

Assignment - 1

Name: P.Pooja

Roll Number: 2303A510F7

Batch - 03

AI Assisted Coding

07-01-2026

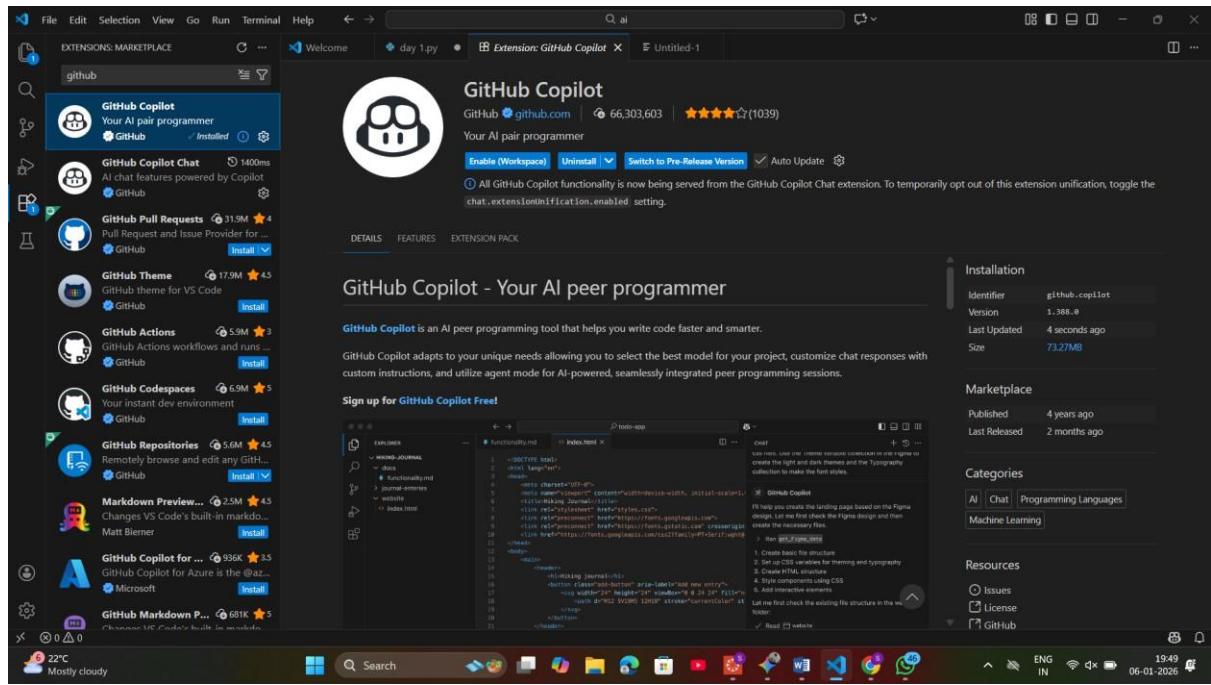
Task 0: Environment Setup:-

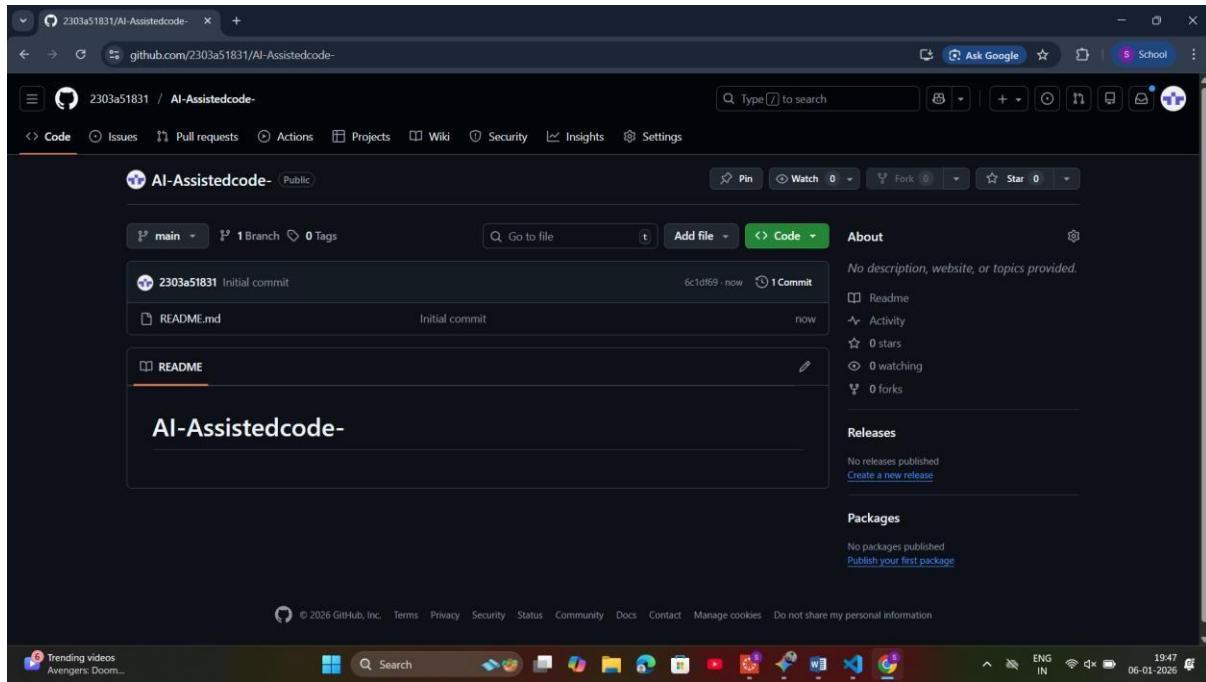
Task 0

- 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 window displaying Python code for calculating factorial:

```
C:\> Users > hp > OneDrive > 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}")
```

Below the code editor, the terminal window shows the execution of the script:

```
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

The right side of the interface features the AI Chat sidebar, which includes a message from the AI asking for clarification on Python code requests.

This screenshot shows the Microsoft Visual Studio Code interface with the terminal tab active. It displays the output of the previously run Python script:

```
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>
```

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. The Explorer sidebar on the left shows a folder named 'HPC' containing 'lab1.py'. The main editor area displays the following Python code:

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
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}")
```

The terminal at the bottom shows the output of running the script:

```
PS C:\Users\hp\OneDrive\Desktop\ai> & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions/ms-python.debugger-2025.18.0-win32-x64\bundled\libs\debug\launcher' '56820' '<1>' 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter a number: 120
PS C:\Users\hp\OneDrive\Desktop\ai>
```

A Chat window on the right side of the interface is open, showing a conversation about Python code optimization. The user has asked for clarification on a Python code request, and the AI has responded with several options and a note to let them know what they're trying to accomplish.

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.

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The code editor displays a Python script named `task1.py` with the following content:

