

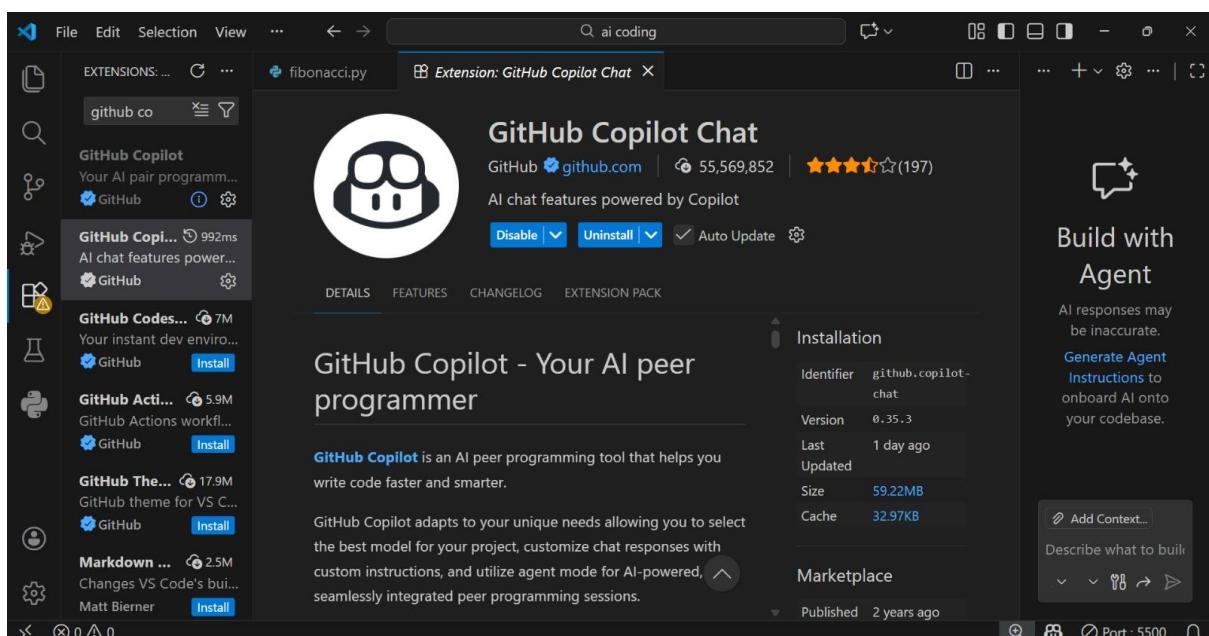
# Course Title: AI-Assisted Coding

Batch – 05

Hall no. – 2303A51305

**Question:** Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

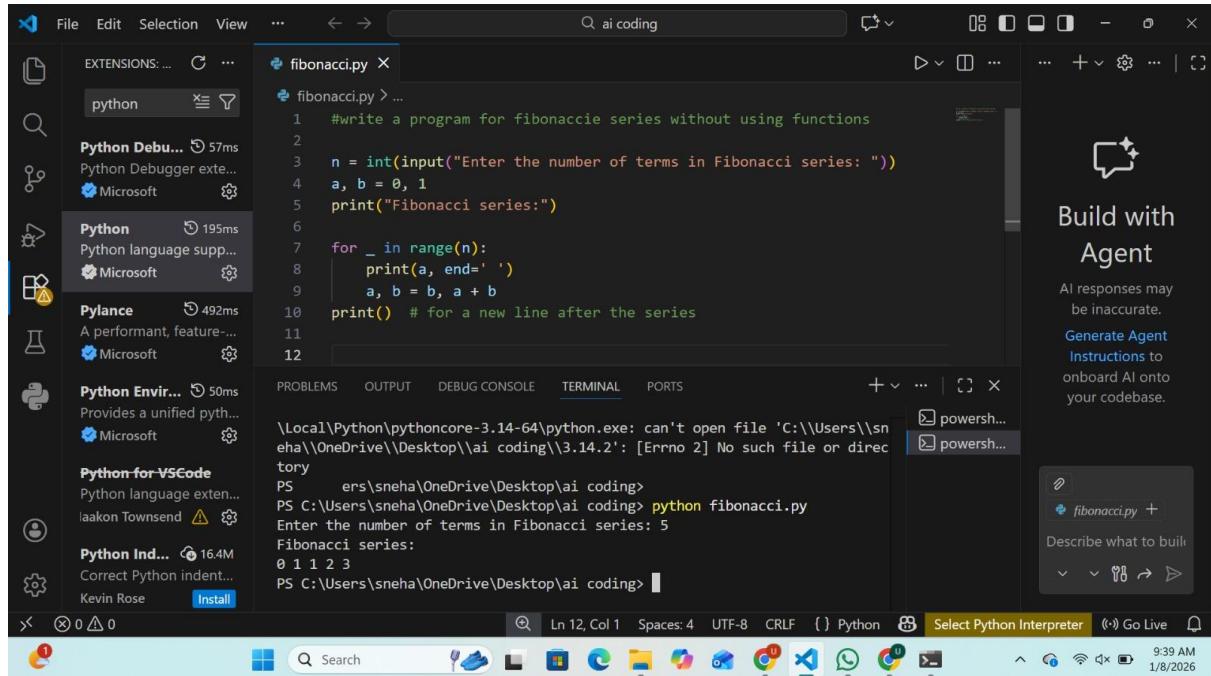
**Task 0 ● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.**



