AI ASSISTED CODING – Assignment Number 1.1

Course Code - 24CS002PC215

Course Title - AI Assisted Coding

Year / Sem - II / I ( 2 – 1 )

Regulation - R24

Date : 24-08-2025

Student Name : G.Sanjansah

Hall Ticket Number : 2503a52l20

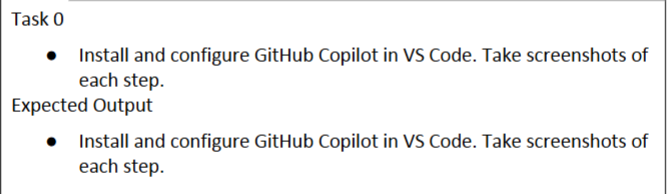
Student Mail-ID : 2503a52l20@sru.edu.in

# Task 0: Install and Configure GitHub Copilot

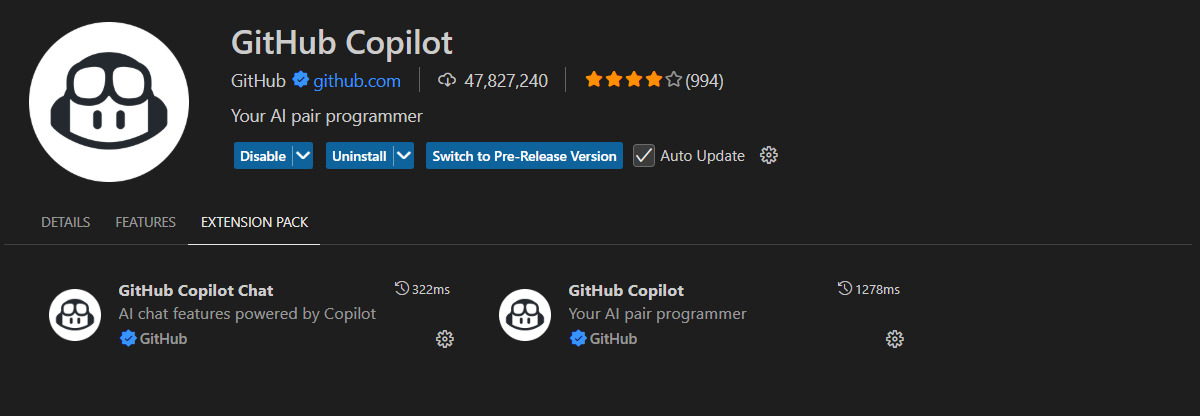
Description:  
Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Expected Output:  
- Install and configure GitHub Copilot in VS Code with screenshots.

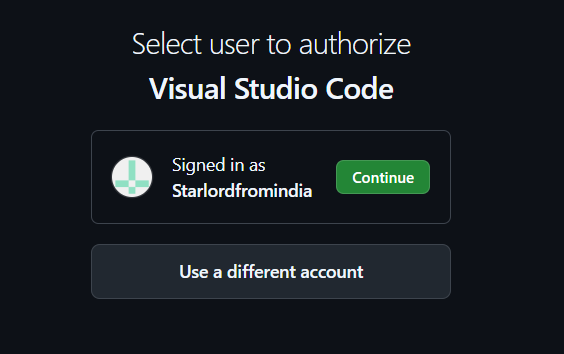
Step 1: Open VS Code Extension Marketplace



