

# ASSIGNMENT-01

**Name:** Praneeth Banda

**HT.No:** 2303A51711

**Batch:** 20

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

### Scenario

You are asked to write a quick numerical sequence generator for a learning platform prototype.

**Prompt: #Fibonacci series upto n terms**

```
assign1.py < ...
assign1.py > ...
#Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)
#fibonacci series up to n terms
# Accept user input
n = int(input("Enter the number of terms: "))
# Initialize first two terms
a = 0
b = 1
# Handle edge cases
if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print(a)
else:
    print(a, b, end=" ")
    for i in range(2, n):
        c = a + b
        print(c, end=" ")
        a = b
        b = c
23
```

### Result:

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v ... | { } X
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>
```

The terminal window also shows a sidebar with two Python environments listed.

## Observation: AI-Generated Logic Without Modularization

GitHub Copilot was able to produce an accurate Fibonacci sequence using simple, direct logic without implementing user-defined functions. The program was concise and appropriate for short, one-time tasks. However, the close coupling of computation and input/output reduced the code's clarity and reusability. This activity illustrated Copilot's strength in rapidly generating working solutions while also revealing its limitations in terms of scalability and code organization.

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

### Scenario

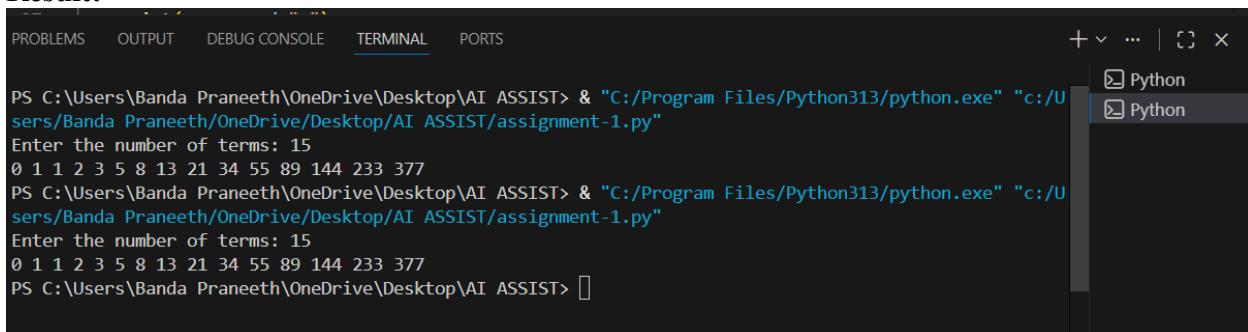
The prototype will be shared with other developers and needs optimization.

**Prompt:** Optimize this code

**Code:**

```
assign1.py > ...
#Task 2: AI Code Optimization & Cleanup (Improving Efficiency)
n = int(input("Enter the number of terms: "))
prev, curr = 0, 1
for i in range(n):
    print(prev, end=" ")
    prev, curr = curr, prev + curr
```

**Result:**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... | [] X
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> []
```

## Observation: AI Code Optimization & Cleanup

When asked to optimize the program, Copilot effectively streamlined the original code by eliminating extra variables and avoiding unnecessary conditional statements. The resulting version was more concise, easier to read, and preserved the same time complexity. This exercise demonstrated Copilot's capability to enhance existing code when provided with clear instructions focused on optimization.

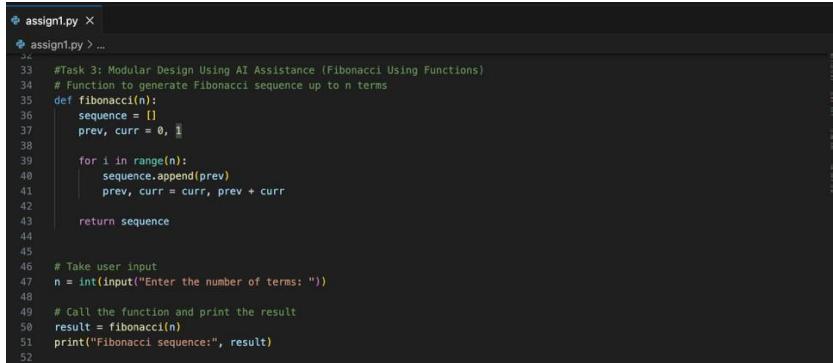
## Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)

