# AI ASSISTED CODING

**Sai Thrishool** **2303A51127**

**BATCH – 03** **07 – 01 – 2026**

---

## ASSIGNMENT – 1.1

**Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow Lab Objectives:**

**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.**

**Lab Outcomes (LOs):**

**After completing this lab, students will be able to:**

**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.**

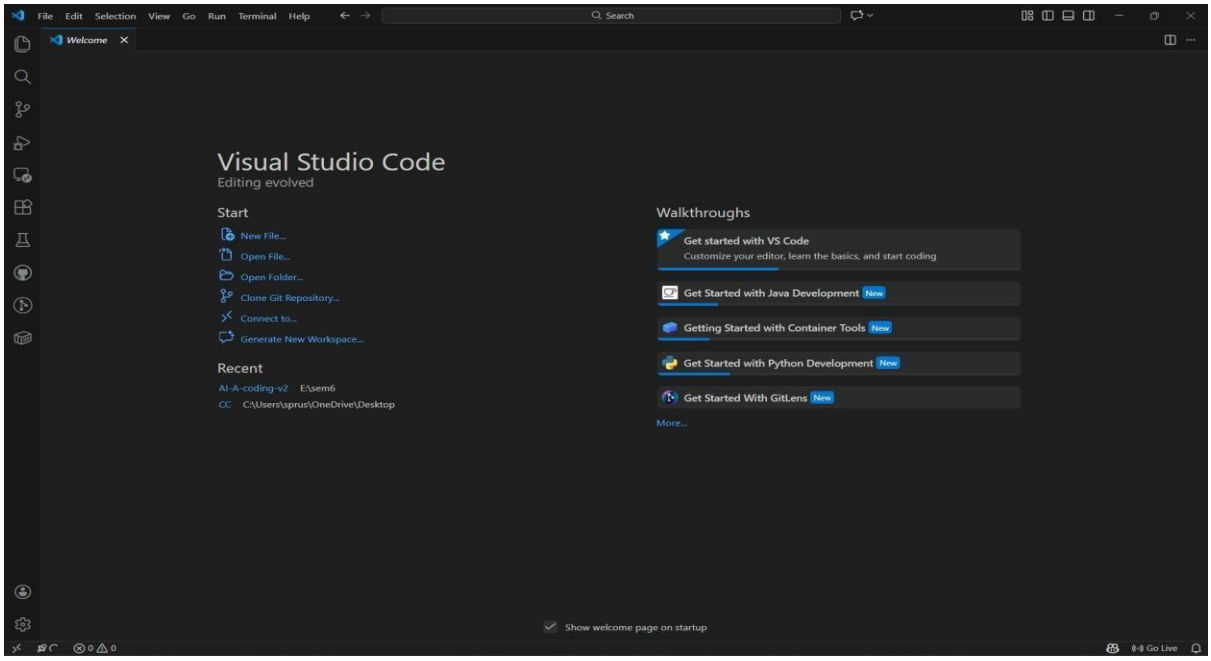
**Task 0**

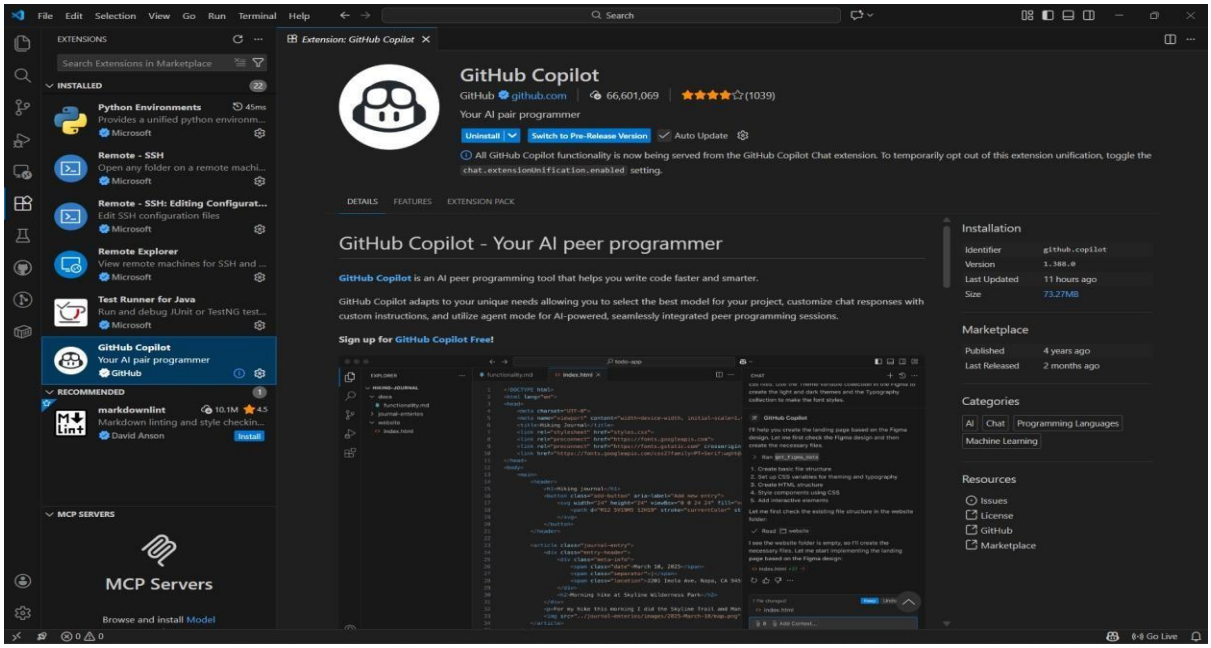
- **Install and configure GitHub Copilot in VS Code. Take screenshots of each step.**

