

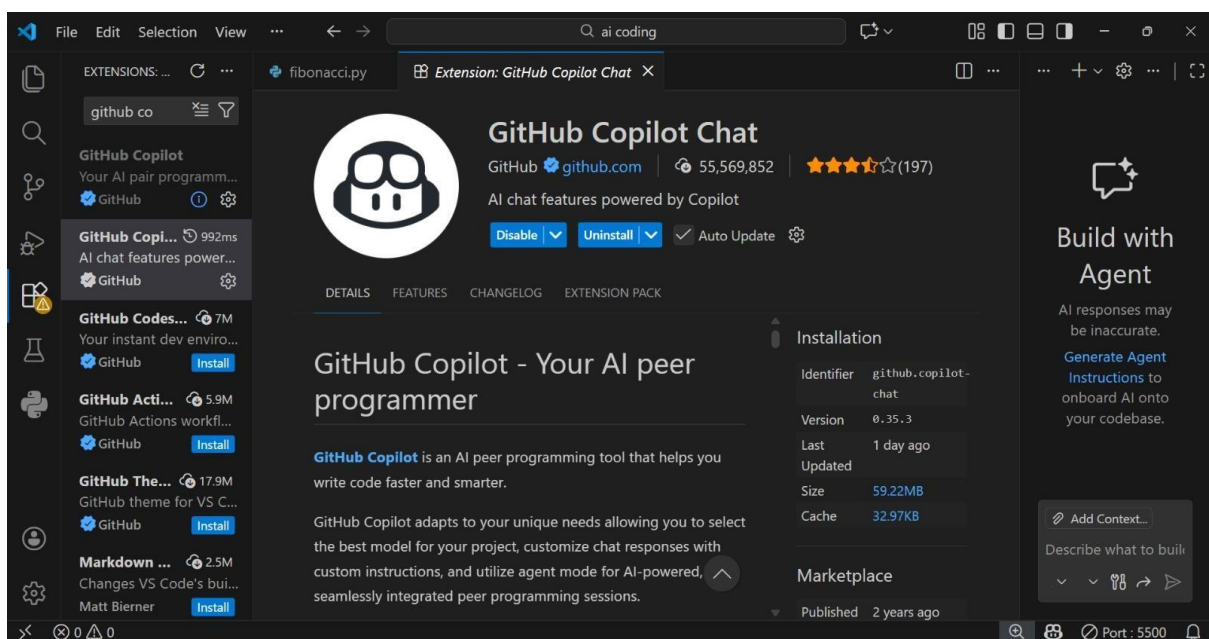
Course Title: AI-Assisted Coding

Batch – 05

Hall no. – 2303A51288

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

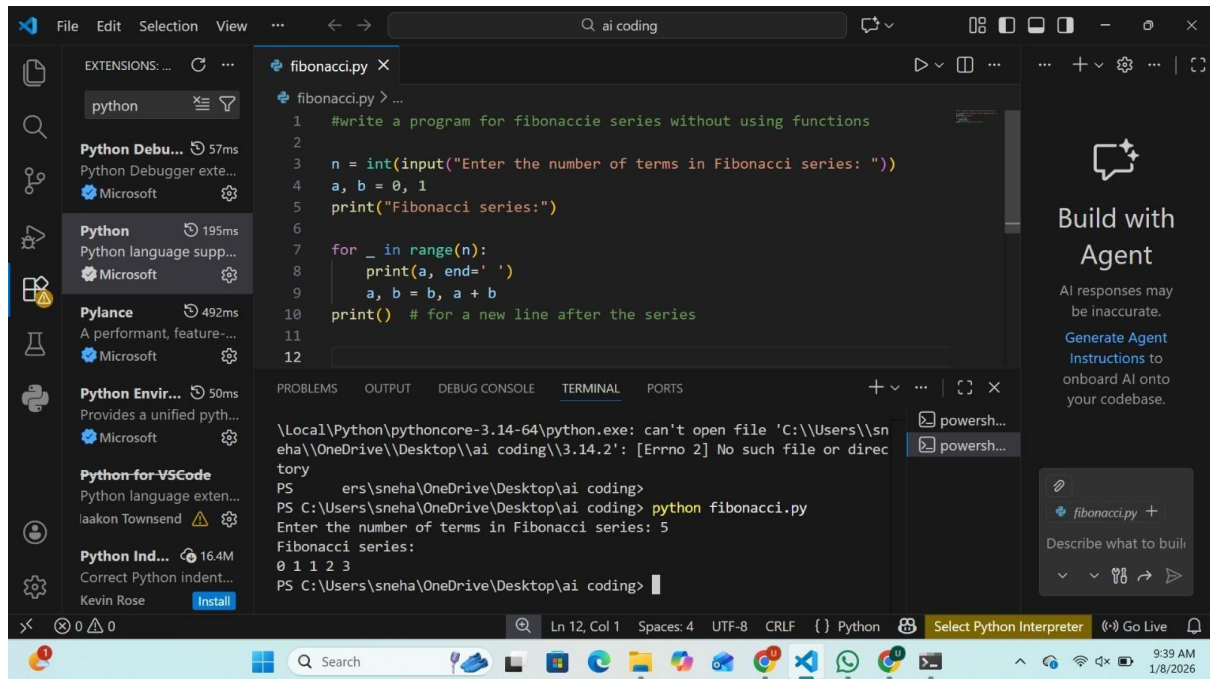
Task 0 ● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



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 :



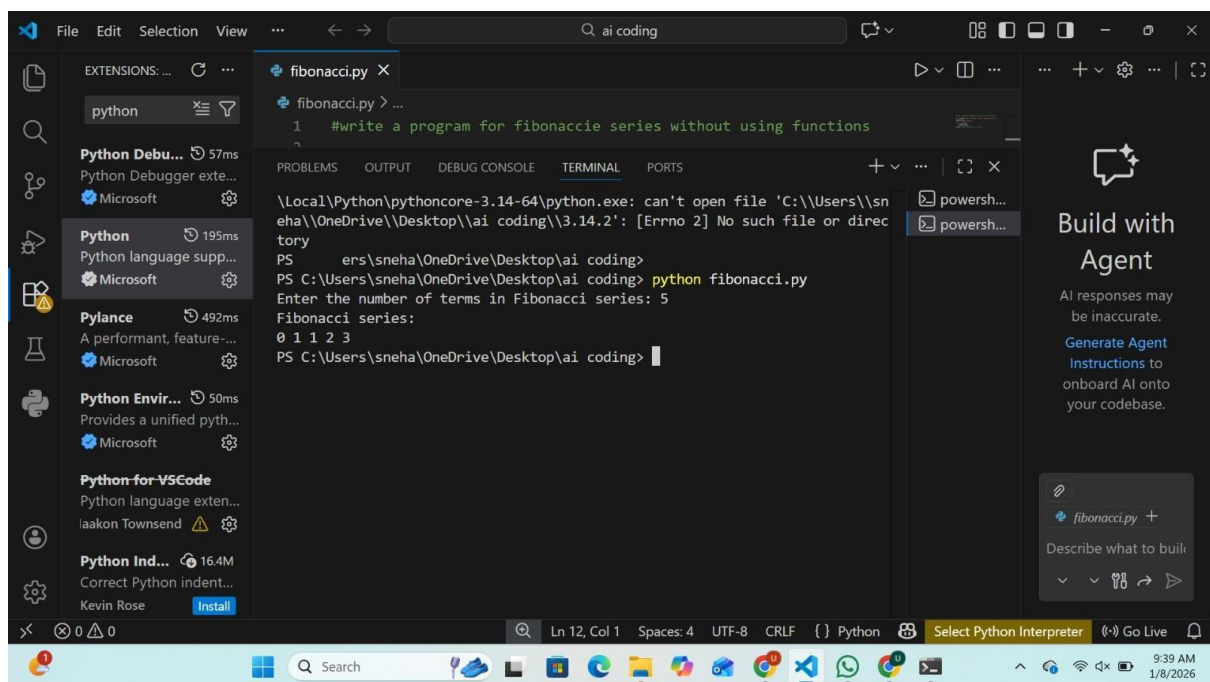
The screenshot shows the Visual Studio Code interface. The left sidebar displays the 'EXTENSIONS' view with a search for 'python'. The main editor window shows a file named 'fibonacci.py' with the following code:

```
1 #write a program for fibonacci series without using functions
2
3 n = int(input("Enter the number of terms in Fibonacci series: "))
4 a, b = 0, 1
5 print("Fibonacci series:")
6
7 for _ in range(n):
8     print(a, end=' ')
9     a, b = b, a + b
10 print() # for a new line after the series
11
12
```

