

Python

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

Input Goes Here..

Copy

Run

Run Program(Ctrl+Enter)

URL as well)

Time(sec) : 0.001

Memory(MB) : 6.1796875

Output:

Copy

ada

Python

```
1 def count_common(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 print(count_common(nums1, nums2))
8
```

Input Goes Here..

Copy

Run

Run Program (Ctrl+Enter)

URL as well)

Time(sec) : 0.012







Memory(MB) : 6.16796875

Output:

Copy

[2, 1]

Python



```
1 def sum_of_squares(nums):
2     result = 0
3     n = len(nums)
4     for i in range(n):
5         distinct = set()
6         for j in range(i, n):
7             distinct.add(nums[j])
8             result += len(distinct) ** 2
9     return result
10 nums = [1, 2, 1]
11 print(sum_of_squares(nums))
12
```

Input Goes Here..

Copy

Run

Run Program(Ctrl+Enter)

Time(sec) : 0.010

Memory(MB) : 6.1953125

Output:

Copy

15



Copy

Run Program(Ctrl+Enter)

Memory(MB) : 6.203125

Copy

4

Python

1- def find_max(arr):

2- if not arr:

3- return None

4- return max(arr)

5 arr = [1, 2, 3, 4, 5]

6 print(find_max(arr))

7

Input Goes Here..

Copy

Run

Run Program (Ctrl+Enter)

URL as well)

Time(sec) : 0.007

Memory(MB) : 6.22265625

Output:

Copy

5

Python

```
1 def sort_and_find_max(arr):
2     if not arr:
3         return None
4     arr.sort()
5     return arr[-1]
6 arr = [64, 34, 25, 12, 22, 11, 90]
7 print(sort_and_find_max(arr))
8
```

Input Goes Here..

Copy

> Run

> Run+URL (Generates URL as well)

Time(sec) : 0.007

Memory(MB) : 6.19140625

Output:

Copy

90

Python

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

Input Goes Here..

Copy

➤ Run

➤ Run+URL (Generates URL as well)

Time(sec) : 0.013

Memory(MB) : 6.1640625

Output:

Copy

[9, 2, 3, 5, 7]

Python



```
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 mid
7         elif arr[mid] < key:
8             left = mid + 1
9         else:
10            right = mid - 1
11    return -1
12 arr = sorted([3, 4, 6, -9, 10, 8, 9, 30])
13 key = 10
14 print(binary_search(arr, key))
15
```

Input Goes Here..

Copy

Run

Run+URL (Generates URL as well)

Time(sec) : 0.000

Memory(MB) : 6.140625

Output:

Copy

6

Python



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

Input Goes Here..

Copy

Run

Run+URL (Generates URL as well)

Time(sec) : 0.015

Memory(MB) : 6.2109375

Output:

Copy

[1, 1, 2, 3, 6, 8, 10]

Python



```
1- def find_ways(m, n, N, i, j):
2-     memo = {}
3-     def dfs(x, y, steps_left):
4-         if x < 0 or y < 0 or x >= m or y >= n:
5-             return 1
6-         if steps_left == 0:
7-             return 0
8-         if (x, y, steps_left) in memo:
9-             return memo[(x, y, steps_left)]
10-        moves = dfs(x - 1, y, steps_left - 1) + dfs(x + 1, y, steps_left - 1) + dfs(x, y - 1, steps_left - 1) + dfs(x, y + 1, steps_left - 1)
11-        memo[(x, y, steps_left)] = moves
12-        return moves
13-     return dfs(i, j, N)
14- print(find_ways(2, 2, 2, 0, 0))
15- print(find_ways(1, 3, 3, 0, 1))
16-
17-
```

Input Goes Here..

Copy

➤ Run

➤ Run+URL (Generates URL as well)

Time(sec) : 0.011

Memory(MB) : 6.20703125

Output:

Copy

6
12