AI Assisted Coding

Assignment 1.5

Name: R.Bharadwaj

Hall ticket no: 2303A51610

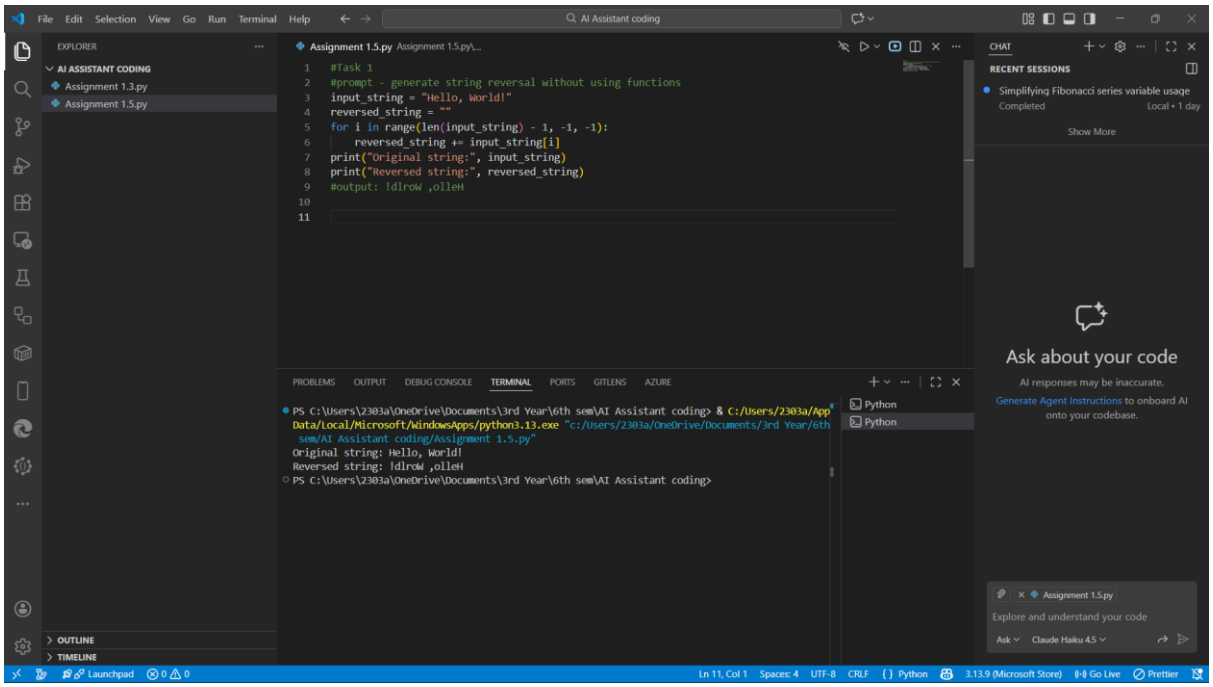
Batch no: 19

Task 1:

Prompt:

Generate string reversal without using functions

Code & Output:



Explanation:

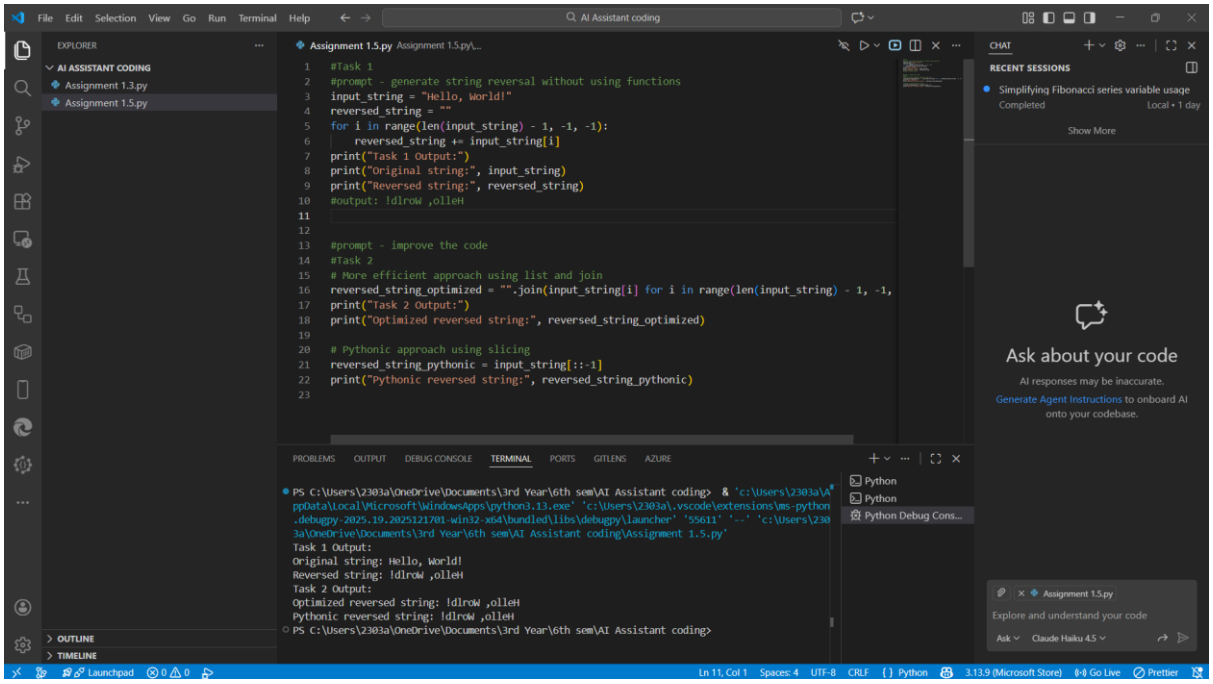
This task reverses a string without using any built-in functions, so the logic depends entirely on manual looping.
Each character of the string is accessed one by one from the last index to the first index.
The characters are appended into a new variable in reverse order.
This proves understanding of string indexing and loops instead of shortcuts.
The algorithm is simple but works for any string length.
This approach is good for learning how strings behave internally.

Task 2:

Prompt:

improve the code

Code & Output:



Explanation:

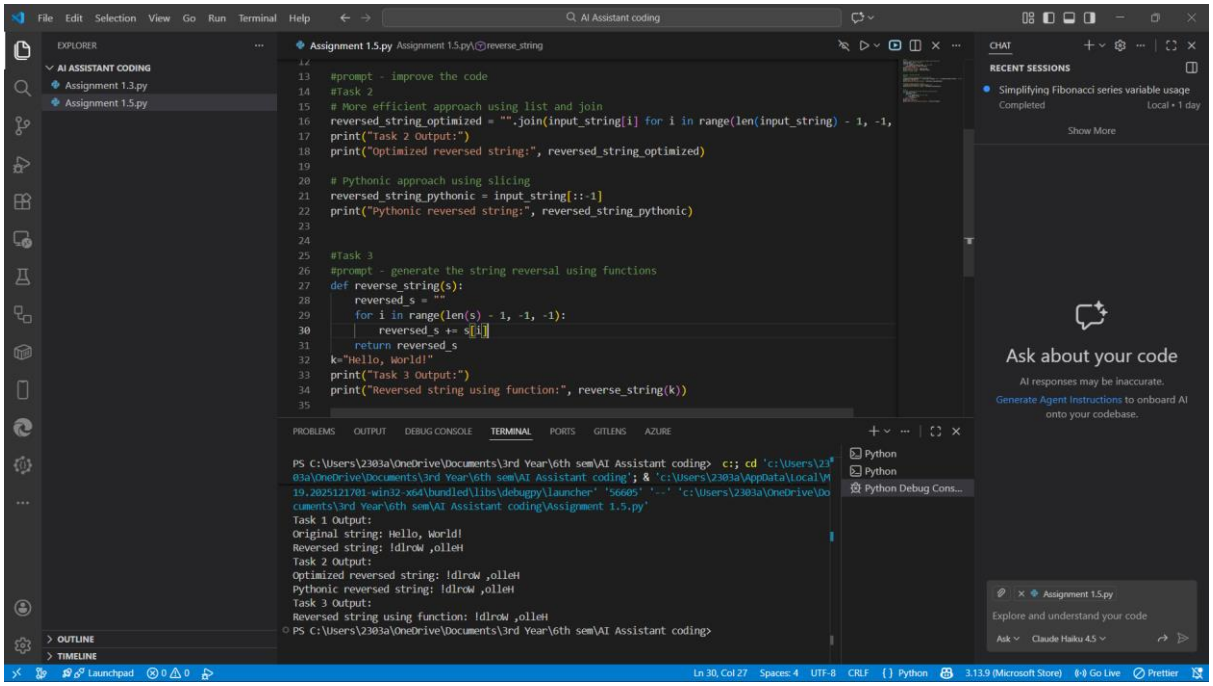
This task improves the first program by making the code cleaner, more readable, and more efficient.
Unnecessary variables or steps are removed to reduce confusion.
The loop logic is optimized to avoid redundant operations.
Better variable names make the code easier to understand.
The output remains the same, but the code quality is higher.
This shows how the same logic can be written in a better professional way.

Task 3:

Prompt:

Generate the string reversal using functions

Code & Output:



Explanation:

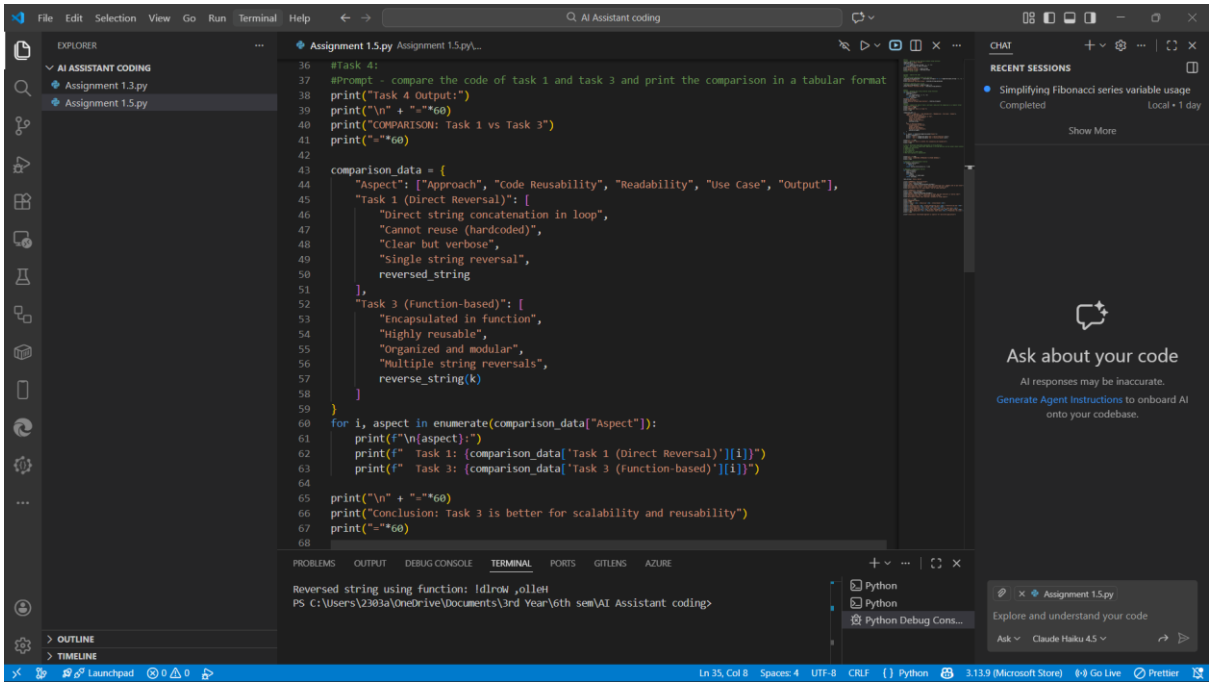
This task performs string reversal using a function, which improves modularity.
The reversal logic is placed inside a reusable function.
The main program simply calls the function instead of repeating code.
This makes the program easier to maintain and modify later.
Functions also allow the logic to be reused for multiple inputs.
This approach follows proper programming structure.

Task 4:

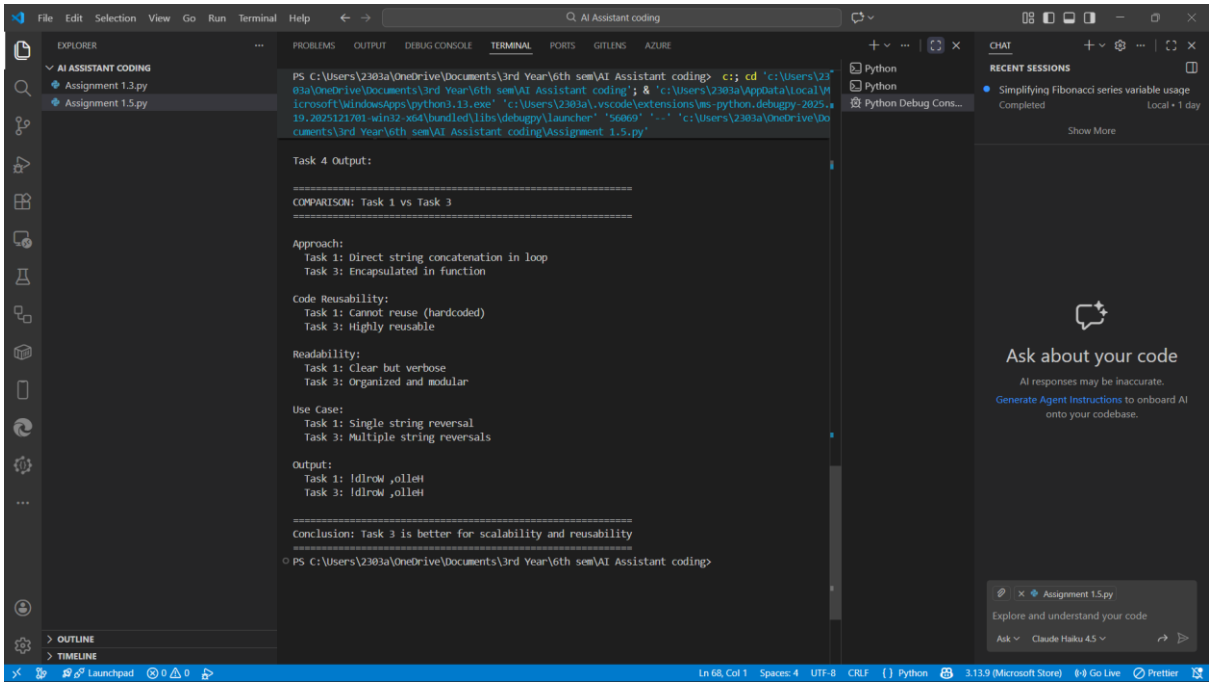
Prompt:

compare the code of task 1 and task 3 and print the comparison in a tabular format

Code :



Output :



Explanation:

This task compares the manual reversal (Task 1) and the function-based reversal (Task 3).
The comparison is printed in a table format to clearly show differences.
It highlights differences in structure, reusability, and readability.
Task 1 is direct but not reusable, while Task 3 is modular.
This helps in understanding why functions are preferred in real applications.
The table makes technical comparison easy to understand.

