**07 – Tuple/Set**

**Ex. No. : 7.1 Date: 27.05.24**

**Register No.: 230701347 Name: VERAN. V**

**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: str = "01010101010"

Output: Yes

Input: str = "REC101"

Output: No

**For example:**

| **Input** | **Result** |
| --- | --- |
| 01010101010 | Yes |
| 010101 10101 | No |

PROGRAM:

str1=set(input())

if not(str1-{'0','1'}):

print("Yes")

else:

print("No")

|  | **Input** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | 01010101010 | Yes | Yes |  |
|  | REC123 | No | No |  |
|  | 010101 10101 | No | No |  |

**Ex. No. : 7.2 Date: 27.05.24**

**Register No.: 230701347 Name: VERAN. V**

**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**.

**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 find\_pairs\_with\_sum(numbers, target\_sum):

numbers\_list = list(numbers)

pairs = set()

visited = set()

for number in numbers\_list:

complement = target\_sum - number

if complement in visited:

pair = tuple(sorted((number, complement)))

pairs.add(pair)

visited.add(number)

return pairs

numbers\_input = input("")

target\_sum = int(input(""))

numbers = tuple(map(int, numbers\_input.split(',')))

pairs = find\_pairs\_with\_sum(numbers, target\_sum)

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

| **Input** | **Expected** | **Got** |  |
| --- | --- | --- | --- |
|  | 5,6,5,7,7,8  13 | 2 | 2 |  |
|  | 1,2,1,2,5  3 | 1 | 1 |  |
|  | 1,2  0 | 0 | 0 |  |

**Ex. No. : 7.3 Date: 27.05.24**

**Register No.: 230701347 Name: VERAN. V**

**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:** s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

**Output:** ["AAAAACCCCC","CCCCCAAAAA"]

**Example 2:**

**Input:** s = "AAAAAAAAAAAAA"

**Output:** ["AAAAAAAAAA"]

**For example:**

| **Input** | **Result** |
| --- | --- |
| AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT | AAAAACCCCC  CCCCCAAAAA |

 PROGRAM:

a=input()

b=[]

for i in range(0,len(a),10):

b.append(a[i:i+10])

print(b[0])

for i in range(len(b)-1):

if(b[i]==b[i+1]):

print(b[i+1][::-1])

|  | **Input** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT | AAAAACCCCC  CCCCCAAAAA | AAAAACCCCC  CCCCCAAAAA |  |
|  | AAAAAAAAAAAAA | AAAAAAAAAA | AAAAAAAAAA |  |

**Ex. No. : 7.4 Date: 27.05.24**

**Register No.: 230701347 Name: VERAN. V**

**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](http://118.185.187.137/moodle/mod/resource/view.php?id=734).

**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:**

a=list(input().split(" "))

a=[int(x) for x in a]

for i in a:

if a.count(i)>1:

print(i)

break

|  | **Input** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | 1 3 4 4 2 | 4 | 4 |  |
|  | 1 2 2 3 4 5 6 7 | 2 | 2 |  |

**Ex. No. : 7.5 Date: 27.05.24**

**Register No.: 230701347 Name: VERAN. V**

**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: 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:

str1=input().split()

str2=input()

count=0

for word in str1:

word=word.lower()

present=0

for i in str2:

if i in word:

present=1

break

if(present==0):

count+=1

print(count)

|  | **Input** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | hello world  ad | 1 | 1 |  |
|  | Welcome to REC  e | 1 | 1 |  |
|  | Faculty Upskilling in Python Programming  ak | 2 | 2 |  |