Aprende a programar con Minecraft

### **Bucles for anidados y listas multidimensionales**

Dentro de tus programas, puedes utilizar varias listas juntas por diversas razones. Es posible incluir listas dentro de listas, que se denominan listas *multidimensionales*. En esta sección, utilizaremos listas bidimensionales (2D) y tridimensionales (3D) para construir estructuras en Minecraft.

#### **Pensar en dos dimensiones**

Ya has aprendido a escribir listas, concretamente listas *unidimensionales*. Se llaman listas unidimensionales porque cada posición de la lista contiene un solo elemento.

Por ejemplo, mira la siguiente lista, llamada lista unidimensionalArcoiris. El formato de esta lista es un poco diferente sólo para destacar que cada posición contiene un único elemento; por lo demás, es igual que otras listas con las que hayas trabajado:

oneDimensionalRainbowList = [0,  
1,  
2,  
3,  
4,  
5]

Hay seis elementos en esta lista: los números del 0 al 5. Cada elemento de la lista tiene un único valor, por lo que la lista es unidimensional.

[El listado 10-5](ch10.xhtml#ch10ex5) muestra esta lista en Minecraft como bloques de lana. El archivo del programa, *rainbowStack1.py*, está disponible en los recursos del libro. Descarga los archivos de código de [*https://www.nostarch.com/programwithminecraft/*](https://www.nostarch.com/programwithminecraft/) o escríbelo tú mismo y ¡juega!

*rainbowStack1.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ oneDimensionalRainbowList = [0, 1, 2, 3, 4, 5]  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➋ for color in oneDimensionalRainbowList:  
mc.setBlock(x, y, z, 35, color)  
y += 1

*Listado 10-5: Construir una pila arco iris de bloques*

El programa crea una lista de colores de bloques ➊ y luego utiliza un bucle for para crear una pila de bloques de lana con colores basados en los colores de la lista ➋.

Cuando ejecutes el programa, obtendrás una única pila de bloques de lana, como puedes ver en la [Figura 10-7](ch10.xhtml#ch10fig7). Observa que la pila tiene seis bloques de alto y uno de ancho. Has utilizado las variables x, y y z a lo largo de este libro. Cada una de estas variables también puede denominarse *dimensión*. Este programa crea una pila de seis bloques en la dimensión y. Cambiando la variable x en la última línea del código en lugar de la variable y, puedes construir una pila de bloques en la dimensión x, que puedes ver en la [Figura 10-8](ch10.xhtml#ch10fig8).



Figura 10-7:*La pila arco iris de bloques creada por* rainbowStack1.py



Figura*10-8: Cambiando la* *variable* y *por la* *variable* x *en la última línea del programa se construyen los bloques en una fila horizontal.*

Como la lista es unidimensional, sólo puedes cambiar el valor de una variable de una dimensión cada vez. En otras palabras, puedes cambiar el valor de la variable y, de la variable x o de la variable z, pero no puedes cambiarlas todas a la vez.

Así que, ¡es hora de empezar a pensar en dos dimensiones! Las listas unidimensionales te permiten tener una sola lista con un solo valor en cada posición, pero las listas bidimensionales te permiten tener muchos valores en cada posición de una lista. Para ello, coloca una lista en cada posición de la lista original, como se indica a continuación.

➊ twoDimensionalRainbowList = [[0, 0, 0],  
➋ [1, 1, 1],  
➌ [2, 2, 2],  
[3, 3, 3],  
[4, 4, 4],  
➍ [5, 5, 5]]

Fíjate bien y verás un corchete de apertura en la primera línea, seguido de una lista llena de ceros y luego una coma ➊. ¡Eso es una lista dentro de otra lista! Podemos llamar *lista exterior* a la lista principal y decir que contiene *listas anidadas*.

En la posición de índice 1 hay una lista que contiene tres 1 ➋. En la posición de índice 2 hay otra lista, que contiene tres 2 ➌. Esto se repite en cada línea. En la última línea hay una lista de tres 5s, seguida de un corchete, que cierra la lista exterior ➍. Este código muestra una lista con seis elementos, cada uno de los cuales es también una lista. ¡Se trata de una lista bidimensional!

Entenderás mejor las listas bidimensionales cuando las utilices en Minecraft. Veamos un ejemplo. Modificando *rainbowStack1.py*, podemos hacer que funcione con la lista bidimensional. Este nuevo programa se llama *rainbowRows.py*:

*rainbowRows.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
twoDimensionalRainbowList = [[0, 0, 0],  
[1, 1, 1],  
[2, 2, 2],  
[3, 3, 3],  
[4, 4, 4],  
[5, 5, 5]]  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➊ startingX = x  
  
➋ for row in twoDimensionalRainbowList:  
➌ for color in row:  
➍ mc.setBlock(x, y, z, 35, color)  
➎ x += 1  
➏ y += 1  
➐ x = startingX

Antes de explicar el código, mira la [Figura 10-9](ch10.xhtml#ch10fig9) para ver la salida de *rainbowRows*.py, que es un conjunto de bloques de seis bloques de alto en la dimensión y y tres bloques de ancho en la dimensión x.



