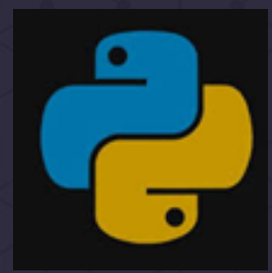
PREGRADO

UNIDAD 2 | SEMANA 6

PROGRAMACIÓN ORIENTADA A OBJETOS EN PYTHON



1ACC0201| Programación Orientada a Objetos

Logro

Al finalizar la sesión, el estudiante adquiere el entendimiento necesario para aplicar la Programación Orientada a Objetos en Python.

Programación Orientada a Objetos en Python

- **Python**, como lenguaje de programación, permite la utilización de diversos paradigmas, incluyendo la Programación Orientada a Objetos.
- **Python** ha sido desarrollado con **Orientación a Objetos**, lo que significa que en Python, **todo se considera un objeto**. Por lo tanto, al crear una variable y asignarle un valor numérico, ese valor se transforma en un objeto; lo mismo aplica para las funciones, así como para listas, tuplas, diccionarios, conjuntos y cadenas, todos ellos considerados también como objetos.
- Consideremos las siguientes definiciones:
  - **Clase**: un modelo o plantilla.
  - **Método o mensaje**: una función que se define dentro de una clase.
  - **Campo o atributo**: los datos que pertenecen a una clase.
  - **Objeto**: una instancia particular de una clase.

Creando una clase

- Para definir una clase en Python, se emplea la palabra clave “class” seguida del nombre de la clase y dos puntos.

```
class Clase:
    # Código de la clase aquí
    pass
```

- De acuerdo con la PEP 8 – Guía de estilo para código, se sugiere que el nombre de la clase comience con una letra mayúscula, mientras que las siguientes deben ser en minúscula.

Creando un objeto a partir de una clase.

- Tenemos la siguiente clase

```
class Laptop():
    """
    Clase empleada para trabajar con Laptops
    """
    tiene_ssd = True


# Programa Principal
laptop1 = Laptop()
```

- Se ha creado una instancia de la clase Laptop utilizando la clase Laptop.
- laptop1 es un ordenador portátil equipado con un disco de almacenamiento SSD.

Constructor en Python

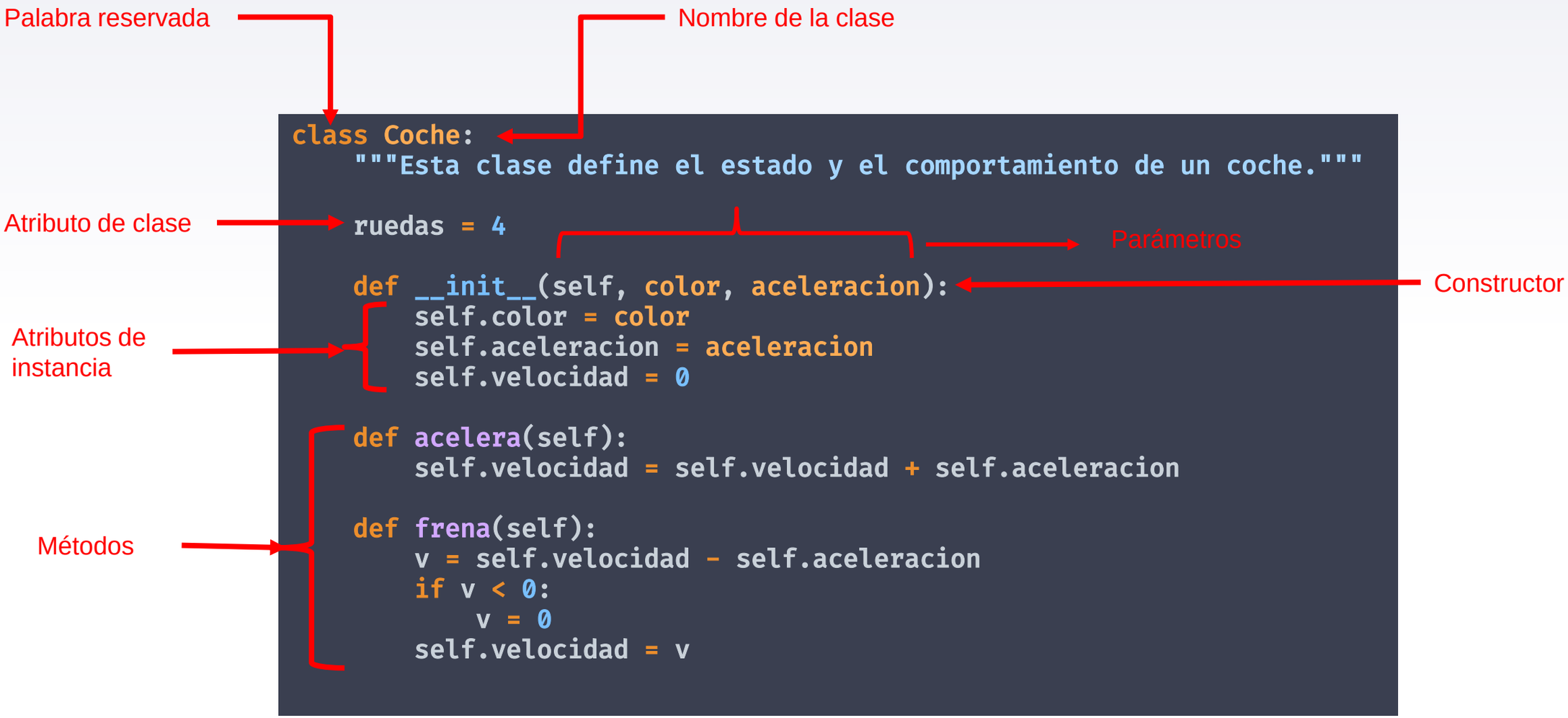
- En Python, un **constructor** es un método especial que se invoca al crear un objeto.
- La función principal de un **constructor** en Python es asignar valores a los **atributos** de la clase al momento de inicializar un objeto.
- El nombre del **método constructor** es siempre **init** (note que tiene dos guiones bajos al principio y al final).
- El método **init** puede aceptar cualquier número de **parámetros**, y al igual que las funciones, estos **pueden ser definidos con valores por defecto**, lo que los hace opcionales para quien los llama.

