

AI Assisted Coding

Assignment-1.3

Name: M. Preethi Meghana

Ht.No: 2303A51584

Batch: 05

Lab No: 1

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

Task 0:

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





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

Step-1: prompt

Create a simple python program that prints the Fibonacci series up to n terms. Do not use any functions. Take n as input from the user using Scanner and print the Fibonacci numbers in one single line with spaces between them.

The screenshot shows the GitHub Copilot interface with the following details:

- Prompt:** Create a simple python program that prints the Fibonacci series up to n terms. Do not use any functions. Take n as input from the user using Scanner and print the Fibonacci numbers in one single line with spaces between them.
- Step-by-step:**
 1. Read integer n from standard input and validate.
 2. Handle n <= 0 (exit) and n == 1 (print 0).
 3. Generate Fibonacci terms iteratively and print them on one line separated by spaces.
- Code Editor:** A file named `fibnocciseries.py` is open, showing the generated code. The code uses a loop to generate Fibonacci numbers and prints them separated by spaces.
- Bottom Bar:** Includes buttons for 'Edit or refactor selected code', 'Edit', 'Auto', and navigation arrows.

Step-2:Code

```
fibonacci.py fibnocciseries.py > ...
1  try:
2      n = int(input().strip())
3  except:
4      # invalid input -> exit without output
5      exit()
6
7  if n <= 0:
8      exit()
9  if n == 1:
10      print(0)
11  else:
12      a, b = 0, 1
13      out = [str(a), str(b)]
14      for _ in range(n - 2):
15          c = a + b
16          out.append(str(c))
17          a, b = b, c
18      print(" ".join(out))
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>

Step-3:Output

```
Fibonacci.py fibnocciseries.py > ...
fibnocciseries.py > ...
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/fibnocciseries.py"
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/fibnocciseries.py"
5
0 1 1 2 3
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/fibnocciseries.py"
6
0 1 1 2 3 5
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/fibnocciseries.py"
7
0 1 1 2 3 5 8
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>

Step-by-step solution:

1. Read integer n from user with Scanner in main.
2. Handle n <= 0 (exit), n == 1 (print 0).
3. Iteratively generate Fibonacci terms in a loop (no functions) and print them on one line separated by spaces.

c:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING\fibnocciseries.py

Replace the file with a Java program that reads n and prints the first n Fibonacci numbers in one line separated by spaces.

