COURSE: AI Assisted Coding

NAME:K.Srija

BATCH-34

HALLTICKET.NO:2303A52189

TASK-1

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.

• Constraint:

➢ Do not define any custom function

➢ Logic must be implemented using loops and variables only

• Expected Deliverables

➢ A working Python program generated with Copilot assistance

➢ Screenshot(s) showing:

➢ The prompt you typed

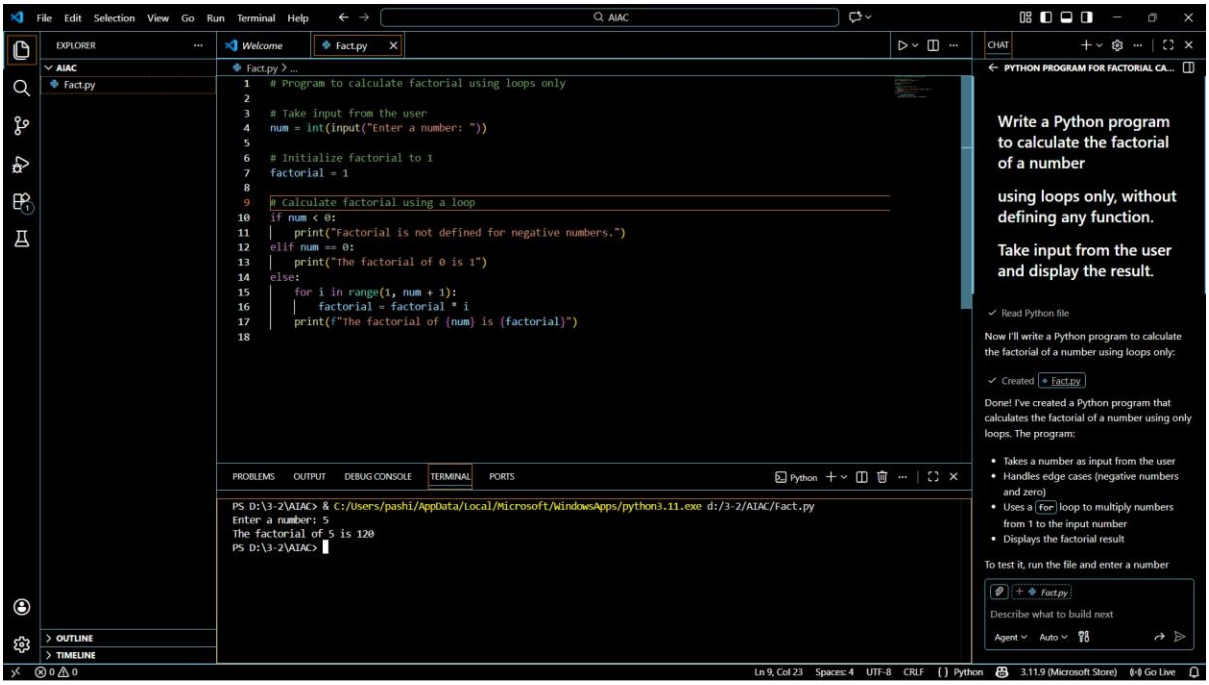
➢ Copilot's suggestions

➢ Sample input/output screenshots

➢ Brief reflection (5–6 lines):

➢ How helpful was Copilot for a beginner?

➢ Did it follow best practices automatically?



Brief Reflection

GitHub Copilot was very helpful for a beginner because it quickly generated correct and readable code.
It understood the comment prompt and followed the constraint of not using functions.
The generated logic was simple and easy to understand.
Copilot automatically used a loop and proper variable naming.
However, beginners should still verify the logic instead of blindly trusting the output.
Overall, Copilot improves productivity and learning speed.

Conclusion

This lab helped in understanding how AI tools like GitHub Copilot assist in coding tasks.
It demonstrated prompt-based programming and highlighted the importance of reviewing AI-generated code.

