

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

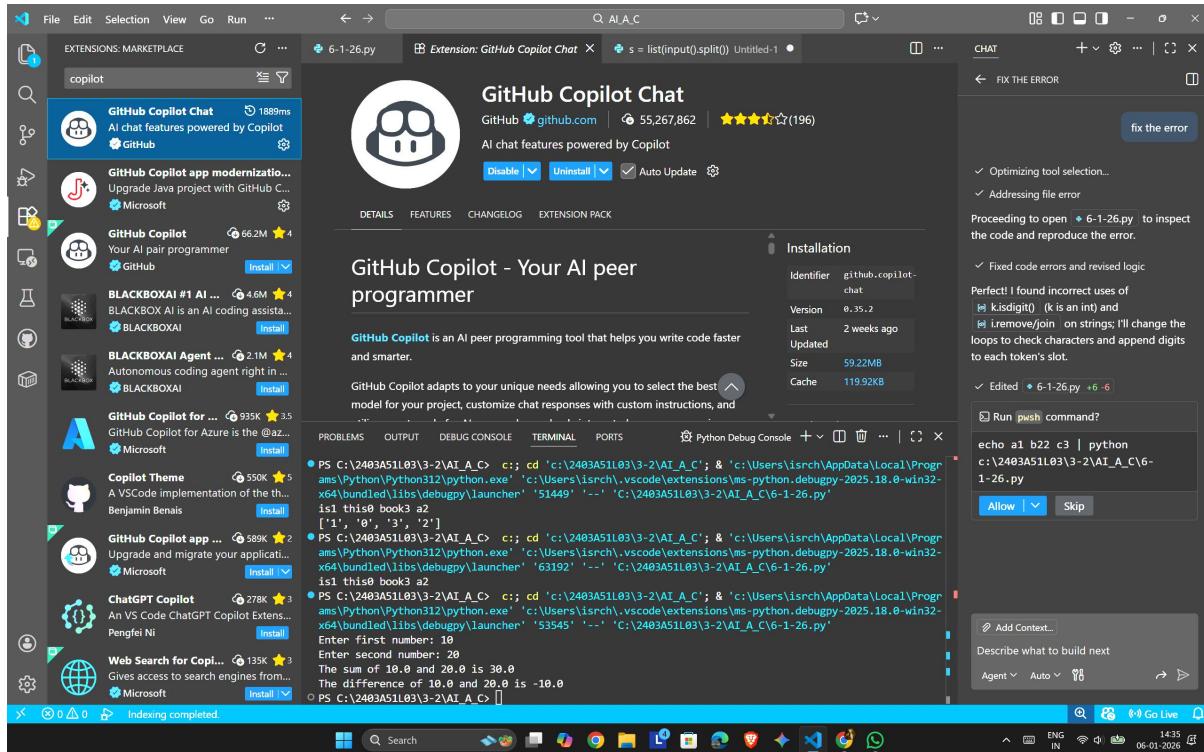
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: II
Academic Session	: 2025-2026
Name of Student	: I.Sathwik Rajeshwara Chary
Enrollment No.	: 2403A51L03
Batch No.	: 51
Date	: 06/01/26

Submission Starts here

Screenshots:

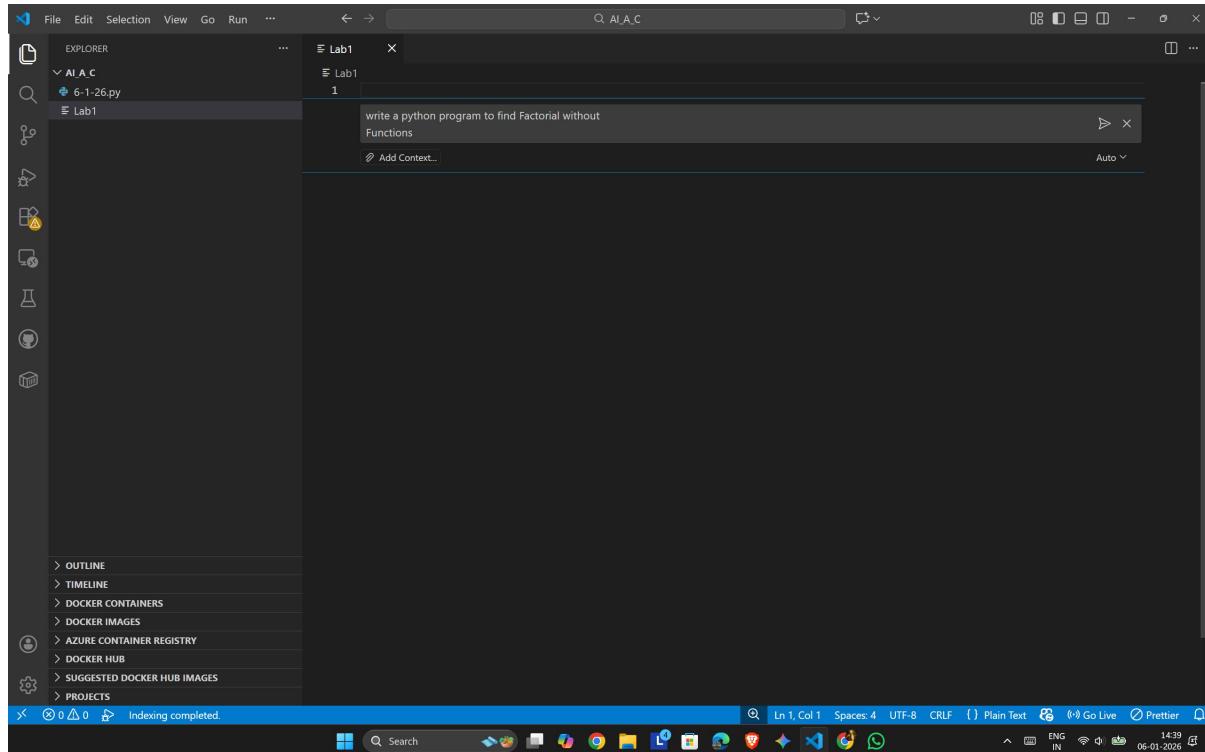
TASK - 0

- Install “Github Copilot” in Visual Studio Code



- 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.



A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows an 'EXPLORER' view with a folder 'AI A.C' containing files '6-1-26.py' and 'Lab1'. The main editor area shows a generated Python script:

```
# Factorial without Functions
n = int(input("Enter a number: "))
factorial = 1
for i in range(1, n + 1):
    factorial = factorial * i
print(f"Factorial of {n} is {factorial}")
```

The code is highlighted in green. A 'Completion' bar at the top right says 'Keep (Ctrl+Y)' and 'Undo'. The bottom status bar shows 'Indexing completed'.

OUTPUT:

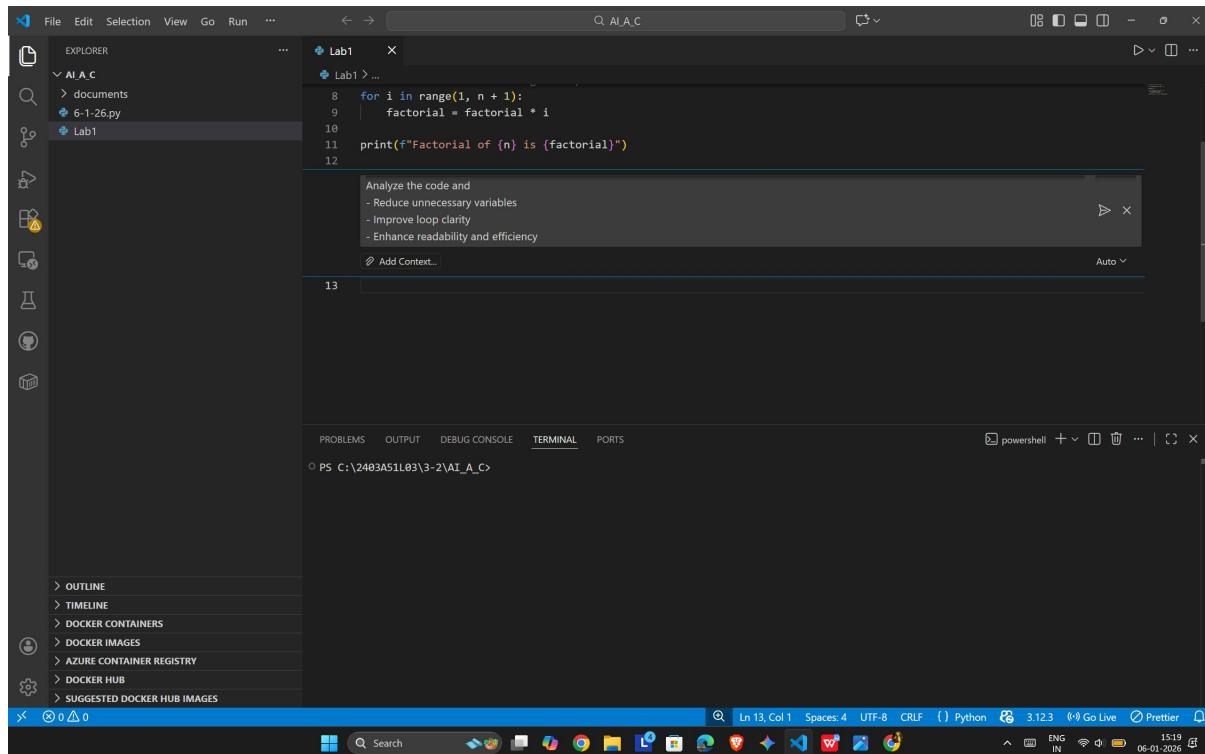
```
File Edit Selection View Go Run ... < > Q AI_A_C Explorer Lab1 x 6-1-26.py Lab1 < ... > ... # Factorial without Functions 1 2 3 n = int(input("Enter a number: ")) 4 5 factorial = 1 6 7 # Calculate factorial using a loop 8 for i in range(1, n + 1): 9     factorial = factorial * i 10 11 print(f"Factorial of {n} is {factorial}") PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + < > ... | < > x PS C:\2403A51L03\3-2\AI_A_C> c: cd 'c:\2403A51L03\3-2\AI_A_C'; & 'c:\Users\lsrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\lsrch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62962' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1' Enter a number: 5 Factorial of 5 is 120 PS C:\2403A51L03\3-2\AI_A_C> Indexing completed Search 3.12.3 ENG IN 06-01-2026 14:40
```

