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.**

```
words = ["abc", "car", "ada", "racecar", "cool"]
palindrome=""
for i in words:
    if i==i[::-1]:
        palindrome=i
        break
print(palindrome)
```

```
ada
```

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].**

```
nums1 = [2, 3, 2]
nums2 = [1, 2]
answer1 = 0
answer2 = 0
for i in range(len(nums1)):
    for j in range(len(nums2)):
        if nums1[i] == nums2[j]:
            answer1 += 1
            break
for i in range(len(nums2)):
    for j in range(len(nums1)):
        if nums2[i] == nums1[j]:
            answer2 += 1
            break
print([answer1, answer2])
```

```
[2, 1]
```

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.**

```
nums = [1, 2, 1]
total_sum_of_squares = 0
for i in range(len(nums)):
    distinct_elements = set()
    for j in range(i, len(nums)):
        distinct_elements.add(nums[j])
        distinct_count = len(distinct_elements)
        total_sum_of_squares += distinct_count * distinct_count
print(total_sum_of_squares)
```

```
15
```

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

   Test Cases: -

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

```
nums1 = [1, 2, 3, 4, 5]
print(max(nums1))
```

```
5
```

5. **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]**

```
def find_unique_elements(nums):
    unique_elements = list(set(nums))
    return unique_elements
input1 = [3, 7, 3, 5, 2, 5, 9, 2]
print(find_unique_elements(input1))
```

```
[2, 3, 5, 7, 9]
```

6. **You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security systems connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.**

```
def rob_linear(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    prev1 = 0
    prev2 = 0
    for num in nums:
        current = max(prev1, prev2 + num)
        prev2 = prev1
        prev1 = current
    return prev1
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    max_profit1 = rob_linear(nums[:-1])
    max_profit2 = rob_linear(nums[1:])
    return max(max_profit1, max_profit2)
nums = [2, 3, 2]
print(rob(nums))
```

```
3
```

7. **A robot is located at the top-left corner of a m×n grid .The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid. How many possible unique paths are there?**

```
def uniquePaths(m,n):
    dp = [[0] * n for _ in range(m)]
    dp[0][0] = 1
    for j in range(1, n):
        dp[0][j] = 1
    for i in range(1, m):
        dp[i][0] = 1
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[m-1][n-1]
```

```
m = 7
n = 3
print(uniquePaths(m,n))
```
```
28
```

8. **In a string S of lowercase letters, these letters form consecutive groups of the same character. For example, a string like s = "abbxxxxzyy" has the groups "a", "bb", "xxxx", "z", and "yy". A group is identified by an interval [start, end], where start and end denote the start and end indices (inclusive) of the group. In the above example, "xxxx" has the interval [3,6]. A group is considered large if it has 3 or more characters. Return the intervals of every large group sorted in increasing order by start index.**

```
def largeGroupPositions(s):
    n = len(s)
    if n == 0:
        return []
    result = []
    start = 0
    for end in range(1, n):
        if s[end] != s[end - 1]:
            if end - start >= 3:
                result.append([start, end - 1])
            start = end
    if n - start >= 3:
        result.append([start, n - 1])
    return result
s = "abbxxxxzzy"
print(largeGroupPositions(s))
```
```
[[3, 6]]
```

9. **A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time.**

```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] < nums[mid + 1]:
            left = mid + 1
        else:
            right = mid
    return left
nums = [1, 2, 3, 1]
print("Peak Element Index:", find_peak_element(nums))
```
```
Peak Element Index: 2
```

10. **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**

```
def count_pairs(nums,k):
    n = len(nums)
```

```
    index_map = {}
    count = 0
    for idx, num in enumerate(nums):
        if num in index_map:
            index_map[num].append(idx)
        else:
            index_map[num] = [idx]
    for num in index_map:
        indices = index_map[num]
        m = len(indices)
        for i in range(m):
            for j in range(i + 1, m):
                if (indices[i] * indices[j]) % k == 0:
                    count += 1
    return count
nums = [3, 1, 2, 2, 2, 1, 3]
k = 2
print(count_pairs(nums,k))
```

```
4
```