Rajalakshmi Engineering College

Name: Prasham Jaganathan
Email: 241501148@rajalakshmi.edu.in
Roll no: 241501148
Phone: 9445840008
Branch: REC
Department: I AI & ML FB
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 3_COD

Attempt : 1
Total Mark : 50
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

You have a string containing a phone number in the format "(XXX) XXX-XXXX". You need to extract the area code from the phone number and create a new string that contains only the area code.

Write a Python program for the same.

Note

(XXX) - Area code

XXX-XXXX - Phone number

Input Format

The input consists of a string, representing the phone number in the format

"(XXX) XXX-XXXX".

Output Format

The output displays "Area code: " followed by a string representing the area code for the given phone number.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: (123) 456-7890
Output: Area code: 123

Answer

```
def extract_area_code(phone_number):

    area_code = phone_number[1:4]

    print(f"Area code: {area_code}")


phone_number = input()


extract_area_code(phone_number)
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

Dhruv wants to write a program to slice a given string based on user-defined start and end positions.

The program should check whether the provided positions are valid and then return the sliced portion of the string if the positions are within the string's length.

Input Format

The first line consists of the input string as a string.

The second line consists of the start position (0-based index) as an integer.

The third line consists of the end position (0-based index) as an integer.

Output Format

The output displays the following format:

If the start and end positions are valid, print the sliced string.

If the start and end positions are invalid, print "Invalid start and end positions".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: pythonprogramming
0
5
Output: python

Answer

```
def slice_string(input_string, start, end):
    if 0 <= start <= end < len(input_string):
        print(input_string[start:end+1])
    else:
        print("Invalid start and end positions")

input_string = input()

start = int(input())
end = int(input())
```

slice_string(input_string, start, end)

Status : Correct ***Marks : 10/10***

3. Problem Statement

Alex is working on a Python program to manage a list of elements. He needs to append multiple elements to the list and then remove an element from the list at a specified index.

Your task is to create a program that helps Alex manage the list. The program should allow Alex to input a list of elements, append them to the existing list, and then remove an element at a specified index.

Input Format

The first line contains an integer n, representing the number of elements to be appended to the list.

The next n lines contain integers, representing the elements to be appended to the list.

The third line of input consists of an integer M, representing the index of the element to be popped from the list.

Output Format

The first line of output displays the original list.

The second line of output displays the list after popping the element of the index M.

The third line of output displays the popped element.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5

64
98
-1
5
26
3

Output: List after appending elements: [64, 98, -1, 5, 26]
List after popping last element: [64, 98, -1, 26]
Popped element: 5

Answer

```
# You are using Python
def manage_list():
    # Step 1: Read the number of elements to append
    n = int(input())

    # Step 2: Create an empty list to store the elements
    elements = []

    # Step 3: Append elements to the list
    for _ in range(n):
        elements.append(int(input()))

    # Step 4: Read the index M where the element will be popped
    M = int(input())

    # Step 5: Display the original list
    print('List after appending elements:',elements)

    # Step 6: Pop the element at index M
    popped_element = elements.pop(M)

    # Step 7: Display the modified list after popping the element
    print('List after popping last element:',elements)

    # Step 8: Display the popped element
    print('Popped element:',popped_element)

# Call the function to execute
manage_list()
```

Status : Correct ***Marks : 10/10***

4. Problem Statement

Given a list of positive and negative numbers, arrange them such that all negative integers appear before all the positive integers in the array. The order of appearance should be maintained.

Example

Input:

[12, 11, -13, -5, 6, -7, 5, -3, -6]

Output:

List = [-13, -5, -7, -3, -6, 12, 11, 6, 5]

Explanation:

The output is the arranged list where all the negative integers appear before the positive integers while maintaining the original order of appearance.

Input Format

The input consists of a single line containing a list of integers enclosed in square brackets separated by commas.

Output Format

The output displays "List = " followed by an arranged list of integers as required, separated by commas and enclosed in square brackets.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: [12, 11, -13, -5, 6, -7, 5, -3, -6]
Output: List = [-13, -5, -7, -3, -6, 12, 11, 6, 5]

Answer

```
def rearrange_numbers(input_list):
```

```
    negative_numbers = [num for num in input_list if num < 0]
    positive_numbers = [num for num in input_list if num >= 0]

    arranged_list = negative_numbers + positive_numbers

    output = "List = [" + ", ".join(map(str, arranged_list)) + "]"
    return output

input_str = "[12, 11, -13, -5, 6, -7, 5, -3, -6]"

input_list = eval(input_str)

result = rearrange_numbers(input_list)
print(result)
```

Status : Wrong ***Marks : 0/10***

5. Problem Statement

Ram is working on a program to manipulate strings. He wants to create a program that takes two strings as input, reverses the second string, and then concatenates it with the first string.

Ram needs your help to design a program.

Input Format

The input consists of two strings in separate lines.

Output Format

The output displays a single line containing the concatenated string of the first string and the reversed second string.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: hello
word

Output: hellodrow

Answer

```
def concatenate_reversed_strings():

    first_string = input().strip()

    second_string = input().strip()


    reversed_second_string = ""
    for char in second_string:
        reversed_second_string = char + reversed_second_string


    result = first_string + reversed_second_string


    print(result)

concatenate_reversed_strings()
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Prasham Jaganathan
Email: 241501148@rajalakshmi.edu.in
Roll no: 241501148
Phone: 9445840008
Branch: REC
Department: I AI & ML FB
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 3_PAH

Attempt : 1
Total Mark : 60
Marks Obtained : 45

Section 1 : Coding

1. 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():

    A = input().strip()


    length_of_A = len(A)


    B = A


    print(f"Length of the string: {length_of_A}")
    print(f"Copied string: {B}")


string_operations()
```

Status : Correct ***Marks : 10/10***

2. 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 = int(input().strip())


    original_list = []
    for _ in range(N):
```

```
    original_list.append(int(input().strip()))

    print(f"Original List: {original_list}")

    original_list.sort()

    replaced_list = []

    for index in range(N):
        if index == 0:
            replaced_list.append(0)
        elif index % 2 == 0:
            replaced_list.append(original_list[index] ** 3)
        else:
            replaced_list.append(original_list[index] ** 3)

    print(f"Replaced List: {replaced_list}")

transform_list()
```

Status : Wrong **Marks : 0/10**

3. 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 = list(map(int, input().strip().split()))
```

```
    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
            (0 < i < len(measurements) - 1 and measurements[i] > measurements[i - 1]
and measurements[i] > measurements[i + 1])
        ]


    print(f"Peaks: {peaks}")


find_peaks()
```

Status : Correct **Marks : 10/10**

4. 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_occurrences():
    # Read the input string
    str1 = input().strip()
    # Read the word to be counted
    str2 = input().strip()

    # Split the string into words
    # We can use str1.split() to split by whitespace
    words = str1.split()

    # Count the occurrences of str2 in the list of words
    count = words.count(str2)

    # Print the result
    print(count)

# Example usage
count_word_occurrences()
```

Status : Partially correct ***Marks : 5/10***

5. 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_log_entry():
    # Read the input log entry
    log_entry = input().strip()

    # Initialize counters
    uppercase_count = 0
    lowercase_count = 0
    digits_count = 0
```

```
    special_count = 0

    # Iterate through each character in the log entry
    for char in log_entry:
        if char.isupper():
            uppercase_count += 1
        elif char.islower():
            lowercase_count += 1
        elif char.isdigit():
            digits_count += 1
        else:
            special_count += 1  # Any character that is not upper, lower, or digit is
special

    # Print the results
    print(f"Uppercase letters: {uppercase_count}")
    print(f"Lowercase letters: {lowercase_count}")
    print(f"Digits: {digits_count}")
    print(f"Special characters: {special_count}")

# Example usage
analyze_log_entry()
```

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():
    # Read the number of elements
    n = int(input().strip())
    # Read the list of integers
    arr = list(map(int, input().strip().split()))

    # Create a set of positive integers from the array
    positive_numbers = set(x for x in arr if x > 0)

    # Initialize the smallest missing positive number
    smallest_missing = 1

    # Check for the smallest missing positive number
    while smallest_missing in positive_numbers:
        smallest_missing += 1

    # Print the result
    print(smallest_missing)


find_smallest_missing_positive()
```

Status : Correct ***Marks : 10/10***