SR UNIVERSITY

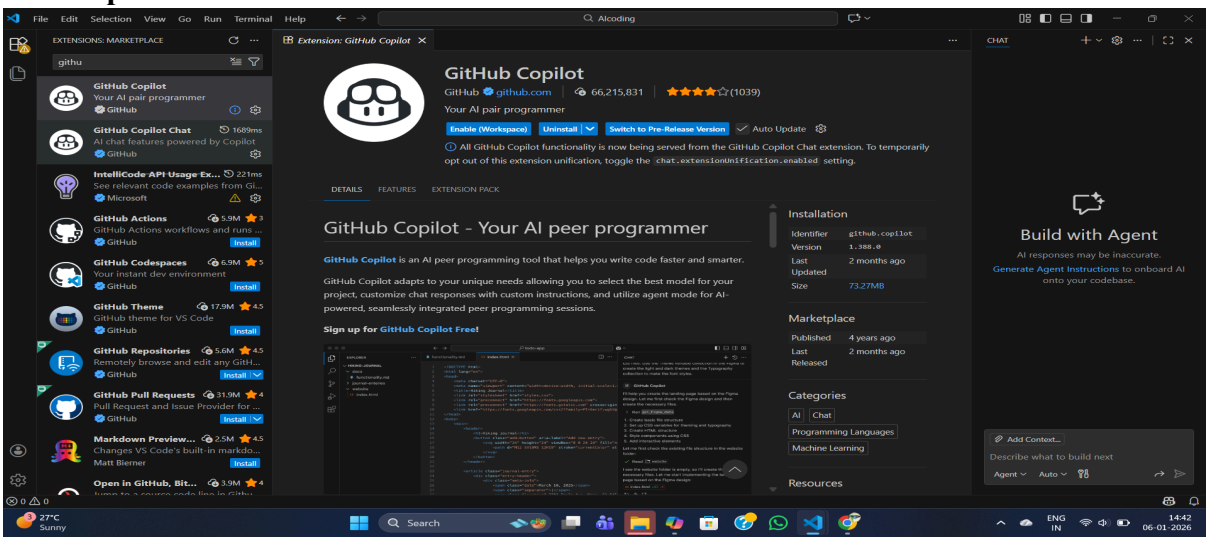
School of Computer Science and Artificial Intelligence

Lab Assignment # 1

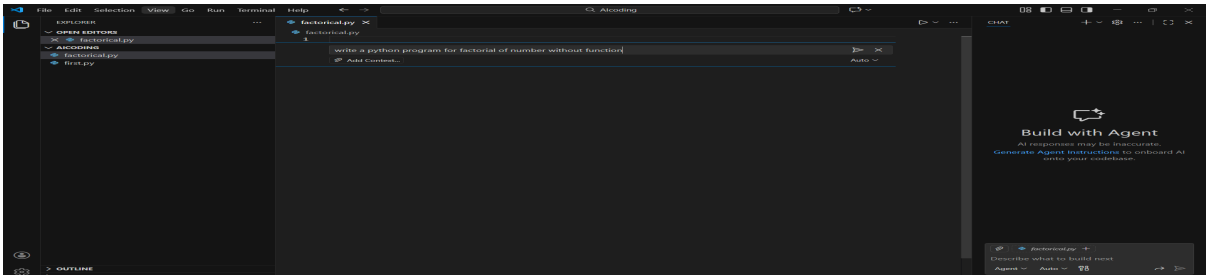
Program	**: B. Tech (CSE)**
Specialization	**:**
Course Title	**: AI Assisted coding**
Course Code	**: 23CS201PC302**
Semester	**: II**
Academic Session	**: 2025-2026**
Name of Student	**: Mohammed Hashir Sayam Hussain**
Enrollment No.	**: 2403A51L27**
Batch No.	**: 51**
Date	**:06-01-2026**

