Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Docente:

Ing. Coyla Idme Leonel

Alumno:

Ticona Miramira Roberto Angel

Polimorfismo en POO

En POO el polimorfismo es la capacidad de un objeto de tomar distintas formas, permitiendo que una misma operación o método se comporte de una manera diferente según el objeto que lo invoque. En python, esto se logra principalmente mediante herencia y redefinición de métodos, gracias a la tipifación dinámica, es muy flexible. En pocas palabras: En python, el polimorfismo permimte usar la misma función, mmétodo u operador para distintos tipos de objetos

» Sobreescritura de métodos

Una clase redefine un método de la superclase par acambiar su comportamiento.

Ejemplos

1.- Crear la clase animal y heredar en las clases perro y gato con el mismo metodo hacer_sonido()

» Código

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

class Perro:
    def hacer_sonido(self):
        print("Guau")

class Gato:
    def hacer_sonido(self):
        print("Miau")

animales = [Perro(), Gato(), Animal()]
for animal in animales:
    animal.hacer_sonido()
```

» Ejecución

```
Guau
Miau
Sonido genérico
```

2.- Crear la clase Vehiculo y heredar a las clases Coche, Bicicleta y Barco, definir el mismo método iniciar()

Código

```python
class Vehiculo:
    def iniciar(self):
        print("El vehiculo esta listo para moverse")

```

```
class Coche(Vehiculo):
    def iniciar(self):
        print("El coche arranca y acelera")

class Bicicleta(Vehiculo):
    def iniciar(self):
        print("La bicicleta empieza a pedalear")

class Barco(Vehiculo):
    def iniciar(self):
        print("El barco enciende el motor y zarpa")

vehiculos = [Coche(), Bicicleta(), Barco()]

for v in vehiculos:
    v.iniciar()
```

Ejecución

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

» Polimorfismo por duck typing

Permite que cualquier objeto que tenga un método con el mismo nombre, aunque no exista relación de herencia.

Ejemplos

1.- Crear las clases Pajaro y Avion, utilizar el mismo método volar().

Código

```
class Pajaro:
    def volar(self):
        print("El pajaro vuela")

class Avion:
    def volar(self):
        print("El avión vuela")

def hacer_volar(obj):
    obj.volar()

hacer_volar(Pajaro())
hacer_volar(Avion())
```

Ejecución

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

2.- Crear las clases Pajaro, Pez y Persona con el mismo método mover()

Código

```
class Pajaro:
    def mover(self):
        print("El pajaro vuela")

class Pez:
    def mover(self):
```

```
        print("El pez nada")

class Persona:
    def mover(self):
        print("La persona camina")

def desplazar(objeto):
    objeto.mover()

objetos = [Pajaro(), Pez(), Persona()]

for objeto in objetos:
    desplazar(objeto)
```

Ejecución

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

» Flexibilidad y reutilización

Permite escribir código que funcione con distintos tipos de objetos sin cambiar la lógica.

» Uso de interfaces o clases bases

En python no tiene interfaces estrictas como java, se pueden usar clases base abstractas (a, b, c) para garantizar que ciertos métodos estén implementados en las subclases.

» Reducción de acoplamiento

El código se vuelve más genérico y fácil de mantener, ya que depende de los métodos de los objetos, no de sus tipos exactos.

Ejemplos

1.- Crear la clase Figura, heredar en las clases Cuadrado, Circulo y Triangulo, utilizando el mismo metodo area().

Código

```
import math
class Figura:
    def area(self):
        pass

class Cuadrado(Figura):
    def __init__(self, lado):
        self.lado = lado

    def area(self):
        return self.lado ** 2

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

    def area(self):
        return math.pi * (self.radio ** 2)

class Triangulo(Figura):
    def __init__(self, base, altura):
        self.base = base
```

```
        self.altura = altura

    def area(self):
        return ((self.base * self.altura) / 2)

figuras = [Cuadrado(4), Circulo(3), Triangulo(5,2)]

for figura in figuras:
    print(f"Área : {figura.area()}")
```

Ejecución

```
Área : 16
Área : 28.274333882308138
Área : 5.0
```

2.- Crear la clase principal MetodoPago, crear las clases heredadas TarjetaCredito, Paypal y Efectivo para registrar pagos.

Código

```
class MetodoPago:
    def pago(self):
        pass

class TarjetaCredito(MetodoPago):
    def pago(self):
        print("Se realizo el pago con tarjeta de credito")

class Paypal(MetodoPago):
    def pago(self):
        print("Se realizo el pago con Paypal")

class Efectivo(MetodoPago):
    def pago(self):
        print("Se realizo el pago con efectivo")

pagos = [TarjetaCredito(), Paypal(), Efectivo()]

for p in pagos:
    p.pago()
```

Ejecución

```
Se realizo el pago con tarjeta de credito
Se realizo el pago con Paypal
Se realizo el pago con efectivo
```