SR UNIVERSITY

AI Assistant Coding
Assignment-1

Name of Student : K.Sai Kumar Reddy
Batch : 42
Enrollment No. : 2303A52223

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

Code:

Assign 1.py

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

Output:

Justification:
The code was generated using an inline prompt in GitHub Copilot within VS Code to implement the logic for generating the Fibonacci series up to *n* terms without using a function. It accepts user input for *n*, iterates up to that value using a for loop, and prints the Fibonacci sequence in order.

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 in Fibonacci series: "))
print("Fibonacci series up to", n, "terms:")
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b

```

Output:

Justification:

The program accurately produces the Fibonacci sequence using an iterative method beginning with 0 and 1. It generates exactly *n* terms based on the given input (9), without any missing or additional values. The output is neatly formatted with space-separated numbers, making it easy to verify results and debug if needed.

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:

```python
# Write a code for printing the fibonacci series up to n terms using a function and include meaningful comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """

    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    print("Fibonacci series up to", n, "terms:")
    for _ in range(n):
        print(a, end=' ')  # Print the current term
        a, b = b, a + b  # Update to the next two terms

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
fibonacci_series(num_terms)  # Call the function to print the series
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI
Assistant Coding/Assign 1.py"
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 terms:
0 1 1 2 3 5
PS C:\Users\Abhi\Documents\AI Assistant Coding>
```

Justification:

In the above code, the Fibonacci series logic is implemented using a modular design, along with meaningful comments that explain the purpose of each line. The Fibonacci series function can be invoked by the user as many times as required.

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:

Assign 1.py

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

Modular:

```python
# Write a code for printing the fibonacci series up to n terms using a function and include meaningful comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """

    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    print("Fibonacci series up to", n, "terms:")
    for _ in range(n):
        print(a, end=' ')  # Print the current term
        a, b = b, a + b  # Update to the next two terms

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
fibonacci_series(num_terms)  # Call the function to print the series
```

Output:

Procedural:

Modular:

Justification:

Procedural code is easy to write initially, but it becomes difficult to maintain as the size of the system increases. Modular code enhances readability by separating logic into independent units, making the program's intent clearer. It also promotes reusability, allowing modules to be integrated into larger projects and pipelines. Additionally, modular code simplifies debugging and maintenance.

SR UNIVERSITY

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.

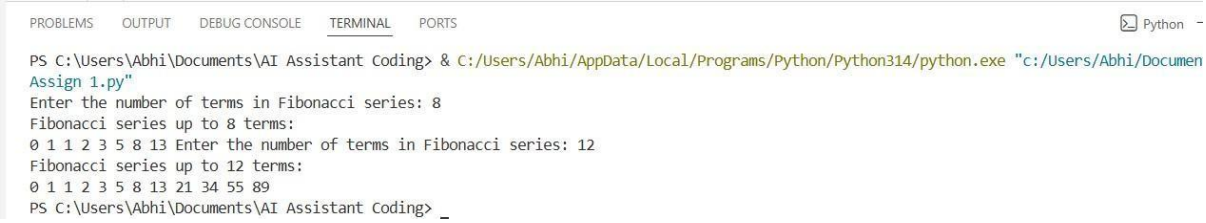
Write a code for printing the Fibonacci series up to n terms using recursion.

Code:

Assign 1.py

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

# Write a code for printing the fibonacci series up to n terms using recursion
def fibonacci_recursive(n):
    """
    This function returns the nth term of the Fibonacci series using recursion.

    Parameters:
    n (int): The position of the term in the Fibonacci series.

    Returns:
    int: The nth term of the Fibonacci series.
    """
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Get the number of terms from the user
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, "terms:")
for i in range(num_terms):
    print(fibonacci_recursive(i), end=' ')

```

Output:



Justification:

Since the iterative approach is efficient and uses a constant amount of memory, it performs well even for large values of *n*. The recursive method is mathematically elegant, but it requires exponential time unless memorization is applied. For large inputs, recursion can lead to stack overflow and poor performance. Therefore, iteration is the more practical choice for scalable systems and high-performance applications