AI_ASSISTANT CODING

ASSIGNMENT-01

Name: AVIRENDLA YASHWANTH

Roll No: 2303A52432

Batch: 42

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

Prompt:

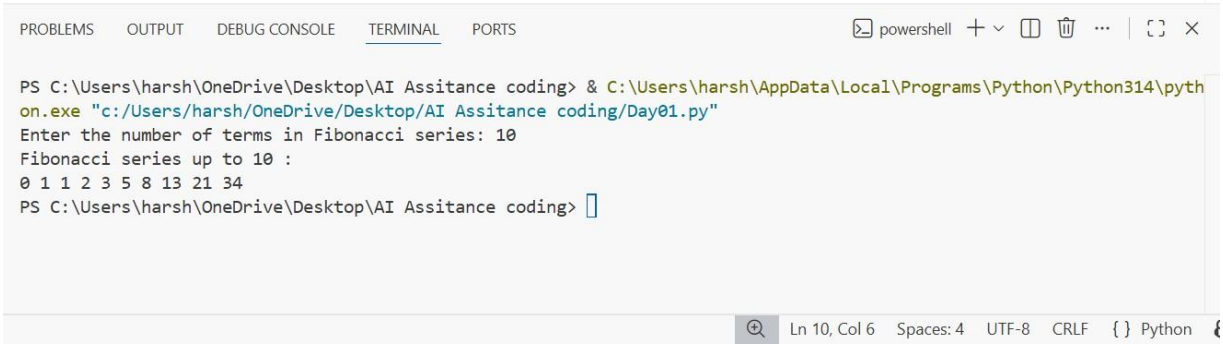
#write a code for printing a Fibonacci series up to n terms without using a function.

Code:

Day01.py > ...

```
#write a code for printing a Fibonacci series up to n terms without using  a function
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
   print("Please enter a positive integer")
elif n == 1:
   print("Fibonacci series up to", n, ":")
   print(a)
else:
    print("Fibonacci series up to", n, ":")
    while count < n:
        print(a, end=' ')
        nth = a + b
        a = b
        b = nth
        count += 1
```

Output:



Justification:

In this task, I used GitHub Copilot to generate Python code for printing the Fibonacci series without using functions. I understood that how the AI is suggesting the prompt by the prior knowledge of us. I executed the code, provided user input, and verified that the output matched the correct Fibonacci sequence.

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

Prompt:

#optimize the code below by removing unnecessary variables and improving readability and avoiding unused computations **Code:**

```python
#optimize the code below by removing unnecessary variables and improving readability and avoidin
#Fibonacci series up to n terms
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
   print("Please enter a positive integer")
elif n == 1:
   print("Fibonacci series up to", n, ":")
   print(a)
else:
    print("Fibonacci series up to", n, ":")
    while count < n:
        print(a, end=' ')
        nth = a + b
        a = b
        b = nth
        count += 1
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   powershell

PS C:\Users\harsh\OneDrive\Desktop\AI Assitance coding> & C:\Users\harsh\AppData\Local\Programs\Python\Python314\pyth
on.exe "c:/Users/harsh/OneDrive/Desktop/AI Assitance coding/Day01.py"
Enter the number of terms in Fibonacci series: 10
Fibonacci series up to 10 :
0 1 1 2 3 5 8 13 21 34
PS C:\Users\harsh\OneDrive\Desktop\AI Assitance coding>

Ln 2, Col 57  Spaces: 4  UTF-8  CRLF  { } Python
```

Justification:

In this task , we have got the optimized version of the previous code, as we observe that the code in the task-1 is already a optimized code. Optimization of the code is very much important, as it reduces the time complexity and space complexity of the code. The code which has more time complexity takes more time to run(slower) and which has more space complexity will utilize maximum space. So, for completing our task efficiently we need optimized code.

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

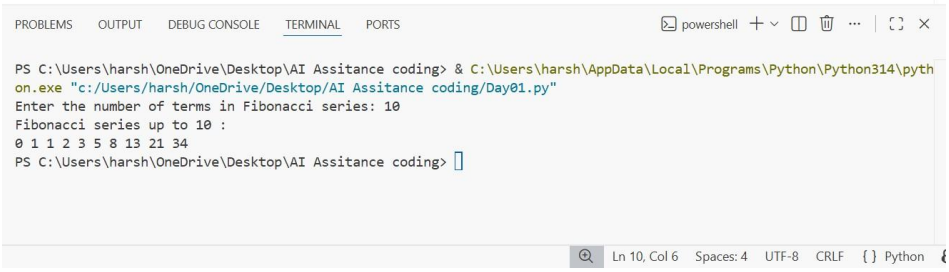
Prompt:

#optimized version of Fibonacci series up to n terms using functions.

Code:

```python
# optimized version of fibonacci series up to n terms with functions
def fibonacci_series(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b  # tuple unpacking for optimization
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series:")
fibonacci_series(n)
```

Output:



Justification:

In this task, we are able to print the Fibonacci series up to n numbers using functions by using the above code. At first we are calling the user-defined function using a function calling statement and then the function runs. In our function the main for loop runs and prints the output.

Task 4: Comparative Analysis - Modular Design Using AI Assistance (Fibonacci Using Functions)

Prompt:

Procedural: #write a code for printing a Fibonacci series up to n terms without using a functions.

