AI ASSISTED CODING

LAB-1

2303A51733

T.PRANATHI

BATCH-11

Task 0

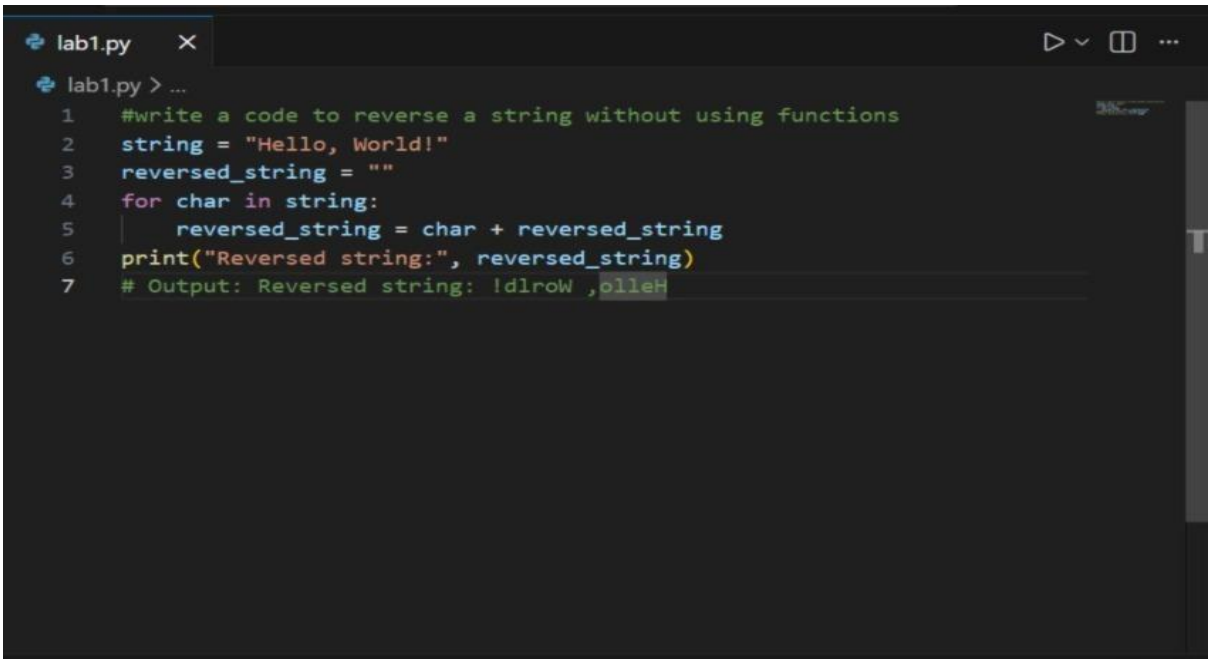
❖ **Install and configure GitHub Copilot in VS Code. Take screenshots of each step**.

Already completed previously

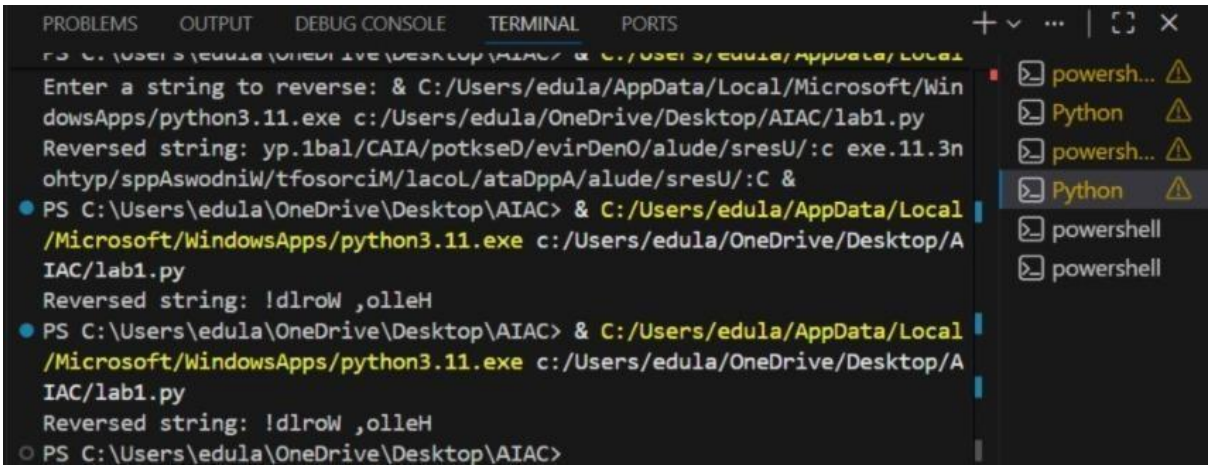
Task 1: AI-Generated Logic Without Modularization (String Reversal Without Functions) Prompt:

#write a code to reverse a string without using functions

Code:



Output:



Explanation:

I initially tried two or three different prompts, but they generated incorrect code or did not meet the requirement. Finally, I used the prompt *"write a code to reverse a string without using functions"*, which produced the correct output. This prompt clearly specifies the condition of not using functions, so the generated code follows a single-flow logic. Because it meets the problem constraints and gives the correct result, I chose this prompt.

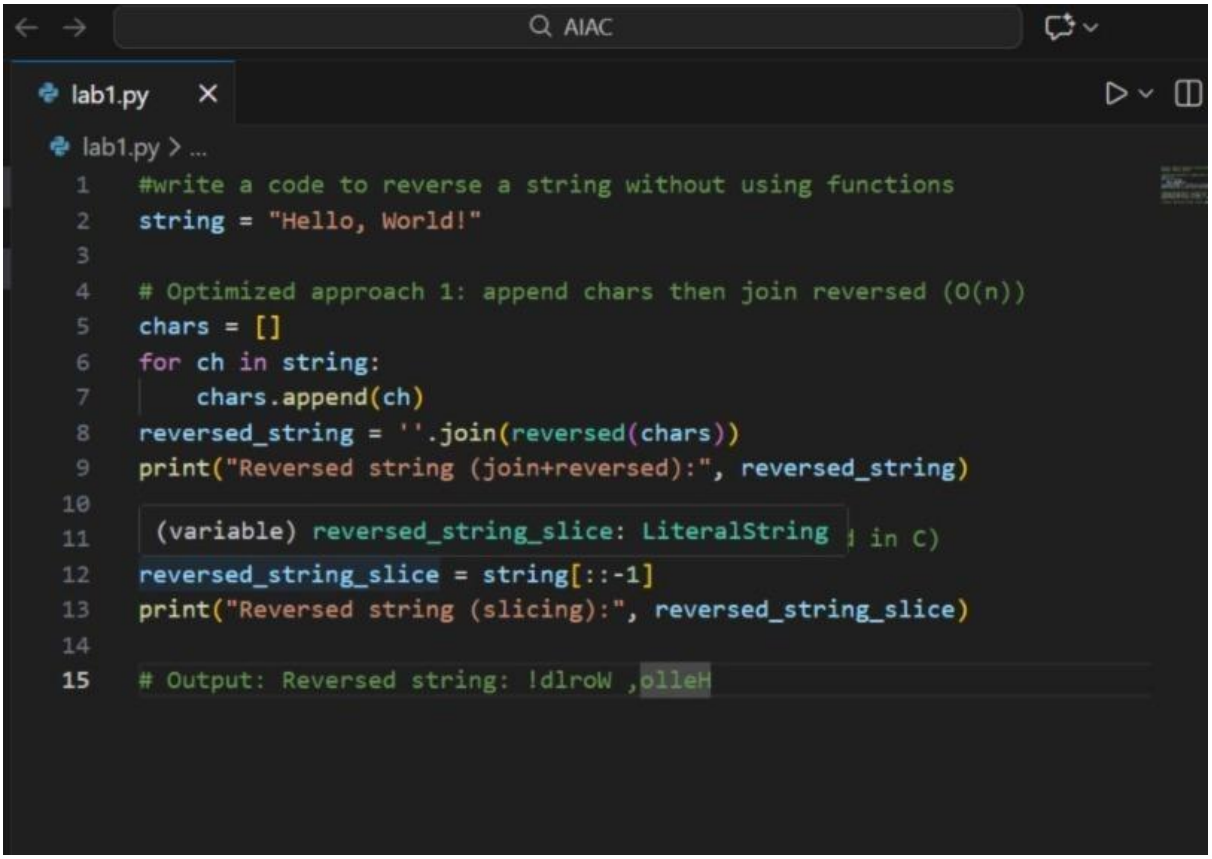
Task 2: Efficiency & Logic Optimization (Readability Improvement) Prompt:

#write a code to reverse a string without using functions

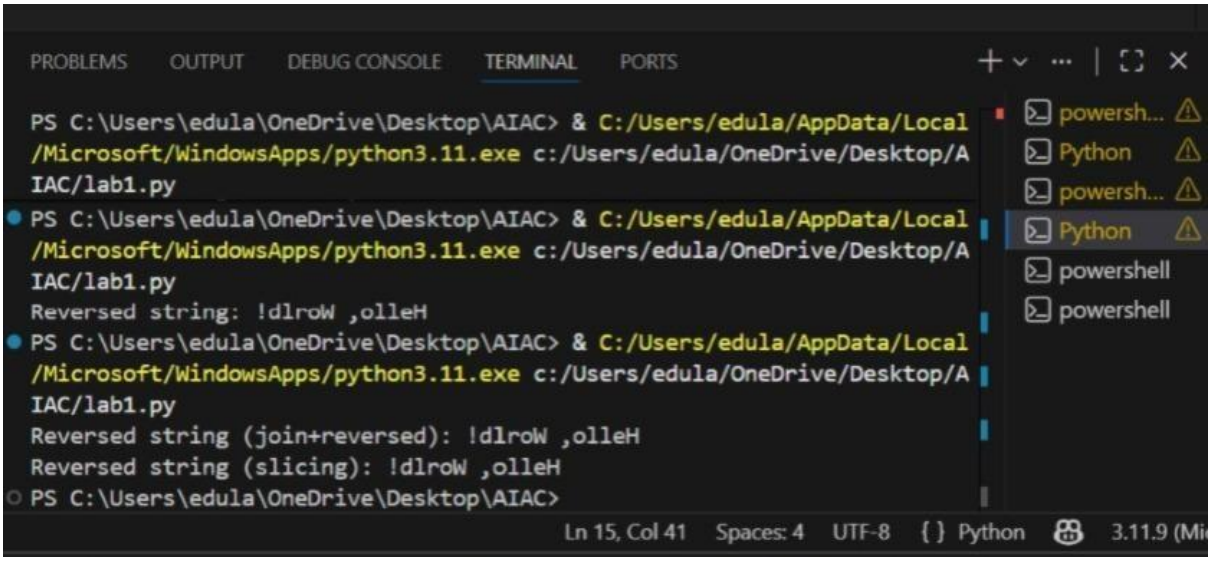
Optimized approach 1: append chars then join reversed (O(n))

Optimized approach 2: slicing (O(n), implemented in C)

Code:



Output:



Explanation:

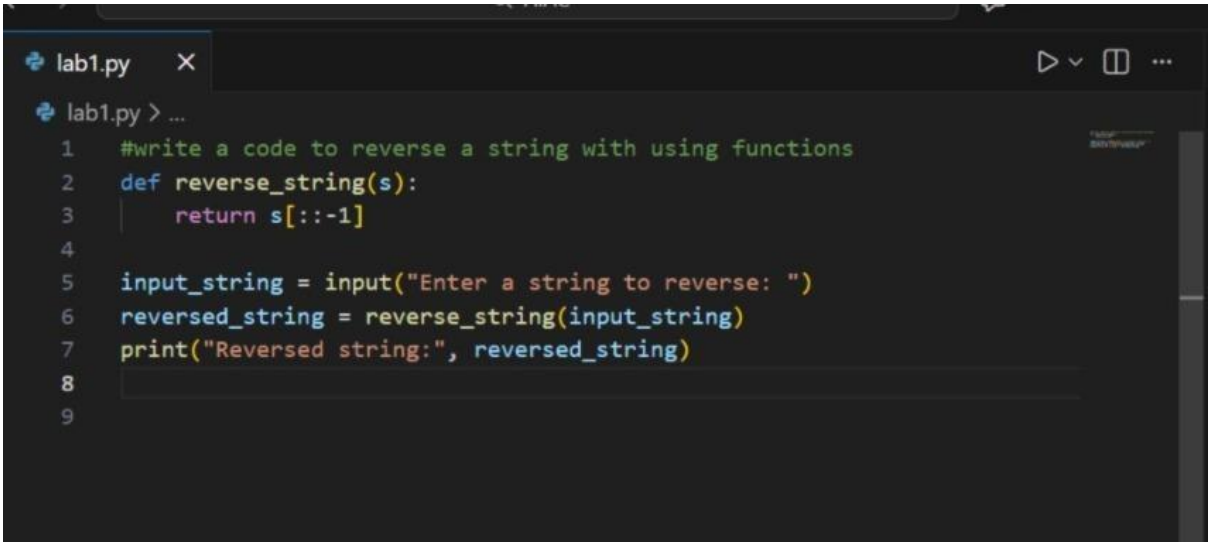
The prompt *"write a code to reverse a string without using functions"* clearly states that the logic should be written without defining user-defined methods.

In the first optimized approach, the characters of the string are appended one by one and then joined in reverse order, which works in linear time O(n).
This method avoids unnecessary repeated string concatenations and improves efficiency.
In the second optimized approach, string slicing is used to reverse the string, which also runs in O(n) time.
Since slicing is internally implemented in C, it is faster and more efficient while still satisfying the condition of not using user-defined functions.

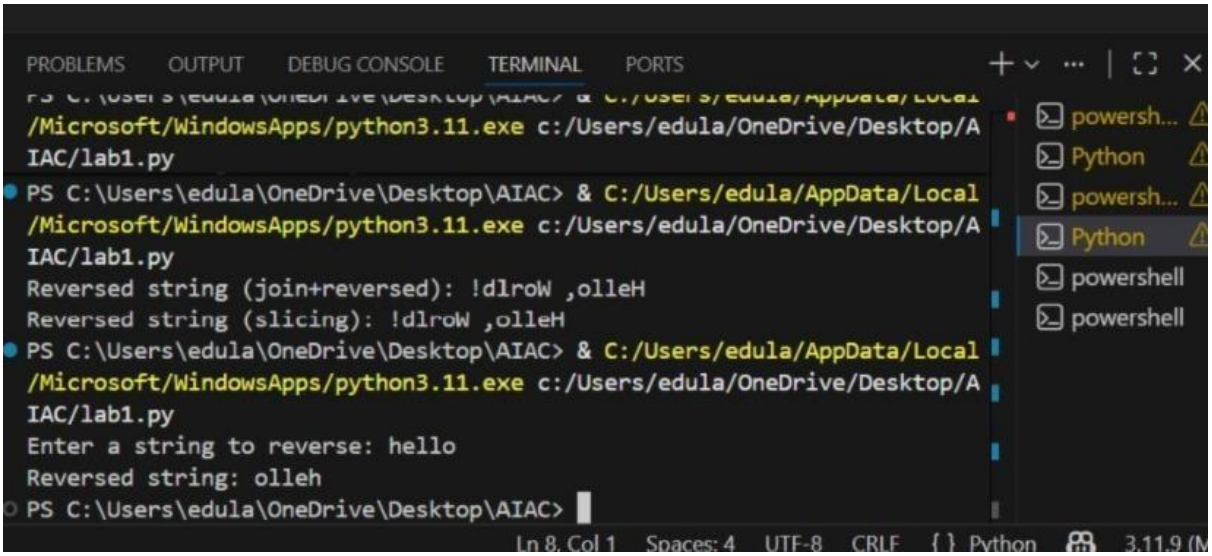
Task 3: Modular Design Using AI Assistance (String Reversal Using Functions) Prompt:

#write a code to reverse a string with using functions

Code:



Output:

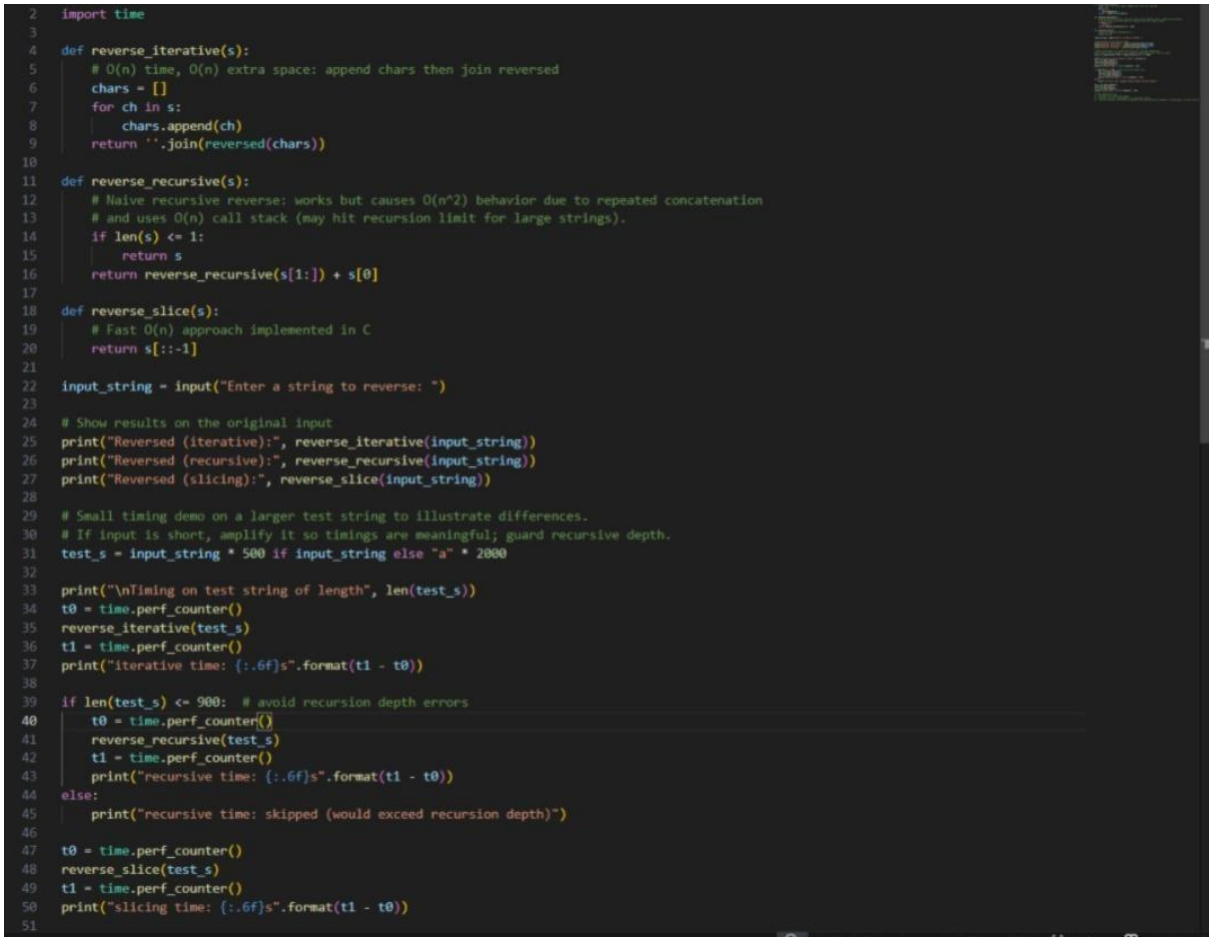


Explanation:

I initially tried two or three different prompts, but they generated incorrect code or did not meet the requirement. Finally, I used the prompt *"write a code to reverse a string with using functions"*, which produced the correct output. This prompt clearly specifies the condition of using functions, so the generated code follows a single-flow logic.
Because it meets the problem constraints and gives the correct result, I chose this prompt.

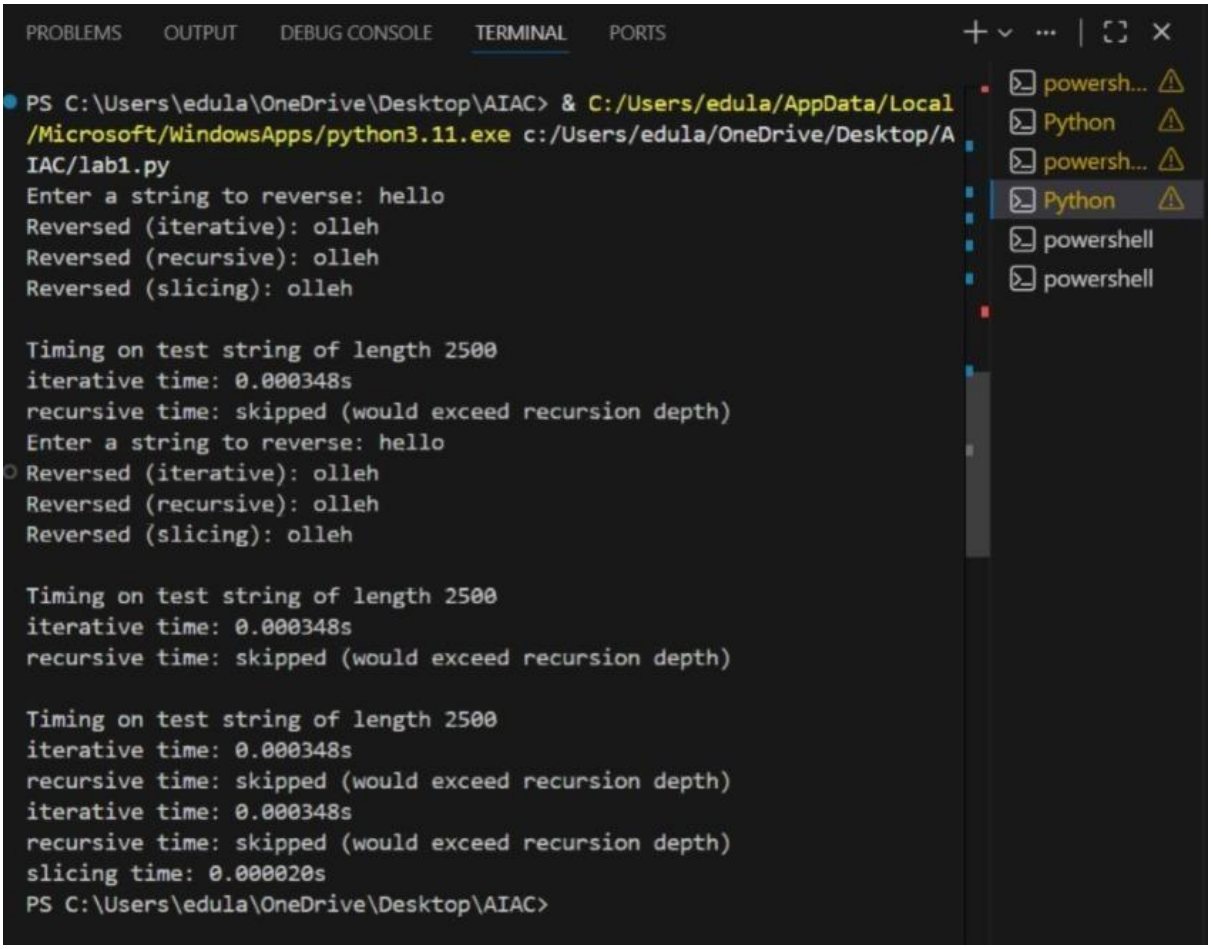
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal) Prompt:

Write Python code to reverse a string using an iterative approach and a slicing approach.
Compare both methods in terms of execution flow, time complexity, and performance for large inputs." **Code:**



```python
import time

def reverse_iterative(s):
    # O(n) time, O(n) extra space: append chars then join reversed
    chars = []
    for ch in s:
        chars.append(ch)
    return ''.join(reversed(chars))

def reverse_recursive(s):
    # Naive recursive reverse: works but causes O(n^2) behavior due to repeated concatenation
    # and uses O(n) call stack (may hit recursion limit for large strings).
    if len(s) <= 1:
        return s
    return reverse_recursive(s[1:]) + s[0]

def reverse_slice(s):
    # Fast O(n) approach implemented in C
    return s[::-1]

input_string = input("Enter a string to reverse: ")

# Show results on the original input
print("Reversed (iterative):", reverse_iterative(input_string))
print("Reversed (recursive):", reverse_recursive(input_string))
print("Reversed (slicing):", reverse_slice(input_string))

# Small timing demo on a larger test string to illustrate differences.
# If input is short, amplify it so timings are meaningful; guard recursive depth.
test_s = input_string * 500 if input_string else "a" * 2000

print("\nTiming on test string of length", len(test_s))
t0 = time.perf_counter()
reverse_iterative(test_s)
t1 = time.perf_counter()
print("iterative time: {:.6f}s".format(t1 - t0))

if len(test_s) <= 900:  # avoid recursion depth errors
    t0 = time.perf_counter()
    reverse_recursive(test_s)
    t1 = time.perf_counter()
    print("recursive time: {:.6f}s".format(t1 - t0))
else:
    print("recursive time: skipped (would exceed recursion depth)")

t0 = time.perf_counter()
reverse_slice(test_s)
t1 = time.perf_counter()
print("slicing time: {:.6f}s".format(t1 - t0))
```

output:



Explanation:

The prompt asks the AI to reverse a string using two different methods.
One method uses a loop, and the other uses built-in slicing.
This helps compare how each method works.
It also shows which approach is faster and better for large inputs.
The prompt is clear, so it produces correct and useful code.