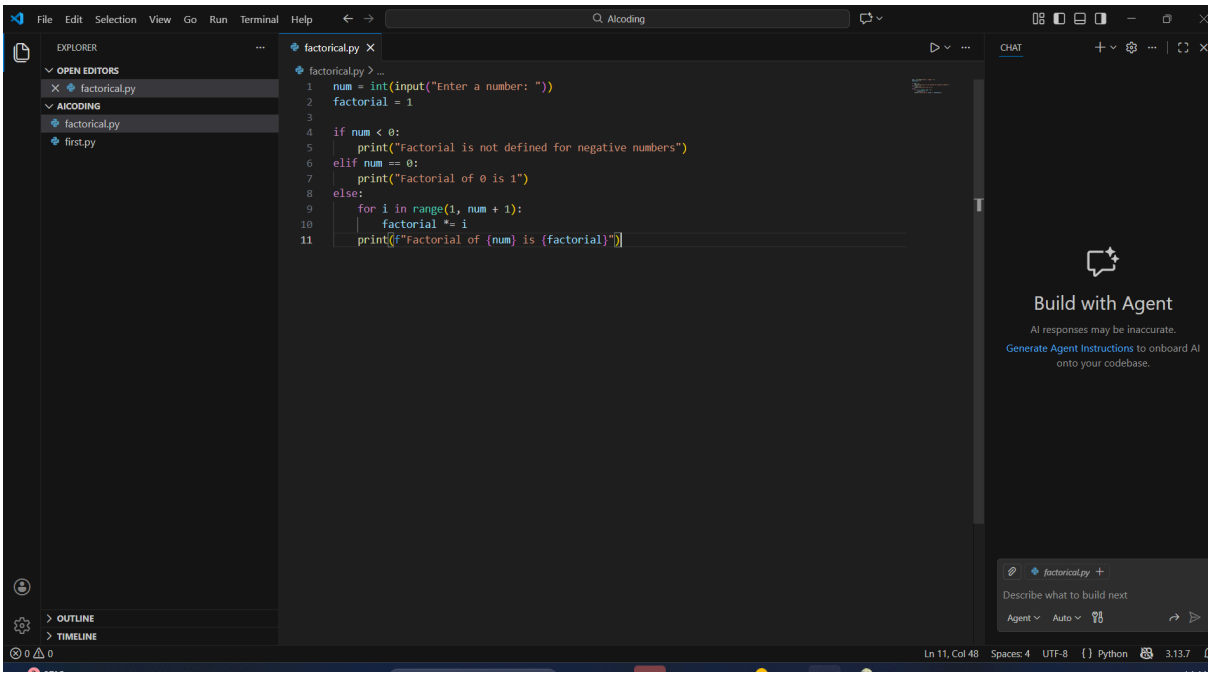
Submission Starts here

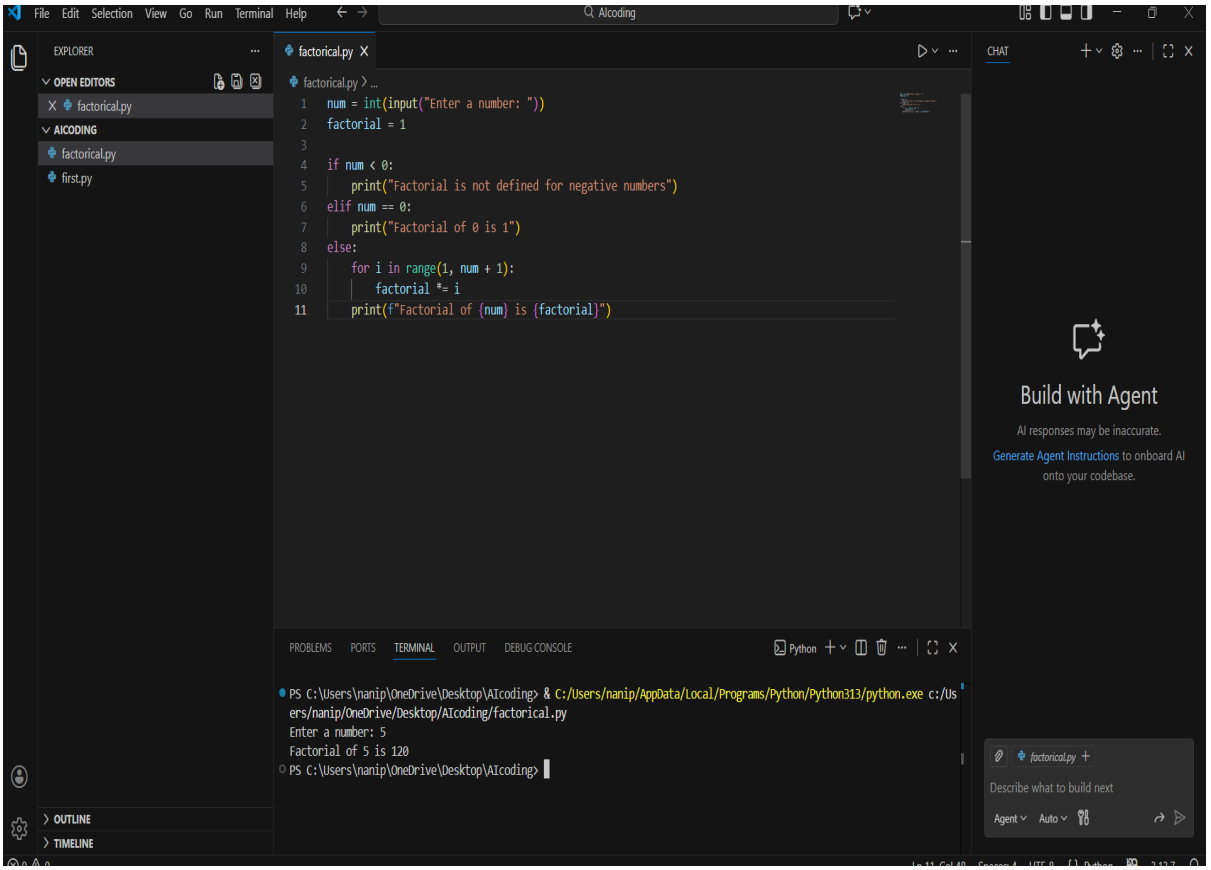
OUTPUT :
SCREENSHOTS:
Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



Task1: Task Description
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.







