Assignment – 1.5

Name: Sony Dodla

Batch:23

Ht No:2303A51666

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

❖ Scenario

You are developing a basic text-processing utility for a messaging application.

❖ Task Description

Use GitHub Copilot to generate a Python program that:

➢ Reverses a given string

➢ Accepts user input

➢ Implements the logic directly in the main code

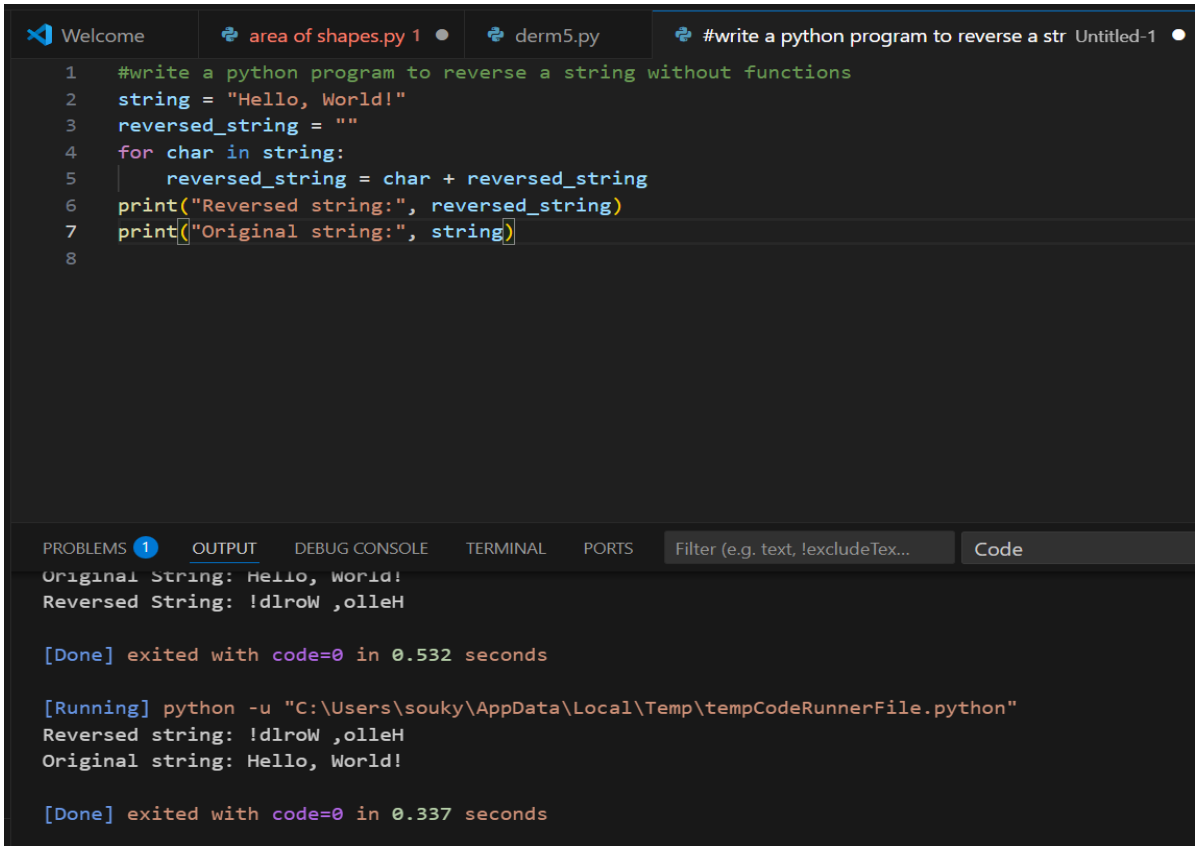
➢ Does not use any user-defined functions

❖ Expected Output

➢ Correct reversed string

➢ Screenshots showing Copilot-generated code suggestions

➢ Sample inputs and outputs



Task 2: Efficiency & Logic Optimization (Readability Improvement)

❖ Scenario

The code will be reviewed by other developers.

❖ Task Description

Examine the Copilot-generated code from Task 1 and improve it by:

➢ Removing unnecessary variables

➢ Simplifying loop or indexing logic

➢ Improving readability

➢ Use Copilot prompts like:

▪ “Simplify this string reversal code”

▪ “Improve readability and efficiency”

Hint:

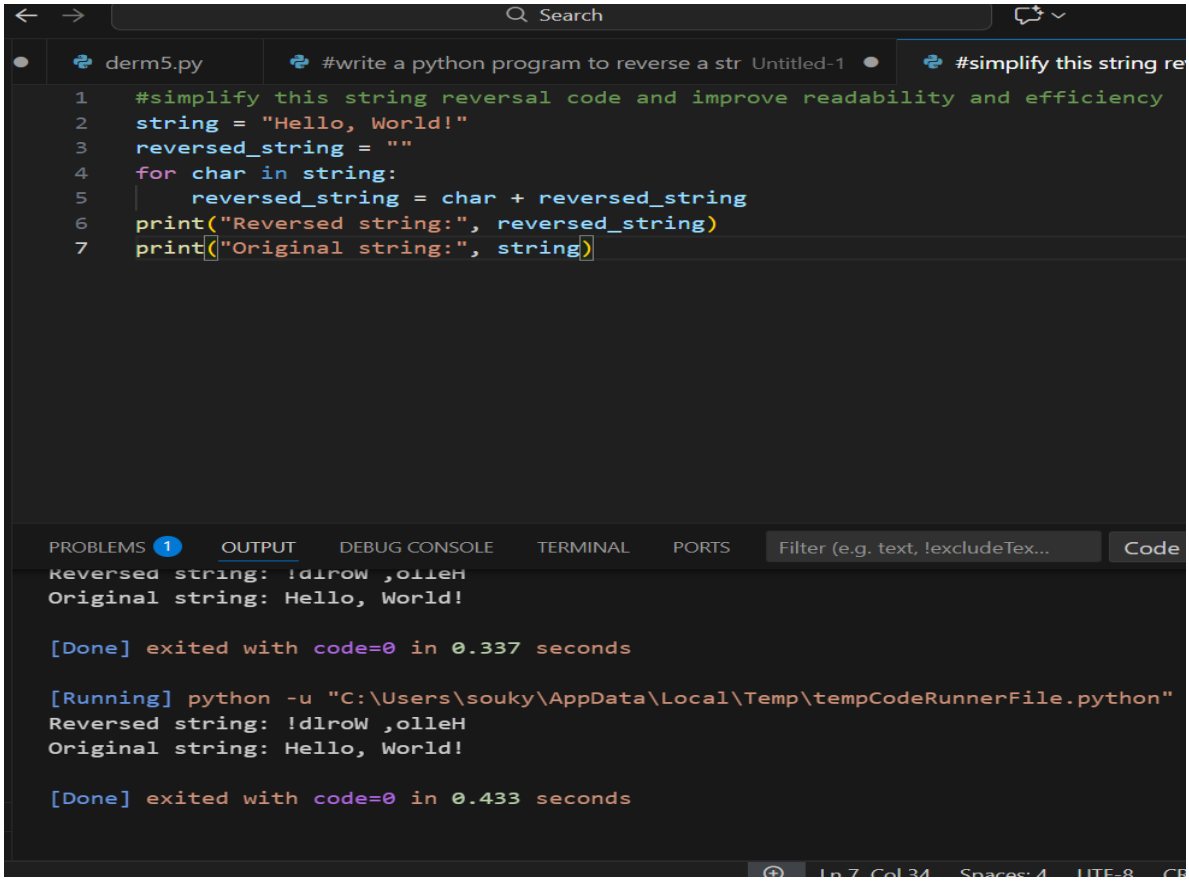
Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Expected Output

➢ Original and optimized code versions

➢ Explanation of how the improvements reduce time complexity



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

❖ Scenario

The string reversal logic is needed in multiple parts of an application.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

