### **Ejercicio 1: Creación básica de un decorador**

Escribe un decorador básico llamado mi\_decorador que imprima "Llamando a la función" antes de ejecutar la función decorada y "Función ejecutada" después de que la función decorada haya terminado.

def mi\_decorador(func):

# Completa aquí el decorador

pass

@mi\_decorador

def saludar():

print("¡Hola!")

# Llamada a la función decorada

saludar()

### **Ejercicio 2: Decorador con argumentos**

Modifica el decorador mi\_decorador para que sea capaz de decorar una función con cualquier número de argumentos. (Empieza con 2 argumentos, usando la función sumar(a, b) como ejemplo de función decorada.)

def mi\_decorador(func):

# Completa el decorador para aceptar funciones con argumentos

pass

@mi\_decorador

def sumar(a, b):

return a + b

# Llamada a la función decorada

resultado = sumar(3, 4)

print(f"Resultado de la suma: {resultado}")

### **Ejercicio 3: Decorador con retorno modificado**

Crea un decorador llamado duplicar\_resultado que modifique la función decorada para que su resultado se duplique. Usa la función multiplicar(a, b) para probar el decorador.

def duplicar\_resultado(func):

# Completa el decorador para duplicar el resultado de la función

pass

@duplicar\_resultado

def multiplicar(a, b):

return a \* b

# Llamada a la función decorada

resultado = multiplicar(5, 3)

print(f"Resultado duplicado: {resultado}")

### **Ejercicio 4: Decorador anidado**

Crea dos decoradores: decorador1 y decorador2. El primero imprimirá "Primero" antes de ejecutar la función decorada, y el segundo imprimirá "Segundo". Luego, aplica ambos decoradores a la función mi\_funcion.

def decorador1(func):

# Completa decorador1

pass

def decorador2(func):

# Completa decorador2

pass

@decorador1

@decorador2

def mi\_funcion():

print("¡Hola desde la función!")

# Llamada a la función decorada

mi\_funcion()

### **Ejercicio 5: Aplicación práctica (autenticación)**

Imagina que estás desarrollando una aplicación y necesitas controlar el acceso a ciertas funciones según si el usuario está autenticado o no. Crea un decorador llamado requiere\_autenticacion que verifique si una variable global usuario\_autenticado es True antes de permitir la ejecución de una función. Si el usuario no está autenticado, imprime un mensaje de error.

usuario\_autenticado = False

def requiere\_autenticacion(func):

# Completa el decorador para verificar la autenticación

pass

@requiere\_autenticacion

def ver\_perfil():

print("Perfil del usuario")

# Intenta ejecutar la función decorada

ver\_perfil()

# Simula la autenticación del usuario

usuario\_autenticado = True

ver\_perfil()

### **Ejercicio 6: Decorador que limita el tiempo**

**Objetivo:** Crear un decorador con parámetros que simule un "timeout" o tiempo límite.

Implementa un decorador llamado timeout(segundos) que limite la ejecución de una función a un máximo de n segundos. Si la función tarda más de ese tiempo en ejecutarse, debe devolver un mensaje como "Tiempo de espera excedido".

import time

def timeout(segundos):

# Implementa aquí el decorador que limite el tiempo de ejecución

pass

@timeout(3)

def tarea\_larga():

time.sleep(5)

print("¡Tarea completada!")

# Llamada a la función decorada

tarea\_larga()

### **Ejercicio 7: Modificar el resultado según un parámetro**

**Objetivo:** Manipular el resultado de una función con un decorador que acepte parámetros.

Crea un decorador llamado multiplica\_resultado(factor) que multiplique el resultado de la función decorada por un valor dado.

def multiplica\_resultado(factor):

# Implementa aquí el decorador con parámetros

pass

@multiplica\_resultado(10)

def obtener\_numero():

return 7

# Llamada a la función decorada

print(obtener\_numero()) # Debería imprimir 70