

Functional Programming Lab 03

K M Anisul Islam

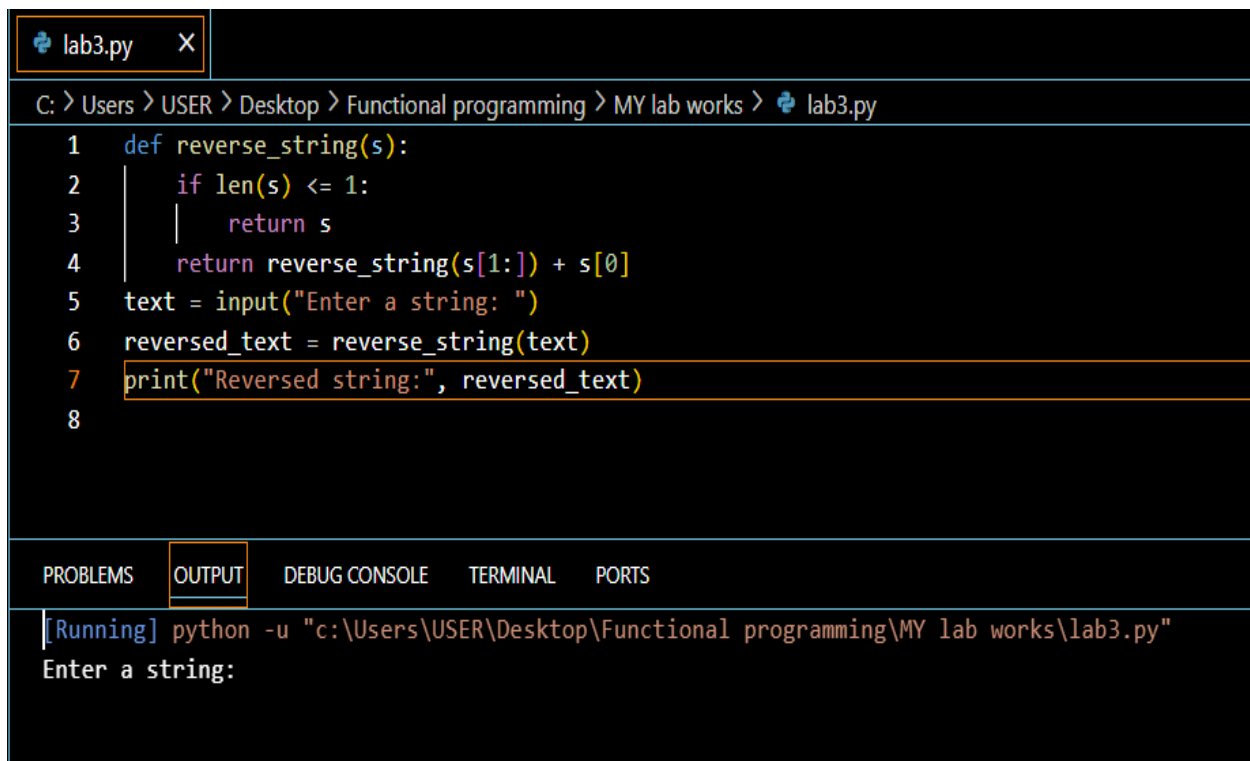
Email: ISLAM.K@stud.satbayev.university

Objective: To study and apply recursive functions in Python for solving classical algorithmic and mathematical problems. The goal of this work is to develop students' skills in writing recursive code, understanding its principles and limitations, and learning to analyze and optimize recursive algorithms.

Individual Task:

String Reversal Using Recursion

Implement a recursive function that reverses a string input by the user.



```
lab3.py X
C: > Users > USER > Desktop > Functional programming > MY lab works > lab3.py
1 def reverse_string(s):
2     if len(s) <= 1:
3         return s
4     return reverse_string(s[1:]) + s[0]
5 text = input("Enter a string: ")
6 reversed_text = reverse_string(text)
7 print("Reversed string:", reversed_text)
8

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[Running] python -u "c:\Users\USER\Desktop\Functional programming\MY lab works\lab3.py"
Enter a string:
```

1. 1. What is recursion, and how does it work in the context of programming?
It is a programming technique where a function calls itself to solve a smaller part of the same problem. It continues until it reaches a base case, which stops further calls.
2. What are the main components of a recursive function?
Base case and Recursive case
3. What are the advantages and disadvantages of using recursion?
It makes code shorter and easier to understand but can use more memory and may cause a stack overflow if not written carefully.
4. Provide an example of a problem best solved using recursion.
Problems like factorial calculation, Fibonacci numbers, or tree traversal.
5. How can stack overflow be prevented in recursion?
It could be prevented by defining a clear base case and avoiding too many recursive calls. Converting recursion to iteration also helps.
6. What is tail recursion, and why is it important?
Tail recursion means the recursive call is the last operation in the function. It helps some languages optimize memory use, though Python does not support this optimization.
7. How can memoization be implemented in a recursive function?
Memoization stores results of previous function calls in a dictionary so repeated inputs do not trigger new recursive calls.
8. Can you compare recursive and iterative approaches to solving problems?
Recursion is easier to write and understand, while iteration is faster and uses less memory. Both solve problems differently but achieve the same goal.
9. What common mistakes occur when writing recursive functions, and how can they be avoided?
Common mistakes include missing the base case or writing a wrong return statement. Testing small inputs first can help catch these errors.
10. How can recursion be used to traverse data structures such as trees and graphs?
Recursion can visit each node or element of trees and graphs one by one by calling the same function on connected or child nodes.