

Lab Assignment # 1

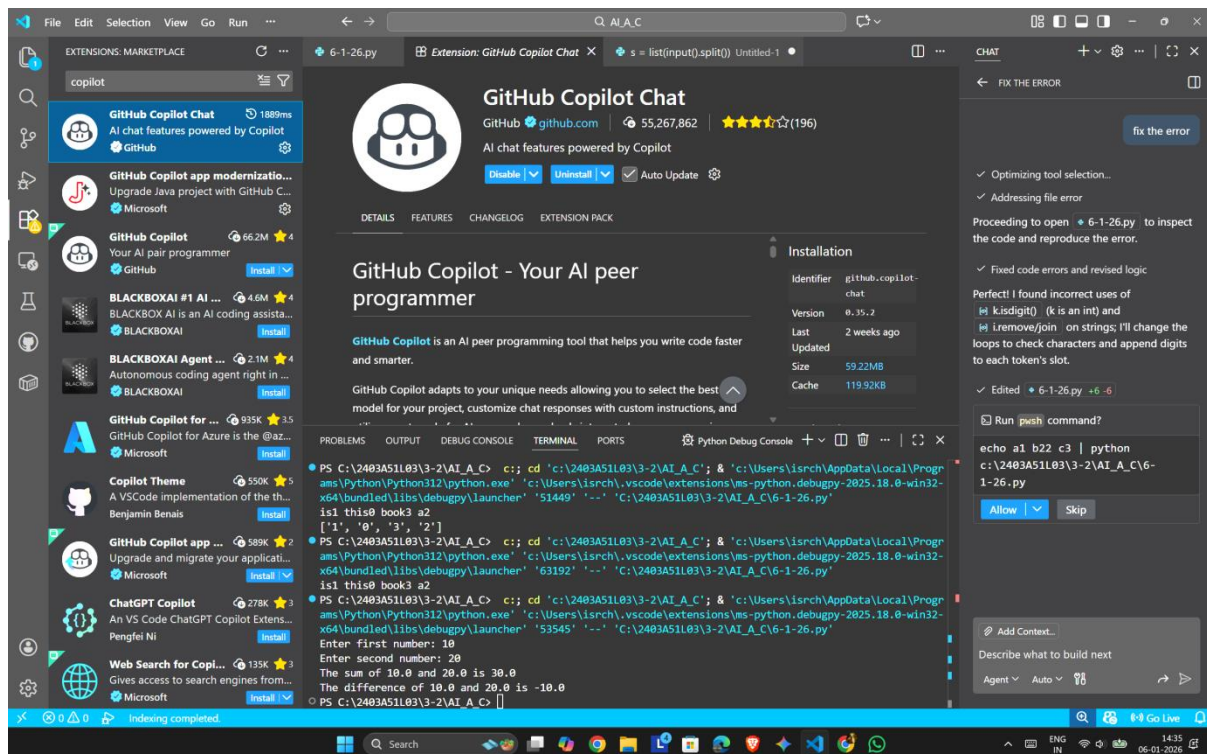
Program : B. Tech (CSE)
Specialization :
Course Title : AI Assisted Coding
Course Code : 23CS002PC304
Semester : II
Academic Session : 2025-2026
Name of Student : J Harshith yadav
Enrollment No. : 2403A51L49
Batch No. : 52
Date : 06/01/26

Submission Starts here

Screenshots:

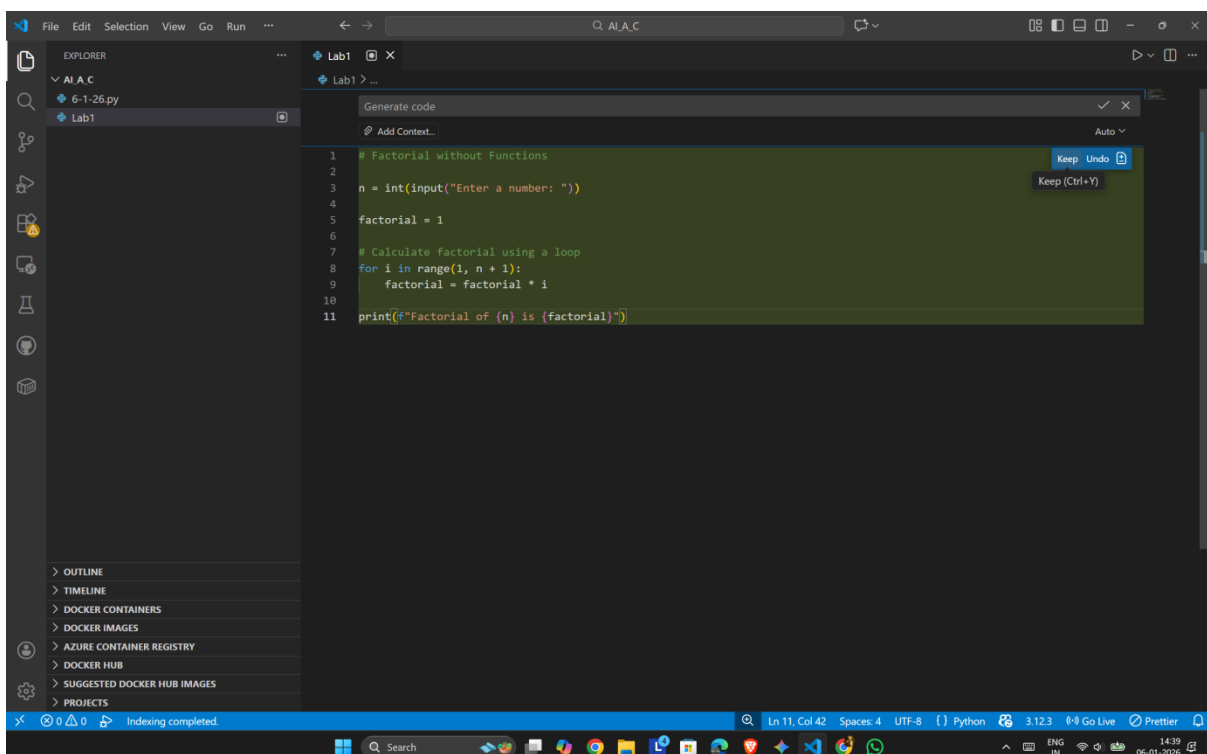
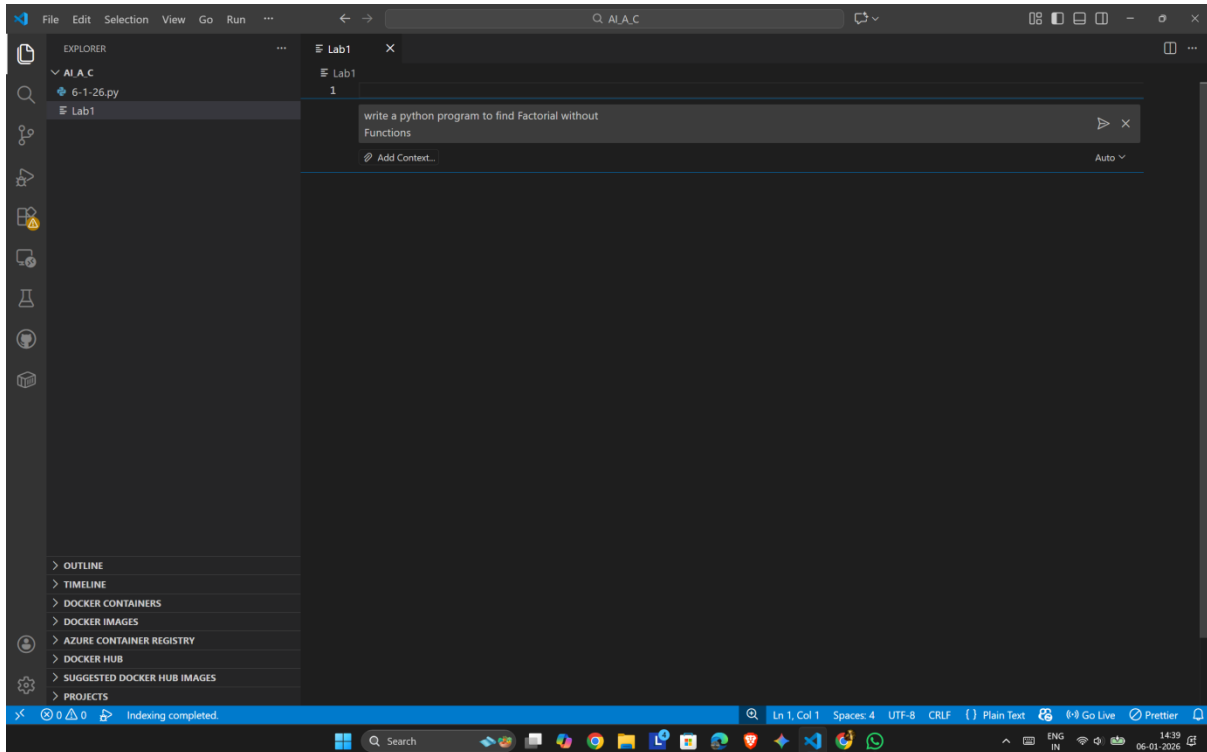
TASK - 0