*Figura 10-9: Uso de una lista bidimensional para hacer un muro arco iris*

Como estamos trabajando con dos dimensiones, necesitamos dos bucles for para dar salida a los valores de la lista bidimensionalRainbowList. El primer bucle itera por cada elemento de la lista exterior ➋. El segundo bucle ➌, llamado bucle *anidado* porque está dentro de otro bucle, recorre cada elemento de cada lista anidada.

Por ejemplo, la primera vez que se ejecuta el bucle externo, obtiene el elemento almacenado en la posición índice 0 de la lista bidimensionalRainbowList y lo almacena en una variable llamada fila ➋. El valor de row es [0, 0, 0] porque es el primer elemento de la lista.

A continuación, el segundo bucle recorre cada elemento de la lista fila y lo almacena en la variable color ➌. En este caso, cada elemento será 0. A continuación, el programa coloca los bloques utilizando la variable de color para determinar el color de cada bloque de lana ➍. El bucle anidado finaliza cuando ha colocado los tres bloques de esa fila, y entonces el bucle exterior se ejecuta de nuevo. A continuación, el bucle externo se desplaza a la posición de índice 1 y almacena el valor en la variable de fila, que ahora es [1, 1, 1]. A continuación, ejecuta de nuevo el bucle anidado para establecer los bloques e itera de nuevo hasta llegar al final de la lista TwoDimensionalRainbowList.

Cuando trabajas en dos dimensiones, puedes cambiar dos variables de coordenadas al mismo tiempo. En este ejemplo, incrementamos la variable y en la penúltima línea del bucle for exterior ➏ para que cada fila de bloques se coloque encima de la fila anterior. También incrementamos la variable x dentro del bucle for anidado ➎ para asegurarnos de que los bloques se colocan en fila. Luego tenemos que restablecer la variable x a su valor original (que se almacena en la variable startingX ➊) cada vez que itere el bucle for exterior ➐. Restablecer la variable x hace que el primer bloque de cada fila se coloque directamente encima del primer bloque de la fila anterior, y así sucesivamente, para que las filas se alineen correctamente entre sí.

#### **Acceder a valores en listas 2D**

Cuando obtengas o establezcas un valor en una lista unidimensional, utiliza corchetes y la posición del índice. Por ejemplo, este código crea una lista llamada puntuaciones que registra las puntuaciones de un jugador, y luego cambia el elemento en la posición índice 2 de 6 a 7:

scores = [1, 5, 6, 1]  
scores[2] = 7

Utilizar o cambiar valores en una lista bidimensional no es muy diferente. Sigues utilizando corchetes y la posición del índice, pero como estás accediendo a dos listas al mismo tiempo, utilizas dos conjuntos de índices y corchetes. ¡Echemos un vistazo!

Aquí está la lista que has visto antes:

twoDimensionalRainbowList = [[0, 0, 0],  
[1, 1, 1},  
[2, 2, 2],  
[3, 3, 3],  
[4, 4, 4],  
[5, 5, 5]]

Si quisiéramos cambiar el segundo elemento (posición de índice 1) de la primera lista (posición de índice 0) al valor 7, utilizaríamos este código:

twoDimensionalRainbowList[0][1] = 7

Como estamos utilizando dos listas y una lista está anidada dentro de la otra, necesitamos utilizar dos conjuntos de corchetes. El primero recoge la posición 0 del índice de la lista TwoDimensionalRainbowList, que es su primera lista anidada. En el segundo corchete ponemos la posición de índice a la que queremos acceder en la lista anidada, 1. A continuación, establecemos el valor de esta posición en 7 utilizando el signo igual.

Añadí este código al programa *rainbowRows.py*[(página 215](ch10.xhtml#page_215)) y lo volví a ejecutar. [La Figura 10-10](ch10.xhtml#ch10fig10) muestra el resultado. Observa que el segundo bloque de la primera fila ha cambiado porque hemos cambiado el valor de la lista anidada a 7.

Si quisieras obtener el valor de un elemento de una lista bidimensional, también utilizarías dos conjuntos de corchetes. Por ejemplo, si quisieras imprimir el valor de la primera posición (índice 0) de la última fila (índice 5), utilizarías este código:

print(twoDimensionalRainbowList[5][0])

Este código imprime el valor 5.



*Figura 10-10: Cambiar uno de los valores de una lista anidada para obtener un resultado diferente*

#### **Misión nº 61: Arte del píxel**

*Los píxeles* son cuadrados de un solo color que forman las imágenes de tu ordenador. Combinando montones de píxeles en una cuadrícula, tu ordenador puede mostrar texto, imágenes, vídeos y todo lo que aparece en tu monitor. Todas las fotos y dibujos de tu ordenador se muestran utilizando píxeles.

El arte de los píxeles es muy popular en Minecraft. Utilizando bloques de diferentes colores en el juego Minecraft, los jugadores construyen imágenes en Minecraft. Los dibujos de personajes de videojuegos en 2D son algunos de los más populares. Puedes crear pixel art a mano, o por supuesto, puedes utilizar un programa Python para generar el pixel art.

