AI ASSISTED CODING

LAB-I

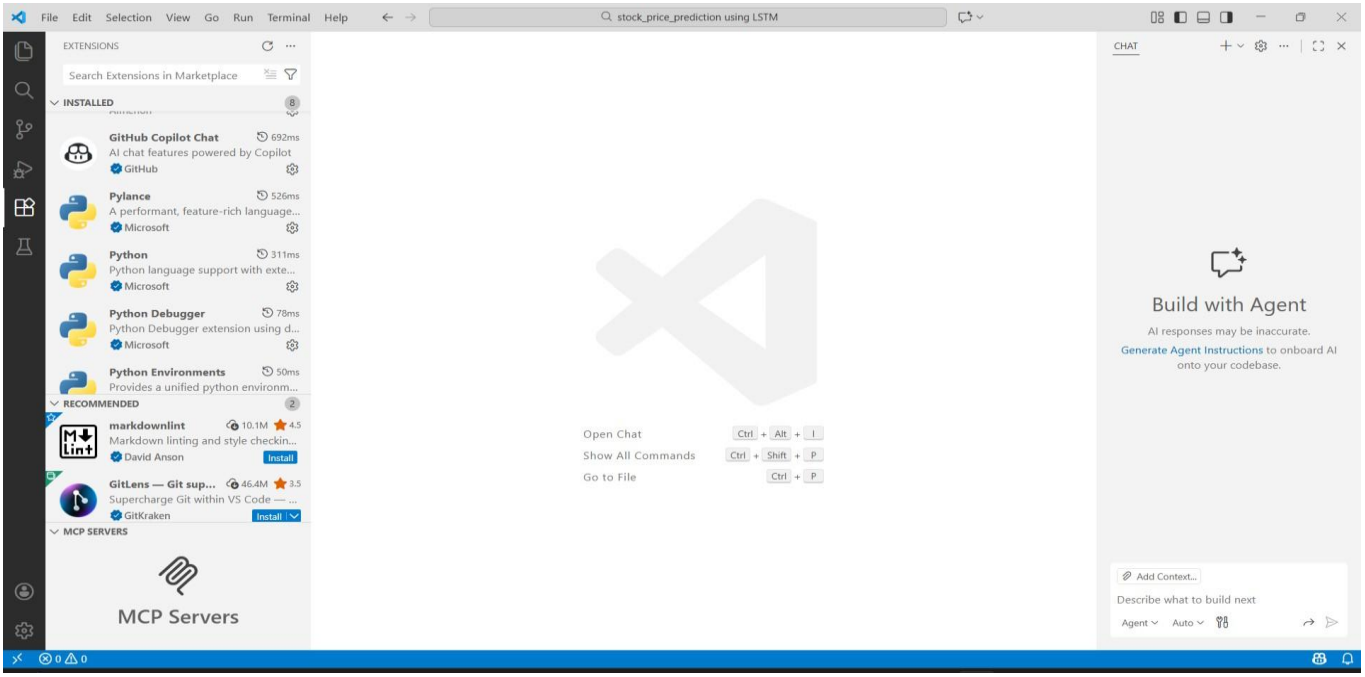
ASSIGNMENT-1

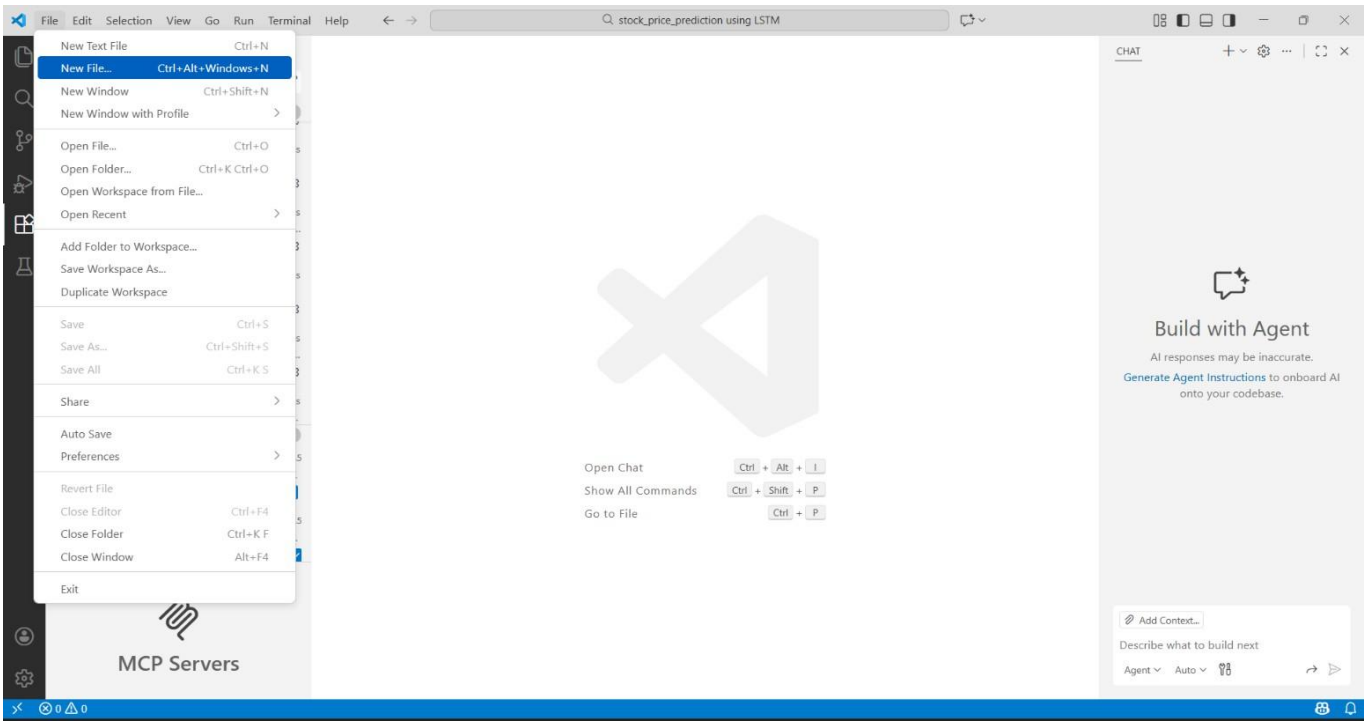
BATCH:28