TASK-2

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”

❖ Expected Deliverables

➢ Original AI-generated code

➢ Optimized version of the same code

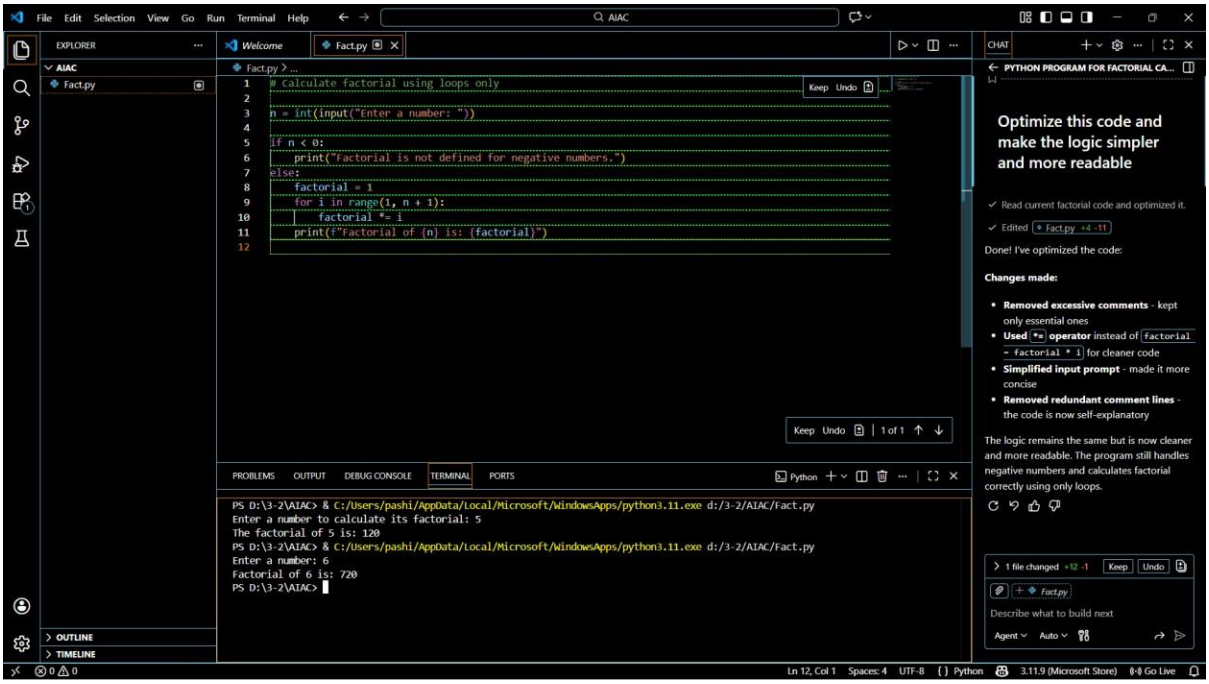
➢ Side-by-side comparison

➢ Written explanation:

▪ What was improved?

▪ Why the new version is better (readability, performance,

maintainability.



Side-by-Side Comparison

Original Code	**Optimized Code**
Uses variable name result	Uses clearer name factorial
Loop starts from 1	Loop starts from 2 (avoids unnecessary multiplication)
Traditional multiplication syntax	Uses *= shorthand
Normal print statement	Uses formatted string (f-string)

Written Explanation

What was improved?

- Variable naming was improved for better clarity
- Loop logic was simplified by starting from 2 instead of 1
- Shorthand operator (*=) reduced verbosity
- Output formatting became cleaner using f-strings

Why the new version is better?

- **Readability:** Clearer variable names and cleaner print statement
- **Performance:** Avoids unnecessary multiplication by 1
- **Maintainability:** Shorter and easier to understand code
- **Professional quality:** Follows cleaner coding style

Conclusion

Using Copilot for optimization helps refine initial AI-generated code into cleaner and more maintainable versions.
However, human review is still essential before committing code to production.

TASK-3

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:

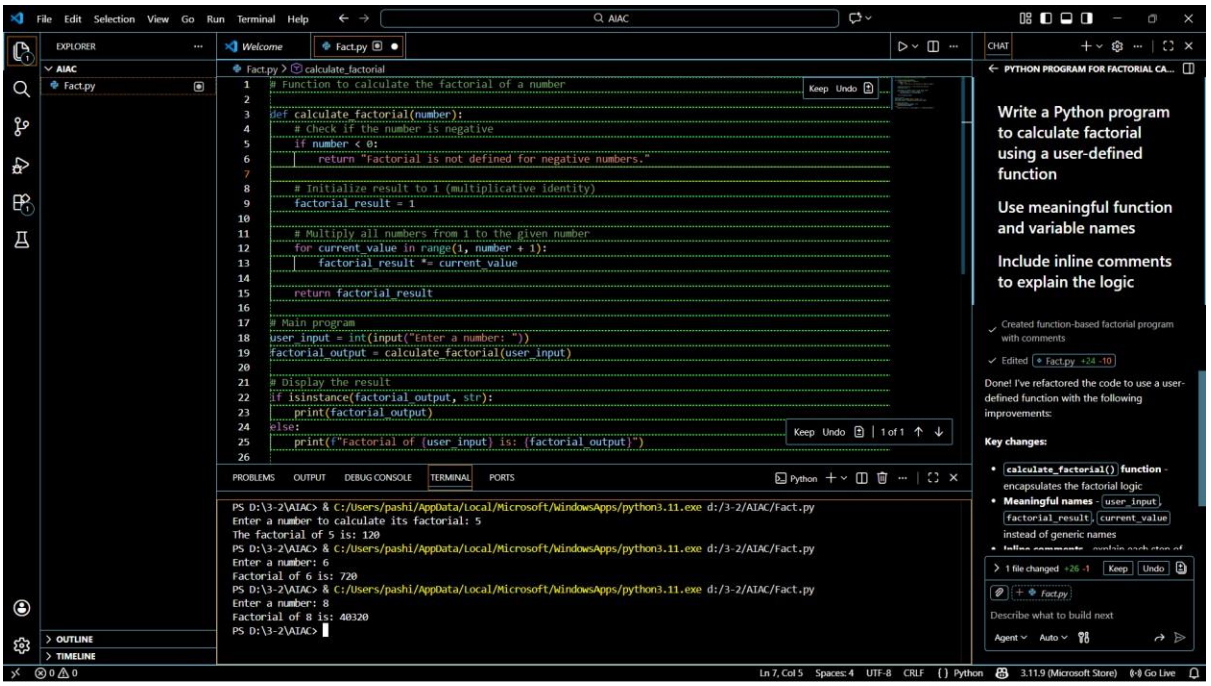
o Prompt evolution

o Copilot-generated function logic

➢ Sample inputs/outputs

➢ Short note:

o How modularity improves reusability.



How Modularity Improves Reusability

Modularity allows the same function to be reused in multiple programs without rewriting the logic.
It improves code readability and makes debugging easier.
Functions help separate logic into meaningful blocks.
If changes are needed, they can be made in one place.
Overall, modular design makes the code more maintainable and scalable

Conclusion

Using GitHub Copilot for modular programming helps generate clean, reusable code quickly.
However, understanding the logic is essential before using AI-generated functions in real projects.

TASK-4

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).

Comparative Analysis – Procedural vs Modular AI Code

Comparison: Without Functions vs With Functions (AI-Generated Code)

Criteria	**Procedural Code (Without Functions)**	**Modular Code (With Functions)**
Logic Clarity	Logic is written in a single block, which is easy to follow for very small programs but becomes cluttered as code grows.	Logic is separated into a well-named function, making the program structure clearer and easier to understand.
Reusability	Code cannot be reused easily and must be rewritten if needed in another program.	The same function can be reused across multiple scripts without duplication.
Debugging Ease	Debugging is harder because all logic is mixed together in the main flow.	Debugging is easier since errors can be isolated within the function.
Suitability for Large Projects	Not suitable for large projects due to poor structure and scalability issues.	Highly suitable for large projects as modular design supports scalability and teamwork.
AI Dependency Risk	Beginners may depend heavily on AI-generated logic without understanding the flow.	Encourages understanding of function design and reduces blind AI dependency.

Conclusion

Procedural code generated by AI is useful for quick and simple tasks, especially for beginners.
However, modular code provides better readability, reusability, and maintainability.
For real-world and large-scale projects, function-based modular design is always preferred.

TASK-5

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

❖ Expected Deliverables

Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

➢ Readability

➢ Stack usage

➢ Performance implications

➢ When recursion is not recommended

Execution Flow Explanation:

In the iterative approach, the program uses a loop to repeatedly multiply numbers until the final result is obtained.
In the recursive approach, the function repeatedly calls itself by reducing the problem size until it reaches the base condition.
Iteration completes the task in a single function call, while recursion breaks the problem into smaller subproblems.
Both approaches follow the same mathematical logic but differ in execution style.

Comparison: Iterative vs Recursive AI-Generated Code

Criteria	**Iterative Approach**	**Recursive Approach**
Readability	Easier to understand for beginners	Cleaner and closer to mathematical definition
Stack Usage	Uses constant memory	Uses call stack for each function call
Performance Implications	Faster and memory-efficient	Slower due to function call overhead
When Recursion Is Not Recommended	—	For large inputs due to stack overflow risk

Conclusion

AI can successfully generate both iterative and recursive solutions when prompted correctly.
Iterative solutions are safer for large inputs, while recursive solutions are more elegant but resource-intensive.
Understanding both approaches helps developers choose the right paradigm for real-world applications.