**Explanation:** I installed GitHub Copilot in VS Code using the Extensions option. Then I signed in with my GitHub account and allowed permissions. Copilot started giving code suggestions while typing, which made coding easier.

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

Input :



A screenshot of the Visual Studio Code interface. The left sidebar shows extensions like Python, Pylance, and Python for VS Code. The main editor window contains a Python script named fibonacci.py with the following code:

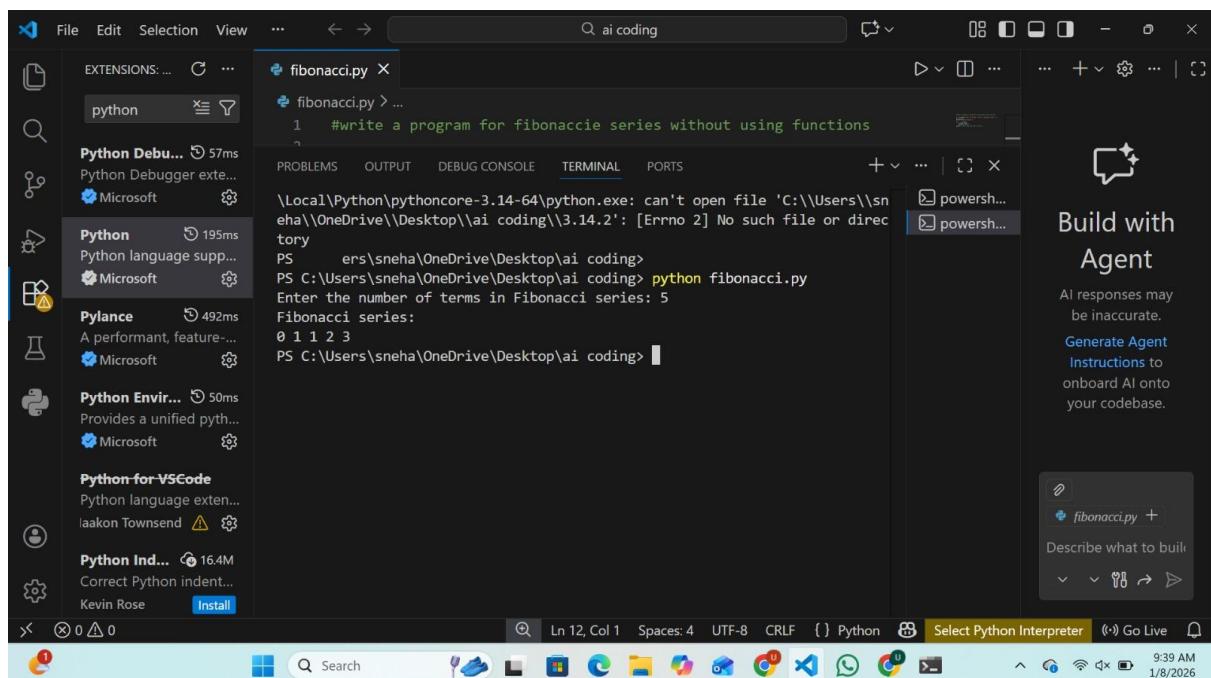
```
#write a program for fibonacci series without using functions
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("fibonacci series:")
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
print() # for a new line after the series
```

The terminal tab shows the output of running the script:

```
\Local\Python\pythoncore-3.14-64\python.exe: can't open file 'C:\\Users\\sneha\\OneDrive\\Desktop\\ai coding\\3.14.2': [Errno 2] No such file or directory
PS C:\Users\sneha\OneDrive\Desktop\ai coding> python fibonacci.py
Enter the number of terms in Fibonacci series: 5
Fibonacci series:
0 1 1 2 3
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

The status bar at the bottom indicates the file is 12 lines long, has 4 spaces, and is in UTF-8 encoding. The date is 1/8/2026.

Output :



A screenshot of the Visual Studio Code interface, identical to the previous one but with AI suggestions. The right sidebar features a "Build with Agent" section with a speech bubble icon and the text "AI responses may be inaccurate." Below it is a "Generate Agent Instructions to onboard AI onto your codebase." button. A tooltip for the "fibonacci.py +" button says "Describe what to build". The rest of the interface is the same, showing the Python script and its execution in the terminal.

**Explanation:** The Fibonacci code is written in one place. No functions are used in this program. The code works, but it looks messy.

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

Input :

The screenshot shows the VS Code interface with the file 'fibonacci.py' open. The code implements memoization for calculating Fibonacci numbers. A sidebar on the right displays AI-related features like 'Build with Agent' and 'Generate Agent Instructions'. The status bar at the bottom indicates the file is base (3.13.5) and port 5500.

```
#write a code for Ai Code Optimization & Cleanup (Improving Efficiency)
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n
    memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)
    return memo[n]

# Example usage
if __name__ == "__main__":
    num = 10
    print(f"fibonacci of {num} is {fibonacci(num)}")
# This function calculates the nth Fibonacci number using memoization for efficiency.
# The memo dictionary stores previously computed Fibonacci numbers to avoid redundant calculations.
```

Output :

The screenshot shows the VS Code interface with the terminal tab active. The output shows the AI has removed unnecessary code from the Fibonacci function, resulting in a much shorter and cleaner implementation. The status bar at the bottom indicates the file is base (3.13.5) and port 5500.

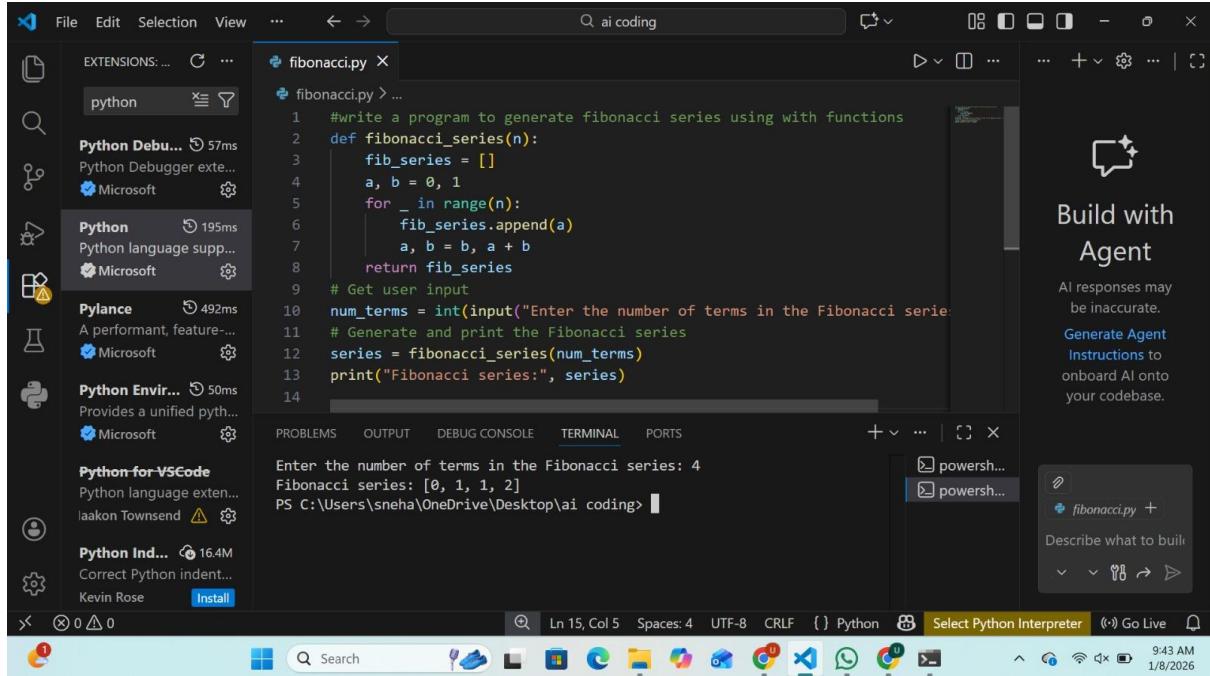
```
Fibonacci of 10 is 55
Fibonacci of 10 is 55
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

**Explanation :** AI removed extra and useless code. The program became short and clean.

Now it is easy to understand.

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

Input :

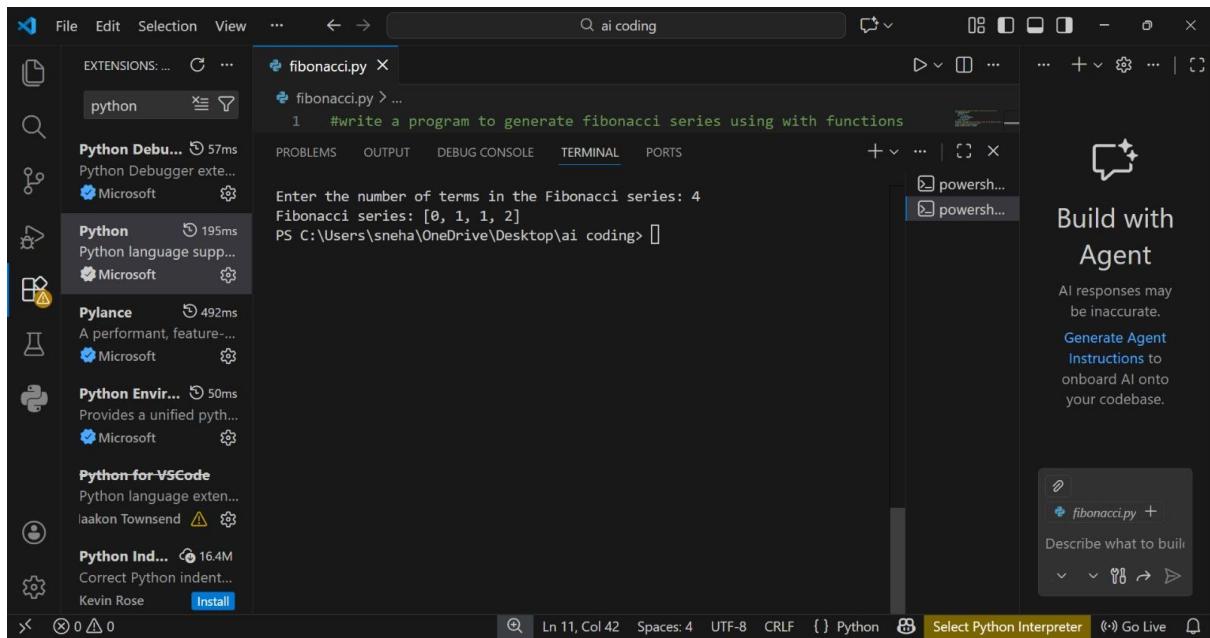


```
fibonacci.py
1 #write a program to generate fibonacci series using with functions
2 def fibonacci_series(n):
3     fib_series = []
4     a, b = 0, 1
5     for _ in range(n):
6         fib_series.append(a)
7         a, b = b, a + b
8     return fib_series
9
10 # Get user input
11 num_terms = int(input("Enter the number of terms in the Fibonacci series"))
12 # Generate and print the Fibonacci series
13 series = fibonacci_series(num_terms)
14 print("Fibonacci series:", series)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

Enter the number of terms in the Fibonacci series: 4  
Fibonacci series: [0, 1, 1, 2]

Output :



```
fibonacci.py
1 #write a program to generate fibonacci series using with functions
2 def fibonacci_series(n):
3     fib_series = []
4     a, b = 0, 1
5     for _ in range(n):
6         fib_series.append(a)
7         a, b = b, a + b
8     return fib_series
9
10 # Get user input
11 num_terms = int(input("Enter the number of terms in the Fibonacci series"))
12 # Generate and print the Fibonacci series
13 series = fibonacci_series(num_terms)
14 print("Fibonacci series:", series)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

Enter the number of terms in the Fibonacci series: 4  
Fibonacci series: [0, 1, 1, 2]

Explanation : The code is written using a function. This makes the program neat.