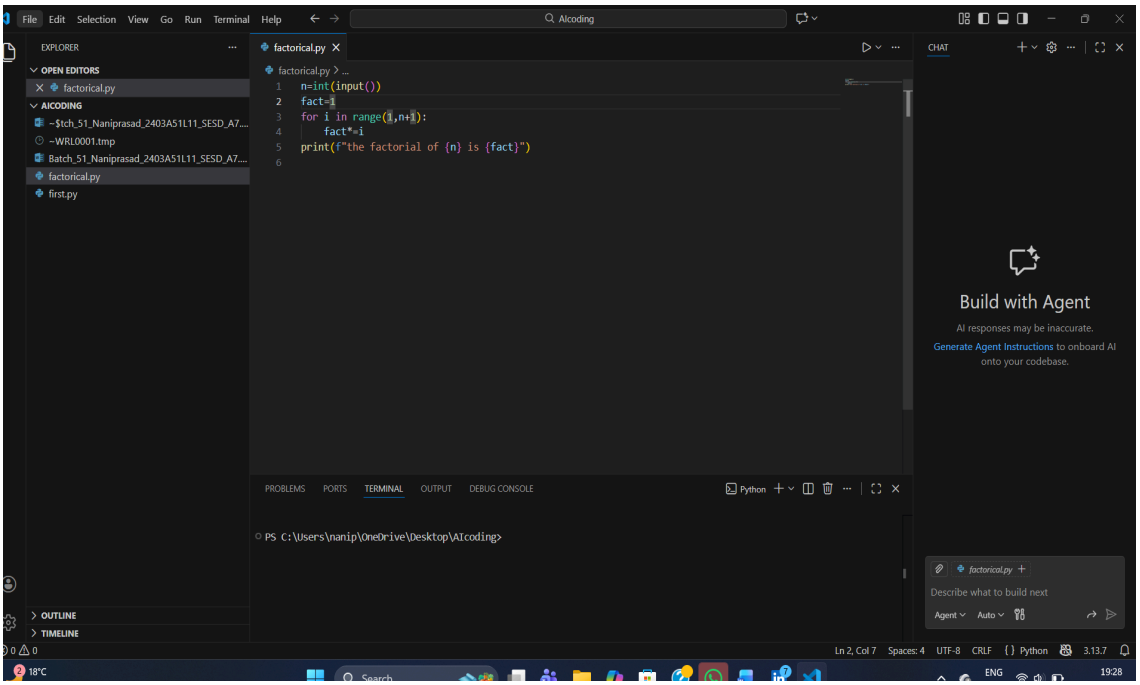
- **The Copilot is very helpful because we can generate code by just giving a prompt in Copilot Chat (ctrl + I)**

- **The code generated was as requested in the prompt**

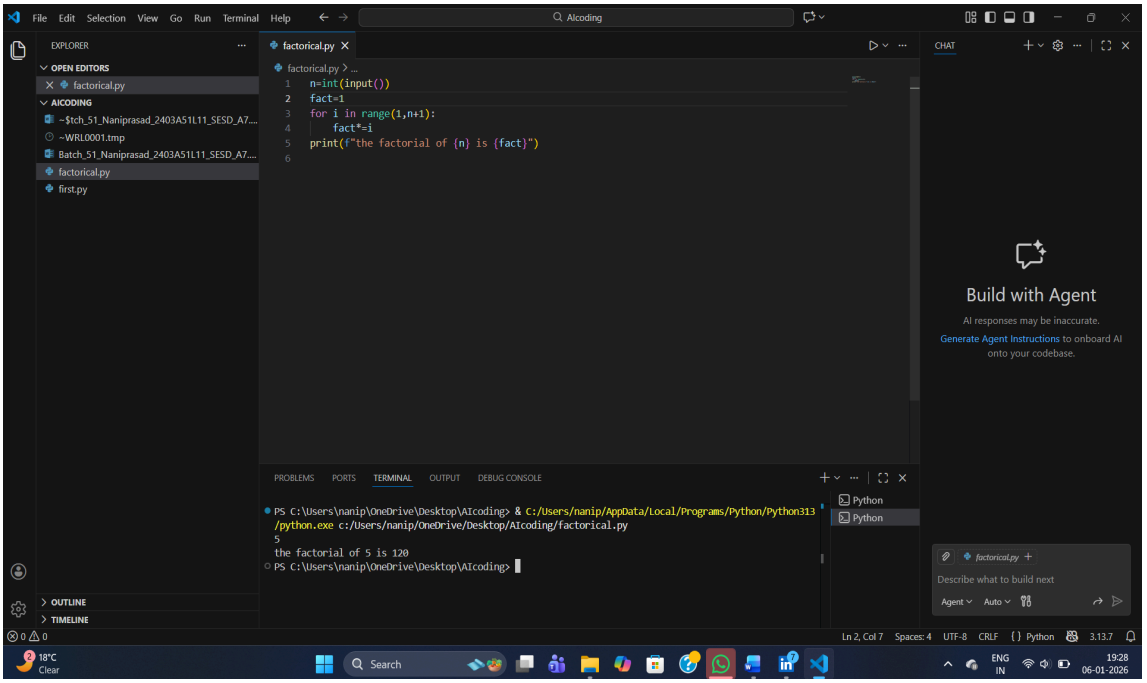
 # TASK - 2

Task Description
Analyze the code generated in Task 1 and use Copilot again to:

- **Reduce unnecessary variables**
- **Improve loop clarity**
- **Enhance readability and efficiency**



SR UNIVERSITY



 # What was improved?

- **Shorter multiplication statement**
- **factorial = factorial * i → factorial *= i**
- **Removed unnecessary comment**

❖ **The loop logic is self-explanatory, so the comment was removed.**

❖ **# Why the new version is better?**

❖ **Readability**

❖ ***= is clearer and more concise.**

- **Fewer lines and less clutter make the code easier to read.**

❖ **Maintainability**

- **Cleaner code is easier to modify and debug.**
- **Reduced redundancy lowers the chance of mistakes.**

❖ **Performance**

- **Performance is effectively the same.**

❖ ***= is marginally optimized at the bytecode level, but the difference is negligible.**

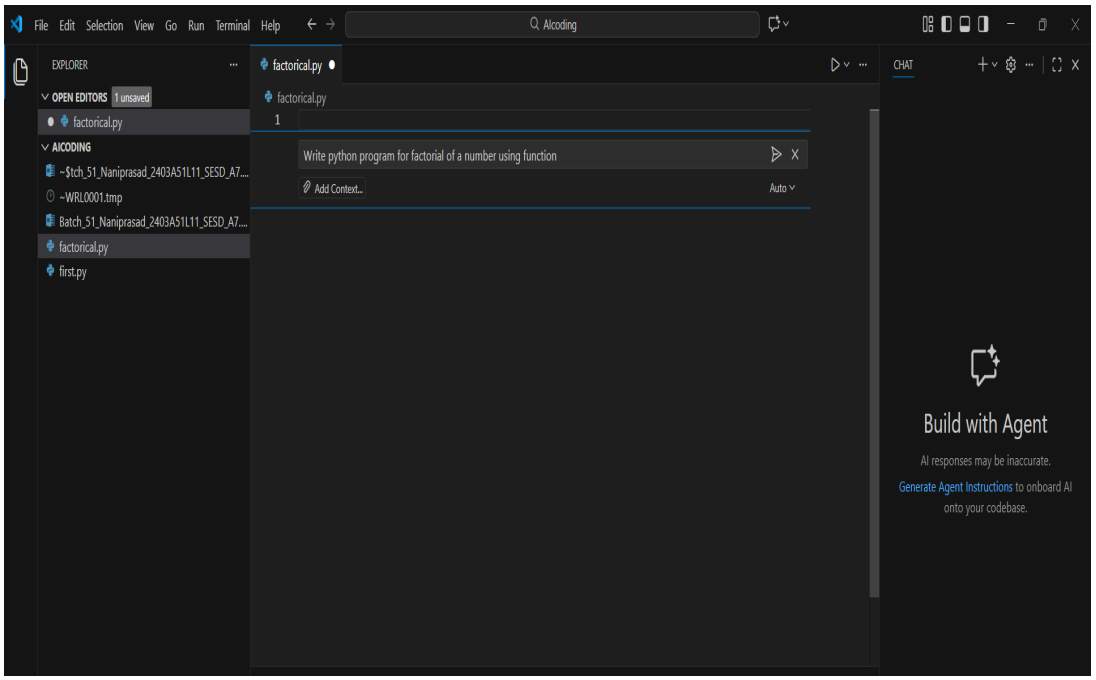
 # Task3

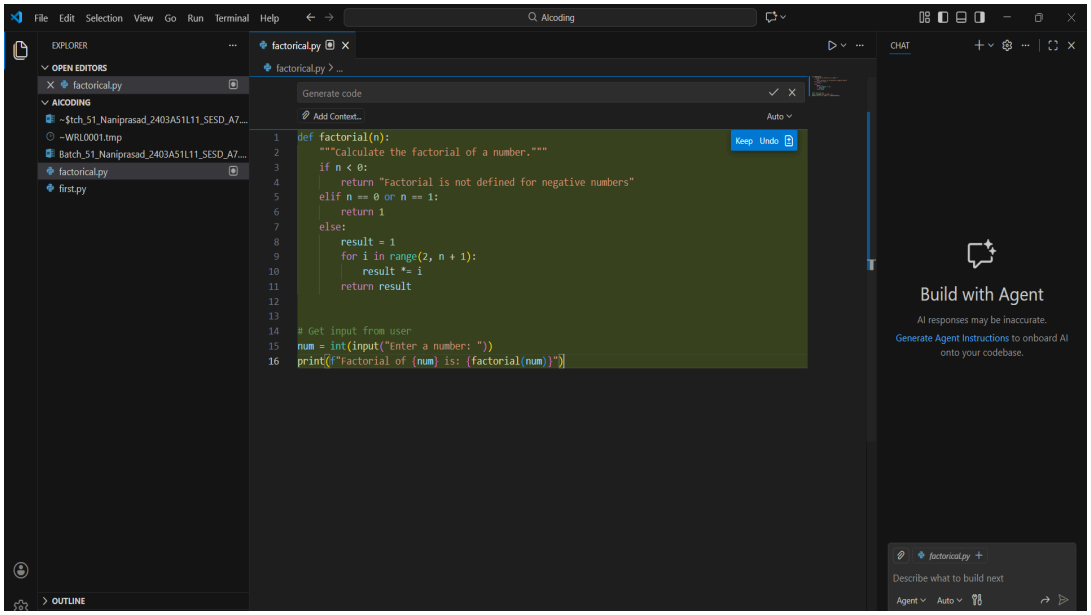
Task Description

SR UNIVERSITY

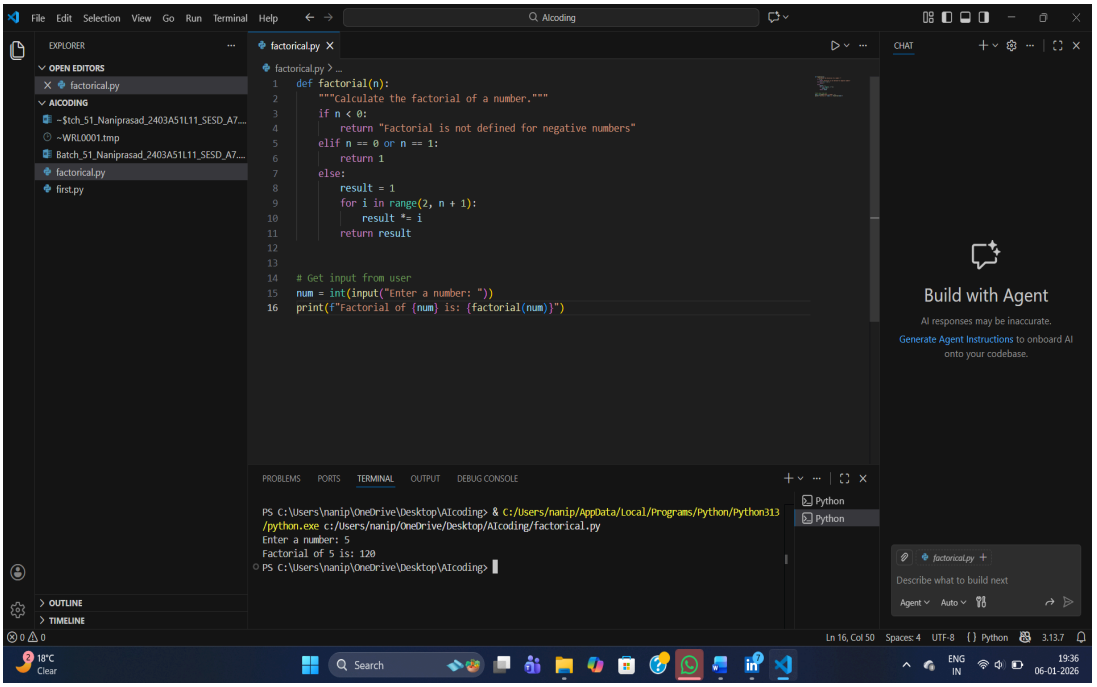
Use GitHub Copilot to generate a modular version of the program by:

- **Creating a user-defined function**
- **Calling the function from the main block**





SR UNIVERSITY



- **Modularity improves reusability by:**

  Allowing the factorial() function to be reused in multiple programs without rewriting code.

  Making the program easier to test, update, and debug.

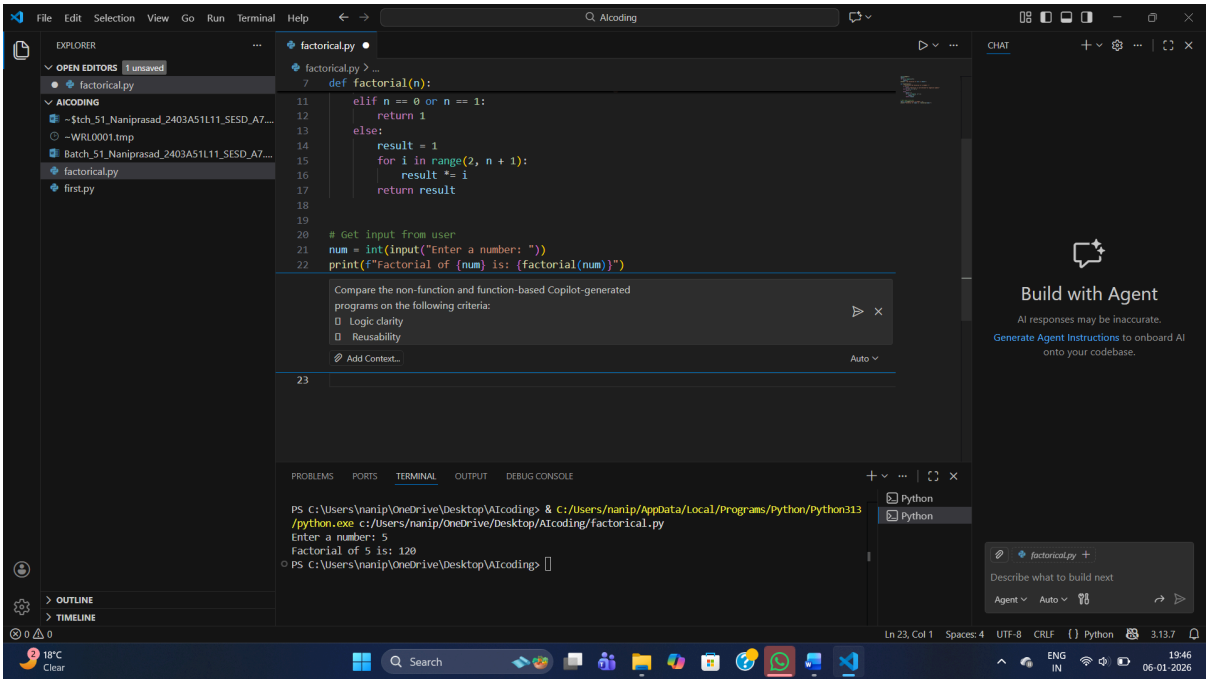
  Improving code organization, where logic is separated from input/output handling.

