Name: Achi Vaishnavi

Hall ticket:2303A51554

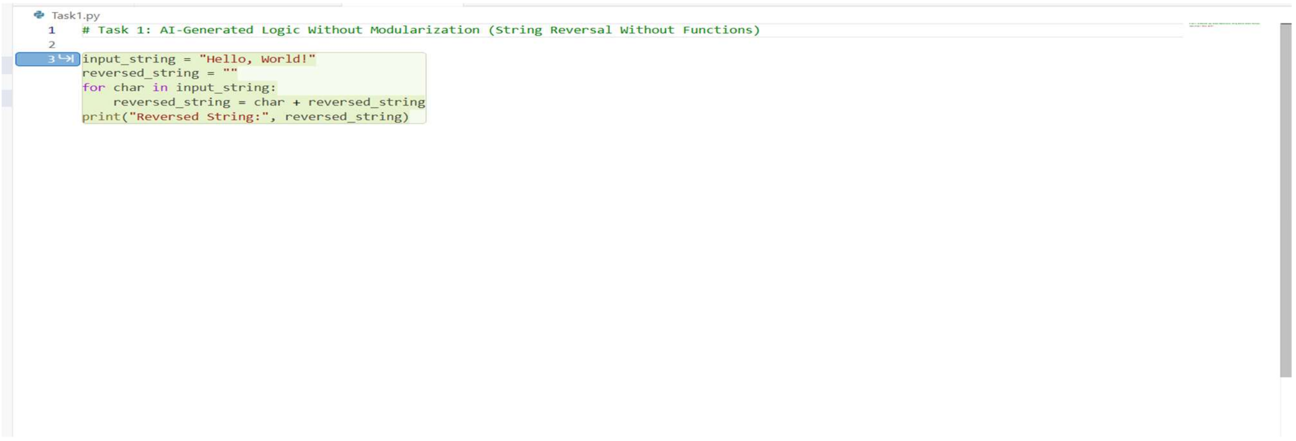
Batch:29

Assessment: 1.5

Task 1:

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

Code:

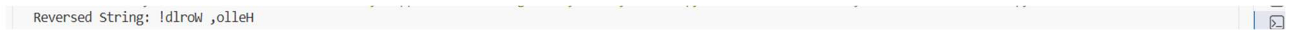


Task1.py

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

input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String:", reversed_string)
```

Output:



```
Reversed String: !dlroW ,olleH
```

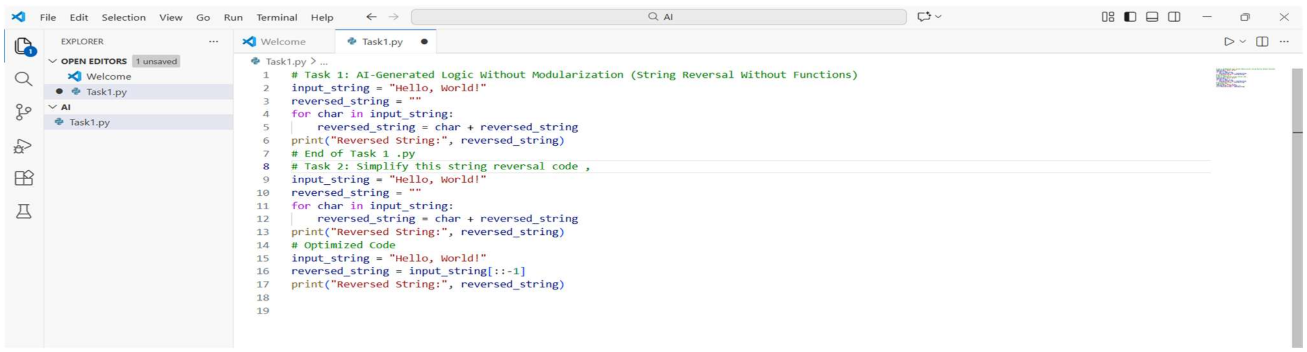
Justification:

The program performs string reversal directly within the main section of the code, without applying any user-defined functions or modular design. It reverses the string by looping through each character of the input and constructing a new string in the opposite order. This method effectively illustrates how basic programming constructs such as loops and variables can be used for string operations. Because the entire logic is written sequentially, the flow of execution is simple to follow, which makes it helpful for understanding the core concept of string reversal. The final output successfully displays the reversed version of the user-provided string, confirming that the program meets the intended requirements.

Task:2:

Prompt: Simplify this string reversal code

Code:



```python
# Task 1: AI-Generated Logic Without Modularization (String Reversal Without Functions)
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String:", reversed_string)
# End of Task 1 .py
# Task 2: Simplify this string reversal code ,
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String:", reversed_string)
# Optimized Code
input_string = "Hello, World!"
reversed_string = input_string[::-1]
print("Reversed String:", reversed_string)
```

Output:

```
Reversed String: !dlroW ,olleH
Reversed String befor simpilification : !dlroW ,olleH
Reversed String after simpilification : !dlroW ,olleH
```

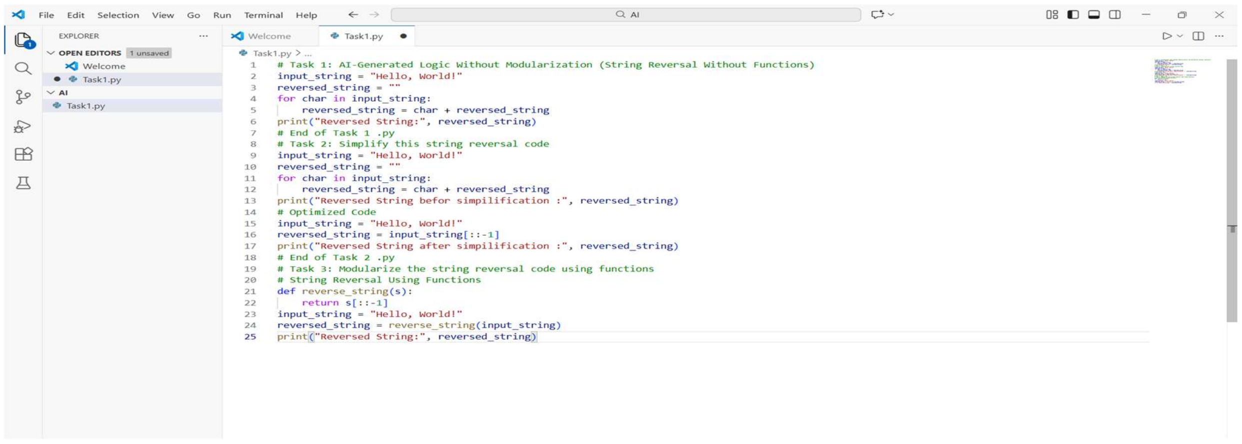
Justification:

In the improved version, Python's slicing technique is used to reverse the string in a single line of code. This significantly reduces the length of the program and improves readability when compared to the loop-based method. Since it eliminates the need for additional variables and repeated steps, the execution becomes more efficient and the code appears more organized. This approach is particularly effective for handling longer strings, as it minimizes extra processing. Overall, the optimized solution is simpler to understand, easier to maintain, and more practical for real-world use.

Task 3:

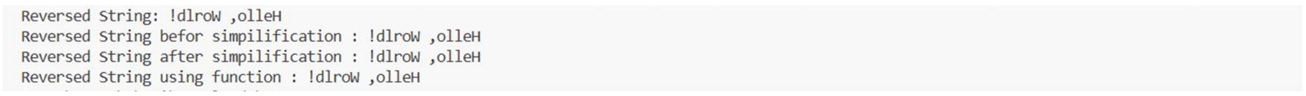
Prompt: String Reversal Using Functions

Code:



```python
# Task 1: AI-Generated Logic Without Modularization (String Reversal Without Functions)
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String:", reversed_string)
# End of Task 1 .py
# Task 2: Simplify this string reversal code
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Reversed String befor simpilification :", reversed_string)
# Optimized Code
input_string = "Hello, World!"
reversed_string = input_string[::-1]
print("Reversed String after simpilification :", reversed_string)
# End of Task 2 .py
# Task 3: Modularize the string reversal code using functions
# String Reversal Using Functions
def reverse_string(s):
    return s[::-1]
input_string = "Hello, World!"
reversed_string = reverse_string(input_string)
print("Reversed String:", reversed_string)
```

Output:



```
Reversed String: !dlroW ,olleH
Reversed String befor simpilification : !dlroW ,olleH
Reversed String after simpilification : !dlroW ,olleH
Reversed String using function : !dlroW ,olleH
```

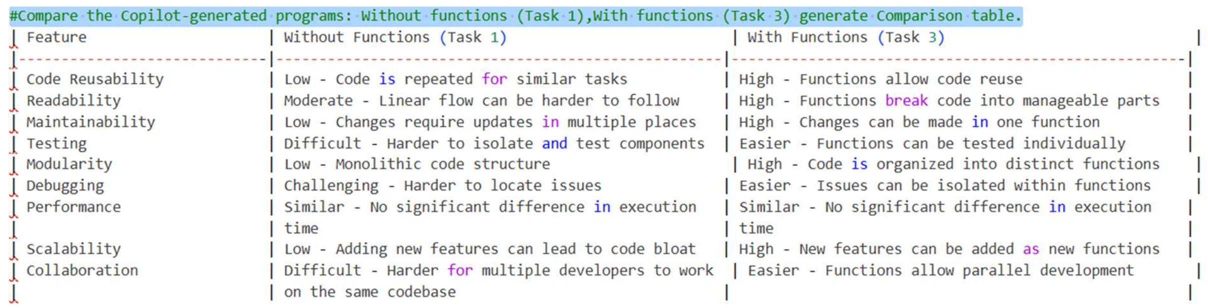
Justification :

By moving the string reversal logic into a separate function, the program becomes better structured and more readable. Rather than having all the code in a single block, the function clearly specifies the task it performs. This approach makes the program easier to test, simplifies debugging, and allows for greater flexibility in future modifications. Additionally, if the same functionality is required elsewhere, it can be reused without duplicating code. Overall, using modular design enhances the program's clarity, maintainability, and overall reliability.

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     | Similar - No significant difference in execution     |
|                              | time                                                 | 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   | Easier - Functions allow parallel development        |
|                              | on the same codebase                                 |                                                      |
```

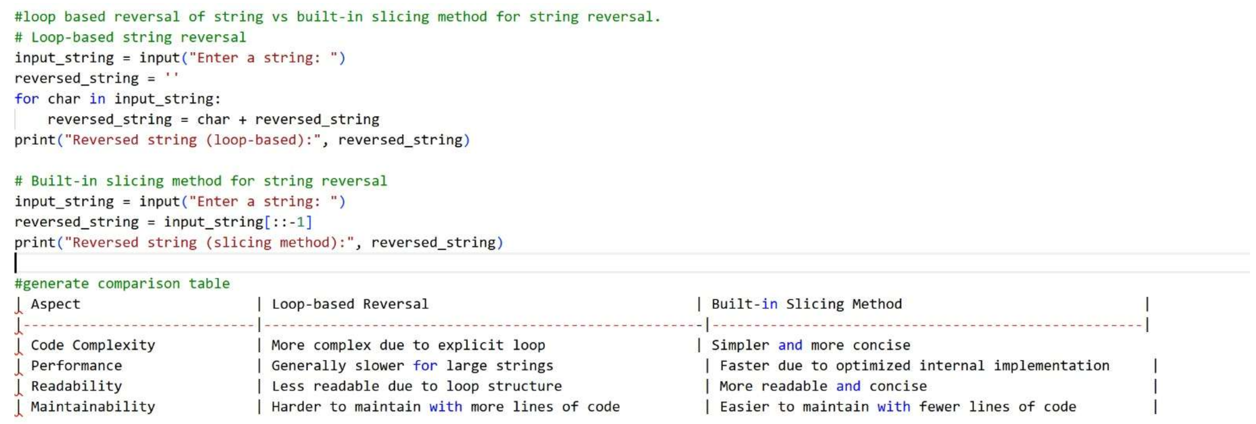
Justification:

The Copilot-generated program that does not use functions works well for very small tasks since all the logic is written in a single block. Although this approach keeps the code straightforward, it becomes harder to handle as the program increases in size. Code reuse is limited, and identifying errors takes more time because the logic is not separated.In contrast, the Copilot-generated program that uses functions follows a more organized and professional structure. Placing the string reversal logic inside a function improves readability and makes the code easier to understand. It also allows the same functionality to be reused in different parts of a program without duplication. Testing and debugging are more efficient because the function can be examined on its own. For larger or real-world applications, a modular design is more effective, easier to maintain, and more dependable. Therefore, using functions is a better approach for long-term software 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 method clearly demonstrates the process of reversing a string one character at a time, which makes it useful for understanding how the logic works internally. However, since strings in Python are immutable, each concatenation results in the creation of a new string, which reduces performance when handling large inputs. In contrast, the built-in slicing technique is more concise and significantly faster. It reverses the string in a single step and benefits from Python's internal optimizations. As a result, this approach is better suited for real-world scenarios where efficiency, readability, and clean code are essential. Overall, the loop-based approach is ideal for learning purposes, while the slicing method is preferred for efficient and professional development.