Name:Kolipaka Sanjana

Roll.no:2303a52139

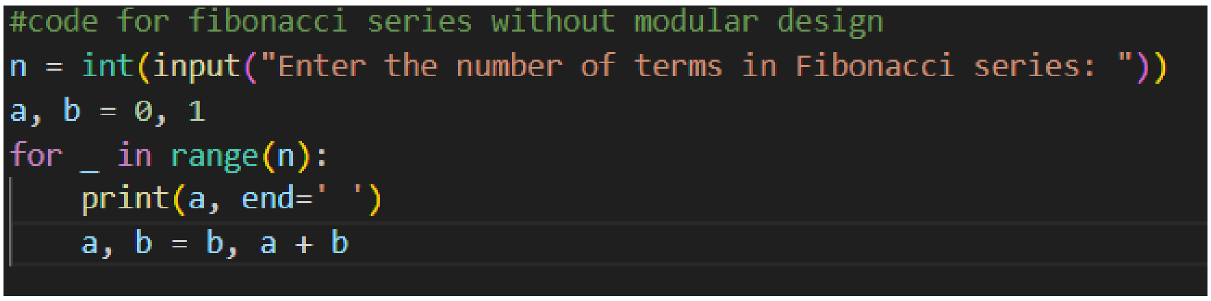
Batch:41

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

Without Functions)

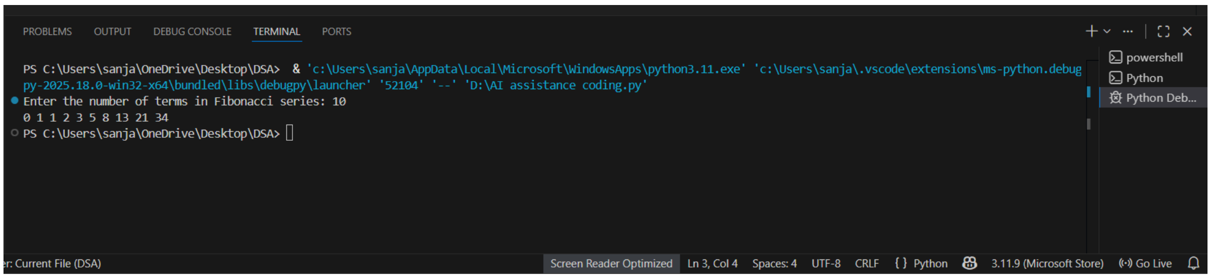
Prompt:Code for fibonacci series without modular design

Code:



```python
#code for fibonacci series without modular design
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

Output:



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debug
py-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA>
```

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)

❖ Scenario

The prototype will be shared with other developers and needs

optimization.

❖ Task Description

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

➢ Removing redundant variables

➢ Simplifying loop logic

➢ Avoiding unnecessary computations

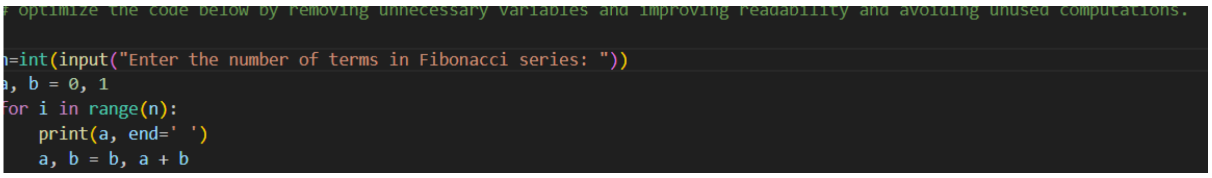
➢ Use Copilot prompts such as:

▪ “Optimize this Fibonacci code”

▪ “Simplify variable usage”

Prompt: optimize the code below by removing unnecessary variables and improving readability and avoiding unused computations.

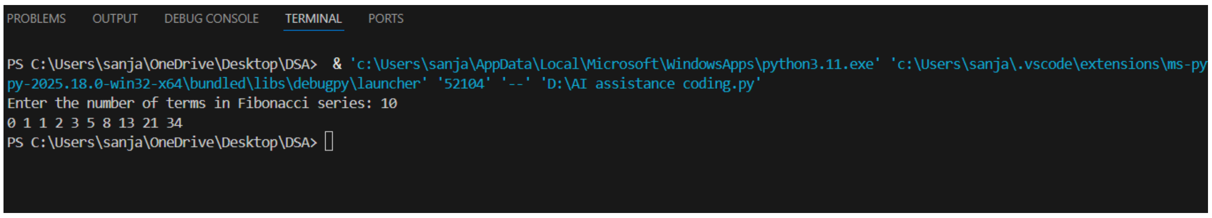
Code:



```
# optimize the code below by removing unnecessary variables and improving readability and avoiding unused computations.

=int(input("Enter the number of terms in Fibonacci series: "))
, b = 0, 1
or i in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

Output:



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-py
py-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA> 
```

