

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

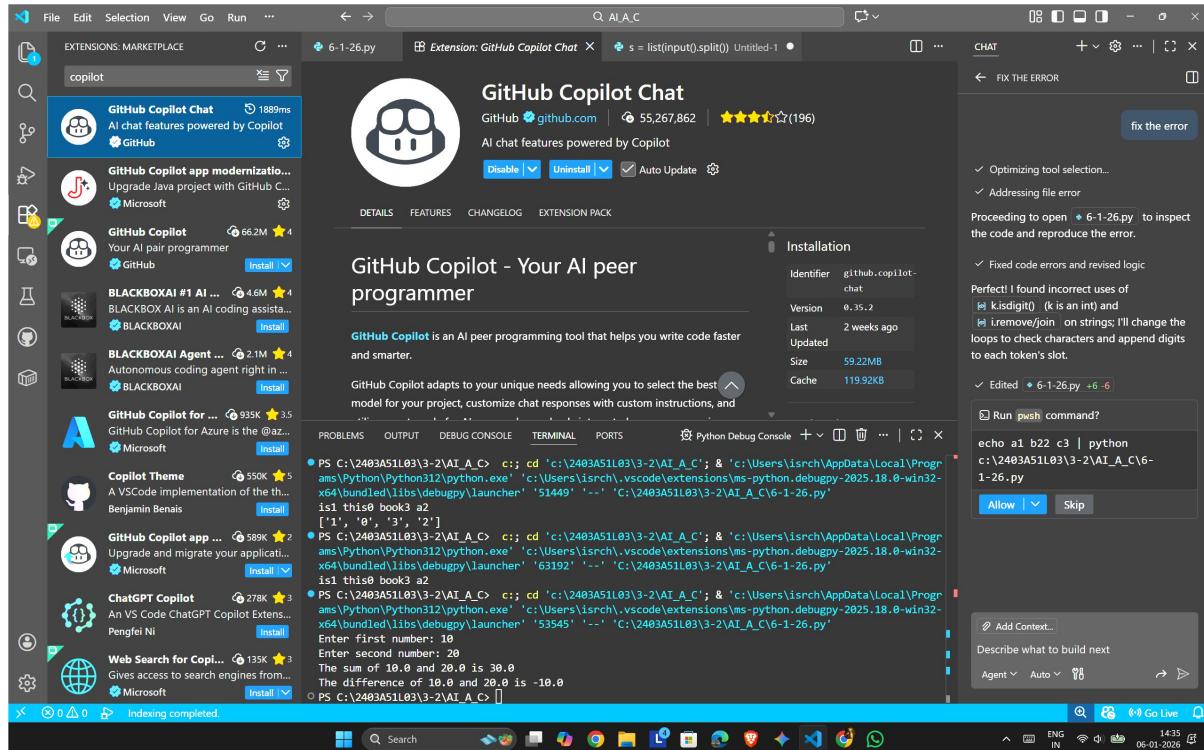
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: II
Academic Session	: 2025-2026
Name of Student	: Ganesh
Enrollment No.	: 2403A51L55
Batch No.	: 52
Date	: 06/01/26

Submission Starts here

Screenshots:

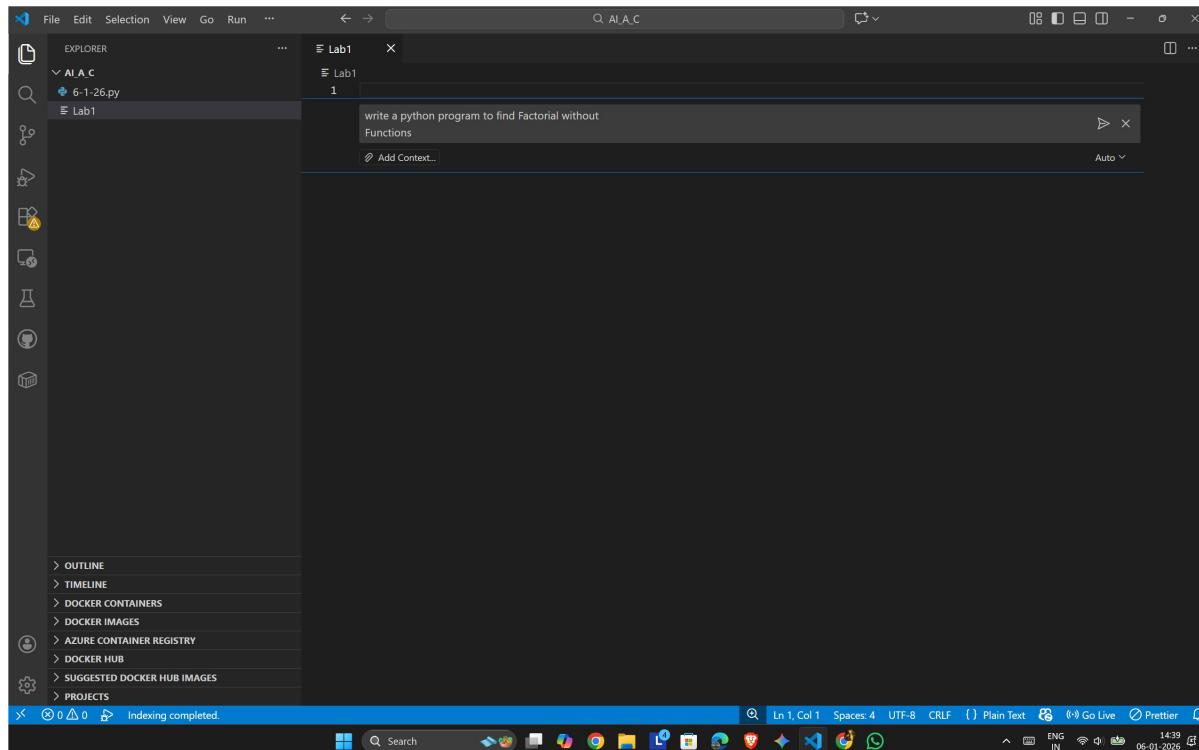
TASK - 0

- Install “Github Copilot” in Visual Studio Code



- 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.



File Edit Selection View Go Run ...

REPL Lab1

AI A.C

6-1-26.py

Lab1

1

write a python program to find Factorial without Functions

Add Context...

