

**Started on** Tuesday, 23 September 2025, 10:45 PM

**State** Finished

**Completed on** Thursday, 25 September 2025, 7:20 PM

**Time taken** 1 day 20 hours

**Marks** 10.00/10.00

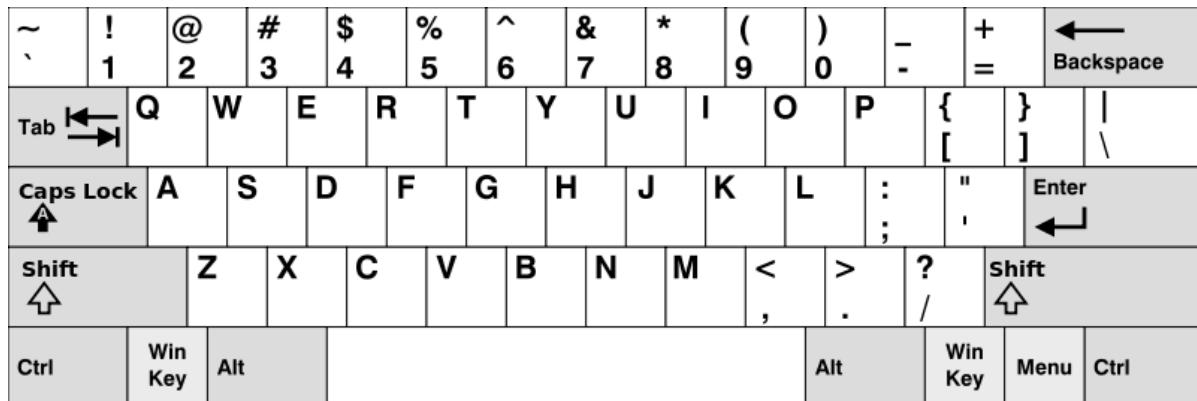
**Grade** **100.00** out of 100.00

**Question 1** | Correct Mark 1.00 out of 1.00

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".

**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      | Alaska |
| Hello  | Dad    |
| Alaska |        |
| Dad    |        |
| Peace  |        |
| 2      | adsfd  |
| adsfd  | afd    |
| afd    |        |

**Answer:** (penalty regime: 0 %)

```

1 n=int(input())
2 words=[input() for _ in range(n)]
3 row1=set("qwertyuiop")

```

```
4 row2=set("asdfghjkl")
5 row3=set("zxcvbnm")
6 result=[]
7 for i in words:
8     lower_word=set(i.lower())
9     if lower_word<=row1 or lower_word<=row2 or lower_word<=row3:
10         result.append(i)
11 if not result:
12     print("No words")
13 else:
14     print(*result,sep='\n')
```

|   | Input                                | Expected      | Got           |   |
|---|--------------------------------------|---------------|---------------|---|
| ✓ | 4<br>Hello<br>Alaska<br>Dad<br>Peace | Alaska<br>Dad | Alaska<br>Dad | ✓ |
| ✓ | 1<br>omk                             | No words      | No words      | ✓ |
| ✓ | 2<br>adsfd<br>afd                    | adsfd<br>afd  | adsfd<br>afd  | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 2** | Correct Mark 1.00 out of 1.00

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 = "AAAAACCCCCAAAAACCCCCAAAAAGGGTTT"
```

```
Output: ["AAAAACCCCC", "CCCCAAAAAA"]
```

**Example 2:**

```
Input: s = "AAAAAAAAAAAAAA"
```

```
Output: ["AAAAAAAAAA"]
```

**For example:**

| Input                           | Result                   |
|---------------------------------|--------------------------|
| AAAAACCCCCAAAAACCCCCAAAAAGGGTTT | AAAAACCCCC<br>CCCCAAAAAA |

**Answer:** (penalty regime: 0 %)

```

1 s=input()
2 n=10
3 result=[]
4 seen=set()
5 for i in range(len(s)-n+1):
6     substring=s[i:i+n]
7     if substring in seen:
8         if substring not in result:
9             result.append(substring)
10    else:
11        seen.add(substring)
12 print(*result,sep='\n')
```

|   | Input                           | Expected                 | Got                      |   |
|---|---------------------------------|--------------------------|--------------------------|---|
| ✓ | AAAAACCCCCAAAAACCCCCAAAAAGGGTTT | AAAAACCCCC<br>CCCCAAAAAA | AAAAACCCCC<br>CCCCAAAAAA | ✓ |
| ✓ | AAAAAAAAAAAAAA                  | AAAAAAAAAA               | AAAAAAAAAA               | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 3** | Correct Mark 1.00 out of 1.00

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.

**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      |

**Answer:** (penalty regime: 0 %)

