

AI ASSISTANT CODING-1

Name: Siri Vempati

HT.NO:2303A51309

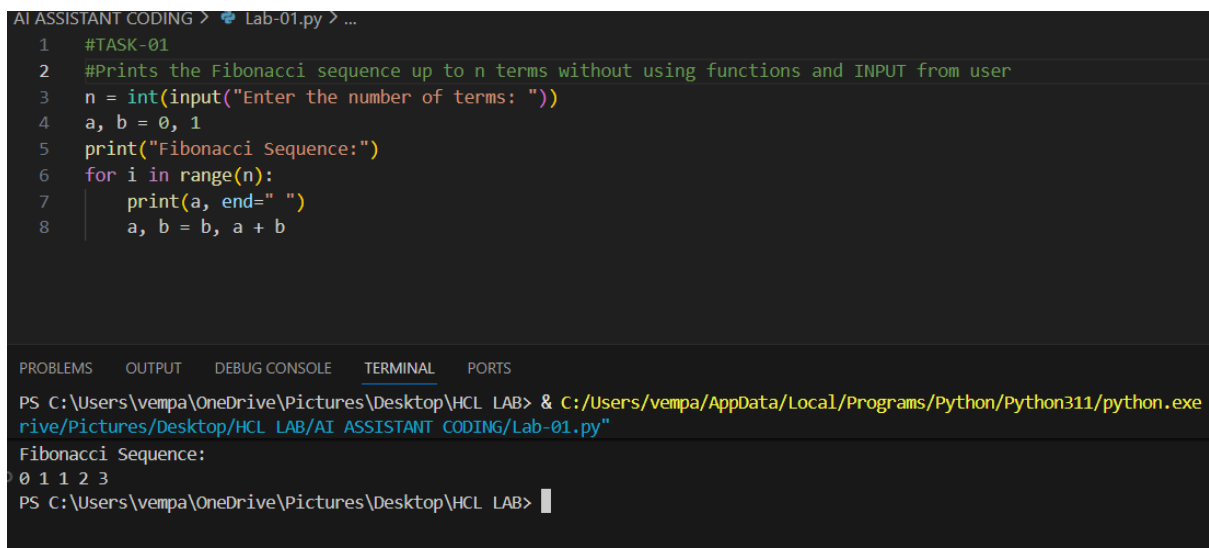
BATCH NO:05

TASK-01:

Prompt: Prints the Fibonacci sequence up to n terms without using functions and INPUT from user

Code:

```
n = int(input("Enter the number of terms: "))
a, b = 0, 1
print("Fibonacci Sequence:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
```



The screenshot shows a code editor with a dark theme. The top part displays the Python code for generating the Fibonacci sequence. Below the code, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, showing the command prompt where the code was executed. The output of the program is displayed in the terminal, showing the Fibonacci sequence: 0 1 1 2 3.

```
AI ASSISTANT CODING > Lab-01.py > ...
1 #TASK-01
2 #Prints the Fibonacci sequence up to n terms without using functions and INPUT from user
3 n = int(input("Enter the number of terms: "))
4 a, b = 0, 1
5 print("Fibonacci Sequence:")
6 for i in range(n):
7     print(a, end=" ")
8     a, b = b, a + b

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/Lab-01.py"
Fibonacci Sequence:
0 1 1 2 3
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
```

Explanation:

This code generates the Fibonacci sequence up to 'n' terms. It initializes the first two terms (0 and 1) and iteratively computes the next terms by updating the values of 'a' and 'b' in each iteration of the loop.

TASK-02

Prompt: Fibonacci series without using functions optimise the code vs original code

Code:

```
n = int(input("\nEnter the number of terms for Fibonacci: "))
```

```
a, b = 0, 1
```

```
print("Fibonacci Sequence:")
```

```
for i in range(n):
```

```
    print(a, end=" ")
```

```
    a, b = b, a + b
```

#code after optimization

```
n = int(input("\nEnter the number of terms for optimized Fibonacci: "))
```

```
a, b = 0, 1
```

```
print("Optimized Fibonacci Sequence:")
```

```
for _ in range(n):
```

```
    print(a, end=" ")
```

```
    a, b = b, a + b
```

```
AI ASSISTANT CODING > Lab-01.py > _
13 #TASK-02
14 #Fibonacci series without using functions optimise the code vs original code
15 n = int(input("\nEnter the number of terms for Fibonacci: "))
16 a, b = 0, 1
17 print("Fibonacci Sequence:")
18 for i in range(n):
19     print(a, end=" ")
20     a, b = b, a + b
21 #code after optimization
22 n = int(input("\nEnter the number of terms for optimized Fibonacci: "))
23 a, b = 0, 1
24 print("Optimized Fibonacci Sequence:")
25 for _ in range(n):
26     print(a, end=" ")
27     a, b = b, a + b
28

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/Lab-01.py" ...
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/Lab-01.py"
Enter the number of terms: 5
Fibonacci Sequence:
0 1 1 2 3
Enter the number of terms for Fibonacci: 6
Fibonacci Sequence:
0 1 1 2 3 5
Enter the number of terms for optimized Fibonacci: 
```

Explanation:

The original code was already quite efficient for generating the Fibonacci sequence.

