NAME : KOVI SAI GANESH

REGISTER NUMBER : 192124028

DAY 4

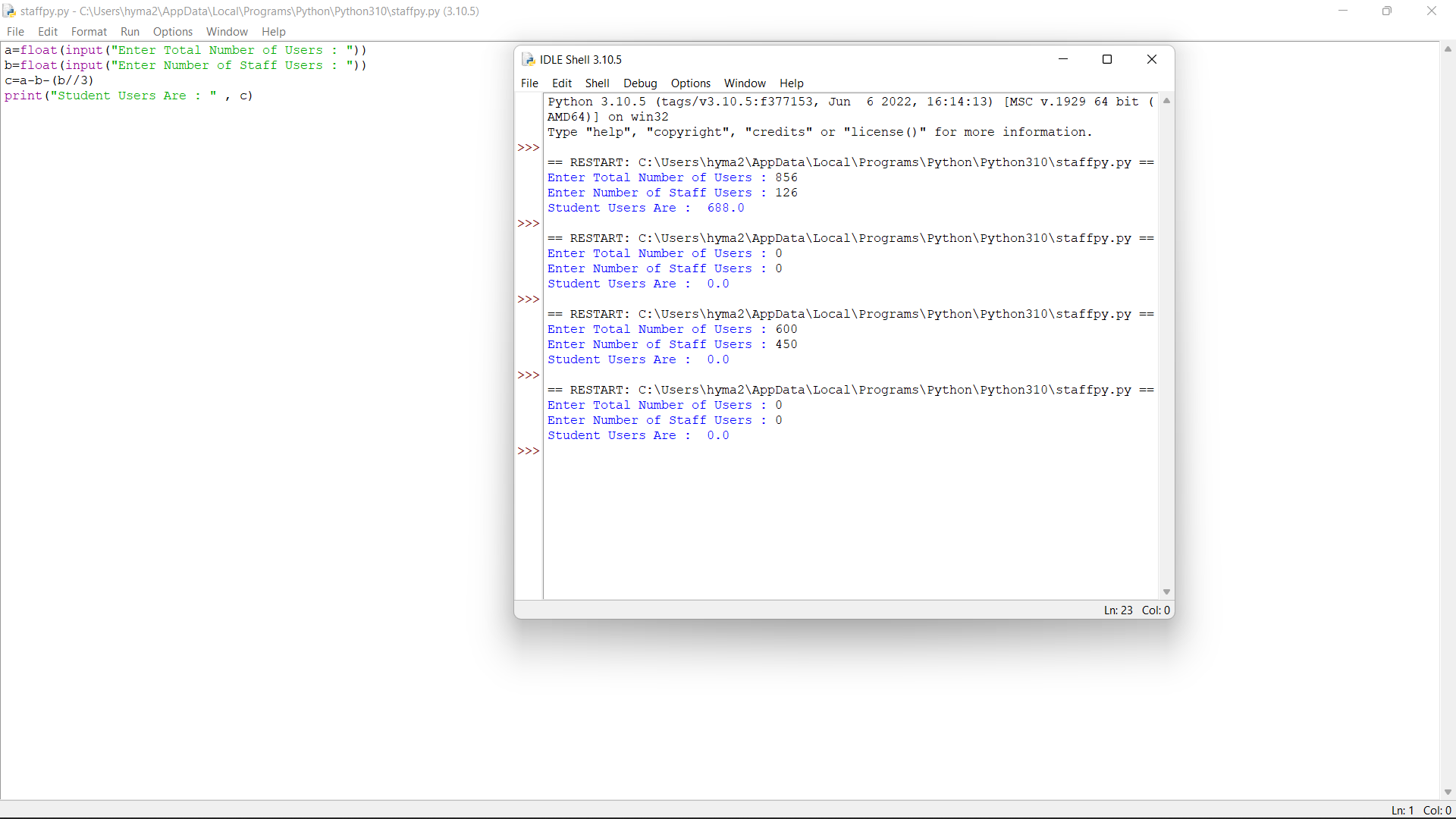
2Q Write a program to find the number of student users in the college, get the total users, staff users details from the client. Note for every 3 staff user there is one Non-teaching staff user assigned by default. Sample Input: Total Users: 856 Staff Users: 126 Sample Output: Student Users: 688 Test Cases: 1. Total User: 0 2. Total User: -143 3. Total User: 1026, Staff User: 1026 4. Total User: 450, Staff User: 540 5. Total User: 600, Staff User: 450

a=float(input("Enter Total Number of Users : "))

b=float(input("Enter Number of Staff Users : "))

c=a-b-(b//3)

print("Student Users Are : " , c)



