AI Assisted Coding

Assignment Number:1.3

Name:Chirra Jyothika

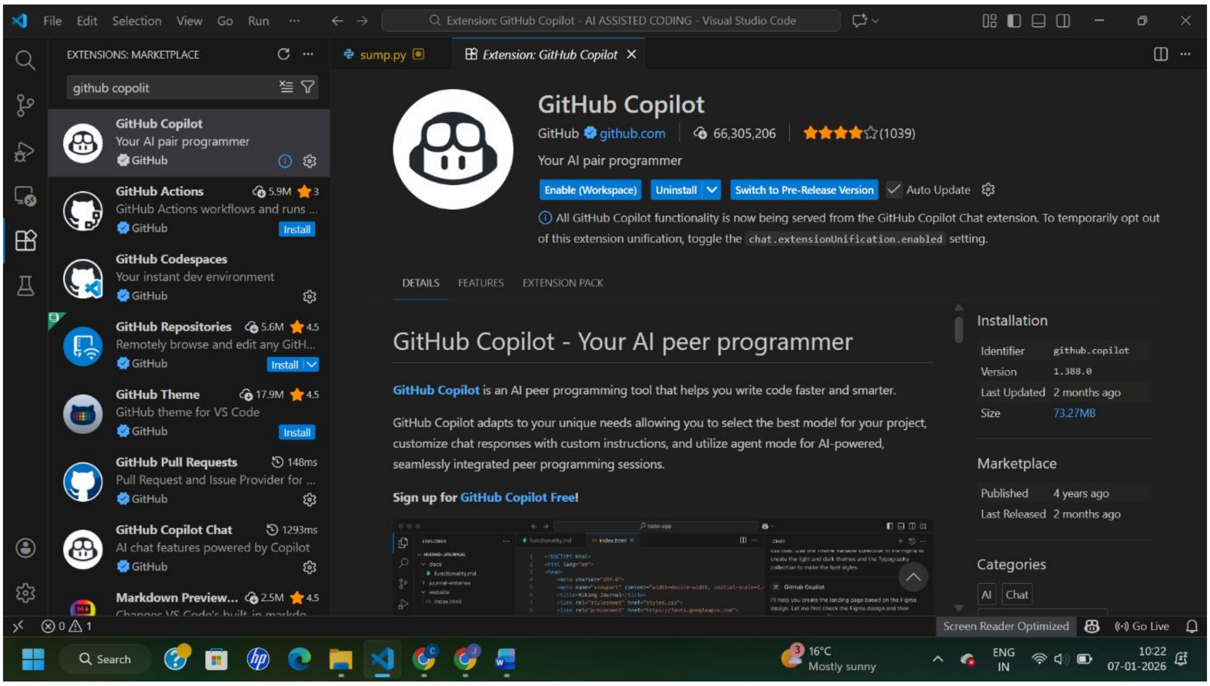
HtNo:2303A51280

BtNo:05

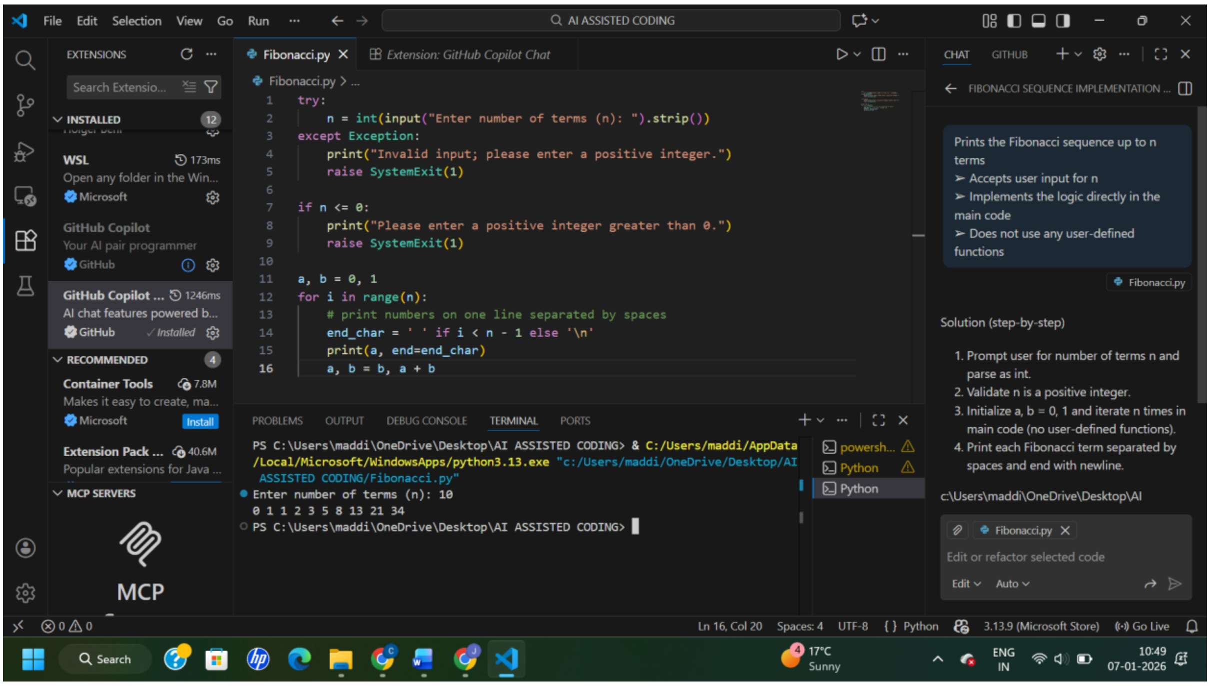
Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

<u>Task 0:</u>

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

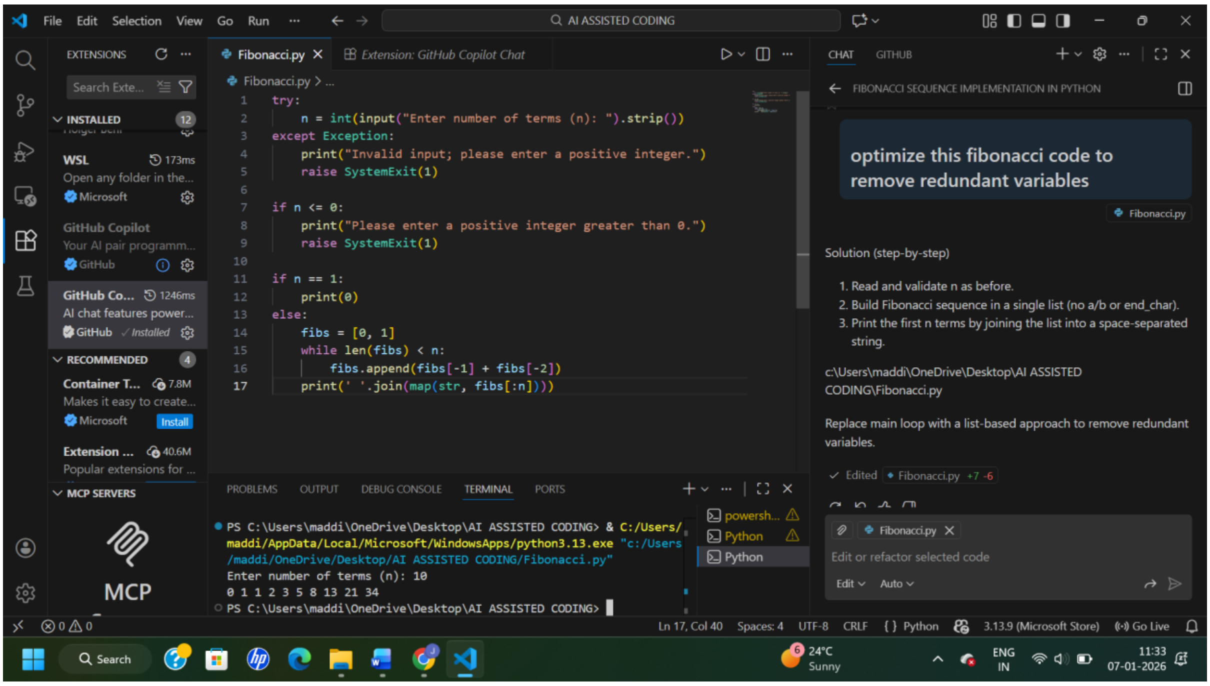


Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)

