Name: P.Sai Venkatesh

H.No: 2303A53039 - B46

Lab Assignment – 1.3

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

Lab Objectives:

- To install and configure GitHub Copilot in Visual Studio Code.
- To explore AI-assisted code generation using GitHub Copilot.
- To analyze the accuracy and effectiveness of Copilot's code suggestions.
- To understand prompt-based programming using comments and code context

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

Task 0

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

Expected Output

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

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.

❖ **Task Description**

Use GitHub Copilot to generate a Python program that:

- Prints the Fibonacci sequence up to *n* terms
- Accepts user input for *n*
- Implements the logic directly in the main code
- Does not use any user-defined functions

❖ **Expected Output**

- Correct Fibonacci sequence for given *n*
- Screenshot(s) showing Copilot-generated suggestions
- Sample inputs and outputs

Prompt:

1. **#write a python program to print the fibonacci series up to n terms using user input implements the logic directly in the main code without using any function.**

Code:

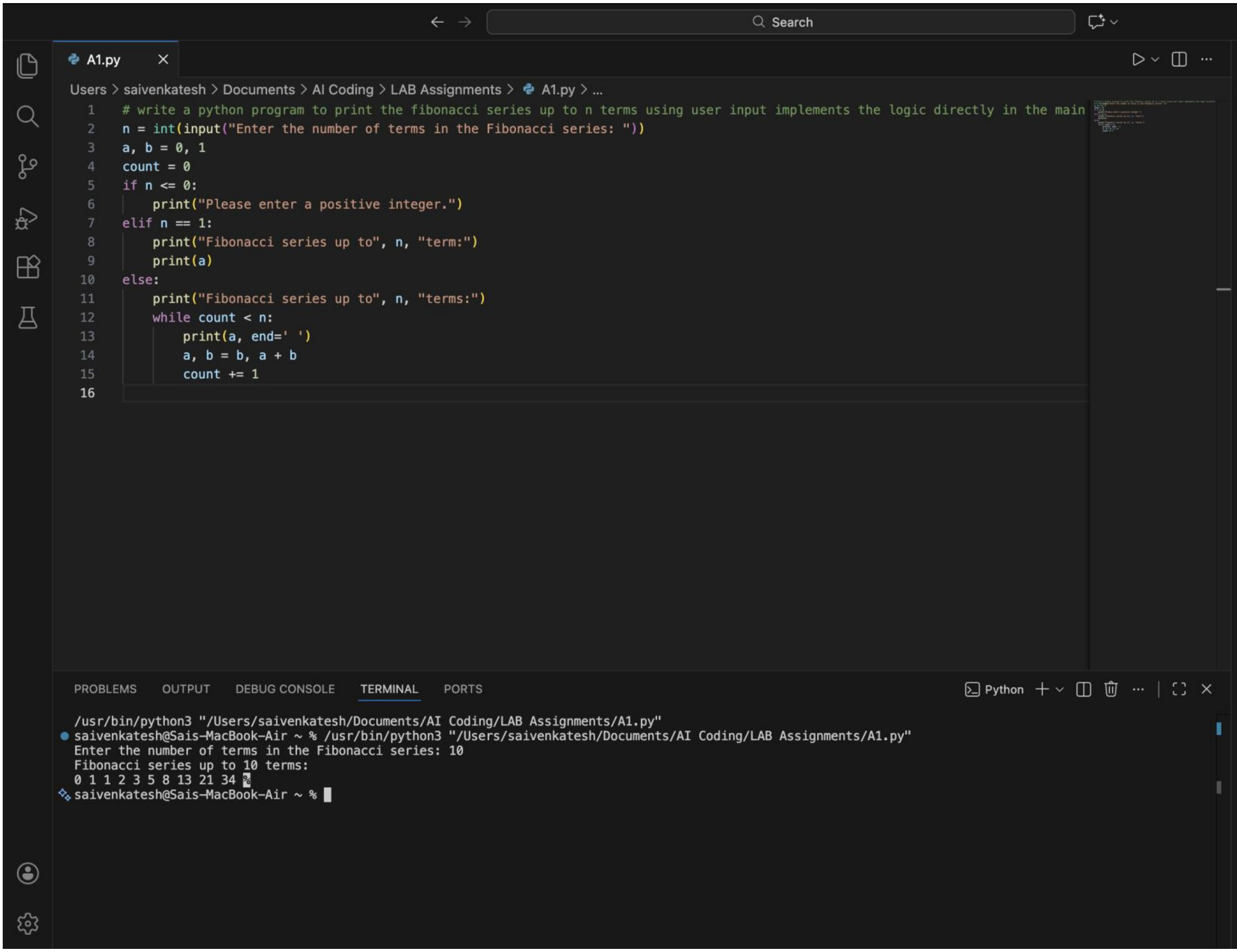
```
n = int(input("Enter the number of terms in the Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print("Fibonacci series up to", n, "term:")
    print(a)
else:
    print("Fibonacci series up to", n, "terms:")
    while count < n:
        print(a, end=' ')
        a, b = b, a + b
        count += 1
```