4Q Valid Palindrome A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers. Given a string s, return true if it is a palindrome, or false otherwise. Test Cases: 1.Input: s = "A man, a plan, a canal: Panama" Output: true 2.Input: s = "race a car" Output: false 3.Input: s = " " Output: true 4. s= “madam” 5.s= “honest”

def first\_letter\_index(str, left, right):

index = -1

for i in range(left, right + 1):

if str[i] >= 'a' and str[i] <= 'z' :

index = i

break

return index

def last\_letter\_index(str, left, right):

index = -1

for i in range(left, right - 1, -1) :

if str[i] >= 'a' and str[i] <= 'z':

index = i

break

return index

def solve(str):

left = 0

right = len(str) - 1

flag = True

for i in range(len(str)) :

left = first\_letter\_index(str, left, right)

right = last\_letter\_index(str, right, left)

if right < 0 or left < 0:

break

if str[left] == str[right]:

left += 1

right -= 1

continue

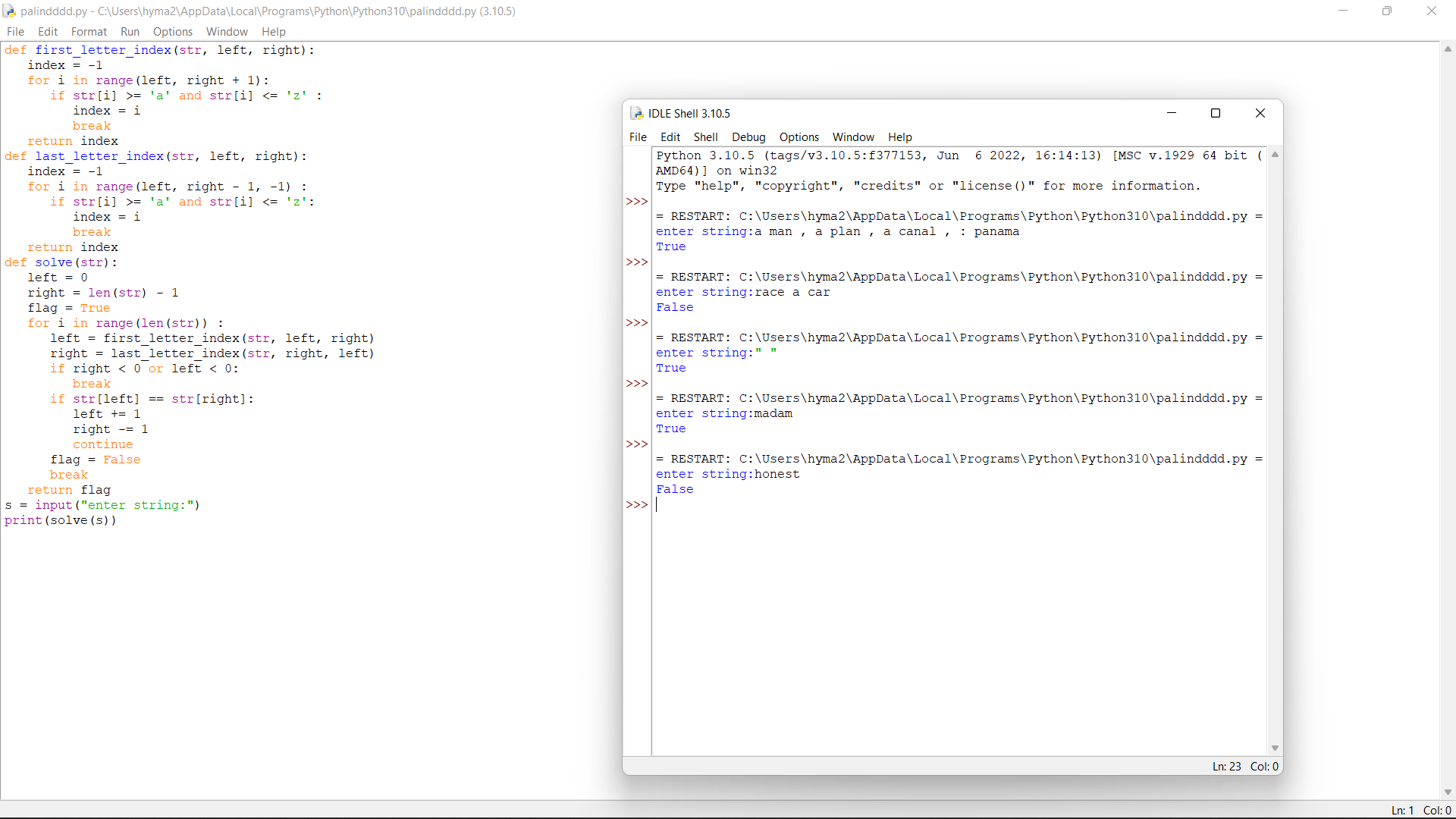
flag = False

break

return flag

s = input("enter string:")

print(solve(s))



