Functional Programming, Decorators & Recursion – Exercises

Task 1: Execution-time decorator

Create a decorator that measures how long a function takes to execute.
The decorator should print the execution time after the function has finished.
Apply the decorator to a function that processes a list of numbers.

Task 2: Functional list transformation

Use a functional approach to transform a list of numbers.
Apply one operation that changes the values and one operation that filters values.
The solution must use lambda expressions together with built-in higher-order functions.

Task 3: Closure-based configuration

Create a function that returns another function.
The returned function should behave differently depending on a value captured from the outer function.
Create at least two functions from the same factory function and demonstrate the difference in behavior.

Task 4: Decorator that modifies return values

Create a decorator that intercepts the return value of a function.
Modify the return value in some meaningful way before returning it.
Apply the decorator to at least one function and demonstrate the effect.

Task 5: Static utility methods

Create a class that acts as a utility container.
The class should contain multiple static methods.
Each static method should perform a small, independent operation.
Demonstrate usage without creating any class instances.

Task 6: Higher-order function pipeline

Create a function that takes another function as input.
Chain multiple function calls together to process a value step by step.
Use both named functions and lambda expressions in the pipeline.

Task 7: Preserving function metadata

Create a decorator that wraps a function.
Ensure that the wrapped function retains its original metadata.
Verify this by inspecting the function’s name and documentation before and after decoration.

Challenge Task 8: Flexible decorator with arguments

Create a decorator that accepts its own arguments.
The decorator should alter its behavior based on the provided arguments.
Apply the decorator to multiple functions using different decorator arguments.

Challenge Task 9: Recursive problem design

Design a problem that can naturally be solved using recursion.
Implement a recursive solution.
Optimize the solution using memoization.
Demonstrate the difference in performance or behavior.

Challenge Task 10: Decorators + recursion interaction

Create a recursive function that performs a calculation.
Apply a decorator that logs information about each function call.
Observe how the decorator behaves during recursive execution and explain the result.