# AI Assisted Coding Lab 1

Pranitha Gujja

2303A52171

Batch – 41

## Question 1:

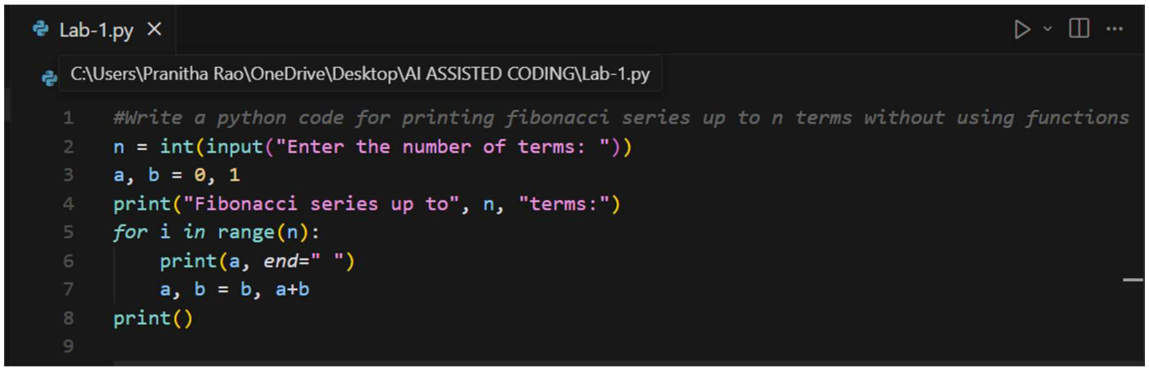
Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence

Without Functions)

## Prompt:

write a python code for printing fibonacci series upto n terms without using functions

## Code:

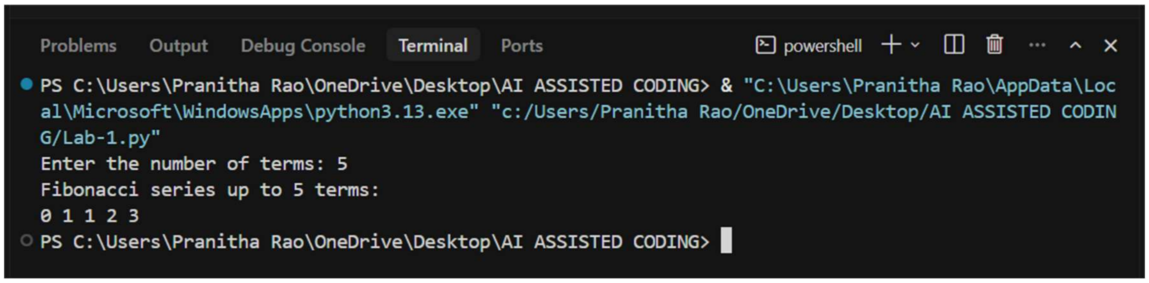


Lab-1.py

C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-1.py

```python
#Write a python code for printing fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a+b
print()
```

## Output:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Loc
al\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODIN
G/Lab-1.py"
Enter the number of terms: 5
Fibonacci series up to 5 terms:
0 1 1 2 3
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## Justification:

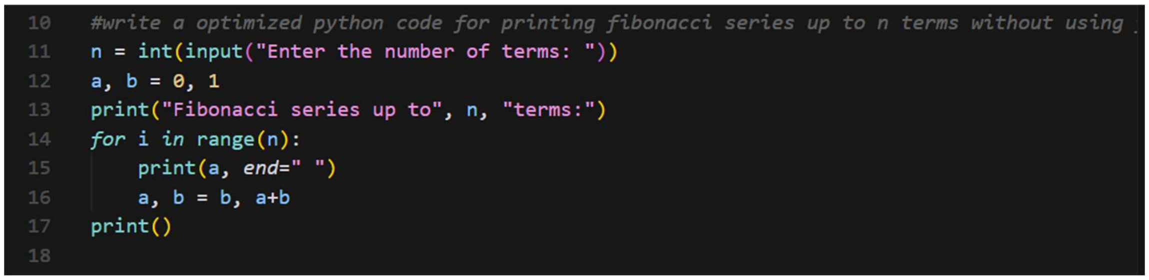
This program generates the Fibonacci series up to a given number of terms without using any functions. All the logic is written directly inside the main program, which makes it easy to follow and understand. It works properly for the provided input values. This method is suitable for small programs or for quick practice, but it is not very efficient for large numbers. The program takes input from the user and displays the Fibonacci sequence up to the specified nth term.

Question 2:

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

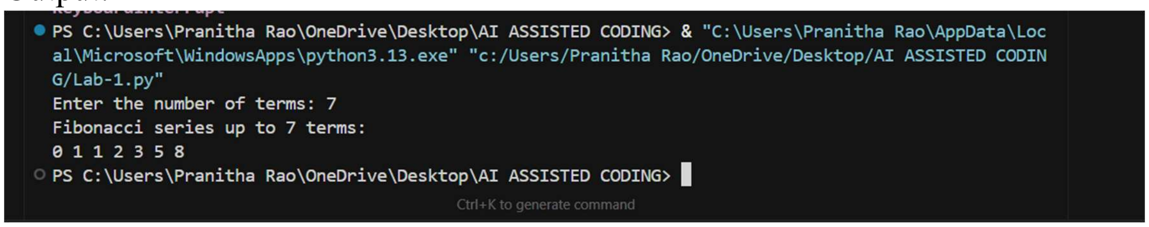
write a optimized python code for printing fibonacci series up to n terms without using functions

Code:



```python
#write a optimized python code for printing fibonacci series up to n terms without using
n = int(input("Enter the number of terms: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a+b
print()
```

Output:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Loc
al\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODIN
G/Lab-1.py"
Enter the number of terms: 7
Fibonacci series up to 7 terms:
0 1 1 2 3 5 8
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

Justification:

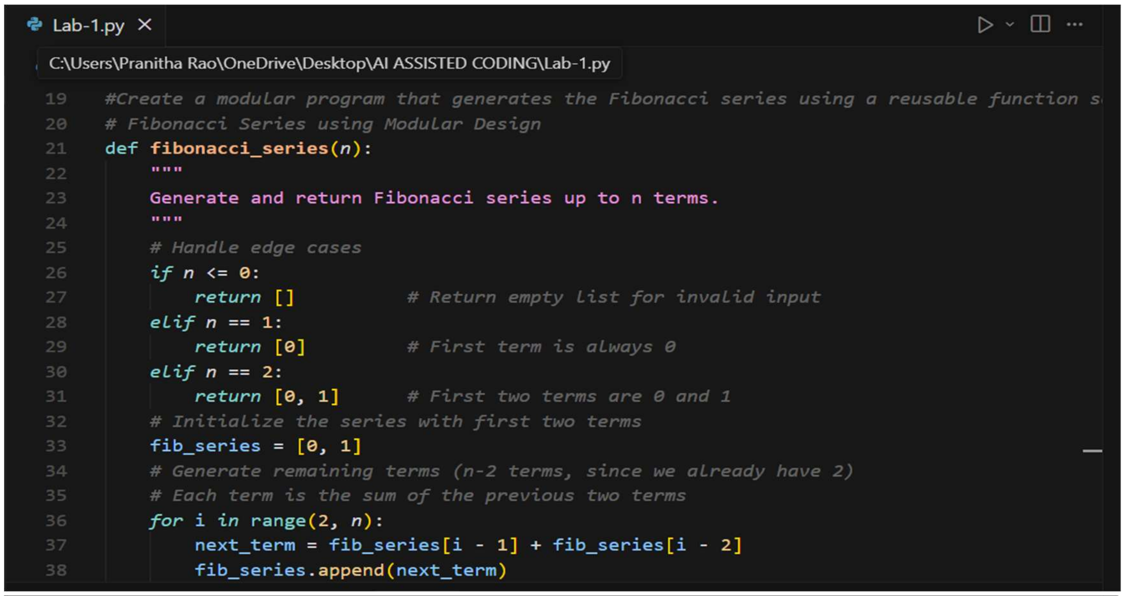
This program shows an optimized version of the Fibonacci series code by reducing extra variables and simplifying the logic. When compared to Task 1, unnecessary lines of code are removed. The program uses fewer lines, making it easier to understand and maintain. Since extra steps are avoided, the execution is faster. Even after optimization, the output remains accurate. Overall, the code becomes cleaner and more efficient.

Question 3:

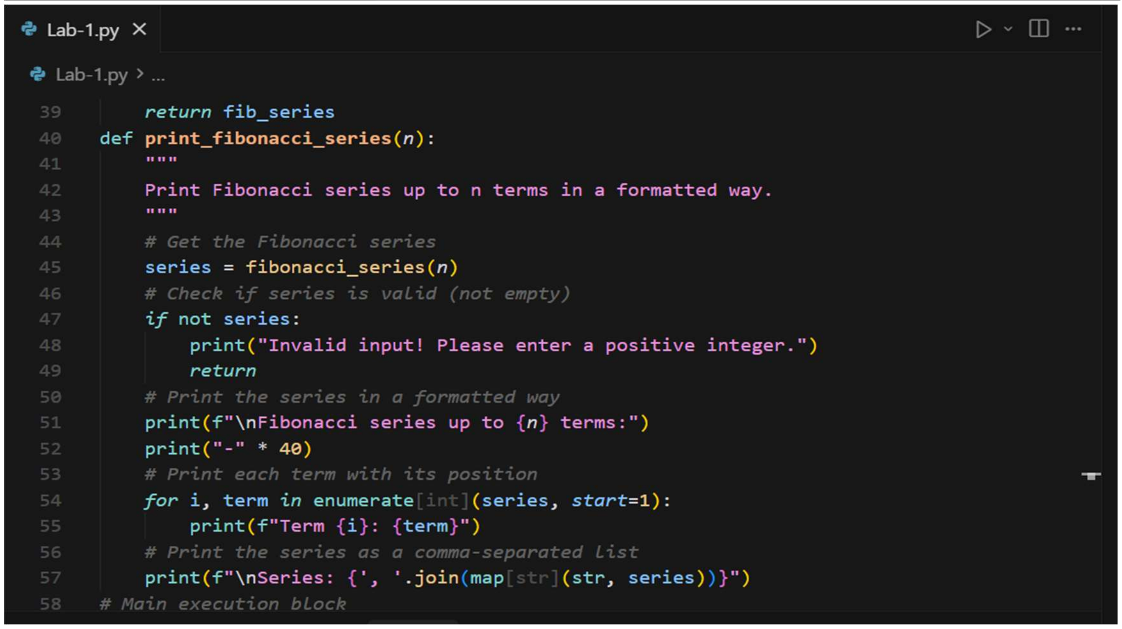
Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions) Prompt:

Write a python code for printing fibonacci series upto n terms and add meaningful comments

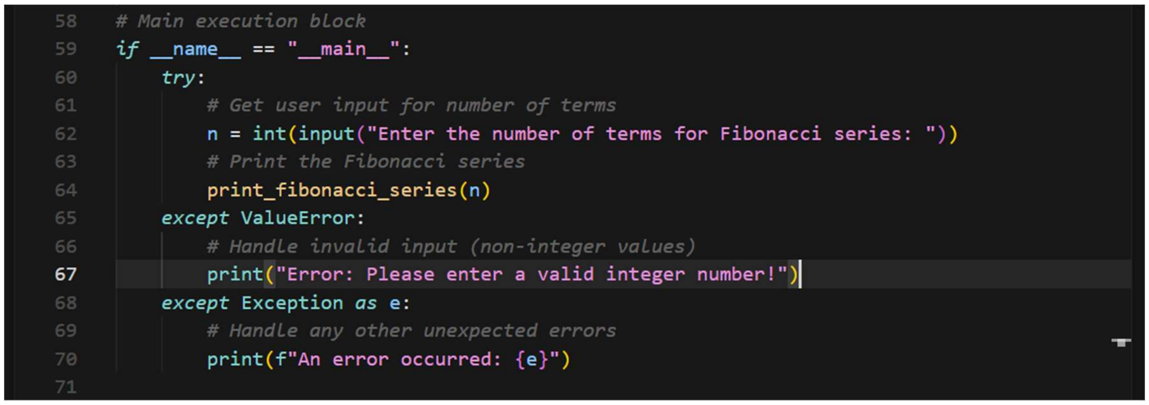
Code:


Lab-1.py
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-1.py

```python
#Create a modular program that generates the Fibonacci series using a reusable function s
# Fibonacci Series using Modular Design
def fibonacci_series(n):
    """
    Generate and return Fibonacci series up to n terms.
    """
    # Handle edge cases
    if n <= 0:
        return []           # Return empty list for invalid input
    elif n == 1:
        return [0]          # First term is always 0
    elif n == 2:
        return [0, 1]       # First two terms are 0 and 1
    # Initialize the series with first two terms
    fib_series = [0, 1]
    # Generate remaining terms (n-2 terms, since we already have 2)
    # Each term is the sum of the previous two terms
    for i in range(2, n):
        next_term = fib_series[i - 1] + fib_series[i - 2]
        fib_series.append(next_term)
```

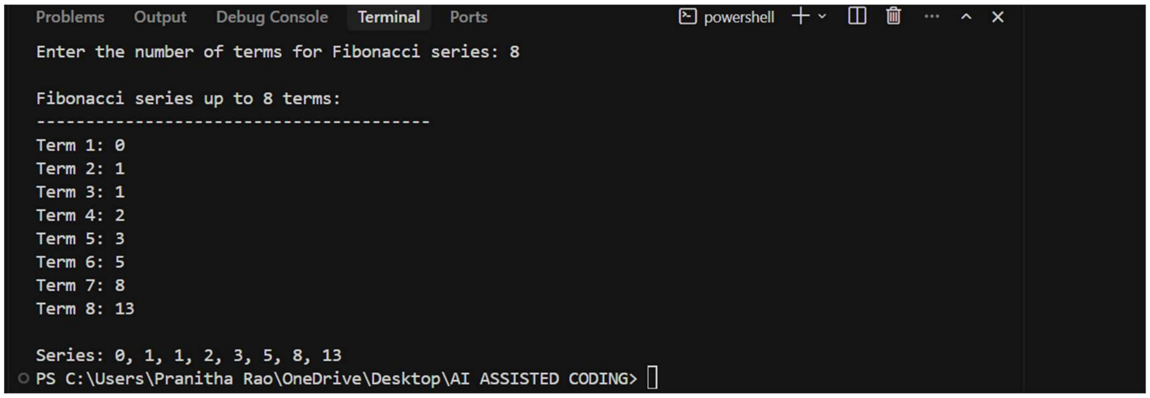


```python
    return fib_series
def print_fibonacci_series(n):
    """
    Print Fibonacci series up to n terms in a formatted way.
    """
    # Get the Fibonacci series
    series = fibonacci_series(n)
    # Check if series is valid (not empty)
    if not series:
        print("Invalid input! Please enter a positive integer.")
        return
    # Print the series in a formatted way
    print(f"\nFibonacci series up to {n} terms:")
    print("-" * 40)
    # Print each term with its position
    for i, term in enumerate[int](series, start=1):
        print(f"Term {i}: {term}")
    # Print the series as a comma-separated list
    print(f"\nSeries: {', '.join(map[str](str, series))}")
# Main execution block
```



```python
# Main execution block
if __name__ == "__main__":
    try:
        # Get user input for number of terms
        n = int(input("Enter the number of terms for Fibonacci series: "))
        # Print the Fibonacci series
        print_fibonacci_series(n)
    except ValueError:
        # Handle invalid input (non-integer values)
        print("Error: Please enter a valid integer number!")
    except Exception as e:
        # Handle any other unexpected errors
        print(f"An error occurred: {e}")
```

Output:



```
Enter the number of terms for Fibonacci series: 8

Fibonacci series up to 8 terms:
----------------------------------------
Term 1: 0
Term 2: 1
Term 3: 1
Term 4: 2
Term 5: 3
Term 6: 5
Term 7: 8
Term 8: 13

Series: 0, 1, 1, 2, 3, 5, 8, 13
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

Justifications:

In this task, the program displays the Fibonacci series with clear and meaningful comments. The Fibonacci logic is placed inside a user-defined function, which makes the program modular and properly structured. This allows the function to be reused in other programs whenever needed. The comments make the code easy to read and understand. Since the logic is clearly separated, finding and fixing errors becomes easier. This method follows good programming standards and is suitable for larger and real-world applications.

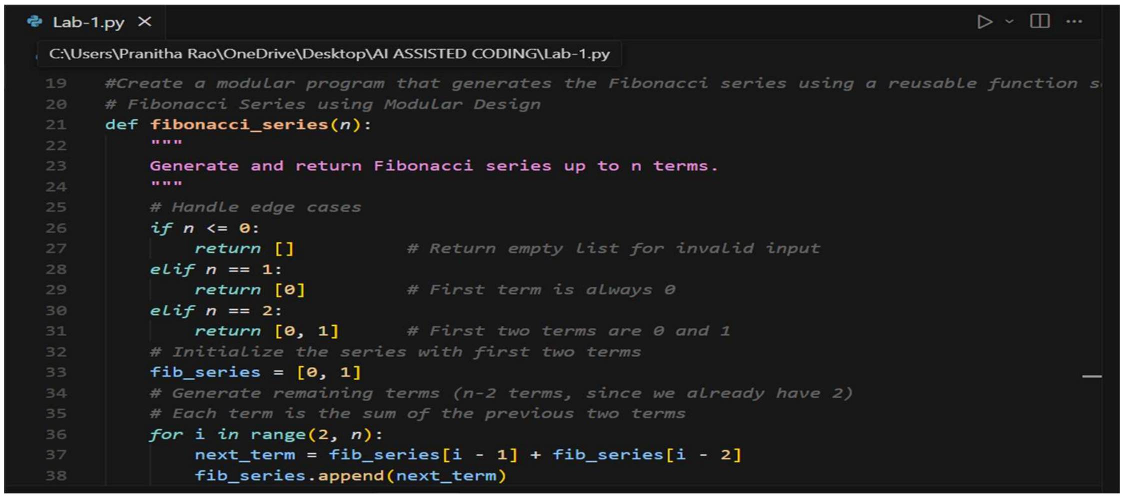
Question 4

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

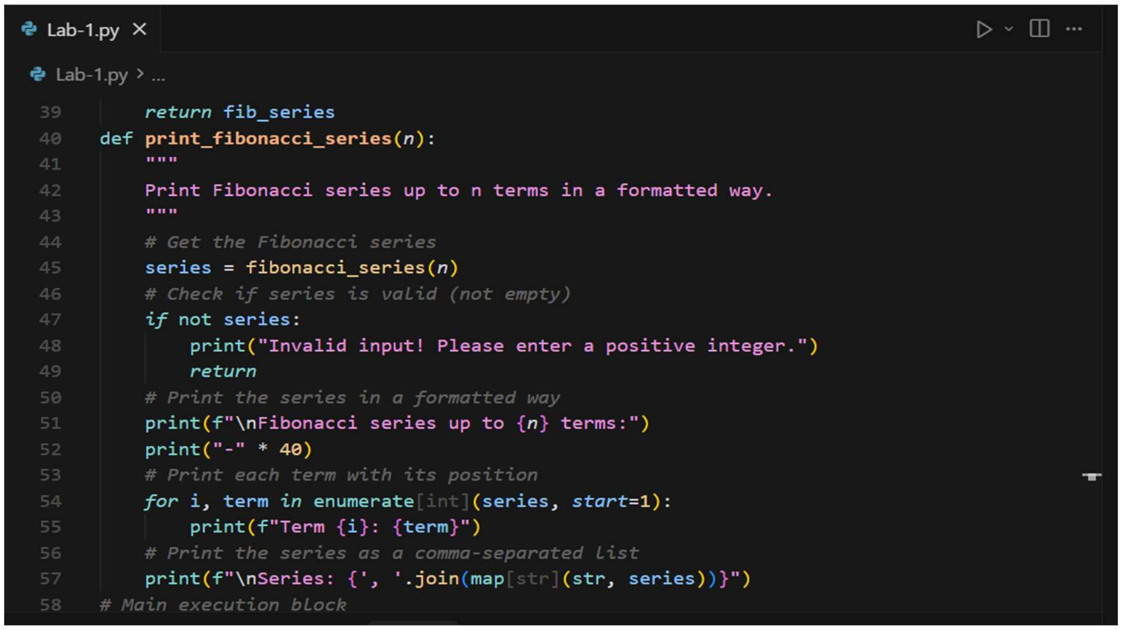
Procedural: write a python code for printing fibonacci series upto n terms without using functions

Modular: Write a python code for printing fibonacci series upto n terms and add meaningful comments Code:

Modular:

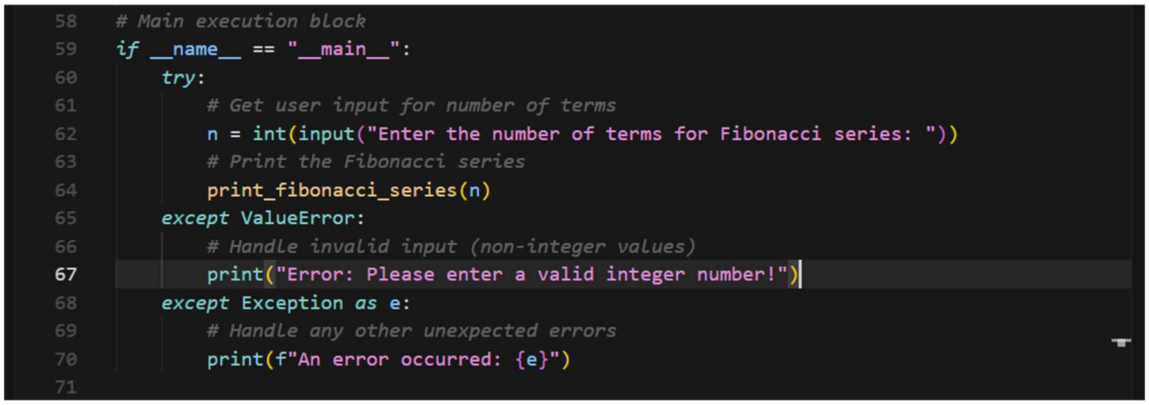

Lab-1.py
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-1.py

```python
#Create a modular program that generates the Fibonacci series using a reusable function s
# Fibonacci Series using Modular Design
def fibonacci_series(n):
    """
    Generate and return Fibonacci series up to n terms.
    """
    # Handle edge cases
    if n <= 0:
        return []           # Return empty list for invalid input
    elif n == 1:
        return [0]          # First term is always 0
    elif n == 2:
        return [0, 1]       # First two terms are 0 and 1
    # Initialize the series with first two terms
    fib_series = [0, 1]
    # Generate remaining terms (n-2 terms, since we already have 2)
    # Each term is the sum of the previous two terms
    for i in range(2, n):
        next_term = fib_series[i - 1] + fib_series[i - 2]
        fib_series.append(next_term)
```


Lab-1.py ×

Lab-1.py > ...

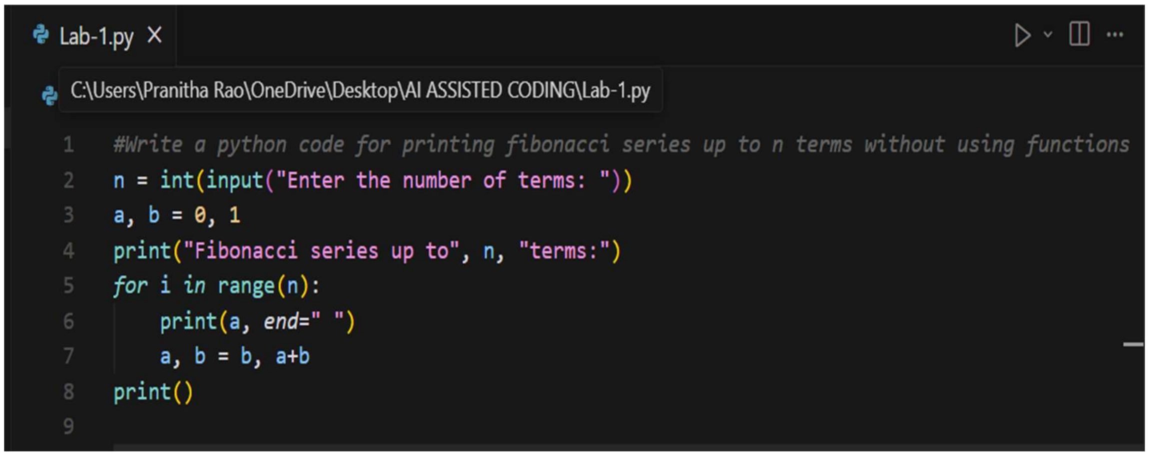
```python
    return fib_series
def print_fibonacci_series(n):
    """
    Print Fibonacci series up to n terms in a formatted way.
    """
    # Get the Fibonacci series
    series = fibonacci_series(n)
    # Check if series is valid (not empty)
    if not series:
        print("Invalid input! Please enter a positive integer.")
        return
    # Print the series in a formatted way
    print(f"\nFibonacci series up to {n} terms:")
    print("-" * 40)
    # Print each term with its position
    for i, term in enumerate[int](series, start=1):
        print(f"Term {i}: {term}")
    # Print the series as a comma-separated list
    print(f"\nSeries: {', '.join(map[str](str, series))}")
# Main execution block
```



```python
# Main execution block
if __name__ == "__main__":
    try:
        # Get user input for number of terms
        n = int(input("Enter the number of terms for Fibonacci series: "))
        # Print the Fibonacci series
        print_fibonacci_series(n)
    except ValueError:
        # Handle invalid input (non-integer values)
        print("Error: Please enter a valid integer number!")
    except Exception as e:
        # Handle any other unexpected errors
        print(f"An error occurred: {e}")

```

Procedular:


Lab-1.py X

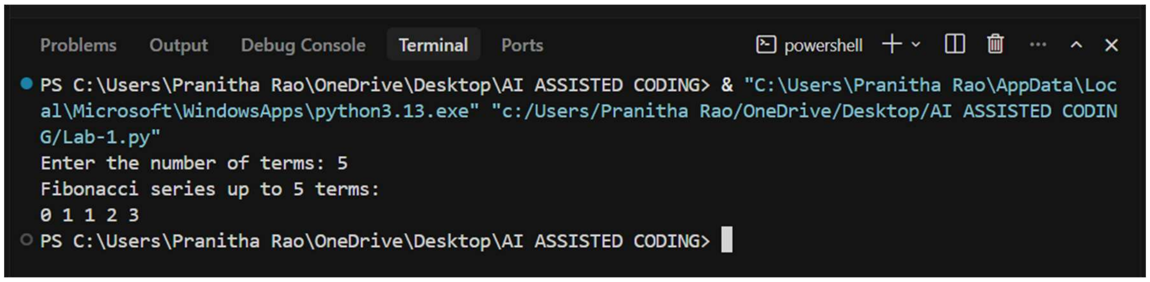
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-1.py

