

# Assignment - 1

**Name :** Adama Supreeth Reddy  
**Hall Ticket No. :** 2303A51392  
**Batch No. :** 06

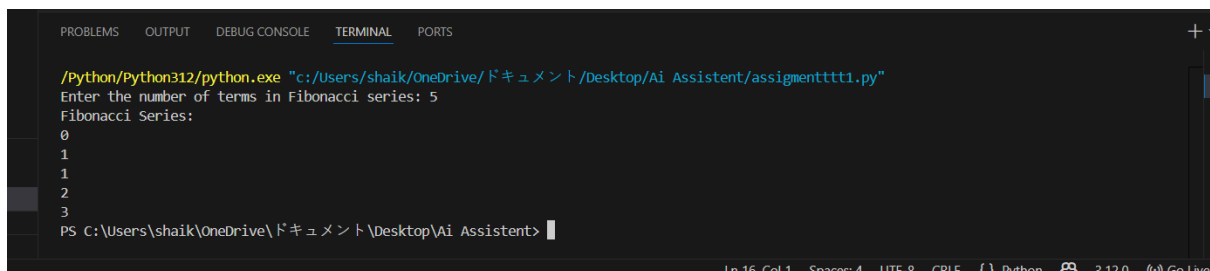
## **#Task 1**

**Prompt:** #Fibonacci Series without Functions and logic should be in main code body

**Code:**

```
n = int(input("Enter the number of terms in Fibonacci series: "))
a, b = 0, 1
print("Fibonacci Series:")
#handling edge cases
if n <= 0:
    print("Please enter a positive integer.")
elif n == 1:
    print(a)
else:
    print(a)
    print(b)
    for _ in range(2, n):
        c = a + b
        print(c)
        a, b = b, c
```

## **Output :**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python/Python312/python.exe "c:/Users/shaik/OneDrive/ドキュメント/Desktop/Ai Assistant/assignmentttt1.py"
Enter the number of terms in Fibonacci series: 5
Fibonacci Series:
0
1
1
2
3
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant>
```

**Explanation :** The algorithm starts with the first two numbers of the Fibonacci sequence, which are 0 and 1."This loop is set to run 'n' times." Every successive number is found by adding the two numbers prior to it. The logic is directly coded within the main code body. It is not made modular. This method is fast but cannot be reused in bigger applications.

## **#Task 2**

**Prompt:** #Optimize the code in shorter form


**Code:**

```
n = int(input("Enter the number of terms in Fibonacci series: "))
if n <= 0:
    print("Please enter a positive integer.")
else:
    a, b = 0, 1
    print("Fibonacci Series:")
```

```

for i in range(n):
    print(a)
    a, b = b, a + b

```



```

PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant> & C:/Users/shaik/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/shaik/OneDrive/ドキュメント/Desktop/Ai Assistant/assignmentttt1.py"
Enter the number of terms in Fibonacci series: 5
Fibonacci Series:
0
1
1
2
3
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant>

```

**Explanation : Inefficient-** Additional and not altogether necessary conditional tests. Slightly verbose variable handling. Messages for simple logic – redundancy.

**Optimized** - Fewer number of conditions. Cleaned up and legible loop code. Same output with reduced structure. More understandable and maintainable for programmers.

### #Task 3

**Prompt**: #Fibonacci Series with functions

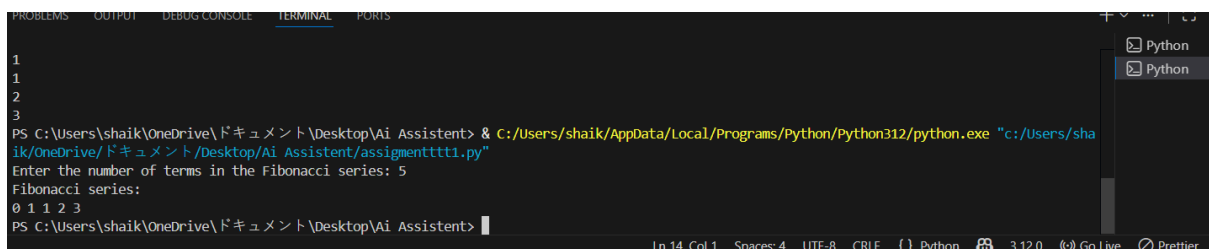
**Code**:

```

def fibonacci(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
fib_series = fibonacci(num_terms)
print("Fibonacci series:")
for num in fib_series:
    print(num, end=' ')

```

**Output :**



```

1
1
2
3
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant> & C:/Users/shaik/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/shaik/OneDrive/ドキュメント/Desktop/Ai Assistant/assignmentttt1.py"
Enter the number of terms in the Fibonacci series: 5
Fibonacci series:
0 1 1 2 3
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant>

```

**Explanation :** Logic is encapsulated inside a function. The function returns the Fibonacci series up to the given number as a list. Improves code reuse, testing, and readability. Suitable for large and modular applications.

### #Task 4

**Prompt** : #Compare the two methods and give differences

#print the differences

# Description on comparison between with functions and without functions

**Code :**

#Fibonacci Series without functions

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

#Fibonacci Series with functions

```
def fibonacci(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
fib_series = fibonacci(num_terms)
print("Fibonacci series:")
for num in fib_series:
    print(num, end=' ')
```

#Compare the two methods and give differences

#print the differences

```
print("\n\nDifferences between the two methods:")
```

```
print("1. The first method does not use functions, while the second method encapsulates the logic in a function.")
```

```
print("2. The first method prints the series directly, while the second method returns a list of Fibonacci numbers.")
```

# Description on comparison between with functions and without functions

```
print("3. The function-based approach is more reusable and modular, allowing for easier testing and maintenance.")
```

**Tabular Format:**

Feature	Without Functions	With Functions
Code Clarity	Moderate	High
Reusability	No	Yes
Debugging	Difficult	Easy
Scalability	Poor	Excellent

Feature	Without Functions	With Functions
Suitability for Large Systems	Low	High

### **Output :**

```

Enter the number of terms in the Fibonacci series: 5
Fibonacci series:
0 1 1 2 3

Enter the number of terms in the Fibonacci series: 5
Fibonacci series:
0 1 1 2 3

Differences between the two methods:
1. The first method does not use functions, while the second method encapsulates the logic in a function.

```

**Explanation :** Analysis -Procedural code is simpler to write but nastier to maintain. Function-oriented code is more cleanable. In real-world software development, it is preferred to be modular.

### **#Task 5**

**Prompt :** #fibanocci series using different algorithms or Recursion or Iteration or Dynamic Programming

### **Code :**

```

# Function to generate Fibonacci series using recursion
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

# Function to generate Fibonacci series using iteration
def fibonacci_iterative(n):
    series = []
    a, b = 0, 1
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

# Function to generate Fibonacci series using dynamic programming
def fibonacci_dynamic(n):
    if n <= 0:

```

```

        return []
    series = [0] * n
    series[0] = 0
    if n > 1:
        series[1] = 1
    for i in range(2, n):
        series[i] = series[i - 1] + series[i - 2]
    return series

# Example usage
n = int(input("Enter the number of terms in Fibonacci series: "))
print("Fibonacci series using recursion:", fibonacci_recursive(n))
print("Fibonacci series using iteration:", fibonacci_iterative(n))
print("Fibonacci series using dynamic programming:", fibonacci_dynamic(n))

```

### Output :

```

1. The first method does not use functions, while the second method encapsulates the logic in a function.
2. The first method prints the series directly, while the second method returns a list of Fibonacci numbers.
3. The function-based approach is more reusable and modular, allowing for easier testing and maintenance.
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant> & c:/Users/shaik/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/shaik/OneDrive/ドキュメント/Desktop/Ai Assistant/assignmentttt1.py"
Enter the number of terms in Fibonacci series: 5
Fibonacci series using recursion: [0, 1, 1, 2, 3]
Fibonacci series using iteration: [0, 1, 1, 2, 3]
Fibonacci series using dynamic programming: [0, 1, 1, 2, 3]
PS C:\Users\shaik\OneDrive\ドキュメント\Desktop\Ai Assistant>

```

### Explanation :

**Iterative:** Iterative solutions employ the use of loops to work with different values.

**Recursive:** It calls itself within its own definition.

The iterative method is efficient. Recursion is good for learning, but not very efficient. For large inputs, recursion should not be used.