School of Computer Science and Artificial Intelligence

Lab Assignment # 01

Course Title	: AI Assistant Coding
Name of Student	: R. Govardhan Sai Ganesh
Enrollment No.	: 2303A54044
Batch No.	: 48

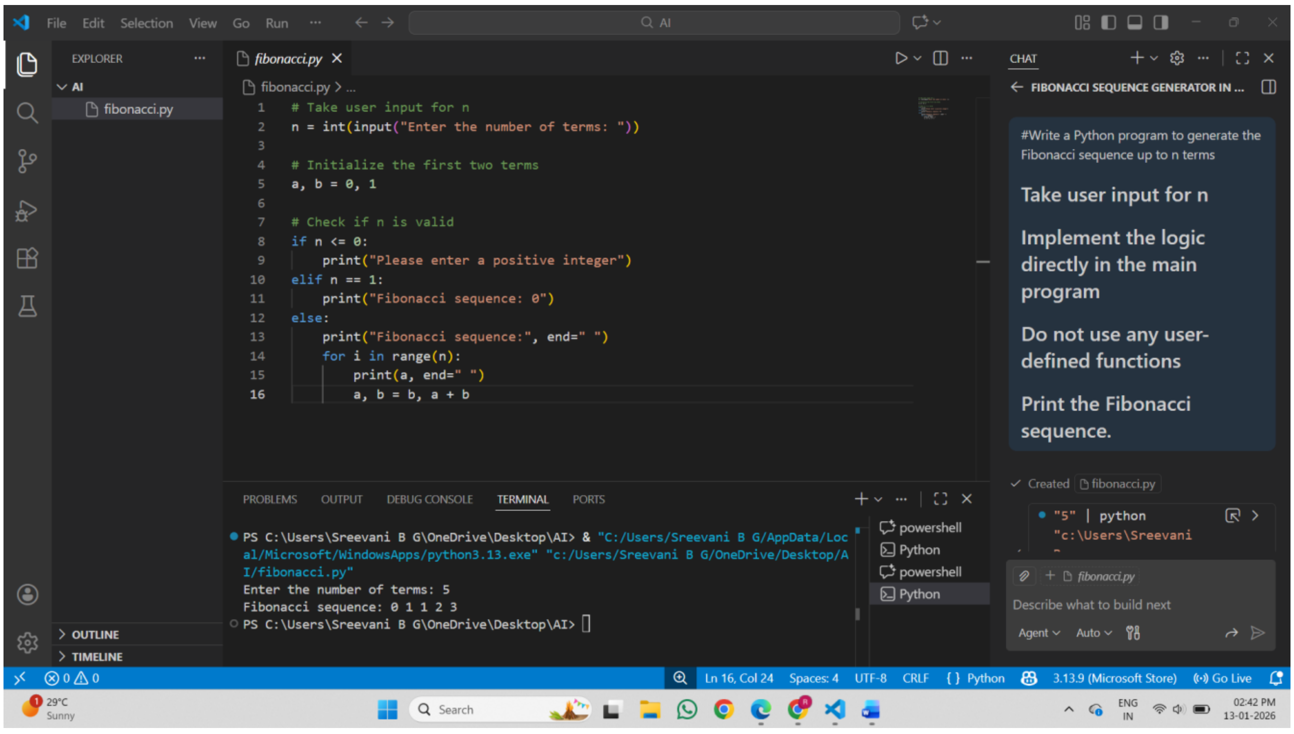
Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

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

❖ ***Scenario: You are asked to write a quick numerical sequence generator for a learning platform prototype.***

• **Prompt used:** #Write a Python program to generate the Fibonacci sequence up to n terms
Take user input for n
# Implement the logic directly in the main program
# Do not use any user-defined functions
# Print the Fibonacci sequence.

• **Screenshot of Generated Code:**



```python
# Take user input for n
n = int(input("Enter the number of terms: "))

# Initialize the first two terms
a, b = 0, 1

# Check if n is valid
if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print("Fibonacci sequence: 0")
else:
    print("Fibonacci sequence:", end=" ")
    for i in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

