listado 3-6](ch03.xhtml#ch3ex6), pero una variable adicional llamada up determina la distancia a la que se colocará el nuevo bloque en el eje y.

*blockStack2.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
x = 6  
y = 5  
z = 28  
blockType = 103  
up = 1  
mc.setBlock(x, y, z, blockType)  
➊ mc.setBlock(x, y + up, z, blockType)

*Listado 3-7: Otra versión del programa de apilamiento que utiliza un operador de suma en los argumentos*

En la última línea, se suman las variables y y up ➊. Como en el [Listado 3-6](ch03.xhtml#ch3ex6), esto hace que el segundo argumento de la función setBlock() sea 6. Las variables son útiles porque si quieres colocar el nuevo bloque dos bloques más arriba en el eje y, todo lo que tienes que hacer es cambiar tu código y poner arriba a 2. Puedes ver el efecto de las tres versiones[(Listados 3-1](ch03.xhtml#ch3ex1) y [3-2](ch03.xhtml#ch3ex2), [3-6](ch03.xhtml#ch3ex6) y [3-7](ch03.xhtml#ch3ex7)) del programa en [la](ch03.xhtml#ch3fig4) Figura 3-4.



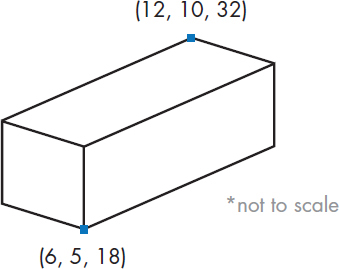
*Figura 3-4: Las tres versiones del programa tienen el mismo efecto, aunque sean diferentes.*

#### **Misión nº 8: Aumentar la velocidad**

Normalmente, pasas tu primer día en Minecraft construyendo un refugio. Con lo que has aprendido hasta ahora, ¡podrás construir una casa sencilla y pasar tu primera noche a lo grande! El programa de esta misión te ayudará a generar rápidamente las paredes, el techo y el suelo de un edificio. En lugar de pasar mucho tiempo colocando cada bloque a mano, puedes construir la estructura básica de tu edificio con unas pocas líneas de código.

Has utilizado setBlock () para crear un solo bloque, pero setBlock() tiene un amigo llamado setBlocks(), que crea varios bloques en forma de cuboide. Un *cuboide* es un rectángulo tridimensional. La longitud, anchura y altura de un cuboide pueden tener valores diferentes.

La función setBlocks () te permite crear muchos bloques en un área grande. Para utilizar setBlocks( ), sólo tienes que pasarle dos conjuntos de coordenadas y el tipo de bloque. El primer conjunto de coordenadas identifica dónde quieres una esquina del cuboide, y el segundo conjunto especifica dónde quieres la esquina opuesta. [La Figura 3-5](ch03.xhtml#ch3fig5) te muestra las esquinas del cuboide, etiquetadas con sus coordenadas.



*Figura 3-5: Un cuboide y las coordenadas utilizadas para fijar sus dimensiones*

Vamos a crear el cuboide de la [Figura 3-5](ch03.xhtml#ch3fig5). Como puedes ver en [el Listado 3-8](ch03.xhtml#ch3ex8), la mina está hecha de adoquines, pero utiliza cualquier tipo de bloque que desees. Bueno, puedes utilizar cualquier cosa excepto lava, agua o aire, ¡eso te daría una casa bastante extraña!

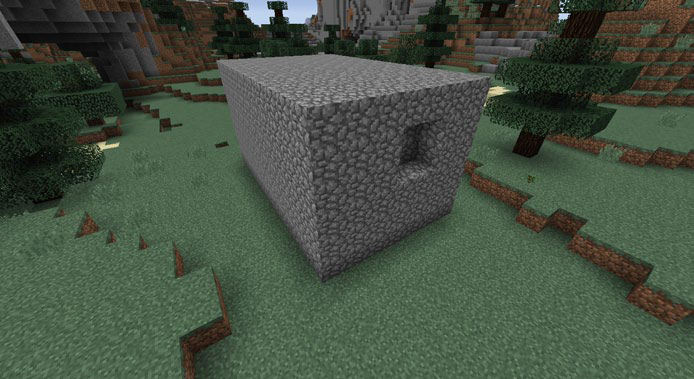
*building.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
➊ pos = mc.player.getPos()  
x = pos.x  
y = pos.y  
z = pos.z  
width = 10  
height = 5  
length = 6  
➋ blockType = 4  
➌ air = 0  
mc.setBlocks(x, y, z, x + width, y + height, z + length, blockType)

