

# Assignment - 1

Name: Sambaraju Vignesh

Roll Number: 2303A51217

Batch - 04

AI Assisted Coding

07-01-2026

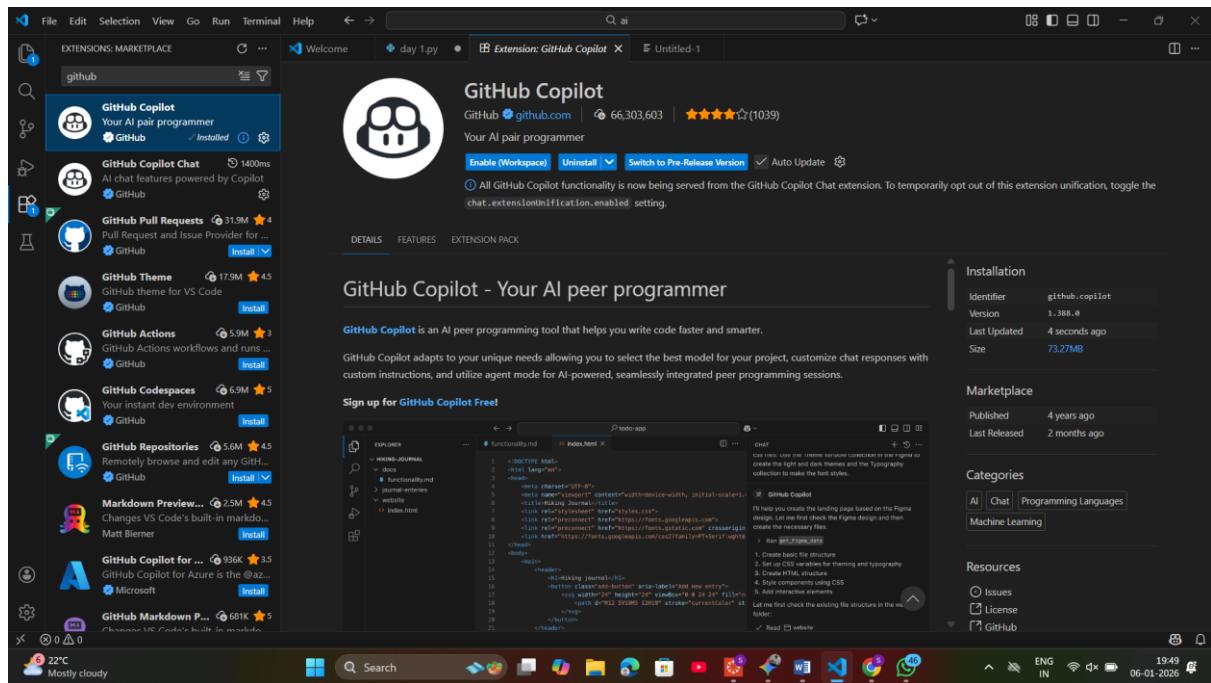
## Task 0: Environment Setup:-

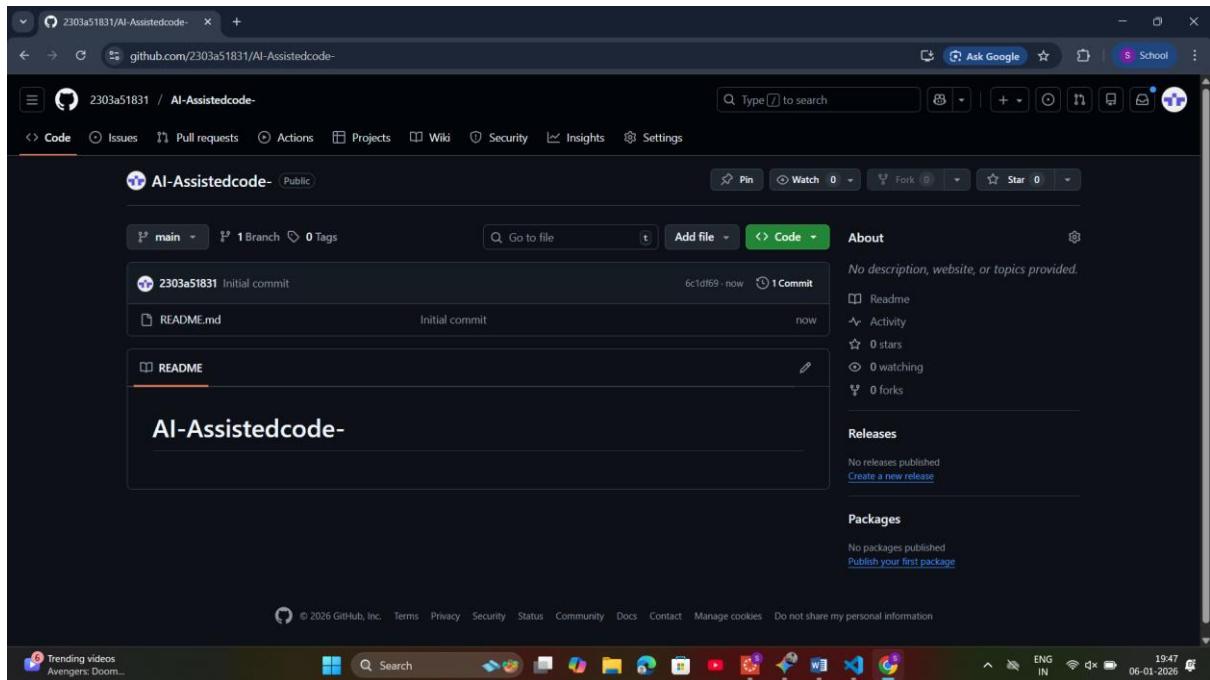
### Task 0

- 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: Non-Modular Logic (Factorial):-

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?

