

# AI-ASSISTED CODING ASSIGNMENT – 1

Thumma Hasini | 2303A52076 | Batch-37

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a file named "Code1.py" under the "AIAC" folder.
- Code Editor:** Displays Python code for generating a Fibonacci sequence up to n terms. The code is as follows:

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

- Terminal:** Shows the command PS C:\Users\lenovo\Desktop\AIAC & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py followed by the output: Enter number of terms: 15 [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
- Right Panel:** Features a "Build with Agent" section with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also includes a "Describe what to build next" input field and dropdowns for "Agent" and "Auto".

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a file named "Code1.py" under the "AIAC" folder.
- Code Editor:** Displays Python code for generating a Fibonacci sequence up to n terms, with an additional line 11. The code is as follows:

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

- Terminal:** Shows the same command and output as the first screenshot: 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]
- Right Panel:** Features a "Build with Agent" section with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also includes a "Describe what to build next" input field and dropdowns for "Agent" and "Auto".

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AIAC" containing a file "Code1.py".
- Code Editor:** Displays Python code to generate a Fibonacci sequence up to n terms. The code uses a loop to calculate the sequence.
- Terminal:** Shows the command PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py being run, followed by the output: Enter number of terms: 15 [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377].
- AI Assistant Panel:** A "Build with Agent" panel is open, featuring a speech bubble icon and the text "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." Below this is a text input field labeled "Describe what to build next" and dropdown menus for "Agent" and "Auto".

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AIAC" containing a file "Code1.py".
- Code Editor:** Displays Python code using a user-defined function "fibonacci\_sequence" to generate the Fibonacci sequence up to n terms.
- Terminal:** Shows the command PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py being run, followed by the output: Enter number of terms: 15 [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377].
- AI Assistant Panel:** A "Build with Agent" panel is open, featuring a speech bubble icon and the text "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." Below this is a text input field labeled "Describe what to build next" and dropdown menus for "Agent" and "Auto".

The image displays two instances of Visual Studio Code (VS Code) running side-by-side. Both instances have the same interface elements: a top bar with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', etc., a title bar 'Code1.py - AIAC - Visual Studio Code', a left sidebar with 'EXPLORER' and 'AIAC' sections, and a main editor area.

**Left Editor (Iterative Implementation):**

```
# Optimise the code below to generate Fibonacci sequence up to n
# Original Code generates fibonacci series upto n terms
# Simplify the logic without using function definition
# Fibonacci code using user defined function
# An iterative Fibonacci implementation

n = int(input("Enter number of terms: "))
sequence = []
a, b = 0, 1
for _ in range(n):
    sequence.append(a)
    a, b = b, a + b
print(sequence)
```

**Right Editor (Recursive Implementation):**

```
# Optimise the code below to generate Fibonacci sequence up to n
# Original Code generates fibonacci series upto n terms
# Simplify the logic without using function definition
# Fibonacci code using user defined function
# An iterative Fibonacci implementation
# A recursive Fibonacci implementation
def fibonacci_recursive(n):
    if n <= 1:
        return n
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

n = int(input("Enter number of terms: "))
for i in range(n):
    print(fibonacci_recursive(i), end=" ")
```

**Terminal Output:**

Both terminals show the command PS C:\Users\lenovo\Desktop\AIAC> & C:/Users/lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/lenovo/Desktop/AIAC/Code1.py followed by the prompt 'Enter number of terms: 15'. The output shows the generated Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377].

### Time and Space Complexity Comparison

Approach	Time Complexity	Space Complexity
Iterative Fibonacci	$O(n)$	$O(1)$
Recursive Fibonacci	$O(2^n)$	$O(n)$

### Performance for Large n

Aspect	Iterative	Recursive
Execution speed	Very fast	Very slow
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  $n = 40$ .