*Listado 3-8: Código que construye un cuboide de bloques*

Observa que he utilizado getPos( ) ➊ en lugar de getTilePos(). La función getPos () es igual que la función getTilePos(), pero devuelve las coordenadas como tres flotantes en lugar de tres enteros.

La anchura, la altura y la longitud de este cuboide son 10, 5 y 6 respectivamente, y he utilizado un ID de bloque de 4 para generar el adoquín ➋. Puedes ver el edificio terminado en la [Figura 3-6](ch03.xhtml#ch3fig6).



*Figura 3-6: El edificio que crea el programa*

Sin embargo, la casa tiene un pequeño problema: ¡es completamente maciza! Después de ejecutar el programa, hice un agujero en el lateral del edificio para que puedas ver que es sólido en el centro. Sin embargo, este cuboide es un buen comienzo, y ahora tú te encargarás de vaciarlo para que el jugador pueda realmente entrar en él.

Tu misión es modificar el programa para crear un edificio con paredes, techo y suelo en la posición del jugador. Para conseguirlo, crearás un cuboide hecho de aire dentro del cuboide sólido que acabas de crear. Los dos cuboides juntos deben producir una caja vacía. Puedes ver el resultado del programa terminado en la [Figura 3-7](ch03.xhtml#ch3fig7). He creado un agujero en el lateral para que puedas ver el centro hueco.



*Figura 3-7: Cuando tu programa esté terminado, debería crear un cuboide hueco. ¡Un cuboide es perfecto para crear edificios rápidamente!*

[El listado 3-8](ch03.xhtml#ch3ex8) ya incluye una variable llamada aire ➌, que puedes utilizar para que los bloques del interior del edificio sean aire. Copia [el Listado 3-8](ch03.xhtml#ch3ex8) en IDLE, guárdalo como *edificio.py*, y adáptalo para crear un segundo cuboide hecho de aire. Tendrás que añadir una función setBlocks() extra en la última línea para crear este cuboide de aire. El cuboide de aire debe estar un bloque dentro de las paredes en todo su perímetro, que es lo que tienes que averiguar cómo hacer utilizando la suma y la resta. Ten paciencia: si tu primer intento no funciona, ¡intenta otra cosa!

**CONSEJO**

*Para crear el cuboide de aire un bloque dentro de las paredes, puedes utilizar los operadores de suma y resta. Crea el cuboide de aire utilizando* setBlocks() *y aumenta* *en 1* *los primeros* *argumentos* x, y *y* z *. Luego resta 1 a los* *argumentos* x + anchura, y + altura *y* z + longitud *.*

**OBJETIVO EXTRA: CONSTRUIR TODO TIPO DE COSAS**

Puedes reutilizar el código que escribiste para crear el edificio siempre que quieras. ¿Y si quieres construir un edificio de otro tamaño? ¿Sabes cómo cambiar la anchura, la altura y la longitud del edificio?

Con unos pocos cambios, tu programa también puede tener muchos otros usos. ¿Puedes averiguar cómo utilizarlo para generar una piscina? Pista: Tendrás que cambiar el tipo de bloque del cuboide interior a agua (bloque ID 8) y quitar la parte superior del cuboide exterior para que el jugador pueda meterse en la piscina.

#### **Multiplicación**

En código Python, la multiplicación tiene un aspecto ligeramente distinto al que probablemente estés acostumbrado a ver. En lugar de escribir una × para multiplicar dos números, se utiliza un asterisco(\*). Pero aparte del símbolo, la multiplicación funciona igual que siempre. La expresión 2 \* 2 es igual a 4, igual que 2 × 2.

Imagina que hay cuatro árboles fuera de tu casa de Minecraft, y de repente el número de árboles se duplica. Puedes representar este cálculo en Python así:

trees = 4  
trees = trees \* 2

El valor de árboles en este ejemplo es 8, que es 4 multiplicado por 2.

