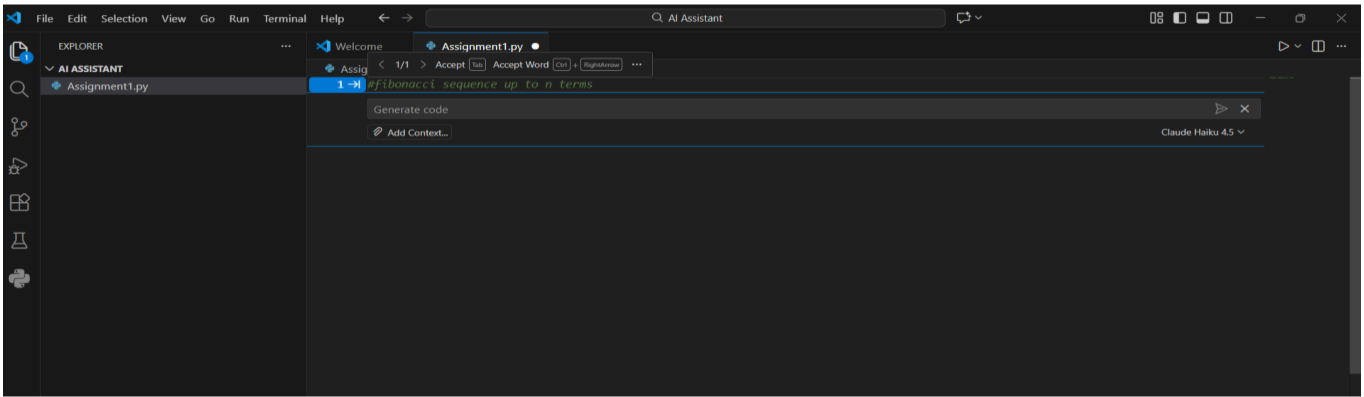
# AI Assistant Coding
# Assignment 1

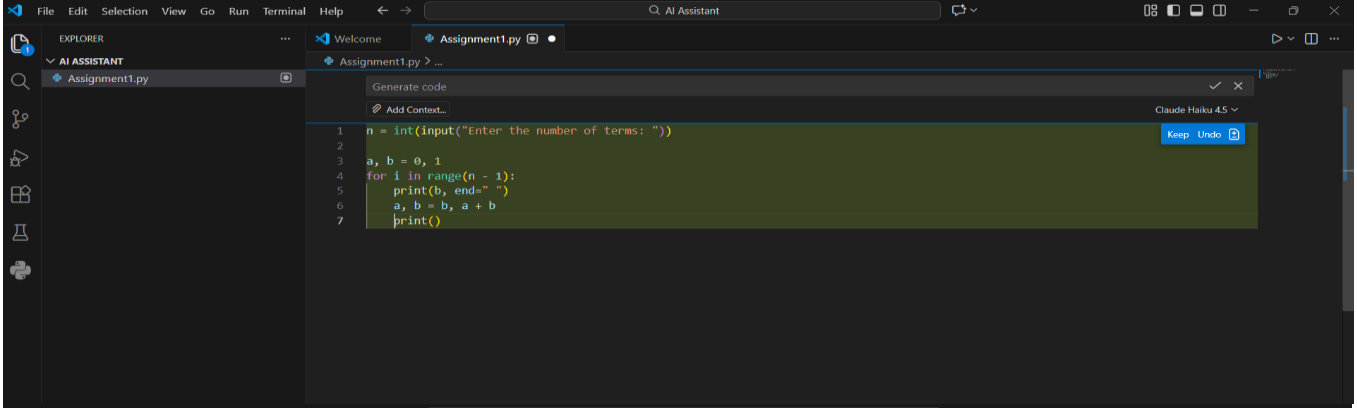
Name :K.sneha
Batch no:20
HT_NO:2303A51332