- 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 Description

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

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



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a folder named 'AI_A_C' containing 'documents', '6-1-26.py', and 'Lab1'. The 'Lab1' folder is expanded, showing a file named 'Lab1.py' with the following code:

```

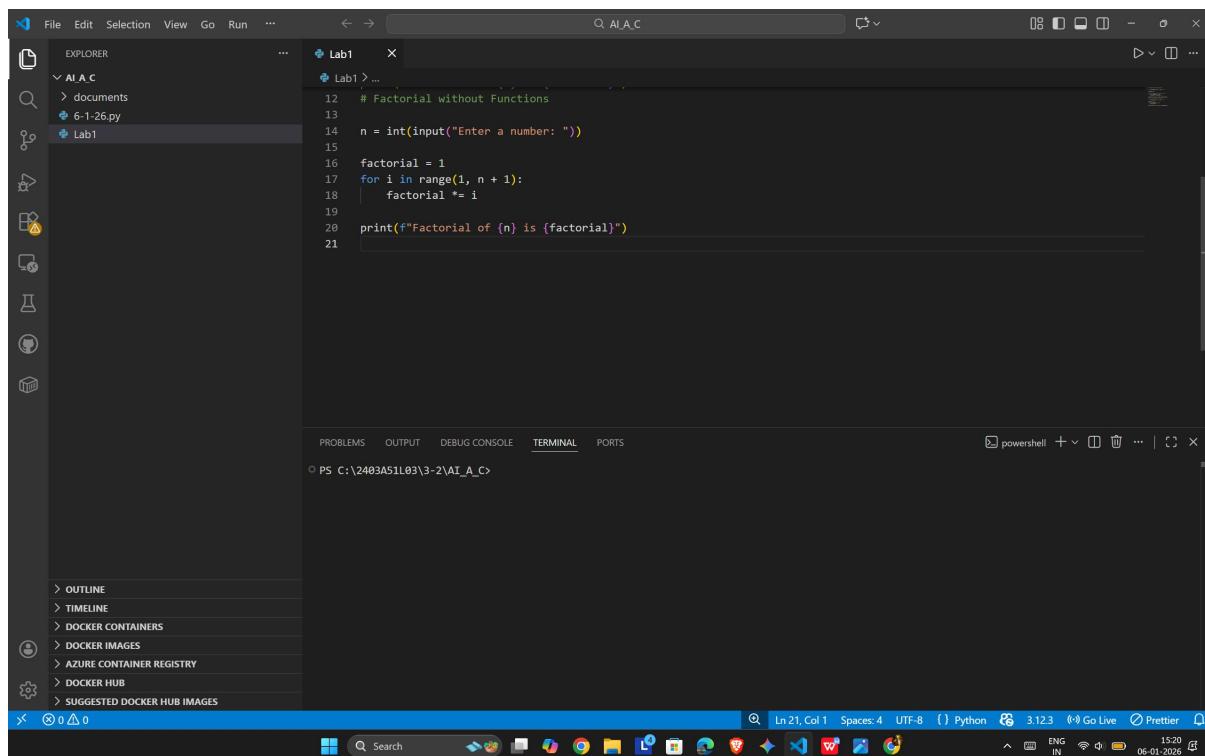
8  for i in range(1, n + 1):
9      factorial = factorial * i
10
11 print(f"Factorial of {n} is {factorial}")
12

```

A tooltip is displayed over the code, listing the tasks to analyze it:

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

The terminal at the bottom shows the command 'PS C:\2403A51L03\3-2\AI_A_C>'.



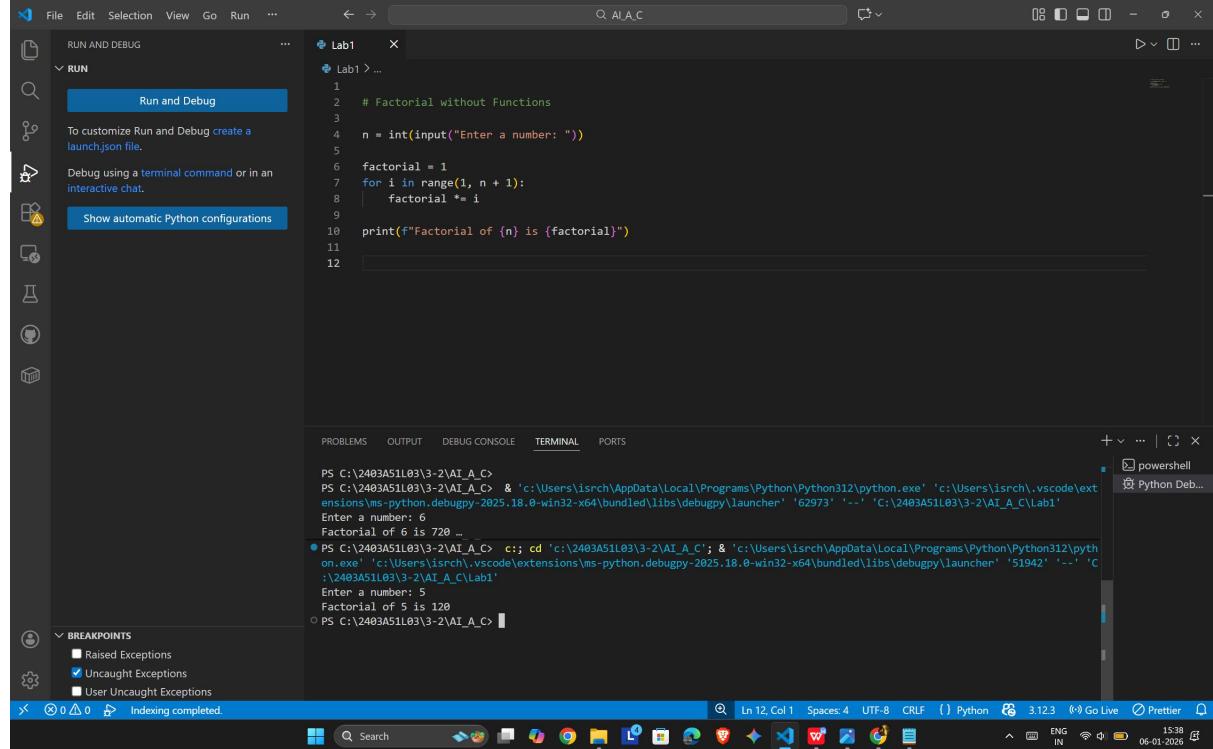
The screenshot shows the Visual Studio Code interface with the same project structure. The 'Lab1.py' file now contains the following code:

```

12 # Factorial without Functions
13
14 n = int(input("Enter a number: "))
15
16 factorial = 1
17 for i in range(1, n + 1):
18     factorial *= i
19
20 print(f"Factorial of {n} is {factorial}")
21

