AI ASSISTANT CODING

ASSIGNMENT-01

Name: K.Archana

HT.No: 2303A51329

Batch: 20

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

Scenario

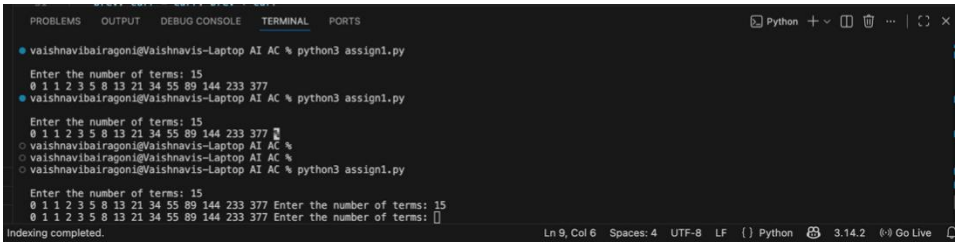
You are asked to write a quick numerical sequence generator for a learning platform prototype.

Prompt: #Fibonacci series upto n terms

Code:

```python
#Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)
#fibonaccci series up to n terms
# Accept user input

n = int(input("Enter the number of terms: "))

# Initialize first two terms
a = 0
b = 1

# Handle edge cases
if n <= 0:
    print("Please enter a positive integer")
elif n == 1:
    print(a)
else:
    print(a, b, end=" ")
    for i in range(2, n):
        c = a + b
        print(c, end=" ")
        a = b
        b = c
```

Result:



Observation:

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

Scenario

The prototype will be shared with other developers and needs optimization.

Prompt: Optimize this code

Code:

Result:

Observation:

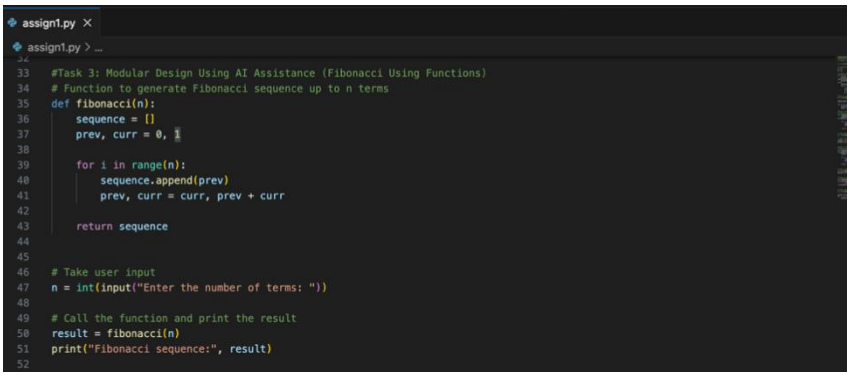
Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)

Scenario

The Fibonacci logic is now required in multiple modules of an application.

Prompt: Optimize this code using functions

Code:



Result:

Observation:

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Scenario

You are participating in a code review session

Prompt: Description on comparision between with functions and without functions

Criteria	Without Functions (Task 1)	With Functions (Task 3)
Code Clarity	Logic is mixed with input/output, making it harder to read	Logic is isolated inside a function, easier to understand
Reusability	Cannot be reused without copying the code	Function can be reused across multiple files or modules
Debugging Ease	Harder to debug because everything runs in one block	Easier to debug by testing the function independently
Maintainability	Changes require editing the entire script	Changes are limited to the function
Suitability for Large Systems	Not suitable; leads to duplicated logic	Highly suitable; supports modular design
Testing	Manual testing only	Function can be unit tested easily
Scalability	Poor scalability	Good scalability

Observation:

The non-function-based Fibonacci program is suitable only for small, one-time scripts or learning exercises. Since the logic is written directly in the main code, it lacks reusability and becomes difficult to maintain or debug as the application grows.