### Scenario

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

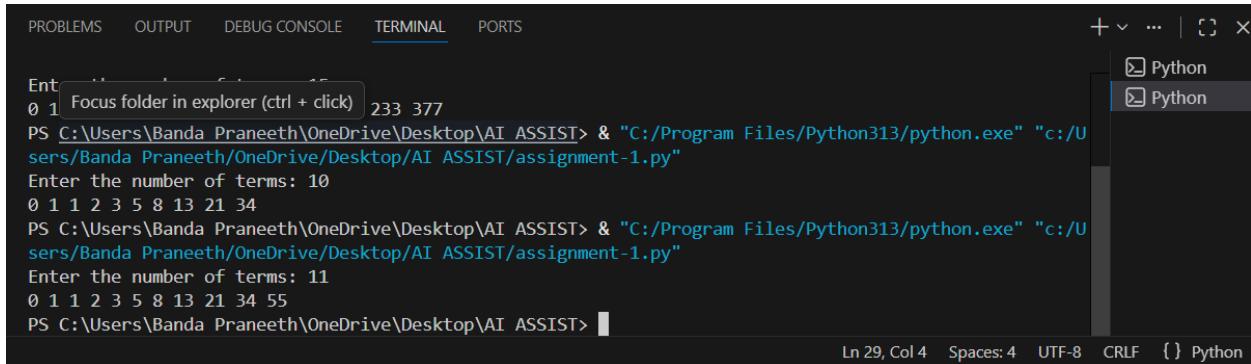
**Prompt:** Optimize this code using functions

### Code:



```
assign1.py X
assign1.py > ...
33 # Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)
34 # Function to generate Fibonacci sequence up to n terms
35 def fibonacci(n):
36     sequence = []
37     prev, curr = 0, 1
38
39     for i in range(n):
40         sequence.append(prev)
41         prev, curr = curr, prev + curr
42
43     return sequence
44
45
46 # Take user input
47 n = int(input("Enter the number of terms: "))
48
49 # Call the function and print the result
50 result = fibonacci(n)
51 print("Fibonacci sequence:", result)
52
```

### Result:



```
+ ... | x
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ent Focus folder in explorer (ctrl + click) 233 377
0 1 Focus folder in explorer (ctrl + click) 233 377
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 11
0 1 1 2 3 5 8 13 21 34 55
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> 
```

### Observation: Function-Based Fibonacci Implementation

Copilot produced a structured Fibonacci solution built around a user-defined function, along with clear and relevant comments. Encapsulating the logic inside a function enhanced the readability of the code, made it reusable, and simplified testing. This exercise highlighted Copilot's understanding of modular programming concepts and its ability to generate code that can be efficiently reused across different parts of an application

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

### **Scenario**

You are participating in a code review session

**Prompt:** Description on comparision between with functions and without function

### **Observation: Code Review Comparison**

The procedural version of the Fibonacci code is straightforward and suitable for small, one-time programs. It directly combines input handling, logic, and output in a single block, which makes it quick to write and easy to understand for beginners. However, this approach reduces readability as the program grows and makes the code difficult to reuse or test independently.

In contrast, the modular version organizes the Fibonacci logic inside a function. This separation improves clarity by isolating the core logic from input and output operations. It also enhances reusability, as the same function can be called from multiple parts of an application, and simplifies testing and maintenance. Overall, while the procedural approach works well for simple tasks, the modular approach is more scalable, maintainable, and better suited for larger applications

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

### **Scenario**

Your mentor wants to assess AI's understanding of different algorithmic paradigms.

**Prompt:** give me a code using iterative and recursive approch

**Code:**

```

#task-5
#iterative and recursive approach

def fibonacci_iterative(n):
    a, b = 0, 1
    result = []
    for i in range(n):
        result.append(a)
        a, b = b, a + b
    return result

def fibonacci_recursive(n):
    if n <= 1:
        return n
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

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

print("Iterative approach:")
print(fibonacci_iterative(n))

print("Recursive approach:")
for i in range(n):
    print(fibonacci_recursive(i), end=" ")

```

## Result:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... ×
Python Python
users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/U
users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.py"
Enter the number of terms: 15
Iterative approach:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
Recursive approach:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>

```

## Observation: Iterative vs Recursive Fibonacci Approaches

The above code demonstrates two algorithmic approaches to generate the Fibonacci sequence: iterative and recursive. The iterative approach uses a loop to compute each term efficiently with minimal memory usage and avoids repeated calculations. The recursive approach follows the mathematical definition of Fibonacci, making it simple and intuitive to understand, but it is less efficient due to repeated function calls. The iterative method is better suited for larger input sizes, while the recursive method is useful for learning and understanding recursion concepts. Selecting the appropriate approach depends on the input size and performance needs of the application.