ASSIGNMENT -1.5

NAME : SYED SUFIYAN

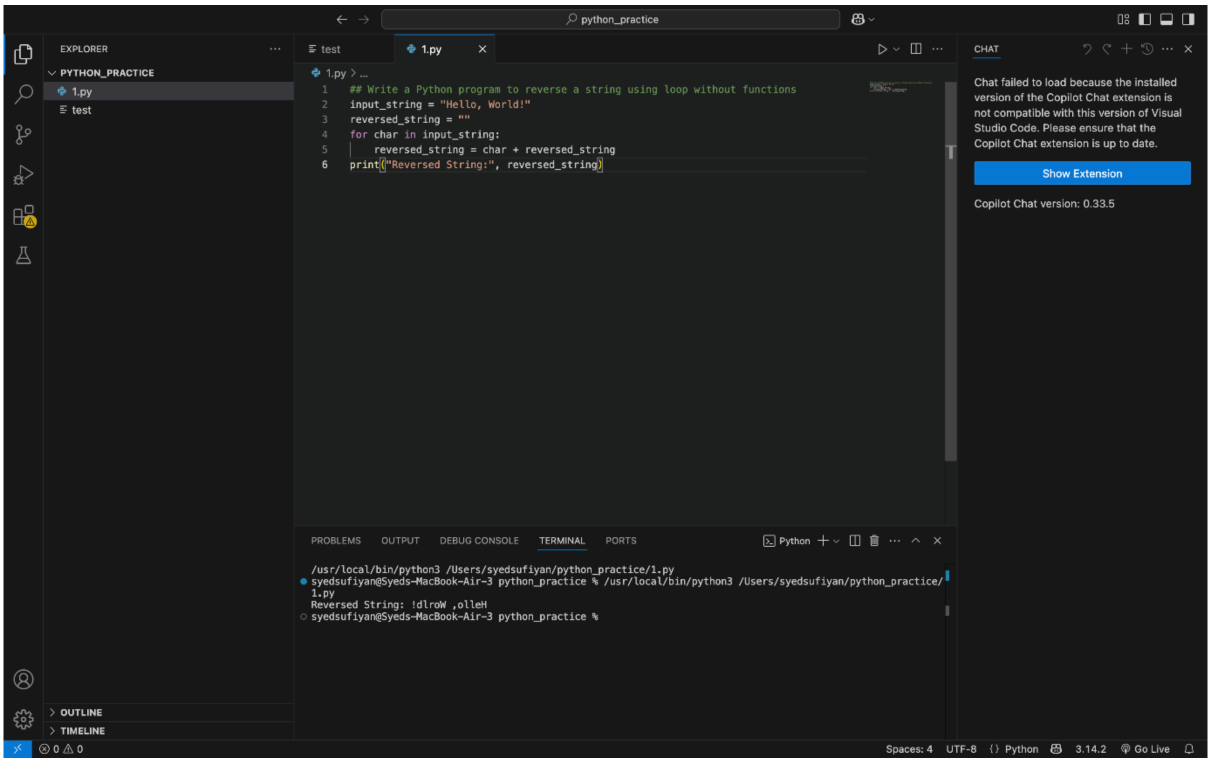
2303A51980

BATCH:30

TASK 1:

PROMPT: AI-GENERATED LOGIC WITHOUT MODULARIZATION (STRING REVERSAL WITHOUT FUNCTIONS)

CODE:



```python
## Write a Python program to reverse a string using loop without functions
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String:", reversed_string)
```

```
/usr/local/bin/python3 /Users/syedsufiyan/python_practice/1.py
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/
1.py
Reversed String: !dlroW ,olleH
syedsufiyan@Syeds-MacBook-Air-3 python_practice %
```

Chat failed to load because the installed version of the Copilot Chat extension is not compatible with this version of Visual Studio Code. Please ensure that the Copilot Chat extension is up to date.

Copilot Chat version: 0.33.5

OBSERVATION:

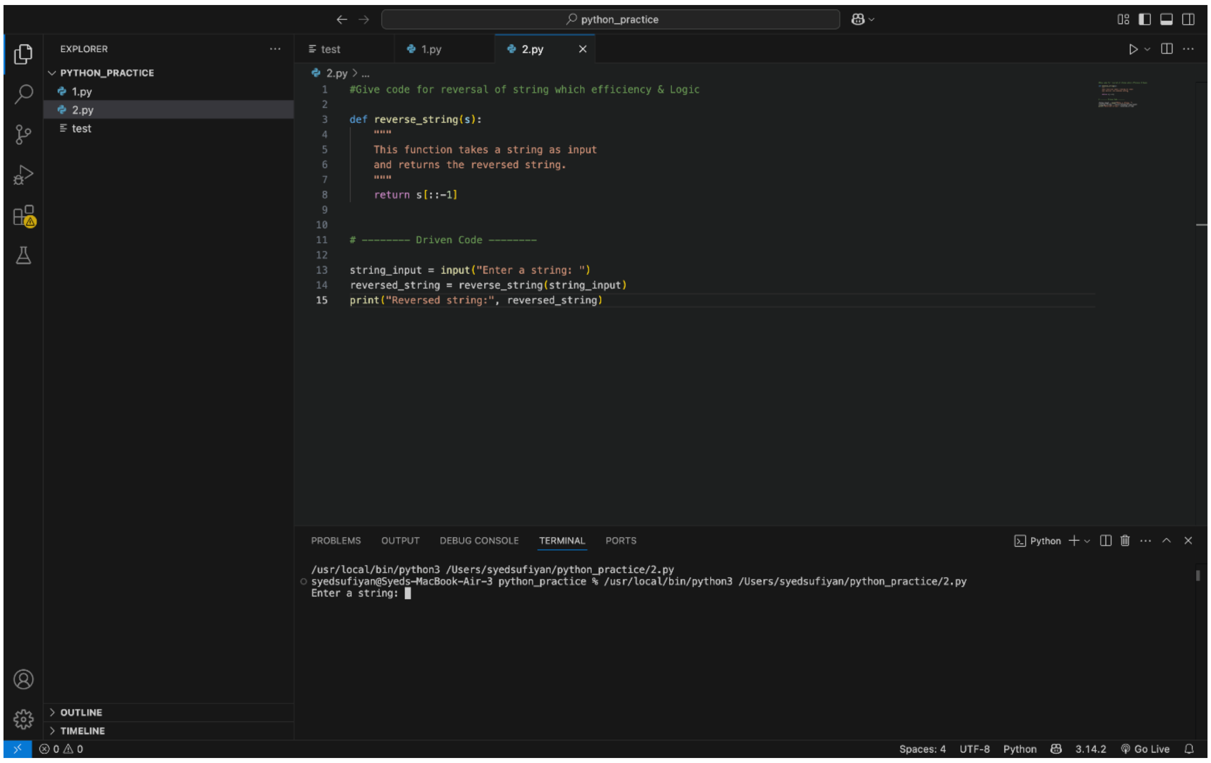
The program successfully reverses the given string using a simple loop without using any functions. Each character is added to the beginning of a new string, which gradually forms the reversed output. The output confirms that the logic works correctly by displaying the reversed string. This approach is easy to understand

and suitable for beginners learning basic string operations. However, for larger programs, a more optimized or modular approach would be better.

TASK:2

PROMPT: GIVE CODE FOR REVERSAL OF STRING WHICH EFFICIENCY & LOGIC OPTIMIZATION

CODE:



```python
#Give code for reversal of string which efficiency & Logic

def reverse_string(s):
    """
    This function takes a string as input
    and returns the reversed string.
    """
    return s[::-1]


# -------- Driven Code --------

string_input = input("Enter a string: ")
reversed_string = reverse_string(string_input)
print("Reversed string:", reversed_string)
```

```
/usr/local/bin/python3 /Users/syedsufiyan/python_practice/2.py
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/2.py
Enter a string: 
```

