

# Rajalakshmi Engineering College

Name: madhavan R  
Email: 240701294@rajalakshmi.edu.in  
Roll no: 240701294  
Phone: 6381021871  
Branch: REC  
Department: I CSE AH  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 3\_PAH

Attempt : 1  
Total Mark : 60  
Marks Obtained : 60

### Section 1 : Coding

#### 1. Problem Statement

Kyara is analyzing a series of measurements taken over time. She needs to identify all the "peaks" in this list of integers.

A peak is defined as an element that is greater than its immediate neighbors. Boundary elements are considered peaks if they are greater than their single neighbor.

Your task is to find and list all such peaks using list comprehension.

Example

Input

1 3 2 4 1 5 7 6 10 2 8

## Output

Peaks: [3, 4, 7, 10, 8]

## Explanation

3 is a peak because it's greater than 1 and 2.

4 is a peak because it's greater than 2 and 1.

7 is a peak because it's greater than 5 and 6.

10 is a peak because it's greater than 6 and 2.

8 is a peak because it is an boundary element and it is greater than 2.

## Input Format

The input consists of several integers separated by spaces, representing the measurements.

## Output Format

The output displays "Peaks: " followed by a list of integers, representing the peak elements in the list.

Refer to the sample output for the formatting specifications.

## Sample Test Case

Input: 1 3 2 4 1 5 7 6 10 2 8

Output: Peaks: [3, 4, 7, 10, 8]

## Answer

```
def find_peaks(measurements):
```

```
    """
```

Finds all the peaks in a list of integers using list comprehension.

Args:

measurements: A list of integers representing the measurements.

Returns:

A list of integers representing the peak elements.

```

"""
peaks = [
    measurements[i]
    for i in range(len(measurements))
    if (i == 0 and len(measurements) > 1 and measurements[i] >
measurements[i + 1]) or
        (i == len(measurements) - 1 and len(measurements) > 1 and
measurements[i] > measurements[i - 1]) or
        (1 <= i < len(measurements) - 1 and measurements[i] > measurements[i -
1] and measurements[i] > measurements[i + 1]) or
        (len(measurements) == 1)
]
return peaks

if __name__ == "__main__":
    input_str = input()
    measurements = [int(x) for x in input_str.split()]
    peak_elements = find_peaks(measurements)
    print(f"Peaks: {peak_elements}")

```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Neha is learning string operations in Python and wants to practice using built-in functions. She is given a string A, and her task is to:

Find the length of the string using a built-in function. Copy the content of A into another string B using built-in functionality.

Help Neha implement a program that efficiently performs these operations.

### **Input Format**

The input consists of a single line containing the string A (without spaces).

### **Output Format**

The first line of output prints the length of the given string.

The second line prints the copied string without an extra newline at the end.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: technology-23

Output: Length of the string: 13

Copied string: technology-23

### **Answer**

```
def string_operations(string_a):
```

```
    """
    Finds the length of a string and copies it to another string using built-in
    functions.
```

```
    Args:
```

```
        string_a: The input string.
```

```
    Returns:
```

```
        A tuple containing the length of the string and the copied string.
```

```
    """
```

```
    length = len(string_a)
```

```
    string_b = string_a[:]
```

```
    return length, string_b
```

```
if __name__ == "__main__":
```

```
    string_a = input()
```

```
    length_a, string_b = string_operations(string_a)
```

```
    print(f"Length of the string: {length_a}")
```

```
    print(f"Copied string: {string_b}")
```

**Status :** Correct

**Marks :** 10/10

### **3. Problem Statement**

Imagine you are developing a text analysis tool for a cybersecurity company. Your task is to analyze input strings to categorize and count the characters into four categories: uppercase letters, lowercase letters, digits, and special characters. The company needs this tool to process log files

and identify potential security threats.

### ***Input Format***

The input consists of the log entry provided as a single string.

### ***Output Format***

The output consists of four lines:

The first line contains an integer representing the count of uppercase letters in the format "Uppercase letters: {uppercase count}".

The second line contains an integer representing the count of lowercase letters in the format "Lowercase letters: {lowercase count}".

The third line contains an integer representing the count of digits in the format "Digits: {digits count}".

The fourth line contains an integer representing the count of special characters in the format "Special characters: {special characters count}".

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: Hello123

Output: Uppercase letters: 1

Lowercase letters: 4

Digits: 3

Special characters: 0

### ***Answer***

```
def analyze_string(log_entry):
```

```
    uppercase_count = 0
```

```
    lowercase_count = 0
```

```
    digits_count = 0
```

```
    special_count = 0
```

```
    for char in log_entry:
```

```
        if 'A' <= char <= 'Z':
```

```

        uppercase_count += 1
    elif 'a' <= char <= 'z':
        lowercase_count += 1
    elif '0' <= char <= '9':
        digits_count += 1
    else:
        special_count += 1
return uppercase_count, lowercase_count, digits_count, special_count

if __name__ == "__main__":
    log_entry = input()
    uppercase, lowercase, digits, special = analyze_string(log_entry)
    print(f"Uppercase letters: {uppercase}")
    print(f"Lowercase letters: {lowercase}")
    print(f"Digits: {digits}")
    print(f"Special characters: {special}")

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

You are tasked with writing a program that takes  $n$  integers as input from the user and stores them in a list. After this, you need to transform the list according to the following rules:

The element at index 0 should be replaced with 0. For elements at even indices (excluding index 0), replace the element with its cube. For elements at odd indices, replace the element with its square.

Additionally, you should sort the list in ascending order before applying these transformations.

##### **Input Format**

The first line of input represents the size of the list,  $N$ .

The elements of the list are represented by the next  $N$  lines.

##### **Output Format**

The first line of output displays "Original List: " followed by the original list.

The second line displays "Replaced List: " followed by the replacement list as per the given condition.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 5

5

1

2

3

4

Output: Original List: [1, 2, 3, 4, 5]

Replaced List: [0, 4, 27, 16, 125]

### **Answer**

```
def transform_list(n, elements):
```

```
    """
```

Takes n integers as input, stores them in a list, sorts the list, and then transforms it according to specific rules.

Args:

n: The number of integers to input.

elements: A list of n integers.

Returns:

A tuple containing the sorted original list and the transformed list.

```
    """
```

```
    original_list = sorted(elements)
```

```
    replaced_list = [0] * n
```

```
    replaced_list[0] = 0
```

```
    for i in range(1, n):
```

```
        if i % 2 == 0:
```

```
            replaced_list[i] = original_list[i] ** 3
```

```
        else:
```

```
            replaced_list[i] = original_list[i] ** 2
```

```
    return original_list, replaced_list
```

```
if __name__ == "__main__":
```

```
n = int(input())
elements = []
for _ in range(n):
    element = int(input())
    elements.append(element)

original, replaced = transform_list(n, elements)
print(f"Original List: {original}")
print(f"Replaced List: {replaced}")
```

**Status :** Correct

**Marks :** 10/10

## 5. Problem Statement

Gowri was doing her homework. She needed to write a paragraph about modern history. During that time, she noticed that some words were repeated repeatedly. She started counting the number of times a particular word was repeated.

Your task is to help Gowri to write a program to get a string from the user. Count the number of times a word is repeated in the string.

Note: Case-sensitive

### **Input Format**

The first line of input consists of a string, str1.

The second line consists of a single word that needs to be counted, str2.

### **Output Format**

The output displays the number of times the given word is in the string.

If the second string str2 is not present in the first string str1, it prints 0.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**



Input: I felt happy because I saw the others were happy and because I knew I should feel happy  
happy

Output: 3

**Answer**

```
def count_word_occurrence(str1, str2):  
    count = 0  
    index = 0  
    while index < len(str1):  
        found_index = str1.find(str2, index)  
        if found_index == -1:  
            break  
        count += 1  
        index = found_index + 1  
    return count  
  
if __name__ == "__main__":  
    str1 = input()  
    str2 = input()  
    occurrence = count_word_occurrence(str1, str2)  
    print(occurrence)
```

**Status :** Correct

**Marks : 10/10**

**6. Problem Statement**

Accept an unsorted list of length  $n$  with both positive and negative integers, including 0. The task is to find the smallest positive number missing from the array. Assume the  $n$  value is always greater than zero.

**Input Format**

The first line consists of  $n$ , which means the number of elements in the array.

The second line consists of the values in the list as space-separated integers.

**Output Format**

The output displays the smallest positive number, which is missing from the array.

Refer to the sample output for the formatting specifications.

**Sample Test Case**

Input: 6

-5 2 0 -1 -10 2

Output: 1

**Answer**

# You are using Python

```
def find_smallest_missing_positive(arr):
```

```
    n = len(arr)
```

```
    # Step 1: Filter the array to contain only positive numbers and remove
    # duplicates
```

```
    positive_set = set(x for x in arr if x > 0)
```

```
    # Step 2: Start checking from 1 upwards to find the missing number
```

```
    smallest_missing = 1
```

```
    while smallest_missing in positive_set:
```

```
        smallest_missing += 1
```

```
    return smallest_missing
```

```
# Input handling
```

```
n = int(input())
```

```
arr = list(map(int, input().split()))
```

```
# Output the result
```

```
print(find_smallest_missing_positive(arr))
```

**Status :** Correct

**Marks :** 10/10