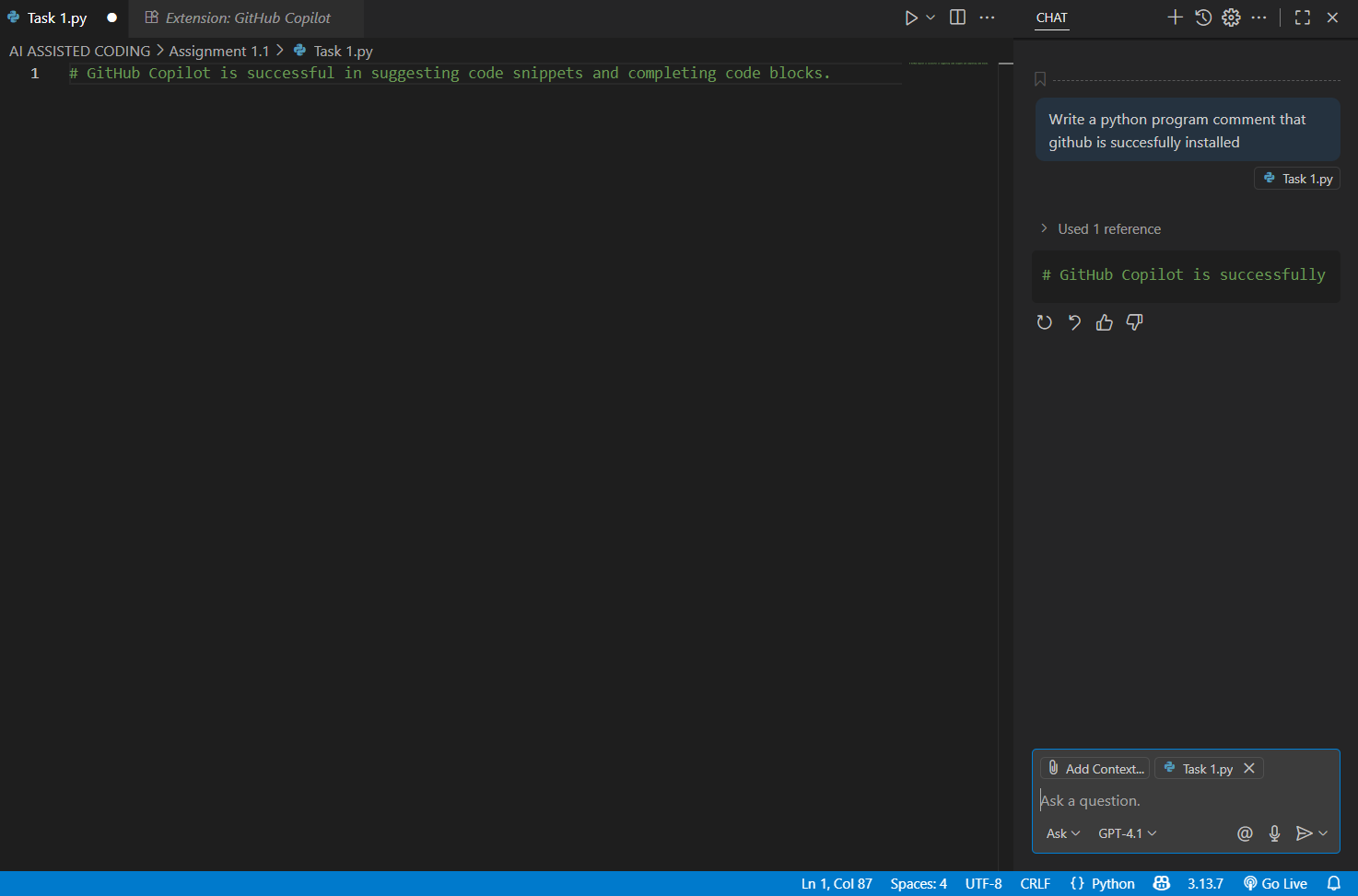
Step 2: Install GitHub Copilot extension



Step 3: Authorize GitHub account



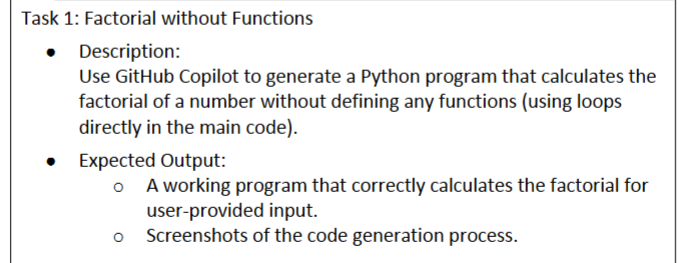
Step 4: Verify Copilot suggestions in VS Code

