1. match\_ends

Given a list of strings, return the count of the number of strings where the string length is 2 or more and the first and last chars of the string are the same.

**CODE:**

**flag = 0**

**list\_of\_strings = []**

**count=0**

**# check valid strings**

**def check\_str(string):**

**if len(string) > 1 and string[0] == string[len(string) - 1]:**

**return True**

**return False**

**while flag != 1:**

**list\_of\_strings.append(input("Enter the string:\t"))**

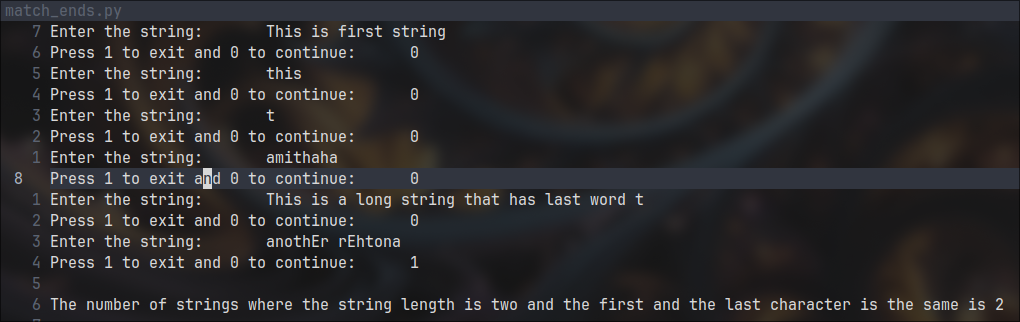
**flag = int(input("Press 1 to exit and 0 to continue:\t"))**

**for string in list\_of\_strings:**

**count += check\_str(string)**

**print(f"\nThe number of strings where the string length is two and the first and the last character is the same is {count}")**

**OUTPUT:**



2. front\_x

Given a list of strings, return a list with the strings in sorted order, except group all the strings that begin with 'x' first.

e.g. ['mix', 'xyz', 'apple', 'xanadu', 'aardvark'] yields ['xanadu', 'xyz', 'aardvark', 'apple', 'mix']

**CODE:**

**flag = 0**

**list\_of\_strings = []**

**list\_of\_strings\_x = []**

**while flag != 1:**

**list\_of\_strings.append(input("Enter the string:\t"))**

**flag = int(input("Press 1 to exit and 0 to continue:\t"))**

**list\_of\_strings.sort()**

**for string in list\_of\_strings[::-1]:**

**if string[0] == 'x':**

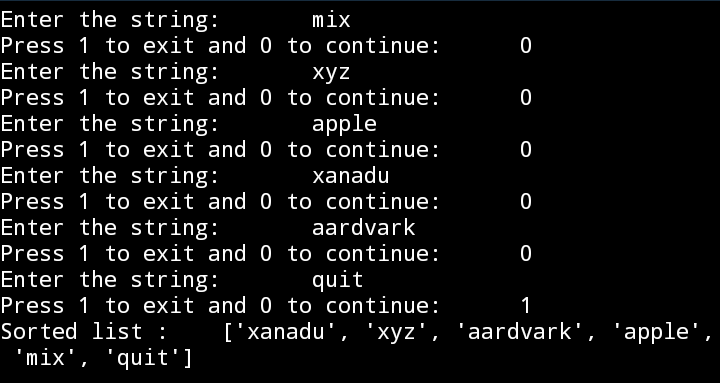
**list\_of\_strings.remove(string)**

**list\_of\_strings\_x.insert(0,string)**

**list\_of\_strings\_x += list\_of\_strings**

**print("Sorted list :\t",list\_of\_strings\_x)**

**OUTPUT:**

****

Hint: this can be done by making 2 lists and sorting each of them before combining them.

3. Given two lists sorted in increasing order, create and return a merged list of all the elements in sorted order. You may modify the passed in lists. Ideally, the solution should work in "linear" time, making a single pass of both lists.

