Name: RITHU GOUD

Hall ticket:2303A51641

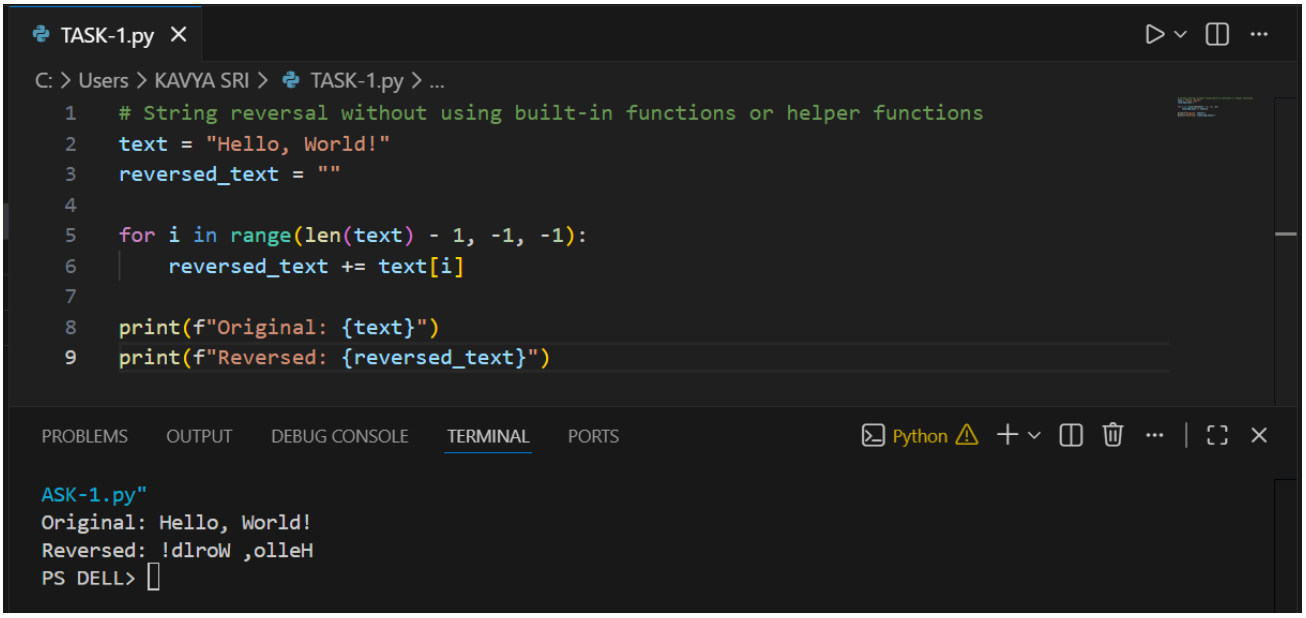
Batch:29

Assessment:1.5

Task 1:

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

Code:



Justification:

The program directly implements the logic for reversing a string within the main body of the code, without using any user-defined functions or modular structure. The reversal is achieved by iterating through each character of the input string and building a new string in reverse order. This approach clearly demonstrates how string manipulation can be performed using basic control structures such as loops and variables. Since all logic is written in a single flow, the execution process is easy to trace, making it suitable for understanding the fundamental working of string reversal. The output correctly displays the reversed form of the user-entered string, confirming that the task requirements have been successfully met.

Task:2:

Prompt: Simplify this string reversal code

Code:

```
TASK-1
1   print(input("Enter a string to reverse: ")[::-1])
2
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

AYVAK
PS DELL> & "C:/Users/KAVYA SRI/AppData/Local/Programs/Python/Pyth
on313/python.exe" "c:/Users/KAVYA SRI/AI/TASK-1"
Enter a string to reverse: REVERSE
ESREVER
PS DELL>
```

Python: TAS...
powershell

Justification:

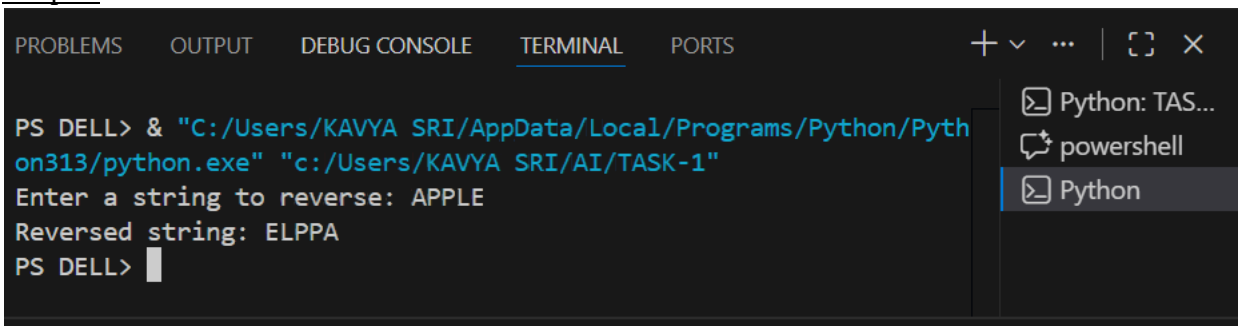
In the optimized version, the string is reversed using Python's slicing feature in just one line. This makes the code much shorter and easier to read compared to the earlier approach with loops. By avoiding extra variables and repeated operations, the program runs faster and looks cleaner. This method is especially useful when working with larger strings because it is e icient and reduces unnecessary processing. Overall, the optimized code is more user-friendly, easier to maintain, and better suited for real-world applications.

