Assignment 1

Name: A. sathwik

Roll no: 2303A51178

Batch No: 29

Task 1: AI-Generated Logic Without Modularization (String Reversal Without Functions)

Prompt:

string Reversal Without Functions

```python
# String reversal without using functions like reverse() or [::-1]

string = input("Enter a string: ")
reversed_string = ""

for i in range(len(string) - 1, -1, -1):
    reversed_string += string[i]

print("Original string:", string)
print("Reversed string:", reversed_string)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python

PS C:\Users\adepu\OneDrive\Desktop\Ai assistant assignemnts> & C:/Python314/python.exe "c:/Users/adepu/OneDrive/Desktop/Ai assistant assignemnts/Assignmnet1.py"
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
```

Observation:

A Python program that correctly implements string reversal without using built-in functions such as reverse () or slicing ([::-1]). The program takes user input, iterates through the string from the last character to the first using a for loop with a negative step, and builds the reversed string by appending each character one by one. The execution output confirms that the logic works as expected, as the input hello world is accurately reversed to dlrow olleh. The code runs without any errors, follows the given constraints, and clearly demonstrates proper understanding of string indexing, loop control, and manual string manipulation in Python, making it suitable and correct for a basic programming assignment.

Task 2: Efficiency & Logic Optimization (Readability Improvement)

Prompt:

Simplify this string reversal code

```python
# String reversal without using functions like reverse() or [::-1]

string = input("Enter a string: ")
reversed_string = ""

for i in range(len(string) - 1, -1, -1):
    reversed_string += string[i]

print("Original string:", string)
print("Reversed string:", reversed_string)
reversed_string = string[::-1]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python
PS C:\Users\adepu\OneDrive\Desktop\Ai assistant assignemnts> & C:/Python314/python.exe "c:/Users/adepu/OneDrive/Desktop/Ai assistant assignemnts/Assignmnet1.py"
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
PS C:\Users\adepu\OneDrive\Desktop\Ai assistant assignemnts> & C:/Python314/python.exe "c:/Users/adepu/OneDrive/Desktop/Ai assistant assignemnts/Assignmnet1.py"
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
```

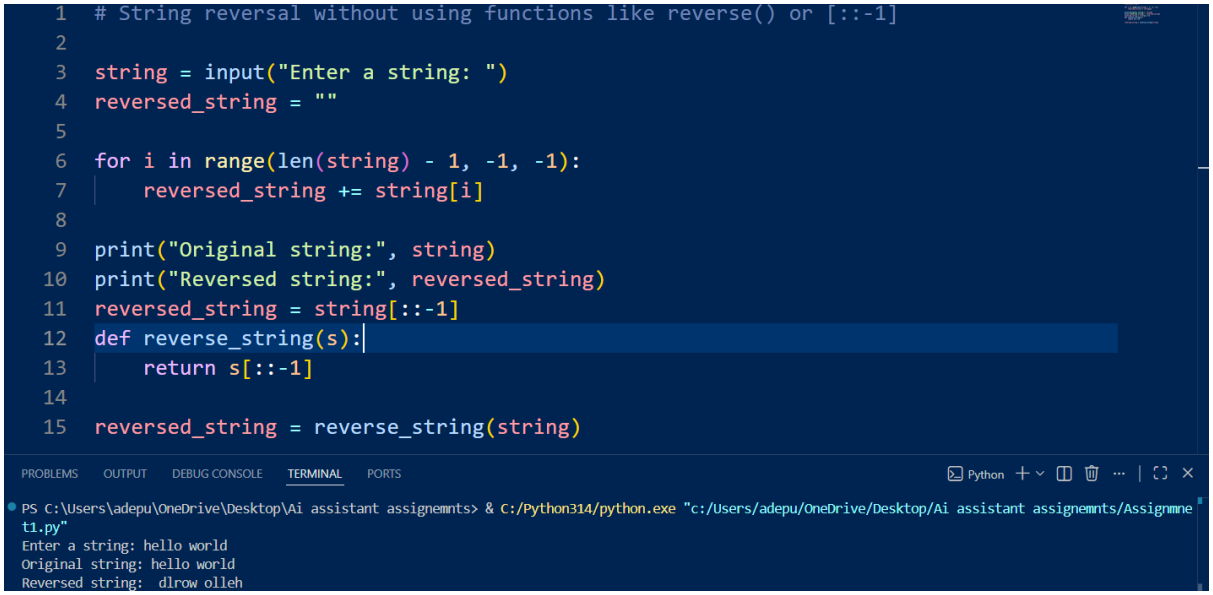
Observations:

A Python program intended to reverse a string without using built-in methods like reverse () or slicing, and the loop-based logic correctly performs the reversal by iterating from the last character to the first and constructing the reversed string manually. The output confirms that the string hello world is reversed correctly to dlrow olleh, demonstrating that the loop-based approach works as expected. However, the presence of the statement reversed_string = string [: -1] at the end contradicts the stated constraint, as slicing is a built-in shortcut for reversing strings. Although this line does not affect the displayed output because it is placed after the print statements, it conceptually violates the problem requirement and should be removed to keep the solution fully aligned with the objective of manual string reversal.