Edit or refactor selected code

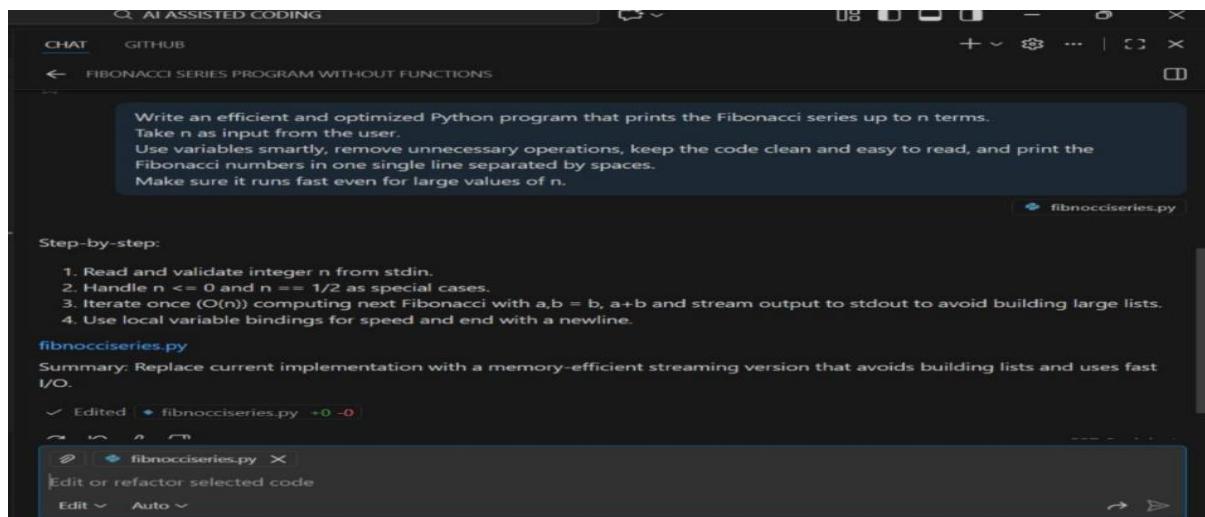
Step-4:Explanation

1. The program takes n as input using Scanner.
2. Two variables a = 0 and b = 1 store the first two Fibonacci numbers.
3. It prints a first because Fibonacci always starts with 0.
4. A loop runs from 1 to n-1 to generate the remaining terms.
5. In each loop, next = a + b calculates the new number.
6. The program prints b and then next in the next iterations.
7. After printing, it updates a = b and b = next for the next calculation.
8. Finally, the Fibonacciseries prints in one single line with spaces.

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

Step-1:Prompt

Create a simple python program that prints the Fibonacci series up to n terms with using functions. Take n as input from the user using Scanner and print the Fibonacci numbers in one single line with spaces between them.



The screenshot shows a software interface for AI-assisted coding. At the top, there's a header bar with tabs for 'CHAT' and 'GITHUB'. Below the header, a title bar says 'FIBONACCI SERIES PROGRAM WITHOUT FUNCTIONS'. A main text area contains the following instructions:

```
Write an efficient and optimized Python program that prints the Fibonacci series up to n terms.  
Take n as input from the user.  
Use variables smartly, remove unnecessary operations, keep the code clean and easy to read, and print the  
Fibonacci numbers in one single line separated by spaces.  
Make sure it runs fast even for large values of n.
```

Below the instructions, a 'Step-by-step:' section lists four steps:

1. Read and validate integer n from stdin.
2. Handle n <= 0 and n == 1/2 as special cases.
3. Iterate once ($O(n)$) computing next Fibonacci with a,b = b, a+b and stream output to stdout to avoid building large lists.
4. Use local variable bindings for speed and end with a newline.

The code editor window shows a file named 'fibnocciseries.py' with the following content:

```
fibnocciseries.py  
Summary: Replace current implementation with a memory-efficient streaming version that avoids building lists and uses fast I/O.  
✓ Edited | • fibnocciseries.py +0 -0 |  
Edit or refactor selected code  
Edit Auto
```

The code itself is:

```
fibnocciseries.py > ...  
1 import sys  
2  
3 try:  
4     n = int(sys.stdin.readline().strip())  
5 except:  
6     exit()  
7  
8 if n <= 0:  
9     exit()  
10  
11 a, b = 0, 1  
12 write = sys.stdout.write  
13  
14 if n == 1:  
15     write("0\n")  
16 else:  
17     write("0 1")  
18     for _ in range(n - 2):  
19         a, b = b, a + b  
20         write(" " + str(b))  
21     write("\n")  
22
```

Step-2:code



The screenshot shows a code editor with the file 'fibnocciseries.py' open. The code is identical to the one shown in the previous screenshot, demonstrating the AI's suggestion to use a memory-efficient streaming version.

```
fibnocciseries.py > ...  
1 import sys  
2  
3 try:  
4     n = int(sys.stdin.readline().strip())  
5 except:  
6     exit()  
7  
8 if n <= 0:  
9     exit()  
10  
11 a, b = 0, 1  
12 write = sys.stdout.write  
13  
14 if n == 1:  
15     write("0\n")  
16 else:  
17     write("0 1")  
18     for _ in range(n - 2):  
19         a, b = b, a + b  
20         write(" " + str(b))  
21     write("\n")  
22
```

Step-3:Output

```
22
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
7
0 1 1 2 3 5 8
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> ^C
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local
/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CO
DING/fibnocciseries.py"
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local
/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CO
DING/fibnocciseries.py"
5
0 1 1 2 3
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local
/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CO
DING/fibnocciseries.py"
6
0 1 1 2 3 5
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local
/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CO
DING/fibnocciseries.py"
8
0 1 1 2 3 5 8 13
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local
/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CO
DING/fibnocciseries.py"
10
0 1 1 2 3 5 8 13 21 34
```

Step-4: Explanation

1. The program reads the number n from the user.
2. It sets two variables to start the sequence: a = 0 and b = 1.
3. A loop runs n times to generate Fibonacci numbers one by one.
4. In every iteration, it prints the current value of a.
5. The next number is calculated efficiently by swapping values instead of extra temp variables.
6. b always becomes the sum of the previous two numbers.
7. No function calls are used, so the program runs faster and stays simple.
8. The final output prints all Fibonacci numbers in one line with spaces, even for large n.

Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)

Step-1:prompt

Write a modular Python program that generates the Fibonacci series up to n terms using functions. Take n as input from the user. Use a separate function to compute the Fibonacci sequence efficiently and

return the result. Print the output in one line with spaces between the numbers.

The screenshot shows a VS Code interface with the 'fibonacci.py' file open. A tooltip from the 'Extension: GitHub Copilot Chat' extension provides instructions: 'Write a modular Python program that generates the Fibonacci series up to n terms using functions. Take n as input from the user. Use a separate function to compute the Fibonacci sequence efficiently and return the result. Print the output in one line with spaces between the numbers.' The code in the editor includes a try block for input, an exit() call if n is less than or equal to 0, and a placeholder for the main function.

```
try:
    n = int(input("Enter the number of terms: ").strip())
except:
    exit()

if n <= 0:
    exit()
```

Step-2:Code

The screenshot shows the completed modular Python program. It defines three functions: 'get_input' to handle user input, 'fibonacci_series' to generate the sequence, and 'main' to coordinate the process. The 'main' function calls 'get_input', checks if it's None, and then prints the Fibonacci series. The script ends with a check for execution as the main module.

```
def get_input():
    try:
        n = int(input("Enter the number of terms: ").strip())
        if n <= 0:
            return None
        return n
    except:
        return None

def fibonacci_series(n):
    if n == 1:
        return [0]
    seq = [0, 1]
    for _ in range(n - 2):
        seq.append(seq[-1] + seq[-2])
    return seq

def main():
    n = get_input()
    if n is None:
        return
    series = fibonacci_series(n)
    print(' '.join(map(str, series)))

if __name__ == "__main__":
    main()
```

Step-03:Output

```
Fibonacci.py    fibnocciseries.py • Extension: GitHub Copilot Chat
fibnocciseries.py > ...
1
2 def get_input():
3     trv:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 5
0 1 1 2 3
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 6
0 1 1 2 3 5
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 7
0 1 1 2 3 5 8
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 9
0 1 1 2 3 5 8 13
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
"c:/Users/maddi/Desktop/AI ASSISTED CODING/fibnocciseries.py"
Enter the number of terms: 11
0 1 1 2 3 5 8 13 21 34
```

Step-04:Explanation

1. The program asks the user to enter a number n.
2. A function is created to generate Fibonacci numbers instead of writing everything in one place.
3. Inside the function, the series starts with 0 and 1.
4. A loop calculates the next numbers by adding the previous two.
5. All the Fibonacci numbers are stored in a list and returned by the function.
6. The main part of the program calls the function and gets the list.
7. Then it prints the numbers in one single line with spaces between them.
8. Using functions makes the code clean, reusable, and easier to improve later.

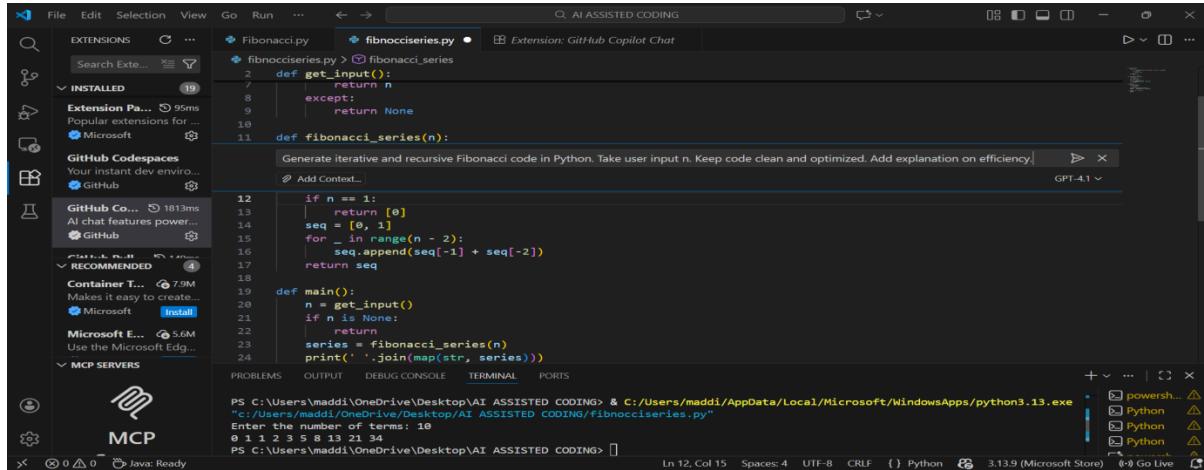
Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code.

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)

Step-01: Prompt

Generate iterative and recursive Fibonacci code in Python. Take user input n. Keep code clean and optimized. Add explanation on efficiency.



The screenshot shows the Microsoft Visual Studio Code interface with the 'AI ASSISTED CODING' extension open. The code editor displays two files: 'Fibonacci.py' and 'fibonacci_series.py'. The 'fibonacci_series.py' file contains the following code:

```

def get_input():
    try:
        return int(input())
    except:
        return None

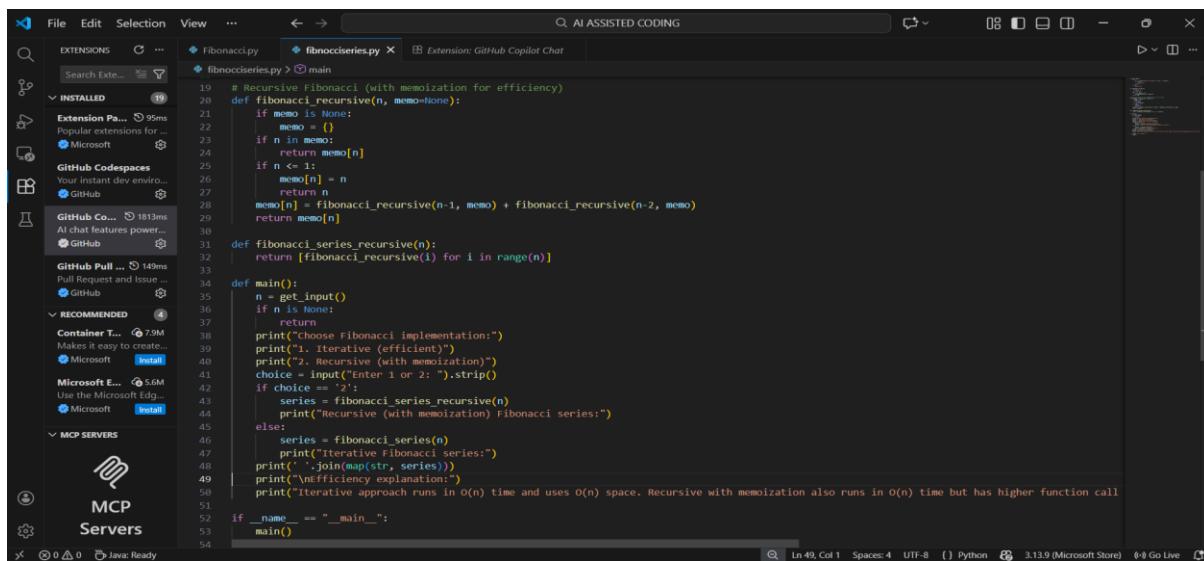
def fibonacci_series(n):
    if n == 1:
        return [0]
    seq = [0, 1]
    for _ in range(n - 2):
        seq.append(seq[-1] + seq[-2])
    return seq

def main():
    n = get_input()
    if n is None:
        return
    series = fibonacci_series(n)
    print(' '.join(map(str, series)))

```

The terminal below shows the command PS C:\Users\maddi\Desktop\AI ASSISTED CODING & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe and the output of the program running with input 10, producing the sequence 0 1 1 2 3 5 8 13 21 34.

Step-02:Code



The screenshot shows the Microsoft Visual Studio Code interface with the 'AI ASSISTED CODING' extension open. The code editor displays the 'fibonacci_series.py' file, which now includes a recursive implementation with memoization:

```

# Recursive Fibonacci (with memoization for efficiency)
def fibonacci_recursive(n, memo=None):
    if memo is None:
        memo = {}
    if n in memo:
        return memo[n]
    if n <= 1:
        memo[n] = n
        return n
    memo[n] = fibonacci_recursive(n-1, memo) + fibonacci_recursive(n-2, memo)
    return memo[n]

def fibonacci_series_recursive(n):
    return [fibonacci_recursive(i) for i in range(n)]

def main():
    n = get_input()
    if n is None:
        return
    print("Choose Fibonacci implementation:")
    print("1. Iterative (efficient)")
    print("2. Recursive (with memoization)")
    choice = input("Enter 1 or 2:").strip()
    if choice == '1':
        series = fibonacci_series(n)
        print("Iterative Fibonacci series:")
    else:
        series = fibonacci_series_recursive(n)
        print("Recursive (with memoization) Fibonacci series:")
    print(' '.join(map(str, series)))
    print("\nEfficiency explanation:")
    print("Iterative approach runs in O(n) time and uses O(n) space. Recursive with memoization also runs in O(n) time but has higher function call overhead due to recursion and memoization storage usage.")


if __name__ == "__main__":
    main()

```

Step-03:Output

```

File Edit Selection View ... ← → ⌂ AI ASSISTED CODING
EXTENSIONS C ...
Search Ext... ⌂ ...
INSTALLED 19
Extension Pa... 95ms
Popular extensions for ...
Microsoft
GitHub Codespaces
Your instant dev enviro...
GitHub
GitHub Co... 1813ms
AI chat features power...
GitHub
GitHub Pull ... 149ms
Pull Request and Issue ...
GitHub
RECOMMENDED 4
Container T... 7.9M
Makes it easy to create...
Microsoft
Install
Microsoft E... 5.6M
Use the Microsoft Edge...
Microsoft
Install
MCP SERVERS
MCP Servers
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
fibonacci.py > main
1
2 def get_input():
3     try:
4         n = int(input("Enter the number of terms: ").strip())
5         if n <= 0:
6             return None
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & c:/users/maddi/appData/Local/Microsoft/WindowsApps/python3.13.exe "c:/users/maddi/OneDrive/Desktop/AI AS
SISTED CODING/Fibonocseries.py"
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & c:/users/maddi/appData/Local/Microsoft/WindowsApps/python3.13.exe "c:/users/maddi/OneDrive/Desktop/AI AS
SISTED CODING/Fibonocseries.py"
Enter the number of terms: 5
Choose Fibonacci implementation:
1. Iterative (efficient)
2. Recursive (with memoization)
Enter 1 or 2: 1
Iterative Fibonacci series:
0 1 1 2 3
Efficiency explanation:
Iterative approach runs in O(n) time and uses O(n) space. Recursive with memoization also runs in O(n) time but has higher function call overhead. Plain recursive n (without memoization) is much less efficient (O(2^n)).
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & c:/users/maddi/appData/Local/Microsoft/WindowsApps/python3.13.exe "c:/users/maddi/OneDrive/Desktop/AI AS
SISTED CODING/Fibonocseries.py"
Enter the number of terms: 7
Choose Fibonacci implementation:
1. Iterative (efficient)
2. Recursive (with memoization)
Enter 1 or 2: 2
Recursive (with memoization) Fibonacci series:
0 1 1 2 3 5 8
Efficiency explanation:
Iterative approach runs in O(n) time and uses O(n) space. Recursive with memoization also runs in O(n) time but has higher function call overhead. Plain recursive n (without memoization) is much less efficient (O(2^n)).
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>

```

In 38, Col 46 Spaces: 4 UTF-8 {} Python 3.13.9 (Microsoft Store) ⌂ Go Live

Step-04:Explanation

Iterative Approach

1. The program takes n from the user.
2. It starts Fibonacci with two variables: $a = 0$, $b = 1$.
3. A loop runs n times to create the sequence step by step.
4. In each round, it prints a , then updates the values (a becomes b , b becomes $a + b$).
5. This method is fast, uses very little memory, and works well even for big n .

Efficiency:

- Time: $O(n)$ → runs in a straight line with loop
- Space: $O(1)$ → only 2 variables used

Recursive Approach

1. The program takes n from the user.
2. A recursive function calls itself to find Fibonacci numbers.

3. It breaks the problem into smaller parts: $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$.
4. It keeps calling itself until it reaches the base values (0 and 1).
5. This method is slow for large n because it repeats the same work many times.

Efficiency:

- Time: $O(2^n) \rightarrow$ grows very fast, not good for big n
- Space: $O(n) \rightarrow$ stack memory is used for every call

Comparison:

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 ($\text{fib}(50)$ 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