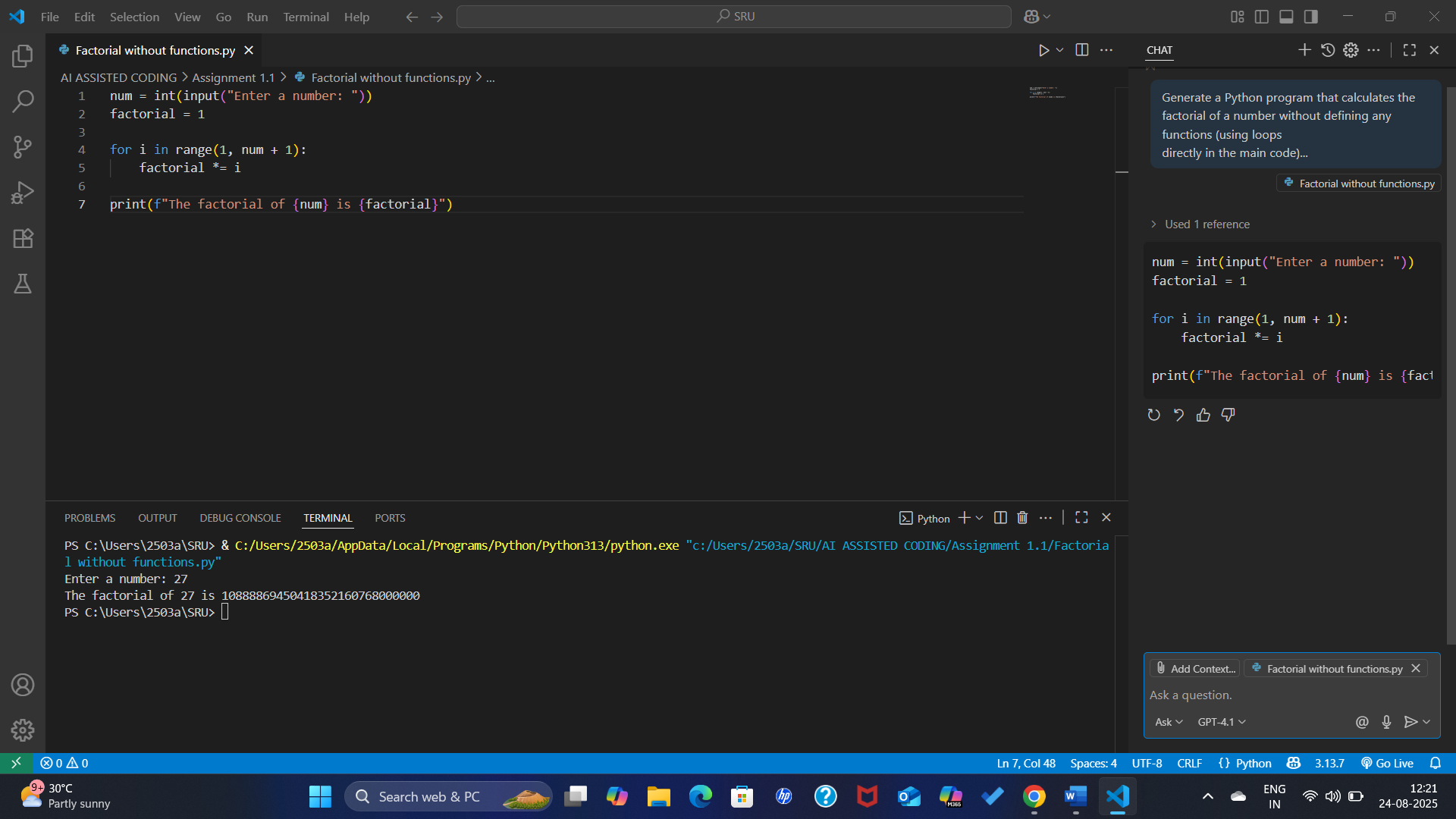


# Task 1: Factorial without Functions

Description:  
Use GitHub Copilot to generate a Python program that calculates the factorial of a number without defining any functions (using loops directly in the main code).

Expected Output:  
- A working program that correctly calculates the factorial for user-provided input.  
- Screenshots of the code generation process.