```
C:\Users\hp>OneDrive>Desktop>ai> 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 terminal below shows the execution of the script:

```
PS C:\Users\hp\OneDrive\Desktop\HPC>
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\python3.11.exe" "C:\Users\hp\vscode\extensions\ms-python.python\debug-2025.18.0-win32-x64\build\lib\debug\launcher" "65497" ... "C:\Users\hp\OneDrive\Desktop\ai\task1.py"
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

The status bar at the bottom indicates the file is indexed and shows system information like weather (28°C, sunny), battery level (80%), and system date (07-01-2026).

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).**

Criteria	Procedural (Task 1 & 2)	Modular (Task 3)
Logic Clarity	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.
Reusability	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.
Debugging Ease	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)
Project Suitability	Suitable only for small, one-off scripts or prototypes.	Essential for enterprise-level, large-scale software development.
AI Dependency Risk	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.

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The left sidebar includes the Explorer, Search, and Activity Bar sections. The main area displays two Python files: `lab1.py` and `task1.py`. The `task1.py` file contains the following code:

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > factorial_iterative
1 def factorial_iterative(n):
2     res = 1
3     for i in range(2, n + 1):
4         res *= i
5     return res
6
7 def factorial_recursive(n):
8     if n == 0 or n == 1:
9         return 1
10    return n * factorial_recursive(n - 1)
```

The terminal below shows the execution of the code:

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

A Chat window on the right side shows a user asking for clarification on Python code requests, and the AI assistant responding with options like View existing code, Write new code, Fix/debug code, Run code, and Explain code.