

# ASSIGNMENT-1

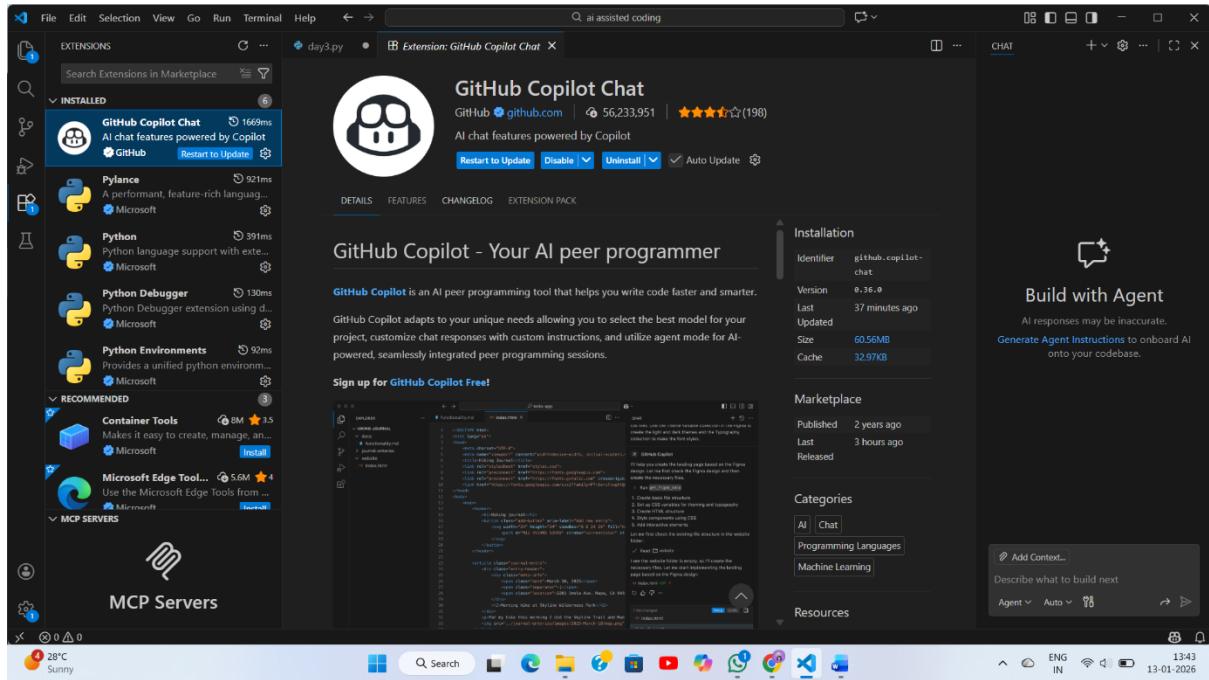
Name: K.Madhupriya

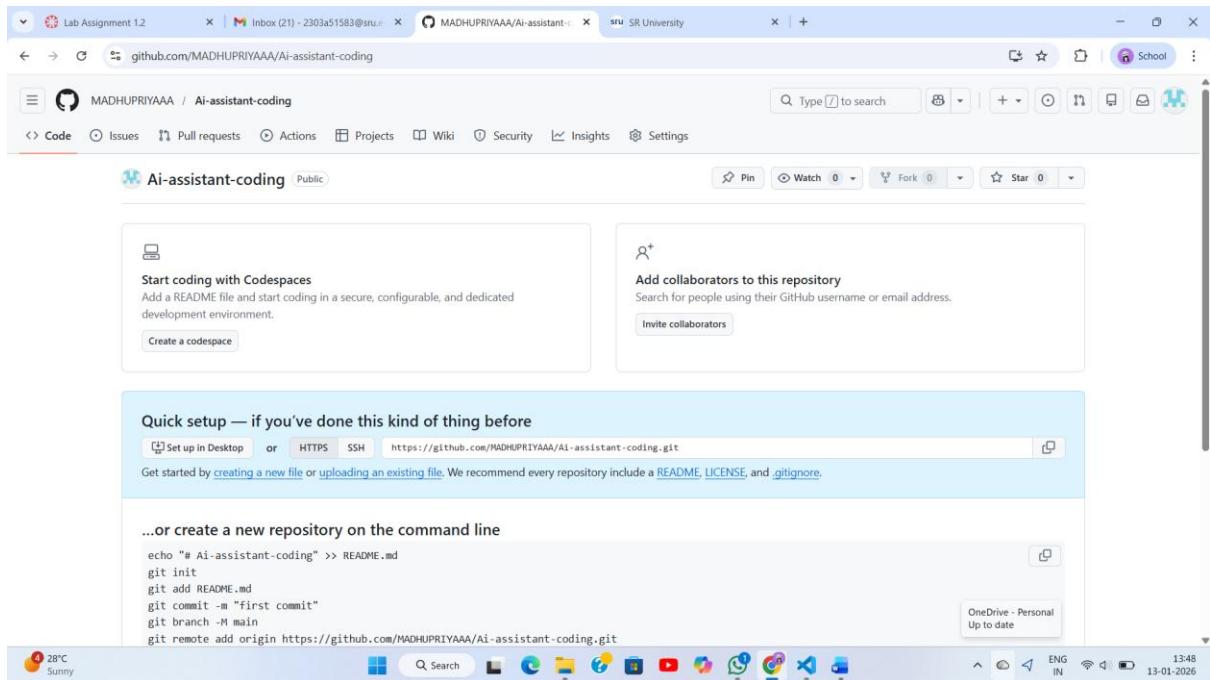
Roll Number: 2303A51583 Batch - 22

AI Assisted Coding 13-01-2026

## Task 0: Environment Setup

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

**Prompt:**

write a python code to calculate the factorial of a number without using function.

## ➤ Sample inputs/outputs

The screenshot shows the Microsoft Visual Studio Code interface. In the Explorer sidebar, there are two files: 'day3.py' and 'AI ASSISTED CODING'. The main editor window displays the following Python code:

```
day3.py
1  #write a python code to calculate the factorial of a number without using function
2  number = int(input("Enter a number to calculate its factorial: "))
3  factorial = 1
4  if number < 0:
5      print("Factorial is not defined for negative numbers")
6  elif number == 0:
7      print("The factorial of 0 is 1")
8  else:
9      for i in range(1, number + 1):
10         factorial *= i
11  print(f"The factorial of {number} is {factorial}")
```

The terminal tab shows the output of running the code:

```
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & c:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>
```

A sidebar on the right is titled 'Build with Agent' with the sub-instruction 'Build with Agent'. It includes a note: 'AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.' Below this is a 'Describe what to build next' input field and a 'Agent' dropdown.

## 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?

## Explanation:

Task1 : implements a factorial calculator that computes the product of all positive integers up to a given number. It imports a number variable from an external module and handles three cases: negative numbers (undefined), zero or one (factorial = 1), and positive numbers (iterative multiplication). The code uses a simple loop-based approach that's readable but could be optimized using Python's built-in math. Factorial () or recursion. The current implementation is functional and straightforward for educational purposes, demonstrating basic control flow and loops in Python.