- Install “Github Copilot” in Visual Studio Code

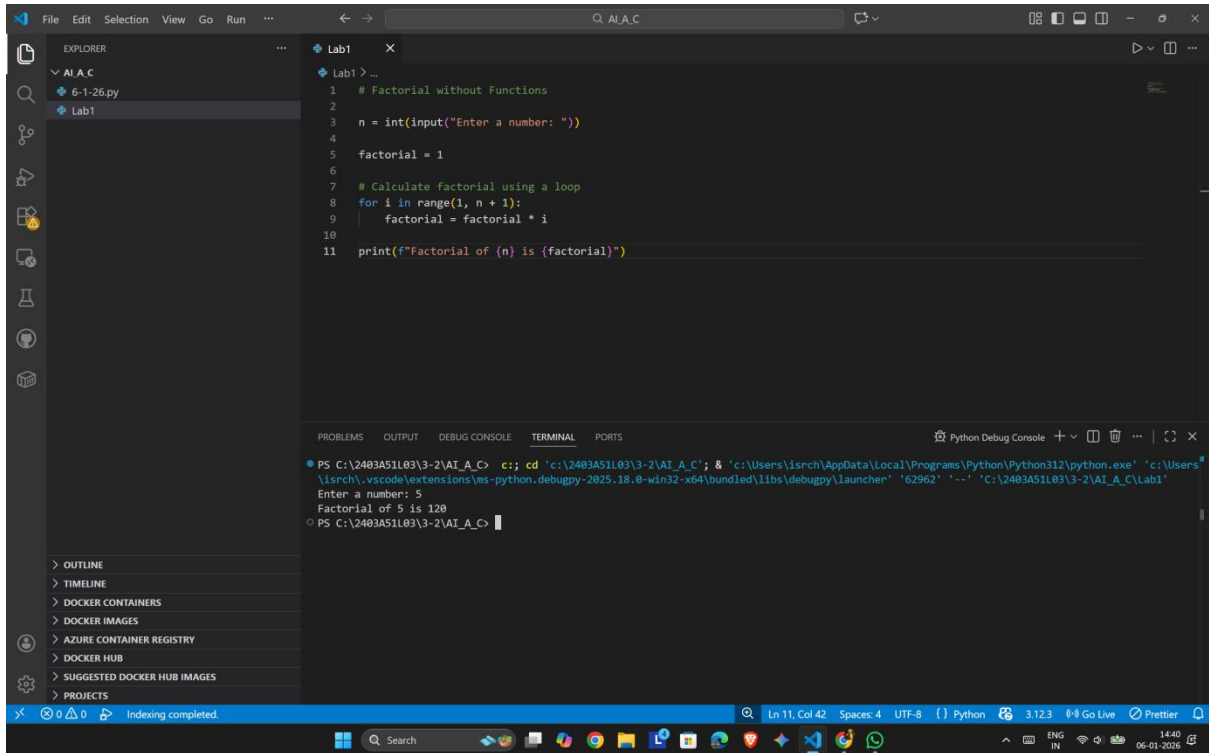


- Task Description

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.



OUTPUT:



The screenshot shows a Visual Studio Code editor window with a Python file named 'Lab1.py' open. The code in the file is as follows:

```
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 at the bottom shows the execution of the script:

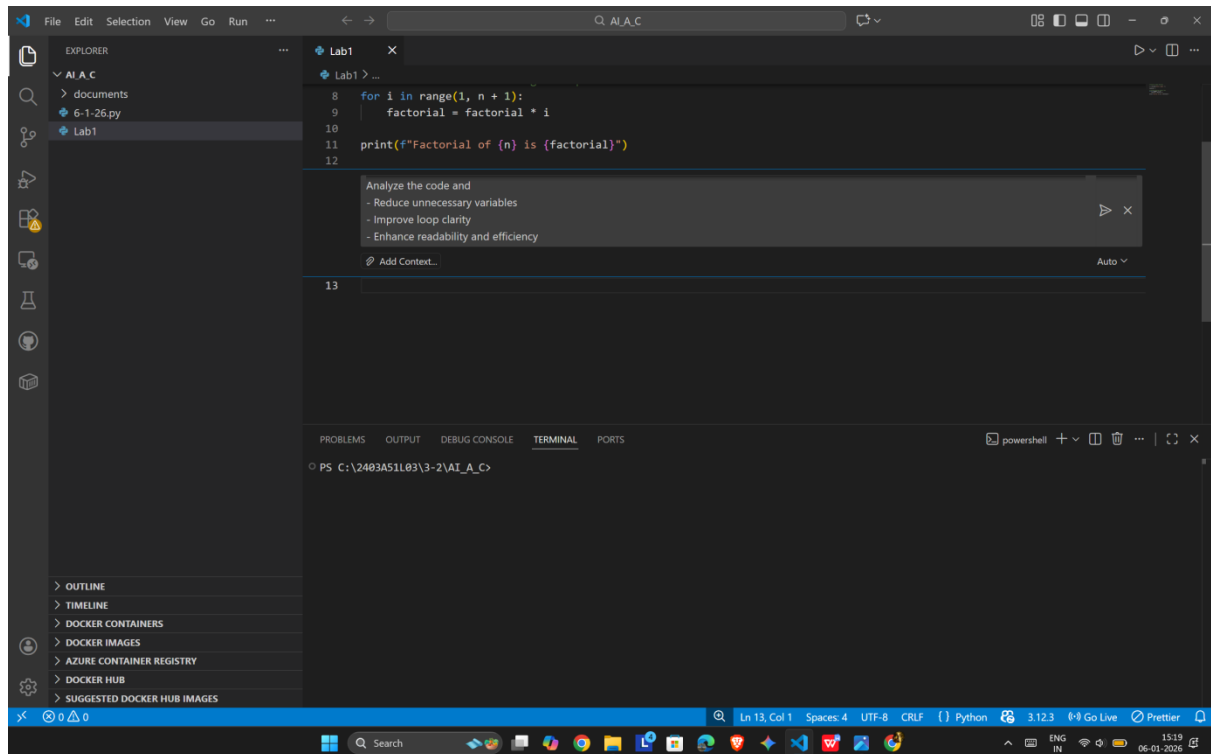
```
PS C:\2403A51103\3-2\AI_A_C> c:: cd 'c:\2403A51103\3-2\AI_A_C'; & 'c:\Users\isrch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\isrch\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '62962' '--' 'c:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 5
Factorial of 5 is 120
PS C:\2403A51103\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



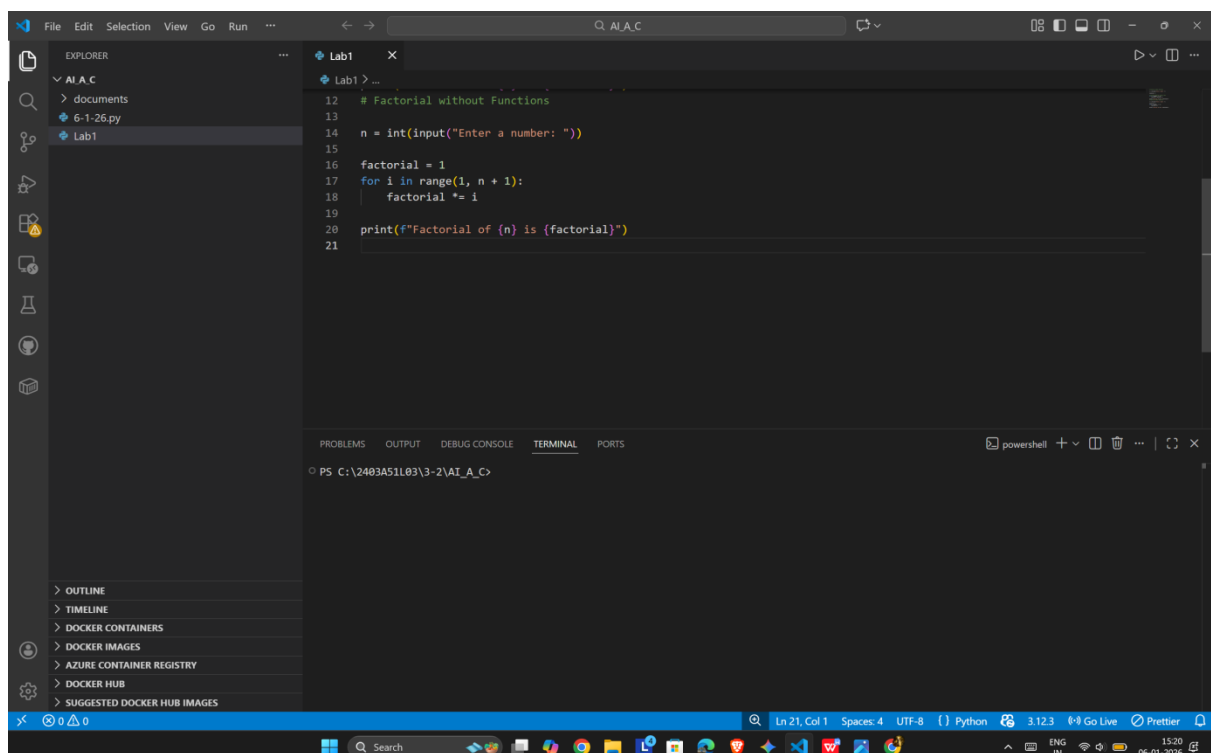
The screenshot shows the Visual Studio Code editor with a file named 'Lab1.py' open. The code in the editor is a simple factorial function:

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

A Copilot suggestion box is visible, displaying the following text:

```
Analyze the code and
- Reduce unnecessary variables
- Improve loop clarity
- Enhance readability and efficiency
Add Context... Auto
```

The bottom of the screen shows the terminal with the command 'PS C:\2403A51103\3-2\AI_A_C>'.



The screenshot shows the Visual Studio Code editor with a file named 'Lab1.py' open. The code in the editor is a more complex factorial function:

```
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 bottom of the screen shows the terminal with the command 'PS C:\2403A51103\3-2\AI_A_C>'.

The screenshot shows the VS Code editor with a Python file named 'Lab1'. The code calculates the factorial of a number using a for loop. The terminal output shows the program running and calculating the factorial of 5, which is 120.

```

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

```

Terminal Output:

```

PS C:\2403A51103\3-2\AI_A_C>
PS C:\2403A51103\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\bundle\libs\debugpy\launcher' '62973' '--' 'c:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720
PS C:\2403A51103\3-2\AI_A_C> c:: cd 'c:\2403A51103\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\bundle\libs\debugpy\launcher' '51942' '--' 'c:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 5
Factorial of 5 is 120
PS C:\2403A51103\3-2\AI_A_C>

```

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

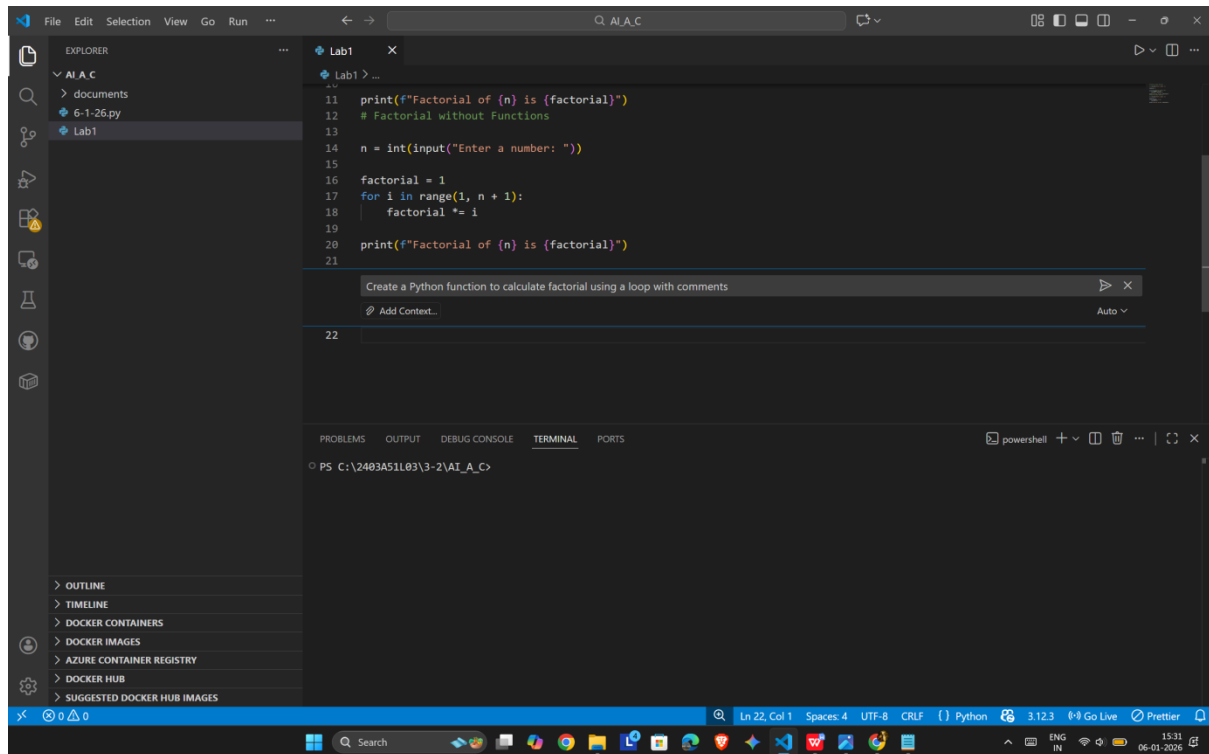
- Performance is effectively the same.

`*=` 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



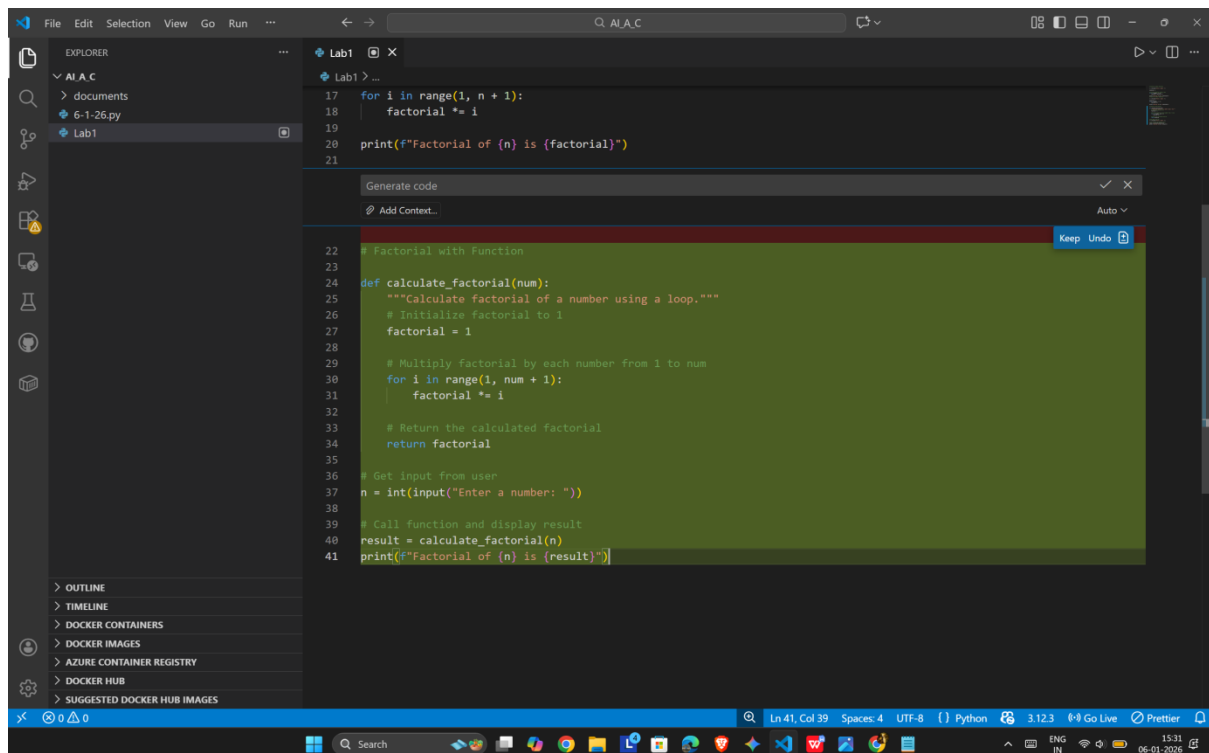
```

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...



```

17 for i in range(1, n + 1):
18     factorial *= i
19
20 print(f"Factorial of {n} is {factorial}")
21

```

Generate code

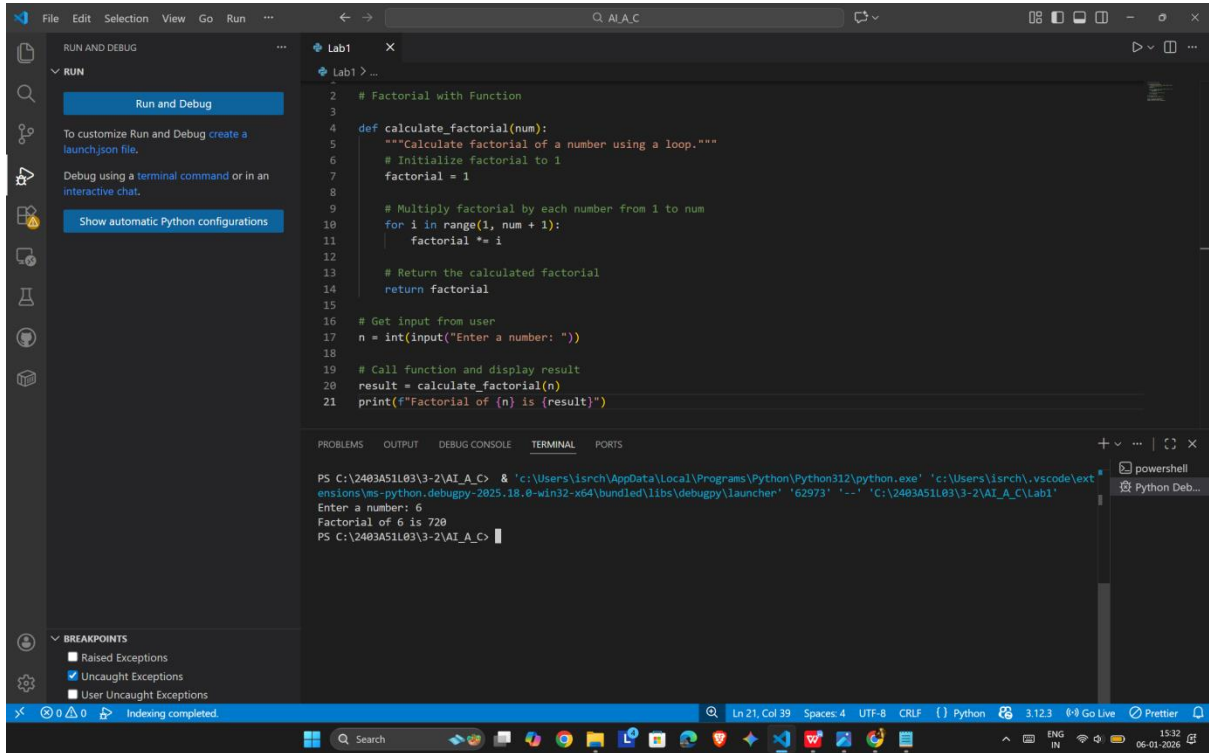
Add Context...

Keep Undo

```

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 Visual Studio Code editor with a Python file named 'Lab1'. The code defines a function 'calculate_factorial' that calculates the factorial of a number using a loop. It then prompts the user for input and prints the result. The terminal output shows the program running successfully, with the input '6' and the output 'Factorial of 6 is 720'.

```
1 # Factorial with Function
2
3
4 def calculate_factorial(num):
5     """Calculate factorial of a number using a loop."""
6     # Initialize factorial to 1
7     factorial = 1
8
9     # Multiply factorial by each number from 1 to num
10    for i in range(1, num + 1):
11        factorial *= i
12
13    # Return the calculated factorial
14    return factorial
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}")
```

Terminal Output:

```
PS C:\2403A51103\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\libs\debugpy\launcher' '62973' '--' 'C:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720
PS C:\2403A51103\3-2\AI_A_C>
```

- 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 a Visual Studio Code editor with a Python file named 'Lab1'. The code defines a function 'calculate_factorial' and uses it to calculate the factorial of a user input. A comparison criteria list is displayed in a box, and the terminal shows the program's execution for inputs 6 and 5.

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

Compare the non-function and function-based programs on the following criteria:

- Logic clarity
- Reusability
- Debugging ease

Terminal Output:

```
PS C:\2403A51103\3-2\AI_A_C>
PS C:\2403A51103\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\bundle\libs\debugpy\launcher' '62973' '-.' 'C:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 6
Factorial of 6 is 720 ...
PS C:\2403A51103\3-2\AI_A_C> c:: cd 'c:\2403A51103\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\bundle\libs\debugpy\launcher' '51942' '-.' 'C:\2403A51103\3-2\AI_A_C\Lab1'
Enter a number: 5
Factorial of 5 is 120
PS C:\2403A51103\3-2\AI_A_C>
```



```

31
32 # Comparison of Non-Function vs Function-Based Factorial Programs
33
34 print("\n=== COMPARISON ===\n")
35
36 comparison = {
37     "Logic Clarity": {
38         "Non-Function": "Direct and simple, but logic is mixed with I/O",
39         "Function-Based": "Clearer separation of concerns; logic isolated in function"
40     },
41     "Reusability": {
42         "Non-Function": "Cannot reuse; must rewrite code for each use",
43         "Function-Based": "Highly reusable; call function multiple times"
44     },
45     "Debugging Ease": {
46         "Non-Function": "Harder; must debug entire script together",
47         "Function-Based": "Easier; test function independently in isolation"
48     },
49     "Large Projects": {
50         "Non-Function": "Not suitable; leads to code duplication and maintenance issues",
51         "Function-Based": "Suitable; promotes modularity and organization"
52     },
53     "AI Dependency Risk": {
54         "Non-Function": "Lower risk; simple, self-contained logic",
55         "Function-Based": "Higher risk if function relies on external AI; more dependencies"
56     }
57 }
58
59 for criterion, comparison_data in comparison.items():
60     print(f"{criterion}:")
61     for approach, description in comparison_data.items():
62         print(f"    {approach}: {description}")
63     print()
  
```

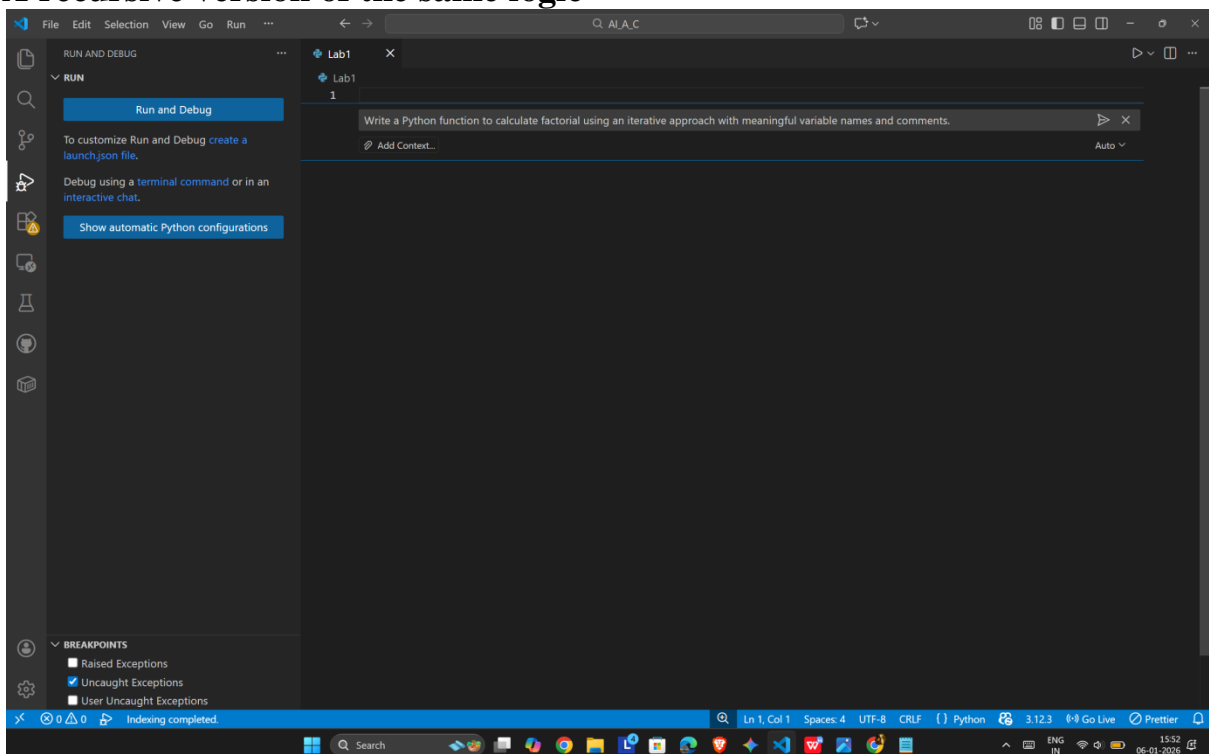
TASK - 5

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 editor with a Python file named `AL_A_C`. The code implements a function `calculate_factorial(n)` using iteration. The left sidebar shows the 'RUN AND DEBUG' panel with the 'Run and Debug' button highlighted. The bottom status bar indicates 'Ln 33, Col 52', 'Spaces: 4', 'UTF-8', and 'Python 3.12.3'.

```

1 def calculate_factorial(n):
2     """
3     Calculate the factorial of a non-negative integer using iteration.
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 is 1
16    if n == 0 or n == 1:
17        return 1
18
19    # Initialize result to 1
20    factorial_result = 1
21
22    # Multiply result by each number from 2 to n
23    for current_number in range(2, n + 1):
24        factorial_result *= current_number
25
26    return factorial_result
27
28 # Example usage
29 if __name__ == "__main__":
30     test_value = 5
31     result = calculate_factorial(test_value)
32     print(f"Factorial of {test_value} is {result}")
33

```

The screenshot shows the Visual Studio Code editor with a Python file named `AL_A_C`. The code implements a function `calculate_factorial(n)` using recursion. The left sidebar shows the 'RUN AND DEBUG' panel with the 'Run and Debug' button highlighted. The bottom status bar indicates 'Ln 36, Col 1', 'Spaces: 4', 'UTF-8', and 'Python 3.12.3'.

```

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    # Base case: factorial of 0 is 1
12    if n == 0 or n == 1:
13        return 1
14
15    # Recursive case: n * factorial(n-1)
16    return n * calculate_factorial(n - 1)
17
18 # Example usage
19 if __name__ == "__main__":
20     test_value = 5
21     result = calculate_factorial(test_value)
22     print(f"Factorial of {test_value} is {result}")
23
24
25
26
27
28
29
30
31
32
33
34
35
36

```

The screenshot shows the Visual Studio Code editor with a Python file named 'Lab1'. The code defines a recursive function 'calculate_factorial_recursive(n)' and includes a main block to test it with the value 5. The function includes docstrings, argument descriptions, and base/recursive cases.

```

33 print(f"Factorial of {test_value} is {result}")
34
35
36 def calculate_factorial_recursive(n):
37     """
38     Calculate the factorial of a non-negative integer using recursion.
39
40     Args:
41         n: A non-negative integer
42
43     Returns:
44         The factorial of n
45     """
46     # Validate input
47     if n < 0:
48         raise ValueError("Factorial is not defined for negative numbers")
49
50     # Base case: factorial of 0 or 1 is 1
51     if n == 0 or n == 1:
52         return 1
53
54     # Recursive case: n! = n * (n-1)!
55     return n * calculate_factorial_recursive(n - 1)
56
57
58 # Example usage with recursive function
59 if __name__ == "__main__":
60     test_value = 5
61     result = calculate_factorial_recursive(test_value)
62     print(f"Factorial of {test_value} (recursive) is {result}")
  
```

OUTPUT:

The screenshot shows the same VS Code editor with the 'Lab1' file. The 'TERMINAL' panel at the bottom displays the output of the script, which is the factorial of 5 calculated recursively.

```

PS C:\2403A51103\3-2\AI_A_C> c:; cd 'c:\2403A51103\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' '61331' '-.' 'c:\2403A51103\3-2\AI_A_C\Lab1'
Factorial of 5 is 120
Factorial of 5 (recursive) is 120
PS C:\2403A51103\3-2\AI_A_C>
  
```

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.**