Programación orientada a objetos: herencia y polimorfismo

1. Herencia

La herencia es un mecanismo que permite a una clase heredar atributos y métodos de otra clase. Facilita la reutilización de código y la creación de clases más específicas.

Herencia simple en Python

Ejemplo de herencia simple en Python:

```
class Empleado(Persona):
    pass
```

La función `super()`

Ejemplo de uso de `super()` en Python:

```
class Empleado(Persona):
    def __init__(self, nombre, edad, puesto):
        super().__init__(nombre, edad)
        self.puesto = puesto
```

Herencia múltiple en Python

Ejemplo de herencia múltiple en Python:

```
class EmpleadoAdministrativo(Persona, Administrativo):
    pass
```

2. Polimorfismo

El polimorfismo se refiere a la capacidad de una clase hija de sobrescribir o modificar el comportamiento de métodos de la clase padre.

Ejemplo de polimorfismo en Python:

```
class Empleado(Persona):
    def __init__(self, nombre, edad, puesto):
        super().__init__(nombre, edad)
        self.puesto = puesto

    def presentarse(self):
        print(f"Hola, mi nombre es {self.nombre}, tengo {self.edad} años y
trabajo como {self.puesto}.")
```

3. Ejemplo práctico

Vamos a crear una clase `Empleado` que herede de la clase `Persona` y utilice polimorfismo para modificar el método `presentarse()`.

Creación de la clase `Empleado`:

```
class Empleado(Persona):
    def __init__(self, nombre, edad, puesto):
        super().__init__(nombre, edad)
        self.puesto = puesto

    def presentarse(self):
        print(f"Hola, mi nombre es {self.nombre}, tengo {self.edad} años y
trabajo como {self.puesto}.")
```

Creación de objetos `Empleado`:

```
empleado1 = Empleado("Laura", 30, "desarrolladora")
empleado2 = Empleado("Pedro", 40, "gerente")

empleado1.presentarse()
empleado2.presentarse()
```

Salida:

```
Hola, mi nombre es Laura, tengo 30 años y trabajo como desarrolladora.
Hola, mi nombre es Pedro, tengo 40 años y trabajo como gerente.
```

```
# Aplicación de herencia y polimorfismo en un ejemplo más completo

Vamos a utilizar la herencia y el polimorfismo para modelar una jerarquía de
clases que representen diferentes tipos de vehículos.

## 1. Creación de la clase base `Vehiculo`

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

    def encender(self):
        print("El vehículo está encendido.")

    def apagar(self):
        print("El vehículo está apagado.")
```
```

2. Creación de clases derivadas

Clase `Automovil`

```python
class Automovil(Vehiculo):
    def __init__(self, marca, modelo, color, num_puertas):
        super().__init__(marca, modelo, color)
        self.num_puertas = num_puertas

    def abrir_puertas(self, num_puertas_abiertas):
        print(f"Se han abierto {num_puertas_abiertas} puertas.")
```

Clase `Motocicleta`

```python
class Motocicleta(Vehiculo):
    def __init__(self, marca, modelo, color, tipo):
        super().__init__(marca, modelo, color)
        self.tipo = tipo

    def encender(self):
        print("La motocicleta está encendida y lista para conducir.")
```

3. Creación de objetos y demostración de polimorfismo

```python
auto = Automovil("Toyota", "Corolla", "Rojo", 4)
moto = Motocicleta("Honda", "CBR 600", "Azul", "Deportiva")

auto.encender()
auto.abrir_puertas(2)
auto.apagar()

moto.encender()
moto.apagar()
```

Salida:

```
El vehículo está encendido.
Se han abierto 2 puertas.
El vehículo está apagado.
La motocicleta está encendida y lista para conducir.
El vehículo está apagado.
```

En este ejemplo, hemos creado una clase base `Vehiculo` y dos clases derivadas, `Automovil` y `Motocicleta`. La clase `Automovil` hereda todos los métodos y atributos de la clase `Vehiculo` y agrega un atributo adicional (`num_puertas`) y un nuevo método (`abrir_puertas`). La clase `Motocicleta` hereda también todos los métodos y atributos de la clase `Vehiculo`, pero sobrescribe el método `encender()` para mostrar un mensaje específico para las motocicletas.

Este ejemplo demuestra cómo la herencia permite reutilizar y extender código de una clase base en clases derivadas, y cómo el polimorfismo permite modificar el comportamiento de un método heredado en una clase derivada.