Aprende a programar con Minecraft

### **Entender los métodos**

Las clases pueden contener métodos, que son funciones asociadas a la clase. Escribir métodos de clase te permite crear funciones que pueden utilizar todas las instancias de esa clase. Es una forma estupenda de ahorrar tiempo y reutilizar código, porque sólo tendrás que escribir un método.

Para crear un método, escribe una función en el cuerpo de una clase utilizando la palabra clave def. Ya has utilizado la palabra clave def en capítulos anteriores para crear funciones. Los métodos también se crean con la palabra clave def, pero están sangrados bajo la clase a la que pertenecen. Por ejemplo, vamos a actualizar la clase Gato en *catClass.py*. Queremos que el gato pueda comer, así que añadamos un método llamado comer() a la clase Gato. Introduce el código y realiza los cambios en catClass *.* py a medida que vayas avanzando:

*catClass.py*

class Cat(object):  
def \_\_init\_\_(self, name, weight):  
self.name = name  
self.weight = weight  
  
def eat(self, food):  
self.weight = self.weight + 0.05  
print(self.name + " is eating " + food)

Observa que la definición del método y el cuerpo del método están sangrados con cuatro espacios extra para que Python sepa que pertenecen a la clase.

Al igual que las funciones, los métodos pueden recibir argumentos. Aquí el método comer( ) toma un argumento llamado comida que indica lo que está comiendo el gato. El método comer () aumenta el atributo peso del gato en 0,05 y luego imprime un mensaje indicando que el gato se está comiendo la comida.

Tras crear un objeto, puedes llamar a cualquiera de los métodos de su clase. Por ejemplo, puedes llamar al método comer() utilizando el objeto pelusa. Añade este código al final de *catClass.py*:

*catClass.py*

fluff = Cat("Fluff", 4.5)  
fluff.eat("tuna")

Aquí vemos nuestro código anterior, en el que creamos un objeto llamado pelusa que forma parte de la clase Gato. Luego llamamos al método comer() y le damos el argumento "atún". Cuando ejecutes el programa, la salida tendrá este aspecto:

Fluff is eating tuna

Ahora Pelusa está feliz comiendo atún. Recuerda que el método comer () también aumenta el atributo peso. Después de llamar al método comer( ), añade el código para imprimir el peso de Pelusa.

También puedes llamar a métodos desde dentro de la clase llamando a un método dentro de otro método. Vamos a crear otro método llamado comerYDormir( ) dentro de la clase Gato. El método comerYDormir () llama al método comer() y luego imprime que el gato está durmiendo. Añade este código a *catClass.py*, justo después del método eat( ) (asegúrate de que indentas el nuevo método como se muestra para que Python sepa que forma parte de la clase):

*catClass.py*

def eatAndSleep(self, food):  
self.eat(food)  
print(self.name + " is now sleeping...")

Para llamar a un método desde dentro de la clase a la que pertenece, añade self. al principio del nombre del método. Aquí se llama al método comer( ) utilizando self. comer(). Ten en cuenta que esto es diferente de llamar a un método fuera de una clase. Cuando haces eso, sólo tienes que introducir el nombre del objeto y el método al que llamas. Por ejemplo, el siguiente código llama al nuevo método comerYDormir () del objeto pelusa. Añádelo a tu archivo *catClass.py*. Debe ser la última línea de código de tu programa:

*catClass.py*

fluff.eatAndSleep("tuna")

Esta es la salida que deberías obtener al ejecutar el programa:

Fluff is eating tuna  
Fluff is now sleeping...

Aquí tienes el programa completo para que veas dónde están todas las piezas:

class Cat(object):  
def \_\_init\_\_(self, name, weight):  
self.name = name  
self.weight = weight  
  
def eat(self, food):  
self.weight = self.weight + 0.05  
print(self.name + " is eating " + food)  
  
def eatAndSleep(self, food):  
self.eat(food)  
print(self.name + " is now sleeping...")  
  
fluff = Cat("Fluff", 4.5)  
print(fluff.weight)  
fluff.eat("tuna")  
fluff.eatAndSleep("tuna")

¡Llevemos las nuevas habilidades que has aprendido al mundo de Minecraft!

#### **Misión nº 69: Casa Fantasma**

Lo mejor de programar con Python y Minecraft es que puedes empezar con una idea tonta y correr con ella. Tu idea puede empezar siendo pequeña, pero con sólo unas pocas líneas de código, puedes construir un programa divertido muy rápidamente.

¿No sería divertido construir una casa fantasma que apareciera en un juego, sólo para desaparecer 30 segundos después? La casa podría reaparecer en otro lugar y volver a desaparecer si tú quisieras.

Aquí tienes la primera versión del programa de la casa fantasma. Guarda [el Listado 12-2](ch12.xhtml#ch12ex2) en un archivo llamado *casafantasma.py* en la carpeta de *clases*.

