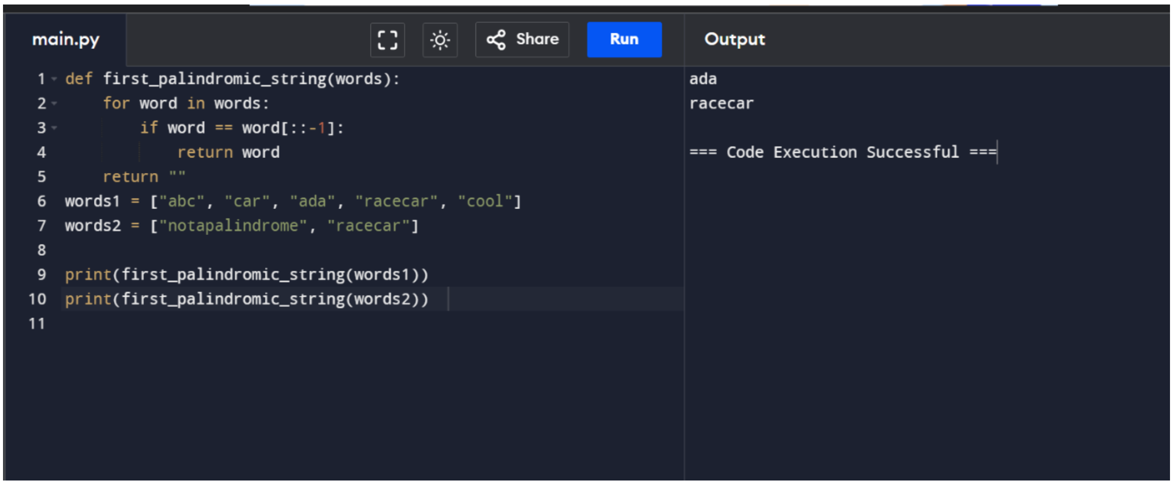
DAY-1 PROGRAMS

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" Explanation: The first string that is palindromic is "ada". Note that "racecar" is also palindromic, but it is not the first.

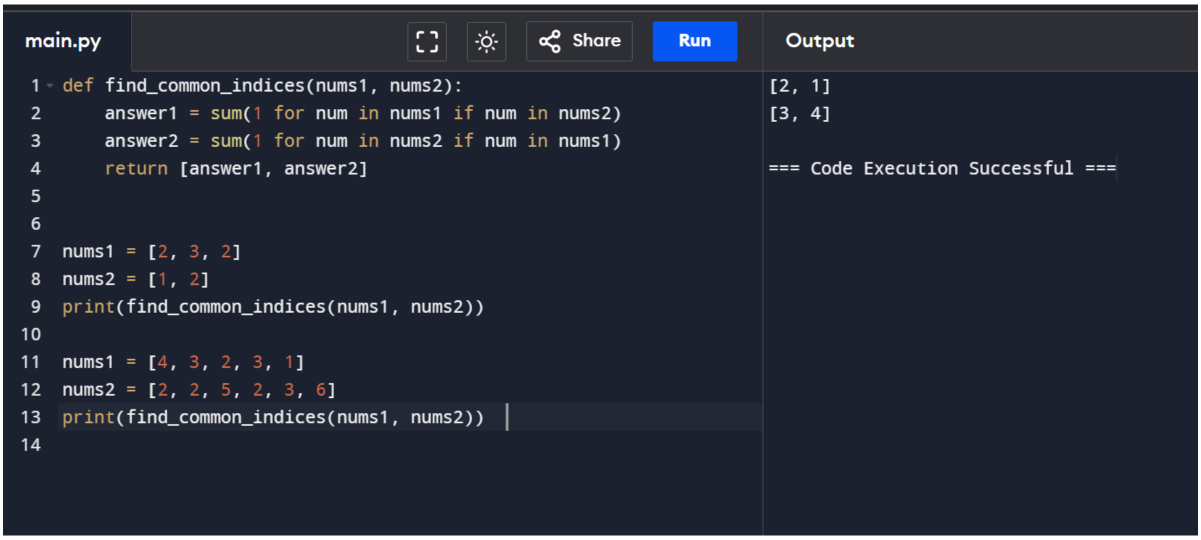
Example 2: Input: words = ["notapalindrome","racecar"] Output: "racecar" Explanation: The first and only string that is palindromic is "racecar"



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] Explanation:

Example 2: Input: nums1 = [4,3,2,3,1], nums2 = [2,2,5,2,3,6] Output: [3,4] Explanation: The elements at indices 1, 2, and 3 in nums1 exist in nums2 as well. So answer1 is 3. The elements at indices 0, 1, 3, and 4 in nums2 exist in nums1. So answer2 is 4.



