(POO) en Python

Constructor de una clase

```python
def __init__(self, color, aceleracion):
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0
```

Un constructor es el primer método que se llama en la creación de objetos (un concepto de programación orientada a objetos). Siempre es parte de una clase (los métodos de un objeto se definen en una clase).

El constructor siempre se llama al crear un nuevo objeto. Se puede utilizar para inicializar variables de clase y rutinas de inicio.

Atributos de clase y atributos de instancia

```
>>> c1 = Coche('rojo', 20)
>>> c2 = Coche('azul', 20)
>>> print(c1.color)
rojo
>>> print(c2.color)
azul
>>> print(c1.ruedas)  # Atributo de clase
4
>>> print(c2.ruedas)  # Atributo de clase
4
>>> Coche.ruedas = 6  # Atributo de clase
>>> print(c1.ruedas)  # Atributo de clase
6
>>> print(c2.ruedas)  # Atributo de clase
6
```

Una clase puede definir dos tipos diferentes de atributos de datos: atributos de clase y atributos de instancia.

Los atributos de clase son atributos compartidos por todas las instancias de esa clase.

Los atributos de instancia, por el contrario, son únicos para cada uno de los objetos pertenecientes a dicha clase.

Herencia múltiple

```python
class A:
    def print_a(self):
        print('a')


class B:
    def print_b(self):
        print('b')


class C(A, B):
    def print_c(self):
        print('c')


c = C()
c.print_a()
c.print_b()
c.print_c()
```

Python es un lenguaje de programación que permite herencia múltiple. Esto quiere decir que una clase puede heredar de más de una clase a la vez.

Polimorfismo

```python
class Perro:
    def sonido(self):
        print('Guauuuuu!!!')

class Gato:
    def sonido(self):
        print('Miaaauuuu!!')

class Vaca:
    def sonido(self):
        print('Múuuuuuuu!!!')
```

Las tres clases implementan un método llamado `sonido()`. Ahora observa el siguiente script:

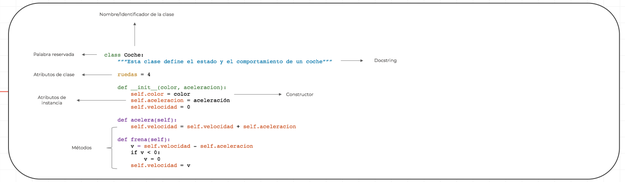
```python
def a_cantar(animales):
    for animal in animales:
        animal.sonido()

if __name__ == '__main__':
    perro = Perro()
    gato = Gato()
    gato_2 = Gato()
    vaca = Vaca()
    perro_2 = Perro()
    granja = [perro, gato, vaca, gato_2, perro_2]
    a_cantar(granja)
```

Polimorfismo es la capacidad de una entidad de referenciar en tiempo de ejecución a instancias de diferentes clases.

Clases y objetos

Básicamente, una clase es una entidad que define una serie de elementos que determinan un estado (datos) y un comportamiento (operaciones sobre los datos que modifican su estado). Por su parte, un objeto es una concreción o instancia de una clase.



Atributos, atributos de datos y métodos

Una vez que sabemos qué es un objeto, tengo que decirte que la única operación que pueden realizar los objetos es referenciar a sus atributos por medio del operador .

Los atributos de datos definen el estado del objeto. En otros lenguajes son conocidos simplemente como atributos o miembros.

Los métodos son las funciones definidas dentro de la clase.

```
>>> c1 = Coche('rojo', 20)
>>> c2 = Coche('azul', 10)
>>> print(c1.color)
rojo
>>> print(c2.color)
azul
>>> c1.marchas = 6
>>> print(c1.marchas)
6
>>> print(c2.marchas)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: 'Coche' object has no attribute 'marchas'
```

Herencia

la herencia es la capacidad de reutilizar una clase extendiendo su funcionalidad. Una clase que hereda de otra puede añadir nuevos atributos, ocultarlos, añadir nuevos métodos o redefinirlos.

En Python, podemos indicar que una clase hereda de otra de la siguiente manera:

```python
class CocheVolador(Coche):

    ruedas = 6

    def __init__(self, color, aceleracion, esta_volando=False):
        super().__init__(color, aceleracion)
        self.esta_volando = esta_volando

    def vuela(self):
        self.esta_volando = True

    def aterriza(self):
        self.esta_volando = False
```

Encapsulación: atributos privados

Hace referencia a la capacidad que tiene un objeto de ocultar su estado, de manera que sus datos solo se puedan modificar por medio de las operaciones (métodos) que ofrece.

```python
class A:
    def __init__(self):
        self._contador = 0  # Este atributo es privado

    def incrementa(self):
        self._contador += 1

    def cuenta(self):
        return self._contador


class B(object):
    def __init__(self):
        self.__contador = 0  # Este atributo es privado

    def incrementa(self):
        self.__contador += 1

    def cuenta(self):
        return self.__contador
```