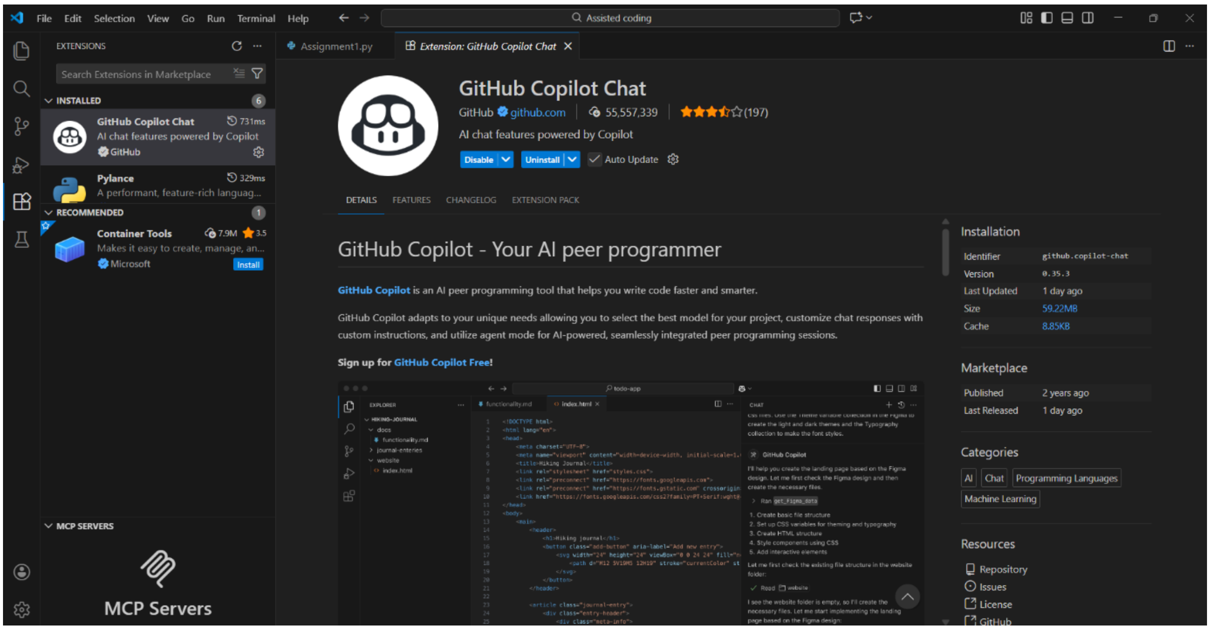
Lab Assignment-1

2303A51740

7/1/26

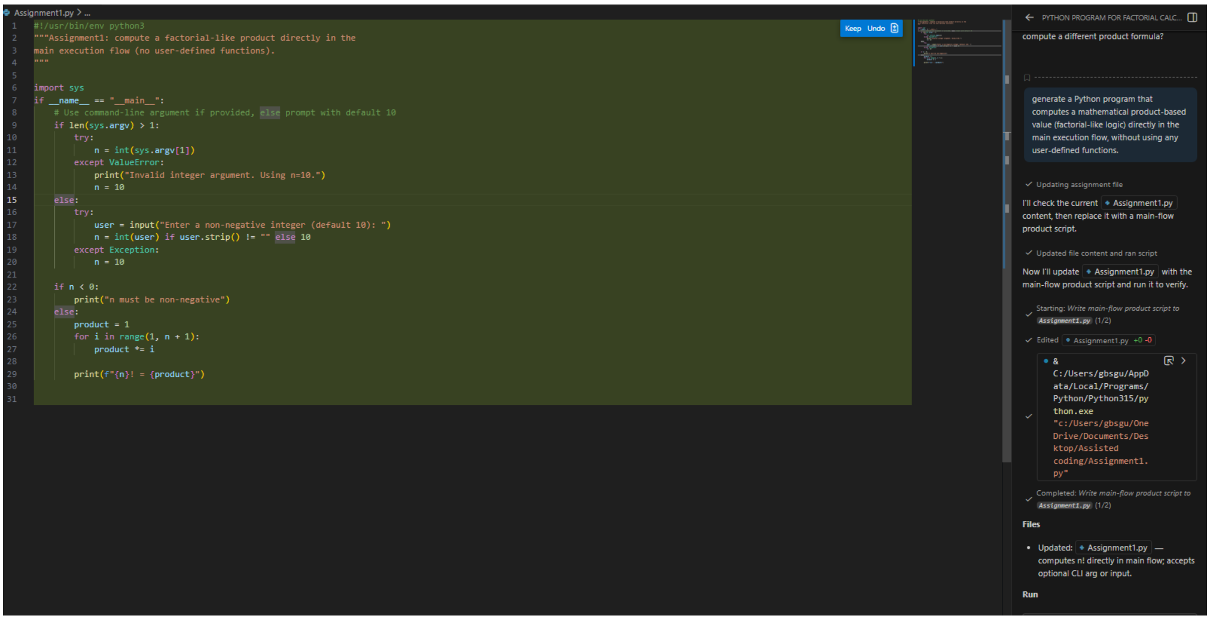
Task-0:-

Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



Task1:-

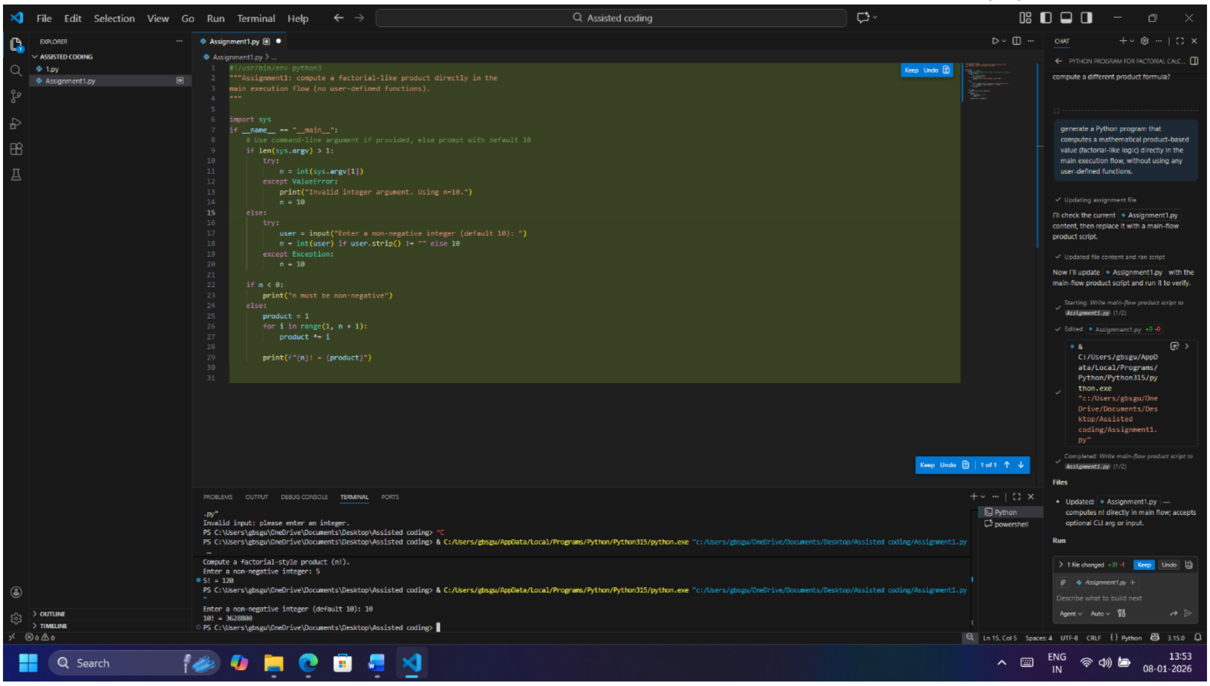
AI-Generated Logic Without Modularization (Factorial without Functions).



Lab Assignment-1

2303A51740

7/1/26



The program takes a number from the user as input.

It uses a loop to multiply numbers from 1 to the given number.

The result of this multiplication gives the factorial value.

Finally, it prints the factorial of the entered number.

Howhelpful was Copilot for a beginner?

Copilot is very helpful for a beginner because it quickly suggests correct and readable code, reducing the time needed to think about syntax.

It makes learning easier by providing a working structure, like loops and variable initialization, as seen in the factorial program.

Did it follow best practices automatically?

