İbrahim Talha ASAN
COE-64170019

Assignment 2 Report

Problem 1:

Definition: The purpose Problem 1 is to find out how many of the given string is in the given path, without discriminating uppercase and lowercase letters, and return the number it finds.

Program Code:

```
import os
from glob import glob

givenString = input("Enter string: ") #input string
PATH = "C:\src\python_projects" #define path.
files = [b for a in os.walk(PATH) for b in glob(os.path.join(a[0], '*'))] #give all file

takenString = givenString
givenString = givenString.lower()#for discriminating uppercase and lowercase
count = 0

for i in files:
        if i.lower().find(givenString) != -1: #finding file with the same name as input
                count = count+1

print("{} occurences for string {} ".format(count,takenString))#printing the number of files
                                                                # with the same name as input.
```

Program Outputs:

Enter string: list

5 occurences for string list

Discussions: There is no problem with Problem 1.

İbrahim Talha ASAN
COE-64170019

Problem 2:

Definition: The purpose of Problem 2 is to return true if the elements of the two given lists are the same, and if the elements of the two lists are not the same, write a function that returns which elements are different.

Program Code:

```
def isPermutations(list1, list2):#take parameter as two list
  l1 = sorted(list1)#sort a using the sorted (built in) function and assign c.
  l2 = sorted(list2)#sort b using the sorted (built in) function and assign d.

  requiredElementsList1=[]#additional  for l1
  requiredElementsList2 =[] # additional  for l2

  if (l1 == l2):#contol the list elements
    return True
  else:
    count = 0#the number of elements that should be added to the lists.
    for i in range (1,len(l1)):
      if l1[i] not in l2:
        for j in range (len(l1)):
            if(l1[j] not in l2 ):
              requiredElementsList2.append(l1[j])#elements that should be added to l2 were found and these
              count+=1                                  # elements were added to a new list.

      elif l2[i] not in l1:
        for k in range(len(l2)):
          if(l2[k] not in l1):
            requiredElementsList1.append(l2[k])#elements that should be added to l1 were found and these
            count+=1                                  # elements were added to a new list.
      else:
        requiredElementsList2.append(l1[i])#If the same element exists, these elements are added to the list again.
        count+=1

        return print("list1 needs {} ,list2 needs {} to make them permutations
        .".format(requiredElementsList1,requiredElementsList2))

print(isPermutations([10, 9, 11, 1], [9, 1, 11, 10]))
print(isPermutations([10, 9, 1, 10], [8, 1, 11, 10]))
```

Program Outputs:

True

list1 needs [8, 11] ,list2 needs [9, 10] to make them permutations .

Discussions: There is no problem with Problem 2.

İbrahim Talha ASAN
COE-64170019

Problem 3:

Definition: The purpose of Problem 3 is to create all sub strings that may consist of the strings given to it (must be in the order of letters in the given string) and print them in a list format.

Program Code:

```
def findSubString(givenString, lengthOfGivenString):#find and print all sub strings.
   sublists=[]#the resulting sublists are added to this list.
   print("[ ")
   for iter in range(1, lengthOfGivenString + 1):
      for i in range(lengthOfGivenString - iter + 1):
         j = i + iter - 1
         for k in range(i, j + 1):
            sublists.append(givenString[k])#append givenString to the sublists.
            print(sublists[k],end=",")#print sublists.
         print()

   print("]")

subString = input("Enter string: ") #give string
findSubString(subString, len(subString))
```

Program Outputs:

- Testing “akraba”
  [
  a,
  k,
  r,
  a,
  b,
  a,
  a,k,
  k,r,
  r,a,
  a,b,
  b,a,
  a,k,r,
  k,r,a,
  r,a,b,
  a,b,a,
  a,k,r,a,
  k,r,a,b,
  r,a,b,a,
  a,k,r,a,b,
  k,r,a,b,a,
  a,k,r,a,b,a,
  ]

İbrahim Talha ASAN
COE-64170019

- Testing “kaya”
  [
  k,
  a,
  y,
  a,
  k,a,
  a,y,
  y,a,
  k,a,y,
  a,y,a,
  k,a,y,a,
  ]

- Testing “keskin”
  [
  k,
  e,
  s,
  k,
  i,
  n,
  k,e,
  e,s,
  s,k,
  k,i,
  i,n,
  k,e,s,
  e,s,k,
  s,k,i,
  k,i,n,
  k,e,s,k,
  e,s,k,i,
  s,k,i,n,
  k,e,s,k,i,
  e,s,k,i,n,
  k,e,s,k,i,n,
  ]

Discussions: There is no problem with Problem 3.