

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

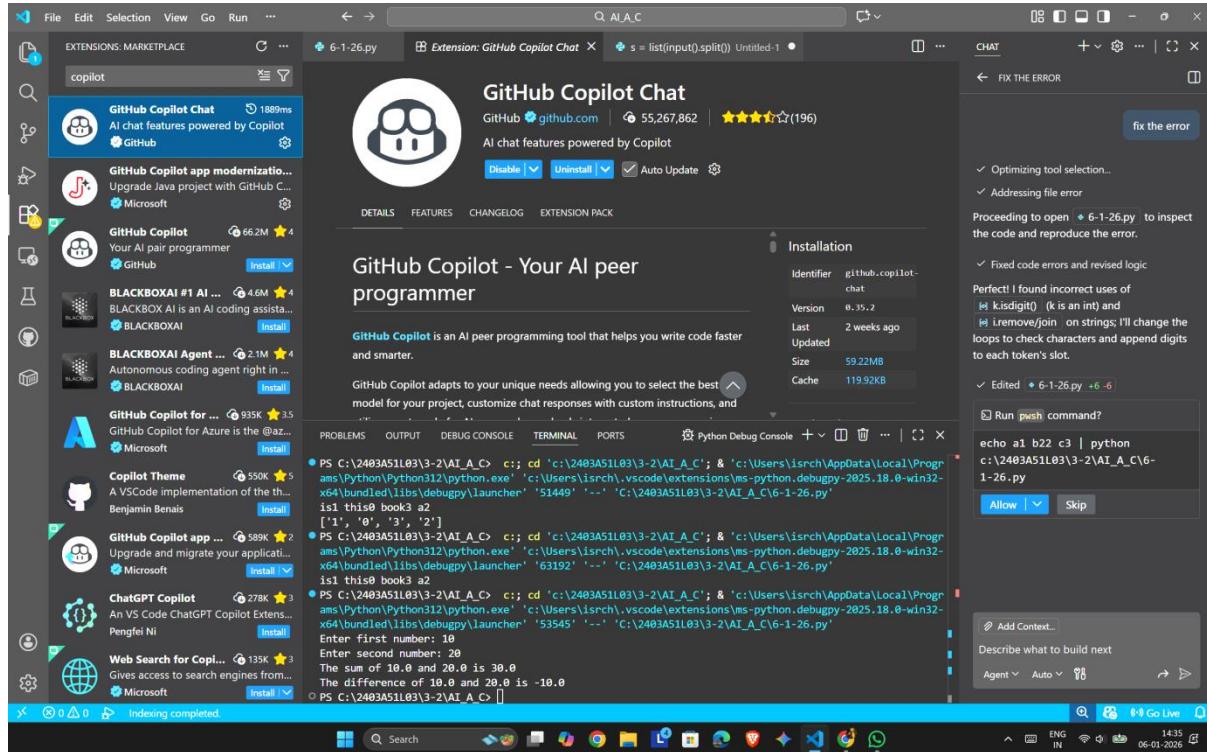
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: II
Academic Session	: 2025-2026
Name of Student	: J Harshith yadav
Enrollment No.	: 2403A51L49
Batch No.	: 52
Date	: 06/01/26

Submission Starts here

Screenshots:

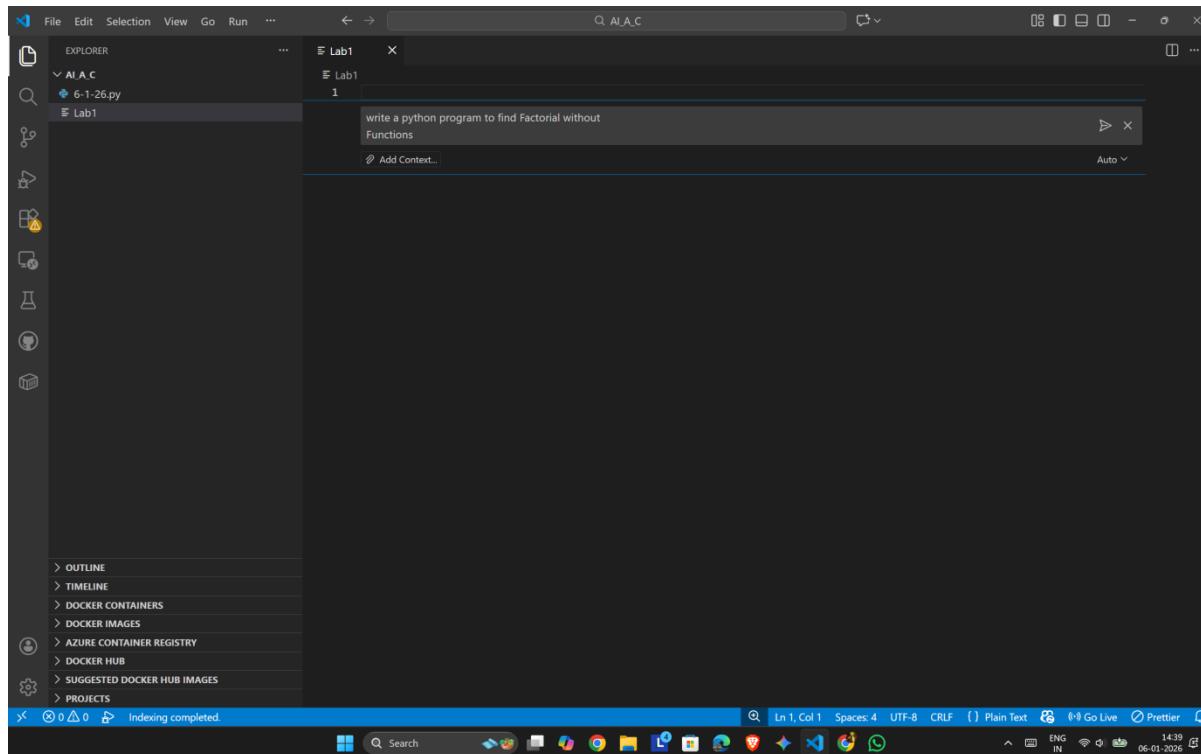
TASK - 0

- 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.



File Edit Selection View Go Run ...

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

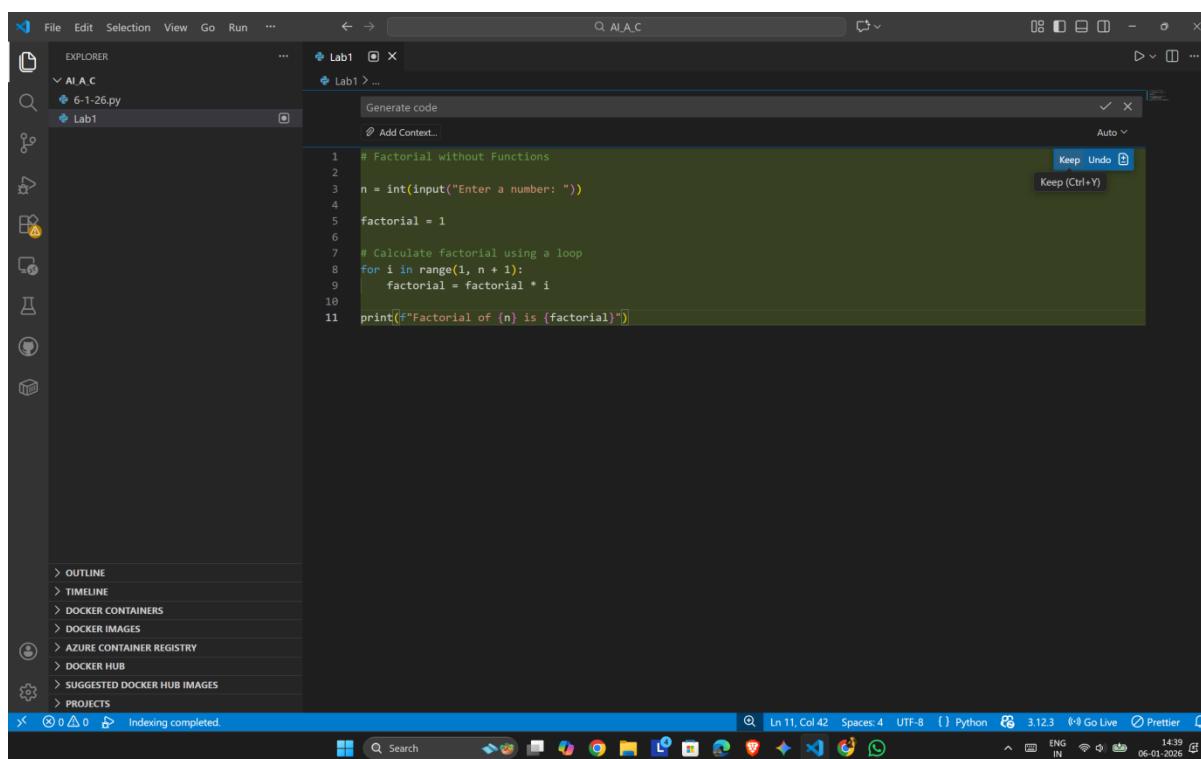
Suggested Docker Hub Images

Projects

Indexing completed.

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

14:39 ENG IN 06-01-2026



```
# 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}")
```

File Edit Selection View Go Run ...

AI_A_C

6-1-26.py

Lab1

Generate code

Add Context...

Keep Undo

Keep (Ctrl+Y)

OUTLINE

Timeline

Docker Containers

Docker Images

Azure Container Registry

Docker Hub

Suggested Docker Hub Images

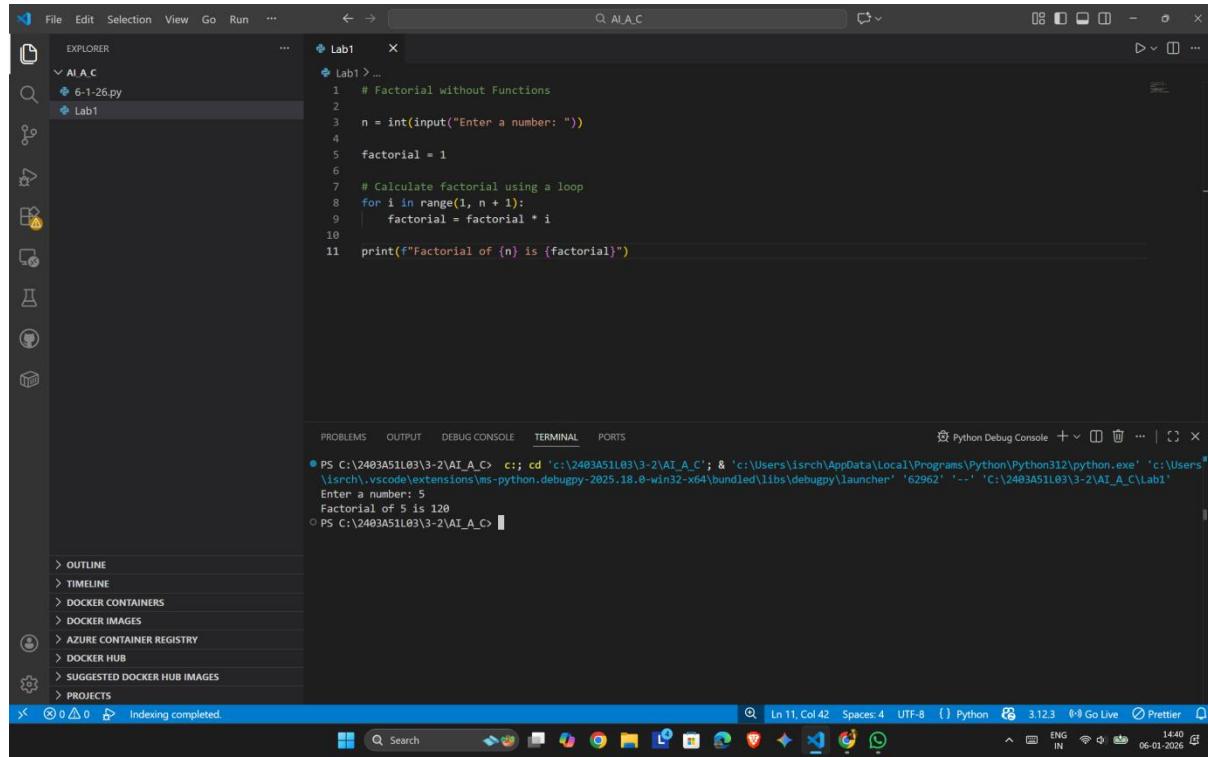
Projects

Indexing completed.

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

14:39 ENG IN 06-01-2026

OUTPUT:



The screenshot shows the Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with a project named 'AI_A_C' containing files '6-1-26.py' and 'Lab1'. The main editor area displays a Python script named 'Lab1.py' with the following code:

```
# Factorial without Functions
n = int(input("Enter a number: "))
factorial = 1
# Calculate factorial using a loop
for i in range(1, n + 1):
    factorial = factorial * i
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> 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' '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 with the following details:

- File Explorer:** Shows a folder named "AI_A_C" containing "documents" and "Lab1". Inside "Lab1" are files "6-1-26.py" and "Lab1.py".
- Code Editor:** Displays Python code for calculating factorial:

```

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

```
- Quick Fix Suggestion:** A tooltip appears over the code editor with the following text:
 - Analyze the code and
 - Reduce unnecessary variables
 - Improve loop clarity
 - Enhance readability and efficiency
- Terminal:** Shows the command "PS C:\2403A51L03\3-2\AI_A_C>".
- Bottom Status Bar:** Includes icons for search, file operations, and system status like battery level and date/time (06-01-2026).

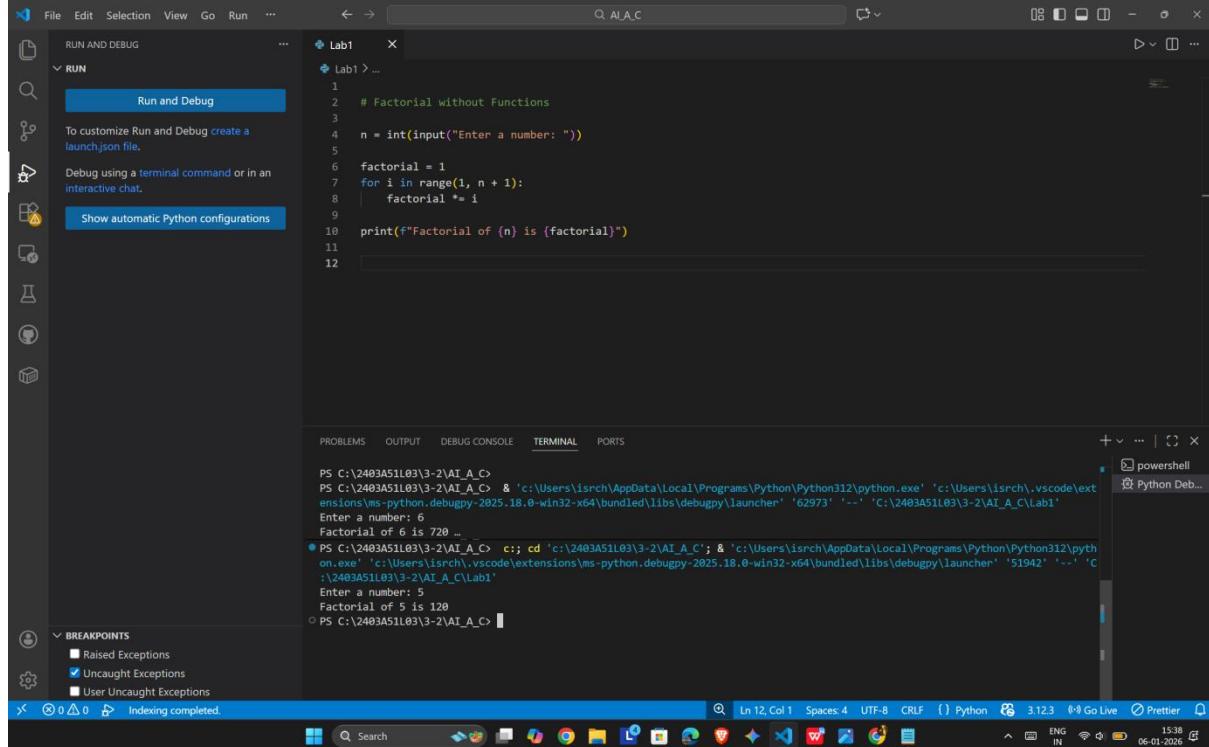
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Same structure as the first screenshot.
- Code Editor:** Displays the same factorial code, but now annotated with comments and a docstring:

```

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

```
- Terminal:** Shows the command "PS C:\2403A51L03\3-2\AI_A_C>".
- Bottom Status Bar:** Includes icons for search, file operations, and system status like battery level and date/time (06-01-2026).



The screenshot shows the Visual Studio Code interface with a dark theme. In the center, there's a code editor window titled 'Lab1' containing the following Python code:

```

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

```

Below the code editor is a terminal window showing the execution of the script. It prints 'Factorial of 6 is 720' and 'Factorial of 5 is 120'. The terminal also displays the command used to run the script: 'python.exe' followed by the path to the script file.

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

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

- File Explorer:** Shows a folder named "AI_A_C" containing "documents", "6-1-26.py", and "Lab1".
- Code Editor:** The "Lab1" tab is active, displaying a Python script:

```
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}")
```
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. It also shows the command prompt: PS C:\2403A51L03\3-2\AI_A_C>.
- Sidebar:** Contains sections for OUTLINE, TIMELINE, DOCKER CONTAINERS, DOCKER IMAGES, AZURE CONTAINER REGISTRY, DOCKER HUB, and SUGGESTED DOCKER HUB IMAGES.
- Status Bar:** Shows file statistics (Ln 22, Col 1), code styling (Spaces: 4, CRLF), language (Python 3.12.3), and other tools like Go Live and Prettier.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, and others. The left sidebar has sections for Explorer, Outline, Timeline, Docker Containers, Docker Images, Azure Container Registry, Docker Hub, and Suggested Docker Hub Images. The main workspace shows a file named 'Lab1.py' with the following Python code:

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

# Factorial with Function

def calculate_factorial(num):
    """Calculate factorial of a number using a loop."""
    # Initialize factorial to 1
    factorial = 1

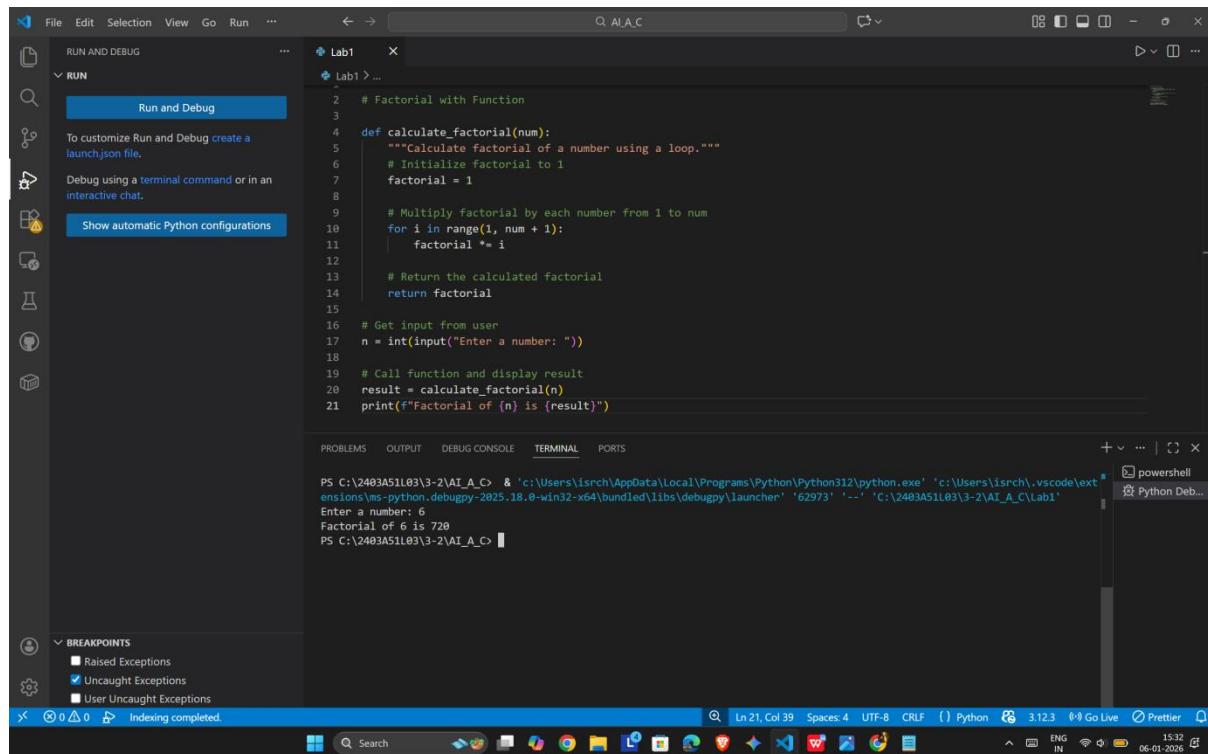
    # Multiply factorial by each number from 1 to num
    for i in range(1, num + 1):
        factorial *= i

    # 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}")
```

The status bar at the bottom shows 'Ln 41, Col 39 Spaces: 4 UTF-8 Python 3.12.3 Go Live'. There are also icons for search, file operations, and system status.



```

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\israch\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\israch\.vscode\extensions\ms-python.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 Visual Studio Code (VS Code) interface. The left sidebar has icons for file operations, run/debug, search, and other development tools. The main area displays a Python script named `Lab1.py` with code to calculate factorials:

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

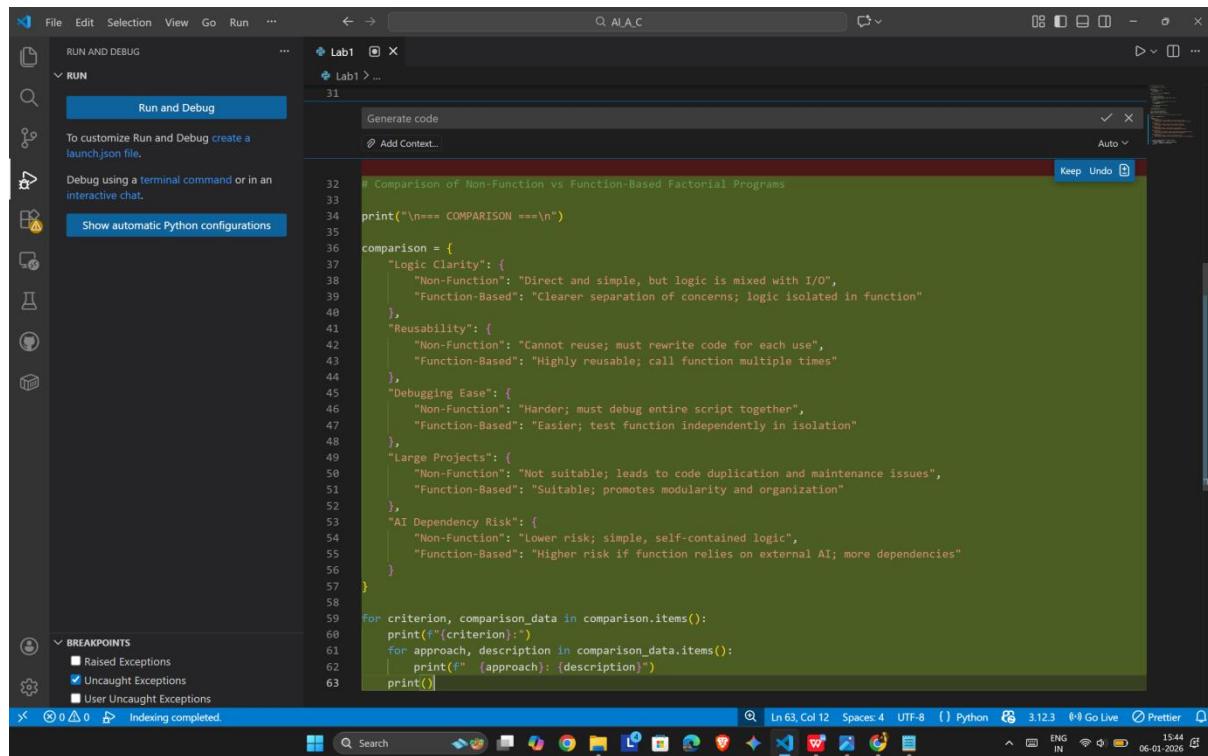
# Get input from user
n = int(input("Enter a number: "))

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

Below the code editor is a tooltip comparing non-function and function-based programs based on logic clarity, reusability, and debugging ease. The bottom part of the screen 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\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' '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: & 'c:\Users\lsrch\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 completed and shows system information like battery level, signal strength, and date.



```

# 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"
    }
}

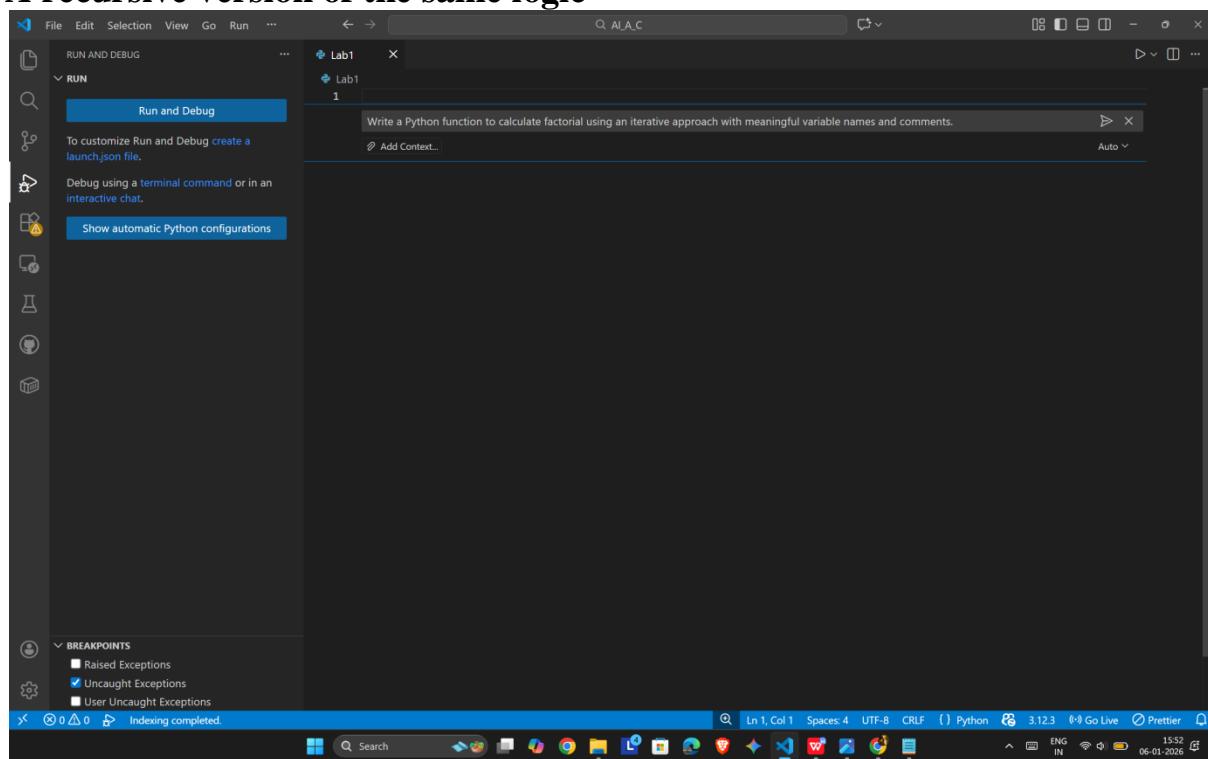
for criterion, comparison_data in comparison.items():
    print(f"\n{criterion}:")
    for approach, description in comparison_data.items():
        print(f"  {approach}: {description}")
    print()

```

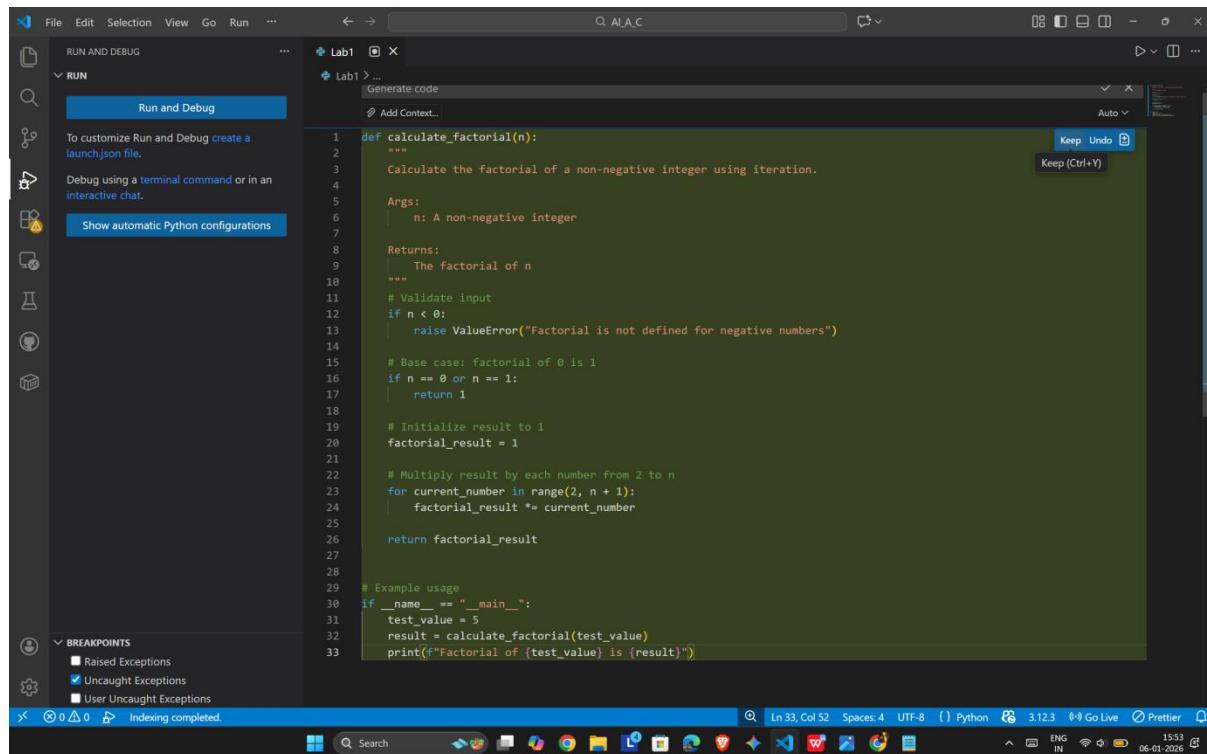
TASK - 5

Task Description

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



Write a Python function to calculate factorial using an iterative approach with meaningful variable names and comments.



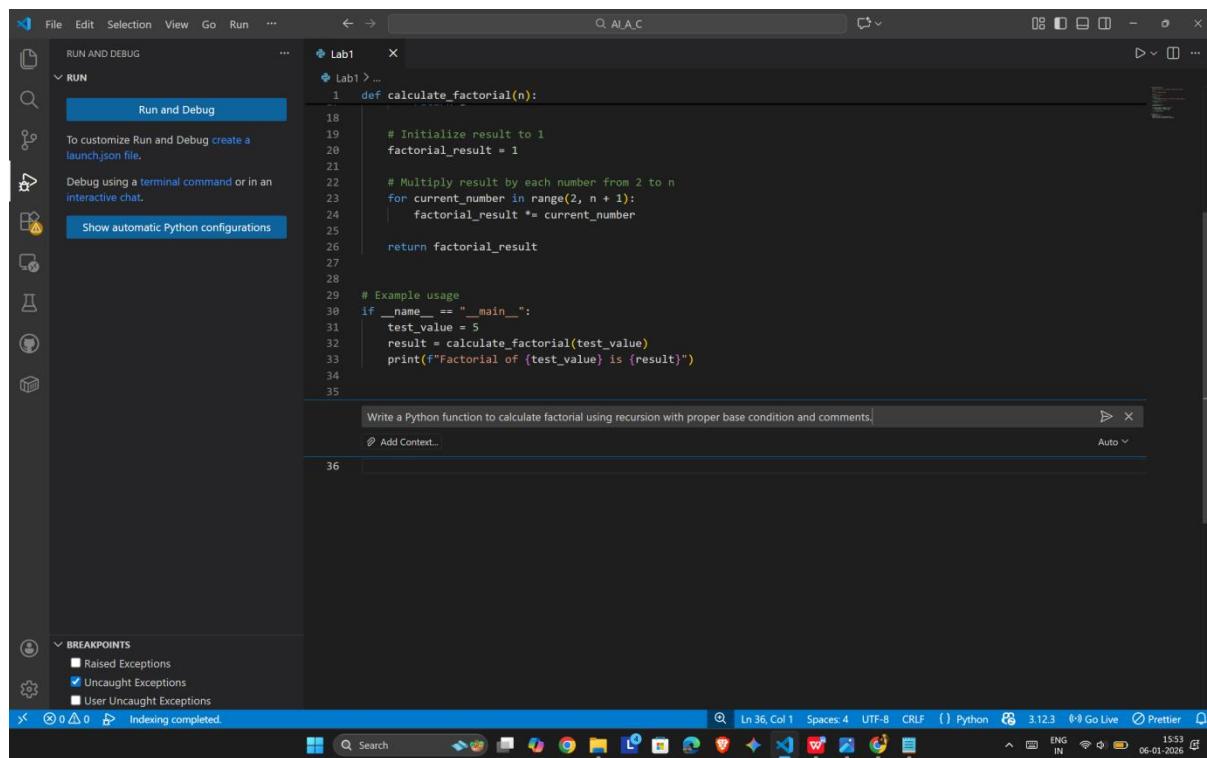
The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The title bar says "Q ALA.C". The left sidebar has sections for "RUN AND DEBUG" and "RUN". Under "RUN", there's a "Run and Debug" button, a note about creating a launch.json file, and a link to "Show automatic Python configurations". The main editor area contains the following Python code:

```

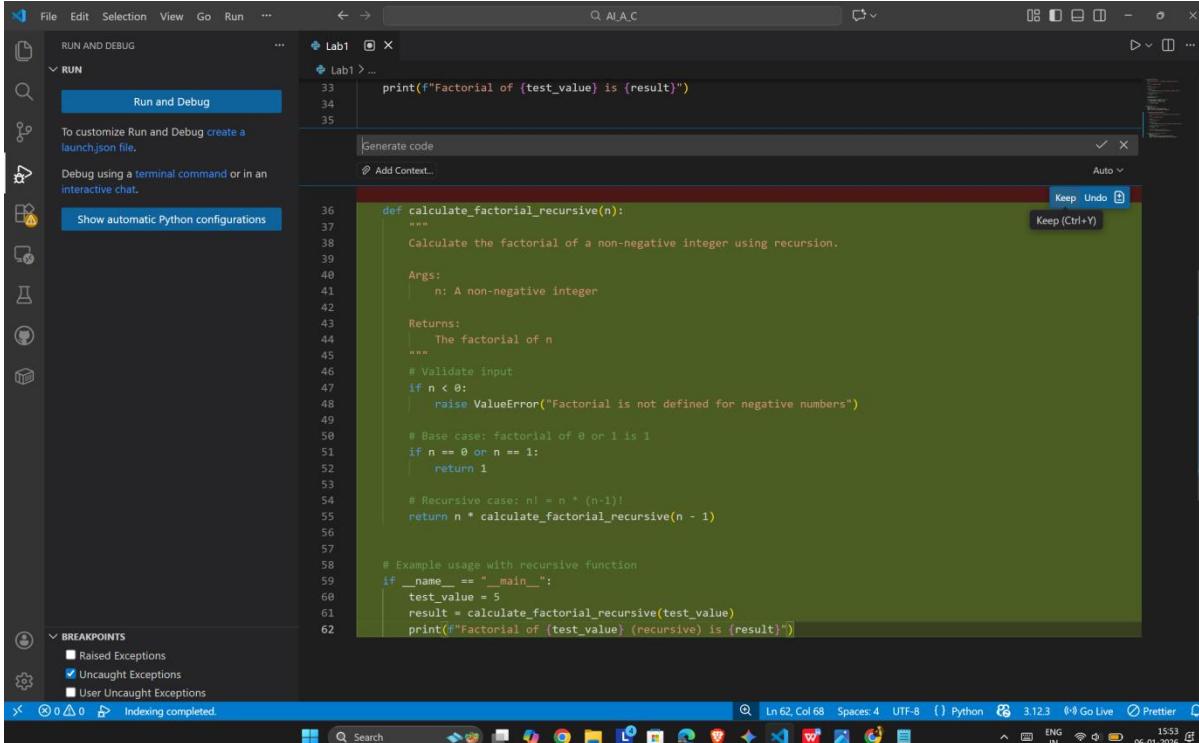
1 def calculate_factorial(n):
2     """
3         Calculate the factorial of a non-negative integer using iteration.
4     """
5     Args:
6         n: A non-negative integer
7
8     Returns:
9         The factorial of n
10    """
11    # Validate input
12    if n < 0:
13        raise ValueError("Factorial 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
29 # Example usage
30 if __name__ == "__main__":
31     test_value = 5
32     result = calculate_factorial(test_value)
33     print(f"Factorial of {test_value} is {result}")

```

The status bar at the bottom shows "Indexing completed.", "Ln 33, Col 52", "Spaces: 4", "UTF-8", "Python 3.12.3", "Go Live", "Prettier", and a timestamp "06-01-2026".



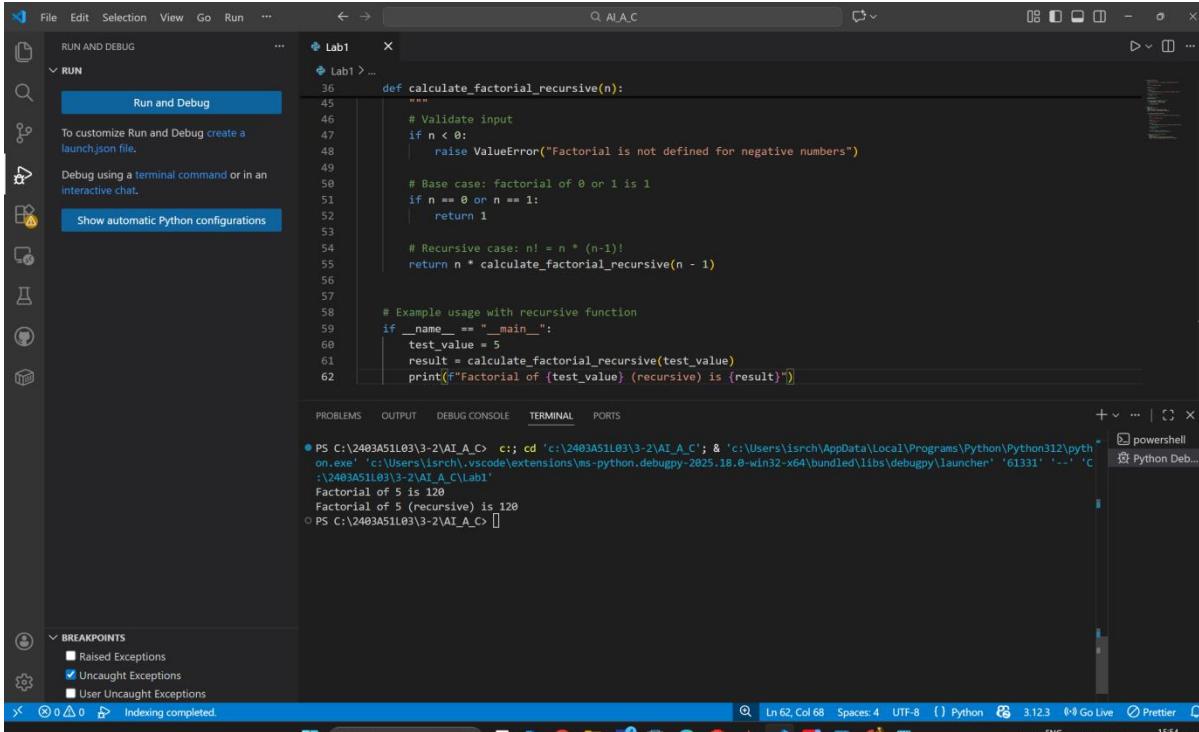
This screenshot is identical to the one above, showing the same Python code for calculating factorial using iteration. The main difference is in the status bar, which now says "Ln 36, Col 1", "Spaces: 4", "CRLF", and "15:53".



The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a folder named "RUN AND DEBUG" containing a "RUN" folder.
- Code Editor:** A file named "Lab1.py" is open, containing Python code for calculating factorial using recursion. The code includes comments explaining the logic and handles negative numbers.
- Terminal:** Shows the command "Indexing completed."
- Status Bar:** Displays "Ln 62, Col 68" and "Python 3.12.3".

OUTPUT:



The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a folder named "RUN AND DEBUG" containing a "RUN" folder.
- Code Editor:** A file named "Lab1.py" is open, containing Python code for calculating factorial using recursion. The code includes comments explaining the logic and handles negative numbers.
- Terminal:** Shows the command "Indexing completed." followed by the execution of the script. The output shows the factorial of 5 being calculated and printed as 120.
- Status Bar:** Displays "Ln 62, Col 68" and "Python 3.12.3".

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.