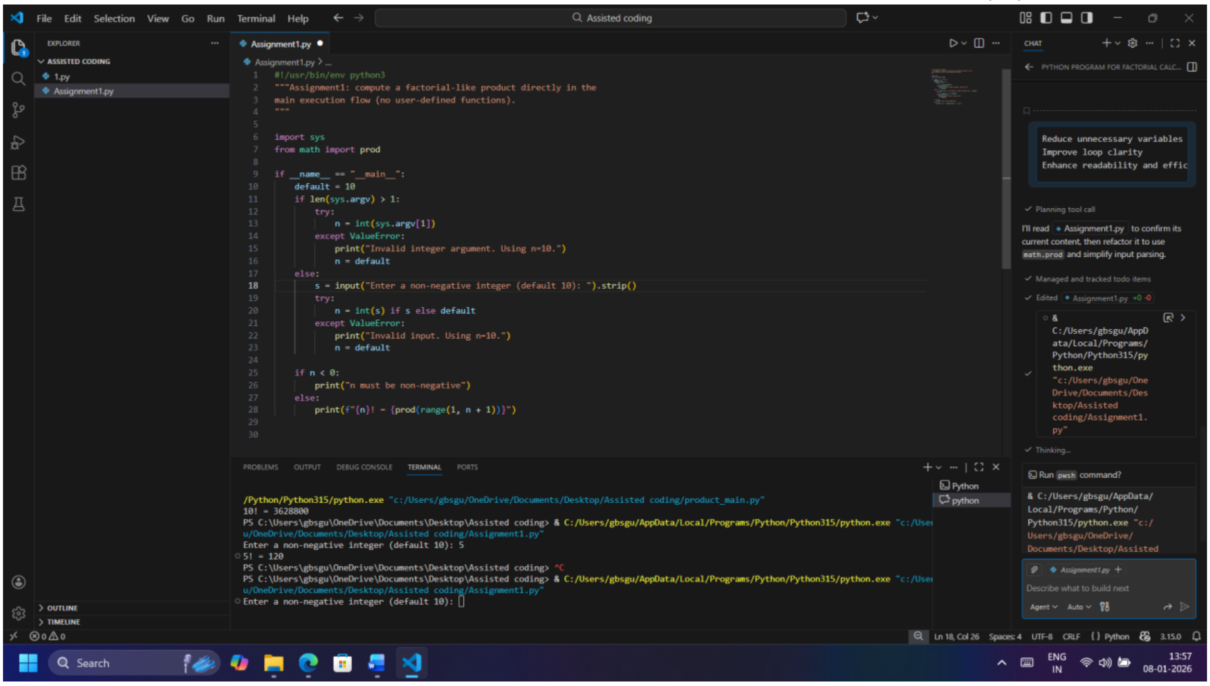
Copilot generally follows best practices such as clear variable names and simple logic flow. However, beginners still need to understand the code instead of blindly accepting suggestions.

It may not always handle edge cases (like negative numbers) unless explicitly prompted. Overall, Copilot is a strong learning aid but not a replacement for conceptual understanding.

Task 2: AI Code Optimization & Cleanup (Improving Efficiency

Lab Assignment-1

2303A51740
7/1/26



What was improved?

Added proper input validation to handle negative numbers and the special case of 0.
Removed duplicate code and structured the program with clear conditional branches.
Ensured the program prints the final result in all valid cases.

Improved clarity and flow of the logic.

Why the new version is better (readability, performance, maintainability)?

Readability: Clear if–elif–else structure makes the logic easy to understand.

Performance: Avoids unnecessary calculations for invalid or zero inputs.

