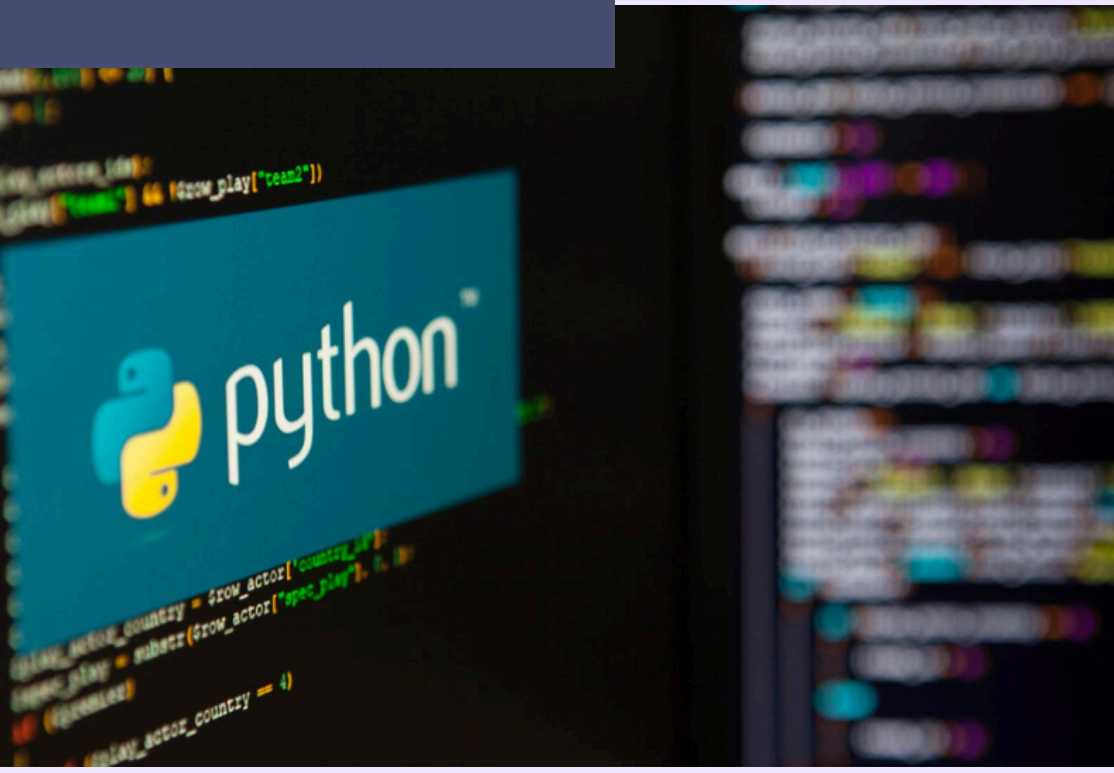
PYTHON

P.ESTRUCTURADA & POO

UD Lógica de Programación

Juan loayza Márquez - Docente DSI



LABORATORIO DE PRÁCTICA01

UD LOGICA DE PROGRAMACIÓN

https://github.com/GrialCompute/Laboratorio_logicaProgramacion

ICICLO - 2025

Programación orientada a objetos (POO)

Esta forma de programar consiste en trasladar la naturaleza de los objetos de la vida real a código de programación, esto recoge todas las funcionalidades y características que tiene un objeto tales como un estado, un comportamiento y sus atributos para así poder crear programas más útiles.
Para dar un ejemplo de objeto usaremos un coche y te preguntaremos lo siguiente:

- ¿Cuál es el estado de un coche? Un coche puede estar parado, circulando, aparcado, etc.
- ¿Qué atributos tiene un coche? Un coche tiene un color, un peso, un tamaño, etc.
- ¿Qué comportamiento tiene un coche? Un coche puede arrancar, frenar, acelerar, girar, etc.

