Programación Orientada a Objetos

Antes de nada, empecemos con una introducción básica a la Programación Orientada a Objetos POO o OOP en inglés. Se trata de un paradigma de programación introducido en los años 1970s, pero que no se hizo popular hasta años más tarde.

Este modo o paradigma de programación nos permite organizar el código de una manera que se asemeja bastante a como pensamos en la vida real, utilizando las famosas clases. Estas nos permiten agrupar un conjunto de variables y funciones que veremos a continuación.

Cosas de lo más cotidianas como un perro o un coche pueden ser representadas con clases. Estas clases tienen diferentes características, que en el caso del perro podrían ser la edad, el nombre o la raza. Llamaremos a estas características, atributos.

Por otro lado, las clases tienen un conjunto de funcionalidades o cosas que pueden hacer. En el caso del perro podría ser andar o ladrar. Llamaremos a estas funcionalidades métodos.

Por último, pueden existir diferentes tipos de perro. Podemos tener uno que se llama Toby o el del vecino que se llama Laika. Llamaremos a estos diferentes tipos de perro objetos. Es decir, el concepto abstracto de perro es la clase, pero Toby o cualquier otro perro particular será el objeto.

La programación orientada a objetos está basada en 6 principios o pilares básicos:

Herencia

Cohesión

Abstracción

Polimorfismo

Acoplamiento

Encapsulamiento

Motivación

Una vez explicada la programación orientada a objetos puede parecer bastante lógica, pero no es algo que haya existido siempre, y de hecho hay muchos lenguajes de programación que no la soportan.

En parte surgió debido a la creciente complejidad a la que los programadores se iban enfrentando conforme pasaban los años. En el mundo de la programación hay gran cantidad de aplicaciones que realizan tareas muy similares y es importante identificar los patrones que nos permiten no reinventar la rueda. La programación orientada a objetos intentaba resolver esto.

Uno de los primeros mecanismos que se crearon fueron las funciones, que permiten agrupar bloques de código que hacen una tarea específica bajo un nombre. Algo muy útil ya que permite también reusar esos módulos o funciones sin tener que copiar todo el código, tan solo la llamada.

Las funciones resultaron muy útiles, pero no eran capaces de satisfacer todas las necesidades de los programadores. Uno de los problemas de las funciones es que sólo realizan unas operaciones con unos datos de entrada para entregar una salida, pero no les importa demasiado conservar esos datos o mantener algún tipo de estado. Salvo que se devuelva un valor en la llamada a la función o se usen variables globales, todo lo que pasa dentro de la función queda olvidado, y esto en muchos casos es un problema.

Imaginemos que tenemos un juego con naves espaciales moviéndose por la pantalla. Cada nave tendrá una posición (x,y) y otros parámetros como el tipo de nave, su color o tamaño. Sin hacer uso de clases y POO, tendremos que tener una variable para cada dato que queremos almacenar: coordenadas, color, tamaño, tipo. El problema viene si tenemos 10 naves, ya que nos podríamos juntar con un número muy elevado de variables. Todo un desastre.

En el mundo de la programación existen tareas muy similares al ejemplo con las naves, y en respuesta a ello surgió la programación orientada a objetos. Una herramienta perfecta que permite resolver cierto tipo de problemas de una forma mucho más sencilla, con menos código y más organizado. Agrupa bajo una clase un conjunto de variables y funciones, que pueden ser reutilizadas con características particulares creando objetos.

Definiendo clases

Vista ya la parte teórica, vamos a ver como podemos hacer uso de la programación orientada a objetos en Python. Lo primero es crear una clase, para ello usaremos el ejemplo de perro.

# Creando una clase vacía

class Perro:

pass

Se trata de una clase vacía y sin mucha utilidad práctica, pero es la mínima clase que podemos crear. Nótese el uso del pass que no hace realmente nada, pero daría un error si después de los : no tenemos contenido.

Ahora que tenemos la clase, podemos crear un objeto de la misma. Podemos hacerlo como si de una variable normal se tratase. Nombre de la variable igual a la clase con (). Dentro de los paréntesis irían los parámetros de entrada si los hubiera.

