Actividad 5

Polimorfismo

Definición del polimorfismo y su papel en la POO

El polimor smo es un concepto clave en la Programación Orientada a Objetos que se re ere a la capacidad de objetos de diferentes clases de responder a un mismo mensaje o invocar un mismo método de manera coherente. El polimor smo permite que objetos de distintas clases puedan ser tratados de manera uniforme si cumplen con una interfaz común, proporcionando exibilidad y extensibilidad en el diseño de software.

Definición de Polimorfismo:

El polimor smo signi ca "muchas formas". En el contexto de la POO, se re ere a la capacidad de objetos de diferentes clases de responder de manera única a una misma interfaz o mensaje. El polimor smo permite tratar objetos de distintas clases de manera uniforme si cumplen con una interfaz común o heredan de una misma clase base.

Definición del polimorfismo y su papel en la POO

Características Clave:

Polimorfismo de Sobrecarga (Compile-Time): Se re ere a la capacidad de una clase de proporcionar diferentes implementaciones para un mismo método, basándose en el tipo o número de parámetros. También se conoce como sobrecarga de métodos.

Polimorfismo de Sobreescritura (Run-Time): Se re ere a la capacidad de una clase derivada de proporcionar su propia implementación de un método que ya está de nido en su clase base. También se conoce como sobreescritura de métodos.

+ info

```python
class Animal:
    def sonido(self):
        pass

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

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

# Función polimórfica
def hacer_sonar(animal):
    return animal.sonido()

# Creación de instancias
perro = Perro()
gato = Gato()

# Llamada a la función polimórfica
print(hacer_sonar(perro))  # Salida: Guau
print(hacer_sonar(gato))   # Salida: Miau
```

Ejemplo Práctico:

En este ejemplo, las clases Perro y Gato heredan de la clase base Animal. La función hacer_sonar toma un objeto de tipo Animal como parámetro y llama al método sonido, permitiendo que objetos de diferentes clases respondan al mismo mensaje de manera coherente.

Definición del polimorfismo y su papel en la POO

Importancia en la POO:

Flexibilidad y Extensibilidad: El polimor smo permite diseñar sistemas más exibles y extensibles al tratar objetos de manera uniforme y permitir la adición fácil de nuevas clases.

Reutilización de Código: Facilita la reutilización de código al permitir que una interfaz común sea implementada por diversas clases.

Interoperabilidad: Favorece la interoperabilidad entre clases, ya que objetos de distintas clases pueden interactuar de manera armoniosa si cumplen con una interfaz común. El polimor smo es esencial en la POO ya que promueve una mayor modularidad y facilita la creación de sistemas más adaptables a cambios futuros. Permite tratar objetos de manera más genérica, mejorando la capacidad de los programas para manejar la complejidad y evolucionar con el tiempo.

Ejemplos de cómo el polimorfismo mejora la flexibilidad del código

El polimorfismo mejora la flexibilidad del código al permitir que objetos de diferentes clases sean tratados de manera uniforme. ejemplos que ilustran cómo el polimorfismo mejora la flexibilidad y extensibilidad del código:

Interfaz Común para Operaciones:

Supongamos que tienes una interfaz común para operaciones matemáticas y dos clases que implementan esa interfaz. El polimor smo permite tratar objetos de ambas clases de manera uniforme al llamar a los métodos de la interfaz común.

```python
# Interfaz común para operaciones matemáticas
class OperacionMatematica:
    def ejecutar(self, a, b):
        pass

# Clases que implementan la interfaz
class Suma(OperacionMatematica):
    def ejecutar(self, a, b):
        return a + b

class Resta(OperacionMatematica):
    def ejecutar(self, a, b):
        return a - b

# Uso del polimorfismo
operacion1 = Suma()
operacion2 = Resta()

resultado1 = operacion1.ejecutar(5, 3)  # Suma
resultado2 = operacion2.ejecutar(5, 3)  # Resta

print(resultado1)  # Salida: 8
print(resultado2)  # Salida: 2
```

```python
# Interfaz común para formas geométricas
class FormaGeometrica:
    def area(self):
        pass

# Clases que implementan la interfaz
class Circulo(FormaGeometrica):
    def __init__(self, radio):
        self.radio = radio

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

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

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

# Uso del polimorfismo
forma1 = Circulo(5)
forma2 = Cuadrado(4)

area1 = forma1.area()  # Área del círculo
area2 = forma2.area()  # Área del cuadrado

print(area1)  # Salida: 78.5
print(area2)  # Salida: 16
```

Tratamiento Genérico de Objetos:

Supongamos que tienes una aplicación de grá cos con varias formas geométricas. El polimor smo permite tratar todas las formas como objetos de la misma interfaz, lo que facilita el tratamiento genérico y la manipulación uniforme.

```python
# Clase base para personajes
class Personaje:
    def atacar(self):
        pass

# Clases que heredan de Personaje
class Guerrero(Personaje):
    def atacar(self):
        return "Espada"

class Mago(Personaje):
    def atacar(self):
        return "Hechizo"

# Uso del polimorfismo
jugador1 = Guerrero()
jugador2 = Mago()

ataque1 = jugador1.atacar()  # Ataque del guerrero
ataque2 = jugador2.atacar()  # Ataque del mago

print(ataque1)  # Salida: Espada
print(ataque2)  # Salida: Hechizo
```

Extensión Fácil de Funcionalidad:

Supongamos que estás construyendo un juego con diferentes personajes. El polimor smo facilita la extensión del juego al agregar nuevos personajes sin afectar el código existente.

+ info

Estos ejemplos muestran cómo el polimor smo permite tratar objetos de diferentes clases de manera uniforme, lo que facilita la creación de código exible, extensible y fácil de mantener. El código puede adaptarse más fácilmente a cambios y adiciones futuras sin necesidad de modi car porciones existentes.

Ejercicios relacionados con Programación Orientada a Objetos (POO) en Python

Ejercicio Creación de Clases y Objeto

Crea una clase Coche con atributos como modelo, color y anio.
Crea dos objetos de la clase Coche con diferentes valores para los atributos.
Imprime los detalles de ambos coches.

Ejercicio Métodos en Clases

Modifica la clase Coche para incluir un método arrancar que imprima un mensaje indicando que el coche está arrancado.
Llama al método arrancar para uno de los coches creados en el Ejercicio 1.

Ejercicios relacionados con Programación Orientada a Objetos (POO) en Python

Ejercicio Herencia

Crea una clase Deportivo que herede de la clase Coche.
Añade un atributo adicional a la clase Deportivo como velocidad_maxima.
Crea un objeto de la clase Deportivo y llama al método arrancar.

Ejercicio Polimorfismo

Define un método llamado desplazarse en ambas clases (Coche y Deportivo) que imprima un mensaje genérico como "Se desplaza por la carretera".
Crea una lista que contenga objetos de ambas clases y, mediante un bucle, llama al método desplazarse para cada objeto.

Ejercicios relacionados con Programación Orientada a Objetos (POO) en Python

Ejercicio Encapsulamiento.

Modifica la clase Coche para que el atributo anio sea privado.
Agrega métodos de acceso (getter y setter) para el atributo anio.

Estos ejercicios proporcionan práctica en la creación y manipulación de clases, el uso de herencia, la implementación de métodos y el manejo de encapsulamiento en Python.

TIC

TALENTO TECH

Universidad Tecnológica de Pereira

faceIT