ASSIGNMENT-1

Roll no:2303A52074
Batch-37

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

Expected Output:
Input:Enter number of terms: 10
Output:Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34

PROMPT:Generate a Python program to print Fibonacci series up to n terms without using functions.

CODE:

```
# Python program to print Fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms: "))
a, b = 0, 1
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
```

OUTPUT:

```
Enter the number of terms: 5
0 1 1 2 3
```

Explanation
The program will require the user to input the number of terms to be used, and Fibonacci will begin with 0 and 1.It then takes a loop to compute each subsequent number and sums the preceding two and displays them sequentially.All the writing is done in the main code and not through any functions and it is simple and easy to comprehend.

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

Expected Output:
Enter number of terms: 10
0 1 1 2 3 5 8 13 21 34

PROMPT:Simplify this Fibonacci program and make it more readable.

Code:

```
# Python program to print Fibonacci series up to n terms
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()
```

Output:

```
Enter the number of terms: 5
0 1 1 2 3
```

Explanation:
The initial code included additional variables and distinct conditions, and they made it longer and a little more difficult to read.The optimized code eliminates the luxury variable and employs one loop with the assignment of the tuples to produce the sequence.This allows the code to be shorter, cleaner and simpler to understand and maintain by other developers.

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

Expected Output:
Enter number of terms: 10
0 1 1 2 3 5 8 13 21 34

PROMPT : Python program to print Fibonacci series up to n terms

Code

```

# Python program to print Fibonacci series up to n terms
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()  # for a new line after the series

```

Output

```
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
```

Explanation:
The fibonacci logic is embedded within a function, which is defined by the user, to allow him to reuse it in various applications.

The function creates the sequence to the n terms with the help of a loop and gives back or prints the values.The code becomes more readable and serviceable with meaningful comments that are added with the help of AI.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Expected output:
A clear comparison between **procedural** and **modular** Fibonacci implementations, showing their differences in clarity, reusability, debugging, and scalability.

Prompt: Create a comparison between prodecural and modular fibonacci code

Code:

```
#Create a comparison between procedural and modular Fibonacci code.
# Procedural approach to print Fibonacci series up to n terms
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()
```

Output:

```
ktop\ai 1.py'
Enter the number of terms: 4
0 1 1 2
```

Explanation
Procedural Fibonacci code is straightforward and combines all the logic at a single location, which is difficult to reuse and maintain.The modular version divides the logic into a function which enhances its readability and makes it easier to debug.In general, the role-based model is more applicable in bigger applications and long-term development.

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

Expected Output
Enter number of terms: 10
Iterative Fibonacci:0 1 1 2 3 5 8 13 21 34
Recursive Fibonacci:0 1 1 2 3 5 8 13 21 34

Prompt: Write iterative and recursive Fibonacci implementations in Python and compare them.

Code

```python
#Write iterative and recursive Fibonacci implementations in Python and compare them.
# Iterative approach to print Fibonacci series up to n terms
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()  # for a new line after the series
# Recursive approach to print Fibonacci series up to n terms
def fibonacci_recursive(n, a=0, b=1):
    if n <= 0:
        return
    print(a, end=" ")
    fibonacci_recursive(n - 1, b, a + b)
n = int(input("Enter the number of terms: "))
fibonacci_recursive(n)
print()  # for a new line after the series
```

Output:

```
ktop\ai 1.py'
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
```

Explanation:

The iterative algorithm applies a loop to get the Fibonacci numbers individually and hence, it is quicker and more memory-efficient.The recursive technique recurs many times and as such, it reworks itself resulting in a slow program.Due to this, recursion cannot be used in large values of n and instead, it is better to use iteration.