#### **División**

En Python, la división se representa con una barra oblicua(/) en lugar del símbolo ÷.

El operador división divide un valor por otro. Sólo tienes que poner el número que quieres dividir a la izquierda de la barra oblicua (/) y el número por el que quieres dividir a la derecha.

Supongamos que hay ocho esqueletos fuera de tu fortaleza de Minecraft, pero la mitad de ellos se alejan. Para saber cuántos quedan, dividirías 8 entre 2. Así es como lo representarías utilizando un operador de división en Python:

skeletons = 8  
skeletons = skeletons / 2

Ahora sólo quedan 4 esqueletos fuera de tu fortaleza. ¡Uf! Probemos estos dos operadores en Minecraft.

#### **Misión 9: Espirales espectaculares**

Uno de los puntos fuertes de las variables es que puedes cambiar el valor de una sola variable, y su valor cambiará en todas las partes del programa donde aparezca la variable. Puedes hacer que un programa haga cosas totalmente distintas con sólo cambiar una única variable utilizando operadores matemáticos, como la multiplicación y la división.

En esta misión, descubrirás cómo utilizar la división y la multiplicación para construir una torre muy alta y delgada, también llamada *aguja*.

[La Figura 3-8](ch03.xhtml#ch3fig8) muestra el aspecto que tendrá la aguja una vez finalizado el programa.



*Figura 3-8: Una aguja creada a partir de bloques de piedra*

El programa utilizará una única variable para establecer la altura de la aguja. Utilizando la multiplicación y la división, establecerás diferentes alturas de chapitel.

En el Listado [3-9](ch03.xhtml#ch3ex9), he iniciado el programa que crea la aguja, pero no he utilizado la variable altura ni los operadores matemáticos para establecer la altura de cada parte.

*spire.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
height = 2  
blockType = 1  
  
# Spire sides: should be same as height  
sideHeight = height  
mc.setBlocks(x + 1, y, z + 1, x + 3, y + sideHeight - 1, z + 3, blockType)  
  
# Spire point: should be two times the height  
➊ pointHeight = 4  
mc.setBlocks(x + 2, y, z + 2, x + 2, y + pointHeight - 1, z + 2, blockType)  
  
# Spire base: should be half the height  
➋ baseHeight = 1  
mc.setBlocks(x, y, z, x + 4, y + baseHeight - 1, z + 4, blockType)

*Listado 3-9: El programa de construcción de la aguja*

Copia [el Listado 3-9](ch03.xhtml#ch3ex9) en un nuevo archivo en IDLE y guárdalo como *aguja.py* en la carpeta de *matemáticas*. Este programa creará una aguja, pero cambiar la variable altura y volver a ejecutar el programa no afectará a la altura de todas las partes de la aguja.

Para arreglar este programa de modo que todas las partes de la altura de la aguja cambien cuando cambies la variable altura, tienes que cambiar las variables alturapunto ➊ y alturabase ➋ para incluir expresiones que utilicen la variable altura y operadores de multiplicación o división. Quieres que pointHeight sea el doble de la altura y baseHeight la mitad de la altura. Por ejemplo, si quisiera que el punto de la aguja fuera tres veces la altura de los lados de la aguja, cambiaría el código a pointHeight = altura \* 3 y baseHeight = altura / 2.

Después de hacer estos cambios, cuando cambies la variable altura, todas las partes de la aguja cambiarán de tamaño.

No necesitas cambiar ninguna otra parte del programa.

Puedes probar el programa cambiando la variable de altura original y volviendo a ejecutarlo. Si cambias la variable altura a 3, tu aguja tendrá el aspecto de [la Figura 3-9](ch03.xhtml#ch3fig9).



*Figura 3-9: Puedes hacer que la aguja sea más alta cambiando la* variable altura *.*

Como utilizas la variable altura para establecer los valores de la altura del punto y la altura de la base, es muy fácil cambiar la aguja. Juega con este código cambiando la variable altura original por algunos números diferentes. ¡Vuelve a ejecutar el programa cada vez para ver qué ocurre!

### **Exponentes**

Puedes utilizar un *exponente* para indicar que un número debe multiplicarse por sí mismo un determinado número de veces. Por ejemplo,34 (tres a la potencia de cuatro) es una forma abreviada de decir 3 \* 3 \* 3 \* 3.

En Python, \*\* es el operador exponencial. El número que quieres multiplicar (la *base*) va a la izquierda del operador, y el número de veces que quieres multiplicarlo por sí mismo (el *exponente*) va a la derecha.

Supongamos que quieres montar una granja en Minecraft. Necesitas labrar cuatro parcelas de tierra. Quieres que cada parcela tenga cuatro manzanas por cuatro manzanas para poder cultivar mucho trigo. Matemáticamente, puedes escribir esto como 4 \* 4 \* 4, o43. Aquí tienes el código para calcular cuánto trigo estás cultivando:

wheat = 4 \*\* 3

Tu respuesta debe ser 64 parcelas individuales de trigo, porque 4 \* 4 es 16 y 16 \* 4 es 64.

### **Paréntesis y orden de las operaciones**

Cuando utilizas varios operadores matemáticos en una expresión, debes tener cuidado con cómo los ordenas. Los distintos operadores tienen distinta *prioridad.* Cuando utilizas varios operadores, la división y la multiplicación se evalúan primero de izquierda a derecha, y luego se calculan la suma y la resta. Veamos cómo se evalúa esta expresión:

mooshroom = 5 \* 2 – 1 + 4 / 2

Como la multiplicación y la división siempre ocurren antes que la suma y la resta, Python empieza por la izquierda multiplicando 5 por 2 para obtener 10, y luego divide 4 por 2 para obtener 2. Esto nos da 10 - 1 + 2. A continuación, Python vuelve a empezar por la izquierda restando 1 a 10, y luego le suma 2, con lo que mooshroom obtiene 11.

Pero puedes controlar el orden de las operaciones utilizando paréntesis. Las expresiones con operadores entre paréntesis evaluarán las operaciones entre paréntesis antes que cualquier otra cosa. Veamos cómo los paréntesis cambian el orden de las operaciones. Para empezar, aquí tienes una expresión que no utiliza ningún paréntesis:

zombiePigmen = 6 \* 3 - 2

Escrita así, zombiePigmen acaba teniendo un valor de 16, porque 6 por 3 es 18, y 18 menos 2 es 16. Sin embargo, con los paréntesis, el resultado cambia:

zombiePigmen = 6 \* (3 - 2)

¡ZombiePigmen tiene ahora un valor de 6! En lugar de seguir el orden habitual, primero Python resta 2 a 3, lo que da como resultado 1, y luego multiplica 6 por 1 para obtener 6.

Cuando quieras que un cálculo se realice en un orden determinado, utiliza paréntesis para decirle a Python qué debe hacer primero. Esto te da aún más control sobre Python.

### **Prácticos trucos matemáticos**

En las siguientes secciones, te enseñaré dos habilidades matemáticas más para que mejores tu programación en Python, y luego combinaremos lo que has aprendido hasta ahora en una misión más.

#### **Operadores abreviados**

Muy a menudo, querrás utilizar un operador en una variable y luego almacenar el resultado en la misma variable. Por ejemplo, puede que quieras añadir cinco ovejas a tu rebaño actual:

sheep = 6  
sheep = sheep + 5

Pero escribir oveja = oveja + 5 probablemente te cansará después de un rato. No te preocupes, ¡hay una forma más corta! Python tiene operadores *abreviados* que te permiten utilizar un operador matemático en una variable y reasignar el resultado a la misma variable. Aquí tienes los cuatro operadores abreviados:

- Suma(+=)

- Resta(-=)

- Multiplicación(\*=)

- División(/=)

Por ejemplo, puedes reescribir el ejemplo de la oveja utilizando el operador abreviado de suma:

sheep = 6  
sheep += 5

El valor de oveja es igual a 11, igual que antes.

#### **Jugar con números aleatorios**

Utilizar números aleatorios es una forma de añadir algo de misterio y diversión a tus programas. ¡Nunca sabes con qué acabarás! Muchos juegos de mesa del mundo real se basan en números aleatorios: piensa en cuántos juegos has jugado en los que tenías que tirar los dados para ver cuántos espacios podías mover. Una tirada de dados es un ejemplo clásico de números aleatorios en acción.