Sabiendo todo esto gracias a la POO puedes trasladarlo al código de un programa y algunos ejemplos de leguajes de POO son: C++, Java, Visual.NET, Python, etc.
La **POO** nos ofrece las siguientes ventajas:

- Programas divididos en “trozos”, “partes”, “módulos” o “clases” permitiendo la modularización del código.
- Muy reutilizable implementando el concepto de herencia.
- Si existe un fallo en alguna línea de código el programa aun así podrá ejecutarse.
- Proporciona la encapsulación un método que permite implementar protección al código para no ser usado fuera de la función donde está escrito.

Conceptos de la POO

Clase

Una clase lo podríamos definir como un módulo donde se redactan las características comunes de un grupo de objetos. Ejemplo: Un objeto es un vehículo y sus clase son los tipos de vehículos tales como aviones, coche, trenes, motos, etc. Cada uno posee características únicas de su clase.

Objeto

Un objeto es un ejemplar perteneciente a una clase. Ejemplo: Si tenemos 2 vehículos un coche y una moto, ambos son objetos y pertenecen a dos clase diferentes, la clase moto y la clase coche.

Atributo

Son las características que tiene una clase.

Comportamiento

Son las acciones u operaciones que un objeto puede realizar.

Modularización
Normalmente cuando creas una aplicación compleja en un leguaje de POO como python es normal que esas aplicaciones estén compuestas de varias clases que trabajan de manera totalmente independiente una de otra y si una de las clases por cualquier razón deja de funcionar el programa no caerá eso es modularización.

Encapsulación
Es una función que impide que el funcionamiento de una clase no pueda ser usado o entendido por otra clase, es decir, por ejemplo: Un equipo de sonido no puede usar o tener las características de una nevera, por lo tanto si llevamos ese ejemplo a código se podría decir que la encapsulación protege la características de cada clase para que no puedan ser usadas fuera de la clase a la que pertenecen.

Habiendo explicado los que es la POO y sus conceptos básicos es hora de ver cómo funciona la POO en **python** y cómo interpretarlos mediante el código.
Para empezar debemos construir en código lo que sería una clase, la cual es la base para poder crear objetos que pertenezcan a esa clase, la sintaxis para construir una clase es escribiendo la palabra **class** seguida del nombre de esa clase con su primera letra en mayúscula seguido en un par de paréntesis "***()***" y dos puntos ***":".***

```
class Coche():
```

Ahora que tenemos una clase procedemos a definir sus atributos, el comportamiento y el estado de esa clase, una clase puede tener una infinidad de atributos tantos como necesitemos. Ya que hemos declarado una clase que se refiere a un objeto de la vida real que es en este caso un coche, procedemos a implementar las siguientes propiedades:

```
class Coche():
    largoChasis=250
    anchoChasis=120
    ruedas=4
    enmarcha=false
```

Ahora que hemos definido los atributos debemos definir su comportamiento el cual viene determinado por lo que se llaman métodos y debemos crear uno para establecer el comportamiento, también podemos establecer infinidad de comportamientos igual que con los atributos, como hemos explicado anteriormente un comportamiento son las operaciones que puede realizar un objeto en este caso es un coche y vamos a definir el método "arrancar", ya que hemos definido que uno de sus atributo es "***enmarcha=false***" que indica que no está en movimiento.
Para crear un método lo hacemos con la palabra ***"def"seguido*** del nombre del método que será *"arrancar"* después unos paréntesis y dentro de ellos la palabra "***self"*** la cual va a hacer referencia al objeto perteneciente a la clase.

```
def arrancar(self):
pass
```

La palabra ***"pass"*** la escribimos dentro del método mientras no le asignemos ningún comportamiento ya que si no lo hacemos nuestro programa tendrá un error al ejecutarse. Ahora para declarar un objeto debemos darle un nombre (el que queramos) junto al nombre de la clase a la que queramos que pertenezca ese objeto.

```
miChoche=Coche()
```

Ahora para acceder a los atributos y los métodos de una clase debamos definirlo escribiendo el nombre del objeto seguido de un punto ***"."*** y el nombre del atributo o método.

Objeto y atributo:
miCoche.largoChasis
Objetos y método:
miCoche.arrancar()
Sabiendo esto ya podemos definir un comportamiento a el objeto **"miCoche"** con los siguientes métodos. Haremos que nuestro atributo **"enmarcha"** que tiene un valor **"False"** que nos indica que nuestro coche está detenido y le indicaremos que se ponga en marcha, para hacer esto debemos establecer dentro del método lo siguiente:

```
def arrancar(self):
self.enmarcha=True
```

Con esto estamos indicando que se cambiaran el valor predeterminado de nuestro atributo ***"enmachar"*** y nuestro objeto coche tendrá la capacidad de ponerse en marcha.
Ahora solo queda comprobar si nuestro objeto está en marcha o no y para eso establecemos otro método al cual podemos denominar ***"estado"*** la cual debemos colocarle un condicional "***if***" de esta manera:

```
def estado(self):
if (self.enmarcha):
return "El coche esta en marcha"
else:
return "El coche esta detenido"
```

Una vez planteándonos todo esto debemos estructurar el código de esta manera:

```
#definir clase
class Coche():
  largoChasis=250
  anchoChasis=120
  ruedas=4
  enmarcha=False
  #definir metodo
  def arrancar(self):
  #definir comportamiento
    self.enmarcha=True
  #definir metodo
  def estado(self):
    #definir comportamiento
    if (self.enmarcha):
   return "El coche esta en marcha"
    else:
      return "El coche esta detenido"


#definir objeto y a que clase pertenece
miChoche=Coche()

#imprimir los siguiente para comprobar si funciona correctamente
print("El largo del coche es: ",miChoche.largoChasis)
print("El coche tiene ", miChoche.ruedas, " ruedas")
miChoche.arrancar()
print(miChoche.estado())
```

Al ejecutar esto nos mostrara la siguiente información:
El largo del coche es: 250
El coche tiene 4 ruedas
El coche esta en marcha

El coche nos muestra que está en marcha porque antes del imprimir el método “estado” le estamos indicando que no ejecute el método “arrancar” que nos cambia el valor del atributo “enmarcha” a “True”, para que nos muestre que “El coche está detenido” comentamos la línea donde nos indica “miCoche.arrancar()” y no imprimirá esto:

El largo del coche es: 250
El coche tiene 4 ruedas
El coche esta detenido

En la POO podemos declarar la cantidad de objetos que queramos cada uno con sus clases, atributos y comportamientos únicos que los diferencia uno de otro y lo podemos hacer de la siguiente manera:

```
#definir clase
class Coche():
  largoChasis=250
  anchoChasis=120
  ruedas=4
  enmarcha=False
  #definir metodo
  def arrancar(self):
    #definir comportamiento
    self.enmarcha=True
 #definir metodo
  def estado(self):
    #definir comportamiento
    if (self.enmarcha):
      return "El coche esta en marcha"
    else:
      return "El coche esta detenido"

#definir objeto y a que clase pertenece
miChoche=Coche()

#imprimir los siguiente para comprobar si funciona correctamente
print("El largo del coche es: ",miChoche.largoChasis)
print("El coche tiene ", miChoche.ruedas, " ruedas")
#miChoche.arrancar()
print(miChoche.estado())

print("----------Segundo objeto----------")
miChoche2=Coche()

#imprimir los siguiente para comprobar si funciona correctamente
print("El largo del coche es: ",miChoche2.largoChasis)
print("El coche tiene ", miChoche2.ruedas, " ruedas")
miChoche2.arrancar()
print(miChoche2.estado())
```

Para poder plantear de mejor manera este código haremos que el método “arrancar” se encargue no solo de arrancar el coche sino también de informarnos cuál es el estado y que el método “estado” nos informe de cada una de los atributos que tiene cada objeto.
Empezamos añadiendo un nuevo parámetro al método “arrancar” el cual vamos a denominar “arrancamos”, eso lo hacemos de la siguiente manera:

```
def arrancar(self,arrancamos)
```

Y establecemos el método “arrancar” de la siguiente manera:

```
def arrancar(self,arrancamos):
  self.enmarcha=arrancamos
  if (self.enmarcha):
   return "El coche esta en marcha"
  else:
   return "El coche esta detenido"
```

Con esto el método puede hacer la función de arrancar y de indicarnos el estado en el que el coche se encuentra es decir si está en marcha o esta detenido.
El método “estado” va a informarnos del estado de cada uno de los otros atributos que tiene el objeto y para eso hacemos lo siguiente:

```
def estado(self):
  #definir comportamiento
  print("El coche tiene ", self.ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)
```

Y de esta manera el método “arrancar” recibe dos parámetros y el método “estado” nos informa del estado de actual de los atributos, ahora procedemos a mostrar en pantalla todo lo anteriormente mencionada de la siguiente manera

```
miChoche=Coche()

print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.estado()
```

Veamos el ejemplo completo y ver lo que nos muestra en pantalla:

```
#definir clase
class Coche():
  largoChasis=250
  anchoChasis=120
  ruedas=4
  enmarcha=False
  #definir metodo
  def arrancar(self,arrancamos):
    #definir comportamiento
    self.enmarcha=arrancamos
    if (self.enmarcha):
      return "El coche esta en marcha"
    else:
      return "El coche esta detenido"
  #definir metodo
  def estado(self):
    #definir comportamiento
    print("El coche tiene ", self.ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)

def estado(self):  a que clase pertenece
  #definir comportamiento
  print("El coche tiene ", self.ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)

print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche=Coche()

print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.estado()
```

Y nos mostrara lo siguiente:
El coche esta en marcha
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250
----------Segundo objeto----------
El coche esta en detenido
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250

Estado inicial

Con todo lo explicado anteriormente está claro que todo los objetos de la clase coche van a tener ciertos atributos en común que van a ser un ancho, largo, tener una cantidad de ruedas y que todos por defecto van a estar detenidos, a la hora de usar la programación orientada a objetos es habitual que los atributos formen parte de un estado inicial, estos quiere decir que queremos que los atributo que tenga la clase sean predeterminados, es decir, que en cuanto creemos un objeto todos sus atributos tengan un estado inicial. Para especificar el estado inicial de los atributos de una clase lo hacemos con un constructor, un constructor es un método que le da estado inicial a los objetos y esto lo hacemos de la siguiente manera:

```
def __init__(self):
  self.largoChasis=250
  self.anchoChasis=120
  self.ruedas=4
  self.enmarcha=False
```

Con la nomenclatura "__init__" estamos especificando que es el método constructor y va a dar un estado inicial a los atributos que pertenezcan a la clase donde lo definas. Si ejecutamos el ejemplo agregándole el constructor se ejecutara de la misma manera de antes:

```
#definir clase
class Coche():
  def __init__(self):
    self.largoChasis=250
    self.anchoChasis=120
    self.ruedas=4
    self.enmarcha=False
  #definir metodo
  def arrancar(self,arrancamos):
    #definir comportamiento
    self.enmarcha=arrancamos
    if (self.enmarcha):
      return "El coche esta en marcha"
    else:
      return "El coche esta detenido"
  #definir metodo
  def estado(self):
    #definir comportamiento
    print("El coche tiene ", self.ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)


#definir objeto y a que clase pertenece
miChoche=Coche()

print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.estado()
```

Deberá mostrarnos el mismo resultado anterior:
El coche esta en marcha
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250
----------Segundo objeto----------
El coche esta en detenido
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250

Encapsulación en Python

Para entender el concepto de encapsulación a la hora de llevarlo a código de python vamos a fijarnos bien en la secuencia de llamadas que estamos haciendo la cual es esta:

```
miChoche=Coche()

print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.estado()
```

