08 – Tuple/Set

Examples:

Input: str = "01010101010"

Output: Yes

Input: str = "REC101"

Output: No

For example:

Input	Result
01010101010	Yes
010101 10101	No

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'}0:
    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

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**.

```python
def find_pairs_with_sum(numbers, target_sum):
    # Convert the tuple to a list to enable mutability
    numbers_list = list(numbers)

    # Create a set to store the pairs
    pairs = set()

    # Create a set to store the visited numbers
    visited = set()

    for number in numbers_list:
        complement = target_sum - number
        if complement in visited:
            # To ensure pairs are stored in a consistent order (e.g., (small, large))
            pair = tuple(sorted((number, complement)))
            pairs.add(pair)
        visited.add(number)

    return pairs

# Taking input from the user
numbers_input = input("")
target_sum = int(input(""))

# Convert the input string to a tuple of integers
numbers = tuple(map(int, numbers_input.split(',')))

# Find pairs
pairs = find_pairs_with_sum(numbers, target_sum)

print(f"{len(pairs)}")
```

Example 1:

Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
Output: ["AAAAACCCCC","CCCCCAAAAA"]
Example 2:

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

For example:

Input	Result
AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT	AAAAACCCCC CCCCCAAAAA

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**.

```python
def findRepeatedDnaSequences(s):
    if len(s) < 10:
        return []

    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)

s1=input()

findRepeatedDnaSequences(s1)
```

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

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.

```python
a=input()
a=a.split(" ")

for i in range(len(a)):
    for j in range(len(a)):
        if(a[i]==a[j] and i!=j):
            c=a[i]
            break
print(c)
```

Input:

5 4

1 2 8 6 5

2 6 8 10

Output:

1 5 10

3

Input:

5 5

1 2 3 4 5

1 2 3 4 5

Output:

NO SUCH ELEMENTS

For example:

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

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.

```
a=[int(i) for i in input().split()]
n1=[int(i) for i in input().split()]
n2=[int(i) for i in input().split()]
com=set(n1)&set(n2)
uni=set(n1)|set(n2)
tot=sorted(uni-com)
for i in tot:
    print(i,end=' ')
print('\n',end='')
print(len(tot))
```

Example 1:

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

Output:

1

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

For example:

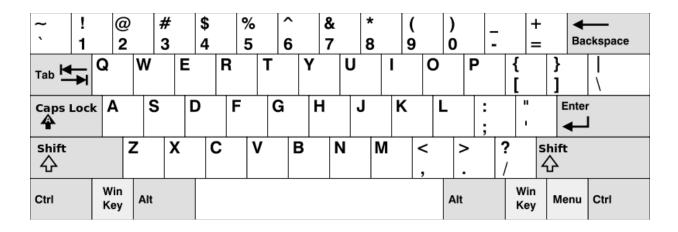| Input	Result
hello world ad	1

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()
b=list(input())

c=0
for i in range(len(b)):
    if b[i] in a:
        c+=1
print(c)
```

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

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".

```
n=int(input())
f=0
a=[input() for i in range(n)]
l1=['qwertyuiop','asdfghjkl','zxcvbnm']
l=[[j for j in i] for i in l1]
for i in a:
    n=[j for j in i.lower()]
    #print(sorted(set(l[1])|set(n))==sorted(set(l[1])))
    #print(set(l[1]),set(n))
    if set(n)|set(l[0])==set(l[0]):
        f=1
        print(i)
        continue
    elif set(n)|set(l[1])==set(l[1]):
        f=1
        print(i)
        continue
    elif set(n)|set(l[2])==set(l[2]):
        f=1
        print(i)
        continue
if not f:
    print('No words')
```