ASSIGNMENT _ 01

V.HRK Sri Harsha Vignesh

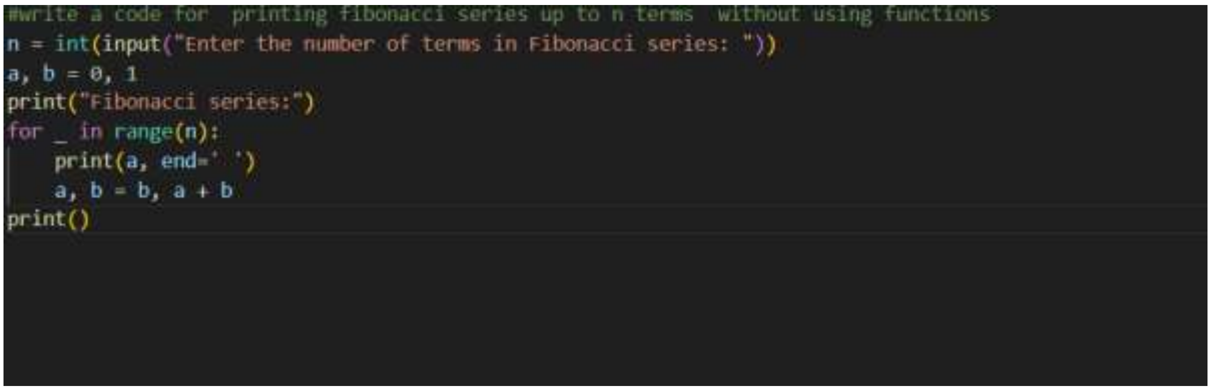
2303A52304

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

PROMPT:

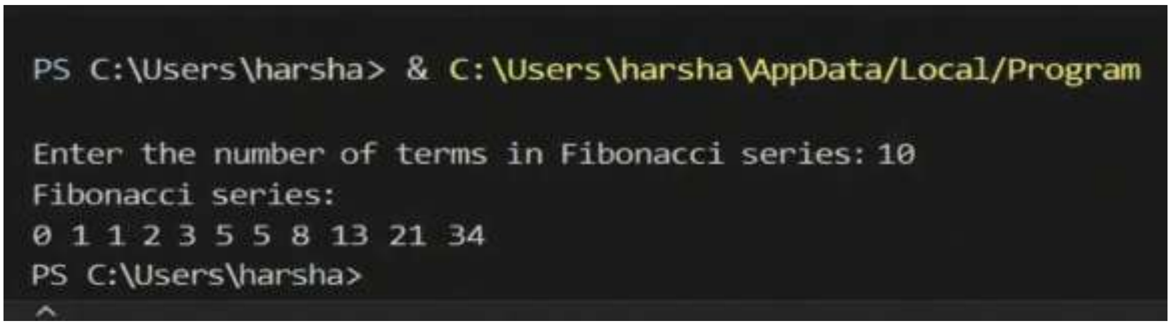
Write a code for printing fibonacci series up to n terms

