# PROGRAMACIÓN ORIENTADA A OBJETOS

## Guía de Estudio N°1

**Docente: Víctor Saldivia Vera - Email: victor.saldivia@ulagos.cl**
Ingeniería Civil en Informática - Departamento de Ciencias de la Ingeniería

UNIVERSIDAD DE LOS LAGOS

Martes 02 de Septiembre de 2025

### Enunciado

Esta guía de estudio tiene como objetivo que los estudiantes adquieran y apliquen los conceptos fundamentales de la Programación Orientada a Objetos (POO) en Python. A lo largo de los ejercicios, se trabajará en la creación de clases, objetos, el uso de métodos, atributos, y métodos mágicos.

### Ejercicios Propuestos

1. Se solicita crear una clase **ReservaHostal** que permita a los usuarios crear reservas de habitaciones en un hostal. Cada reserva tendrá atributos como el nombre del cliente, la fecha de entrada, la fecha de salida, y el número de habitación. Implementar los siguientes requerimientos:

   Métodos:

   - Un método para calcular la duración de la estadía del cliente.
   - Un método mágico para mostrar la información de la reserva.
   - Un método para cambiar la fecha de salida.

   Se debe eliminar un objeto **ReservaHostal**, además de imprimir un mensaje indicando que la reserva ha sido cancelada.

2. Implementar una clase **FuncionTrigonometrica**, que permita representar, graficar y evaluar funciones trigonométricas como seno, coseno y tangente. Se solicita realizar lo siguiente:

   Clase **FuncionTrigonometrica**:

   - **Atributos**:
     - Tipo de función (seno, coseno, tangente)
     - Amplitud y periodo de la función.
   - **Métodos**:
     - Crear un método que evalúe la función trigonométrica en un valor x (en radianes)
     - Crear un método que grafique la función en un intervalo de valores.
     - Crear un método mágico que devuelva una representación de texto de la función trigonométrica.
     - Crear un método que devuelva un valor crítico de la función (por ejemplo, los máximos o mínimos para seno y coseno).

Además de implementar las funciones trigonométricas, se debe demostrar gráficamente la periodicidad de la función y cómo se afecta por cambios en la amplitud y el periodo.

3. Diseñar un sistema de gestión de inventario utilizando una clase **Inventario** que gestione múltiples objetos de tipo **Producto**. Este inventario debe ser un diccionario donde las claves serán los códigos de los productos, y los valores serán instancias de cada **Producto**. Además cada producto tendrá un historial de cambios en el stock utilizando una lista.

   Clase **Producto**:

   - **Atributos**:
     - El nombre del producto.
     - El precio por unidad.
     - La cantidad disponible del producto.
     - El código único (Código de Barra) del producto.
     - Una lista que registre el historial de cambios en el stock (incrementos o decrementos).
   - **Métodos de la clase Producto:**
     - Crear método que actualice el stock del producto y añada el cambio al historial de stock.
     - Implementar método que calcule el valor total de los productos disponibles
     - Crear método mágico para mostrar el estado actual del producto en formato texto.

   Clase **Inventario**:

   - **Atributos**:
     - Un diccionario de productos donde la clave es el código del producto y el valor es una instancia de la clase **Producto**.
   - **Métodos de la clase Inventario:**
     - Crear método que agregue un nuevo producto al inventario.
     - Implementar un método que actualice el stock de un producto en el inventario.
     - Crear método que muestre todos los productos del inventario y sus detalles.
     - Implementar un método que calcule el valor total de todos los productos en el inventario.

Se debe considerar en este ejercicio, la clase **Inventario** contiene productos (instancias de la clase **Producto**), pero no hereda de **Producto**.

4. Implementar una clase **Pedido** que represente un pedido en un restaurante. Cada pedido tiene un número de mesa, una lista de platos pedidos y el total del costo del pedido.

   Clase **Pedido**:

   - **Atributos**:
     - Número de mesa, lista de platos, y el total del pedido.
   - **Métodos**:
     - Crear método para añadir platos al pedido (cada plato tiene un nombre y un precio)
     - Crear método para calcular el total del pedido.
     - Implementar un método mágico para contar el número de platos en el pedido.
     - Implementar un método mágico para combinar dos pedidos de la misma mesa (sumar platos y total).
   - **Finalizador**:
     - Al eliminar un pedido, mostrar un mensaje indicando que el pedido ha sido completado.

5. Desarrollar un sistema de gestión de una biblioteca donde se pueda agregar y administrar libros. Cada libro tiene un título, autor, año de publicación, y cantidad disponible del libro. La biblioteca es responsable de gestionar los libros y permitir la búsqueda de libros por título.

   Considerar crear la clase **Biblioteca** que debe manejar múltiples instancias de la clase **Libro** utilizando un diccionario. Los libros se pueden agregar, buscar y actualizar

.

6. Diseñar un sistema que permita gestionar playlists de música utilizando programación orientada a objetos. Para ello, deberás implementar dos clases: **Cancion** y **Playlist**

   Cada **Cancion** debe tener título, artista y duración en segundos, y debe poder mostrarse en formato legible con su duración en minutos y segundos.

   Cada **Playlist** debe tener un nombre y una lista de canciones, y debe permitir:

   - Agregar canciones.
   - Calcular la duración total en minutos y segundos.
   - Mostrar la información de la playlist de forma legible (nombre, número de canciones y duración total).
   - Contar el número de canciones utilizando un método mágico. (len(playlist)).
   - Combinar dos playlists en una nueva utilizando el operador **+**.

**Requisitos mínimos**

1. Clase **Cancion**
    - Atributos: título, artista, duración en segundos.
    - Método **milisegundos()** que convierta la duración a formato **min:seg**.
      Método mágico para mostrar la canción en formato:
        i. *Título — Artista (m:ss)*
2. Clase **Playlist**
    - Atributos: nombre y lista de canciones.
    - Método **agregar()** para añadir canciones.
    - Método **duracion_total()** que calcule la duración en segundos.
    - Método **milisegundos_total()** que muestre la duración total en **min:seg**.
    - Método mágico para contar el número de canciones.
    - Método mágico para combinar dos playlists.
    - Método mágico para mostrar:
        i. *Playlist: Nombre | X canciones | Total: mm:ss*
        ii. *1. Cancion1*
        iii. *2. Cancion2*
        iv. *...*

**Ejemplo de uso**

- Crear al menos dos playlists distintas.
- Agregar canciones con título, artista y duración.
- Imprimir cada playlist.
- Combinar ambas playlists con el operador **+**.
- Mostrar la playlist combinada y la cantidad de canciones resultantes.

**Se solicita:**

1. Desarrollar cada ejercicio en un archivo diferente en Python
2. Subir los archivos a su repositorio personal en GitHub / GitLab