# Programación Orientada a Objetos

## ¿Qué es el paradigma de objetos y de dónde salen estos obj?

POO organiza el software en "objetos", que son entidades que encapsulan tanto datos (atributos) como comportamientos (métodos).

Los obj Parten de Clases, Herencia, Encapsulamiento, Polimorfismo y Abstracción para crear sistemas modularizados, reutilizables y fáciles de mantener.

### ¿Cuál es la historia de la programación orientada a objetos (POO)?

El POO surgió en la década de 1960 con Simula, el primer lenguaje en introducir clases y objetos. En los años 80, Smalltalk popularizó la POO, y C++ combinó la eficiencia de C con POO. Posteriormente, lenguajes como Java y Python impulsaron su adopción en el desarrollo de software moderno.

## ¿Qué es una clase en POO?

Una clase es un plano abstracto que funciona como plantilla que define propiedades y comportamientos que son comunes para un grupo de objetos/ entidades.

### Por ejemplo

una clase "Coche" puede tener atributos como color y marca, y métodos como acelerar o frenar.

## ¿Qué es un objeto en POO?

Un objeto es una instancia de una clase. Contiene valores para los atributos definidos en la clase.

### Por ejemplo

un coche rojo, marca Toyota y modelo Corolla sería un objeto de la clase "Coche".

## ¿Qué es el encapsulamiento en POO y para que lo usamos?

Es el concepto de agrupar datos y métodos en una misma clase condicionando el acceso y así proteger los datos internos y exponer solo lo necesario.

Esto asegura la estabilidad y simplifica el mantenimiento del software.

## ¿Qué son los atributos públicos y privados en POO?

Son las diferentes condiciones de acceso que podeos dar.

* *Atributos* *públicos*: Pueden ser accedidos desde fuera de la clase. En Python, no tienen prefijo especial.
* *Atributos privados*: No se deben acceder directamente desde fuera. Se indican con un doble guion bajo (`\_\_`) y se accede a ellos a través de métodos como getters y setters.

## ¿Cuáles son los diferentes tipos de métodos?

* Métodos de instancia: Operan sobre una instancia de la clase/objeto y pueden modificar sus atributos.
* Métodos de clase: Se usan para operar en datos compartidos por todas las instancias de una clase. Utilizan el decorador `@classmethod`.

La existencia de ambos tipos de métodos facilita la organización y estructuración del código, promoviendo una mejor reutilización y un diseño más claro.

## ¿Qué es la herencia en POO?

La herencia permite crear nuevas clases basadas en clases existentes, heredando sus atributos y métodos. Esto promueve la reutilización de código y permite organizar las clases de manera jerárquica.

## ¿Qué es el polimorfismo en POO?

El polimorfismo permite que diferentes subclases implementen el *mismo* *método* de manera diferente. Facilita tratar a los objetos de diversas clases derivadas como si fueran de la misma clase base.

## ¿Qué es la composición en POO?

La composición permite construir clases complejas utilizando objetos de *otras* clases. A diferencia de la herencia, en la composición una clase "tiene" (has-a) objetos de otras clases como parte de su estructura interna, lo que promueve el desacoplamiento y la modularidad.

## ¿Qué es una clase abstracta?

Es una plantilla que define un conjunto de métodos y propiedades que deben ser implementados por sus subclases. Esta clase no está diseñada para crear objetos directamente, sino para ser heredada por otras clases que sí implementen los métodos abstractos.

### ¿Por qué no se pueden instanciar métodos o clases abstractas?

Las clases abstractas no pueden instanciarse directamente porque no tienen implementaciones completas. Su propósito es definir una estructura común para que otras clases la sigan, asegurando que las subclases proporcionen implementaciones concretas de los métodos abstractos.

## ¿Qué son las interfaces en POO?

Es una plantilla que Especifica los métodos que deben implementar otras subclase, pero no proporciona código de implementación. En Python, se puede simular con clases abstractas.

## ¿Qué es la delegación en POO?

La delegación es un patrón de diseño en el que un objeto delega la responsabilidad de una tarea a otro objeto. Permite reutilizar código y promover la responsabilidad única, ya que un objeto puede delegar tareas específicas a colaboradores especializados.

