Informe en LaTeX con Control de Versiones en GitHub

Joaquin Alfonso Carrasco Duran

5 de octubre de 2025

Índice

1. Introducción **2**

2. Herencia **2**

3. Clases Abstractas **2**

4. Polimorfismo **3**

5. Interfaces **3**

6. Method Resolution Order (MRO) **3**

7. Conclusión **4**

1. Introducción

En este informe se abordan los conceptos fundamentales de la Programación Orientada a Objetos (POO), incluyendo **herencia**, **clases abstractas**, **polimorfismo** e **interfaces**, junto con el concepto de **Method Resolution Order (MRO)** en Python. El trabajo se gestiona mediante Git y GitHub siguiendo buenas prácticas de control de versiones y convención de commits.

2. Herencia

La herencia permite que una clase (subclase) reutilice y extienda el comportamiento de otra (superclase).

```
class Animal:
    def hablar(self):
        print("Sonido genérico")

class Perro(Animal):
    def hablar(self):
        print("Guau!")

p = Perro()
p.hablar()  # Imprime: Guau!
```

Ventaja: evita duplicar código y promueve la reutilización.

3. Clases Abstractas

Una clase abstracta define una estructura común, pero deja algunos métodos sin implementar.

```
from abc import ABC, abstractmethod

class Figura(ABC):
    @abstractmethod
    def area(self):
        pass

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio
```

```
    def area(self):
        return 3.14 * self.radio ** 2
```

Uso: asegurar que las subclases implementen ciertos métodos.

4. Polimorfismo

El polimorfismo permite usar un mismo método con distintos comportamientos según el tipo de objeto.

```
def hacer_sonido(animal):
    animal.hablar()

hacer_sonido(Perro())
hacer_sonido(Gato())
```

Idea: un mismo mensaje produce diferentes resultados.

5. Interfaces

En Python, las interfaces pueden simularse usando clases abstractas con solo métodos abstractos.

```
class Enviable(ABC):
    @abstractmethod
    def enviar(self):
        pass
```

Una clase que implemente esta interfaz debe definir el método `enviar()`.

6. Method Resolution Order (MRO)

El MRO define el orden en que Python busca métodos y atributos en las clases al usar herencia múltiple.

Ejemplo:

```
class A: pass
class B(A): pass
```

```
class C(B): pass

print(C.mro())
```

Resultado: muestra la jerarquía lineal de búsqueda de métodos.

7. Conclusión

El uso de POO junto con herramientas modernas como GitHub y LaTeX potencia el desarrollo estructurado, reproducible y colaborativo de proyectos grande y pequeños.