

AI ASSISTED CODING

LAB-I

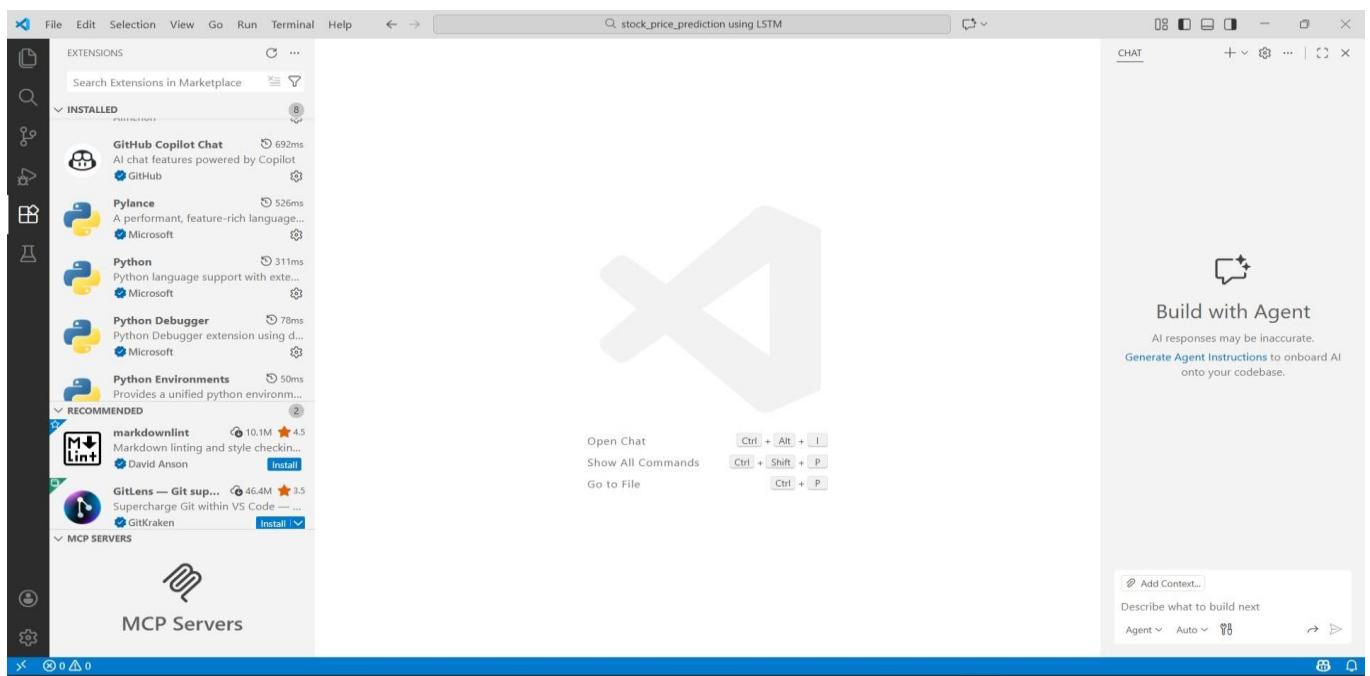
ASSIGNMENT-1