```

The terminal at the bottom shows the command 'PS C:\2403A51L03\3-2\AI_A_C>'.



```

1 # Factorial without Functions
2
3 n = int(input("Enter a number: "))
4
5 factorial = 1
6 for i in range(1, n + 1):
7     factorial *= i
8
9 print(f"Factorial of {n} is {factorial}")
10
11
12

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\2403A51L03\3-2\AI_A_C> PS C:\2403A51L03\3-2\AI_A_C> & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62973' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720 ...
PS C:\2403A51L03\3-2\AI_A_C> c:; cd 'c:\2403A51L03\3-2\AI_A_C'; & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51942' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1'
Enter a number: 5
Factorial of 5 is 120
PS C:\2403A51L03\3-2\AI_A_C>

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- User Uncaught Exceptions

Indexing completed.

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.**

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

```

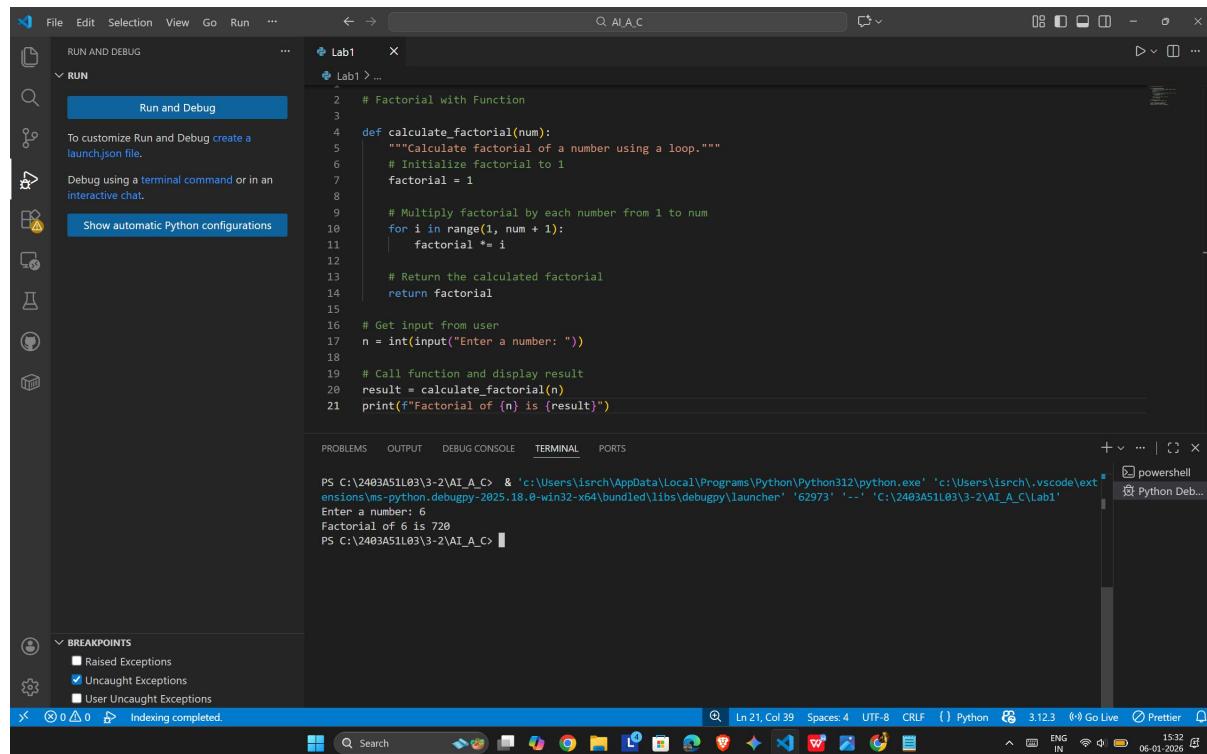
File Edit Selection View Go Run ...
EXPLORER Lab1 > ...
AI_A_C
documents
6-1-26.py
Lab1
...
11 print(f"Factorial of {n} is {factorial}")
12 # Factorial without Functions
13
14 n = int(input("Enter a number: "))
15
16 factorial = 1
17 for i in range(1, n + 1):
18     factorial *= i
19
20 print(f"Factorial of {n} is {factorial}")
21
Create a Python function to calculate factorial using a loop with comments
∅ Add Context...
22
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\2403A51L03\3-2\AI_A_C>

```

