

## AI Assistant Coding Assignment-1

---

Name of Student : Nikhil N  
Enrollment No. : 2303A52477  
Batch : 38

---

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

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

#### Prompt:

write a code to print fibonacci series upto  $n$  without using a function

#### Code:

```
# write a code to print fibonacci series upto n without using a function
n = int(input("Enter the number of terms: "))
a, b = 0, 1
count = 0

if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print(f"Fibonacci series up to {n} term:")
    print(a)
else:
    print(f"Fibonacci series up to {n} terms:")
    while count < n:
        print(a, end=' ')
        c = a + b
        a = b
        b = c
        count += 1
```

#### Output:

```
PS C:\Users\nikhil\Projects\AI Assistant Coding> & C:/Users/nikhil/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/nikhil/Projects/AI Assistant Coding/fibo.py"
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS C:\Users\nikhil\Projects\AI Assistant Coding> █
```

#### Justification:

Generated code using inline prompt of GitHub Copilot in VS Code for the logic of Fibonacci series up to n terms without using a function. It takes input from the user to iterate till n using for loop and print them in a sequence.

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

- **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”**

### Prompt:

Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.

### Code:

```
# Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.

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

if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print(f"Fibonacci series up to {n} term:")
    print(0)
else:
    print(f"Fibonacci series up to {n} terms:")
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
```

### Output:

```
PS C:\Users\nikhi\Projects\AI Assistant Coding> & C:/Users/nikhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/nikhi/Projects/AI Assistant Coding/fibo.py"
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS C:\Users\nikhi\Projects\AI Assistant Coding> |
```

### Justification:

Justification for optimizations: Used tuple unpacking ( $a, b = 0, 1$  and  $a, b = b, a + b$ ) for cleaner variable swapping, used underscore (`_`) in the loop since the counter variable isn't needed, combined variable declarations on a single line where appropriate, used f-strings for better string formatting, and the code is already quite optimized with  $O(n)$  time complexity and  $O(1)$  space complexity.

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

#### **Prompt:**

Write a code for printing the Fibonacci series up to  $n$  terms using a function and include meaningful comments.

#### **Code:**

```
# Write a code for printing the Fibonacci series up to n terms using a function and include meaningful comments.
def print_fibonacci(n):
    """
    Prints the Fibonacci series up to n terms.

    Args:
        n (int): Number of terms to print in the Fibonacci series

    The Fibonacci series starts with 0 and 1, and each subsequent number
    is the sum of the previous two numbers.
    """
    if n <= 0:
        print("Please enter a positive integer")
        return
    elif n == 1:
        print(f"Fibonacci series up to {n} term:")
        print(0)
        return

    print(f"Fibonacci series up to {n} terms:")
    # Initialize the first two terms of the series
    a, b = 0, 1

    # Generate and print each term
    for _ in range(n):
        print(a, end=' ')
        # Update values: a becomes b, b becomes sum of previous a and b
        a, b = b, a + b
    print() # Print newline at the end

# Get input from user
n = int(input("Enter the number of terms: "))
print_fibonacci(n)
```

## Output:

```
PS C:\Users\nikhi\Projects\AI Assistant Coding> & C:/Users/nikhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/nikhi/Projects/AI Assistant Coding/fibo.py"
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS C:\Users\nikhi\Projects\AI Assistant Coding> 
```

## Justification:

In the above code Fibonacci Series logic is written in a Modular design with useful comments that explain what that line does. The “ fibonacci\_series ” function can be called by the user as many times as needed.

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

### 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

## Prompt:

Non-Modular:

Write a code for printing a Fibonacci series up to n terms without using a function.

Modular:

Write a code for printing the Fibonacci series up to n terms using a function and include meaningful comments.

## Code:

Procedural:

```
# Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.

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

if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print(f"Fibonacci series up to {n} term:")
    print(0)
else:
    print(f"Fibonacci series up to {n} terms:")
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
```

Modular:

```
# Write a code for printing the Fibonacci series up to n terms using a function and include meaningful comments.
def print_fibonacci(n):
    """
    Prints the Fibonacci series up to n terms.

    Args:
        n (int): Number of terms to print in the Fibonacci series

    The Fibonacci series starts with 0 and 1, and each subsequent number
    is the sum of the previous two numbers.
    """
    if n <= 0:
        print("Please enter a positive integer")
        return
    elif n == 1:
        print(f"Fibonacci series up to {n} term:")
        print(0)
        return

    print(f"Fibonacci series up to {n} terms:")
    # Initialize the first two terms of the series
    a, b = 0, 1

    # Generate and print each term
    for _ in range(n):
        print(a, end=' ')
        # Update values: a becomes b, b becomes sum of previous a and b
        a, b = b, a + b
    print() # Print newline at the end

# Get input from user
n = int(input("Enter the number of terms: "))
print_fibonacci(n)
```

Output:

Procedural:

```
PS C:\Users\nikhi\Projects\AI Assistant Coding> C:\Users\nikhi\AppData\Local\Programs\Python\Python314\python.exe "c:/Users/nikhi/Projects/AI Assistant Coding/fibo.py"
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS C:\Users\nikhi\Projects\AI Assistant Coding> █
```

Modular:

```
PS C:\Users\nikhi\Projects\AI Assistant Coding> & C:/Users/nikhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/nikhi/Projects/AI Assistant Coding/fibo.py"
Enter the number of terms: 8
Fibonacci series up to 8 terms:
0 1 1 2 3 5 8 13
PS C:\Users\nikhi\Projects\AI Assistant Coding> 
```

## Justification:

Procedural code is simple but becomes harder to maintain as system size grows. Modular code improves clarity by isolating logic, making intent easier to understand. Modular code is reusable, which can be later integrated into pipelines and projects. Also, debugging is easier in modular code.

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

### Prompt GitHub Copilot to generate:

- An iterative Fibonacci implementation
- A recursive Fibonacci implementation

### Prompt:

Write a code for printing a Fibonacci series up to n terms without using a function.

Write a code for printing the Fibonacci series up to n terms using recursion.

### Code:

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

# Write a code for printing the Fibonacci series up to n terms using recursion.

def fibonacci(n, a=0, b=1, count=0):
    if count < n:
        print(a, end=' ')
        fibonacci(n, b, a + b, count + 1)

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

### Output:

```
PS C:\Users\nikhi\Projects\AI Assistant Coding> & C:/Users/nikhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/nikhi/Projects/AI Assistant Coding/fibo.py"
Enter number of terms: 8
0 1 1 2 3 5 8 13
Enter number of terms: 8
0 1 1 2 3 5 8 13
PS C:\Users\nikhi\Projects\AI Assistant Coding> █
```

### Justification:

Since the iterative approach is fast and uses a fixed amount of memory, it works well for large  $n$  value. The recursive method is mathematically neat, but it takes exponential time unless you use memorization. Recursion should be avoided for large inputs due to the risk of stack overflow and slow performance. Iteration is the practical choice for scalable systems and high-performance needs.