

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:

```
#Write a simple python code for printing fibonacci series up to n terms without using functions
n = int(input("Enter the number of terms: "))
a, b = 0, 1
count = 0
while count < n:
    print(a, end=" ")
    a, b = b, a + b
    count += 1
print() # for a new line after printing the series
```

Output:

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

Explanation:

The program generates the Fibonacci series up to a given number of terms directly within the main program, without using any user-defined functions.

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

Prompt:

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

Code:

```
1 # Optimize version of 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 count = 0
5 if n <= 0:
6     print("Please enter a positive integer.")
7 elif n == 1:
8     print("Fibonacci series up to", n, ":")
9     print(a)
10 else:
11     print("Fibonacci series up to", n, ":")
12     while count < n:
13         print(a, end=" ")
14         a, b = b, a + b
15         count += 1
16     print()
17
```

Output:

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

Explanation:

The task results in an optimized version of the earlier program, improving efficiency without increasing complexity. Optimization helps reduce execution time and memory consumption, ensuring faster and more effective use of system resources.

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:

The Fibonacci series is generated using a user-defined function that handles the complete logic. This modular design improves code clarity, reusability, and makes the program easier to manage and debug.

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

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:

This task helps compare procedural and modular programming methods.

Using functions makes the code cleaner, reusable, and better suited for large programs.

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:

Iterative 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)}")
```

Output:

```
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:

Using an iterative loop to generate Fibonacci numbers is more efficient as it is faster and uses less memory. Recursive methods are slower and memory-intensive, making loops a better choice for practical applications.