Output:

Enter the number of terms in the Fibonacci series: 10

Fibonacci series up to 10 terms:

0 1 1 2 3 5 8 13 21 34



Explaination(3 to 6 lines):

In this program, the number of terms for the Fibonacci series is taken from the user.

The first two values are initialized, and the remaining terms are generated using a loop.
Each new number is obtained by adding the previous two numbers.
The entire logic is written directly in the main program without using any functions.

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

❖ **Scenario**

The prototype will be shared with other developers and needs optimization.

❖ **Task Description**

- ➢ Examine the Copilot-generated code from Task 1 and improve it by:
- ➢ Removing redundant variables
- ➢ Simplifying loop logic
- ➢ Avoiding unnecessary computations
- ➢ Use Copilot prompts such as:
  - *"Optimize this Fibonacci code"*
  - *"Simplify variable usage"*

Hint:
Prompt Copilot with phrases like
"optimize this code", *"simplify logic"*, or *"make it more readable"*

❖ **Expected Output**

- ➢ Original vs improved code
- ➢ Written explanation of:

- What was inefficient
- How the optimized version improves performance and readability

Prompt:

#Optimize this fibonacci code and simplify variable usage

Code:

```
n = int(input("Enter the number of terms in the Fibonacci series: "))
fib1, fib2 = 0, 1
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print("Fibonacci series up to", n, ":")
    print(fib1)
else:
    print("Fibonacci series up to", n, ":")
    for _ in range(n):
        print(fib1, end=' ')
        fib1, fib2 = fib2, fib1 + fib2
```

Explaination:

In this optimized version, unnecessary variables and conditions from the previous code were removed.
The Fibonacci numbers are generated using only two variables, which makes the code simpler.
A for loop is used instead of a while loop to reduce extra control variables.
Tuple assignment is used to update values efficiently, improving readability and performance.

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

❖ **Scenario**

The Fibonacci logic is now required in multiple modules of an application.

❖ **Task Description**

Use GitHub Copilot to generate a function-based Python program that:

- Uses a user-defined function to generate Fibonacci numbers
- Returns or prints the sequence up to *n*
- Includes meaningful comments (AI-assisted)

- **Expected Output**
  - Correct function-based Fibonacci implementation
  - Screenshots documenting Copilot’s function generation
  - Sample test cases with outputs

Prompt:

#wrtie a python program that Uses a user-defined function to generate Fibonacci numbers Returns or prints the sequence up to n Includes meaningful comments (AI-assisted)

Code:

```
def generate_fibonacci(n):
    """
    Generate Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to generate.

    Returns:
    list: A list containing the Fibonacci series up to n terms.
    """
    fibonacci_series = []
    a, b = 0, 1
    for _ in range(n):
```

