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.

Aim:

To write a program to read the same forward and backward in a string is a palindrome

Algorithm:

Step 1: Start

Step 2: Read the array of strings words

Step 3: For each string word in words, do the following:

a. Reverse the string → rev = word[::-1]

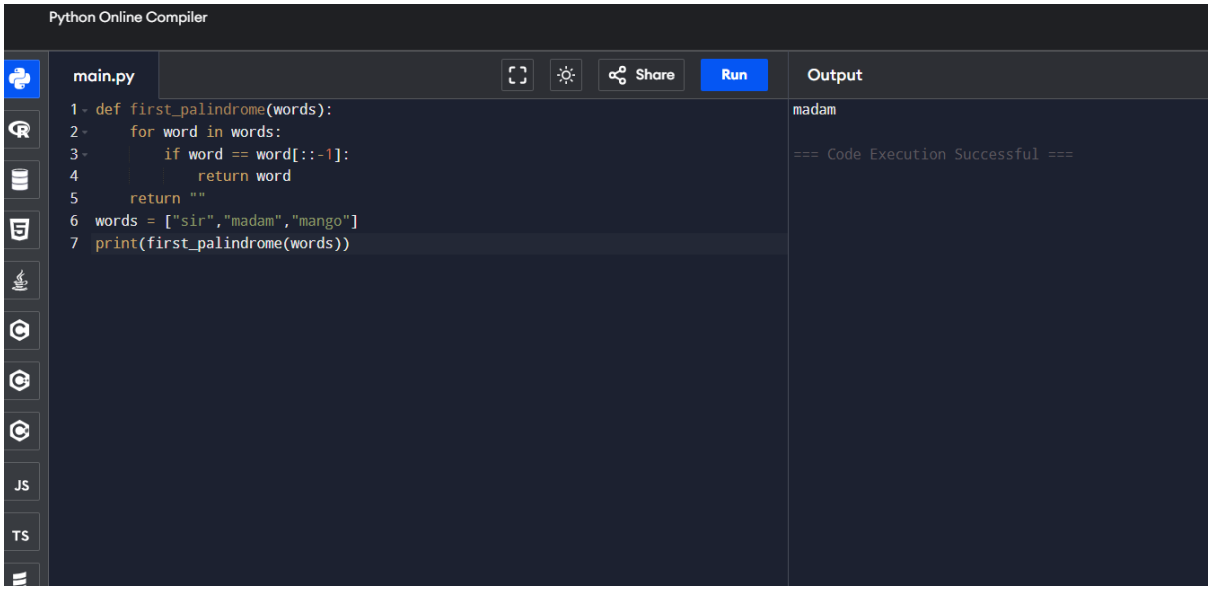
b. Compare the original string with its reverse

If word == rev, then

Return word (This is the first palindrome)

Step 4: If no palindromic string is found after checking all words, return ""

Step 5: Stop



Result :

The program has been executed successfully

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].

Aim:

To find how many elements from nums1 exist in nums2 and how many elements from nums2 exist in nums1, and return both counts as [answer1, answer2].

Algorithm:

Step 1: Start

Step 2: Read arrays nums1 and nums2

Step 3: Initialize answer1 = 0 and answer2 = 0

Step 4:

For each element x in nums1:

If x exists in nums2, increment answer1 by 1

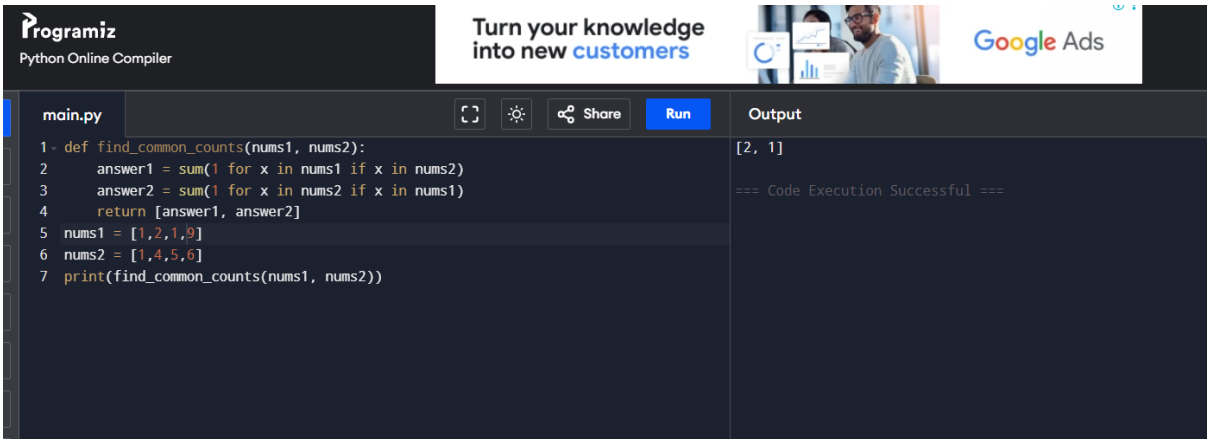
Step 5:

For each element x in nums2:

If x exists in nums1, increment answer2 by 1

Step 6: Return [answer1, answer2]

Step 7: Stop



Result:

The program has been executed successfully

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.

