

Programación Orientada a Objetos (POO)

Polimorfismo con argumentos variables

El polimorfismo con argumentos variables permite que métodos en diferentes clases acepten un número variable de argumentos, ya sea posicionales o de palabra clave

Uso de **args*** y *kwargs*:

- ****args:** Permite que una función o método acepte un número variable de argumentos posicionales.
- ****kwargs:** Permite que una función o método acepte un número variable de argumentos de palabra clave.

El siguiente código muestra un ejemplo de polimorfismo utilizando argumentos variables (***args** y ****kwargs**). En este caso, se usa el polimorfismo para manejar diferentes tipos de operaciones con una interfaz común, permitiendo que diferentes clases derivadas procesen los datos de manera específica.

Clase Base: `Operacion`

- `def procesar(self, *args, **kwargs)`
 - **Propósito:** Define una interfaz común para todas las operaciones, especificando que el método `procesar()` debe ser implementado por las subclasses.
 - **Implementación:** Lanza una excepción `NotImplementedError` para indicar que las subclasses deben proporcionar su propia implementación del método `procesar()`.

Clase Derivada 1: `Suma`

- `def procesar(self, *args, **kwargs)`
 - **Propósito:** Implementa el método `procesar()` para realizar una suma de los argumentos.
 - **Implementación:**
 - * Utiliza `sum(args)` para calcular la suma de todos los argumentos numéricos.

- * Usa `kwargs.get('mensaje', 'Resultado de la suma:')` para obtener un mensaje opcional que se muestra junto con el resultado.
- * Devuelve un string con el mensaje y el total de la suma.

Clase Derivada 2: Multiplicar

- `def procesar(self, *args, **kwargs)`
 - **Propósito:** Implementa el método `procesar()` para realizar una multiplicación de los argumentos.
 - **Implementación:**
 - * Utiliza `reduce(lambda x, y: x * y, args)` para calcular el producto de todos los argumentos numéricos.
 - * Usa `kwargs.get('mensaje', 'Resultado de la multiplicación:')` para obtener un mensaje opcional que se muestra junto con el resultado.
 - * Devuelve un string con el mensaje y el total de la multiplicación.

Clase Derivada 3: Concatenar

- `def procesar(self, *args, **kwargs)`
 - **Propósito:** Implementa el método `procesar()` para concatenar cadenas.
 - **Implementación:**
 - * Utiliza `''.join(args)` para concatenar todos los argumentos de tipo cadena.
 - * Usa `kwargs.get('separador', '')` para obtener un separador opcional que se coloca entre las cadenas.
 - * Usa `kwargs.get('sufijo', '')` para obtener un sufijo opcional que se añade al final de la cadena concatenada.
 - * Devuelve la cadena concatenada con el separador y sufijo opcionales.

Función que Utiliza Polimorfismo con Argumentos Variables

- `def realizar_operacion(operacion, *args, **kwargs)`
 - **Propósito:** Acepta un objeto de tipo `Operacion` (o una de sus subclases) y llama al método `procesar()` de ese objeto con argumentos variables.
 - **Implementación:**
 - * Llama al método `procesar()` del objeto `operacion`, pasando todos los argumentos y palabras clave.
 - * Imprime el resultado del método `procesar()`.

Detalles adicionales:

- En el código, Suma, Multiplicar y Concatenar son clases derivadas que implementan el método procesar() de manera específica, pero todas utilizan argumentos variables (*args y **kwargs). La función realizar_operacion() demuestra el uso del polimorfismo, permitiendo que diferentes tipos de operaciones se realicen con la misma interfaz de llamada, sin necesidad de conocer los detalles específicos de cada operación.
- El uso de *args permite que las clases derivadas acepten un número variable de argumentos, y **kwargs permite opciones adicionales configurables. Esto proporciona una interfaz flexible y extensible para procesar datos de manera consistente.

```
from functools import reduce

# Clase base
class Operacion:
    def procesar(self, *args, **kwargs):
        raise NotImplementedError("Subclases deben implementar este método")

# Clase derivada 1: Suma
class Suma(Operacion):
    def procesar(self, *args, **kwargs):
        total = sum(args)
        mensaje = kwargs.get('mensaje', 'Resultado de la suma:')
        return f"{mensaje} {total}"

# Clase derivada 2: Concatenar
class Multiplicar(Operacion):
    def procesar(self, *args, **kwargs):
        total = reduce(lambda x, y: x * y, args)
        mensaje = kwargs.get('mensaje', 'Resultado de la multiplicación:')
        return f"{mensaje} {total}"

# Clase derivada 2: Concatenar
class Concatenar(Operacion):
    def procesar(self, *args, **kwargs):
        concatenado = ''.join(args)
        separador = kwargs.get('separador', '')
        return separador.join(args) + kwargs.get('sufijo', '')

# Función que utiliza el polimorfismo con argumentos variables
def realizar_operacion(operacion, *args, **kwargs):
    resultado = operacion.procesar(*args, **kwargs)
    print(f"Resultado: {resultado}")

# Crear instancias de las clases derivadas
```

```
suma = Suma()
mult = Multiplicar()
concatenar = Concatenar()

# Llamar a la función con diferentes tipos de operaciones y argumentos variables
realizar_operacion(suma, 1, 2, 3, 4,
                   mensaje="Suma de los números:")
realizar_operacion(mult, 1, 2, 3, 4,
                   mensaje="Multiplicación de los números:")
realizar_operacion(concatenar, 'Hola', 'Mundo', 'Python',
                   separador=' ', sufijo='!')
```

Resultado: Suma de los números: 10

Resultado: Multiplicación de los números: 24

Resultado: Hola Mundo Python!