

# Programación Orientada a Objetos (POO)

## Herencia múltiple

La herencia múltiple permite que una clase herede de más de una clase base. Esto significa que una subclase puede tener múltiples superclases y, por lo tanto, puede heredar atributos y métodos de todas ellas.

El código muestra un ejemplo de herencia múltiple en Python, donde una clase derivada hereda de dos clases base. Esto permite que una clase tenga funcionalidades combinadas de varias clases base.

### Clase Base 1: Animal

- `__init__(self, nombre)`
  - **Propósito:** Inicializa el atributo privado `__nombre` para almacenar el nombre del animal.
  - **Parámetro:**
    - \* `nombre`: Nombre del animal.
- `@property def nombre(self)`
  - **Propósito:** Proporciona un método para acceder al atributo privado `__nombre`.
  - **Getter:** Devuelve el valor del atributo `__nombre`.
- `def hablar(self)`
  - **Propósito:** Define el comportamiento general de hablar para los animales.
  - **Comportamiento:** Imprime un mensaje genérico que indica que el animal hace un sonido.

### Clase Base 2: Mascota

- `__init__(self, dueño)`
  - **Propósito:** Inicializa el atributo privado `__dueño` para almacenar el nombre del dueño de la mascota.

- **Parámetro:**
  - \* dueño: Nombre del dueño de la mascota.
- `@property def dueño(self)`
  - **Propósito:** Proporciona un método para acceder al atributo privado `__dueño`.
  - **Getter:** Devuelve el valor del atributo `__dueño`.
- `@dueño.setter def dueño(self, nuevo_dueño)`
  - **Propósito:** Permite modificar el atributo privado `__dueño`.
  - **Setter:** Establece el valor de `__dueño` a `nuevo_dueño`.
- `def mostrar_dueño(self)`
  - **Propósito:** Muestra el nombre del dueño de la mascota.
  - **Comportamiento:** Imprime un mensaje indicando quién es el dueño de la mascota.

#### Clase Derivada: Perro

- `__init__(self, nombre, dueño, raza)`
  - **Propósito:** Inicializa la clase `Perro`, que hereda de `Animal` y `Mascota`.
  - **Parámetros:**
    - \* nombre: Nombre del perro.
    - \* dueño: Nombre del dueño del perro.
    - \* raza: Raza del perro.
  - **Comportamiento:**
    - \* `Animal.__init__(self, nombre)`: Llama al constructor de `Animal` para inicializar el atributo `__nombre`.
    - \* `Mascota.__init__(self, dueño)`: Llama al constructor de `Mascota` para inicializar el atributo `__dueño`.
    - \* Inicializa el atributo privado `__raza` específico de `Perro`.
- `@property def raza(self)`
  - **Propósito:** Proporciona un método para acceder al atributo privado `__raza`.
  - **Getter:** Devuelve el valor del atributo `__raza`.
- `def hablar(self)`
  - **Propósito:** Sobreescribe el método `hablar` de `Animal` para proporcionar una implementación específica para los perros.
  - **Comportamiento:**
    - \* `super().hablar()`: Llama al método `hablar` de la clase base `Animal`, que imprime un mensaje genérico.
    - \* Añade la impresión de “Guau”, que es característico de los perros.

```

# Primera clase base
class Animal:
    def __init__(self, nombre):
        self.__nombre = nombre

    @property
    def nombre(self):
        return self.__nombre

    def hablar(self):
        print(f"{self.nombre} hace el siguiente sonido: ")

# Segunda clase base
class Mascota:
    def __init__(self, dueño):
        self.__dueño = dueño

    @property
    def dueño(self):
        return self.__dueño

    @dueño.setter
    def dueño(self, nuevo_dueño):
        self.__dueño = nuevo_dueño

    def mostrar_dueño(self):
        print(f"{self.nombre} es propiedad de {self.dueño}")

# Subclase o clase derivada
class Perro(Animal, Mascota):
    def __init__(self, nombre, dueño, raza):
        Animal.__init__(self, nombre)
        Mascota.__init__(self, dueño)
        self.__raza = raza

    @property
    def raza(self):
        return self.__raza

    def hablar(self):
        super().hablar()
        print("Guau")

```

```
mi_perro = Perro("Helena", "Fidel", "Pastor Alemán")  
  
mi_perro.hablar()  
mi_perro.mostrar_dueño()
```

Helena hace el siguiente sonido:

Guau

Helena es propiedad de Fidel