Instituto Politécnico Nacional

Escuela Superior De Ingeniería Mecánica y Eléctrica

Programación Orientada a Objetos: Conceptos Clave

De la Rosa Vázquez Josué

Índice

Introducción

La **Programación Orientada a Objetos (POO)** es un paradigma que organiza el software en términos de "objetos", los cuales representan entidades del mundo real o conceptos abstractos. En este documento exploraremos conceptos clave de la POO: polimorfismo, sobrecarga de operadores y herencia, junto con sus tipos y alcances.

Desarrollo de los Temas

Polimorfismo

Definición: El polimorfismo permite que distintas clases respondan de manera única a un método común, haciendo que el código sea más flexible y extensible.

Ventajas: Facilita la extensibilidad y reduce la duplicación de código.

Ejemplo en Python:

In [1]:
```python
class Animal:
    def hacer_sonido(self):
        pass

class Perro(Animal):
    def hacer_sonido(self):
        return "¡Guau!"

class Gato(Animal):
    def hacer_sonido(self):
        return "¡Miau!"

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

```
¡Guau!
¡Miau!
```

Polimorfismo Paramétrico

Definición: Este tipo de polimorfismo permite que funciones o métodos operen con diferentes tipos de datos mediante el uso de generics.

Ejemplo en Python:

In [2]:
```python
from typing import TypeVar, List

T = TypeVar('T')  # T puede ser cualquier tipo

def obtener_elemento_medio(lista: List[T]) -> T:
    return lista[len(lista) // 2]

numeros = [10, 20, 30, 40, 50]
palabras = ["uno", "dos", "tres", "cuatro"]

print(obtener_elemento_medio(numeros))
print(obtener_elemento_medio(palabras))
```

```
30
tres
```

Sobrecarga de Operadores

Definición: Permite redefinir operadores para su uso en objetos personalizados.

Ejemplo en Python:

In [3]:
```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, otro):
        return Vector(self.x + otro.x, self.y + otro.y)

    def __str__(self):
        return f"({self.x}, {self.y})"

v1 = Vector(2, 4)
v2 = Vector(3, 1)
resultado = v1 + v2
print(resultado)
```

```
(5, 5)
```

Herencia

Definición: La herencia permite crear una clase derivada que reutiliza y extiende la funcionalidad de una clase base.

Ejemplo en Python:

In [4]:
```python
class Vehiculo:
    def encender_motor(self):
        return "El motor está encendido."

class Coche(Vehiculo):
    def conducir(self):
        return "El coche está en movimiento."

mi_coche = Coche()
print(mi_coche.encender_motor())
print(mi_coche.conducir())
```

```
El motor está encendido.
El coche está en movimiento.
```

Herencia Simple

Definición: Una clase hereda de una única clase base.

Ejemplo en Python:

In [5]:
```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

class Estudiante(Persona):
    def __init__(self, nombre, id_estudiante):
        super().__init__(nombre)
        self.id_estudiante = id_estudiante

estudiante = Estudiante("Juan", 123)
print(estudiante.nombre)
print(estudiante.id_estudiante)
```

```
Juan
123
```

Herencia Múltiple

Definición: Una clase hereda de múltiples clases base, lo cual puede provocar ambigüedad si no se gestiona correctamente.

Ejemplo en Python:

In [6]:
```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

class Empleado:
    def __init__(self, salario):
        self.salario = salario

class Doctor(Persona, Empleado):
    def __init__(self, nombre, salario, especialidad):
        Persona.__init__(self, nombre)
        Empleado.__init__(self, salario)
        self.especialidad = especialidad

doctor = Doctor("Ana", 50000, "Cardiología")
print(doctor.nombre)
print(doctor.salario)
print(doctor.especialidad)
```

