

## Shruthi Addagudi Batch – 37

### AI-ASSISTED CODING ASSIGNMENT 1

The screenshot shows three instances of Visual Studio Code running side-by-side, demonstrating the AI-assisted coding feature for generating a Fibonacci sequence.

**Top Window:** The code editor displays a Python script named `Code1.py` with the following content:

```
1  # Write a code to print the Fibonacci sequence up to n terms
2  def fibonacci_sequence(n):
3      sequence = []
4      a, b = 0, 1
5      for _ in range(n):
6          sequence.append(a)
7          a, b = b, a + b
8      return sequence
9  n = int(input("Enter number of terms: "))
10 print(fibonacci_sequence(n))
```

The terminal below shows the output of running the code in Python 3.11:

```
PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py
Enter number of terms: 15
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
PS C:\Users\lenovo\Desktop\AIAC>
```

An AI assistance panel titled "Build with Agent" is visible on the right, with the message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."

**Middle Window:** The code editor displays the same `Code1.py` script, but with a different implementation:

```
1  # Optimise the code below to generate Fibonacci sequence up to n
2  # Original Code generates fibonacci series upto n terms
3  def fibonacci_sequence(n):
4      sequence = []
5      a, b = 0, 1
6      for _ in range(n):
7          sequence.append(a)
8          a, b = b, a + b
9      return sequence
10 n = int(input("Enter number of terms: "))
11 print(fibonacci_sequence(n))
```

The terminal output is identical to the first window.

**Bottom Window:** The code editor displays the same `Code1.py` script, with the same implementation as the middle window.

The terminal output is identical to the first window.

The image shows two identical instances of the Visual Studio Code interface side-by-side, demonstrating the transformation of a loop-based Fibonacci sequence generator into a recursive one.

**Top Window (Original Loop-Based Code):**

```
1 # Optimise the code below to generate Fibonacci sequence up to n terms
2 # Original Code generates fibonacci series upto n terms
3 # Simplify the logic without using function definition
4 n = int(input("Enter number of terms: "))
5 sequence = []
6 a, b = 0, 1
7 for _ in range(n):
8     sequence.append(a)
9     a, b = b, a + b
10 print(sequence)
```

**Bottom Window (Optimized Recursive Function):**

```
1 # Optimise the code below to generate Fibonacci sequence up to n terms
2 # Original Code generates fibonacci series upto n terms
3 # Simplify the logic without using function definition
4 # Fibonacci code using user defined function
5 def fibonacci_sequence(n):
6     sequence = []
7     a, b = 0, 1
8     for _ in range(n):
9         sequence.append(a)
10        a, b = b, a + b
11    return sequence
12 n = int(input("Enter number of terms: "))
13 print(fibonacci_sequence(n))
14 |
```

Both windows show the same terminal output:

```
PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py
Enter number of terms: 15
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
PS C:\Users\lenovo\Desktop\AIAC> []
```

The interface includes standard VS Code features like the Explorer, Welcome, Chat, Problems, Output, Terminal, and Python extension bar. A "Build with Agent" sidebar is visible in both windows, with the message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."

```

1 # Optimise the code below to generate Fibonacci sequence up to n
2 # Original Code generates fibonacci series upto n terms
3 # Simplify the logic without using function definition
4 # Fibonacci code using user defined function
5 # An iterative Fibonacci implementation
6
7 n = int(input("Enter number of terms: "))
8 sequence = []
9 a, b = 0, 1
10 for _ in range(n):
11     sequence.append(a)
12     a, b = b, a + b
13 print(sequence)

PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py
Enter number of terms: 15
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
PS C:\Users\lenovo\Desktop\AIAC> []

----- Bottom Implementation -----

1 # Optimise the code below to generate Fibonacci sequence up to n
2 # Original Code generates fibonacci series upto n terms
3 # Simplify the logic without using function definition
4 # Fibonacci code using user defined function
5 # A recursive Fibonacci implementation
6
7 def fibonacci_recursive(n):
8     if n <= 1:
9         return n
10    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
11
12 n = int(input("Enter number of terms: "))
13 for i in range(n):
14     print(fibonacci_recursive(i), end=" ")
15

PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py
Enter number of terms: 15
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
PS C:\Users\lenovo\Desktop\AIAC>

```

## Approach

**Iterative Fibonacci**

## Time Complexity Space Complexity

**O(1)**

**Recursive Fibonacci**

**O(n)**

## Performance for Large n

### Aspect

**Iterative**

**Recursive**

**Execution speed**

**Very fast**

**Very slow**

Aspect	Iterative	Recursive
Memory usage	Minimal	High (call stack)
Scalability	Excellent	Poor
Risk of crash	None	Stack overflow

**Conclusion:**

- Iterative Fibonacci works efficiently even for large values like  $n = 10000$ .
- Recursive Fibonacci becomes extremely slow and may crash for values above  $n = 40$ .