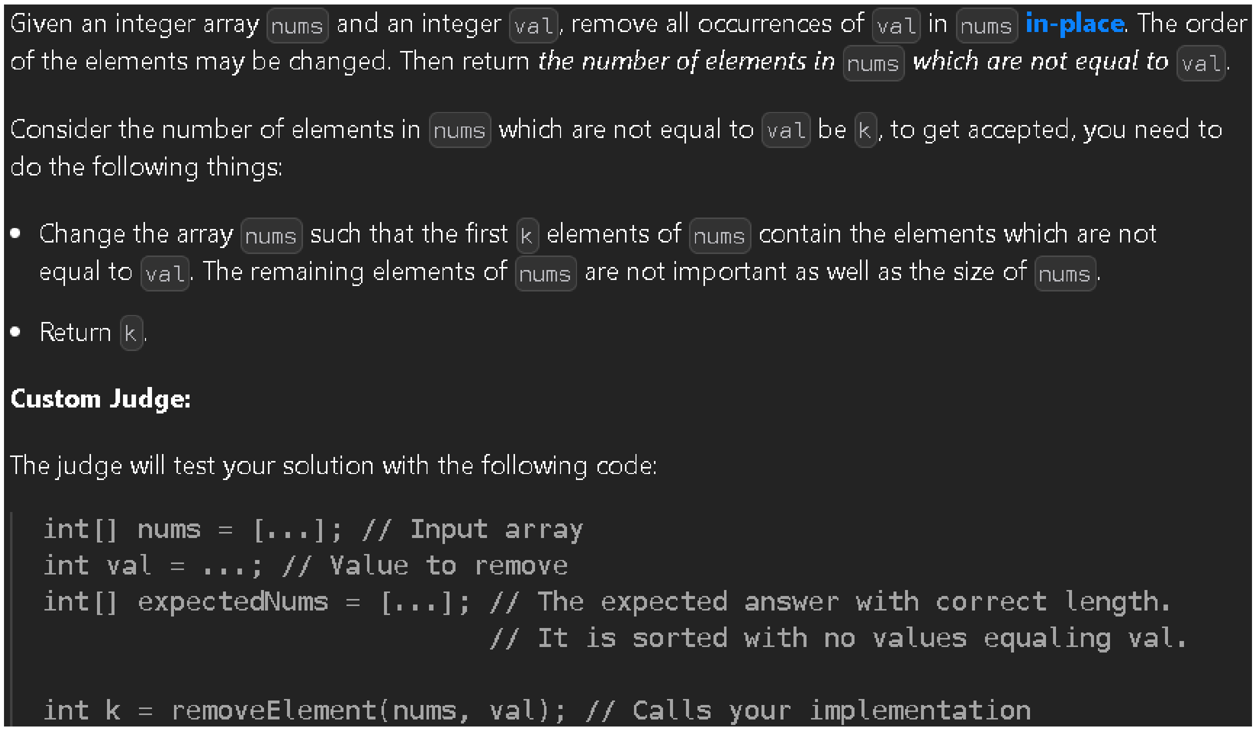


Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

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:

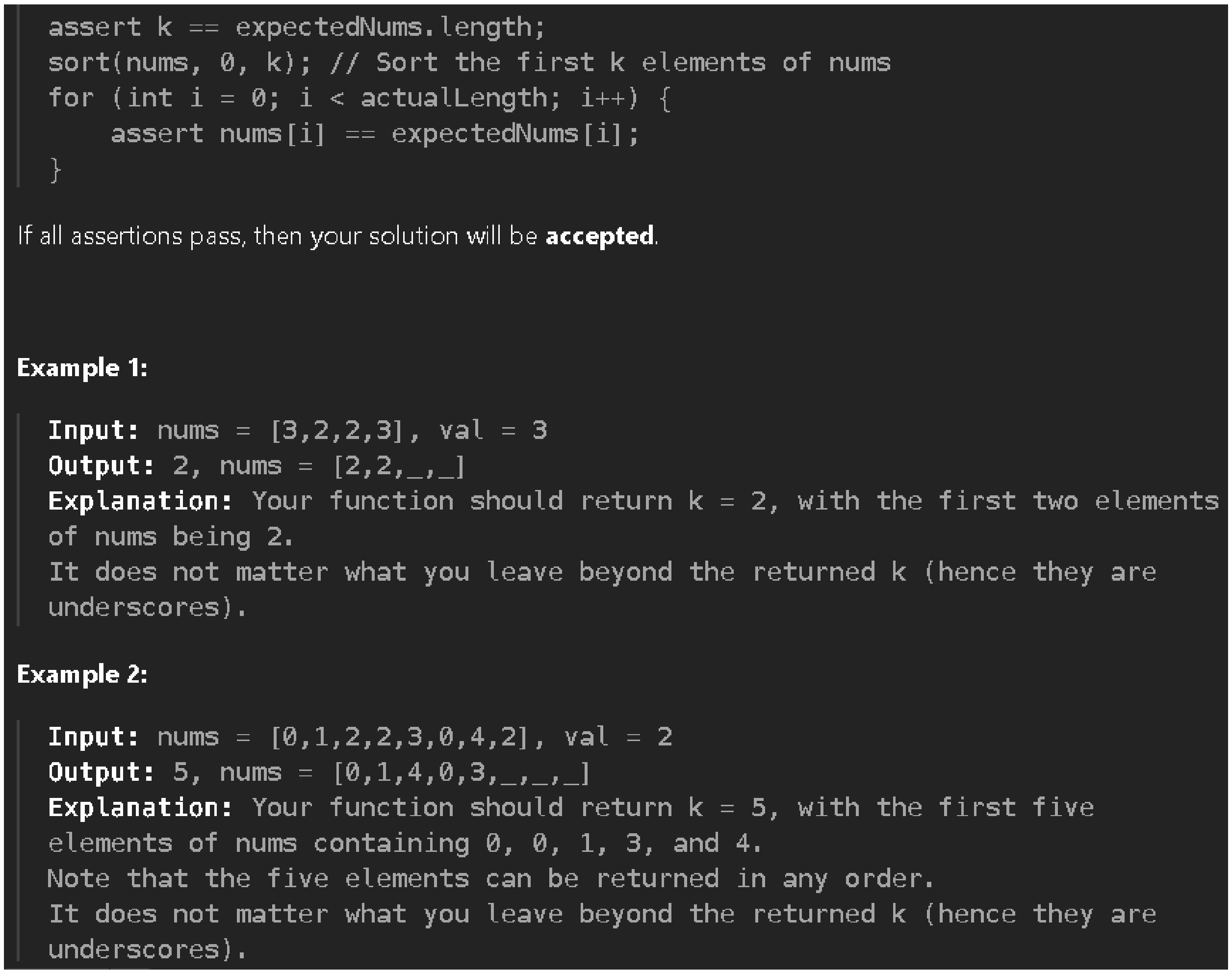
- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
                            // It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation
```



```
assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

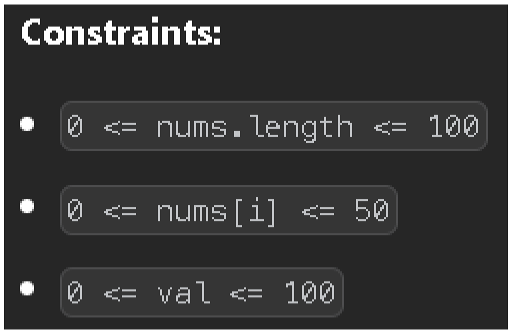
If all assertions pass, then your solution will be **accepted**.

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).
```



Constraints:

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

Python:

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        # Pointer to place the next valid element
        k = 0

        for i in range(len(nums)):
            if nums[i] != val:
                nums[k] = nums[i]  # Move non-val element to the front
        k += 1

    return k
```

JavaScript:

```
/**
 * @param {number[]} nums
 * @param {number} val
 * @return {number}
 */
var removeElement = function(nums, val) {
    let k = 0; // Pointer for placing non-val elements

    for (let i = 0; i < nums.length; i++) {
        if (nums[i] !== val) {
            nums[k] = nums[i]; // Move valid element forward
            k++;
        }
    }

    return k; // Return count of valid elements
};
```

Java:

```
class Solution {
    public int removeElement(int[] nums, int val) {
        int k = 0; // pointer for placing non-val elements

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != val) {
                nums[k] = nums[i]; // move non-val element to front
                k++;
            }
        }
        return k; // number of elements not equal to val
    }
}
```