PROGRAMACIÓN ORIENTADA A OBJETOS(POO)

El lenguaje Python tiene la característica de permitir programar con las siguientes metodologías:

- **Programación Lineal:** Es cuando desarrollamos todo el código sin emplear funciones. El código es una secuencia lineal de comando.
- **Programación Estructurada:** Es cuando planteamos funciones que agrupan actividades a desarrollar y luego dentro del programa llamamos a dichas funciones que pueden estar dentro del mismo archivo (módulo) o en una librería separada.
- **Programación Orientada a Objetos:** Es cuando planteamos clases y definimos objetos de las mismas (Este es el objetivo de los próximos conceptos, aprender la metodología de programación orientada a objetos y la sintaxis particular de Python para la POO)

PROGRAMACIÓN ORIENTADA A OBJETOS

Dos paradigmas: Estructurado u orientado a procedimientos C
Basic
Cobol
Pascal

Desventajas:
Muchas líneas de Código
Pocos reutilizables
Difícil depurar en caso de errores o actualizaciones:

"La POO es la *representación del mundo real en código de programación*"

Conceptos básicos de Objetos

Ejemplo de objetos: Silla, Mesa, Coche, Persona, Animal, etc.

Todos los objetos tienen un **estado** y un **comportamiento**:

Ejemplo:
El **objeto** Persona
Tiene las siguientes **atributos o propiedades**:

Peso
Altura
Raza
Religión
Nacionalidad

Tiene los siguientes **métodos o comportamientos**:
Trabajar
Estudiar

Viajar
Dormir

Lenguajes que soportan el paradigma de POO:

C++
Java
JavaScript
Visual.NET
Python
PHP

Ventajas de la POO:

Mejor Modularización
Mejor reutilización de Código
Posee herencia
Mejora el manejo de errores Encapsulamiento Polimorfismo
Tiene vocabulario propio…
ejemplos:

Clases
Objetos
Modularización
Encapsulamiento
Herencia
Polimorfismo
Instancia

CLASES:

Las clases actúan como plantillas que se utilizan para construir instancias o ejemplos de una clase de cosas. Como ejemplo, cada instancia de la clase Persona puede tener un nombre, una edad, una dirección, etc., pero cada instancia tiene sus propios valores para el nombre, edad y dirección etc.

Podríamos representar a las personas de una familia instanciando al padre, con sus atributos y a la madre con sus atributos, ambos pertenecen a la clase **Persona,** pero son objetos (instancias) distintas**. (Instanciar quiere decir crear)**

Las clases permiten a los programadores especificar la estructura (es decir, sus atributos o campos, etc.) y su comportamiento para todos los objetos de esa clase.

Por ejemplo, al definir la clase Persona podríamos darle:

Un campo o atributo para el nombre de la persona,
Un campo o atributo para su edad,
Un campo o atributo para su correo electrónico,
Algunos comportamientos como cumpleaños (lo que aumentará su edad),
Permitir enviarles y recibir un mensaje por correo electrónico, etc.

En Python las clases se utilizan como:

- Plantilla para crear instancias (u objetos) de esa clase
- Definir métodos de instancia o comportamiento común para una clase de objetos
- Definir atributos o campos para contener datos dentro de los objetos
- Enviar mensajes entre objetos.

Por otro lado, los objetos, pueden:

- Ser creados a partir de una clase,
- Mantener sus propios valores para variables de instancia
- Recibir mensajes
- Ejecutar métodos de instancia.
- Tener muchas copias en el sistema (todas con sus propios datos).

