

# Assignment - 1

Name: K.Priya

Roll Number: 2303A51194

Batch - 04

AI Assisted Coding

07-01-2026

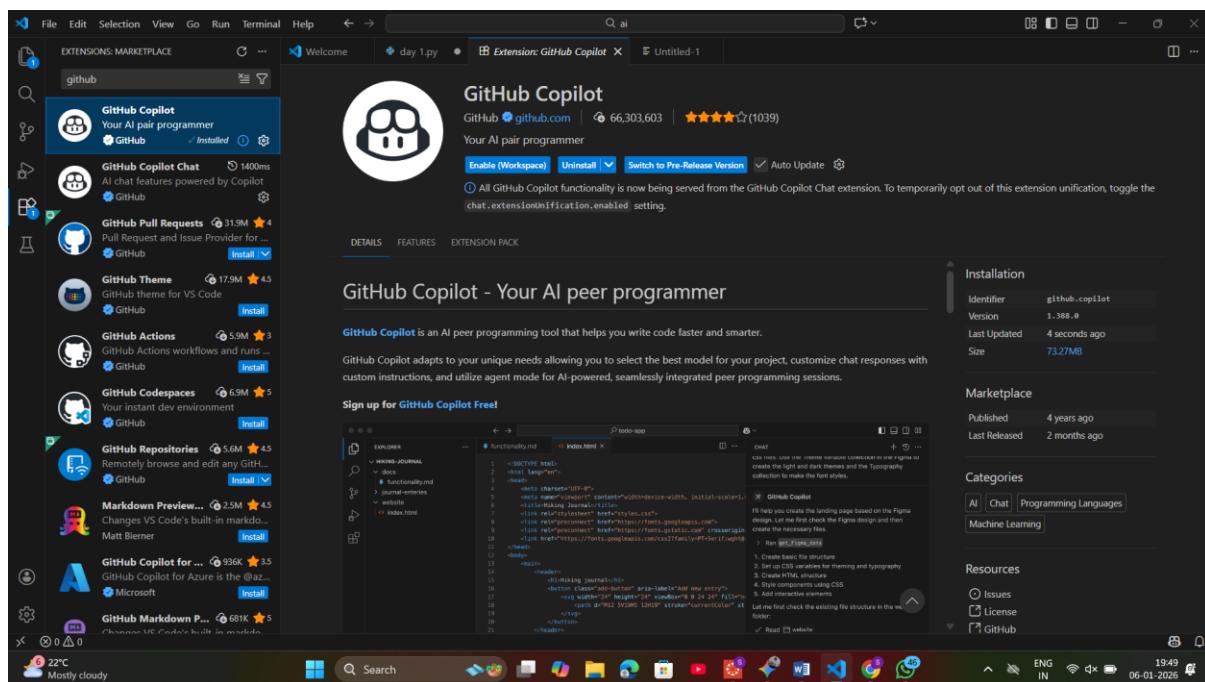
## Task 0: Environment Setup:-

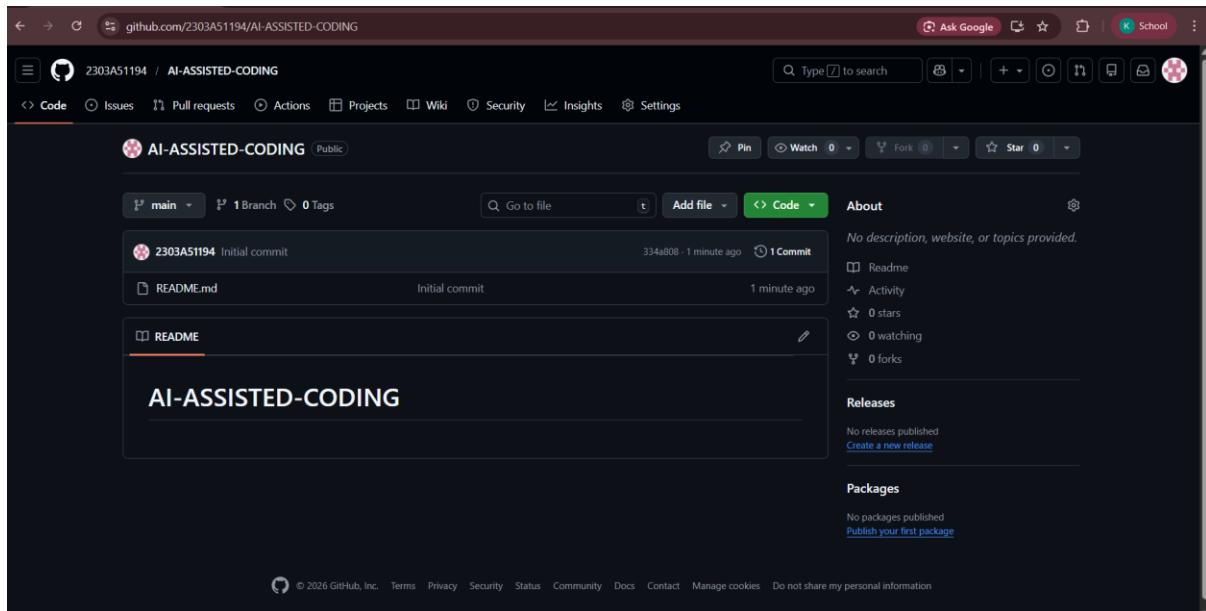
### Task 0

- 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 > 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}")

```

To the right of the code editor, there is a sidebar titled "CHAT" which contains the following text:

- Asked for clarification on Python code request
- I'd be happy to help! However, your request is unclear. Could you clarify what you'd like me to do?
- For example:
  - 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?
- Please let me know what you're trying to accomplish!

At the bottom of the interface, there is a terminal window showing the execution of the code:

```

Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC> ~c
PS C:\Users\hp\OneDrive\Desktop\HPC>
PS C:\Users\hp\OneDrive\Desktop\HPC> c; cd 'c:\Users\hp\OneDrive\Desktop\HPC'; & "c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe" '<:vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62630' '' 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

The screenshot shows the Microsoft Visual Studio Code interface with the terminal tab active. The terminal window displays the following output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... X
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 code editor displays a Python script named `task1.py` with the following content:

```

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("Factorial: ", factorial)

```

The terminal below shows the execution of the script:

```

PS C:\Users\hp\OneDrive\Desktop\HPC> & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.0-x64\python.exe' 'c:\Users\hp\vscode\extensions\ms-python.python.debugpy-2025.18.0-win32-x64\bundled\libs\debug\launcher' "56828" >> "C:\Users\hp\OneDrive\Desktop\HPC\task1.py"
Enter a number: 5
Factorial: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

The status bar at the bottom indicates indexing completed and the system is at 28°C.

The right side of the interface features the AI Chat feature. A message from the AI says: "Asked for clarification on Python code request. I'd be happy to help! However, your request is unclear. Could you clarify what you'd like me to do?". Below this, a list of options is provided:

- 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 AI panel, there is a prompt: "Please let me know what you're trying to accomplish!" followed by a text input field containing "task1.py" and a button labeled "Describe what to build next".

## 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 code editor displays a Python script named `task1.py` with the following content:

```
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 command-line output of running the script:

```
PS C:\Users\hp\OneDrive\Desktop\HPC> task1.py
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.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65497' --> 'c:\Users\hp\OneDrive\Desktop\HPC>
Enter number: 120
Result: 479001600
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

The status bar at the bottom indicates the system is at 28°C and sunny. A floating panel on the right provides a "Clarification on Python code request" and a "Build" interface for the file `task1.py`.

## 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)
<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.**

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The Explorer sidebar on the left shows a project named "HPC" containing a file "lab1.py". The main editor area displays two Python files: "factorial\_iterative.py" and "factorial\_recursive.py".

```
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
```

The status bar at the bottom indicates indexing completed, the current file is "task1.py", and the system status is 28°C, Sunny.

A Chat window on the right side of the interface is open, showing a conversation about Python code requests. It includes a list of common code-related questions and a message asking for clarification on the user's request.