# AI Assistant Coding
## <u>Assignment-1.3</u>

V.Sai Nithish
2303A52410
BATCH-38

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

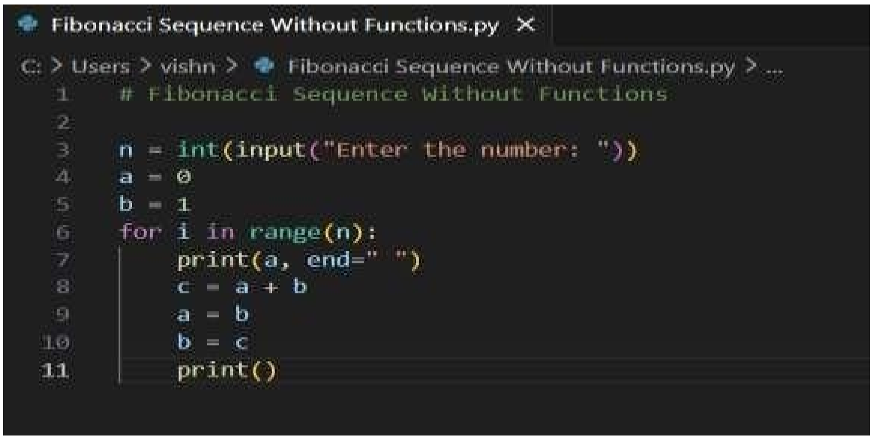
**Use GitHub Copilot to generate a Python program that:**

- **➢ Prints the Fibonacci sequence up to *n* terms**
- **➢ Accepts user input for *n***
- **➢ Implements the logic directly in the main code ➢ Does not use any user-defined functions**

**Prompt:**

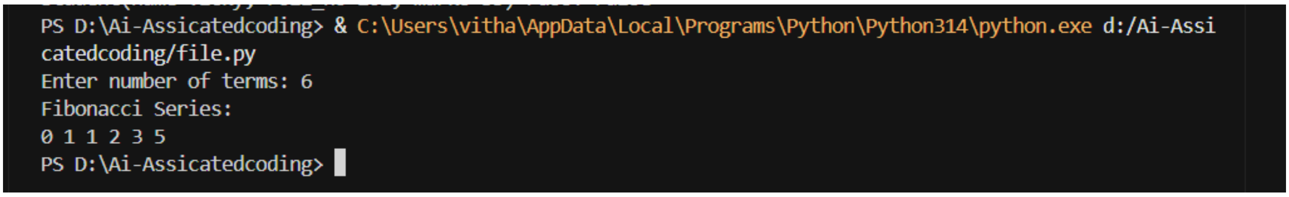
write a code to print fibonacci series upto n without using a function

**Code:**


Fibonacci Sequence Without Functions.py
C: > Users > vishn > Fibonacci Sequence Without Functions.py > ...

```python
# Fibonacci Sequence Without Functions

n = int(input("Enter the number: "))
a = 0
b = 1
for i in range(n):
    print(a, end=" ")
    c = a + b
    a = b
    b = c
    print()
```

**Output:**



```
PS D:\Ai-Assicatedcoding> & C:\Users\vitha\AppData\Local\Programs\Python\Python314\python.exe d:/Ai-Assi
catedcoding/file.py
Enter number of terms: 6
Fibonacci Series:
0 1 1 2 3 5
PS D:\Ai-Assicatedcoding> 
```

**Explanation:**

The user enters a number `n`, which represents how many terms of the Fibonacci series should be printed. The program starts with `a = 0` and `b = 1`. Inside a loop that runs `n` times, it prints the current value of `a`, then updates the values by adding the previous two numbers. This continues until `n` terms are printed.

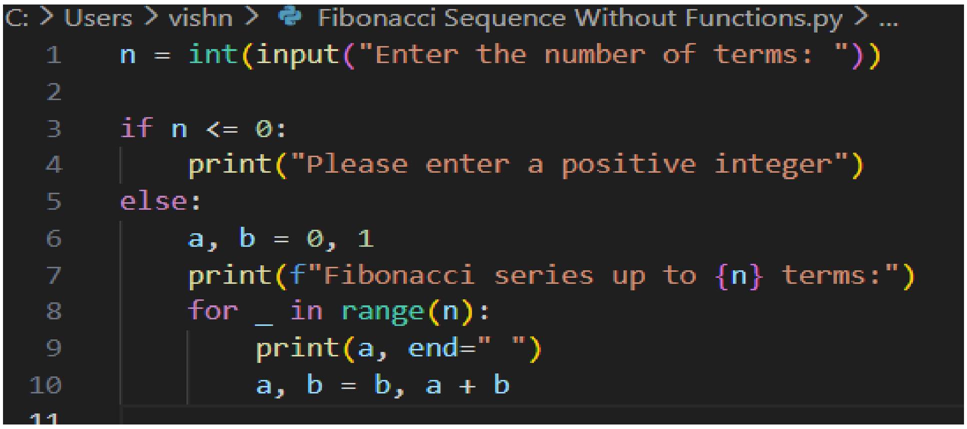
### Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