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

Prompt: Fibonacci Sequence up to n terms Without Functions

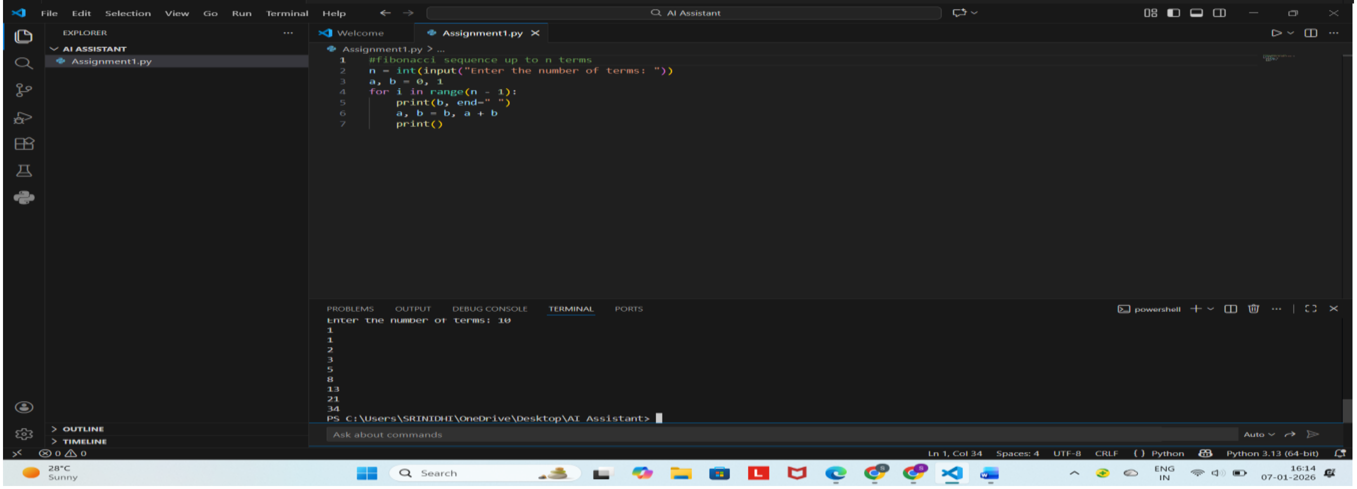


```
#fibonacci sequence up to n terms
```



```
n = int(input("Enter the number of terms: "))

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


File Edit Selection View Go Run Terminal Help
AI Assistant
EXPLORER
AI ASSISTANT
Assignment1.py
Welcome
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Ask about commands
OUTLINE
TIMELINE
Ln 1, Col 34 Spaces: 4 UTF-8 CRLF Python Python 3.13 (64-bit)
28°C Sunny
Search
ENG IN
16:14
07-01-2026

```
#fibonacci sequence up to n terms
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for i in range(n - 1):
    print(b, end=" ")
    a, b = b, a + b
    print()
```

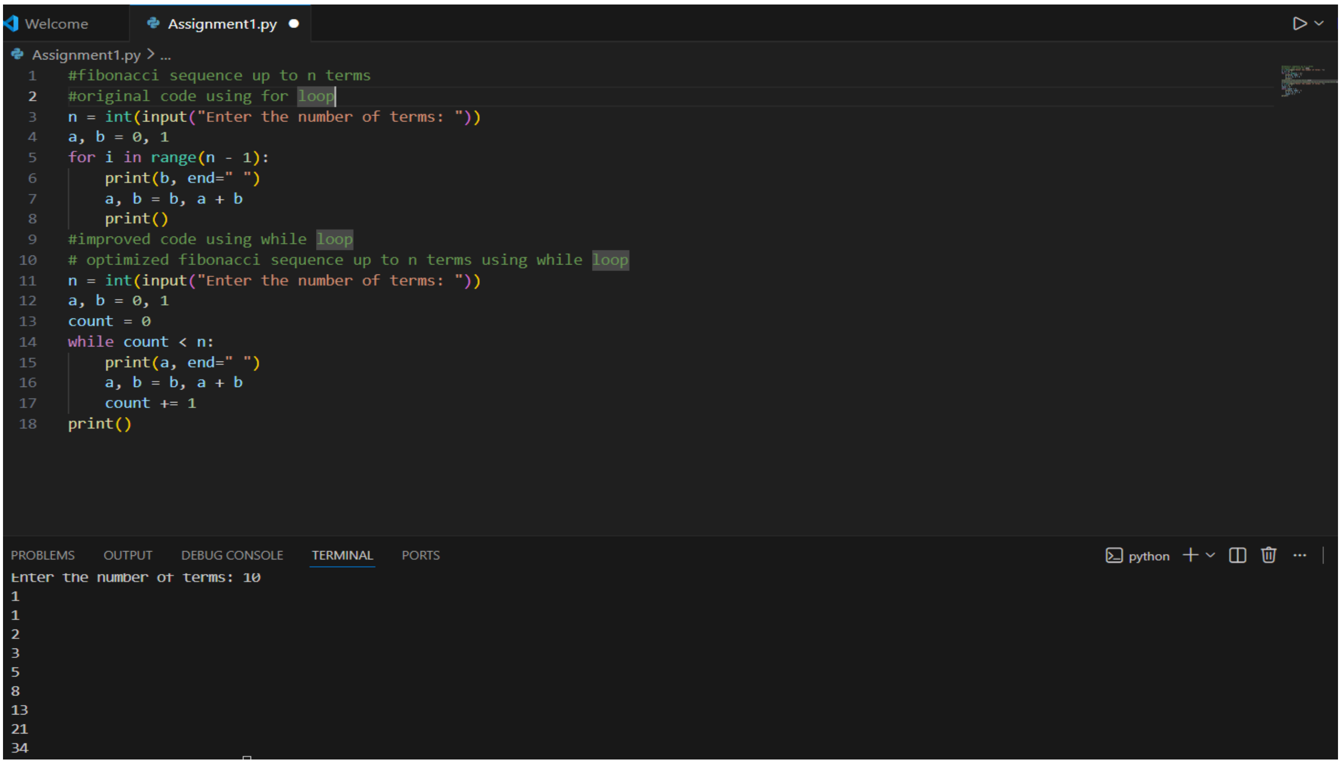
```
Enter the number of terms: 10
1
1
2
3
5
8
13
21
34
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant>
```

Explanation:

This program generates the Fibonacci sequence up to n terms using direct logic without any user-defined functions, which matches the requirement of quick prototyping. The program first takes an integer input n from the user to decide how many terms of the Fibonacci sequence should be printed. Two variables, a and b, are initialized to store consecutive Fibonacci values. A for loop is used to calculate the sequence iteratively. During each iteration of the loop, the current Fibonacci value (b) is printed, and then the variables are updated so that the next Fibonacci number can be computed. This approach avoids recursion and uses simple variable updates, making the logic easy to understand. Overall, the code demonstrates a straightforward, non-modular implementation of the Fibonacci sequence suitable for a learning platform prototype.

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

Prompt: Optimize this Fibonacci code Simplify variable usage