5Q Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, they cannot move through that element. If the end isn’t reachable, return -1. Test Case: 1.Input: arr[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9] Output: 3 (1-> 3 -> 9 -> 9) 2.Input: arr[] = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] Output: 10 3.Input: arr[] = [2,3,1,1,4] Output: 2 4.Input: arr[] = [1, 3, 6, 1, 0, 9] Output: 3 5.Input: arr[] = [2,3,0,1,4] Output: 2

def minJumps(arr, l, h):

if (h == l):

return 0

if (arr[l] == 0):

return float('inf')

min = float('inf')

for i in range(l + 1, h + 1):

if (i < l + arr[l] + 1):

jumps = minJumps(arr, i, h)

if (jumps != float('inf') and

jumps + 1 < min):

min = jumps + 1

return min

arr=eval(input("Enter list:"))

n=len(arr)

print('Minimum number of jumps to reach',

'end is', minJumps(arr, 0, n-1))

7Q Count Sorted Vowel Strings Given an integer n, return the number of strings of length n that consist only of vowels (a, e, i, o, u) and are lexicographically sorted. A string s is lexicographically sorted if for all valid i, s[i] is the same as or comes before s[i+1] in the alphabet. Test Cases: 1.Input: n = 1 Output: 5 Explanation: The 5 sorted strings that consist of vowels only are ["a","e","i","o","u"]. 2.Input: n = 2 Output: 15 Explanation: The 15 sorted strings that consist of vowels only are ["aa","ae","ai","ao","au","ee","ei","eo","eu","ii","io","iu","oo","ou","uu"]. Note that "ea" is not a valid string since 'e' comes after 'a' in the alphabet. 3.Input: n = 33 Output: 66045 4.n=55 5=32

def countstrings(n, start):

if n == 0:

return 1

cnt = 0

for i in range(start, 5):

cnt += countstrings(n - 1, i)

return cnt

def countVowelStrings(n):

return countstrings(n, 0)

n = int(input("Enter the N Value : "))

print(countVowelStrings(n))

9Q The year is divided into four seasons: spring, summer, fall and winter. While the exact dates that the seasons change vary a little bit from year to year because of the way that the calendar is constructed, we will use the following dates for this exercise: Season First day Summer March 20 Spring June 21 Fall September 22 Winter December 21 Create a program that reads a month and day from the user. The user will enter the name of the month as a string, followed by the day within the month as an integer. Then your program should display the season associated with the date that was entered. Note: Enter First three letter for month example: Jan for January, Feb for February and so on....and first letter of the month should be capital Input: Enter the month: march Enter the date: 21 Output: The season is currently summer Test Cases: 1. July, 29 2. September, 5 3. December, 30 4. March, 12 5. June, 27

month = input("Input the month (e.g. January, February etc.): ")

day = int(input("Input the day: "))

if month in ('January', 'February', 'March'):

season = 'winter'

elif month in ('April', 'May', 'June'):

season = 'summer'

elif month in ('July', 'August', 'September'):

season = 'spring'

else:

season = 'autumn'

if (month == 'March') and (day > 19):

season = 'summer'

elif (month == 'June') and (day > 20):

season = 'summer'

elif (month == 'September') and (day > 21):

season = 'autumn'

elif (month == 'December') and (day > 20):

season = 'winter'

print("Season is",season)

10Q 10. Scramble String We can scramble a string s to get a string t using the following algorithm: If the length of the string is 1, stop. If the length of the string is > 1, do the following: Split the string into two non-empty substrings at a random index, i.e., if the string is s, divide it to x and y where s = x + y. Randomly decide to swap the two substrings or to keep them in the same order. i.e., after this step, s may become s = x + y or s = y + x. Apply step 1 recursively on each of the two substrings x and y. Given two strings s1 and s2 of the same length, return true if s2 is a scrambled string of s1, otherwise, return false. Test cases: 1.Input: s1 = "great", s2 = "rgeat" Output: true 2.Input: s1 = "abcde", s2 = "caebd" Output: false 3.Input: s1 = "a", s2 = "a" Output: true 4.s1=”ab” s2= “ad” 5s1=10 s2=-5

def isScramble(S1: str, S2: str):

if len(S1) != len(S2):

return False

n = len(S1)

if not n:

return True

if S1 == S2:

return True

if sorted(S1) != sorted(S2):

return False

for i in range(1, n):

if (isScramble(S1[:i], S2[:i]) and

isScramble(S1[i:], S2[i:])):

return True

if (isScramble(S1[-i:], S2[:i]) and

isScramble(S1[:-i], S2[i:])):

return True

return False

S1 = input("enter string1:")

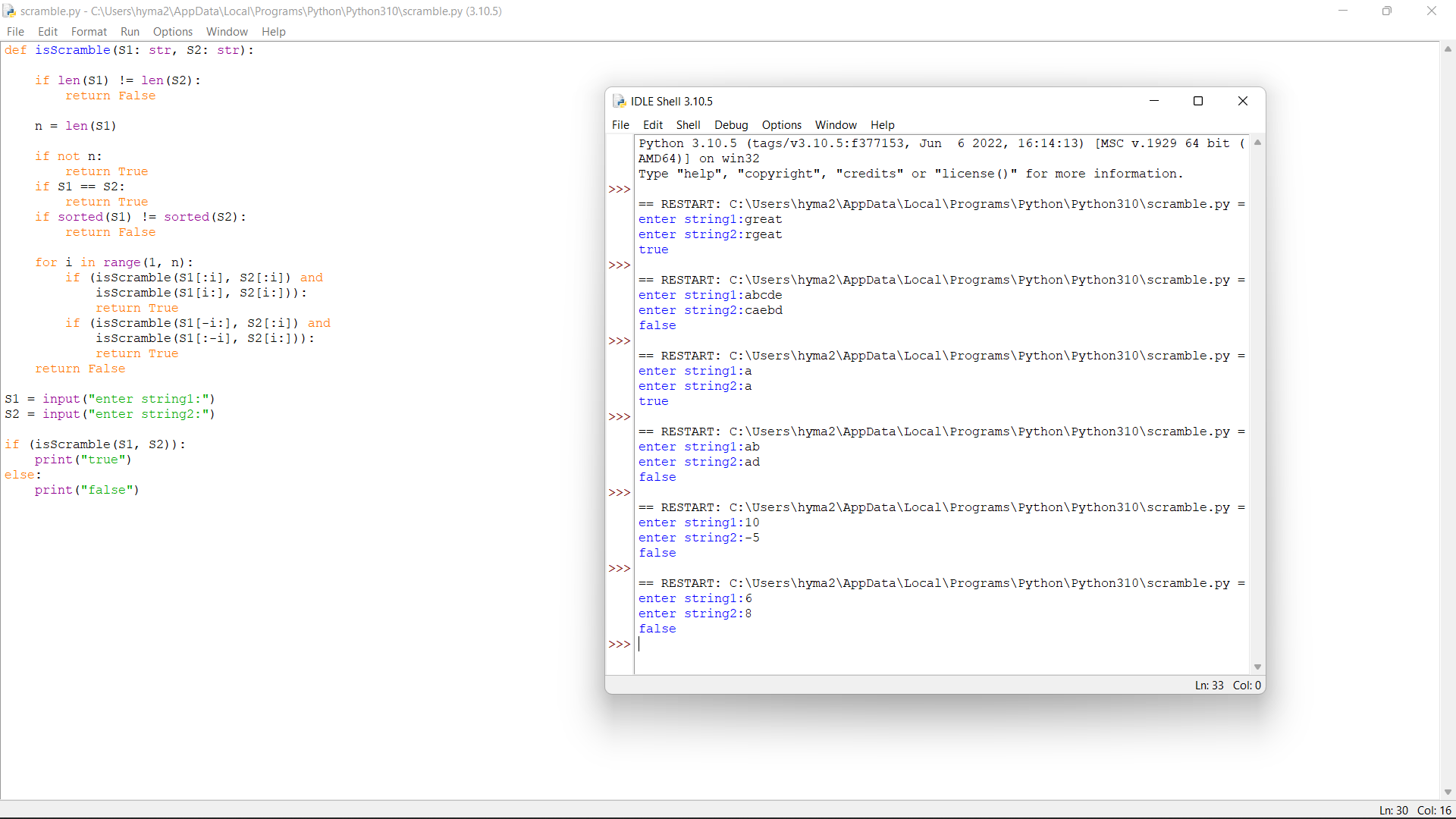
S2 = input("enter string2:")

if (isScramble(S1, S2)):

print("Yes")

else:

print("No")



number of elements:"))