- **➢ Examine the Copilot-generated code from Task 1 and improve it by:**

- ➢ **Removing redundant variables**
- ➢ **Simplifying loop logic**
- ➢ **Avoiding unnecessary computations ➢ Use Copilot prompts such as: ▪ “Optimize this Fibonacci code”**
- ➢ **“Simplify variable usage”**

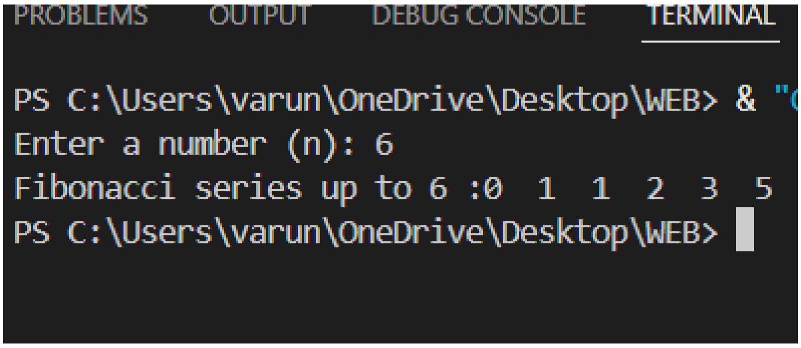
**Prompt:**
Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.

**Code:**


C: > Users > vishn > Fibonacci Sequence Without Functions.py > ...

```python
n = int(input("Enter the number of terms: "))

if n <= 0:
    print("Please enter a positive integer")
else:
    a, b = 0, 1
    print(f"Fibonacci series up to {n} terms:")
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

**Output:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "
Enter a number (n): 6
Fibonacci series up to 6 :0  1  1  2  3  5
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

**Explanation:**

This optimized version of the Fibonacci program removes extra variables and makes the logic cleaner. Instead of using a third variable, it updates both values in a single line using tuple assignment (`a, b = b, a + b`). This makes the program shorter, more readable, and more efficient while producing the same output.

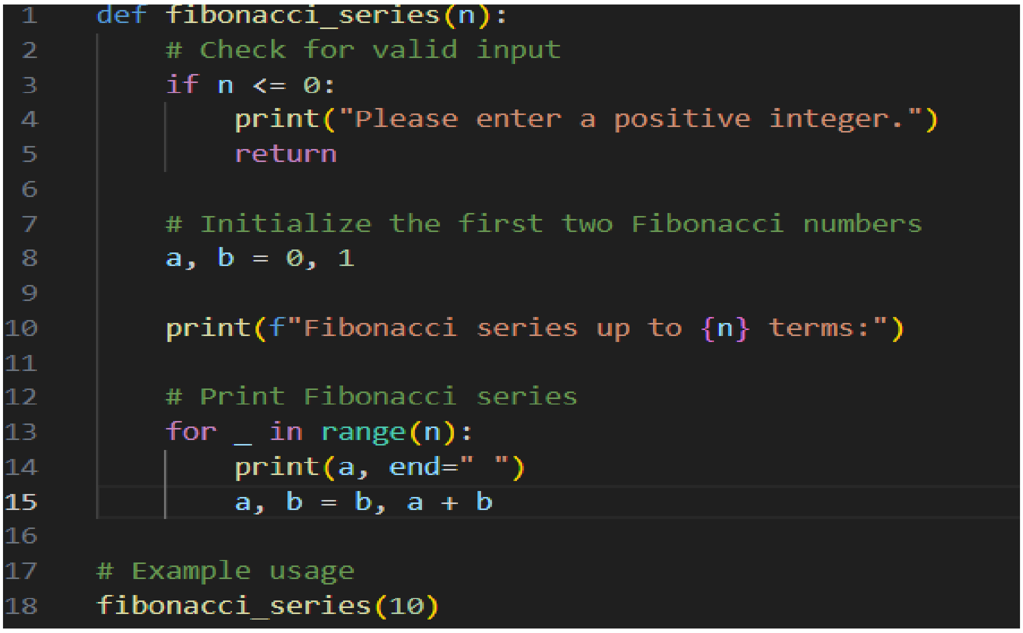
## Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions) Use GitHub Copilot to generate a function-based Python program that:

- **Uses a user-defined function to generate Fibonacci numbers**
- **Returns or prints the sequence up to n**
- **Includes meaningful comments (AI-assisted)**

**Prompt:**

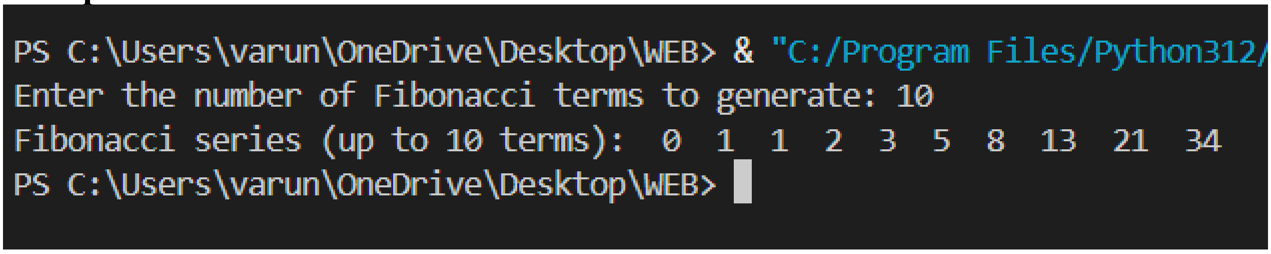
Write a code for printing the Fibonacci series up to n terms using a function and include meaningful comments.

**Code:**