Podemos ver que hemos creado dos objetos, al primer objeto “miCoche” le hemos dicho que arranque y hemos mostrado su estado, al segundo objeto “miCoche2” le hemos que no arranque y también hemos mostrado su estado.
Ahora si al objeto “miCoche2” le especificamos que tiene 2 ruedas antes de mostrar su estado al ejecutarlos nos dirá que tiene 2 ruedas, esto podemos hacerlo de la siguiente manera:

```
#definir clase
class Coche():
  def __init__(self):
    self.largoChasis=250
    self.anchoChasis=120
    self.ruedas=4
    self.enmarcha=False
  #definir metodo
  def arrancar(self,arrancamos):
    #definir comportamiento
    self.enmarcha=arrancamos
    if (self.enmarcha):
     return "El coche esta en marcha"
    else:
     return "El coche esta detenido"
  #definir metodo
  def estado(self):
    #definir comportamiento
    print("El coche tiene ", self.ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)
```

```
#definir objeto y a que clase pertenece
miChoche=Coche()


print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.ruedas=2
miChoche2.estado()
```

Nos mostrara en pantalla lo siguiente:
El coche esta en marcha
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250
----------Segundo objeto----------
El coche esta en detenido
El coche tiene 2 ruedas. Un ancho 120 y un largo de 250

Si utilizamos la lógica y llevamos este ejemplo a un caso de la vida real no tiene sentido un coche con 2 ruedas o más de 4 ruedas no tendría sentido y no se debería permitir que el programa ejecute algo que no tiene sentido.
Te preguntaras que es lo que se puede hacer en estos casos y es aquí donde entra el concepto de encapsulación. La encapsulación se encarga de proteger un atributo para que no se pueda modificar desde afuera de la clase, cómo pudiste ver el atributo ruedas se encuentra dentro del constructor y nosotros lo hemos modificado fuera de la clase y eso no debe ocurrir. Para evitar esto debemos encapsular ese atributo y eso lo hacemos precediendo su nombre de dos guiones bajos:

```
def __init__(self):
  self.largoChasis=250
  self.anchoChasis=120
  self.__ruedas=4 <----------------
  self.enmarcha=False
```

Ahora estas indicando que al atributo ruedas no sea accesible desde el exterior pero si será accesible desde dentro de la clase y ahora en adelante si estas encapsulando el atributo ruedas siempre debes especificarlo con los dos guiones bajos tal y como lo declaraste dentro de la clase.
Si ejecutamos el código de la siguiente manera veras que el valor del atributo ruedas no será modificado:

```
#definir clase
class Coche():
  def __init__(self):
    self.largoChasis=250
    self.anchoChasis=120
    self.__ruedas=4
    self.enmarcha=False
  #definir metodo
  def arrancar(self,arrancamos):
    #definir comportamiento
    self.enmarcha=arrancamos
    if (self.enmarcha):
      return "El coche esta en marcha"
    else:
      return "El coche esta detenido"
  #definir metodo
  def estado(self):
    #definir comportamiento
    print("El coche tiene ", self.__ruedas, " ruedas. Un ancho de ", self.anchoChasis, " y un larde de ", self.largoChasis)

#definir objeto y a que clase pertenece
miChoche=Coche()


print(miChoche.arrancar(True))
miChoche.estado()

print("----------Segundo objeto----------")

miChoche2=Coche()

print(miChoche2.arrancar(False))
miChoche2.ruedas=2
miChoche2.estado()
```

En pantalla se mostrara de la siguiente manera:
El coche esta en marcha
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250
----------Segundo objeto----------
El coche esta en detenido
El coche tiene 4 ruedas. Un ancho 120 y un largo de 250
Para concluir como puedes ver hemos aplicado la encapsulación para proteger el un parte del código de nuestro programa, la encapsulación puede usarse para todos los atributos y comportamiento que tiene una clase solo mientras creas que sea necesario.