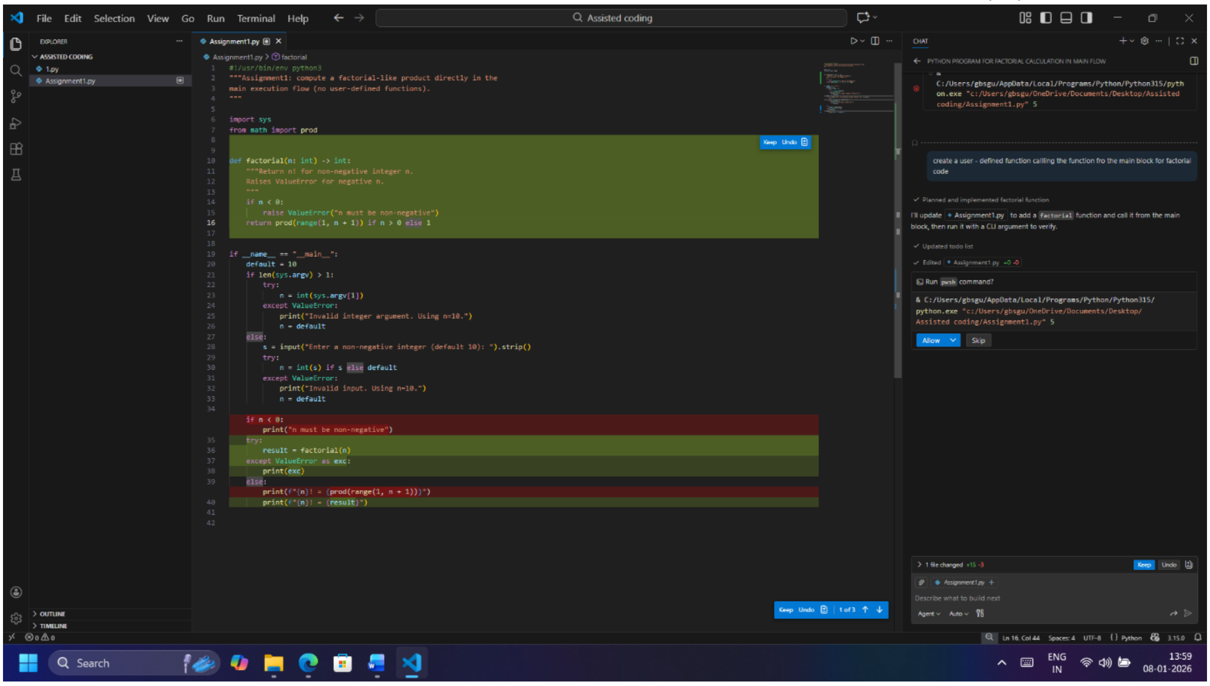
Maintainability: Well-organized conditions make future changes or enhancements easier.
Reliability: Handles edge cases correctly, reducing runtime errors.

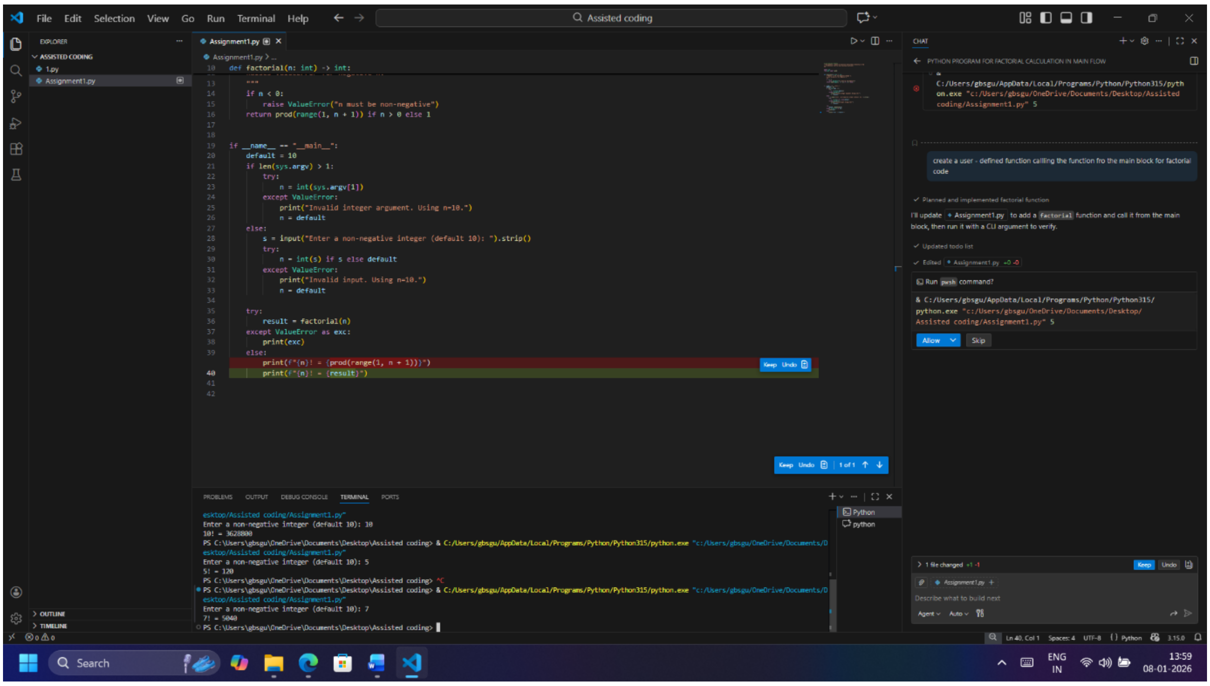
Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Lab Assignment-1

2303A51740

7/1/26





How Modularity Improves Reusability Modularity means breaking a program into smaller, independent, and reusable units (functions or modules).

