

# AI Assisted Coding Lab 1

*Veekshitha Adharasani*

*2303A52175*

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

### Output:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python3
13\python.exe d:/veekshitha/AI_Coding/LAB-1.py
● Enter the number of terms: 8
0 1 1 2 3 5 8 13
○ PS D:\veekshitha\AI_Coding> █
```

### Justification:

This program prints the Fibonacci series upto n terms without using functions. All the logics are written in the main program, which makes it simple to understand. It works correctly for the given input. This approach is useful for small programs or quick testing. But it is a bit hard for larger numbers. The program accepts user input and prints the Fibonacci series upto the 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

## Optimized Fibonacci series in Python

5m · +24 -3 · Auto

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

∞ Agent ▾ Auto ▾



### Code:

```
LAB-1.py > ...
1
2 # Optimized Fibonacci Series up to n terms (without functions)
3 n = int(input("Enter the number of terms: "))
4
5 # First two terms of Fibonacci series
6 a, b = 0, 1
7
8 # Print first term
9 if n >= 1:
10     print(a, end=" ")
11
12 # Print second term
13 if n >= 2:
14     print(b, end=" ")
15
16 # Generate and print remaining terms
17 for i in range(2, n):
18     # Calculate next term
19     c = a + b
20     print(c, end=" ")
21     # Update values for next iteration
22     a, b = b, c
23
24 print() # New Line at the end
```

### Output:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:\veekshitha\AI_Coding\LAB-1.py
Enter the number of terms: 8
0 1 1 2 3 5 8 13
PS D:\veekshitha\AI_Coding>
```

### Justification:

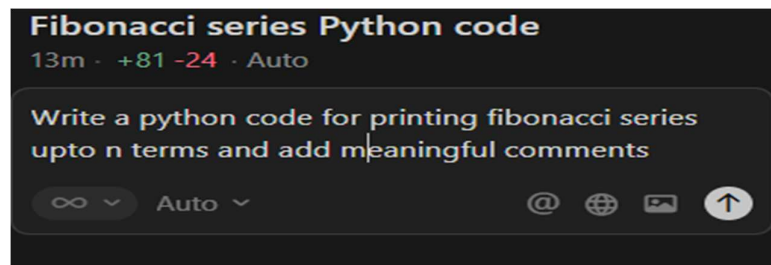
This program prints the optimized code of Fibonacci series that removes extra variables and simplifies the logic. Compared to Task 1, it removes the unwanted and extra lines. It uses fewer lines of code and is easier to read. The program runs faster because unnecessary steps are avoided. The output remains correct after optimization. This makes the code more efficient and clean.

### 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 > print_fibonacci_series
7 def fibonacci_series(n):
8     """
9     Generate and return Fibonacci series up to n terms.
10    """
11    # Handle edge cases
12    if n <= 0:
13        return [] # Return empty List for invalid input
14    elif n == 1:
15        return [0] # First term is always 0
16    elif n == 2:
17        return [0, 1] # First two terms are 0 and 1
18
19    # Initialize the series with first two terms
20    fib_series = [0, 1]
21
22    # Generate remaining terms (n-2 terms, since we already have 2)
23    # Each term is the sum of the previous two terms
24    for i in range(2, n):
25        next_term = fib_series[i - 1] + fib_series[i - 2]
26        fib_series.append(next_term)
27
28    return fib_series
29
30
31 def print_fibonacci_series(n):
32     """
33     Print Fibonacci series up to n terms in a formatted way.
34     """
35     # Get the Fibonacci series
36     series = fibonacci_series(n)
37
38     # Check if series is valid (not empty)
39     if not series:
40         print("Invalid input! Please enter a positive integer.")
41         return
```

```
LAB-1.py > print_fibonacci_series
31 def print_fibonacci_series(n):
32     """
33     Print Fibonacci series up to n terms in a formatted way.
34     """
35     # Get the Fibonacci series
36     series = fibonacci_series(n)
37
38     # Check if series is valid (not empty)
39     if not series:
40         print("Invalid input! Please enter a positive integer.")
41         return
42
43     # Print the series in a formatted way
44     print(f"\nFibonacci series up to {n} terms:")
45     print("-" * 40)
46
47     # Print each term with its position
48     for i, term in enumerate(series, start=1):
49         print(f"Term {i}: {term}")
50
51     # Print the series as a comma-separated list
52     print(f"\nSeries: {' '.join(map(str, series))}")
53
54 # Main execution block
55 if __name__ == "__main__":
56     try:
57         # Get user input for number of terms
58         n = int(input("Enter the number of terms for Fibonacci series: "))
59
60         # Print the Fibonacci series
61         print_fibonacci_series(n)
62
63     except ValueError:
64         # Handle invalid input (non-integer values)
65         print("Error: Please enter a valid integer number!")
66
67     except Exception as e:
68         # Handle any other unexpected errors
69         print(f"An error occurred: {e}")
```

Output:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:/veekshitha/AI_Coding/LAB-1.py
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 D:\veekshitha\AI_Coding>
```

