SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

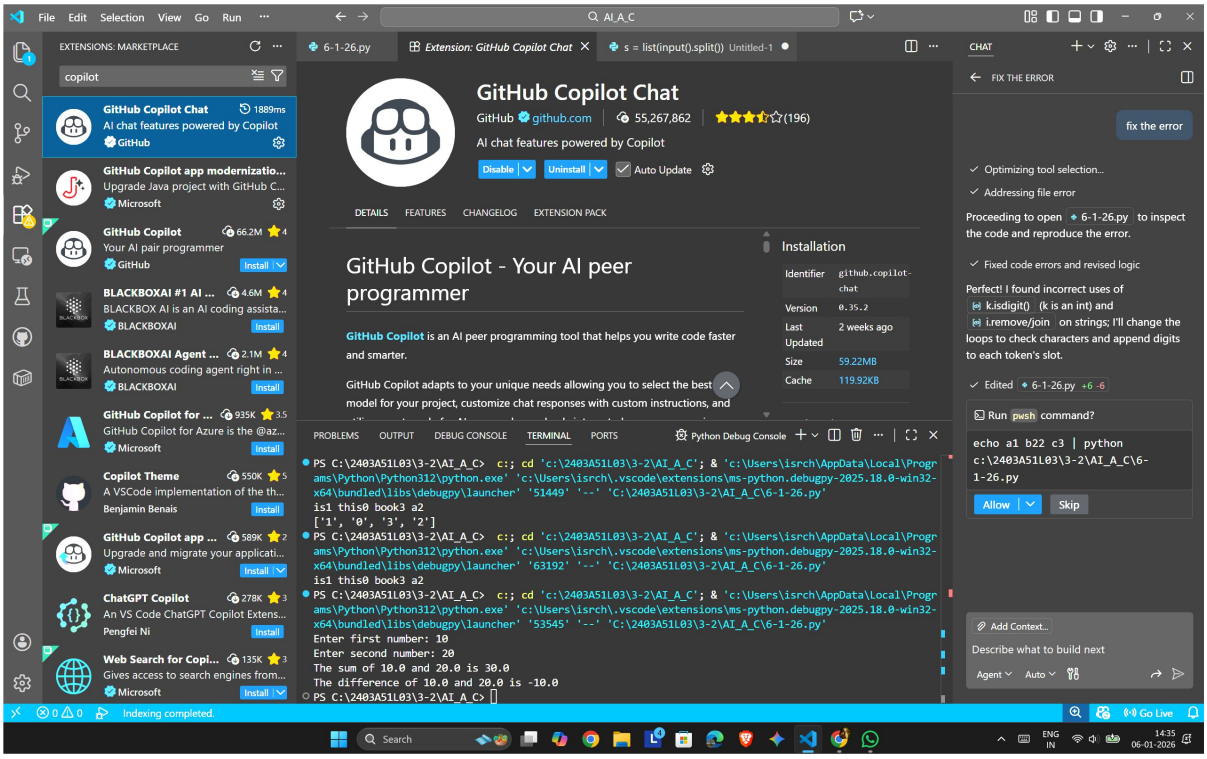
Program	**: B. Tech (CSE)**
Specialization	**:**
Course Title	**: AI Assisted Coding**
Course Code	**: 23CS002PC304**
Semester	**: II**
Academic Session	**: 2025-2026**
Name of Student	**: Ganesh**
Enrollment No.	**: 2403A51L55**
Batch No.	**: 52**
Date	**: 06/01/26**

Submission Starts here

Screenshots:

TASK - 0

- Install “Github Copilot” in Visual Studio Code

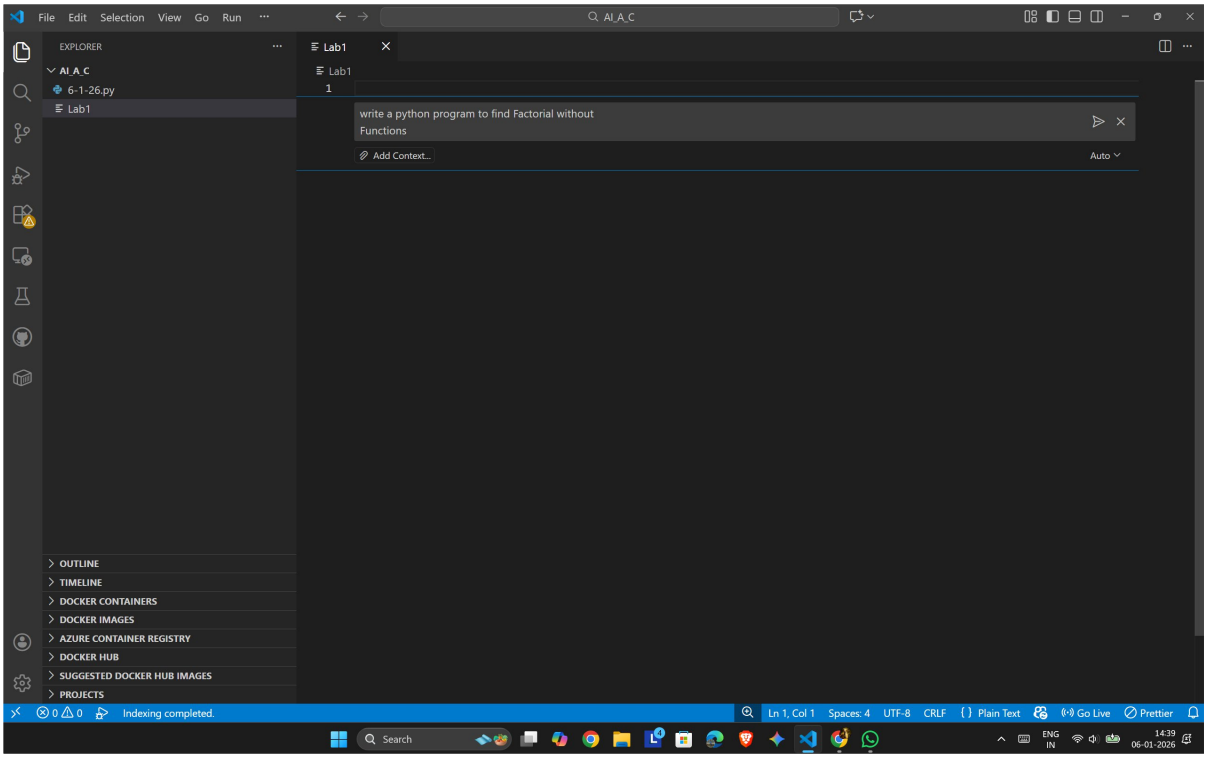


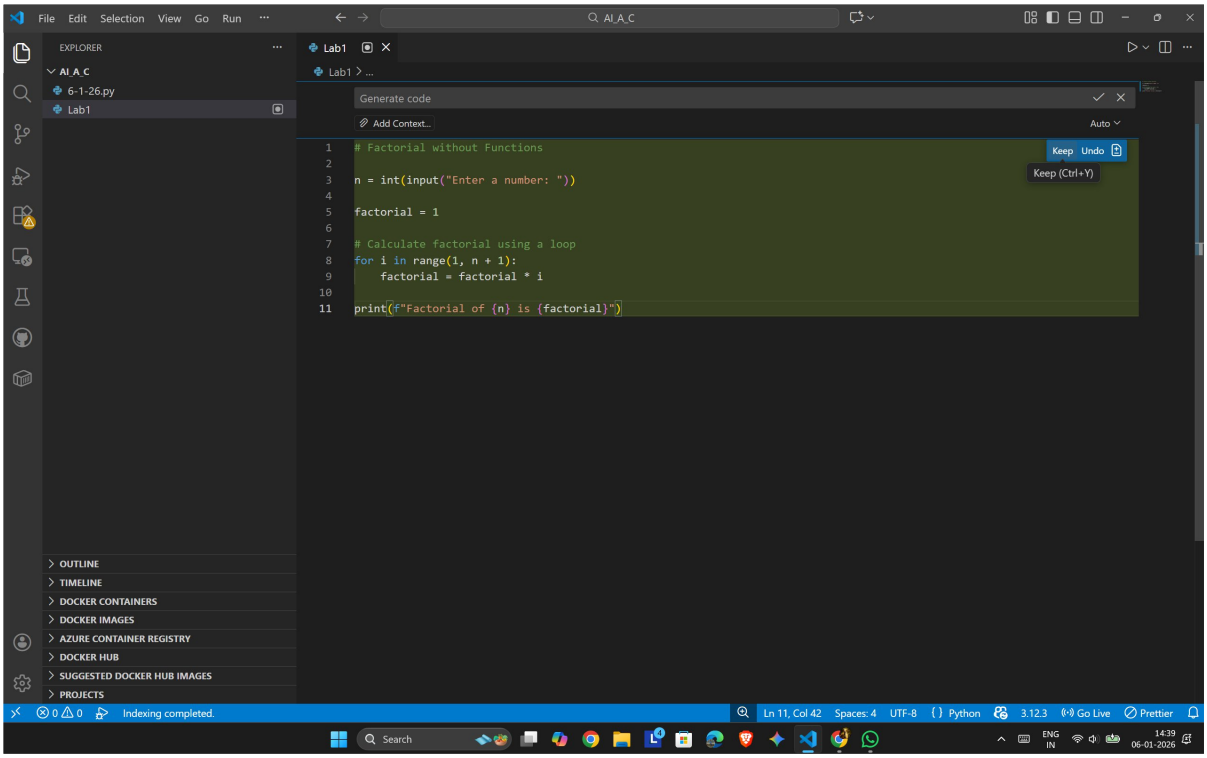
SR UNIVERSITY

TASK - 1

- Task Description

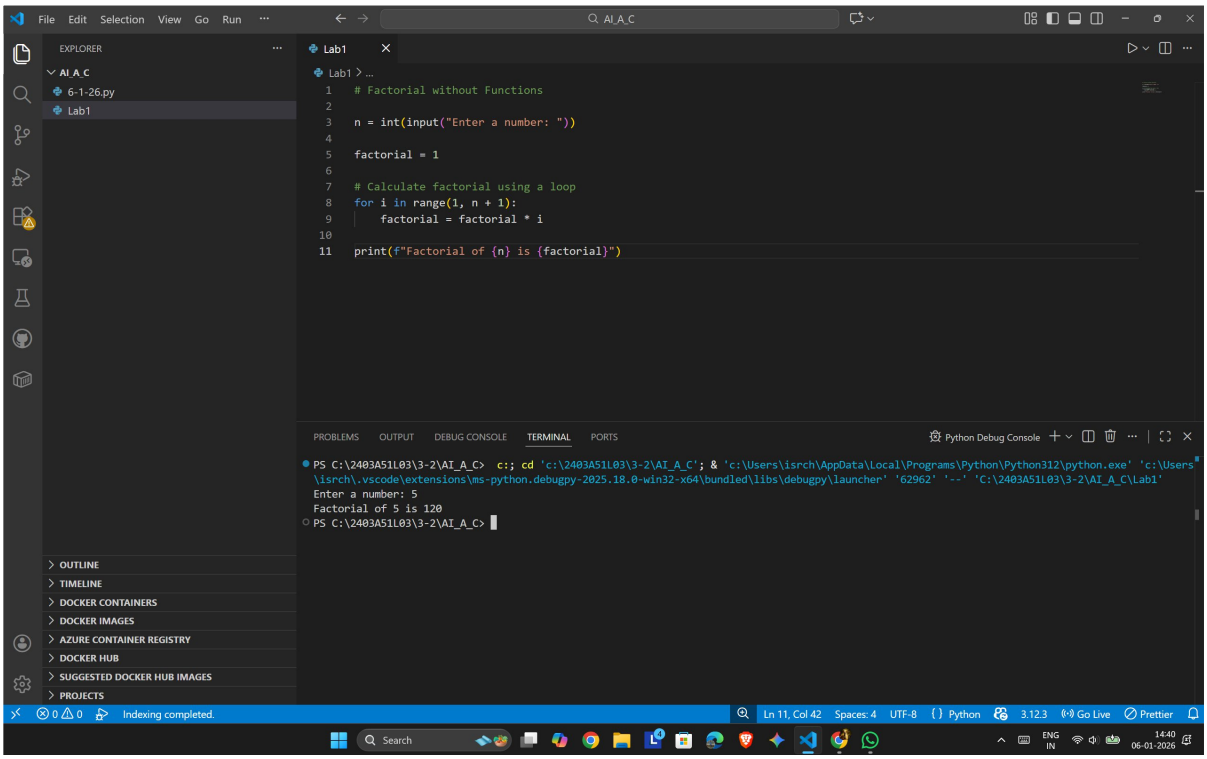
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.





