AI Assisted Coding Lab 1

Shivani Pambidi

2303A52172

Batch-41

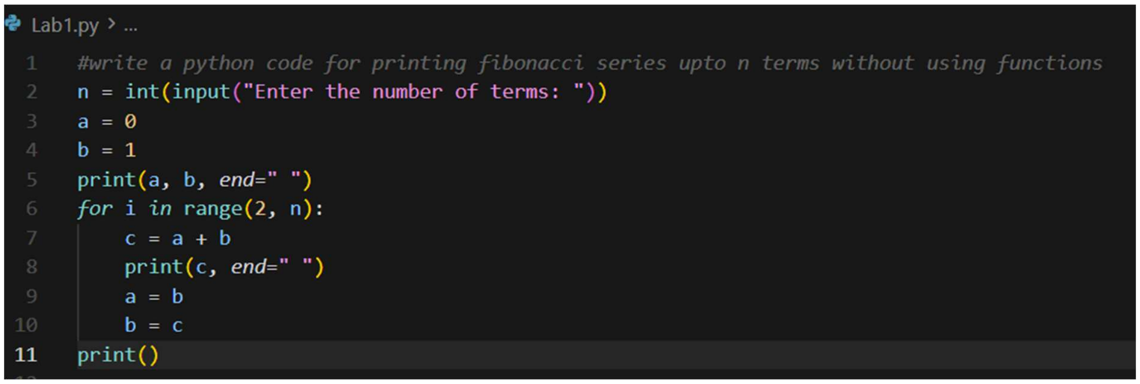
Question 1:

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

Prompt:

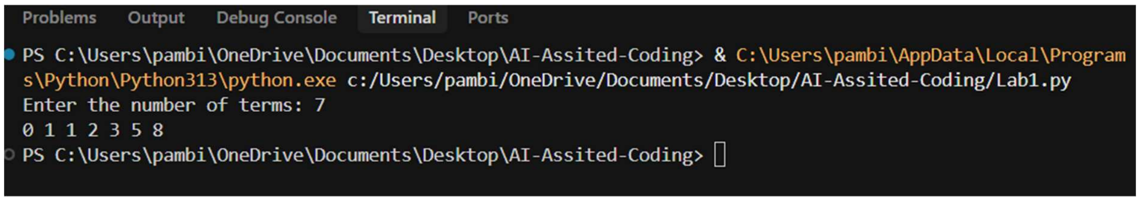
write a python code for printing fibonacci series upto n terms without using functions

Code:



```
Lab1.py > ...
#write a python code for printing fibonacci series upto n terms without using functions
n = int(input("Enter the number of terms: "))
a = 0
b = 1
print(a, b, end=" ")
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
print()
```

Output:



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Program
s\Python\Python313\python.exe c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab1.py
Enter the number of terms: 7
0 1 1 2 3 5 8
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

Justification:

This program generates the Fibonacci series up to n terms without using any functions. The entire logic is written directly in the program, so it is easy to follow. It produces the correct output for the given input. This method is suitable for small tasks and basic practice. The program takes input from the user and prints the Fibonacci numbers up to the required term.

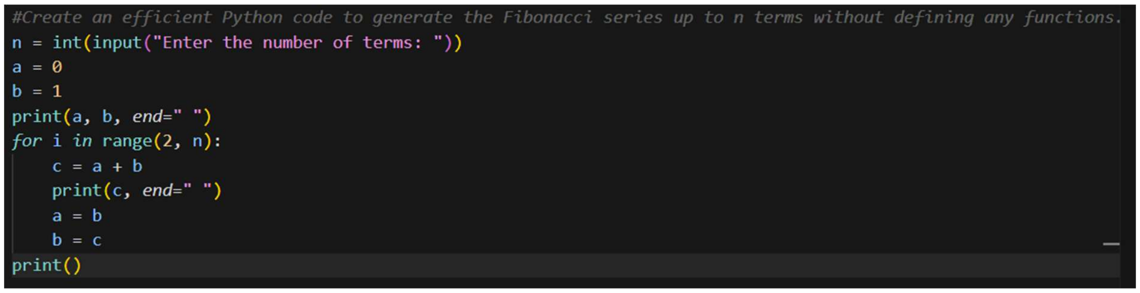
Question 2:

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

Prompt :

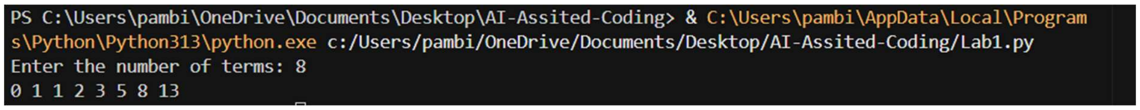
Create an efficient Python code to generate the Fibonacci series up to n terms without defining any functions.