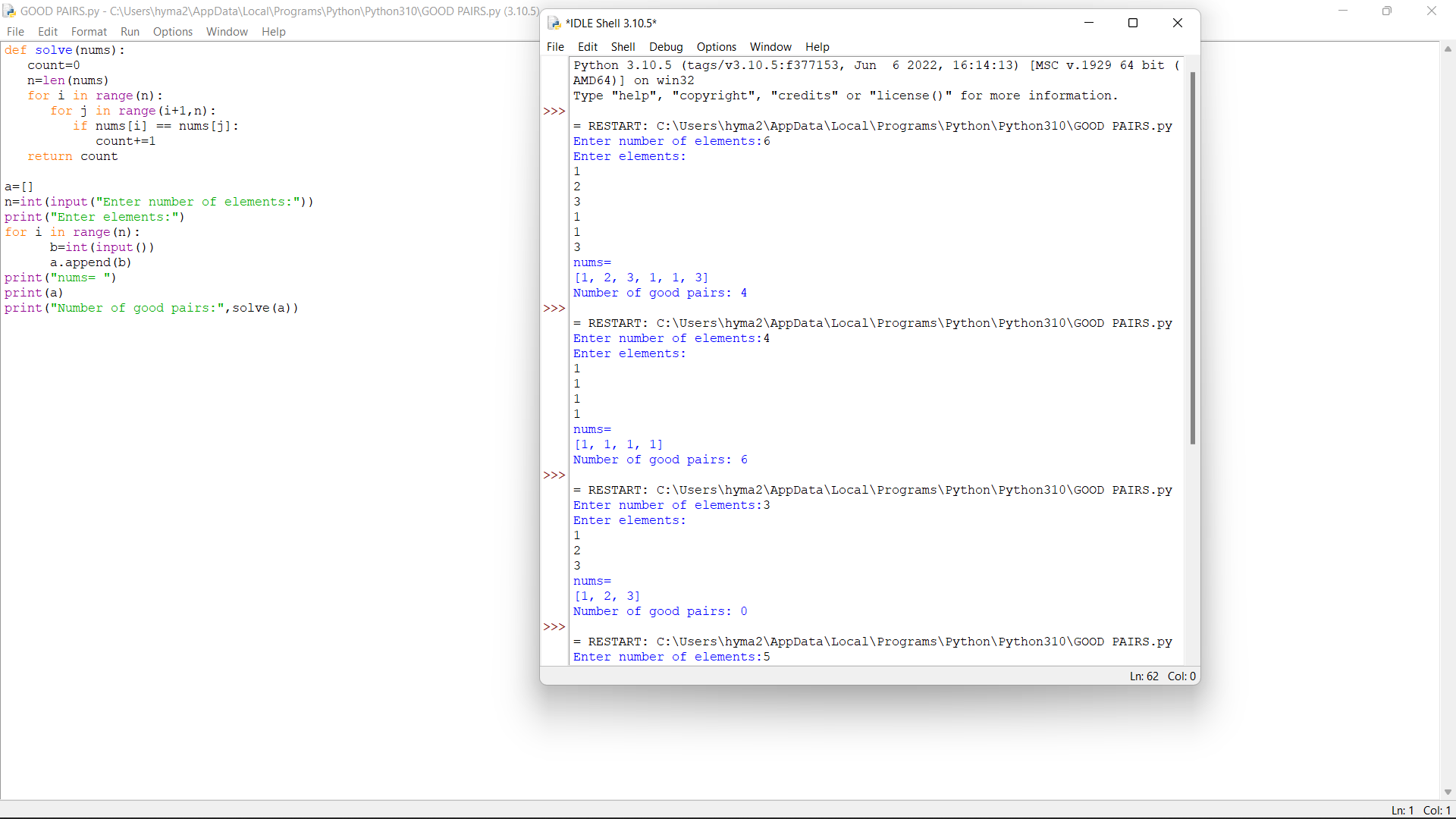
print("Enter elements:")