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

Explanation: Six possible subarrays are: [1]: 1 distinct value [2]: 1 distinct value [1]: 1 distinct value [1,2]: 2 distinct values [2,1]: 2 distinct values [1,2,1]: 2 distinct values The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 + 22 + 22 + 22 = 15.

main.py Share Run Output

```python
def sum_of_squares_of_distinct_counts(nums):
    n = len(nums)
    result = 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)
            result += distinct_count ** 2

    return result

nums = [1, 2, 1]
print(sum_of_squares_of_distinct_counts(nums))
```

```
15

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

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
Explanation: There are 4 pairs that meet all the requirements:
- nums[0] == nums[6], and 0 * 6 == 0, which is divisible by 2.
- nums[2] == nums[3], and 2 * 3 == 6, which is divisible by 2.
- nums[2] == nums[4], and 2 * 4 == 8, which is divisible by 2.
- nums[3] == nums[4], and 3 * 4 == 12, which is divisible by 2.
Example 2: Input: nums = [1,2,3,4], k = 1 Output: 0
Explanation: Since no value in nums is repeated, there are no pairs (i,j) that meet all the requirements.

main.py

```python
def count_pairs(nums, k):
    count = 0
    n = len(nums)
    for i in range(n):
        for j in range(i + 1, n):
            if nums[i] == nums[j] and (i * j) % k == 0:
                count += 1

    return count
nums1 = [3, 1, 2, 2, 2, 1, 3]
k1 = 2
print(count_pairs(nums1, k1))

nums2 = [1, 2, 3, 4]
k2 = 1
print(count_pairs(nums2, k2))
```

Output

```
4
0

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

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

main.py

```python
def find_max(arr):
    max_value = arr[0]
    for num in arr[1:]:
        if num > max_value:
            max_value = num

    return max_value

print(find_max([1, 2, 3, 4, 5]))
print(find_max([7, 7, 7, 7, 7]))
print(find_max([-10, 2, 3, -4, 5]))
```

Output

```
5
7
5

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

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.
2. Single Element List 1. Input: [5] 2. Expected Output: 5
3. All Elements are the Same 1. Input: [3, 3, 3, 3, 3] 2. Expected Output: 3

main.py

```python
def process_list(nums):
    if not nums:
        return "The list is empty."
    nums.sort()
    return nums[-1]

print(process_list([]))
print(process_list([5]))
print(process_list([3, 3, 3, 3, 3]))
```

Output

```
The list is empty.
5
3

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

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)
Negative and Positive Numbers
• Input: [-1, 2, -1, 3, 2, -2]
• Expected Output: [-1, 2, 3, -2] (Order may vary)
List with Large Numbers
• Input: [1000000, 999999, 1000000]
• Expected Output: [1000000, 999999]

main.py

```python
def unique_elements(nums):
    unique_set = set(nums)
    unique_list = list(unique_set)
    return unique_list

print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
print(unique_elements([-1, 2, -1, 3, 2, -2]))
print(unique_elements([1000000, 999999, 1000000]))
```

Output

```
[2, 3, 5, 7, 9]
[2, 3, -1, -2]
[1000000, 999999]

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

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

main.py

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break


arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print("Sorted array:", arr)
```

Output

```
Sorted array: [11, 12, 22, 25, 34, 64, 90]

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

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

Example X={ 3,4,6,-9,10,8,9,30} KEY=100

Output : Element 100 is not found

main.py

```python
def binary_search(arr, key):
    arr.sort()
    def binary_search_helper(arr, key, left, right):
        if left <= right:
            mid = (left + right) // 2
            if arr[mid] == key:
                return mid
            elif arr[mid] < key:
                return binary_search_helper(arr, key, mid + 1, right)
            else:
                return binary_search_helper(arr, key, left, mid - 1)
        return -1
    index = binary_search_helper(arr, key, 0, len(arr) - 1)
    if index != -1:
        return f"Element {key} is found at position {index}"
```

Output

```
Element 10 is found at position 6
Element 100 is not found

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

main.py

```python
                right)
            else:
                return binary_search_helper(arr, key, left, mid - 1)
        return -1
    index = binary_search_helper(arr, key, 0, len(arr) - 1)
    if index != -1:
        return f"Element {key} is found at position {index}"
    else:
        return f"Element {key} is not found"

arr1 = [3, 4, 6, -9, 10, 8, 9, 30]
key1 = 10
print(binary_search(arr1, key1))
arr2 = [3, 4, 6, -9, 10, 8, 9, 30]
key2 = 100
print(binary_search(arr2, key2))
```

Output


```
Element 10 is found at position 6
Element 100 is not found

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

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.

main.py

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    sorted_arr = []
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] <= right[right_idx]:
            sorted_arr.append(left[left_idx])
            left_idx += 1
        else:
            sorted_arr.append(right[right_idx])
            right_idx += 1
```

Output

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

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

main.py

```python
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] <= right[right_idx]:
            sorted_arr.append(left[left_idx])
            left_idx += 1
        else:
            sorted_arr.append(right[right_idx])
            right_idx += 1

    sorted_arr.extend(left[left_idx:])
    sorted_arr.extend(right[right_idx:])

    return sorted_arr

nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_nums = merge_sort(nums)
print(sorted_nums)
```

Output

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

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