Universidad Autónoma de Occidente

Fundamentación

Lenguaje Python: Programación Orientada a Objetos(POO)

Sergio A. Cantillo

sacantillo@uao.edu.co

Facultad de Ingeniería y Ciencias Básicas
Universidad Autónoma de Occidente

Agenda

Agenda

1. Introducción
2. Clases y Objetos

Agenda

1. Introducción
2. Clases y Objetos
3. Encapsulamiento

Agenda

1. Introducción
2. Clases y Objetos
3. Encapsulamiento
4. Herencia

Agenda

1. Introducción
2. Clases y Objetos
3. Encapsulamiento
4. Herencia
5. Polimorfismo

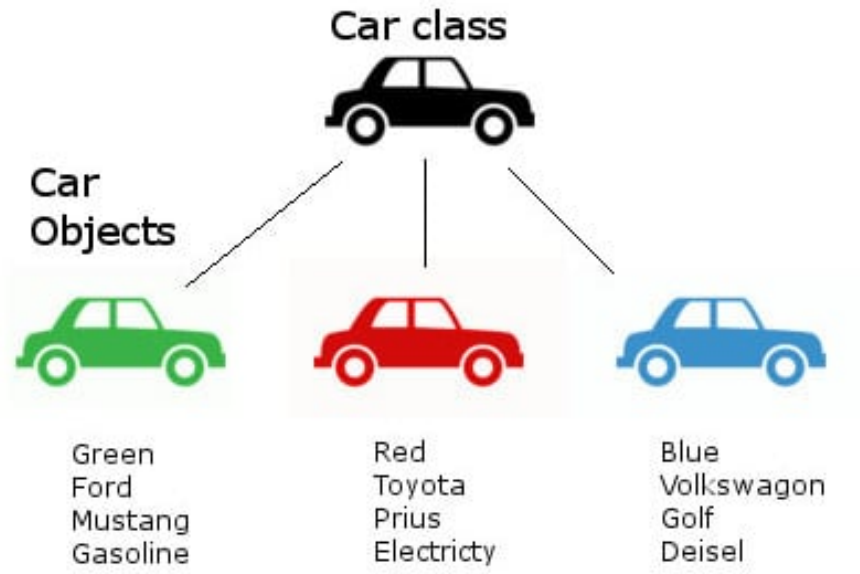
Agenda

1. Introducción
2. Clases y Objetos
3. Encapsulamiento
4. Herencia
5. Polimorfismo
6. Funcion Main

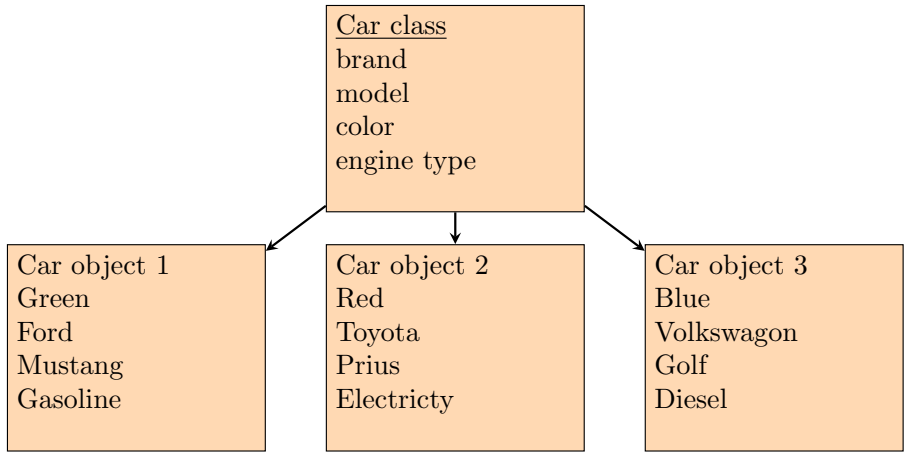
Agenda

Programación Orientada a Objetos

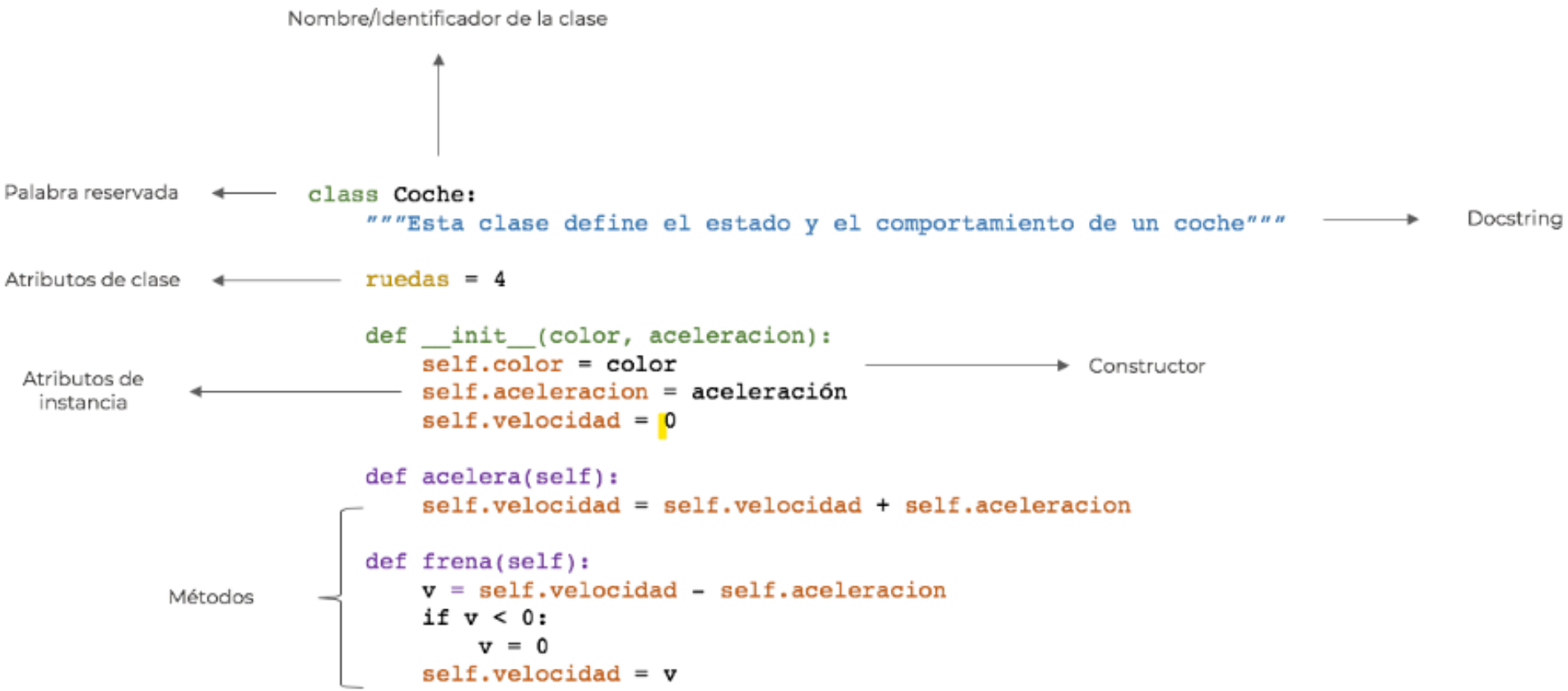
- Paradigma de programación que busca simplificar y representar características esenciales de las entidades del mundo real en forma de **objetos** a partir de **clases**.
- La POO hace mas eficiente y fácil de entender los programas.
- Permite la modularidad y reutilización de código.



Una **clase** es un *molde* o *plantilla* para la creación de objetos, compuesta por **atributos** (propiedades) y métodos (funciones).



Programación Orientada a Objetos



Clases y Objetos en Python

- La elaboración de las clases esta fundamentada en el constructor.
- Los nombres de variables y métodos en la clase que presentan el doble guión bajo al comienzo, surgen de un proceso de **name mangling** para limitar el acceso externo a estas variables.

```
1  class Car():
2      #initializer or constructor
3      puertas = 4
4      def __init__(self, brand, model, color, engine_type, puertas):
5          self.brand = brand
6          self.model = model
7          self.color = color
8          self.engine_type = engine_type
9          print("se creo el objeto",puertas+1)
10
11     def __str__(self) -> str:
12         cadena=self.brand+","+self.model+","+self.color
13         return cadena
14
15
16 car1 = Car("kia", "picanto", "blanco", "gasolina", 8)
17 #car2 = Car("nissan", "sentra", "negro", "diesel")
18
19 print(car1.puertas)
20 print(car1.brand)
21 print(car1)
```