**Expected Output**

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

**Task 0: Environment Setup:-**




## Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

- **Scenario**

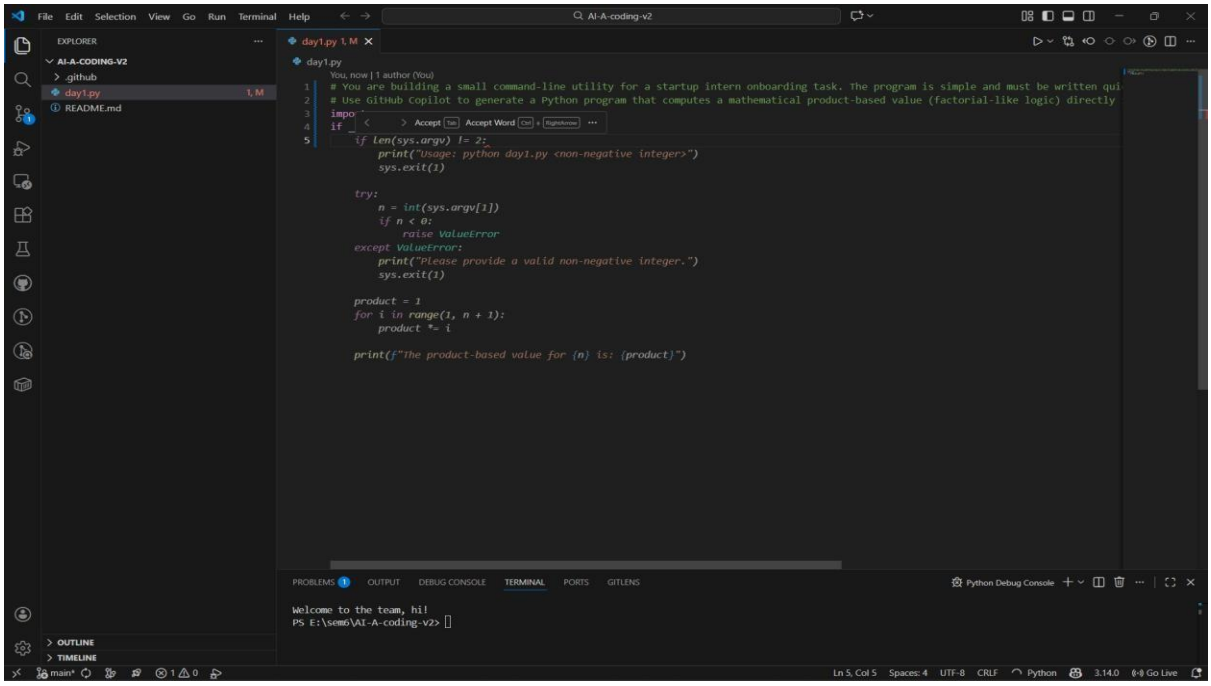
**You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.**