Aim:

To find the sum of squares of the number of distinct elements in all possible subarrays of a given integer array.

Algorithm:

Step 1: Start

Step 2: Initialize total = 0

Step 3: For each index i from 0 to n-1:

a. Create an empty set seen

b. For each index j from i to n-1:

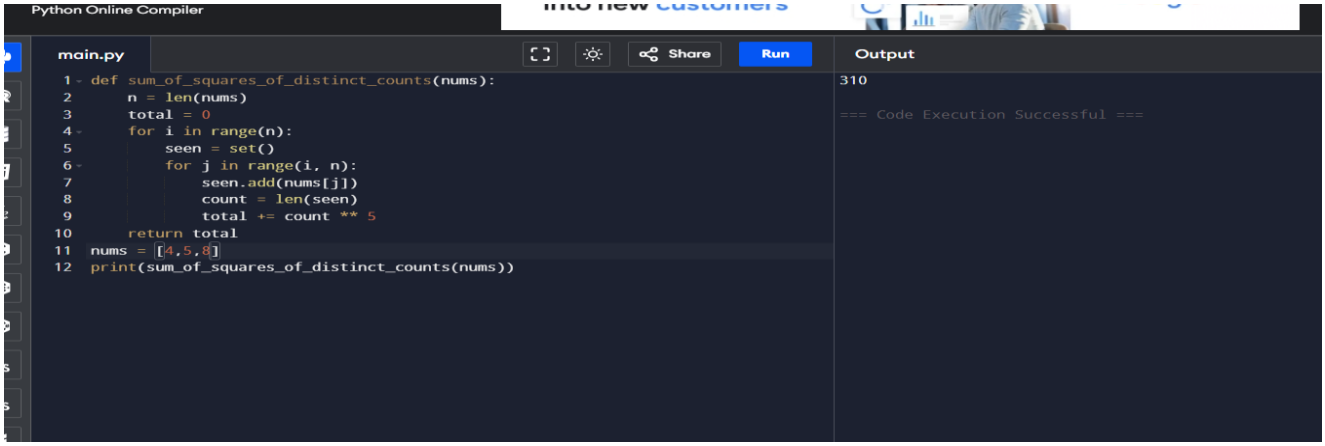
i. Add nums[j] to seen

ii. Find count = len(seen)

iii. Add count² to total

Step 4: After all subarrays are processed, return total

Step 5: Stop



Result :

The program has been executed successfully

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.

Aim:

To count the number of index pairs (i, j) where nums[i] equals nums[j] and the product (i * j) is divisible by a given integer k.

Algorithm:

Step 1: Start

Step 2: Initialize count = 0

Step 3: Loop i from 0 to n - 1

Loop j from i + 1 to n - 1

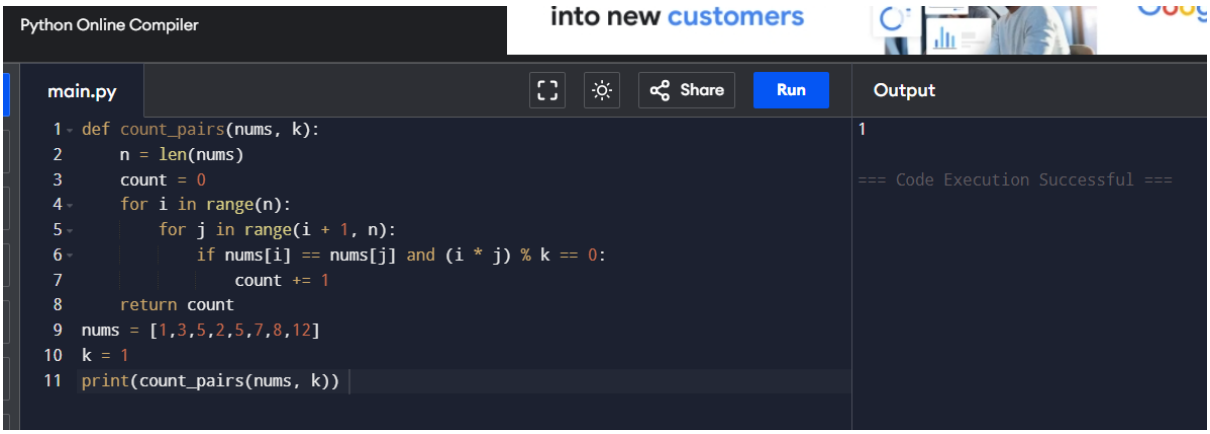
If both conditions are true:

- nums[i] == nums[j]
- (i * j) % k == 0

Then increment count by 1

Step 4: After checking all pairs, return count

Step 5: Stop



Result :

The program has been executed successfully

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

Aim:

To write the program for the below test cases with least time complexity.

