Informe Lab 2: POO en Python

Juan Ignacio Carrera Saavedra

Octubre 2025 – UCT

Índice

1. Introducción **2**

2. Herencia **2**

3. Polimorfismo **2**

4. Clases Abstractas **3**

5. Interfaces **3**

6. Method Resolution Order (MRO) **4**

7. Conclusión **5**

1. Introducción

La Programación Orientada a Objetos (POO) organiza el código en .ºbjetos"que combinan atributos y metodos. En el siguiente informe se cubriran algunos de los conceptos fundamentales que este paradigma de programación ofrece.

2. Herencia

La herencia permite crear clases hijas como se les suele decir, que heredan atributos y métodos de clases padre, especializando su comportamiento.

```
class Vehiculo:
    def transportar(self):
        return "Transportando genericamente"

class Auto(Vehiculo):
    def transportar(self):
        return "Rodando por la carretera"

mi_auto = Auto()
print(mi_auto.transportar())  # Salida: Rodando por la
   carretera
```

Listing 1: Herencia básica en Python

Ventajas:

- Evita duplicación de código
- Crea jerarquías lógicas
- Facilita el mantenimiento

3. Polimorfismo

El polimorfismo permite que diferentes clases respondan distinto ante los mismos metodos.

```
class Pato:
    def sonido(self): return "Cuac!"

class Vaca:
    def sonido(self): return "Muu!"
```

```
animales = [Pato(), Vaca()]
for animal in animales:
    print(animal.sonido())
```

Listing 2: Polimorfismo en acción

Beneficio: Nos ofrece una gran flexibilidad a la hora de tratar a los objetos.

4. Clases Abstractas

Las clases abstractas definen métodos que las subclases deben implementar, se usa como plantilla para otras subclases. Una clase abstracta no se puede instanciar directamente.

```
from abc import ABC, abstractmethod

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

class Circulo(Forma):
    def __init__(self, radio):
        self.radio = radio
    def area(self):
        return 3.1416 * self.radio ** 2

c = Circulo(5)
print(f"Area: {c.area()}")  # Area: 78.54
```

Listing 3: Clase abstracta con implementación

Fórmula del círculo:

$$A = \pi r^2$$

5. Interfaces

Una interfaz es un contrato que define qué métodos debe implementar una clase, pero no cómo debe implementarlos. En Python se pueden usar con Protocol, o simulando una con ABC.

```
from typing import Protocol

class Pagable(Protocol):
```

```
    def procesar_pago(self, monto: float) -> str: ...

class TarjetaCredito:
    def procesar_pago(self, monto: float) -> str:
        return f"Pago de {monto} con tarjeta de credito"

class PayPal:
    def procesar_pago(self, monto: float) -> str:
        return f"Pago de {monto} via PayPal"

# Ambas clases cumplen con el protocolo Pagable
metodos_pago: list[Pagable] = [TarjetaCredito(), PayPal()
   ]

for metodo in metodos_pago:
    print(metodo.procesar_pago(150))
```

Listing 4: Interfaz para servicios de pago

6. Method Resolution Order (MRO)

El MRO es la manera en la que Python busca metodos y atributos en una jerarquia de herencia, sobretodo en herencia multiple.

```
class A:
    def metodo(self): print("A")

class B(A):
    def metodo(self): print("B")

class C(A):
    def metodo(self): print("C")

class D(B, C): pass

d = D()
d.metodo()  # Salida: B (porque B esta primero)
```

Listing 5: MRO resolviendo herencia múltiple

Orden de búsqueda:

$$D \to B \to C \to A \to object$$

7. Conclusión

Los conceptos repasados en este informe crean en conjunto un paradigma de programación que permite crear codigo modular, reutilizable y escalable. La flexibilidad que permiten estos conceptos en su uso conforman una gran herramienta para aquellos que desean construir software robusto.

Las posibilidades que nos abre al estudiar y practicar constantemente estos conceptos nos permite, en conclusión, ser mejores desarrolladores.