08 – Tuple/Set

Examples:

Input: str = "01010101010"

Output: Yes

Input: str = "REC101"

Output: No

For example:

Input	Result
01010101010	Yes
010101 10101	No

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

Register No.: **Name:**

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.

```
Str1 = input()

If set(str1).issubset({‘0’, ‘1’}):

    Print(“Yes”)

Else:

    Print(“No”)
```

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

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

Register No.: **Name:**

Check Pair

Given a tuple and a positive integer k, the task is to find the count of distinct pairs in the tuple whose sum is equal to **K**.

```
T=tuple(input().split(‘,’))

K=int(input())

D=[]

For I in t:

   For j in t:

      If int(i)+int(j)==k:

         If (I,j) not in d:

            d.append((I,j))

print(len(d)//2)
```

Example 1:

Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
Output: ["AAAAACCCCC","CCCCCAAAAA"]

Example 2:

Input: s = "AAAAAAAAAAAAA"
Output: ["AAAAAAAAAA"]

For example:

Input	Result
AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT	AAAAACCCCC CCCCCAAAAA

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

Register No.: **Name:**

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**.

```
S = input()
If len(s) < 10:
    Result = []
Else:
    Sequences = {}
    Result = []
    For I in range(len(s) – 9):
        Substring = s[i:i+10]
        If substring in sequences:
            Sequences[substring] += 1
        Else:
            Sequences[substring] = 1
    For sequence, count in sequences.items():
        If count > 1:
```

Result.append(sequence)

For I in result:

Print(i)

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

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

Register No.: **Name:**

Print repeated no

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.There is only **one repeated number** in nums, return *this repeated number*. Solve the problem using set.

```
Def find_duplicate(nums):
    Seen = set()
    For num in nums:
        If num in seen:
            Return num
        Seen.add(num)
```

Sample Input:

5 4

1 2 8 6 5

2 6 8 10

Sample Output:

1 5 10

3

Sample Input:

5 5

1 2 3 4 5

1 2 3 4 5

Sample Output:

NO SUCH ELEMENTS

For example:

Input	Result
5 4 1 2 8 6 5 2 6 8 10	1 5 10 3

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

Register No.: **Name:**

Remove repeated

Write a program to eliminate the common elements in the given 2 arrays and print only the non-repeating elements 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.

```
Sizes = input().split()

Size1 = int(sizes[0])

Size2 = int(sizes[1])

Array1 = input().split()

Array2 = input().split()

Array1 = list(map(int, array1))

Array2 = list(map(int, array2))

Set1 = set(array1)

Set2 = set(array2)
```

```
Common_elements = set1.intersection(set2)

Unique_set1 = set1 – common_elements

Unique_set2 = set2 – common_elements

Unique_elements = unique_set1.union(unique_set2)

If unique_elements:

    Unique_elements_list = sorted(list(unique_elements))

    Print(“ “.join(map(str, unique_elements_list)))

    Print(len(unique_elements_list))

Else:

    Print(“NO SUCH ELEMENTS”)
```

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

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

Register No.: **Name:**

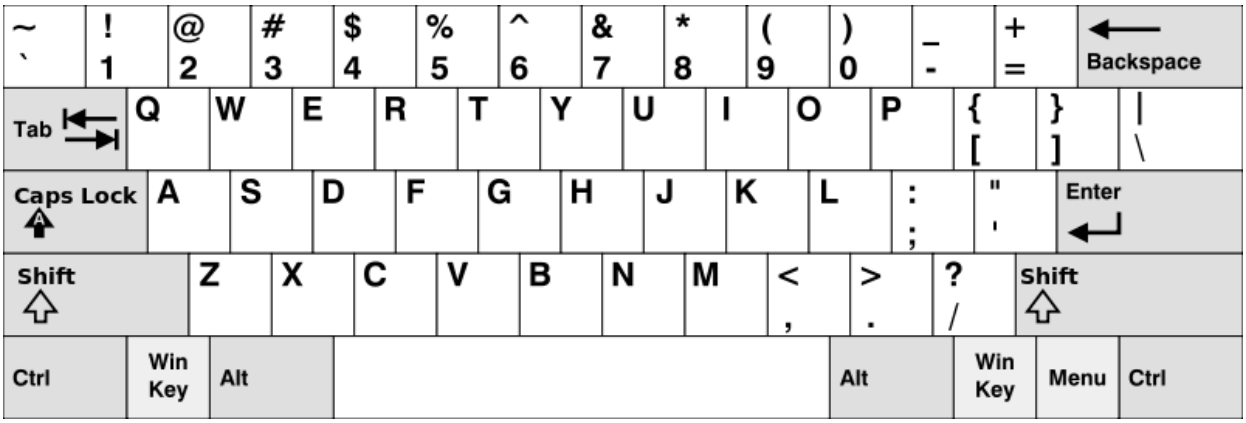
Malfunctioning Keyboard

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.

```
A=input().lower().split()

B=input()

C=0

For I in a:

    Flag=0

    For j in i:

        If j in b:

            Flag=1

            Break
```

```
If(flag==0):

    C+=1

Print©
```



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

Ex. No. : 8.7 **Date:**

Register No.: **Name:**

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".

Def find_words_in_one_row(words):

Row1 = set("qwertyuiop")

Row2 = set("asdfghjkl")

Row3 = set("zxcvbnm")

Def can_be_typed_on_one_row(word):

Lower_word = set(word.lower())

Return lower_word <= row1 or lower_word <= row2 or lower_word <= row3

Return [word for word in words if can_be_typed_on_one_row(word)]