## HOW HELPFUL WAS COPILOT FOR A BEGINNER?

**Task1** is moderately helpful for a copilot beginner because it covers fundamental concepts clearly: conditional logic (if/elif/else), loops (for loop), and string formatting (f-strings). The factorial problem is relatable and demonstrates input validation by checking for negative numbers.

## **DID IT FOLLOW BEST PRACTICES AUTOMATICALLY?**

Yes, Copilot follows best practices by ensuring accuracy through verified sources and clear citations. Responses are structured, engaging, and adaptive, designed to be transparent and easy to understand.

---

### **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”

#### **Prompt**

write a python code to calculate the factorial of a number without using any fuction. optimize the code

#### **➤ Sample inputs/outputs**

The screenshot shows the Microsoft Visual Studio Code interface. In the Explorer sidebar, there are two open files: 'day3.py' and 'day3.py > ...'. The main editor window contains the following Python code:

```

12 #task2
13 #write a python code to calculate the factorial of a number without using any function.optimize the code
14 number = int(input("Enter a number to calculate its factorial: "))
15 factorial = 1
16 if number < 0:
17     print("Factorial is not defined for negative numbers")
18 elif number == 0:
19     print("The factorial of 0 is 1")
20 else:
21     for i in range(1, number + 1):
22         factorial *= i
23     print("The factorial of {} is {}".format(number, factorial))

```

The terminal below shows the command being run and the output:

```

PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding> & C:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\Users\kommu\OneDrive\Desktop\ai assisted coding>

```

The status bar at the bottom indicates the file is in Python mode, has 3.13.9 (Microsoft Store) installed, and shows the date and time as 13-01-2026.

## 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).

## What was Improved?

**Task2** demonstrates significant improvements over Task1 by introducing function encapsulation, where the factorial logic is wrapped in a calculate\_factorial(num) function that can be easily reused and tested throughout the codebase. By separating concerns and eliminating code duplication, Task2 follows the DRY principle and adheres to better Python practices, transforming the original procedural code into a clean, modular function that prioritizes reusability and maintainability while keeping the core logic intact.