## ¿Qué son las clases y métodos finales?

Las clases y métodos finales no pueden ser heredados ni sobrescritos. Esto asegura que ciertos comportamientos críticos no puedan ser modificados, protegiendo la integridad del sistema y simplificando la estructura del código.

## ¿Qué es un método estático en POO?

Un método estático pertenece a la clase en sí, no a una instancia específica, y se define usando el decorador `@staticmethod`. No puede acceder a atributos ni métodos de instancia y es útil para funciones que no dependen del estado del objeto.

# Complejidad Computacional

## ¿Qué es la complejidad computacional?

La complejidad computacional clasifica los problemas computacionales según los recursos (tiempo y espacio) necesarios para resolverlos. Esto permite medir la eficiencia de los algoritmos que los abordan.

## ¿Qué características debe tener un algoritmo?

- Precisión: Debe expresarse sin ambigüedad.

- Determinismo: Responder de la misma manera ante las mismas condiciones.

- Finitud: La descripción del algoritmo debe ser finita y terminar en un número limitado de pasos.

## ¿Cómo se mide la complejidad algorítmica?

Se mide por los recursos temporales (como tiempo de ejecución) necesarios para resolver un problema, y permite evaluar la eficiencia del algoritmo.

## ¿Qué factores influyen en el costo de la complejidad algorítmica?

- Costo esperado o promedio: Basado en el comportamiento general.

- Mejor costo: El tiempo mínimo necesario.

- Peor costo: El máximo tiempo que puede tomar.

## ¿Qué tipos de costos se consideran en la complejidad temporal?

Se consideran diferentes casos como el mejor, peor y promedio. Por ejemplo, al buscar un elemento en un vector de tamaño \*n\*, el mejor costo es cuando está en la primera posición, el peor costo cuando se recorre todo el vector, y el costo promedio es aproximadamente \*n/2\*.

## ¿Qué son las clases de problemas P, NP, NP-completos y NP-duros?

- Clase P: Son problemas con complejidad polinómica, que son tratables en la práctica.

- Clase NP: Problemas que no se pueden resolver en tiempo polinómico, pero cuya solución puede ser verificada en tiempo polinómico.

- Clase NP-completos: Problemas extremadamente complejos que pertenecen a la clase NP y son los más difíciles de resolver en esa clase.

- Clase NP-duros: Problemas que pueden ser transformados en NP-completos y no pueden resolverse en tiempo polinómico, a menos que P = NP.

## ¿Qué es el orden de magnitud en la complejidad algorítmica?

El orden de magnitud describe el crecimiento del tiempo de ejecución o el uso de recursos en función del tamaño de la entrada. Se clasifica en diferentes tipos como constante, lineal, cuadrático, cúbico, entre otros.

## Ejemplos

### Ejemplo de análisis de complejidad utilizando Python y la biblioteca Big-O

Se puede utilizar la biblioteca `big\_o` para calcular la complejidad temporal de un algoritmo que busca el valor máximo en una lista de números. El resultado puede variar dependiendo del tamaño de la lista y se clasifica en diferentes órdenes de magnitud como tiempo lineal, cuadrático, cúbico, etc.

### Ejemplo con Programación Orientada a Objetos (POO): Clase MaxFinder

Se implementa una clase `MaxFinder` que contiene métodos para generar datos, buscar el valor máximo y calcular la complejidad temporal de la búsqueda. La clase puede mostrar los resultados en diferentes formatos de tiempo.

### Ejemplo de la clase Tiempo con sobrecarga de operadores

Se crea una clase `Tiempo` que puede devolver los minutos de un objeto de tiempo y sobrecargar el operador de resta para calcular la diferencia entre dos tiempos.

### Ejemplo de una clase Progression y sus derivadas

Se implementa una clase `Progression` que representa una progresión numérica. A partir de ella, se derivan clases como `ArithmeticProgression` (progresión aritmética) y `GeometricProgression` (progresión geométrica) que extienden su comportamiento añadiendo lógica de avance específica para cada tipo de progresión.