```python
def fibonacci_series(n):
    # Check for valid input
    if n <= 0:
        print("Please enter a positive integer.")
        return

    # Initialize the first two Fibonacci numbers
    a, b = 0, 1

    print(f"Fibonacci series up to {n} terms:")

    # Print Fibonacci series
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

# Example usage
fibonacci_series(10)
```

**Output:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Files/Python312/
Enter the number of Fibonacci terms to generate: 10
Fibonacci series (up to 10 terms):  0  1  1  2  3  5  8  13  21  34
PS C:\Users\varun\OneDrive\Desktop\WEB> 
```

**Explanation:**

This program uses a **user-defined function** named `print_fibonacci(n)` to generate the Fibonacci series. The function first checks whether the input is valid. If valid, it initializes two variables and uses a loop to generate and print the series. The main program only takes input and calls the function, making the code more organized and reusable.

## Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

**Compare the Copilot-generated Fibonacci programs:**

- **Without functions (Task 1)**
- **With functions (Task 3)**

- **Analyze them in terms of:**
    - **Code clarity**
    - **Reusability**
    - **Debugging ease**
    - **Suitability for larger systems**

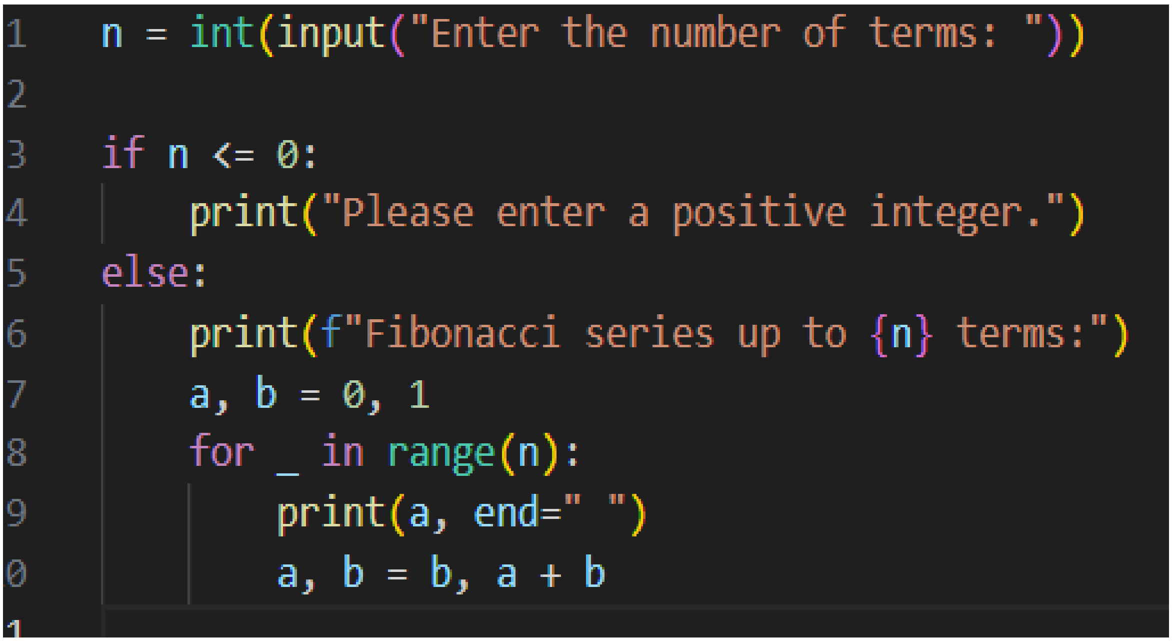
**Prompt:**

Non-Modular: Write a code for printing a Fibonacci series up to n terms without using a function.

Modular: Write a code for printing the Fibonacci series up to n terms using a function and include meaningful comments.

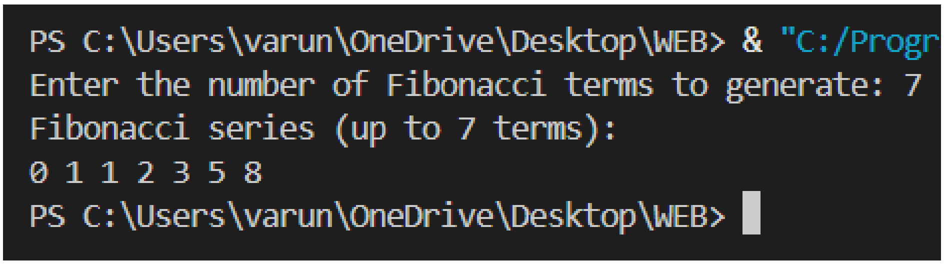
**Code:**

**Procedural:**



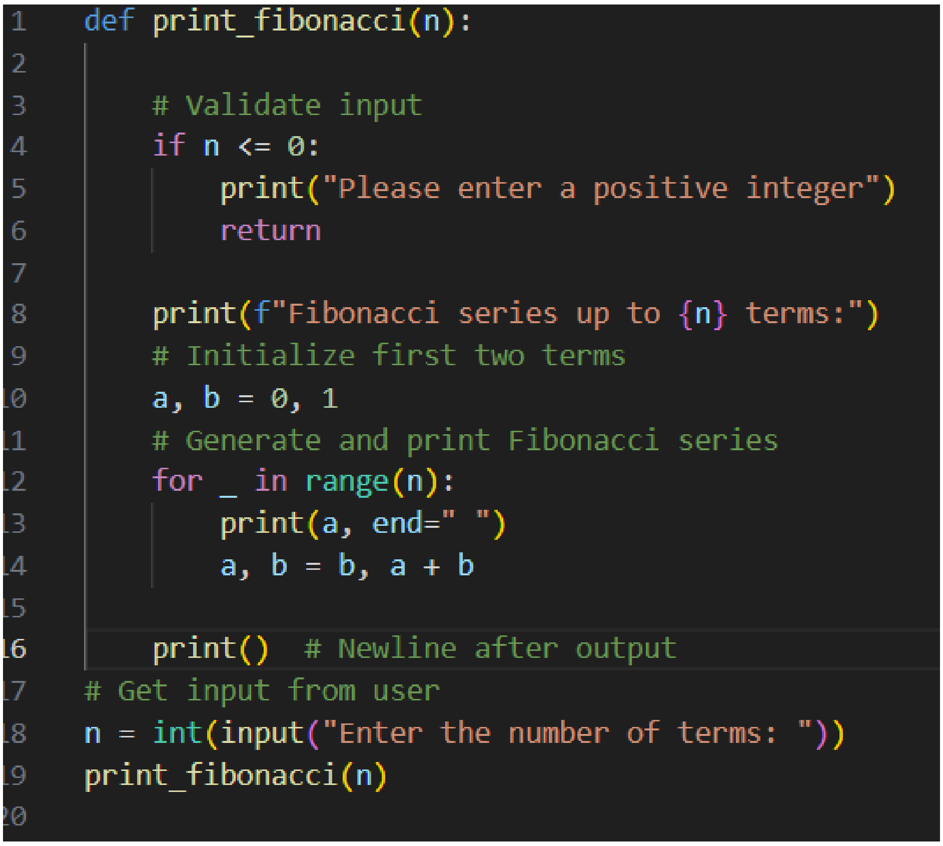
```python
n = int(input("Enter the number of terms: "))

if n <= 0:
    print("Please enter a positive integer.")
else:
    print(f"Fibonacci series up to {n} terms:")
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
```

