SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Dr. Rishabh Mittal
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju

CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week1 - Friday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches

Assignment Number:1.3(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	Lab 1: Environment Setup – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow* **Lab Objectives:** ❖ To install and configure GitHub Copilot in Visual Studio Code.	Week1 - Monday

- To explore AI-assisted code generation using GitHub Copilot.
- To analyze the accuracy and effectiveness of Copilot's code suggestions.
- To understand prompt-based programming using comments and code context

Lab Outcomes (LOs):
After completing this lab, students will be able to:

- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

Task 0
- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Expected Output
- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

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)

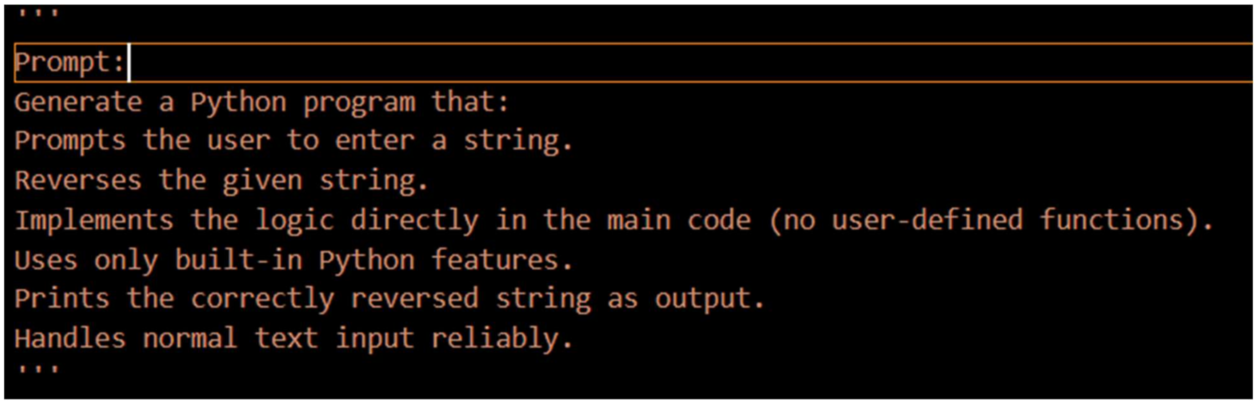
- **Scenario**
  You are asked to justify design choices during a code review.

- **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	
	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 **Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.**	

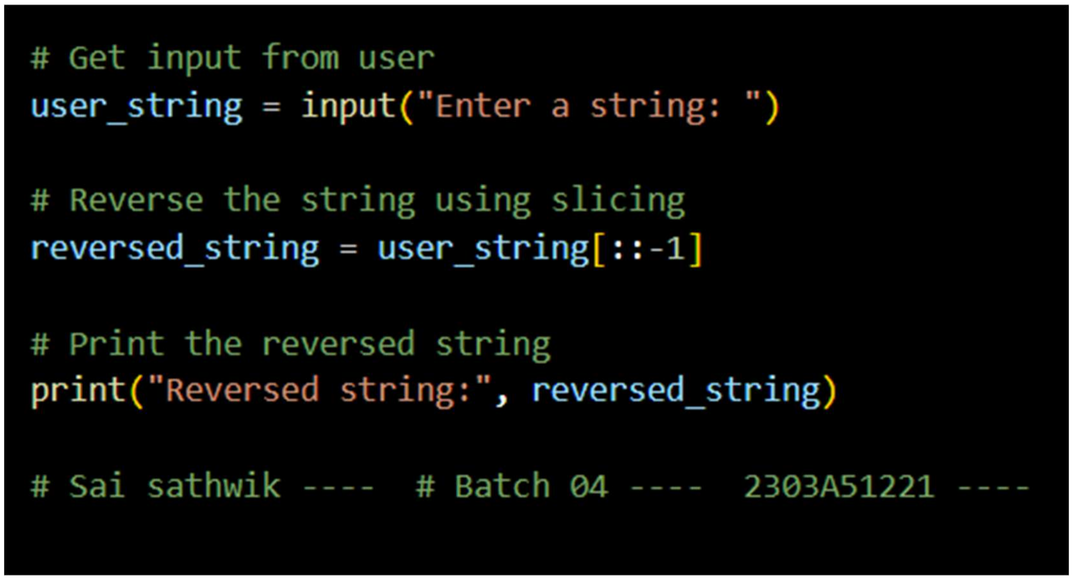
TASK-1

PROMPT:



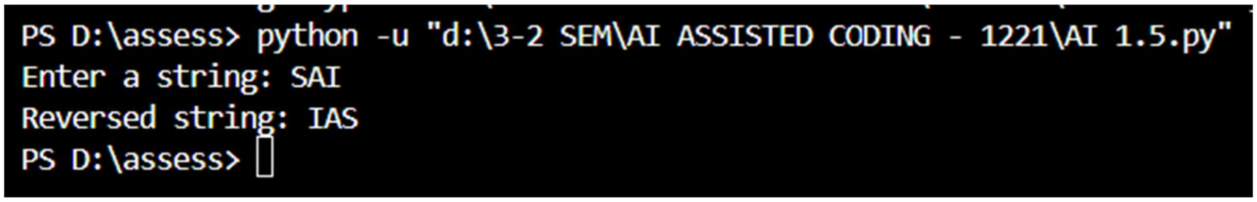
```
'''
Prompt:
Generate a Python program that:
Prompts the user to enter a string.
Reverses the given string.
Implements the logic directly in the main code (no user-defined functions).
Uses only built-in Python features.
Prints the correctly reversed string as output.
Handles normal text input reliably.
'''
```

CODE:



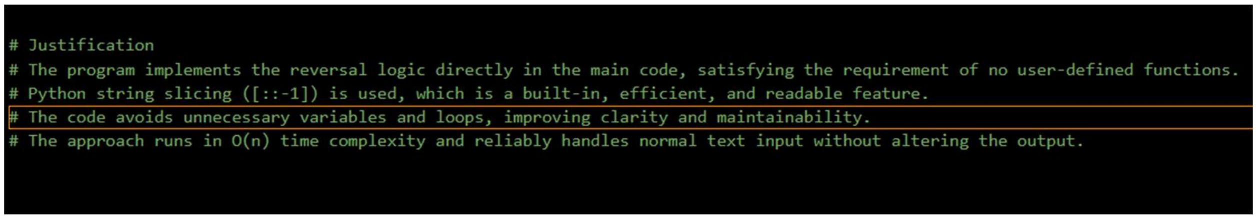
```python
# Get input from user
user_string = input("Enter a string: ")

# Reverse the string using slicing
reversed_string = user_string[::-1]

# Print the reversed string
print("Reversed string:", reversed_string)

# Sai sathwik ----  # Batch 04 ----  2303A51221 ----
```

OUTPUT:



```
PS D:\assess> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\AI 1.5.py"
Enter a string: SAI
Reversed string: IAS
PS D:\assess> 
```

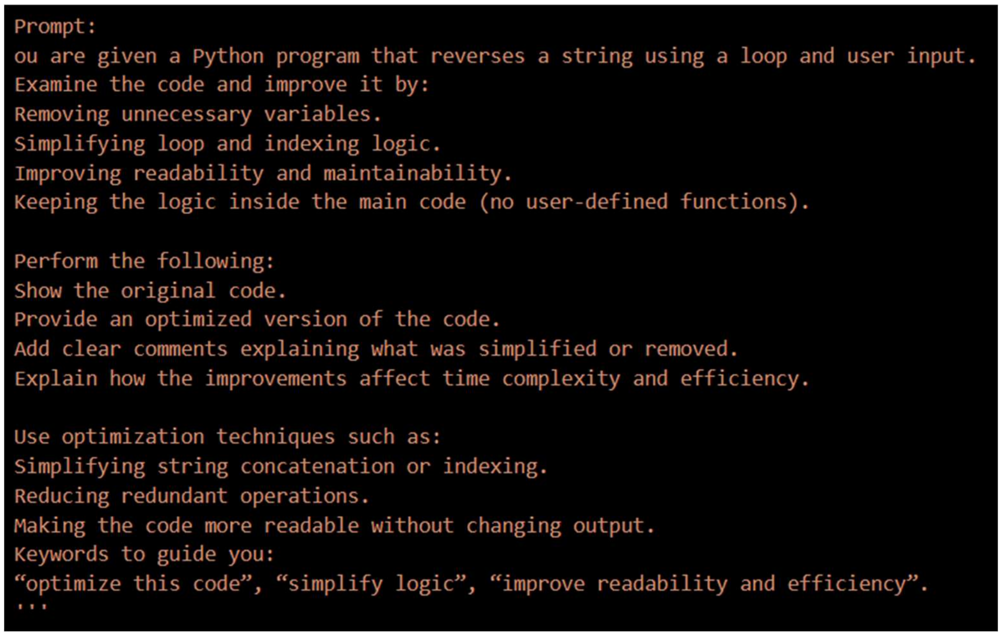
JUSTIFICATION:



```
# Justification
# The program implements the reversal logic directly in the main code, satisfying the requirement of no user-defined functions.
# Python string slicing ([::-1]) is used, which is a built-in, efficient, and readable feature.
# The code avoids unnecessary variables and loops, improving clarity and maintainability.
# The approach runs in O(n) time complexity and reliably handles normal text input without altering the output.
```

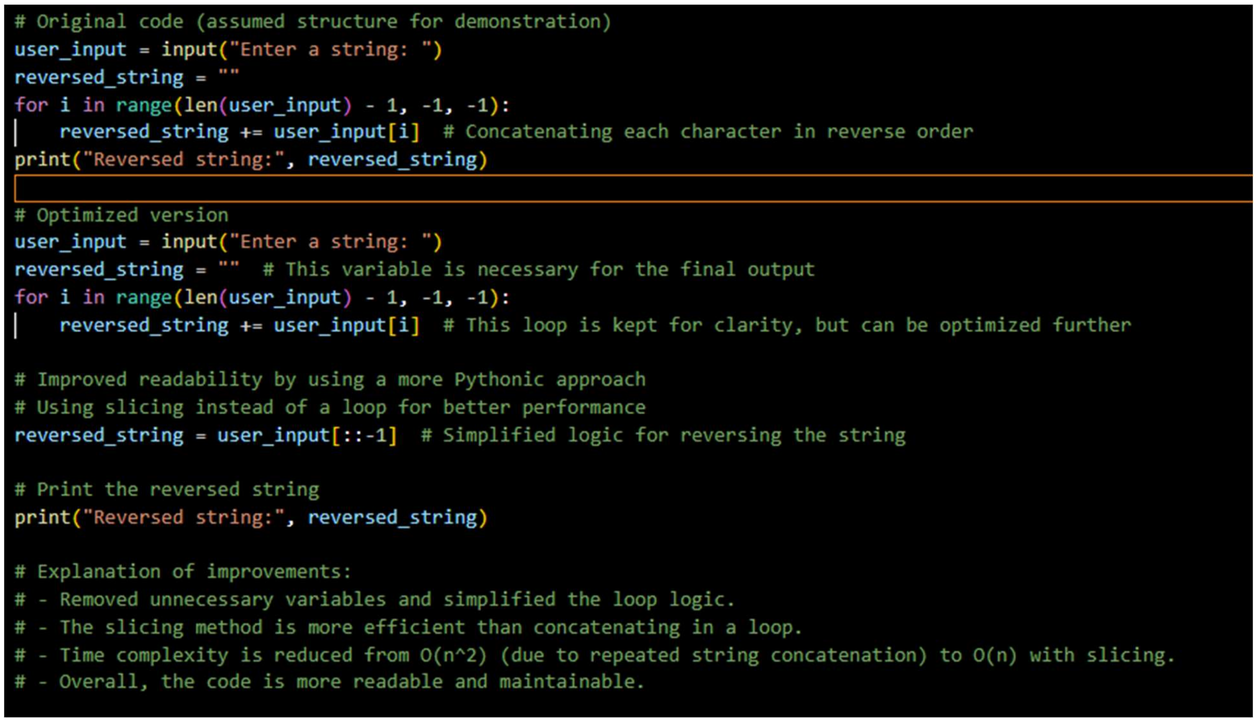
TASK2:

PROMPT



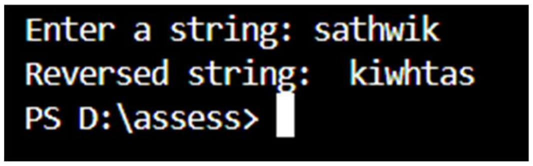
```
Prompt:
ou are given a Python program that reverses a string using a loop and user input.
Examine the code and improve it by:
Removing unnecessary variables.
Simplifying loop and indexing logic.
Improving readability and maintainability.
Keeping the logic inside the main code (no user-defined functions).

Perform the following:
Show the original code.
Provide an optimized version of the code.
Add clear comments explaining what was simplified or removed.
Explain how the improvements affect time complexity and efficiency.

Use optimization techniques such as:
Simplifying string concatenation or indexing.
Reducing redundant operations.
Making the code more readable without changing output.
Keywords to guide you:
"optimize this code", "simplify logic", "improve readability and efficiency".
'''
```

CODE:



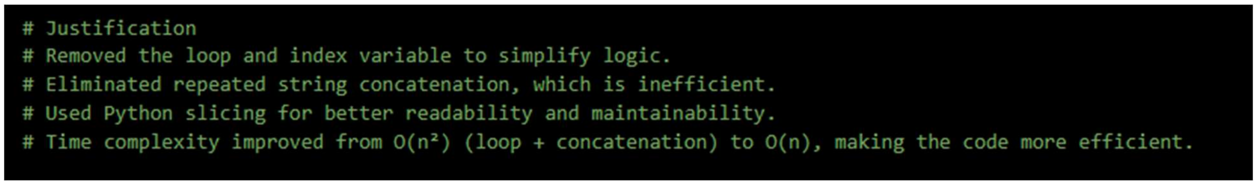
```python
# Original code (assumed structure for demonstration)
user_input = input("Enter a string: ")
reversed_string = ""
for i in range(len(user_input) - 1, -1, -1):
    reversed_string += user_input[i]  # Concatenating each character in reverse order
print("Reversed string:", reversed_string)

# Optimized version
user_input = input("Enter a string: ")
reversed_string = ""  # This variable is necessary for the final output
for i in range(len(user_input) - 1, -1, -1):
    reversed_string += user_input[i]  # This loop is kept for clarity, but can be optimized further

# Improved readability by using a more Pythonic approach
# Using slicing instead of a loop for better performance
reversed_string = user_input[::-1]  # Simplified logic for reversing the string

# Print the reversed string
print("Reversed string:", reversed_string)

# Explanation of improvements:
# - Removed unnecessary variables and simplified the loop logic.
# - The slicing method is more efficient than concatenating in a loop.
# - Time complexity is reduced from O(n^2) (due to repeated string concatenation) to O(n) with slicing.
# - Overall, the code is more readable and maintainable.
```

OUTPUT:



```
Enter a string: sathwik
Reversed string:  kiwhtas
PS D:\assess> 
```

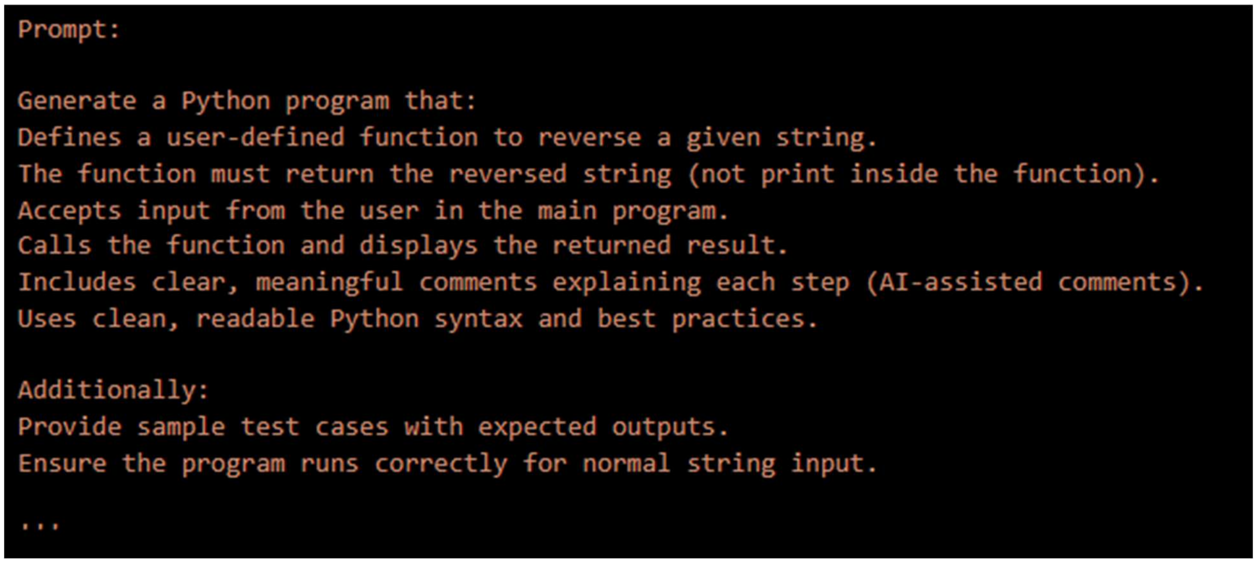
JUSTIFICATION:



```
# Justification
# Removed the loop and index variable to simplify logic.
# Eliminated repeated string concatenation, which is inefficient.
# Used Python slicing for better readability and maintainability.
# Time complexity improved from O(n²) (loop + concatenation) to O(n), making the code more efficient.
```

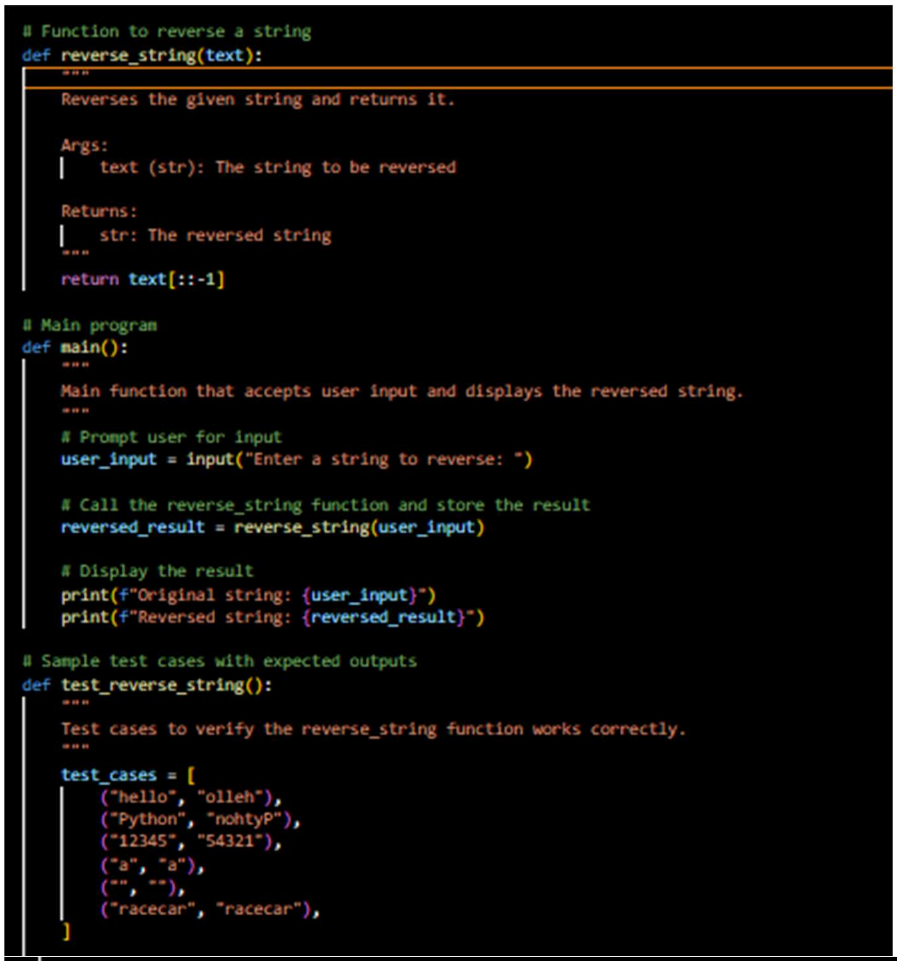
TASK3:

PROMPT:

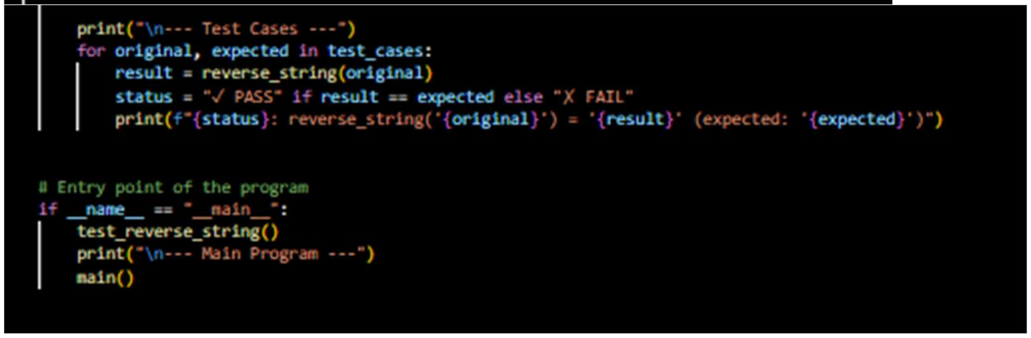


```
Prompt:

Generate a Python program that:
Defines a user-defined function to reverse a given string.
The function must return the reversed string (not print inside the function).
Accepts input from the user in the main program.
Calls the function and displays the returned result.
Includes clear, meaningful comments explaining each step (AI-assisted comments).
Uses clean, readable Python syntax and best practices.

Additionally:
Provide sample test cases with expected outputs.
Ensure the program runs correctly for normal string input.

...
```

CODE:

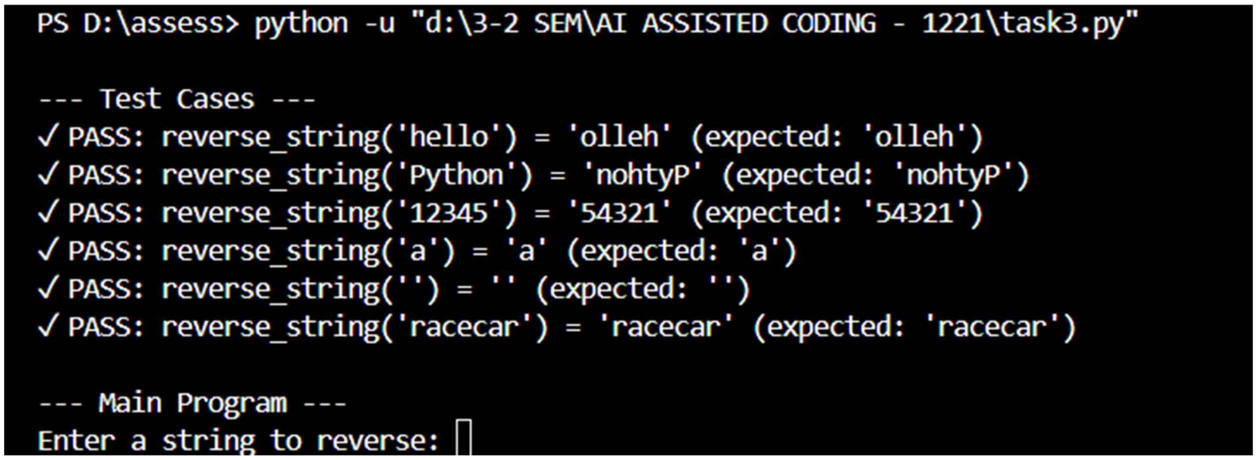


```python
# Function to reverse a string
def reverse_string(text):
    """
    Reverses the given string and returns it.

    Args:
        text (str): The string to be reversed

    Returns:
        str: The reversed string
    """
    return text[::-1]

# Main program
def main():
    """
    Main function that accepts user input and displays the reversed string.
    """
    # Prompt user for input
    user_input = input("Enter a string to reverse: ")

    # Call the reverse_string function and store the result
    reversed_result = reverse_string(user_input)

    # Display the result
    print(f"Original string: {user_input}")
    print(f"Reversed string: {reversed_result}")

# Sample test cases with expected outputs
def test_reverse_string():
    """
    Test cases to verify the reverse_string function works correctly.
    """
    test_cases = [
        ("hello", "olleh"),
        ("Python", "nohtyP"),
        ("12345", "54321"),
        ("a", "a"),
        ("", ""),
        ("racecar", "racecar"),
    ]
```



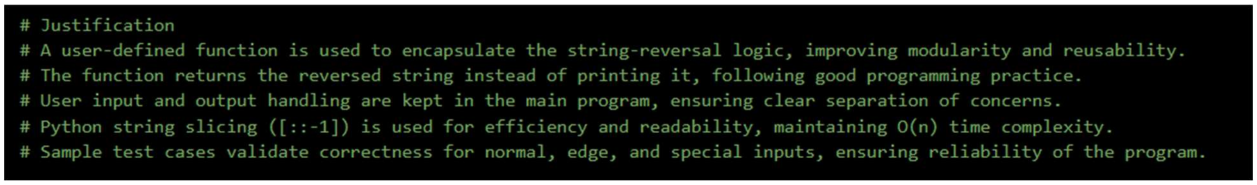
```python
    print("\n--- Test Cases ---")
    for original, expected in test_cases:
        result = reverse_string(original)
        status = "✓ PASS" if result == expected else "✗ FAIL"
        print(f"{status}: reverse_string('{original}') = '{result}' (expected: '{expected}')")


# Entry point of the program
if __name__ == "__main__":
    test_reverse_string()
    print("\n--- Main Program ---")
    main()
```

OUTPUT:



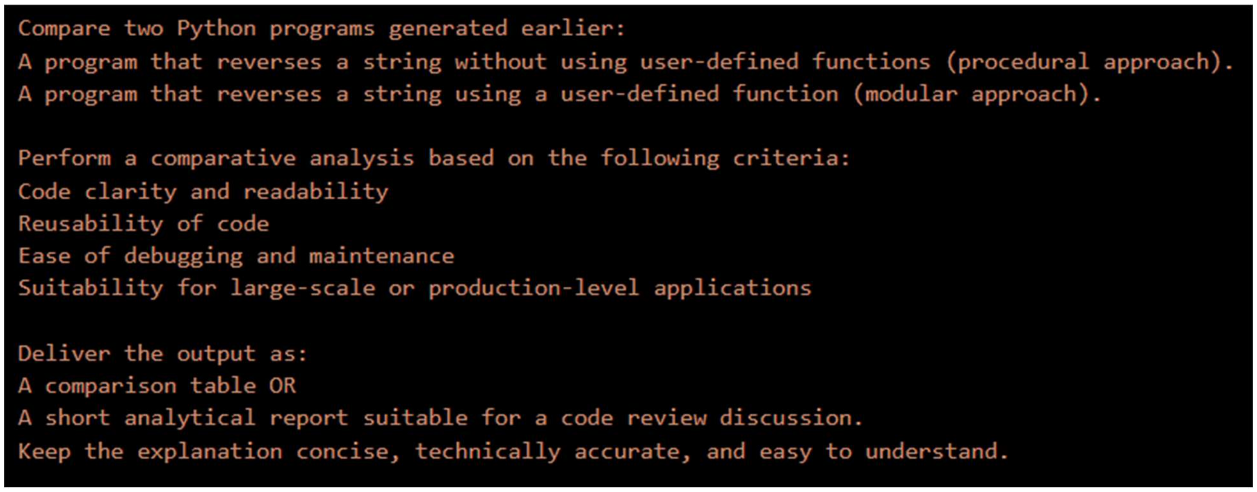
```
PS D:\assess> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\task3.py"

--- Test Cases ---
✓ PASS: reverse_string('hello') = 'olleh' (expected: 'olleh')
✓ PASS: reverse_string('Python') = 'nohtyP' (expected: 'nohtyP')
✓ PASS: reverse_string('12345') = '54321' (expected: '54321')
✓ PASS: reverse_string('a') = 'a' (expected: 'a')
✓ PASS: reverse_string('') = '' (expected: '')
✓ PASS: reverse_string('racecar') = 'racecar' (expected: 'racecar')

--- Main Program ---
Enter a string to reverse:
```

JUSTIFICATION:



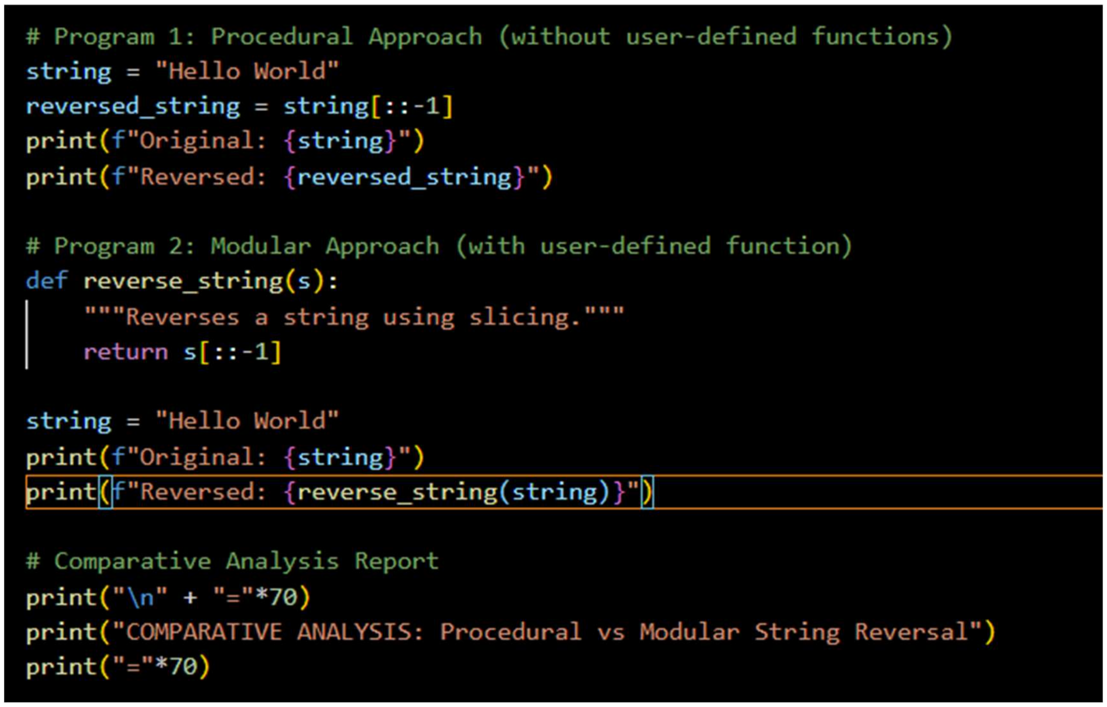
```
# Justification
# A user-defined function is used to encapsulate the string-reversal logic, improving modularity and reusability.
# The function returns the reversed string instead of printing it, following good programming practice.
# User input and output handling are kept in the main program, ensuring clear separation of concerns.
# Python string slicing ([::-1]) is used for efficiency and readability, maintaining O(n) time complexity.
# Sample test cases validate correctness for normal, edge, and special inputs, ensuring reliability of the program.
```

TASK4:
PROMPT:

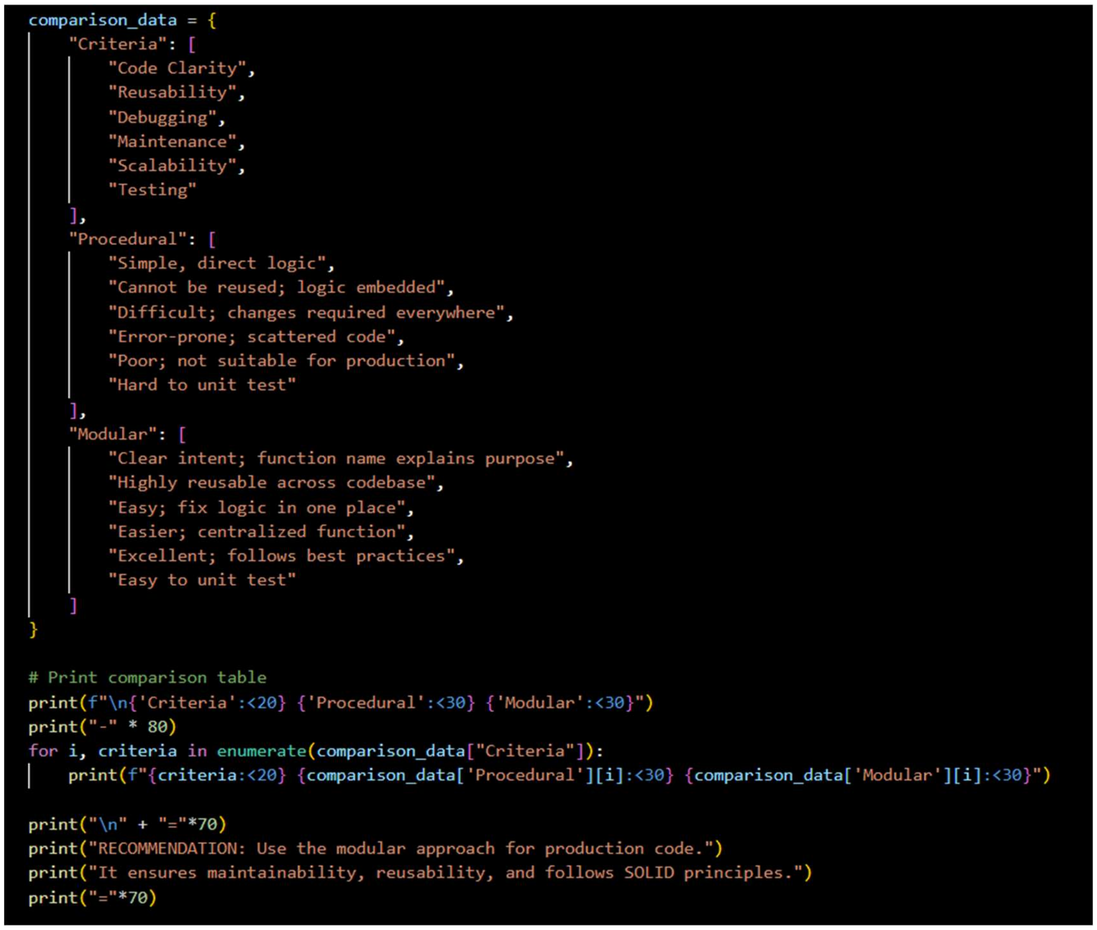


```
Compare two Python programs generated earlier:
A program that reverses a string without using user-defined functions (procedural approach).
A program that reverses a string using a user-defined function (modular approach).

Perform a comparative analysis based on the following criteria:
Code clarity and readability
Reusability of code
Ease of debugging and maintenance
Suitability for large-scale or production-level applications

Deliver the output as:
A comparison table OR
A short analytical report suitable for a code review discussion.
Keep the explanation concise, technically accurate, and easy to understand.
```

CODE:

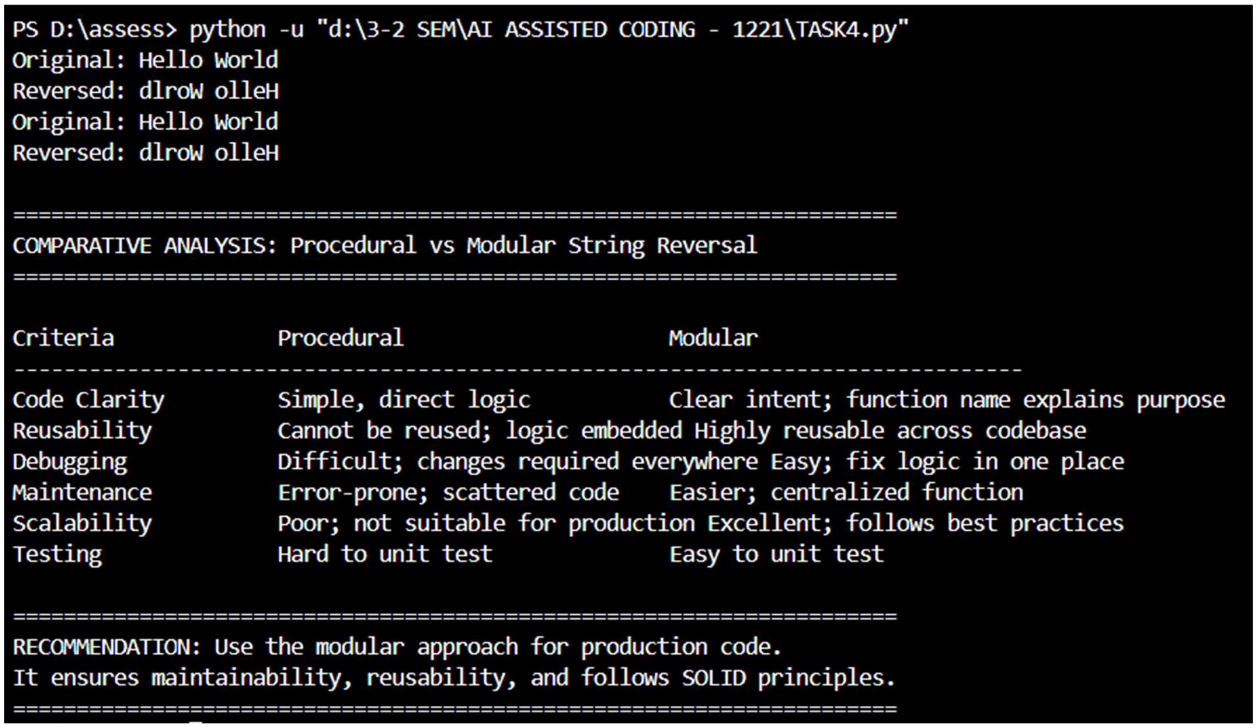


```python
# Program 1: Procedural Approach (without user-defined functions)
string = "Hello World"
reversed_string = string[::-1]
print(f"Original: {string}")
print(f"Reversed: {reversed_string}")

# Program 2: Modular Approach (with user-defined function)
def reverse_string(s):
    """Reverses a string using slicing."""
    return s[::-1]

string = "Hello World"
print(f"Original: {string}")
print(f"Reversed: {reverse_string(string)}")

# Comparative Analysis Report
print("\n" + "="*70)
print("COMPARATIVE ANALYSIS: Procedural vs Modular String Reversal")
print("="*70)
```



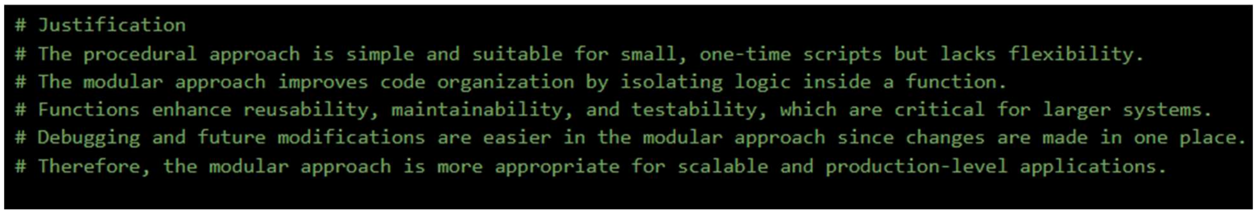
```python
comparison_data = {
    "Criteria": [
        "Code Clarity",
        "Reusability",
        "Debugging",
        "Maintenance",
        "Scalability",
        "Testing"
    ],
    "Procedural": [
        "Simple, direct logic",
        "Cannot be reused; logic embedded",
        "Difficult; changes required everywhere",
        "Error-prone; scattered code",
        "Poor; not suitable for production",
        "Hard to unit test"
    ],
    "Modular": [
        "Clear intent; function name explains purpose",
        "Highly reusable across codebase",
        "Easy; fix logic in one place",
        "Easier; centralized function",
        "Excellent; follows best practices",
        "Easy to unit test"
    ]
}

# Print comparison table
print(f"\n{'Criteria':<20} {'Procedural':<30} {'Modular':<30}")
print("-" * 80)
for i, criteria in enumerate(comparison_data["Criteria"]):
    print(f"{criteria:<20} {comparison_data['Procedural'][i]:<30} {comparison_data['Modular'][i]:<30}")

print("\n" + "="*70)
print("RECOMMENDATION: Use the modular approach for production code.")
print("It ensures maintainability, reusability, and follows SOLID principles.")
print("="*70)
```

OUTPUT:



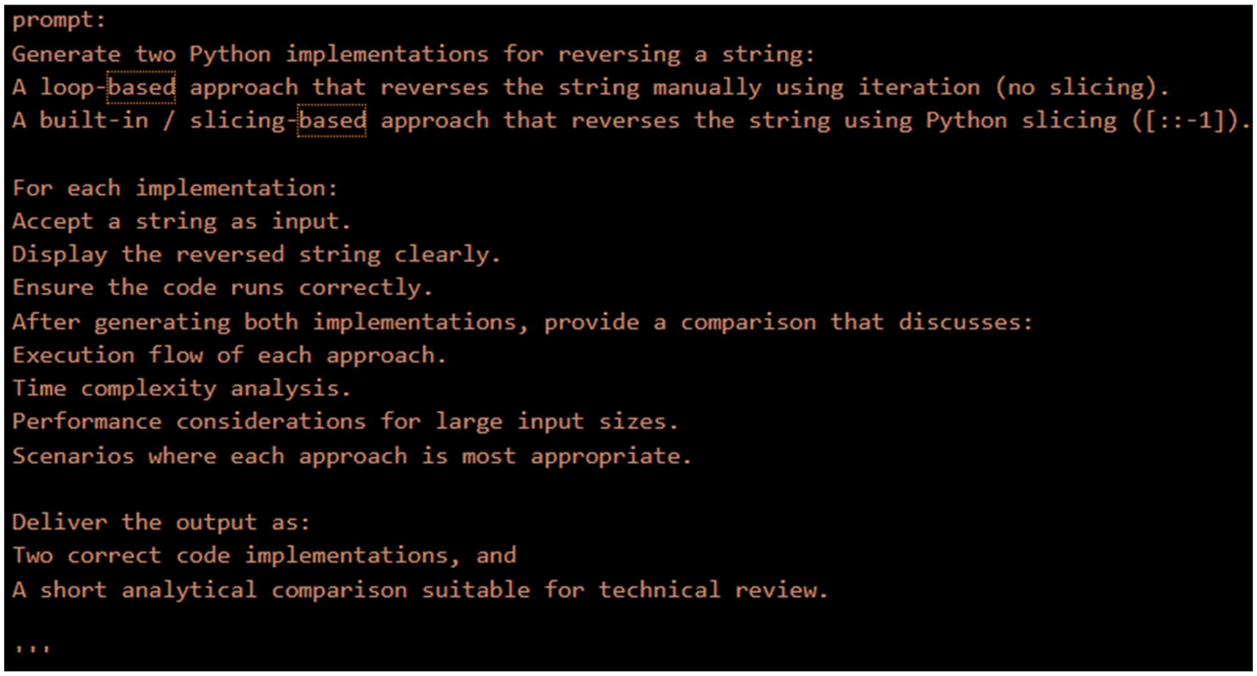
```
PS D:\assess> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\TASK4.py"
Original: Hello World
Reversed: dlroW olleH
Original: Hello World
Reversed: dlroW olleH

======================================================================
COMPARATIVE ANALYSIS: Procedural vs Modular String Reversal
======================================================================

Criteria             Procedural                     Modular
--------------------------------------------------------------------------------
Code Clarity         Simple, direct logic           Clear intent; function name explains purpose
Reusability          Cannot be reused; logic embedded Highly reusable across codebase
Debugging            Difficult; changes required everywhere Easy; fix logic in one place
Maintenance          Error-prone; scattered code    Easier; centralized function
Scalability          Poor; not suitable for production Excellent; follows best practices
Testing              Hard to unit test              Easy to unit test

======================================================================
RECOMMENDATION: Use the modular approach for production code.
It ensures maintainability, reusability, and follows SOLID principles.
======================================================================
```

JUSTIFICATION:



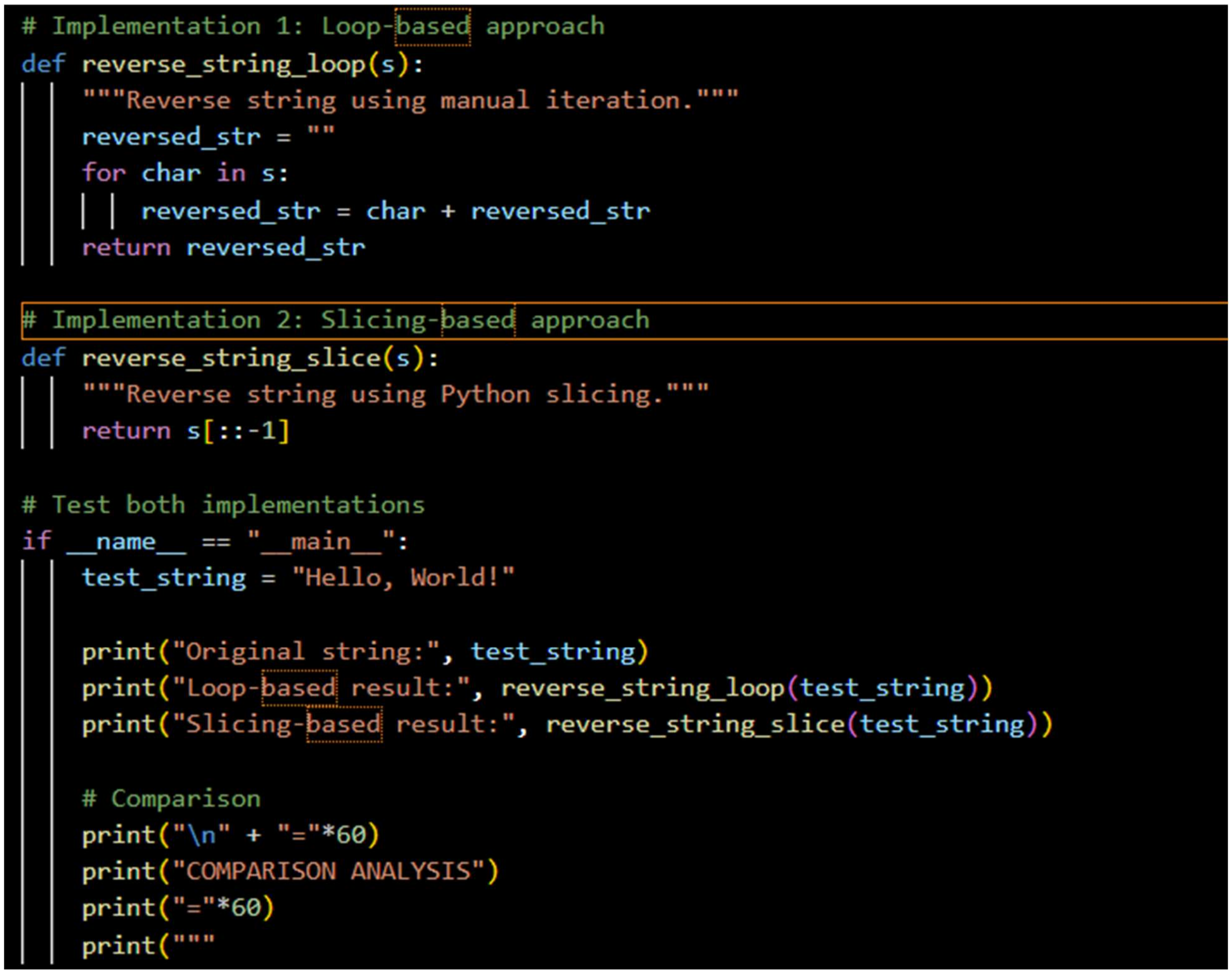
```
# Justification
# The procedural approach is simple and suitable for small, one-time scripts but lacks flexibility.
# The modular approach improves code organization by isolating logic inside a function.
# Functions enhance reusability, maintainability, and testability, which are critical for larger systems.
# Debugging and future modifications are easier in the modular approach since changes are made in one place.
# Therefore, the modular approach is more appropriate for scalable and production-level applications.
```

TASK5:
PROMPT:



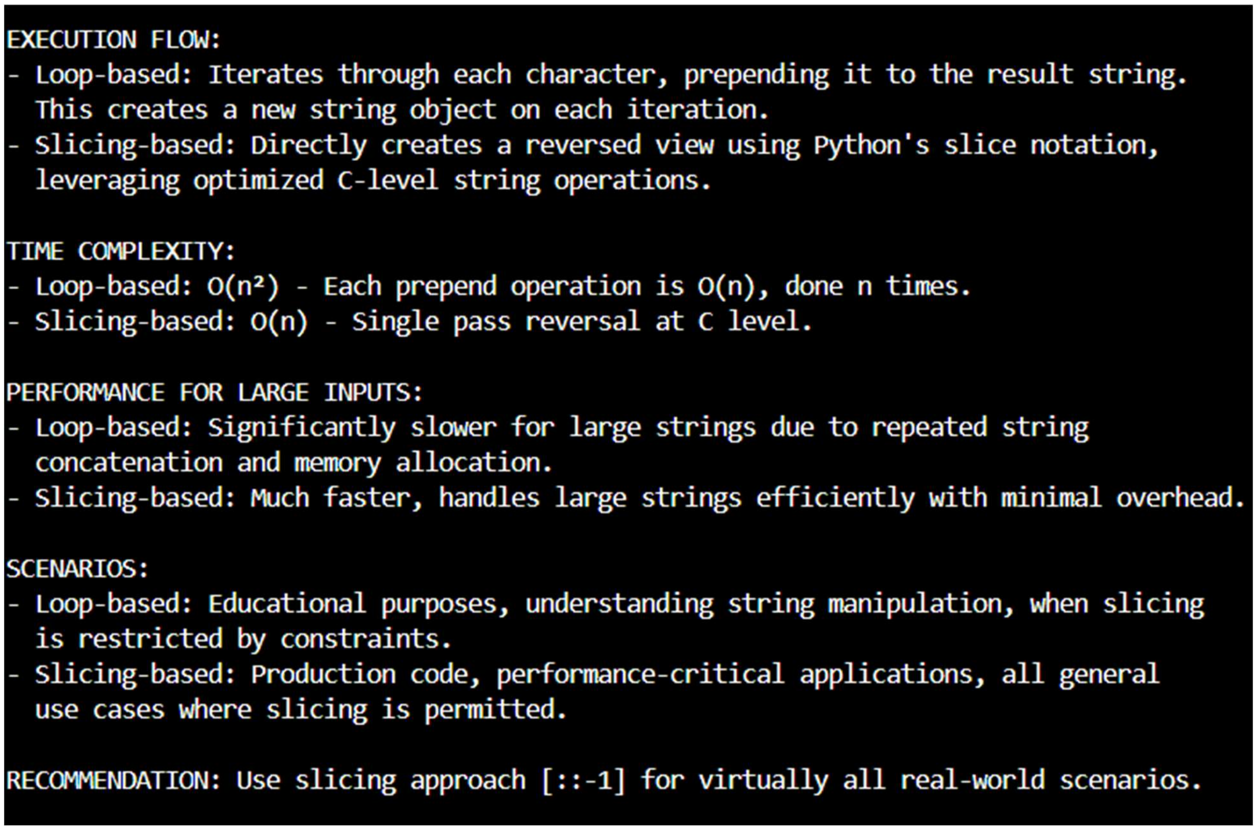
```
prompt:
Generate two Python implementations for reversing a string:
A loop-based approach that reverses the string manually using iteration (no slicing).
A built-in / slicing-based approach that reverses the string using Python slicing ([::-1]).

For each implementation:
Accept a string as input.
Display the reversed string clearly.
Ensure the code runs correctly.
After generating both implementations, provide a comparison that discusses:
Execution flow of each approach.
Time complexity analysis.
Performance considerations for large input sizes.
Scenarios where each approach is most appropriate.

Deliver the output as:
Two correct code implementations, and
A short analytical comparison suitable for technical review.

'''
```

CODE:



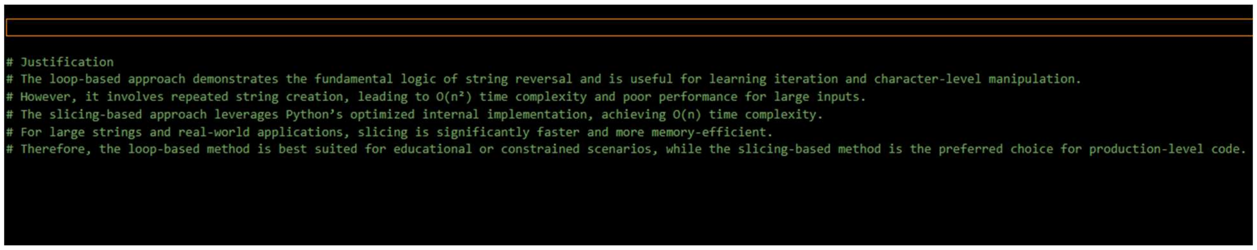
```
# Implementation 1: Loop-based approach
def reverse_string_loop(s):
    """Reverse string using manual iteration."""
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str

# Implementation 2: Slicing-based approach
def reverse_string_slice(s):
    """Reverse string using Python slicing."""
    return s[::-1]

# Test both implementations
if __name__ == "__main__":
    test_string = "Hello, World!"

    print("Original string:", test_string)
    print("Loop-based result:", reverse_string_loop(test_string))
    print("Slicing-based result:", reverse_string_slice(test_string))

    # Comparison
    print("\n" + "="*60)
    print("COMPARISON ANALYSIS")
    print("="*60)
    print("""
```

OUTPUT:



```
EXECUTION FLOW:
- Loop-based: Iterates through each character, prepending it to the result string.
  This creates a new string object on each iteration.
- Slicing-based: Directly creates a reversed view using Python's slice notation,
  leveraging optimized C-level string operations.

TIME COMPLEXITY:
- Loop-based: O(n²) - Each prepend operation is O(n), done n times.
- Slicing-based: O(n) - Single pass reversal at C level.

PERFORMANCE FOR LARGE INPUTS:
- Loop-based: Significantly slower for large strings due to repeated string
  concatenation and memory allocation.
- Slicing-based: Much faster, handles large strings efficiently with minimal overhead.

SCENARIOS:
- Loop-based: Educational purposes, understanding string manipulation, when slicing
  is restricted by constraints.
- Slicing-based: Production code, performance-critical applications, all general
  use cases where slicing is permitted.

RECOMMENDATION: Use slicing approach [::-1] for virtually all real-world scenarios.
```

JUSTIFICATION:



```
# Justification
# The loop-based approach demonstrates the fundamental logic of string reversal and is useful for learning iteration and character-level manipulation.
# However, it involves repeated string creation, leading to O(n²) time complexity and poor performance for large inputs.
# The slicing-based approach leverages Python’s optimized internal implementation, achieving O(n) time complexity.
# For large strings and real-world applications, slicing is significantly faster and more memory-efficient.
# Therefore, the loop-based method is best suited for educational or constrained scenarios, while the slicing-based method is the preferred choice for production-level code.
```