SR UNIVERSITY

OUTPUT:



- 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

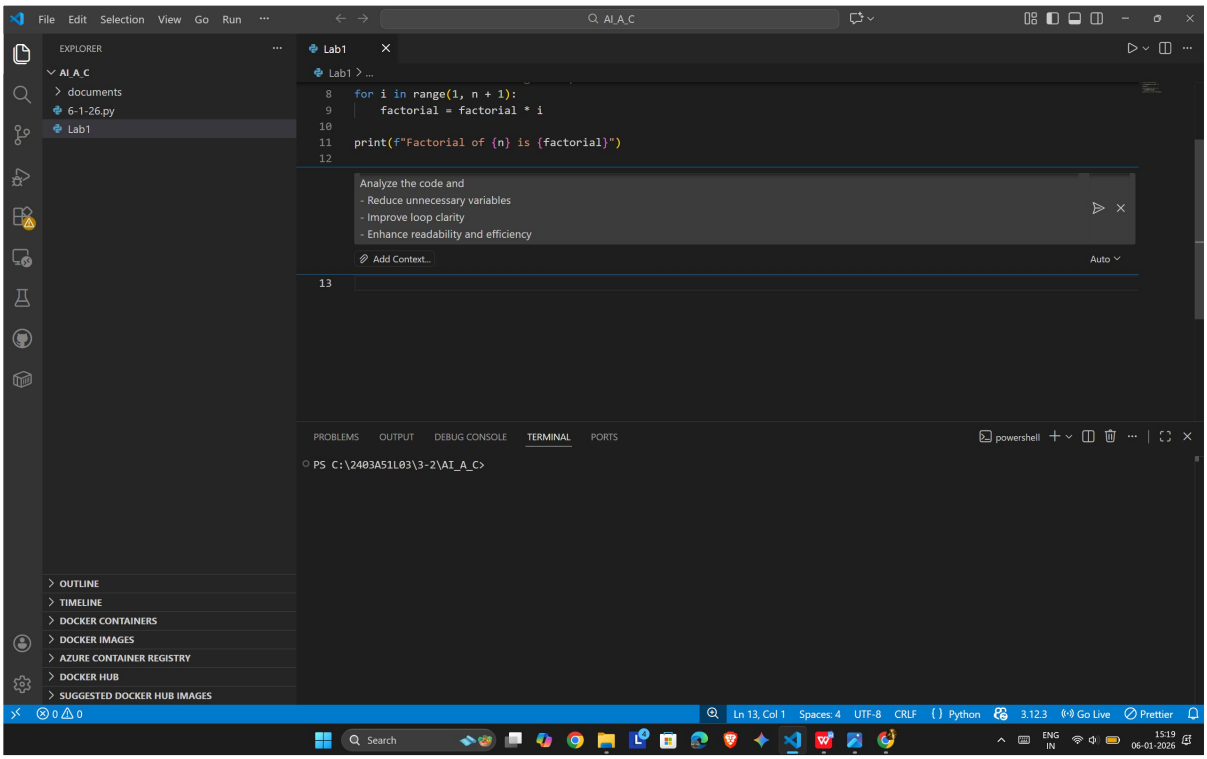
SR UNIVERSITY

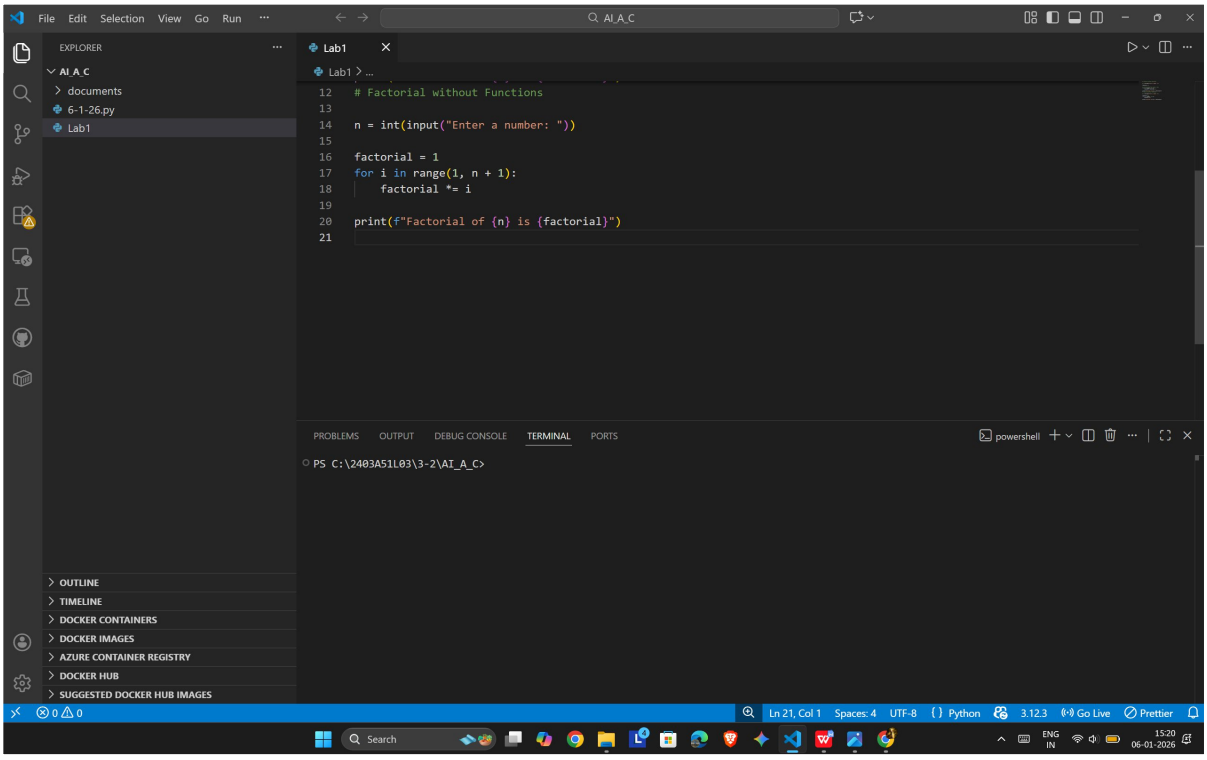
TASK - 2

Task Description

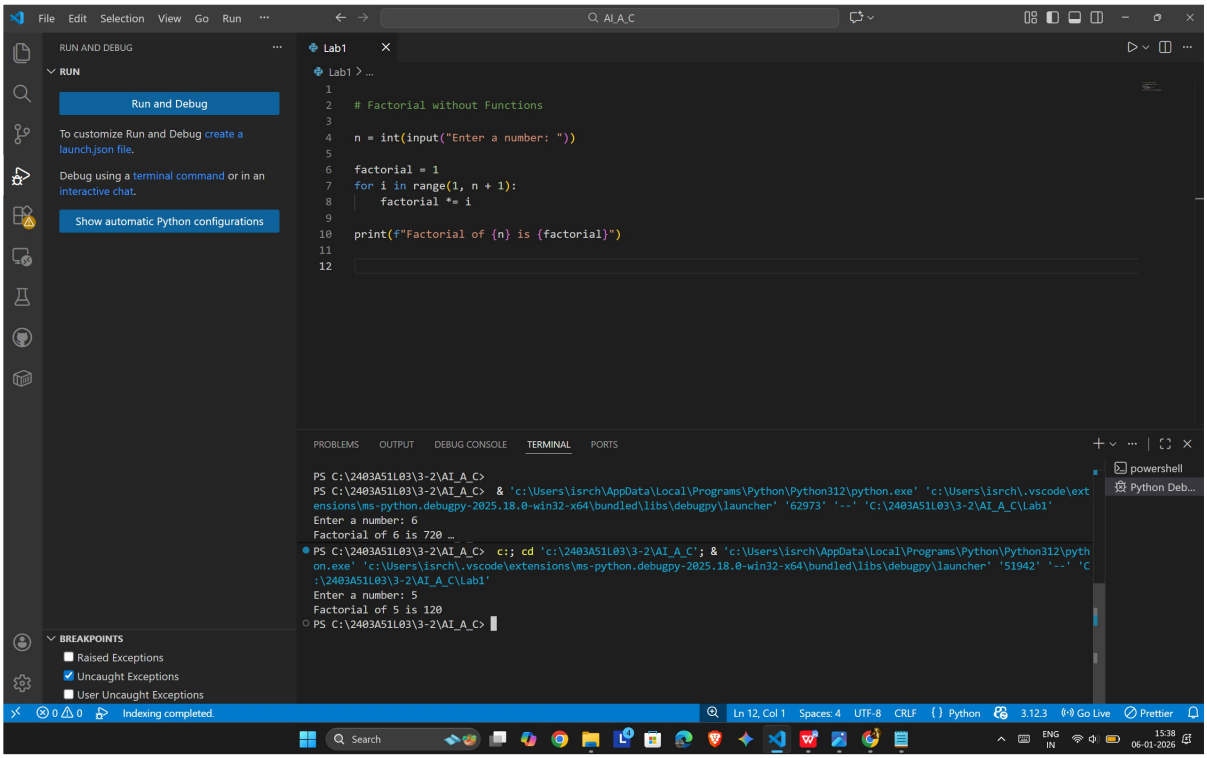
Analyze the code generated in Task 1 and use Copilot again to:

- **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?

1. Readability
***= is clearer and more concise.**

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

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

3. Performance
- Performance is effectively the same.

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

SR UNIVERSITY

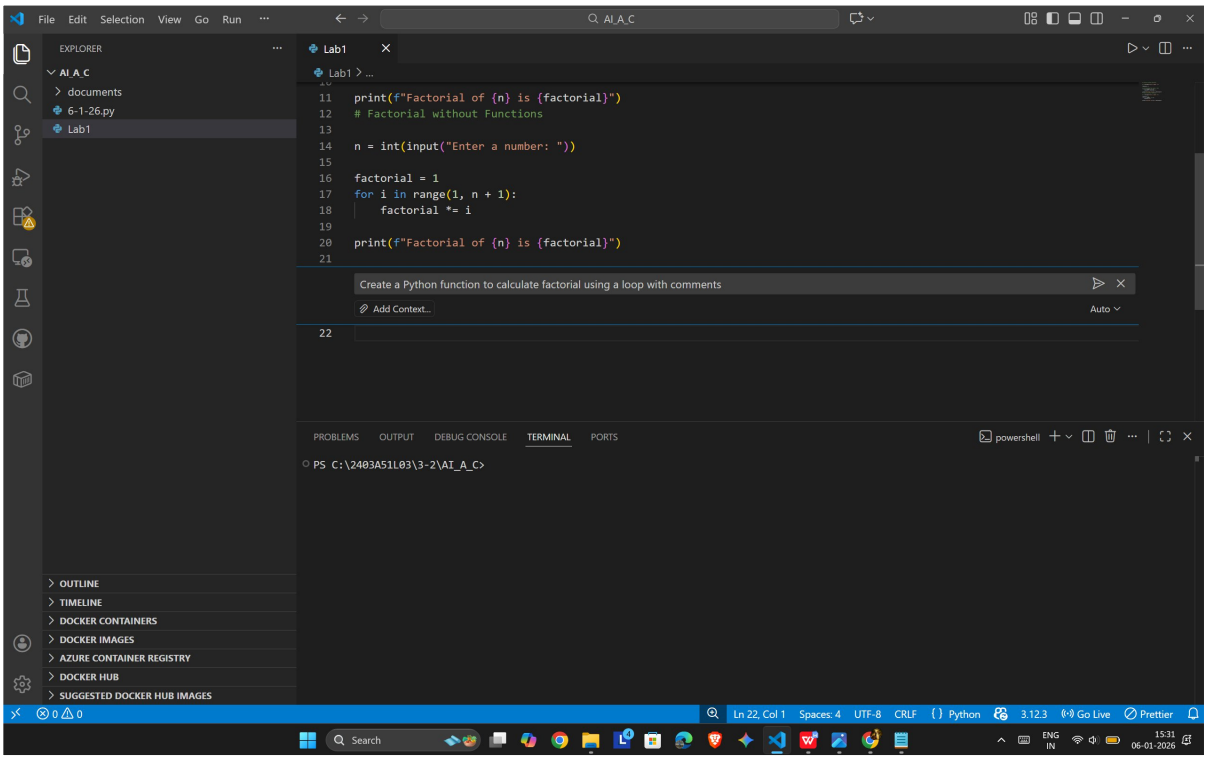
TASK - 3