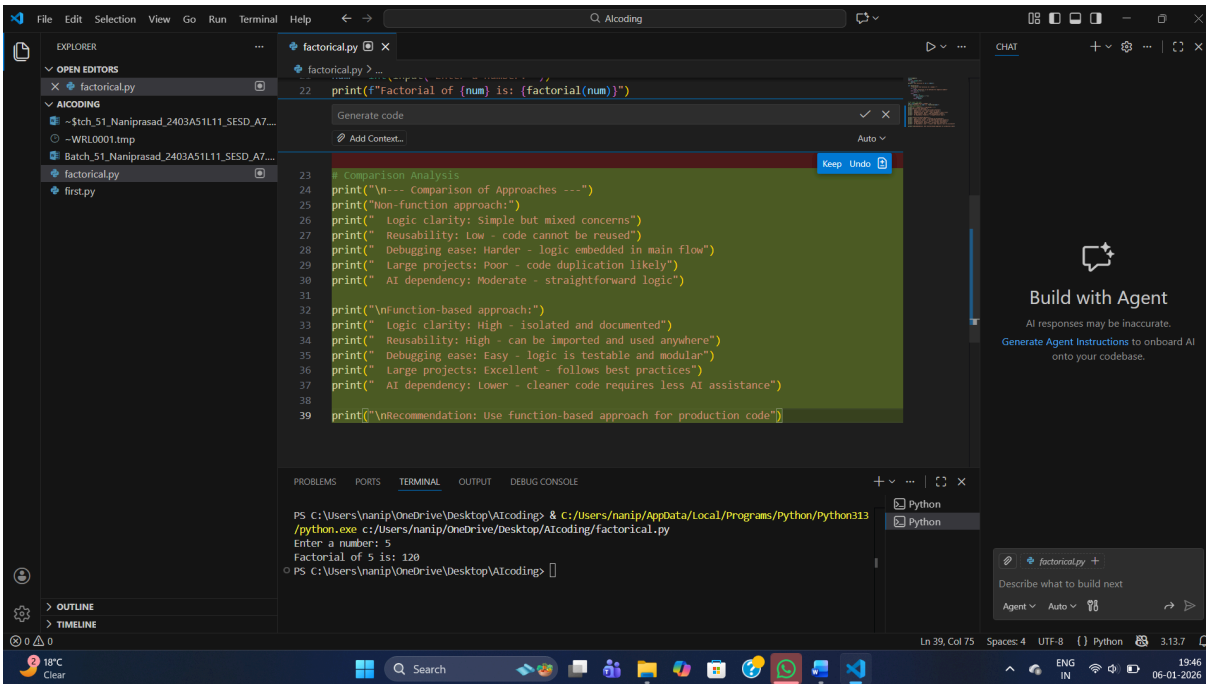
  Supporting scalability, as the same function can be extended or integrated into larger projects.

#Task4
Task Description
Compare the non-function and function-based Copilot-generated programs on the following criteria:

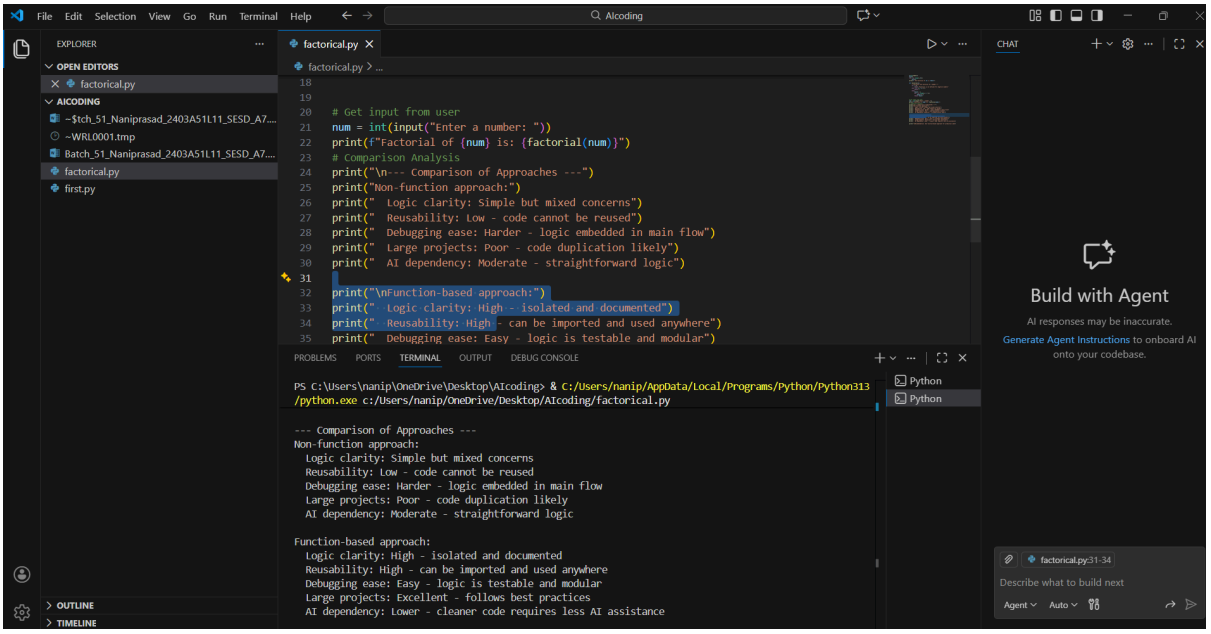
- **Logic clarity**
- **Reusability**
- **Debugging ease**
- **Suitability for large projects**
- **AI dependency risk**

