Programación orientada a objetos

La programación orientada a objetos es un paradigma el cual propone modelar todo en base a clases y a objetos. Este paradigma consiste en el uso de los conceptos de herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Elementos de la POO

- **Clases**

  Las clases son nuestros modelos donde después vamos a crear nuestros objetos. Ejemplo de creación del objeto Antena, Pelo y Ojo:

```
class Antena:
  pass

class Pelo:
  pass

class Ojo:
  pass
```

- **Propiedades**

  Características de nuestros objetos, se representan como variables:

```
class Antena():

  color=('rojo')
  longitud=('3.5m')

class Pelo():

  color=('negro')
  longitud=('8cm')

class Cosa():
  antenas=Antena()
  pelos=Pelo()
  antenas=Antena() #propiedad compuesta por el objeto Cosa Antena
  pelos=Pelo() #propiedad compuesta por el objeto Cosa Pelo

print(Antena.color)
print(Antena.longitud)
print(Pelo.color)
print(Pelo.longitud)
```

- **Métodos**

Son funciones en sí, pero en POO se llaman métodos y representan acciones que puede realizar el objeto (solo el objeto):

```
class Pie():
  color=('marron')
  longitud=('42')


  def saltar(self):
      pass



class Objeto():
  color=('negro')
  longitud=('2')

  def flotar(self):
      pass
```

- **Objeto**

Se podría decir que las clases son como modelos que nos sirven para crear objetos. Usualmente se dice que el objeto es la materialización de una clase.

- **Herencia:** una característica principal de la POO

  Algunos objetos comparten las mismas propiedades y métodos que otros objetos y a esto de le denomina **herencia**. Además, en Python cuando una clase no hereda de ninguna otra, debe heredar de object que es la clase principal de Python.

```
class comida:
  pass
class pan(comida):
  pass
class leche(comida):
  pass
```

- **Polimorfismo**

Se denomina a la capacidad que tiene objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. En otras palabras “si la clase B hereda de la clase A, no tiene que heredar todo acerca de la clase A, sino que puede hacer algunas de las cosas

diferentes” aunque el Polimorfismo no es de gran importancia en Python a diferencia de Java o C++.

- **Encapsulación**

La encapsulación define los accesos a los métodos o atributos de una clase haciendo el lenguaje de programación más seguro y estructurado. Este acceso en Python viene determinado por su nombre, si el nombre comienza con dos guiones bajos (y no termina también con dos guiones bajos) se trata de un atributo o método privada, si no es así éstos son públicos**.**

- público: se pueden ver los métodos y atributos fuera de la clase
- privada: los métodos y atributos solo son accesibles por métodos dentro de la clase

```
class ClaseTest:
  def a(self):
    pass
  def __b(self):
    pass

MyObject = ClaseTest()

MyObject.a()

MyObject.__b() #aqui no podemos acceder ya que es privada y da un error
```