```
PS C:\Users\Sreevani B G\OneDrive\Desktop\AI> & "C:/Users/Sreevani B G/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Sreevani B G/OneDrive/Desktop/AI/fibonacci.py"
Enter the number of terms: 5
Fibonacci sequence: 0 1 1 2 3
PS C:\Users\Sreevani B G\OneDrive\Desktop\AI>
```

#Write a Python program to generate the Fibonacci sequence up to n terms

Take user input for n

Implement the logic directly in the main program

Do not use any user-defined functions

Print the Fibonacci sequence.

✓ Created fibonacci.py

"5" | python "c:\Users\Sreevani

• **Sample Input:**
Enter no of terms : 5

• **Sample Output:**
Fibonacci sequence up to 5 terms: 0 1 1 2 3

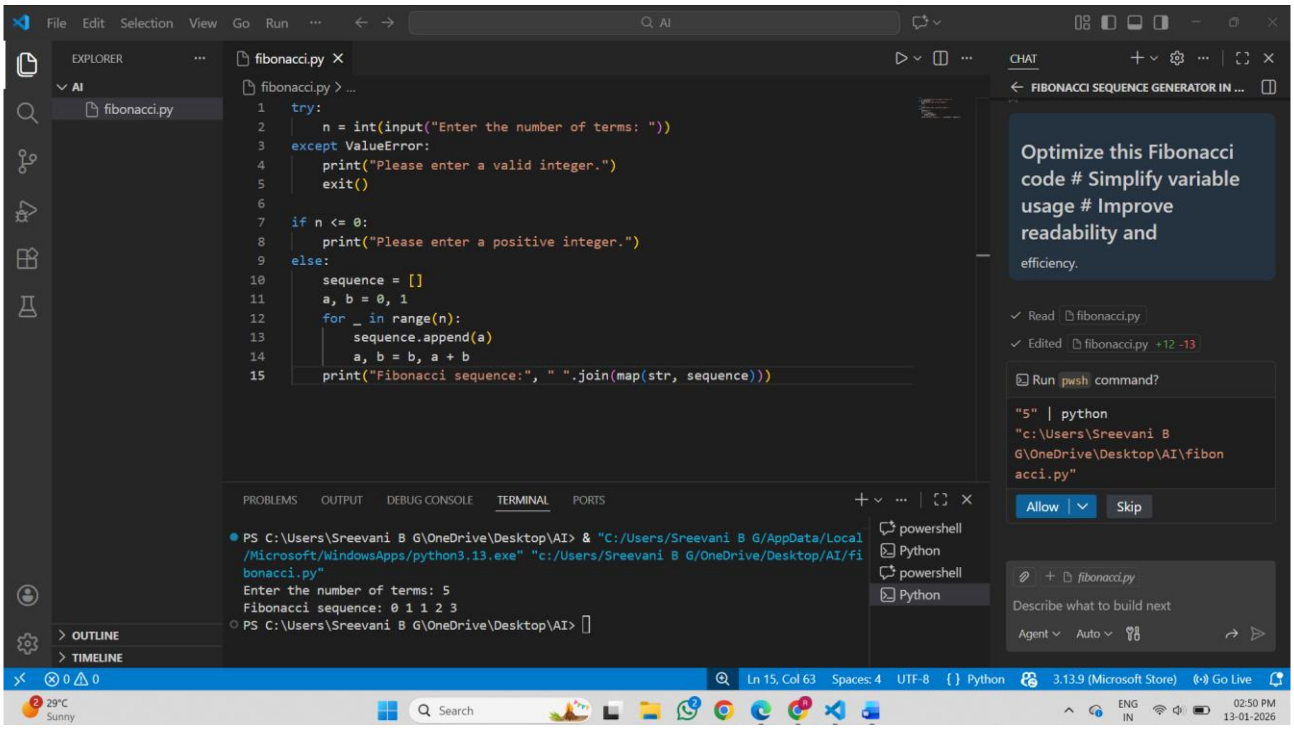
• Short Explanation of Logic:
The program first takes an integer n as input from the user, which represents the number of Fibonacci terms to be printed. It initializes the first two Fibonacci numbers as 0 and 1. Using a loop, the program prints each term and updates the values by adding the previous two numbers to generate the next term in the sequence. The entire logic is written directly in the main program without using any user-defined functions.

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

❖ ***Scenario: The prototype will be shared with other developers and needs optimization.***

• Prompt used:
# Optimize this Fibonacci code
# Simplify variable usage
# Improve readability and efficiency.

• Screenshot of Generated Code:



```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer.")
    exit()

if n <= 0:
    print("Please enter a positive integer.")
else:
    sequence = []
    a, b = 0, 1
    for _ in range(n):
        sequence.append(a)
        a, b = b, a + b
    print("Fibonacci sequence:", " ".join(map(str, sequence)))
```

Optimize this Fibonacci code # Simplify variable usage # Improve readability and efficiency.