CODE:



```python
#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\harsha> & C:\Users\harsha\AppData/Local/Program

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

JUSTIFICATION

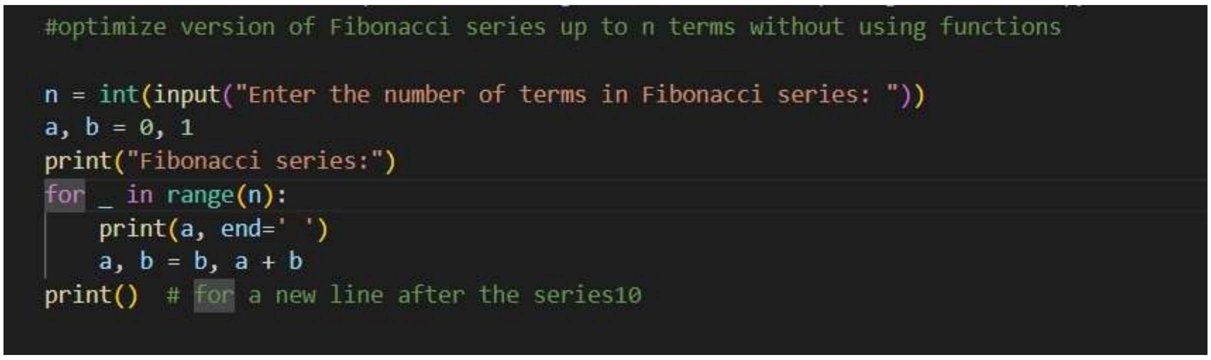
In this task, the Fibonacci series is printed up to n terms without using functions. The logic is written directly in the main program using a loop

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

PROMPT

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

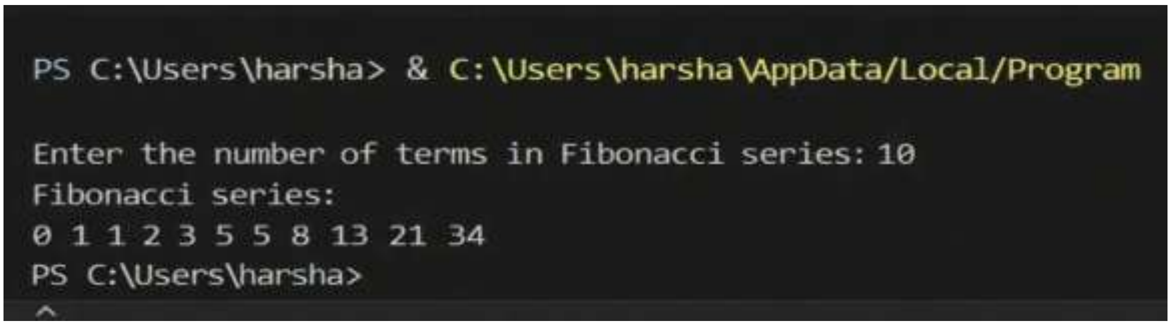
CODE:



```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\harsha> & C:\Users\harsha\AppData/Local/Program

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

JUSTIFICATION

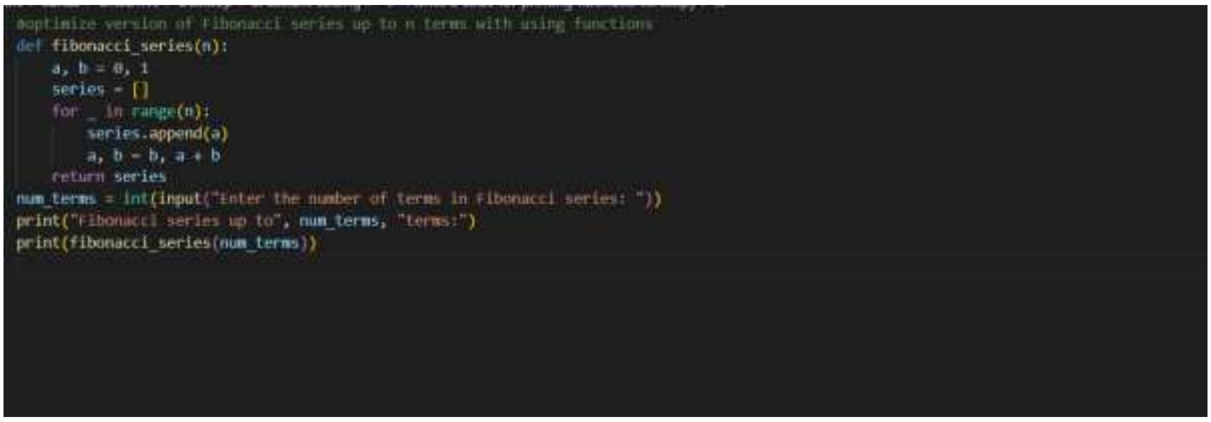
This task shows an optimized version of the Fibonacci code. Code optimization is important because it helps the program run faster and use less memory.

