Programación Orientada a Objeto (POO)

- POO:
  - Es la evolución natural de la programación estructurada. Se basa en una representación más cercana a como expresaríamos las cosas en la vida real.
  - Se basa en dividir el programa en pequeñas unidades lógicas de código. A estas **pequeñas unidades lógicas de código se les llama objetos**.
  - Los objetos son estructuras independientes que **combinan datos** o estados **(variables) con** acciones asociadas o **comportamiento (métodos)**.
  - La lógica de negocio se modela e implementa mediante una serie de objetos que **interactúan entre si.**
  - Este enfoque aumenta la capacidad para administrar la complejidad del software, lo cual resulta especialmente importante cuando se desarrollan y mantienen aplicaciones y estructuras de datos de gran tamaño.

Programación Orientada a Objeto (POO)

- Características principales:
  - **Abstracción**: Los objetos permiten modelar las características esenciales y el comportamiento de entidades reales sin revelar "cómo" se implementan.
  - **Encapsulamiento**: Consiste en agrupar en una clase las características y el comportamiento de un mismo nivel a abstracción.
  - **Ocultamiento**: Es la capacidad de ocultar los detalles internos de una clase y exponer sólo los detalles que sean necesarios para el resto del sistema.
  - **Polimorfismo:** Se refiere a la propiedad de invocar acciones sintácticamente iguales a objetos de tipos distintos.

Programación Orientada a Objeto (POO)

- Características principales:
  - **Herencia**:
    - Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
  - **Recolección de basura**:
    - Es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente los objetos que hayan quedado sin ninguna referencia a ellos.
    - Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno asignará memoria al crear un nuevo objeto y la liberará la memoria cuando nadie esté usando el objeto.

POO en Python

- Características principales:
  - **Abstracción**: soportado
  - **Encapsulamiento**: soportado
  - **Ocultamiento**: brinda está característica por convención.
  - **Polimorfismo:** natural por su enfoque dinámico (enlazado tardío).
  - **Herencia:** soporta herencia simple y múltiple
  - **Recolección de basura**: soportado

POO: Clases

- Una clase es un tipo de dato definido por el usuario. La clase es como el plano (la definición) de los objetos. Sintaxis:
- **# Clase base:**

```
class <NombreDeLaClase>:
    """documentación"""
    <Sentencias de la clase>
```

- **# Herencia (simple y multiple)**

```
class <NombreDeLaClase>(<ClaseBase1, ..., ClaseBaseN>):
    """documentación"""
    <Sentencias de la clase>
```

POO: Clases

- Crear una clase:

```
class MiClase:
  x = 5
```

- Crear un objeto:

```
o = MiClase()
print(o.x)
```

- Borrar un objeto:

```
del o
```

- Todo es un objeto:

```
[1]: a=5
     type(a)

[1]: int

[4]: a=int(5)
     type(a)

[4]: int
```

POO: Clases: __init__()

- Todas las clases tienen una función llamada __init __ (), que siempre se ejecuta cuando se instancia la clase.
- Se le llama “constructor” y se usa para asignar valores a los atributos del objeto u otras operaciones que sean necesarias cuando se crea el objeto.
- Nota: El parámetro self es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

```
class Persona:
  def __init__(self, nombre, edad):
    self.nombre = nombre
    self.edad = edad

p1 = Persona("Carlos", 36)
print(p1.nombre)
print(p1.edad)
```

POO: Clases: Métodos

- Los objetos pueden contener métodos (son funciones que pertenecen al objeto).
- Todos los métodos de clase deben tener el argumento adicional self como primer argumento.

```
class Persona:
  def __init__(self, nombre, edad):
    self.nombre = nombre
    self.edad = edad

  def mimetodo(self):
    print("Hola, mi nombre  es " + self.nombre)

p1 = Persona("Carlos", 36)
p1.mimetodo()
```

POO: Clases: Variables

- Las variables del objeto se pueden modificar o borrar de la siguiente manera:

```
class Persona:
  def __init__(self, edad):
    self.edad = edad
p1 = Persona(36)
# Modificar
p1.edad = 40
# Borrar
del p1.edad
```

POO: Clases: Variables

- Dentro de una clase hay dos tipos principales de variables:
  - Variables de la clase:
    - Tienen el mismo valor en todas las instancias de la clase (es decir, variables estáticas)
    - Se accede a la variable de la clase usando:

      <NombreDeLaClase>.<NombreDeLaVariable>
  - Variables de la instancia:
    - Suelen tener valores diferentes en todas las instancias de la clase.
    - Se accede a la variable de la instancia usando

      <NombreDeLaInstancia>.<NombreDeLaVariable>

POO: Clases: Variables

```
class Auto:
    # Variable de la clase
    ruedas = 4

    def __init__(self, marca):
        # Variable de la instancia
        self.marca = marca

auto1= Auto("Renault")
auto2= Auto("Suzuki")
print(f"Marca: {auto1.marca}, Ruedas: {Auto.ruedas}")
print(f"Marca: {auto2.marca}, Ruedas: {Auto.ruedas}")
```

POO: Clases: Variables

- Cuando se crean variables de instancia realmente se generan dos, una de la clase y otra de instancia con el mismo nombre. Y cual se está usando depende de como se acceda.

```
class Auto:
    ruedas = 4
    def __init__(self, marca):
        self.marca = marca


instancia= Auto("Renault")
instancia.ruedas = 5
print(instancia.ruedas)
print(Auto.ruedas)
```

POO: Encapsulamiento

- Nada en Python es privado. Todos los atributos de Python (variables y métodos) son públicos.
- Por convención se usa un solo guión bajo antes del nombre para señalar que es un atributo internos y no debería usarse externamente.

```
class Auto:
    def __init__(self, marca):
        self._marca = marca
auto= Auto("Renault")
auto._marca
```

POO: Encapsulamiento

- Un doble guión bajo __ al principio de una variable lo “hace privado”. Da una fuerte sugerencia de no tocarlo desde fuera de la clase. Cualquier intento de acceder dará como resultado un AttributeError ya que python cambia el nombre al siguiente formato:

  <_NombreDeLaClase>__<NombreDeLaVariable>.

```
class Auto:
    def __init__(self, marca):
        self.__marca = marca
auto= Auto("Renault")
auto._Auto__marca
```

POO: Herencia

- La herencia nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.
  - La clase padre es la clase de la que se hereda, también llamada clase base.
  - La clase hija es la clase que hereda de otra clase, también llamada clase derivada

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
    def printNombre(self):
        print(self.nombre)
class Estudiante(Persona):
    pass
x = Estudiante("Alejandro")
x.printNombre()
```

POO: Herencia

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

class Estudiante(Persona):
    def __init__(self, nombre, fecha): # Constructor propio
        super().__init__(nombre) # Para mantener la herencia
        self.fechagraduacion = fecha
    def welcome(self):
        print(f"Bienvenido {self.nombre} ({self.fechagraduacion})")

x = Estudiante("Alejandro", "2019")
x.welcome()
```

POO: Herencia Múltiple

```
class A:
    def a(self):
        print('b')
class B:
    def b(self):
        print('c')
class C(A, B):
    def c(self):
        print('d')
c = C()
c.a()
c.b()
c.c()
```