However, minor improvements were made by removing the loop variable 'i' which was unused, replacing it with an underscore '_' to indicate that it's intentionally ignored.

This enhances readability by signal that the loop variable is not needed.

TASK-03:

Prompt: Fibonacci series using functions with Modular design

Code:

```
def fibonacci_function(n):
    a, b = 0, 1
    sequence = []
    for _ in range(n):
```

```

sequence.append(a)

a, b = b, a + b

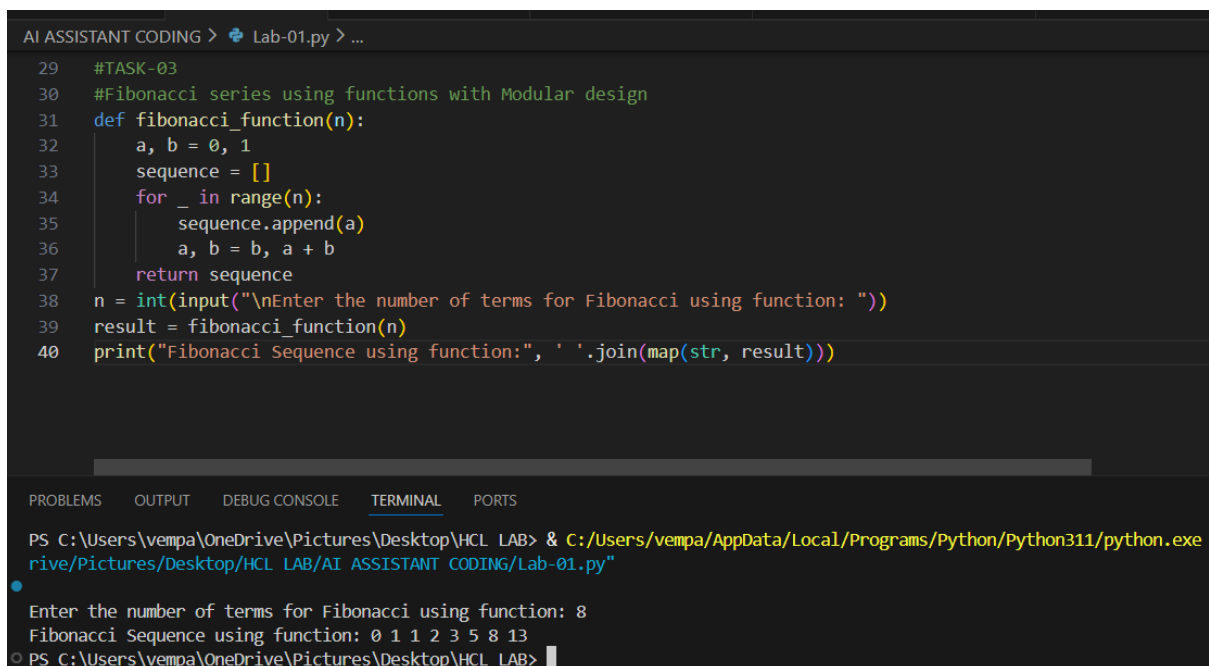
return sequence

n = int(input("\nEnter the number of terms for Fibonacci using function: "))

result = fibonacci_function(n)

print("Fibonacci Sequence using function:", ' '.join(map(str, result)))

```



The screenshot shows a code editor with a dark theme. The top part displays Python code for a Fibonacci function. The bottom part shows a terminal window where the code is executed. The terminal output shows the user entering '8' and the program printing the Fibonacci sequence up to 8 terms: '0 1 1 2 3 5 8 13'.

```

AI ASSISTANT CODING > Lab-01.py > ...
29 #TASK-03
30 #Fibonacci series using functions with Modular design
31 def fibonacci_function(n):
32     a, b = 0, 1
33     sequence = []
34     for _ in range(n):
35         sequence.append(a)
36         a, b = b, a + b
37     return sequence
38 n = int(input("\nEnter the number of terms for Fibonacci using function: "))
39 result = fibonacci_function(n)
40 print("Fibonacci Sequence using function:", ' '.join(map(str, result)))

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/Lab-01.py
Enter the number of terms for Fibonacci using function: 8
Fibonacci Sequence using function: 0 1 1 2 3 5 8 13
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB>

```

Explanation:

This code defines a function 'fibonacci_function' that generates the Fibonacci sequence up to 'n' terms.

It uses a list to store the sequence and returns it after computation.