*ghostHouse.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
import time  
➊ class Building(object):  
➋ def \_\_init\_\_(self, x, y, z, width, height, depth):  
self.x = x  
self.y = y  
self.z = z  
  
self.width = width  
self.height = height  
self.depth = depth  
  
➌ def build(self):  
mc.setBlocks(self.x, self.y, self.z,  
self.x + self.width, self.y + self.height,  
self.z + self.depth, 4)  
  
mc.setBlocks(self.x + 1, self.y + 1, self.z + 1,  
self.x + self.width - 1, self.y + self.height - 1,  
self.z + self.depth - 1, 0)  
➍ # Call the buildDoor() and buildWindows() methods here  
  
➎ def clear(self):  
mc.setBlocks(self.x, self.y, self.z,  
self.x + self.width, self.y + self.height,  
self.z + self.depth, 0)  
➏ # Remove the doors and windows here  
  
pos = mc.player.getTilePos()  
x = pos.x  
y = pos.y  
z = pos.z  
  
➐ ghostHouse = Building(x, y, z, 10, 6, 8) ghostHouse.build()  
time.sleep(30)  
  
ghostHouse.clear()  
➑ ghostHouse.x = 8

*ghostHouse.py*

Listado 12-2*: La* clase Edificio *crea un edificio.*

[El](ch12.xhtml#ch12ex2) listado 12-2 utiliza una clase llamada Edificio ➊ con un método \_\_init\_\_( ) para establecer la posición y el tamaño de la casa ➋. Crea un objeto Edificio con el nombre CasaFantasma ➐. El edificio aparece y luego desaparece misteriosamente al cabo de 30 segundos utilizando los métodos build() ➌ y clear() ➎. El único problema es que no parece una casa. Ahora mismo parece un gran cascarón vacío hecho de adoquines.

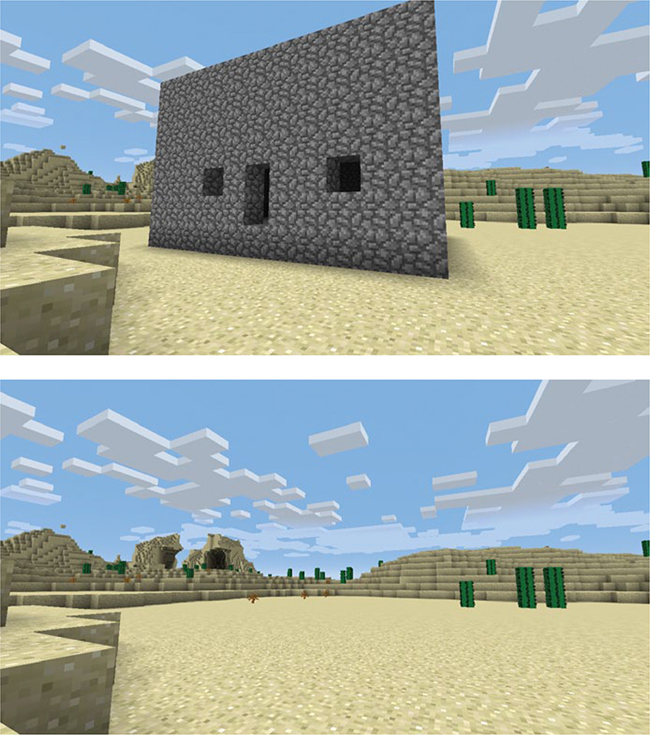
Tienes que hacer que la casa fantasma se parezca más a una casa y menos a un cascarón, porque los cascarones fantasma no dan tanto miedo como las casas fantasma. Para que el edificio parezca más una casa, tu misión es añadir un método que construya una puerta en la parte delantera de la casa y un segundo método que añada ventanas. Llama a estos dos métodos desde dentro del método build( ) para que se construyan al mismo tiempo ➍.

Después de añadir los métodos para construir la puerta y las ventanas, tendrás que actualizar el método clear () para eliminarlos ➏; de lo contrario, se quedarán atrás cuando desaparezca la casa.

Cuando hayas añadido los métodos adicionales, traslada el edificio a una nueva ubicación cambiando los atributos x, y y z del objeto ghostHouse y añadiendo más llamadas a los métodos build() y clear ( ). He empezado esto por ti cambiando la posición x de la casa ➑.

Cuando ejecutes el programa, la casa fantasma debería aparecer de repente y desaparecer 30 segundos después, para reaparecer en otro lugar. ¡Espeluznante!

[La Figura 12-2](ch12.xhtml#ch12fig2) muestra mi casa fantasma.



*Figura 12-2: La casa fantasma aparece y luego desaparece.*

**OBJETIVO EXTRA: MEJORA DE LA CASA**

De momento, la casa fantasma es muy básica. Utilizando las increíbles habilidades de Python que has aprendido en este libro, añade lo que quieras a la función build( ) para personalizar tu casa.

[anterior](ch12_3.html)[Subtema 4 de 10: (Ver todo)](ch12.html)[siguiente](ch12_5.html)