Aprende a programar con Minecraft

### **Crear una clase**

Empezarás creando una clase y luego crearás todos tus objetos a partir de esa clase. Para crear una clase, utiliza la palabra clave class, el nombre con el que quieras llamar a la clase y la clase objeto entre paréntesis (explicaré la clase objeto en "[Heredar una clase](ch12.xhtml#ch12lev2sec07)" en [la página 278](ch12.xhtml#page_278)):

class ClassName(object):  
def \_\_init\_\_(self):  
# Body of init

Es una buena práctica poner en mayúsculas los nombres de tus clases. Así es más fácil distinguir las clases de las funciones, que deben empezar por minúscula.

Cuando crees una nueva clase, debes incluir el método \_\_init\_\_() y pasarle self como argumento. El argumento self es necesario para todos los métodos de una clase. Hace referencia a la clase a la que pertenece el método. El método \_\_init\_\_ () indica a Python lo que quieres que haga la clase cuando la utilices por primera vez en un programa. Esto se llama *inicializar* la clase, que es la abreviatura de \_\_init\_\_().

Por ejemplo, vamos a crear una clase llamada Gato y a crear algunos objetos gato. La clase Gato almacenará dos atributos para cada gato, su nombre y su peso en kilogramos. Cada objeto gato tendrá sus propios valores de nombre y peso. Abre un nuevo archivo en el editor de texto de IDLE y guárdalo como claseGato *.py* en una nueva carpeta llamada *clases*. Introduce el siguiente código para crear una clase llamada Gato:

*catClass.py*

class Cat(object):  
➊ def \_\_init\_\_(self, name, weight):  
➋ self.name = name  
➌ self.weight = weight

En este ejemplo, el método \_\_init\_\_( ) toma tres argumentos ➊. El primero es self, que es un argumento obligatorio en todo método de clase. El segundo argumento, nombre, y el último argumento, peso, son argumentos adicionales para crear atributos para todos los gatos.

Las dos últimas líneas crean los atributos nombre ➋ y peso ➌ y los establecen en los valores de los argumentos nombre y peso. Cuando creas atributos dentro de una clase, utilizas la notación de puntos con self. Los atributos siempre se identifican con un self, que indica a Python que un atributo pertenece a la clase.

A continuación, aprenderás a utilizar esta clase para crear instancias de objetos.

#### **Crear un objeto**

Utilizando la clase recién creada, vamos a crear algunos objetos gato, o instancias de la clase Gato.

Inicializar un objeto es similar a crear una variable. Para inicializar un objeto, introduces el nombre del objeto, un signo igual(=) y el nombre de la clase. Pasas argumentos a la clase entre paréntesis, igual que haces con una llamada a una función.

Por ejemplo, adoptemos un gato y llamémosle Pelusa. Utilizando la clase Gato, podemos crear un objeto gato llamado pelusa añadiendo el código siguiente en la última línea de *catClass.* py (fíjate en que no tiene sangría):

*catClass.py*

class Cat(object):  
def \_\_init\_\_(self, name, weight):  
self.name = name  
self.weight = weight  
  
fluff = Cat("Fluff", 4.5)

Cuando creas un objeto, el número de argumentos que proporcionas depende de los argumentos de su función \_\_init\_\_(). Aquí incluimos dos argumentos, uno para el nombre ("Pelusa") y otro para el peso(4,5). No necesitas incluir el argumento self al crear un objeto porque Python lo añade automáticamente.

Crear un objeto también se conoce como *llamar a un constructor*. El método \_\_init\_\_ () suele denominarse constructor porque construye una clase cuando se le llama. El método \_\_init\_\_ () es un tipo especial de método porque no haces referencia a él por su nombre. En cambio, se ejecuta cuando creas un objeto utilizando el nombre de la clase. Por ejemplo, aquí el código pelusa = Gato("Pelusa", 4.5) llama al método \_\_init\_\_ (), que construye un objeto Gato llamado pelusa.

A continuación, aprenderás a acceder a los atributos del objeto pelusa.