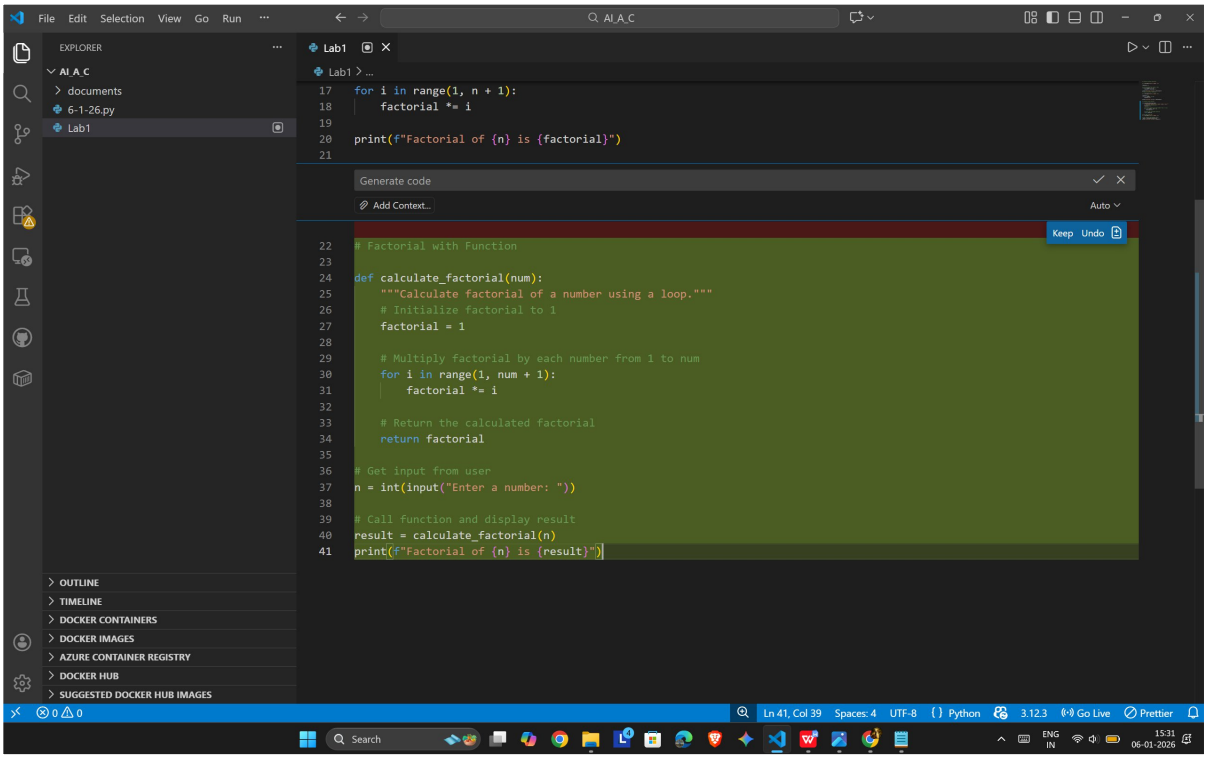
Task Description

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

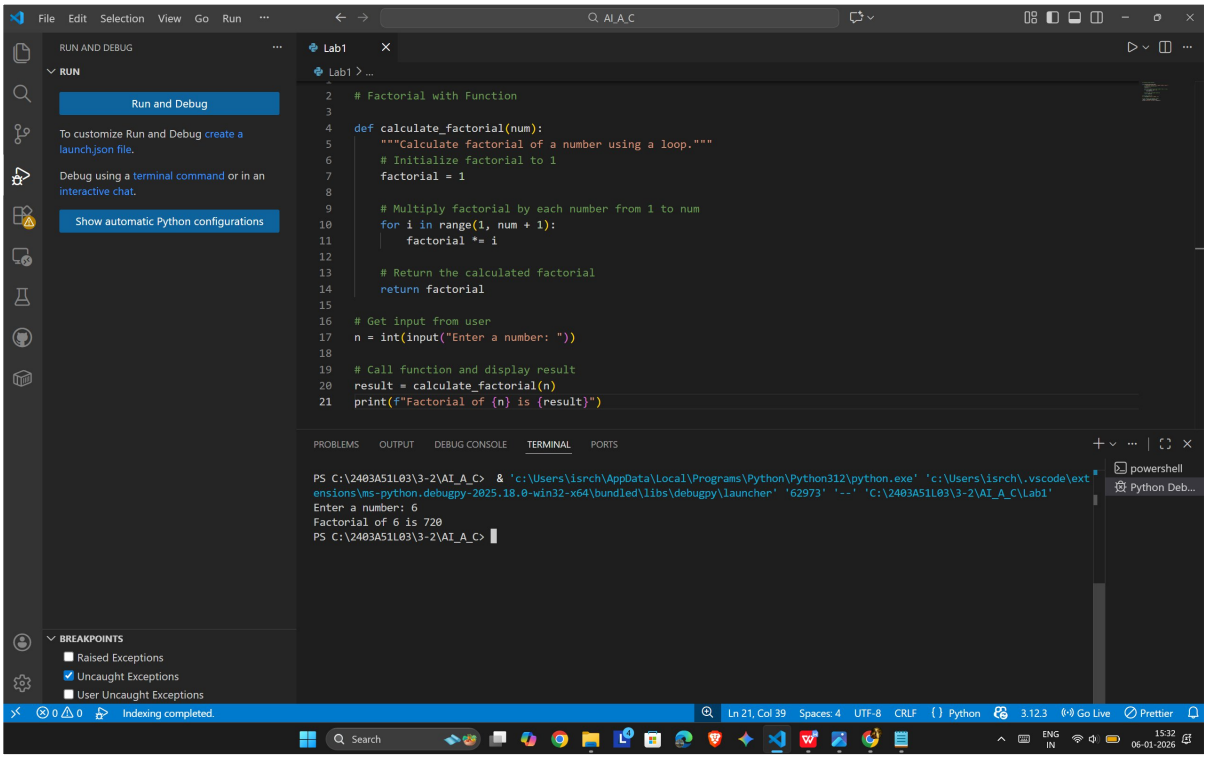
➢ Creating a user-defined function

➢ Calling the function from the main block





SR UNIVERSITY



- Modularity improves reusability by:

Allowing the calculate_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.