```
Ana
50000
Cardiología
```

Herencia de Elementos Públicos, Privados y Protegidos

Definicion: En Python, los atributos protegidos se nombran con un guion bajo (_). Los atributos privados se nombran con dos guiones bajos (__), y los públicos se definen sin guiones.

Ejemplo en Python:

In [26]:
```python
class ClaseBase:
    def __init__(self):
        self.publico = "Soy público"
        self._protegido = "Soy protegido"
        self.__privado = "Soy privado"

    def obtener_privado(self):
        return self.__privado

class ClaseDerivada(ClaseBase):
    def mostrar(self):
        return f"{self.publico}, {self._protegido}, {self.obtener_privado()}"

objeto = ClaseDerivada()
print(objeto.mostrar())
```

```
Soy público, Soy protegido, Soy privado
```

Ejemplos de Cada Tema

A continuación, se incluyen cinco ejemplos para cada tema abordado en el desarrollo.

Polimorfismo

1. Ejemplo Animales

In [7]:
```python
class Animal:
    def hacer_sonido(self):
        return "Sonido genérico"

class Perro(Animal):
    def hacer_sonido(self):
        return "Guau"

class Gato(Animal):
    def hacer_sonido(self):
        return "Miau"

for animal in [Perro(), Gato()]:
    print(animal.hacer_sonido())
```

```
Guau
Miau
```

2. Ejemplo Vehiculos

In [8]:
```python
class Vehiculo:
    def mover(self):
        return "El vehículo se mueve"
```

```
class Coche(Vehiculo):
    def mover(self):
        return "El coche avanza"

class Avion(Vehiculo):
    def mover(self):
        return "El avión despega"

for vehiculo in [Coche(), Avion()]:
    print(vehiculo.mover())
```

```
El coche avanza
El avión despega
```

3. Ejemplo Herramientas

In [9]:
```
class Herramienta:
    def usar(self):
        return "Usando herramienta"

class Martillo(Herramienta):
    def usar(self):
        return "Martillazo"

class Sierra(Herramienta):
    def usar(self):
        return "Corte"

for herramienta in [Martillo(), Sierra()]:
    print(herramienta.usar())
```

```
Martillazo
Corte
```

4. Ejemplo Electrodomesticos

In [10]:
```
class Electrodomestico:
    def encender(self):
        return "Encendiendo..."

class Lavadora(Electrodomestico):
    def encender(self):
        return "Lavadora encendida"

class Microondas(Electrodomestico):
    def encender(self):
        return "Microondas calentando"

for aparato in [Lavadora(), Microondas()]:
    print(aparato.encender())
```

```
Lavadora encendida
Microondas calentando
```

5. Ejemplo Fruta

In [11]:
```
class Fruta:
    def descripcion(self):
        return "Fruta genérica"

class Manzana(Fruta):
    def descripcion(self):
        return "Soy una manzana"

class Naranja(Fruta):
    def descripcion(self):
        return "Soy una naranja"

for fruta in [Manzana(), Naranja()]:
    print(fruta.descripcion())
```

```
Soy una manzana
Soy una naranja
```

Polimorfismo Paramétrico

1. Función que devuelve el valor medio en listas de cualquier tipo

In [12]:
```
from typing import List, TypeVar

T = TypeVar('T')

def obtener_medio(lista: List[T]) -> T:
    return lista[len(lista) // 2]

print(obtener_medio([1, 2, 3]))
print(obtener_medio(["uno", "dos", "tres"]))
```

```
2
dos
```

2. Función de búsqueda de máximo en listas de cualquier tipo

In [13]:
```
def encontrar_maximo(lista: List[T]) -> T:
    return max(lista)

print(encontrar_maximo([3, 5, 2]))
print(encontrar_maximo(["a", "b", "c"]))
```

```
5
c
```

3. Función de intercambio de valores en una tupla

def intercambiar(a: T, b: T) -> tuple: return b, a

