

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.

```
s=input()
s1=set(s)
if(s1=={'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 = "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"

Output: ["AAAAACCCC", "CCCCAAAAA"]

Example 2:

Input: s = "AAAAAAAAAAAAA"

Output: ["AAAAAAAAA"]

For example:

Input	Result
AAAAACCCCCAAAAACCCCCAAAAAGGGTTT	AAAAACCCC CCCCAAAAA

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()
sub={}
r=[]
for i in range(len(s)-9):
    str=s[i:i+10]
    if str in sub:
        sub[str]+=1
    else:
        sub[str]=1
    if(sub[str]==2):
        r.append(str)
for x in r:
    print(x)
```

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](#).

```
n=input().split()
for i in n:
    if n.count(i)>=2:
        print(i)
        break
```

[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 5 10
1 2 8 6 5	3
2 6 8 10	

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.

```
s1,s2=input().split()
str1=int(s1)
str2=int(s2)
arr1=input()
arr2=input()
aar1=[int(i) for i in arr1.split()]
aar2=[int(i) for i in arr2.split()]
set1=set(aar1)
set2=set(aar2)
n=set1.symmetric_difference(set2)
if n:
    print(*n)
    print(len(n))
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.

```
s1=list(input())
s2=list(input())
b=[]
count=0
for i in range(len(s1)):
    for j in range(len(s2)):
        if s2[j] in s1[i]:
            if s2[j] not in b:
                b.append(s2[j])
for i in range(len(b)):
    count+=1
print(count)
```

~ 1	! 2	@ 3	# 4	\$ 5	% 6	^ 7	& 8	* 9	(0) -	+ =	← Backspace	
Tab ↔	Q	W	E	R	T	Y	U	I	O	P	{ [}]	 \
Caps Lock ⬆	A	S	D	F	G	H	J	K	L	:	" '	Enter ↵	
Shift ⬆	Z	X	C	V	B	N	M	< ,	> .	?	/	Shift ⬆	
Ctrl	Win Key	Alt								Alt	Win Key	Menu	Ctrl

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".

```
lst1=[]
for i in range (int(input())):
    lst1.append(input())
lst2="qwertyuiop"
lst3="asdfghjkl"
lst4="zxcvbnm"
l=0
m=0
n=0
lst5=[]
for i in lst1:
    x=i
    i=i.lower()
    l=0
    m=0
    n=0
    j=i
    b=len(j)
    for k in range(0,b):
        if(i[k] not in lst3 and i[k] not in lst4):
            l+=1
        elif(i[k] not in lst2 and i[k] not in lst4):
            m+=1
        elif(i[k] not in lst2 and i[k] not in lst3):
            n+=1
    if(l==b or m==b or n==b):
        lst5.append(x)
```

p=0

Department of Computer Science and Engineering | Rajalakshmi Engineering College

p+=1

if i!=0.

```
    print(i)
if(p==0):
    print("No words")
```