Assignment 01

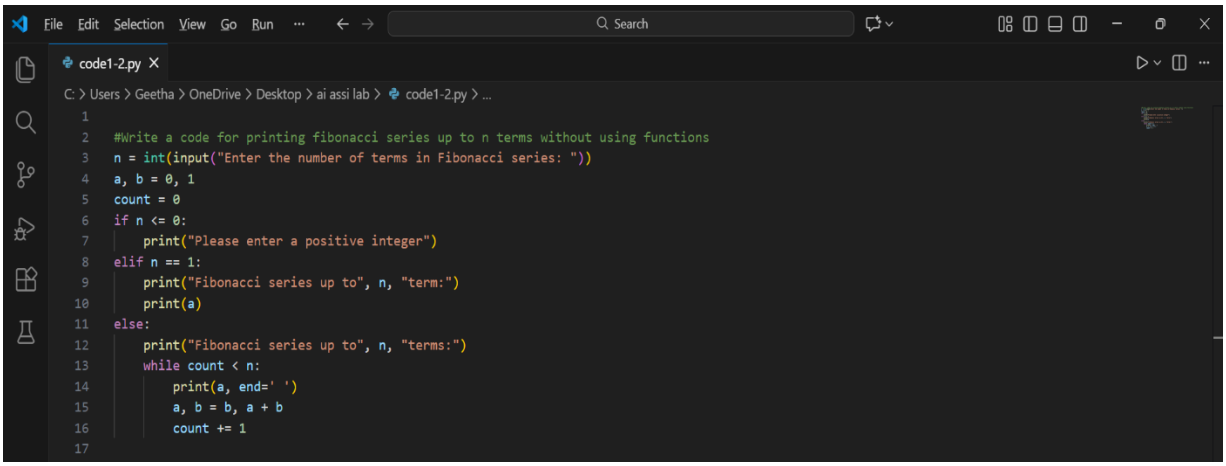
D. VIDHU SREE
2303A52400 (BATCH 41)

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

PROMPT:

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

CODE:



OUTPUT:

JUSTIFICATION:

By completing this task , we have printed the required fibonocci series up to n terms without modularization. In this code we are printing the series with only one for loop

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

PROMPT:

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

CODE:

OUTPUT:

JUSTIFICATION:

By completing this task , we have got the optimized version of the previous code , as we can see the code which we have got the in task 1 is already optimized one.

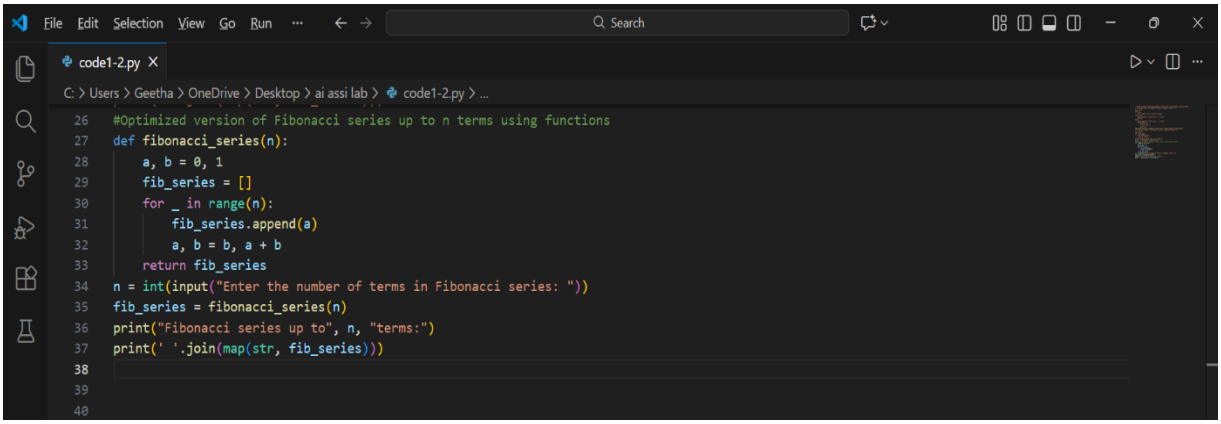
Optimization of the code is very much important, as it reduces the time complexity and space complexity of the code, for the code which has more time complexity takes more time to run(slower) and which has more space complexity will utilize maximum space. So for completing our task efficiently , we need to Optimize our code

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

PROMPT:

Optimized version of Fibonacci series up to n terms using functions

CODE:



OUTPUT:

JUSTIFICATION:

By completing this task , we are able to print the Fibonacci series up to n numbers using functions by using the above code. At first we are calling the user-defined function using function calling statement and then our function runs. In our function the main for loop runs and prints the output