En este programa, utilizarás una lista 2D y bucles anidados para crear pixel art en Minecraft. [El listado 10-6](ch10.xhtml#ch10ex6) contiene el principio del programa. Cópialo en un nuevo archivo llamado *pixelArt.py* y guárdalo en la carpeta *forLoops*.

*pixelArt.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
pos = mc.player.getTilePos()  
x, y, z = pos.x, pos.y, pos.z  
  
➊ blocks = [[35, 35, 35, 35, 35, 35, 35, 35],  
[35, 35, 35, 35, 35, 35, 35, 35],  
[35, 35, 35, 35, 35, 35, 35, 35],  
[35, 35, 35, 35, 35, 35, 35, 35]]  
➋ for row in reversed(blocks):  
for block in row:  
mc.setBlock(x, y, z, block)  
x += 1  
y += 1  
x = pos.x

*Listado 10-6: Una lista bidimensional que dibuja una cara sonriente*

El programa crea una lista bidimensional llamada bloques que contiene IDs de bloques ➊ y luego utiliza dos bucles para establecer los bloques en el mundo Minecraft ➋. Para asegurarte de que la primera fila de la lista está en la parte superior cuando se coloca en Minecraft y la fila inferior de la lista se coloca en la parte inferior, se incluye la función reversed() con el primer bucle for ➋. Si no fuera así, la imagen estaría al revés en comparación con el orden de la lista de bloques.

De momento, los bloques son todos bloques de lana blanca y no muestran ninguna imagen. Para terminar el programa, tienes que reescribir la lista de bloques bidimensional para que dibuje una cara sonriente, como se muestra en la Figura [10-11](ch10.xhtml#ch10fig11).



*Figura 10-11: Una cara sonriente dibujada con bloques*

Cambia los valores dentro de las listas para que la salida coincida con la Figura [10-11](ch10.xhtml#ch10fig11). Tendrás que cambiar algunos de los valores de las listas de bloques de lana (ID de bloque 35) a bloques de lapislázuli (ID de bloque 22). Por ejemplo, cambia la primera línea por ésta:

blocks = [[35, 35, 22, 22, 22, 22, 35, 35],

También tendrás que añadir más filas a la lista de bloques para que la altura de la imagen coincida con la del dibujo.

**OBJETIVO EXTRA: DIBÚJALO TÚ MISMO**

Prueba a cambiar los valores de la lista bidimensional en *pixelArt.* py para mostrar una imagen diferente. También puedes cambiar las longitudes de las listas. Esboza primero tus diseños en papel cuadriculado. Luego conviértelos en listas bidimensionales para poder crearlas en Minecraft.

#### **Generar listas bidimensionales con bucles**

Los programas que utilizan números aleatorios son divertidos porque se comportan de forma diferente cada vez que los ejecutas. En el pasado, he creado muchos programas que utilizan números aleatorios en listas bidimensionales para crear imágenes. Cada número aleatorio puede mostrar un color o, en el caso de Minecraft, un bloque diferente.

Aquí tienes el principio de un programa que genera números aleatorios y los almacena en un conjunto bidimensional de listas:

import random  
➊ randomNumbers = []  
for outer in range(10):  
➋ randomNumbers.append([])  
for inner in range(10):  
➌ number = random.randint(1, 4)  
randomNumbers[outer].append(number)  
print(randomNumbers)

El programa comienza con una lista vacía llamada NúmerosAleatorios ➊. Cada vez que se repite el bucle for exterior, añade una nueva lista vacía a la lista NúmerosAleatorios ➋. En el bucle interno, el programa genera un número aleatorio entre 1 y 5 y lo almacena en la lista interna ➌. El bucle interno se repite 10 veces para generar 10 elementos en cada lista interna.

Después de añadir saltos de línea para facilitar la lectura, la salida del programa tiene este aspecto (fíjate en los 10 elementos de las 10 listas internas):

[[3, 1, 4, 1, 4, 1, 2, 3, 2, 2],  
[1, 3, 4, 2, 4, 3, 4, 1, 3, 2],  
[4, 2, 4, 1, 4, 3, 2, 3, 4, 4],  
[1, 4, 3, 4, 3, 4, 3, 3, 4, 4],  
[3, 1, 4, 2, 3, 3, 3, 1, 4, 2],  
[4, 1, 4, 2, 3, 2, 4, 3, 3, 1],  
[2, 4, 2, 1, 2, 1, 4, 2, 4, 3],  
[3, 1, 3, 4, 1, 4, 2, 2, 4, 1],  
[4, 3, 1, 2, 4, 2, 2, 3, 1, 2],  
[3, 1, 3, 3, 1, 3, 1, 4, 1, 2]]

Incorporando números aleatorios a tus creaciones 2D de Minecraft, ¡puedes crear efectos muy chulos que serían difíciles de hacer a mano!

#### **Misión nº 62: Un muro desgastado por la intemperie**

Cuando construyo muros en Minecraft, no uso un único tipo de bloque. Cambiando algunos bloques de adoquines por bloques de adoquines musgosos, puedo convertir un muro liso en un muro con aspecto dañado, desgastado por la intemperie, orgánico y chulo. Por muy divertido que sea construir un muro a mano, nunca consigo que los bloques que he añadido al azar tengan un aspecto suficientemente aleatorio. Probablemente habrás adivinado que la solución para que los muros rotos parezcan más aleatorios es utilizar un programa Python.

