

Name: K.Likitha

Hall Ticket: 2303A52144

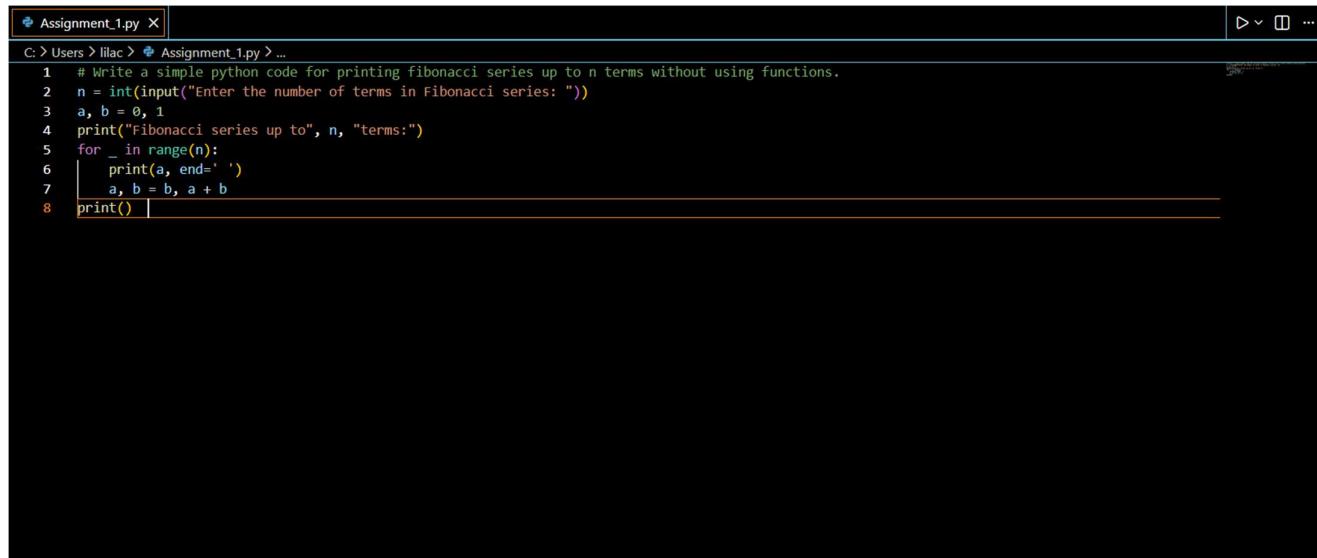
Batch: 41

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.

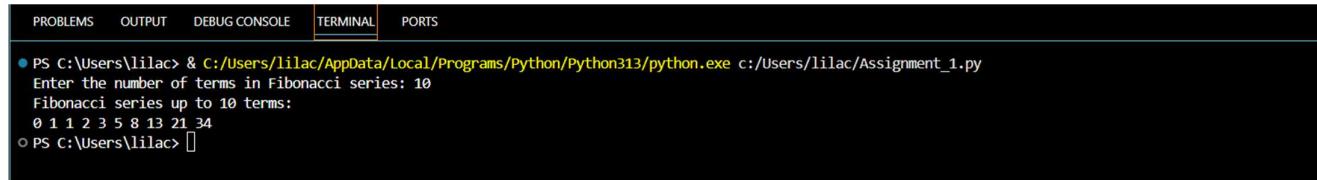
Code:



A screenshot of a terminal window titled "Assignment_1.py". The window shows a command prompt at "C:\Users>". Below the prompt is a block of Python code. The code defines a function named "fibonacci" that takes an integer "n" as input. It initializes two variables, "a" and "b", both set to 1. It then enters a loop that runs "n" times. In each iteration, it prints the value of "a" followed by a space, then updates "a" to be the sum of "a" and "b", and "b" to be the previous value of "a". After the loop, the function returns "a". The code is as follows:

```
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 = 1, 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:



A screenshot of a terminal window titled "TERMINAL". The window shows a command prompt at "PS C:\Users\lilac>". The user runs the command "c:/users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/users/lilac/Assignment_1.py". The program prompts for the number of terms and prints the Fibonacci sequence up to 10 terms. The output is as follows:

```
PS C:\Users\lilac> & c:/users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/users/lilac/Assignment_1.py
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> []
```

Explanation:

By completing this task, the Fibonacci series up to n terms is generated directly within the main program without using modularization or user-defined functions. The program takes user input for the number of terms and produces the Fibonacci sequence 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:

```
Assignment_1.py
C:> Users > lilac > Assignment_1.py > ...
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:

```
PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
● Enter the number of terms in Fibonacci series: 5
Fibonacci series up to 5 :
0 1 1 2 3
```

Explanations:

By completing this task, we obtained an optimized version of the earlier code, improving its efficiency while maintaining simplicity. Code optimization reduces execution time and memory usage, enabling programs to run faster and use resources more effectively.

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

Prompt:

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

Code:

```
Assignment_1.py X
C:> Users > lilac > Assignment_1.py > ...
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:

```
0 1 1 2 3
PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 :
0 1 1 2 3 5
```

Explanation:

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.

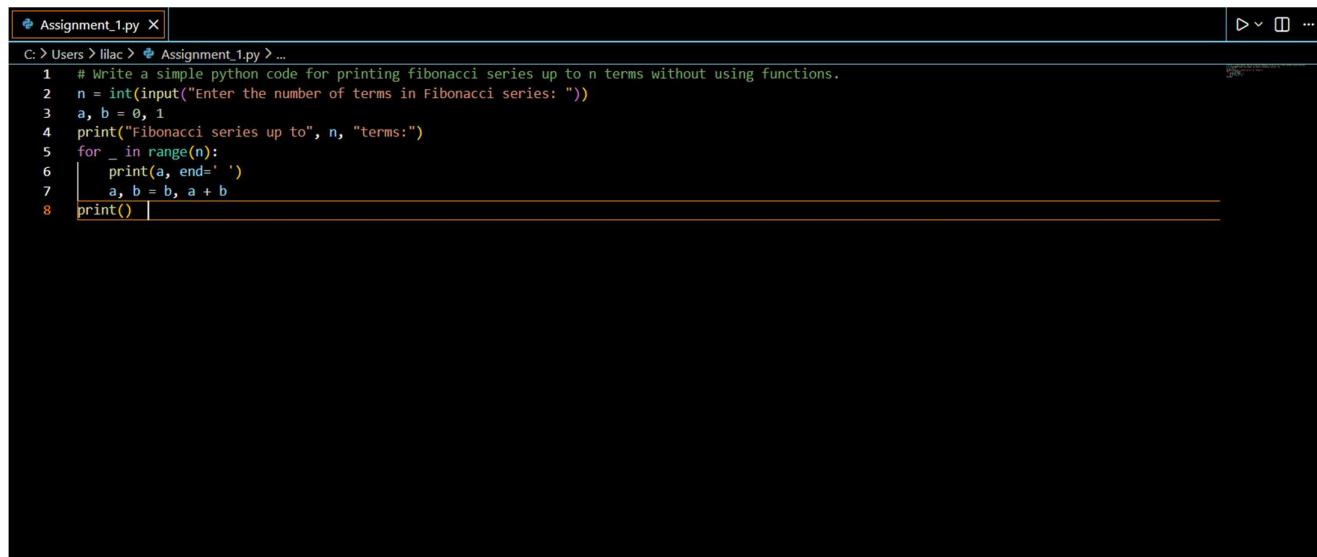
Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Prompt:

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

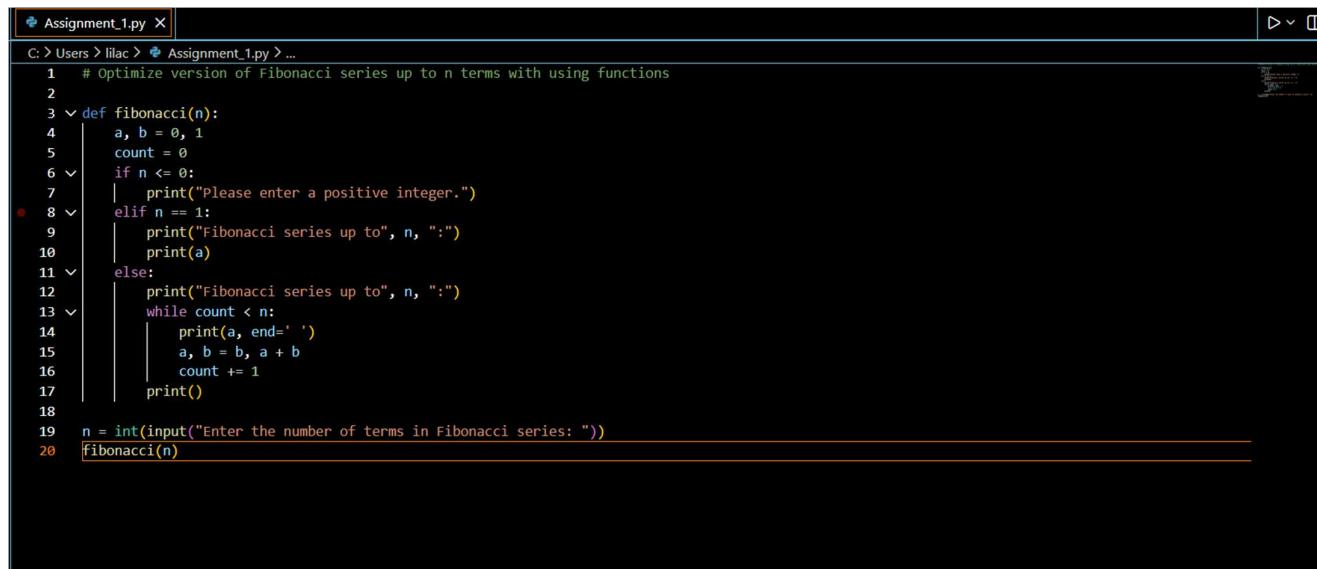
Modular: Optimized version of Fibonacci series up to n terms using functions

Code:



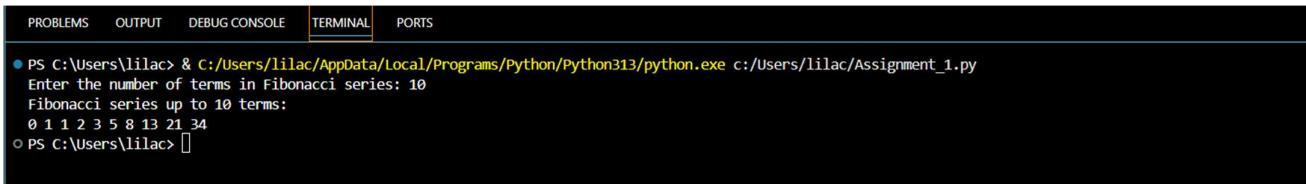
```
Assignment_1.py X
C:\> Users > lilac > Assignment_1.py > ...
1 # Write a simple python code for printing fibonacci series up to n terms without using functions.
2 n = int(input("Enter the number of terms in Fibonacci series: "))
3 a, b = 0, 1
4 print("Fibonacci series up to", n, "terms:")
5 for _ in range(n):
6     print(a, end=' ')
7     a, b = b, a + b
8 print()
```

Code:



```
Assignment_1.py X
C:\> Users > lilac > Assignment_1.py > ...
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:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
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> []
```

Output:



```
0 1 1 2 3
● PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
Enter the number of terms in Fibonacci series: 6
Fibonacci series up to 6 :
0 1 1 2 3 5
```

Explanations:

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)

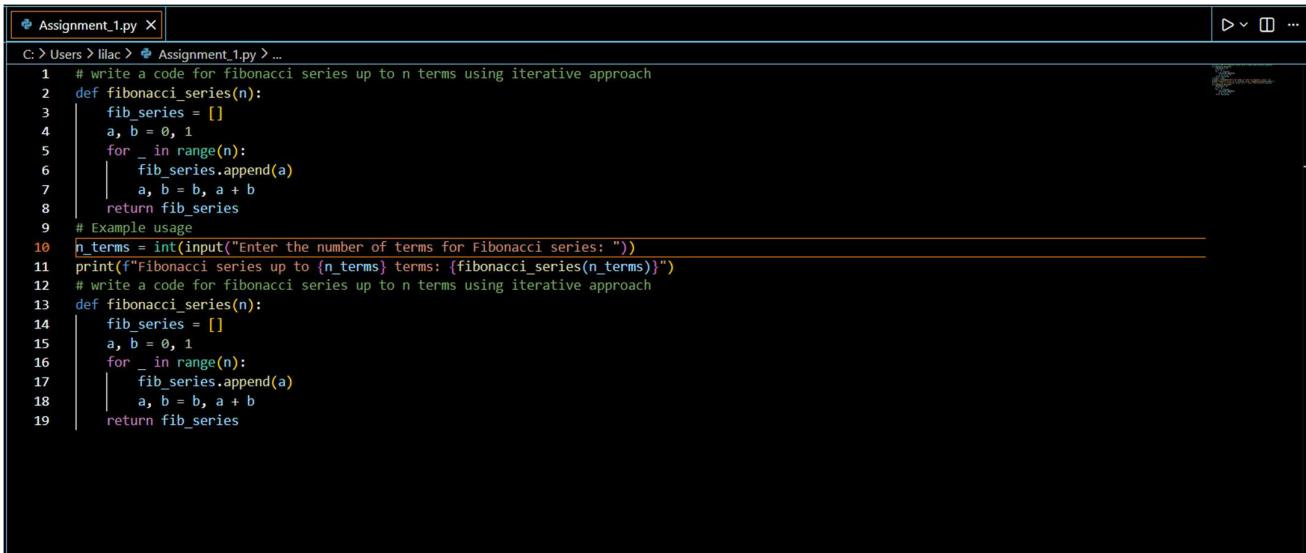
Prompt:

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:



```
Assignment_1.py X
C:> Users > lilac > Assignment_1.py >
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 # Example usage
10 n_terms = int(input("Enter the number of terms for Fibonacci series: "))
11 print(f"Fibonacci series up to {n_terms} terms: {fibonacci_series(n_terms)}")
12 # write a code for fibonacci series up to n terms using iterative approach
13 def fibonacci_series(n):
14     fib_series = []
15     a, b = 0, 1
16     for _ in range(n):
17         fib_series.append(a)
18         a, b = b, a + b
19     return fib_series
```

Recursive Approach:

```
C:\> Users > lilac > Assignment_1.py > ...
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:

```
PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
Enter the number of terms for Fibonacci series: 4
Fibonacci series up to 4 terms: [0, 1, 1, 2]
```

Recursive Approach:

```
PS C:\Users\lilac> & C:/Users/lilac/AppData/Local/Programs/Python/Python313/python.exe c:/Users/lilac/Assignment_1.py
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 because it is faster, uses less memory, and handles large values easily. Recursive methods are slower, consume more memory, and may fail for large inputs, so loops are preferred in real-world programs.