# Rajalakshmi Engineering College

Name: n.madhu narayanan
Email: 240701295@rajalakshmi.edu.in
Roll no: 240701295
Phone: 8870065218
Branch: REC
Department: I CSE AH
Batch: 2028
Degree: B.E - CSE

Scan to verify results

## NeoColab_REC_CS23221_Python Programming

## REC_Python_Week 5_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 37.5

## Section 1 : Coding

1. Problem Statement

Alex is tasked with managing the membership lists of several exclusive clubs. Each club has its own list of members, and Alex needs to determine the unique members who are part of exactly one club when considering all clubs together.

Your goal is to help Alex by writing a program that calculates the symmetric difference of membership lists from multiple clubs and then finds the total number of unique members.

***Input Format***

The first line of input consists of an integer k, representing the number of clubs.

The next k lines each contain a space-separated list of integers, where each

integer represents a member's ID.

***Output Format***

The first line of output displays the symmetric difference of the membership lists as a set.

The second line displays the sum of the elements in this symmetric difference.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 3
1 2 3
2 3 4
5 6 7
Output: {1, 4, 5, 6, 7}
23

***Answer***

```
def main():
    # Read the number of clubs
    k = int(input().strip())

    # Initialize a list to hold the sets of members for each club
    club_members = []

    # Read each club's members and store them as sets
    for _ in range(k):
        members = set(map(int, input().strip().split()))
        club_members.append(members)

    # Calculate the symmetric difference across all club member sets
    symmetric_difference = set()

    # Start with the first club's members
    symmetric_difference = club_members[0]

    # Compute the symmetric difference with each subsequent club
    for i in range(1, k):
```

```
        symmetric_difference ^= club_members[i]  # XOR operation for symmetric
difference

    # Calculate the sum of the elements in the symmetric difference
    total_sum = sum(symmetric_difference)

    # Print the results
    print(symmetric_difference)
    print(total_sum)

if __name__ == "__main__":
    main()
```

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

2. Problem Statement

Riley is analyzing DNA sequences and needs to determine which bases match at the same positions in two given DNA sequences. Each DNA sequence is represented as a tuple of integers, where each integer corresponds to a DNA base.

Your task is to write a program that compares these two sequences and identifies the bases that match at the same positions and print it.

***Input Format***

The first line of input consists of an integer n, representing the size of the first tuple.

The second line contains n space-separated integers, representing the elements of the first DNA sequence tuple.

The third line of input consists of an integer m, representing the size of the second tuple.

The fourth line contains m space-separated integers, representing the elements of the second DNA sequence tuple.

***Output Format***

The output is a space-separated integer of the matching bases at the same

positions in both sequences.

Refer to the sample output for format specifications.

***Sample Test Case***

Input: 4
5 1 8 4
4
4 1 8 2

Output: 1 8

***Answer***

```
def main():
    # Read the size of the first tuple
    n = int(input().strip())
    # Read the first DNA sequence
    dna_sequence_1 = tuple(map(int, input().strip().split()))

    # Read the size of the second tuple
    m = int(input().strip())
    # Read the second DNA sequence
    dna_sequence_2 = tuple(map(int, input().strip().split()))

    # Find the minimum length to avoid index errors
    min_length = min(n, m)

    # Identify matching bases at the same positions
    matching_bases = [dna_sequence_1[i] for i in range(min_length) if
dna_sequence_1[i] == dna_sequence_2[i]]

    # Print the matching bases as space-separated integers
    print(" ".join(map(str, matching_bases)))

if __name__ == "__main__":
    main()
```

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

3. Problem Statement

James is an engineer working on designing a new rocket propulsion system. He needs to solve a quadratic equation to determine the optimal launch trajectory. The equation is of the form ax2 +bx+c=0.

Your task is to help James find the roots of this quadratic equation. Depending on the discriminant, the roots might be real and distinct, real and equal, or complex. Implement a program to determine and display the roots of the equation based on the given coefficients.

***Input Format***

The first line of input consists of an integer N, representing the number of coefficients.

The second line contains three space-separated integers a,b, and c representing the coefficients of the quadratic equation.

***Output Format***

The output displays:

1. If the discriminant is positive, display the two real roots.
2. If the discriminant is zero, display the repeated real root.
3. If the discriminant is negative, display the complex roots as a tuple with real and imaginary parts.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 3
1 5 6
Output: (-2.0, -3.0)

***Answer***

```
import math
def solve_quadratic():
    N = int(input())
```

```
    if N != 3:
        print("Invalid input: Number of coefficients must be 3.")
        return
    a, b, c = map(int, input().split())
    discriminant = b**2 - 4*a*c

    if discriminant > 0:
        root1 = (-b + math.sqrt(discriminant)) / (2 * a)
        root2 = (-b - math.sqrt(discriminant)) / (2 * a)
        print((root1, root2))
    elif discriminant == 0:
        root = -b / (2 * a)
        print((root,))
    else:
        real_part = -b / (2 * a)
        imaginary_part = math.sqrt(abs(discriminant)) / (2 * a)
        print(((real_part, imaginary_part), (real_part, -imaginary_part)))
solve_quadratic()
```

***Status :*** Partially correct **Marks : 7.5/10**

4. Problem Statement

Emily is a librarian who keeps track of books borrowed and returned by her patrons. She maintains four sets of book IDs: the first set represents books borrowed, the second set represents books returned, the third set represents books added to the collection, and the fourth set represents books that are now missing. Emily wants to determine which books are still borrowed but not returned, as well as those that were added but are now missing. Finally, she needs to find all unique book IDs from both results.

Help Emily by writing a program that performs the following operations on four sets of integers:

Compute the difference between the borrowed books (first set) and the returned books (second set).Compute the difference between the added books (third set) and the missing books (fourth set).Find the union of the results from the previous two steps, and sort the final result in descending order.

***Input Format***

The first line of input consists of a list of integers representing borrowed books.

The second line of input consists of a list of integers representing returned books.

The third line of input consists of a list of integers representing added books.

The fourth line of input consists of a list of integers representing missing books.

***Output Format***

The first line of output displays the difference between sets P and Q, sorted in descending order.

The second line of output displays the difference between sets R and S, sorted in descending order.

The third line of output displays the union of the differences from the previous two steps, sorted in descending order.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

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

Output: [1]
[5]
[5, 1]

***Answer***

```
def main():
    # Read input sets
    borrowed_books = set(map(int, input().strip().split()))
    returned_books = set(map(int, input().strip().split()))
    added_books = set(map(int, input().strip().split()))
    missing_books = set(map(int, input().strip().split()))
```

```
    # Compute the difference between borrowed and returned books
    borrowed_not_returned = borrowed_books - returned_books
    # Compute the difference between added and missing books
    added_not_missing = added_books - missing_books

    # Sort the results in descending order
    borrowed_not_returned_sorted = sorted(borrowed_not_returned, reverse=True)
    added_not_missing_sorted = sorted(added_not_missing, reverse=True)

    # Compute the union of the two results
    final_result = sorted(borrowed_not_returned.union(added_not_missing),
reverse=True)

    # Print the results
    print(borrowed_not_returned_sorted)
    print(added_not_missing_sorted)
    print(final_result)

if __name__ == "__main__":
    main()
```

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