```

File Edit Selection View Go Run ...
EXPLORER Lab1 > ...
AI_A_C
documents
6-1-26.py
Lab1
...
17 for i in range(1, n + 1):
18     factorial *= i
19
20 print(f"Factorial of {n} is {factorial}")
21
Generate code ✓ ×
∅ Add Context...
22 # Factorial with Function
23
24 def calculate_factorial(num):
25     """Calculate factorial of a number using a loop."""
26     # Initialize factorial to 1
27     factorial = 1
28
29     # Multiply factorial by each number from 1 to num
30     for i in range(1, num + 1):
31         factorial *= i
32
33     # Return the calculated factorial
34     return factorial
35
36 # Get input from user
37 n = int(input("Enter a number: "))
38
39 # Call function and display result
40 result = calculate_factorial(n)
41 print(f"Factorial of {n} is {result}")

```



```

1 # Factorial with Function
2
3 def calculate_factorial(num):
4     """Calculate factorial of a number using a loop."""
5     # Initialize factorial to 1
6     factorial = 1
7
8     # Multiply factorial by each number from 1 to num
9     for i in range(1, num + 1):
10         factorial *= i
11
12     # Return the calculated factorial
13     return factorial
14
15
16 # Get input from user
17 n = int(input("Enter a number: "))
18
19 # Call function and display result
20 result = calculate_factorial(n)
21 print(f"Factorial of {n} is {result}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\2403A51L03\3-2\AI_A_C> & 'c:\Users\isrchr\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrchr\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62973' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720
PS C:\2403A51L03\3-2\AI_A_C>

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- User Uncaught Exceptions

Indexing completed.

- **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.

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

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, and other standard options. A tab labeled 'Lab1' is active, showing Python code for calculating factorials:

```
def calculate_factorial(num):
    # Return the calculated factorial
    return factorial

    # Get input from user
    n = int(input("Enter a number: "))

    # Call function and display result
    result = calculate_factorial(n)
    print(f"Factorial of {n} is {result}")

31
```

Below the code editor, a tooltip provides instructions for comparing non-function and function-based programs:

Compare the non-function and function-based programs on the following criteria:
- Logic clarity
- Reusability
- Debugging ease

The bottom section of the interface shows the terminal window with the following output:

```
PS C:\2403A51L03\3-2\AI_A_C>
PS C:\2403A51L03\3-2\AI_A_C> & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62973' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720 -
PS C:\2403A51L03\3-2\AI_A_C> :: cd 'c:\2403A51L03\3-2\AI_A_C'; & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51942' '--' 'C:\2403A51L03\3-2\AI_A_C\Lab1'
Enter a number: 5
Factorial of 5 is 120
PS C:\2403A51L03\3-2\AI_A_C>
```

The bottom status bar indicates indexing is completed. The bottom right corner shows the date and time: 06-01-2024 15:44.

The screenshot shows a code editor interface with a Python script open. The script compares Non-Function and Function-Based Factorial Programs across various criteria. The code uses f-strings for printing results.

```
# Comparison of Non-Function vs Function-Based Factorial Programs

print("\n==== COMPARISON ===\n")

comparison = {
    "Logic Clarity": {
        "Non-Function": "Direct and simple, but logic is mixed with I/O",
        "Function-Based": "Clearer separation of concerns; logic isolated in function"
    },
    "Reusability": {
        "Non-Function": "Cannot reuse; must rewrite code for each use",
        "Function-Based": "Highly reusable; call function multiple times"
    },
    "Debugging Ease": {
        "Non-Function": "Harder; must debug entire script together",
        "Function-Based": "Easier; test function independently in isolation"
    },
    "Large Projects": {
        "Non-Function": "Not suitable; leads to code duplication and maintenance issues",
        "Function-Based": "Suitable; promotes modularity and organization"
    },
    "AI Dependency Risk": {
        "Non-Function": "Lower risk; simple, self-contained logic",
        "Function-Based": "Higher risk if function relies on external AI; more dependencies"
    }
}

for criterion, comparison_data in comparison.items():
    print(f"\n{criterion}:")
    for approach, description in comparison_data.items():
        print(f" | {approach}: {description}")
    print()
```