**Output:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Progr
Enter the number of Fibonacci terms to generate: 7
Fibonacci series (up to 7 terms):
0 1 1 2 3 5 8
PS C:\Users\varun\OneDrive\Desktop\WEB> 
```

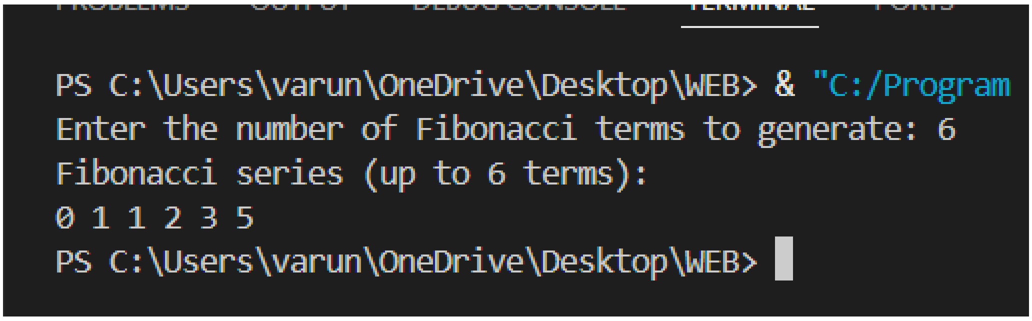
**Modular:**



```python
def print_fibonacci(n):

    # Validate input
    if n <= 0:
        print("Please enter a positive integer")
        return

    print(f"Fibonacci series up to {n} terms:")
    # Initialize first two terms
    a, b = 0, 1
    # Generate and print Fibonacci series
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

    print()  # Newline after output
# Get input from user
n = int(input("Enter the number of terms: "))
print_fibonacci(n)
```

**Output:**

**Modular:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program
Enter the number of Fibonacci terms to generate: 6
Fibonacci series (up to 6 terms):
0 1 1 2 3 5
PS C:\Users\varun\OneDrive\Desktop\WEB> 
```

**Explanation:**
In the **procedural approach**, the Fibonacci series code is written directly in the main program without using any function. The steps run one after another, and this method is simple but not reusable.
In the **modular approach**, the Fibonacci logic is written inside a function. The main program only calls the function. This makes the code more organized, easy to understand, and reusable.