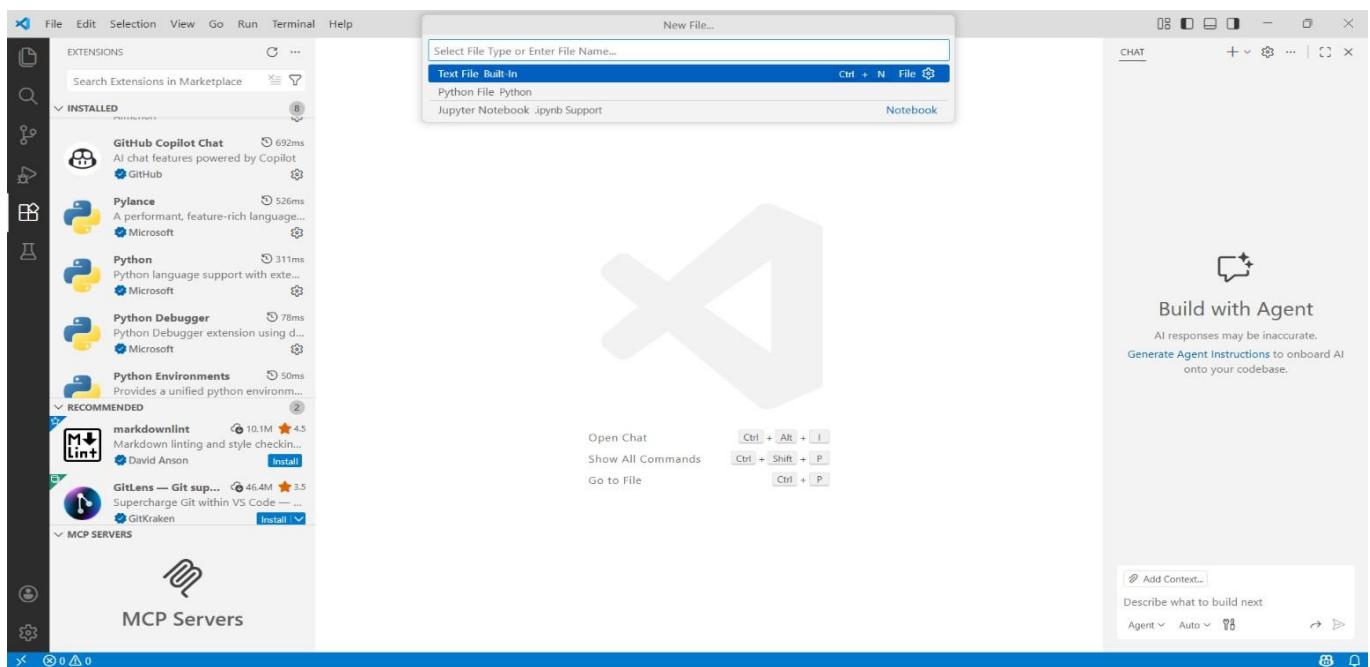
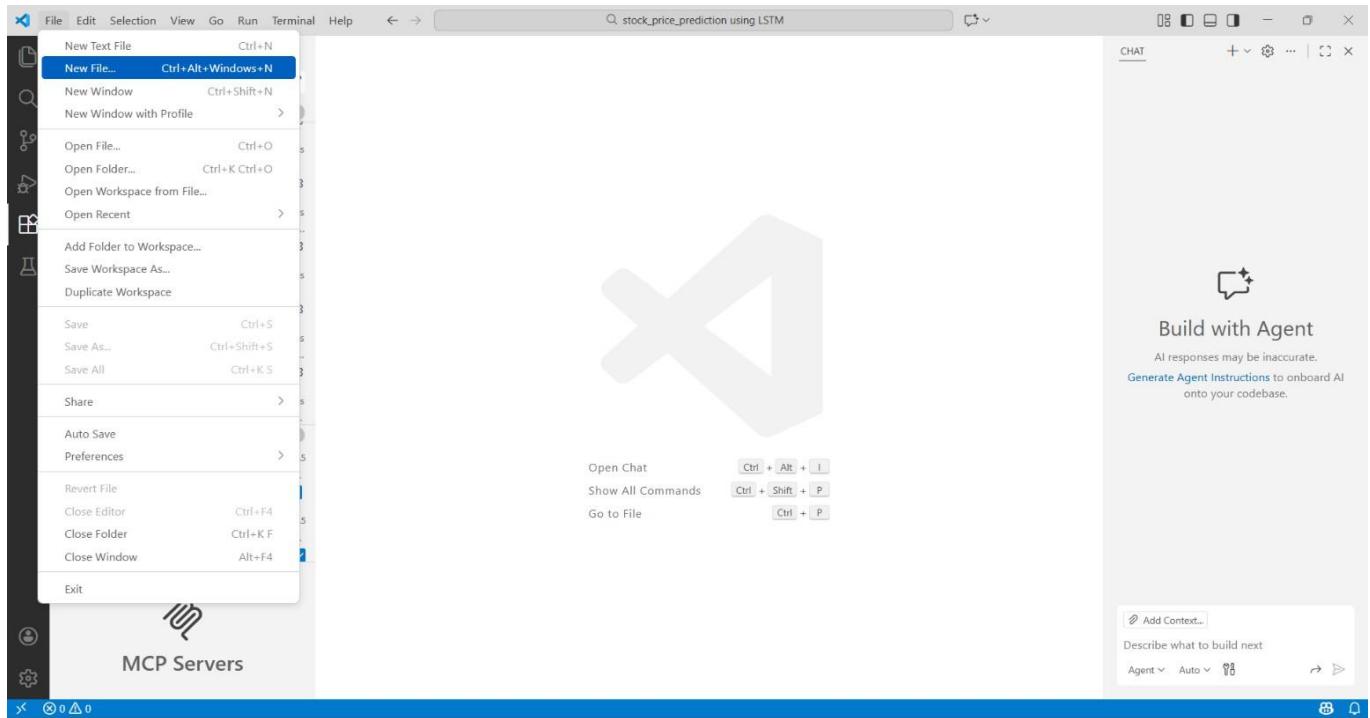
BATCH:28

