Conceptos Claves de la Programación Orientada a Objetos

1. **Clases y Objetos**
   - **Clase:** Es una plantilla o un plano para crear objetos. Define un tipo de dato abstracto que incluye atributos (propiedades) y métodos (funciones) que los objetos de esa clase tendrán.
   - **Objeto:** Es una instancia de una clase. Representa una entidad concreta con los atributos y métodos definidos por su clase.

   Ejemplo:

```
# Definición de una clase
class Coche:
    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año

    def describir(self):
        return f"{self.marca} {self.modelo} del año {self.año}"

# Creación de un objeto de la clase Coche
mi_coche = Coche("Toyota", "Corolla", 2020)
print(mi_coche.describir())  # Imprime: Toyota Corolla del año 2020
```

2. **Encapsulamiento**
   - **Encapsulamiento:** Es el principio de ocultar los detalles internos del funcionamiento de un objeto y solo exponer lo necesario a través de métodos públicos. Esto protege el estado interno del objeto de cambios no controlados.

   Ejemplo:

```
class CuentaBancaria:
    def __init__(self, saldo):
        self.__saldo = saldo  # Atributo privado

    def depositar(self, cantidad):
        if cantidad > 0:
            self.__saldo += cantidad

    def retirar(self, cantidad):
        if 0 < cantidad <= self.__saldo:
            self.__saldo -= cantidad

    def mostrar_saldo(self):
        return self.__saldo

# Uso de la clase CuentaBancaria
cuenta = CuentaBancaria(1000)
cuenta.depositar(500)
cuenta.retirar(200)
print(cuenta.mostrar_saldo())  # Imprime: 1300
```

3. **Herencia**
    - **Herencia:** Es el mecanismo por el cual una clase puede heredar atributos y métodos de otra clase. La clase que hereda es conocida como "subclase" o "clase derivada", y la clase de la que se hereda es conocida como "superclase" o "clase base". La herencia permite la reutilización del código y la creación de una jerarquía de clases.

Ejemplo:

```
# Clase base
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        return "El animal hace un sonido"

# Clase derivada
class Perro(Animal):
    def hablar(self):
        return "El perro dice: ¡Guau!"

# Uso de la clase Perro
mi_perro = Perro("Rex")
print(mi_perro.nombre)  # Imprime: Rex
print(mi_perro.hablar())  # Imprime: El perro dice: ¡Guau!
```

4. **Polimorfismo**
    - **Polimorfismo:** Es el principio que permite que diferentes clases puedan ser tratadas a través de una interfaz común. En otras palabras, permite que se pueda usar un método en diferentes clases que implementan ese método de maneras distintas.

Ejemplo:

```
class Gato(Animal):
    def hablar(self):
        return "El gato dice: ¡Miau!"

# Uso del polimorfismo
animales = [Perro("Rex"), Gato("Whiskers")]

for animal in animales:
    print(animal.hablar())  # Imprime: El perro dice: ¡Guau!  y luego:
El gato dice: ¡Miau!
```

Resumen de Conceptos

- **Clases y Objetos:** Definen y crean instancias que encapsulan datos y comportamientos.
- **Encapsulamiento:** Protege los datos internos y limita el acceso a ellos.
- **Herencia:** Permite crear nuevas clases basadas en clases existentes, reutilizando y extendiendo su funcionalidad.
- **Polimorfismo:** Permite que el mismo método actúe de manera diferente en diferentes clases, facilitando la flexibilidad del código.

Ejercicio: Gestión de Libros con Herencia

Paso 1: Definición de la Clase Base `Libro`

1. **Crear la clase base `Libro`:**
   - Define la clase `Libro` con atributos básicos y métodos para mostrar información y cambiar la categoría.

```
class Libro:
    def __init__(self, titulo, autor, paginas, categoria):
        self.titulo = titulo
        self.autor = autor
        self.paginas = paginas
        self.categoria = categoria

    def info_libro(self):
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"Páginas: {self.paginas}")
        print(f"Categoría: {self.categoria}")

    def cambiar_categoria(self, nueva_categoria):
        self.categoria = nueva_categoria
        print(f"La categoría se ha actualizado a: {self.categoria}")
```

Paso 2: Definición de la Clase Derivada `LibroDigital`

2. **Crear la clase derivada `LibroDigital`:**
   - Extiende la clase `Libro` para incluir un nuevo atributo `tamano_mb` (tamaño en megabytes) y un método adicional `info_digital` para mostrar información adicional.

```
class LibroDigital(Libro):
    def __init__(self, titulo, autor, paginas, categoria, tamano_mb):
        super().__init__(titulo, autor, paginas, categoria)
        self.tamano_mb = tamano_mb
```

```
    def info_digital(self):
        self.info_libro()  # Llama al método de la clase base
        print(f"Tamaño: {self.tamano_mb} MB")
```

Paso 3: Definición de la Clase Derivada `LibroFísico`

3. **Crear la clase derivada `LibroFísico`:**
   - Extiende la clase `Libro` para incluir un nuevo atributo `peso_kg` (peso en kilogramos) y un método adicional `info_fisico` para mostrar información adicional.

```
class LibroFisico(Libro):
    def __init__(self, titulo, autor, paginas, categoria, peso_kg):
        super().__init__(titulo, autor, paginas, categoria)
        self.peso_kg = peso_kg

    def info_fisico(self):
        self.info_libro()  # Llama al método de la clase base
        print(f"Peso: {self.peso_kg} kg")
```

Paso 4: Creación de Instancias y Uso de las Clases

4. **Crear instancias y demostrar el uso de la herencia:**
   - Crea instancias de `LibroDigital` y `LibroFisico` y muestra cómo acceder a sus atributos y métodos.

```
# Crear instancias de libros digitales y físicos
libro_digital = LibroDigital("Python Avanzado", "Ana Ruiz", 400,
"Programación", 1.2)
libro_fisico = LibroFisico("Aprende Python", "Luis Martínez", 350,
"Programación", 0.8)

# Mostrar información de libros digitales
print("Información del libro digital:")
libro_digital.info_digital()
print()  # Salto de línea

# Mostrar información de libros físicos
print("Información del libro físico:")
libro_fisico.info_fisico()
```

Explicación Adicional:

- **Herencia:** Permite que `LibroDigital` y `LibroFisico` hereden atributos y métodos de `Libro`, pero también pueden agregar sus propios atributos y métodos. Esto promueve la reutilización del código y la extensión de funcionalidades sin duplicar código.
- **Método `super()`:** Se usa para llamar a los métodos de la clase base (`Libro`) desde la clase derivada, permitiendo que la clase derivada aproveche el comportamiento ya definido en la clase base.