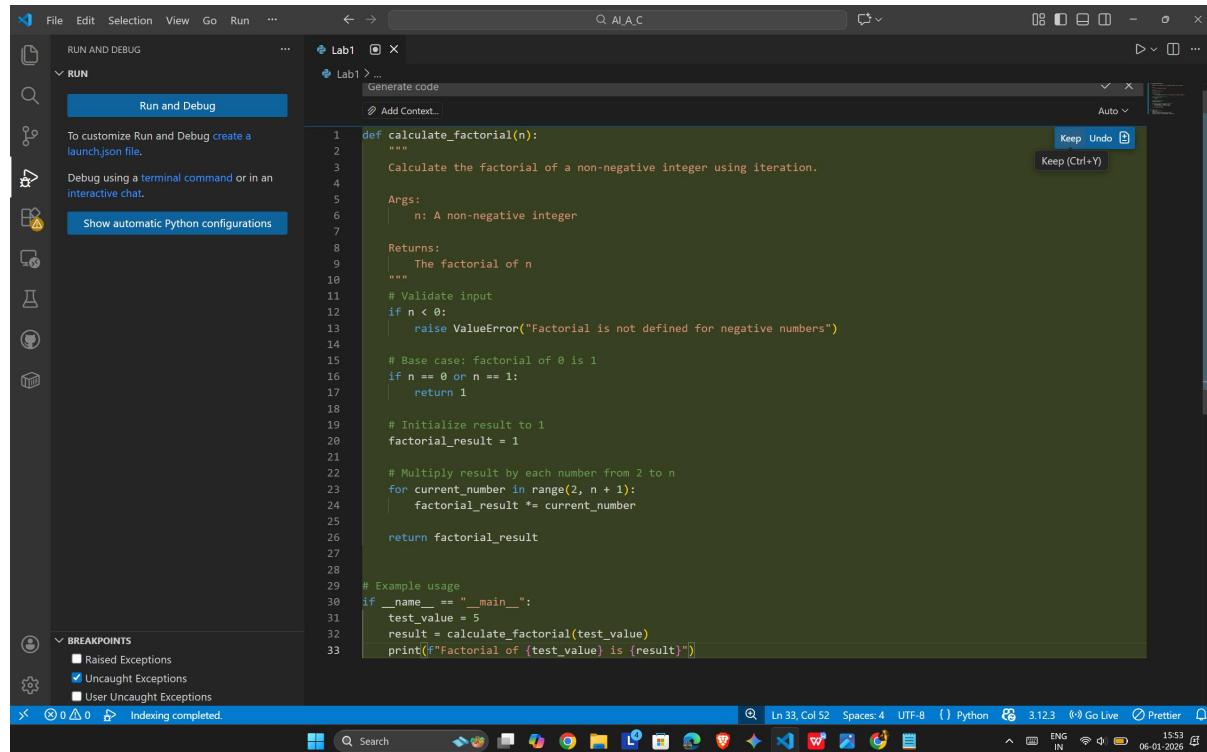
TASK - 5

Task Description

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

The screenshot shows the Visual Studio Code interface with the following details:

- File**, **Edit**, **Selection**, **View**, **Go**, **Run**, **...** menu bar.
- Search** bar at the top right with the text "AI_A_C".
- Run and Debug** sidebar on the left:
 - RUN** section: "Run and Debug" button (highlighted), "To customize Run and Debug create a launch.json file.", "Debug using a terminal command or in an interactive chat.", "Show automatic Python configurations" button.
 - BREAKPOINTS** section: "Raised Exceptions" (unchecked), "Uncaught Exceptions" (checked), "User Uncaught Exceptions" (unchecked).
- Code Editor**:
 - Tab: Lab1
 - Text: "Write a Python function to calculate factorial using an iterative approach with meaningful variable names and comments."
 - Buttons: "Add Context...", "Auto".
- Bottom Status Bar**: "Indexing completed." and various status icons.
- Bottom Taskbar**: Icons for Search, GitHub, Git, Terminal, Python, Go Live, Prettier, and others.



