**Abstracción**

La abstracción es el proceso de simplificar un sistema complejo ocultando los detalles innecesarios y mostrando solo las partes esenciales. En POO, las clases son un ejemplo de abstracción porque encapsulan datos y comportamiento.

**Encapsulación**

La encapsulación es el concepto de agrupar datos (atributos) y métodos que operan sobre esos datos en una sola unidad, que es la clase. Además, controla el acceso a esos datos para protegerlos de modificaciones no deseadas.

**Herencia**

La herencia permite que una clase (clase hija o subclase) herede atributos y métodos de otra clase (clase padre o superclase). Esto promueve la reutilización del código y establece una relación "es un" entre las clases.

**Polimorfismo**

El polimorfismo permite que una sola interfaz sea utilizada para diferentes tipos de objetos. En otras palabras, diferentes clases pueden ser tratadas como instancias de una clase padre común.

**Clase y Objeto**

* **Clase:** Es un plano o plantilla para crear objetos. Define atributos y métodos.
* **Objeto:** Es una instancia de una clase. Cuando creas un objeto, estás creando una instancia de la clase con valores específicos.

**Constructor \_\_init\_\_ y su propósito**

El constructor \_\_init\_\_ es un método especial en Python que se ejecuta automáticamente cuando se crea una nueva instancia de la clase. Se utiliza para inicializar los atributos del objeto.

python

Copiar código

class Persona:

def \_\_init\_\_(self, nombre, edad):

self.nombre = nombre

self.edad = edad

**Atributos**

Los atributos son variables que pertenecen a una clase o a sus instancias (objetos). Definen las propiedades de los objetos.

**Diferencias entre métodos y atributos públicos, privados y protegidos**

* **Públicos:** Accesibles desde cualquier parte del programa.
* **Privados:** Solo accesibles dentro de la propia clase (convención: se usa un guion bajo doble, \_\_).
* **Protegidos:** Accesibles dentro de la clase y sus subclases (convención: se usa un guion bajo simple, \_).

**Getters y Setters**

* **Getters:** Métodos que obtienen el valor de un atributo.
* **Setters:** Métodos que establecen o modifican el valor de un atributo.

python

Copiar código

class Persona:

def \_\_init\_\_(self, nombre):

self.\_\_nombre = nombre

def get\_nombre(self):

return self.\_\_nombre

def set\_nombre(self, nombre):

self.\_\_nombre = nombre

**Métodos estáticos, de instancia y de clase**

* **Métodos de instancia:** Operan sobre instancias de la clase.
* **Métodos de clase:** Operan sobre la clase en sí y se declaran con @classmethod.
* **Métodos estáticos:** No dependen ni de la clase ni de la instancia, se declaran con @staticmethod.

python

Copiar código

class MiClase:

def metodo\_instancia(self):

print("Método de instancia")

@classmethod

def metodo\_clase(cls):

print("Método de clase")

@staticmethod

def metodo\_estatico():

print("Método estático")

**Variables de instancia y variables de clase**

* **Variables de instancia:** Pertenecen a una instancia específica de una clase.
* **Variables de clase:** Compartidas por todas las instancias de la clase.

**Sobrecarga de métodos**

La sobrecarga de métodos (no soportada directamente en Python) permite definir múltiples métodos con el mismo nombre pero con diferentes parámetros.

**Sobreescritura de métodos**

La sobrescritura de métodos permite que una subclase proporcione una implementación específica de un método que ya está definido en su superclase.

**Clase abstracta**

Una clase abstracta no se puede instanciar y se utiliza para definir métodos que deben ser implementados por sus subclases. Se usa abc.ABC y @abstractmethod.

**Herencia múltiple**

La herencia múltiple permite que una clase herede de más de una clase base.

**Agregación**

La agregación es una relación en la que una clase contiene otra clase pero las dos pueden existir independientemente.

**Composición**

La composición es una relación más fuerte donde una clase contiene otra clase y la existencia de una depende de la otra.

**Diferencias entre agregación y composición**

* **Agregación:** Relación "tiene un", las partes pueden existir por sí solas.
* **Composición:** Relación "es parte de", las partes no pueden existir sin el todo.

***Agregación***

**Definición:** La agregación es una relación entre objetos en la que un objeto (la clase contenedora) contiene referencias a otros objetos (las clases contenidas), pero los objetos contenidos pueden existir independientemente de la clase contenedora. Es una relación "tiene un".

**Características:**

* La relación entre los objetos es débil.
* La existencia de los objetos contenidos no depende de la existencia del objeto contenedor.
* Los objetos contenidos pueden ser compartidos por varios objetos contenedores.

**Ejemplo:** Imaginemos una clase Departamento que contiene una colección de objetos de la clase Empleado. Aunque Departamento contiene Empleado, los empleados pueden existir sin un departamento específico.

***Composición***

**Definición:** La composición es una relación más fuerte entre objetos en la que un objeto (la clase contenedora) contiene otros objetos (las clases contenidas) y los objetos contenidos no pueden existir independientemente del objeto contenedor. Es una relación "es parte de".

**Características:**

* La relación entre los objetos es fuerte.
* La existencia de los objetos contenidos depende de la existencia del objeto contenedor.
* Los objetos contenidos no pueden ser compartidos por varios objetos contenedores.

**Ejemplo:** Imaginemos una clase Casa que contiene objetos de la clase Habitacion. Las habitaciones no pueden existir sin la casa a la que pertenecen.

***Diferencias entre Agregación y Composición***

* **Relación de Existencia:**
  + **Agregación:** Los objetos contenidos pueden existir independientemente del objeto contenedor. Por ejemplo, un Empleado puede existir sin un Departamento.
  + **Composición:** Los objetos contenidos no pueden existir sin el objeto contenedor. Por ejemplo, una Habitacion no puede existir sin una Casa.
* **Fortaleza de la Relación:**
  + **Agregación:** La relación es débil. Los objetos contenidos pueden ser referenciados por múltiples contenedores.
  + **Composición:** La relación es fuerte. Los objetos contenidos están estrictamente vinculados a un solo contenedor.
* **Independencia:**
  + **Agregación:** Los objetos pueden ser compartidos entre varios contenedores.
  + **Composición:** Los objetos no se comparten y son destruidos cuando el contenedor se destruye.

*En resumen, la agregación y la composición son dos tipos de relaciones en la POO que permiten estructurar sistemas complejos. La agregación permite una relación más flexible y débil, mientras que la composición establece una relación fuerte y dependiente entre los objetos.*

***RECURSIVA***

Las funciones recursivas son funciones que se llaman a sí mismas dentro de su propio código. Estas funciones son útiles para resolver problemas que pueden dividirse en subproblemas más pequeños y similares al original.

**Características de las funciones recursivas**

1. **Caso base:** Es la condición que termina la recursión. Sin un caso base, la función continuaría llamándose a sí misma indefinidamente, lo que resultaría en un desbordamiento de pila (stack overflow).
2. **Caso recursivo:** Es la parte de la función donde se llama a sí misma con argumentos modificados, acercándose al caso base.

**Ejemplo de función recursiva: Factorial**

El factorial de un número nnn (denotado como n!n!n!) se define como el producto de todos los enteros positivos desde 1 hasta nnn. Matemáticamente, se puede definir como: n!=n×(n−1)!n! = n \times (n-1)!n!=n×(n−1)! con el caso base: 0!=10! = 10!=1