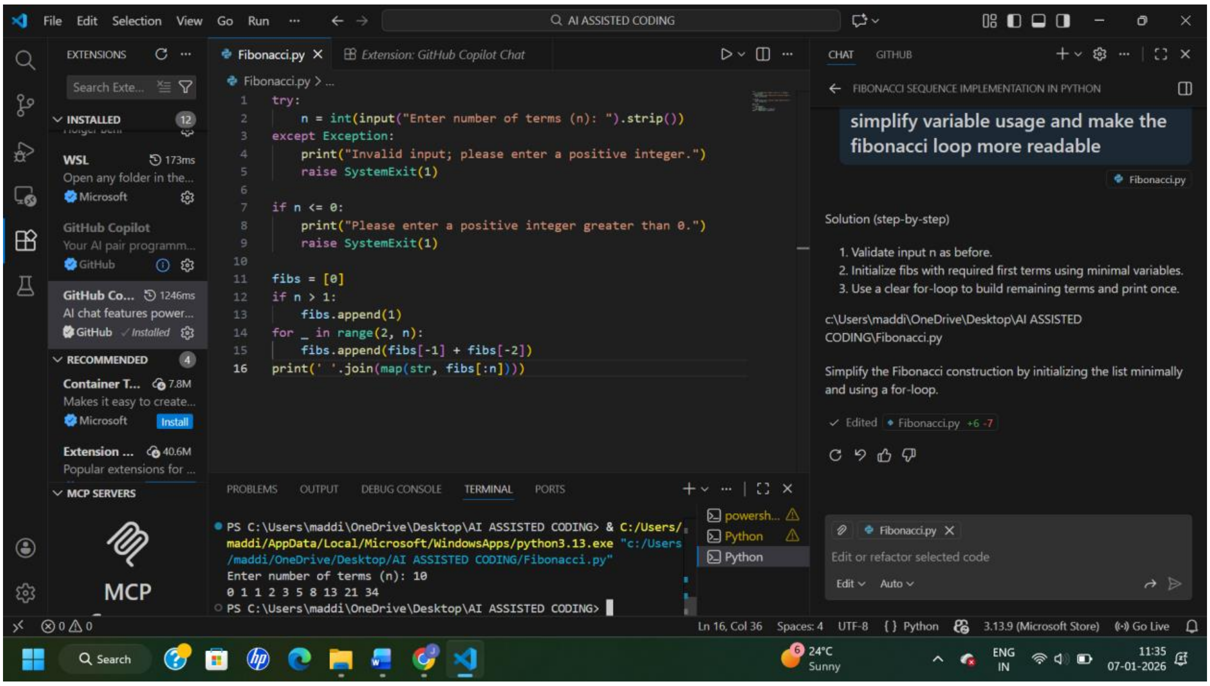


Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

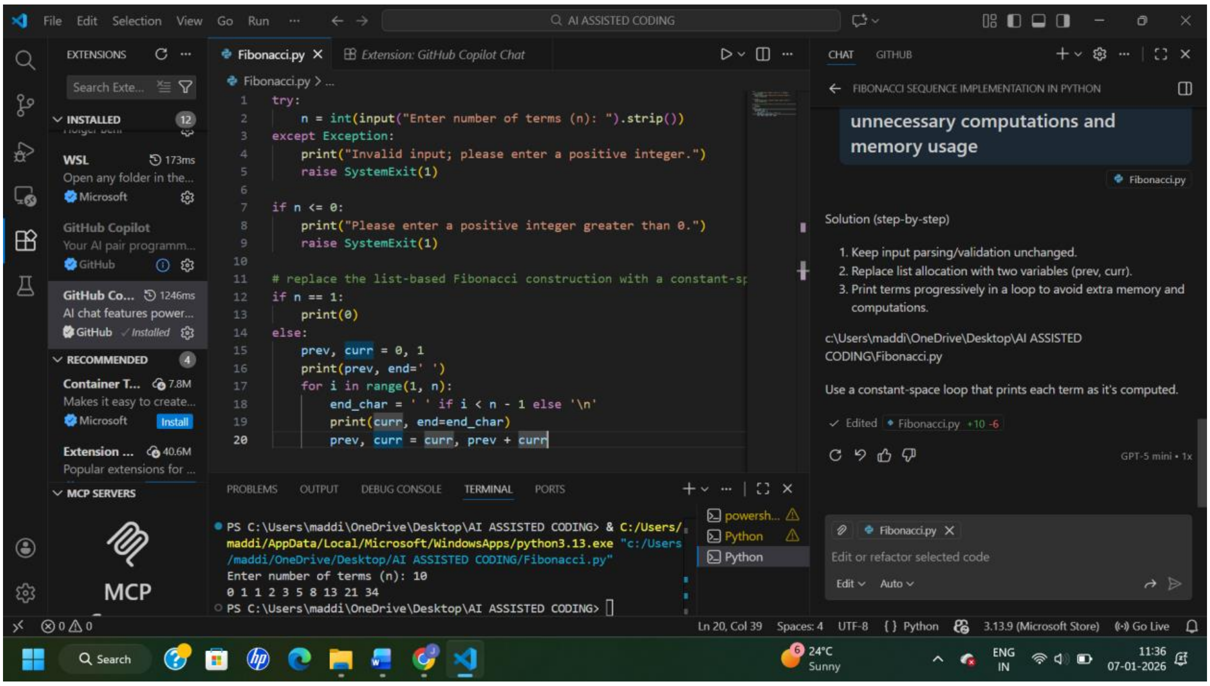
optimize this fibonacci code to remove redundant variables