```
print(intercambiar(3, 5))
```

4. Función que devuelve el primer elemento de una lista de cualquier tipo

In [14]:
```
def primer_elemento(lista: List[T]) -> T:
    return lista[0]

print(primer_elemento([10, 20, 30]))
```

```
10
```

5. Función que compara dos valores de cualquier tipo

In [15]:
```
def son_iguales(a: T, b: T) -> bool:
    return a == b

print(son_iguales(5, 5))
print(son_iguales("hola", "adios"))
```

```
True
False
```

Sobrecarga de Operadores

1. Suma de vectores

In [16]:
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, otro):
        return Vector(self.x + otro.x, self.y + otro.y)

v1 = Vector(2, 3)
v2 = Vector(1, 1)
resultado = v1 + v2
print(resultado.x, resultado.y)
```

```
3 4
```

2. Sobrecarga de operador de igualdad (==) en una clase Persona

In [17]:
```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def __eq__(self, otro):
        return self.nombre == otro.nombre

p1 = Persona("Juan")
```

```
p2 = Persona("Juan")
print(p1 == p2)
```

```
True
```

3. Multiplicación de números en una clase personalizada

In [18]:
```
class Numero:
    def __init__(self, valor):
        self.valor = valor

    def __mul__(self, otro):
        return Numero(self.valor * otro.valor)

n1 = Numero(3)
n2 = Numero(4)
resultado = n1 * n2
print(resultado.valor)
```

```
12
```

4. Sobrecarga de **str** para representar una clase Producto

In [19]:
```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def __str__(self):
        return f"{self.nombre} cuesta {self.precio} USD"

producto = Producto("Manzana", 2)
print(producto)
```

```
Manzana cuesta 2 USD
```

5. Comparación (<) entre objetos Persona por edad

In [20]:
```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def __lt__(self, otro):
        return self.edad < otro.edad

p1 = Persona("Juan", 20)
p2 = Persona("Ana", 30)
print(p1 < p2)
```

```
True
```

Herencia

1. Clase base Animal con clase derivada Perro

In [21]:
```
class Animal:
    def sonido(self):
        return "Sonido genérico"

class Perro(Animal):
    def sonido(self):
        return "Guau"

perro = Perro()
print(perro.sonido())
```

```
Guau
```

2. Clase base Vehiculo con clase derivada Coche

In [22]:
```
class Vehiculo:
    def encender(self):
        return "Vehículo encendido"

class Coche(Vehiculo):
    def encender(self):
        return "Coche encendido"

coche = Coche()
print(coche.encender())
```

```
Coche encendido
```

3. Clase base Persona con clase derivada Estudiante

In [23]:
```
class Persona:
    def hablar(self):
        return "Hola"

class Estudiante(Persona):
    def estudiar(self):
        return "Estudiando"

estudiante = Estudiante()
print(estudiante.hablar())
print(estudiante.estudiar())
```

```
Hola
Estudiando
```

4. Clase base Electrodomestico con clase derivada Lavadora

In [24]:
```
class Electrodomestico:
    def encender(self):
        return "Electrodoméstico encendido"
```

```
class Lavadora(Electrodomestico):
    def lavar(self):
        return "Lavando ropa"

lavadora = Lavadora()
print(lavadora.encender())
print(lavadora.lavar())
```

```
Electrodoméstico encendido
Lavando ropa
```

5. Clase base Figura con clase derivada Circulo

In [25]:
```
class Figura:
    def area(self):
        return "Área no definida"

class Circulo(Figura):
    def area(self):
        return "Área del círculo calculada"

circulo = Circulo()
print(circulo.area())
```

```
Área del círculo calculada
```

Mapa Mental

Mi Mapa Mental

Conclusión

La POO permite una organización modular y flexible en el desarrollo de software. Conceptos como el polimorfismo y la herencia simplifican la reutilización de código, aunque también presentan retos como la gestión de la ambigüedad en herencia múltiple y sobrecarga de operadores.

Bibliografía

1. "Lutz, M. (2013). Learning Python. O'Reilly Media."
2. Barry, P. (2010). Head First Python: A Brain-Friendly Guide. O'Reilly Media
3. Beazley, D. M., & Jones, B. K. (2013). Python Cookbook. O'Reilly Media.