#### **Acceder a los atributos**

Puedes acceder a los atributos de un objeto para obtener más información sobre él. Por ejemplo, añade el siguiente código a *catClass.* py después del objeto pelusa para imprimir el atributo peso del objeto pelusa:

*catClass.py*

print(fluff.weight)

El valor que se imprime al ejecutar el programa debe ser 4,5, porque ése es el valor que le pusiste al atributo peso cuando creaste el objeto.

Observa que estamos utilizando la notación de puntos entre el nombre del objeto, pelusa, y el atributo peso. El punto significa que quieres utilizar el atributo que pertenece a un objeto concreto. En este caso, el valor del atributo peso pertenece al objeto pelusa. Siempre que obtengas o establezcas el valor del atributo de un objeto, utilizarás la notación de punto.

Puedes cambiar el valor de un atributo como lo harías con cualquier otra variable, utilizando un signo igual(=). Por ejemplo, cambiemos el peso de Pelusa a 5 porque ha engordado durante las vacaciones de invierno. Para ello, cambia el atributo peso del objeto pelusa a 5:

*catClass.py*

fluff.weight = 5

Ahora, siempre que accedas al atributo peso del objeto pelusa, será 5.

Utilizando los conocimientos que ahora tienes sobre cómo crear una clase y crear una instancia de la misma, vamos a hacer cosas chulas en Minecraft.

#### **Misión nº 68: Objetos de Ubicación**

A lo largo del libro, has almacenado ubicaciones, como tu casa, un castillo o un palacio, en tu mundo Minecraft. Has utilizado variables, listas, tuplas y diccionarios para hacerlo de diversas formas.

También puedes crear y almacenar información relacionada, como ubicaciones, utilizando la programación orientada a objetos. Por ejemplo, puedes utilizar objetos para almacenar las coordenadas de un montón de ubicaciones diferentes.

Cada ubicación tiene una coordenada x, y y z, pero los valores de cada ubicación son diferentes. Creando una clase de ubicación, puedes almacenar y acceder a las coordenadas de distintas ubicaciones. Eso te ayudará a llevar un registro de todas las cosas increíbles que construyas en Minecraft. ¡Podrás acceder fácilmente a las coordenadas de todas tus creaciones Minecraft para poder teletransportar al jugador a ellas en un instante!

[El listado 12-1](ch12.xhtml#ch12ex1) contiene el inicio de la clase Ubicación. Cuando el código esté terminado, se podrá utilizar para almacenar las coordenadas de una ubicación en un único objeto. Copia el código en un nuevo archivo llamado *locationClass.py* en la carpeta de *clases*.

*locationClass.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
➊ class Location(object):  
def \_\_init\_\_(self, x, y, z):  
➋ self.x = x  
➌ # Add the y and z attributes here  
  
➍ bedroom = Location(64, 52, -8)  
➎ mc.player.setTilePos(bedroom.x, bedroom.y, bedroom.z)

*Listado 12-1: El inicio de la* *clase* Ubicación

Para iniciar la clase, he incluido la palabra clave class y he llamado a la clase Ubicación ➊. En ➍ está el código para inicializar un objeto llamado dormitorio, que almacenará la ubicación del dormitorio en mi casa de Minecraft. El método setTilePos( ) establece la posición del jugador en la ubicación del dormitorio -los atributos x, y y z del objeto dormitorio ➎. Sin embargo, el programa está incompleto. Tienes que terminar el método \_\_init\_\_( ) de la clase y establecer los atributos y y z a los valores de los argumentos pasados al método \_\_init\_\_(). Yo he fijado el valor del atributo x ➋, pero es tarea tuya hacer lo mismo con los atributos y y z ➌. ¡No olvides utilizar la ubicación de tu propio dormitorio en ➍!

[La Figura 12-1](ch12.xhtml#ch12fig1) muestra el programa completado en acción, ya que teletransporta al jugador a mi dormitorio.



*Figura 12-1: El programa ha teletransportado al jugador a mi dormitorio.*

**OBJETIVO EXTRA: HOGAR DULCE HOGAR**

¿A qué otras habitaciones de tu casa quieres teletransportarte? Crea más objetos con la clase Ubicación para recorrer tu casa con estilo.

[anterior](ch12_2.html)[Subtema 3 de 10: (Ver todo)](ch12.html)[siguiente](ch12_4.html)