Programación orientada a objetos

- Paradigma de programación más utilizado
- Todos los datos con los que se trabaja son objetos

Un objeto

- Tiene atributos (color, marca)
- Tiene métodos (transportar, arrancar, frenar...)

Pilares

- Abstracción (definir el plano de un objeto)
- Encapsulamiento (evitar que los objetos se manipulen de manera incorrecta)
- Herencia (podemos definir objetos hijos a partir del plano del objeto padre)
- Polimorfismo (hace referencia a cada objeto creado a partir del mismo plano)

Clases e instancias

- Los objetos por sí mismos no tienen propiedades, deben definirse en la clase
- Las propiedades se definen a través de un método llamado init dentro de la clase, es su constructor

```
# clase persona

class Person:
    def __init__(self):
        print("Estoy en el constructor")

desire = Person() # Imprimirá "Estoy en el constructor"
```

Atributos

Las variables que definen las características del objeto.

Hay dos tipos:

- los de instancia (se definen dentro de la función `init`)
- los de la clase (se definen fuera de la función `init`)

Atributos de instancia:

```
# clase persona

class Person:
    def __init__(self, name, age): # atributos de instancia
        self.name = name
        self.age = age

desire = Person("Desire", 28)

print(desire.name) # Imprimirá "Desire"
print(desire.age) # Imprimirá 28
```

Atributos de clase:

```
# clase persona
```

```
class Person:
    species = "human" # atributo de clase

    def __init__(self, name, age):
        self.name = name
        self.age = age

desire = Person("Desire", 28)

print(desire.name) # Imprimirá "Desire"
print(desire.age) # Imprimirá 28
print(desire.species) # Imprimirá "human"
```

Métodos de una clase

- Funciones que se definen dentro de una clase
- Cada objeto creado a partir de esa clase, puede acceder a esos métodos

```
# clase persona

class Person:
    species = "human" # atributo de clase

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def birthday(self):
        self.age += 1
        print(f"Happy birthday, {self.name}! {self.age}")

desire = Person("Desire", 28)
desire.birthday() # Ejecutará "Happy birthday, Desire! 29"
```

Herencia entre clases

- Permite que una clase sea creada a partir de otra
- Hay clases padres y clases hijo
- Las clases hijo heredan atributos y métodos de la clase padre
- Las clases padres e hijas deben estar relacionadas entre sí, por ejemplo: Persona → Empleado

```
# clase persona

class Person:
    species = "human" # atributo de clase

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def birthday(self):
        self.age += 1
        print(f"Happy birthday, {self.name}! {self.age}")

# clase empleado
class Employee(Person):
    def work(self, work_hours):
        print(f"You've been working for {work_hours} hours")

# creating an instance of employee

desire = Employee("Desire", 28)
desire.work(6)
```

Si quisiéramos añadir a la clase empleado una variable de instancia del total de horas que ha trabajado:

```
# clase empleado
class Employee(Person):
    def __init__(self, name, age, total_work_hours):
        super(Employee, self).__init__(name, age)
        self.total_work_hours = total_work_hours

    def work(self, work_hours):
        self.total_work_hours += work_hours
        print(f"You've been working for {work_hours} hours no
        print(f"You've been working for {self.total_work_hour
```

@Desiré Marrón