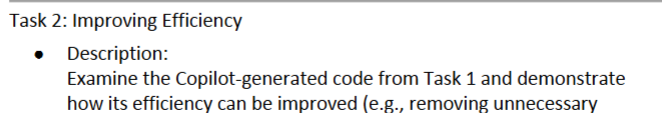


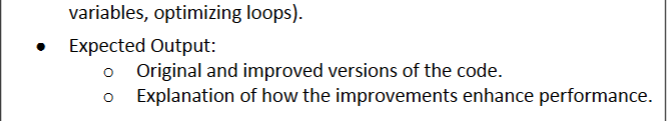


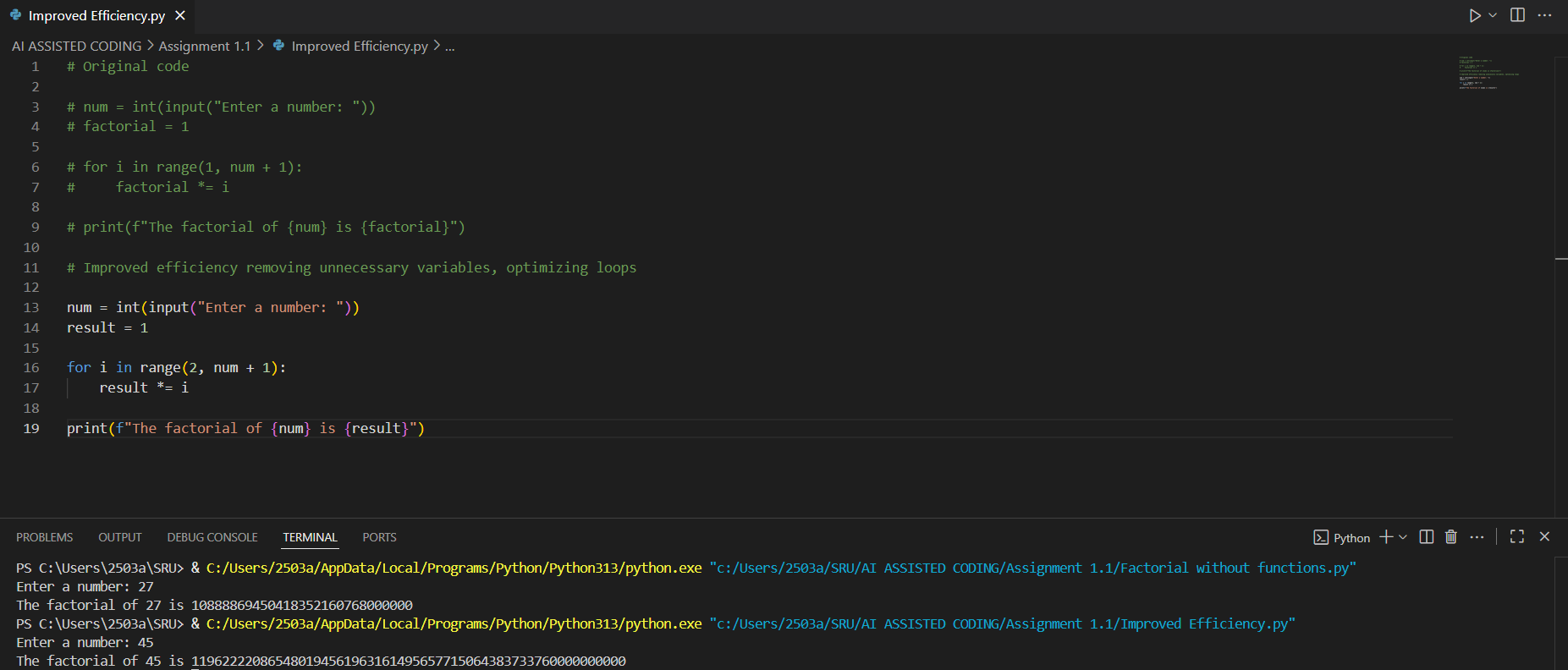
# Task 2: Improving Efficiency

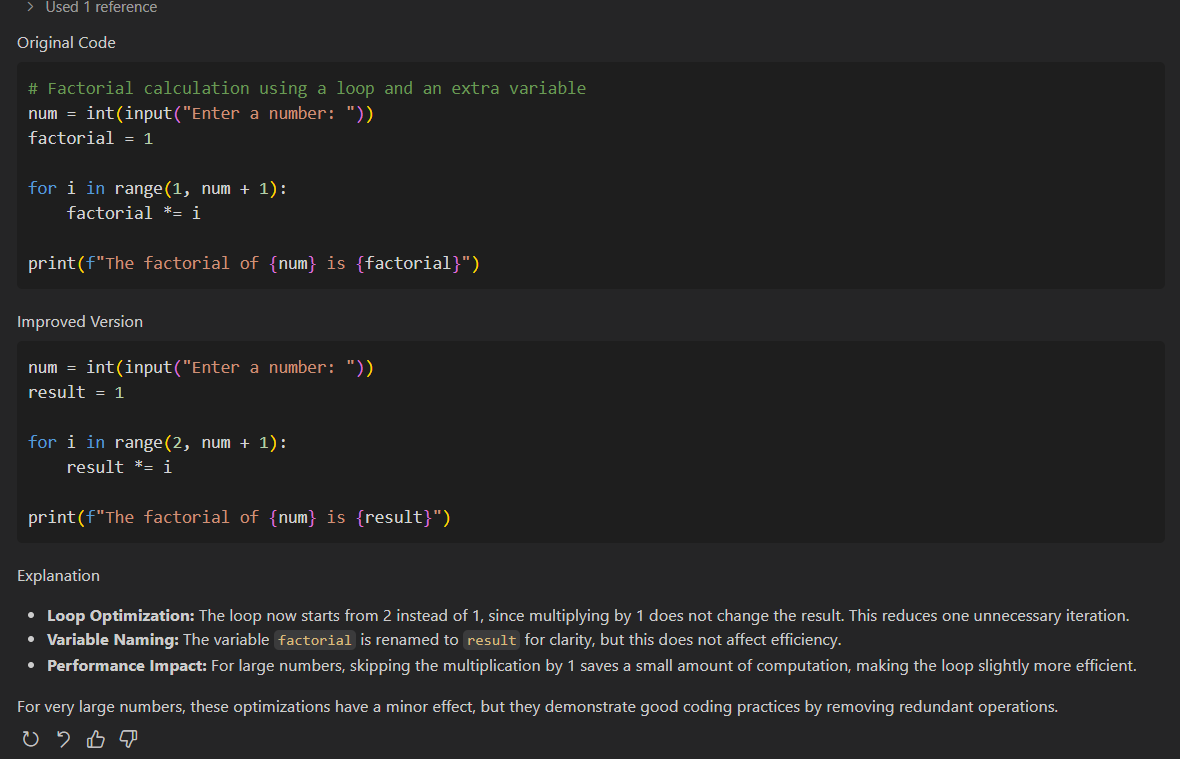
Description:  
Examine the Copilot-generated code from Task 1 and demonstrate how its efficiency can be improved (e.g., removing unnecessary variables, optimizing loops).