Algorithm:

Step 1: Start

Step 2: Create an empty dictionary indices to group indices by the values in nums.

Step 3: For each index i and element num in nums:

Add i to the list indices[num].

Step 4: Initialize count = 0.

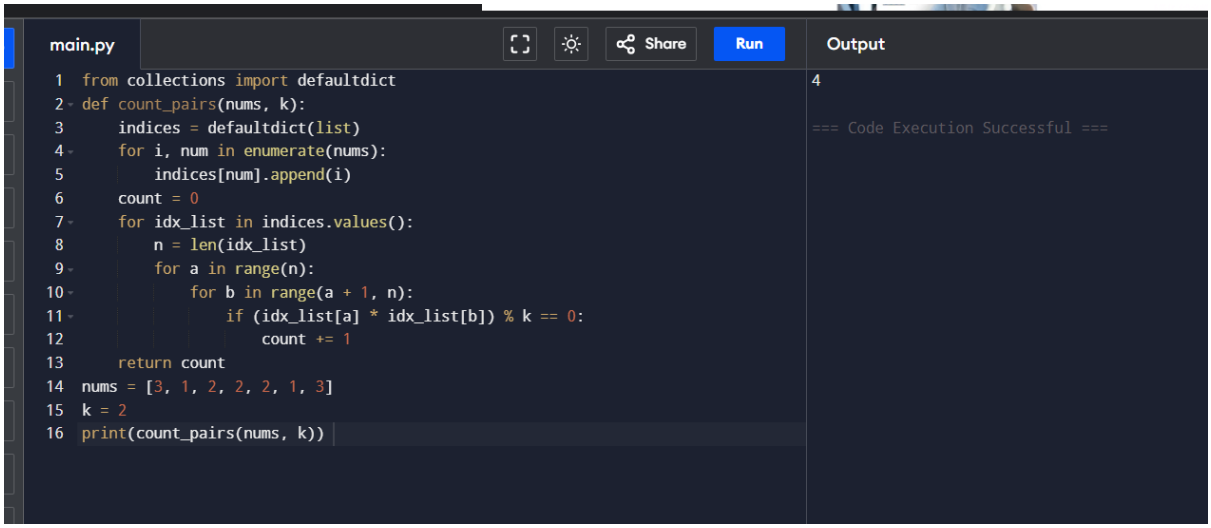
Step 5: For each list of indices idx_list in the dictionary indices:

For every pair of indices (a, b) in idx_list where a < b:

If (a * b) % k == 0, increment count by 1.

Step 6: After checking all pairs, return count.

Step 7: Stop



Result:

The program has been executed successfully

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.

Aim:

To sort a list of numbers efficiently and then find the maximum element from the sorted list using minimal time complexity.

Algorithms:

Step 1: Start

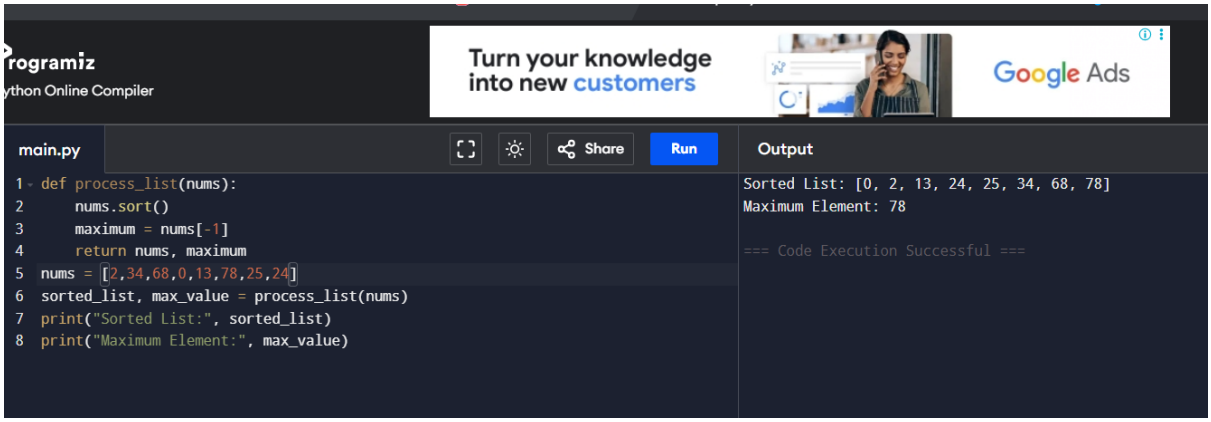
Step 2: Read the list nums

Step 3: Sort the list using an efficient sorting algorithm (Python uses Timsort, which is O(n log n))

Step 4: Retrieve the last element of the sorted list as the maximum value

Step 5: Return the sorted list and the maximum element

Step 6: Stop



Result:

The program has been executed successfully

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?

Aim:

To create a new list containing only the unique elements from a given list and analyse its time and space efficiency.

Algorithms:

Step 1: Start

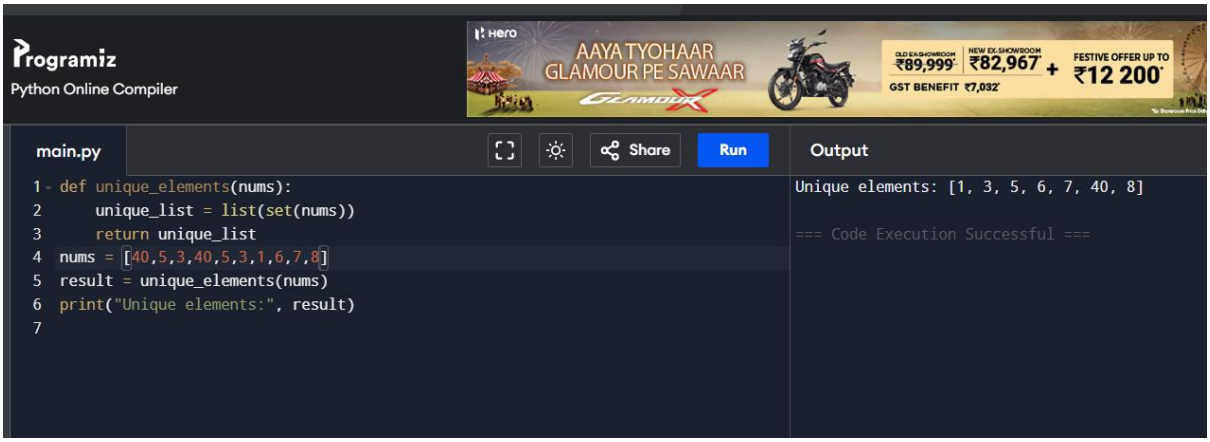
Step 2: Read the input list nums

Step 3: Convert nums to a set to remove duplicates (set(nums))

Step 4: Convert the set back to a list (list(set(nums)))

Step 5: Return the new list of unique elements

Step 6: Stop



Result:

The program has been executed successfully

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

Aim:

To sort an array of integers using the Bubble Sort technique and analyse its time complexity in Big-O notation.

Algorithms:

Step 1: Start
Step 2: Read the input array arr of size n
Step 3: Repeat for i from 0 to n-1
a. Set swapped = False
b. For j from 0 to n - i - 2:
If arr[j] > arr[j + 1], then swap them and set swapped = True
c. If no swaps were made in this pass, stop early (array is sorted)
Step 4: Return the sorted array
Step 5: Stop

Result:

The program has been executed successfully

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

Aim:

To determine whether a given number exists in a sorted array using the binary search technique and analyze its time complexity (O(log n)

Algorithms:

Step 1: Start
Step 2: Initialize low = 0, high = n - 1
Step 3: Repeat while low <= high
a. Compute mid = (low + high) // 2
b. If arr[mid] == x, return True
c. If arr[mid] < x, set low = mid + 1
d. Else, set high = mid - 1
Step 4: If loop ends, return False (element not found)
Step 5: Stop

Result:

The program has been executed successfully

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.

Aim:

To sort an array of integers in ascending order using Heap Sort without built-in functions, achieving O(n log n) time complexity and O(1) space complexity.

Algorithms:

Step 1: Start
Step 2: Build a **max heap** from the array
Step 3: Repeat until the heap size is 1
a. Swap the root (maximum element) with the last element
b. Reduce heap size by 1
c. Heapify the root to maintain the heap property
Step 4: The array is now sorted in ascending order
Step 5: Stop

main.py

Share Run

```
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and arr[left] > arr[largest]:
        largest = left
    if right < n and arr[right] > arr[largest]:
        largest = right
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)
def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr
nums = [56,55,68,74,12,34,36,69]
sorted_nums = heap_sort(nums)
print("Sorted Array:", sorted_nums)
```

Output

```
Sorted Array: [12, 34, 36, 55, 56, 68, 69, 74]

=== Code Execution Successful ===
```

Result:

The program has been executed successfully