ASSIGNMENT-1.3

Name: J.Vyshnavi

HT. No: 2303A51895

Batch: 08

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

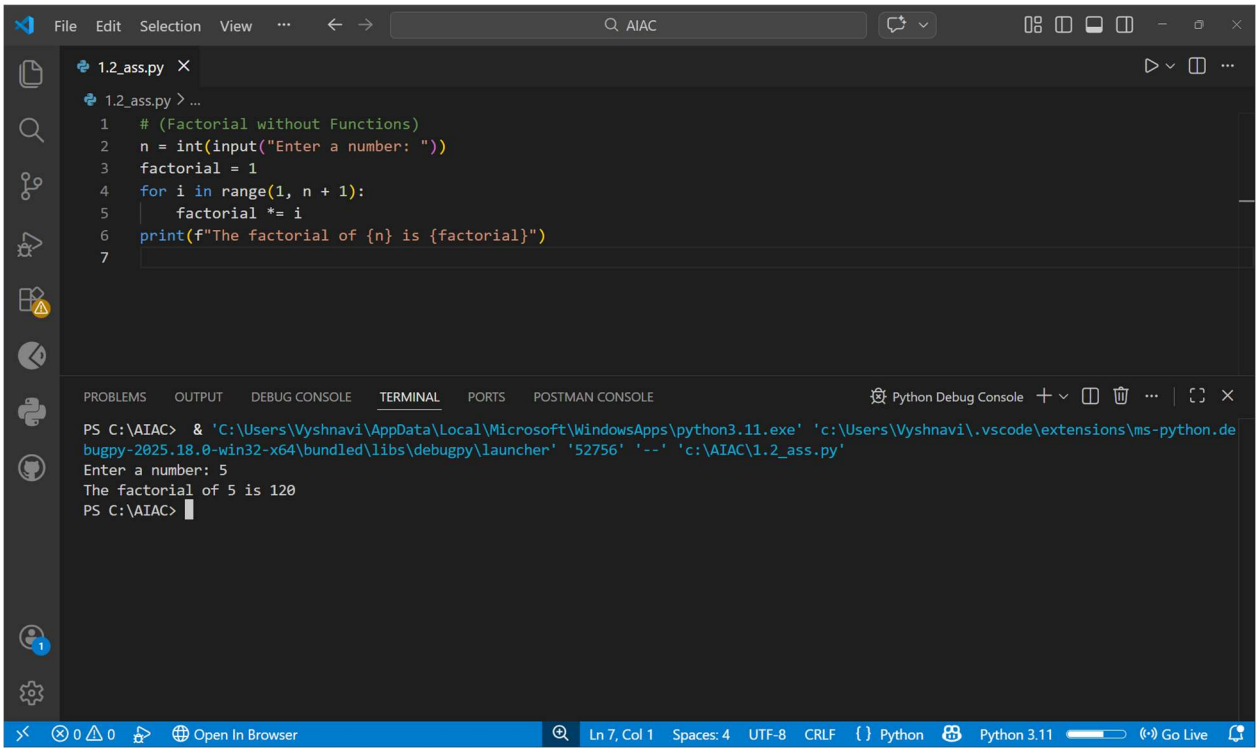
• Scenario

You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

• 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.

Output:



Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

❖ Scenario

Your team lead asks you to review AI-generated code before committing it to a shared repository.

❖ Task Description

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

➢ Reduce unnecessary variables

➢ Improve loop clarity

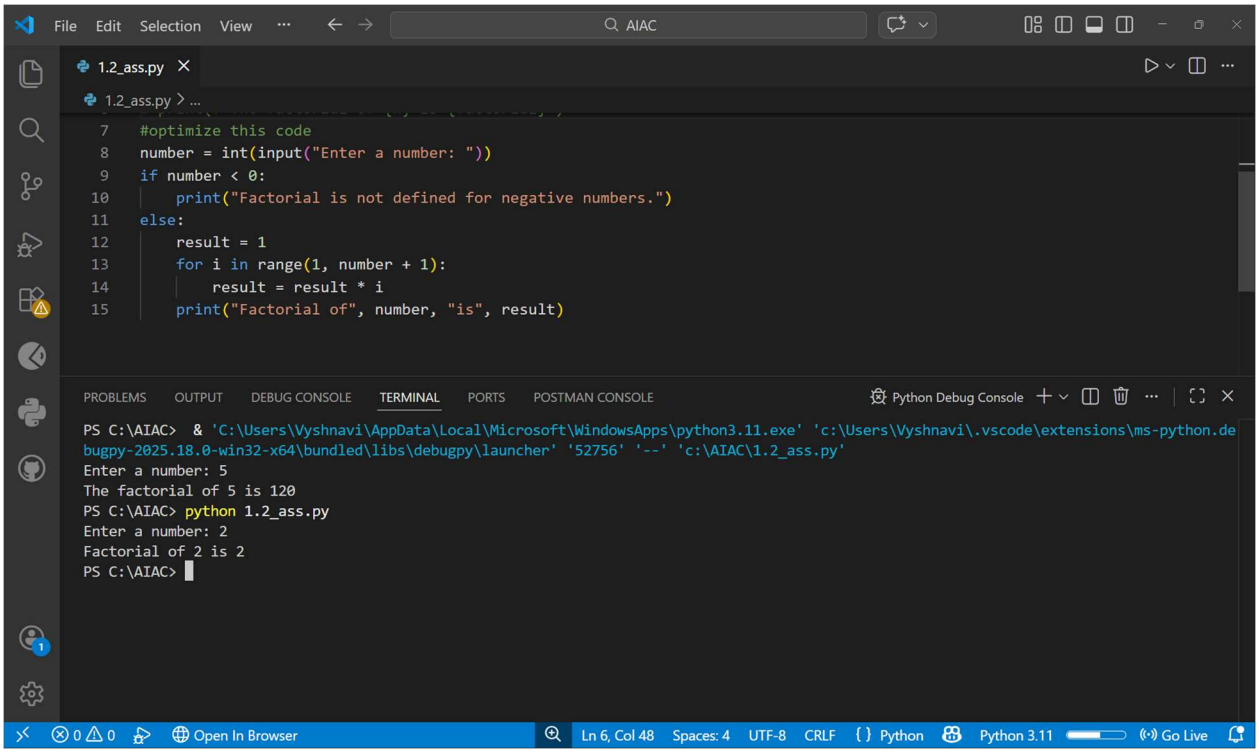
➢ Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

Output:



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

❖ Scenario

The same logic now needs to be reused in multiple scripts.

❖ Task Description

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

➢ Creating a user-defined function

➢ Calling the function from the main block

❖ Constraints

➢ Use meaningful function and variable names

➢ Include inline comments (preferably suggested by Copilot)

❖ Expected Deliverables

➢ AI-assisted function-based program

➢ Screenshots showing:

o Prompt evolution

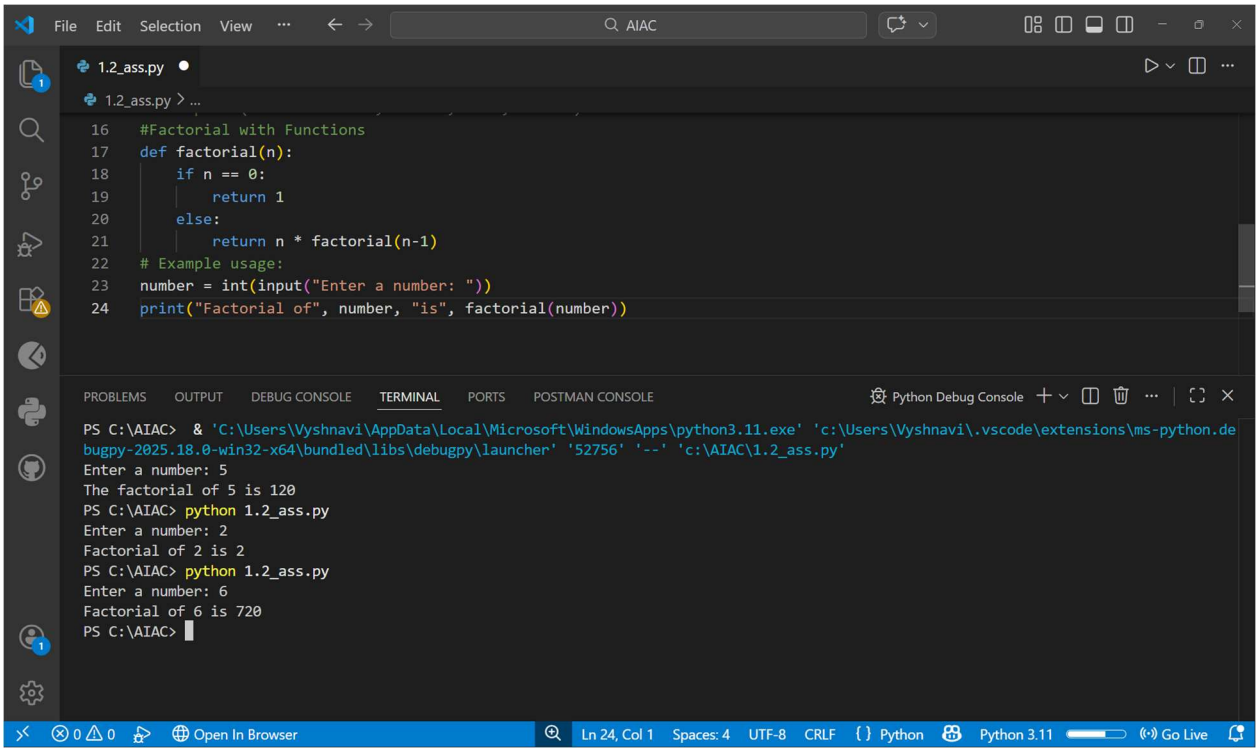
o Copilot-generated function logic

➢ Sample inputs/outputs

➢ Short note:

o How modularity improves reusability.

Output:



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

❖ Scenario

As part of a code review meeting, you are asked to justify design choices.

❖ 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

❖ Expected Deliverables

Choose one:

➢ A comparison table

OR

➢ A short technical report (300–400 words).

1. Logic Clarity:
The procedural version is straightforward and easy to follow for small programs because all logic is written in one place. However, as the program grows, mixing input handling and computation logic in the same block reduces clarity. The modular version improves logic clarity by separating the factorial computation into a dedicated function. This clear separation makes the code easier to read and understand.

2. Reusability:
The non-function version lacks reusability because the logic is tightly coupled with the main script. If the same factorial logic is needed elsewhere, the entire code must be copied. In contrast, the modular version allows the function to be reused in multiple scripts or projects simply by importing it, making it significantly more flexible.

3. Debugging Ease:
Debugging procedural code becomes difficult when the file size increases, as there is no logical separation of concerns. In the modular version, bugs can be isolated within the function, allowing easier testing and troubleshooting. Functions also make unit testing possible, which improves reliability.

4. Suitability for Large Projects:
Procedural style is acceptable for small scripts or quick prototypes but is not suitable for large-scale applications. Modular design follows better software engineering principles, making it more scalable, maintainable, and adaptable for larger systems.

5. AI Dependency Risk:
In the procedural approach, beginners may rely entirely on AI-generated code without understanding the structure. Modular design encourages structured thinking and better coding habits. While AI tools accelerate development, developers must review and validate the generated logic to avoid blind dependency.

Task 5: AI-Generated Iterative vs Recursive Thinking

❖ Scenario

Your mentor wants to test how well AI understands different computational paradigms.

❖ Task Description

Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic

❖ Constraints

Both implementations must produce identical outputs

Students must not manually write the code first

Output: