poo-teoria

December 22, 2023

#

09 - PROGRAMACIÓN ORIENTADA A OBJETOS - PARTE 1

Python es un lenguaje multiparadigma, es decir, permite programar utilizando diferentes estilos. Los paradigmas de programación son formas de clasificar los lenguajes de programación en función de su estilo de programación. Los paradigmas más conocidos son:

- Programación imperativa
- Programación declarativa
- Programación orientada a objetos

Hasta ahora hemos trabajado con el paradigma imperativo, es decir, hemos programado utilizando instrucciones que modifican el estado del programa.

En este tema vamos a introducir el paradigma de programación orientada a objetos. En este paradigma, el programa se estructura en entidades llamadas objetos, que interactúan entre sí para realizar las tareas necesarias. Cada objeto es una instancia de una clase, que define la estructura y el comportamiento de los objetos de ese tipo.

0.1 1. Clases y objetos

Una clase es una plantilla que define la estructura y el comportamiento de algo. Por ejemplo, la clase `str` define la estructura y el comportamiento de las cadenas de texto. Una clase se define mediante la palabra reservada `class`, seguida del nombre de la clase y dos puntos. Por ejemplo, la siguiente clase define un tipo de objeto que representa a una persona:

[ ]:
```
class Persona:
    pass
```

La clase `Persona` no define ninguna estructura ni comportamiento, pero podemos crear objetos de esa clase. Los objetos son instancias de una clase. Para crear un objeto de una clase, se llama a la clase como si fuera una función. Por ejemplo, para crear un objeto de la clase `Persona`, se llama a la clase `Persona()` como si fuera una función:

[ ]:
```
mi_persona = Persona()
type(mi_persona)
```

Hemos creado un objeto de la clase `Persona` y lo hemos asignado a la variable `mi_persona`.

Pero un objeto que no presenta atributos ni métodos no tiene mucho sentido. Podemos agregar características a un objeto mediante atributos. Los atributos son variables que pertenecen a un

objeto. Por ejemplo, podemos agregar el atributo `nombre` a nuestra clase `Persona`.

Para lograrlo debemos definir además nuestro primer método `__init__`. Este método es especial, ya que se ejecuta automáticamente al crear un objeto de la clase.

[ ]:
```
class Persona:
    def __init__(self):
        print('Se ejecutó el método __init__()')

mi_persona = Persona()
```

El método `__init__` recibe como primer parámetro el objeto que se está creando, y los parámetros que se pasan al llamar a la clase. Por convención, el primer parámetro se llama `self`. El método `__init__` debe inicializar los atributos del objeto. Por ejemplo, el siguiente método `__init__` inicializa el atributo `nombre` y `edad` del objeto:

[ ]:
```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

Para crear la instancia de esa clase vamos a tener que pasar entre parentesis los atributos que definimos en el método `__init__`.

[ ]:
```
mi_persona = Persona('Francisco', 28)
```

Ahora podemos acceder a los atributos de un objeto mediante la sintaxis `objeto.atributo`. Por ejemplo, podemos acceder al atributo `nombre` del objeto `mi_persona`:

[ ]:
```
print(mi_persona.nombre,mi_persona.edad)
```

También podemos modificar el valor de un atributo de un objeto:

[ ]:
```
mi_persona.nombre = 'José'
mi_persona.edad = 30

print(mi_persona.nombre,mi_persona.edad)
```

En Python, los atributos de un objeto pueden ser de dos tipos:

- **Atributos de instancia**: Son atributos que pertenecen a un objeto en particular. Por ejemplo, el atributo `nombre` de la clase `Persona` es un atributo de instancia.
- **Atributos de clase**: Son atributos que pertenecen a la clase en general. Por ejemplo, podemos agregar el atributo `especie` a la clase `Persona`. Este atributo es un atributo de clase, ya que es común a todos los objetos de la clase `Persona`.

[ ]:
```
class Gato:
    reino = 'animalia'                  # atributo de clase
    def __init__(self, nombre, raza):
        self.nombre = nombre            # atributos de instancia
```

```
        self.raza = raza

mi_gato = Gato('Garfield', 'persa')
otro_gato = Gato('Tom', 'siames')

print(mi_gato.reino, otro_gato.reino)
```

Para acceder a un atributo de clase, se utiliza la sintaxis `Clase.atributo`. Por ejemplo, podemos acceder al atributo `especie` de la clase `Persona`:

[ ]:
```
print(Gato.reino)
```

Asi como existe el método especial `__init__`, existen otros métodos especiales que permiten definir el comportamiento de los objetos. Por ejemplo, el método `__str__` permite definir la representación en forma de cadena de un objeto. Por ejemplo, el siguiente método `__str__` define la representación en forma de cadena de un objeto de la clase `Persona`:

[ ]:
```
class Persona2:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def __str__(self):
        return f'Persona: {self.nombre}, {self.edad} años'
```

[ ]:
```
mi_persona = Persona('Francisco', 28)
otra_persona = Persona2('Lucas', 24)

print(mi_persona)
print(otra_persona)
```

0.2 2. Métodos

Los métodos son funciones que pertenecen a una clase. Por ejemplo, podemos agregar el método `saludar` a la clase `Persona`. Este método muestra un saludo por pantalla:

[ ]:
```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def saludar(self):
        print(f'Hola, soy {self.nombre} y tengo {self.edad} años')
```

Para llama a un método de un objeto, se utiliza la sintaxis `objeto.metodo()`. Por ejemplo, podemos llamar al método `saludar` del objeto `mi_persona`:

[ ]:
```
mi_persona = Persona('Maria', 31)
mi_persona.saludar()
```

Los métodos pueden recibir parámetros. Por ejemplo, el siguiente método `saludar` recibe un parámetro `otro_nombre` y muestra un saludo personalizado:

[ ]:
```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def saludar(self, otra_persona):
        print(f'Hola {otra_persona}, soy {self.nombre} y tengo {self.edad}␣
 ↪años')
```

[ ]:
```
mi_persona = Persona('Pablo', 48)
mi_persona.saludar('Julia')
```