

2303A52366

Batch no:31

Assignment-1.2		
Q.No.	Question	Expected Time to complete
1	<p>Lab 1: Environment Setup – <i>GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow</i></p> <p>Lab Objectives:</p> <ul style="list-style-type: none">• 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 <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none">• 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. <hr/> <p>Task 0</p> <ul style="list-style-type: none">• Install and configure GitHub Copilot in VS Code. Take screenshots of each step. <p>Expected Output</p> <ul style="list-style-type: none">• Install and configure GitHub Copilot in VS Code. Take screenshots of each step. <hr/> <p>Task 1: AI-Generated Logic Without Modularization (Factorial without</p>	Week1 - Monday

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

```
 2 number = 5
 3 factorial = 1
 4 for i in range(1, number + 1):
 5     factorial *= i
 6 print(f"The factorial of {number} is {factorial}")
```

- Screenshot(s) showing:

The screenshot shows a terminal window with the title 'Task1.py'. The code is as follows:

```
Task1.py
Assignment1 > Task1.py > ...
1 # Write a Python code that takes input from user and calculates the factorial of a number us
2 num = int(input("Enter a number to calculate its factorial: "))
3 factorial = 1
4 for i in range(1, num + 1):
```

- The prompt you typed

```
#Write a Python code to find a factorial. Use a simple for loop. Do not use
functions
```

- Copilot's suggestions

```
# write a Python code that takes input from user and checks whether the number is prime or not
```

➤ Sample input/output screenshots

```
PS C:\Users\moham> & C:/Users/moham/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/moham/all/Packages/5319275A.What'sApp/Desktop_cv1g1gvanyjgm/LocalState/sessions/F7AF872B6F59FC2D30BE62A461868691ansfers/2026-02/aiac_lab.py"
The factorial of 5 is 120
PS C:\Users\moham>
```

➤ Brief reflection (5–6 lines):

We start with factorial = 1 because 1 is the neutral starting point for multiplication. The range(1, number + 1) creates a list of numbers from 1 up to 5. The loop takes each number one by one and multiplies it into our total.

➤ How helpful was Copilot for a beginner?

We can generate code **in** just seconds simply by writing a prompt. It **is** very helpful **for learning** and allows us to engage **in** a much more efficient way.

➤ Did it follow best practices automatically?

Yes, it automatically implemented several coding best practices

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

```
num = int(input("Enter a number to calculate its
factorial: "))
factorial = 1
for i in range(1, num + 1):
    factorial *= i
print(f"The factorial of {num} is {factorial}")
```

- Optimized version of the same code

```
num = int(input("Enter a number to calculate its
factorial: "))
if num < 0:
    print("Factorial is not defined for negative
numbers.")
elif num == 0 or num == 1:
    print(f"The factorial of {num} is 1")
else:
    factorial = 1
    for i in range(2, num + 1):
        factorial *= i
    print(f"The factorial of {num} is
{factorial}")
```

- Side-by-side comparison

```

Task1.py
Assignment1 > Task1.py > ...
1 # Write a Python code that takes input from user
2 num = int(input("Enter a number to calculate factorial"))
3 factorial = 1
4 for i in range(1, num + 1):
5     factorial *= i
6 print(f"The factorial of {num} is {factorial}")
7
8
9

Task2.py
Assignment1 > Task2.py > ...
1 # Write a Python code that takes input from user
2
3 num = int(input("Enter a number to calculate factorial"))
4 if num < 0:
5     print("Factorial is not defined for negative numbers")
6 elif num == 0 or num == 1:
7     print(f"The factorial of {num} is {num}")
8 else:
9     factorial = 1
10    for i in range(2, num + 1):
11        factorial *= i
12    print(f"The factorial of {num} is {factorial}")
13

```

➤ Written explanation:

- What was improved?

In the optimized code it first checks for negative numbers and if the user enters 1 or 0 it runs faster than the original code.

- Why the new version is better (readability, performance, maintainability).

using if-else blocks for edge cases like 0 and 1 it makes faster to run compared to orginal code

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

```

def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage
number = int(input("Enter a number to calculate its factorial: "))
print(f"The factorial of {number} is: {factorial(number)}")

```

➤ Screenshots showing:

- Prompt evolution
- Copilot-generated function logic

```

def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

```

➤ Sample inputs/outputs

```

PS C:\Users\moham> & C:/Users/moham/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/moham/AppData/Local/PA
275A.WhatApp/Desktop_cv1g1gvanyjgw/LocalState/sessions/F7AF872B6F59FC2D30BE62AA61868691D61FBCA2/transfers/2026-02/aiac
Enter a number to calculate its factorial: -6
The factorial of -6 is: Factorial is not defined for negative numbers.
PS C:\Users\moham>

```

➤ Short note:

- How modularity improves reusability

modules that can be reused across projects without rewriting, reducing duplication and enhancing maintainability.

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

==== PROCEDURAL VERSION ==== Enter a number to calculate its factorial: 5 The factorial of 5 is 120 ⌚ Execution Time (Procedural): 2207.452900 ms =====		
==== MODULAR VERSION ==== Enter a number to calculate its factorial: 5 The factorial of 5 is 120 ⌚ Execution Time (Modular): 1376.439900 ms =====		
===== PERFORMANCE ANALYSIS =====		
===== Procedural Version: 2207.452900 ms Modular Version: 1376.439900 ms Modular was faster by 831.013000 ms		
Feature	Non-Function Approach (Task 1)	Function-Based Approach (Task 2)
Structure	Monolithic: Logic, input, and output are mixed in one global block.	Modular: Logic is encapsulated in a specific function, separating it from I/O.
Reusability	Low: Code must be copied and pasted to be used elsewhere.	High: Function can be imported and called by any other script or module.
Error Handling	Basic: Uses simple <code>print</code> statements (e.g., "Error: Negative number").	Robust: Uses Exceptions (<code>raise ValueError</code>) allowing programs to catch and manage errors.
Testing	Difficult: Requires manual user input for every test case.	Easy: Can be automated using Unit Tests to verify logic instantly.
Maintainability	Poor: Changes to logic might break the input/output flow.	Excellent: You can upgrade the math logic without touching the rest of the code.

OR

- A short technical report (300–400 words).

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

Iterative

```
def factorial_iterative(n):
    if n < 0:
        return "Factorial is not defined for negative
numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

num = int(input("Enter a number to calculate its
factorial: "))
factorial_result = factorial_iterative(num)
print(f"The factorial of {num} is
{factorial_result}")
```

recursive

```
def factorial_recursive(n):
    if n < 0:
        return "Factorial is not defined for negative
numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)
num = int(input("Enter a number to calculate its
```

```
factorial: "))
factorial_result = factorial_recursive(num)
print(f"The factorial of {num} is
{factorial_result}")
```

➤ Stack usage

an iterative one using a loop for factorial calculation and a recursive one..both handling negatives and base cases with user input and output.

➤ Performance implications

Time Complexity: Both iterative and recursive approaches are $O(n)$

Space Complexity: Iterative uses $O(1)$ and Recursive uses $O(n)$ space due to call stack.

➤ When recursion is *not* recommended.

when input n is large as it can cause stack overflow due to deep call stacks

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.