ROLL NO.:2303A51644

TASK 0:

- Install and configure GitHub Copilot in VS Code.





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

PROMPT: Write a python program to Fibonacci without functions.

CODE AND OUPUT:

```
C:\> Users > HP > OneDrive > Desktop > AI ASSISTED CODING > AI (Assignment-1.3).py > ...
1  #write a python program to fibonacci sequence without functions.
2  n = int(input("Enter the number of terms in Fibonacci sequence: "))
3  a, b = 0, 1
4  count = 0
5  if n <= 0:
6      print("Please enter a positive integer.")
7
8  elif n == 1:
9      print("Fibonacci sequence up to", n, ":")
10     print(a)
11 else:
12     print("Fibonacci sequence:")
13
14     while count < n:
15         print(a, end=' ')
16         nth = a + b
17         a = b
18         b = nth
19         count += 1
20
21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI ASSISTED CODING/AI (Assignment-1.3).py"
Enter the number of terms in Fibonacci sequence: 9
Fibonacci sequence:
0 1 1 2 3 5 8 13 21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM>
```

```
C:\> Users > HP > OneDrive > Desktop > AI ASSISTED CODING > AI (Assignment-1.3).py > ...
1  #write a python program to fibonacci sequence without functions.
2  n = int(input("Enter the number of terms in Fibonacci sequence: "))
3  a, b = 0, 1
4  count = 0
5  if n <= 0:
6      print("Please enter a positive integer.")
7
8  elif n == 1:
9      print("Fibonacci sequence up to", n, ":")
10     print(a)
11 else:
12     print("Fibonacci sequence:")
13
14     while count < n:
15         print(a, end=' ')
16         c = a + b
17         a = b
18         b = c
19         count += 1
20
21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI ASSISTED CODING/AI (Assignment-1.3).py"
Enter the number of terms in Fibonacci sequence:
0 1 1 2 3 5 8 13 21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM>
```