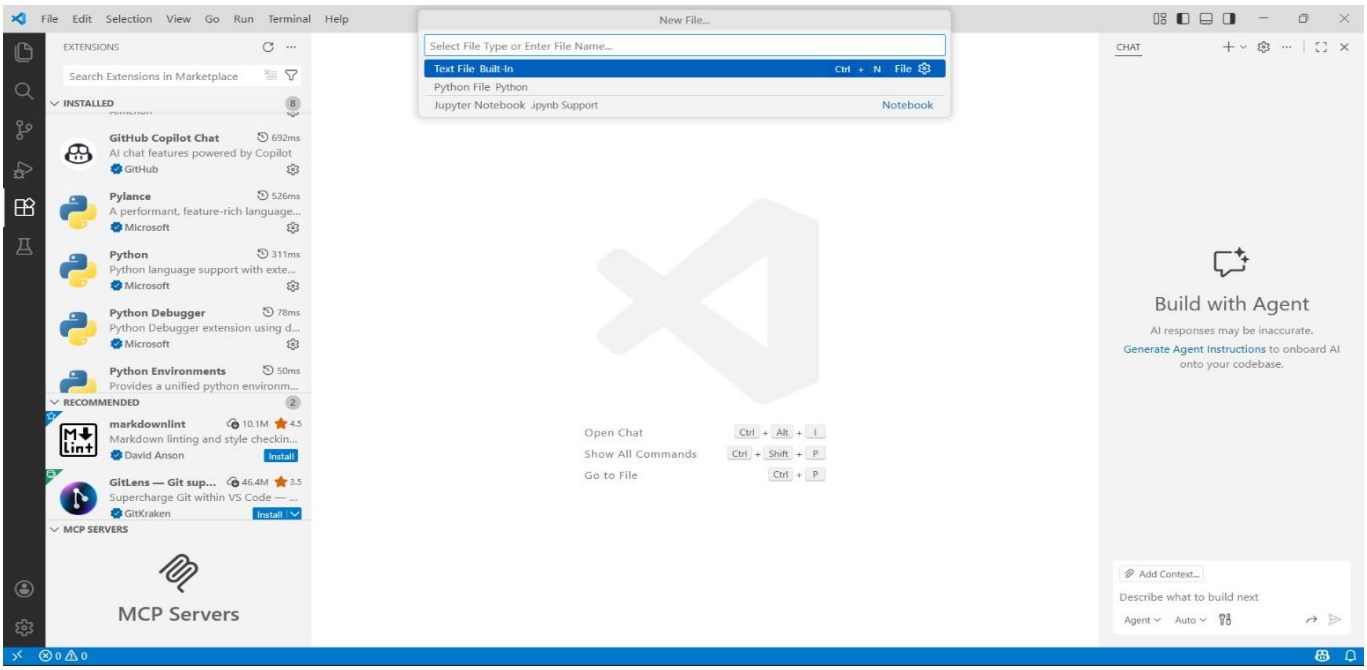
ROLL NO.:2303A51644

TASK 0:

● Install and configure GitHub Copilot in VS Code.