The bottom panel shows the 'TERMINAL' view with the following output:

```
\Local\Python\pythoncore-3.14-64\python.exe: can't open file 'C:\Users\sneha\OneDrive\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 :

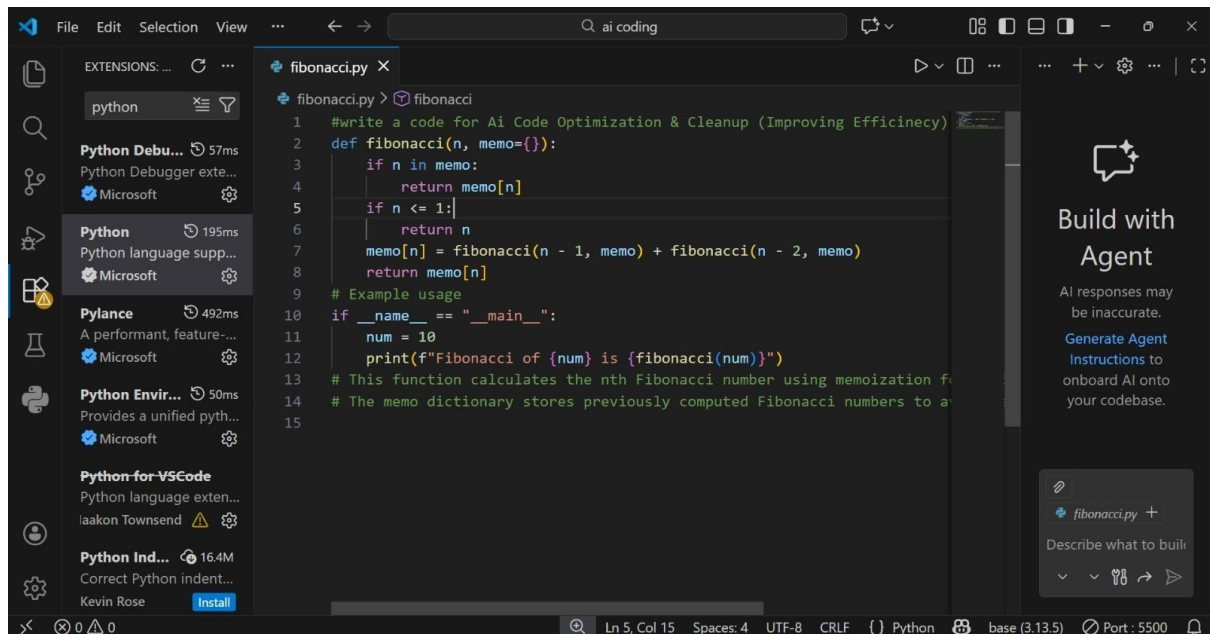


This screenshot is identical to the one above, showing the same VS Code interface with the 'fibonacci.py' file and the terminal output. The code in the editor is the same, and the terminal shows the same execution results.

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

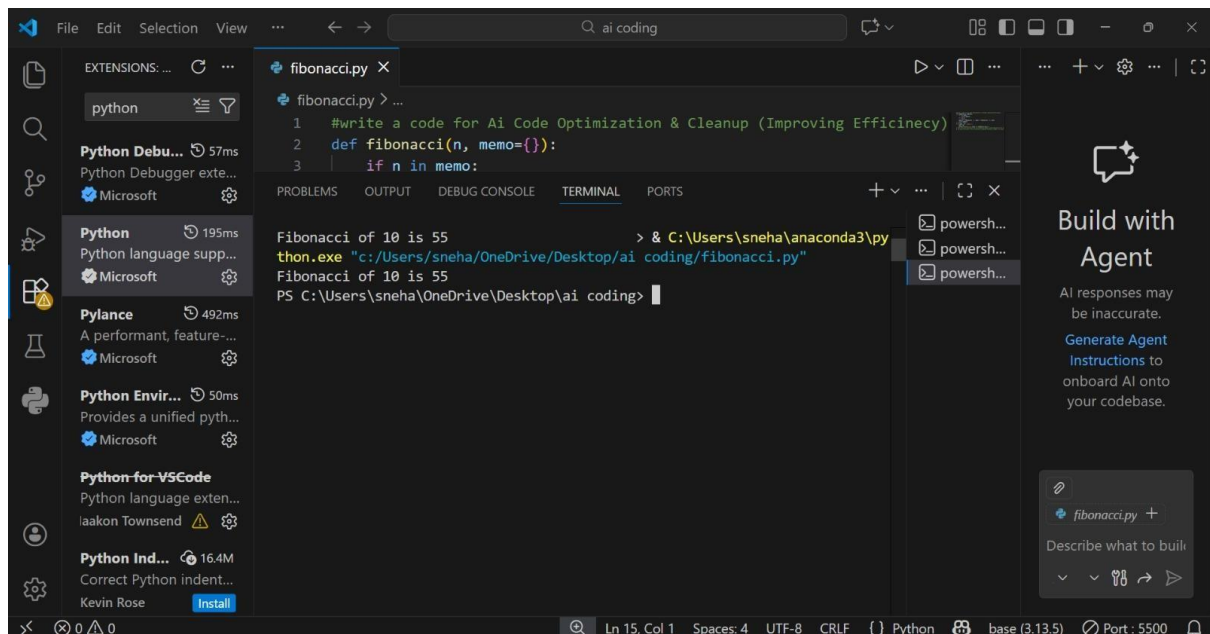
Task 2: AI-Code Optimisation & Cleanup (Improving Efficiency)

Input :