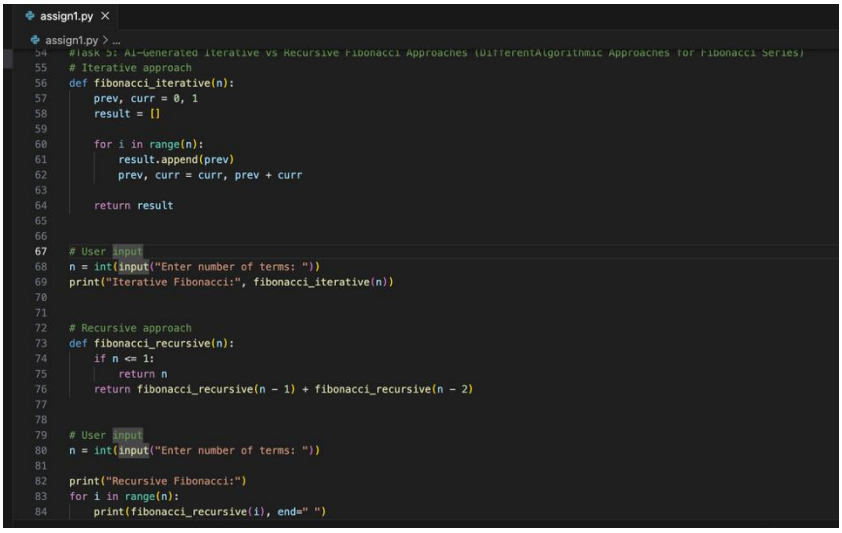
The function-based implementation significantly improves code quality. By separating the Fibonacci logic into a user-defined function, the program becomes modular, reusable, and easier to test. This approach aligns with best practices in software development and is more appropriate for larger systems where the same logic may be required in multiple modules.

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

Scenario

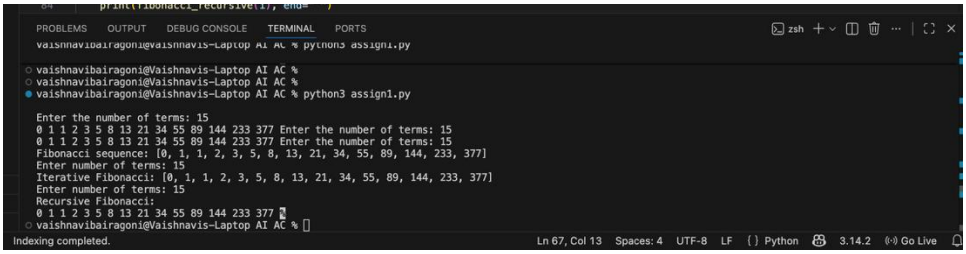
Your mentor wants to assess AI's understanding of different algorithmic paradigms.

Code:



```python
#Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (DifferentAlgorithmic Approaches for Fibonacci Series)
# Iterative approach
def fibonacci_iterative(n):
    prev, curr = 0, 1
    result = []

    for i in range(n):
        result.append(prev)
        prev, curr = curr, prev + curr

    return result


# User input
n = int(input("Enter number of terms: "))
print("Iterative Fibonacci:", fibonacci_iterative(n))


# Recursive approach
def fibonacci_recursive(n):
    if n <= 1:
        return n
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)


# User input
n = int(input("Enter number of terms: "))

print("Recursive Fibonacci:")
for i in range(n):
    print(fibonacci_recursive(i), end=" ")
```

Result:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

vaishnavibairagoni@Vaishnavis-Laptop AI AC %
vaishnavibairagoni@Vaishnavis-Laptop AI AC %
vaishnavibairagoni@Vaishnavis-Laptop AI AC % python3 assign1.py

Enter the number of terms: 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 Enter the number of terms: 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 Enter the number of terms: 15
Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
Enter number of terms: 15
Iterative Fibonacci: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
Enter number of terms: 15
Recursive Fibonacci:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
vaishnavibairagoni@Vaishnavis-Laptop AI AC %

Indexing completed.   Ln 67, Col 13  Spaces: 4  UTF-8  LF  Python  3.14.2  Go Live
```

Observation: