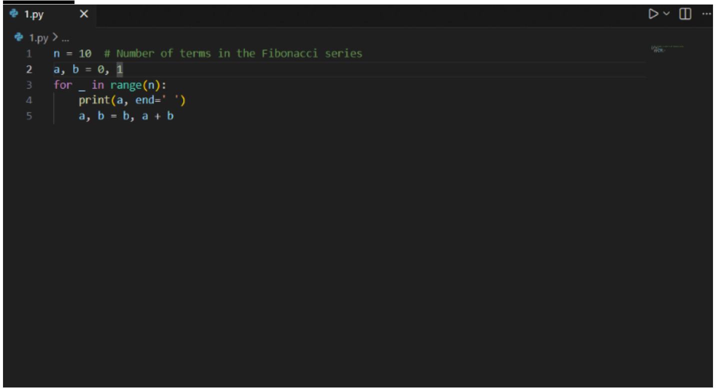
Assignment - 1

Name : Uthuri Karthikeya
Hall Ticket No. : 2303A51306
Batch No. : 05

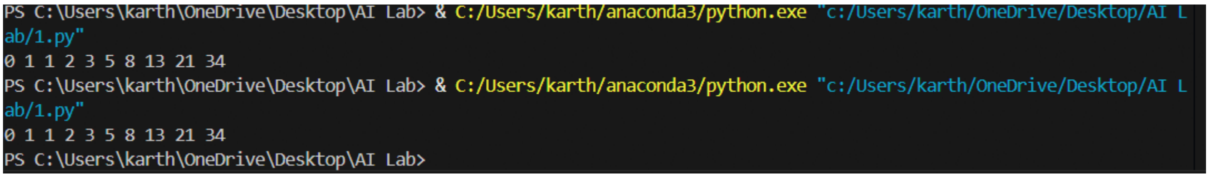
#Task 1
Prompt: #Generate a python program on fibanocci series without using functions

Code:



```python
n = 10  # Number of terms in the Fibonacci series
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

Output :

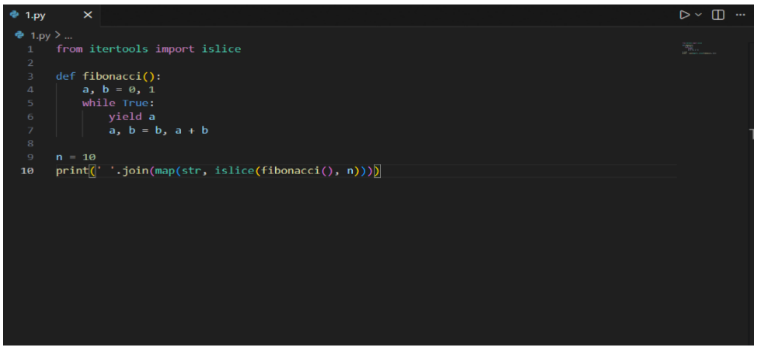


```
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\karth\OneDrive\Desktop\AI Lab>
```

Explanation : This Python code generates the Fibonacci series using an iterative approach. The variable n specifies how many terms to print, while a and b are initialized to the first two Fibonacci numbers (0 and 1). The for loop runs n times, printing the current value of a each time and then updating a and b so that a takes the value of b and b becomes the sum of the previous two numbers. As a result, the program efficiently prints the first 10 Fibonacci numbers in a single line.

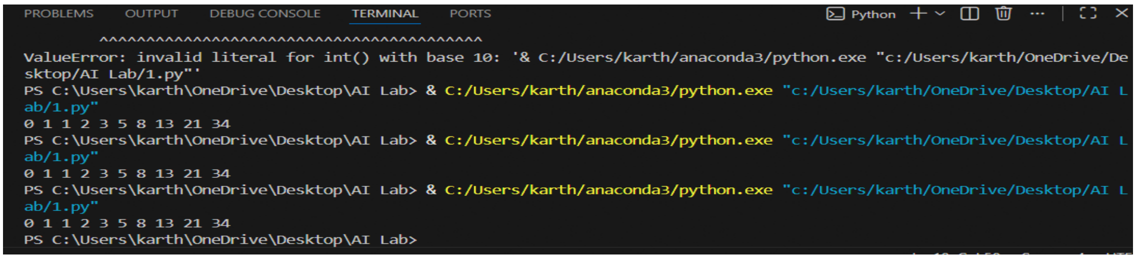
#Task 2
Prompt: #Optimize this code and simply this code

Code:



```python
from itertools import islice

def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

n = 10
print(' '.join(map(str, islice(fibonacci(), n))))
```

Output :



```
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: '& C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/De
sktop/AI Lab/1.py"'
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\karth\OneDrive\Desktop\AI Lab>
```

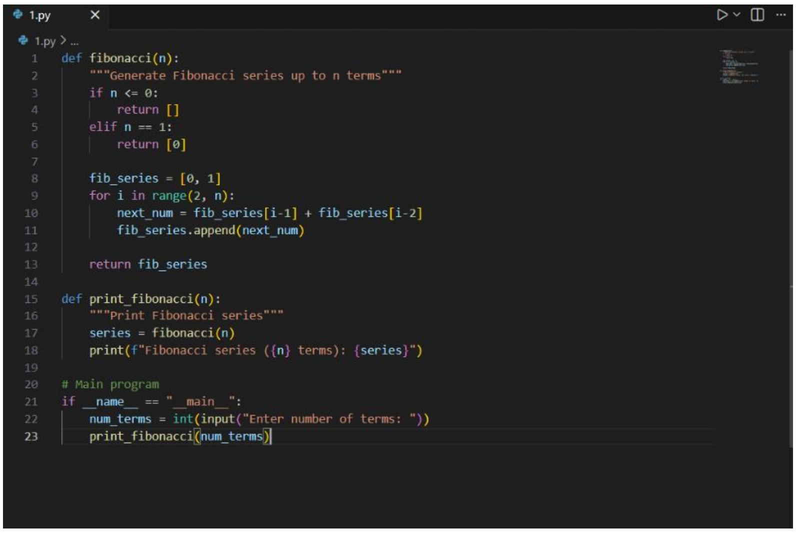
Explanation **: Inefficient-** Additional and not altogether necessary conditional tests. Slightly verbose variable handling. Messages for simple logic – redundancy.

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

#Task 3

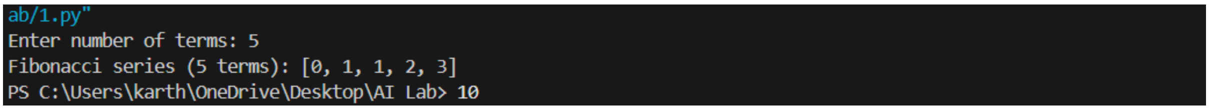
Prompt: #Generate a python program on fabinocci series using functions

Code:



```python
def fibonacci(n):
    """Generate Fibonacci series up to n terms"""
    if n <= 0:
        return []
    elif n == 1:
        return [0]

    fib_series = [0, 1]
    for i in range(2, n):
        next_num = fib_series[i-1] + fib_series[i-2]
        fib_series.append(next_num)

    return fib_series

def print_fibonacci(n):
    """Print Fibonacci series"""
    series = fibonacci(n)
    print(f"Fibonacci series ({n} terms): {series}")

# Main program
if __name__ == "__main__":
    num_terms = int(input("Enter number of terms: "))
    print_fibonacci(num_terms)
```

Output :



```
ab/1.py"
Enter number of terms: 5
Fibonacci series (5 terms): [0, 1, 1, 2, 3]
PS C:\Users\karth\OneDrive\Desktop\AI Lab> 10
```

Explanation **:** The logic is written inside a function, which makes the code clean and organized. This function calculates the Fibonacci series up to the given number and returns it as a list. Using a function helps reuse the same code in different places, makes testing easier, and improves readability. This approach is especially useful for large programs and modular applications.

#Task 4

Prompt : #Compare the two methods and give differences
#print the differences
# Description on comparision between with functions and without functions

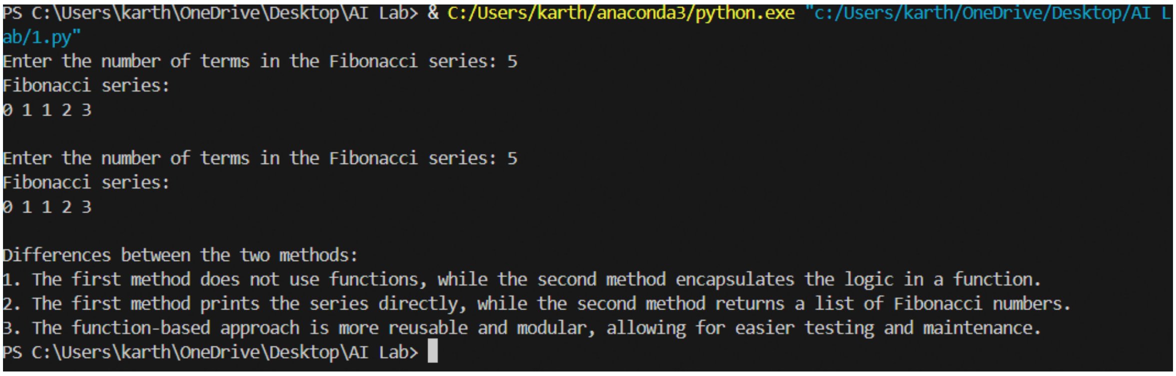
Code :

```
#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
Scalability	Poor	Excellent
Suitability for Large Systems	Low	High

Output :



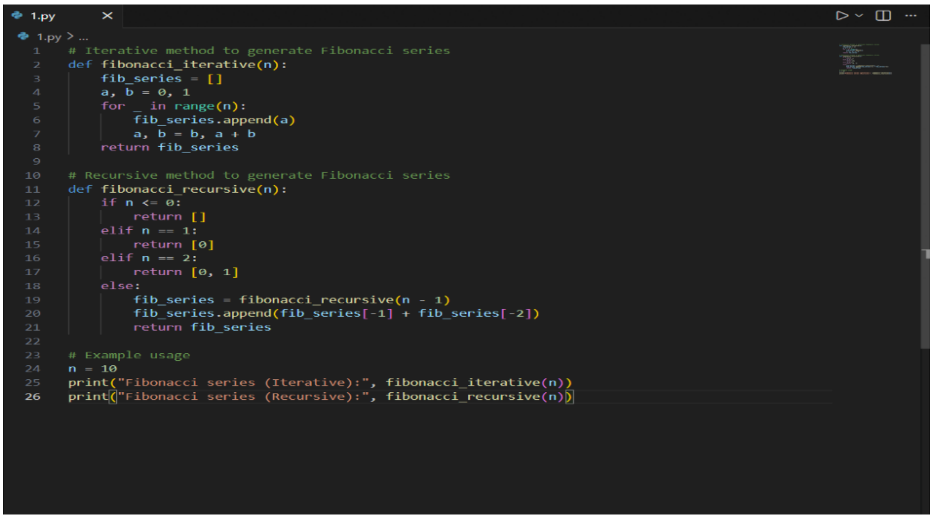
```
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
Enter the number of terms in the Fibonacci series: 5
Fibonacci series:
0 1 1 2 3

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.
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\karth\OneDrive\Desktop\AI Lab>
```

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

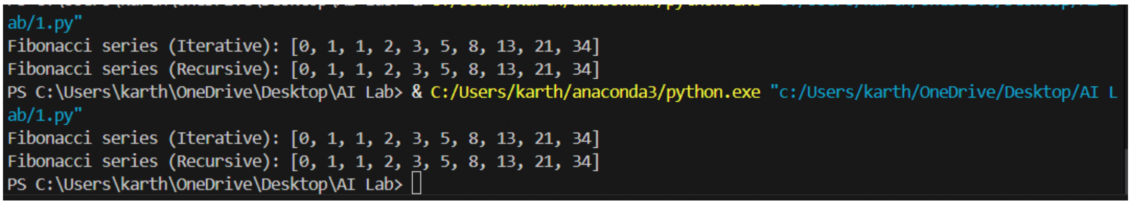
Prompt : #Generate Fibonacci series using both iterative and recursive methods. Provide code examples, explain how each approach works, and compare their efficiency.

Code :



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

# Recursive method to generate Fibonacci series
def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        fib_series = fibonacci_recursive(n - 1)
        fib_series.append(fib_series[-1] + fib_series[-2])
        return fib_series

# Example usage
n = 10
print("Fibonacci series (Iterative):", fibonacci_iterative(n))
print("Fibonacci series (Recursive):", fibonacci_recursive(n))
```

Output :



```
ab/1.py"
Fibonacci series (Iterative): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci series (Recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\karth\OneDrive\Desktop\AI Lab> & C:/Users/karth/anaconda3/python.exe "c:/Users/karth/OneDrive/Desktop/AI L
ab/1.py"
Fibonacci series (Iterative): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci series (Recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\karth\OneDrive\Desktop\AI Lab> 
```

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.

.