Auto

OUTLINE

TIMELINE

DOCKER CONTAINERS

DOCKER IMAGES

AZURE CONTAINER REGISTRY

DOCKER HUB

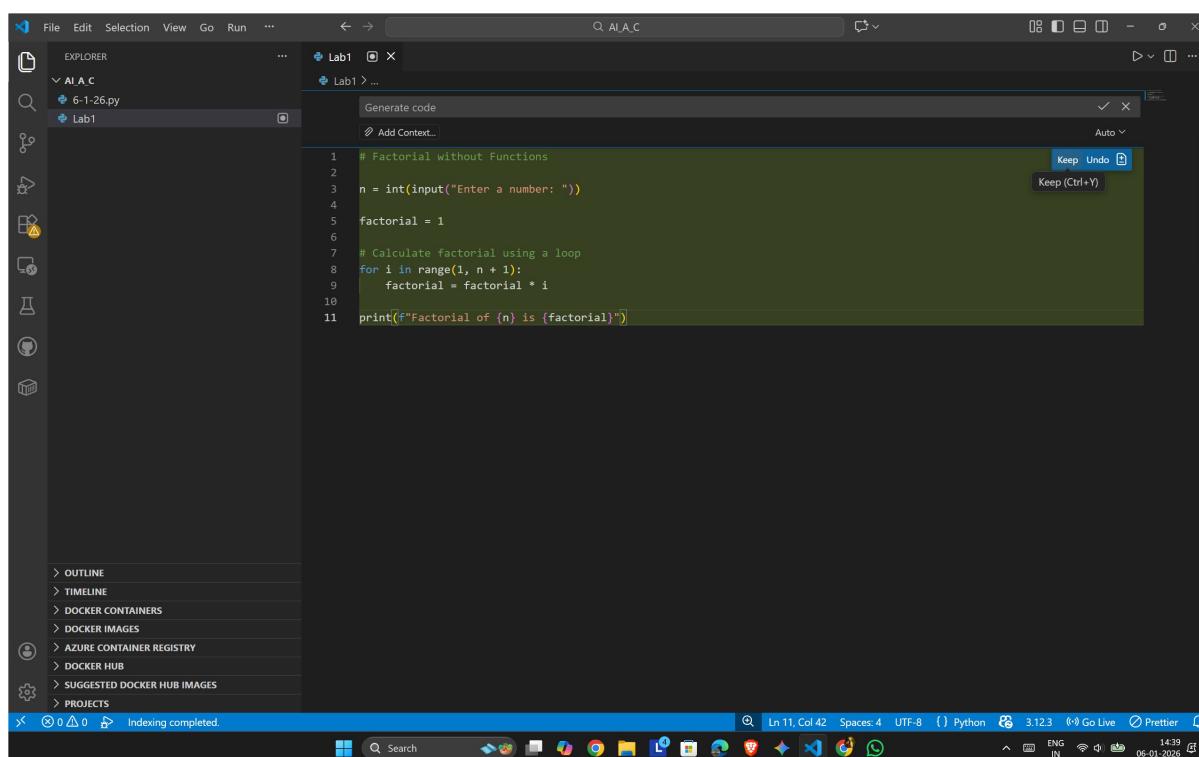
SUGGESTED DOCKER HUB IMAGES

PROJECTS

Indexing completed

Ln 1, Col 1 Spaces: 4 UTF-8 Plain Text Go Live Prettier

14:39 06-01-2026



File Edit Selection View Go Run ...

REPL Lab1

AI A.C

6-1-26.py

Lab1

Generate code

Add Context...

```
# 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}")
```

Keep Undo

Keep (Ctrl+Y)