El parámetro self

- En Python, todos los métodos que se encuentran dentro de una clase comienzan con un parámetro denominado **self**, seguido de los parámetros de inicialización. Este parámetro actúa como una referencia a la instancia actual de la clase.
- Aunque se puede utilizar un nombre diferente si se prefiere, la convención establece el uso de **self**. Esta no es una palabra reservada en Python.
- **self** se emplea **dentro de la clase** para **acceder** a cada uno de los **atributos** en los diferentes **métodos** que se desarrollarán dentro de la clase.
- Aunque es necesario incluir **self** de manera explícita al definir el método, no es necesario especificarlo al llamar al método; Python lo añadirá automáticamente.

Estructura de una clase en Python

- Veamos el siguiente ejemplo




```python
class Coche:
    """Esta clase define el estado y el comportamiento de un coche."""

    ruedas = 4

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

    def acelera(self):
        self.velocidad = self.velocidad + self.aceleracion

    def frena(self):
        v = self.velocidad - self.aceleracion
        if v < 0:
            v = 0
        self.velocidad = v
```

Estructura de una clase

```
class Coche:
    """Esta clase define el estado y el comportamiento de un coche."""

    def __init__(self, color, aceleracion):
        self.color = color
        self.aceleracion = aceleracion
        self.velocidad = 0
        self. ruedas = 4

    def acelera(self):
        self.velocidad = self.velocidad + self.aceleracion

    def frena(self):
        v = self.velocidad - self.aceleracion
        if v < 0:
            v = 0
        self.velocidad = v
```

- El diagrama presentado previamente ilustra la clase Coche.
- Esta clase establece diversos atributos, tales como ruedas, color, aceleración y velocidad.
- También incorpora las funciones acelera() y frena().

Creación de objetos

```
class Coche:
    """Esta clase define el estado y el comportamiento de un coche."""

    def __init__(self, color, aceleracion):
        self.color = color
        self.aceleracion = aceleracion
        self.velocidad = 0
        self. ruedas = 4

    def acelera(self):
        self.velocidad = self.velocidad + self.aceleracion

    def frena(self):
        v = self.velocidad - self.aceleracion
        if v < 0:
            v = 0
        self.velocidad = v


# Programa Principal
coche1 = Coche("rojo", 20)
print(f"{coche1.color} {coche1.ruedas}")
```

- coche1 es una instancia generada a partir de la clase Coche.
- coche1 cuenta con atributos como ruedas, color y aceleración.
- coche1 tiene disponibles los métodos acelera y frena.
- Al instanciar el objeto coche1, se activa el constructor.
- En este caso, el constructor establece los atributos color, aceleración y velocidad.

Creación de objetos

```
# Programa Principal
coche1 = Coche("rojo", 20)
print(f"{coche1.color} {coche1.ruedas}")

coche1.acelera()
print(f"{coche1.velocidad} {coche1.aceleracion}")
coche1.acelera()
print(f"{coche1.velocidad} {coche1.aceleracion}")

coche1.aceleracion = 10
coche1.frena()
print(f"{coche1.velocidad} {coche1.aceleracion}")
```

rojo 4
20 20
40 20
30 10

- A continuación, se ilustra la ejecución de los métodos pertenecientes al objeto coche1, y se muestra la actualización de uno de sus atributos.
- En secciones posteriores, se explorará el concepto de encapsulación, donde los atributos serán protegidos y solo podrán ser modificados a través de métodos desarrollados para tal fin.

Ejercicio 1

- Se requiere la implementación de una clase Punto en Python, diseñada para modelar un punto en el plano cartesiano. La clase deberá cumplir con las siguientes especificaciones:
- Atributos:
  - x: Representa la coordenada horizontal del punto.
  - y: Representa la coordenada vertical del punto.
- Métodos:
  - __init__(self, x, y): Constructor que inicializa los atributos x e y con los valores proporcionados.
  - determinar_cuadrante(self): Método que evalúa los valores de x e y para determinar el cuadrante del plano cartesiano en el que se ubica el punto. El método deberá imprimir el resultado de la evaluación.

```python
class Punto:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def imprimir(self):
        print("Coordenada del punto")
        print(f"({self.x}, {self.y})")

    def imprimir_cuadrante(self):
        if self.x > 0 and self.y > 0:
            print("Primer cuadrange")
        elif self.x < 0 and self.y > 0:
            print("Segundo cuadrante")
        elif self.x < 0 and self.y < 0:
            print("Tercer cuadrante")
        else:
            print("Cuarto cuadrante")


# Programa principal
punto1 = Punto(10, -2)
punto1.imprimir()
punto1.imprimir_cuadrante()
```

Coordenada del punto
(10, -2)
Cuarto cuadrante

Ejercicio 2

- Se requiere el desarrollo de un programa en Python que defina la clase Esfera. Esta clase deberá modelar una esfera geométrica y proporcionar métodos para calcular sus propiedades fundamentales:
- Atributos:
  - Radio: representa el radio de la esfera.
- Métodos:
  - Cálculo del área: un método que calcule y retorne el área de la superficie de la esfera.
  - Cálculo del volumen: un método que calcule y retorne el volumen de la esfera.

```python
import math


class Esfera:
    def __init__(self, radio):
        self.radio = radio

    def area(self):
        print(f"El área de la esfera es {round(self.radio**2 * math.pi, 4)}")

    def volumen(self):
        print(
            f"El volumen de la esfera es "
            f"{round(self.radio**3 * math.pi * 4 / 3, 4)}"
        )


# Programa principal
esfera1 = Esfera(7)
esfera1.area()
esfera1.volumen()
```

Ejercicio 3

Se requiere la implementación de un programa en Python que cree las clases Banco y Cuenta.

- La clase Cuenta deberá definir los atributos numCta (cadena) y saldo (numérico), y un método constructor para inicializarlos.
- La clase Banco deberá contener un método transferirDinero(ctaOrigen, ctaDestino, monto), que realice la transferencia del monto desde ctaOrigen hacia ctaDestino, siempre que ctaOrigen tenga saldo suficiente.

El programa deberá:

- Crear dos instancias de Cuenta con valores iniciales diferentes.
- Crear una instancia de Banco.
- Invocar el método transferirDinero() de la instancia de Banco, pasando las instancias de Cuenta y el monto a transferir.
- Mostrar los saldos de las cuentas antes y después de la transferencia.