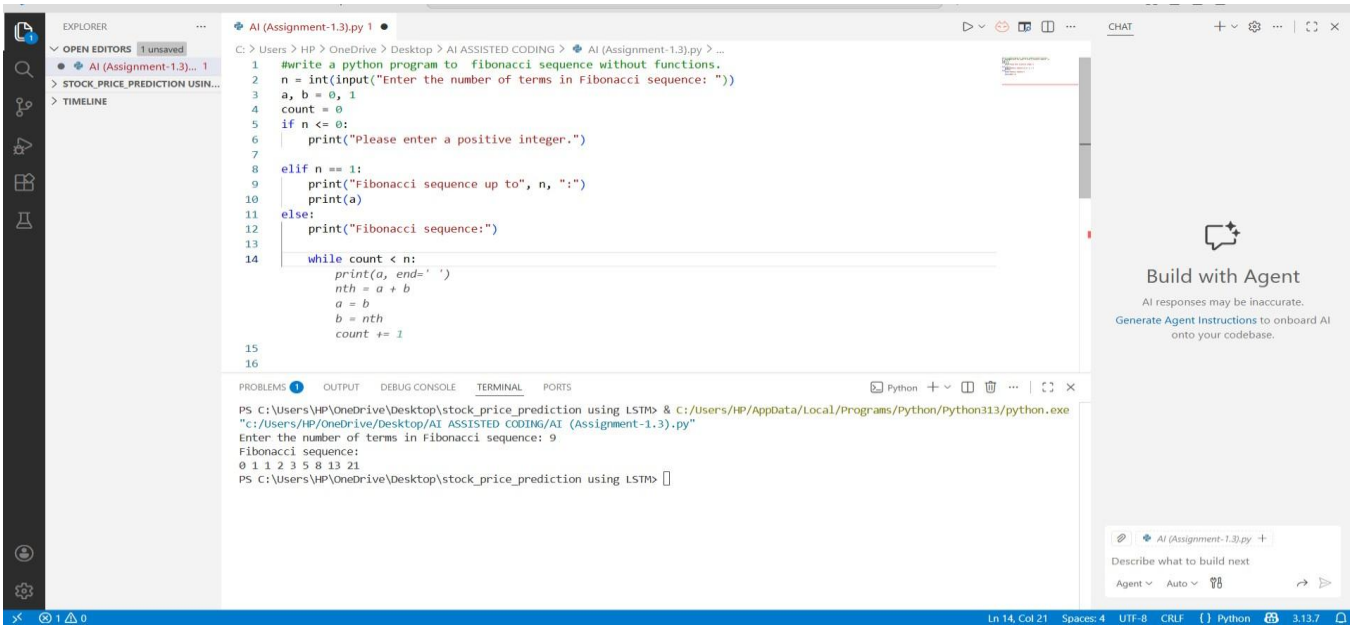


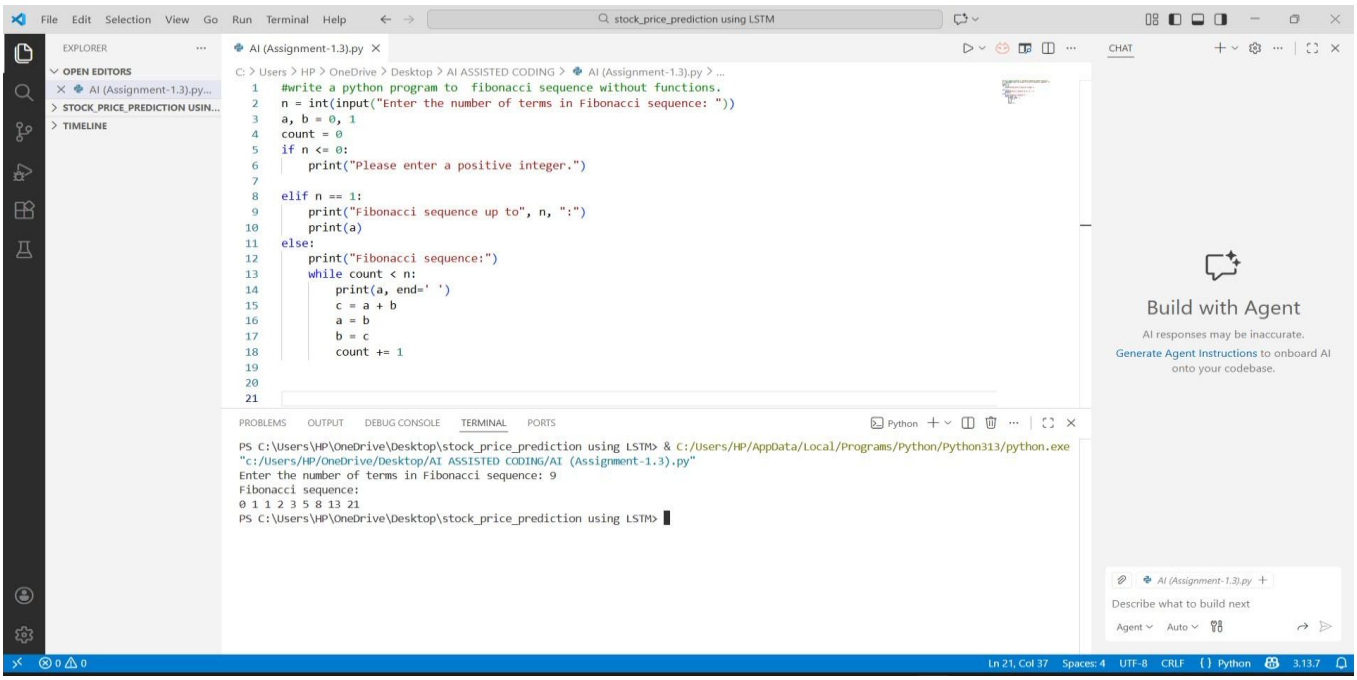


<u>TASK 1:</u> AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions).

PROMPT:Write a python program to Fibonacci without functions.

CODE AND OUPUT:





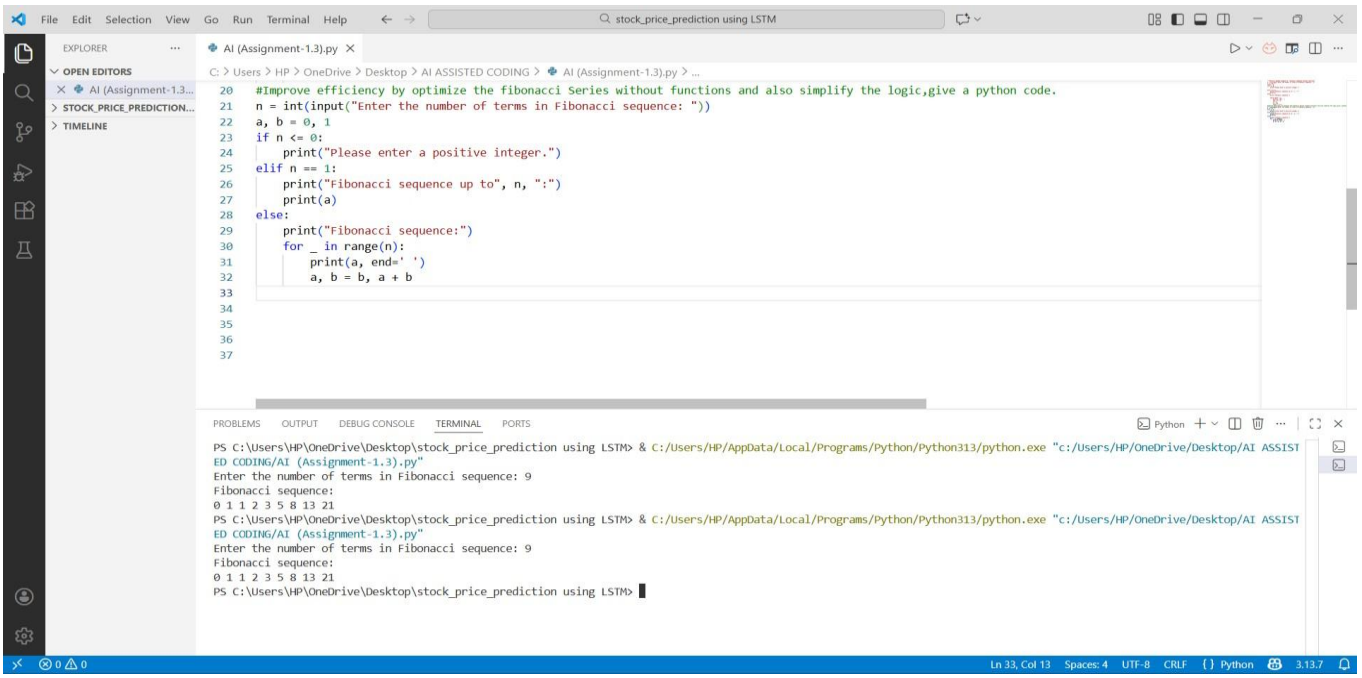
JUSTIFICATION:

1. All Fibonacci logic is written in a single block of code.
2. Easy for beginners to understand basic sequence generation.
3. Code readability decreases as the program grows.
4. Logic cannot be reused in other programs.
5. Suitable only for small and simple applications.

TASK:2:- AI Code Optimization & Cleanup (Improving Efficiency).

PROMPT: Improve efficiency by optimize the fibonacci sequence without using functions and also simplify the logic,give a python code.

