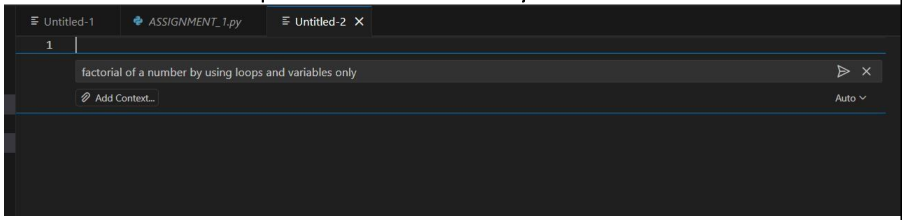
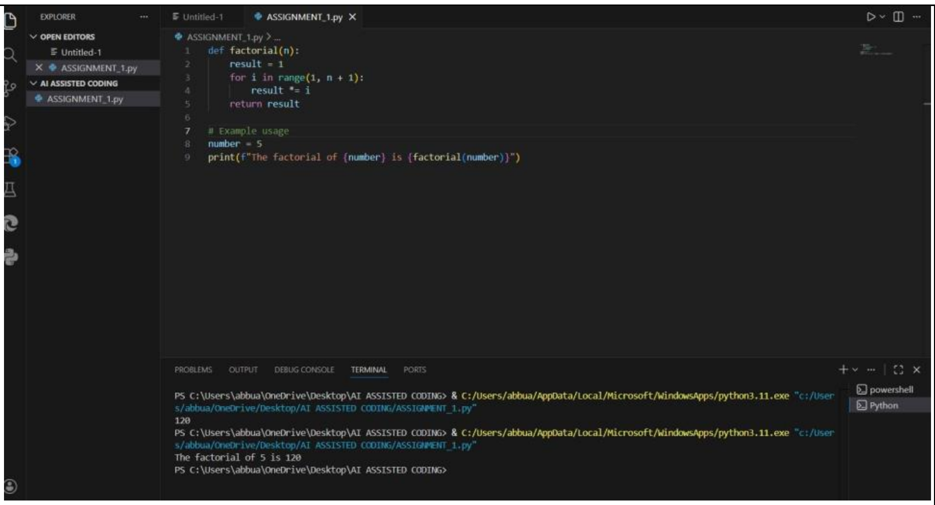
ASSIGNMENT_1

Gangula Navyasri

2303A51739
B_25

1	**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?  Untitled-1 ASSIGNMENT_1.py Untitled-2 Add Context... Auto factorial of a number by using loops and variables only	Week1 - Monday



```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Example usage
number = 5
print(f"The factorial of {number} is {factorial(number)}")
```

```
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
120
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\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**

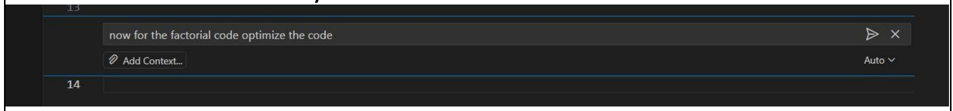
Analyze the code generated in **Task 1** and use Copilot again to:

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

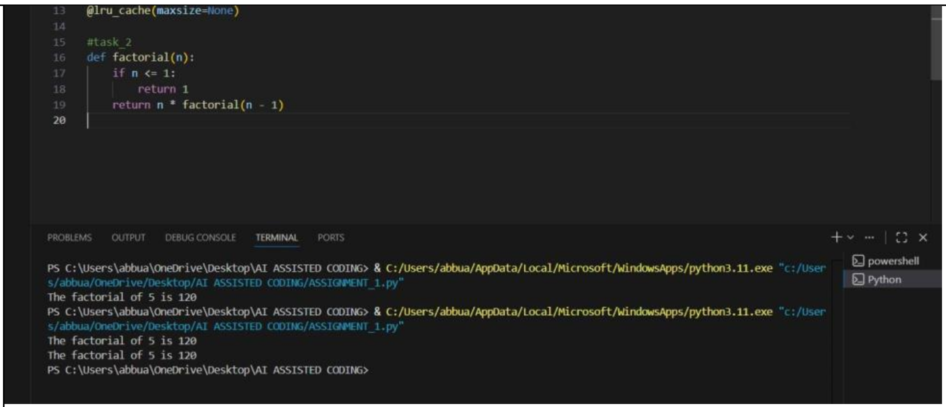
Hint:
Prompt Copilot with phrases like
"optimize this code", *"simplify logic"*, or *"make it more readable"*

❖ **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.



now for the factorial code optimize the code



```
@lru_cache(maxsize=None)

#task_2
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```

```
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 5 is 120
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\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:

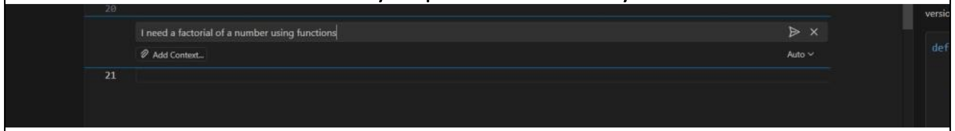
- ➢ Creating a **user-defined function**
- ➢ Calling the function from the main block

❖ **Constraints**

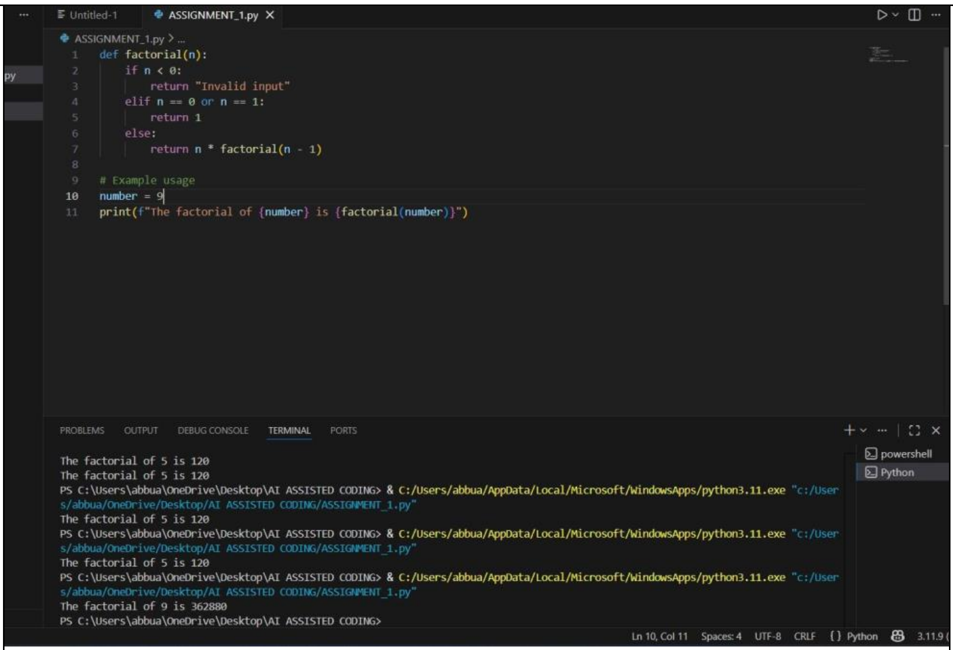
- ➢ Use meaningful function and variable names
- ➢ Include inline comments (preferably suggested by Copilot)

❖ **Expected Deliverables**

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



I need a factorial of a number using functions



```python
def factorial(n):
    if n < 0:
        return "Invalid input"
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage
number = 9
print(f"The factorial of {number} is {factorial(number)}")
```

```
The factorial of 5 is 120
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 5 is 120
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/abbua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/User
s/abbua/OneDrive/Desktop/AI ASSISTED CODING/ASSIGNMENT_1.py"
The factorial of 9 is 362880
PS C:\Users\abbua\OneDrive\Desktop\AI ASSISTED CODING>
```

Untitled-1
ASSIGNMENT_1.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python
Ln 10, Col 11 Spaces: 4 UTF-8 CRLF {} Python 3.11.9

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**

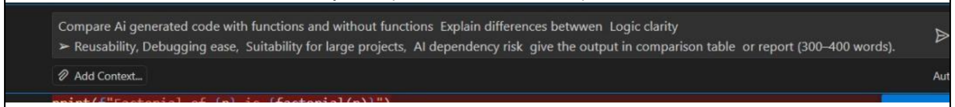
Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:

- ➢ Logic clarity
- ➢ Reusability
- ➢ Debugging ease
- ➢ Suitability for large projects
- ➢ AI dependency risk

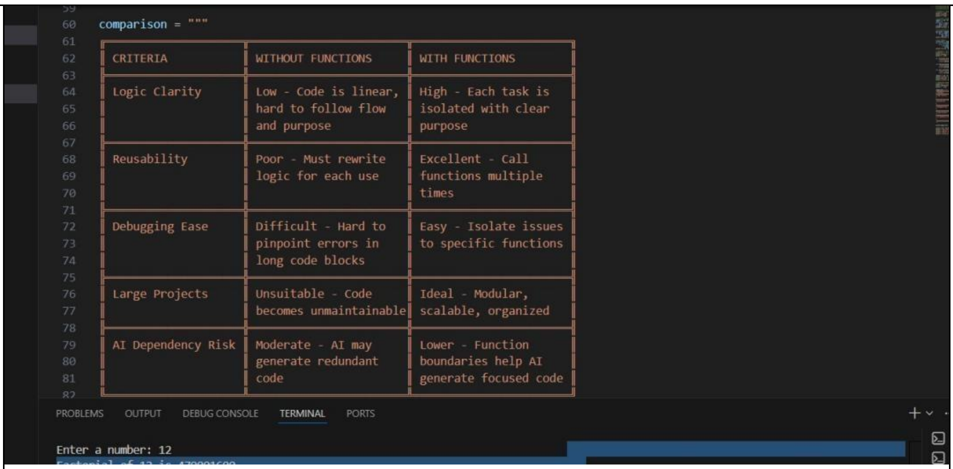
❖ **Expected Deliverables**

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).



```
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
```

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**

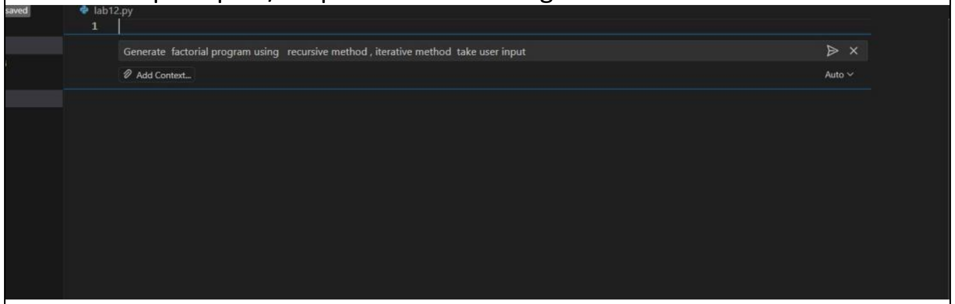
Two AI-generated implementations
Execution flow explanation (in your own words)
Comparison covering:

- ➢ Readability
- ➢ Stack usage
- ➢ Performance implications
- ➢ When recursion is *not* recommended.

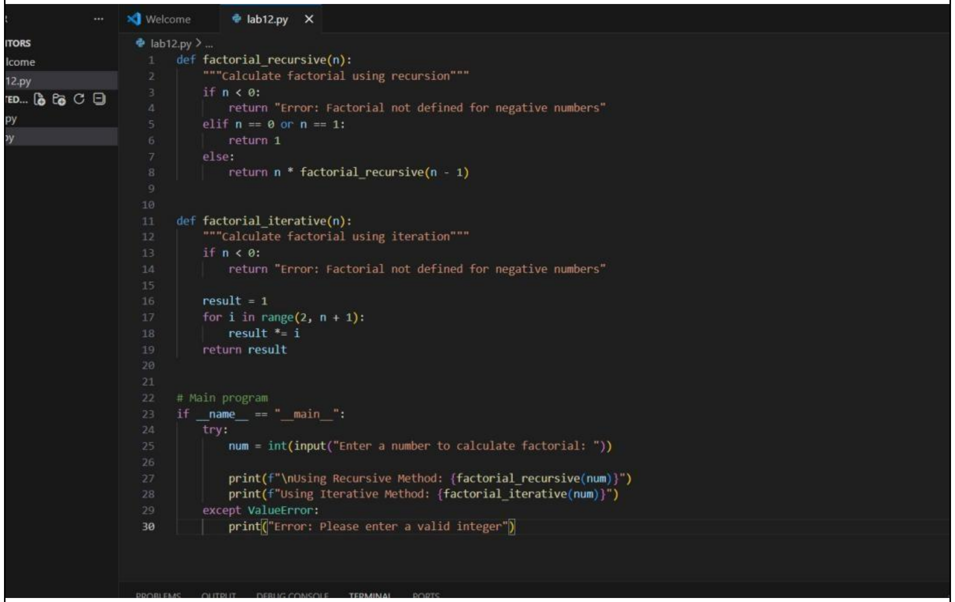
Submission Requirements

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.



Generate factorial program using recursive method , iterative method take user input



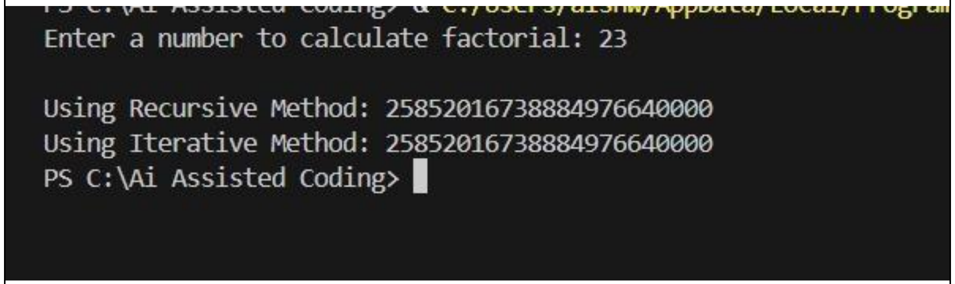
```python
def factorial_recursive(n):
    """Calculate factorial using recursion"""
    if n < 0:
        return "Error: Factorial not defined for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)


def factorial_iterative(n):
    """Calculate factorial using iteration"""
    if n < 0:
        return "Error: Factorial not defined for negative numbers"

    result = 1
    for i in range(2, n + 1):
        result *= i
    return result


# Main program
if __name__ == "__main__":
    try:
        num = int(input("Enter a number to calculate factorial: "))

        print(f"\nUsing Recursive Method: {factorial_recursive(num)}")
        print(f"Using Iterative Method: {factorial_iterative(num)}")
    except ValueError:
        print("Error: Please enter a valid integer")
```



```
PS C:\Ai Assisted Coding> & c:/Users/aishw/AppData/Local/Program
Enter a number to calculate factorial: 23

Using Recursive Method: 25852016738884976640000
Using Iterative Method: 25852016738884976640000
PS C:\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.