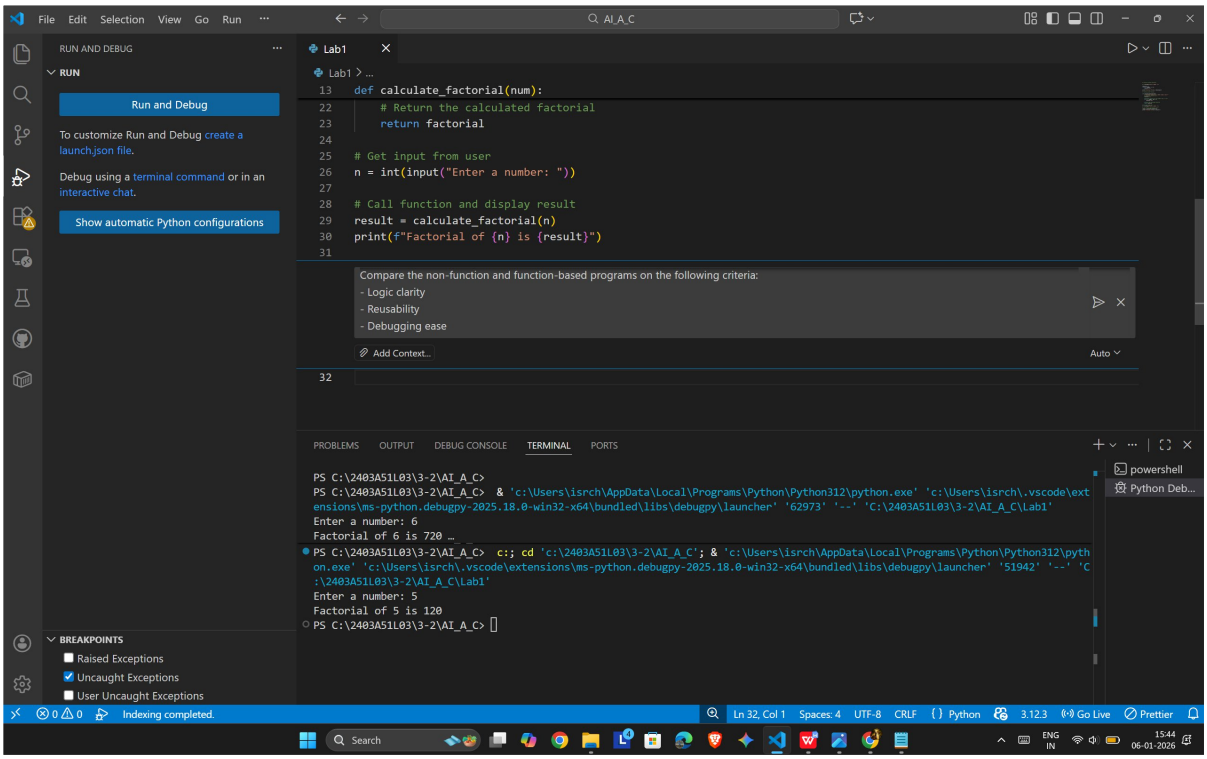
SR UNIVERSITY

TASK - 4

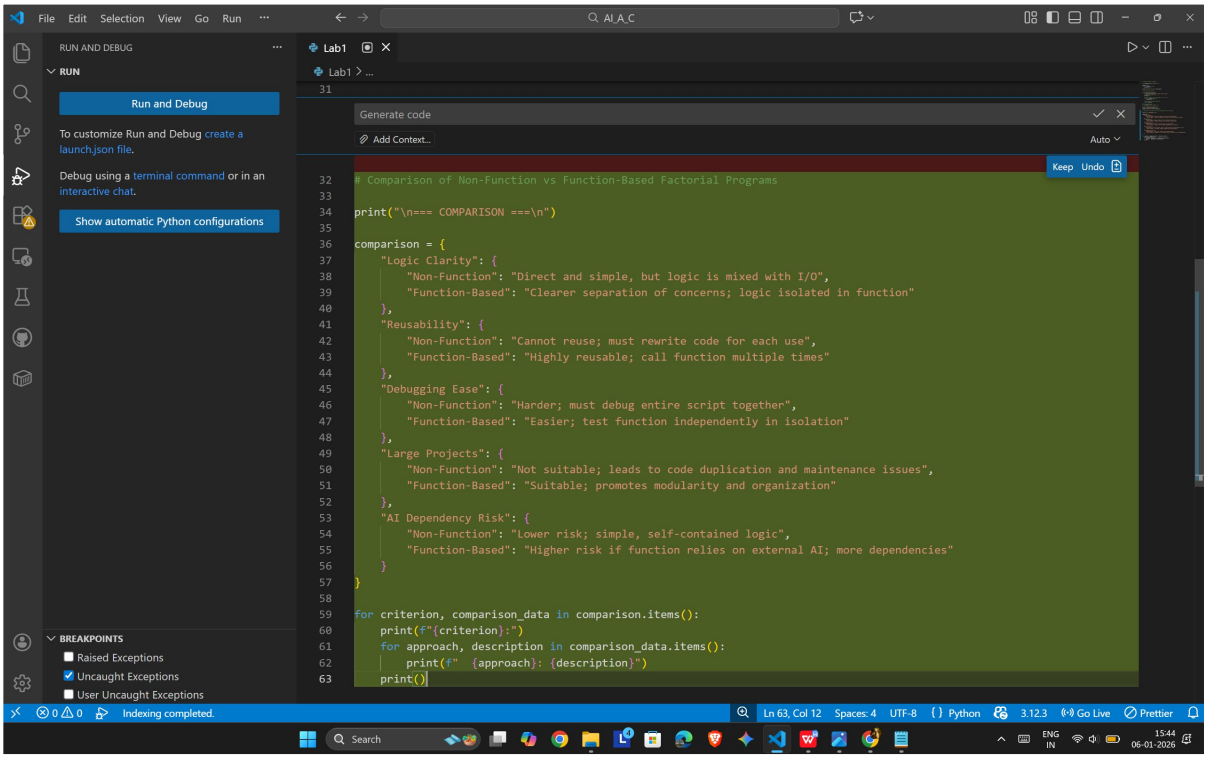
Task Description

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**



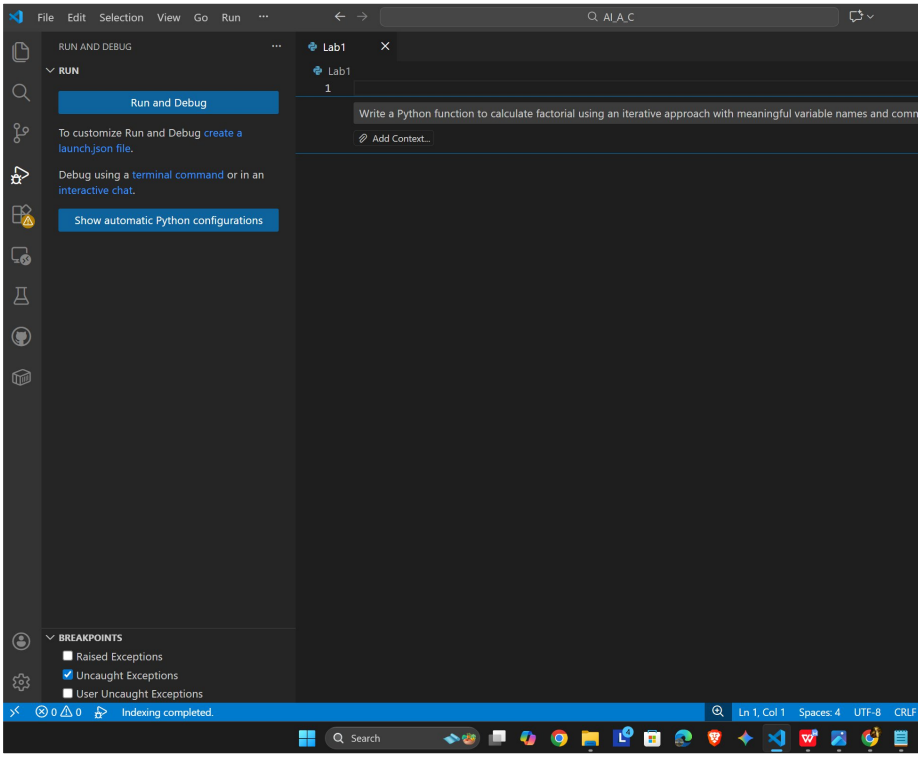