Para generar un muro desgastado con Python, tienes que dividir el programa en dos pasos principales:

1. Crear una lista bidimensional y almacenar los valores de los bloques en la lista.
2. Dar salida a la lista bidimensional en el mundo Minecraft.

Para empezar, [el Listado 10-7](ch10.xhtml#ch10ex7) incluye el código para elegir un valor de bloque aleatorio, configurar la lista y obtener la posición del jugador. Copia el listado en un nuevo archivo llamado *brokenWall.py* y guárdalo en la carpeta *forLoops*.

*brokenWall.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import random  
  
  
➊ def brokenBlock():  
brokenBlocks = [48, 67, 4, 4, 4, 4]  
block = random.choice(brokenBlocks)  
return block  
  
pos = mc.player.getTilePos()  
x, y, z = pos.x, pos.y, pos.z  
  
brokenWall = []  
height, width = 5, 10  
  
# Create the list of broken blocks  
  
# Set the blocks

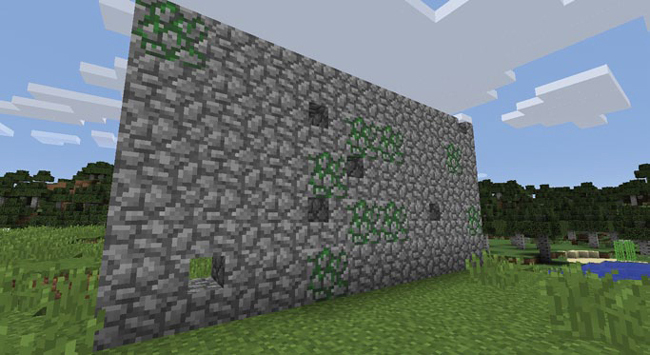
*Listado 10-7: Inicio del programa para crear un muro roto*

La función brokenBlock( ) devuelve un valor de bloque aleatorio que se utiliza para construir el muro ➊. Las variables anchura y altura establecen la anchura y la altura del muro.

Para terminar el programa, tienes que generar una lista bidimensional de valores de bloques, y luego utilizar esos valores para construir el diseño en Minecraft.

Empieza con la lista en blanco brokenWall. Utilizando un bucle for anidado dentro de otro bucle for, genera valores de bloque aleatorios con la función brokenBlock(). Almacena los valores de bloque en listas, y guarda esas listas en la lista brokenWall. A continuación, utiliza otro conjunto de bucles anidados para colocar los bloques en Minecraft.

Cuando tu programa esté completo, desplázate al lugar de tu mundo Minecraft donde quieras construir tu muro desgastado por el tiempo y ejecuta el código. Puedes utilizar el programa para decorar un castillo o crear ruinas de aspecto espeluznante en el bosque. ¡Experimenta con distintas ubicaciones para ver cuál te gusta más! [La Figura 10-12](ch10.xhtml#ch10fig12) muestra el aspecto que tendrá un muro cuando ejecutes el programa.



*Figura 10-12: Un muro con bloques rotos generados aleatoriamente. ¡Parece que podría estar embrujado!*

**OBJETIVO EXTRA: CREA UN MURO COLORIDO**

En el programa *brokenWall.py*, cambia los valores de los bloques en la lista brokenBlock () de la función brokenBlocks para crear todo tipo de muros. Prueba a cambiar los valores de los bloques por diferentes colores de lana y ¡a ver qué pasa!

#### **Pensar en tres dimensiones**

Por supuesto, Minecraft es un juego que utiliza tres dimensiones. Y has utilizado tres dimensiones a lo largo de este libro. Cada una de las variables x, y y z que has utilizado en la mayoría de los programas representa una dimensión.

Has visto cómo poner un grupo de listas dentro de otro para obtener una lista bidimensional y crear un genial arte de píxeles y paredes desgastadas. Poniendo un tercer grupo de listas dentro de una lista bidimensional se crea una lista tridimensional, ¡que te permite llevar tus habilidades de construcción a una dimensión completamente nueva!

Las listas tridimensionales son extremadamente útiles en Minecraft porque puedes usarlas para duplicar estructuras 3D, como edificios, esculturas y muchas otras cosas.

La lista tridimensional del [Listado 10-8](ch10.xhtml#ch10ex8) tiene cuatro listas anidadas en su interior. ¡Lo bueno es que dentro de cada índice de esas listas anidadas hay otra lista! Básicamente, cada elemento de esta lista es una lista 2D. He añadido comentarios en blanco para facilitar la lectura de la lista.

cube = [[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]],  
#  
[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]],  
#  
[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]],  
#  
[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]]]

*Listado 10-8: Una lista tridimensional con listas anidadas*

¡Un código como éste puede utilizarse para crear una estructura cúbica muy chula! A continuación veremos un programa que hace precisamente eso.