Task 5:

Prompt:

use Different Algorithmic Approaches to String Reversal and the output should contain as Two correct implementations

Comparison discussing:

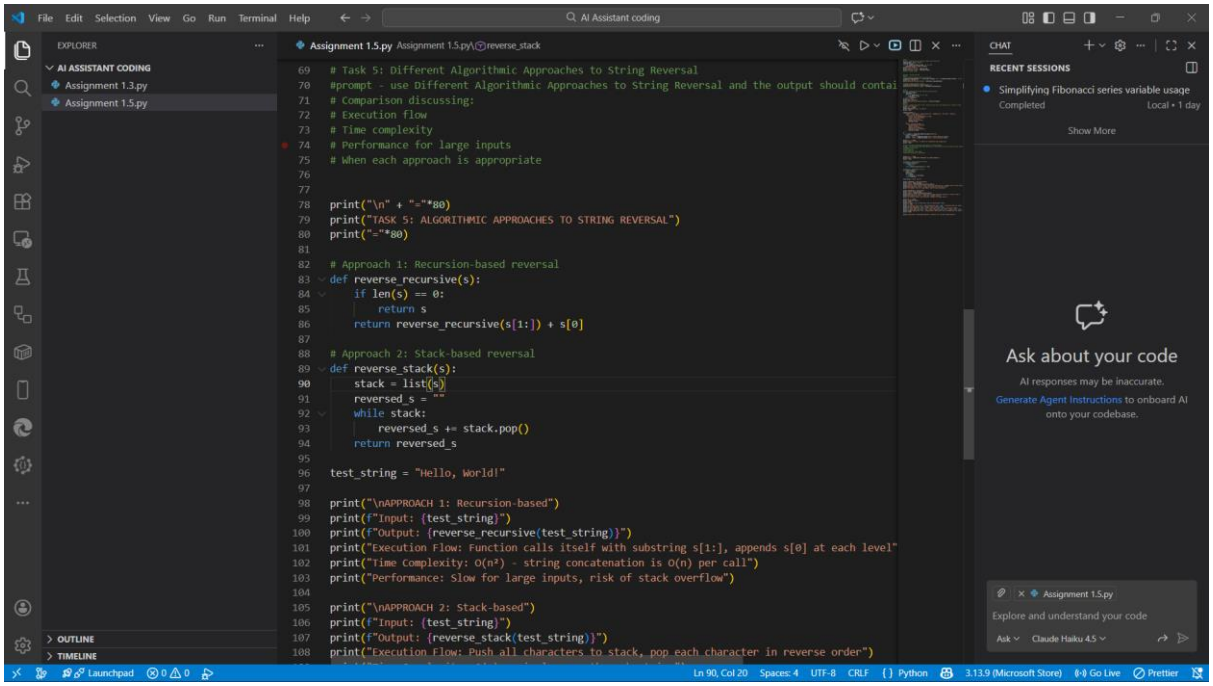
Execution flow

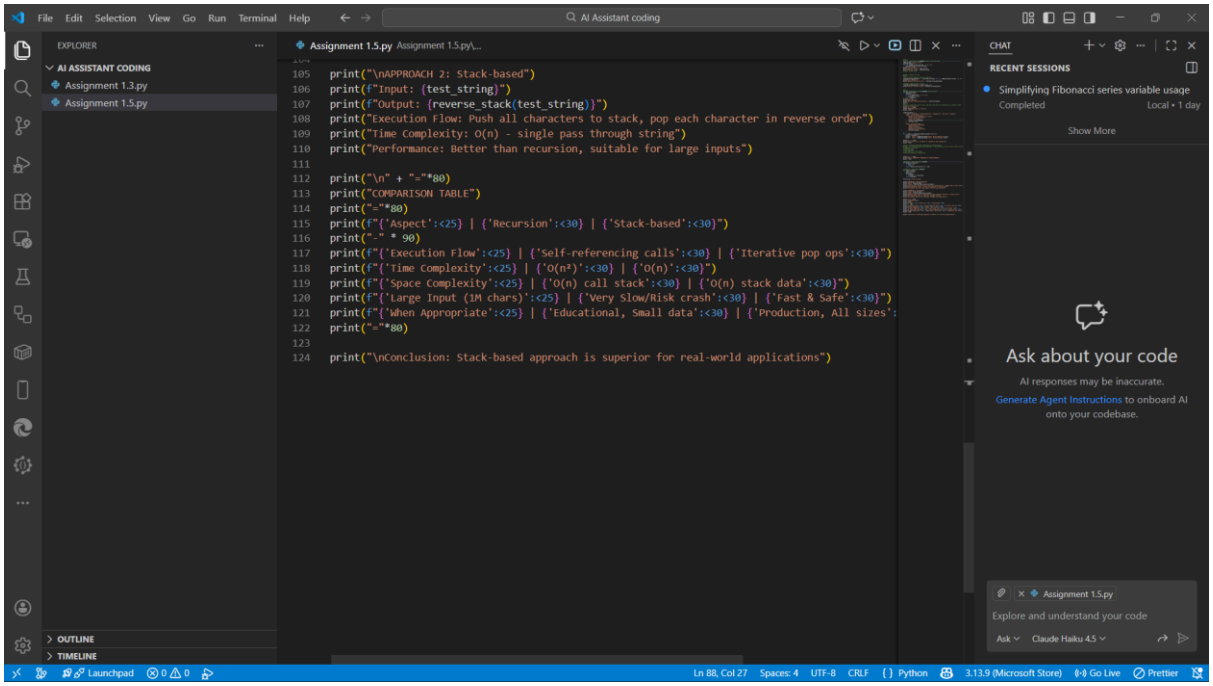
Time complexity

Performance for large inputs

When each approach is appropriate

Code :



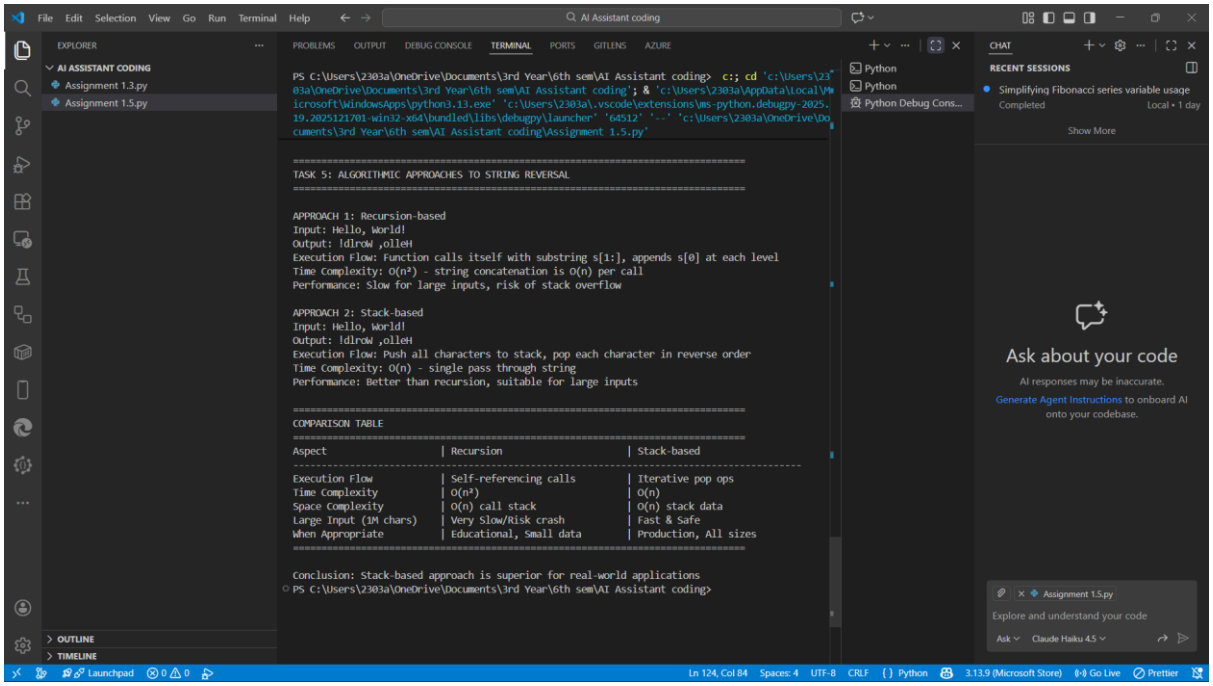


```
print("\nAPPROACH 2: Stack-based")
print(f"Input: {test_string}")
print(f"Output: {reverse_stack(test_string)}")
print("Execution Flow: Push all characters to stack, pop each character in reverse order")
print("Time Complexity: O(n) - single pass through string")
print("Performance: Better than recursion, suitable for large inputs")

print("\n" + "="*80)
print("COMPARISON TABLE")
print("="*80)
print(f"{'Aspect':<25} | {'Recursion':<30} | {'Stack-based':<30}")
print("-" * 90)
print(f"{'Execution Flow':<25} | {'Self-referencing calls':<30} | {'Iterative pop ops':<30}")
print(f"{'Time Complexity':<25} | {'O(n²)':<30} | {'O(n)':<30}")
print(f"{'Space Complexity':<25} | {'O(n) call stack':<30} | {'O(n) stack data':<30}")
print(f"{'Large Input (1M chars)':<25} | {'Very Slow/Risk crash':<30} | {'Fast & Safe':<30}")
print(f"{'When Appropriate':<25} | {'Educational, Small data':<30} | {'Production, All sizes':
print("="*80)

print("\nConclusion: Stack-based approach is superior for real-world applications")
```