Métodos de una Clase

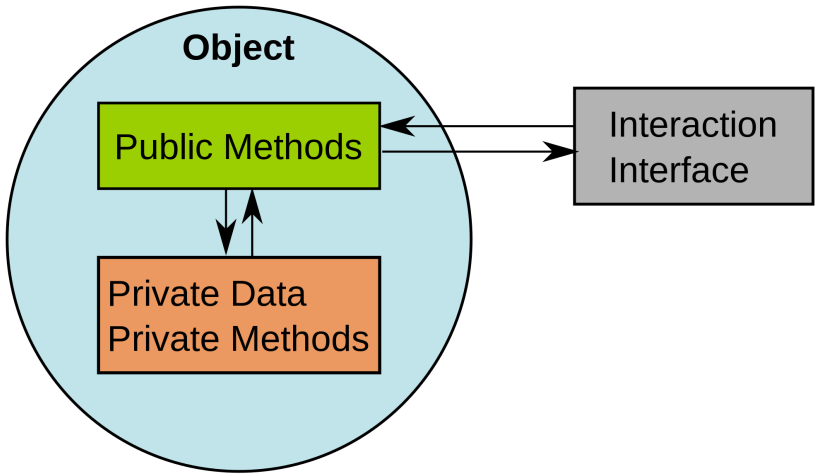
- Los métodos son **funciones** definidas dentro del cuerpo de la clase.
- Los métodos permiten definir el comportamiento de un objeto, es decir, lo que puede hacer.
- Los métodos que presenten doble guión bajo al comienzo como al final de su nombre, son métodos especiales en el contexto de POO.

```
1  class Car():
2      #initializer or constructor
3      def __init__(self, brand, model, color, engine_type):
4          self.brand = brand
5          self.model = model
6          self.color = color
7          self.engine_type = engine_type
8
9      #method definition (parameters)
10     def update_color(self, new_color):
11         self.color = new_color
12
13
14 car = Car("kia", "picanto", "blanco", "gasolina")
15 print(car.color)
16
17 # method invocation (arguments)
18 car.update_color("negro")
19 print(car.color)
```

Encapsulamiento

- El encapsulamiento permite restringir el acceso a métodos y variables de un objeto con el fin de prevenir la modificación directa y externa de los datos.
- Es una propiedad que permite asegurar que el contenido de un objeto se pueda ocultar del mundo exterior dejándose ver lo que cada objeto necesite hacer publico.
- **Ejemplo:** Es posible verificar que solo se asigne un valor entero positivo y menor que 120 a la edad de una persona.

```
1  class Car():
2      #initializer or constructor
3      def __init__(self, brand, model, color,
         engine_type):
4          self.brand = brand
5          self.model = model
6          self.color = color
7          self.engine_type = engine_type
8          self.__maxprice = 900
9
10     def sell(self):
11         print("precio de venta: ",self.__maxprice)
12     # setter method
13     def set_max_price(self, price):
14         self.__maxprice = price
15
16 car = Car("kia", "picanto", "rojo", "gasolina")
17 car.sell()
18
19 # change the price
20 car.__maxprice = 1000
21 car.sell()
22
23 # using setter function
24 car.set_max_price(1000)
25 car.sell()
```



Herencia

- Forma de crear nuevas clases utilizando funciones o atributos de clase(s) ya existente(s) sin necesidad de modificarla(s).
- Propiedad que permite a los objetos heredar comportamientos (variables, funciones) de otras clases a partir de una jerarquía de clases.

```python
#parent class
class User:
    def __init__(self):
        print("user is ready")

    def who_is_this(self):
        print("User")

    def read(self):
        print("read faster")

#child class
class Teacher(User):
    def __init__(self):
        #call super() function
        super().__init__
        print("Teacher is ready")

    def who_is_this(self):
        print("Teacher")

    def write(self):
        print("write faster")

#child class
class Student(User):
    def __init__(self):
        #call super() function
        super().__init__
        print("Student is ready")

    def who_is_this(self):
        print("Student")

    def check(self):
        print("check a lot")

jose = Student()
jose.who_is_this()
jose.read()
jose.check()

luis = Teacher()
luis.who_is_this()
luis.read()
luis.write()
```

