AI Assistant Coding
Assignment-1

A.HARSHITH
2303A52422
BATCH-37

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:

```
# Fibonacci series up to n without using functions
n = int(input("Enter the number of terms: "))

a, b = 0, 1
print("Fibonacci series:")

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

Output:

```
PS C:\Users\vidya\OneDrive\Desktop\ai coding> python a1.py
Enter the number of terms: 6
Fibonacci series:
0 1 1 2 3 5
```

Explanation:

This program prints the Fibonacci series.
The user enters a number n. The program starts with 0 and 1. In each loop, it prints the current number and adds the previous two numbers to get the next one. This repeats n times and prints the Fibonacci series.

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:

```
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for _ in range(n): a, b = b, print(a, end=" ") or a + b
print()
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        powershell

PS C:\Users\vidya\OneDrive\Desktop\ai coding> python opa1.py
Enter the number of terms: 5
0 1 1 2 3
```

Explanation:

This program prints the Fibonacci series up to a given number of terms.
The user enters a number n. If the number is not valid, the program shows an error message. Otherwise, it starts with 0 and 1 and prints the Fibonacci numbers using a loop. In each step, the next number is found by adding the previous two numbers. The series is printed until n terms are displayed.

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:

```
# Function to generate and print Fibonacci series up to n terms
def fibonacci_series(n):
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    # Validate input
    if n <= 0:
        print("Please enter a positive number")
        return

    print("Fibonacci series up to", n, "terms:")

    # Loop through n iterations to generate Fibonacci numbers
    for _ in range(n):
        # Print current Fibonacci number
        print(a, end=" ")
        # Update a and b to next consecutive Fibonacci numbers
        a, b = b, a + b

    # Print newline for clean output
    print()
# Main program
if __name__ == "__main__":
    # Get input from user
    n = int(input("Enter the number of terms: "))

    # Call the function to print Fibonacci series
    fibonacci_series(n)
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\Users\vidya\OneDrive\Desktop\ai coding> python a13.py
Enter the number of terms: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\vidya\OneDrive\Desktop\ai coding>
```

Explanation:

This program prints the Fibonacci series using a function.

The user enters a number n. The function print_fibonacci checks whether the number is valid. If it is valid, the program starts with 0 and 1 and prints Fibonacci numbers by adding the previous two numbers each time. The series is printed up to n terms

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:

```
# Non-modular approach: Fibonacci series without using a function
n = int(input("Enter the number of terms: "))

# Validate input
if n <= 0:
    print("Please enter a positive number")
else:
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    print("Fibonacci series up to", n, "terms:")

    # Generate and print Fibonacci numbers
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

    print()
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\Users\vidya\OneDrive\Desktop\ai coding> python a13.py
Enter the number of terms: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\vidya\OneDrive\Desktop\ai coding>
```

Modular:

```python
# Function to generate and print Fibonacci series up to n terms
def fibonacci_series(n):
    """
    Generates and prints the Fibonacci series up to n terms.

    Args:
        n (int): The number of Fibonacci terms to generate and print
    """
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    # Validate input
    if n <= 0:
        print("Please enter a positive number")
        return

    print("Fibonacci series up to", n, "terms:")

    # Loop through n iterations to generate Fibonacci numbers
    for _ in range(n):
        # Print current Fibonacci number
        print(a, end=" ")
        # Update a and b to next consecutive Fibonacci numbers
        a, b = b, a + b

    # Print newline for clean output
    print()


# Main program
if __name__ == "__main__":
    # Get input from user
    n = int(input("Enter the number of terms: "))

    # Call the function to print Fibonacci series
    fibonacci_series(n)
```

Output:

Modular:

```
PS C:\Users\vidya\OneDrive\Desktop\ai coding> python a1op4.py
Enter the number of terms: 6
Fibonacci series up to 6 terms:
0 1 1 2 3 5
```

Explanation:
In the **procedural approach**, the Fibonacci series code is written directly in the main program without using any function. The steps run one after another, and this method is simple but not reusable.

In the **modular approach**, the Fibonacci logic is written inside a function. The main program only calls the function. This makes the code more organized, easy to understand, and reusable.

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 the 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=" ")
        # Recursive call with updated values
        fibonacci(n, b, a + b, count + 1)

# Main program
n = int(input("Enter the number of terms: "))
fibonacci(n)
print()
```

Output:

```
PS C:\Users\vidya\OneDrive\Desktop\ai coding> python ai.py
Enter the number of terms: 6
0 1 1 2 3 5
Enter the number of terms: 6
0 1 1 2 3 5
```

Explanation:
This program prints the Fibonacci series in two ways .
In the first part, the Fibonacci series is printed **without using a function**. The user enters the number of terms, and the program uses a loop to print the series by adding the previous two numbers each time.
In the second part, the Fibonacci series is printed **using recursion**. A function is defined that prints one Fibonacci number and then calls itself to print the next one. This process continues until the required number of terms is printed. Both methods produce the same Fibonacci series but use different approaches.