The screenshot shows a dark-themed instance of Visual Studio Code. The main editor pane displays a Python script named `Lab1.py`. The code defines a function `calculate_factorial` that calculates the factorial of a non-negative integer using iteration. The code includes docstrings, type hints, and a base case for zero. A terminal tab at the bottom shows the message "Indexing completed". The status bar at the bottom right indicates the file is 3.12.3, has 1553 words, and was last modified on 06-01-2026.

```

def calculate_factorial(n):
    """
    Calculate the factorial of a non-negative integer using iteration.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """
    # Validate input
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    # Base case: factorial of 0 is 1
    if n == 0 or n == 1:
        return 1

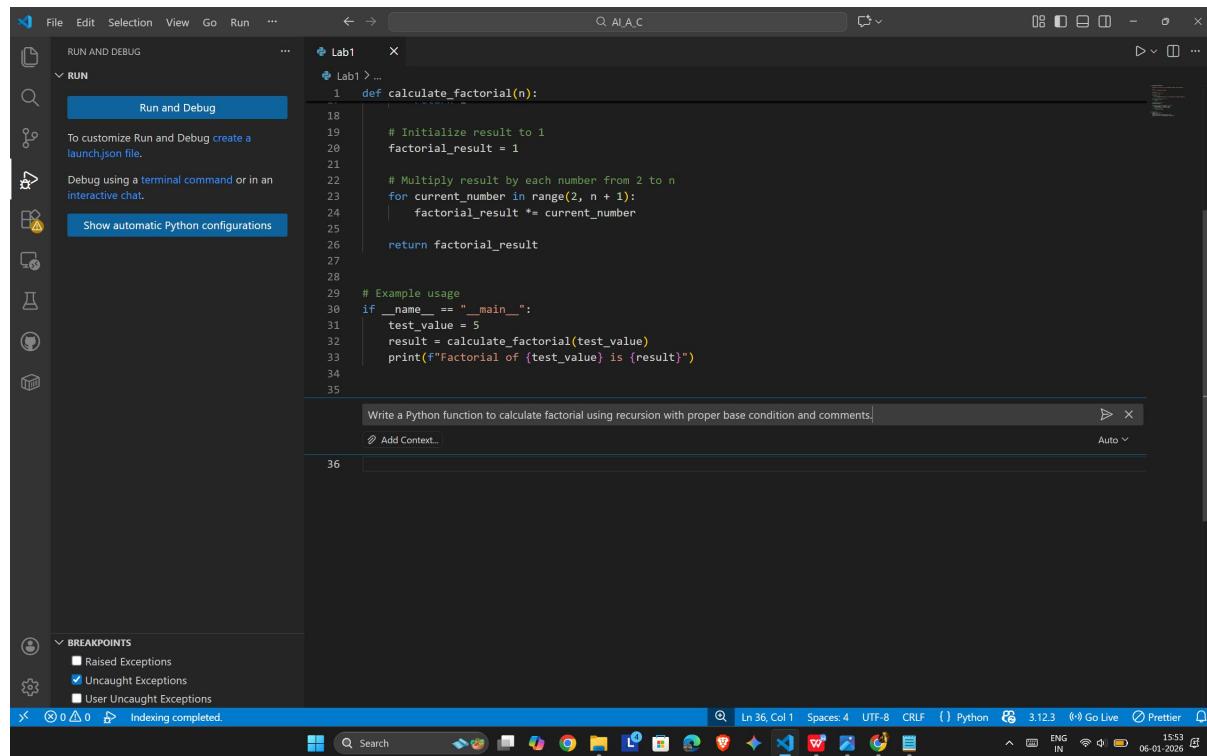
    # Initialize result to 1
    factorial_result = 1

    # Multiply result by each number from 2 to n
    for current_number in range(2, n + 1):
        factorial_result *= current_number

    return factorial_result

# Example usage
if __name__ == "__main__":
    test_value = 5
    result = calculate_factorial(test_value)
    print(f"Factorial of {test_value} is {result}")

```



This screenshot shows the same instance of VS Code after the user has modified the code to use recursion. The function definition remains the same, but the explanatory comments have been removed. A new comment at the bottom of the code pane reads "Write a Python function to calculate factorial using recursion with proper base condition and comments.". The status bar at the bottom right indicates the file is 3.12.3, has 1553 words, and was last modified on 06-01-2026.

```

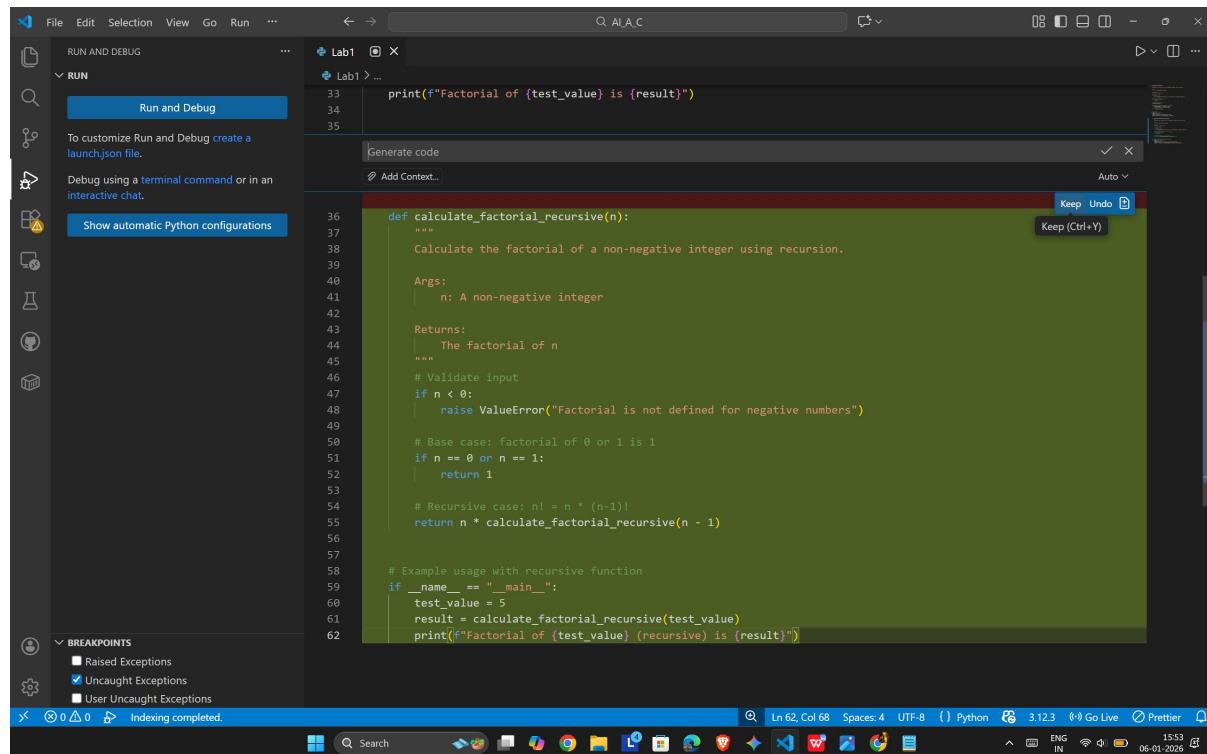
def calculate_factorial(n):
    # Initialize result to 1
    factorial_result = 1

    # Multiply result by each number from 2 to n
    for current_number in range(2, n + 1):
        factorial_result *= current_number

    return factorial_result

# Example usage
if __name__ == "__main__":
    test_value = 5
    result = calculate_factorial(test_value)
    print(f"Factorial of {test_value} is {result}")

