Aprende a programar con Minecraft

### **Recuperarse de los errores**

Python utiliza el manejo de *excepciones* para asegurarse de que tu programa pueda recuperarse de los errores y continuar ejecutándose cuando se produzcan. Por ejemplo, el manejo de excepciones es una forma de gestionar las entradas incorrectas del usuario.

Supongamos que tu programa solicita un número entero, pero el usuario introduce una cadena. Normalmente, el programa mostraría un mensaje de error, lo que también se llama *lanzar una* excepción, y dejaría de ejecutarse.

Con el manejo de excepciones, puedes gestionar tú mismo ese error: puedes mantener el programa funcionando sin problemas, mostrar un mensaje de error útil al usuario -como "Por favor, introduzca un número en su lugar"- y darle la oportunidad de solucionar su problema sin reiniciar el programa.

Una sentencia try-except es una herramienta que puedes utilizar para gestionar errores. Es especialmente buena para proporcionar información útil al usuario cuando introduce datos incorrectos, y puede evitar que tu programa se detenga cuando se produce un error.

La sentencia se compone de dos partes: el try y el except. La primera parte, try, es el código que quieres ejecutar si no se produce ningún error. Este código puede tomar una entrada o imprimir una cadena. La parte except de la sentencia se ejecutará sólo si se produce un error en la parte try.

Imagina un código que te pregunta cuántos pares de gafas de sol tienes (tengo tres pares):

try:  
➊ noOfSunglasses = int(input("How many sunglasses do you own? "))  
except:  
➋ print("Invalid input: please enter a number")

Este programa requiere un número. Si introduces letras o símbolos, imprimirá "Entrada no válida: introduce un número". El error se produce porque la función int( ) sólo puede convertir cadenas que contengan sólo números enteros ➊. Si introduces un número, el código funcionará, pero si introduces algo que no es un número, como muchas gafas de sol, esta entrada provocará un error en la función int( ).

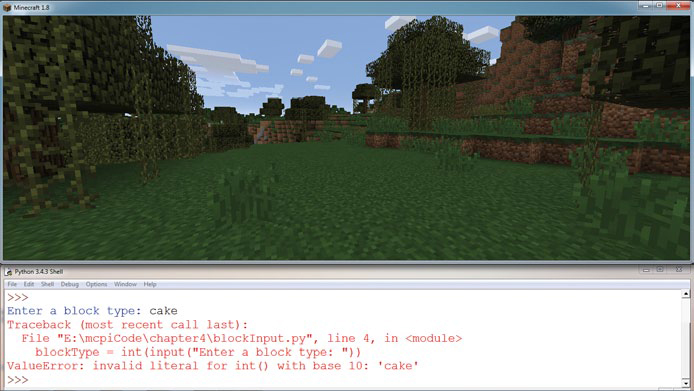
Por cierto, ¿has notado algo diferente en este trozo de código? Es la primera vez que utilizamos una sentencia que requiere *sangría*, que es cuando escribes varios espacios *antes de* escribir cualquier texto. Hablaré más de la sangría cuando trate las sentencias if en [el Capítulo 6](ch06.xhtml#ch06) y los bucles for en [los Capítulos 7](ch07.xhtml#ch07) y [9](ch09.xhtml#ch09). Por ahora, asegúrate de que escribes tu código exactamente como aparece en este libro.

Normalmente, cuando se produce un error, Python muestra un mensaje difícil de entender y no indica claramente al usuario cómo solucionar el problema. Pero con try-except, puedes evitar que los mensajes de error de Python se muestren al usuario cuando introduzca un tipo de datos incorrecto; en su lugar, puedes darle instrucciones sencillas y útiles sobre lo que debe hacer. A veces el usuario simplemente pulsa INTRO en lugar de introducir datos. Normalmente, esto crea un error, pero con el código dentro de la sentencia try-except ➊, en su lugar se imprimirá la cadena " Entrada no válida: por favor, introduzca un número ➋.

Puedes poner casi cualquier código dentro de una sentencia try-except, incluso otras sentencias try-except. ¡Pruébalo en la siguiente misión!

#### **Misión nº 15: Sólo se permiten números**

¿Recuerdas el programa que escribiste en la Misión 14? Cuando introducías un valor entero, el programa funcionaba exactamente como debía y creaba un bloque. Pero si introducías una cadena, el programa dejaba de funcionar y mostraba un error, como se muestra en la [Figura 4-5](ch04.xhtml#ch4fig5).



*Figura4-5:* cake *no es un número, por lo que el programa no creó un bloque.*

Este mensaje de error tiene sentido cuando estás acostumbrado a Python. Pero, ¿qué pasaría si alguien que nunca antes hubiera utilizado Python intentara introducir una cadena en lugar de un número entero? Recibiría un mensaje de error que no entendería. Tu misión es utilizar el tratamiento de errores para escribir un mensaje que sea fácil de entender.

Abre el programa bloqueEntrada *.py* que creaste en la Misión 14. Guarda el programa como *blockInputFix.py* en la carpeta *strings*.