OUTLINE

TIMELINE

DOCKER CONTAINERS

DOCKER IMAGES

AZURE CONTAINER REGISTRY

DOCKER HUB

SUGGESTED DOCKER HUB IMAGES

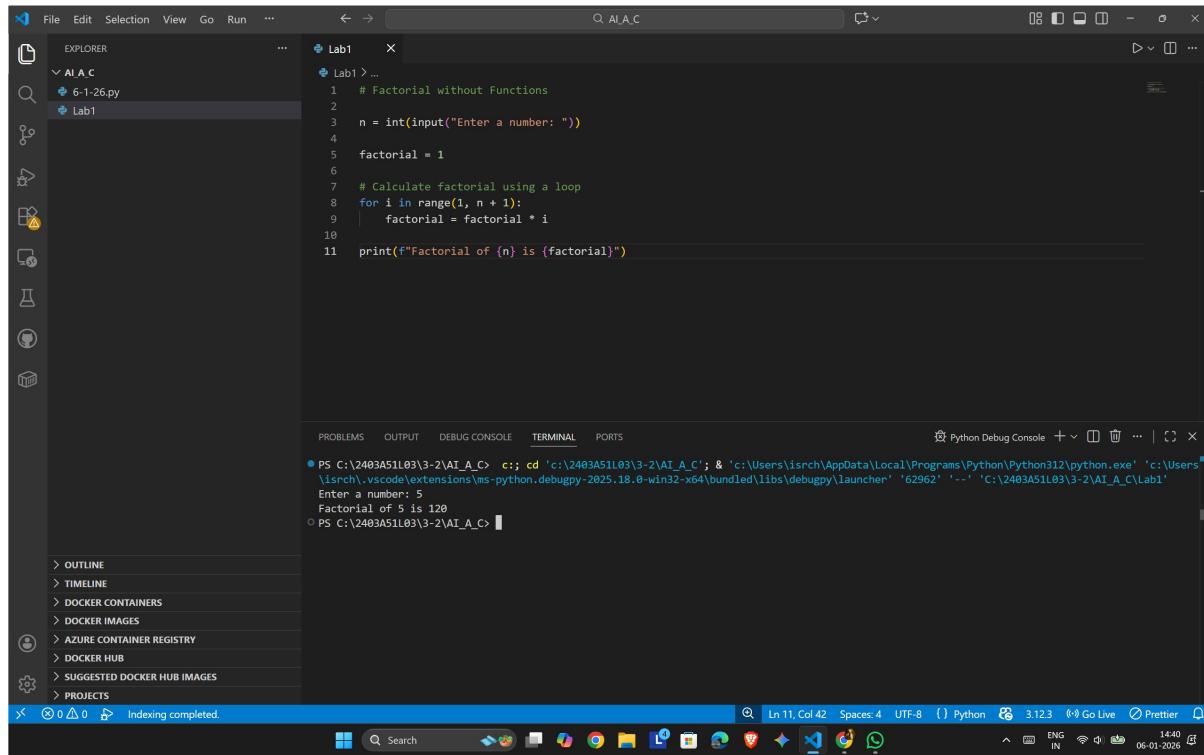
PROJECTS

Indexing completed

Ln 11, Col 42 Spaces: 4 UTF-8 Python 3.12.3 Go Live Prettier

14:39 06-01-2026

OUTPUT:



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left has a tree view with 'AI_A_C' expanded, showing '6-1-26.py' and 'Lab1'. The main editor area displays a Python script named 'Lab1.py' with the following code:

```

1 # Factorial without Functions
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}")

```

The terminal tab at the bottom shows the output of running the script:

```

PS C:\2403A51L03\3-2\AI_A_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>

```

- 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'. The main editor area displays the following Python code:

```

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

```

A tooltip window titled 'Analyze the code and' provides three suggestions:

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

The bottom status bar shows the path 'C:\2403A51L03\3-2\AI_A_C>' and the date '06-01-2026'.

The screenshot shows the Visual Studio Code interface with the same project structure and file. The main editor area now contains the following Python code, reflecting the changes suggested by Copilot:

```

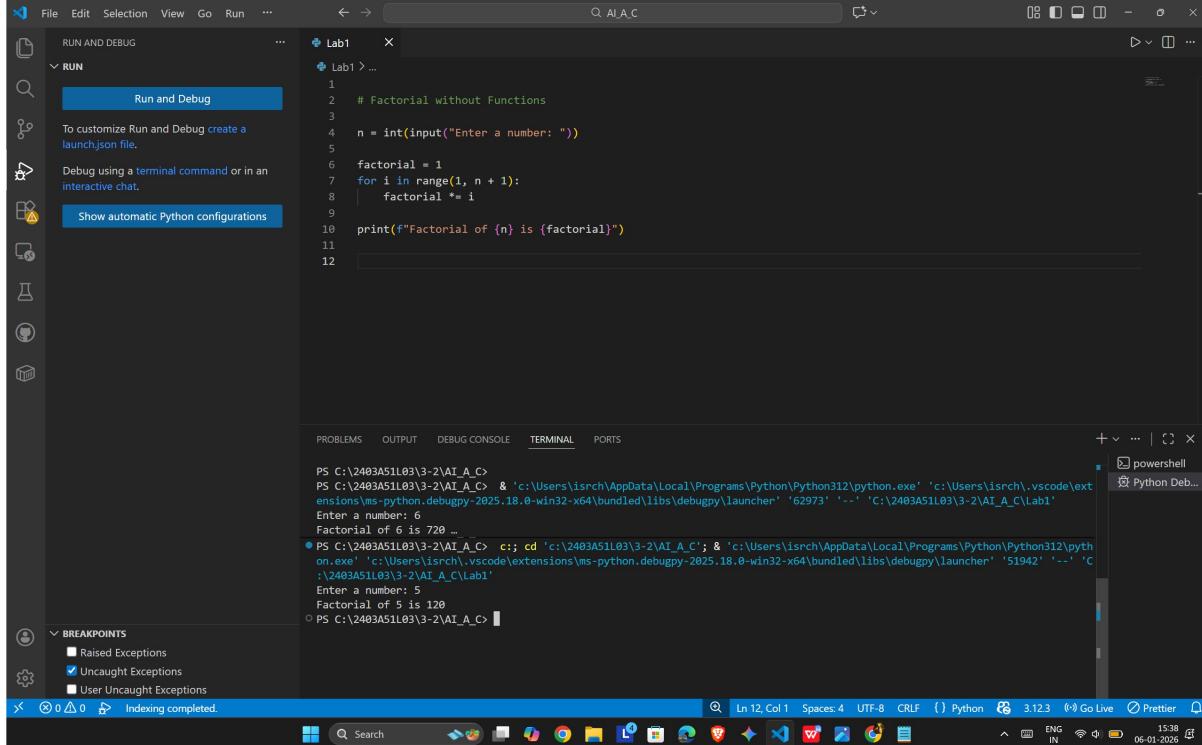
# Factorial without Functions
n = int(input("Enter a number: "))

factorial = 1
for i in range(1, n + 1):
    factorial *= i

print(f"Factorial of {n} is {factorial}")