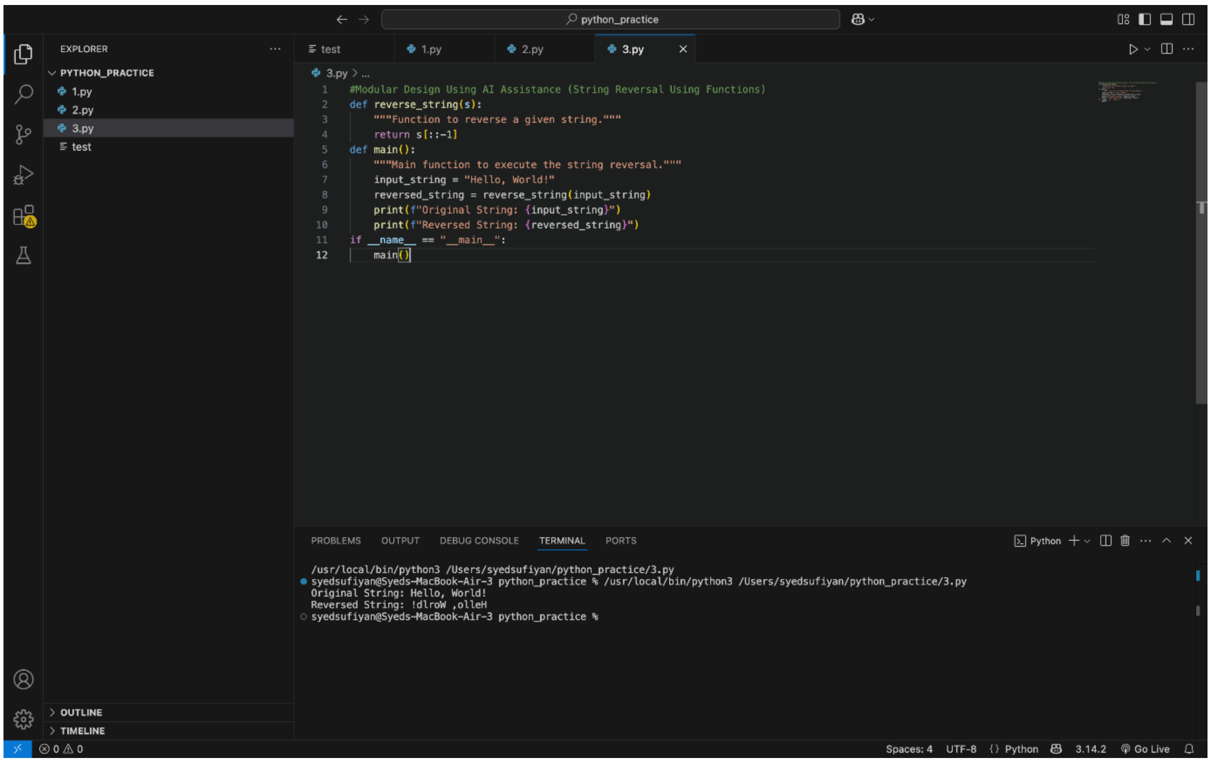
OBSERVATION:

The function uses Python slicing to reverse the string in a single step. No extra variables or loops are used, which makes the code easy to read. The logic is efficient and executes faster than manual reversal methods. Overall, the code is clean, readable, and suitable for review by other developers. The optimized approach reduces unnecessary operations and improves performance. It follows Python best practices, making the code more maintainable and reliable.

TASK:3

PROMPT: MODULAR DESIGN USING AI ASSISTANCE (STRING REVERSAL USING FUNCTIONS)

CODE:



```python
#Modular Design Using AI Assistance (String Reversal Using Functions)
def reverse_string(s):
    """Function to reverse a given string."""
    return s[::-1]
def main():
    """Main function to execute the string reversal."""
    input_string = "Hello, World!"
    reversed_string = reverse_string(input_string)
    print(f"Original String: {input_string}")
    print(f"Reversed String: {reversed_string}")
if __name__ == "__main__":
    main()
```

```
/usr/local/bin/python3 /Users/syedsufiyan/python_practice/3.py
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/3.py
Original String: Hello, World!
Reversed String: !dlroW ,olleH
syedsufiyan@Syeds-MacBook-Air-3 python_practice %
```