CODE AND OUTPUT:



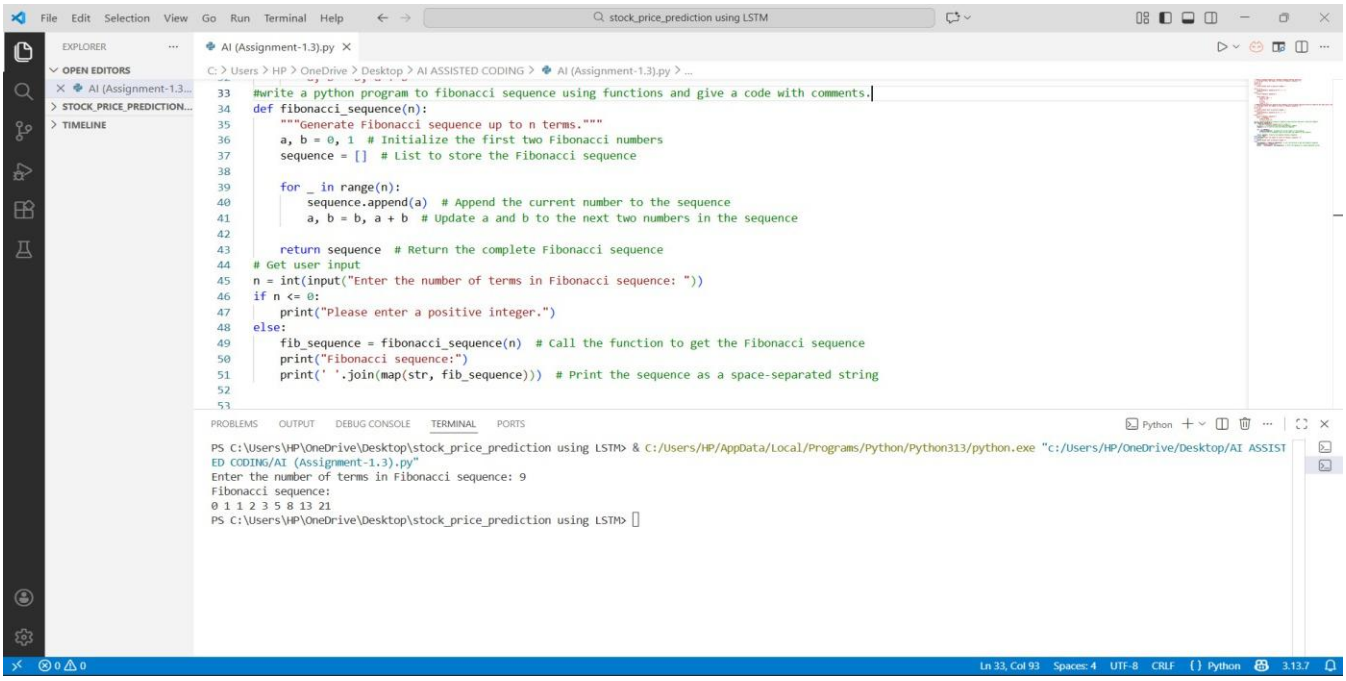
JUSTIFICATION:

1. Optimized logic reduces unnecessary calculations.
2. Code becomes simpler and easier to read.
3. Execution efficiency is improved without using functions.
4. Shows how AI can enhance performance through optimization.
5. Still limited in scalability due to lack of modularization.

TASK 3: Modular Design Using AI Assistance (Fibonacci Using Functions).

PROMPT: Write a python program to fibonacci sequence using functions and give a code with comments.

CODE AND OUTPUT:



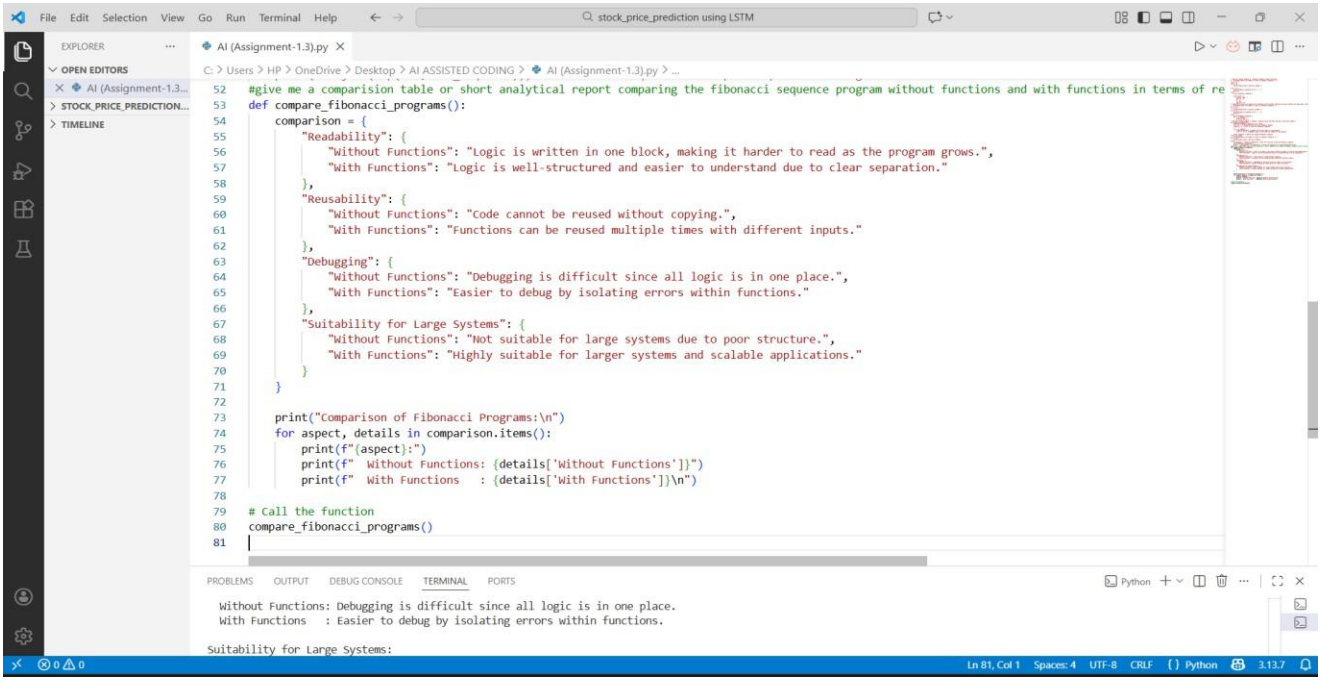
JUSTIFICATION:

1. Fibonacci logic is separated into a function.
2. Improves code readability and structure.
3. Enables reusability of the Fibonacci function.
4. Makes debugging easier by isolating errors.
5. Suitable for larger and real-world software systems.

TASK 4: Comparative Analysis – Procedural vs Modular Fibonacci Code.

PROMPT: give me a comparision table or short analytical report comparing the fibonacci sequence program without functions and with functions in terms of readability, reusability, and efficiency,give a code in python .

CODE AND OUTPUT:



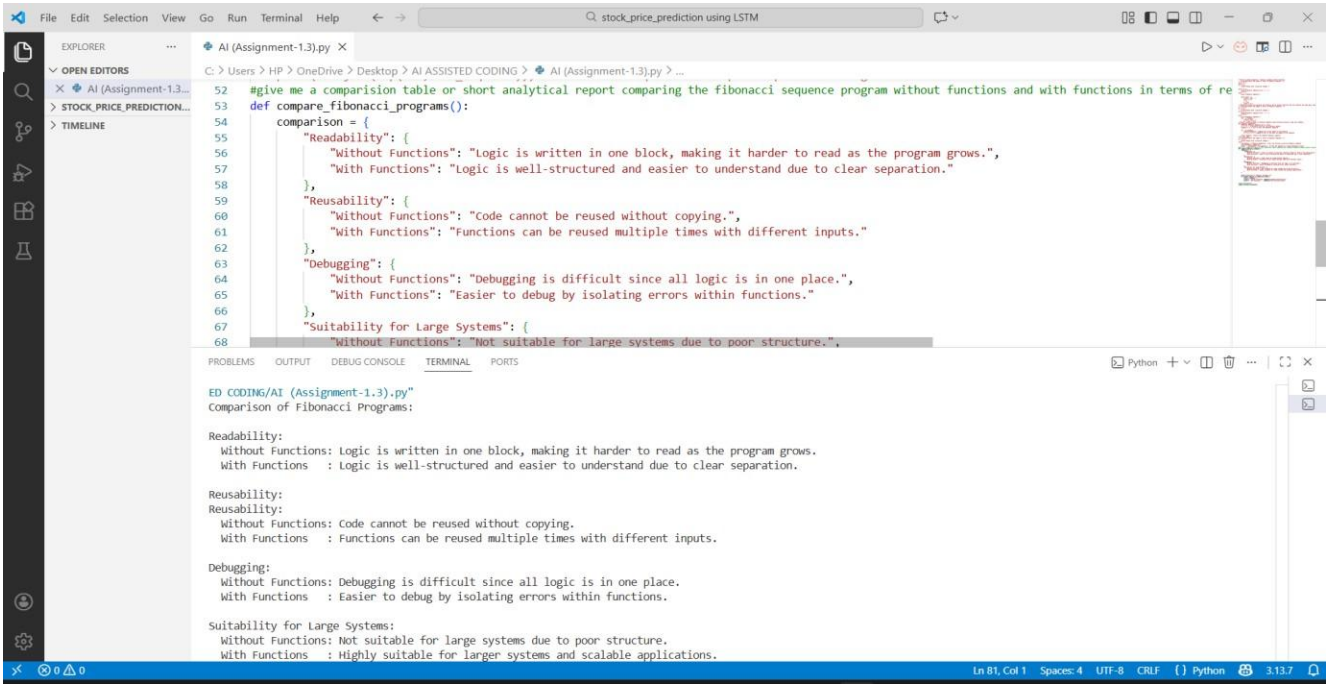
```
#give me a comparision table or short analytical report comparing the fibonacci sequence program without functions and with functions in terms of re
def compare_fibonacci_programs():
    comparison = {
        "Readability": {
            "Without Functions": "Logic is written in one block, making it harder to read as the program grows.",
            "With Functions": "Logic is well-structured and easier to understand due to clear separation."
        },
        "Reusability": {
            "Without Functions": "Code cannot be reused without copying.",
            "With Functions": "Functions can be reused multiple times with different inputs."
        },
        "Debugging": {
            "Without Functions": "Debugging is difficult since all logic is in one place.",
            "With Functions": "Easier to debug by isolating errors within functions."
        },
        "Suitability for Large Systems": {
            "Without Functions": "Not suitable for large systems due to poor structure.",
            "With Functions": "Highly suitable for larger systems and scalable applications."
        }
    }

    print("Comparison of Fibonacci Programs:\n")
    for aspect, details in comparison.items():
        print(f"{aspect}:")
        print(f"  Without Functions: {details['Without Functions']}")
        print(f"  With Functions   : {details['With Functions']}\n")

# Call the function
compare_fibonacci_programs()
```

```
Without Functions: Debugging is difficult since all logic is in one place.
With Functions   : Easier to debug by isolating errors within functions.

Suitability for Large Systems:
```



```
#give me a comparision table or short analytical report comparing the fibonacci sequence program without functions and with functions in terms of re
def compare_fibonacci_programs():
    comparison = {
        "Readability": {
            "Without Functions": "Logic is written in one block, making it harder to read as the program grows.",
            "With Functions": "Logic is well-structured and easier to understand due to clear separation."
        },
        "Reusability": {
            "Without Functions": "Code cannot be reused without copying.",
            "With Functions": "Functions can be reused multiple times with different inputs."
        },
        "Debugging": {
            "Without Functions": "Debugging is difficult since all logic is in one place.",
            "With Functions": "Easier to debug by isolating errors within functions."
        },
        "Suitability for Large Systems": {
            "Without Functions": "Not suitable for large systems due to poor structure.",
```

```
ED CODING/AI (Assignment-1.3).py"
Comparison of Fibonacci Programs:

Readability:
  Without Functions: Logic is written in one block, making it harder to read as the program grows.
  With Functions   : Logic is well-structured and easier to understand due to clear separation.

Reusability:
Reusability:
  Without Functions: Code cannot be reused without copying.
  With Functions   : Functions can be reused multiple times with different inputs.

Debugging:
  Without Functions: Debugging is difficult since all logic is in one place.
  With Functions   : Easier to debug by isolating errors within functions.

Suitability for Large Systems:
  Without Functions: Not suitable for large systems due to poor structure.
  With Functions   : Highly suitable for larger systems and scalable applications.
```

JUSTIFICATION:

Procedural vs Modular Fibonacci Programs

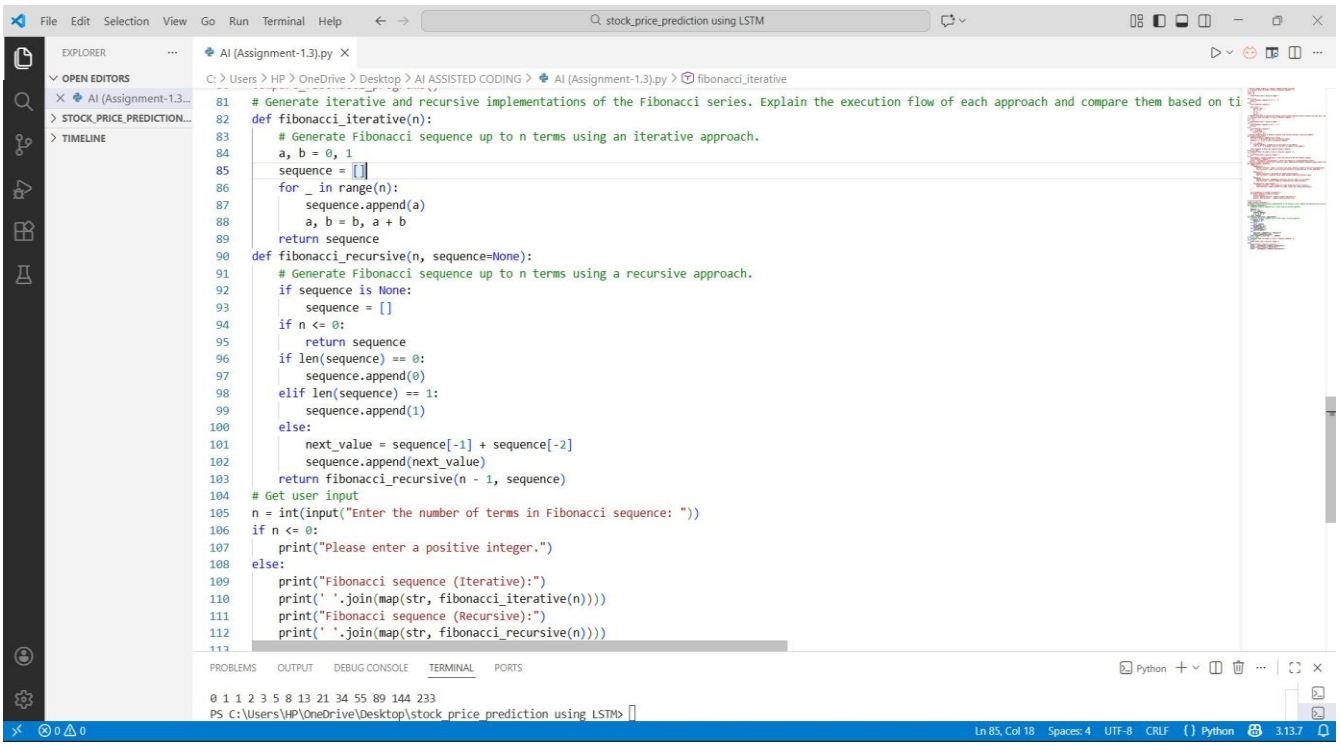
1. **Code Clarity:**
   Procedural code is harder to read as all logic is written together, while modular code is clearer due to separation into functions.

