ASSIGNMENT-1.2

HT.NO:2303A51556

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**	
Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026	
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Rapelly Nandini Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week1 - Tuesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:1.2(Present assignment number)/**24**(Total number of assignments)			

Q.No.	**Question**	***Expected Time to complete***
1	Lab 1: Environment Setup – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow*	Week1 - Monday

Lab Objectives:

- To install and configure GitHub Copilot in Visual Studio Code.
- To explore AI-assisted code generation using GitHub Copilot.
- To analyze the accuracy and effectiveness of Copilot's code suggestions.
- To understand prompt-based programming using comments and code context

Lab Outcomes (LOs):
After completing this lab, students will be able to:

- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

Task 0

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Expected Output

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

- **Scenario**

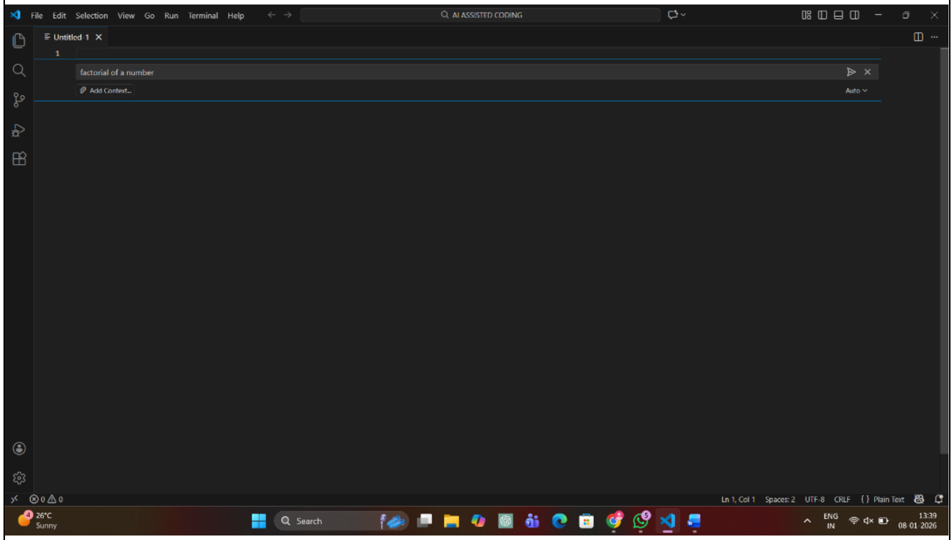
You are building a **small command-line utility** for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

- **Task Description**

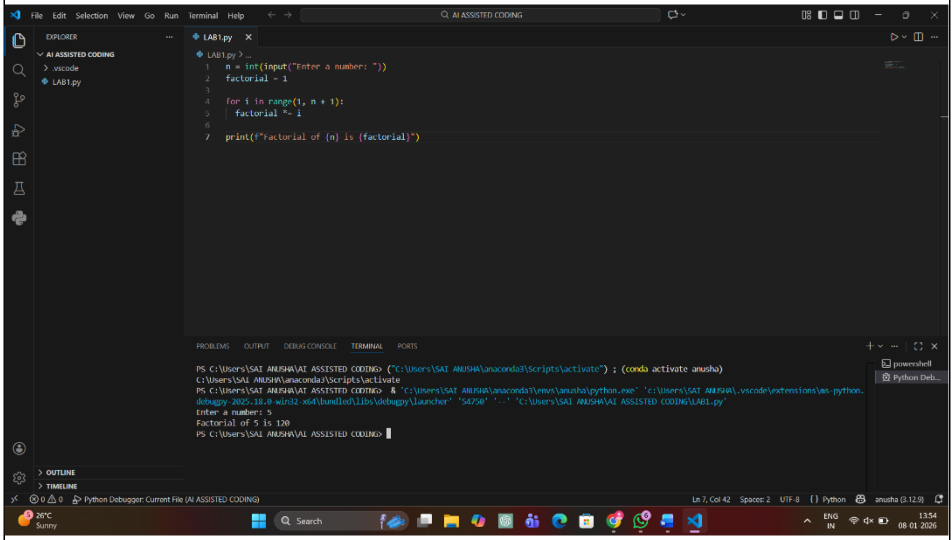
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- **Constraint:**
- ➢ Do not define any custom function
- ➢ Logic must be implemented using loops and variables only

- **Expected Deliverables**
- ➢ A working Python program generated with Copilot assistance
- ➢ Screenshot(s) showing:
- ➢ The prompt you typed
- ➢ Copilot's suggestions
- ➢ Sample input/output screenshots
- ➢ Brief reflection (5–6 lines):
- ➢ How helpful was Copilot for a beginner?
- ➢ Did it follow best practices automatically?



factorial of a number



```python
n = int(input("Enter a number: "))
factorial = 1

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

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

```
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> ("C:\Users\SAI ANUSHA\anaconda3\Scripts\activate") ; (conda activate anusha)
C:\Users\SAI ANUSHA\anaconda3\Scripts\activate
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING>  & 'C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe' 'c:\Users\SAI ANUSHA\.vscode\extensions\ms-python.
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54750' '--' 'C:\Users\SAI ANUSHA\AI ASSISTED CODING\LAB1.py'
Enter a number: 5
Factorial of 5 is 120
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> 
```

EXPLANATION:-

This program calculates the factorial of a number entered by the user.

If the number is negative, it prints that factorial is not defined; if the number is 0 or 1, it prints 1, otherwise it uses a loop to multiply numbers from 2 to the given number and prints the final factorial

---

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

❖ **Scenario**

Your team lead asks you to **review AI-generated code** before committing it to a shared repository.

❖ **Task Description**

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

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

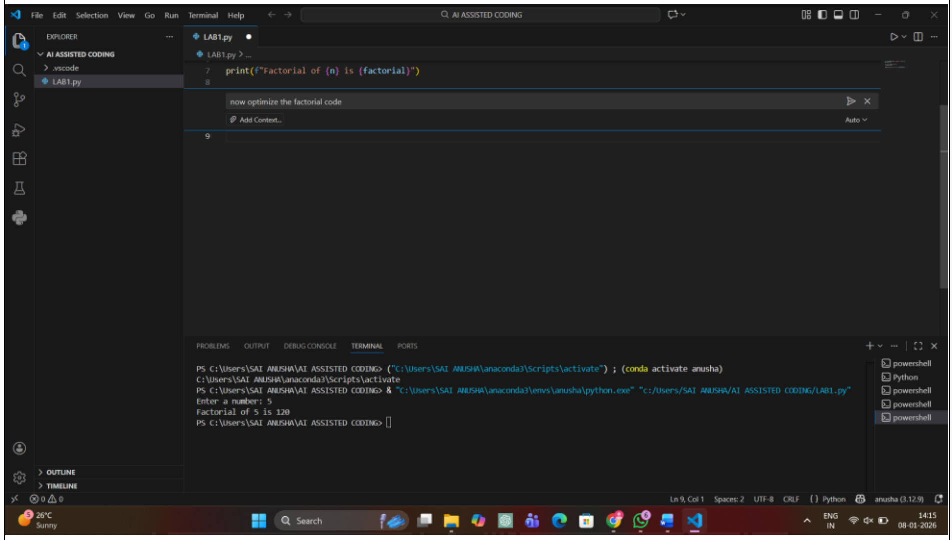
Hint:

Prompt Copilot with phrases like

"optimize this code", *"simplify logic"*, or *"make it more readable"*

❖ **Expected Deliverables**

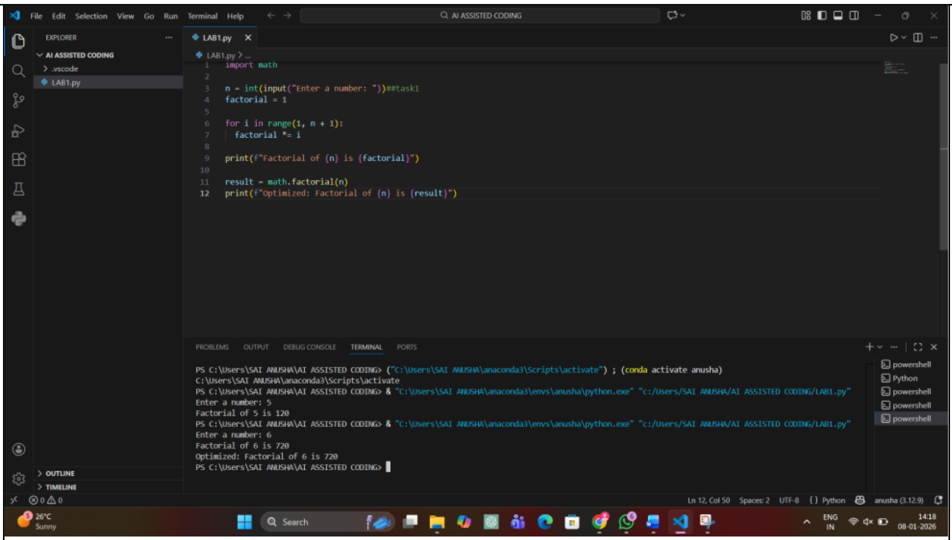
- ➢ Original AI-generated code
- ➢ Optimized version of the same code
- ➢ Side-by-side comparison
- ➢ Written explanation:
  - ▪ What was improved?
  - ▪ Why the new version is better (readability, performance, maintainability.



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

now optimize the factorial code

```
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> ("C:\Users\SAI ANUSHA\anaconda3\Scripts\activate") ; (conda activate anusha)
C:\Users\SAI ANUSHA\anaconda3\Scripts\activate
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/LAB1.py"
Enter a number: 5
Factorial of 5 is 120
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> []
```



```python
import math

n = int(input("Enter a number: "))##task1
factorial = 1

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

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

result = math.factorial(n)
print(f"Optimized: Factorial of {n} is {result}")
```

```
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> ("C:\Users\SAI ANUSHA\anaconda3\Scripts\activate") ; (conda activate anusha)
C:\Users\SAI ANUSHA\anaconda3\Scripts\activate
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/LAB1.py"
Enter a number: 5
Factorial of 5 is 120
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/LAB1.py"
Enter a number: 6
Factorial of 6 is 720
Optimized: Factorial of 6 is 720
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING>
```

EXPLANATION:-

The optimized version removes unnecessary conditions while keeping the logic correct. This makes the code shorter and easier to read, which helps in understanding and maintaining it. Even after simplification, the program produces the same output and runs with the same performance as the original version.

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

❖ **Scenario**

The same logic now needs to be reused in **multiple scripts**.

❖ **Task Description**

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

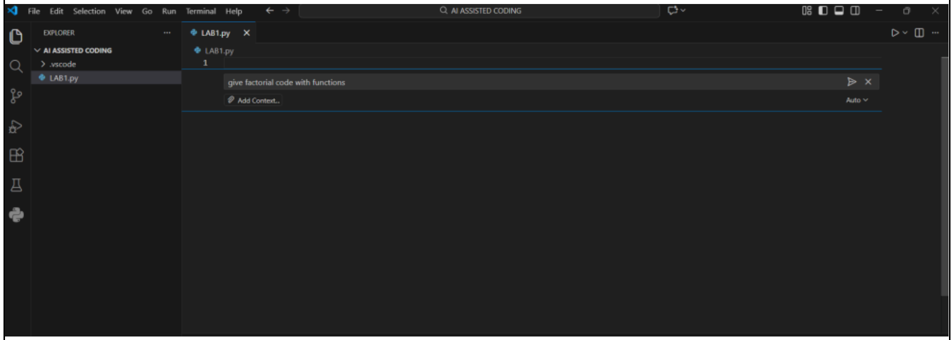
- ➢ Creating a **user-defined function**
- ➢ Calling the function from the main block

❖ **Constraints**

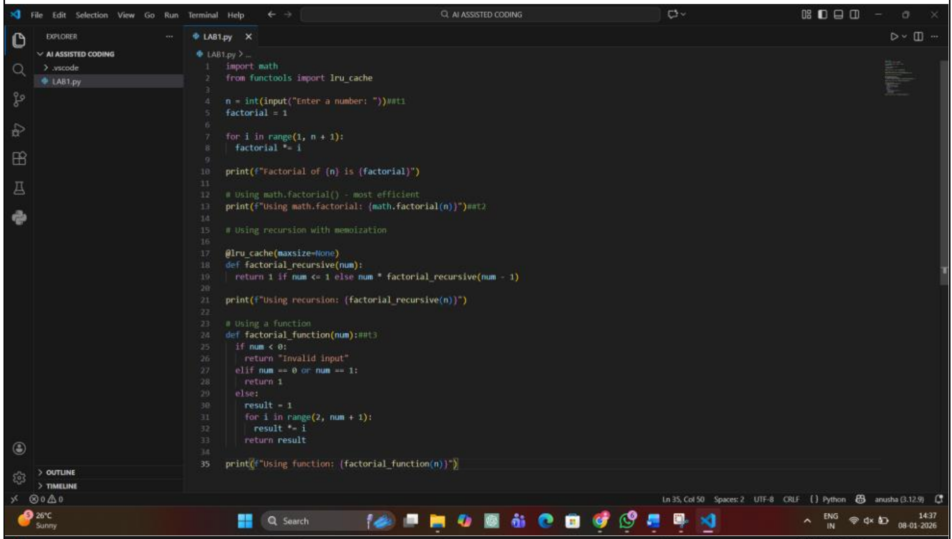
- ➢ Use meaningful function and variable names
- ➢ Include inline comments (preferably suggested by Copilot)

❖ **Expected Deliverables**

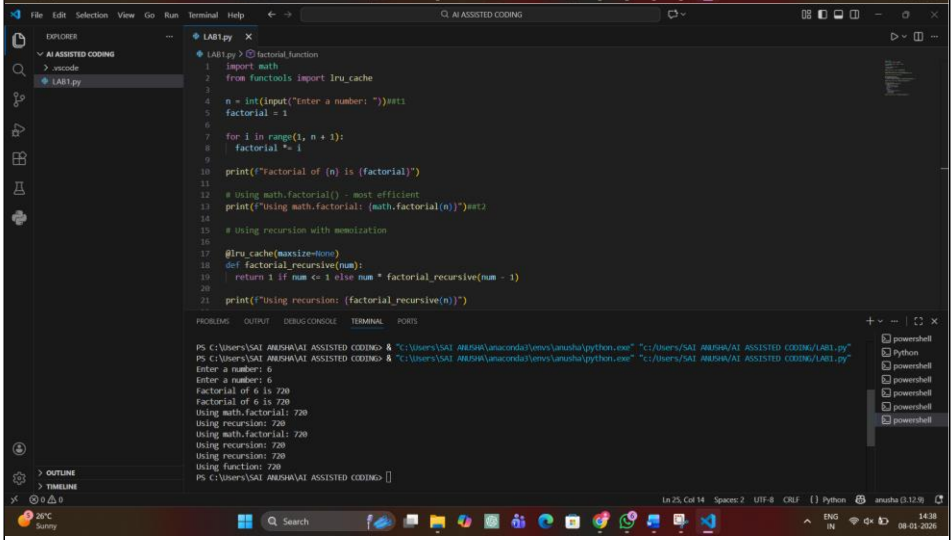
- ➢ AI-assisted function-based program
- ➢ Screenshots showing:
  - ○ Prompt evolution
  - ○ Copilot-generated function logic
- ➢ Sample inputs/outputs
- ➢ Short note:
  - ○ How modularity improves reusabil



give factorial code with functions



```python
import math
from functools import lru_cache

n = int(input("Enter a number: "))##t1
factorial = 1

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

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

# Using math.factorial() - most efficient
print(f"Using math.factorial: {math.factorial(n)}")##t2

# Using recursion with memoization

@lru_cache(maxsize=None)
def factorial_recursive(num):
  return 1 if num <= 1 else num * factorial_recursive(num - 1)

print(f"Using recursion: {factorial_recursive(n)}")

# Using a function
def factorial_function(num):##t3
  if num < 0:
    return "Invalid input"
  elif num == 0 or num == 1:
    return 1
  else:
    result = 1
    for i in range(2, num + 1):
      result *= i
    return result

print(f"Using function: {factorial_function(n)}")
```



```python
import math
from functools import lru_cache

n = int(input("Enter a number: "))##t1
factorial = 1

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

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

# Using math.factorial() - most efficient
print(f"Using math.factorial: {math.factorial(n)}")##t2

# Using recursion with memoization

@lru_cache(maxsize=None)
def factorial_recursive(num):
  return 1 if num <= 1 else num * factorial_recursive(num - 1)

print(f"Using recursion: {factorial_recursive(n)}")
```

```
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/LAB1.py"
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/LAB1.py"
Enter a number: 6
Enter a number: 6
Factorial of 6 is 720
Factorial of 6 is 720
Using math.factorial: 720
Using recursion: 720
Using math.factorial: 720
Using recursion: 720
Using recursion: 720
Using function: 720
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> 
```

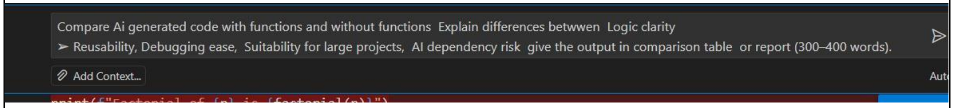
EXPLANATION:-
The improved version removes unnecessary conditions but keeps the logic intact. This makes the code shorter and easier to read, which aids in understanding and maintaining it. Even after this simplification, the program still produces the same output and runs at the same performance level as the original version.

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

- **Scenario**
  As part of a **code review meeting**, you are asked to justify design choices.

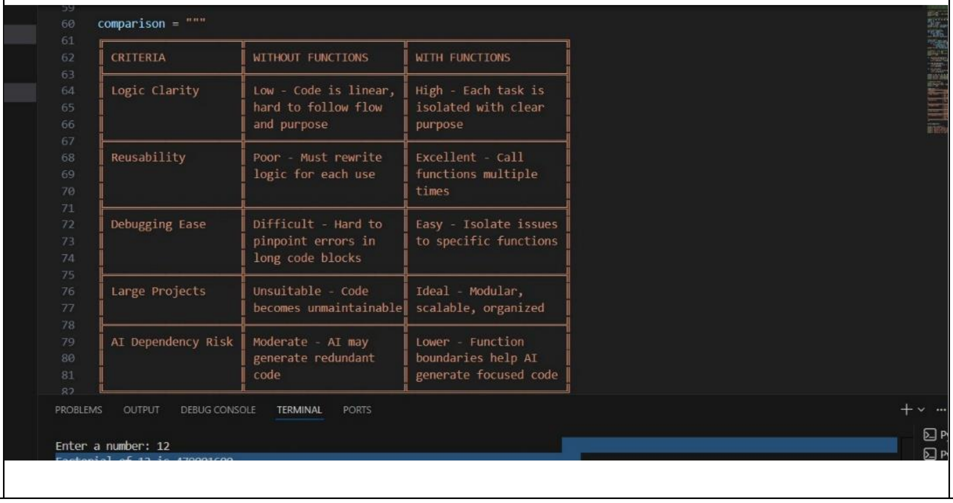
- **Task Description**
  Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:
  - Logic clarity
  - Reusability
  - Debugging ease
  - Suitability for large projects
  - AI dependency risk

- **Expected Deliverables**
  Choose **one**:
  - A comparison table
    OR
  - A short technical report (300–400 words).



Compare Ai generated code with functions and without functions Explain differences betwwen Logic clarity
➢ Reusability, Debugging ease, Suitability for large projects, AI dependency risk give the output in comparison table or report (300–400 words).

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



```
comparison = """
╔═══════════════════╦══════════════════════╦═══════════════════════╗
║ CRITERIA          ║ WITHOUT FUNCTIONS    ║ WITH FUNCTIONS        ║
╠═══════════════════╬══════════════════════╬═══════════════════════╣
║ Logic Clarity     ║ Low - Code is linear,║ High - Each task is   ║
║                   ║ hard to follow flow  ║ isolated with clear   ║
║                   ║ and purpose          ║ purpose               ║
╠═══════════════════╬══════════════════════╬═══════════════════════╣
║ Reusability       ║ Poor - Must rewrite  ║ Excellent - Call      ║
║                   ║ logic for each use   ║ functions multiple    ║
║                   ║                      ║ times                 ║
╠═══════════════════╬══════════════════════╬═══════════════════════╣
║ Debugging Ease    ║ Difficult - Hard to  ║ Easy - Isolate issues ║
║                   ║ pinpoint errors in   ║ to specific functions ║
║                   ║ long code blocks     ║                       ║
╠═══════════════════╬══════════════════════╬═══════════════════════╣
║ Large Projects    ║ Unsuitable - Code    ║ Ideal - Modular,      ║
║                   ║ becomes unmaintainable║ scalable, organized  ║
╠═══════════════════╬══════════════════════╬═══════════════════════╣
║ AI Dependency Risk║ Moderate - AI may    ║ Lower - Function      ║
║                   ║ generate redundant   ║ boundaries help AI    ║
║                   ║ code                 ║ generate focused code ║
```

```
Enter a number: 12
Factorial of 12 is 479001600
```

EXPLANATION:-
Function-based code is better than non-function code because it makes the program easier to understand, reuse, and debug. Functions divide the logic into clear parts, which helps in finding errors quickly and makes the code suitable for large projects. Non-function code may work for small or quick tasks, but it becomes hard to manage as the program grows. Overall, using functions is the professional and reliable approach, especially when working with AI-generated code.

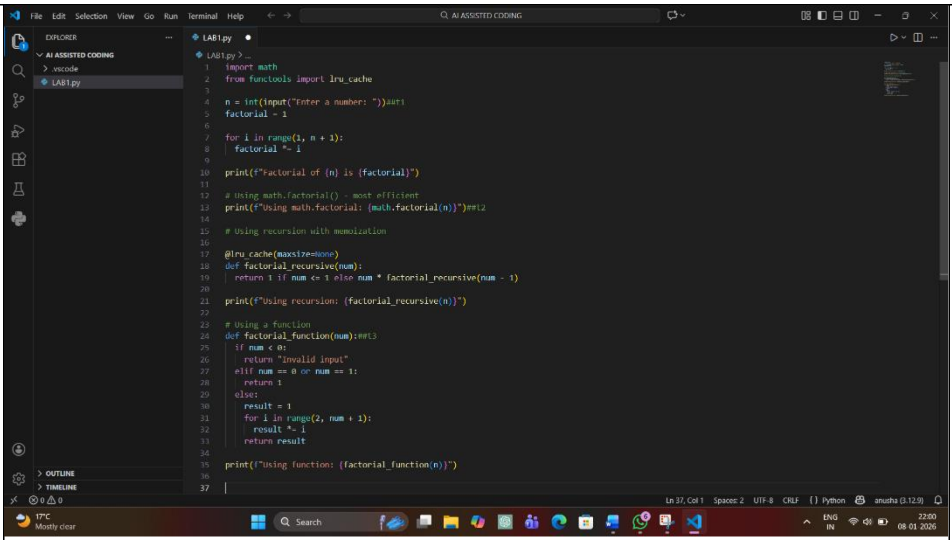
Task 5: AI-Generated Iterative vs Recursive Thinking

- **Scenario**
  Your mentor wants to test how well AI understands different computational paradigms.

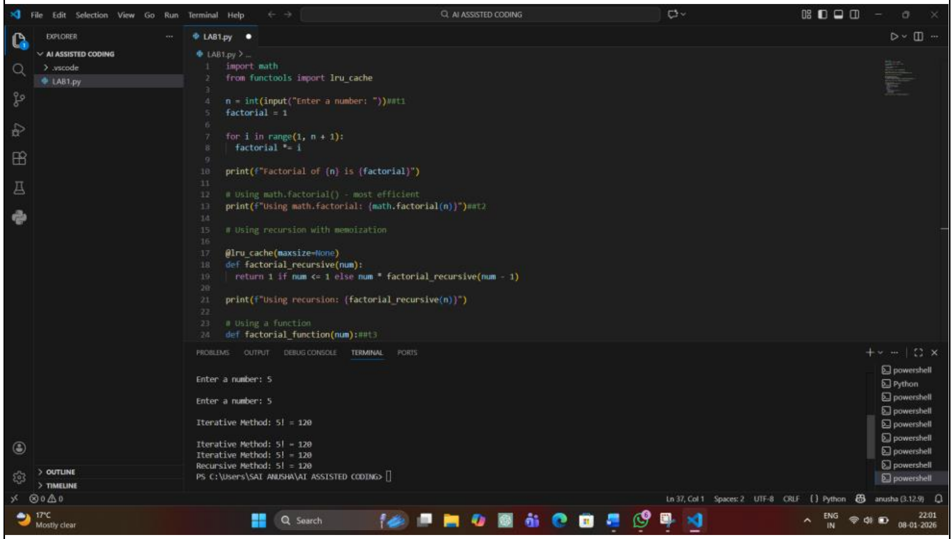
- **Task Description**
  Prompt Copilot to generate:
  An **iterative** version of the logic
  A **recursive** version of the same logic

- **Constraints**
  Both implementations must produce identical outputs
  Students must **not manually write the code first**

- **Expected Deliverables**
  Two AI-generated implementations
  Execution flow explanation (in your own words)
  Comparison covering:
  - Readability
  - Stack usage
  - Performance implications
  - When recursion is *not* recommended.



```python
import math
from functools import lru_cache

n = int(input("Enter a number: "))##t1
factorial = 1

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

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

# Using math.factorial() - most efficient
print(f"Using math.factorial: {math.factorial(n)}")##t2

# Using recursion with memoization

@lru_cache(maxsize=None)
def factorial_recursive(num):
  return 1 if num <= 1 else num * factorial_recursive(num - 1)

print(f"Using recursion: {factorial_recursive(n)}")

# Using a function
def factorial_function(num):##t3
  if num < 0:
    return "Invalid input"
  elif num == 0 or num == 1:
    return 1
  else:
    result = 1
    for i in range(2, num + 1):
      result *= i
    return result

print(f"Using function: {factorial_function(n)}")

```



```python
import math
from functools import lru_cache

n = int(input("Enter a number: "))##t1
factorial = 1

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

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

# Using math.factorial() - most efficient
print(f"Using math.factorial: {math.factorial(n)}")##t2

# Using recursion with memoization

@lru_cache(maxsize=None)
def factorial_recursive(num):
  return 1 if num <= 1 else num * factorial_recursive(num - 1)

print(f"Using recursion: {factorial_recursive(n)}")

# Using a function
def factorial_function(num):##t3
```

```
Enter a number: 5

Enter a number: 5

Iterative Method: 5! = 120

Iterative Method: 5! = 120
Iterative Method: 5! = 120
Recursive Method: 5! = 120
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> 
```

EXPLANATION:-

The program calculates the factorial of a number using both a loop (iterative method) and a function that calls itself (recursive method), and both give the same result.

The iterative method uses less memory and is safer, while the recursive method is simpler to write but uses more stack memory.

Submission Requirements

1. Generate code for each task with comments.
2. Screenshots of Copilot suggestions.
3. Comparative analysis reports (Task 4 and Task 5).
4. Sample inputs/outputs demonstrating correctness.

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output

	and if required, screenshots.	