```



```

File Edit Selection View Go Run ... ← → 🔍 AI/AI
RUN AND DEBUG ... Lab1 ✎
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
Breakpoints
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
Indexing completed.
33     print(f"Factorial of {test_value} is {result}")
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

def calculate_factorial_recursive(n):
    """
    Calculate the factorial of a non-negative integer using recursion.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """

    # Validate input
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1

    # Recursive case: n! = n * (n-1)!
    return n * calculate_factorial_recursive(n - 1)

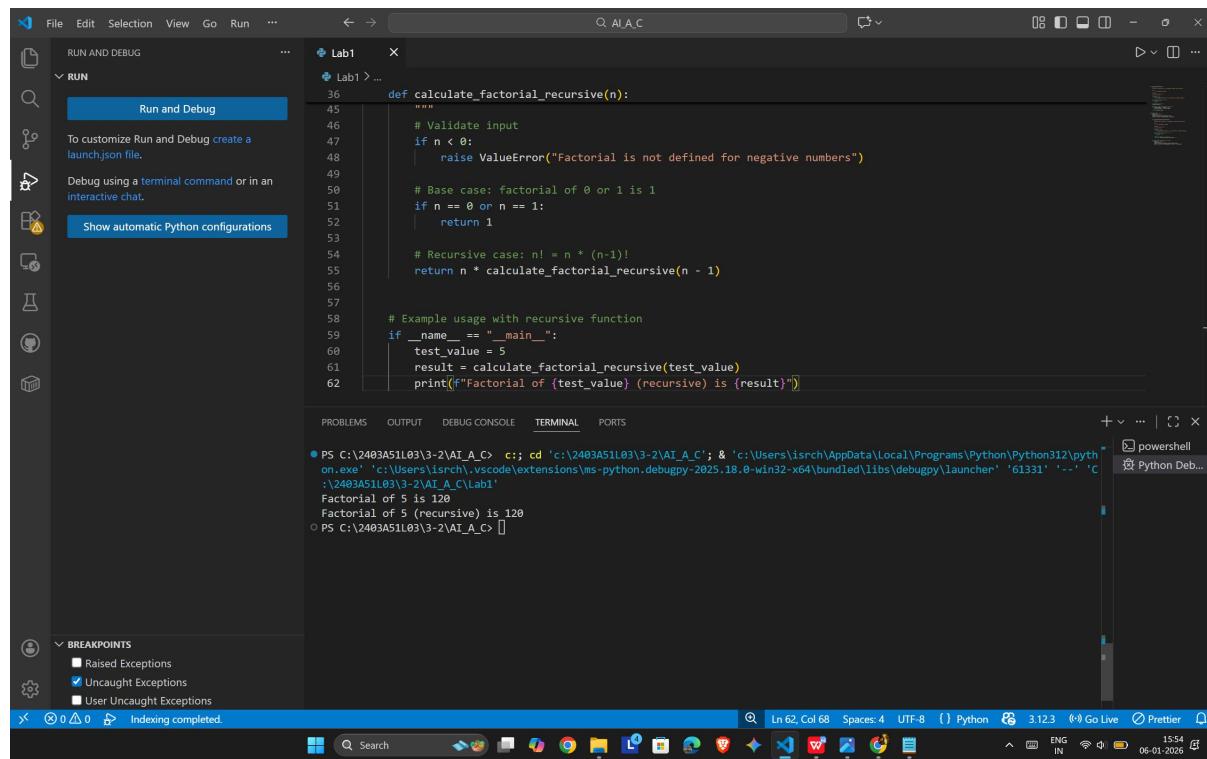
# Example usage with recursive function
if __name__ == "__main__":
    test_value = 5
    result = calculate_factorial_recursive(test_value)
    print(f"Factorial of {test_value} (recursive) is {result}")

```

Ln 62, Col 68 Spaces: 4 UTF-8 ⓘ Python 3.12.3 ⓘ Go Live ⓘ Prettier ⓘ

15:53 06-01-2026 ⓘ

OUTPUT:



```

File Edit Selection View Go Run ... ← → 🔍 AI/AI
RUN AND DEBUG ... Lab1 ✎
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
PS C:\2403A51L03\3-2\AT_A_C> c:; cd 'c:\2403A51L03\3-2\AT_A_C'; & 'c:\Users\lrsch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\lrsch\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61331' '--' 'C:\2403A51L03\3-2\AT_A_C\Lab1'
Factorial of 5 is 120
Factorial of 5 (recursive) is 120
PS C:\2403A51L03\3-2\AT_A_C> []

```

Ln 62, Col 68 Spaces: 4 UTF-8 ⓘ Python 3.12.3 ⓘ Go Live ⓘ Prettier ⓘ

15:54 06-01-2026 ⓘ

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.