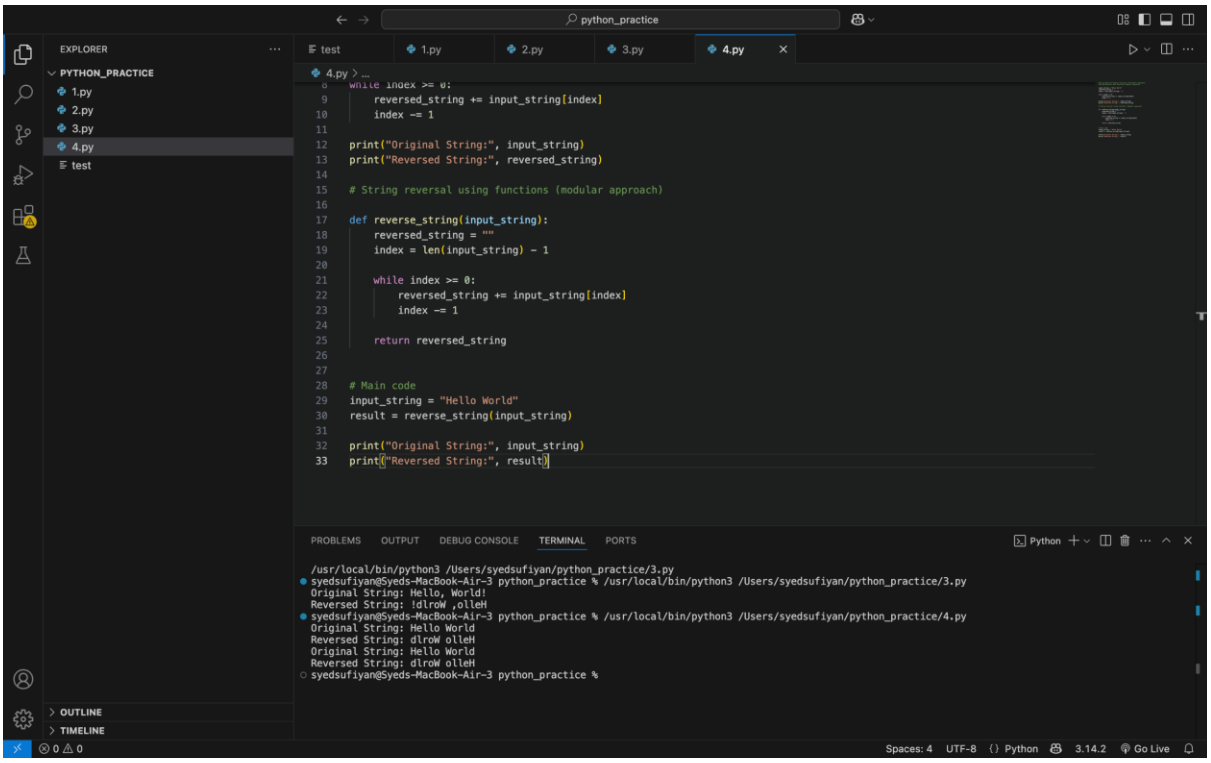
OBSERVATION:

This program follows a modular design by separating the string reversal logic into a reusable function. The use of clear function names and meaningful comments makes the code easy to understand and maintain. Since the reversal logic is written only once, it can be reused in multiple parts of the application without duplication. Overall, the structure improves readability, reusability, and makes future modifications simple.

TASK:4

PROMPT: COMPARATIVE ANALYSIS – PROCEDURAL VS MODULAR APPROACH (WITH VS WITHOUT FUNCTIONS)

CODE:



```python
    while index >= 0:
        reversed_string += input_string[index]
        index -= 1

print("Original String:", input_string)
print("Reversed String:", reversed_string)

# String reversal using functions (modular approach)

def reverse_string(input_string):
    reversed_string = ""
    index = len(input_string) - 1

    while index >= 0:
        reversed_string += input_string[index]
        index -= 1

    return reversed_string


# Main code
input_string = "Hello World"
result = reverse_string(input_string)

print("Original String:", input_string)
print("Reversed String:", result)
```

```
/usr/local/bin/python3 /Users/syedsufiyan/python_practice/3.py
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/3.py
Original String: Hello, World!
Reversed String: !dlroW ,olleH
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/4.py
Original String: Hello World
Reversed String: dlroW olleH
Original String: Hello World
Reversed String: dlroW olleH
syedsufiyan@Syeds-MacBook-Air-3 python_practice %
```

OBSERVATION:

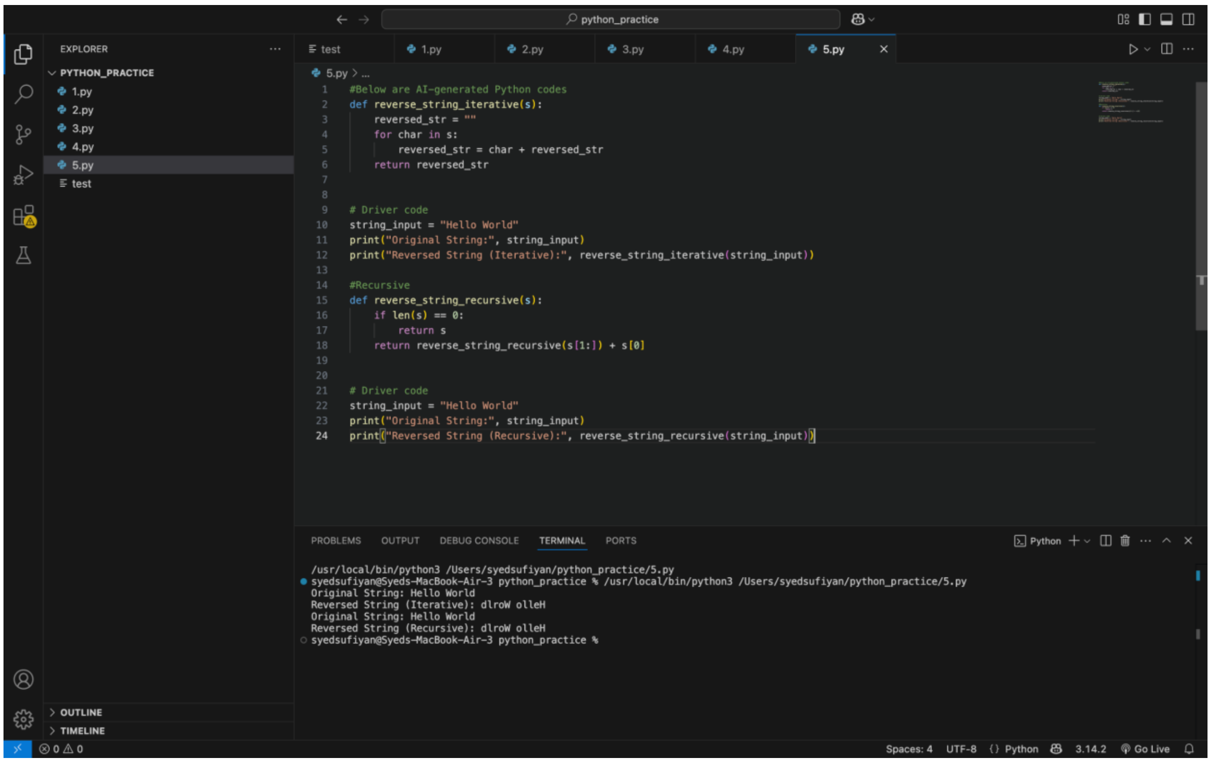
The procedural approach places all logic in one block, making the code harder to reuse and maintain. The modular approach separates logic into a function, improving clarity and structure. Functions allow easy reuse of code without duplication. Debugging is simpler in the modular approach because errors can be isolated. Overall, modular design is better suited for large and scalable applications

TASK:5

PROMPT:

AI-GENERATED PYTHON CODES ITERATIVE VS RECURSION

CODE:



```python
#Below are AI-generated Python codes
def reverse_string_iterative(s):
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str


# Driver code
string_input = "Hello World"
print("Original String:", string_input)
print("Reversed String (Iterative):", reverse_string_iterative(string_input))

#Recursive
def reverse_string_recursive(s):
    if len(s) == 0:
        return s
    return reverse_string_recursive(s[1:]) + s[0]


# Driver code
string_input = "Hello World"
print("Original String:", string_input)
print("Reversed String (Recursive):", reverse_string_recursive(string_input))
```

```
/usr/local/bin/python3 /Users/syedsufiyan/python_practice/5.py
syedsufiyan@Syeds-MacBook-Air-3 python_practice % /usr/local/bin/python3 /Users/syedsufiyan/python_practice/5.py
Original String: Hello World
Reversed String (Iterative): dlroW olleH
Original String: Hello World
Reversed String (Recursive): dlroW olleH
syedsufiyan@Syeds-MacBook-Air-3 python_practice %
```

OBSERVATION:

The iterative approach reverses the string by looping through each character, which makes the execution flow easy to follow but slightly slower due to repeated string concatenation. The recursive approach breaks the problem into smaller parts, which is conceptually clean but uses more memory because of function calls and stack usage. Both methods have linear time complexity, but recursion adds extra overhead. For large input strings, the iterative approach is generally safer and more efficient. The recursive method is better suited for learning and understanding recursion rather than performance-critical applications.