Expected Output:  
- Original and improved versions of the code.  
- Explanation of how the improvements enhance performance.





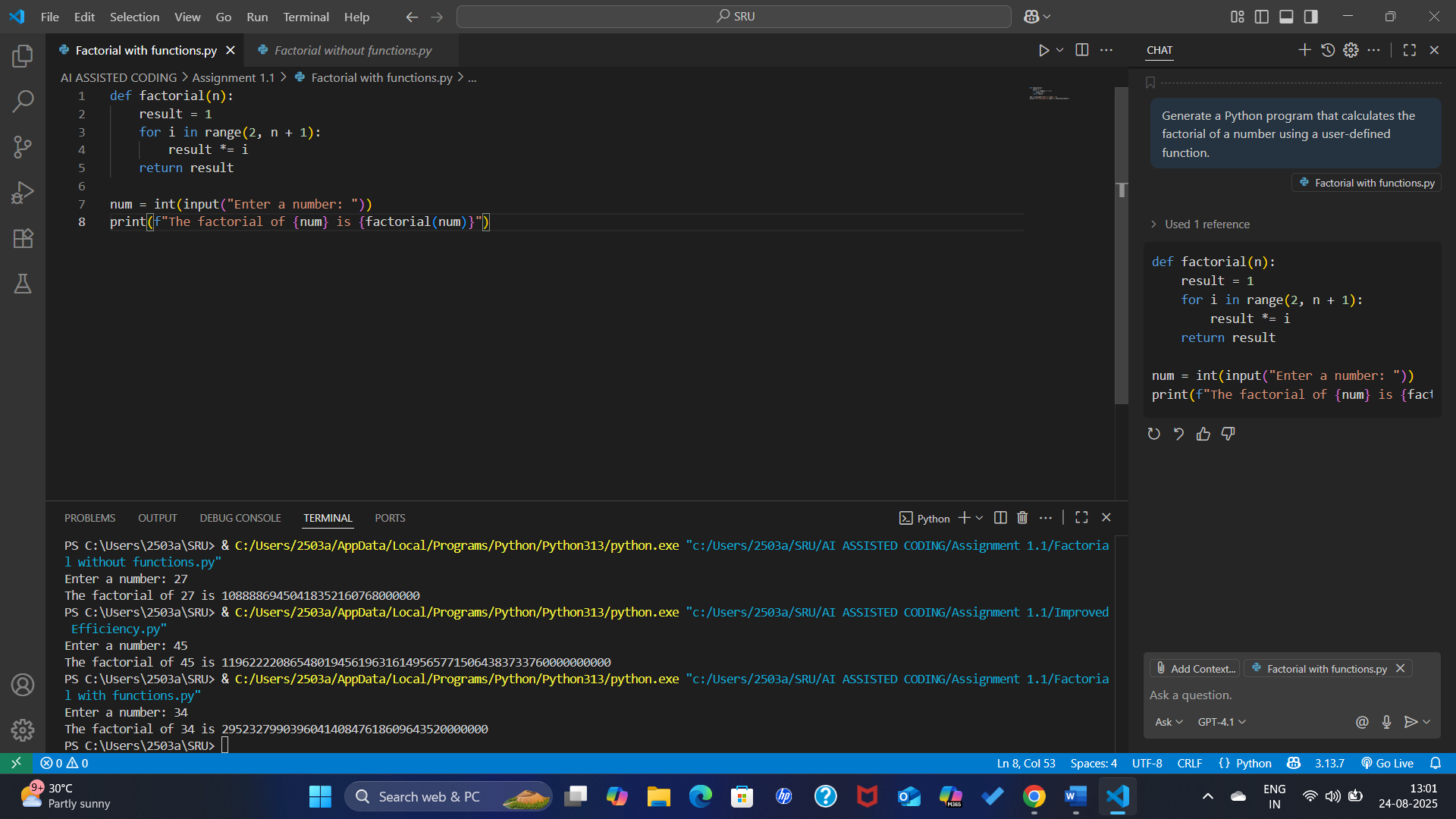


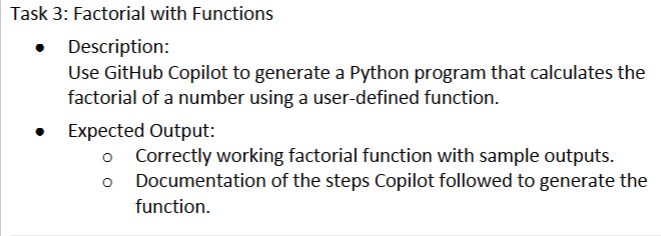


# Task 3: Factorial with Functions

Description:  
Use GitHub Copilot to generate a Python program that calculates the factorial of a number using a user-defined function.

Expected Output:  
- Correctly working factorial function with sample outputs.  
- Documentation of the steps Copilot followed to generate the function.





# Task 4: Comparative Analysis – With vs Without Functions

Description:  
Differentiate between the Copilot-generated factorial program with functions and without functions in terms of logic, reusability, and execution.

Expected Output:  
- A comparison table or short report explaining the differences.

Report: Difference Between Copilot-Generated Factorial Program With Functions and Without Functions

1. Logic  
- With Functions: The logic is modular. The calculation of factorial is separated into a function (factorial(n)) which takes an input, processes it, and returns the result.  
- Without Functions: The logic is written directly in the main program. The factorial calculation is done inline without separation, making it less structured.

2. Reusability  
- With Functions: Highly reusable. The factorial() function can be called multiple times for different inputs without rewriting the logic. It can also be imported and used in other programs.  
- Without Functions: Not reusable. If the program needs factorial in multiple places, the same loop logic has to be rewritten each time.

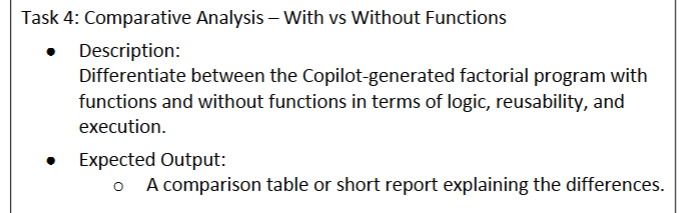
3. Execution  
- With Functions: Slightly more organized execution. The program first defines the function and then executes it when called. Execution is clean, and debugging is easier.  
- Without Functions: Executes directly in sequence. While it works fine for simple tasks, the code can become messy if the program grows larger.

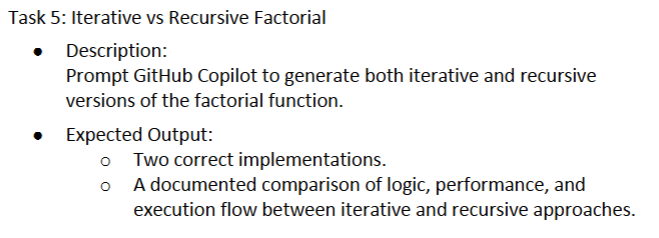
Conclusion:  
The function-based factorial program is more structured, reusable, and suitable for larger applications, while the non-function version is simpler but limited to one-time use.

