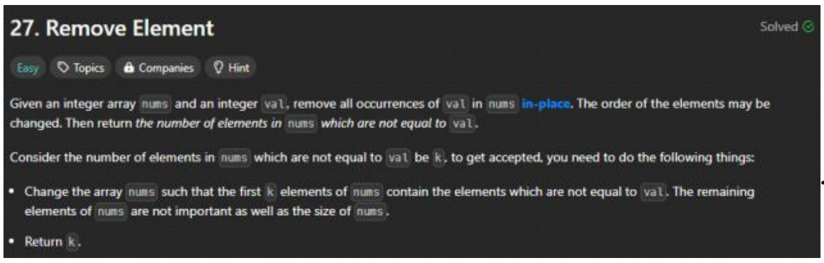
Remove Element

02 March 2025 09:50



27. Remove Element

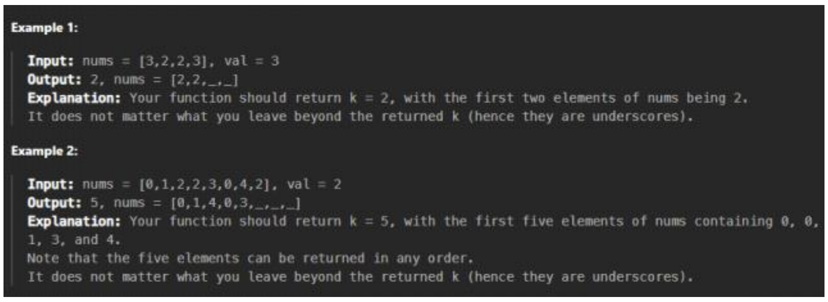
Solved

Easy · Topics · Companies · Hint

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` in-place. The order of the elements may be changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.



Example 1:

```
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 2.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,1,2,2,3,0,4,2], val = 2
Output: 5, nums = [0,1,4,0,3,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0,
1, 3, and 4.
Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

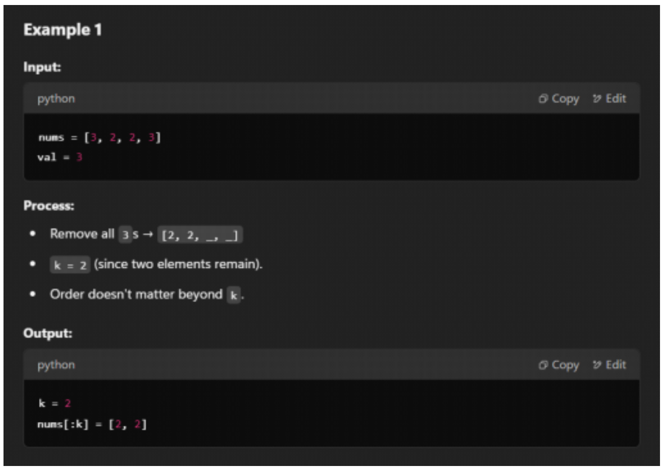
Understanding the Question

You're given:

1. **An integer array nums**
2. **An integer val**

Your task:

- **Remove all occurrences of val from nums in-place** (without using extra space).
- **Return k**, the number of elements in nums that are **not** equal to val.
- The order of the remaining elements **can be changed**.
- The values beyond the first k elements **don't matter**.



Example 1

Input:

python

```python
nums = [3, 2, 2, 3]
val = 3
```

Process:

- Remove all `3`s → `[2, 2, _, _]`
- `k = 2` (since two elements remain).
- Order doesn't matter beyond `k`.

Output:

python