TASK-04

Prompt: Fibonacci series with procedural vs Modular AI Code(with vs without functions)

Code:

#Procedural Code

```
n = int(input("\nEnter the number of terms for Procedural Fibonacci: "))
```

```
a, b = 0, 1
print("Procedural Fibonacci Sequence:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
#Modular Code
def fibonacci_modular(n):
    a, b = 0, 1
    sequence = []
    for _ in range(n):
        sequence.append(a)
        a, b = b, a + b
    return sequence
n = int(input("\nEnter the number of terms for Modular Fibonacci: "))
result = fibonacci_modular(n)
print("Modular Fibonacci Sequence:", ' '.join(map(str, result)))
```

```
AI ASSISTANT CODING > Lab-01.py > ...

42 #TASK-04
43 #Fibonacci series with procedural vs Modular AI Code(with vs without functions)
44 #Procedural Code
45 n = int(input("\nEnter the number of terms for Procedural Fibonacci: "))
46 a, b = 0, 1
47 print("Procedural Fibonacci Sequence:")
48 for i in range(n):
49     print(a, end=" ")
50     a, b = b, a + b
51 #Modular Code
52 def fibonacci_modular(n):
53     a, b = 0, 1
54     sequence = []
55     for _ in range(n):
56         sequence.append(a)
57         a, b = b, a + b
58     return sequence
59 n = int(input("\nEnter the number of terms for Modular Fibonacci: "))
60 result = fibonacci_modular(n)
61 print("Modular Fibonacci Sequence:", ' '.join(map(str, result)))
62

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe
rive/Pictures/Desktop/HCL LAB/AI ASSISTANT CODING/Lab-01.py"
3
Enter the number of terms for Procedural Fibonacci: 6
Procedural Fibonacci Sequence:
0 1 1 2 3 5
Enter the number of terms for Modular Fibonacci: 6
Modular Fibonacci Sequence: 0 1 1 2 3 5
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> |
```

Explanation:

In the procedural code, the Fibonacci sequence is generated directly within the main body of the code, making it less reusable and harder to maintain.

In contrast, the modular code encapsulates the Fibonacci logic within a function, allowing for easier reuse and better organization.

The modular approach enhances readability and maintainability, as changes to the Fibonacci logic can be made within the function without affecting the main code.

TASK-05:

Prompt: Fibonacci series using recursion vs iterative approach of Fibonacci series

Code:

```
def recursive_fibonacci(n):
```

```
if n <= 0:
    return []
elif n == 1:
    return [0]
elif n == 2:
    return [0, 1]
else:
    seq = recursive_fibonacci(n - 1)
    seq.append(seq[-1] + seq[-2])
    return seq

n = int(input("\nEnter the number of terms for Recursive Fibonacci: "))
result = recursive_fibonacci(n)
print("Fibonacci Sequence using recursion:", ' '.join(map(str, result)))

#Iterative approach

n = int(input("\nEnter the number of terms for Iterative Fibonacci: "))
a, b = 0, 1
print("Fibonacci Sequence using iteration:")
for i in range(n):
    print(a, end=" ")
    a, b = b, a + b
```

```
AI ASSISTANT CODING > Lab-01.py > ...
63 #TASK-05
64 #Fibonacci series using recursion vs iteative approach of Fibonacci series
65 def recursive_fibonacci(n):
66     if n <= 0:
67         return []
68     elif n == 1:
69         return [0]
70     elif n == 2:
71         return [0, 1]
72     else:
73         seq = recursive_fibonacci(n - 1)
74         seq.append(seq[-1] + seq[-2])
75         return seq
76 n = int(input("\nEnter the number of terms for Recursive Fibonacci: "))
77 result = recursive_fibonacci(n)
78 print("Fibonacci Sequence using recursion:", ' '.join(map(str, result)))
79 #Iterative approach
80 n = int(input("\nEnter the number of terms for Iterative Fibonacci: "))
81 a, b = 0, 1
82 print("Fibonacci Sequence using iteration:")
83 for i in range(n):
84     print(a, end=" ")
85     a, b = b, a + b
86
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> & C:/Users/vempa/AppData/Local/Programs/Python/Python311/python.exe

Enter the number of terms for Recursive Fibonacci: 9
Fibonacci Sequence using recursion: 0 1 1 2 3 5 8 13 21

Enter the number of terms for Iterative Fibonacci: 9
Fibonacci Sequence using iteration:
0 1 1 2 3 5 8 13 21
PS C:\Users\vempa\OneDrive\Pictures\Desktop\HCL LAB> 
```

Explanation:

This code defines a recursive function to generate the Fibonacci sequence. The function builds the sequence by calling itself to get the sequence up to $n-1$ terms and then appending the next term.

The iterative approach is also provided for comparison, which uses a loop to generate the sequence.