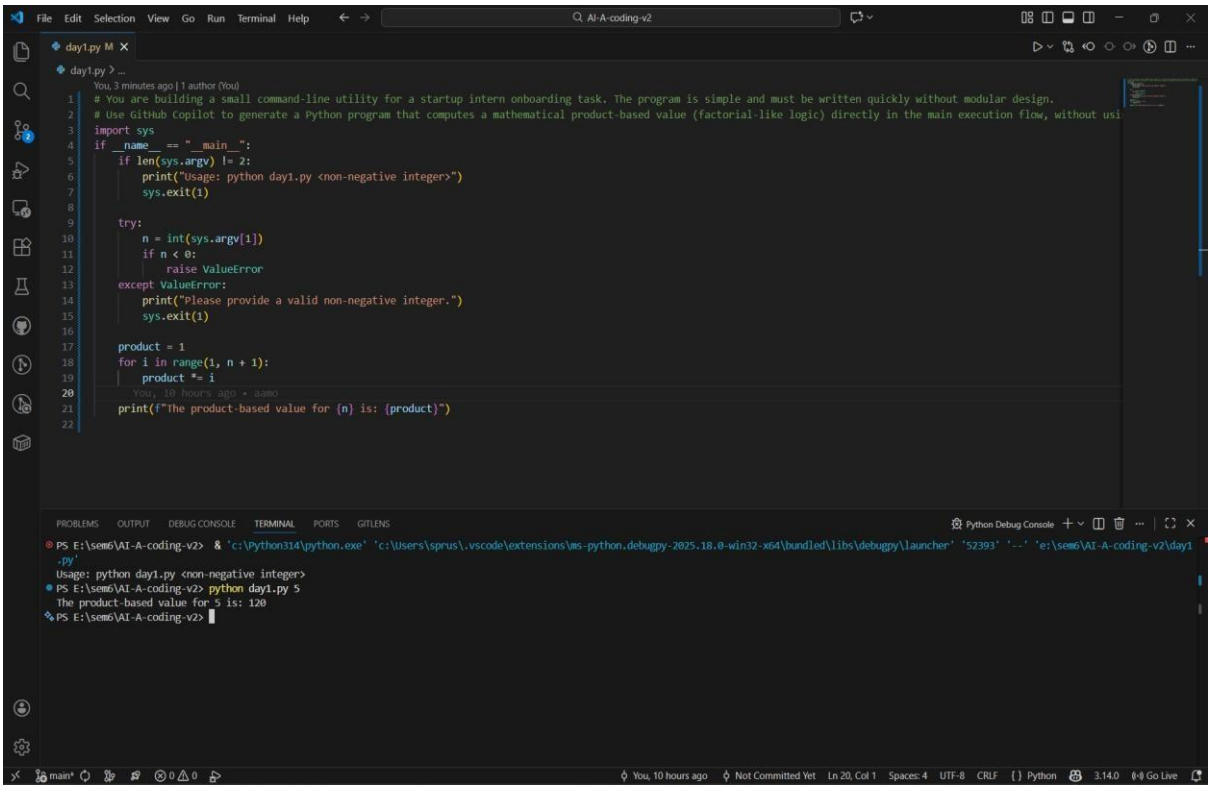
- **Task Description**

**Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factoriallike 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?**

**Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)**




## 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**

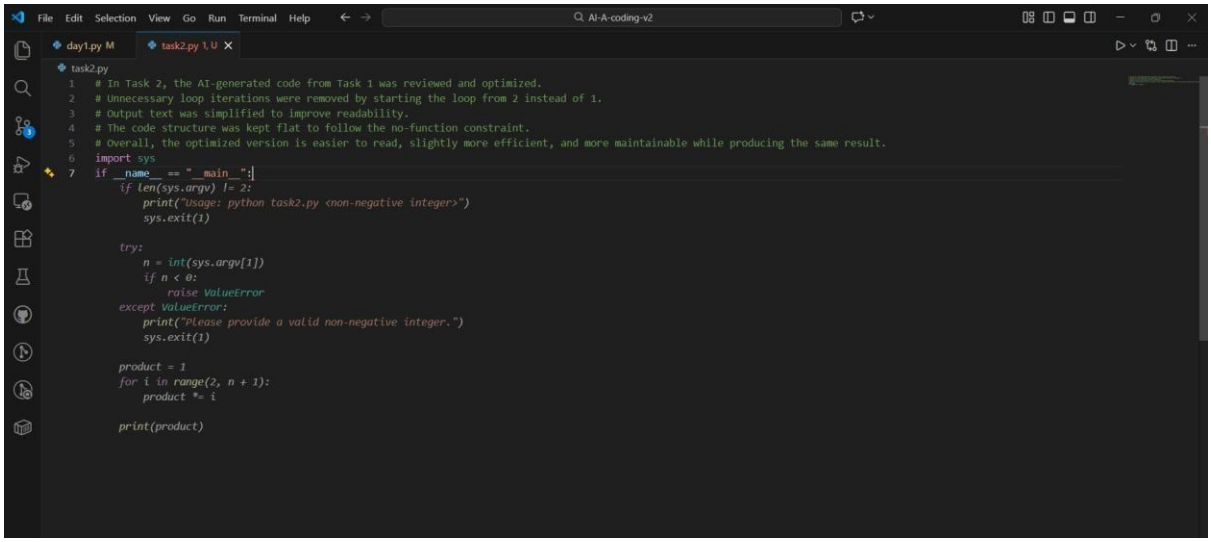
➢ **Original AI-generated code**

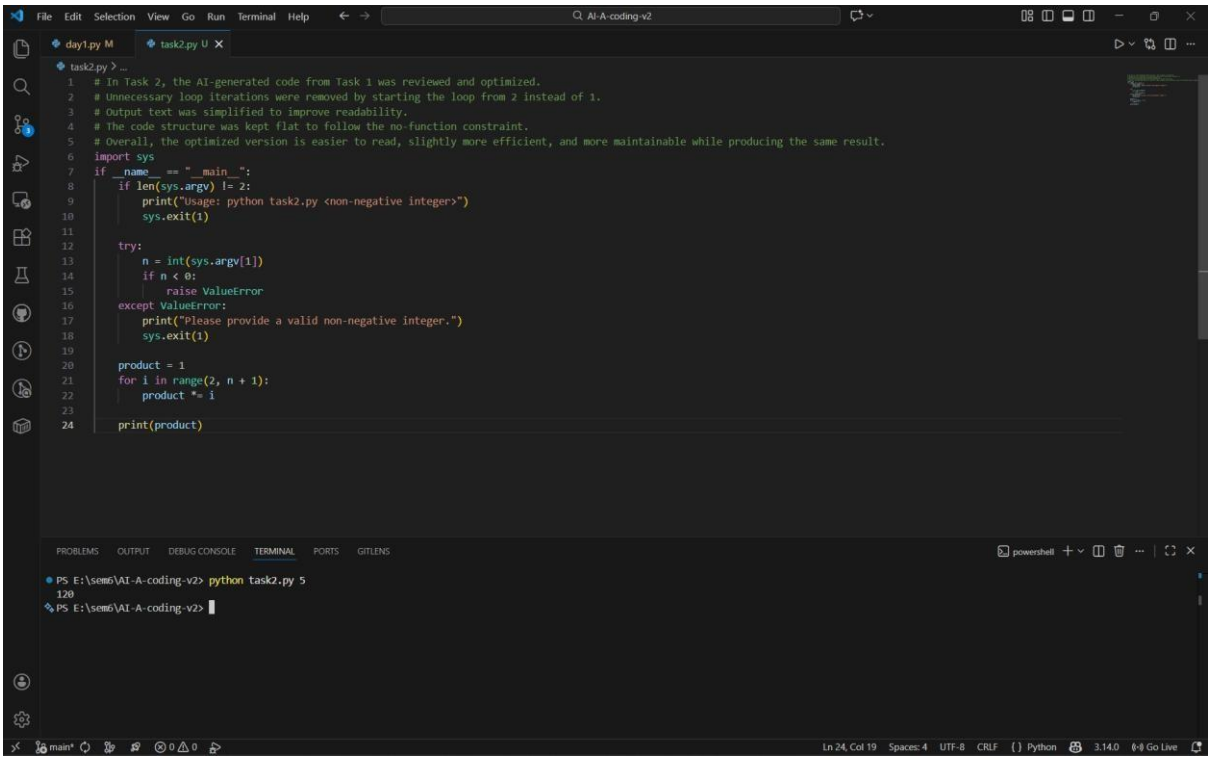
➢ **Optimized version of the same code** ➢ **Side-by-side comparison** ➢ **Written explanation:**

▪ **What was improved?**

▪ **Why the new version is better (readability, performance, maintainability.**

**Task 2: AI Code Optimization & Cleanup (Improving Efficiency)**




## 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

**# Task 3: Modular Design Using AI Assistance (Factorial with Functions)**

**➢ Use meaningful function and variable names**

**➢ Include inline comments (preferably suggested by Copilot)**

**❖ Expected Deliverables**

**➢ AI-assisted function-based program ➢**

**Screenshots showing: o Prompt**

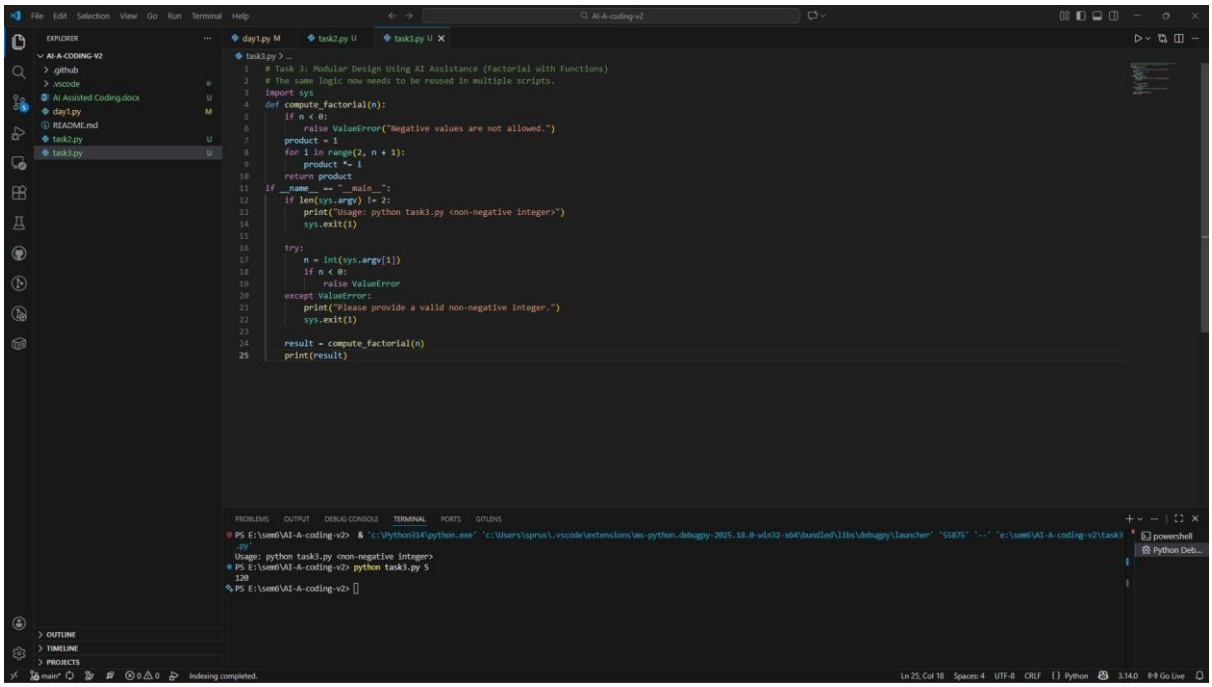
**evolution o Copilot-generated function**

**logic**

**➢ Sample inputs/outputs**

**➢ Short note: o How modularity improves**

**reusability.**




**Short Note: How Modularity Improves Reusability**

**Modularity helps in reusability by helping separate logic in terms of different functions which may be reused in multiple programs. The factorial computation is put in a function which makes the code easier to maintain and test. If the logic has to be changed, changes can be made at**

**one place without having any impact on the whole program. Modular code is also more readable and easier to work on in a team environment.**

**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**

**Copilotgenerated 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).**

**Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)**

**Task 5: AI-Generated Iterative vs Recursive Thinking**

**❖ Scenario**

**Your mentor wants to test how well AI**

**understands different computational paradigms.**

| Criteria | Procedural Code (Without Functions) | Modular Code (With Functions) |
|---|---|---|
| Logic Clarity | Logic is written in one continuous flow, which is easy to follow for very small programs but becomes cluttered as code grows. | Logic is clearly separated into functions, making the purpose of each part easier to understand. |
| Reusability | Code cannot be reused easily because the logic is tightly coupled to the main execution block. | Function-based logic can be reused across multiple scripts by importing or calling the function. |
| Debugging Ease | Debugging is harder since all logic exists in one block, making it difficult to isolate issues. | Debugging is easier because errors can be traced to specific functions. |
| Suitability for Large Projects | Not suitable for large projects as it leads to poor organization and low maintainability. | Well suited for large projects due to better structure, scalability, and teamwork support. |
| AI Dependency Risk | High risk, as beginners may copy AIgenerated code without understanding the full flow. | Lower risk, as modular structure encourages understanding of individual components. |

❖ 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

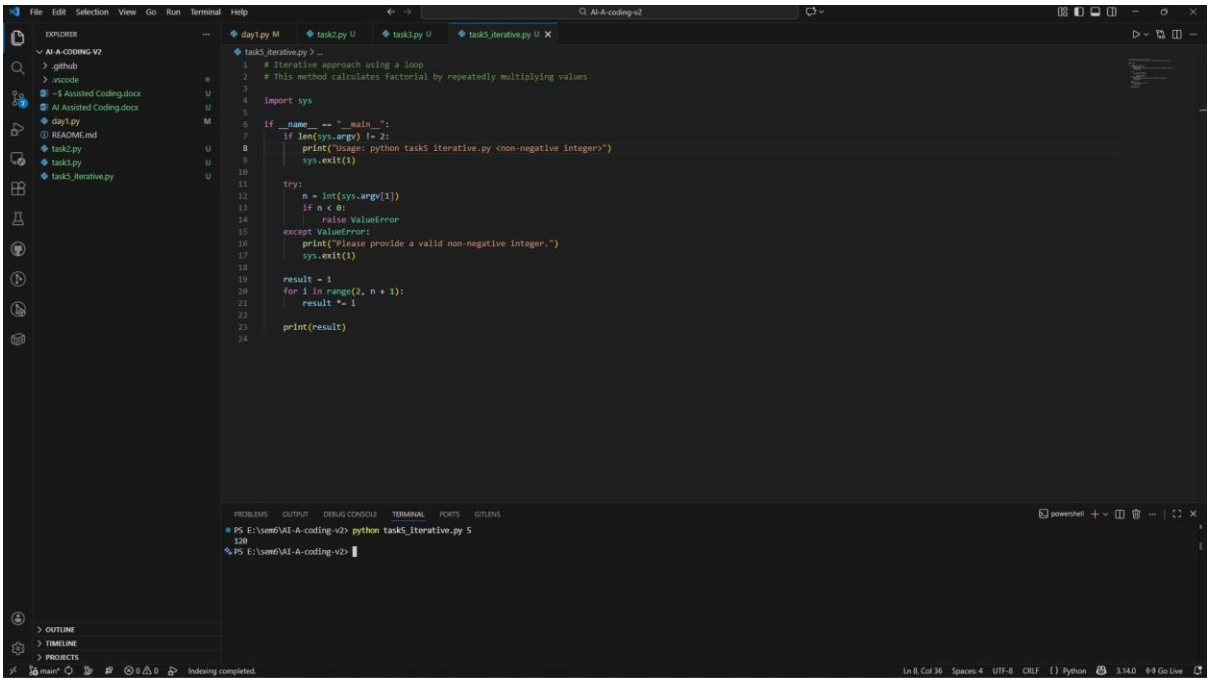
❖ Expected Deliverables

Two AI-generated implementations

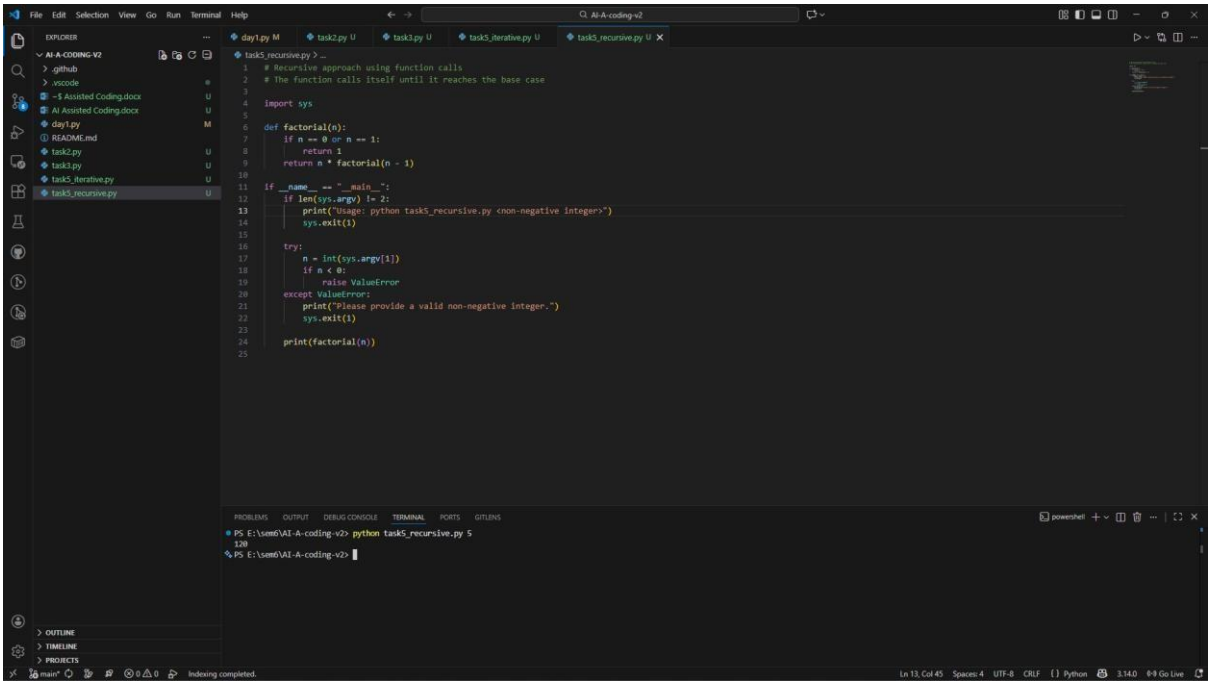
**Execution flow explanation (in your own words)**
**Comparison covering:**
**➢ Readability**
**➢ Stack usage**
**➢ Performance implications**
**➢ When recursion is not recommended.**

## Task 5: AI-Generated Iterative vs Recursive Thinking

**Iterative Thinking –**



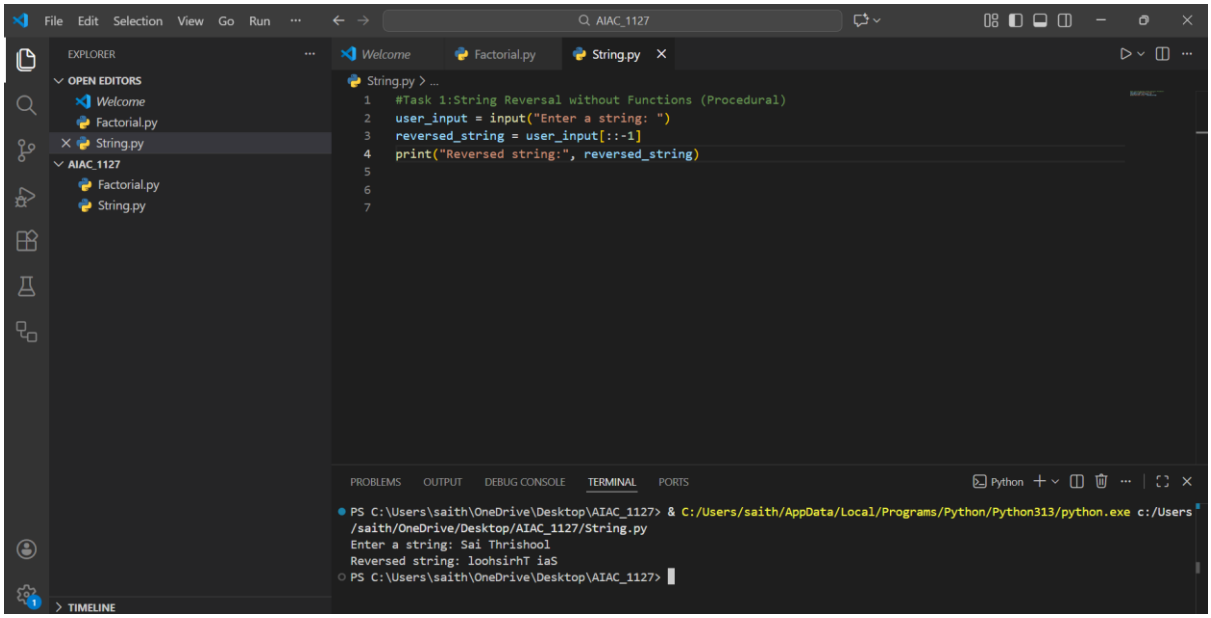

**Recursive thinking –**



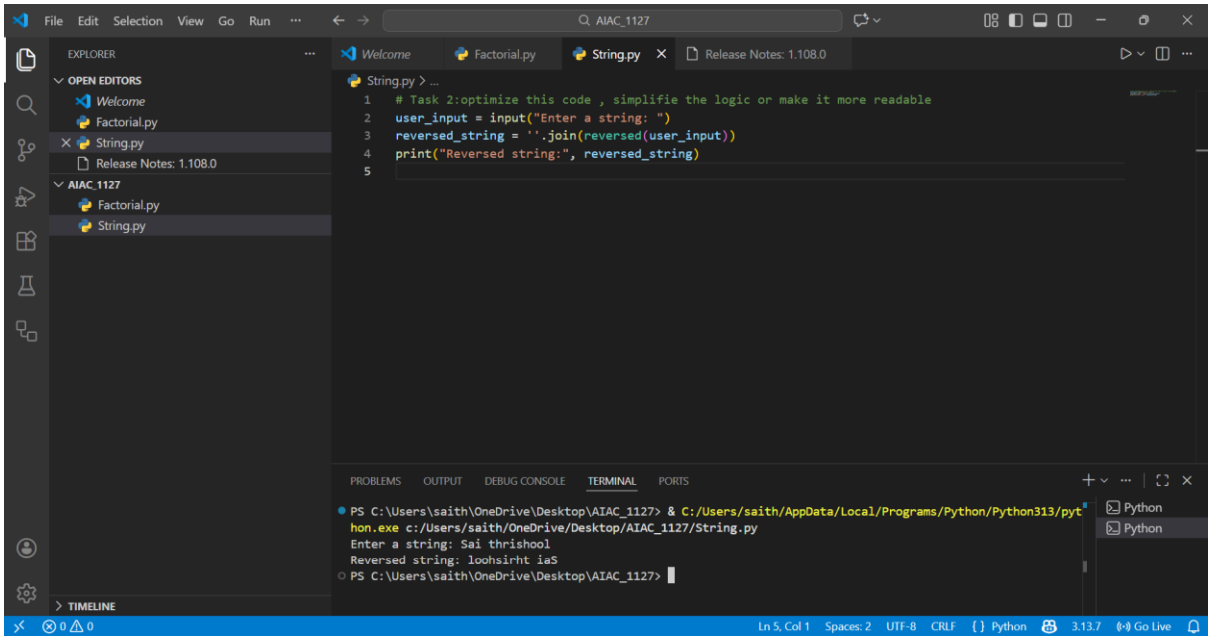

| Aspect | Iterative Approach | Recursive Approach |
|---|---|---|
| Readability | Easy to understand for beginners | More mathematical and elegant |
| Stack Usage | Uses constant memory | Uses additional stack memory |
| Performance | Faster and memory efficient | Slower for large inputs |
| Error Risk | Low | Risk of stack overflow |
| When Not Recommended | — | Not recommended for large input values |

## ASSIGNMENT – 1.5

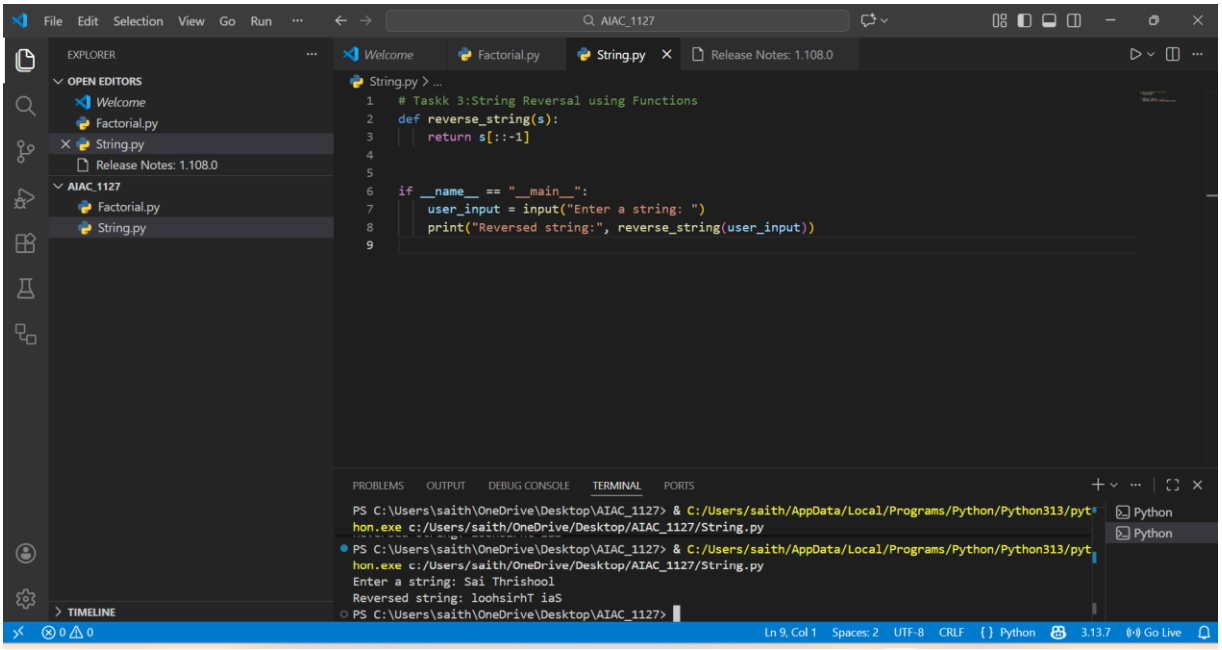
## TASK 1: String Reversal without Functions (Procedural)