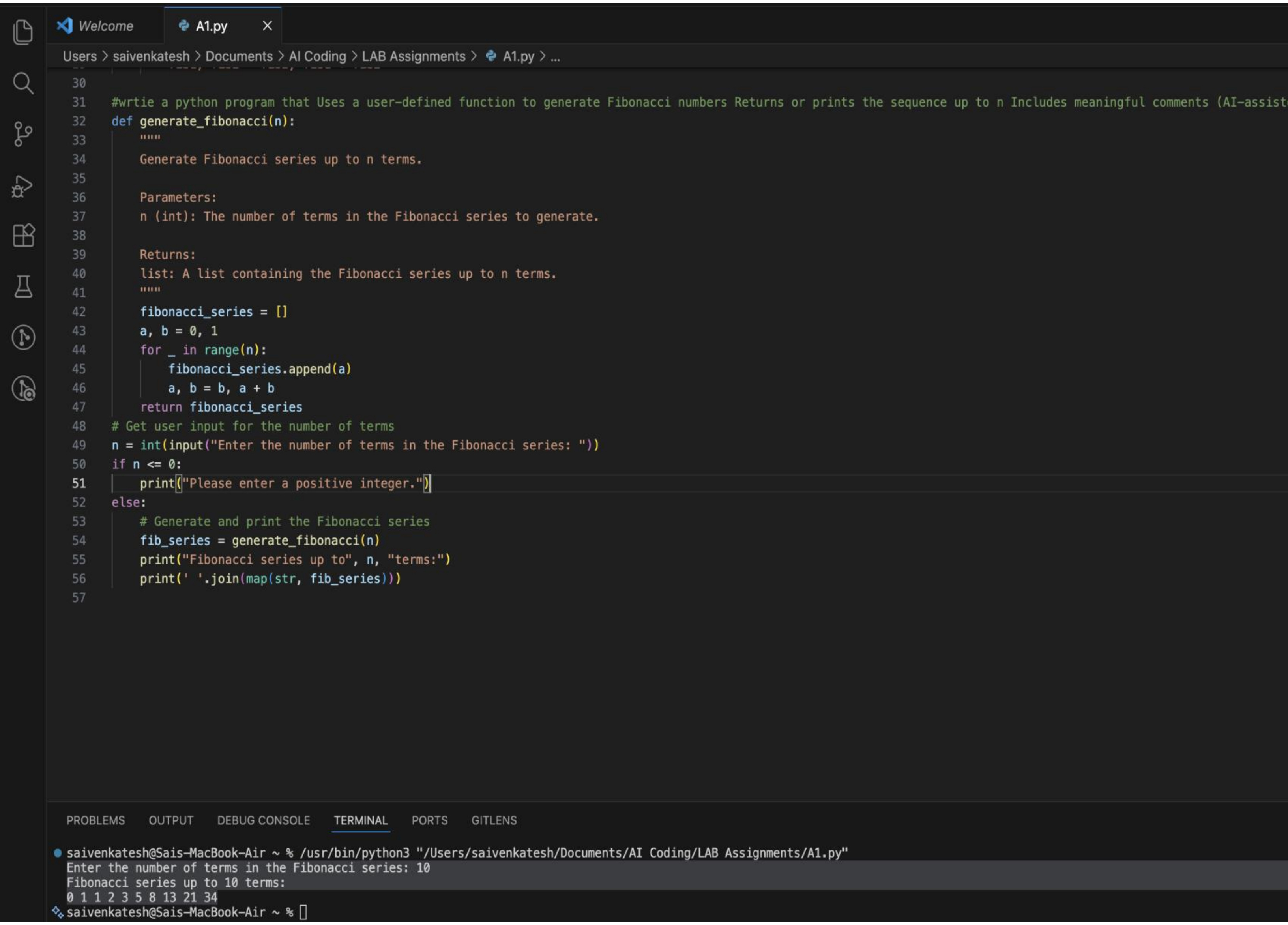
```
        fibonacci_series.append(a)
        a, b = b, a + b
    return fibonacci_series
# Get user input for the number of terms
n = int(input("Enter the number of terms in the Fibonacci series: "))
if n <= 0:
    print("Please enter a positive integer.")
else:
    # Generate and print the Fibonacci series
    fib_series = generate_fibonacci(n)
    print("Fibonacci series up to", n, "terms:")
    print(' '.join(map(str, fib_series)))
```

Output:

Enter the number of terms in the Fibonacci series: 10

Fibonacci series up to 10 terms:

0 1 1 2 3 5 8 13 21 34



Explaination:

In this program, a user-defined function is used to generate the Fibonacci series up to a given number of terms.

The function calculates each Fibonacci number using a loop and stores the values in a list.

User input is taken in the main program and passed to the function as an argument.

The function returns the generated series, which is then displayed as output.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

❖ **Scenario**

You are participating in a code review session.