```

The bottom status bar shows the path 'C:\2403A51L03\3-2\AI_A_C>' and the date '06-01-2026'.



The screenshot shows the Visual Studio Code interface with a Python script named `Lab1.py` open in the editor. The code calculates the factorial of a user-specified number without using a function. It includes a comment explaining the logic and uses an f-string for output. The terminal below shows the execution of the script and its output for n=6 and n=5.

```

RUN AND DEBUG
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

Lab1 > ...
1
2 # Factorial without Functions
3
4 n = int(input("Enter a number: "))
5
6 factorial = 1
7 for i in range(1, n + 1):
8     factorial *= i
9
10 print(f"Factorial of {n} is {factorial}")
11
12

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
PS C:\2403A51L03\3-2\AI_A_C>
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> c:; cd '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' '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
Indexing completed.

Ln 12, Col 1 Spaces: 4 UTF-8 Python 3.12.3 Go Live Prettier
ENG IN 15:38 06-01-2026

```

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 AI_A_C
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>

```

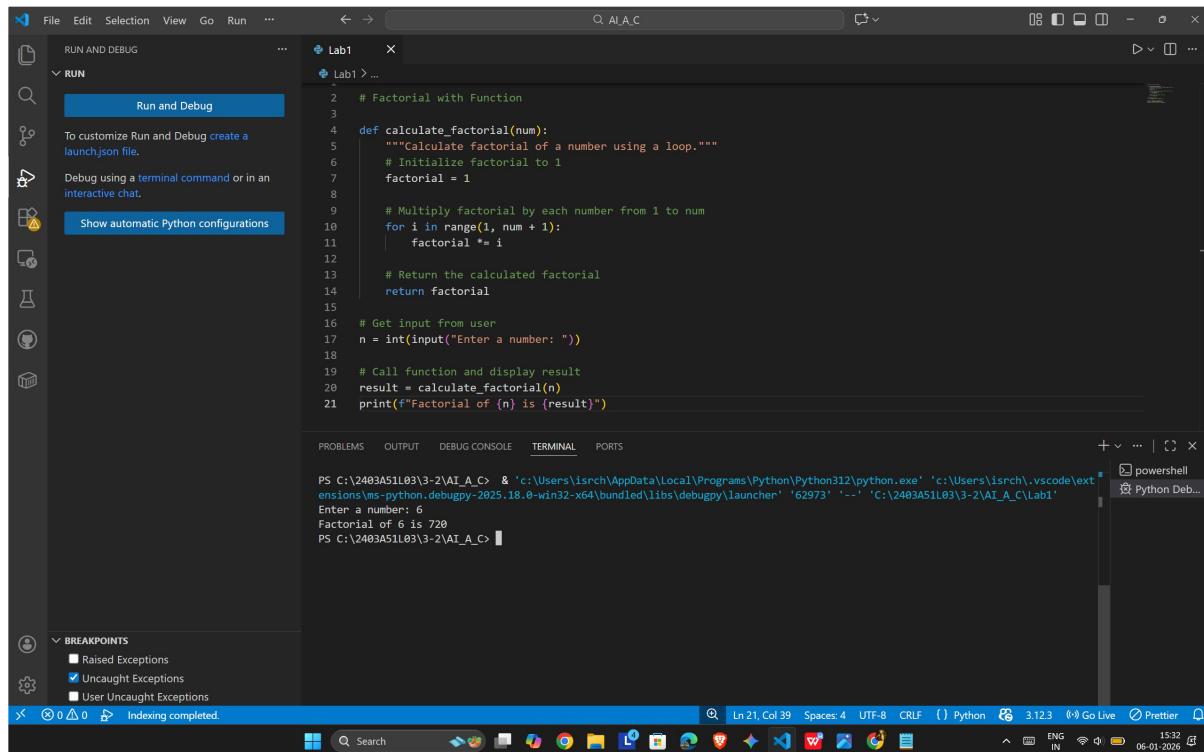
The screenshot shows a dark-themed VS Code interface. The Explorer sidebar on the left lists a folder 'AI_A_C' containing files '6-1-26.py' and 'Lab1'. The 'Lab1' file is open in the editor, showing a script that calculates the factorial of a given number using a for loop. A tooltip from GitHub Copilot is visible, prompting to 'Create a Python function to calculate factorial using a loop with comments'. The terminal at the bottom shows a PowerShell prompt in the directory 'C:\2403A51L03\3-2\AI_A_C'.

```

File Edit Selection View Go Run ...
EXPLORER AI_A_C
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}")

```

The screenshot shows the same VS Code interface after the modularization process. The 'Lab1' file now contains a modularized version of the factorial program. It includes a new function 'calculate_factorial' that takes a parameter 'num' and returns the factorial. The original loop-based calculation has been moved into this function. The main block still calls this function and prints the result. The GitHub Copilot tooltip is no longer present, indicating the task is complete.



```

File Edit Selection View Go Run ... ← → ⌂ Q AI A C RUN AND DEBUG ... Lab1 x
RUN > ...
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 + v ... | x
powershell
Python Deb...
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. On the left, the sidebar includes sections for RUN AND DEBUG, RUN (with 'Run and Debug' selected), and BREAKPOINTS. The main editor area displays a Python script named 'Lab1.py' with the following code:

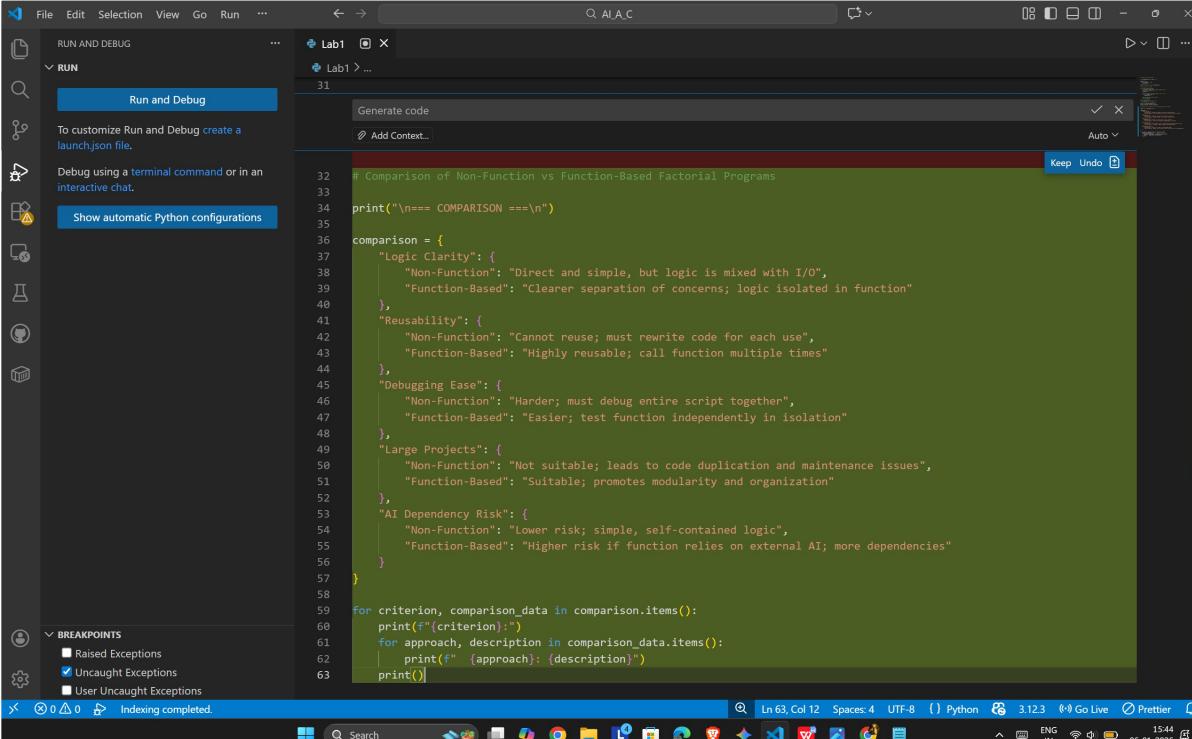
```

13 def calculate_factorial(num):
14     # Return the calculated factorial
15     return factorial
16
17 # Get input from user
18 n = int(input("Enter a number: "))
19
20 # Call function and display result
21 result = calculate_factorial(n)
22 print(f"Factorial of {n} is {result}")
23
24
25
26
27
28
29
30
31
32

```

A tooltip box is overlaid on the code, containing the task description: "Compare the non-function and function-based programs on the following criteria:" followed by a bulleted list: "- Logic clarity", "- Reusability", and "- Debugging ease".

The bottom right corner of the interface shows the status bar with information such as 'Ln 32, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.3', 'Go Live', and a date/time stamp '06-01-2026 15:44'.



```

# 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": "Cleaner 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"
    ]
}

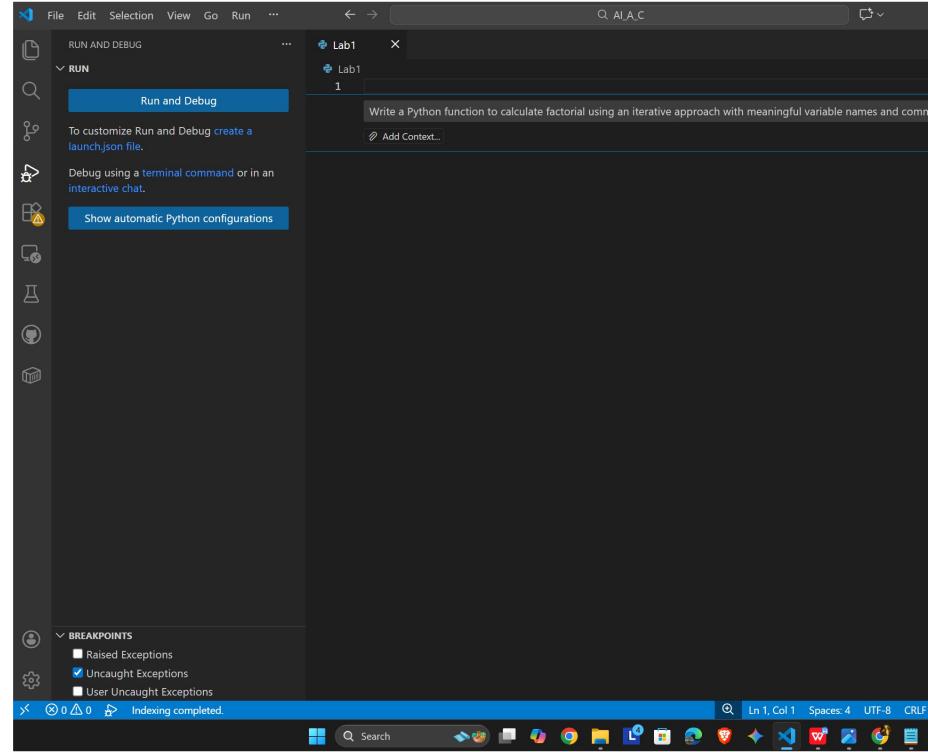
for criterion, comparison_data in comparison.items():
    print(f"\n{criterion}:")
    for approach, description in comparison_data.items():
        print(f"  {approach}: {description}")
    print()