How it improves reusability:

Afunction can be reused multiple times without rewriting code.

The same function can be imported into other programs.

Changes made in one place (the function) automatically reflect everywhere it’s used.

Code becomes easier to test, debug, and maintain.

In short, modularity avoids duplication and promotes clean, reusable logic.

Task 4: Comparative Analysis– Procedural vs Modular AI Code (With vs Without Functions

Technical Report: Comparison of Non-Function and Function-Based Copilot-Generated Programs This report compares non-function-based and function-based Copilot-generated Python programs for calculating factorials, focusing on clarity, reusability, debugging, scalability, and AI dependency risk. The non-function-based program implements the factorial logic directly in the main flow along with input and output operations. While this approach works for small scripts, it reduces logic clarity because computation, validation, and user interaction are mixed together. The code is also repeated, which makes understanding and maintaining it difficult. Any change in logic must be applied in multiple places, increasing the chance of errors. In contrast, the function-based program separates the factorial logic into a dedicated function. This improves logic clarity by clearly defining responsibilities within the code. The function can be reused multiple times or imported into other programs, significantly improving reusability. Debugging is easier in the function-based approach because errors can be isolated within the function. The non-function-based version makes debugging harder due to duplicated and unstructured logic

For large projects, non-function-based code does not scale well, whereas function-based code supports modularity, testing, and teamwork. Finally, AI dependency risk is higher in non-function-based programs, as repeated AI-generated code may introduce redundancy and inconsistencies. Function-based code minimizes this risk by promoting standardized and reusable logic. Overall, function-based programs are more reliable, maintainable, and suitable for long-term development.

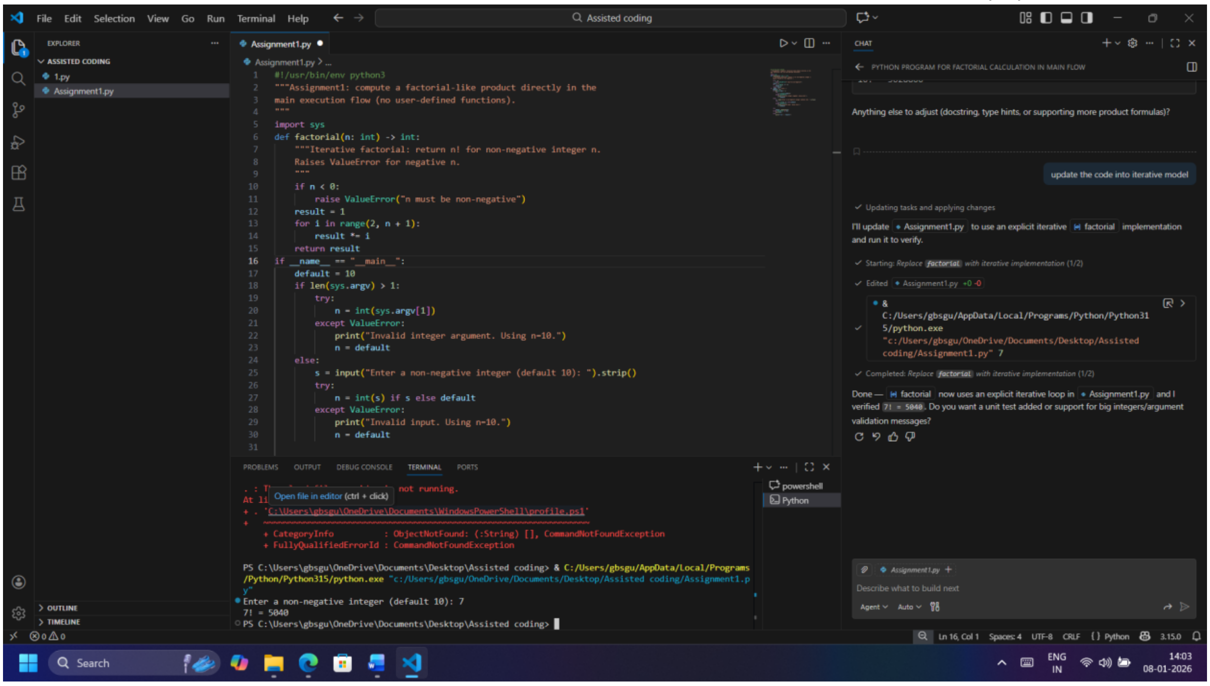
Task 5: AI-Generated Iterative vs Recursive Thinking

