LIST - TRAINING TASKS

- I will only accept tasks submitted **by the deadline**.
- I'll give you feedback on whether the solution is correct or not. The most important thing to me in your code is the **output**.
- The program **must display** some information (e.g., popups or messages) when it receives input or is running in a loop, indicating what action the user should take.
- It will **not** be possible to resend the code.
- An incorrect solution may negatively affect your activity grade.

1. Goal

Create a **decorator factory** `@requires_role(role_name)` that restricts access to a function based on user roles.

🧠 Concepts to Use

- **Decorator factory** → a function that returns a decorator.
- **Global user data** → simulate current logged-in user with a `current_user` dictionary.
- **Function wrapping** → use `functools.wraps` to preserve metadata.
- **Error handling** → raise `PermissionError` if the role is missing.

```
# Try calling the functions

print(delete_post(42))  # ✅ Should work (user has "admin")
print(edit_post(7))     # ✅ Should work (user has "editor")
print(view_post(99))    # ❌ Should raise PermissionError (user lacks "viewer")
```

2. Simple Logging Decorator with Level

Objective:

Create a decorator factory `@log(level)` that prints a message before and after the execution of the function, based on the specified logging level.

📋 Instructions:

1. The decorator factory should accept a parameter `level`,
   e.g., `"INFO"`, `"DEBUG"`, `"WARNING"`, etc.
2. Before the function runs, print:
   `"LEVEL: Starting <function_name>"`
3. After the function runs, print:
   `"LEVEL: Finished <function_name>"`
4. Use `functools.wraps` to preserve function metadata.

3. Timing Decorator

Objective:

Create a decorator `@timer` that logs the time it takes to run a function.

📋 Instructions:

1. The decorator should print the execution time of the function.
2. Use the `time` module to calculate the time taken by the function.
3. Use `functools.wraps` to preserve the function's metadata.