**08 – Tuple/Set**

**Ex. No. : 8.1 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**Binary String**

Coders here is a simple task for you, Given string str. Your task is to check whether it is a binary string or not by using python set.

**Examples:**

**Input**: t = (5, 6, 5, 7, 7, 8 ), K = 13   
**Output**: 2   
Explanation:   
Pairs with sum K( = 13) are  {(5, 8), (6, 7), (6, 7)}.   
Therefore, distinct pairs with sum K( = 13) are { (5, 8), (6, 7) }.   
Therefore, the required output is 2.

For example:

| Input | Result |
| --- | --- |
| 1,2,1,2,5  3 | 1 |
| 1,2  0 | 0 |

**PROGRAM:**

def is\_binary\_string(input\_str):

return set(input\_str) <= {'0', '1'}

# Test the function

input\_str1 = input()

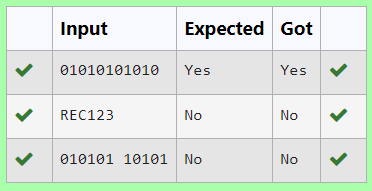
if is\_binary\_string(input\_str1):

print("Yes")

else:

print("No")

**OUTPUT:**

****

**Ex. No. : 8.2 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**Check Pair**

There is a malfunctioning keyboard where some letter keys do not work. All other keys on the keyboard work properly.

Given a string text of words separated by a single space (no leading or trailing spaces) and a string brokenLetters of all distinct letter keys that are broken, return the number of words in text you can fully type using this keyboard.

Example 1:

Input: text = "hello world", brokenLetters = "ad"

Output:

1

Explanation: We cannot type "world" because the 'd' key is broken.

**For example:**

| **Input** | **Result** |
| --- | --- |
| hello world  ad | 1 |
| Faculty Upskilling in Python Programming  ak | 2 |

**PROGRAM:**

def count\_words\_typed(text, brokenLetters):

count = 0

for word in text.split():

if all(letter not in brokenLetters for letter in word):

count += 1

return count

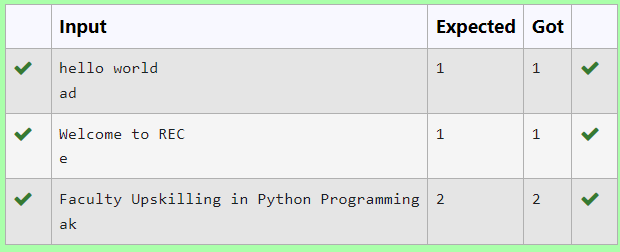
# Test the function

text = input().lower()

brokenLetters = input()

print(count\_words\_typed(text, brokenLetters))

**OUTPUT:**

****

**Ex. No. : 8.3 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**DNA Sequence**

The **DNA sequence** is composed of a series of nucleotides abbreviated as 'A', 'C', 'G', and 'T'.

For example, "ACGAATTCCG" is a **DNA sequence**.

When studying **DNA**, it is useful to identify repeated sequences within the DNA.

Given a string s that represents a **DNA sequence**, return all the **10-letter-long** sequences (substrings) that occur more than once in a DNA molecule. You may return the answer in **any order**.

**Example 1:**

**Input:**nums = [1,3,4,2,2]

**Output:** 2

**Example 2:**

**Input:**nums = [3,1,3,4,2]

**Output:** 3

**For example:**

| **Input** | **Result** |
| --- | --- |
| 1 3 4 4 2 | 4 |

**PROGRAM:**

s = input()

substrings = {}

repeated\_substrings = []

for i in range(len(s) - 9):

substring = s[i:i+10]

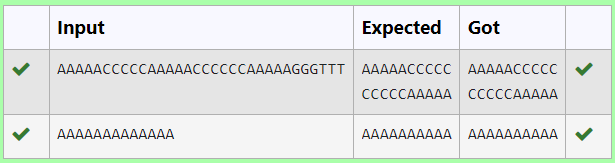
substrings[substring] = substrings.get(substring, 0) + 1

if substrings[substring] == 2:

repeated\_substrings.append(substring)

print(\*repeated\_substrings,sep="\n")

**OUTPUT:**

****

**Ex. No. : 8.4 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**Remove repeated**

Write a program to eliminate the common elements in the given 2 arrays and print only the non-repeatingelements and the total number of such non-repeating elements.

Input Format:

The first line contains space-separated values, denoting the size of the two arrays in integer format respectively.

The next two lines contain the space-separated integer arrays to be compared.

Example 1:

Input: text = "hello world", brokenLetters = "ad"

Output:

1

Explanation: We cannot type "world" because the 'd' key is broken.

**For example:**

| **Input** | **Result** |
| --- | --- |
| hello world  ad | 1 |

**PROGRAM:**

s = input()

substrings = {}

repeated\_substrings = []

for i in range(len(s) - 9):

substring = s[i:i+10]

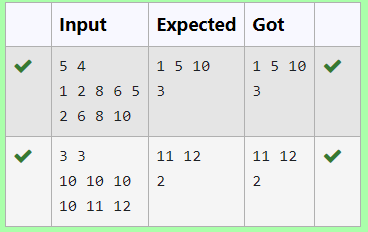
substrings[substring] = substrings.get(substring, 0) + 1

if substrings[substring] == 2:

repeated\_substrings.append(substring)

print(\*repeated\_substrings,sep="\n")

**OUTPUT:**

****

**Ex. No. : 8.5 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**Malfunctioning Keyboard**

There is a malfunctioning keyboard where some letter keys do not work. All other keys on the keyboard work properly.

Given a string text of words separated by a single space (no leading or trailing spaces) and a string brokenLetters of all distinct letter keys that are broken, return the number of words in text you can fully type using this keyboard.



**Example 1:**

**Input:** words = ["Hello","Alaska","Dad","Peace"]

**Output:** ["Alaska","Dad"]

**Example 2:**

**Input:** words = ["omk"]

**Output:** []

**Example 3:**

**Input:** words = ["adsdf","sfd"]

**Output:** ["adsdf","sfd"]

**For example:**

| **Input** | **Result** |
| --- | --- |
| 4  Hello  Alaska  Dad  Peace | Alaska  Dad |

**PROGRAM:**

def count\_words\_typed(text, brokenLetters):

count = 0

for word in text.split():

if all(letter not in brokenLetters for letter in word):

count += 1

return count

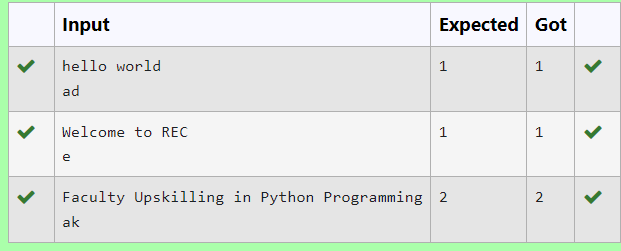
# Test the function

text = input().lower()

brokenLetters = input()

print(count\_words\_typed(text, brokenLetters)) # Output: 1

**OUTPUT:**

****

**Ex. No. : 8.6 Date:**

**Register No.: 231801024 Name: Boopesh.R**

**American keyboard**

Given an array of strings words, return *the words that can be typed using letters of the alphabet on only one row of American keyboard like the image below*.

In the **American keyboard**:

* the first row consists of the characters "qwertyuiop",
* the second row consists of the characters "asdfghjkl", and
* the third row consists of the characters "zxcvbnm".

**PROGRAM:**

rows = ['qwertyuiop', 'asdfghjkl', 'zxcvbnm']

words = []

num\_words = int(input())

for \_ in range(num\_words):

words.append(input())

found\_words = False

for word in words:

lowercase\_word = word.lower()

for row in rows:

if lowercase\_word[0] in row:

row\_set = set(row)

break

if all(char in row\_set for char in lowercase\_word):

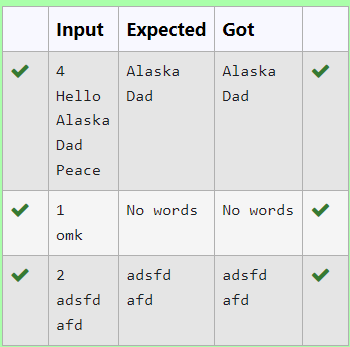
print(word)

found\_words = True

if not found\_words:

print("No words")

**OUTPUT:**

****