Task 3: Modular Design Using AI Assistance (String Reversal Using Functions)

Prompt:

String Reversal Using Functions



Observation:

A Python program where multiple approaches to string reversal are written within the same script. The loop-based method correctly reverses the input string by iterating from the last character to the first and produces the expected output, as seen in the terminal where hello world is reversed to dlrow olleh. However, after printing the result, the code additionally includes a slicing-based reversal and a function that also uses slicing ([::-1]), which contradicts the original comment stating that built-in reversal techniques should not be used. Although these extra lines do not change the displayed output because they appear after the print statements, they introduce redundancy and violate the stated constraint. Overall, the program demonstrates correct manual reversal logic, but it should retain only the loop-based approach to remain consistent, clean, and aligned with the problem requirement.

Task4: Comparative Analysis – Procedural vs Modular Approach (With vs

Without Functions)

Prompt:

With vs Without Functions

```python
# String reversal without using functions or slicing

string = input("Enter a string: ")
reversed_string = ""

for i in range(len(string) - 1, -1, -1):
    reversed_string += string[i]

print("Original string:", string)
print("Reversed string:", reversed_string)

# String reversal using a function (no slicing, no built-in reverse)

def reverse_string(s):
    result = ""
    for i in range(len(s) - 1, -1, -1):
        result += s[i]
    return result

string = input("Enter a string: ")
reversed_string = reverse_string(string)

print("Original string:", string)
print("Reversed string:", reversed_string)
```

```
PS C:\Users\adepu\OneDrive\Desktop\Ai assistant assignemnts> & C:/Python314/python.exe "c:/Users/adepu/OneDrive/Desktop/Ai assistant assignemnts/Assignmne
t1.py"
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
Enter a string: hello world
Original string: hello world
Reversed string:  dlrow olleh
```

Observation:

codes correctly demonstrate two different design approaches for solving the same problem of string reversal without using built-in reverse methods. In the procedural version, the logic is written in a single continuous flow, which makes it simple and easy to understand for beginners, but tightly couples input, processing, and output, limiting reusability and scalability. In contrast, the modular version separates the core reversal logic into a dedicated function, improving code readability, reusability, and maintainability while producing the same correct output. Both programs generate accurate results, but the function-based approach reflects better programming practices and is more suitable for debugging and extension in larger applications, whereas the procedural approach is best suited for small, one-time tasks or learning purposes.

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal)

Prompt:

Different Algorithmic Approaches to String Reversal

```python
# String reversal using recursion

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

string = input("Enter a string: ")
reversed_string = reverse_string_recursive(string)

print("Original string:", string)
print("Reversed string:", reversed_string)

# String reversal using a stack (list)

def reverse_string_stack(s):
    stack = list(s)
    result = ""
    while stack:
        result += stack.pop()
    return result

string = input("Enter a string: ")
reversed_string = reverse_string_stack(string)
```

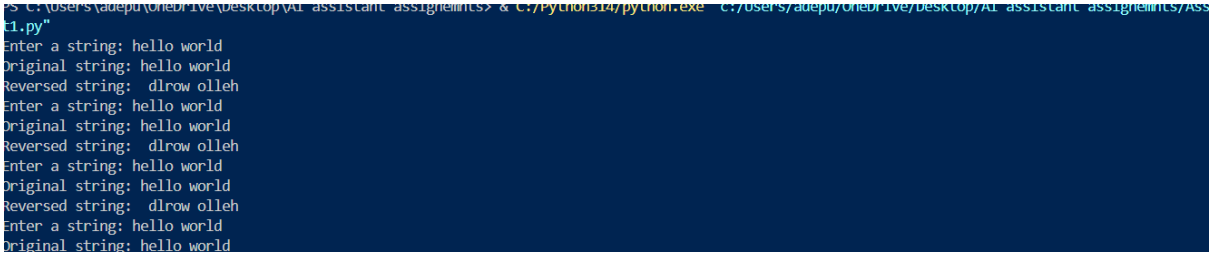
```python
print("Original string:", string)
print("Reversed string:", reversed_string)

# String reversal using two pointers

def reverse_string_two_pointers(s):
    s_list = list(s)
    left, right = 0, len(s_list) - 1
    while left < right:
        s_list[left], s_list[right] = s_list[right], s_list[left]
        left += 1
        right -= 1
    return "".join(s_list)

string = input("Enter a string: ")
reversed_string = reverse_string_two_pointers(string)

print("Original string:", string)
print("Reversed string:", reversed_string)
```



Observation:

Demonstrates three different algorithmic approaches to reverse a string—recursion, stack-based processing, and the two-pointer technique—each correctly producing the reversed output while highlighting different problem-solving paradigms. The recursive method illustrates a clear mathematical and logical approach by breaking the string into smaller subproblems, though it may be inefficient for very long strings due to function call overhead and stack limitations. The stack-based approach leverages the LIFO property of lists to reverse the string iteratively, offering better control over execution flow but using additional memory. The two-pointer method is the most efficient in terms of performance, as it reverses the string in place using index swapping with minimal extra space. Overall, the code effectively compares multiple strategies for the same task, demonstrating a strong understanding of data structures, recursion, and algorithmic efficiency, and shows how different approaches can be chosen based on performance and scalability requirements.