

# Polimorfismo en POO

Alumno: Puma Huanca Anthony Rusbel

Docente: Ing. Coyla Idme Leonel

Lenguajes de Programación II – FINESI

Universidad Nacional del Altiplano

Facultad de Ingeniería Estadística e Informática

## Introducción

En la Programación Orientada a Objetos (POO), el **polimorfismo** es la capacidad que tienen los objetos de tomar distintas formas. Permite que un mismo método o función se comporte de manera diferente según el tipo de objeto que lo invoque. En Python, esto se logra mediante **herencia**, **redefinición de métodos** y el **duck typing** (tipado implícito).

## 1 Sobrescritura de Métodos

### Ejemplo 1: Clases Animales

```
1 class Animal:
2     def hacer_sonido(self):
3         print("Sonido gen ríco")
4
5 class Perro(Animal):
6     def hacer_sonido(self):
7         print("Guau")
8
9 class Gato(Animal):
10    def hacer_sonido(self):
11        print("Miau")
12
13 animales = [Perro(), Gato(), Animal()]
14 for animal in animales:
15     animal.hacer_sonido()
```

Listing 1: Polimorfismo mediante herencia

### Ejecución:

```
Guau
Miau
Sonido genérico
```

## Ejemplo 2: Clases Vehiculos

```
1 class Vehiculo:
2     def iniciar(self):
3         print("El vehiculo esta listo para moverse")
4
5 class Coche(Vehiculo):
6     def iniciar(self):
7         print("El coche arranca y acelera")
8
9 class Bicicleta(Vehiculo):
10    def iniciar(self):
11        print("La bicicleta empieza a pedalear")
12
13 class Barco(Vehiculo):
14     def iniciar(self):
15         print("El barco enciende el motor y zarpa")
16
17 vehiculos = [Coche(), Bicicleta(), Barco()]
18 for v in vehiculos:
19     v.iniciar()
```

Listing 2: Método iniciar() polimórfico

### Ejecución:

```
El coche arranca y acelera
La bicicleta empieza a pedalear
El barco enciende el motor y zarpa
```

## 2 Polimorfismo por Duck Typing

El duck typing permite que cualquier objeto con un método del mismo nombre sea usado, sin necesidad de herencia explícita.

### Ejemplo 1: Clase Voladores

```
1 class Pajaro:
2     def volar(self):
3         print("El pajaro vuela")
4
5 class Avion:
6     def volar(self):
7         print("El avion vuela")
8
9 def hacer_volar(obj):
10    obj.volar()
11
12 hacer_volar(Pajaro())
13 hacer_volar(Avion())
```

Listing 3: Duck typing con método volar()

### Ejecución:

```
El pajaro vuela  
El avión vuela
```

### Ejemplo 2: Clase Movimiento4

```
1 class Pajaro:  
2     def mover(self):  
3         print("El pajaro vuela")  
4  
5 class Pez:  
6     def mover(self):  
7         print("El pez nada")  
8  
9 class Persona:  
10    def mover(self):  
11        print("La persona camina")  
12  
13 def desplazar(objeto):  
14     objeto.mover()  
15  
16 objetos = [Pajaro(), Pez(), Persona()]  
17 for objeto in objetos:  
18     desplazar(objeto)
```

Listing 4: Método `mover()` en distintas clases

### Ejecución:

```
El pajaro vuela  
El pez nada  
La persona camina
```

## 3 Ventajas del Polimorfismo

- **Flexibilidad y reutilización:** Permite escribir código que funciona con distintos tipos de objetos sin modificar la lógica.
- **Reducción de acoplamiento:** El código depende de interfaces (métodos), no de tipos específicos.
- **Uso de clases base abstractas:** En Python, aunque no hay interfaces formales como en Java, se pueden usar clases base para forzar la implementación de ciertos métodos.