Python puede generar números aleatorios por ti fácilmente, así que vamos a simular la tirada de un dado. El número generado debe estar entre 1 y 6:

➊ import random  
➋ diceValue = random.randint(1, 6)

Cuando quieras crear números aleatorios, asegúrate de incluir import random ➊ al principio de tu programa. La función randint() ➋ genera un valor entero que el programa puede utilizar como cualquier otro número. Los números que colocas dentro de los paréntesis como argumentos indican a randint () que genere valores entre el primer número y el segundo. En este ejemplo, el número generado puede tener un valor de 1, 2, 3, 4, 5 ó 6.

Puedes utilizar randint () para añadir un número aleatorio al valor de una variable, e incluso puedes generar números negativos. ¡Veamos cómo!

import random  
score = 0  
score += random.randint(0, 99)  
points = random.randint(-99, 99)

El número más bajo que Python puede generar para añadir a la puntuación es 0, y el más alto es 99. Por otra parte, debido al argumento negativo, ¡los puntos podrían establecerse en un número tan bajo como -99!

#### **Misión nº 10: ¡Super Salta a algún sitio nuevo!**

En la última misión de este capítulo, harás que el jugador salte una distancia aleatoria en los ejes x, y y z almacenando la posición actual del jugador y añadiendo después un número aleatorio a cada una de las tres coordenadas. Utiliza los números aleatorios entre -10 y 10 para cambiar los valores x y z, y asegúrate de que los valores aleatorios para y están entre 0 y 10.

Para empezar, copia [el Listado 3-10](ch03.xhtml#ch3ex10) en un nuevo archivo en IDLE y guárdalo como *randomJump.py*.

*randomJump.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
import random  
  
pos = mc.player.getPos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➊ x = x + random.randint(-10, 10)  
mc.player.setPos(x, y, z)

*Listado 3-10: El programa de salto aleatorio incompleto*

Falta el código de generación de números aleatorios para las variables y y z, y depende de ti añadirlo. Una vez que lo hagas, el jugador tendrá la posibilidad de saltar a cualquier parte, como hice yo en la [Figura 3](ch03.xhtml#ch3fig10)-10. ¡Deja que el azar te lleve a lugares nuevos y emocionantes!

De momento, el código no utiliza operadores abreviados para cambiar los valores de las variables. Prueba a cambiar la expresión de suma en ➊ para que utilice taquigrafía en su lugar.



*Figura 3-10: He saltado en una dirección aleatoria y he acabado en lo alto de este árbol. ¿Dónde has acabado tú?*

**OBJETIVO EXTRA: TELETRANSPORTE DE BLOQUES ALEATORIOS**

¡Hagamos que el programa *randomJump.* py sea aún más aleatorio! Después de que el jugador salte a una posición aleatoria, coloca un bloque aleatorio debajo de él. También podrías adaptar el programa de recorrido de teletransporte de la Misión 1 (ver *teletransporte*.py en [la página 38](ch02.xhtml#page_38)) para que el jugador se teletransporte a una posición aleatoria cada vez. Si accidentalmente te teletransportas a algún lugar y te quedas atascado, siempre puedes volver a ejecutar teleport *.*py, que debería teletransportarte a un lugar seguro.

### **Lo que has aprendido**

En este capítulo has aprendido a hacer operaciones matemáticas en Python. Utilizarás la suma, la resta, la multiplicación y la división con frecuencia en los programas Python del resto de este libro y en los programas que crees en el futuro. También has aprendido a generar números aleatorios y, de paso, has creado algunos programas muy útiles en Minecraft. ¡Buen trabajo!

En [el Capítulo 4](ch04.xhtml#ch04), aprenderás sobre el tipo de datos cadena en Python, que se utiliza para contener letras, símbolos y números. Las cadenas son muy útiles en Minecraft, porque puedes utilizarlas para enviar mensajes al chat de Minecraft. También explorarás las cadenas para manipular el chat de Minecraft y hacer otras cosas chulas.