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

PROMPT:

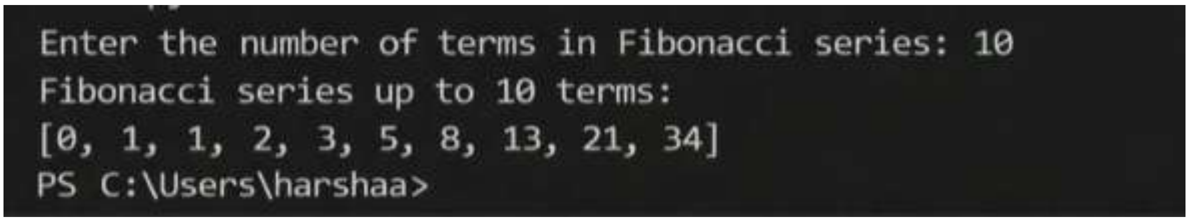
Optimize version of Fibonacci series up to n terms with using functions

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, "terms:")
print(fibonacci_series(num_terms))
```

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]
PS C:\Users\harshaa>
```

JUSTIFICATION:

In this task, Fibonacci series is generated using a user-defined function. Using functions makes the code reusable and easy to understand.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

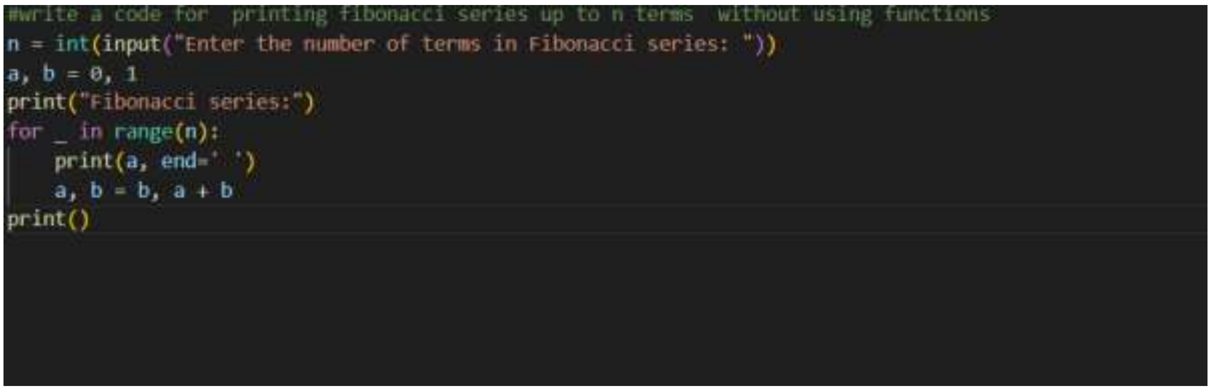
PROMPTS:

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

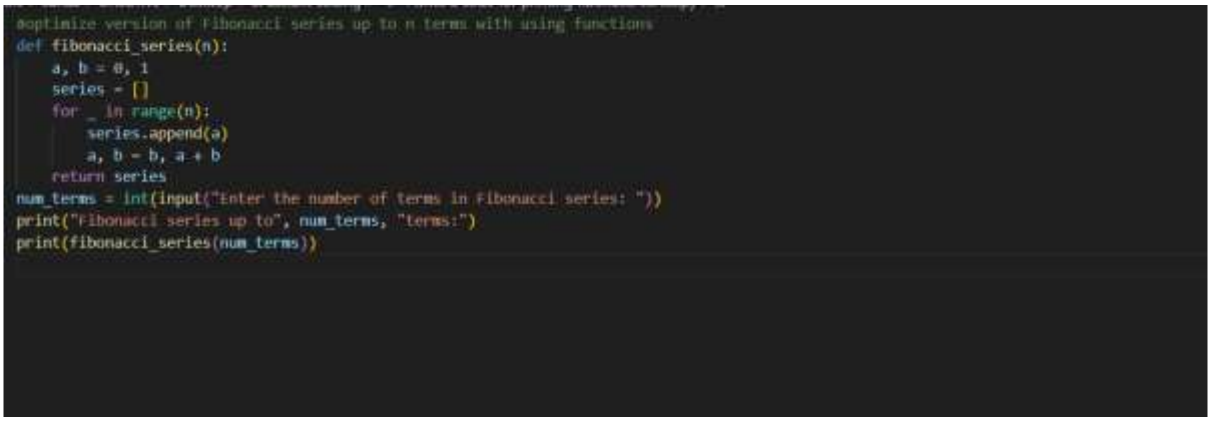
CODE:

procedural



```python
#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()
```

