SR UNIVERSITY

AI Assistant Coding
Assignment-1

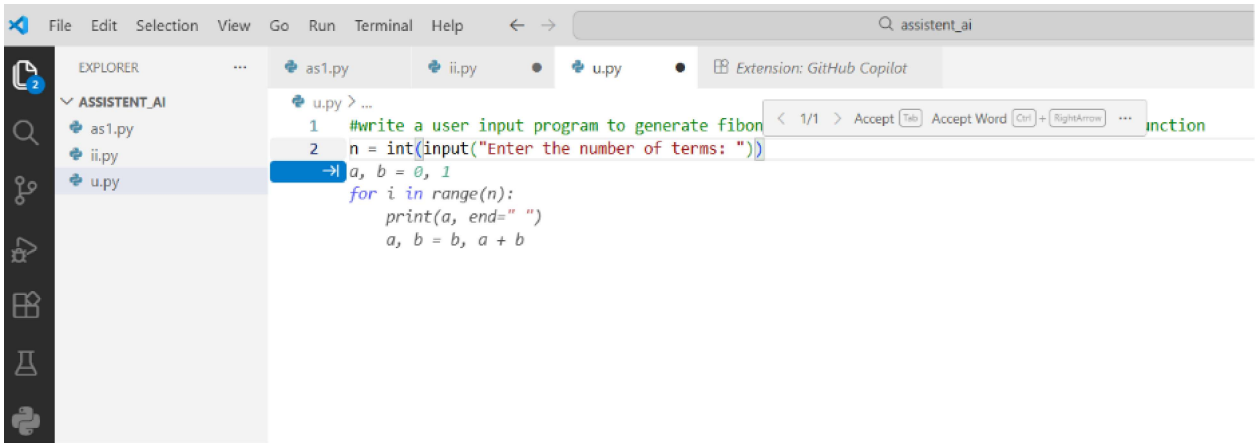
Name of Student : V. Abhinav Batch : 41
Enrollment No. : 2303A52174

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

Use GitHub Copilot to generate a Python program that:
- **Prints the Fibonacci sequence up to *n* terms**
- **Accepts user input for *n***
- **Implements the logic directly in the main code**
- **Does not use any user-defined functions**

Prompt:
Write a user defined program for printing a Fibonacci series up to n terms without using a function.

Code:



Output:

SR UNIVERSITY

JUSTIFICATION:
Generated code using inline prompt of GitHub Copilot in VS Code for the logic of Fibonacci series up to n terms without using a function. It takes input from the user to iterate till n using for loop and print them in a sequence.

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

- **Examine the Copilot-generated code from Task 1 and improve it by:**
- **Removing redundant variables**
- **Simplifying loop logic**
- **Avoiding unnecessary computations**
- **Use Copilot prompts such as:**
  - **"Optimize this Fibonacci code"**
  - **"Simplify variable usage"**

Prompt:
Optimize the below code by removing unnecessary variables and using loop to generate fibonacci series

Code:

Output:

JUSTIFICATION:

The program correctly generates Fibonacci numbers using an iterative approach starting from 0 and 1. It computes exactly *n* terms as input (10), ensuring no extra or missing values. The output format is clean, space-separated, and suitable for debugging or result verification.

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

Use GitHub Copilot to generate a function-based Python program that:

- ➢ **Uses a user-defined function to generate Fibonacci numbers**
- ➢ **Returns or prints the sequence up to n**
- ➢ **Includes meaningful comments (AI-assisted)**

Prompt:

Write a user defined function code to print fibonacci series upto n terms and add meaningfull comments.

CODE:

as1.py | ii.py | u.py | Extension: GitHub Copilot

as1.py > ...

```python
#write the user defined function code to print fibonacci series upto n terms and add meaningfull comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """
    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    count = 0    # Initialize a counter to keep track of the number of terms printed

    # Check if the number of terms is valid
    if n <= 0:
        print("Please enter a positive integer.")
    elif n == 1:
        print("Fibonacci series up to", n, "term:")
        print(a)  # Print the first term
    else:
        print("Fibonacci series up to", n, "terms:")
        while count < n:
            print(a, end=' ')  # Print the current term
            a, b = b, a + b   # Update to the next two terms
            count += 1        # Increment the counter

num = int(input("Enter the number of terms: "))
fibonacci_series(num)  # Print the first num terms of the Fibonacci series
```

SR UNIVERSITY

OUTPUT:

JUSTIFICATION:

In the above code Fibonacci Series logic is written in a Modular design with meaningfull comments that explain what that line does. The " fibonacci_series " function can be called by the user as many times as needed.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code Compare the Copilot-generated Fibonacci programs:

- **Without functions (Task 1)**
- **With functions (Task 3)**
- **Analyze them in terms of:**
  - **Code clarity**
  - **Reusability**
  - **Debugging ease**
  - **Suitability for larger systems**

Prompt:

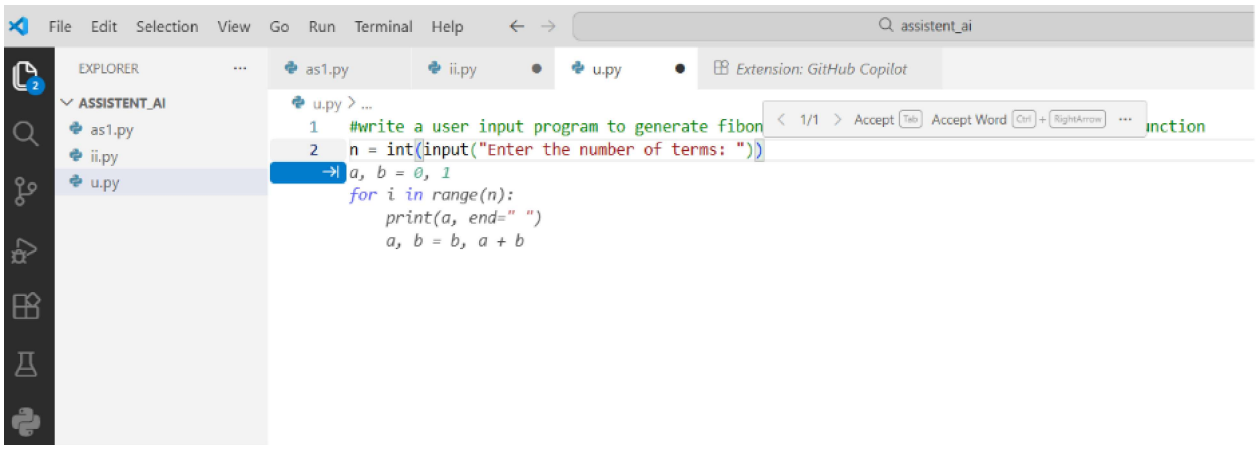
Non-Modular:
Write a user defined program for printing a Fibonacci series up to n terms without using a function.

Modular:
Write a user defined function code to print fibonacci series upto n terms and add meaningfull comments.

Code:

Procedural:

SR UNIVERSITY



Modular:

as1.py ii.py u.py Extension: GitHub Copilot

as1.py > ...

```python
#write the user defined function code to print fibonacci series upto n terms and add meaningfull comments
def fibonacci_series(n):
    """
    This function prints the Fibonacci series up to n terms.

    Parameters:
    n (int): The number of terms in the Fibonacci series to be printed.
    """
    a, b = 0, 1  # Initialize the first two terms of the Fibonacci series
    count = 0    # Initialize a counter to keep track of the number of terms printed

    # Check if the number of terms is valid
    if n <= 0:
        print("Please enter a positive integer.")
    elif n == 1:
        print("Fibonacci series up to", n, "term:")
        print(a)  # Print the first term
    else:
        print("Fibonacci series up to", n, "terms:")
        while count < n:
            print(a, end=' ')  # Print the current term
            a, b = b, a + b   # Update to the next two terms
            count += 1        # Increment the counter

num = int(input("Enter the number of terms: "))
fibonacci_series(num)  # Print the first num terms of the Fibonacci series
```

OUTPUT:

PROCEDURAL

SR UNIVERSITY

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\aksha\Downloads\assistent_ai> & C:\Python313\python.exe c:/Users/aksha/Downloads/assistent_ai/u.py
Enter the number of terms: 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\aksha\Downloads\assistent_ai>
```

OUTPUT:
MODURAL

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\aksha\Downloads\assistent_ai> & C:\Python313\python.exe c:/Users/aksha/Downloads/assistent_ai/as1.py
Enter the number of terms: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\aksha\Downloads\assistent_ai>
```

Justification:

Procedural code is simple but becomes harder to maintain as system size grows. Modular code improves clarity by isolating logic, making intent easier to understand. Modular code is reusable, which can be later integrated into pipelines and projects. Also, debugging is easier in modular code.

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

Prompt GitHub Copilot to generate:

- **An iterative Fibonacci implementation**
- **A recursive Fibonacci implementation**

Prompt:

Write a code for printing a Fibonacci series up to n terms without using a function.

Write a code for printing the Fibonacci series up to n terms using recursion.

SR UNIVERSITY

CODE:

as1.py ● ii.py ● u.py 9+ ●

as1.py > ...

```python
#write a code for printing the Fibonacci series up to n terms without using function.
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci series up to", n, "terms:")
for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b

print()  # for newline


#write a code for printing the Fibonacci series up to n terms using recursion.
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        seq = fibonacci(n - 1)
        seq.append(seq[-1] + seq[-2])
        return seq
n_terms = int(input("Enter the number of terms in Fibonacci series: "))
fibo_series = fibonacci(n_terms)
print("Fibonacci series up to", n_terms, "terms:")
print(*fibo_series)
```

OUTPUT:

PROBLEMS 44 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\aksha\Downloads\assistent_ai> & C:\Python313\python.exe c:/Users/aksha/Downloads/assistent_ai/as1.py
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
Enter the number of terms in Fibonacci series: 12
Fibonacci series up to 12 terms:
0 1 1 2 3 5 8 13 21 34 55 89
PS C:\Users\aksha\Downloads\assistent_ai>
```

Justification:

Since the iterative approach is fast and uses a fixed amount of memory, it works well for large ***n*** value. The recursive method is mathematically neat, but it takes exponential time unless you use memorization. Recursion should be avoided for large inputs due to the risk of stack overflow and slow performance. Iteration is the practical choice for scalable systems and high-performance needs.