**CODE:**

list1 = [1, 2, 3, 5, 7]

list2 = [2, 4, 6, 8]

merged\_list = []

i = 0

j = 0

while i < len(list1) and j < len(list2):

if list1[i] < list2[j]:

merged\_list.append(list1[i])

i += 1

else:

merged\_list.append(list2[j])

j += 1

# Add any remaining elements from list1.

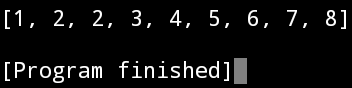
merged\_list.extend(list1[i:])

# Add any remaining elements from list2.

merged\_list.extend(list2[j:])

print(merged\_list)

**OUTPUT:**

****

4. both\_ends

Given a string s, return a string made of the first 2 and the last 2 chars of the original string, so 'spring' yields 'spng'. However, if the string length is less than 2, return instead the empty string.

**CODE:**

def both\_ends(string):

if len(string) < 2 :

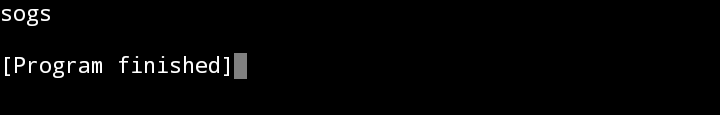
return ""

return string[:2] + string[-2:]

if \_\_name\_\_ == '\_\_main\_\_' :

print(both\_ends("somethings"))

**OUTPUT:**

****

5. fix\_start

Given a string s, return a string where all occurences of its first char have been changed to '\*', except do not change the first char itself.

e.g. 'babble' yields 'ba\*\*le'

Assume that the string is length 1 or more.

**CODE:**

**def fix\_start(string):**

**if len(string) == False:**

**return**

**i = 1**

**new\_string= string[0]**

**while i < len(string):**

**if string[i] == string[0]:**

**new\_string += '\*'**

**else:**

**new\_string += string[i]**

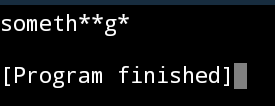
**i += 1**

**return new\_string**

**if \_\_name\_\_ == '\_\_main\_\_' :**

**print(fix\_start("somethssgs"))**

**OITPUT:**

****

6. MixUp

Given strings a and b, return a single string with a and b separated by a space '<a> <b>', except swap the first 2 chars of each string.

e.g.

'mix', pod' -> 'pox mid'

'dog', 'dinner' -> 'dig donner'

Assume a and b are length 2 or more.

**CODE:def mix\_up(a, b):**

**if len(a) >= 2 and len(b) >= 2:**

**return b[:2] + a[2:] + ' ' + a[:2] + b[2:]b**

**else:**

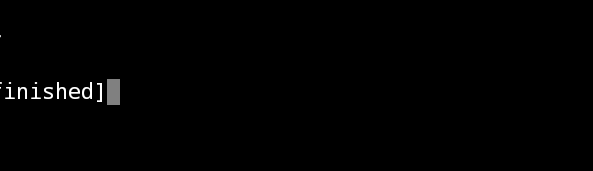
**return "Invalid input: Both strings should be of length 2 or more."**

**if \_\_name\_\_ == '\_\_main\_\_':**

**print(mix\_up('mix', 'pod'))**

**print(mix\_up('dog', 'dinner'))**

**OUTPUT:**

****

7. not\_bad

Given a string, find the first appearance of the substring 'not' and 'bad'. If the 'bad' follows the 'not', replace the whole 'not'...'bad' substring with 'good'. Return the resulting string.

So 'This dinner is not that bad!' yields: This dinner is good!

**CODE:**

**def not\_bad(string):**

**index\_not = string.find("not") # returns the index of not**

**index\_bad = string[index\_not:].find("bad")**

**if index\_bad == -1:**

**return string**

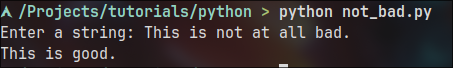
**return string[:index\_not] + "good" + string[index\_bad + index\_not + 3 :]**

**if \_\_name\_\_ == "\_\_main\_\_":**

**string = input("Enter a string:\t")**

**print(not\_bad(string))**

**OUTPUT:**

****