Vas a cambiar el programa para que utilice una sentencia try-except cuando te pida un número de bloque. Busca la última línea de código en el programa. Debe tener este aspecto:

mc.setBlock(x, y, z, blockType)

Añade una sentencia try en la línea anterior a ésta y añade cuatro espacios al principio de la línea antes de la función mc.setBlock(). Después, en la línea anterior a setBlock( ), añade este código para obtener una entrada del usuario: blockType = int(input("Introduce un tipo de bloque: ")).

A continuación, en la línea justo después de la función setBlock (), escribe una sentencia except. Dentro de la sentencia except añade una línea de código que publique un mensaje en el chat de Minecraft para decir que el tipo de bloque debe ser un número; por ejemplo, " ¡No has introducido un número! Introduce un número la próxima vez". Este es el aspecto que debería tener el código modificado (fíjate en los cuatro espacios, o sangría, al principio de las líneas ➊ y ➋):

try:  
➊ blockType = int(input("Enter a block type: "))  
mc.setBlock(x, y, z, blockType)  
except:  
➋ mc.postToChat("You did not enter a number! Enter a number next time.")

La función int( ) espera convertir la entrada introducida por el usuario en un número entero ➊. Como hemos añadido la sentencia try-except al programa, si el usuario introduce algo que no es un número (como letras o símbolos), se producirá un error. En lugar de mostrar el mensaje de error normal de Python, el programa enviará una cadena al chat pidiendo al usuario que introduzca sólo un número ➋. Quizá quieras cambiar el mensaje del chat para que sea un poco más educado.

Cuando hayas terminado de introducir un mensaje de error más amable, guarda el archivo *blockInputFix.py* y ejecútalo para admirar tu obra. El resultado debería parecerse a [la Figura 4-6](ch04.xhtml#ch4fig6).



*Figura 4-6: El mensaje de error mostrado en el chat es mucho más fácil de entender.*

#### **Misión nº 16: Sprint Record**

La misión final de este capítulo combina todo lo que has aprendido sobre variables[(Capítulo 2](ch02.xhtml#ch02)) y operadores matemáticos[(Capítulo 3](ch03.xhtml#ch03)) con la publicación de mensajes en el chat. Tu tarea consiste en crear un registrador: el programa calculará la distancia que recorre el jugador en 10 segundos y mostrará los resultados en el chat.

Recuerda que puedes utilizar el siguiente código para hacer que tus programas esperen, o duerman, un determinado número de segundos:

import time # Place this somewhere near the top of your program  
time.sleep(30) # Makes the program wait 30 seconds

Utiliza este ejemplo de sleep() y escribe el siguiente código para empezar con este nuevo programa:

*sprint.py*

import time  
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ pos1 = mc.player.getTilePos()  
x1 = pos1.x  
y1 = pos1.y  
z1 = pos1.z  
  
time.sleep(10)  
  
➋ pos2 = mc.player.getTilePos()  
x2 = pos2.x  
y2 = pos2.y  
z2 = pos2.z  
  
# Compare the difference between the starting position and ending position  
➌ xDistance = x2 – x1  
yDistance =  
zDistance =  
  
# Post the results to the chat  
➍ mc.postToChat("")

Desglosemos este código. El programa obtiene la posición inicial del jugador ➊, espera 10 segundos y luego obtiene la posición final del jugador ➋. Para terminar el programa, tienes que calcular la diferencia entre las posiciones inicial y final. Para ello, establece los valores de las variables yDistancia y zDistancia, que comienzan en ➌. Para ayudarte, he incluido el valor de la variable xDistancia, que debe ser x2 - x1. Los valores de las variables yDistancia y zDistancia deben ser similares a éste, pero utilizando variables distintas en lugar de x1 y x2.

En la última línea, envía los resultados al chat de Minecraft ➍. Los resultados deben tener el siguiente formato "El jugador se ha movido x: 10, y: 6, y z: -3". Utiliza cadenas, concatenación y los valores de las variables xDistancia, yDistancia y zDistancia para hacerlo.

Guarda este programa en la carpeta *cadenas* como *sprint.py* y ejecútalo. [La Figura 4-7](ch04.xhtml#ch4fig7) muestra el resultado del programa.



*Figura 4-7: Al finalizar el programa se muestra la distancia recorrida.*

Si tienes el programa en marcha pero te resulta difícil cambiar entre la línea de comandos y Minecraft lo suficientemente rápido, prueba a añadir una cuenta atrás de tres segundos antes del paso 2. Publica esta cuenta atrás en el chat.

**OBJETIVO EXTRA: A VUELO DE PÁJARO**

En este punto, el programa muestra la distancia recorrida a lo largo de cada eje por separado: la dirección x, la dirección y y la dirección z. ¿Cómo crearías y mostrarías la distancia recorrida como un único número entero, es decir, a vuelo de pájaro? Pista: Tendrás que utilizar la trigonometría, concretamente el teorema de Pitágoras. Si no estás seguro de cómo hacerlo ahora, no te preocupes; verás un programa similar con código para calcular la distancia recorrida como un único número entero en la Misión 40[(página 145](ch07.xhtml#page_145)).

[anterior](ch04_6.html)[Subtema 7 de 8: (Ver todo)](ch04.html)[siguiente](ch04_8.html)