#### **Cómo crear listas tridimensionales**

Las listas que tienen tres dimensiones son perfectas para almacenar datos sobre objetos tridimensionales, como tus impresionantes edificios de Minecraft. Almacenar objetos tridimensionales es importante, y darles salida correctamente en Minecraft es igual de importante. Como una lista tridimensional es una lista dentro de otra lista que está dentro de otra lista, puedes utilizar un bucle for dentro de otro bucle for que también está dentro de otro bucle for para acceder a todos los datos. En otras palabras, puedes utilizar tres bucles for anidados.

En [el Listado 10-9](ch10.xhtml#ch10ex9), he copiado la lista tridimensional del [Listado 10-8](ch10.xhtml#ch10ex8) y he creado un programa llamado *cubo.py*. Este programa utiliza tres bucles for anidados para sacar todos los valores de la lista tridimensional de uno en uno para construir una estructura cúbica en el mundo de Minecraft.

*cube.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
cube = [[[57, 57, 57, 57], [57, 0, 0, 57], [57, 0, 0, 57], [57, 57, 57, 57]],  
[[57, 0, 0, 57], [0, 0, 0, 0], [0, 0, 0, 0], [57, 0, 0, 57]],  
[[57, 0, 0, 57], [0, 0, 0, 0], [0, 0, 0, 0], [57, 0, 0, 57]],  
[[57, 57, 57, 57], [57, 0, 0, 57], [57, 0, 0, 57], [57, 57, 57, 57]]]  
  
startingX = x  
➊ startingY = y  
➋ for depth in cube:  
for height in reversed(depth):  
for block in height:  
mc.setBlock(x, y, z, block)  
x += 1  
y += 1  
x = startingX  
➌ z += 1  
➍ y = startingY

*Listado 10-9: Código para crear un cubo tridimensional hecho de diamantes*

[La Figura 10-13](ch10.xhtml#ch10fig13) muestra el resultado de este programa.



*Figura 10-13: El cubo creado por el* *programa* cube.py

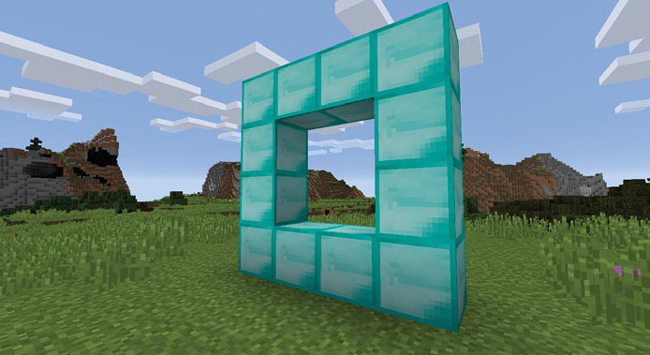
El código de cube. *py* es muy similar al del programa bidimensional *rainbowRows.py*[(página 215](ch10.xhtml#page_215)) que construye un muro arco iris. La principal diferencia es que cubo. *py* utiliza tres bucles for en lugar de dos, porque trabaja con una lista tridimensional. El bucle for adicional añade una dimensión más a la estructura, la profundidad ➋. Así que ahora la estructura tiene anchura, altura y profundidad.

Cada vez que se ejecuta el bucle exterior para profundidad en cubo, crea una lista bidimensional utilizando los dos bucles anidados, para altura en invertido(profundidad) y para bloque en altura. El código de los dos bucles anidados es similar al código del programa *rainbowRows.py*, lo que significa que estos bucles construyen un muro en Minecraft.

Veamos el resultado del bucle exterior cada vez que se repite para que podamos ver cómo construye el cubo paso a paso. La primera vez que se ejecuta el bucle exterior, emite los bloques en la posición índice 0 de la lista del cubo. Esa lista tiene este aspecto:

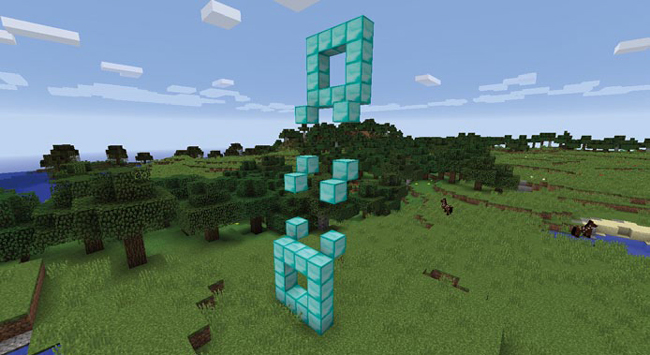
[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]]

[La Figura 10-14](ch10.xhtml#ch10fig14) muestra la salida: nuestro primer muro de bloques.



*Figura 10-14:* Resultado*del primer bucle bidimensional,* *índice* 0 del cubo

Después de construir cada lista bidimensional en el juego, el valor de la variable z en *cube.py* ➌ se incrementa para mover un bloque más lejos a lo largo del eje z. Esto da profundidad al cubo, por lo que no sólo estamos construyendo un muro. También tenemos que restablecer el valor de la variable y en ➍ a su valor original ➊ para que los bloques de la parte inferior del cubo se alineen entre sí cada vez que se repita el bucle exterior. Si no se reseteara la variable y, la coordenada y de cada conjunto de bloques seguiría subiendo cada vez más, ¡creando unas escaleras de aspecto extraño! [La Figura 10-15](ch10.xhtml#ch10fig15) muestra el aspecto que tendría esto.

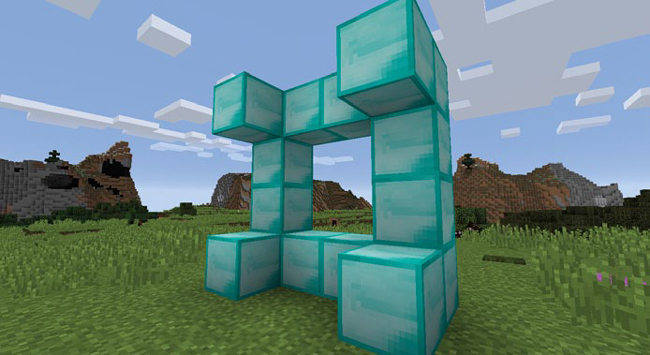


*Figura 10-15:* ¡*Reiniciamos la* *variable y para que esto no ocurra!*

La segunda vez que se ejecuta el bucle exterior, da salida a los bloques en la posición índice 1 de la lista del cubo, que tiene este aspecto:

[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]]

Esto añade la siguiente parte del cubo, como puedes ver en la [Figura 10-16](ch10.xhtml#ch10fig16). Una vez construida esta parte del cubo, la variable z aumenta en 1 ➌ y la variable y vuelve a su valor original ➍.



*Figura 10-16: Resultado del segundo bucle bidimensional,* *índice del* cubo *1*

La siguiente vez que se repite el bucle, sale la lista bidimensional en la posición índice 2 del cubo:

[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]]

La Figura[10-17](ch10.xhtml#ch10fig17) muestra el resultado. De nuevo, el valor z se incrementa en 1 y el valor y se restablece.

A continuación, el bucle se repite una cuarta y última vez, dando como salida la posición índice 3 del cubo:

[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]]

La[Figura 10-18](ch10.xhtml#ch10fig18) muestra la estructura terminada del cubo.

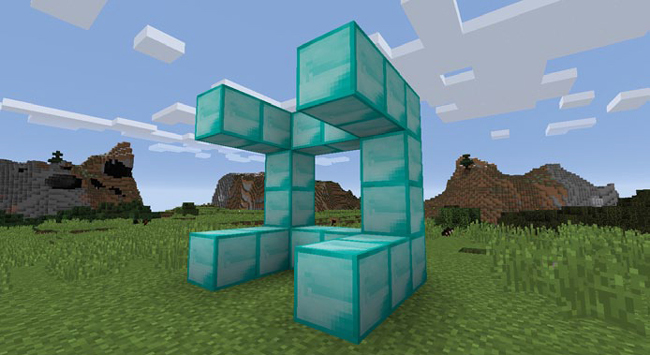
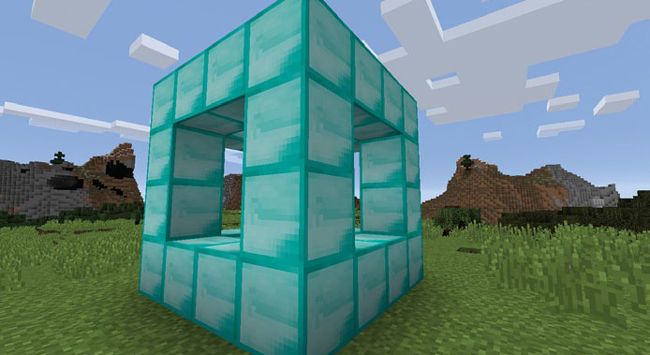


Figura 10-17: Resultado del*tercer bucle bidimensional,* índice *2* del cubo



Figura*10-18*: El resultado del último bucle bidimensional, en la posición final *del* índice del cubo

Experimenta con este programa: utiliza un tipo de bloque diferente, intenta hacer un cubo más grande, ¡o cualquier otra cosa que puedas imaginar! En la siguiente sección, te mostraré cómo acceder a valores en listas tridimensionales para que puedas hacer algunos de estos cambios.

#### **Acceder a valores en listas tridimensionales**

Los valores dentro de las listas tridimensionales pueden cambiarse igual que en las listas unidimensionales y bidimensionales, utilizando corchetes y posiciones de índice.

Empecemos con nuestra lista tridimensional de cubos de diamante:

cube = [[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]],  
#  
[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]],  
#  
[[57, 0, 0, 57],  
[0, 0, 0, 0],  
[0, 0, 0, 0],  
[57, 0, 0, 57]],  
#  
[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]]]

Quiero cambiar el bloque inferior izquierdo de la parte frontal de mi cubo a dorado.