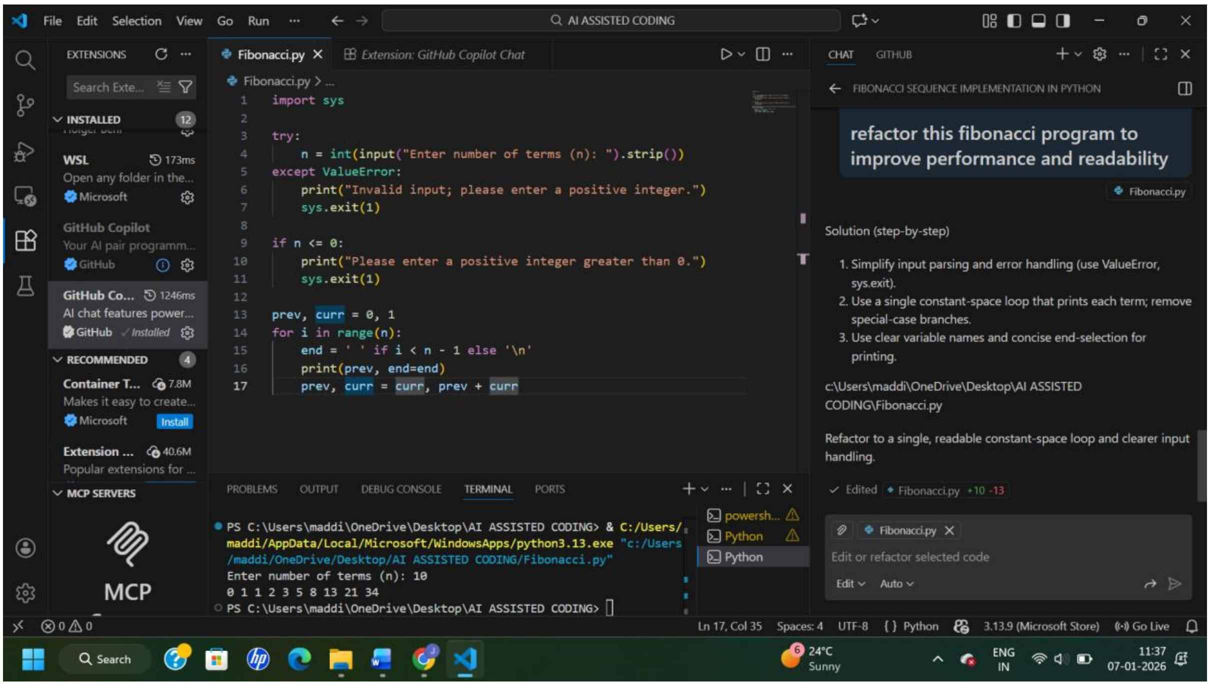
simplify variable usage and make the fibonacci loop more readable