The function can be reused.

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

Input :

The screenshot shows the VS Code interface with the file 'fibonacci.py' open. The code contains two functions: 'fibonacci\_procedural(n)' and 'fibonacci\_modular(n)'. The 'fibonacci\_procedural' function uses a loop to calculate the sequence. The 'fibonacci\_modular' function uses a helper function and handles edge cases for n=0 and n=1.

```
#write a program for comparative analysis - procedural vs modular fibonacci
# Procedural approach to calculate Fibonacci numbers
def fibonacci_procedural(n):
    a, b = 0, 1
    fib_sequence = []
    for _ in range(n):
        fib_sequence.append(a)
        a, b = b, a + b
    return fib_sequence

# Modular approach to calculate Fibonacci numbers

def fibonacci_modular(n):
    def fib_helper(n):
        if n == 0:
            return 0
        elif n == 1:
            return 1
        else:
```

This screenshot is identical to the one above, showing the same code in 'fibonacci.py' and its execution output in the terminal. Both the procedural and modular approaches produce the same sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34].

```
Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

Output :

```
fibonacci.py X
fibonacci > ...
13 def fibonacci_modular(n):
14     def fib_helper(n):
15         if n <= 0:
16             return 0
17         elif n == 1:
18             return 1
19         else:
20             return fib_helper(n - 1) + fib_helper(n - 2)
21
22     fib_sequence = []
23     for i in range(n):
24         fib_sequence.append(fib_helper(i))
25     return fib_sequence
26
27 # Example usage
28 n = 10
29 print("Procedural Fibonacci:", fibonacci_procedural(n))
30 print("Modular Fibonacci:", fibonacci_modular(n))
31
32
33
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

Ln 32, Col 9 Spaces: 4 UTF-8 CRLF {} Python Select Python Interpreter ⓘ Go Live

Explanation ; Procedural code is written in one block. Modular code uses functions.

Modular code is better and clearer.

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

Input :

```
fibonacci.py X
fibonacci > ...
1 # write a code for Ai generated iterative vs recursive fibonacci approach
2 def fibonacci_recursive(n):
3     if n <= 0:
4         return 0
5     elif n == 1:
6         return 1
7     else:
8         return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
9 def fibonacci_iterative(n):
10    if n <= 0:
11        return 0
12    elif n == 1:
13        return 1
14    else:
15        a, b = 0, 1
16        for _ in range(2, n + 1):
17            a, b = b, a + b
18        return b
19
20 # Example usage:
21 n = 10
22 print("Recursive Fibonacci of", n, "is:", fibonacci_recursive(n))
23 print("Iterative Fibonacci of", n, "is:", fibonacci_iterative(n))
```

Ln 23, Col 1 Spaces: 4 UTF-8 CRLF {} Python base (3.13.5) Port: 5500

## Output :

The screenshot shows a dark-themed instance of Visual Studio Code. In the center, there is a code editor window titled "fibonacci.py" containing Python code. The code defines two functions: "fibonacci\_recursive" and "fibonacci\_iterative". The "fibonacci\_recursive" function uses recursion, while the "fibonacci\_iterative" function uses a loop. Both functions handle base cases (n=0 and n=1) and calculate the result by summing the previous two numbers in the sequence. Below the code editor, the status bar displays file information like "Ln 23, Col 1" and settings like "Spaces: 4", "UTF-8", and "Python". To the left of the code editor is a sidebar with various extensions listed under the "python" category, including "Python Debug", "Python", "Pylance", "Python Envir...", "Python-for-VSCode", "Python Ind...", and "Python Indent...". On the right side of the interface, there is a "Build with Agent" panel with a message about AI responses being inaccurate and instructions to onboard AI onto the codebase.

```
# write a code for Ai generated iterative vs recursive fibonacci approach
def fibonacci_recursive(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

def fibonacci_iterative(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b

# Example usage:
n = 10
print("Recursive Fibonacci of", n, "is:", fibonacci_recursive(n))
print("Iterative Fibonacci of", n, "is:", fibonacci_iterative(n))
```

## Explanation :

**Iterative method uses a loop. Recursive method calls itself. The loop method is faster.**