```
1 #write a code for Ai Code Optimization & Cleanup (Improving Efficiency)
2 def fibonacci(n, memo={}):
3     if n in memo:
4         return memo[n]
5     if n <= 1:
6         return n
7     memo[n] = fibonacci(n - 1, memo) + fibonacci(n - 2, memo)
8     return memo[n]
9 # Example usage
10 if __name__ == "__main__":
11     num = 10
12     print(f"Fibonacci of {num} is {fibonacci(num)}")
13 # This function calculates the nth Fibonacci number using memoization for
14 # The memo dictionary stores previously computed Fibonacci numbers to a
```

Output :



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

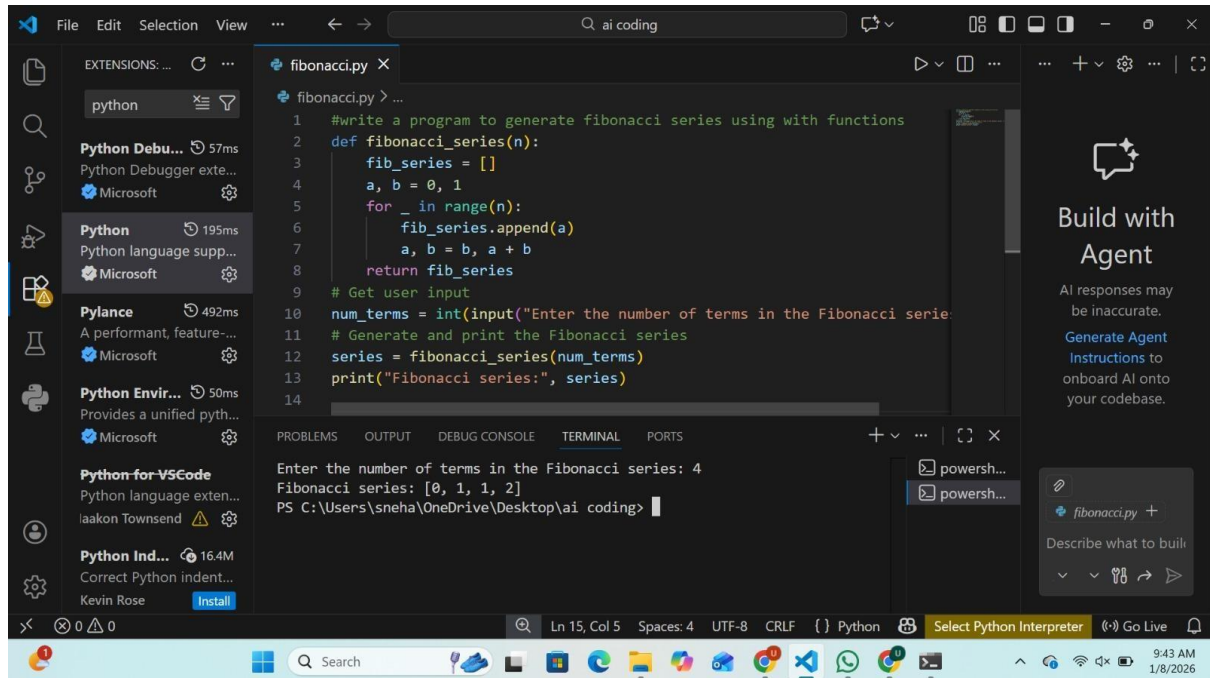
```
Fibonacci of 10 is 55
> & C:\Users\sneha\anaconda3\python.exe "c:/Users/sneha/OneDrive/Desktop/ai coding/fibonacci.py"
Fibonacci of 10 is 55
PS C:\Users\sneha\OneDrive\Desktop\ai coding>
```

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 :

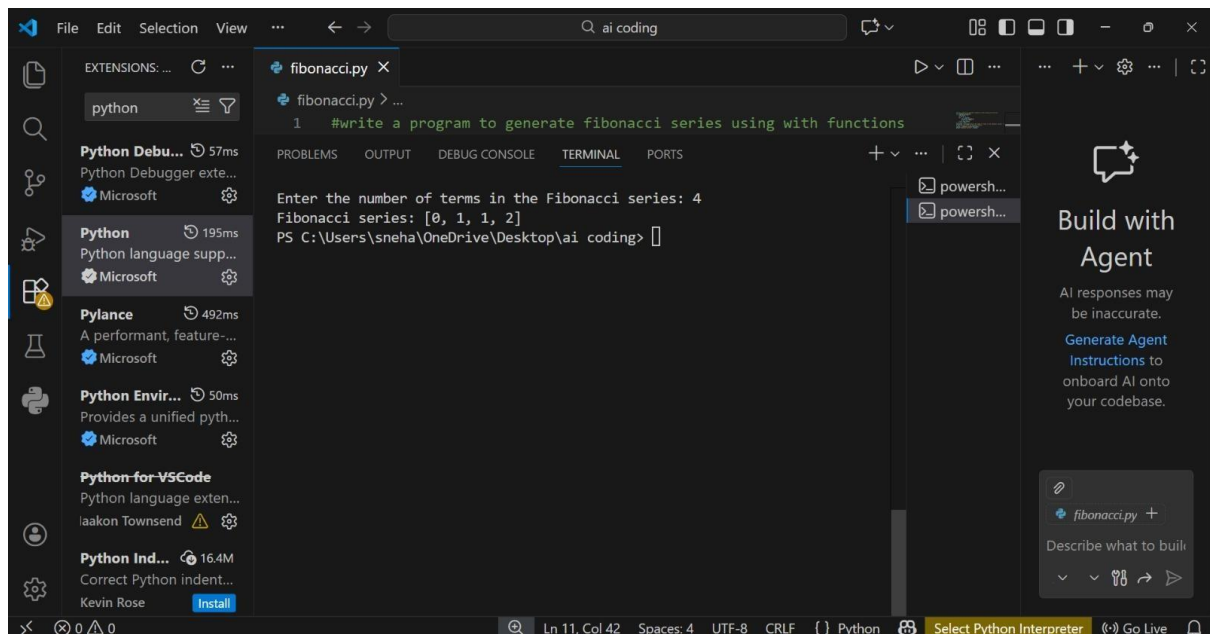


The screenshot shows the Visual Studio Code editor with a Python file named `fibonacci.py`. The code is as follows:

```
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 # Get user input
10 num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
11 # Generate and print the Fibonacci series
12 series = fibonacci_series(num_terms)
13 print("Fibonacci series:", series)
14
```

The left sidebar shows the Extensions view with several Python-related extensions installed, including Python Debugger, Python language support, Pylance, Python Environment, Python for VS Code, and Python Indentation. The bottom status bar indicates the current line and column (Ln 15, Col 5) and the selected Python interpreter.

Output :



The screenshot shows the Visual Studio Code editor with the same Python file `fibonacci.py`. The output of the program is displayed in the terminal window at the bottom:

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

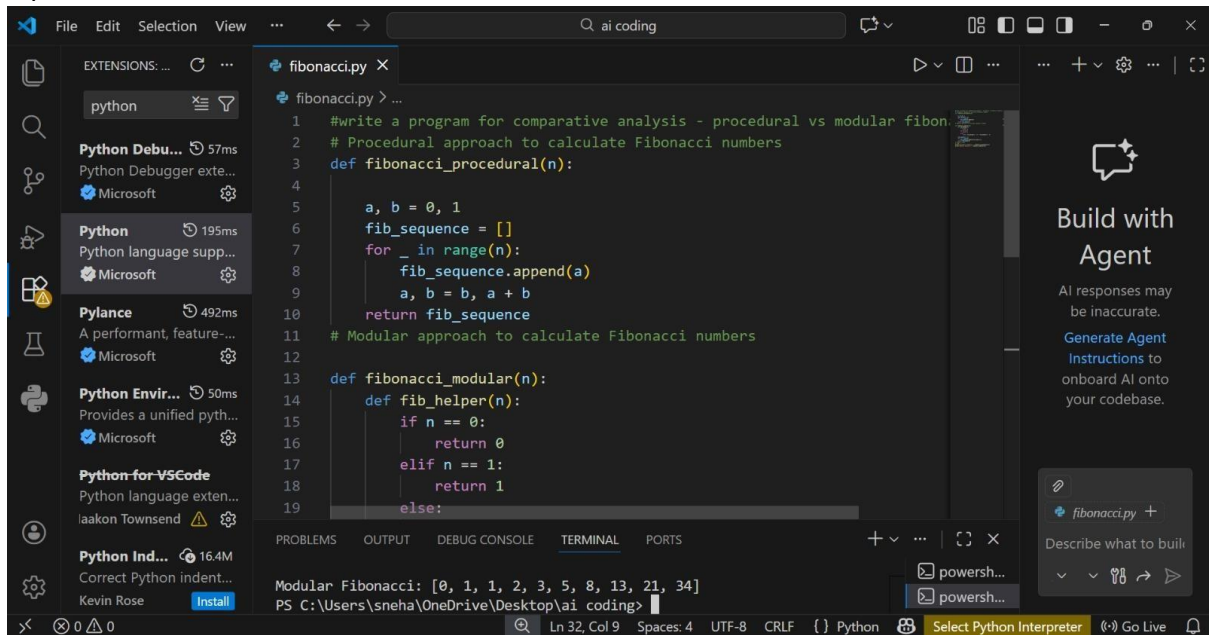
The terminal window shows the command prompt prompt `PS C:\Users\sneha\OneDrive\Desktop\ai coding>`. The status bar at the bottom indicates the current line and column (Ln 11, Col 42) and the selected Python interpreter.

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

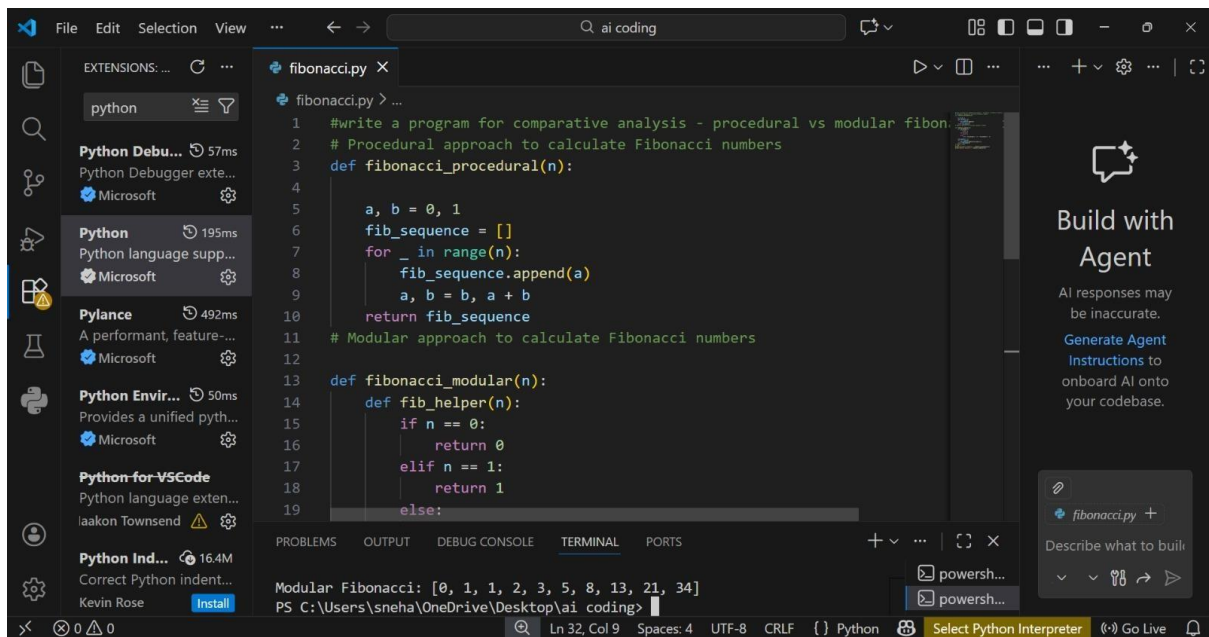
Input :



The screenshot shows the VS Code editor with a file named `fibonacci.py`. The code contains two functions: `fibonacci_procedural(n)` and `fibonacci_modular(n)`. The procedural function uses a loop to calculate the Fibonacci sequence, while the modular function uses a helper function `fib_helper(n)` to calculate the sequence. The terminal output shows the result of the modular function: `Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]`.