Code:



```python
#Create an efficient Python code to generate the Fibonacci series up to n terms without defining any functions.
n = int(input("Enter the number of terms: "))
a = 0
b = 1
print(a, b, end=" ")
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
print()
```

Output:



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Program
s\Python\Python313\python.exe c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab1.py
Enter the number of terms: 8
0 1 1 2 3 5 8 13
```

Justification:

This prompt clearly asks for an efficient Python program to print the Fibonacci series up to n terms. It mentions not to use functions, which keeps the code straightforward. The focus on optimization encourages better use of loops and variables. It is easy to understand and suitable for beginners.

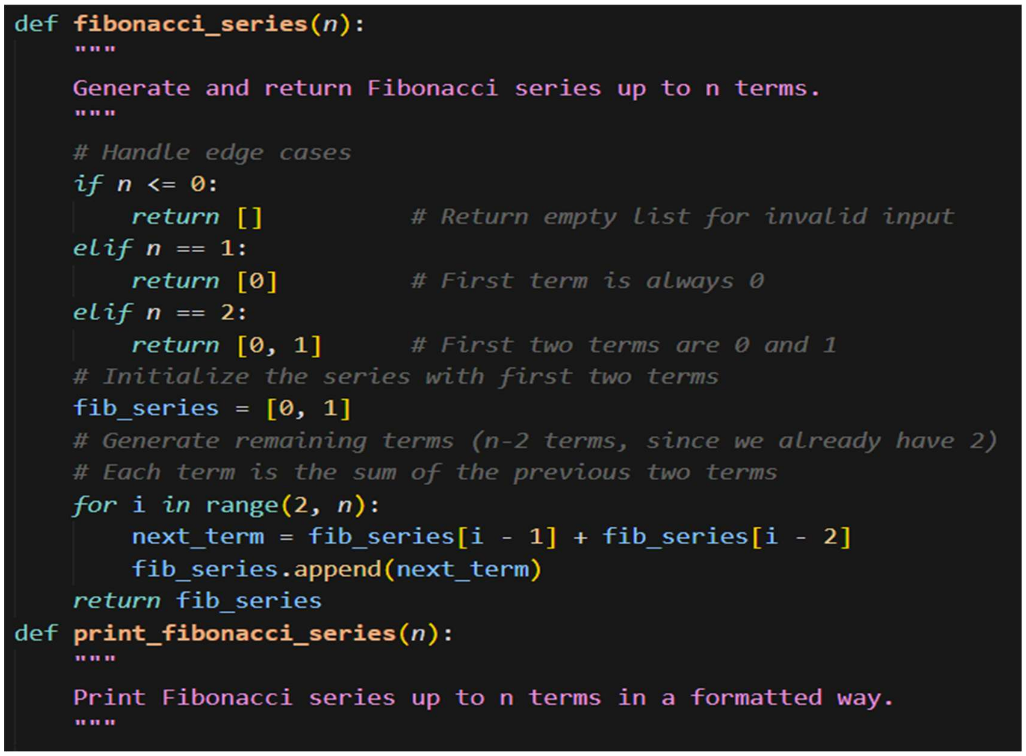
Question 3:

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

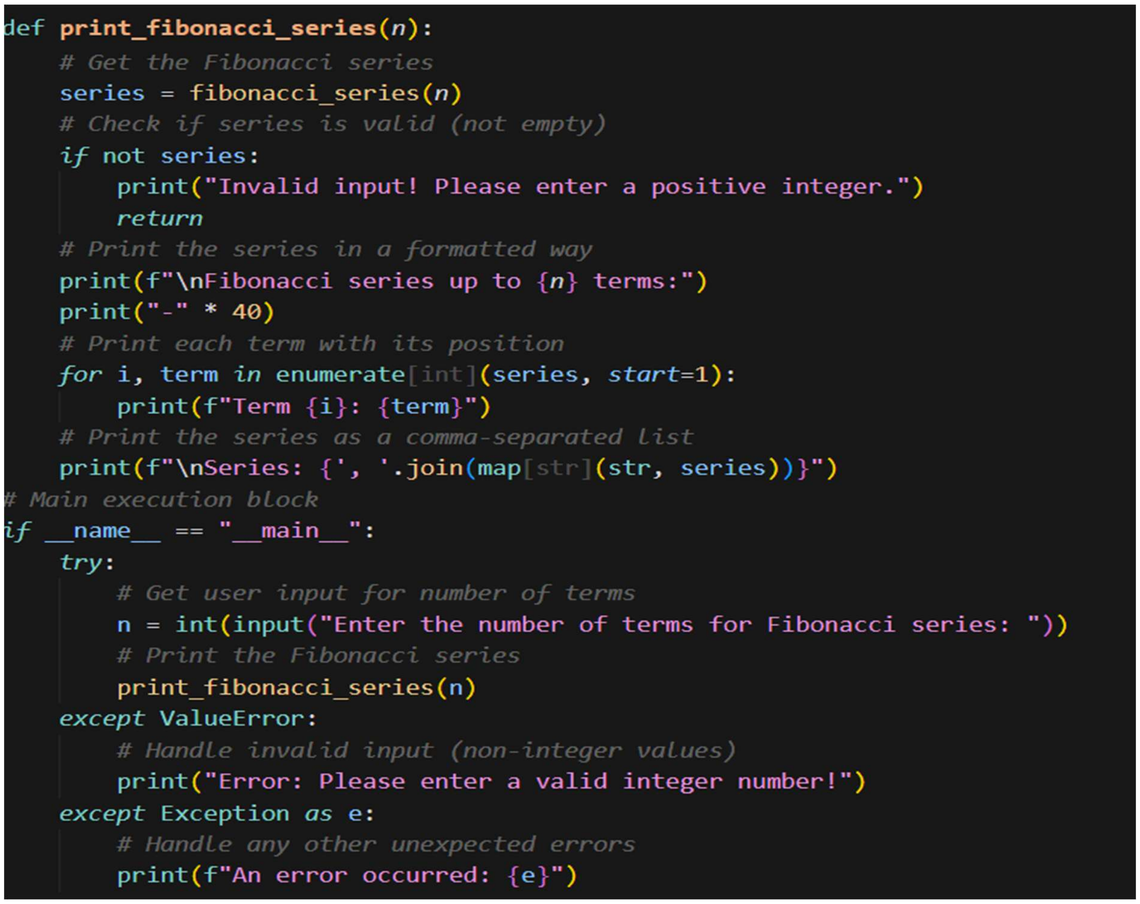
Prompt:

Write a Python program that prints the Fibonacci sequence up to a user-given limit by updating values inside a loop, and explain each step using comments.

Code:

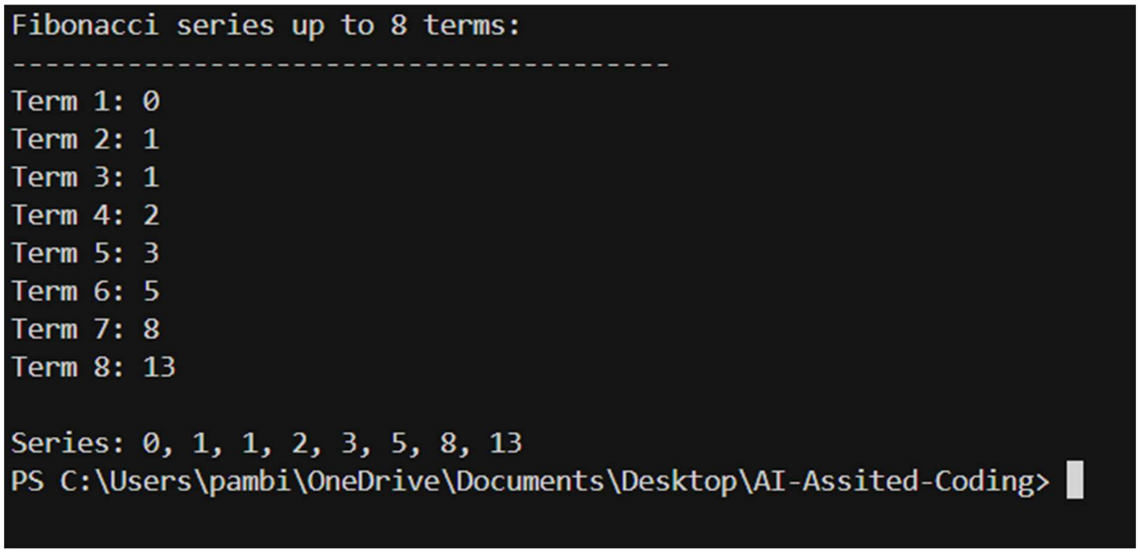


```python
def fibonacci_series(n):
    """
    Generate and return Fibonacci series up to n terms.
    """
    # Handle edge cases
    if n <= 0:
        return []           # Return empty list for invalid input
    elif n == 1:
        return [0]          # First term is always 0
    elif n == 2:
        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)
    return fib_series
def print_fibonacci_series(n):
    """
    Print Fibonacci series up to n terms in a formatted way.
    """
```



```python
def print_fibonacci_series(n):
    # 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
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:



```
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\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

Justification:

This program generates the Fibonacci series up to *n* terms using functions, which makes the code organized and easy to read. It includes input validation to handle incorrect or non-integer values, ensuring the program runs safely. The series is printed in a clear, formatted way with each term labeled, making it easy to understand.

Question 4

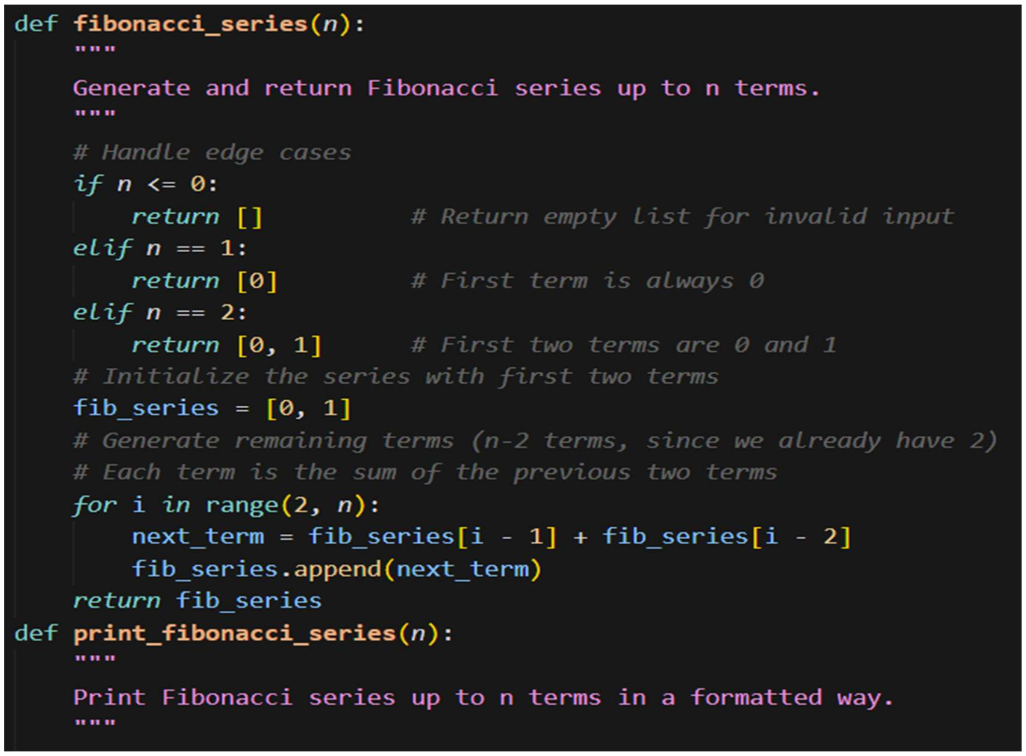
Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code
Prompt:

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

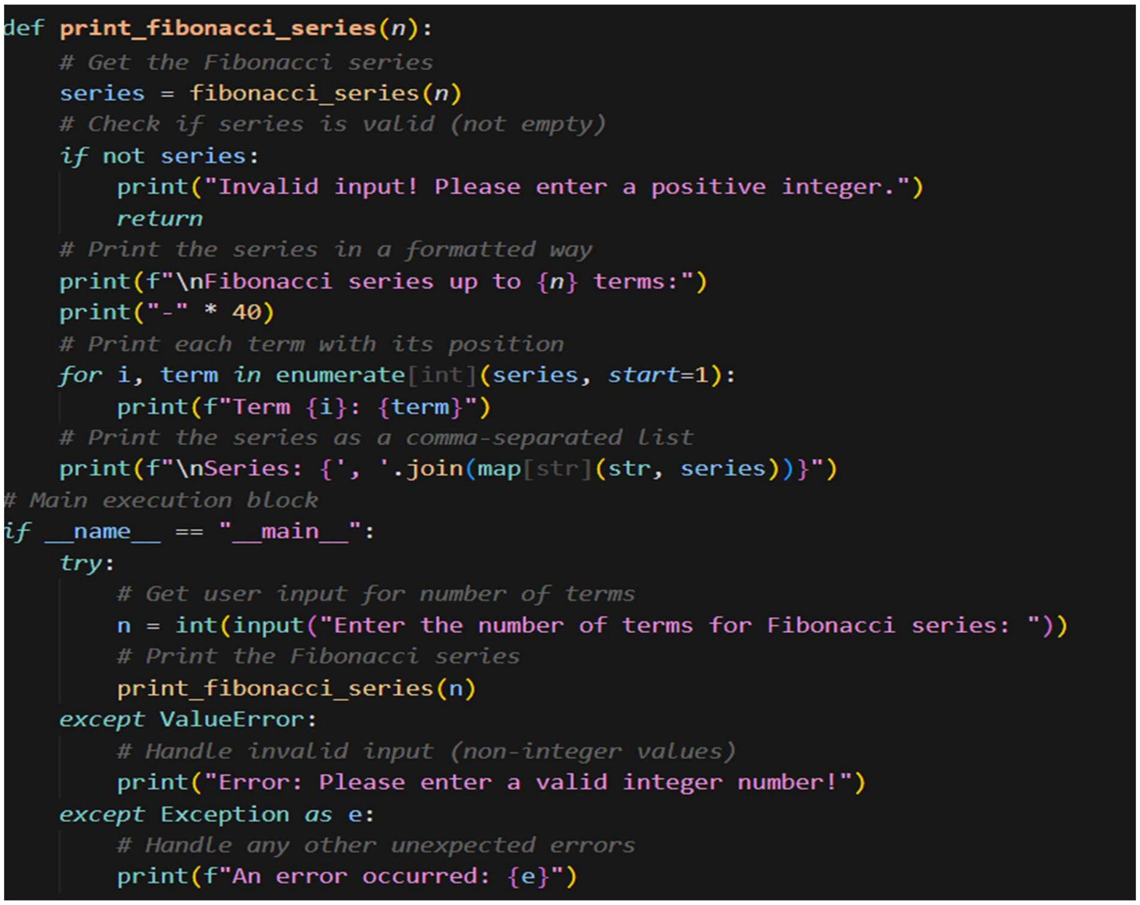
Modular: Write a Python program that prints the Fibonacci sequence up to a user-given limit by updating values inside a loop, and explain each step using comments.

Code:

Modular:

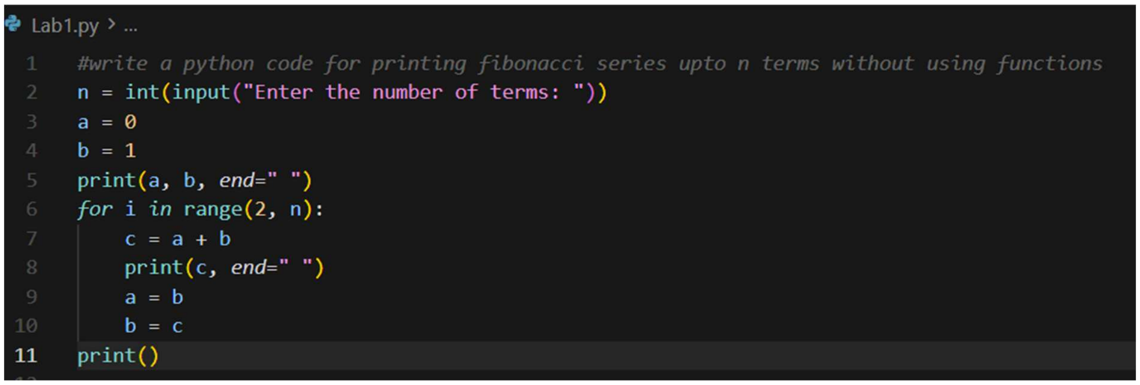