```
"5" | python "c:\Users\Sreevani B G\OneDrive\Desktop\AI\fibonacci.py"
```

```
PS C:\Users\Sreevani B G\OneDrive\Desktop\AI> & "C:/Users/Sreevani B G/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "c:/Users/Sreevani B G/OneDrive/Desktop/AI/fibonacci.py"
Enter the number of terms: 5
Fibonacci sequence: 0 1 1 2 3
PS C:\Users\Sreevani B G\OneDrive\Desktop\AI>
```

• Sample Input:
Enter the number of terms: 5

• Sample Output: Fibonacci sequence up to 5 terms: 0 1 1 2 3

• Short Explanation of Logic:
The optimized program improves the original Fibonacci code by removing unnecessary variables and simplifying the loop logic. It generates each Fibonacci number using only the previous two values, avoiding redundant calculations. This makes the code more readable, efficient, and easier to understand while producing the same correct output.

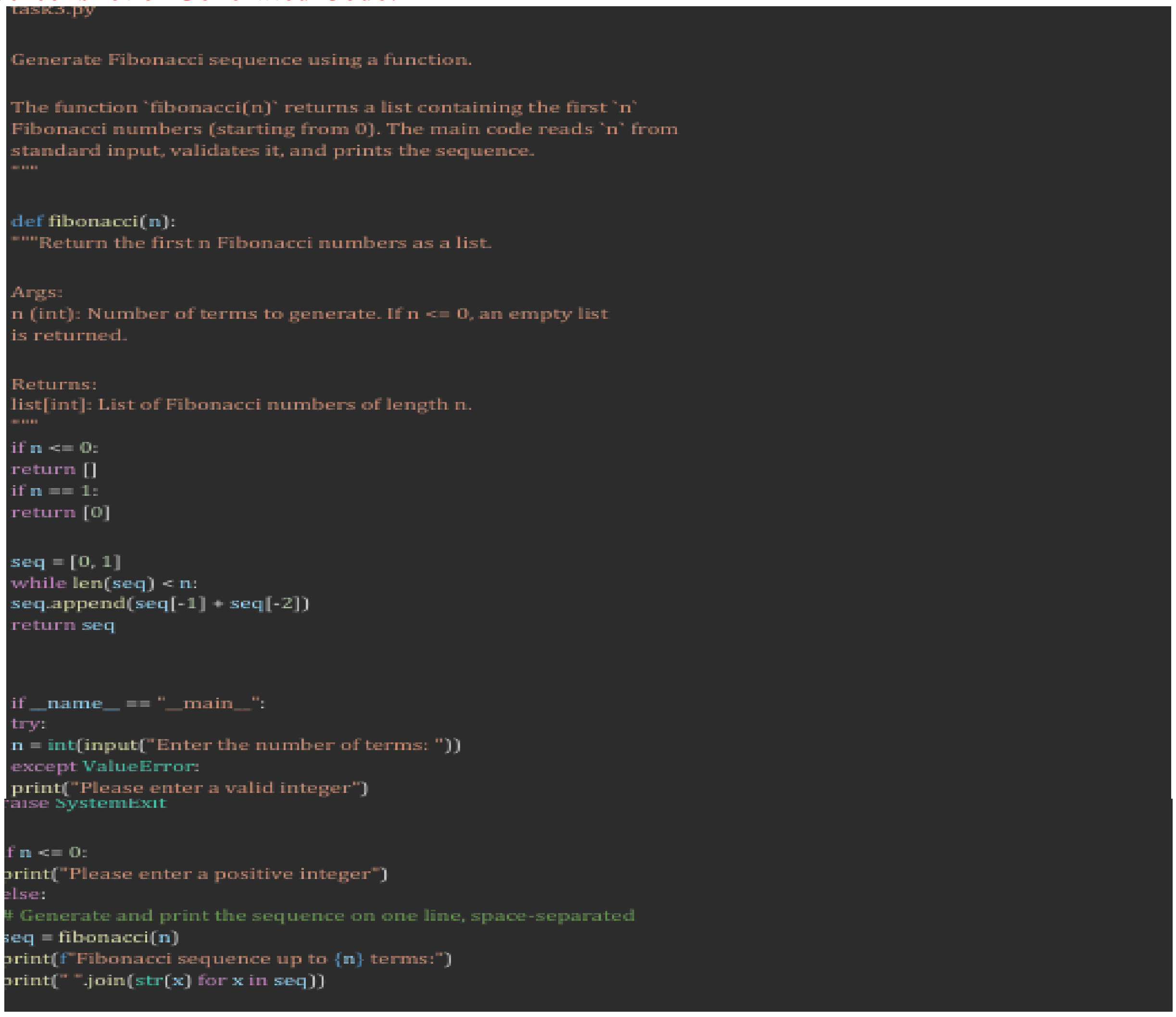
Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)

❖ ***Scenario :The Fibonacci logic is now required in multiple modules of an application.***

• Prompt used:

# Write a Python program using a function to generate Fibonacci sequence
# The function should take n as input
# Include meaningful comments

• Screenshot of Generated Code:



```
task3.py

Generate Fibonacci sequence using a function.

The function `fibonacci(n)` returns a list containing the first `n`
Fibonacci numbers (starting from 0). The main code reads `n` from
standard input, validates it, and prints the sequence.
"""

def fibonacci(n):
"""Return the first n Fibonacci numbers as a list.

Args:
n (int): Number of terms to generate. If n <= 0, an empty list
is returned.

Returns:
list[int]: List of Fibonacci numbers of length n.
"""
if n <= 0:
return []
if n == 1:
return [0]

seq = [0, 1]
while len(seq) < n:
seq.append(seq[-1] + seq[-2])
return seq


if __name__ == "__main__":
try:
n = int(input("Enter the number of terms: "))
except ValueError:
print("Please enter a valid integer")
raise SystemExit