❖ **Task Description**

Compare the Copilot-generated Fibonacci programs:

- Without functions (Task 1)
- With functions (Task 3)
- Analyze them in terms of:
  - Code clarity
  - Reusability
  - Debugging ease
  - Suitability for larger systems

❖ **Expected Output**

Comparison table or short analytical report

Comparison Table

Aspect	Procedural Fibonacci (Without Functions)	Modular Fibonacci (With Functions)

Code Clarity	Simple but becomes messy as code grows	Clear and well-organized
Reusability	Cannot be reused easily	Function can be reused multiple times
Debugging	Difficult to isolate errors	Easier to debug inside the function
Maintainability	Hard to modify for large programs	Easy to update and maintain
Suitability for Large Systems	Not suitable	Highly suitable

Analysis:

The procedural Fibonacci program is simple and easy to understand for small tasks.
However, it is not reusable and becomes difficult to manage in larger programs.
The modular approach using functions improves code clarity and reusability.
Function-based design makes debugging and maintenance easier and is preferred for real-world applications.

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.

❖ **Task Description**

Prompt GitHub Copilot to generate:

An iterative Fibonacci implementation

A recursive Fibonacci implementation

- **Expected Output**
  - Two correct implementations
  - Explanation of execution flow for both
  - Comparison covering:
    - Time and space complexity
    - Performance for large *n*
    - When recursion should be avoided

Prompt:

#Write an iterative Python program to generate the Fibonacci series up to n terms using a loop

#Write a recursive Python program to generate the Fibonacci series up to n terms with proper base conditions

Code:

```
#Write an iterative Python program to generate the Fibonacci series up to n terms using a loop

n = int(input("Enter the number of terms in the Fibonacci series: "))
a, b = 0, 1

if n <= 0:

    print("Please enter a positive integer.")
```

```
elif n == 1:
        print("Fibonacci series up to", n, ":") print(a)
else:
        print("Fibonacci series up to", n, ":")
        for _ in range(n):
                print(a, end=' ')
                a, b = b, a + b
```

#Write a recursive Python program to generate the Fibonacci series up to n terms with proper base conditions

