AI-Assisted String Reversal Using GitHub Copilot

Name: P.Koushik

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
```

```python
Assignment1 > task1.py > ...
1    #task1
2    # write a code to reverse a string without using any functions
3    # accept user input for the string to be reversed
4
5    input_string = input("Enter a string to reverse: ")
6    reversed_string = ""
7    for char in input_string:
8        reversed_string = char + reversed_string
9    print("Reversed string (without functions):", reversed_string)
10
11
```

Sample Output

```
PS C:\Users\admin\OneDrive\Desktop\AI ASSISTED> python task1.py
Enter a string to be reversed: koushik
Reversed string (without functions): kihsuok
PS C:\Users\admin\OneDrive\Desktop\AI ASSISTED>
```

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
```

```
Assignment1 > 🐍 task2.py > ...
  1    #task2
  2    #simplify this string reversal code and improve its performance and readability
  3    input_string = "Hello, World!"
  4    reversed_string = input_string[::-1]
  5    print(reversed_string)
```

Sample Output

```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\AI ASSISTED\task2.py"
!dlroW ,olleH

[Done] exited with code=0 in 0.317 seconds
```

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

```
Assignment1 > 🐍 task3.py > ...
 1   #task3
 2   # write code with function to reverse a string and comment each line and function call and no sample input and output
 3   def reverse_string(s):
 4       # Initialize an empty string to hold the reversed string
 5       reversed_s = ""
 6       # Loop through each character in the input string
 7       for char in s:
 8           # Prepend the character to the reversed string
 9           reversed_s = char + reversed_s
10       # Return the reversed string
11       return reversed_s
12   # Test case
13   print(reverse_string("Hello World"))
```

Sample Output

```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\AI ASSISTED\task3.py"
!dlroW ,olleH

[Done] exited with code=0 in 0.176 seconds
```

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

Aspect	Without Functions	With Functions
Code Clarity	Less organized	More structured
Reusability	Cannot be reused	Easily reusable
Debugging	Harder	Easier
Scalability	Not suitable	Suitable
Best Use	Small scripts	Large applications

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

```
Assignment1 > task5.py > ...
1   #task5
2   # a loop based string reversal function with comments and a built in slicing based string reversal function and comparision disc
3   def reverse_string_loop(s):
4       # Initialize an empty string to hold the reversed string
5       reversed_s = ""
6       # Loop through each character in the input string
7       for char in s:
8           # Prepend the character to the reversed string
9           reversed_s = char + reversed_s
10      # Return the reversed string
11      return reversed_s
12
13  #built-in slicing based string reversal function
14  def reverse_string_slicing(s):
15      # Use slicing to reverse the string
16      return s[::-1]
17  print(reverse_string_loop("Python"))
18  print(reverse_string_slicing("Python"))
19
20
21  # Comparison discussion
22  """
23  The loop-based string reversal function iterates through each character of the input string
24  and constructs the reversed string by prepending each character. This method is straightforward
25  and easy to understand, but it may be less efficient for very long strings due to the repeated string concatenation,
26  which can lead to higher time complexity."""
27  |
28
```

Output:

```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\AI ASSISTED\task4.py"
nohtyP
nohtyP

[Done] exited with code=0 in 0.294 seconds
```

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.