Name :B. Sangeeth Reddy

Hall ticket no:2303A53012

Task 1: AI-Generated Logic Without Modularization

Fibonacci Sequence Without Functions

Scenario

A quick Fibonacci sequence generator is required for a learning platform prototype.

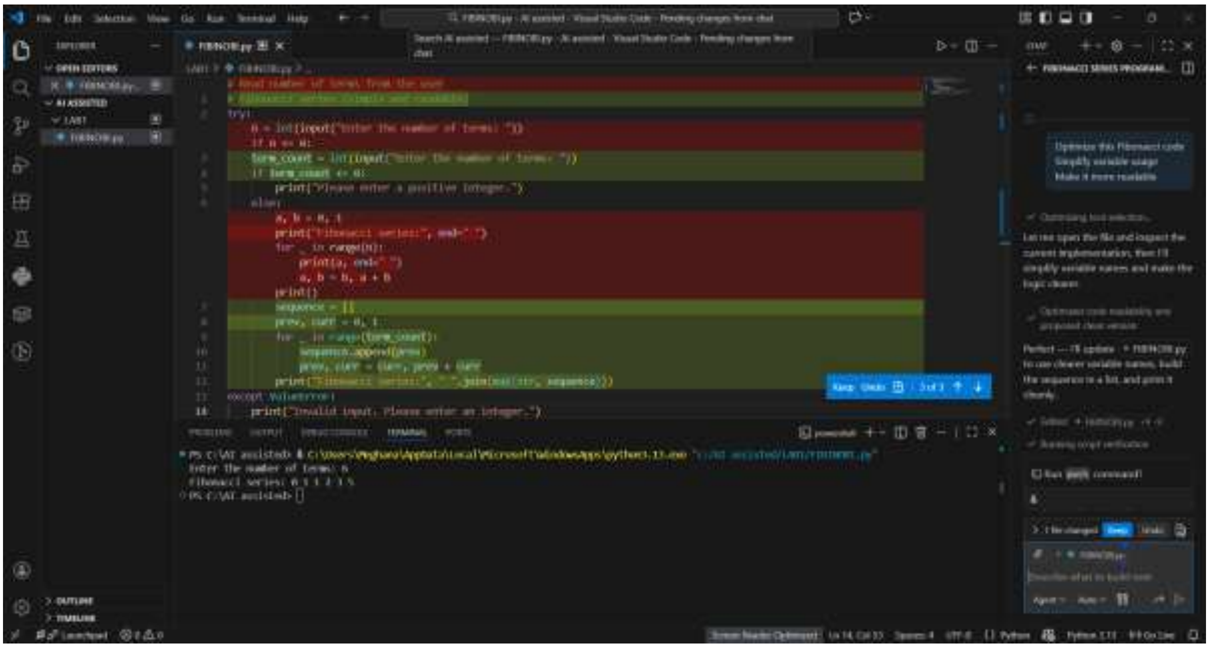
Prompt Given to Copilot

# Write a Python program to print Fibonacci series up to n terms

# Take user input

# Do not use functions

Copilot-Generated Code



Sample Input

Enter number of terms: 7

Sample Output

0 1 1 2 3 5 8

Task 2: AI Code Optimization & Cleanup

Original Code Issues

- Extra temporary variable (c)
- Repeated reassignment logic

- Less readable variable naming
- Initial values printed separately

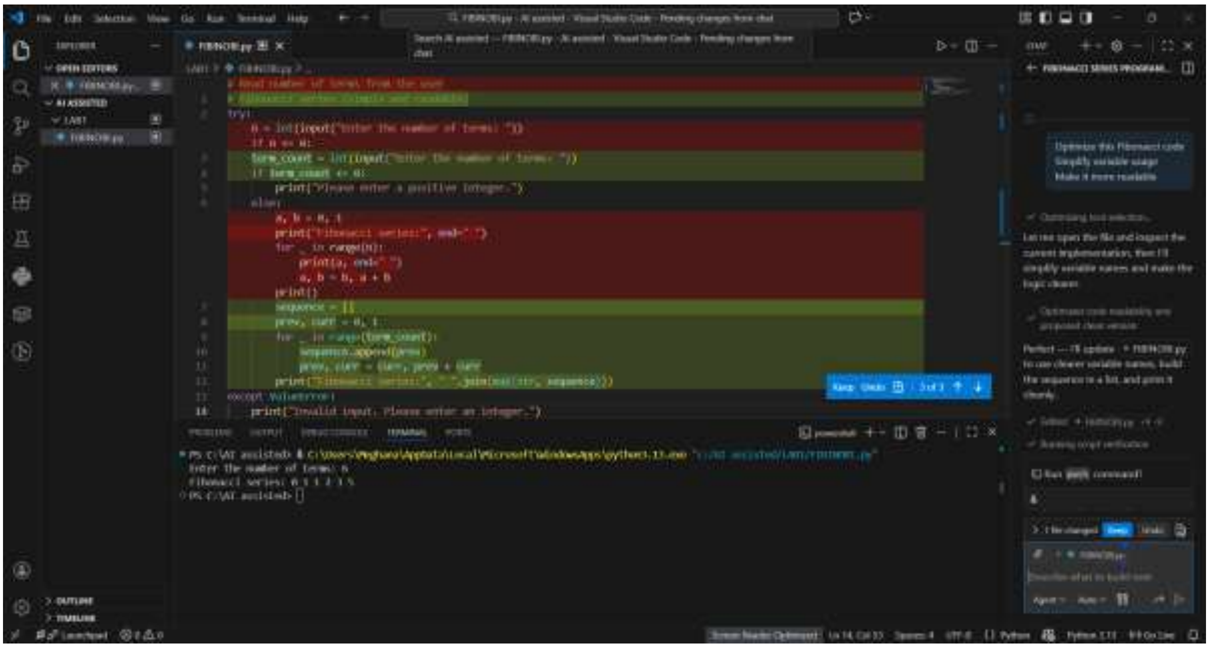
Prompt Given to Copilot

Optimize this Fibonacci code

Simplify variable usage

Make it more readable

Optimized Code



```
# Read number of terms from the user
# Fibonacci series (simple and readable)
try:
    n = int(input("Enter the number of terms: "))
    if n <= 0:
    term_count = int(input("Enter the number of terms: "))
    if term_count <= 0:
        print("Please enter a positive integer.")
    else:
        a, b = 0, 1
        print("Fibonacci series:", end=" ")
        for _ in range(n):
            print(a, end=" ")
            a, b = b, a + b
        print()
        sequence = []
        prev, curr = 0, 1
        for _ in range(term_count):
            sequence.append(prev)
            prev, curr = curr, prev + curr
        print("Fibonacci series:", " ".join(map(str, sequence)))
except ValueError:
    print("Invalid input. Please enter an integer.")
```

```
Enter the number of terms: 6
Fibonacci series: 0 1 1 2 3 5
PS C:\AI assisted>
```

Improvements Explained

Aspect	**Original Code**	**Optimized Code**
Variables	Multiple temporary variables	Minimal variables
Readability	Moderate	High
Loop Logic	Complex	Simplified
Performance	Same complexity	Cleaner execution

Conclusion:
The optimized version improves readability, reduces redundancy, and follows Pythonic coding practices while maintaining the same time complexity.

