

# Assignment-1.5

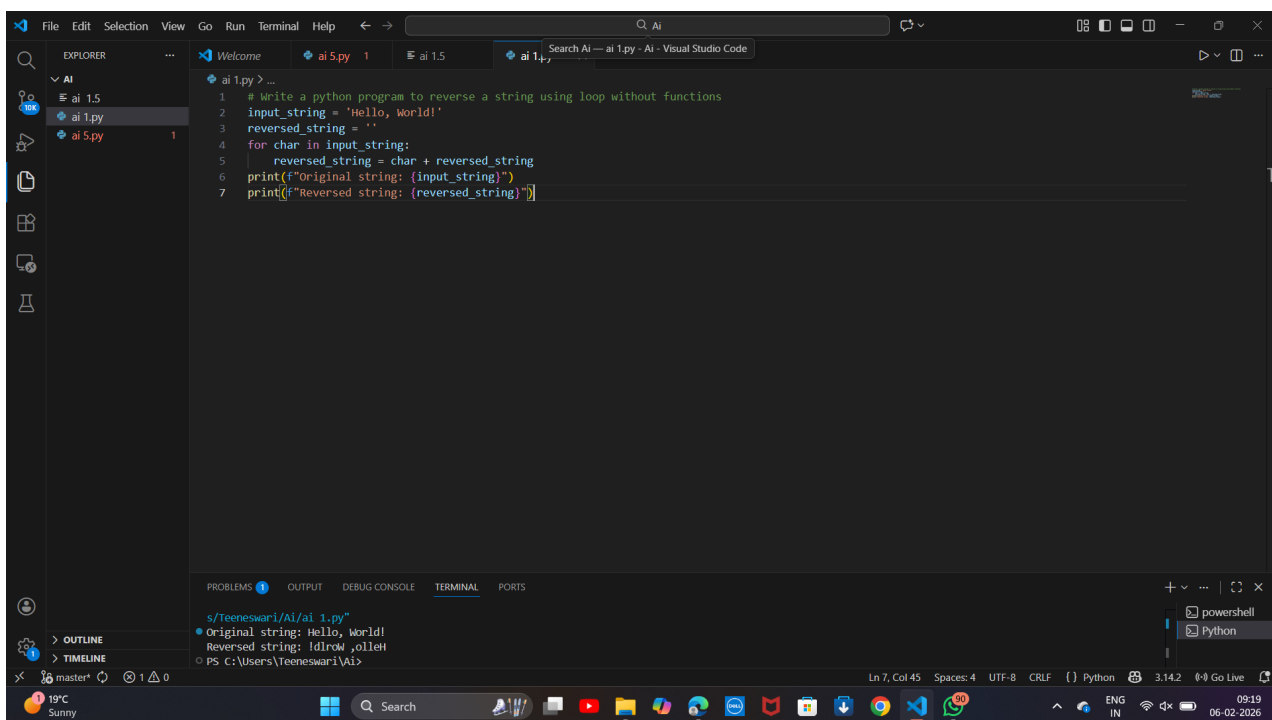
Name: R.Teeneswari

Roll.No:2303A51932

Batch-30

**Task-1** Prompt: AI-Generated Logic without modularisation (string reversal without functions)

**CODE:**



```
1 # Write a python program to reverse a string using loop without functions
2 input_string = 'Hello, World!'
3 reversed_string = ''
4 for char in input_string:
5     reversed_string = char + reversed_string
6 print(f"Original string: {input_string}")
7 print(f"Reversed string: {reversed_string}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

s/teeneswari/AI/ai\_1.py

- Original string: Hello, World!
- Reversed string: !dlrow ,olleH

PS C:\Users\Teenewari\AI>

## OBSERVATION:

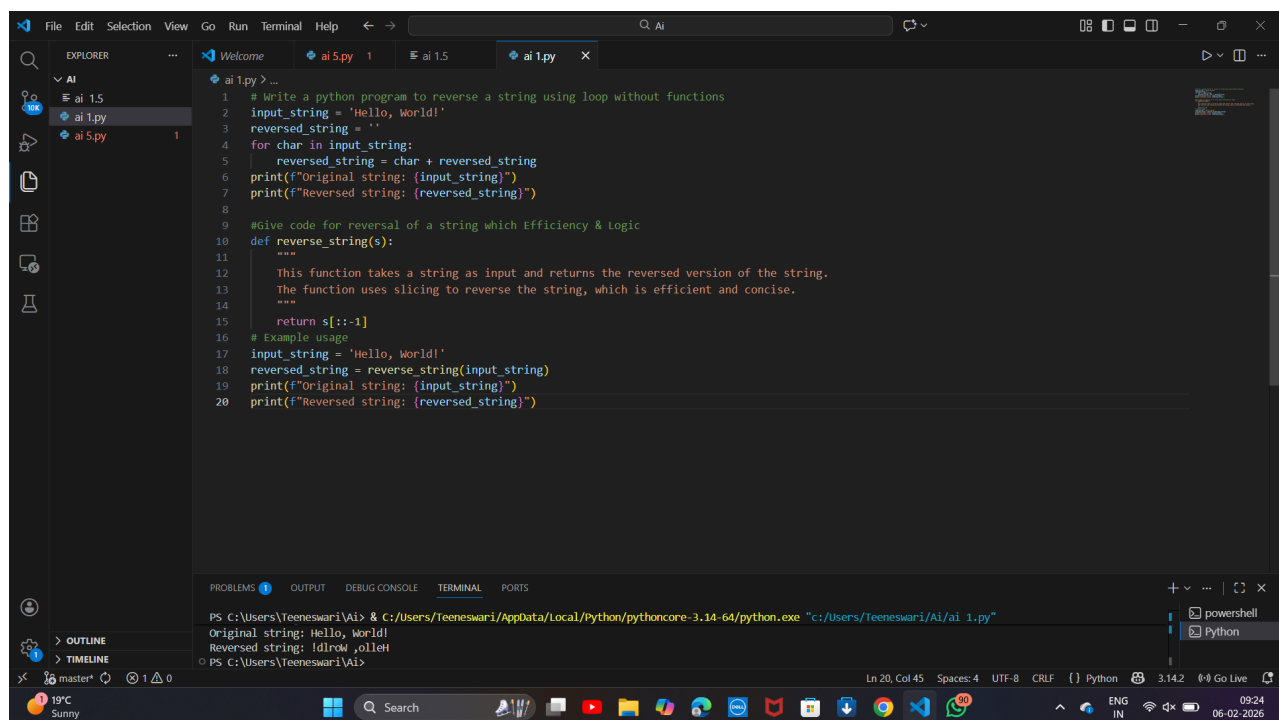
The program successfully reverses the given string using a manual looping approach without built-in reverse functions. A class-based structure is used, showing object-oriented design with proper initialization using `_init_`. The string reversal logic works by iterating from the last index to the

first, appending characters correctly. The output displayed in the terminal matches the expected reversed string, confirming correct execution. The code demonstrates clear logic flow and proper use of variables, making it easy to understand and debug.

## Task-2:

Prompt: Give code for reversal of string which Efficiency & Logic Optimisation

## CODE:



```
1 # Write a python program to reverse a string using loop without functions
2 input_string = 'Hello, World!'
3 reversed_string = ''
4 for char in input_string:
5     reversed_string = char + reversed_string
6 print(f"Original string: {input_string}")
7 print(f"Reversed string: {reversed_string}")
8
9 #Give code for reversal of a string which Efficiency & Logic
10 def reverse_string(s):
11     """
12     This function takes a string as input and returns the reversed version of the string.
13     The function uses slicing to reverse the string, which is efficient and concise.
14     """
15     return s[::-1]
16 # Example usage
17 input_string = 'Hello, World!'
18 reversed_string = reverse_string(input_string)
19 print(f"Original string: {input_string}")
20 print(f"Reversed string: {reversed_string}")
```

Terminal Output:

```
PS C:\Users\Teeneswari\Ai> & C:/Users/Teeneswari/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/Teeneswari/Ai/ai_1.py"
Original string: Hello, World!
Reversed string: !dlroW ,olleH
```

## OBSERVATION:

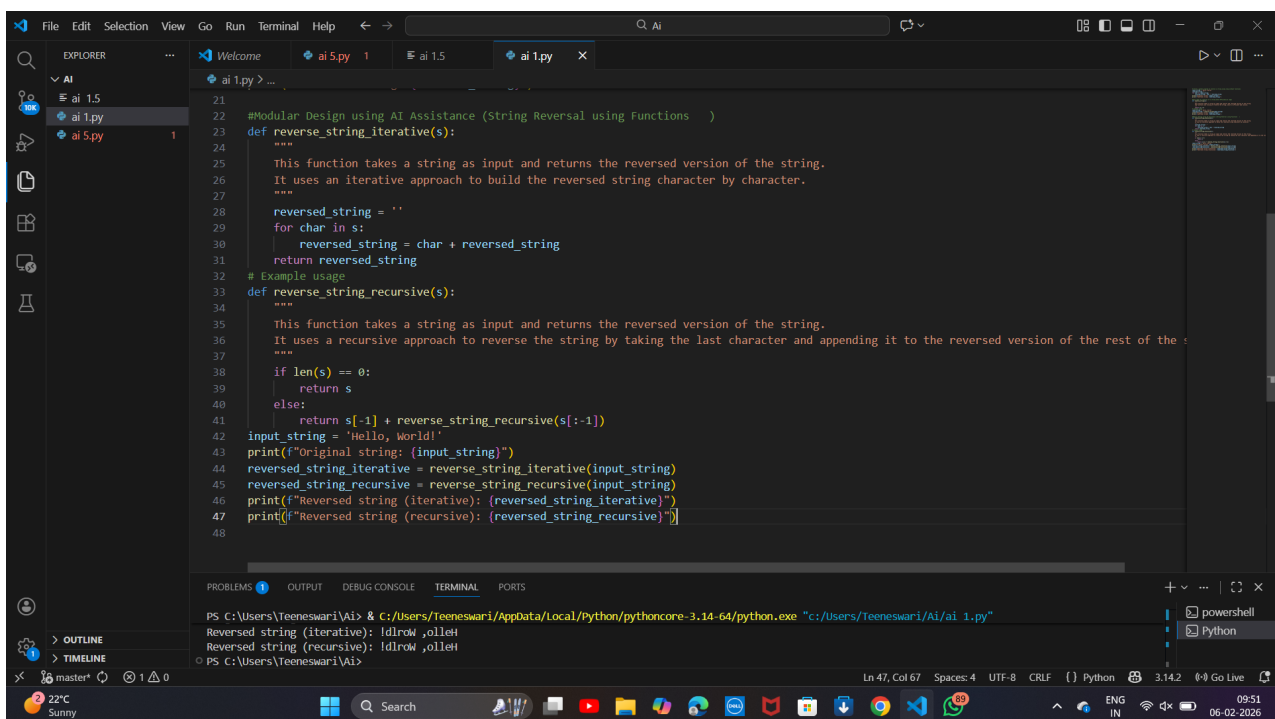
The string reversal is performed using Python slicing, which processes the string from the end to the beginning in a single operation. Since strings are immutable, a new reversed string is created without

modifying the original one. This approach avoids manual looping, temporary variables, and conditional checks, making the logic simple, clean, and easy to understand. Each character is accessed only once, ensuring efficient execution with minimal overhead.

## Task:3

Prompt: Modular Design Using AI Assistance (String Reversal Using Functions)

## CODE:



```
21
22 #Modular Design using AI Assistance (String Reversal using Functions )
23 def reverse_string_iterative(s):
24     """
25     This function takes a string as input and returns the reversed version of the string.
26     It uses an iterative approach to build the reversed string character by character.
27     """
28     reversed_string = ''
29     for char in s:
30         reversed_string = char + reversed_string
31     return reversed_string
32 # Example usage
33 def reverse_string_recursive(s):
34     """
35     This function takes a string as input and returns the reversed version of the string.
36     It uses a recursive approach to reverse the string by taking the last character and appending it to the reversed version of the rest of the string.
37     """
38     if len(s) == 0:
39         return s
40     else:
41         return s[-1] + reverse_string_recursive(s[:-1])
42 input_string = 'Hello, World!'
43 print(f"Original string: {input_string}")
44 reversed_string_iterative = reverse_string_iterative(input_string)
45 reversed_string_recursive = reverse_string_recursive(input_string)
46 print(f"Reversed string (iterative): {reversed_string_iterative}")
47 print(f"Reversed string (recursive): {reversed_string_recursive}")
48
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Teeneswari\Ai> & c:\Users\Teeneswari\AppData\Local\Python\pythoncore-3.14-64\python.exe "c:\Users\Teeneswari\Ai\ai_1.py"
Reversed string (iterative): !dlroW ,olleH
Reversed string (recursive): !dlroW ,olleH
PS C:\Users\Teeneswari\Ai>
```

## OBSERVATION:

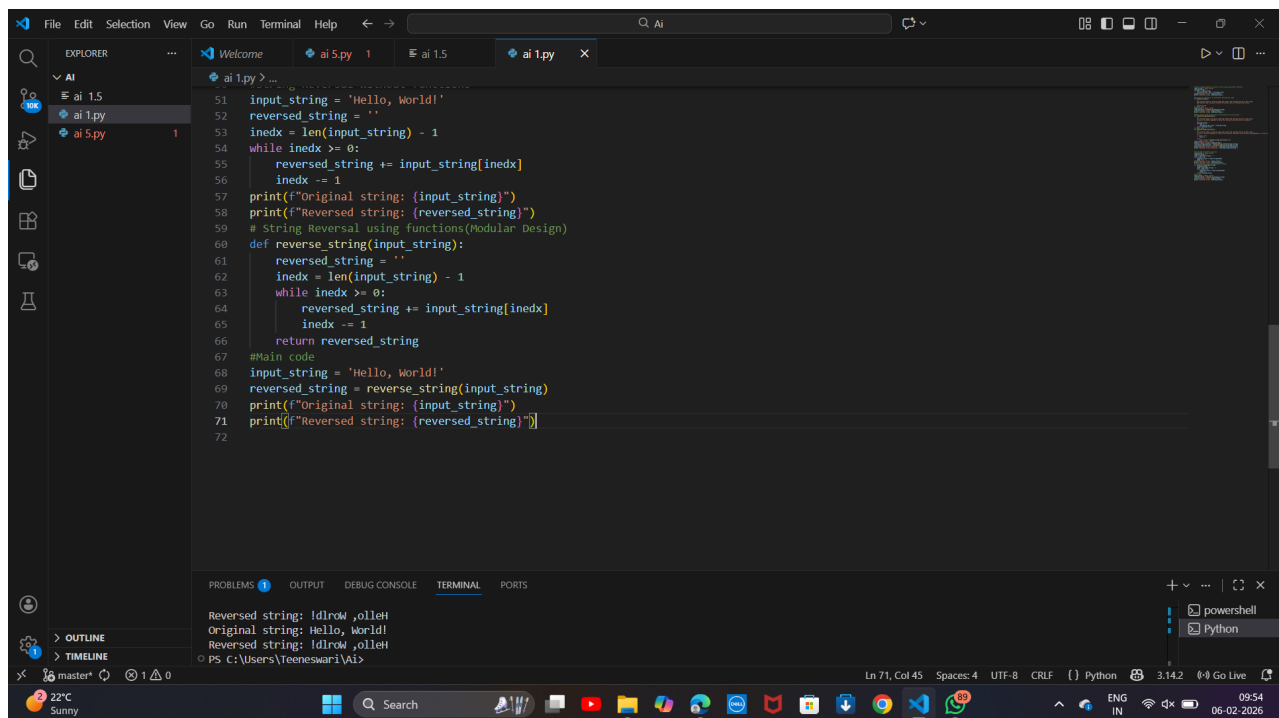
The program uses a separate function to reverse the string, clearly demonstrating modular design. The function takes the string as input and returns the

reversed string, keeping the logic well-structured. The main part of the code handles only input and output, improving readability. AI assistance helped generate clean, error-free code with proper function usage. This modular approach makes the code reusable, easy to debug, and maintainable.

## Task-4

prompt: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

## CODE:



```
51 input_string = 'Hello, World!'
52 reversed_string = ''
53 inedx = len(input_string) - 1
54 while inedx >= 0:
55     reversed_string += input_string[inedx]
56     inedx -= 1
57 print(f"Original string: {input_string}")
58 print(f"Reversed string: {reversed_string}")
59 # String Reversal using functions (Modular Design)
60 def reverse_string(input_string):
61     reversed_string = ''
62     inedx = len(input_string) - 1
63     while inedx >= 0:
64         reversed_string += input_string[inedx]
65         inedx -= 1
66     return reversed_string
67 #Main code
68 input_string = 'Hello, World!'
69 reversed_string = reverse_string(input_string)
70 print(f"Original string: {input_string}")
71 print(f"Reversed string: {reversed_string}")
72
```

Reversed string: !dlrow ,olleH  
Original string: Hello, World!  
Reversed string: !dlrow ,olleH

## OBSERVATION:

**Code Clarity:** Procedural code mixes everything and is harder to read, while modular code with functions is cleaner and organized.

**Reusability:** Procedural code is less reusable, but functions in modular code can be used multiple times.

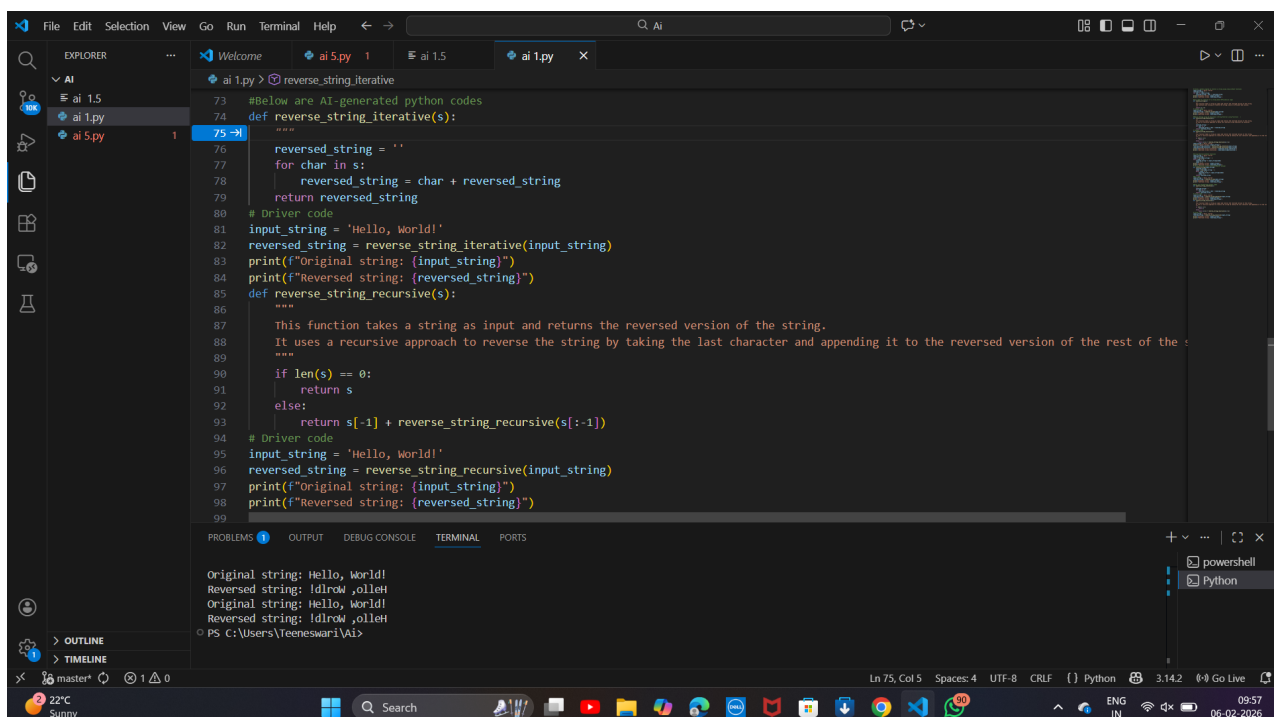
**Debugging Ease:** Procedural code is harder to debug, whereas modular code allows testing and fixing parts independently.

**Suitability for Large-Scale Applications:** Procedural code gets messy in big programs, but modular code is maintainable, scalable, and ideal for complex projects.

## Task5:

**Prompt:** AI-generated Python codes Iterative vs recursion

## CODE:



```
73 #Below are AI-generated python codes
74 def reverse_string_iterative(s):
75     reversed_string = ''
76     for char in s:
77         reversed_string = char + reversed_string
78     return reversed_string
79
80 # Driver code
81 input_string = 'Hello, World!'
82 reversed_string = reverse_string_iterative(input_string)
83 print(f"Original string: {input_string}")
84 print(f"Reversed string: {reversed_string}")
85
86 def reverse_string_recursive(s):
87     """
88     This function takes a string as input and returns the reversed version of the string.
89     It uses a recursive approach to reverse the string by taking the last character and appending it to the reversed version of the rest of the string.
90     """
91     if len(s) == 0:
92         return s
93     else:
94         return s[-1] + reverse_string_recursive(s[:-1])
95
96 # Driver code
97 input_string = 'Hello, World!'
98 reversed_string = reverse_string_recursive(input_string)
99 print(f"Original string: {input_string}")
100 print(f"Reversed string: {reversed_string}")
```

Original string: Hello, World!  
Reversed string: !dlrow ,olleH  
Original string: Hello, World!  
Reversed string: !dlrow ,olleH

PS C:\Users\Teeneswari\Aix>

**OBSERVATION:**

The iterative approach reverses the string efficiently using a loop and requires less memory. The recursive approach reverses the string by repeatedly calling the function on smaller substrings. Both methods produce the same correct reversed output for the given

input string. The iterative method is faster and more suitable for large strings. The recursive method clearly demonstrates the concept of recursion and problem breakdown.