

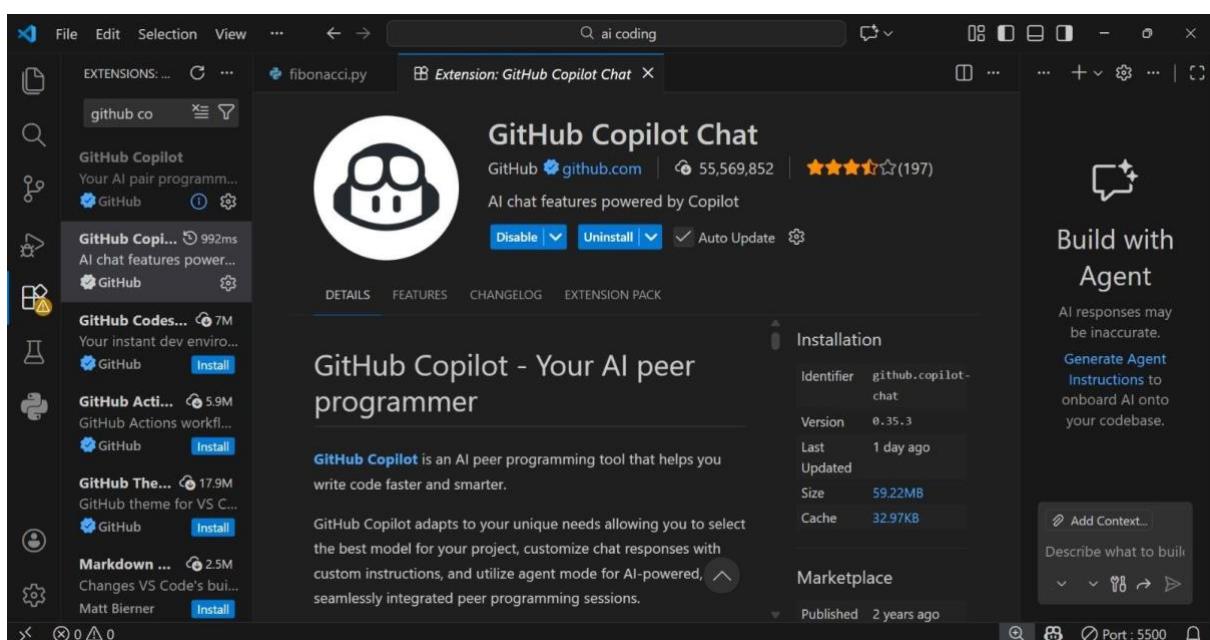
Course Title: AI-Assisted Coding

Batch – 06

Hall no. – 2303A510C1

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 window shows the execution of the script and its output:

```
\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>
```

Output :

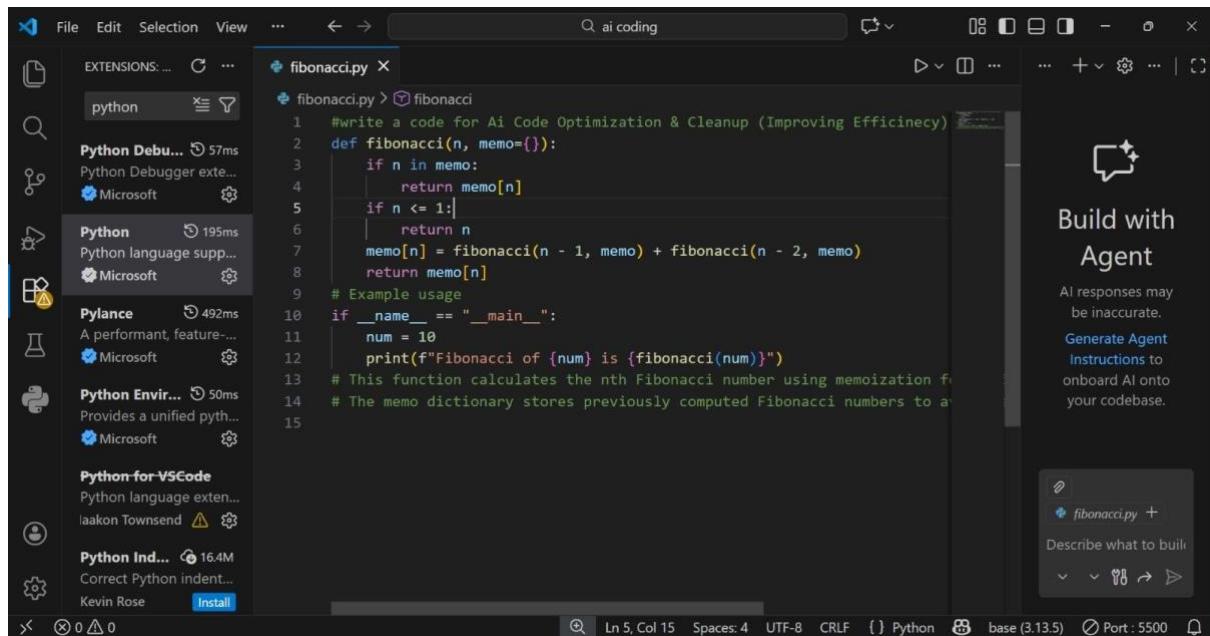
A screenshot of the Visual Studio Code interface, identical to the previous one but with a different terminal output. The terminal shows the script was run successfully and produced the expected output.

```
\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>
```