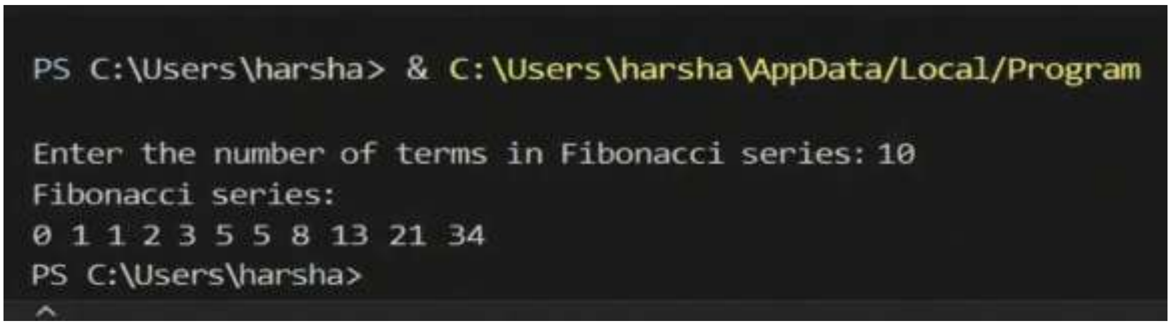
modular



```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, "terms:")
print(fibonacci_series(num_terms))
```

OUTPUTS:

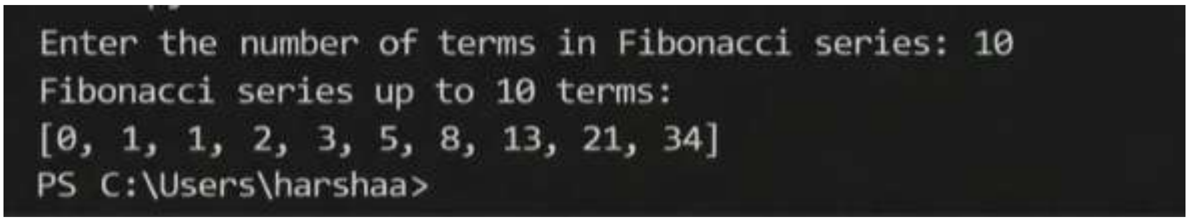
Procedural:



```
PS C:\Users\harsha> & C:\Users\harsha\AppData/Local/Program

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

Modular:



```
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\harshaa>
```

JUSTIFICATION:

This task explains the difference between procedural and modular code. Modular approach is more efficient and suitable for large programs.

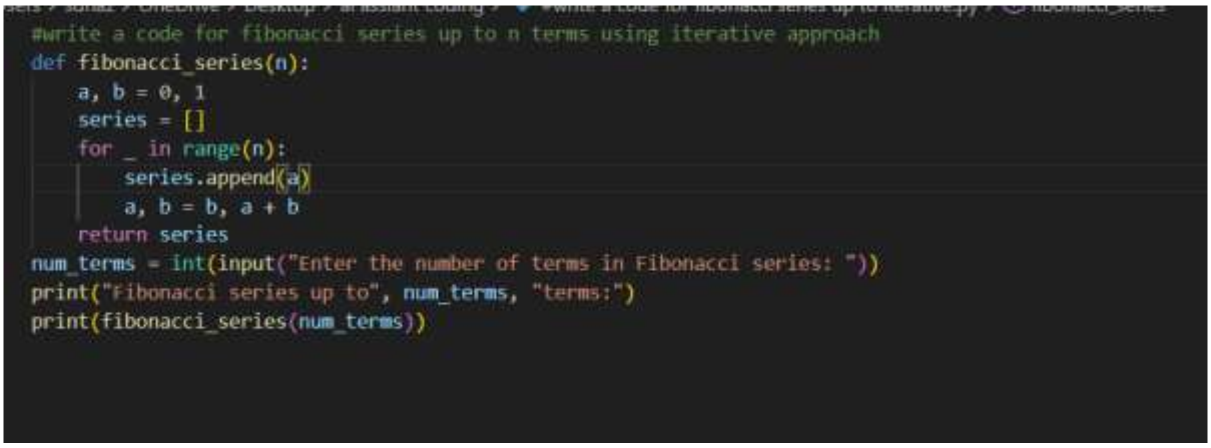
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

PROMPTS:

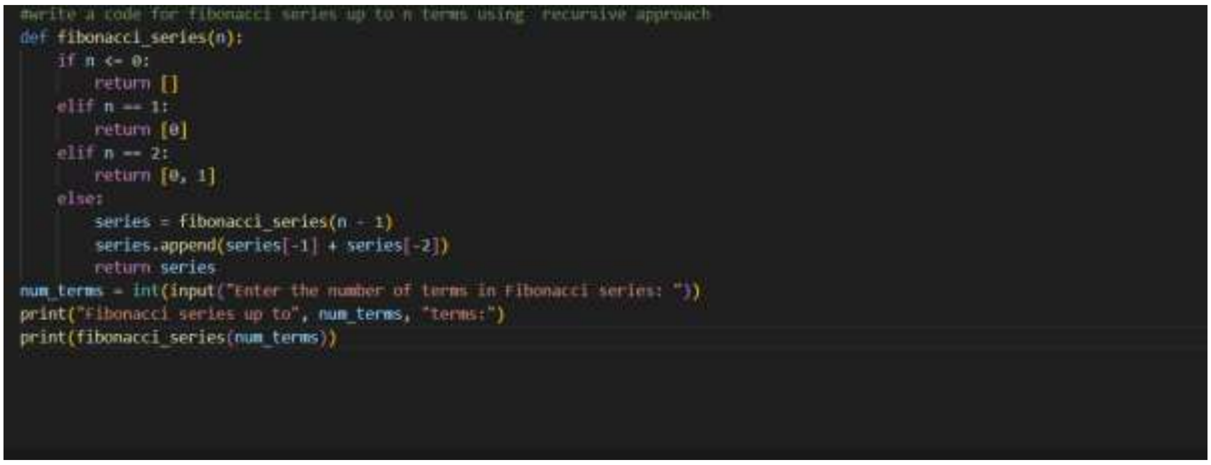
Iterative approach: fibonacci series up to n terms using iterative approach

Recursive approach : fibonacci series up to n terms using recursive approach

CODE:



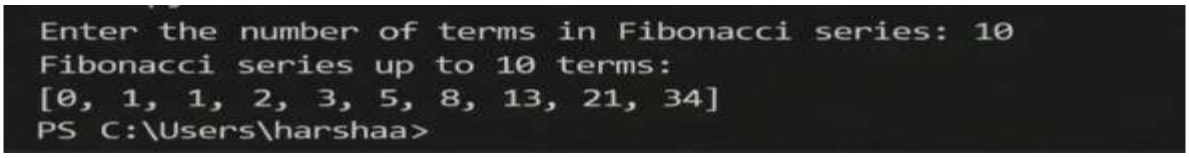
```python
#write a code for fibonacci series up to n terms using iterative approach
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = int(input("Enter the number of terms in Fibonacci series: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

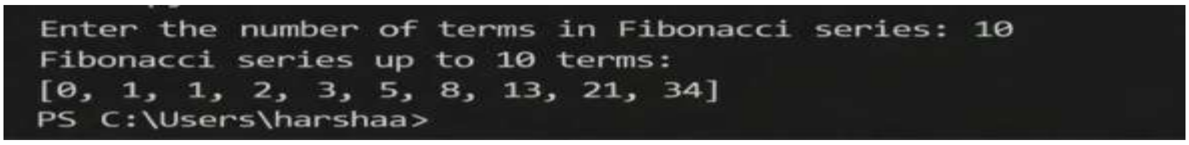
OUTPUT:

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\harshaa>
```

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\harshaa>
```

Justification:

In this task, The iterative Fibonacci approach is usually the better choice because it’s faster, uses very little memory, and works well even when the number of terms is large. Recursive Fibonacci, on the other hand makes many function calls, which slows things down and uses extra memory. For bigger inputs, it can even crash due to recursion limits. That’s why, in real-world programs, iteration is generally preferred over recursion.