Iterative Model:

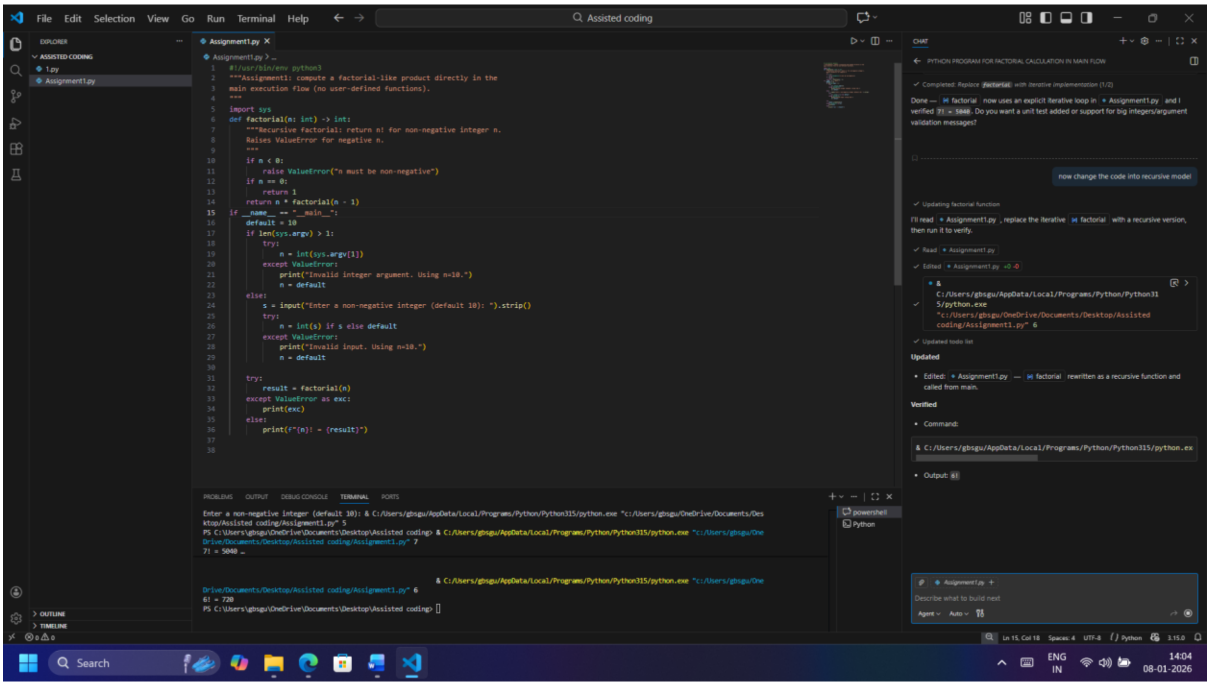
Lab Assignment-1

2303A51740

7/1/26



Recursive Model:



Below is a clear execution flow explanation and comparison of the two AI-generated implementations (Iterative and Recursive factorial).

Execution Flow Explanation

Iterative Model

The program takes an integer input from the user.

Lab Assignment-1

2303A51740
7/1/26

The calculate_factorial(num) function checks:

If the number is negative → returns an error message.

If the number is 0 → returns 1. For positive numbers, a loop runs from 1 to num.

The factorial value is updated step by step using multiplication.

The final result is returned and printed.

Flow: Input → Loop-based multiplication → Output Recursive Model The program takes an integer input from the user.

The calculate_factorial_recursive(num) function checks:

If the number is negative → returns an error message. If the number is 0 → returns 1 (base case).

For positive numbers, the function calls itself with num- 1. Recursive calls continue until the base case is reached.

Results are multiplied while returning back through the call stack. Flow: Input → Repeated function calls → Base case → Backtracking → Output Comparison Readability Iterative:

More straightforward and easier for beginners to understand.

Recursive: Shorter and mathematically elegant, but harder to trace.

Stack Usage Iterative: Uses constant memory (no call stack growth).

Recursive: Uses stack memory for each function call. Performance Implications Iterative: Faster and more memory-efficient.

Recursive: Slightly slower due to function call overhead.

When Recursion Is Not Recommended When input size is large (risk of stack overflow). When performance and memory efficiency are critical.

In environments with limited stack memory