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

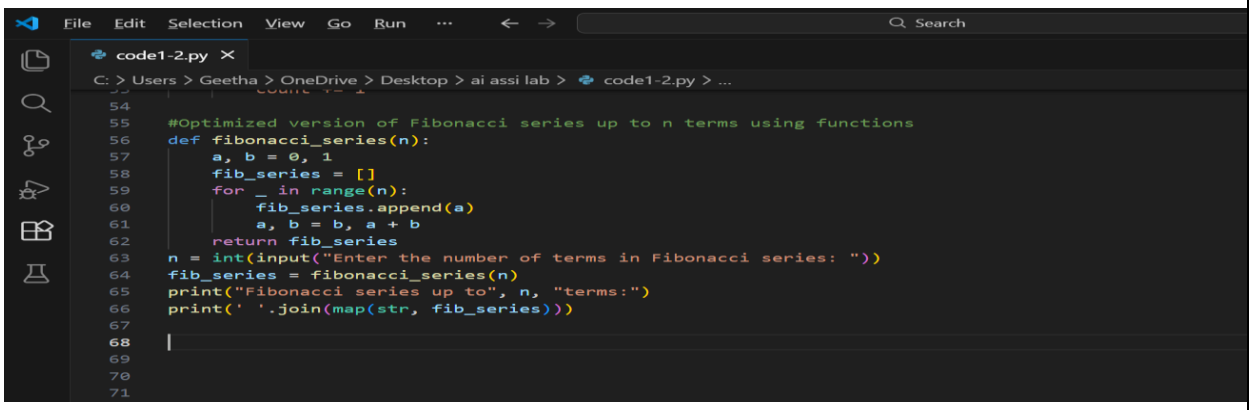
CODE:

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
count = 0
if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print("Fibonacci series up to", n, "term:")
    print(a)
else:
    print("Fibonacci series up to", n, "terms:")
    while count < n:
        print(a, end=' ')
        a, b = b, a + b
        count += 1

```

Modular



OUTPUT:

Procedural

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 aproch

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

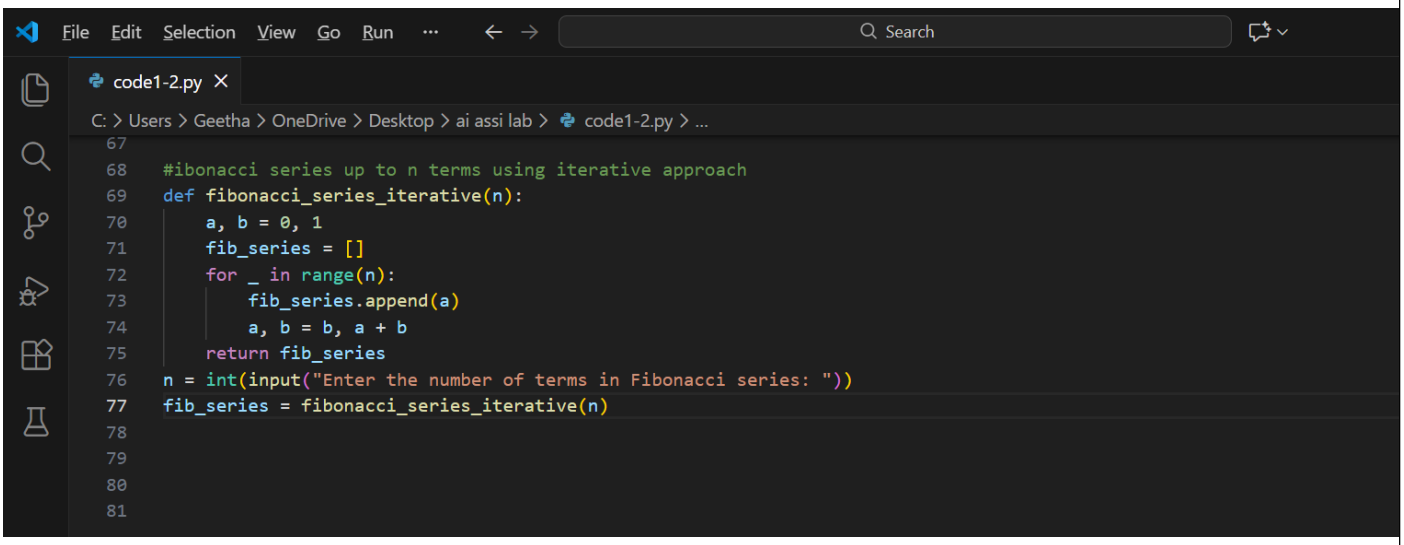
PROMPTS:

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

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

CODE:

Iterative approach:



Recursive approach :

code1-2.py

C: > Users > Geetha > OneDrive > Desktop > ai assi lab > code1-2.py > ...

```
78
79  #fibonacci series up to n terms using recursive approach
80  def fibonacci_series_recursive(n, a=0, b=1, fib_series=None):
81      if fib_series is None:
82          fib_series = []
83      if n == 0:
84          return fib_series
85      fib_series.append(a)
86      return fibonacci_series_recursive(n-1, b, a + b, fib_series)
87  n = int(input("Enter the number of terms in Fibonacci series: "))
88  fib_series = fibonacci_series_recursive(n)
89  print("Fibonacci series up to", n, "terms:")
90
91
```

OUTPUT:

Iterative approach:

Recursive approach

JUSTIFICATION:

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.