➢ Uses a user-defined function to reverse a string

➢ Returns the reversed string

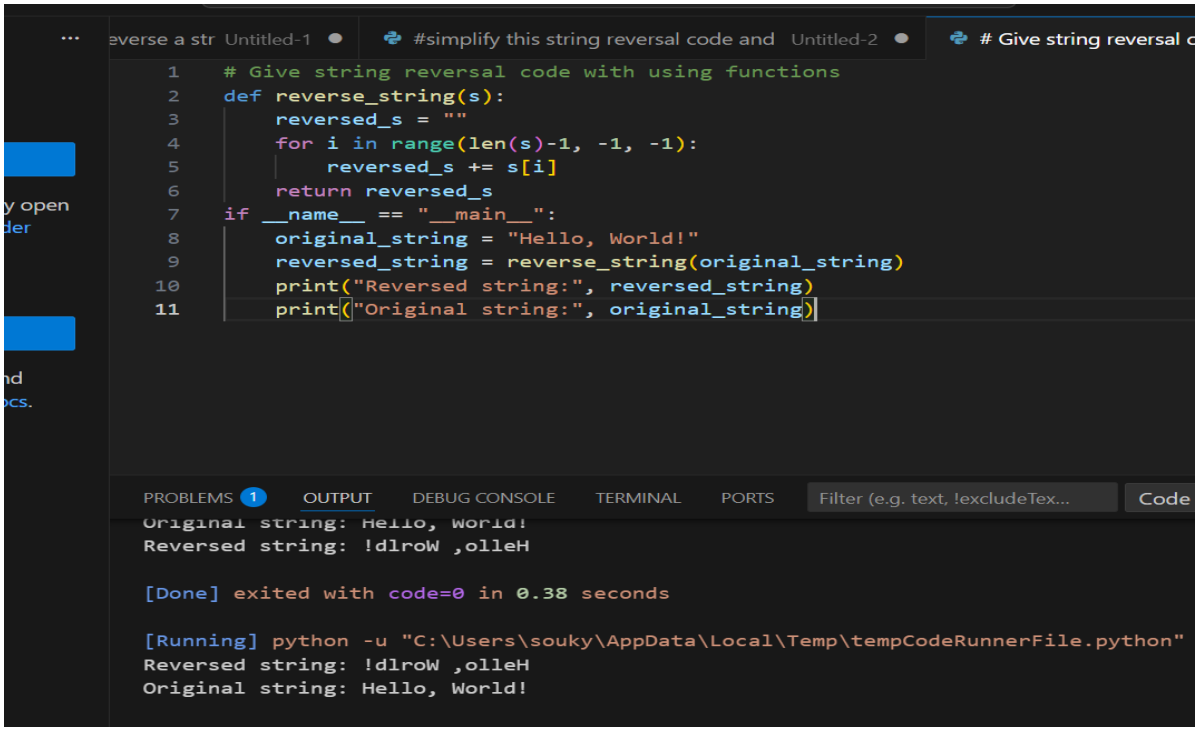
➢ Includes meaningful comments (AI-assisted)

❖ Expected Output

➢ Correct function-based implementation

➢ Screenshots documenting Copilot's function generation

➢ Sample test cases and outputs



Task 4: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

❖ Scenario

You are asked to justify design choices during a code review.

❖ Task Description

Compare the Copilot-generated programs:

➢ Without functions (Task 1)

➢ With functions (Task 3)

Analyze them based on:

➢ Code clarity

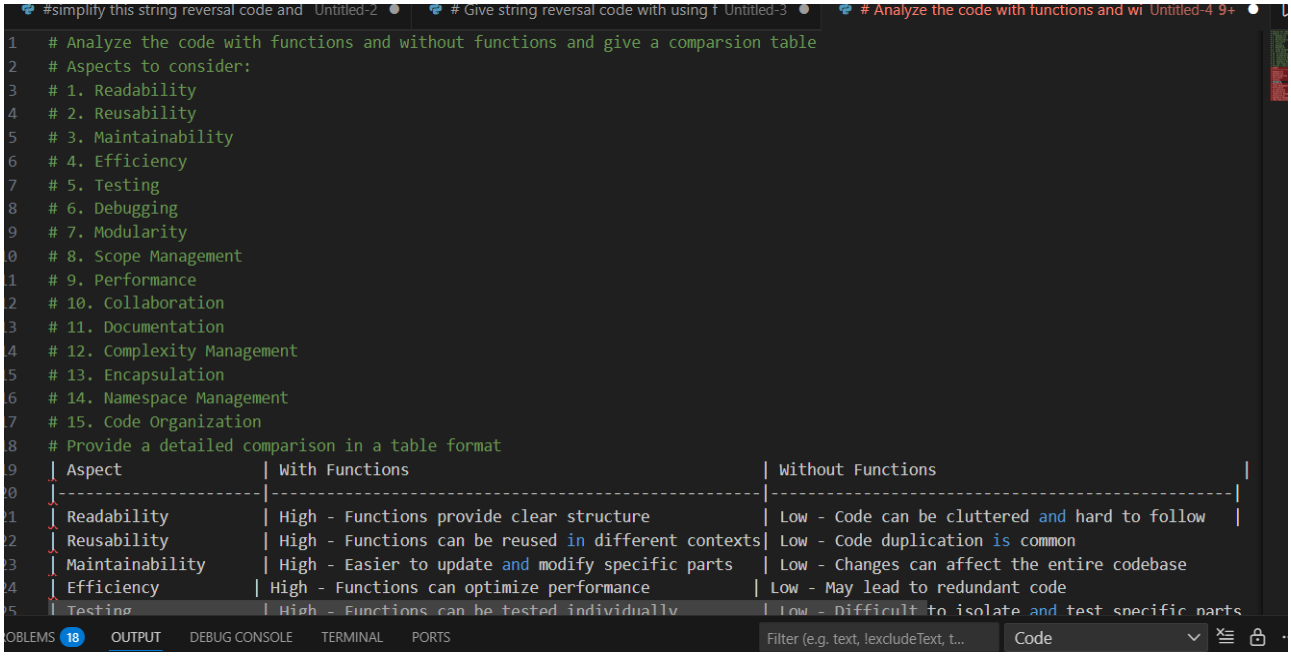
➢ Reusability

➢ Debugging ease

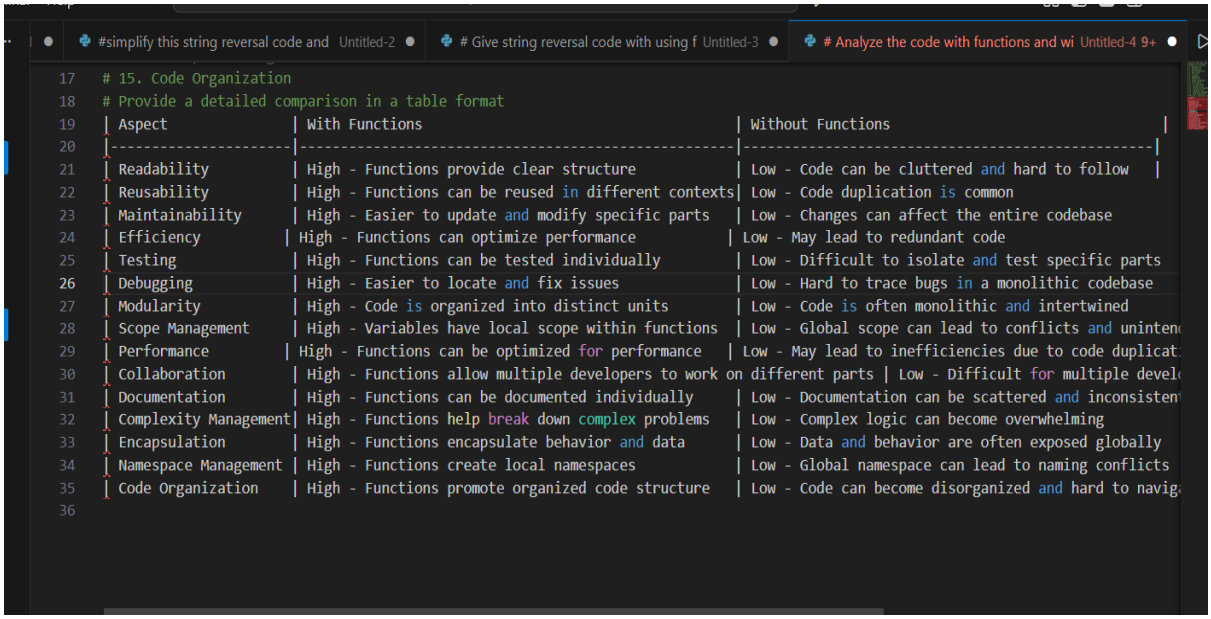
➢ Suitability for large-scale applications

