**DAY-1**

1. Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward.

Example 1:

Input: words = ["abc","car","ada","racecar","cool"]

Output: "ada"

**# Program:-**

# Given array of strings

words = ["abc", "car", "ada", "racecar", "cool"]

# Initialize the result as an empty string

result = ""

# Iterate through each word in the array

for word in words:

    # Check if the word is a palindrome

    if word == word[::-1]:

        # If it is, set the result to this word and break the loop

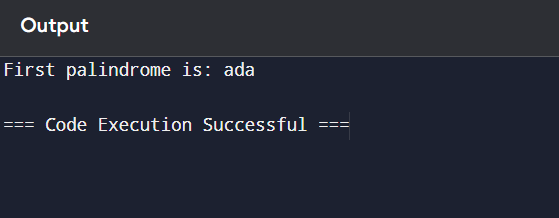
        result = word

        break

# Print the result

print(result)

**output:-**



2.You are given two integer arrays nums1 and nums2 of sizes n and m, respectively. Calculate the following values: answer1 : the number of indices i such that nums1[i] exists in nums2. answer2 : the number of indices i such that nums2[i] exists in nums1 Return [answer1,answer2].

Example 1:

Input: nums1 = [2,3,2], nums2 = [1,2]

Output: [2,1]

**Program:-**

# Given arrays

nums1 = [1, 2, 3, 4]

nums2 = [3, 4, 5, 6]

# Convert nums2 to a set for O(1) average time complexity lookups

set\_nums2 = set(nums2)

# Calculate answer1

answer1 = sum(1 for num in nums1 if num in set\_nums2)

# Convert nums1 to a set for O(1) average time complexity lookups

set\_nums1 = set(nums1)

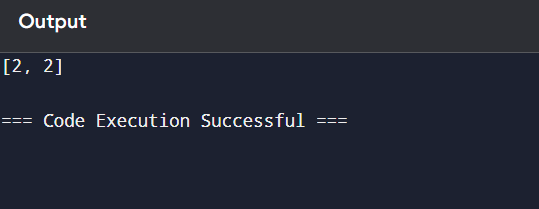
# Calculate answer2

answer2 = sum(1 for num in nums2 if num in set\_nums1)

result = [answer1, answer2]

print(result)

**output:-**

****

3.You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let nums[i..j] be a subarray of nums consisting of all the indices from i to j such that 0 <= i <= j < nums.length. Then the number of distinct values in nums[i..j] is called the distinct count of nums[i..j]. Return the sum of the squares of distinct counts of all subarrays of nums. A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

Input: nums = [1,2,1]

Output: 15

**Program:-**

# Given input

nums = [1, 2, 1]

n = len(nums)

total\_sum = 0

for i in range(n):

    distinct\_elements = set()

    for j in range(i, n):

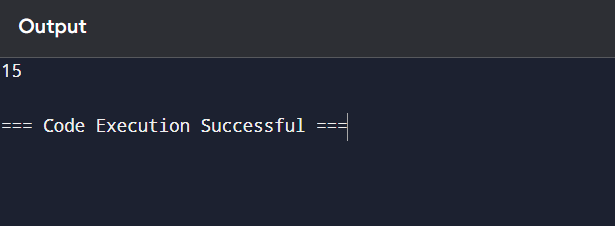
        distinct\_elements.add(nums[j])

        distinct\_count = len(distinct\_elements)

        total\_sum += distinct\_count \*\* 2

print(total\_sum)

**output:-**



4.Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i \* j) is divisible by k.

Example 1:

Input: nums = [3,1,2,2,2,1,3], k = 2

Output: 4

**Program:-**

# Given input

nums = [3, 1, 2, 2, 2, 1, 3]

k = 2

n = len(nums)

count = 0

# Iterate over all possible pairs (i, j)

for i in range(n):

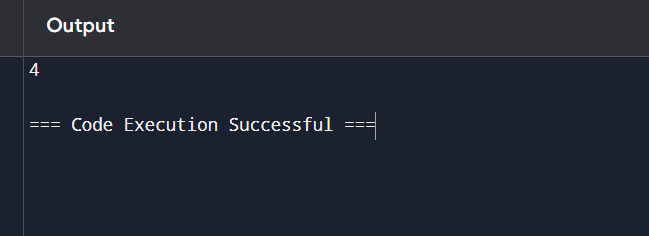
    for j in range(i + 1, n):

        if nums[i] == nums[j] and (i \* j) % k == 0:

            count += 1

print(count)

**output:-**



5.Write a program FOR THE BELOW TEST CASES with least time complexity

Test Cases: -

1) Input: {1, 2, 3, 4, 5} Expected Output: 5

2) Input: {7, 7, 7, 7, 7} Expected Output: 7

3) Input: {-10, 2, 3, -4, 5} Expected Output: 5

**Program:-**

test\_cases = [

    [1, 2, 3, 4, 5],

    [7, 7, 7, 7, 7],

    [-10, 2, 3, -4, 5]