The screenshot shows the Microsoft Visual Studio Code interface. In the center, there is a code editor with the following Python code:

```
C:\> Users > hp > Desktop > ai > task1.py > ...
1 # Task 1: Procedural Factorial Implementation
2 num = int(input("Enter a number: "))
3 factorial = 1
4
5 if num < 0:
6     print("Factorial does not exist for negative numbers")
7 elif num == 0:
8     print("The factorial of 0 is 1")
9 else:
10    temp = num
11    while temp > 0:
12        factorial *= temp
13        temp -= 1
14    print(f"The factorial of {num} is {factorial}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

The terminal below shows the execution of the code:

```
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC> ^C
PS C:\Users\hp\OneDrive\Desktop\HPC>
PS C:\Users\hp\OneDrive\Desktop\HPC> cd 'c:\Users\hp\OneDrive\Desktop\HPC'; & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\vscode\extensions\ms-python.python\2025.18.0-win32-x64\bundles\libs\debug\launcher' '62630' '--' 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

The status bar at the bottom indicates: Line 14, Col 52, Spaces: 4, UTF-8, CR/LF, Python, 3.13.9 (Microsoft Store), 1549, ENG IN, 07-01-2026.

The screenshot shows a terminal window with the following command and output:

```
PS C:\Users\hp\OneDrive\Desktop\HPC & c:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/hp/OneDrive/Desktop/ai/task1.py
Enter a number: 5
Factorial is: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

The status bar at the bottom indicates: Line 0, Col 20, Spaces: 4, UTF-8, CR/LF.

## Task 2: AI Code Optimization:-

### 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).

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. In the center, there's a code editor with the following Python code:

```

1 # Task 2: Optimized Factorial
2 num = int(input("Enter a number: "))
3 factorial = 1
4
5 for i in range(1, num + 1):
6     factorial *= i
7
8 print(f"Factorial: {factorial}")

```

To the right of the code editor is a "CHAT" panel. A message from "Copilot" says: "python code". A response from "User" says: "Asked for clarification on Python code request". Another response from "User" says: "I'd be happy to help! However, your request is unclear. Could you clarify what you'd like me to do?". Below this, a "For example:" section lists several options with icons:

- View existing code: Should I look at lab1.py or task1.py?
- Write new code: What should the code do?
- Fix/debug code: What problem needs fixing?
- Run code: Which file should I execute?
- Explain code: Which code would you like me to explain?

At the bottom of the CHAT panel, there's a message input field: "Describe what to build next" and a "Send" button.

Below the code editor is a "TERMINAL" tab. The terminal window shows the command line output of running the script:

```

PS C:\Users\hp\OneDrive\Desktop\HPC> & "c:\users\hp\appdata\local\microsoft\windowsapps\python3.13.exe" "c:\users\hp\vscode\extensions\ms-python.python.debug-2025.18.0-win32-x64\bundled\libs\debug\Launcher" "56820" "<" "c:\users\hp\OneDrive\Desktop\ai\task1.py"
Enter a number: 5
Factorial: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

The status bar at the bottom indicates the file is indexed and shows the date and time: 07-01-2026 15:35.

## 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.

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays the 'EXPLORER' view with a file named 'lab1.py'. The main editor area shows a Python script named 'task1.py' with the following code:

```
C:\> Users > hp > OneDrive > Desktop > ai > lab1.py > ... > task1.py > ...
1 def calculate_factorial(n):
2     """Calculates the factorial of a given number iteratively."""
3     result = 1
4     for i in range(1, n + 1):
5         result *= i
6     return result
7
8 if __name__ == "__main__":
9     user_input = int(input("Enter number: "))
10    print(f"Result: {calculate_factorial(user_input)}")
```

The bottom terminal window shows the command-line output of running the script:

```
PS C:\Users\hp\OneDrive\Desktop\ai> task1.py
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\ai>
```

The right side of the interface features a 'CHAT' window titled 'CLARIFICATION ON PYTHON CODE REQUEST'. It contains a message from GitHub Copilot asking for clarification on the Python code request. Below the message, there are several options for interacting with the code, such as 'View existing code', 'Write new code', and 'Fix/debug code'. A message at the bottom encourages the user to let them know what they're trying to accomplish.

## Task 4: Comparative Analysis:-

### 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).**

<b>Criteria</b>	<b>Procedural (Task 1 &amp; 2)</b>	<b>Modular (Task 3)</b>
<b>Logic Clarity</b>	Linear and straightforward for very small tasks but becomes "spaghetti code" as complexity grows.	High clarity; the mathematical logic is isolated from the input/output logic.
<b>Reusability</b>	None. To use the logic elsewhere, the code must be manually copied and pasted.	High. The function can be imported into other Python files or called multiple times in one script.
<b>Debugging Ease</b>	Difficult. Errors in logic are mixed with errors in user input handling.	Simple. You can test the function with specific values (Unit Testing) to ensure the math is correct.

Criteria	Procedural (Task 1 & 2)	Modular (Task 3)
<b>Project Suitability</b>	Suitable only for small, one-off scripts or prototypes.	Essential for enterprise-level, large-scale software development.
<b>AI Dependency Risk</b>	High. AI might generate redundant variables or inefficient loops in long scripts.	Low. AI is highly specialized and accurate when asked to write specific, single-purpose functions.

### **Task 5: Iterative vs Recursive Thinking:-**

**: 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.**

