## Python Object Oriented Programming (OOP)

Object-Oriented Programming (OOP) es muy útil porque nos permite escribir un bloque de código y reutilizarlo creando diferentes clases e instancias de esas clases. Cada una de estas clases puede agregar nuevas funcionalidades sin necesidad de reescribir, cambiar o modificar el código existente. Los programas complejos requieren el uso de clases y objetos porque podemos dividir el código en múltiples partes manejables, lo que facilita su organización, mantenimiento y expansión a lo largo del tiempo.

### *Los principios de la Programación Orientada a Objetos*

Los siguientes principios son generalmente aplicables a todos los lenguajes de programación orientados a objetos:

- **Abstracción**: Permite representar ideas complejas de forma simple mediante clases, ocultando los detalles internos y mostrando solo lo esencial.
- **Encapsulamiento**: Consiste en agrupar datos y métodos que operan sobre ellos en una sola unidad (la clase) y restringir el acceso directo a algunos de los componentes.
- **Herencia**: Permite que una clase hereda atributos y métodos de otra, fomentando la reutilización del código.
- **Polimorfismo**: Permite que objetos de distintas clases respondan al mismo método de diferentes formas.

### *¿Qué otros lenguajes soportan la Programación Orientada a Objetos?*

El lenguaje orientado a objetos más antiguo y conocido es Java. Luego está C#, otro lenguaje OOP desarrollado por Microsoft. Algunos de los lenguajes OOP más famosos son PHP, Ruby, TypeScript y Python.

### *¿Cómo podemos crear clases en Python?*

Para crear una clase en Python, usamos la palabra clave class, seguida del nombre de la clase. Por ejemplo:

---

```
class MiClase:
    pass
```

---

Esta estructura nos permite definir atributos y métodos que darán forma al comportamiento y características de los objetos que creemos a partir de ella.

Como puedes ver, el nombre de la clase comienza con letra mayúscula, y esta es una convención de nombres que deberías adoptar. Otra regla importante es que el nombre de la clase debe ser singular.

Si el nombre de la clase necesita estar compuesto por dos palabras, debes utilizar la convención PascalCase, donde cada palabra empieza con mayúscula y no se utilizan guiones bajos. Por ejemplo:

---

```
class UnEstudiante:
    pass
```

---

Esto mejora la legibilidad del código y sigue las buenas prácticas de programación en Python.

### *¿Qué son las Clases y los Objetos en Python?*

En Python, las clases son como un plano o plantilla que usamos para crear objetos. Dentro de una clase, definimos una serie de propiedades (atributos) y métodos (funciones) que van a representar las características y comportamientos comunes de todos los objetos que creemos a partir de esa clase.

Por otro lado, los objetos (también llamados instancias) son ejemplares concretos creados a partir de esa clase. Cada objeto tiene su propia copia de las propiedades y puede usar los métodos definidos en la clase.

Palabras clave importantes:

- **Clase (Class)**: Es la estructura que define qué atributos y métodos va a tener el objeto.
- **Objeto (Object/Instance):** Es una instancia real creada a partir de una clase.
- **Propiedad (Property):** Es una variable que representa una característica del objeto (por ejemplo, nombre, color, edad).
- **Método (Method):** Es una función definida dentro de una clase que define acciones o comportamientos que el objeto puede realizar.

---

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

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")

persona1 = Persona("Ana", 30) # Crear un objeto (instancia) de la clase Persona

persona1.saludar() # Usar un método
```

---

### *Construir clases, atributos y métodos*

En Python, los métodos de clase que comienzan y terminan con doble guion bajo (por ejemplo, __init__) se conocen como métodos especiales, dunder methods (por “double underscore”) o métodos mágicos.

Estos métodos tienen un propósito especial dentro de las clases. Uno de los más comunes es __init__, también llamado constructor, que se ejecuta automáticamente cuando se crea un nuevo objeto de esa clase.

Aunque los métodos usan la misma sintaxis que las funciones (es decir, comienzan con la palabra clave def), la diferencia clave es que:

- Las funciones existen en el ámbito global (fuera de una clase).
- Los métodos están definidos dentro de una clase y son usados por los objetos que se crean a partir de esa clase.

Por eso, en la documentación o libros, muchas veces se habla de funciones dentro de clases como si fueran intercambiables, pero técnicamente, si están dentro de una clase, se consideran métodos.

En resumen:

- def: se usa tanto para definir funciones como métodos.
- __init__: es un método especial llamado automáticamente al crear un objeto.
- __nombre__: cualquier método con doble guion bajo es un método especial (como __str__, __len__, etc.).

Un atributo de objeto de clase (o atributo de clase) es distinto de los demás atributos definidos dentro del método __init__ porque no pertenece a una instancia específica, sino a la clase en sí. Es decir, es compartido por todos los objetos creados a partir de esa clase.

---

```
class Persona:
    is_person = True  # Atributo de clase

    def __init__(self, nombre, apellido, edad):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad

    def saludar(self):
        print(f"Hola, me llamo {self.nombre} {self.apellido} y tengo {self.edad} años.")
```

---

En este caso:

- is_person es un atributo de clase: se define fuera de __init__ y no depende de un objeto particular.
- Se puede acceder a él desde cualquier instancia (por ejemplo, persona1.is_person) o directamente desde la clase (Persona.is_person).
- Los atributos como nombre, apellido y edad son atributos de instancia, porque se asignan en __init__ con self.

---

```
persona1 = Persona("Ana", "López", 28)
print(persona1.is_person)     # True
print(Persona.is_person)      # True
```

---

La palabra clave self se usa en los métodos para referirse a la instancia que invoca el método, y así acceder a sus atributos individuales. En cambio, los atributos estáticos como is_person no necesitan self para ser llamados desde la clase.

Los atributos de clase en Python son variables que se definen directamente dentro de la clase y fuera de cualquier método, por lo que son compartidos por todas las instancias u objetos que se creen a partir de esa clase. Se consideran estáticos, ya que su valor no cambia entre objetos, a menos que se modifique desde la clase misma. En cambio, los atributos de instancia se definen dentro del método __init__ usando self, y cada objeto creado tendrá su propia copia de estos atributos con valores individuales. Esto permite que una clase contenga tanto información general (común a todos los objetos) como información específica de cada instancia.

### *El método __init__()*

El método __init__ en Python es conocido como el constructor de una clase y se ejecuta automáticamente cada vez que se crea una nueva instancia de dicha clase. Su función principal es inicializar los atributos del objeto, es decir, asignar los valores iniciales que definen su estado. Se utiliza la palabra clave self para referirse a la propia instancia que se está creando. Aunque ya se explicó su uso básico, es importante destacar que el __init__ puede aceptar tantos parámetros como se necesiten, lo que permite personalizar cada objeto desde su creación. Además, se pueden incluir dentro del constructor validaciones, condiciones o incluso llamadas a otros métodos si se requiere una configuración más compleja del objeto al momento de su instalación.

El constructor __init__ cumple con la función de crear una nueva instancia u objeto a partir de una clase, como por ejemplo la clase Person. Este nuevo objeto podrá usar los atributos y métodos definidos en la clase. El primer parámetro del constructor es siempre la palabra clave self, que actúa como una referencia al propio objeto que se está creando. Aunque self

parece un parámetro más, es especial porque permite acceder y diferenciar los datos de cada instancia. Cada objeto que creamos con la clase tendrá su propio self, lo que garantiza que cada instancia sea única y maneje su propia información de manera independiente. Al principio puede parecer confuso, pero self es esencial para entender cómo funcionan los objetos en Python.