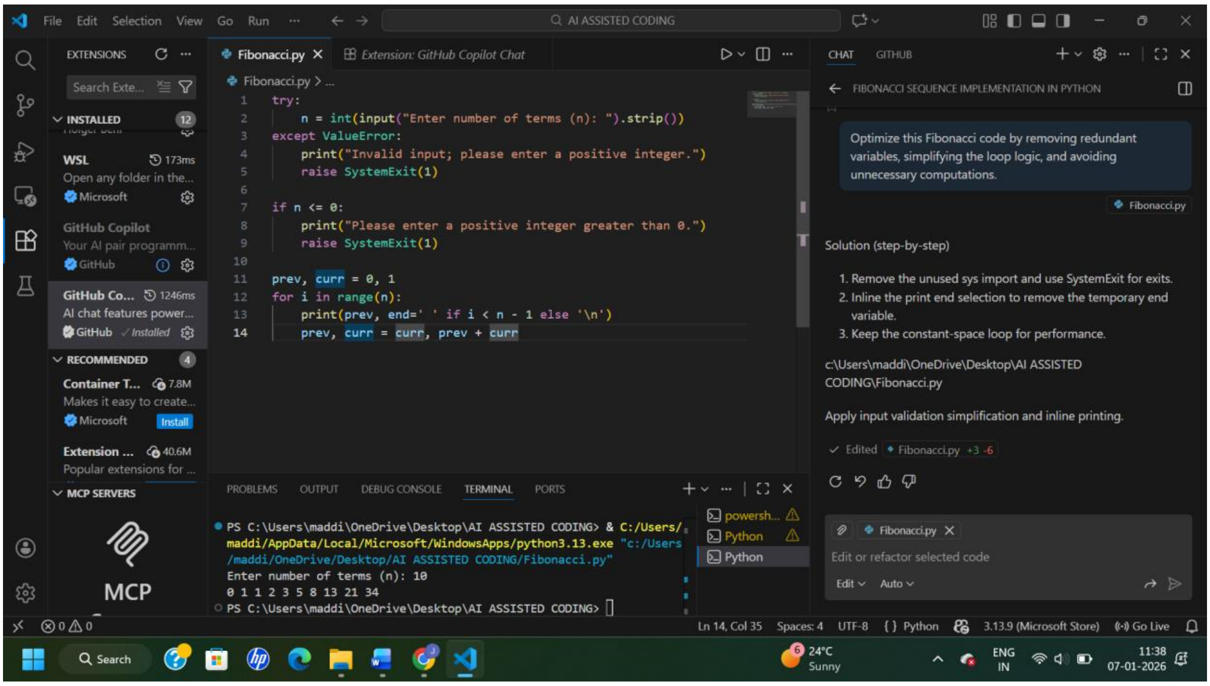
optimize this code to avoid unnecessary computations and memory usage



refactor this fibonacci program to improve performance and readability



#Optimize this Fibonacci code by removing redundant variables, simplifying the loop logic, and avoiding unnecessary computations.