```python
#fibonacci sequence up to n terms
#original code using for loop
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for i in range(n - 1):
    print(b, end=" ")
    a, b = b, a + b
    print()
#improved code using while loop
# optimized fibonacci sequence up to n terms using while loop
n = int(input("Enter the number of terms: "))
a, b = 0, 1
count = 0
while count < n:
    print(a, end=" ")
    a, b = b, a + b
    count += 1
print()
```

```
Enter the number of terms: 10
1
1
2
3
5
8
13
21
34
```

Explanation:

1)What was inefficient

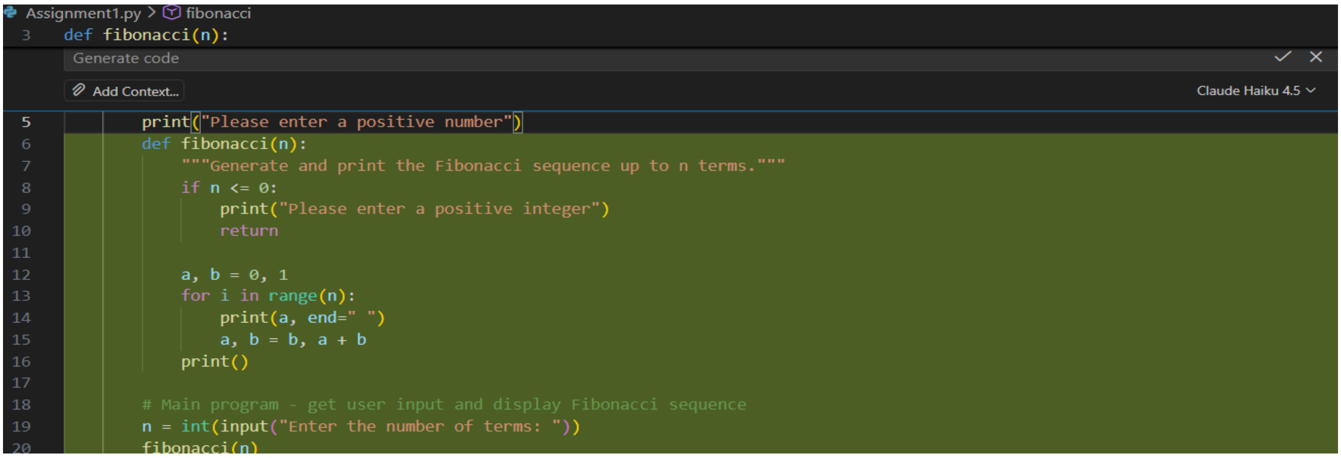
- The original code printed each Fibonacci number on a new line due to an extra `print()` statement inside the loop.
- The loop variable was not used, which made the code less clear.
- The loop range (`n - 1`) made the logic harder to understand.
- The sequence did not clearly start from the first Fibonacci number (`0`), which can confuse readers.
- Output logic and calculation logic were mixed, reducing readability.

2) How the optimized version improves performance and readability

- Unnecessary statements were removed, resulting in cleaner execution.
- The loop runs directly for n terms, making the logic easy to follow.
- The Fibonacci sequence is generated correctly starting from 0.
- Output is printed in a single line, improving clarity.
- Fewer variables and simpler control flow make the code easier to read, maintain, and share with other developers.

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

Prompt: Generate a Python program with a user-defined function that takes user input n and prints or returns the Fibonacci sequence up to n terms, with clear and meaningful comments

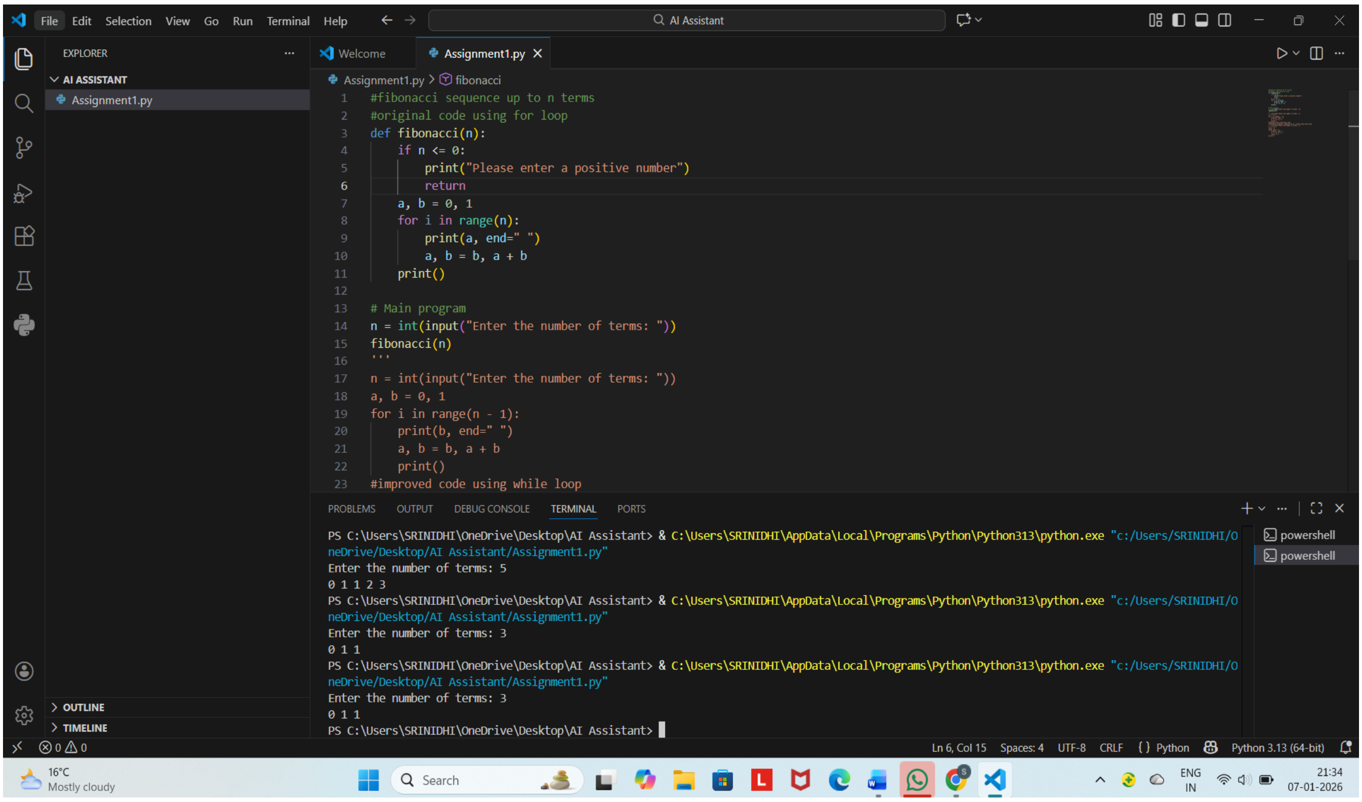