```python
#Write a python code for printing fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a+b
print()

```

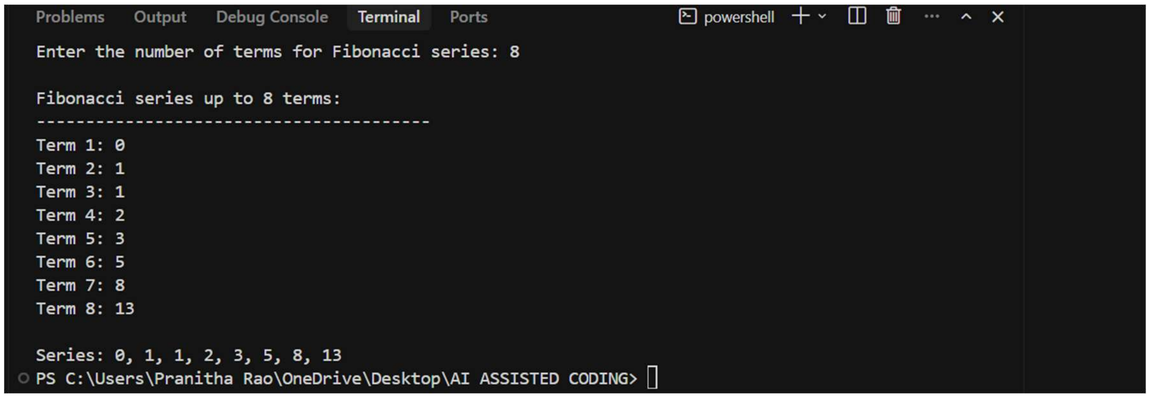
Outputs:

Procedular:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-1.py"
Enter the number of terms: 5
Fibonacci series up to 5 terms:
0 1 1 2 3
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

Modular:



```
Enter the number of terms for Fibonacci series: 8

Fibonacci series up to 8 terms:
----------------------------------------
Term 1: 0
Term 2: 1
Term 3: 1
Term 4: 2
Term 5: 3
Term 6: 5
Term 7: 8
Term 8: 13

Series: 0, 1, 1, 2, 3, 5, 8, 13
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## Justification:

In this task, Task 1 and Task 3 programs are compared. The procedural approach is easy to write but does not support reusability. Since all the logic is placed in a single block, the code becomes less structured. On the other hand, the modular approach divides the logic into functions, making the program well organized. Modular programs are easier to read and maintain. Finding and fixing errors is also simpler in modular code. Because it supports code reuse and future expansion, modular programming is more suitable for large applications.

## Question 5:

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

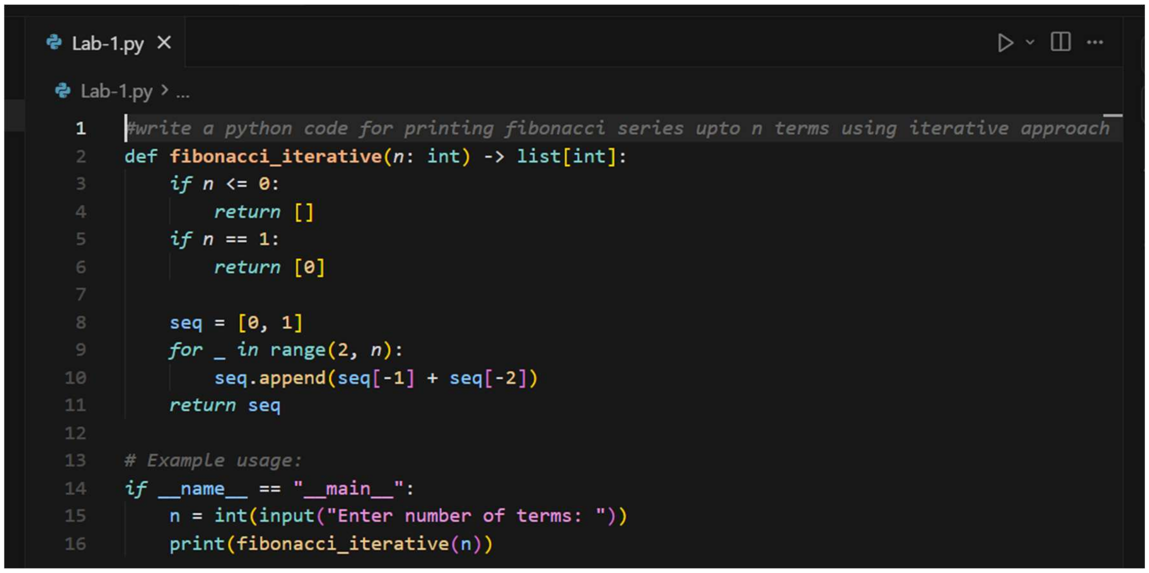
Algorithmic Approaches for Fibonacci Series) Prompts:

Iterative Approach: write a python code for printing fibonacci series upto n terms using iterative approach

Recursive Approach: write a python code for printing fibonacci series upto n terms using recursion

## Codes:

Iterative Approach:

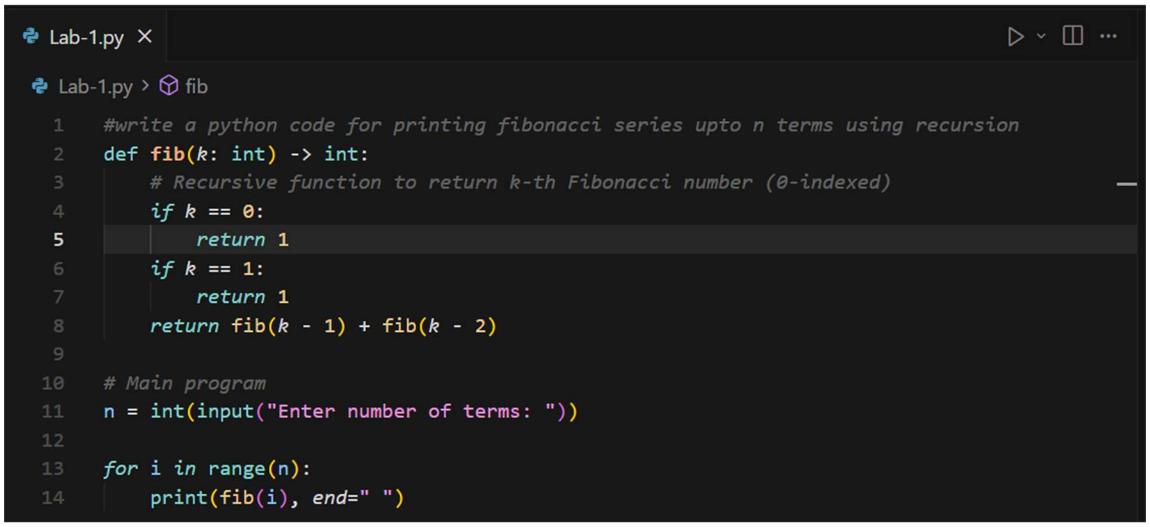


```python
#write a python code for printing fibonacci series upto n terms using iterative approach
def fibonacci_iterative(n: int) -> list[int]:
    if n <= 0:
        return []
    if n == 1:
        return [0]

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

# Example usage:
if __name__ == "__main__":
    n = int(input("Enter number of terms: "))
    print(fibonacci_iterative(n))
```

Recursive Approach:



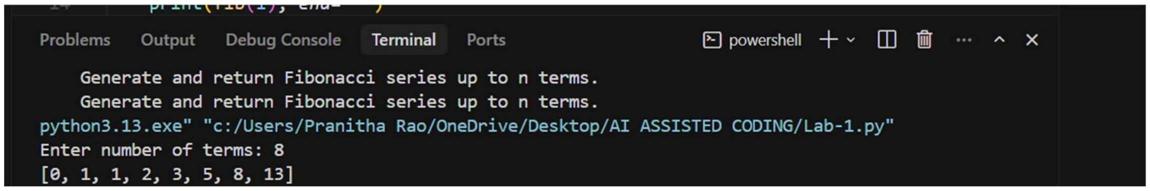
```python
#write a python code for printing fibonacci series upto n terms using recursion
def fib(k: int) -> int:
    # Recursive function to return k-th Fibonacci number (0-indexed)
    if k == 0:
        return 1
    if k == 1:
        return 1
    return fib(k - 1) + fib(k - 2)

# Main program
n = int(input("Enter number of terms: "))

for i in range(n):
    print(fib(i), end=" ")
```

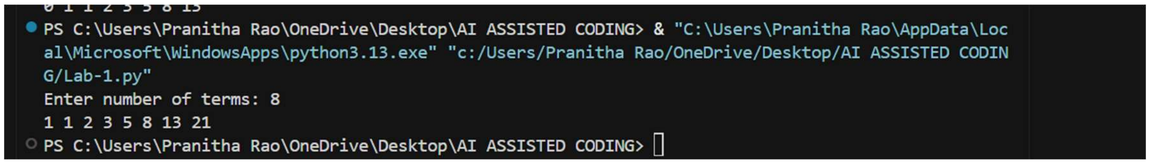
Outputs:

Iterative Approach:



```
    Generate and return Fibonacci series up to n terms.
    Generate and return Fibonacci series up to n terms.
python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-1.py"
Enter number of terms: 8
[0, 1, 1, 2, 3, 5, 8, 13]
```

Recursive Approach:



```
0 1 1 2 3 5 8 13
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Loc
al\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODIN
G/Lab-1.py"
Enter number of terms: 8
1 1 2 3 5 8 13 21
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## Justification:

The program demonstrates two methods to generate the Fibonacci series: iterative and recursive. The iterative method uses loops to calculate the Fibonacci numbers, making it faster and more memory-efficient. This approach is suitable for handling larger values of *n*. The recursive method computes Fibonacci numbers by making repeated function calls. Because it performs the same calculations multiple times, it takes more execution time. It also consumes more memory due to the function call stack. Therefore, recursion is not recommended for large input values.