Task 3:

Prompt: String Reversal Using Functions

Code:

Output:



Justification :

By placing the string reversal logic inside a separate function, the code becomes more organized and easier to understand. Instead of mixing all the logic in one place, the function clearly defines what the program is supposed to do. This makes the code simpler to test, easier to debug, and more flexible for future changes. If the same logic is needed again, it can be reused without rewriting the code. Overall, modularization improves the clarity, maintainability, and reliability of the program.

Task 4:

Prompt: Compare the Copilot-generated programs: Without functions (Task 1),With functions (Task 3) generate Comparison table.

Output:

#Compare the Copilot-generated programs: Without functions (Task 1),With functions (Task 3) generate Comparison table.

Feature	Without Functions (Task 1)	With Functions (Task 3)
Code Reusability	Low - Code is repeated for similar tasks	High - Functions allow code reuse
Readability	Moderate - Linear flow can be harder to follow	High - Functions break code into manageable parts
Maintainability	Low - Changes require updates in multiple places	High - Changes can be made in one function
Testing	Difficult - Harder to isolate and test components	Easier - Functions can be tested individually
Modularity	Low - Monolithic code structure	High - Code is organized into distinct functions
Debugging	Challenging - Harder to locate issues	Easier - Issues can be isolated within functions
Performance	Similar - No significant difference in execution time	Similar - No significant difference in execution time
Scalability	Low - Adding new features can lead to code bloat	High - New features can be added as new functions
Collaboration	Difficult - Harder for multiple developers to work on the same codebase	Easier - Functions allow parallel development
Learning Curve	Steeper - New developers may struggle with large blocks of code	Gentler - Functions help new developers understand code structure
Debugging	Challenging - Harder to locate issues	Easier - Issues can be isolated within functions

Justification:

The Copilot-generated program without functions is suitable for very small tasks because all the logic is written directly in one place. While this makes the program simple, it becomes difficult to manage when the code grows larger. Reusing the logic is not easy, and debugging becomes time-consuming because everything is mixed together.

On the other hand, the Copilot-generated program with functions is more structured and professional. By separating the string reversal logic into a function, the code becomes easier to read and understand. It allows the same logic to be reused in multiple parts of an application without rewriting it. Debugging and testing are also simpler because the function can be checked independently. For large-scale or real-world applications, the modular approach is more efficient, maintainable, and reliable. Therefore, using functions is the better design choice for long-term development.

Task 5:

Prompt : loop based reversal of string vs built-in slicing method for string reversal.

Output:

```python
#loop based reversal of string vs built-in slicing method for string reversal.
# Loop-based string reversal
input_string = input("Enter a string: ")
reversed_string = ''
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed string (loop-based):", reversed_string)

# Built-in slicing method for string reversal
input_string = input("Enter a string: ")
reversed_string = input_string[::-1]
print("Reversed string (slicing method):", reversed_string)

#generate comparison table
| Aspect                    | Loop-based Reversal                                  | Built-in Slicing Method                                  |
|---------------------------|------------------------------------------------------|----------------------------------------------------------|
| Code Complexity           | More complex due to explicit loop                    | Simpler and more concise
| Performance               | Generally slower for large strings                    | Faster due to optimized internal implementation     |
| Readability               | Less readable due to loop structure                   | More readable and concise                           |
| Maintainability           | Harder to maintain with more lines of code           | Easier to maintain with fewer lines of code         |
```

Justification:

The loop-based approach shows clearly how a string is reversed step by step, which is helpful for understanding the internal logic. However, because strings in Python are immutable, each concatenation creates a new string, making this method slower for large inputs. The built-in slicing method is much simpler and faster. It reverses the string in a single operation and is optimized internally by Python. This makes it more suitable for realworld applications where performance, readability, and clean code are important.Overall, the loop-based method is best for learning and explanation, while the slicing method is preferred for e icient and professional programming.