

# AI-Assisted String Reversal Using GitHub Copilot

**Name:** V.Abhilash

**Hall.No:** 2303A51922

**Batch :** 12

## Task 1: AI-Generated Logic Without Modularization

### Scenario

A basic text-processing utility is developed for a messaging application.

### Copilot Prompt Used

```
# write a code to reverse a string without using any functions  
# accept user input for the string to be reversed
```

Task1.py X

Task1.py > ...

```
1  
2 # task1  
3 # write a code to reverse a string without using any functions  
4 # accept user input for the string to be reversed  
5  
6 input_string = input("Enter a string to reverse: ")  
7 reversed_string = ""  
8  
9 for char in input_string:  
10     reversed_string = char + reversed_string  
11  
12 print("Reversed string (without functions):", reversed_string)  
13
```

### Sample Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python312/python.exe c:/Users/HP/la.py  
Enter a string: Abhilash  
Reversed string: hsalihbA  
PS C:\Users\HP>
```

## Explanation

The program reads user input and reverses the string using a loop without defining any functions. The logic is implemented directly in the main code.

## Task 2: Efficiency & Logic Optimization

### Scenario

The code is reviewed for readability and efficiency.

### Copilot Prompt Used

```
#simplify this string reversal code and improve its performance  
and readability
```



```
Task2.py  X  
Task2.py > ...  
1  # task2  
2  # simplify this string reversal code and improve its performance and readability  
3  
4  input_string = "Hello, World!"  
5  reversed_string = input_string[::-1]  
6  print(reversed_string)  
7
```

### Sample Output

```
!dlrow ,olleH
```

## Explanation

The slicing method simplifies the logic and removes unnecessary variables and loops.

The slicing method is faster in practice due to internal optimizations in Python.

Readability is significantly improved.

## Task 3: Modular Design Using Functions

### Scenario

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

## Copilot Prompt Used

```
# write code with function to reverse a string and comment each line and function call and no sample input and output
```

Code

```
Task3.py X
Task3.py > ...
1  # task3
2  # write code with function to reverse a string and comment each line and function call
3  # no sample input and output
4
5  def reverse_string(s):
6      # Initialize an empty string to hold the reversed string
7      reversed_s = ""
8
9      # Loop through each character in the input string
10     for char in s:
11         # Prepend the character to the reversed string
12         reversed_s = char + reversed_s
13
14     # Return the reversed string
15     return reversed_s
16
17 # Test case
18 print(reverse_string("Hello World"))
19
```

Sample Output

dlrow olleH

## Explanation

Using a function improves reusability, readability, and maintainability. The logic can now be reused across different modules.

## Task 4: Comparative Analysis – With vs Without Functions

### Procedural vs Modular Approach □ With vs Without Functions

## Copilot Prompt Used

#Create a Python program that prints a formatted comparison table showing the advantages of using functions over not using functions, and display a final recommendation.

```
#Task4
#Create a Python program that prints a formatted comparison table showing the advantages of using functions over not using fun
print("\n" + "="*80)
print("ANALYSIS: WITH vs WITHOUT FUNCTIONS")
print("="*80)
comparison_data = {
    "Criteria": ["Code Clarity", "Reusability", "Debugging Ease", "Large-Scale Suitability"],
    "Without Functions": [
        "Low - inline logic is scattered",
        "Poor - must rewrite code each time",
        "Difficult - hard to isolate errors",
        "Not viable - code duplication"
    ],
    "With Functions": [
        "High - logic is encapsulated",
        "Excellent - call function multiple times",
        "Easy - test individual functions",
        "Ideal - modular and maintainable"
    ]
}

# Print comparison table

print(f"\n{'Criteria':<25}{'Without Functions':<35}{'With Functions':<35}")
print("="*95)
for i, criteria in enumerate(comparison_data["Criteria"]):
    print(f"{'Criteria':<25} {'Without Functions':<35} {'With Functions':<35}")
print("\n" + "="*80)
print("RECOMMENDATION: Use functions for better maintainability and scalability")
print("="*80)
```

```
=====
ANALYSIS: WITH vs WITHOUT FUNCTIONS
=====

Criteria                Without Functions                With Functions
-----
Code Clarity            Low - inline logic is scattered    High - logic is encapsulated
Reusability             Poor - must rewrite code each time Excellent - call function multiple times
Debugging Ease          Difficult - hard to isolate errors Easy - test individual functions
Large-Scale Suitability Not viable - code duplication      Ideal - modular and maintainable

=====
RECOMMENDATION: Use functions for better maintainability and scalability
=====
```

## Conclusion

Function-based design is preferred for large-scale applications due to better organization and reusability.

# Task 5: Loop-Based vs Built-In String Reversal Copilot

## Prompt Use

```
Task4.py x
Task4.py > ...
1 # task5
2 # a loop based string reversal function with comments and a built-in slicing based
3 # string reversal function and comparison discussion
4
5 def reverse_string_loop(s):
6     # Initialize an empty string to hold the reversed string
7     reversed_s = ""
8     # Loop through each character in the input string
9     for char in s:
10         # Prepend the character to the reversed string
11         reversed_s = char + reversed_s
12
13     # Return the reversed string
14     return reversed_s
15 # built-in slicing based string reversal function
16 def reverse_string_slicing(s):
17     # Use slicing to reverse the string
18     return s[::-1]
19 print(reverse_string_loop("Python"))
20 print(reverse_string_slicing("Python"))
21
22 # Comparison discussion
23 """
24 The loop-based string reversal function iterates through each character of the input string
25 and constructs the reversed string by prepending each character. This method is straightforward
26 and easy to understand, but it may be less efficient for very long strings due to the repeated
27 string concatenation, which can lead to higher time complexity.
28 """
```

Output:

nohtyp

nohtyp

## Comparison Discussion

The loop-based approach reverses the string by checking each character one by one. It helps beginners understand how string reversal works.

The slicing-based approach uses Python's built-in feature to reverse the string. It is shorter and easier to read.

Both methods take the same amount of time, which is **O(n)**, where *n* is the length of the string.

However, slicing works faster for large strings because it is optimized inside Python.

The loop-based method is good for learning, while the slicing method is better for real applications.

## **Conclusion**

GitHub Copilot helped in writing, improving, and comparing different string reversal programs. Using functions and built-in features makes the code easier to read, faster, and better for large programs.

## **Declaration**

All the code and explanations were created using GitHub Copilot and were checked manually to ensure correctness.