Assignment-1.2

K. Akshay
2303A51817 - Batch-26

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

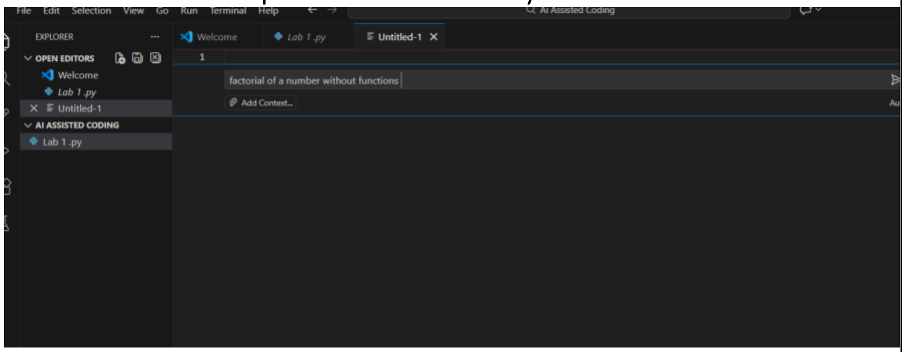
Assignment Number:1.2(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***

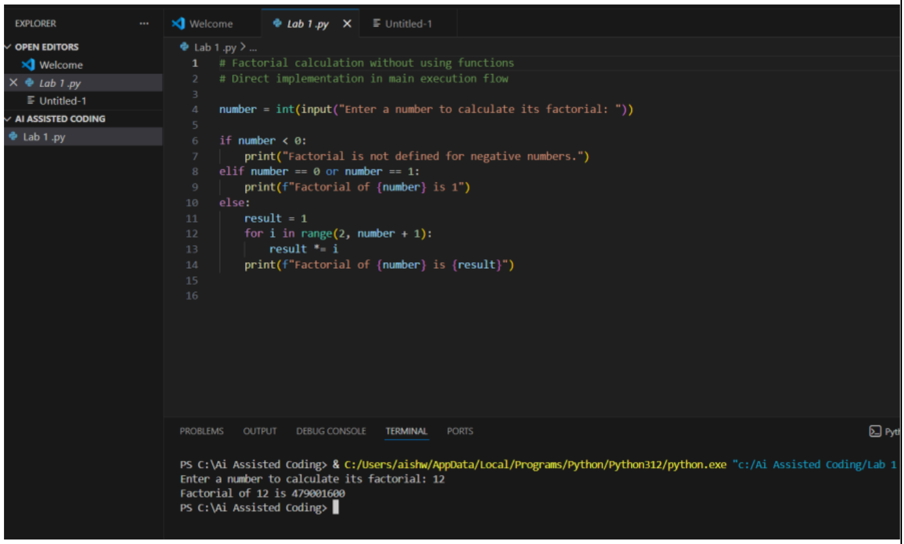
1	Lab 1: Environment Setup – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow* **Lab Objectives:** ● To install and configure GitHub Copilot in Visual Studio Code. ● To explore AI-assisted code generation using GitHub Copilot. ● To analyze the accuracy and effectiveness 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.	Week1 - Monday
	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 without functions



```python
# Factorial calculation without using functions
# Direct implementation in main execution flow

number = int(input("Enter a number to calculate its factorial: "))

if number < 0:
    print("Factorial is not defined for negative numbers.")
elif number == 0 or number == 1:
    print(f"Factorial of {number} is 1")
else:
    result = 1
    for i in range(2, number + 1):
        result *= i
    print(f"Factorial of {number} is {result}")
```

```
PS C:\Ai Assisted Coding> & C:/Users/aishw/AppData/Local/Programs/Python/Python312/python.exe "c:/Ai Assisted Coding/Lab 1
Enter a number to calculate its factorial: 12
Factorial of 12 is 479001600
PS C:\Ai Assisted Coding> 
```

Explanation: This Program Calculates the factorial of a number.If negative number is entered the factorial is not defined .

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**

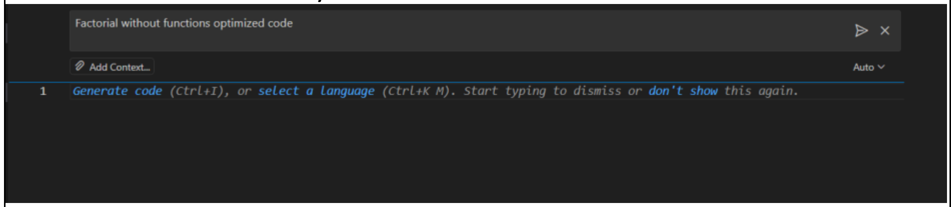
Analyze the code generated in **Task 1** and use Copilot again to:

- 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”*

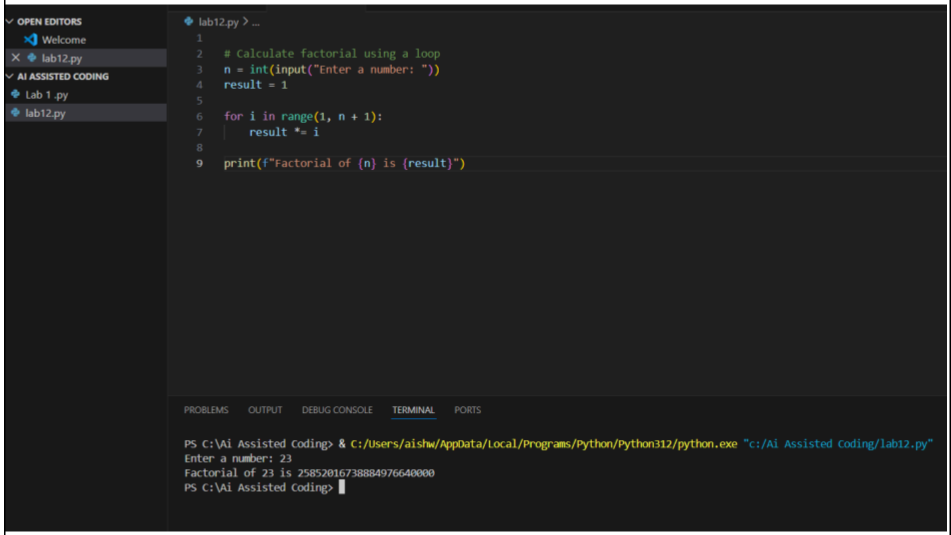
❖ **Expected Deliverables**

- 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.



Factorial without functions optimized code

```
Generate code (Ctrl+I), or select a language (Ctrl+K M). Start typing to dismiss or don't show this again.
```



```python

# Calculate factorial using a loop
n = int(input("Enter a number: "))
result = 1

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

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

```
PS C:\Ai Assisted Coding> & C:/Users/aishw/AppData/Local/Programs/Python/Python312/python.exe "c:/Ai Assisted Coding/lab12.py"
Enter a number: 23
Factorial of 23 is 25852016738884976640000
PS C:\Ai Assisted Coding>
```

Explain : Optimized code removes unnecessary conditions simplifies the logic which makes it more understandable and maintainable .

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

❖ **Scenario**

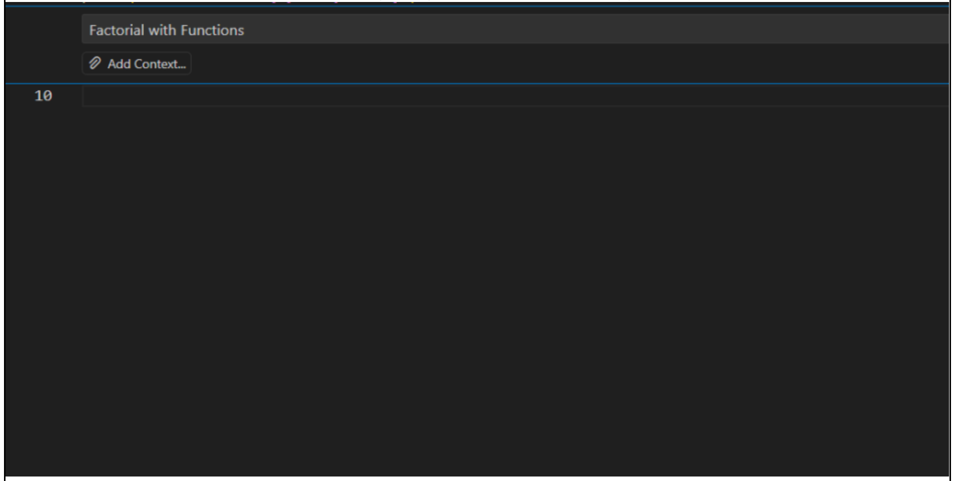
The same logic now needs to be reused in **multiple scripts**.

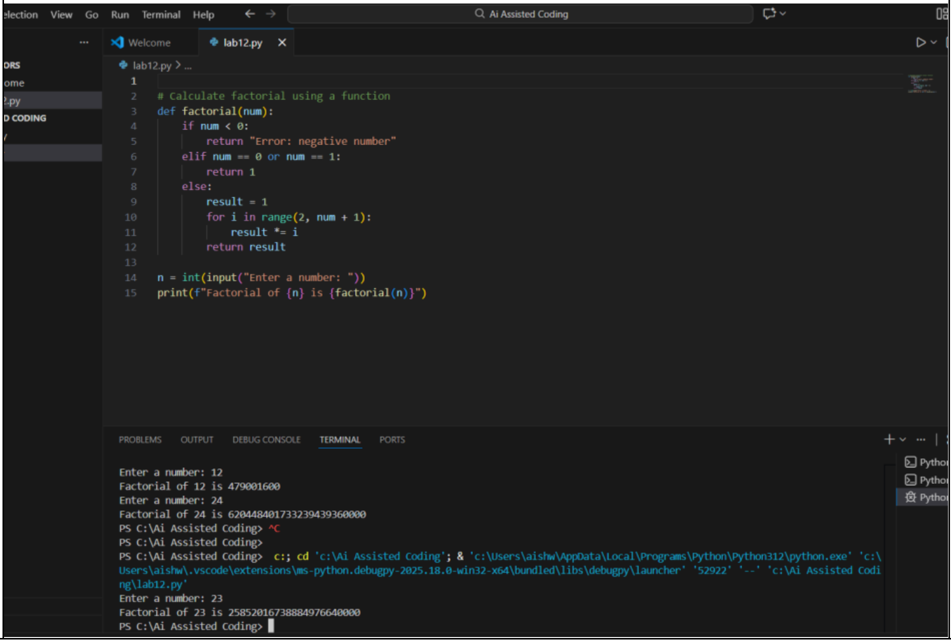
❖ **Task Description**

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 reusability.




election View Go Run Terminal Help
Ai Assisted Coding
Welcome
lab12.py
lab12.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python

# Calculate factorial using a function
def factorial(num):
    if num < 0:
        return "Error: negative number"
    elif num == 0 or num == 1:
        return 1
    else:
        result = 1
        for i in range(2, num + 1):
            result *= i
        return result

n = int(input("Enter a number: "))
print(f"Factorial of {n} is {factorial(n)}")
```

```
Enter a number: 12
Factorial of 12 is 479001600
Enter a number: 24
Factorial of 24 is 620448401733239439360000
PS C:\Ai Assisted Coding> ^C
PS C:\Ai Assisted Coding>
PS C:\Ai Assisted Coding>  c:; cd 'c:\Ai Assisted Coding'; & 'c:\Users\aishw\AppData\Local\Programs\Python\Python312\python.exe' 'c:\
Users\aishw\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52922' '--' 'c:\Ai Assisted Codi
ng\lab12.py'
Enter a number: 23
Factorial of 23 is 25852016738884976640000
PS C:\Ai Assisted Coding> 
```

Explanation : This Program calculates the factorial using user defined function . This condition avoids unnecessary calculations .

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

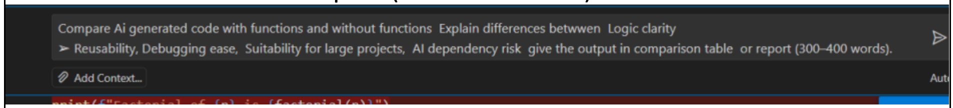
❖ **Scenario**
As part of a **code review meeting**, you are asked to justify design choices.

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

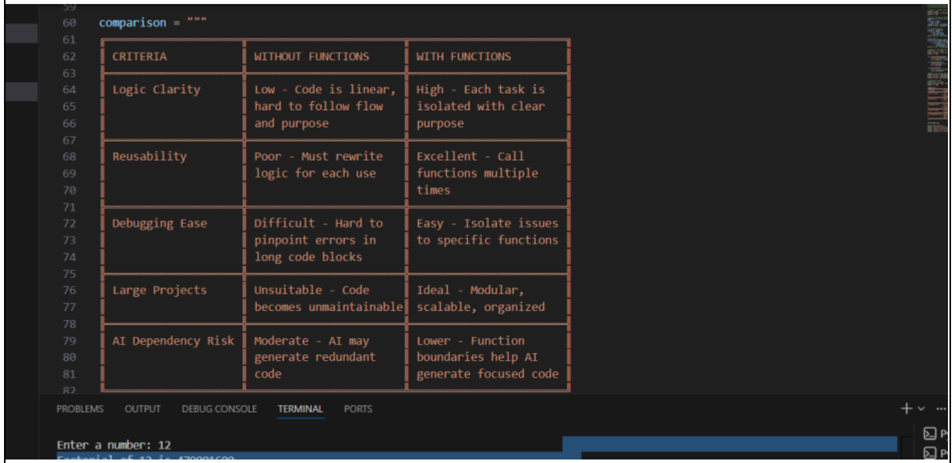
- ➢ Logic clarity
- ➢ Reusability
- ➢ Debugging ease
- ➢ Suitability for large projects
- ➢ AI dependency risk

❖ **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 : This table shows that writing code with functions is better than writing code without functions because functions make the logic clearer, allow code reuse, simplify debugging, and help manage large projects efficiently. Code without functions is harder to understand, difficult to maintain, and often leads to repetitive or messy code, while functions keep programs modular and organized.

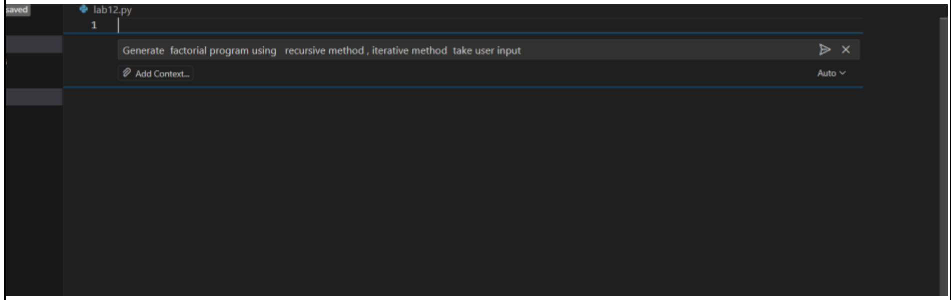
Task 5: AI-Generated Iterative vs Recursive Thinking

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

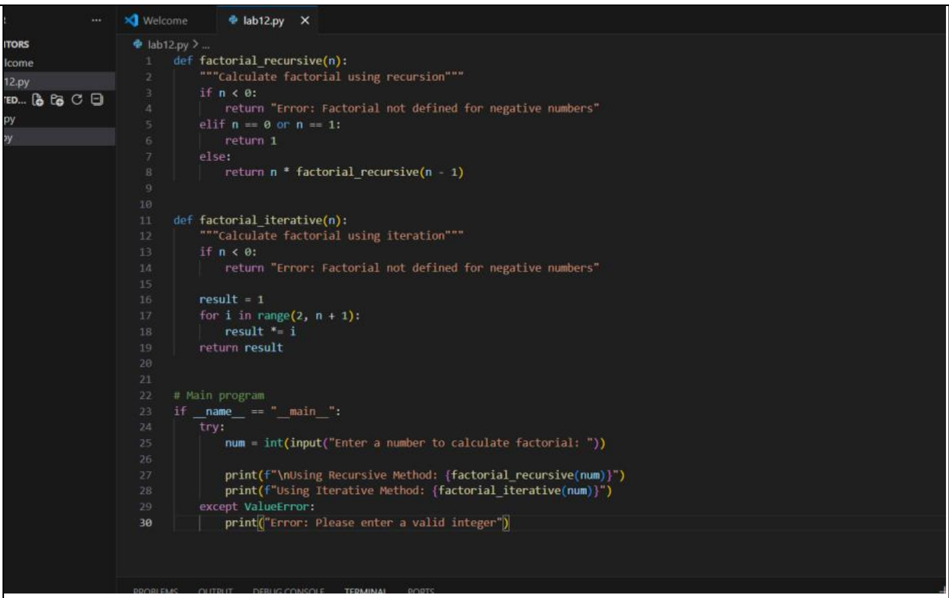
- **Task Description**
  Prompt Copilot to generate:
  An **iterative** version of the logic
  A **recursive** version of the same logic

- **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.



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



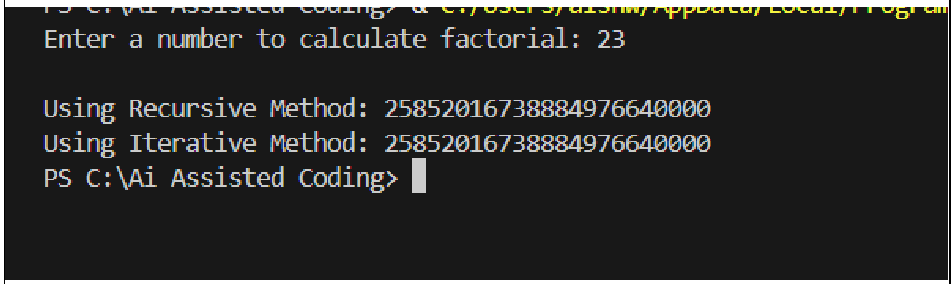
```python
def factorial_recursive(n):
    """Calculate factorial using recursion"""
    if n < 0:
        return "Error: Factorial not defined for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)


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")
```



```
Enter a number to calculate factorial: 23

Using Recursive Method: 25852016738884976640000
Using Iterative Method: 25852016738884976640000
PS C:\Ai Assisted Coding>
```

Explanation : This program finds the factorial of a number using both recursive and iterative methods. The recursive function calls itself until it reaches the base case, while the iterative function uses a loop to calculate the result. Both methods give the same output for valid inputs. Recursion is easier to understand but uses more memory, whereas iteration is more efficient and safer for large numbers.

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.