❖ Expected Output

Comparison table or short analytical report



```
# Analyze the code with functions and without functions and give a comparsion table
# Aspects to consider:
# 1. Readability
# 2. Reusability
# 3. Maintainability
# 4. Efficiency
# 5. Testing
# 6. Debugging
# 7. Modularity
# 8. Scope Management
# 9. Performance
# 10. Collaboration
# 11. Documentation
# 12. Complexity Management
# 13. Encapsulation
# 14. Namespace Management
# 15. Code Organization
# Provide a detailed comparison in a table format
| Aspect              | With Functions                                      | Without Functions                                      |
|---------------------|-----------------------------------------------------|--------------------------------------------------------|
| Readability         | High - Functions provide clear structure            | Low - Code can be cluttered and hard to follow         |
| Reusability         | High - Functions can be reused in different contexts| Low - Code duplication is common                       |
| Maintainability     | High - Easier to update and modify specific parts   | Low - Changes can affect the entire codebase           |
| Efficiency          | High - Functions can optimize performance           | Low - May lead to redundant code                       |
| Testing             | High - Functions can be tested individually         | Low - Difficult to isolate and test specific parts     |
```



```
# 15. Code Organization
# Provide a detailed comparison in a table format
| Aspect              | With Functions                                      | Without Functions                                      |
|---------------------|-----------------------------------------------------|--------------------------------------------------------|
| Readability         | High - Functions provide clear structure            | Low - Code can be cluttered and hard to follow         |
| Reusability         | High - Functions can be reused in different contexts| Low - Code duplication is common                       |
| Maintainability     | High - Easier to update and modify specific parts   | Low - Changes can affect the entire codebase           |
| Efficiency          | High - Functions can optimize performance           | Low - May lead to redundant code                       |
| Testing             | High - Functions can be tested individually         | Low - Difficult to isolate and test specific parts     |
| Debugging           | High - Easier to locate and fix issues              | Low - Hard to trace bugs in a monolithic codebase      |
| Modularity          | High - Code is organized into distinct units        | Low - Code is often monolithic and intertwined         |
| Scope Management    | High - Variables have local scope within functions  | Low - Global scope can lead to conflicts and uninten   |
| Performance         | High - Functions can be optimized for performance   | Low - May lead to inefficiencies due to code duplicat  |
| Collaboration       | High - Functions allow multiple developers to work on different parts | Low - Difficult for multiple devel |
| Documentation       | High - Functions can be documented individually     | Low - Documentation can be scattered and inconsisten   |
| Complexity Management| High - Functions help break down complex problems  | Low - Complex logic can become overwhelming            |
| Encapsulation       | High - Functions encapsulate behavior and data      | Low - Data and behavior are often exposed globally     |
| Namespace Management| High - Functions create local namespaces            | Low - Global namespace can lead to naming conflicts    |
| Code Organization   | High - Functions promote organized code structure   | Low - Code can become disorganized and hard to navig   |
```

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

❖ Scenario

Your mentor wants to evaluate how AI handles alternative logic paths.

