Assignment - 01

Name :M.Shiva Shankar

Hall Ticket: 2303A51294

Batch No. : 05

#Task 1 Prompt: Write a python code for fibanocci series without using functions?

Code:

```python
# Fibonacci series without using functions

n = 10   # Number of terms
a = 0
b = 1

print("Fibonacci Series:")
for i in range(n):
    print(a, end=' ')
    c = a + b
    a = b
    b = c
```

Output :

```
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/    powershell
python/new.py                                                                                                                                           Python
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Administrator\Desktop\HCI>
```
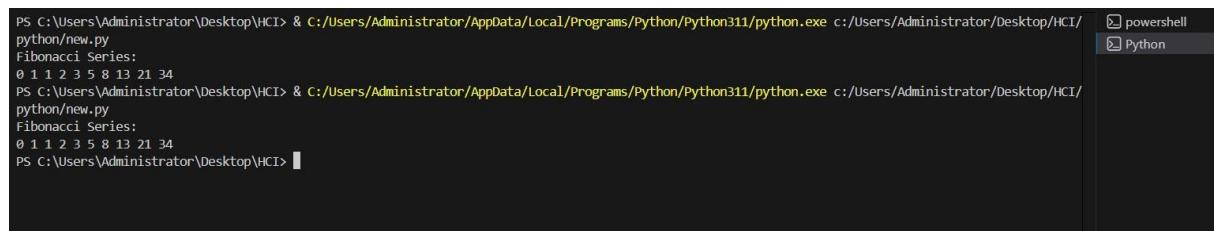
Explanation :

This Python program prints the Fibonacci series using an iterative method. The variable n defines how many terms to generate, while a and b start with the values 0 and 1. A for loop runs n times, printing the value of a in each step. After each print, the values are updated so that a becomes b and b becomes the sum of the previous two numbers. This continues until the required number of Fibonacci terms are printed, resulting in the first 10 numbers displayed in a single line.

#Task 2 Prompt: #Optimize this code and simplify the code.

Code:

```
python > 🐍 new.py > ...
   1   n = 10   # Number of terms
   2   a, b = 0, 1
   3
   4   print("Fibonacci Series:")
   5   for _ in range(n):
   6       print(a, end=' ')
   7       a, b = b, a + b
```

Output :

```
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/
python/new.py
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/
python/new.py
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34
PS C:\Users\Administrator\Desktop\HCI>
```

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: #Generate a python program on fibanocci series using functions?

CODE:

```python
python >  new.py > ...
  1   def fibonacci_series(n):
  2       series = []
  3       a, b = 0, 1
  4       for _ in range(n):
  5           series.append(a)
  6           a, b = b, a + b
  7       return series
  8
  9   def main():
 10       num = int(input("Enter the number of terms: "))
 11       if num <= 0:
 12           print("Please enter a positive integer.")
 13       else:
 14           print("Fibonacci series:")
 15           print(fibonacci_series(num))
 16
 17   if __name__ == "__main__":
 18       main()
```

Output :

```
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/
python/new.py
Enter the number of terms: 5
Fibonacci series:
[0, 1, 1, 2, 3]
PS C:\Users\Administrator\Desktop\HCI>
                                                                                Ln 18, Col 11   Spaces: 4   UTF-8   {} Python   3.11.5   (·) Go Live
```

Explanation :

This Python program generates the Fibonacci series using functions. The function fibanocci series (n) takes an integer n as input and initializes an empty list called series to store the Fibonacci numbers. The variables a and b are initialized to 0 and 1, which are the first two Fibonacci numbers. A for loop runs n times, and in each iteration, the current value of a is appended to the list. Then, the values of a and b are updated so that a becomes b and b becomes the sum of the previous two numbers. After completing the loop, the function returns the list containing the Fibonacci series. The main() function takes user input for the number of terms, checks whether the input is a positive integer, and then prints the Fibonacci series by calling the fibonacci_series function. The condition if __name__ == "__main__": ensures that the main() function runs only when the program is executed directly.

#Task 4
Prompt : #Compare the two methods and give differences
#print the differences
Description on comparision 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")


#Fibinocci 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 comparision 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
Feature	Without Functions	With Functions
Scalability	Poor	Excellent
Suitability for Large Systems	Low	High

Output :

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 : #Generate Fibonacci series using both iterative and recursive methods. Provide code examples, explain how each approach works, and compare their efficiency. **Code** :

```python
# Fibonacci Series: Iterative Method
def fibonacci_iterative(n):
    series = []
    a, b = 0, 1
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

# Fibonacci Series: Recursive Method
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

# Example usage:
n = 10
```

Output :

```
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/
python/new.py
Enter the number of terms: 5
Fibonacci series:
[0, 1, 1, 2, 3]
PS C:\Users\Administrator\Desktop\HCI> & C:/Users/Administrator/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Administrator/Desktop/HCI/
python/new.py
Iterative: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Recursive: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\Administrator\Desktop\HCI>
```

Ln 35, Col 71   Spaces: 4   UTF-8   { } Python   3.11.5   (•) Go Live

Explanation : **Iterative**

vs Recursive :

The iterative method uses loops to repeat steps and is fast and memory-efficient. The recursive method works by a function calling itself, which makes the logic easy to understand but uses more memory. Because of this, recursion is not suitable for large inputs, while iteration is a better and more practical choice.

.