# PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos". Estos objetos son entidades que encapsulan datos y comportamientos relacionados. La idea principal es organizar el código en torno a estos objetos para que sea más modular, reutilizable y fácil de mantener.

### DIFERENCIAS CON LA PROGRAMACION FUNCIONAL

**1. Unidad básica de programación: Objetos vs. Funciones**

* **Programación Orientada a Objetos (POO):**
  + **Unidad básica:** Objetos. Un objeto es una instancia de una clase que encapsula datos (atributos) y comportamientos (métodos).
  + El enfoque está en crear objetos que representen entidades del mundo real, con estados y comportamientos específicos.
  + Los métodos de los objetos modifican su estado interno y la interacción entre objetos es fundamental.
* **Programación Funcional:**
  + **Unidad básica:** Funciones puras. Una función pura es una que siempre devuelve el mismo resultado dado el mismo input y no tiene efectos secundarios (no modifica el estado global ni el de otras funciones).
  + El enfoque está en la aplicación y composición de funciones para transformar datos. Los datos fluyen a través de funciones que transforman y retornan nuevos valores sin modificar el estado.
  + Las funciones son tratadas como ciudadanos de primera clase, lo que significa que pueden ser pasadas como argumentos, retornadas como resultados, y asignadas a variables.

**2. Estado y mutabilidad: Estado mutable vs. inmutabilidad**

* **Programación Orientada a Objetos (POO):**
  + Los objetos tienen estado mutable; es decir, los atributos de un objeto pueden cambiar durante la ejecución del programa.
  + La POO permite y a menudo depende de la modificación del estado interno de los objetos, lo que puede hacer que el seguimiento de los cambios de estado sea más complejo.
* **Programación Funcional:**
  + Promueve la inmutabilidad, donde los datos no cambian después de ser creados. En lugar de modificar el estado de una estructura de datos, se crea una nueva con los cambios aplicados.
  + El estado global se evita en gran medida, lo que conduce a un código más predecible y fácil de razonar, ya que no hay efectos secundarios inesperados.

**3. Enfoque: Comportamiento vs. Transformación**

* **Programación Orientada a Objetos (POO):**
  + Se centra en describir las entidades (objetos) y cómo interactúan entre sí. Los métodos de los objetos describen comportamientos que cambian el estado interno o interactúan con otros objetos.
  + El diseño del código suele girar en torno a las entidades y sus interacciones.
* **Programación Funcional:**
  + Se centra en describir qué hacer con los datos en lugar de como hacerlo. La atención se dirige hacia las transformaciones de datos mediante la aplicación de funciones.
  + Las funciones pueden ser compuestas entre sí para formar operaciones más complejas.

**4. Gestión del flujo de control: Secuencial vs. Declarativo**

* **Programación Orientada a Objetos (POO):**
  + El flujo de control suele ser más secuencial y explícito, con estructuras como bucles y condicionales.
  + Los métodos de los objetos a menudo controlan directamente el flujo del programa.
* **Programación Funcional:**
  + Prefiere un enfoque más declarativo, donde se describe qué se quiere lograr en lugar de como hacerlo en pasos secuenciales.
  + Funciones como map, filter, y reduce son comunes para manejar listas de manera declarativa.

**5. Abstracción y reutilización del código: Clases vs. Funciones de orden superior**

* **Programación Orientada a Objetos (POO):**
  + La abstracción se logra a través de clases, herencia y polimorfismo. Las clases pueden ser reutilizadas, extendidas y modificadas.
  + Se promueve la encapsulación, donde los detalles internos de una clase están ocultos y solo se expone una interfaz pública.
* **Programación Funcional:**
  + La abstracción se logra mediante funciones de orden superior (funciones que toman otras funciones como argumentos o retornan funciones).
  + La reutilización se promueve mediante la composición de funciones, donde funciones más simples se combinan para crear operaciones más complejas.

### Principios fundamentales de la POO:

1. Clases y objetos:

- Clase: Es una plantilla o blueprint que define un tipo de objeto. Una clase especifica los atributos (datos) y los métodos (comportamientos) que los objetos de ese tipo tendrán.

- Objeto: Es una instancia de una clase. Es un ente concreto creado a partir de una clase que tiene sus propios valores para los atributos definidos en la clase.

2. Encapsulamiento:

- Consiste en ocultar los detalles internos de un objeto y exponer solo lo necesario para su uso. Esto se logra mediante el uso de modificadores de acceso como `public`, `private`, y `protected`, que controlan la visibilidad de los atributos y métodos de un objeto.

3. Herencia:

- Permite que una clase (subclase o clase derivada) herede atributos y métodos de otra clase (superclase o clase base). Esto fomenta la reutilización del código y permite crear jerarquías de clases que representan relaciones "es un".

4. Polimorfismo:

- Es la capacidad de los objetos de diferentes clases de ser tratados como objetos de una clase común. Específicamente, se refiere a la capacidad de diferentes objetos para responder a la misma llamada de método de maneras diferentes. El polimorfismo se manifiesta en dos formas principales:

- Sobrecarga de métodos: Múltiples métodos en la misma clase con el mismo nombre, pero diferentes firmas.

- Sobreescritura de métodos: Una subclase puede proporcionar una implementación específica de un método que ya está definido en su superclase.

5. Abstracción:

- Se refiere a la capacidad de centrarse en lo esencial de un objeto, ocultando los detalles no relevantes. A través de la abstracción, se crean clases abstractas o interfaces que definen un contrato para las clases derivadas.

6. Cohesión

La **cohesión** se refiere al grado en que los elementos de un módulo (por ejemplo, una clase o un método) están relacionados entre sí. En términos sencillos, mide lo bien que las partes de un módulo trabajan juntas para cumplir con una única responsabilidad o propósito.

* **Alta cohesión**: Un módulo con alta cohesión tiene un propósito claro y todas sus partes están estrechamente relacionadas para cumplir ese propósito. Esto hace que el código sea más fácil de entender, mantener y reutilizar.
* **Baja cohesión**: Un módulo con baja cohesión tiene responsabilidades dispersas o inconexas, lo que puede dificultar su mantenimiento y comprensión.

**Ejemplo**:

* Una clase con alta cohesión podría ser una clase Factura, que maneja todo lo relacionado con la facturación, como calcular el total, aplicar descuentos, y generar un recibo.

7.Acoplamiento

El **acoplamiento** se refiere al grado de dependencia entre diferentes módulos (clases, métodos, etc.). Mide cuánto depende un módulo de otro para funcionar correctamente.

* **Bajo acoplamiento**: Es deseable tener bajo acoplamiento, ya que significa que los módulos son independientes y pueden cambiarse sin afectar a otros módulos. Esto aumenta la flexibilidad del diseño y facilita la modificación y el mantenimiento del código.
* **Alto acoplamiento**: Ocurre cuando un módulo depende en gran medida de otro. Esto puede hacer que cualquier cambio en un módulo afecte a otros, dificultando la actualización y el mantenimiento del sistema.

**Ejemplo**:

* Si la clase Factura depende fuertemente de una clase Cliente, sería un ejemplo de alto acoplamiento. Si Cliente cambia, Factura podría romperse o necesitar modificaciones.

Beneficios de la POO:

- Modularidad: Facilita la división del código en partes manejables (clases).

- Reutilización: Las clases y objetos pueden reutilizarse en diferentes programas.

- Mantenibilidad: Los cambios en el código son más fáciles de gestionar.

- Escalabilidad: Facilita la expansión y actualización de software.

Ejemplos en la práctica:

- Un ejemplo clásico es el uso de una clase `Coche`, que puede tener atributos como `color`, `marca`, y `modelo`, y métodos como `acelerar()`, `frenar()`, etc. A partir de esta clase, se pueden crear diferentes objetos como `miCoche` o `tuCoche`, cada uno con sus propias características.

#### EJEMPLOS EN PYTHON

Captura de pantalla de un celular

Descripción generada automáticamente

#### EJEMPLO EN PHP

Captura de pantalla de computadora

Descripción generada automáticamente

#### EJEMPLO EN JAVASCRIPT

Captura de pantalla de computadora

Descripción generada automáticamente