# Creamos un objeto de la clase perro

mi\_perro = Perro()

Definiendo atributos

A continuación vamos a añadir algunos atributos a nuestra clase. Antes de nada es importante distinguir que existen dos tipos de atributos:

Atributos de instancia: Pertenecen a la instancia de la clase o al objeto. Son atributos particulares de cada instancia, en nuestro caso de cada perro.

Atributos de clase: Se trata de atributos que pertenecen a la clase, por lo tanto serán comunes para todos los objetos.

Empecemos creando un par de atributos de instancia para nuestro perro, el nombre y la raza. Para ello creamos un método \_\_init\_\_ que será llamado automáticamente cuando creemos un objeto. Se trata del constructor.

class Perro:

# El método \_\_init\_\_ es llamado al crear el objeto

def \_\_init\_\_(self, nombre, raza):

print(f"Creando perro {nombre}, {raza}")

# Atributos de instancia

self.nombre = nombre

self.raza = raza

Ahora que hemos definido el método init con dos parámetros de entrada, podemos crear el objeto pasando el valor de los atributos. Usando type() podemos ver como efectivamente el objeto es de la clase Perro.

mi\_perro = Perro("Toby", "Bulldog")

print(type(mi\_perro))

# Creando perro Toby, Bulldog

# <class '\_\_main\_\_.Perro'>

Seguramente te hayas fijado en el self que se pasa como parámetro de entrada del método. Es una variable que representa la instancia de la clase, y deberá estar siempre ahí.

El uso de \_\_init\_\_ y el doble \_\_ no es una coincidencia. Cuando veas un método con esa forma, significa que está reservado para un uso especial del lenguaje. En este caso sería lo que se conoce como constructor. Hay gente que llama a estos métodos mágicos.

Por último, podemos acceder a los atributos usando el objeto y .. Por lo tanto.

print(mi\_perro.nombre) # Toby

print(mi\_perro.raza) # Bulldog

Hasta ahora hemos definido atributos de instancia, ya que son atributos que pertenecen a cada perro concreto. Ahora vamos a definir un atributo de clase, que será común para todos los perros. Por ejemplo, la especie de los perros es algo común para todos los objetos Perro.

class Perro:

# Atributo de clase

especie = 'mamífero'

# El método \_\_init\_\_ es llamado al crear el objeto

def \_\_init\_\_(self, nombre, raza):

print(f"Creando perro {nombre}, {raza}")

# Atributos de instancia

self.nombre = nombre

self.raza = raza

Dado que es un atributo de clase, no es necesario crear un objeto para acceder al atributos. Podemos hacer lo siguiente.

print(Perro.especie)

# mamífero

Se puede acceder también al atributo de clase desde el objeto.

mi\_perro = Perro("Toby", "Bulldog")

mi\_perro.especie

# 'mamífero'

De esta manera, todos los objetos que se creen de la clase perro compartirán ese atributo de clase, ya que pertenecen a la misma.

Definiendo métodos

En realidad cuando usamos \_\_init\_\_ anteriormente ya estábamos definiendo un método, solo que uno especial. A continuación vamos a ver como definir métodos que le den alguna funcionalidad interesante a nuestra clase, siguiendo con el ejemplo de perro.

Vamos a codificar dos métodos, ladrar y caminar. El primero no recibirá ningún parámetro y el segundo recibirá el número de pasos que queremos andar. Como hemos indicado anteriormente self hace referencia a la instancia de la clase. Se puede definir un método con def y el nombre, y entre () los parámetros de entrada que recibe, donde siempre tendrá que estar self el primero.

class Perro:

# Atributo de clase

especie = 'mamífero'

# El método \_\_init\_\_ es llamado al crear el objeto

def \_\_init\_\_(self, nombre, raza):

print(f"Creando perro {nombre}, {raza}")

# Atributos de instancia

self.nombre = nombre

self.raza = raza

def ladra(self):

print("Guau")

def camina(self, pasos):

print(f"Caminando {pasos} pasos")

Por lo tanto si creamos un objeto mi\_perro, podremos hacer uso de sus métodos llamándolos con . y el nombre del método. Como si de una función se tratase, pueden recibir y devolver argumentos.

mi\_perro = Perro("Toby", "Bulldog")

mi\_perro.ladra()

mi\_perro.camina(10)

# Creando perro Toby, Bulldog

# Guau

# Caminando 10 pasos