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

**Prompt GitHub Copilot to generate:**

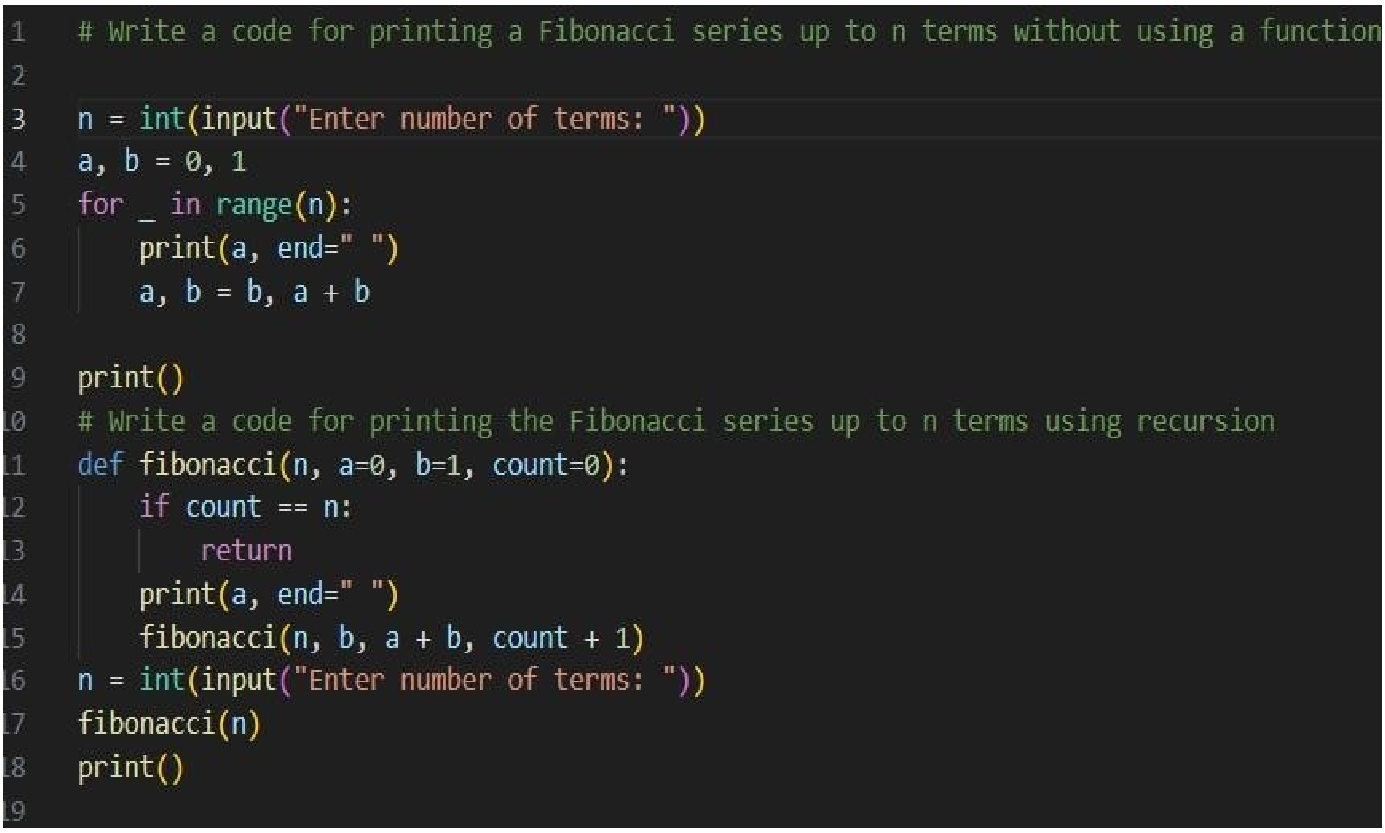
**An iterative Fibonacci implementation**
**A recursive Fibonacci implementation**

**Prompt:**

Write a code for printing a Fibonacci series up to n terms without using a function.

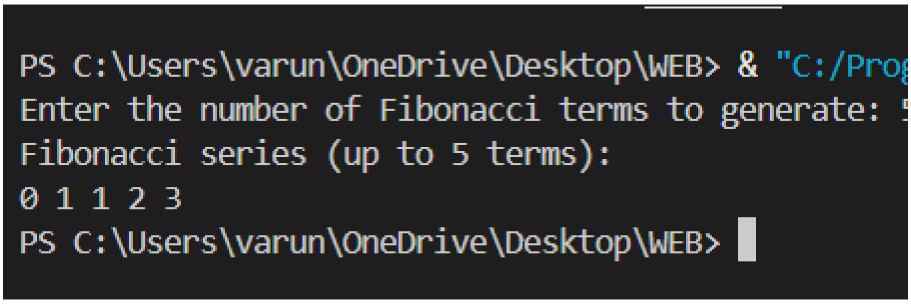
Write a code for printing the Fibonacci series up to n terms using recursion.

**Code:**



```python
# Write a code for printing a Fibonacci series up to n terms without using a function

n = int(input("Enter number of terms: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b

print()
# Write a code for printing the Fibonacci series up to n terms using recursion
def fibonacci(n, a=0, b=1, count=0):
    if count == n:
        return
    print(a, end=" ")
    fibonacci(n, b, a + b, count + 1)
n = int(input("Enter number of terms: "))
fibonacci(n)
print()

```

**Output:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Prog
Enter the number of Fibonacci terms to generate: 
Fibonacci series (up to 5 terms):
0 1 1 2 3
PS C:\Users\varun\OneDrive\Desktop\WEB> 
```

**Explanation:**

This program prints the Fibonacci series in two ways .

In the first part, the Fibonacci series is printed **without using a function**. The user enters the number of terms, and the program uses a loop to print the series by adding the previous two numbers each time.

In the second part, the Fibonacci series is printed **using recursion**. A function is defined that prints one Fibonacci number and then calls itself to print the next one. This process continues until the required number of terms is printed. Both methods produce the same Fibonacci series but use different approaches.