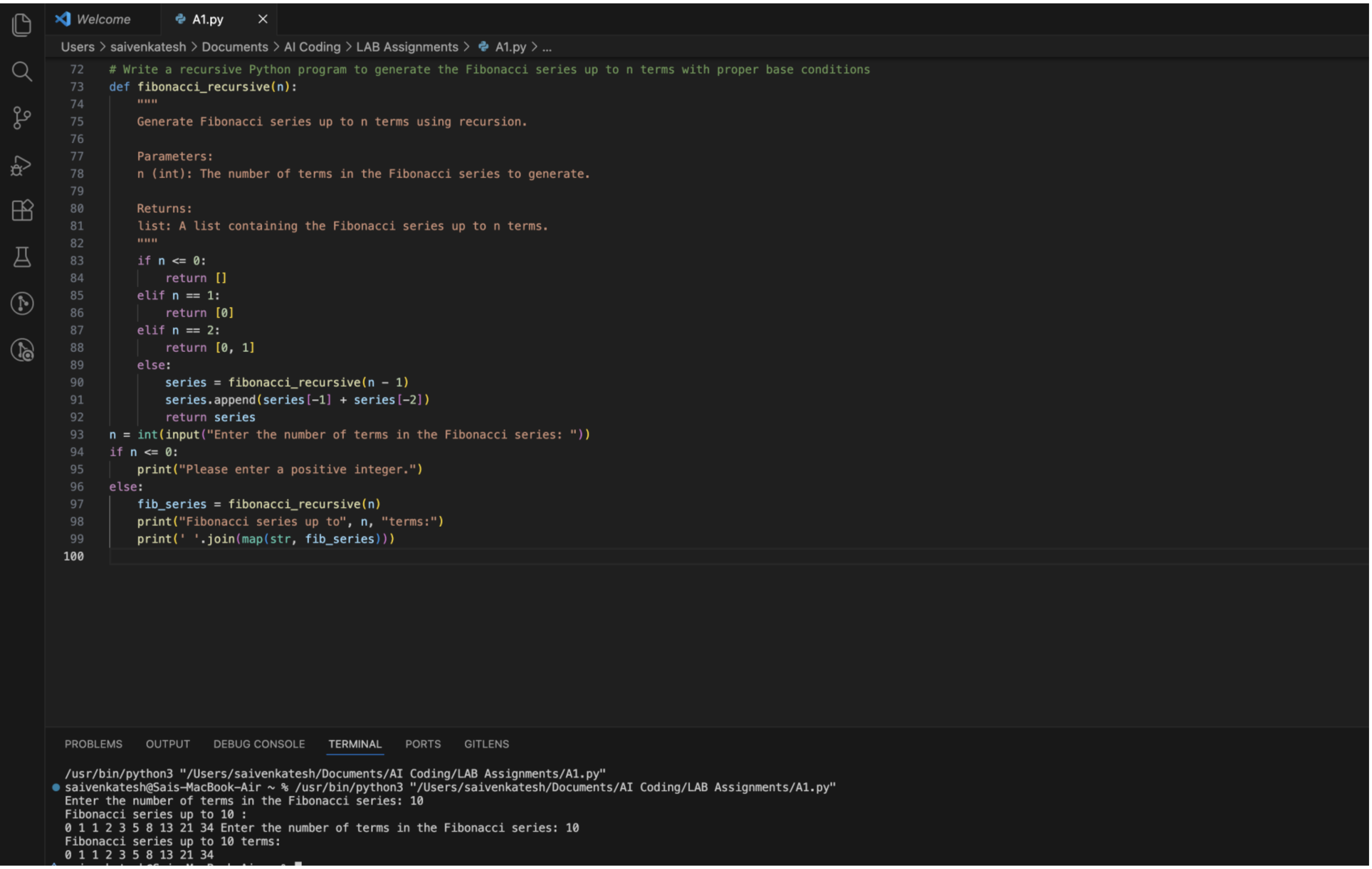
```
def fibonacci_recursive(n):
 """ Generate Fibonacci series up to n terms using recursion.
Parameters:
n (int): The number of terms in the Fibonacci series to generate.
Returns:
list: A list containing the Fibonacci series up to n terms.
"""
if n <= 0:
   return []
elif n == 1:
   return [0]
elif n == 2:
   return [0, 1]
else:
   series = fibonacci_recursive(n - 1)
   series.append(series[-1] + series[-2])
```

```
   return series

n = int(input("Enter the number of terms in the Fibonacci series: "))
 if n <= 0:
        print("Please enter a positive integer.")
else:
         fib_series = fibonacci_recursive(n)
        print("Fibonacci series up to", n, "terms:")
         print(' '.join(map(str, fib_series)))
```

Output:

```
Enter the number of terms in the Fibonacci series: 10
Fibonacci series up to 10 :
0 1 1 2 3 5 8 13 21 34
Enter the number of terms in the Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
```




```python
# Write a recursive Python program to generate the Fibonacci series up to n terms with proper base conditions
def fibonacci_recursive(n):
    """
    Generate Fibonacci series up to n terms using recursion.

    Parameters:
    n (int): The number of terms in the Fibonacci series to generate.

    Returns:
    list: A list containing the Fibonacci series up to n terms.
    """
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci_recursive(n - 1)
        series.append(series[-1] + series[-2])
        return series
n = int(input("Enter the number of terms in the Fibonacci series: "))
if n <= 0:
    print("Please enter a positive integer.")
else:
    fib_series = fibonacci_recursive(n)
    print("Fibonacci series up to", n, "terms:")
    print(' '.join(map(str, fib_series)))
```

```
/usr/bin/python3 "/Users/saivenkatesh/Documents/AI Coding/LAB Assignments/A1.py"
saivenkatesh@Sais-MacBook-Air ~ % /usr/bin/python3 "/Users/saivenkatesh/Documents/AI Coding/LAB Assignments/A1.py"
Enter the number of terms in the Fibonacci series: 10
Fibonacci series up to 10 :
0 1 1 2 3 5 8 13 21 34 Enter the number of terms in the Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
```

Explaination:

In this task, Fibonacci series is implemented using both iterative and recursive approaches.

The iterative method uses a loop and updates values step by step, making it efficient and fast.

The recursive method generates the series by calling the function repeatedly with base conditions.

Iteration uses less memory and works well for large values of n, while recursion is easier to understand but slower.

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.