

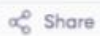


Win big with foundit

Foundit

[apply now](#)

main.py



Share

Run

Output

```
1- def count_intersections(nums1, nums2):
2     answer1 = sum(1 for i in nums1 if i in nums2)
3     answer2 = sum(1 for i in nums2 if i in nums1)
4     return [answer1, answer2]
5 nums1 = [2, 3, 2]
6 nums2 = [1, 2]
7 output = count_intersections(nums1, nums2)
8 print(output)
9
```

[2, 1]

=== Code Execution Successful ===

main.py



Run

Output

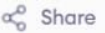
Clear

```
1- def first_palindromic_string(words):
2-     for word in words:
3-         if word == word[::-1]:
4-             return word
5-     return ""
6 words = ["aba", "car", "ada", "racecar", "cooloo"]
7 print(first_palindromic_string(words))
8
```

aba

=== Code Execution Successful ===

main.py



Share

Run

Output

```
1- def get_unique_elements(input_list):  
2     return list(set(input_list))  
3 input1 = [3, 7, 3, 5, 2, 5, 9, 2]  
4 print(get_unique_elements(input1))
```

[2, 3, 5, 7, 9]

=== Code Execution Successful ===

```
1 a=[1,2,3,48,6,9]
2 b=max(a)
3 print(b)
```

48

=== Code Execution Successful ===



```
1 def countPairs(nums, k):
2     count = 0
3     n = len(nums)
4     num_indices = {}
5
6     for i in range(n):
7         if nums[i] in num_indices:
8             for j in num_indices[nums[i]]:
9                 if (i * j) % k == 0:
10                     count += 1
11             num_indices[nums[i]].append(i)
12         else:
13             num_indices[nums[i]] = [i]
14     return count
15 print(countPairs([3,1,2,2,2,1,3], 2))
16 print(countPairs([1,2,3,4], 1))
17
```

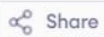
4

0

=== Code Execution Successful ===



main.py



Share

Run

Output

```
1- def sum_of_squares_distinct_counts(nums):
2     result = 0
3-     for i in range(len(nums)):
4-         for j in range(i, len(nums)):
5             distinct_count = len(set(nums[i:j+1]))
6             result += distinct_count ** 2
7     return result
8 nums = [1, 2, 1]
9 output = sum_of_squares_distinct_counts(nums)
10 print(output)
11
```

15

=== Code Execution Successful ===

```
def process_numbers(numbers):
```

```
    if not numbers:
```

```
        return "The list is empty."
```

```
    sorted_numbers = sorted(numbers)
```

```
    return sorted_numbers[-1]
```

```
print(process_numbers([]))
```

```
print(process_numbers([5]))
```

The list is empty.

5

=== Code Execution Successful ===



```
1 def binary_search(arr, key):
2     left, right = 0, len(arr) - 1
3     while left <= right:
4         mid = (left + right) // 2
5         if arr[mid] == key:
6             return f"Element {key} is found at position {mid}"
7         elif arr[mid] < key:
8             left = mid + 1
9         else:
10            right = mid - 1
11    return f"Element {key} is not found"
12 arr = sorted([3, 4, 6, -9, 10, 8, 9, 30])
13 print(binary_search(arr, 10))
14 print(binary_search(arr, 100))
```

Element 10 is found at position 6

Element 100 is not found

=== Code Execution Successful ===





```
1 def bubble_sort(arr):  
2     n = len(arr)  
3     for i in range(n):  
4         for j in range(0, n-i-1):  
5             if arr[j] > arr[j+1]:  
6                 arr[j], arr[j+1] = arr[j+1], arr[j]  
7     return arr  
8 arr = [64, 34, 25, 12, 22, 11, 90]  
9 sorted_arr = bubble_sort(arr)  
10 print("Sorted array:", sorted_arr)
```

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

=== Code Execution Successful ===



```
1 def quick_sort(nums):
2     if len(nums) <= 1:
3         return nums
4     pivot = nums[len(nums) // 2]
5     left = [x for x in nums if x < pivot]
6     middle = [x for x in nums if x == pivot]
7     right = [x for x in nums if x > pivot]
8     return quick_sort(left) + middle + quick_sort(right)
9 nums = [5, 2, 9, 1, 5, 6]
10 sorted_nums = quick_sort(nums)
11 print(sorted_nums)
12
```

[1, 2, 5, 5, 6, 9]

=== Code Execution Successful ===