Práctico 4

Decoradores en Python

1. **Decorador para imprimir antes y después**
   Crea un decorador que imprima un mensaje antes y después de ejecutar una función. Probalo con una función que imprima "Hola mundo".

2. Identifique cual es el error en este pequeño código con un decorador.

```python
def decorador_saludo():
    def wrapper():
        print("Hola desde el decorador")
    return wrapper

@decorador_saludo
def saludar():
    print("Hola desde la función")

saludar()
```

3. **Decorador para convertir a mayúsculas**
   Escribí un decorador que modifique el resultado de una función que devuelve texto, convirtiéndolo todo a mayúsculas.

4. **Decorador para repetir una función N veces**
   Escribí un decorador que reciba como parámetro un número entero y repita la ejecución de la función decorada esa cantidad de veces.

5. **Decorador para verificar contraseña antes de ejecutar**
   Creá un decorador que pregunte una contraseña antes de ejecutar la función. Solo si la contraseña es correcta, la función continuará.

6. **Decorador para registrar llamadas**
   Implementá un decorador que, cada vez que se llama a una función, guarde en un archivo de texto el nombre de la función y la hora de la llamada.

7. **Decorador para validar tipos de argumentos**
   Crea un decorador que verifique que todos los argumentos de una función sean enteros. Si no lo son, que imprima un mensaje de error. *(Utilizar la función type() por ejemplo)*

8. **Decorador para simular acceso restringido por rol**
   Crea un decorador que reciba un *rol* como parámetro. Si el rol es "admin", permite ejecutar la función. Si no, muestra un mensaje de acceso denegado.

9. Investigar los siguientes decoradores built-in de Python, que se encuentran entre los más utilizados: @*staticmethod*, @*property*, @*classmethod* Realizar un mini apunte personal con explicación de cada uno, y ejemplo de uso.