En Python todo es un objeto y como tal, es un ejemplo de un tipo o clase de cosas. Por ejemplo, los enteros son de la **clase *int***, los números reales son ejemplos de la **clase *float***, etc. Esto se ilustra a continuación para varios tipos diferentes dentro de *Python:*

```
print(type(4))
print(type(5.6))
print(type(True))
print(type('Buen día'))
print(type([1, 2, 3, 4]))
```

Salidas:

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
<class 'list'>
```

Definición de Clases en Python:

La programación orientada a objetos se basa en la definición de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (lo que conocemos como variables) y métodos (lo que conocemos como funciones).

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Siempre conviene buscar un nombre de clase lo más próximo a lo que representa. La palabra clave para declarar la clase es ***class***, seguidamente el nombre de la clase y luego dos puntos.

Los métodos de una clase se definen utilizando la misma sintaxis que para la definición de funciones.

Como veremos todo método tiene como primer parámetro el identificador ***self*** que tiene la referencia del objeto que llamó al método.

El método **constructor**, al igual que todos los métodos de cualquier clase, recibe como primer parámetro a la instancia sobre la que está trabajando. Por convención a ese primer parámetro se lo suele llamar ***self*** (que podríamos traducir *como yo mismo*), pero puede llamarse de cualquier forma.

Para referirse a los **atributos de objetos** hay que hacerlo a partir del objeto ***self***.

Luego dentro del método diferenciamos los atributos del objeto antecediendo el identificador ***self:***

```
class Personal():
    #atributos
    nombre = "Pedro"
    edad = 20
    altura = 1.75
```

Es recomendable guardar una clase en un archivo con el nombre de esa clase. Por ejemplo, el código anterior se almacenaría en un archivo llamado *Persona.py*;

Una aplicación normalmente está compuesta por varias Clases

En resumen:

Palabra reservada ***class***
Llamamos **clase** a la representación abstracta de un concepto. Por ejemplo, “perro”, “número entero” o “servidor web”.

Las clases se componen de **atributos** y **métodos**.

Un **objeto** es cada una de las instancias de una clase.
Los **atributos** definen las características propias del objeto y modifican su estado.

Son datos asociados a las clases y a los objetos creados a partir de ellas.

Un **atributo de objeto** se define dentro de un método y pertenece a un objeto determinado de la clase instanciada.

Los **métodos** son bloques de código (o funciones) de una clase que se utilizan para definir el comportamiento de los objetos.

Atributos de objetos

Para definir **atributos de objetos**, basta con definir una variable dentro de los métodos, es una buena idea definir todos los atributos de nuestras instancias en el **constructor**, de modo que se creen con algún valor válido.

ENCAPSULAMIENTO.

Se refiere a que un objeto oculta la implementación de sus **métodos**,además, una clase oculta su implementación a las demás clases. (**clases** encapsuladas)

Las clases se comunican (o conectan) entre si a través de sus métodos

Es conveniente (estándar) que los nombres de las clases comiencen con mayúscula

MÉTODO CONSTRUCTOR INIT

El método constructor define todos los atributos que tiene la clase.
El objetivo fundamental del método __init__ es inicializar los atributos del objeto que creamos.

Básicamente el método __init__ remplaza al método inicializar que habíamos hecho en el concepto anterior.

Las ventajas de implementar el método __init__ en lugar del método inicializar son:

1. El método __init__ es el primer método que se ejecuta cuando se crea un objeto.
2. El método __init__ se llama automáticamente. Es decir, es imposible de olvidarse de llamarlo ya que se llamará automáticamente.

Otras características del método __init__ son:

- Se ejecuta inmediatamente luego de crear un objeto.
- El método __init__ no puede retornar dato.

- el método __init__ puede recibir parámetros que se utilizan normalmente para inicializar atributos.
- El método __init__ es un método opcional, de todos modos, es muy común declararlo

```
def __init__([parámetros]):
    [algoritmo]
```

Se puede mezclar el orden de la definición de atributos y métodos según sea necesario dentro de una sola clase.

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

Como hemos visto anteriormente los atributos de objetos no se crean hasta que no hemos ejecutado el método.
Tenemos un método especial, llamado **constructor __init__,** que nos permite inicializar los atributos de objetos.
Este método se llama cada vez que se crea una nueva instancia de la clase (objeto).