```python
def fibonacci(n):
```

```python
    print("Please enter a positive number")
    def fibonacci(n):
        """Generate and print the Fibonacci sequence up to n terms."""
        if n <= 0:
            print("Please enter a positive integer")
            return

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

    # Main program - get user input and display Fibonacci sequence
    n = int(input("Enter the number of terms: "))
    fibonacci(n)
```



```python
#fibonacci sequence up to n terms
#original code using for loop
def fibonacci(n):
    if n <= 0:
        print("Please enter a positive number")
        return
    a, b = 0, 1
    for i in range(n):
        print(a, end=" ")
        a, b = b, a + b
    print()

# Main program
n = int(input("Enter the number of terms: "))
fibonacci(n)
'''
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for i in range(n - 1):
    print(b, end=" ")
    a, b = b, a + b
    print()
#improved code using while loop
```

```
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/SRINIDHI/O
neDrive/Desktop/AI Assistant/Assignment1.py"
Enter the number of terms: 5
0 1 1 2 3
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/SRINIDHI/O
neDrive/Desktop/AI Assistant/Assignment1.py"
Enter the number of terms: 3
0 1 1
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/SRINIDHI/O
neDrive/Desktop/AI Assistant/Assignment1.py"
Enter the number of terms: 3
0 1 1
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> 
```

## Explanation

- A user-defined function fibonacci(n) is created to generate the Fibonacci sequence.
- The function takes n as a parameter, making it reusable across different modules.
- The Fibonacci sequence is generated iteratively using two variables to maintain efficiency.
- Meaningful comments are included to explain the purpose of each part of the code.
- The main program handles user input and calls the function, keeping logic modular.

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

## Analytical Explanation

The procedural Fibonacci code is suitable for quick demonstrations and small scripts. However, it mixes input, logic, and output, which reduces clarity and makes reuse difficult.

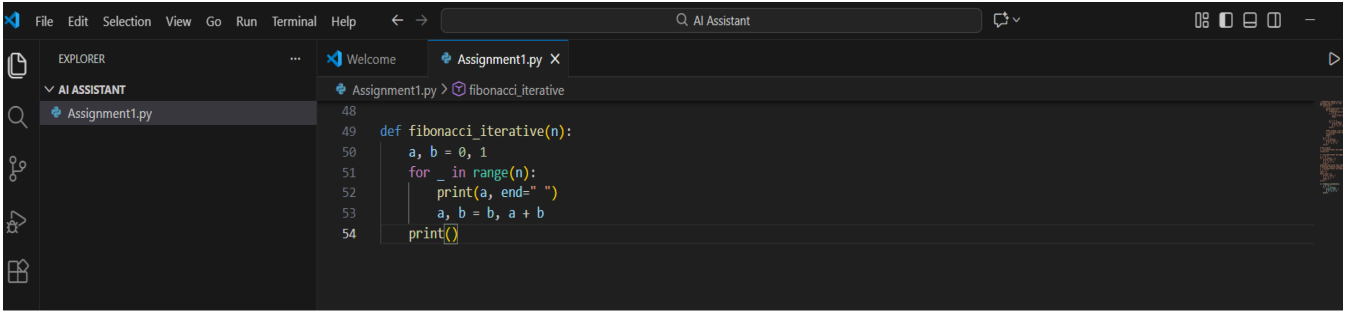
The modular Fibonacci implementation improves structure by placing the logic inside a user-defined function. This allows the same logic to be reused across different modules, simplifies debugging, and makes the code more maintainable. For larger applications, the modular approach is clearly more appropriate.

**Comparison Table**

| Aspect | Procedural Code (Task 1 – Without Function) | Modular Code (Task 3 – With Function) |
|---|---|---|
| Code Clarity | Simple but logic and output are mixed in one place | Clear separation of logic and execution |
| Reusability | Cannot be reused without copying code | Function can be reused in multiple modules |
| Debugging Ease | Harder to isolate logic-related errors | Easier to test and debug the function independently |
| Suitability for Larger Systems | Not suitable; causes code duplication | Suitable; supports scalability and maintainability |

## Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

1. **Iterative Fibonacci Implementation**

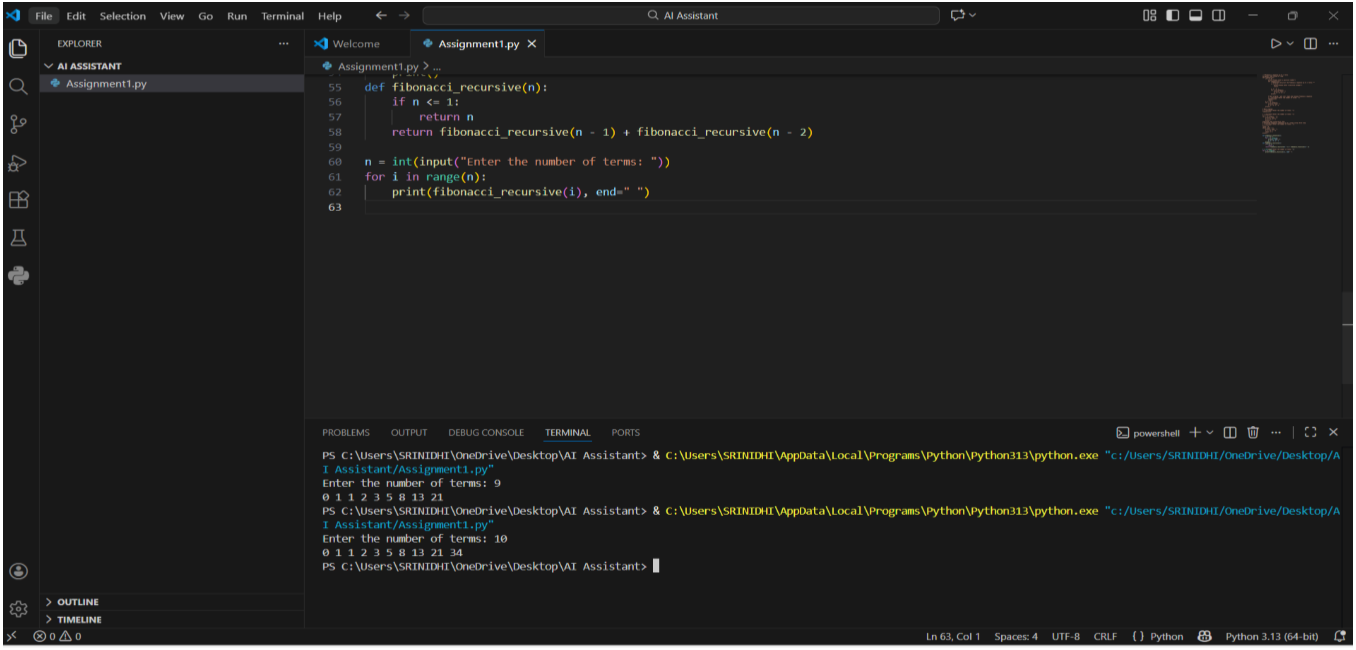