Task 3: Modular Design Using AI Assistance

Fibonacci Using Functions

Scenario

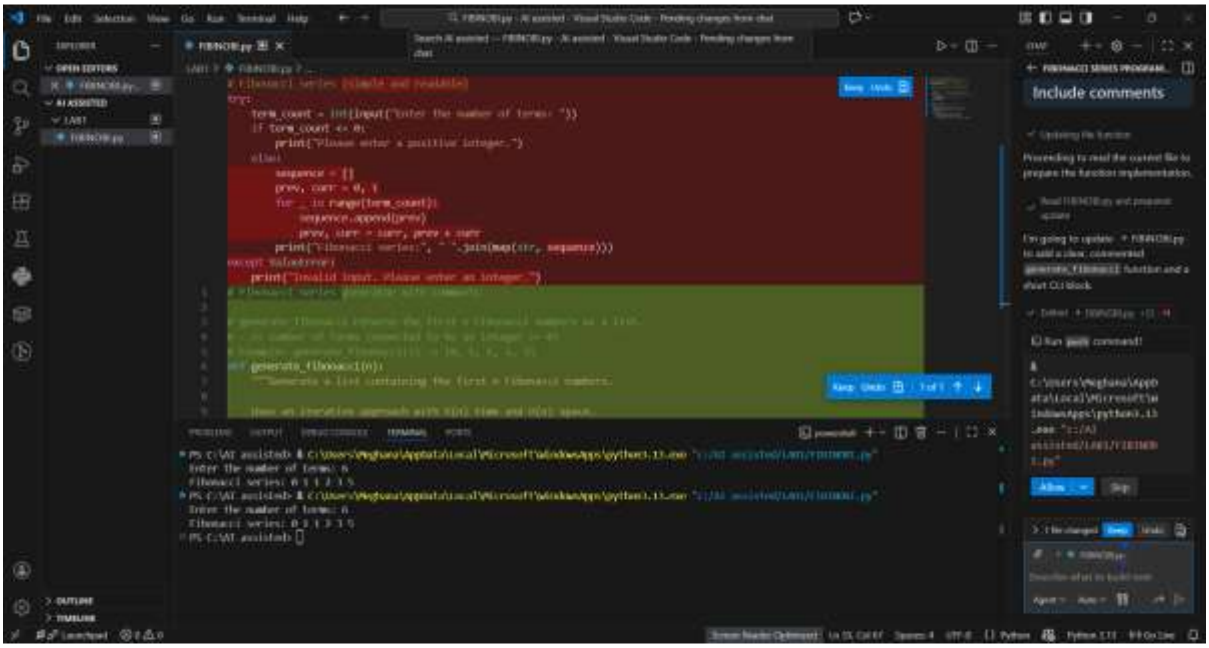
The Fibonacci logic must be reused across multiple modules.

Prompt Given to Copilot

# Write a Python function to generate Fibonacci series up to n terms

# Include comments

Function-Based Code



Include comments

Sample Input

Enter number of terms: 6

Sample Output

0 1 1 2 3 5

Task 4: Comparative Analysis – Procedural vs Modular Code

Criteria	**Without Functions**	**With Functions**
Code Clarity	Lower	Higher
Reusability	Poor	Excellent
Debugging	Difficult	Easier
Scalability	Not suitable	Suitable
Maintainability	Low	High

Conclusion:
Function-based (modular) code is better suited for large systems due to improved readability, reusability, and maintainability.

Task 5: Iterative vs Recursive Fibonacci Approaches

Iterative Fibonacci (Copilot Generated)

```
def fibonacci_iterative(n):
    a, b = 0, 1
    for i in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

Execution Flow

- Uses loop
- Updates values step-by-step
- Efficient memory usage

Recursive Fibonacci (Copilot Generated)

```
def fibonacci_recursive(n):
    if n <= 1:
        return n
    return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

n = int(input("Enter number: "))
for i in range(n):
    print(fibonacci_recursive(i), end=" ")
```

Execution Flow

- Function calls itself
- Uses call stack
- Recomputes values repeatedly

Comparison

Aspect	Iterative	Recursive
Time Complexity	$O(n)$	$O(2^n)$

Aspect	**Iterative**	**Recursive**
Space Complexity	O(1)	O(n)
Performance (large n)	Fast	Very slow
Stack Overflow Risk	No	Yes
Recommended Usage	Large inputs	Educational only

Conclusion:
Recursion should be avoided for large n due to high time complexity and memory overhead. Iterative solutions are more efficient and practical.

Overall Conclusion

This lab successfully demonstrated:

- Installation and usage of GitHub Copilot
- AI-assisted code generation
- Optimization through prompt engineering
- Modular vs procedural design
- Iterative vs recursive algorithmic approaches

GitHub Copilot significantly improves development speed, but human evaluation is essential to ensure correctness, efficiency, and maintainability.