SR UNIVERSITY



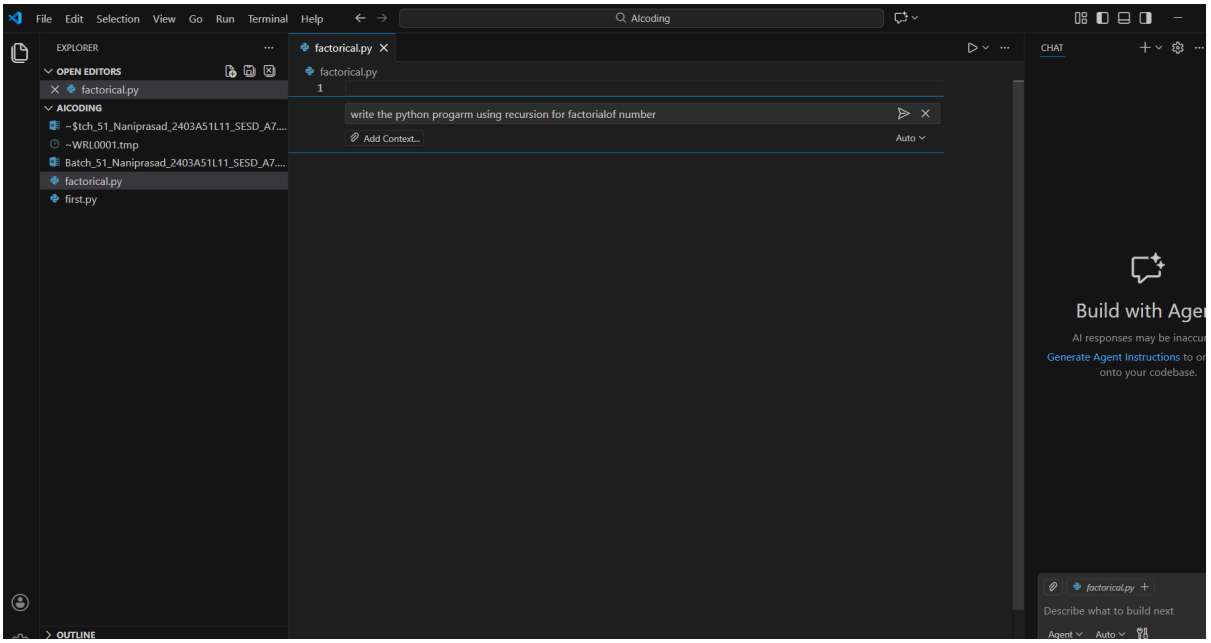


SR UNIVERSITY

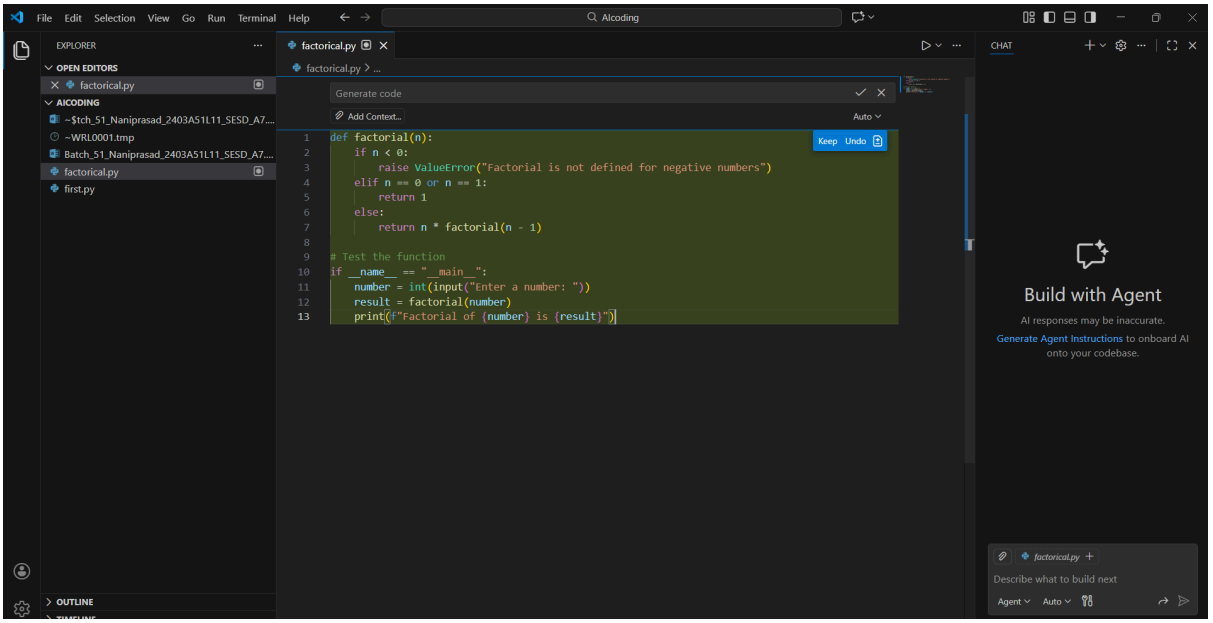


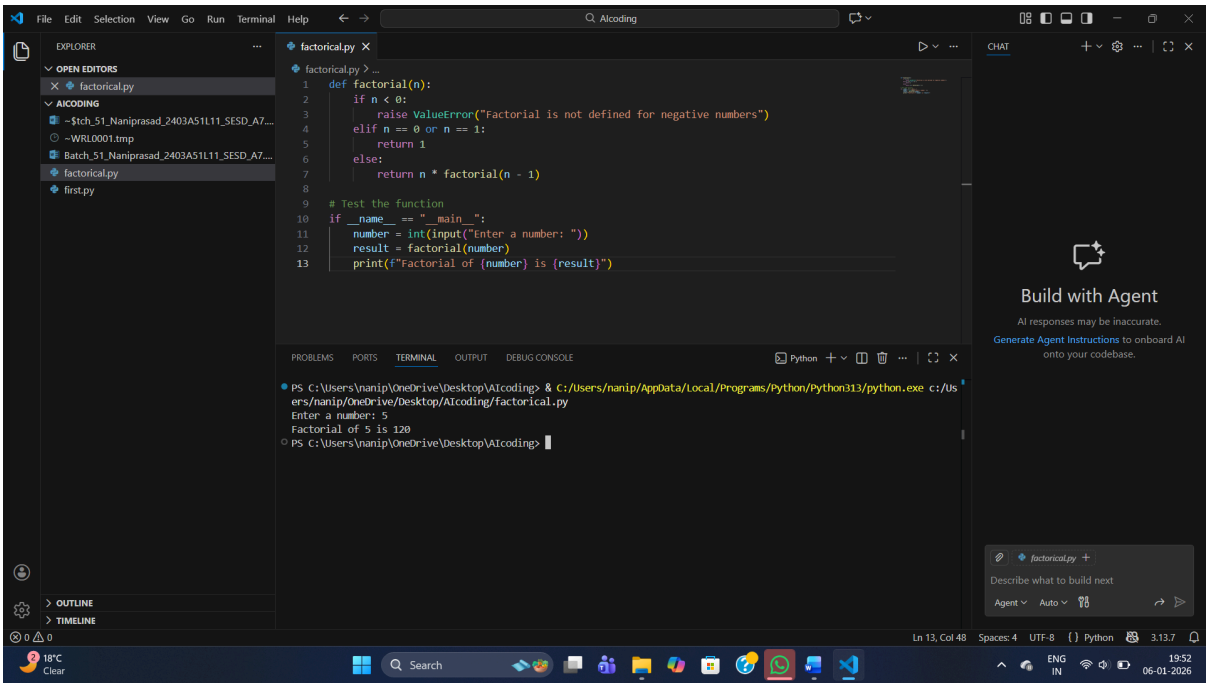
TASK - 5

Task Description
Prompt Copilot to generate:
An iterative version of the logic
A recursive version of the same logic



SR UNIVERSITY





Explanation:

How the Function Works

1. **Negative number check**
   Factorials are not defined for negative numbers. If the input is negative, the program raises an error message.
2. **Base cases**
   For 0 and 1, the factorial is defined as 1. This acts as the stopping condition for recursion.
3. **Recursive case**
   For numbers greater than 1, the function calls itself with n-1. This recursive process continues until it reaches the base case.
   Example:

- **To compute 5!, the function calculates 5\times 4!.**

- **Then 4! becomes 4\times 3!, and so on, until it reaches 1!.**

- **Main Program Flow**

- **The program asks the user to enter a number.**
- **It then calls the factorial function with that number.**
- **Finally, it prints the result in a clear message.**

- **Example Execution**

If the user enters 5:

- **The recursive calls break it down step by step until reaching 1.**
- **The final result is 120.**

So the program outputs: *Factorial of 5 is 120*.

Summary

This program demonstrates:

- **Recursion (function calling itself).**
- **Error handling (for negative inputs).**
- **Base cases (to stop recursion).**
- **User interaction (taking input and displaying output).**

SR UNIVERSITY