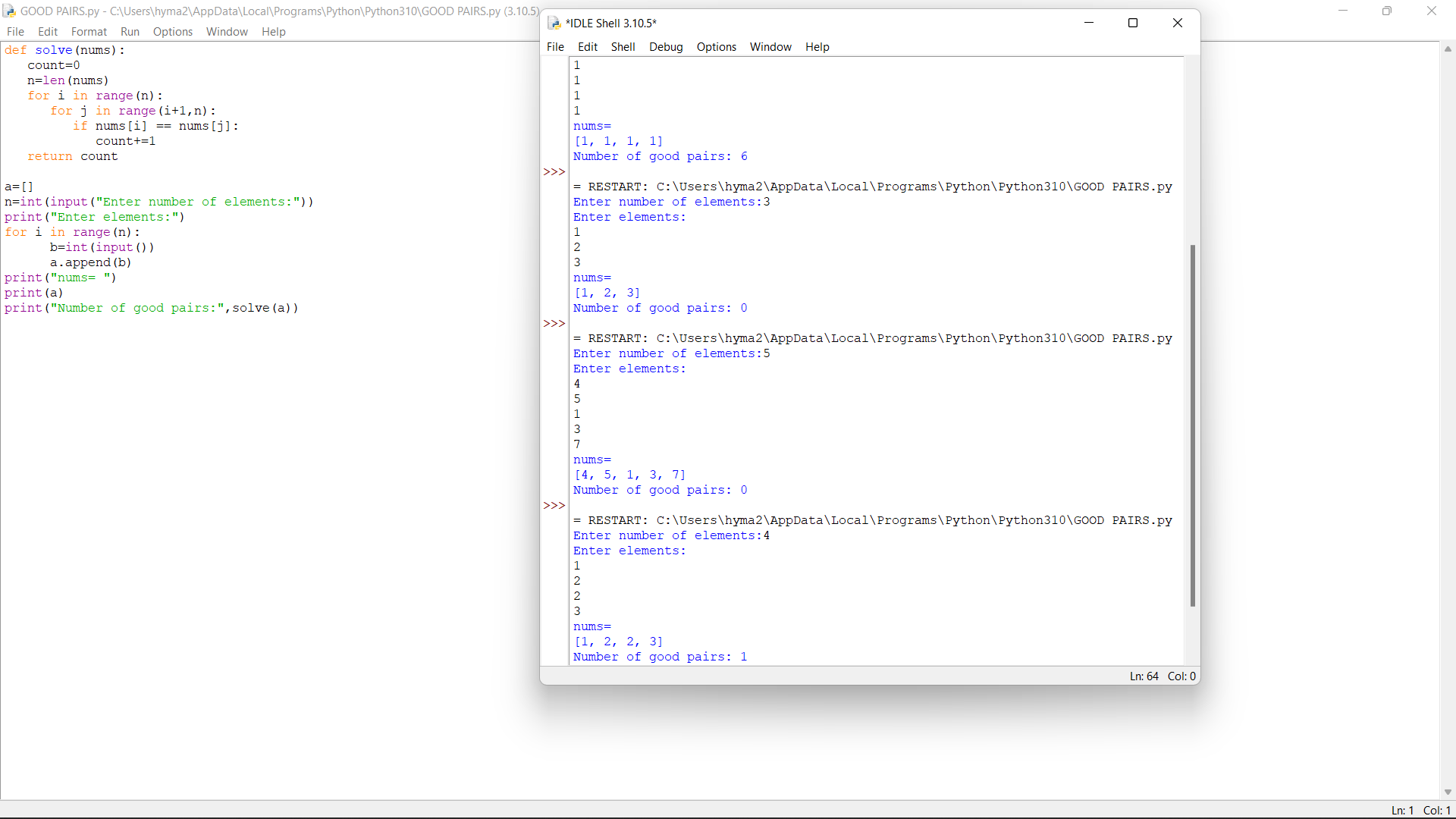
for i in range(n):

b=int(input())

a.append(b)

print("Number of good pairs:",solve(a))





7q Generate Parentheses Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

def printParenthesis(str, n):

if(n > 0):

\_printParenthesis(str, 0,

n, 0, 0)

return

def \_printParenthesis(str, pos, n,

open, close):

if(close == n):

for i in str:

print(i, end="")

print()

return

else:

if(open > close):

str[pos] = ')'

\_printParenthesis(str, pos + 1, n,

open, close + 1)

if(open < n):

str[pos] = '('

\_printParenthesis(str, pos + 1, n,

open + 1, close)

n = int(input("n="))

str = [" "] \* 2 \* n

printParenthesis(str , n)