Ejercicio 3

```python
class Cuenta:
    def __init__(self, numCta, saldo):
        self.numCta = numCta
        self.saldo = saldo

    def deposito(self, monto):
        self.saldo += monto

    def retiro(self, monto):
        self.saldo -= monto

    def imprimeSaldo(self):
        print(f"Saldo de la cuenta {self.numCta} es {self.saldo}")
```

Esta es la clase cuenta, tiene tres métodos, además del constructor

Ejercicio 3

```python
class Banco:
    def __init__(self, nombre):
        self.nombre = nombre

    def transferirDinero(self, ctaOrigen, ctaDestino, monto):
        if ctaOrigen.numCta != ctaDestino.numCta:
            if ctaOrigen.saldo >= monto:
                ctaOrigen.retiro(monto)
                ctaDestino.deposito(monto)
                print("Transferencia exitosa")
            else:
                print(f"Fondos insuficientes en cuenta {ctaOrigen.numCta} ")
        else:
            print("No se puede transferir hacia si mismo")
```

Esta es la clase banco

Ejercicio 3

```
# Programa Principal
banco = Banco("El usurero")
print(f"\nBanco: {banco.nombre}")
cuenta1 = Cuenta(123, 1000)
cuenta1.imprimeSaldo()
cuenta2 = Cuenta(345, 300)
cuenta2.imprimeSaldo()
banco = Banco("El usurero")
banco.transferirDinero(cuenta1, cuenta2, 300)
print(f"\nBanco: {banco.nombre}")
cuenta1.imprimeSaldo()
cuenta2.imprimeSaldo()
```

Se crean dos objetos cuenta, luego se crea el objeto banco. Se invoca el método transferirDinero del objeto banco y se envía como parámetros las cuentas y el monto a transferir

Banco: El usurero
Saldo de la cuenta 123 es 1000
Saldo de la cuenta 345 es 300
Transferencia exitosa

Banco: El usurero
Saldo de la cuenta 123 es 700
Saldo de la cuenta 345 es 600

El método __str__

- Veamos el siguiente ejemplo

```
class Persona:
    def __init__(self, nom, ape):
        self.nombre = nom
        self.apellido = ape

    def __str__(self):
        cadena = f"{self.nombre} {self.apellido}"
        return cadena


persona1 = Persona("Robert", "Fischer")
print(persona1)
```

- Vemos que en el programa la función principal usa print y manda a “imprimir el objeto”.
- Si se definió el método __str__, entonces en ese caso este devolverá un string, el cual será el que use el print
- El método __str__ se ejecutará sin ser llamado directamente cuando se envía un objeto como parámetro, aun cuando también se puede ejecutar si se le llamada de manera específica

El método destructor __del__

- Veamos el siguiente ejemplo

```
class Pelicula:
    def __init__(self, titulo, duracion, lanzamiento):
        self.titulo = titulo
        self.duracion = duracion
        self.lanzamiento = lanzamiento
        print(f"Se ha creado la película {self.titulo}")

    def __del__(self):
        print(f"Se ha borrado la película {self.titulo}")

    def __str__(self):
        return (
            f"{self.titulo} lanzada el {self.lanzamiento} con una duración de "
            f"{self.duracion} minutos"
        )


p1 = Pelicula("Misión imposible 5 - Nación secreta", 132, 2015)
print(p1)
p1 = Pelicula("El Padrino I", 180, 1972)
print(p1)
```

El método destructor __del__

- El resultado del código anterior será

```
Se ha creado la película Misión imposible 5 - Nación secreta
Misión imposible 5 - Nación secreta lanzada el 2015 con una duración de 132 minutos
Se ha creado la película El Padrino I
Se ha borrado la película Misión imposible 5 - Nación secreta
El Padrino I lanzada el 1972 con una duración de 180 minutos
Se ha borrado la película El Padrino I
```

- Se puede observar que el método destructor __del__ se ha ejecutado automáticamente eliminando el primer objeto creado, cuando su etiqueta p1, fue usada para otro objeto, es decir cuando el objeto ya no tiene referencias

Variables de clase y variables de instancia

- Una variable de clase es única y compartida por todas sus instancias.
- Una variable de instancia es exclusiva y particular de cada instancia.
- En Python, las variables de clase se definen fuera de los métodos y las de instancia dentro de ellos.

```
class Perro:
    tipo = "canino"  # Variable de clase que comparten las instancias

    def __init__(self, nombre):
        self.nombre = nombre  # Variables de instancia, únicas en cada instancia


d = Perro("Roc")
e = Perro("Luna")

print(d.nombre, d.tipo)
print(e.nombre, e.tipo)
```

Roc canino
Luna canino

PREGRADO

Ingeniería de Sistemas de Información

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC

Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco

Lima 33 - Perú

T 511 313 3333

https://www.upc.edu.pe

exígete, innova

LAUREATE
INTERNATIONAL
UNIVERSITIES