❖ Task Description

Prompt GitHub Copilot to generate:

➢ A loop-based string reversal approach

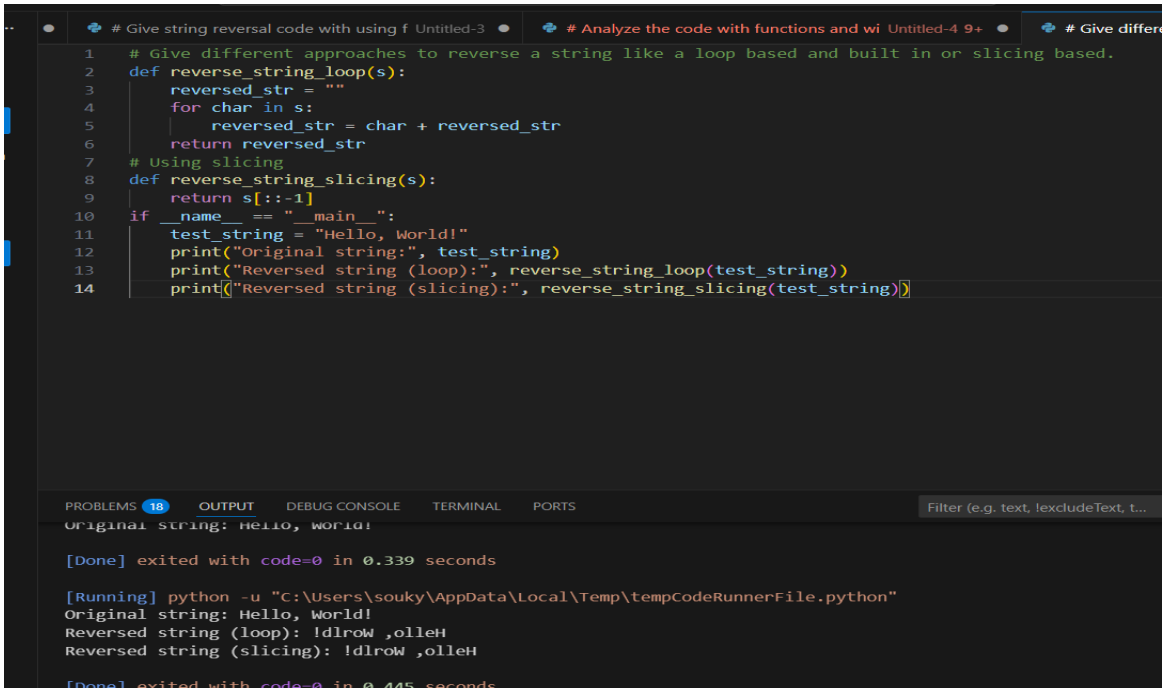
➢ A built-in / slicing-based string reversal approach

❖ Expected Output

➢ Two correct implementations

➢ Comparison discussing:

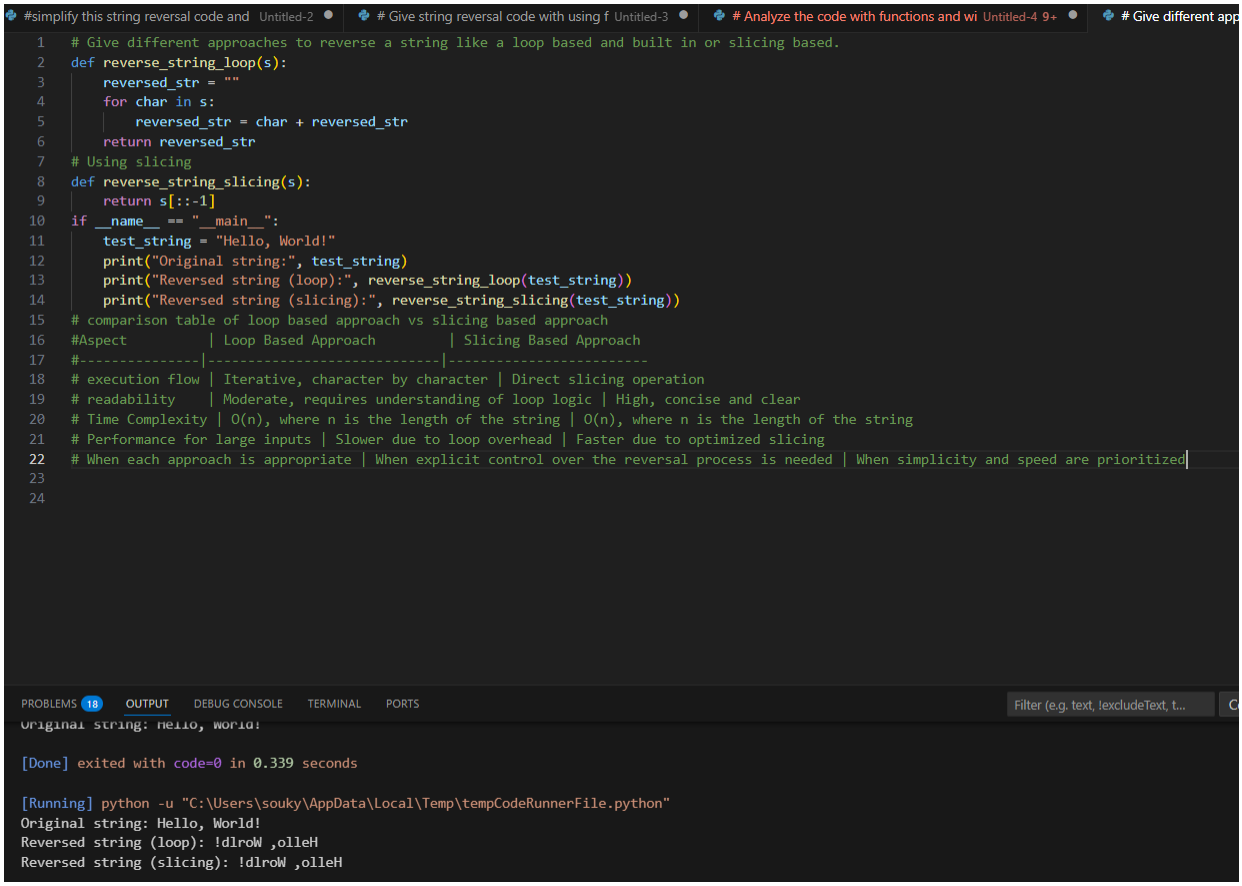
- Execution flow
- Time complexity
- Performance for large inputs
- When each approach is appropriate



```python
# Give different approaches to reverse a string like a loop based and built in or slicing based.
def reverse_string_loop(s):
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str
# Using slicing
def reverse_string_slicing(s):
    return s[::-1]
if __name__ == "__main__":
    test_string = "Hello, World!"
    print("Original string:", test_string)
    print("Reversed string (loop):", reverse_string_loop(test_string))
    print("Reversed string (slicing):", reverse_string_slicing(test_string))
```

```
[Done] exited with code=0 in 0.339 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Original string: Hello, World!
Reversed string (loop): !dlroW ,olleH
Reversed string (slicing): !dlroW ,olleH
```



```python
# Give different approaches to reverse a string like a loop based and built in or slicing based.
def reverse_string_loop(s):
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str
# Using slicing
def reverse_string_slicing(s):
    return s[::-1]
if __name__ == "__main__":
    test_string = "Hello, World!"
    print("Original string:", test_string)
    print("Reversed string (loop):", reverse_string_loop(test_string))
    print("Reversed string (slicing):", reverse_string_slicing(test_string))
# comparison table of loop based approach vs slicing based approach
#Aspect          | Loop Based Approach         | Slicing Based Approach
#----------------|-----------------------------|-----------------------
# execution flow | Iterative, character by character | Direct slicing operation
# readability    | Moderate, requires understanding of loop logic | High, concise and clear
# Time Complexity | O(n), where n is the length of the string | O(n), where n is the length of the string
# Performance for large inputs | Slower due to loop overhead | Faster due to optimized slicing
# When each approach is appropriate | When explicit control over the reversal process is needed | When simplicity and speed are prioritized
```

```
[Done] exited with code=0 in 0.339 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Original string: Hello, World!
Reversed string (loop): !dlroW ,olleH
Reversed string (slicing): !dlroW ,olleH
```