## Justifications:

In this task, the program prints the Fibonacci series with meaningful comments. The Fibonacci logic is written inside a user defined function. This makes the program modular and well organized. The function can be reused in different programs. Comments help in understanding the code easily. Debugging becomes simpler due to clear separation of logic. This approach follows good programming practices. It is suitable for large 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 > print_fibonacci_series
7 def fibonacci_series(n):
8     """
9     Generate and return Fibonacci series up to n terms.
10    """
11    # Handle edge cases
12    if n <= 0:
13        return [] # Return empty List for invalid input
14    elif n == 1:
15        return [0] # First term is always 0
16    elif n == 2:
17        return [0, 1] # First two terms are 0 and 1
18
19    # Initialize the series with first two terms
20    fib_series = [0, 1]
21
22    # Generate remaining terms (n-2 terms, since we already have 2)
23    # Each term is the sum of the previous two terms
24    for i in range(2, n):
25        next_term = fib_series[i - 1] + fib_series[i - 2]
26        fib_series.append(next_term)
27
28    return fib_series
29
30
31 def print_fibonacci_series(n):
32     """
33     Print Fibonacci series up to n terms in a formatted way.
34     """
35     # Get the Fibonacci series
36     series = fibonacci_series(n)
37
38     # Check if series is valid (not empty)
39     if not series:
40         print("Invalid input! Please enter a positive integer.")
41         return
```

```
LAB-1.py > print_fibonacci_series
31 def print_fibonacci_series(n):
32     """
33     Print Fibonacci series up to n terms in a formatted way.
34     """
35     # Get the Fibonacci series
36     series = fibonacci_series(n)
37
38     # Check if series is valid (not empty)
39     if not series:
40         print("Invalid input! Please enter a positive integer.")
41         return
42
43     # Print the series in a formatted way
44     print(f"\nFibonacci series up to {n} terms:")
45     print("-" * 40)
46
47     # Print each term with its position
48     for i, term in enumerate(series, start=1):
49         print(f"Term {i}: {term}")
50
51     # Print the series as a comma-separated list
52     print(f"\nSeries: {' '.join(map(str, series))}")
53
54 # Main execution block
55 if __name__ == "__main__":
56     try:
57         # Get user input for number of terms
58         n = int(input("Enter the number of terms for Fibonacci series: "))
59
60         # Print the Fibonacci series
61         print_fibonacci_series(n)
62
63     except ValueError:
64         # Handle invalid input (non-integer values)
65         print("Error: Please enter a valid integer number!")
66     except Exception as e:
67         # Handle any other unexpected errors
68         print(f"An error occurred: {e}")
```

## Procedural:

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

## Outputs:

### Procedural:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:/veekshitha/AI_Coding/LAB-1.py
Enter the number of terms: 8
0 1 1 2 3 5 8 13
PS D:\veekshitha\AI_Coding> █
```

### Modular:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:/veekshitha/AI_Coding/LAB-1.py
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 D:\veekshitha\AI_Coding> █
```

## Justification:

In this task, the previous codes are compared(Task1 and Task3). The procedural code is simple but not reusable. All logic is written in one place, making it less organized. The modular code separates logic using functions. Modular code is easier to read and understand. Debugging is simpler in modular programs. It supports reuse and scalability. Therefore, modular code is better for large systems.

## 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

## Fibonacci series iterative Python code

Now · +38 -24 · Auto

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

## Python Fibonacci series recursion

1m · +35 -24 · Auto

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

**Codes:**

**Iterative Approach:**

```
LAB-1.py > fibonacci_iterative
1 def fibonacci_iterative(n):
2     /
3     if n <= 0:
4         print("Please enter a positive number.")
5         return
6
7     # First two terms of Fibonacci series
8     if n >= 1:
9         print(0, end=" ")
10
11     if n >= 2:
12         print(1, end=" ")
13
14     # Initialize first two terms
15     a, b = 0, 1
16
17     # Generate remaining terms
18     for i in range(2, n):
19         next_term = a + b
20         print(next_term, end=" ")
21         a, b = b, next_term
22
23     print() # New Line after printing all terms
24
25 # Main program
26 if __name__ == "__main__":
27     try:
28         n = int(input("Enter the number of terms: "))
29         print(f"Fibonacci series up to {n} terms:")
30         fibonacci_iterative(n)
31     except ValueError:
32         print("Invalid input! Please enter a valid integer.")
33
34
```

**Recursive Approach:**

```
LAB-1.py > print_fibonacci_series
1 def fibonacci(n):
2     """
3     Recursive function to calculate the nth Fibonacci number.
4     Base cases: F(0) = 0, F(1) = 1
5     Recursive case: F(n) = F(n-1) + F(n-2)
6     """
7     if n <= 0:
8         return 0
9     elif n == 1:
10        return 1
11    else:
12        return fibonacci(n - 1) + fibonacci(n - 2)
13
14 def print_fibonacci_series(n):
15     """
16     Prints the Fibonacci series up to n terms using recursion.
17     """
18    print(f"Fibonacci series up to {n} terms:")
19    for i in range(1, n + 1):
20        print(fibonacci(i), end=" ")
21    print() # New Line after the series
22
23 # Main program
24 if __name__ == "__main__":
25     try:
26         n = int(input("Enter the number of terms: "))
27         if n <= 0:
28             print("Please enter a positive integer.")
29         else:
30             print_fibonacci_series(n)
31     except ValueError:
32         print("Invalid input! Please enter a valid integer.")
33
34
```

**Outputs:**

**Iterative Approach:**

```
PS D:\veekshitha\AI_Coding> & C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:/veekshitha/AI_Coding/LAB-1.py
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS D:\veekshitha\AI_Coding>
```

### Recursive Approach:

```
Python3 (python.exe d:\veekshitha\AI_coding\Lab 1.py)
● Enter the number of terms: 8
  Fibonacci series up to 8 terms:
  1 1 2 3 5 8 13 21
○ PS D:\veekshitha\AI_Coding>
```

### Justification:

The program prints the code by two approaches, Iterative and Recursive. The iterative approach uses a loop to generate Fibonacci numbers. It runs faster and uses less memory. This approach works well for large values of n. The recursive approach uses function calls repeatedly. It takes more time due to repeated calculations. It also uses more memory because of the call stack. Recursion should be avoided for large inputs.