```
1 #write a program for comparative analysis - procedural vs modular fibonacci
2 # Procedural approach to calculate Fibonacci numbers
3 def fibonacci_procedural(n):
4
5     a, b = 0, 1
6     fib_sequence = []
7     for _ in range(n):
8         fib_sequence.append(a)
9         a, b = b, a + b
10    return fib_sequence
11
12 # Modular approach to calculate Fibonacci numbers
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:
```

Terminal Output: `Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]`



The screenshot shows the VS Code editor with the same code as the previous image. The terminal output now shows the result of the procedural function: `Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]`.

```
1 #write a program for comparative analysis - procedural vs modular fibonacci
2 # Procedural approach to calculate Fibonacci numbers
3 def fibonacci_procedural(n):
4
5     a, b = 0, 1
6     fib_sequence = []
7     for _ in range(n):
8         fib_sequence.append(a)
9         a, b = b, a + b
10    return fib_sequence
11
12 # Modular approach to calculate Fibonacci numbers
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:
```

Terminal Output: `Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]`

Output :

```

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

```

Modular Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

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 :

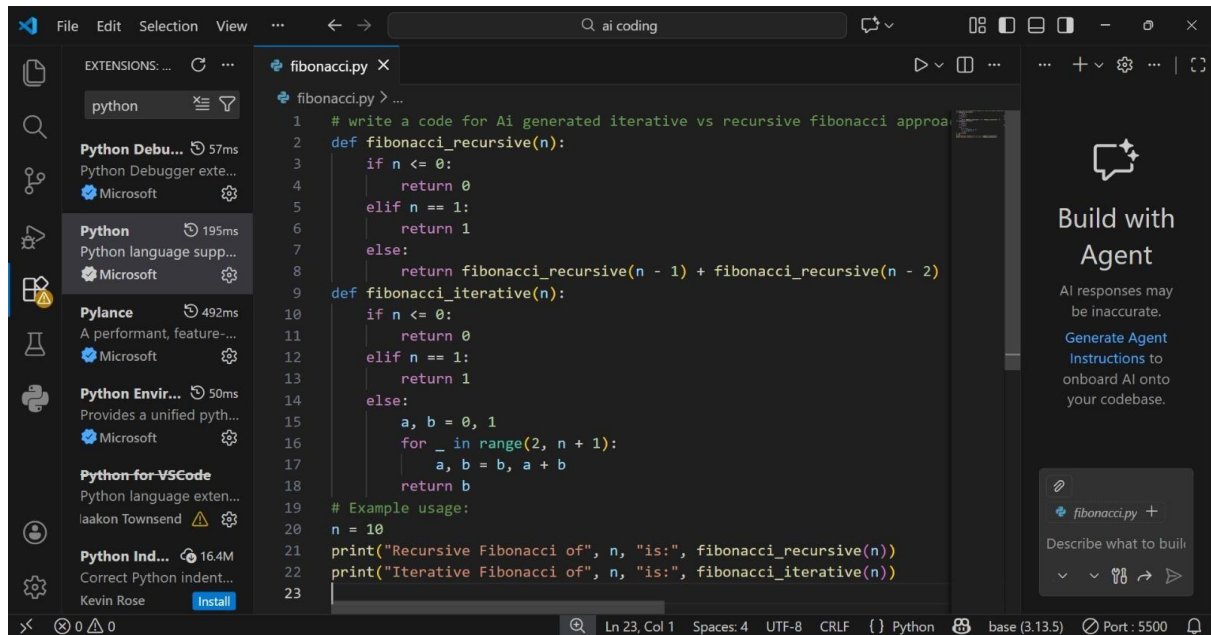
```

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
10 def fibonacci_iterative(n):
11     if n <= 0:
12         return 0
13     elif n == 1:
14         return 1
15     else:
16         a, b = 0, 1
17         for _ in range(2, n + 1):
18             a, b = b, a + b
19         return b
20
21 # Example usage:
22 n = 10
23 print("Recursive Fibonacci of", n, "is:", fibonacci_recursive(n))
24 print("Iterative Fibonacci of", n, "is:", fibonacci_iterative(n))
25

```

Recursive Fibonacci of 10 is: 55
Iterative Fibonacci of 10 is: 55

Output :



```
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
```

Explanation :

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