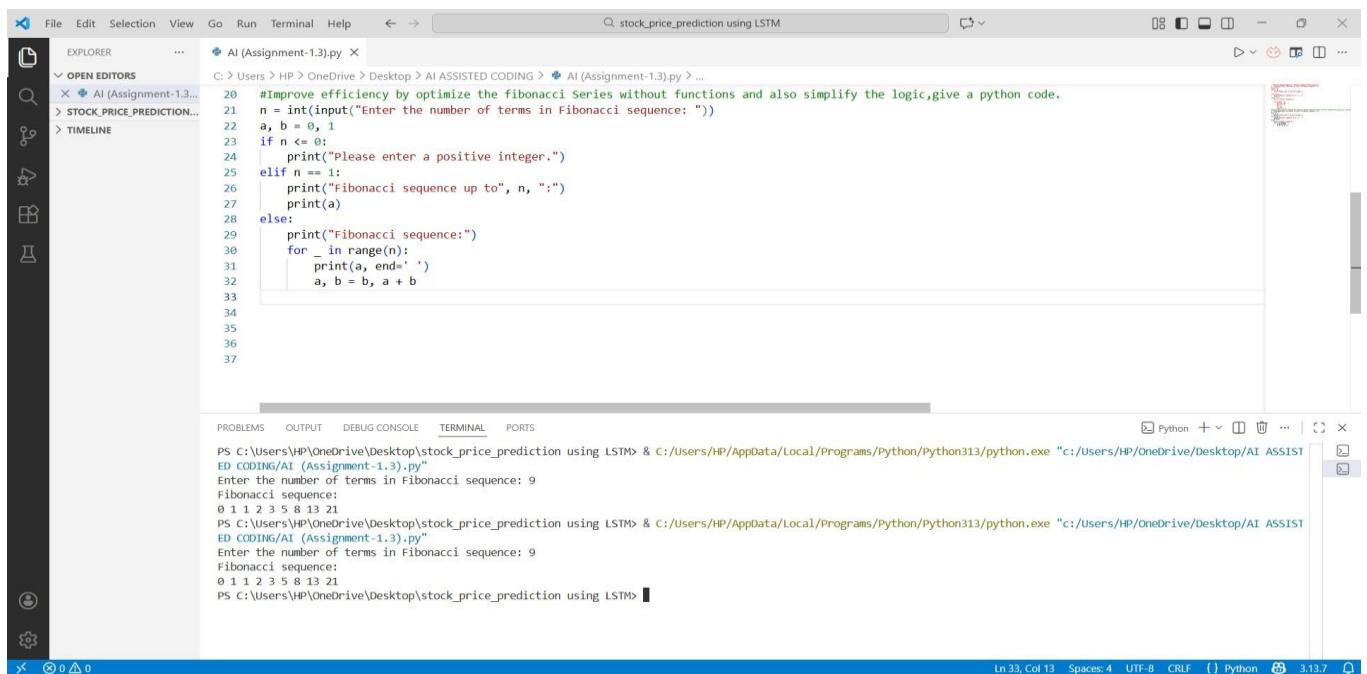
JUSTIFICATION:

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:



The screenshot shows a Visual Studio Code (VS Code) interface. The top bar has the title "stock_price_prediction using LSTM". The left sidebar shows an "EXPLORER" view with a file named "AI (Assignment-1.3).py" selected. The main editor area contains the following Python code:

```
20 #improve efficiency by optimize the fibonacci Series without functions and also simplify the logic,give a python code.
21 n = int(input("Enter the number of terms in Fibonacci sequence: "))
22 a, b = 0, 1
23 if n <= 0:
24     print("Please enter a positive integer.")
25 elif n == 1:
26     print("Fibonacci sequence up to", n, ":")
27     print(a)
28 else:
29     print("Fibonacci sequence:")
30     for _ in range(n):
31         print(a, end=' ')
32         a, b = b, a + b
33
34
35
36
37
```

Below the editor is a terminal window showing the execution of the script. The terminal output is:

```
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI ASSISTED CODING/AI (Assignment-1.3).py"
Enter the number of terms in Fibonacci sequence: 9
Fibonacci sequence:
0 1 1 2 3 5 8 13 21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI ASSISTED CODING/AI (Assignment-1.3).py"
Enter the number of terms in Fibonacci sequence: 9
Fibonacci sequence:
0 1 1 2 3 5 8 13 21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM>
```

The status bar at the bottom indicates "Ln 33, Col 13 Spaces:4 UTF-8 CRLF {} Python 3.13.7".