SR UNIVERSITY

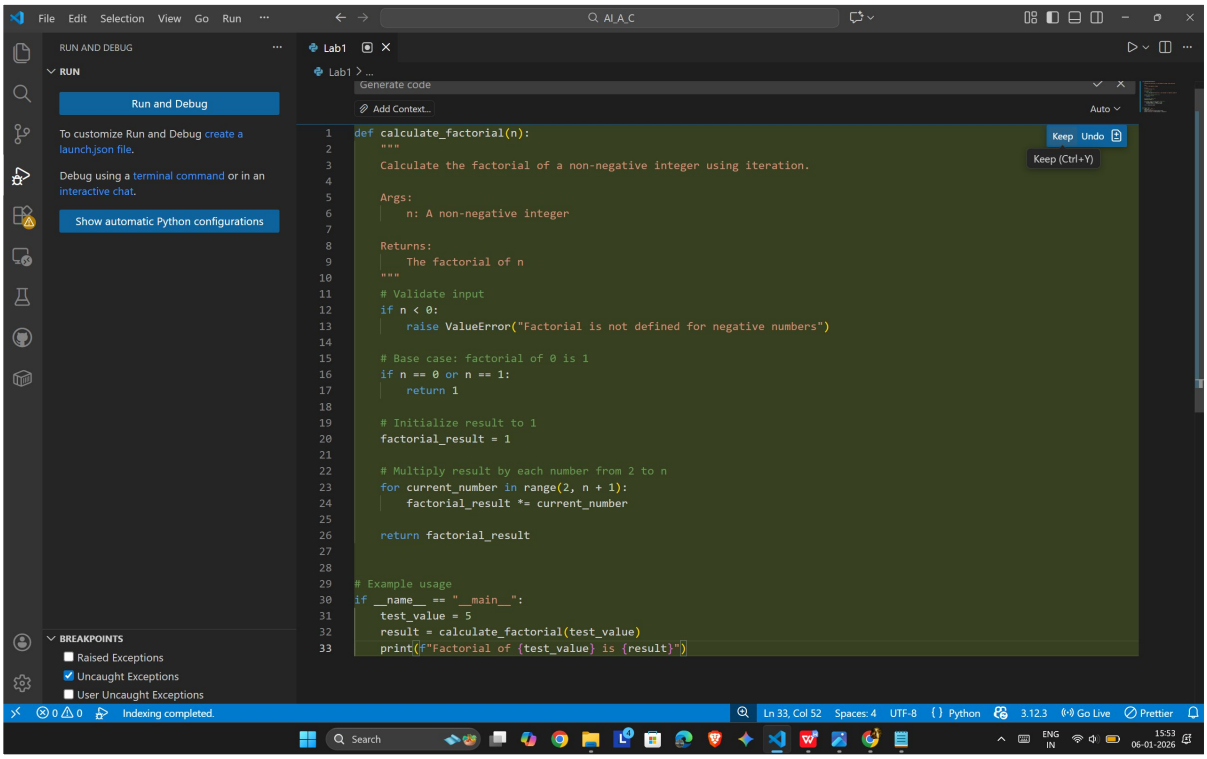


TASK - 5

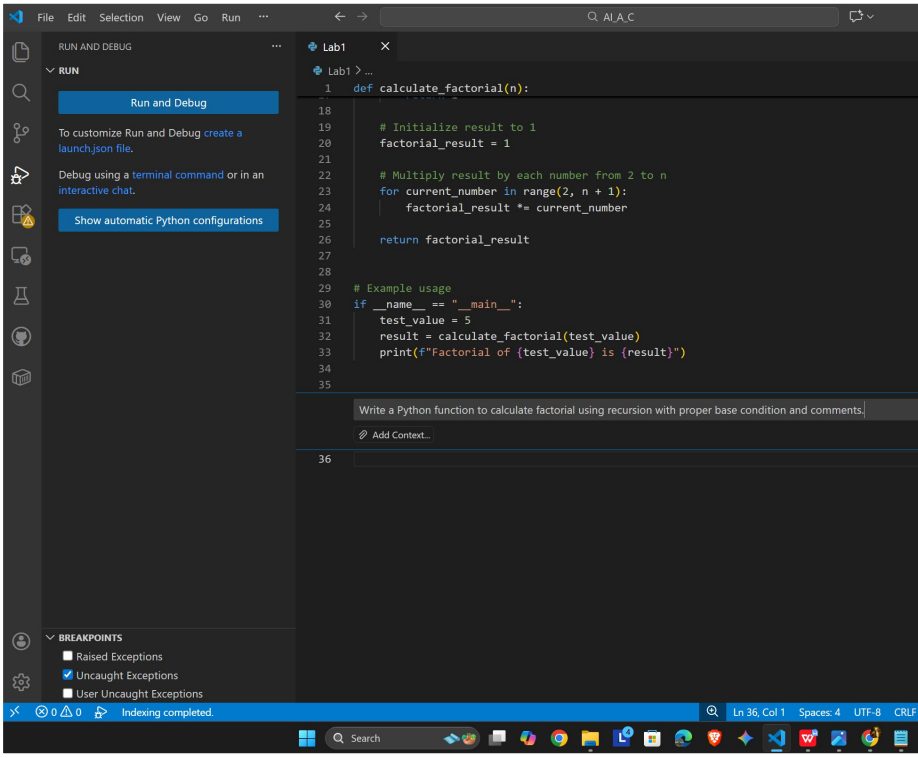
Task Description
Prompt Copilot to generate:
An iterative version of the logic
A recursive version of the same logic