Justification:

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

❖ Scenario

The Fibonacci logic is now required in multiple modules of an application.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

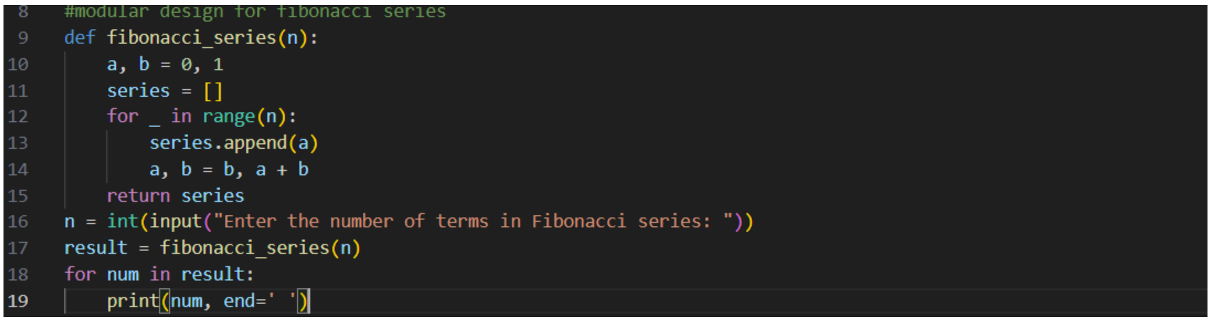
➢ Uses a user-defined function to generate Fibonacci numbers

➢ Returns or prints the sequence up to n

➢ Includes meaningful comments (AI-assisted)

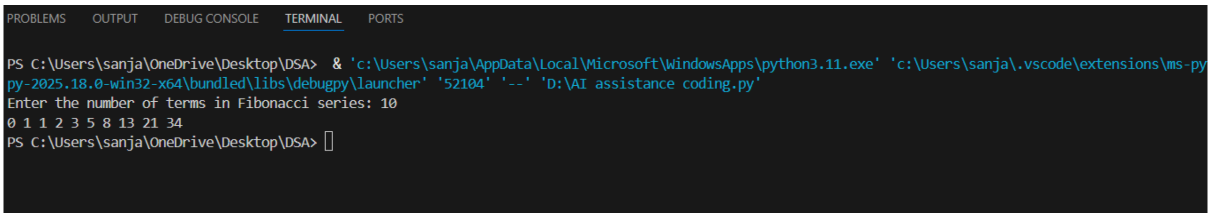
Prompt: #modular design for fibonacci series

Code:



```python
#modular design for fibonacci series
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
n = int(input("Enter the number of terms in Fibonacci series: "))
result = fibonacci_series(n)
for num in result:
    print(num, end=' ')
```

Output:



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-py
py-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA> 
```

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

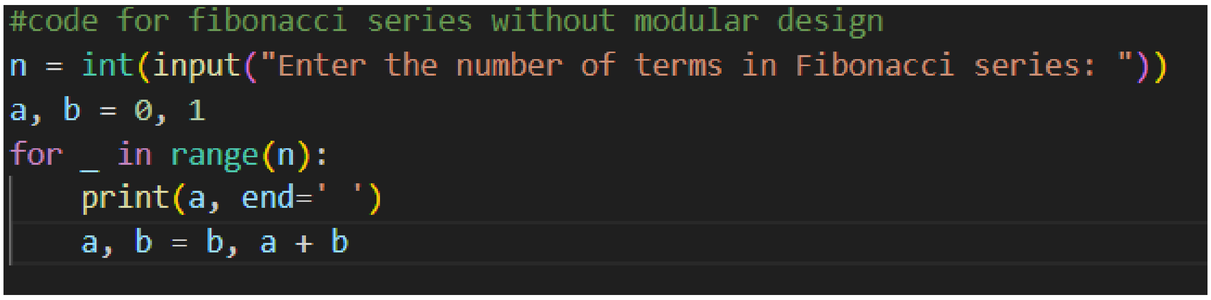
PROMPTS:

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

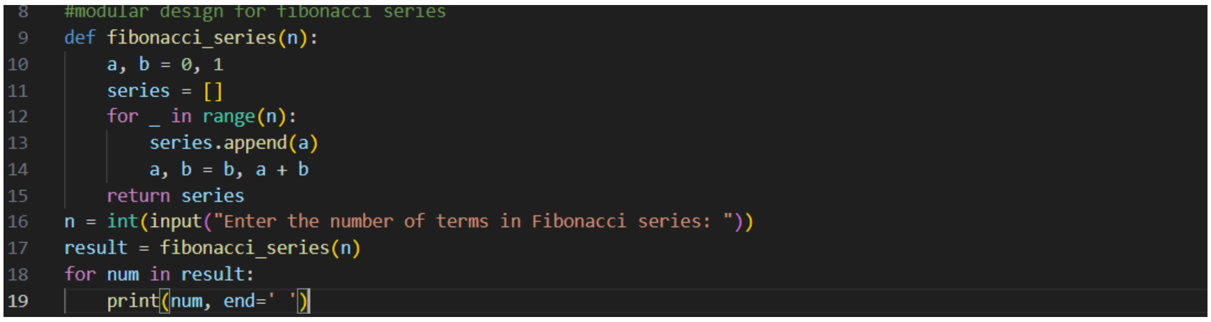
CODE:

Procedural



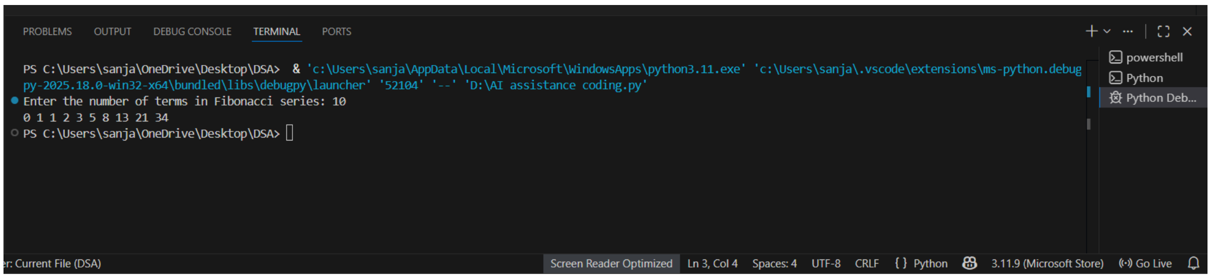
```python
#code for fibonacci series without modular design
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

MODULAR:

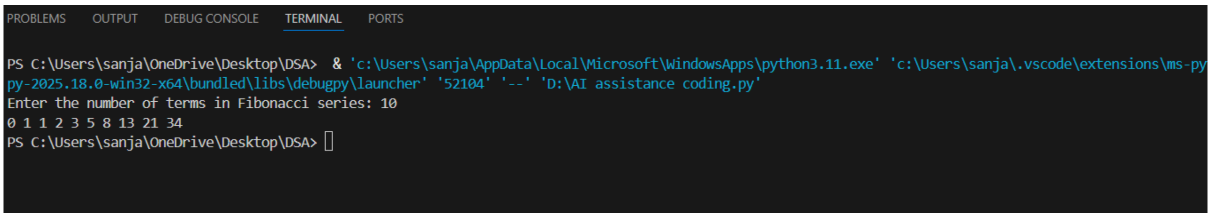


```
#modular design for fibonacci series
def fibonacci_series(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
n = int(input("Enter the number of terms in Fibonacci series: "))
result = fibonacci_series(n)
for num in result:
    print(num, end=' ')
```

Outputs:



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA>
```



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA>
```

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

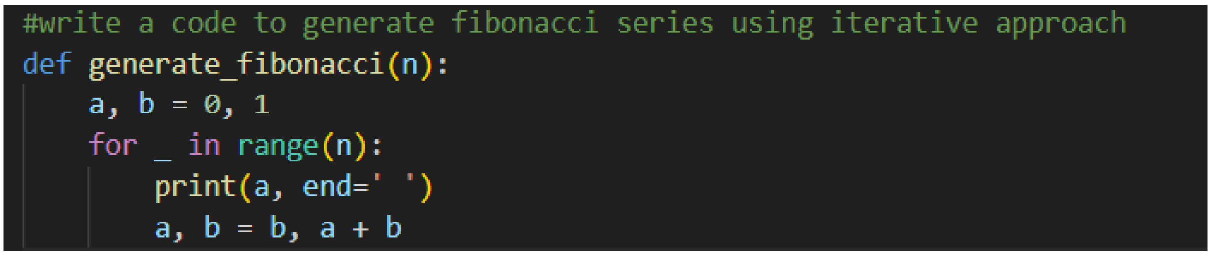
- Reusability of the code
- Easy to Debug
- Code Clarity
- Suitable for large systems

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

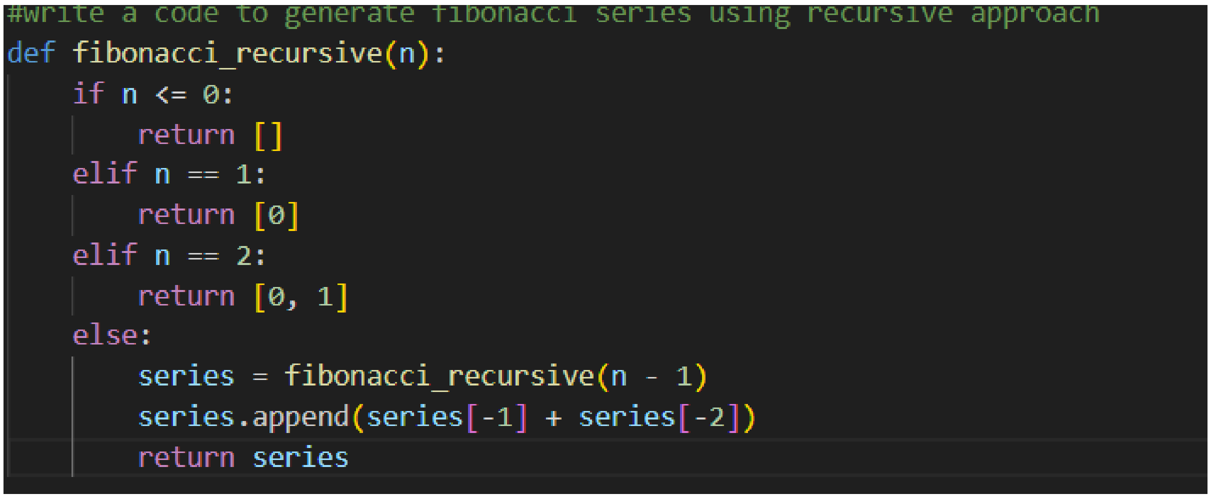
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

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:

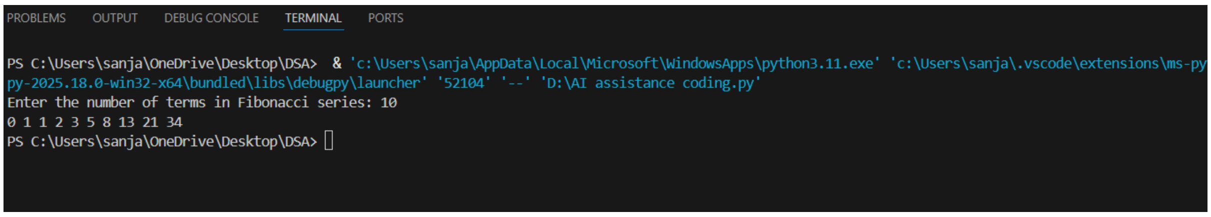


```python
#write a code to generate fibonacci series using iterative approach
def generate_fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
```

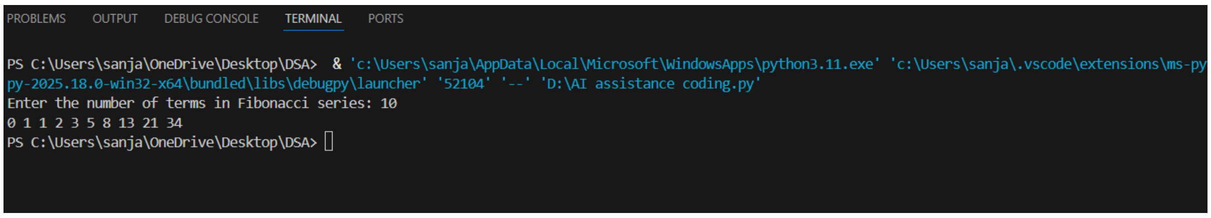


```python
#write a code to generate fibonacci series using recursive approach
def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci_recursive(n - 1)
        series.append(series[-1] + series[-2])
        return series
```

OUTPUT:



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-py
py-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA> 
```



```
PS C:\Users\sanja\OneDrive\Desktop\DSA>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-py
py-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52104' '--' 'D:\AI assistance coding.py'
Enter the number of terms in Fibonacci series: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\sanja\OneDrive\Desktop\DSA> 
```

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.