f n <= 0:
print("Please enter a positive integer")
else:
# Generate and print the sequence on one line, space-separated
seq = fibonacci(n)
print(f"Fibonacci sequence up to {n} terms:")
print(" ".join(str(x) for x in seq))
```

• Short Explanation of Logic: The program defines a user-defined function that generates the Fibonacci sequence up to n terms. The main program takes input from the user and calls this function. Inside the function, the Fibonacci numbers are calculated using a loop and printed (or returned). Using a function makes the code modular, reusable, and easier to maintain.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

❖ ***Scenario: You are participating in a code review session.***

• Prompt used:

• *Prompt 1:* Basic Comparison Compare a Fibonacci program written without functions and a Fibonacci program written using functions. Analyze the comparison in terms of: - Code clarity - Reusability - Ease of debugging - Suitability for larger systems Provide the answer in simple academic language.

• ***Prompt 2:*** Table Format Create a comparison table between procedural Fibonacci code (without functions) and modular Fibonacci code (with functions). Include parameters such as code clarity, reusability, debugging ease, and scalability.

• ***Prompt 3:*** Short Analytical Explanation Write a short analytical report comparing procedural and modular Fibonacci implementations. Explain which approach is better and why. Limit the explanation to 5–6 lines.

• Sample Output:
Comparison (Fibonacci Programs)
Code clarity: Functions: named blocks (e.g., fib(n)) separate intent from implementation, so the main flow is concise and easy to read; No functions: everything is inline, which can be fine for very short scripts but becomes harder to follow as logic grows.