```python
def fibonacci_series(n):
    """
    Generate and return Fibonacci series up to n terms.
    """
    # Handle edge cases
    if n <= 0:
        return []           # Return empty list for invalid input
    elif n == 1:
        return [0]          # First term is always 0
    elif n == 2:
        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)
    return fib_series
def print_fibonacci_series(n):
    """
    Print Fibonacci series up to n terms in a formatted way.
    """
```



```python
def print_fibonacci_series(n):
    # 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
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:

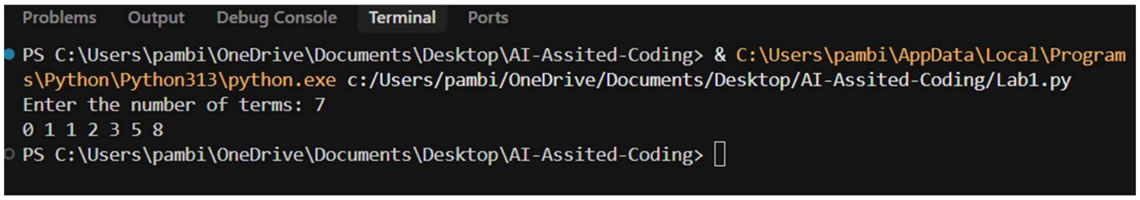


Lab1.py > ...

```python
#write a python code for printing fibonacci series upto n terms without using functions
n = int(input("Enter the number of terms: "))
a = 0
b = 1
print(a, b, end=" ")
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
print()
```

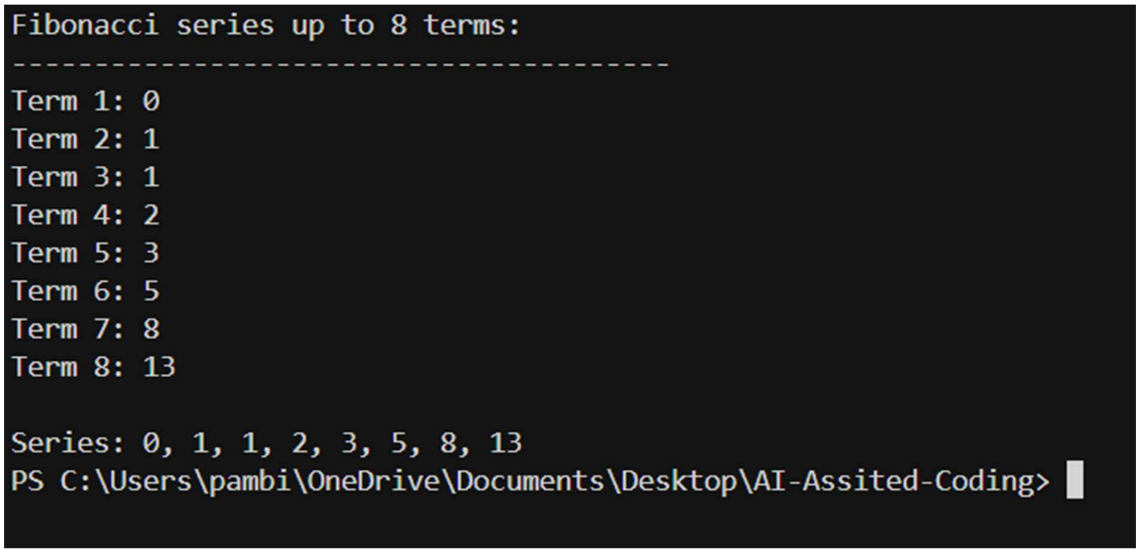
Outputs:

Procedular:



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Program
s\Python\Python313\python.exe c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab1.py
Enter the number of terms: 7
0 1 1 2 3 5 8
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