Primero necesito acceder al índice de la lista de cubos que contiene la parte frontal del cubo, que es 0. Así que la primera parte de la expresión tendrá este aspecto:

cube[0]

Si imprimiera el valor de esta expresión, obtendría la siguiente salida (que he formateado para que sea más fácil de leer):

[[57, 57, 57, 57],  
[57, 0, 0, 57],  
[57, 0, 0, 57],  
[57, 57, 57, 57]]

Esta lista bidimensional representa la parte frontal del cubo. A continuación, quiero acceder a la fila inferior, que es el índice 3. Así que añado [3] a mi expresión:

cube[0][3]

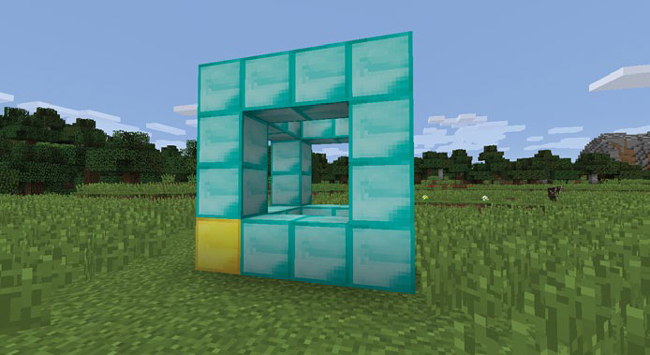
Si imprimiera la lista almacenada en esta posición, obtendría lo siguiente:

[57, 57, 57, 57]

Por último, quiero acceder al bloque situado más a la izquierda de la fila, que es el índice 3. Así que la expresión final para cambiar el bloque de abajo a la izquierda por un bloque dorado tiene el siguiente aspecto:

cube[0][3][3] = 41

Cuando ejecuto el programa *cubo.* py con esta línea añadida, obtengo un cubo hecho de diamantes con un único bloque dorado, como en la [Figura 10-19](ch10.xhtml#ch10fig19).



*Figura 10-19: El cubo modificado con una sola esquina dorada*

#### **Misión nº 63: Duplicar un edificio**

Aunque construir cosas en Minecraft utilizando un programa Python ahorra mucho tiempo, si eres como yo, es posible que sigas dedicando una cantidad considerable de esfuerzo a añadir detalles, como imágenes y muebles, a tus edificios. A veces puede que necesites hacer una copia idéntica de un objeto concreto, y copiar un objeto a mano puede llevar mucho tiempo. Colocar cada bloque uno a uno también es mucho trabajo, y podrías colocar un bloque en el lugar equivocado. La solución obvia es hacer un programa que copie un edificio en Minecraft y construya una copia del mismo en el juego por ti.

El programa terminado tendrá que hacer dos cosas: primero copiará un área del juego y la almacenará en una lista tridimensional, y luego construirá la estructura copiada utilizando esa lista tridimensional.

He incluido el inicio del programa en el Listado [10-10](ch10.xhtml#ch10ex10) para ayudarte. Copia el listado en un archivo nuevo y guárdalo como *duplicarÁrea.py* en la carpeta *forLoops*.

*duplicateArea.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ def sortPair(val1, val2):  
if val1 > val2:  
return val2, val1  
else:  
return val1, val2  
➋ def copyStructure(x1, y1, z1, x2, y2, z2):  
# Sort the highest and lowest x, y, and z values  
x1, x2 = sortPair(x1, x2)  
y1, y2 = sortPair(y1, y2)  
z1, z2 = sortPair(z1, z2)  
  
width = x2 - x1  
height = y2 - y1  
length = z2 - z1  
  
structure = []  
  
print("Please wait...")  
  
➌ # Copy the structure  
  
return structure  
  
➍ def buildStructure(x, y, z, structure):  
  
xStart = x  
yStart = y  
  
➎ # Build the structure  
  
# Get the position of the first corner  
➏ input("Move to the first corner and press enter in this window")  
pos = mc.player.getTilePos()  
x1, y1, z1 = pos.x, pos.y, pos.z  
  
# Get the position of the second corner  
➐ input("Move to the opposite corner and press enter in this window")  
pos = mc.player.getTilePos()  
x2, y2, z2 = pos.x, pos.y, pos.z  
  
# Copy the building  
➑ structure = copyStructure(x1, y1, z1, x2, y2, z2)  
  
# Set the position for the copy  
➒ input("Move to the position you want to create the structure and press ENTER image  
in this window")  
pos = mc.player.getTilePos()  
x, y, z = pos.x, pos.y, pos.z  
buildStructure(x, y, z, structure)

*Listado 10-10: Cuando el programa esté terminado, duplicará edificios.*

Este programa se divide en varias partes. En primer lugar, la función sortPair() ➊ ordena un par de valores en una tupla con el valor más bajo en la primera posición del índice y el valor más alto en la segunda posición del índice. Por ejemplo, si le diera a sortPair () los argumentos 9 y 3, devolvería una tupla con el valor (3, 9) porque 3 es menor que 9. Utilizo esta función para ordenar pares de valores x, y y z, de modo que las variables anchura, longitud y profundidad sean siempre positivas al calcularlas.

A continuación, la función copyStructure () ➋ copia la estructura del mundo del juego, pero está incompleta ➌. La función buildStructure () ➍ construye la estructura, pero también está incompleta ➎. En esta misión completarás ambas cosas.