Optimized: # optimized version of Fibonacci series up to n terms using functions

Code:

Procedural

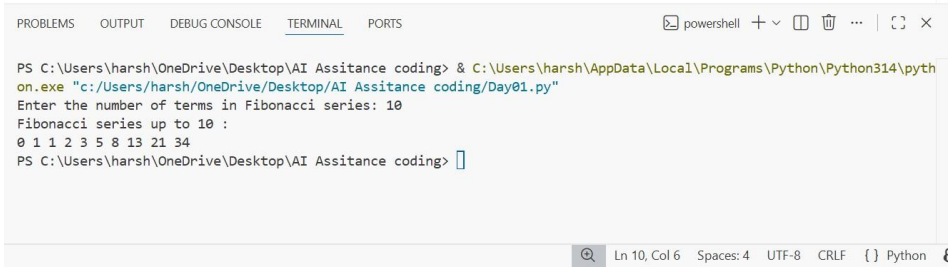
```python
#write a code for printing a Fibonacci series up to n terms without using  a function
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
count = 0
if n <= 0:
   print("Please enter a positive integer")
elif n == 1:
   print("Fibonacci series up to", n, ":")
   print(a)
else:
    print("Fibonacci series up to", n, ":")
    while count < n:
        print(a, end=' ')
        nth = a + b
        a = b
        b = nth
        count += 1
```

Modular

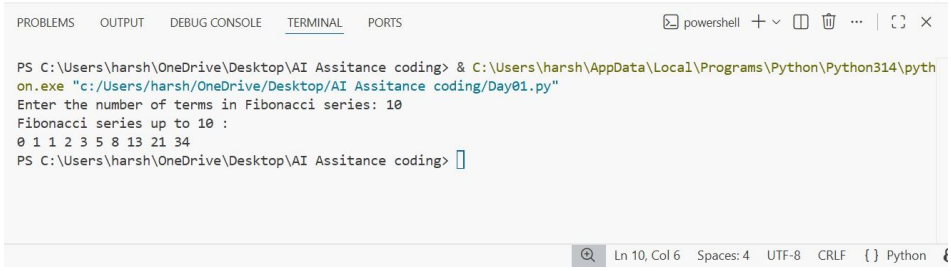
```python
# optimized version of fibonacci series up to n terms with functions
def fibonacci_series(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b  # tuple unpacking for optimization
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series:")
fibonacci_series(n)
```

Output:

Procedural



Modular



Justification:

In this task, we are able to find the difference between Procedural(without using functions) and Modular(using functions). The main use of function is Reusability of the code, Easy to Debug, Code Clarity, Suitable for large systems. By observing, we can analyse that using modular method is a better and clean approach.

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

Prompt:

Iterative Approach: #Fibonacci series up to n terms using iterative approach.

Recursive Approach: #Fibonacci series up to n terms using recursive approach.

Code:

Iterative Approach:

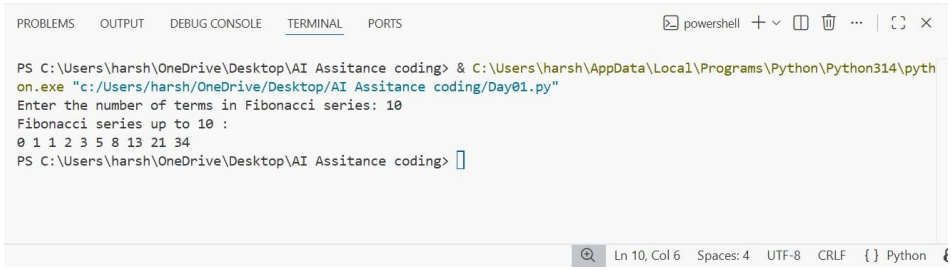
```
# fibonacci series up to n terms using iterative approach
def fibonacci_series(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b  # tuple unpacking for optimization
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series:")
fibonacci_series(n)
```

Recursive Approach:

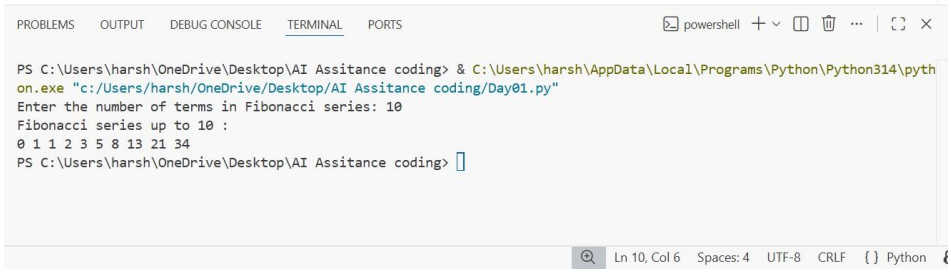
```
# fibonacci series up to n terms using recursive approach
def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci_recursive(n - 1)
        series.append(series[-1] + series[-2])
        return series
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series:")
print(' '.join(map(str, fibonacci_recursive(n))))
```

Output:

Iterative Approach:



Recursive Approach:



Justification:

In this task, The iterative Fibonacci approach is usually the better choice because it's faster, uses very little memory, and works well even when the number of terms is large. Recursive Fibonacci, on the other hand makes many function calls, which slows things down and uses extra memory. For bigger inputs, it can even crash due to recursion limits. That's why, in realworld programs, iteration is generally preferred over recursion.