## TASK 2: Optimize this code , simplify the logic or make it more readable.




## TASK 3: String Reversal using Functions.




## TASK 4: Comparative Analysis – Procedural VS Modular Approach.

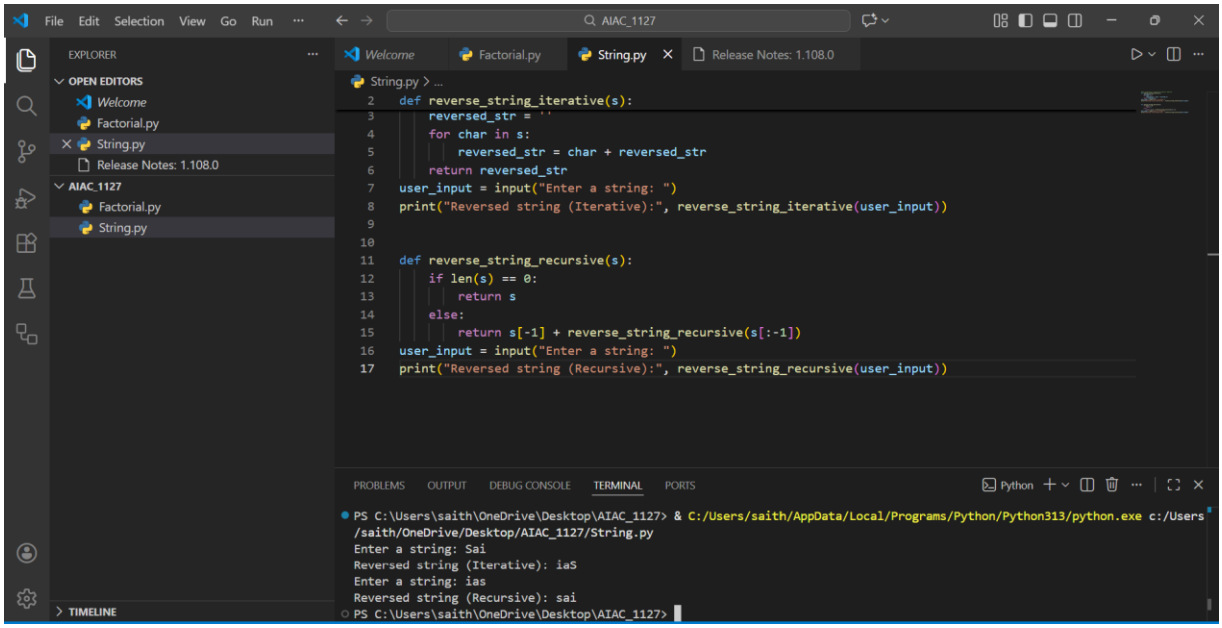
### Procedural Approach (Without Functions):

The string reversal logic is implemented directly in the main code. While this approach is simple and easy to understand for small programs, it lacks reusability and scalability. Debugging becomes difficult as the program grows because all logic is tightly coupled.

### Modular Approach (With Functions):

The function-based implementation separates logic from execution, improving code readability and structure. The function can be reused in multiple parts of an application, making it suitable for large-scale systems. Debugging and maintenance are easier due to isolated logic.

## TASK 5: Iterative vs Recursive Fibonacci Approach.


File Edit Selection View Go Run
AIAC_1127
EXPLORER
OPEN EDITORS
Welcome
Factorial.py
String.py
Release Notes: 1.108.0
AIAC_1127
Factorial.py
String.py
TIMELINE
Welcome
Factorial.py
String.py
Release Notes: 1.108.0
String.py > ...
2 def reverse_string_iterative(s):
3 reversed_str = ''
4 for char in s:
5 reversed_str = char + reversed_str
6 return reversed_str
7 user_input = input("Enter a string: ")
8 print("Reversed string (Iterative):", reverse_string_iterative(user_input))
9
10
11 def reverse_string_recursive(s):
12 if len(s) == 0:
13 return s
14 else:
15 return s[-1] + reverse_string_recursive(s[:-1])
16 user_input = input("Enter a string: ")
17 print("Reversed string (Recursive):", reverse_string_recursive(user_input))
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\Users\saith\OneDrive\Desktop\AIAC_1127> & C:/Users/saith/AppData/Local/Programs/Python/Python313/python.exe c:/Users/saith/OneDrive/Desktop/AIAC_1127/String.py
Enter a string: Sai
Reversed string (Iterative): iaS
Enter a string: ias
Reversed string (Recursive): sai
PS C:\Users\saith\OneDrive\Desktop\AIAC_1127>

**THANK YOU!!**