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 :



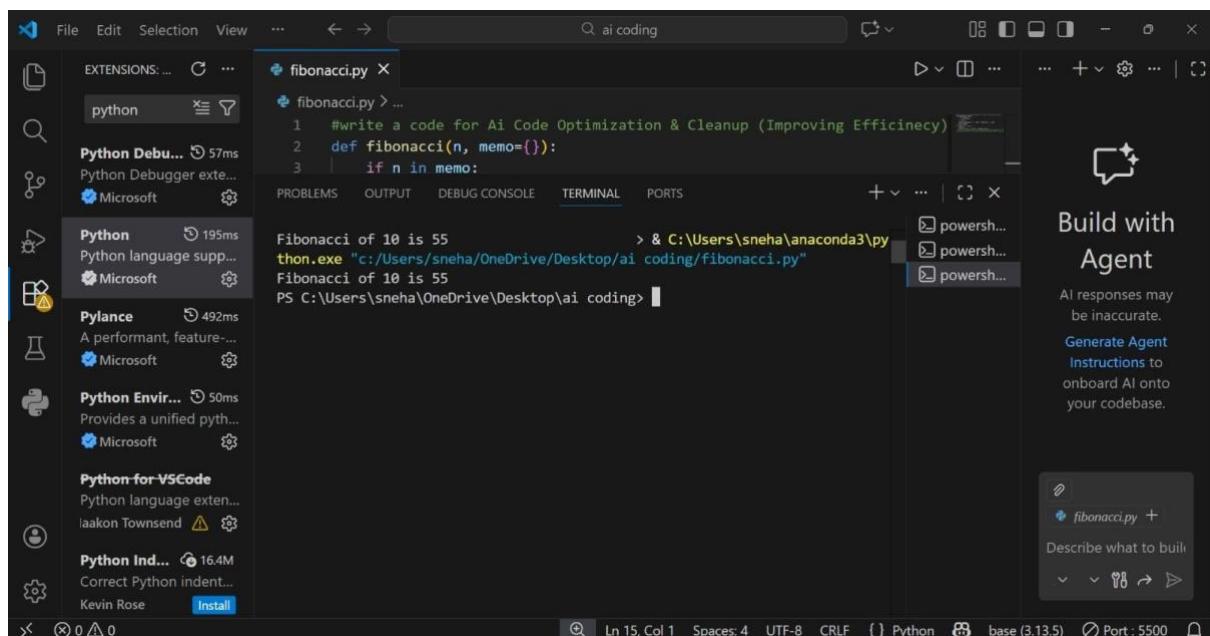
A screenshot of the Visual Studio Code interface. The left sidebar shows extensions like Python, Pylance, and Python for VSCode installed. The main editor window displays the following Python code:

```
#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
# The memo dictionary stores previously computed Fibonacci numbers to a
```

The status bar at the bottom indicates the code is base (3.13.5) and port 5500.

Output :



A screenshot of the Visual Studio Code interface, similar to the input screen but with the terminal tab active. The terminal window shows the command being run and the resulting output:

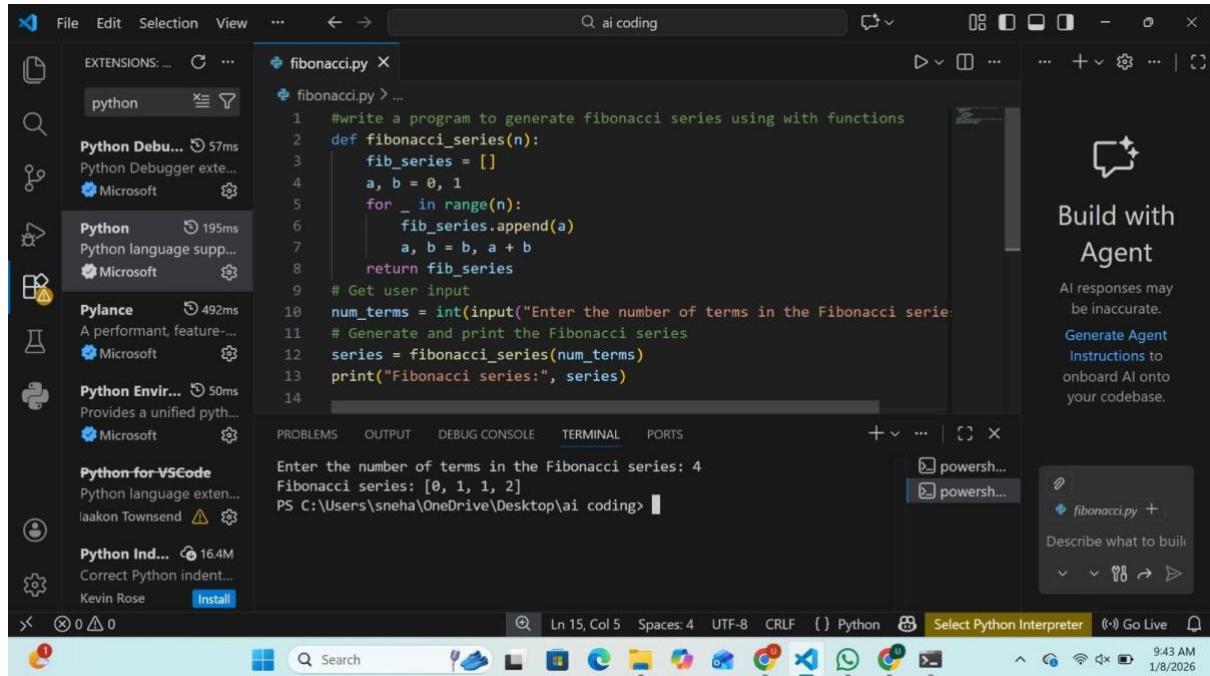
```
Fibonacci of 10 is 55
> & C:\Users\sneha\anaconda3\python.exe "c:/Users/sneha/OneDrive/Desktop/ai coding/fibonacci.py"
Fibonacci of 10 is 55
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

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)

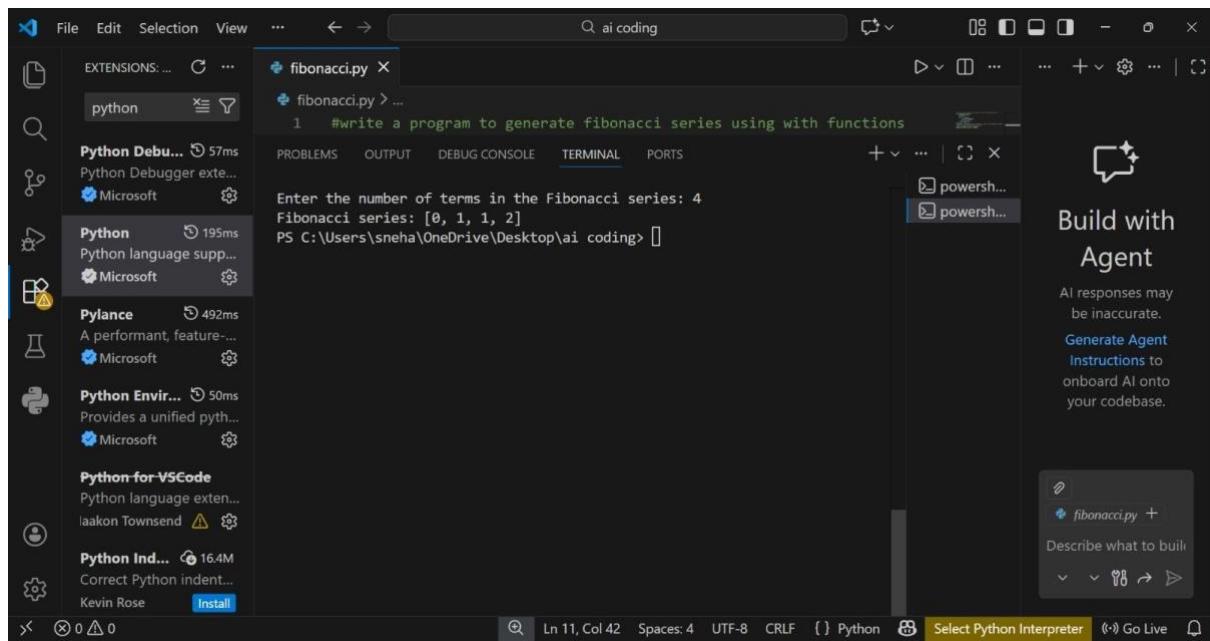
Input :



```
#write a program to generate fibonacci series using with functions
def fibonacci_series(n):
    fib_series = []
    a, b = 0, 1
    for _ in range(n):
        fib_series.append(a)
        a, b = b, a + b
    return fib_series

# Get user input
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
# Generate and print the Fibonacci series
series = fibonacci_series(num_terms)
print("Fibonacci series:", series)
```

Output :



```
#write a program to generate fibonacci series using with functions
def fibonacci_series(n):
    fib_series = []
    a, b = 0, 1
    for _ in range(n):
        fib_series.append(a)
        a, b = b, a + b
    return fib_series

# Get user input
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
# Generate and print the Fibonacci series
series = fibonacci_series(num_terms)
print("Fibonacci series:", series)
```

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 Visual Studio Code interface with the 'fibo' extension installed. The left sidebar displays various Python extensions, with 'fibo' highlighted. The main editor area contains two functions: 'fibonacci_procedural' and 'fibonacci_modular'. The terminal at the bottom shows the output of the modular Fibonacci function.

```
#write a program for comparative analysis - procedural vs modular fibonacci numbers
# 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:
Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

This screenshot is identical to the one above, showing the 'fibo' extension installed in VS Code and the execution of the 'fibo' script. The terminal output remains the same, displaying the modular Fibonacci sequence up to 34.

```
#write a program for comparative analysis - procedural vs modular fibonacci numbers
# 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:
Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

Output :

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

```
def fibonacci_modular(n):
    def fib_helper(n):
        if n <= 0:
            return 0
        elif n == 1:
            return 1
        else:
            return fib_helper(n - 1) + fib_helper(n - 2)
    fib_sequence = []
    for i in range(n):
        fib_sequence.append(fib_helper(i))
    return fib_sequence

# Example usage
n = 10
print("Procedural Fibonacci:", fibonacci_procedural(n))
print("Modular Fibonacci:", fibonacci_modular(n))
```

The terminal at the bottom shows the output: "Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]". The status bar indicates the file is saved and the Python interpreter is selected.

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 :

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

```
# 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))
```

The terminal at the bottom shows the output: "Recursive Fibonacci of 10 is: 55" and "Iterative Fibonacci of 10 is: 55". The status bar indicates the file is saved and the Python interpreter is selected.

Output :

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows extensions installed: Python (195ms), Python Debug (57ms), Pylance (492ms), Python Envir... (50ms), Python for VSCode, Python Ind... (16.4M).
- Search Bar:** ai coding
- Code Editor:** fibonacci.py (selected)
- Code Content:**

```
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 # Example usage:
20 n = 10
21 print("Recursive Fibonacci of", n, "is:", fibonacci_recursive(n))
22 print("Iterative Fibonacci of", n, "is:", fibonacci_iterative(n))
23
```
- Right Panel:**
 - Build with Agent:** AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.
 - Build Status:** fibonacci.py +
 - Description:** Describe what to build
- Bottom Status Bar:** Ln 23, Col 1, Spaces: 4, UTF-8, CRLF, Python, base (3.13.5), Port: 5500

Explanation :

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