

Name:m.sivateja

Hall Ticket: 2303A52145

Batch: 41

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

Prompt:

Write a simple python code for printing fibonacci series up to n terms without using functions.

Code:

```
1 # Write a simple python code for printing fibonacci series up to n terms without using functions.
2 n = int(input("Enter the number of terms in Fibonacci series: "))
3 a, b = 0, 1
4 print("Fibonacci series up to", n, "terms:")
5 for _ in range(n):
6     print(a, end=' ')
7     a, b = b, a + b
8 print()
```

Output:

```
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
```

Explanation:

This program generates the Fibonacci sequence up to n terms directly in the main section without using separate functions or modularization. It takes the number of terms from the user and prints the series using a single loop.

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

Prompt:

Optimize version of Fibonacci series up to n terms without using functions code:

```
# Optimize version of Fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print("Fibonacci series up to", n, ":")
    print(a)
else:
    print("Fibonacci series up to", n, ":")
    while count < n:
        print(a, end=' ')
        a, b = b, a + b
        count += 1
    print()
```

Output:

```
Enter the number of terms in Fibonacci series: 5
Fibonacci series up to 5 :
0 1 1 2 3
```

Explanation:

The task resulted in an improved version of the previous code that works more efficiently while keeping it simple. Code optimization helps reduce execution time and memory usage, making the program faster and more resource-efficient.

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

Prompt:

Optimize version of Fibonacci series up to n terms with using functions

Code:

```
1 # Optimize version of Fibonacci series up to n terms with using functions
2
3 def fibonacci(n):
4     a, b = 0, 1
5     count = 0
6     if n <= 0:
7         print("Please enter a positive integer.")
8     elif n == 1:
9         print("Fibonacci series up to", n, ":")
10        print(a)
11    else:
12        print("Fibonacci series up to", n, ":")
13        while count < n:
14            print(a, end=' ')
15            a, b = b, a + b
16            count += 1
17        print()
18
19 n = int(input("Enter the number of terms in Fibonacci series: "))
20 fibonacci(n)
```

Output:

```
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 :
0 1 1 2 3 5
```

Explanation:

We created the Fibonacci series using a user-defined function that holds all the logic, making the program modular and reusable. This approach improves code organization, readability, and debugging, and is better suited for larger applications.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Prompt:

Procedural: Write a code for printing fibonacci series up to n terms without using functions

Modular: Optimized version of Fibonacci series up to n terms using functions

Code1:

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

Code2:

```
1 # Optimize version of Fibonacci series up to n terms with using functions
2
3 def fibonacci(n):
4     a, b = 0, 1
5     count = 0
6     if n <= 0:
7         print("Please enter a positive integer.")
8     elif n == 1:
9         print("Fibonacci series up to", n, ":")
10        print(a)
11    else:
12        print("Fibonacci series up to", n, ":")
13        while count < n:
14            print(a, end=' ')
15            a, b = b, a + b
16            count += 1
17        print()
18
19 n = int(input("Enter the number of terms in Fibonacci series: "))
20 fibonacci(n)
```

Output1:

```
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\lilac> []
```

Output2:

```
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 :
0 1 1 2 3 5
```

Explanations:

This task helped us understand the difference between procedural programming and modular programming using functions. Functions make code reusable, clearer, easier to debug, and more suitable for large programs, making the modular approach cleaner and better.

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

Prompt:

Iterative approach: write a code fibonacci series up to n terms using iterative approach
Recursive approach : write a code fibonacci series up to n terms using recursive approach

Code:

Itertive approach:

```
1 # write a code for fibonacci series up to n terms using iterative approach
2 def fibonacci_series(n):
3     fib_series = []
4     a, b = 0, 1
5     for _ in range(n):
6         fib_series.append(a)
7         a, b = b, a + b
8     return fib_series
9
10 # Example usage
11 n_terms = int(input("Enter the number of terms for Fibonacci series: "))
12 print(f"Fibonacci series up to {n_terms} terms: {fibonacci_series(n_terms)}")
13 # write a code for fibonacci series up to n terms using iterative approach
14 def fibonacci_series(n):
15     fib_series = []
16     a, b = 0, 1
17     for _ in range(n):
18         fib_series.append(a)
19         a, b = b, a + b
20     return fib_series
```

Recursive Approach:

```
1 # write a code for fibonacci series up to n terms using recursive approach
2 def fibonacci(n):
3     if n <= 0:
4         return []
5     elif n == 1:
6         return [0]
7     elif n == 2:
8         return [0, 1]
9     else:
10        fib_series = fibonacci(n - 1)
11        fib_series.append(fib_series[-1] + fib_series[-2])
12    return fib_series
13 # Example usage
14 n_terms = int(input("Enter the number of terms for Fibonacci series: "))
15 print(f"fibonacci series up to {n_terms} terms: {fibonacci(n_terms)}")
16
```

Output:

Iterative approach:

```
Enter the number of terms for Fibonacci series: 4
Fibonacci series up to 4 terms: [0, 1, 1, 2]
```

Recursive Approach:

```
Enter the number of terms for Fibonacci series: 5
Fibonacci series up to 5 terms: [0, 1, 1, 2, 3]
```

Explanation:

An iterative loop produces Fibonacci numbers quickly and efficiently by using minimal memory and handling large inputs smoothly. In contrast, recursion takes more time and memory, making it less practical for bigger calculations.