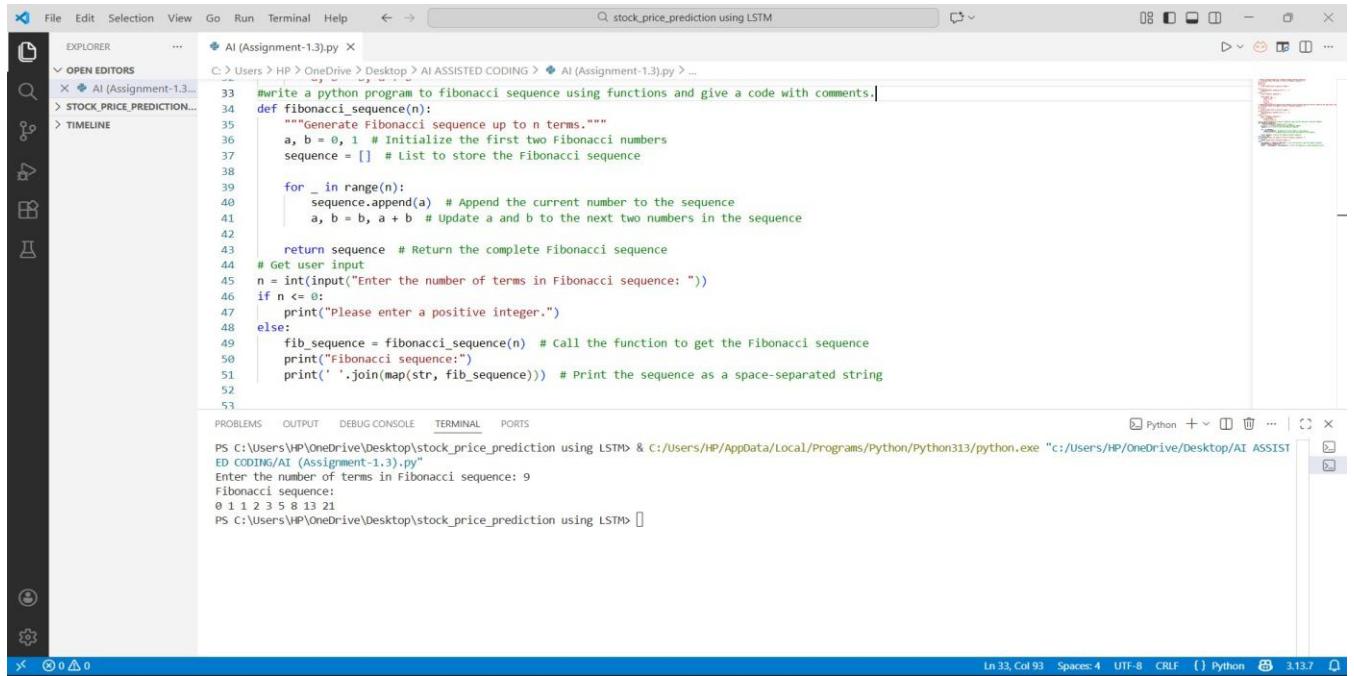
JUSTIFICATION:

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:



```
File Edit Selection View Go Run Terminal Help < > Q stock_price_prediction using LSTM
OPEN EDITORS
EXPLORER
AI (Assignment-1.3) ...
C:\Users>HP>OneDrive>Desktop>AI ASSISTED CODING> AI (Assignment-1.3).py > ...
33 #write a python program to fibonacci sequence using functions and give a code with comments.
34 def fibonacci_sequence(n):
35     """Generate Fibonacci sequence up to n terms."""
36     a, b = 0, 1 # Initialize the first two Fibonacci numbers
37     sequence = [] # List to store the Fibonacci sequence
38
39     for _ in range(n):
40         sequence.append(a) # Append the current number to the sequence
41         a, b = b, a + b # Update a and b to the next two numbers in the sequence
42
43     return sequence # Return the complete Fibonacci sequence
44 # Get user input
45 n = int(input("Enter the number of terms in Fibonacci sequence: "))
46 if n <= 0:
47     print("Please enter a positive integer.")
48 else:
49     fib_sequence = fibonacci_sequence(n) # Call the function to get the Fibonacci sequence
50     print("Fibonacci sequence:")
51     print(' '.join(map(str, fib_sequence))) # Print the sequence as a space-separated string
52
53
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/AI ASSISTED CODING/AI (Assignment-1.3).py"
Enter the number of terms in Fibonacci sequence: 9
Fibonacci sequence:
0 1 1 2 3 5 8 13 21
PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM>
In 33, Col 93 Spaces: 4 UFT-8 CRLF { } Python 3.13.7
```

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:

The screenshot shows a code editor window with the following Python code:

```
#give me a comparison table or short analytical report comparing the fibonacci sequence program without functions and with functions in terms of readability, reusability, debugging, and suitability for large systems.
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"\n{aspect}:")
        print(f" Without Functions: {details['Without Functions']}")\n"
        print(f" With Functions : {details['With Functions']}\\n")
    # Call the function
    compare_fibonacci_programs()
```

The terminal output shows the comparison results:

```
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:
```

The screenshot shows the same code editor window after running the code. The terminal output now includes the comparison results:

```
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:
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:

The screenshot shows a Jupyter Notebook interface with the following details:

- File, Edit, Selection, View, Go, Run, Terminal, Help** menu bar.
- EXPLORER** sidebar with sections: OPEN EDITORS, AI (Assignment-1.3).py, STOCK_PRICE_PREDICTION..., and TIMELINE.
- TERMINAL** tab open, displaying the command: PS C:\Users\HP\OneDrive\Desktop\stock_price_prediction using LSTM> []
- CODE** tab: AI (Assignment-1.3).py content.
- OUTPUT** tab: Shows the output of the code execution: 0 1 1 2 3 5 8 13 21 34 55 89 144 233
- PROBLEMS**, **DEBUG CONSOLE**, and **PORTS** tabs.
- STATUS** bar: Ln 85, Col 18, Spaces: 4, UTF-8, CRLF, Python 3.13.7

```
# Generate iterative and recursive implementations of the Fibonacci series. Explain the execution flow of each approach and compare them based on time complexity and memory usage.
```

```
def fibonacci_iterative(n):  
    # Generate Fibonacci sequence up to n terms using an iterative approach.  
    a, b = 0, 1  
    sequence = []  
    for _ in range(n):  
        sequence.append(a)  
        a, b = b, a + b  
    return sequence
```

```
def fibonacci_recursive(n, sequence=None):  
    # Generate Fibonacci sequence up to n terms using a recursive approach.  
    if sequence is None:  
        sequence = []  
    if n <= 0:  
        return sequence  
    if len(sequence) == 0:  
        sequence.append(0)  
    elif len(sequence) == 1:  
        sequence.append(1)  
    else:  
        next_value = sequence[-1] + sequence[-2]  
        sequence.append(next_value)  
    return fibonacci_recursive(n - 1, sequence)
```

```
# Get user input  
n = int(input("Enter the number of terms in Fibonacci sequence: "))  
if n <= 0:  
    print("Please enter a positive integer.")  
else:  
    print("Fibonacci sequence (Iterative):")  
    print(' '.join(map(str, fibonacci_iterative(n))))  
    print("Fibonacci sequence (Recursive):")  
    print(' '.join(map(str, fibonacci_recursive(n))))
```

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.