What Was Inefficient in the Original Code

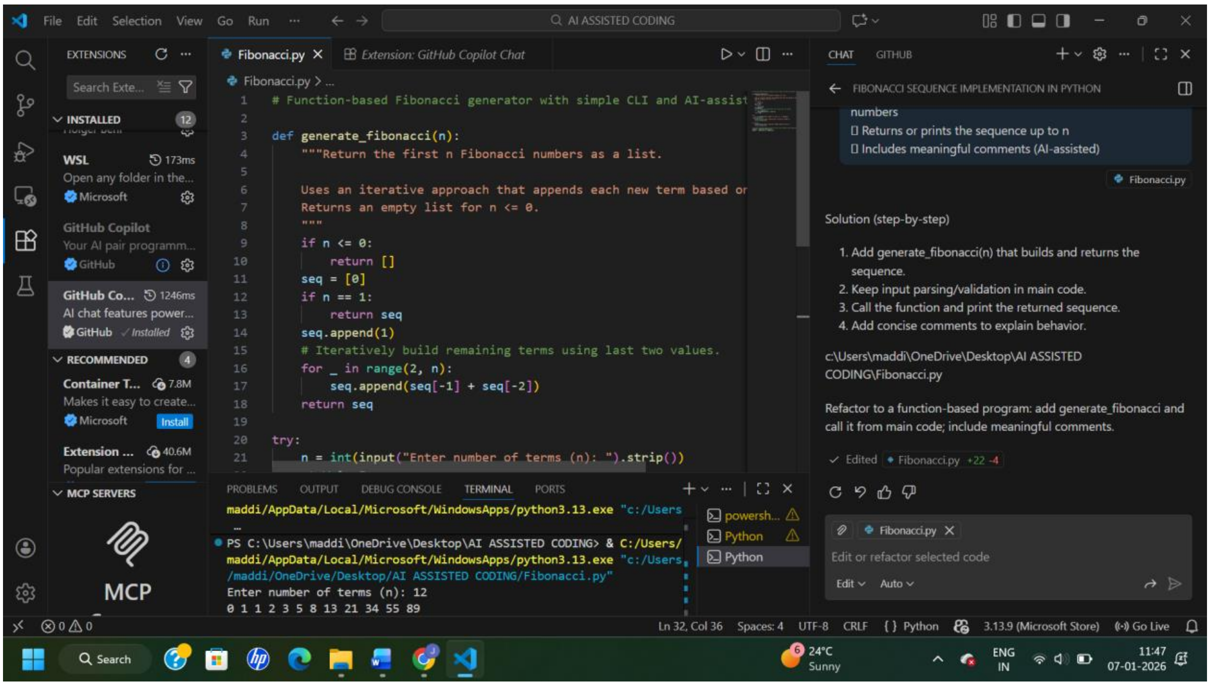
- The original code used **extra variables** that were not necessary to generate the Fibonacci sequence.
- A **temporary variable** was used to store the sum of numbers, which increased code length.
- The program stored the Fibonacci sequence in a **list**, even though the requirement was only to display the output.
- The loop logic was written in **multiple steps**, making the code less readable

How the Optimized Version Improves Performance and Readability

- Redundant variables were removed, and **tuple assignment** was used to update values efficiently.
- The optimized code prints values directly instead of storing them, which **reduces memory usage**.
- The loop logic was simplified, making the code **shorter and easier to understand**.

- Overall performance was improved by using **constant memory** and cleaner logic.

<u>Task 3</u>: Modular Design Using AI Assistance (Fibonacci Using Functions)



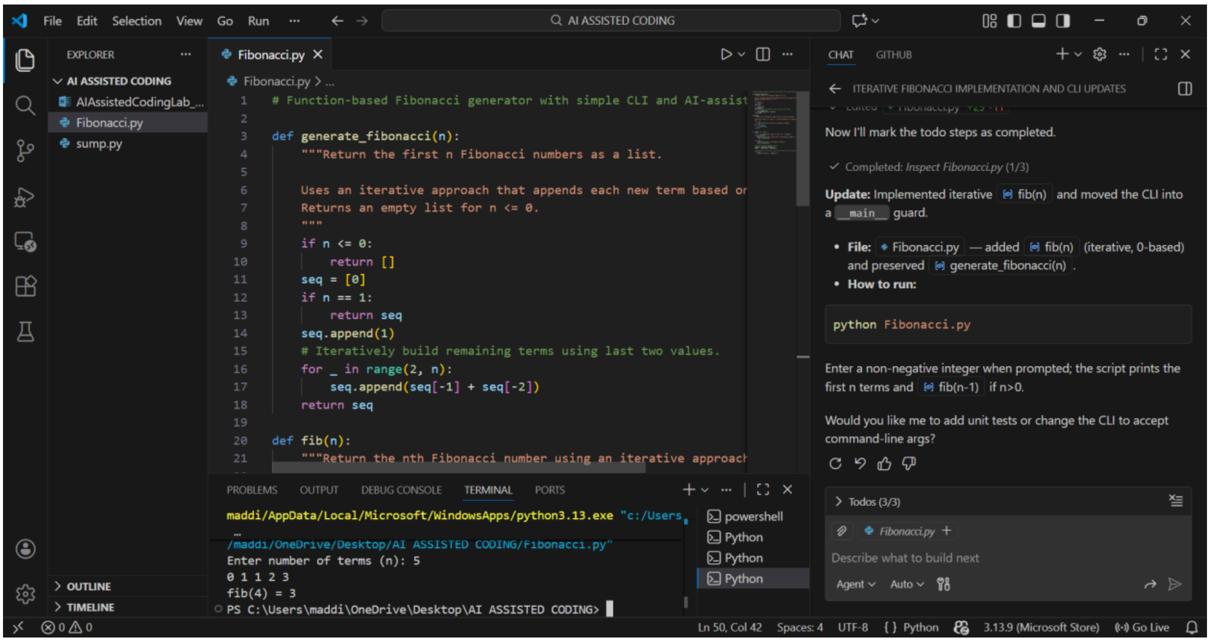
Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Feature	**Without Functions**	**With Functions**
Code Clarity	Logic is written in one block; harder to read when long	Logic is separated into a named function → easier to understand
Reusability	Cannot reuse Fibonacci logic without rewriting	Can call the function anywhere in program
Debugging Ease	Bugs must be traced in main logic, mixed with other code	Errors isolated in function → easier to test & fix
Suitability for Larger Systems	Poor; not scalable, becomes messy with added features	Good; fits into bigger systems, easier to maintain
Testing	Hard to unit test a part of code independently	Function can be tested separately with multiple inputs