```

1 num=input()
2 nums=[int(x) for x in num.split()]
3 seen=set()
4 for n in nums:
5     if n in seen:
6         print(n)
7         break
8     seen.add(n)

```

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

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 4** | Correct Mark 1.00 out of 1.00

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<br>ad                              | 1      |
| Faculty Upskilling in Python Programming<br>ak | 2      |

**Answer:** (penalty regime: 0 %)

```

1 t=input()
2 broken=input()
3 broken_set=set(broken.lower())
4 words=t.split()
5 typec=0
6 for word in words:
7     can_type=True
8     for char in word.lower():
9         if char in broken_set:
10             can_type=False
11             break
12     if can_type:
13         typec+=1
14 print(typec)
15

```

|   | Input               | Expected | Got |   |
|---|---------------------|----------|-----|---|
| ✓ | hello world<br>ad   | 1        | 1   | ✓ |
| ✓ | Welcome to REC<br>e | 1        | 1   | ✓ |

|   | Input  | Expected | Got |   |
|---|--|----------|-----|---|
| ✓ | Faculty Upskilling in Python Programming<br>ak | 2        | 2   | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 5** | Correct Mark 1.00 out of 1.00

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     |

**Answer:** (penalty regime: 0 %)

```

1 s=input()
2 chars=set(s)
3 if chars.issubset({'0','1'}):
4     print("Yes")
5 else:
6     print("No")

```

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

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.



**Question 6** | Correct Mark 1.00 out of 1.00

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 | 1      |
| 3         |        |
| 1,2       | 0      |
| 0         |        |

**Answer:** (penalty regime: 0 %)

```

1 s=input()
2 t=tuple(int(x) for x in s.split(','))
3 k=int(input())
4 seen=set()
5 dist=set()
6 for num in t:
7     comp=k-num
8     if comp in seen:
9         pair=tuple(sorted((num,comp)))
10        dist.add(pair)
11    seen.add(num)
12 print(len(dist))

```

|   | Input             | Expected | Got |   |
|---|-------------------|----------|-----|---|
| ✓ | 5,6,5,7,7,8<br>13 | 2        | 2   | ✓ |
| ✓ | 1,2,1,2,5<br>3    | 1        | 1   | ✓ |

|   | Input     | Expected | Got |   |
|---|-----------|----------|-----|---|
| ✓ | 1, 2<br>0 | 0        | 0   | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 7** | Correct Mark 1.00 out of 1.00

Program to print all the distinct elements in an array. Distinct elements are nothing but the unique (non-duplicate) elements present in the given array.

**Input Format:**

First line take an Integer input from stdin which is array length n.

Second line take n Integers which is inputs of array.

**Output Format:**

Print the Distinct Elements in Array in single line which is space Separated

**Example Input:**

5

1 2 2 3 4

**Output:**

1 2 3 4

**Example Input:**

6

1 1 2 2 3 3

**Output:**

1 2 3

**For example:**

| Input | Result  |
|-------|---------|
| 5     | 1 2 3 4 |
| 1     |         |
| 2     |         |
| 2     |         |
| 3     |         |
| 4     |         |

**Answer:** (penalty regime: 0 %)

```

1 n=int(input())
2 s=set()
3 for i in range(n):
4     k=int(input())
5     s.add(k)
6 print(*s)

```

|   | Input  | Expected  | Got       |   |
|---|--|-----------|-----------|---|
| ✓ | 5<br>1<br>2<br>2<br>3<br>4                           | 1 2 3 4   | 1 2 3 4   | ✓ |
| ✓ | 6<br>1<br>1<br>2<br>2<br>3<br>3                      | 1 2 3     | 1 2 3     | ✓ |
| ✓ | 5<br>11<br>22<br>11<br>22<br>11                      | 11 22     | 11 22     | ✓ |
| ✓ | 10<br>1<br>2<br>3<br>4<br>5<br>1<br>2<br>3<br>4<br>5 | 1 2 3 4 5 | 1 2 3 4 5 | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 8** | Correct Mark 1.00 out of 1.00

## Check if a set is a subset of another set.

Example:

Sample Input1:

mango apple

mango orange

mango

output1:

yes

set3 is subset of set1 and set2

input2:

mango orange

banana orange

grapes

output2:

no

For example:

| Test | Input                                   | Result                                 |
|------|---|--|
| 1    | mango apple<br>mango orange<br>mango    | yes<br>set3 is subset of set1 and set2 |
| 2    | mango orange<br>banana orange<br>grapes | No                                     |

Answer: (penalty regime: 0 %)