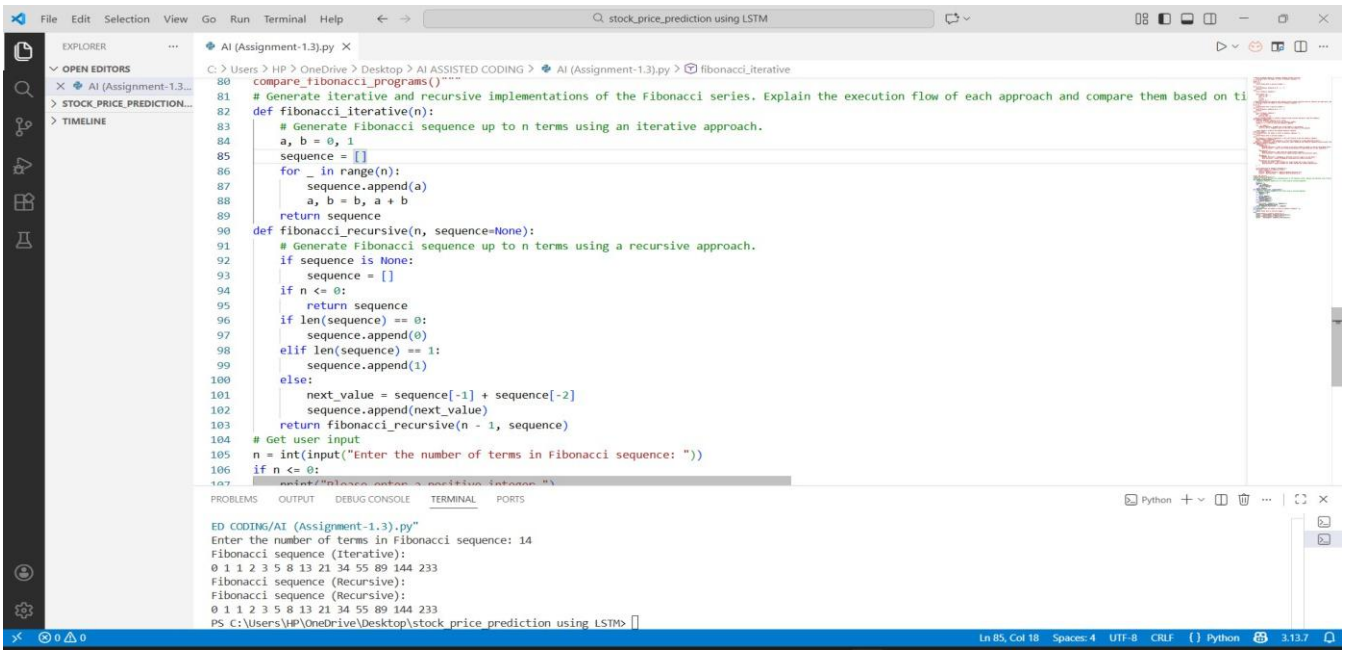
2. **Reusability:**
   Procedural code cannot be reused easily, whereas modular code allows the Fibonacci function to be reused multiple times.
3. **Debugging Ease:**
   Debugging is difficult in procedural code, but modular code makes error identification easier by isolating logic.
4. **Scalability:**
   Procedural code is not suitable for large programs, while modular code supports growth and scalability.
5. **Maintainability:**
   Changes in procedural code affect the entire program, whereas modular code allows easy updates within functions.
6. **Professional Practice:**
   Modular programming follows standard software development practices and is preferred in real-world systems.

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

PROMPT: Generate iterative and recursive implementations of the Fibonacci series. Explain the execution flow of each approach and compare them based on time complexity, space complexity, performance for large n, and cases where recursion should be avoided.give a python code.

CODE AND OUTPUT:





JUSTIFICATION:

1. Iterative approach uses loops and is memory efficient.
2. Recursive approach uses function calls and consumes more memory.
3. Recursive Fibonacci has higher time complexity due to repeated calls.

4. Iterative approach performs better for large values of *n*.
5. Recursion should be avoided in performance-critical applications.
6. Demonstrates different algorithmic paradigms clearly.