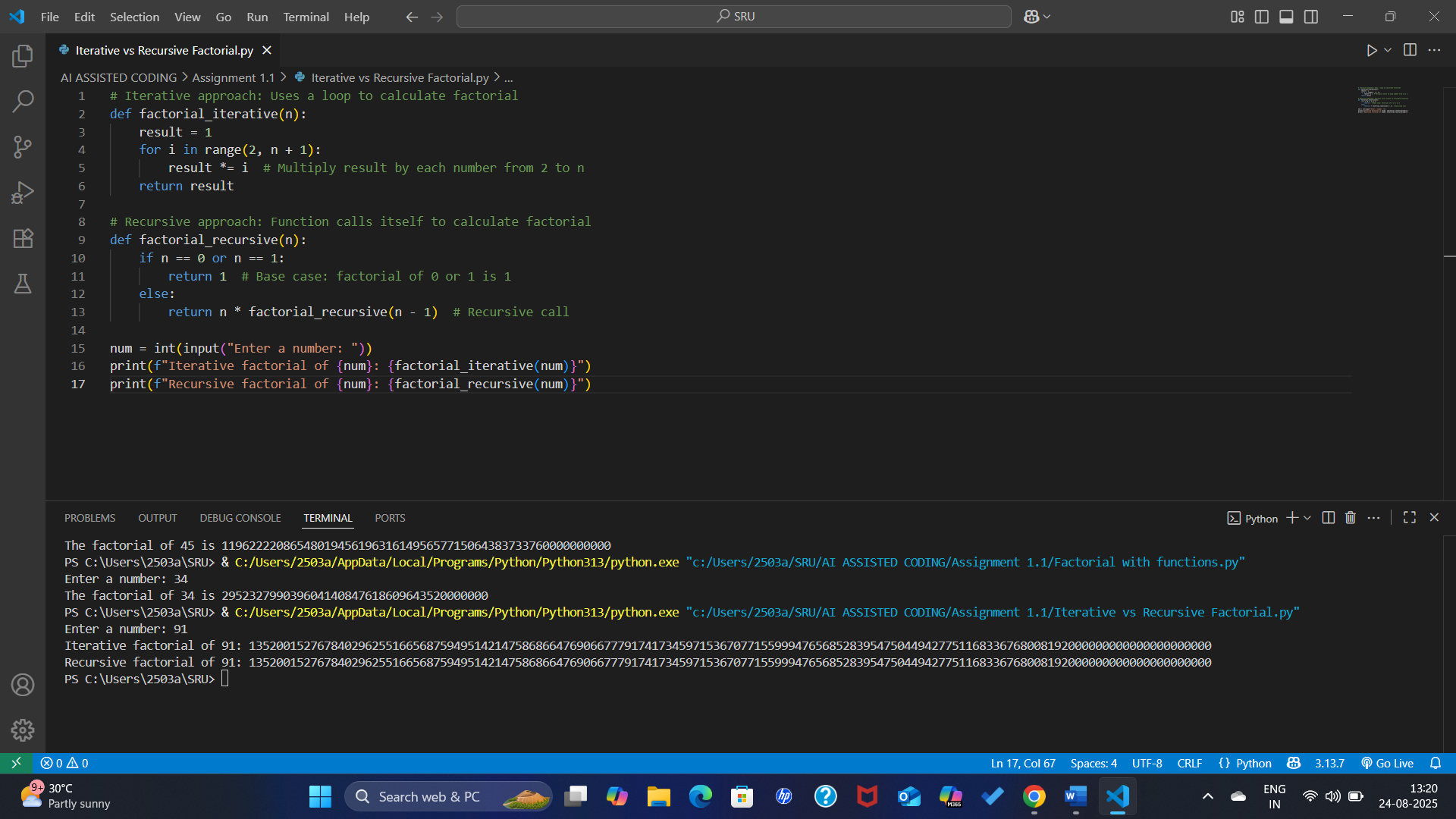
# Task 5: Iterative vs Recursive Factorial