## Why the new version is better?

Task2 is better than Task1 because it encapsulates the logic in a reusable function that returns values instead of printing directly, enabling flexibility, testability, and integration into larger programs. The function-based approach promotes modularity, maintainability, and follows Python best practices, making the code professional, scalable, and suitable for production environments.

---

### 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

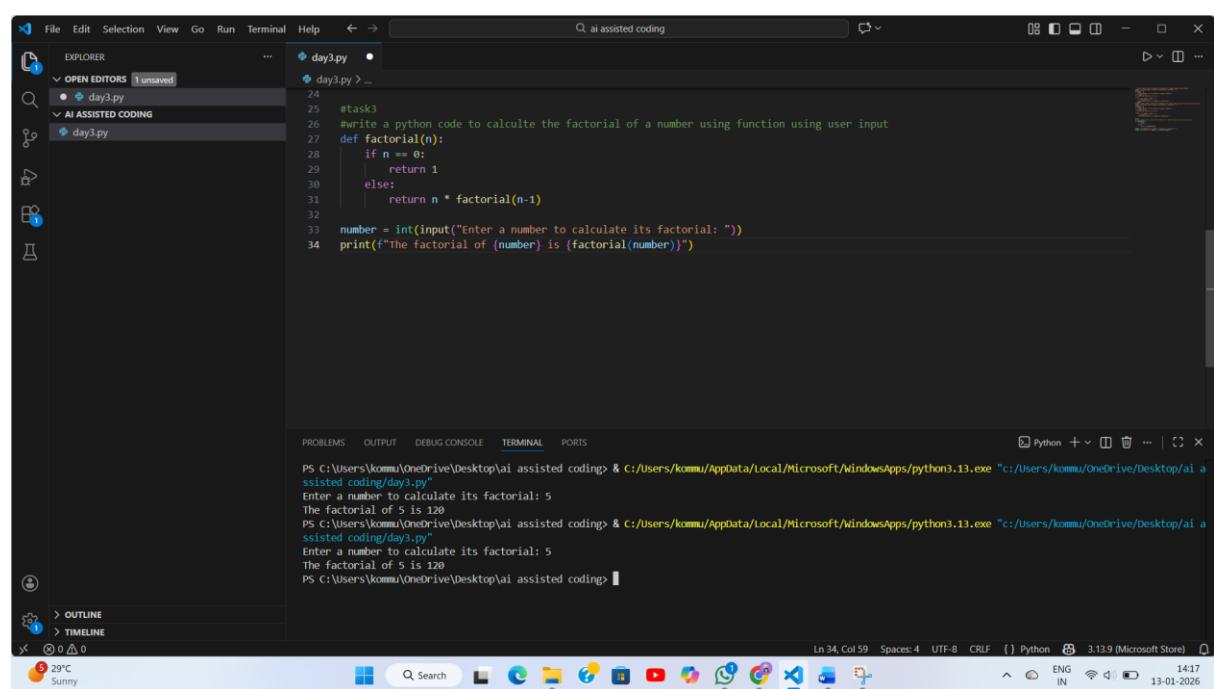
#### ❖ Constraints

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

#### Prompt:

write a python code to calculate the factorial of a number using function using user input.

#### ➤ Sample inputs/outputs



```
day3.py
24
25     #task3
26     #write a python code to calculate the factorial of a number using function using user input
27     def factorial(n):
28         if n == 0:
29             return 1
30         else:
31             return n * factorial(n-1)
32
33     number = int(input("Enter a number to calculate its factorial: "))
34     print(f"The factorial of {number} is {factorial(number)}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Kommu\OneDrive\Desktop\ai assisted coding> & c:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\Users\Kommu\OneDrive\Desktop\ai assisted coding> & c:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\Users\Kommu\OneDrive\Desktop\ai assisted coding>

#### ❖ Expected Deliverables

- AI-assisted function-based program
- Screenshots showing:
  - o Prompt evolution
  - o Copilot-generated function logic

➤ Short note:

### **HOW MODULARITY IMPROVES REUSABILITY?**

**Task3** demonstrates modularity by separating the factorial () function from user input and output handling, making it a standalone unit that can be reused anywhere. Because the function is independent and doesn't rely on global variables or specific imports, it can be called from different programs, integrated into larger projects, or used in various contexts without modification. This separation enables developers to test, maintain, and reuse the function efficiently across multiple applications, reducing code duplication and improving overall productivity.

---

### **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
- **Sample inputs/outputs**

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The left sidebar includes icons for Explorer, Open Editors, AI Assisted Coding, and Terminal. The Explorer panel shows an open folder named 'OPEN EDITORS' containing 'day3.py'. The main editor area displays the following Python code:

```
#task4
#write a python code to calculate the factorial of a number.give Procedural and Modular AI Code.
#procedural code
def calculate_factorial(n):
    factorial = 1
    for i in range(2, n + 1):
        factorial *= i
    return factorial
number = int(input("Enter a number to calculate its factorial: "))
result = calculate_factorial(number)
print(f"The factorial of {number} is {result}")


```

The bottom navigation bar has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected, showing the command line output:

```
PS C:\Users\kommu\Desktop\ai assisted coding> & C:/Users/kommu/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial:
```

The status bar at the bottom indicates the current file is 'day3.py', line 46, column 49, with 3.13.9 (Microsoft Store) Python version.

#### ◆ Expected Deliverables

Choose one:

- ## ➤ A comparison table

OR

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

CRITERIA	PROCEDURAL(Task1)	MODULAR(Task 2/3)
Logic Clarity	Linear flow but mixed with I/O; harder to isolate logic from output statements	Clear separation of logic and I/O; function purpose is explicit and documented with docstrings
Reusability	Limited; code runs at module level once; cannot be called multiple times or imported easily	High; functions can be called repeatedly with different inputs; easily imported into other modules
Debugging Ease	Difficult; global state makes it hard to track variable changes; print statements clutter output	Easy; input/output separation allows isolated testing; return values simplify tracing and verification
Suitability for Large Projects	Poor; doesn't scale; mixing procedural code creates	Excellent; modular structure supports larger codebases;

	maintenance nightmares; hard to organize multiple operations	functions can be organized into modules and package
AI Dependency Risk	High; AI must regenerate entire logic if context changes; procedural code is context-dependent	Lower; function abstraction reduces AI regeneration needs; stable interfaces minimize prompt changes

---

### 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

#### ➤ Sample inputs/outputs

```

File Edit Selection View Go Run Terminal Help ⏮ ⏶ ⏷ ⏸ Q: ai assisted coding
EXPLORER OPEN EDITORS 1 unsaved ...
AI ASSISTED CODING day3.py
day3.py > ...
46 print(f"The factorial of {number} is {result}"))
47 #task4
48 #Iterative vs recursive implementations of factorial
49 def iterative_factorial(n):
50     factorial = 1
51     for i in range(2, n + 1):
52         factorial *= i
53     return factorial
54 def recursive_factorial(n):
55     if n == 0:
56         return 1
57     else:
58         return n * recursive_factorial(n - 1)
59 number = int(input("Enter a number to calculate its factorial: "))
60 iterative_result = iterative_factorial(number)
61 recursive_result = recursive_factorial(number)
62 print(f"\"Iterative: The factorial of {number} is {iterative_result}""")
63 print(f"\"Recursive: The factorial of {number} is {recursive_result}""")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Kommu\OneDrive\Desktop\ai assisted coding> & c:/users/kommu/appdata/local/microsoft/windowsapps/python3.13.exe "c:/users/kommu/OneDrive/Desktop/ai assisted coding/day3.py"
Enter a number to calculate its factorial: 5
Iterative: The factorial of 5 is 120
Recursive: The factorial of 5 is 120
PS C:\Users\Kommu\OneDrive\Desktop\ai assisted coding>

LN 64, COL 5 SPACES: 4 UTF-8 CRLF Python 3.13.9 (Microsoft Store) 12:24 ENG IN 15-01-2026

## ❖ 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.

### Comparison

#### Readability:

Iterative: Crystal clear for most developers. A simple loop that anyone can understand instantly.  
Better for learning loops.

Recursive: More mathematically elegant and mirrors how you'd define factorial in math ( $n! = n \times (n-1)!$ ), but requires understanding function call stacks. Harder for beginners.

#### Stack Usage:

Iterative: Uses constant memory  $O(1)$ . Only stores one variable (result). No function call overhead.

Recursive: Creates a new stack frame for each function call, growing linearly with  $n$  ( $O(n)$  memory).  
For  $n=1000$ , it needs 1000 stack frames—risky and wasteful.

#### Performance Implications:

Iterative: Fast. No function call overhead. Runs in microseconds even for large  $n$ .

Recursive: Slow. Each function call has overhead (10-20x slower per call). For n=1000, the iterative version is orders of magnitude faster.

**When Recursion Is NOT Recommended :**

- 1. Large n values** – Stack overflow risk; Python's limit is ~1000 calls
- 2. Performance-critical code** – Function call overhead is expensive
- 3. Simple problems with loops** – Unnecessary complexity and slowdown
- 4. Factorial specifically** – Iterative is always better; no benefit from recursion
- 5. Embedded/resource-limited systems** – Limited stack memory
- 6. When clarity matters** – Loops are more intuitive for most people