```

TASK - 5

Task Description

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



```

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}")

```

The screenshot shows a Python code editor in VS Code. The code defines a recursive factorial function. A docstring at the top provides a brief description of the function's purpose. Below the docstring, the function handles edge cases (n=0 and n=1) and performs recursive calculations. An example usage block at the bottom demonstrates how to call the function and print its result.

```

1  def calculate_factorial(n):
2      """
3          Calculate the factorial of a non-negative integer using recursion.
4      """
5      Args:
6          n: A non-negative integer
7
8      Returns:
9          The factorial of n
10
11     # Validate input
12     if n < 0:
13         raise ValueError("Factorial is not defined for negative numbers")
14
15     # Base case: factorial of 0 or 1 is 1
16     if n == 0 or n == 1:
17         return 1
18
19     # Recursive case: n! = n * (n-1)!
20     return n * calculate_factorial(n - 1)
21
22
23     # Example usage
24     if __name__ == "__main__":
25         test_value = 5
26         result = calculate_factorial(test_value)
27         print(f"Factorial of {test_value} is {result}")
28
29
30
31
32
33
34
35
36

```

This screenshot shows the same Python code as the first one, but with more extensive documentation. A detailed docstring at the top includes a summary, parameters, and returns section. The code also includes validation for negative numbers and handles the base cases where n is 0 or 1. An example usage block at the bottom shows how to call the function and print its result. A 'Keep' button is visible in the status bar, indicating changes have been saved.

```

1  def calculate_factorial(n):
2      """
3          Calculate the factorial of a non-negative integer using recursion.
4      """
5      Args:
6          n: A non-negative integer
7
8      Returns:
9          The factorial of n
10
11     # Validate input
12     if n < 0:
13         raise ValueError("Factorial is not defined for negative numbers")
14
15     # Base case: factorial of 0 or 1 is 1
16     if n == 0 or n == 1:
17         return 1
18
19     # Recursive case: n! = n * (n-1)!
20     return n * calculate_factorial(n - 1)
21
22
23     # Example usage with recursive function
24     if __name__ == "__main__":
25         test_value = 5
26         result = calculate_factorial(test_value)
27         print(f"Factorial of {test_value} (recursive) is {result}")
28
29
30
31
32
33
34
35
36

```

OUTPUT:

```

File Edit Selection View Go Run ... ← → Q AI_A_C
RUN AND DEBUG ...
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
PS C:\2403A51L03\3-2\AI_A_C> cd 'c:\2403A51L03\3-2\AI_A_C'; & 'c:\Users\isrchr\AppData\Local\Programs\Python\Python38\python.exe' 'c:\Users\isrchr\vscode\extensions\ms-python.python\2025.18.0-win32-x64\bundled\libs\calculate_factorial_recursive.py'
Factorial of 5 is 120
Factorial of 5 (recursive) is 120
PS C:\2403A51L03\3-2\AI_A_C> []

```

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- User Uncaught Exceptions

Explanation

- 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.