```

1 set1=set(input())
2 set2=set(input())
3 set3=set(input())
4 if set3.issubset(set1) and set3.issubset(set2):
5     print("yes")
6     print("set3 is subset of set1 and set2")
7 else:
8     print("No")

```

|   | Test | Input                                   | Expected                               | Got                                    |   |
|---|------|---|--|--|---|
| ✓ | 1    | mango apple<br>mango orange<br>mango    | yes<br>set3 is subset of set1 and set2 | yes<br>set3 is subset of set1 and set2 | ✓ |
| ✓ | 2    | mango orange<br>banana orange<br>grapes | No                                     | No                                     | ✓ |

Passed all tests! ✓ //

Correct

Marks for this submission: 1.00/1.00.

**Question 9** | Correct Mark 1.00 out of 1.00

You are given an integer tuple `nums` containing distinct numbers. Your task is to perform a sequence of operations on this tuple until it becomes empty. The operations are defined as follows:

1. If the first element of the tuple has the smallest value in the entire tuple, remove it.
2. Otherwise, move the first element to the end of the tuple.

You need to return an integer denoting the number of operations required to make the tuple empty.

## Constraints

- The input tuple `nums` contains distinct integers.
- The operations must be performed using tuples and sets to maintain immutability and efficiency.
- Your function should accept the tuple `nums` as input and return the total number of operations as an integer.

Example:

Input: `nums = (3, 4, -1)`

Output: 5

Explanation:

Operation 1: `[3, 4, -1]` -> First element is not the smallest, move to the end -> `[4, -1, 3]`

Operation 2: `[4, -1, 3]` -> First element is not the smallest, move to the end -> `[-1, 3, 4]`

Operation 3: `[-1, 3, 4]` -> First element is the smallest, remove it -> `[3, 4]`

Operation 4: `[3, 4]` -> First element is the smallest, remove it -> `[4]`

Operation 5: `[4]` -> First element is the smallest, remove it -> `[]`

Total operations: 5

For example:

| Test   | Result |
|--|--------|
| <code>print(count_operations((3, 4, -1)))</code> | 5      |

Answer: (penalty regime: 0 %)

Reset answer

```

1 | def count_operations(nums: tuple) -> int:
2 |     nums=list(nums)
3 |     count=0
4 |     while nums:
5 |         if nums[0]==min(nums):
6 |             nums.pop(0)
7 |         else:
8 |             nums.append(nums.pop(0))
9 |         count+=1
10 |    return count
11 |

```

|   | Test                                       | Expected | Got |   |
|---|--|----------|-----|---|
| ✓ | print(count_operations((3, 4, -1)))        | 5        | 5   | ✓ |
| ✓ | print(count_operations((1, 2, 3, 4, 5)))   | 5        | 5   | ✓ |
| ✓ | print(count_operations((5, 4, 3, 2, 1)))   | 15       | 15  | ✓ |
| ✓ | print(count_operations((42, )))            | 1        | 1   | ✓ |
| ✓ | print(count_operations((-2, 3, -5, 4, 1))) | 11       | 11  | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

**Question 10** | Correct Mark 1.00 out of 1.00

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.

**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  |                  |
| 5 5       | NO SUCH ELEMENTS |
| 1 2 3 4 5 |                  |
| 1 2 3 4 5 |                  |

**Answer:** (penalty regime: 0 %)

```
1 n=input()
2 n1,n2=map(int,n.split())
3 arr1n=input()
4 arr1=set(map(int,arr1n.split()))
5 arr2n=input()
6 arr2=set(map(int,arr2n.split()))
7 common_element=arr1.intersection(arr2)
8 nons=sorted(list(arr1.symmetric_difference(arr2)))
9 if nons:
10     print(*nons)
11     print(len(nons))
12 else:
13     print("NO SUCH ELEMENTS")
```

|   | Input                         | Expected         | Got              |   |
|---|-------------------------------|------------------|------------------|---|
| ✓ | 5 4<br>1 2 8 6 5<br>2 6 8 10  | 1 5 10<br>3      | 1 5 10<br>3      | ✓ |
| ✓ | 3 3<br>10 10 10<br>10 11 12   | 11 12<br>2       | 11 12<br>2       | ✓ |
| ✓ | 5 5<br>1 2 3 4 5<br>1 2 3 4 5 | NO SUCH ELEMENTS | NO SUCH ELEMENTS | ✓ |

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.