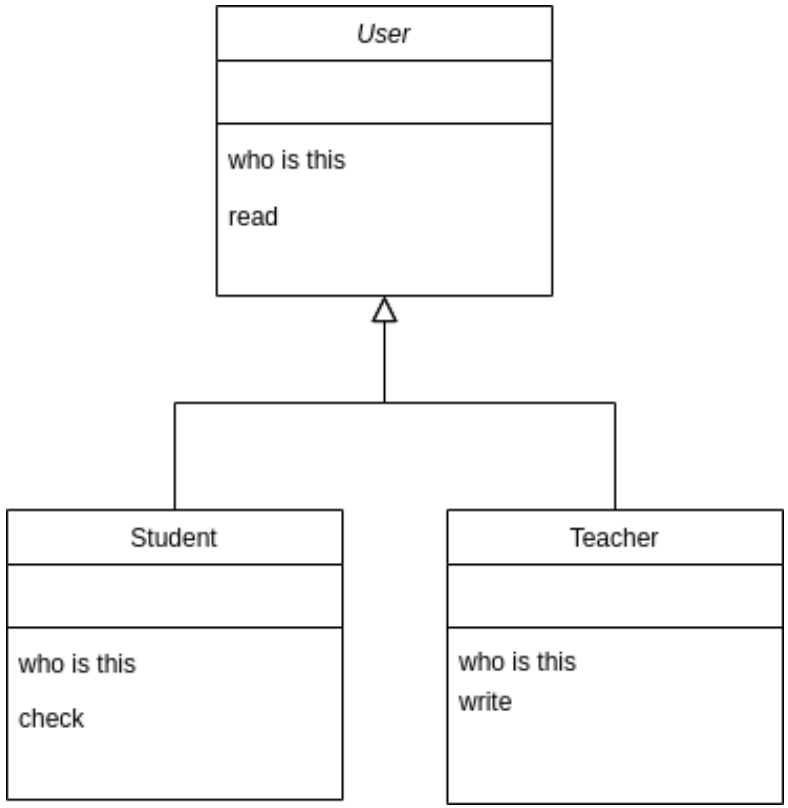


Figura: Diagrama de Clases [1]

Polimorfismo

Propiedad de los objetos de diferentes clases de responder ante el mismo mensaje o método de manera diferente. Esto facilita la reutilización del código y la flexibilidad en el diseño de programas.

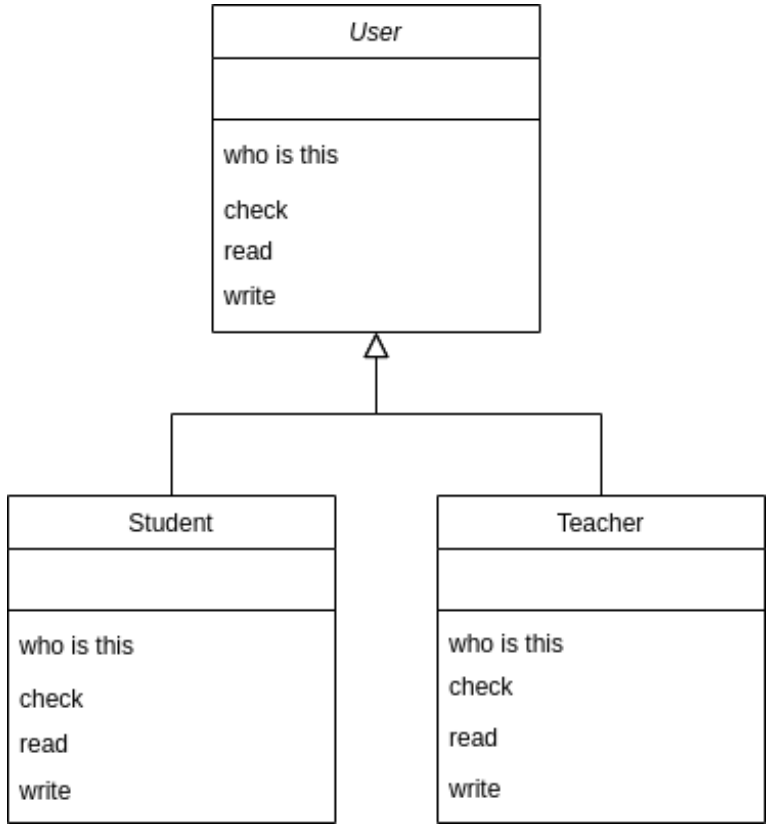


Figura: Diagrama de Clases [1]

Funcion main()

- Un programa de Python no necesita obligatoriamente definir una función **main()** como en otros lenguajes.
- La función **main()** es útil para cuando un archivo python se puede ejecutar directamente, pero alternativamente también se puede importar y reutilizar en otro modulo.

no_main.py

```
1  def add(a, b):
2      return (a + b)
3
4
5  print(add(2, 3))
```

test_no_main.py

```
1  from no_main import add
2
3  print(add(2,4))
```

¿Cuál es el resultado?

main_ex.py

```
1  def add(a, b):
2      return (a + b)
3
4  def substract(a, b):
5      return (a - b)
6
7  def impresion():
8      print(add(2, 3))
9
10 if __name__ == "__main__":
11     impresion()
12     print(add(2, 3))
```

test_main.py

```
1  from main_ex import add
2
3  print(add(2,4))
```

¿Cuál es el resultado?

Referencias


John Hunt.
A Beginners Guide to Python 3 Programming.
Springer, 2019.

`https://www.w3schools.com/python/python_classes.asp`