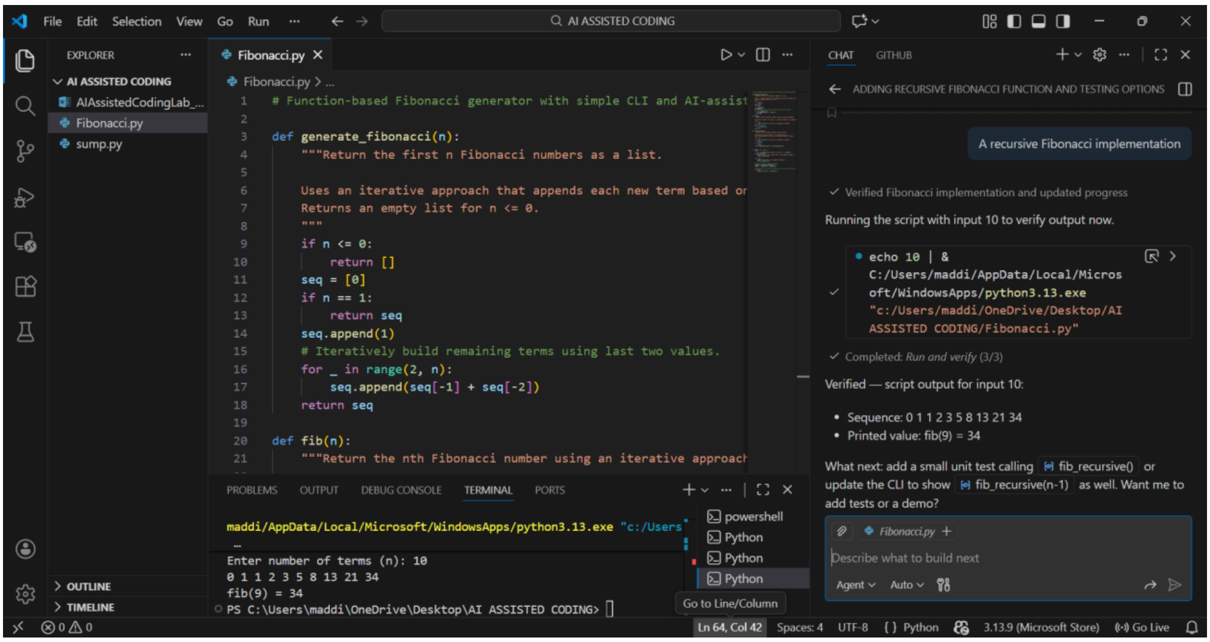
Maintainability	Low; changes affect entire code block	High; changes only in function, no impact on main flow
Performance Impact	No function call overhead (very small benefit)	Minimal overhead but worth it for structure & scaling

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

An iterative Fibonacci implementation



A recursive Fibonacci implementation



Comparison covering:

Aspect	Iterative	Recursive
Time Complexity	O(n)	O(2^n) (very slow due to repeated calls)
Space Complexity	O(1)	O(n) (stack memory for calls)
Performance for Large n	Excellent (can handle 10^7+ if needed)	Poor (fib(50) may take seconds/minutes)
Memory Usage	Very low	High because of recursion stack
Scalability	Best for real systems	Not scalable without optimization
Risk	No crash risk	StackOverflow for large n