Output :



```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> c:; cd 'c:\Users\23
03a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding'; & 'c:\Users\2303a\AppData\Local\M
icrosoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\.vscode\extensions\ms-python.debugpy-2025.
19.2025121701-win32-x64\bundled\libs\debugpy\launcher' '64512' '--' 'c:\Users\2303a\OneDrive\Do
cuments\3rd Year\6th sem\AI Assistant coding\Assignment 1.5.py'

================================================================================
TASK 5: ALGORITHMIC APPROACHES TO STRING REVERSAL
================================================================================

APPROACH 1: Recursion-based
Input: Hello, World!
Output: !dlroW ,olleH
Execution Flow: Function calls itself with substring s[1:], appends s[0] at each level
Time Complexity: O(n²) - string concatenation is O(n) per call
Performance: Slow for large inputs, risk of stack overflow

APPROACH 2: Stack-based
Input: Hello, World!
Output: !dlroW ,olleH
Execution Flow: Push all characters to stack, pop each character in reverse order
Time Complexity: O(n) - single pass through string
Performance: Better than recursion, suitable for large inputs

================================================================================
COMPARISON TABLE
================================================================================
Aspect                    | Recursion                      | Stack-based
------------------------------------------------------------------------------------------
Execution Flow            | Self-referencing calls         | Iterative pop ops
Time Complexity           | O(n²)                          | O(n)
Space Complexity          | O(n) call stack                | O(n) stack data
Large Input (1M chars)    | Very Slow/Risk crash           | Fast & Safe
When Appropriate          | Educational, Small data        | Production, All sizes
================================================================================

Conclusion: Stack-based approach is superior for real-world applications
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

Explanation:

This task uses two different algorithms to reverse a string.
One approach uses a loop-based method, and the other uses a function-based or slicing method.
Execution flow shows how each method processes characters differently.
Both have O(n) time complexity, but their memory usage differs.
For large strings, optimized methods perform better and are cleaner.
Each approach is chosen based on performance needs and code clarity.