He añadido un truco ingenioso para obtener las coordenadas del edificio que quieres copiar y la ubicación en el juego donde quieres construir la copia: Utilizando la función input (), el programa te pide primero que muevas a tu personaje a una esquina del edificio y pulses ENTER ➏. La función input( ) hace que el código espere hasta que hayas movido al jugador hasta donde quieres que esté. En cuanto pulses ENTER, obtiene la posición del jugador mediante la función getTilePos(). Volvemos a hacer lo mismo en la esquina opuesta del edificio ➐. A continuación, la función copyStructure () utiliza estos dos conjuntos de coordenadas para copiar el edificio ➑. (Al copiar estructuras más grandes, esta parte del programa puede tardar un rato en ejecutarse). Por último, te mueves al lugar donde quieres que se construya el edificio y pulsas ENTER ➒ para pasar la última posición del jugador a la función buildStructure().

Para terminar el programa, te toca completar las funciones copyStructure() y buildStructure(). Añade tres bucles anidados a la función copyStructure () para copiar todos los bloques entre las coordenadas dadas en el argumento en una lista tridimensional en ➌. Para terminar la función buildStructure (), añade tres bucles for anidados que emitan los valores de los bloques de la lista tridimensional en ➎. La función debe utilizar las coordenadas dadas en sus argumentos.

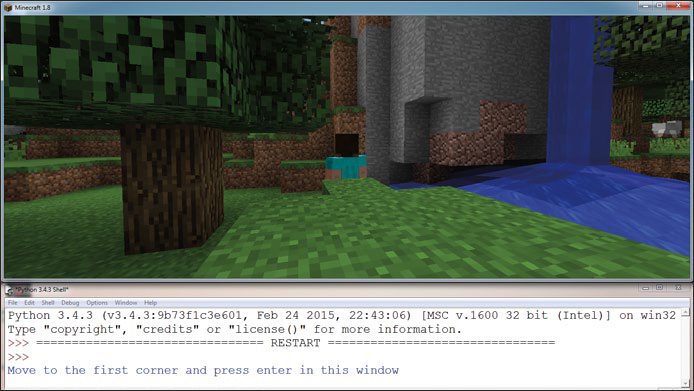
Asegúrate de que el programa trabaja a través de las posiciones x, y, y z dentro de la estructura. Utiliza los bucles for para cambiar las posiciones x, y, y z.

Aunque *duplicarÁrea.py* es un programa largo, es muy útil y merece la pena el esfuerzo. Cuando termines esta misión, ¡podrás construir ciudades enteras en tu mundo Minecraft! Utilicé *duplicateArea.* py para duplicar un interesante acantilado que encontré mientras exploraba. La [Figura 10-20](ch10.xhtml#ch10fig20) muestra el acantilado que quería copiar.



*Figura 10-20: Me gustó el aspecto de este acantilado, así que hice una copia de él.*

Cuando utilices el programa *duplicateArea.* py para hacer copias, colócate primero en el exterior (si tu estructura es un edificio) y cerca de la esquina inferior del objeto que quieres copiar. A continuación, pulsa ENTER en IDLE. [La Figura 10-21](ch10.xhtml#ch10fig21) me muestra de pie en la primera esquina.



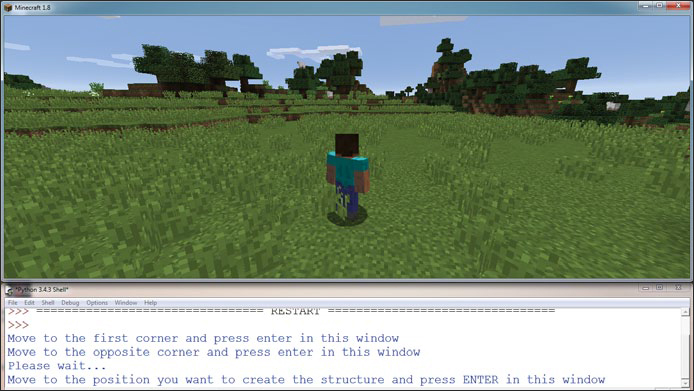
*Figura 10-21: Primero me desplacé a una esquina de la estructura y pulsé ENTER en IDLE.*

Después, vuela hacia arriba y alrededor hasta la esquina opuesta del objeto que quieres copiar y pulsa ENTER por segunda vez. La Figura [10-22](ch10.xhtml#ch10fig22) muestra que he volado hacia arriba y me he movido alrededor del acantilado.



*Figura 10-22:* Después me he desplazado hasta*la esquina opuesta de la estructura y he pulsado* *INTRO*.

Aparecerá un mensaje pidiéndote que esperes un momento mientras el programa copia la estructura. Muévete hasta el lugar donde quieras construir la copia y espera a que aparezca un mensaje preguntándote dónde quieres construir la nueva estructura[(](ch10.xhtml#ch10fig23)Figura 10-23).



*Figura 10-23: Esperé un rato a que la estructura se copiara. Luego me desplacé al lugar donde quería construir la copia y pulsé ENTER para construirla.*

Pulsa INTRO cuando estés en el lugar adecuado para el nuevo edificio, ¡y se construirá una copia justo delante de ti! [La Figura 10-24](ch10.xhtml#ch10fig24) muestra mi copia del acantilado.



*Figura 10-24: ¡La copia del acantilado original!*

[anterior](ch10_5.html)[Subtema 6 de 7: (Ver todo)](ch10.html)[siguiente](ch10_7.html)