Clase 1: Fundamentos de Programación Orientada a Objetos en Python 🐍

1. Introducción a la Programación Orientada a Objetos

La programación orientada a objetos (POO) es un paradigma que organiza el software en **objetos**, que son instancias de **clases**. Aquí, las clases actúan como plantillas que definen atributos y comportamientos.

Conceptos Clave:

- **Clase**: Es una plantilla que define un tipo de objeto. Por ejemplo, la clase `Persona` puede tener atributos como `nombre`, `apellido`, y `fecha_de_nacimiento`.
- **Objeto**: Es una instancia de una clase. Por ejemplo, `persona1` puede ser un objeto de la clase `Persona` con `nombre = "Carla"` y `apellido = "Florida"`.

2. Creación de Clases y Objetos

Para crear una clase en Python, se utiliza la palabra reservada `class`, seguida del nombre de la clase (con mayúscula inicial).

Ejemplo:

```
class Persona:
    def __init__(self, nombre, apellido, fecha_nacimiento):
        self.nombre = nombre
        self.apellido = apellido
        self.fecha_nacimiento = fecha_nacimiento

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} {self.apelli
do}.")
```

Creación de Objetos:

Para crear un objeto de la clase `Persona`, se puede hacer de la siguiente manera:

```
persona1 = Persona("Carla", "Florida", "15 de octubre de 192
4")
persona1.saludar()  # Salida: Hola, mi nombre es Carla Florid
a.
```

3. Métodos en Clases

Los **métodos** son funciones definidas dentro de una clase. Pueden acceder a los atributos del objeto a través de `self`.

Ejemplo de un Método:

```
def saludar(self):
    print(f"Hola, mi nombre es {self.nombre}.")
```

4. Ejemplo Práctico: Cuenta Bancaria 💰

Vamos a implementar una clase `CuentaBancaria` que gestione depósitos y retiros.

Implementación:

```
class CuentaBancaria:
    def __init__(self, titular, balance=0):
        self.titular = titular
        self.balance = balance
        self.activa = True  # La cuenta se inicia activa

    def depositar(self, monto):
        if self.activa:
            self.balance += monto
            print(f"Se ha depositado {monto}. Saldo actual:
{self.balance}.")
        else:
            print("La cuenta está inactiva, no se puede depos
itar.")

    def retirar(self, monto):
        if self.activa:
            if monto <= self.balance:
                self.balance -= monto
                print(f"Se ha retirado {monto}. Saldo actual:
{self.balance}.")
            else:
                print("Saldo insuficiente.")
        else:
            print("La cuenta está inactiva, no se puede retir
ar.")

    def desactivar(self):
        self.activa = False
        print("La cuenta ha sido desactivada.")

    def activar(self):
        self.activa = True
        print("La cuenta ha sido activada.")
```

Creación de Objetos:

```
cuenta_ana = CuentaBancaria("Ana", 500)
cuenta_ana.depositar(200)  # Saldo actual: 700
cuenta_ana.retirar(100)     # Saldo actual: 600
cuenta_ana.desactivar()      # La cuenta ha sido desactivada.
cuenta_ana.depositar(50)     # La cuenta está inactiva, no se
puede depositar.
cuenta_ana.activar()         # La cuenta ha sido activada.
cuenta_ana.depositar(50)     # Saldo actual: 650
```

5. Ejercicios Prácticos

A continuación, se presentan algunos ejercicios prácticos basados en los conceptos tratados.

Ejercicio 1: Crear una clase `Vehículo` 🚗

1. Define una clase `Vehículo` que tenga atributos como `marca`, `modelo`, y `año`.
2. Implementa un método `mostrar_info` que imprima la información del vehículo.
3. Crea un objeto de la clase `Vehículo` y muestra su información.

Solución Paso a Paso:

```
class Vehiculo:
    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año

    def mostrar_info(self):
        print(f"Vehículo: {self.marca} {self.modelo}, Año: {s
elf.año}")

# Crear un objeto de la clase Vehículo
vehiculo1 = Vehiculo("Toyota", "Corolla", 2022)
```

```
vehiculo1.mostrar_info()  # Salida: Vehículo: Toyota Corolla,
Año: 2022
```

Ejercicio 2: Sistema de Reserva de Pasajes ✈️

1. Crea una clase `Pasajero` con atributos `nombre` y `edad`.
2. Crea una clase `Vuelo` que tenga una lista de pasajeros y métodos para agregar y listar pasajeros.
3. Implementa un método que permita verificar la disponibilidad de asientos.

Solución Paso a Paso:

```
class Pasajero:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

class Vuelo:
    def __init__(self, destino, capacidad):
        self.destino = destino
        self.capacidad = capacidad
        self.pasajeros = []

    def agregar_pasajero(self, pasajero):
        if len(self.pasajeros) < self.capacidad:
            self.pasajeros.append(pasajero)
            print(f"Pasajero {pasajero.nombre} agregado al vu
elo.")
        else:
            print("No hay más asientos disponibles.")

    def listar_pasajeros(self):
        print(f"Pasajeros en el vuelo a {self.destino}:")
        for p in self.pasajeros:
            print(f"- {p.nombre}")
```

```
# Crear un vuelo
vuelo1 = Vuelo("Madrid", 3)
vuelo1.agregar_pasajero(Pasajero("Carlos", 30))
vuelo1.agregar_pasajero(Pasajero("Ana", 28))
vuelo1.listar_pasajeros()
vuelo1.agregar_pasajero(Pasajero("Luis", 25))
vuelo1.agregar_pasajero(Pasajero("Sofía", 22))  # Sin asiento
s disponibles
```

Resumen y Conclusión

La programación orientada a objetos es un enfoque poderoso que facilita la organización del código en Python. Aprender a utilizar clases y objetos es fundamental para desarrollar aplicaciones más complejas y estructuradas. Estos ejercicios prácticos no solo refuerzan los conceptos aprendidos, sino que también te preparan para aplicar la POO en problemas del mundo real.
