Understanding and Implementing Python Decorators

Objective: The goal of this task is for students to understand how Python decorators work and to implement their own decorator functions. This exercise will help students get hands-on experience with function wrapping and applying additional functionality to existing code without modifying it directly.

Task 1: Basic Decorator Implementation

Create a simple Python decorator named `logger` that logs the execution of a function. The decorator should print:

1. A message before the function starts, indicating that the function is about to run.
2. The name of the function being called.
3. A message after the function has finished executing.

Use the decorator on the following function:

```
def greet(name):
    print(f"Hello, {name}!")
```

Task 2: Decorator with Arguments

Extend the `logger` decorator to accept an argument that specifies the log level (e.g., "INFO", "DEBUG", "ERROR"). Based on the log level provided, the decorator should print the level of logging in addition to the function call. For example:

```
@logger(log_level="DEBUG")
def calculate_sum(a, b):
    return a + b
```

Expected output when calling `calculate_sum(5, 10)`:

```
DEBUG: Starting execution of calculate_sum
DEBUG: The function calculate_sum executed with result: 15
```

Task 3: Chaining Multiple Decorators

Write a second decorator named `time_tracker` that tracks the time taken by a function to execute. Then, apply both `logger` and `time_tracker` to a function. The `time_tracker` decorator should print:

1. The start time of the function.
2. The time it took for the function to complete.

Apply both `logger` and `time_tracker` to the following function:

```
def compute_factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * compute_factorial(n - 1)
```

Task 4: Optional: Decorator for Input Validation

Create a decorator called `validate_inputs` that ensures the function receives non-negative integers as inputs. If a negative number is passed, the decorator should raise a `ValueError`. Use it on the following function:

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

Instructions:

1. Write a short explanation for each task, describing how the decorator works.
2. Implement each task in Python.
3. Provide test cases for each function, showing the output before and after applying the decorators.