

Ejercicio 1: Gestión de Productos

Crea un programa para gestionar productos en una tienda. Hay tres tipos de productos: **Electrónicos**, **Ropa** y **Comida**. Todos los productos comparten características comunes como el **nombre**, **precio** y **descuento** aplicable. Cada tipo de producto tiene un método para calcular el precio final después del descuento, y reglas adicionales:

- Los **Electrónicos** tienen un descuento máximo del 10%.
- La **Ropa** tiene un descuento máximo del 20%.
- La **Comida** no admite descuentos.

Enunciado:

1. Crea una clase base `Producto` con los atributos y métodos comunes.
2. Define clases derivadas para cada tipo de producto (`Electronico`, `Ropa`, `Comida`).
3. Implementa un método `precio_final()` que calcule el precio con el descuento correspondiente.
4. Crea una lista de productos y muestra el precio final de cada uno.

Ejercicio 2: Sistema de Reservas

Diseña un sistema para gestionar reservas en un hotel. Hay tres tipos de habitaciones: **Individual**, **Doble** y **Suite**. Cada tipo tiene un precio base y características específicas:

- Las **Individuales** tienen un precio base de 50 € por noche.
- Las **Dobles** tienen un precio base de 75 € por noche y permiten un suplemento de desayuno (+10 €).
- Las **Suites** tienen un precio base de 150 € por noche y permiten un descuento del 10% para estancias largas (más de 3 noches).

Enunciado:

1. Crea una clase base `Habitacion` con un método `calcular_precio(noches)`.
2. Implementa clases derivadas para cada tipo de habitación.
3. Crea una lista de habitaciones y calcula el precio total para varias reservas.

Ejercicio 3: Sistema de Vehículos

Crea un sistema para registrar vehículos en una flota de transporte. Hay tres tipos de vehículos: **Coche**, **Camión** y **Moto**. Todos tienen un **modelo**, un **año** y un método para calcular su consumo basado en las siguientes reglas:

- Los **Coches** consumen 5 litros por cada 100 km.
- Los **Camiones** consumen 20 litros por cada 100 km, pero tienen un incremento del 10% por cada tonelada de carga.
- Las **Motos** consumen 3 litros por cada 100 km.

Enunciado:

1. Crea una clase base `Vehiculo` con atributos comunes y un método `calcular_consumo(kilometros)`.
2. Implementa las clases derivadas `Coche`, `Camion` y `Moto` con las reglas específicas.
3. Crea instancias de cada tipo de vehículo y calcula su consumo para un trayecto de 200 km.

Ejercicio 4: Sistema de Gestión de Empleados

Crea un sistema para gestionar empleados de una empresa. Existen tres tipos de empleados: **Asalariados**, **Por Hora**, y **Comisionistas**. Todos los empleados tienen un nombre, un ID único y un método para calcular su salario mensual:

- **Asalariados** tienen un salario fijo mensual.
- **Por Hora** tienen un salario basado en las horas trabajadas (con un máximo de 160 horas al mes) y una tarifa por hora.
- **Comisionistas** reciben un salario base más un porcentaje de las ventas realizadas.

Enunciado:

1. Crea una clase base abstracta `Empleado` con un método abstracto `calcular_salario()`.
2. Implementa las clases derivadas para los diferentes tipos de empleados.
3. Diseña un programa que permita gestionar una lista de empleados y calcular el salario total de la empresa.

```
from abc import ABC, abstractmethod
```

```
class Empleado(ABC):
```

```
.....
```

EJERCICIO 5: Sistema de Gestión de Vehículos de una Empresa de Transporte

La empresa de transporte "TransMovil" necesita un sistema en Python para gestionar los vehículos que operan en su flota. El sistema debe incluir:

1. Clases principales:

- **Vehiculo:** Representa un vehículo genérico.
- **Auto:** Hereda de `Vehiculo` y tiene características específicas para automóviles.
- **Camion:** Hereda de `Vehiculo` y tiene características específicas para camiones.

2. Características del sistema:

- Cada vehículo debe tener:
 - Un **identificador único** (por ejemplo, matrícula o número de registro).
 - El **modelo**.
 - La **capacidad máxima de carga** (en kilogramos, para camiones) o el **número de asientos** (para automóviles).
 - Su **estado actual** (disponible o en servicio).
- Los vehículos deben almacenarse en un **diccionario**, donde la clave sea el identificador único y el valor sea el objeto del vehículo.

3. Requerimientos funcionales:

- Registrar un nuevo vehículo (auto o camión) en el sistema.
- Consultar los datos de un vehículo por su identificador.
- Listar todos los vehículos disponibles en la flota.
- Cambiar el estado de un vehículo (de disponible a en servicio o viceversa).
- Eliminar un vehículo del sistema.

4. Restricciones:

- La capacidad máxima de carga debe ser mayor a 0 para los camiones.

- El número de asientos debe ser mayor a 1 para los automóviles.
-

Tareas a realizar

1. Implementa la clase base `Vehiculo` con los atributos comunes y métodos genéricos, como cambiar el estado o mostrar información.
 2. Implementa las clases `Auto` y `Camion` con atributos y comportamientos específicos.
 3. Crea una clase `GestionFlota` que maneje el diccionario de vehículos y ofrezca métodos para cumplir los requerimientos funcionales.
 4. Escribe un programa principal que simule:
 - El registro de al menos 3 vehículos (2 autos y 1 camión).
 - Consultas de datos y operaciones sobre los vehículos registrados.
-