Reusability: Functions: easily reused, imported, and called with different arguments; No functions: logic is tied to the script and must be copied or refactored to reuse. Ease of debugging: Functions: let you test and isolate small units (unit tests, step through debugging) so locating bugs is simpler; No functions: debugging requires examining the whole linear flow and is more error-prone.

Suitability for larger systems: Functions: enable modular design, easier maintenance, and integration into larger codebases; No functions: poor scalability—hard to extend, test, or integrate into services or libraries.

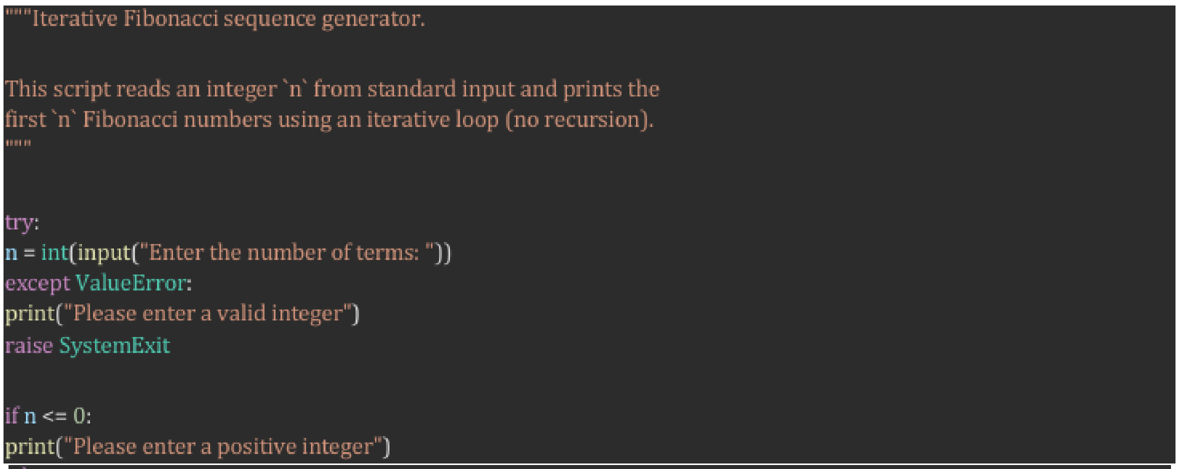
Procedural vs Modular Fibonacci —
Procedural implementations keep logic inline, making very small scripts straightforward but quickly cluttered as logic grows. Modular implementations encapsulate Fibonacci generation in named functions, which improves clarity and documents intent. Functions greatly increase reusability: the same routine can be imported and reused without duplication. Debugging is easier with

functions since units can be tested in isolation and traced independently. For larger systems, modular design scales far better—it's maintainable, testable, and integrates cleanly into other components.

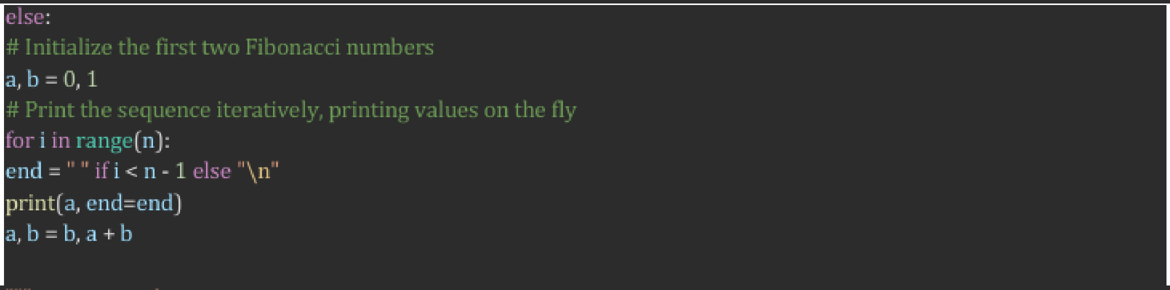
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)
❖ ***Scenario: Your mentor wants to assess AI’s understanding of different algorithmic paradigms.***

• Prompt used:
Iterative Prompt # Generate Fibonacci sequence using iterative approach.
Recursive Prompt # Generate Fibonacci sequence using recursive approach.

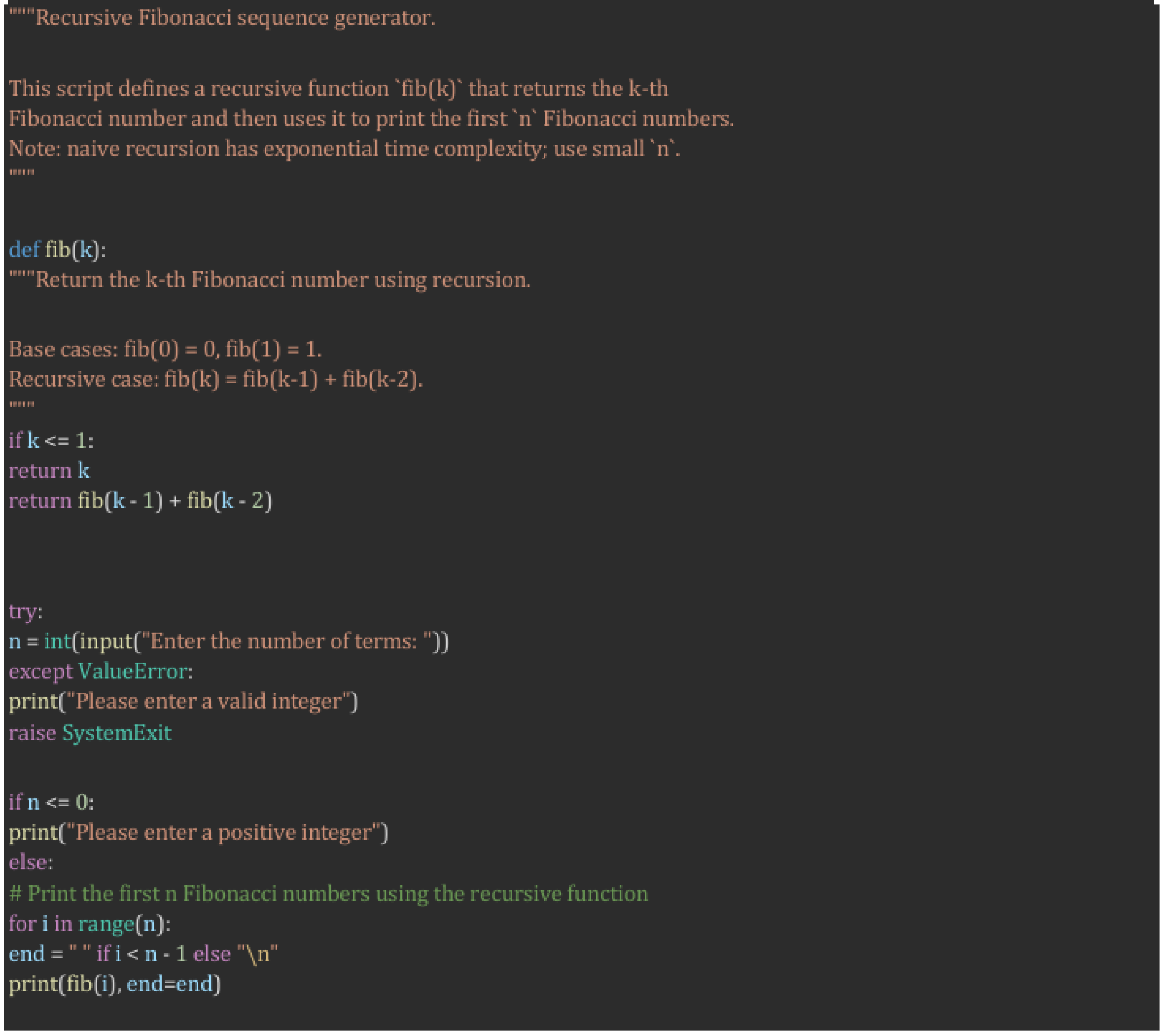
• **Screenshot of Generated Code:**



```
"""Iterative Fibonacci sequence generator.

This script reads an integer `n` from standard input and prints the
first `n` Fibonacci numbers using an iterative loop (no recursion).
"""


try:
n = int(input("Enter the number of terms: "))
except ValueError:
print("Please enter a valid integer")
raise SystemExit

if n <= 0:
print("Please enter a positive integer")
```

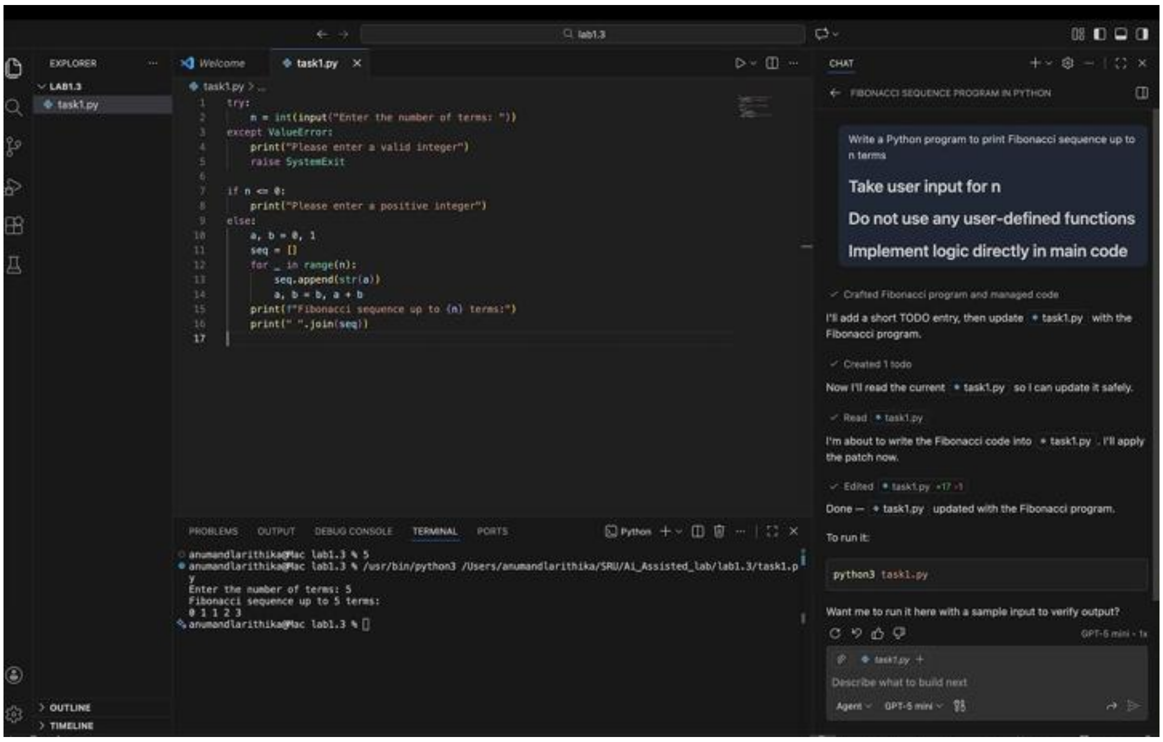


```
else:
# Initialize the first two Fibonacci numbers
a, b = 0, 1
# Print the sequence iteratively, printing values on the fly
for i in range(n):
end = " " if i < n - 1 else "\n"
print(a, end=end)
a, b = b, a + b
```



```
"""Recursive Fibonacci sequence generator.

This script defines a recursive function `fib(k)` that returns the k-th
Fibonacci number and then uses it to print the first `n` Fibonacci numbers.
Note: naive recursion has exponential time complexity; use small `n`.
"""


def fib(k):
"""Return the k-th Fibonacci number using recursion.

Base cases: fib(0) = 0, fib(1) = 1.
Recursive case: fib(k) = fib(k-1) + fib(k-2).
"""
if k <= 1:
return k
return fib(k - 1) + fib(k - 2)



try:
n = int(input("Enter the number of terms: "))
except ValueError:
print("Please enter a valid integer")
raise SystemExit

if n <= 0:
print("Please enter a positive integer")
else:
# Print the first n Fibonacci numbers using the recursive function
for i in range(n):
end = " " if i < n - 1 else "\n"
print(fib(i), end=end)
```

- **Sample Input:** Enter the number of terms: 5
- **Sample Output:** 0 1 1 2 3



```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    a, b = 0, 1
    seq = []
    for _ in range(n):
        seq.append(str(a))
        a, b = b, a + b
    print(f"Fibonacci sequence up to {n} terms:")
    print(" ".join(seq))
```

```
anumandlarithika@Mac lab1.3 % 5
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task1.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % 
```

FIBONACCI SEQUENCE PROGRAM IN PYTHON

Write a Python program to print Fibonacci sequence up to n terms

Take user input for n

Do not use any user-defined functions

Implement logic directly in main code

✓ Crafted Fibonacci program and managed code

I'll add a short TODO entry, then update task1.py with the Fibonacci program.

✓ Created 1 todo

Now I'll read the current task1.py so I can update it safely.

✓ Read task1.py

I'm about to write the Fibonacci code into task1.py. I'll apply the patch now.

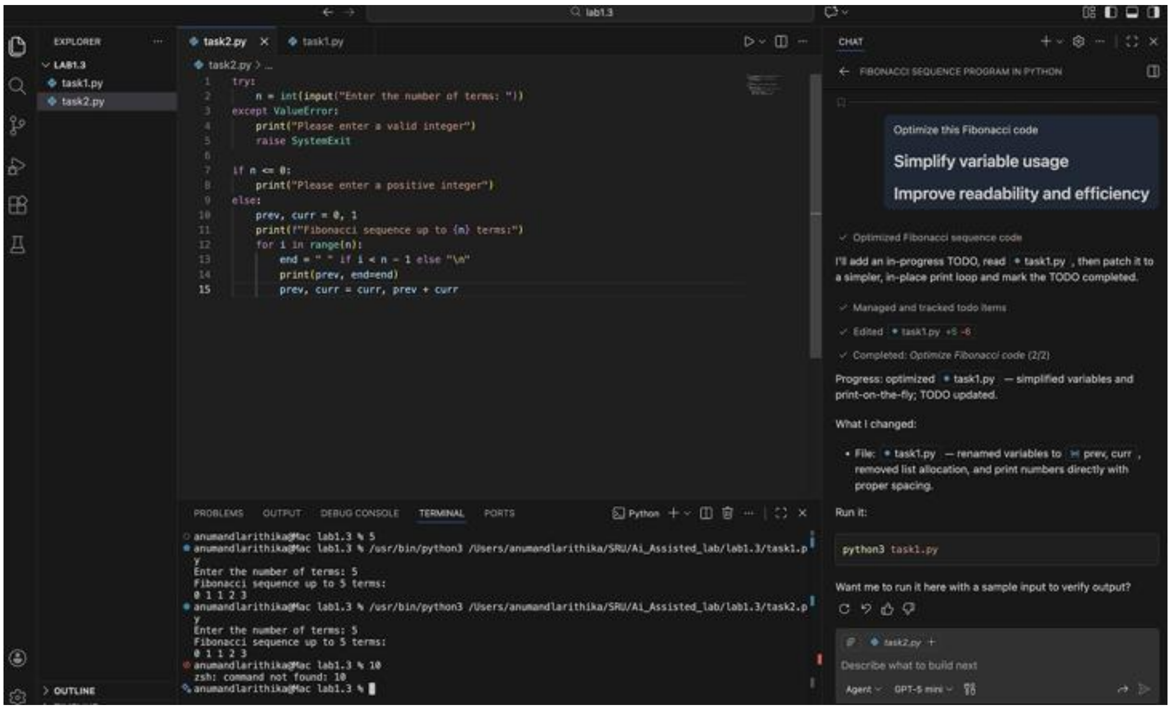
✓ Edited task1.py +17 -1

Done — task1.py updated with the Fibonacci program.

To run it:

```
python3 task1.py
```

Want me to run it here with a sample input to verify output?



```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    prev, curr = 0, 1
    print(f"Fibonacci sequence up to {n} terms:")
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(prev, end=end)
        prev, curr = curr, prev + curr
```

```
anumandlarithika@Mac lab1.3 % 5
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task1.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task2.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % 10
zsh: command not found: 10
anumandlarithika@Mac lab1.3 % 
```

FIBONACCI SEQUENCE PROGRAM IN PYTHON

Optimize this Fibonacci code

Simplify variable usage

Improve readability and efficiency

✓ Optimized Fibonacci sequence code

I'll add an in-progress TODO, read task1.py, then patch it to a simpler, in-place print loop and mark the TODO completed.

✓ Managed and tracked todo items

✓ Edited task1.py +5 -8

✓ Completed: *Optimize Fibonacci code* (2/2)

Progress: optimized task1.py — simplified variables and print-on-the-fly; TODO updated.

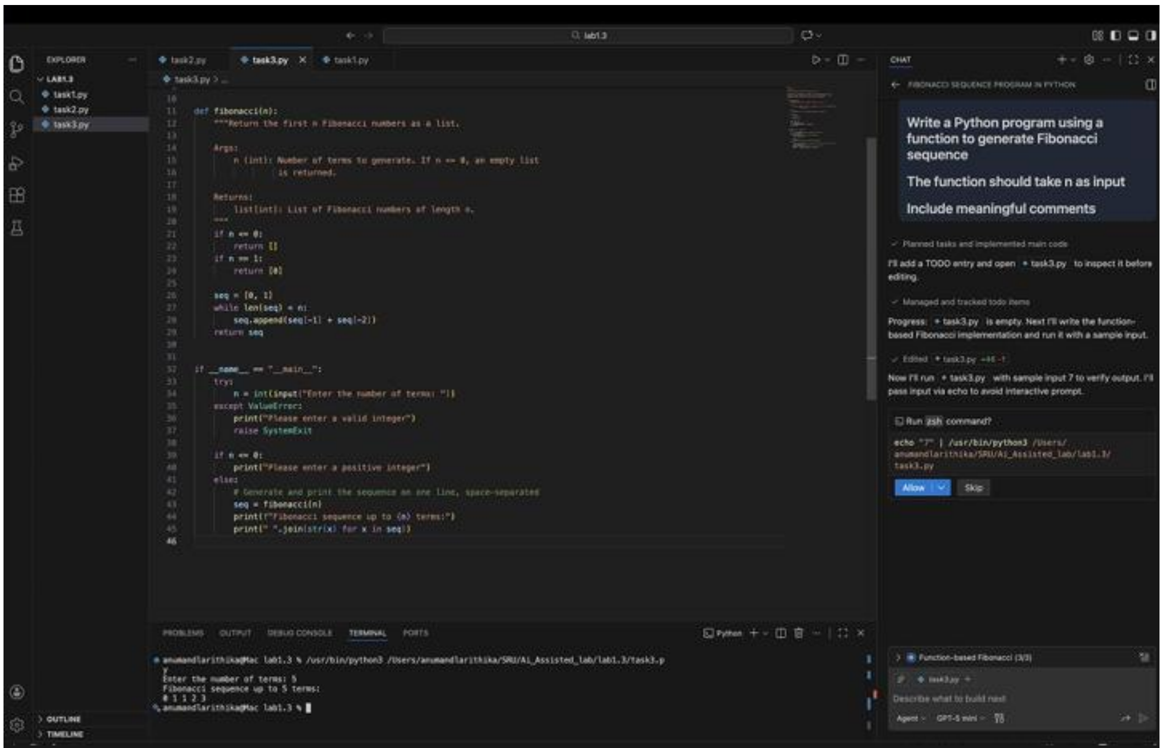
What I changed:

- File: task1.py — renamed variables to prev, curr, removed list allocation, and print numbers directly with proper spacing.

Run it:

```
python3 task1.py
```

Want me to run it here with a sample input to verify output?



```python
def fibonacci(n):
    """Return the first n Fibonacci numbers as a list.

    Args:
        n (int): Number of terms to generate. If n == 0, an empty list
                 is returned.

    Returns:
        list[int]: List of Fibonacci numbers of length n.
    """
    if n == 0:
        return []
    if n == 1:
        return [0]

    seq = [0, 1]
    while len(seq) < n:
        seq.append(seq[-1] + seq[-2])
    return seq


if __name__ == "__main__":
    try:
        n = int(input("Enter the number of terms: "))
    except ValueError:
        print("Please enter a valid integer")
        raise SystemExit

    if n <= 0:
        print("Please enter a positive integer")
    else:
        # Generate and print the sequence on one line, space-separated
        seq = fibonacci(n)
        print(f"Fibonacci sequence up to {n} terms:")
        print(" ".join(str(x) for x in seq))
```

```
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab1.3/task3.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % 
```

FIBONACCI SEQUENCE PROGRAM IN PYTHON

Write a Python program using a function to generate Fibonacci sequence

The function should take n as input

Include meaningful comments

✓ Planned tasks and implemented main code

I'll add a TODO entry and open task3.py to inspect it before editing.

✓ Managed and tracked todo items

Progress: task3.py is empty. Next I'll write the function-based Fibonacci implementation and run it with a sample input.

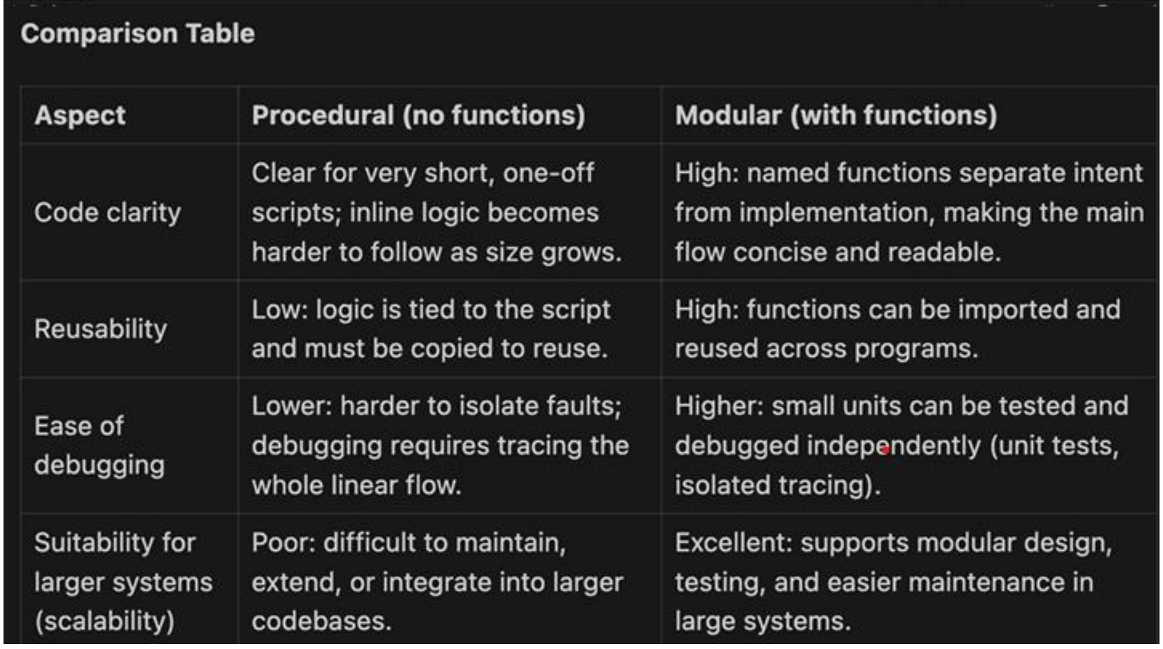
✓ Edited task3.py +46 -1

Now I'll run task3.py with sample input 7 to verify output. I'll pass input via echo to avoid interactive prompt.

Run zsh command?

```
echo "7" | /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab1.3/task3.py
```

EXPLORER
LAB1.3
task1.py
task2.py
task3.py
CHAT
Allow
Skip
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Function-based Fibonacci (3/3)
Describe what to build next
Agent
GPT-5 mini
OUTLINE
TIMELINE



Comparison Table

Aspect	Procedural (no functions)	Modular (with functions)
Code clarity	Clear for very short, one-off scripts; inline logic becomes harder to follow as size grows.	High: named functions separate intent from implementation, making the main flow concise and readable.
Reusability	Low: logic is tied to the script and must be copied to reuse.	High: functions can be imported and reused across programs.
Ease of debugging	Lower: harder to isolate faults; debugging requires tracing the whole linear flow.	Higher: small units can be tested and debugged independently (unit tests, isolated tracing).
Suitability for larger systems (scalability)	Poor: difficult to maintain, extend, or integrate into larger codebases.	Excellent: supports modular design, testing, and easier maintenance in large systems.