Modular:



```
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\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

Justification:

In this task, we compared the previous codes (Task 1 and Task 3). The procedural code is simple but everything is written together, so it's harder to manage and reuse. The modular code breaks the program into functions, making it easier to read, understand, and debug. Functions can be reused, and the code can be scaled for bigger programs

Question 5:

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

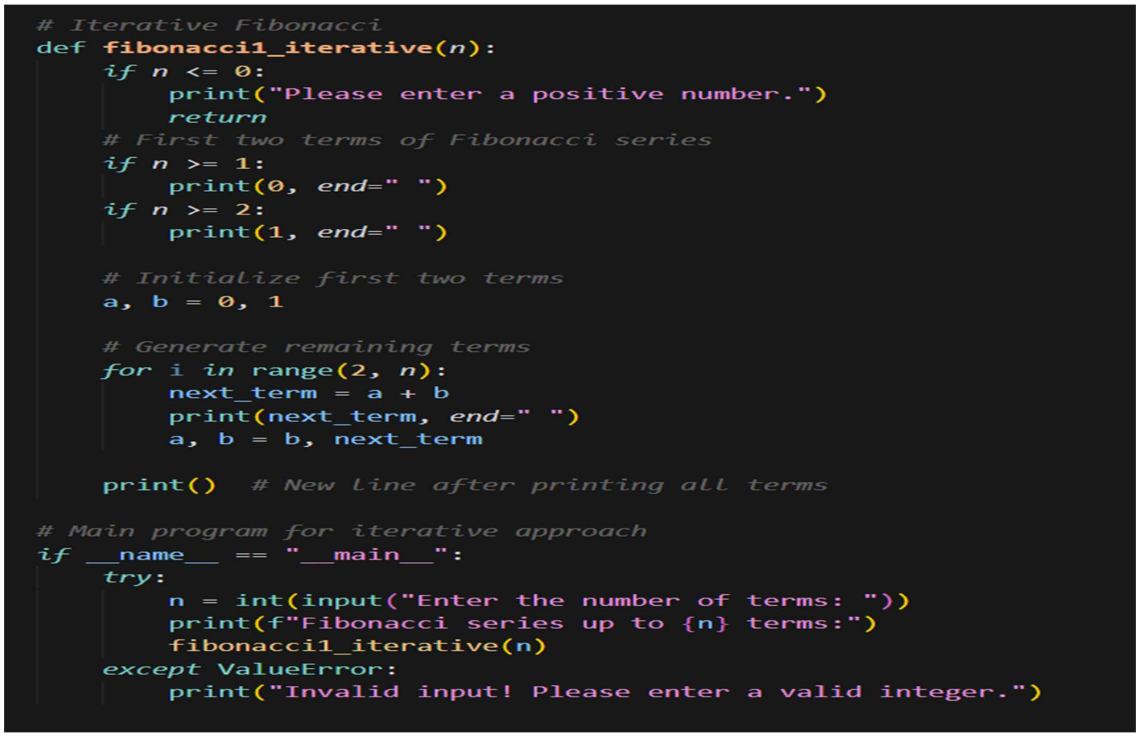
Prompts:

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

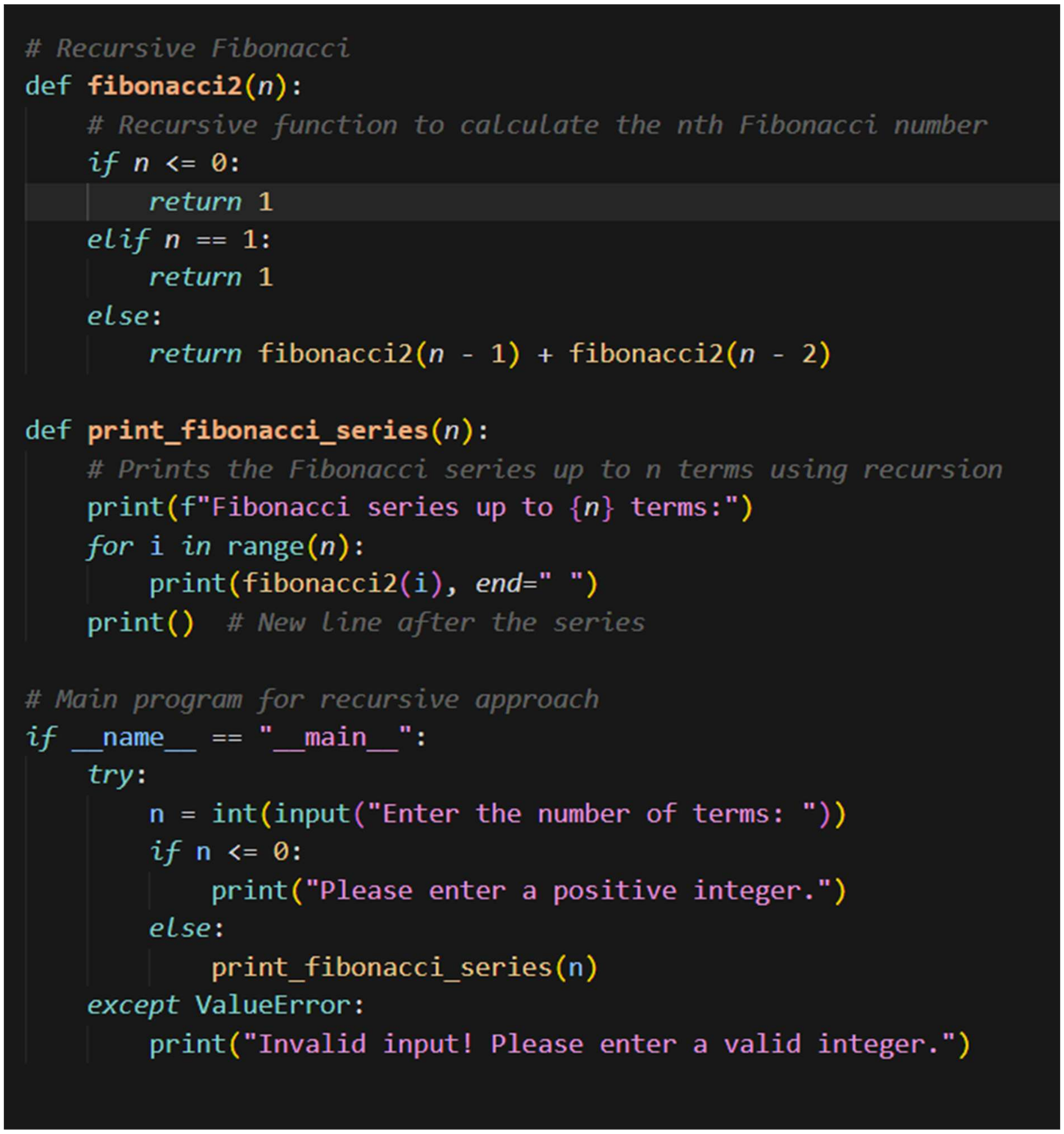
Codes:

Iterative Approach:



```python
# Iterative Fibonacci
def fibonacci1_iterative(n):
    if n <= 0:
        print("Please enter a positive number.")
        return
    # First two terms of Fibonacci series
    if n >= 1:
        print(0, end=" ")
    if n >= 2:
        print(1, end=" ")

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

    # Generate remaining terms
    for i in range(2, n):
        next_term = a + b
        print(next_term, end=" ")
        a, b = b, next_term

    print()  # New line after printing all terms

# Main program for iterative approach
if __name__ == "__main__":
    try:
        n = int(input("Enter the number of terms: "))
        print(f"Fibonacci series up to {n} terms:")
        fibonacci1_iterative(n)
    except ValueError:
        print("Invalid input! Please enter a valid integer.")
```

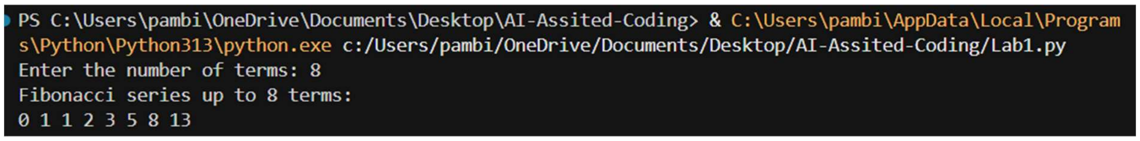
Recursive Approach:



```python
# Recursive Fibonacci
def fibonacci2(n):
    # Recursive function to calculate the nth Fibonacci number
    if n <= 0:
        return 1
    elif n == 1:
        return 1
    else:
        return fibonacci2(n - 1) + fibonacci2(n - 2)

def print_fibonacci_series(n):
    # Prints the Fibonacci series up to n terms using recursion
    print(f"Fibonacci series up to {n} terms:")
    for i in range(n):
        print(fibonacci2(i), end=" ")
    print()  # New line after the series

# Main program for recursive approach
if __name__ == "__main__":
    try:
        n = int(input("Enter the number of terms: "))
        if n <= 0:
            print("Please enter a positive integer.")
        else:
            print_fibonacci_series(n)
    except ValueError:
        print("Invalid input! Please enter a valid integer.")
```

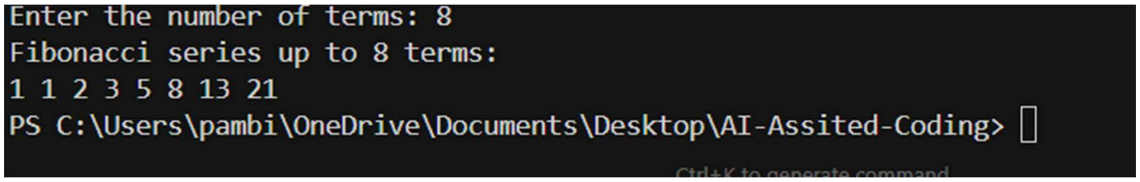
Outputs:

Iterative Approach:



```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\Local\Program
s\Python\Python313\python.exe c:/Users/pambi/OneDrive/Documents/Desktop/AI-Assited-Coding/Lab1.py
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
```

Recursive Approach:



```
Enter the number of terms: 8
Fibonacci series up to 8 terms:
1 1 2 3 5 8 13 21
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding>
```

Justification:

The program shows two ways to print Fibonacci numbers: Iterative and Recursive. The iterative method uses a loop, runs faster, and uses less memory, so it's better for large numbers. The recursive method calls the function many times, which takes more time and memory. For big inputs, recursion is not recommended.