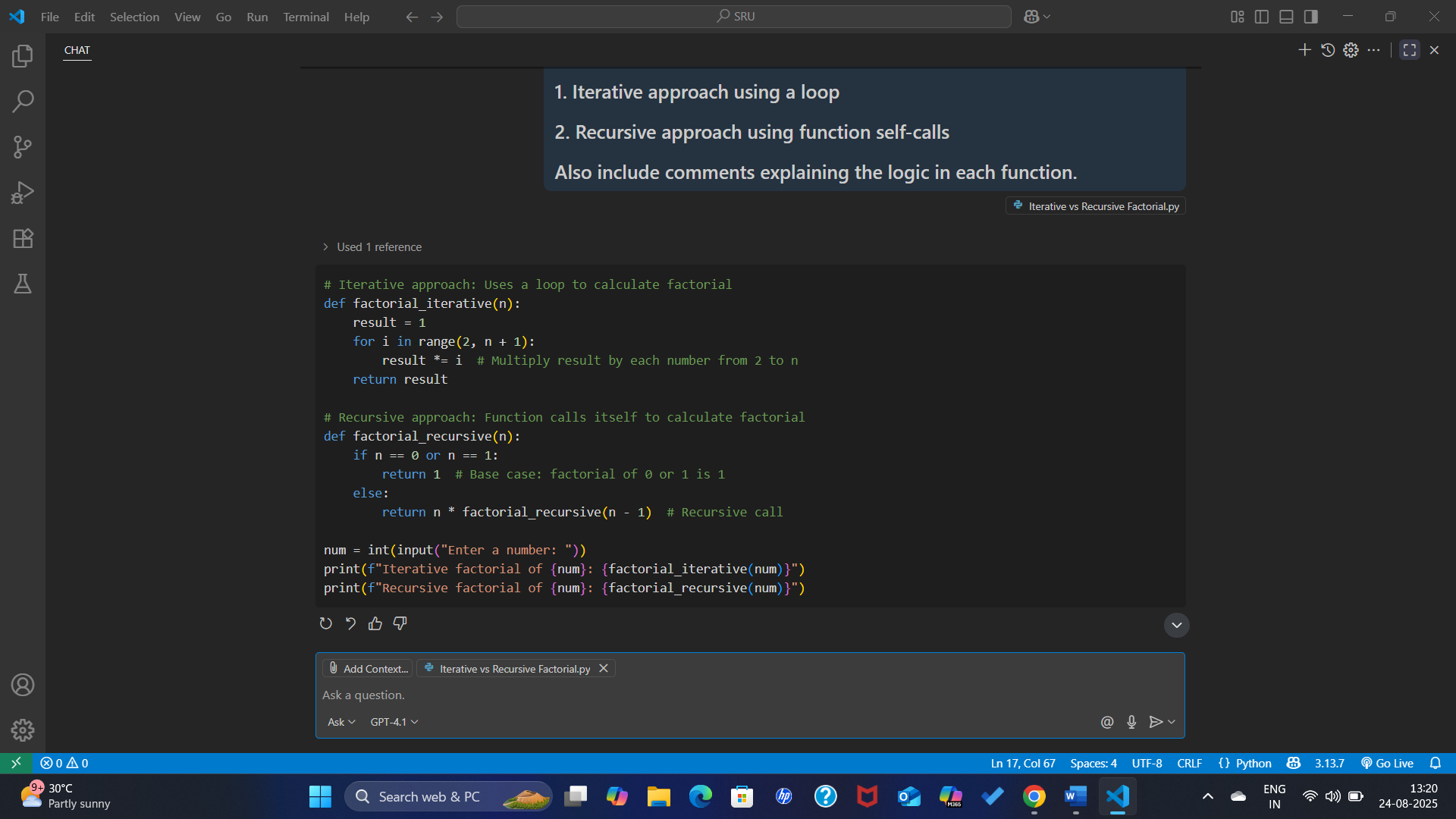
Description:  
Prompt GitHub Copilot to generate both iterative and recursive versions of the factorial function.

Expected Output:  
- Two correct implementations.  
- A documented comparison of logic, performance, and execution flow between iterative and recursive approaches.









Report: Iterative vs Recursive Factorial

1. Logic  
- Iterative Approach: Uses a loop (for) to multiply numbers from 2 up to n. The calculation is done step by step in a single function without extra function calls.  
- Recursive Approach: The function repeatedly calls itself, breaking the problem into smaller subproblems until it reaches the base case (n == 0 or n == 1).

2. Performance  
- Iterative: More memory-efficient since it only uses a loop variable and accumulator. Execution is generally faster because no extra function calls are made.  
- Recursive: Less efficient for large n because each function call consumes stack memory. Deep recursion can lead to a stack overflow if n is very large.

3. Execution Flow  
- Iterative: Executes sequentially in a loop until the result is obtained. Easy to trace and debug.  
- Recursive: Execution flows downward into multiple function calls and then resolves back upward when base cases are reached. More elegant and mathematical, but harder to trace for beginners.

Conclusion:  
Both approaches correctly compute factorial. The iterative version is more efficient and practical for large inputs, while the recursive version is conceptually elegant and mirrors the mathematical definition of factorial.