```python
k = 2
nums[:k] = [2, 2]
```

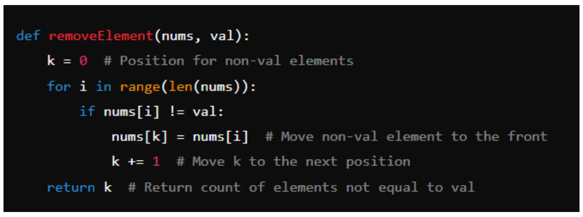
Constraints:

- 0 <= nums.length <= 100
- 0 <= nums[i] <= 50
- 0 <= val <= 100

Approach:

- Use a **slow pointer (k)** to keep track of where the next non-val element should be placed.
- Use a **fast pointer (i)** to iterate over the list.
- If nums[i] is **not equal** to val, place it at nums[k] and increment k.

Code:



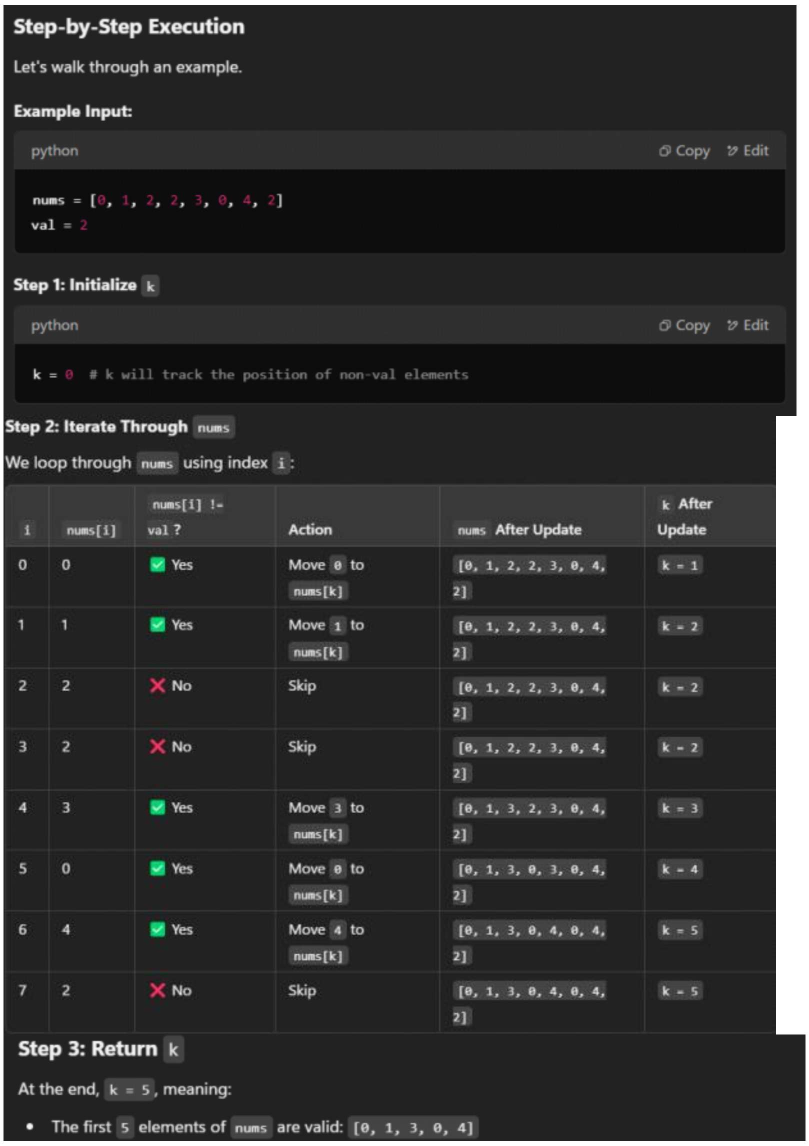
```python
def removeElement(nums, val):
    k = 0  # Position for non-val elements
    for i in range(len(nums)):
        if nums[i] != val:
            nums[k] = nums[i]  # Move non-val element to the front
            k += 1  # Move k to the next position
    return k  # Return count of elements not equal to val
```

Time Complexity:

- **O(n)** → We traverse the list once.

Space Complexity:

- **O(1)** → We modify the list in place.



Step-by-Step Execution

Let's walk through an example.

Example Input:

python

```python
nums = [0, 1, 2, 2, 3, 0, 4, 2]
val = 2
```

Step 1: Initialize `k`

python

```python
k = 0  # k will track the position of non-val elements
```

Step 2: Iterate Through `nums`

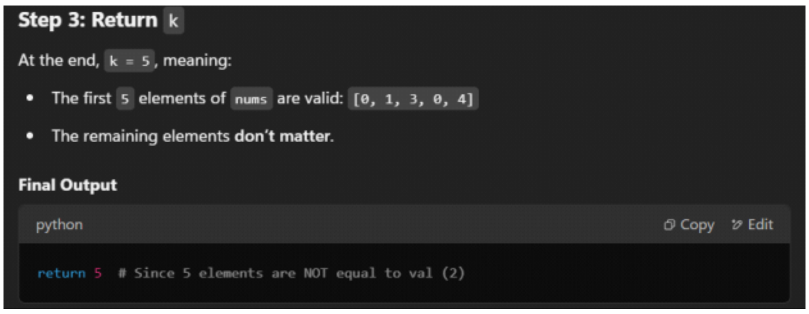
We loop through `nums` using index `i`:

`i`	`nums[i]`	`nums[i] != val` ?	Action	`nums` After Update	`k` After Update
0	0	✅ Yes	Move `0` to `nums[k]`	`[0, 1, 2, 2, 3, 0, 4, 2]`	`k = 1`
1	1	✅ Yes	Move `1` to `nums[k]`	`[0, 1, 2, 2, 3, 0, 4, 2]`	`k = 2`
2	2	❌ No	Skip	`[0, 1, 2, 2, 3, 0, 4, 2]`	`k = 2`
3	2	❌ No	Skip	`[0, 1, 2, 2, 3, 0, 4, 2]`	`k = 2`
4	3	✅ Yes	Move `3` to `nums[k]`	`[0, 1, 3, 2, 3, 0, 4, 2]`	`k = 3`
5	0	✅ Yes	Move `0` to `nums[k]`	`[0, 1, 3, 0, 3, 0, 4, 2]`	`k = 4`
6	4	✅ Yes	Move `4` to `nums[k]`	`[0, 1, 3, 0, 4, 0, 4, 2]`	`k = 5`
7	2	❌ No	Skip	`[0, 1, 3, 0, 4, 0, 4, 2]`	`k = 5`

Step 3: Return `k`

At the end, `k = 5`, meaning:

- The first `5` elements of `nums` are valid: `[0, 1, 3, 0, 4]`



Step 3: Return `k`

At the end, `k = 5`, meaning:

- The first `5` elements of `nums` are valid: `[0, 1, 3, 0, 4]`
- The remaining elements **don't matter**.

Final Output

python Copy Edit

```python
return 5  # Since 5 elements are NOT equal to val (2)
```