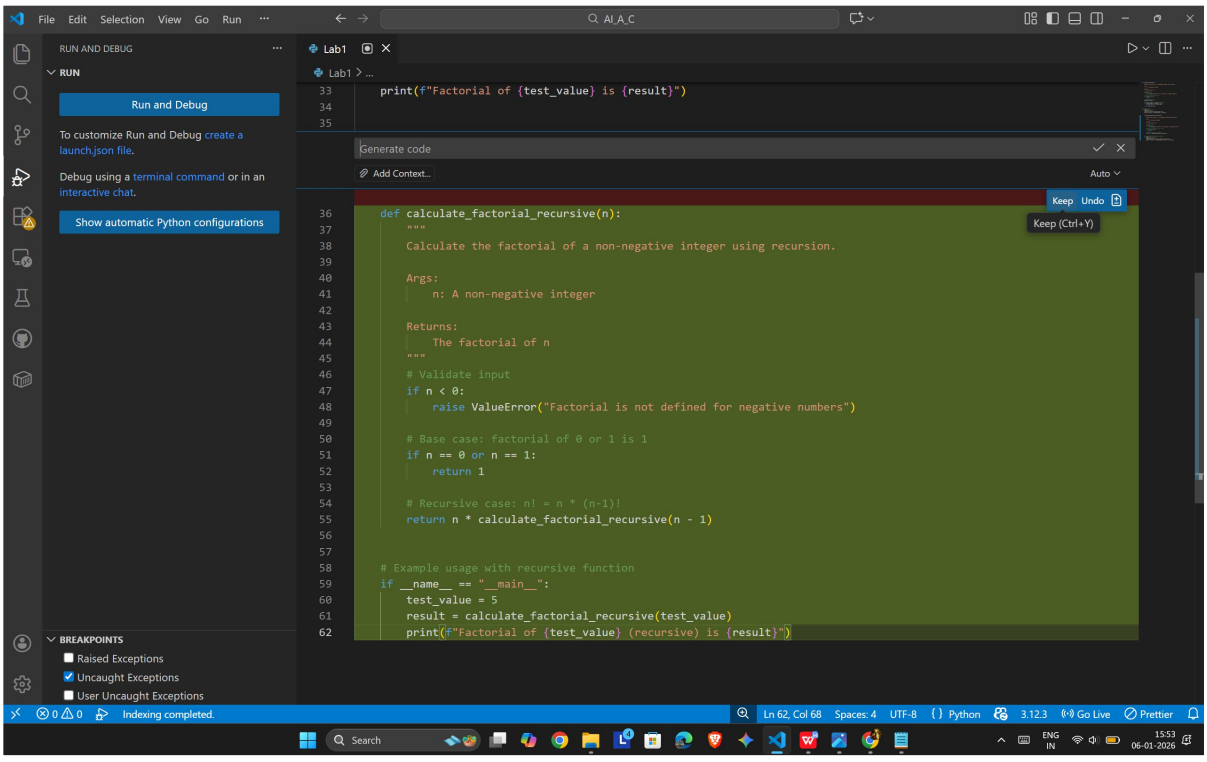
SR UNIVERSITY





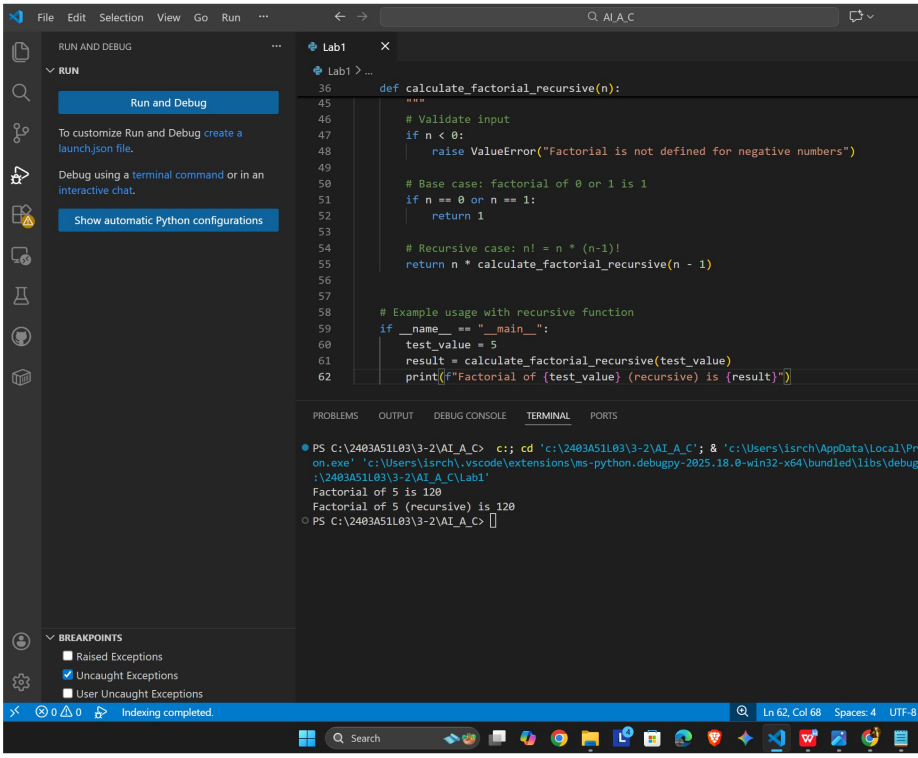
SR UNIVERSITY





OUTPUT:

SR UNIVERSITY



Explaination

- Iterative Approach
- Starts with a result value of 1.
- Repeats multiplication from 1 up to the given number using a loop.
- Stores the intermediate result in the same variable.
- Executes sequentially without extra memory overhead.

- Recursive Approach
- Breaks the problem into smaller subproblems.
- Each function call multiplies the current number by the factorial of the previous number.
- Stops when it reaches the base case (0 or 1).
- Uses the call stack to remember previous function calls.