```python
def fibonacci_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
    print()
```

Execution Flow (Iterative)

- Two variables store the previous two Fibonacci numbers.
- A loop runs n times.
- In each iteration, the next Fibonacci number is calculated using simple addition.
- No repeated calculations are performed.

2. Recursive Fibonacci Implementation



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

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

```
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/SRINIDHI/OneDrive/Desktop/A
I Assistant/Assignment1.py"
Enter the number of terms: 9
0 1 1 2 3 5 8 13 21
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> & C:\Users\SRINIDHI\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/SRINIDHI/OneDrive/Desktop/A
I Assistant/Assignment1.py"
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\SRINIDHI\OneDrive\Desktop\AI Assistant> 
```

**Execution Flow (Recursive)**

- The function calls itself to compute smaller Fibonacci values.
- Base cases (n = 0 or n = 1) stop further recursion.
- Results are calculated as recursive calls return.
- Many Fibonacci values are recalculated multiple times.

## Comparison of Iterative vs Recursive Approaches

| Aspect | Iterative | Recursive |
|---|---|---|
| **Time Complexity** | O(n) | $O(2^n)$ |
| **Space Complexity** | O(1) | O(n) (call stack) |
| **Performance for Large n** | Very efficient | Very slow |
| **Memory Usage** | Low | High |
| **Risk of Stack Overflow** | No | Yes |

**When Recursion Should Be Avoided**

- When n is large.
- When performance is important.
- When memory usage must be minimized.
- In production-level or real-time systems.