ASSIGNMENT _ 01

SYED NAZIYA PARVEEN

2303A52394

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

PROMPT:

Write a code for printing fibonacci series up to n terms

CODE:

```
#write a 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:")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print()
```

OUTPUT:

```
PS C:\Users\sdnaz> & C:/Users/sdnaz/AppData/Local/Progra

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

JUSTIFICATION

By completing this task, we have printed the required Fibonacci series up to *n* terms without using modularization. The logic is written directly in the main program without defining any user-defined functions. The program accepts user input for the number of terms and generates the Fibonacci sequence using a single for loop.

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

PROMPT

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

CODE:

```python
#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
print("Fibonacci series:")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print()  # for a new line after the series10
```

OUTPUT:

```
PS C:\Users\sdnaz> & C:/Users/sdnaz/AppData/Local/Progr

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

JUSTIFICATION

By completing this task, we obtained an optimized version of the previous code. The code used in Task 1 is already simple and efficient. Code optimization is important because it helps reduce the time taken to run the program and the memory used by it. A program with high time complexity runs slower, and a program with high space complexity uses more memory. Therefore, optimizing the code helps us complete tasks faster and more effi**ciently.**

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

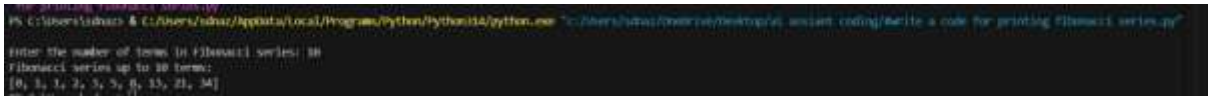
PROMPT:

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

CODE:

```python
#optimize version of Fibonacci series up to n terms with using functions
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, ":")
print(fibonacci_series(num_terms))
```

OUTPUT:



JUSTIFICATION:

By completing this task, we implemented the Fibonacci series using a user-defined function. The function contains all the Fibonacci logic, which makes the code modular and easy to reuse. Using modularization helps to organize the code properly and allows the same function to be used in different programs. This method makes the code easier to read, easier to debug, and more suitable for large applications.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

PROMPTS:

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:

procedural

```
#write a 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:")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print()
```

modular

```
#optimize version of Fibonacci series up to n terms with using functions
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, "terms:")
print(fibonacci_series(num_terms))
```

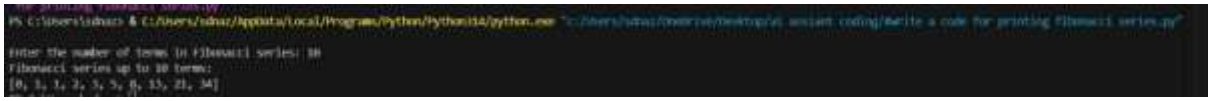
OUTPUTS:

Procedural:

```
PS C:\Users\sdnaz> & C:/Users/sdnaz/AppData/Local/Progra

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

Modular:



JUSTIFICATION:

By this task , we are able to find the difference between Procedural(without using functions) and Modular(with using functions). The main use of function is

- Reusability of the code
- Easy to Debug
- Code Clarity
- Suitable for large systems

By observing, we can analyze that using modular method is a better and clean approach

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

PROMPTS:

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:

```python
#write a code for fibonacci series up to n terms using iterative approach
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, "terms:")
print(fibonacci_series(num_terms))
```

```python
#write a code for fibonacci series up to n terms using recursive approach
def fibonacci_series(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci_series(n - 1)
        series.append(series[-1] + series[-2])
        return series
num_terms = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series up to", num_terms, "terms:")
print(fibonacci_series(num_terms))
```

OUTPUT:

Iterative approach

```
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\sdnaz> & C:/Users/sdnaz/AppData/Local/Programs/Python/Python314/python.exe
```

Recursive approach

```
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\sdnaz> & C:/Users/sdnaz/AppData/Local/Programs/Python/Python314/python.exe
```

Justification:

Using a loop (iterative method) to find Fibonacci numbers is usually better because it's faster, uses less memory, and can handle big numbers easily. The recursive method (using functions that call themselves) is slower, uses more memory, and can crash if the number is too large. That's why in real programs, loops are preferred over recursion.