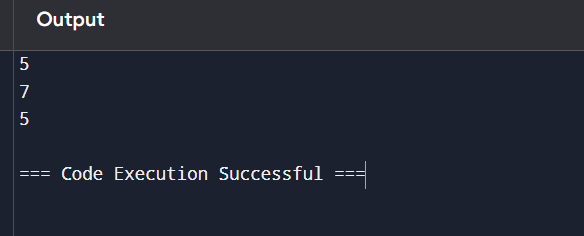
]

for nums in test\_cases:

    max\_value = max(nums)

    print(max\_value)

**Output:-**



6.You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.

Test Cases:-

1. Empty List 1. Input: [] 2. Expected Output: None or an appropriate message indicating that the list is empty.

**Program:-**

# Given list

a = [3,2,5,6,3]

# Step 1: Remove duplicates manually

b = []

for num in a:

    if num not in b:

        b.append(num)

print(b)

# Step 2: Check if the list is empty

if b == []:

    print("none")

else:

    # Step 3: Find the maximum element manually

    maximum = b[0]

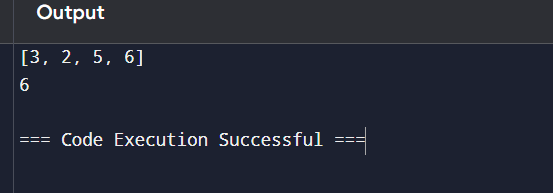
    for num in b:

        if num > maximum:

            maximum = num

    print(maximum)

**output:-**



7.Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm? Test Cases Some Duplicate Elements

• Input: [3, 7, 3, 5, 2, 5, 9, 2]

• Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)

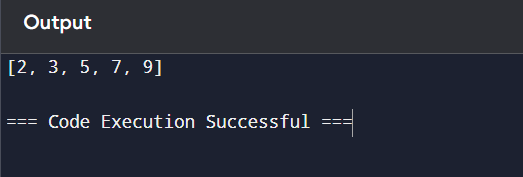
**Program:-**

input\_list = [3, 7, 3, 5, 2, 5, 9, 2]

unique\_elements = list(set(input\_list))

print(unique\_elements)

**Output:-**



8.Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code

**Program:-**

# Input array

arr = [64, 34, 25, 12, 22, 11, 90]

n = len(arr)

# Bubble sort algorithm

for i in range(n):

    swapped = False

    for j in range(0, n - i - 1):  # Last i elements are already sorted

        if arr[j] > arr[j + 1]:

            # Swap if the element found is greater

            arr[j], arr[j + 1] = arr[j + 1], arr[j]

            swapped = True

    # If no two elements were swapped, the array is sorted

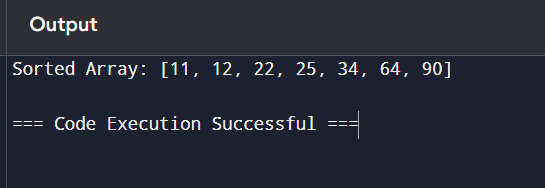
    if not swapped:

        break

# Output the sorted array

print("Sorted Array:", arr)

**output:-**



9.Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.

Test Case:

Example X={ 3,4,6,-9,10,8,9,30} KEY=10

Output: Element 10 is found at position 5

**Program:-**

# Input sorted array

arr = [-9, 3, 4, 6, 8, 9, 10, 30]  # Note: The array should be sorted

key = 10  # Element to search for

# Binary search algorithm

low = 0

high = len(arr) - 1

found = False

while low <= high:

    mid = (low + high) // 2  # Find the middle index

    if arr[mid] == key:

        print(f"Element {key} is found at position {mid}")

        found = True

        break

    elif arr[mid] < key:

        low = mid + 1  # Search in the right half

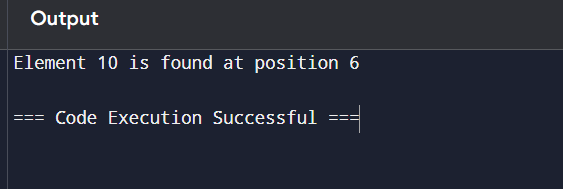
    else:

        high = mid - 1  # Search in the left half

if not found:

    print(f"Element {key} is not found in the array.")

**output:-**



10.Given an array of integers nums, sort the array in ascending order and return it. You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

**Program:**-

arr=[1,5,2,8,4,-9,9,22]

n=len(arr)

for i in range(n):

    for j in range(0,n-i-1):

        if (arr[j]>arr[j+1]):

            arr[j],arr[j+1]=arr[j+1],arr[j]

print(arr)

**output:-**

