Problem Analysis: Mutating Lists During Iteration

Programming Logic Review

Problem Statement: Question 4

Given the following code and input:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(remove_even(nums))
```

Observed Incorrect Output: `[1, 3, 5, 7, 9, 10]`
Note: The number 10 is even but remains in the list.

1 Root Cause Analysis

The error occurs because the list is being mutated (modified) while the loop is iterating over its indices.

- **Index Shifting:** When an even number is removed using `list.remove()`, all subsequent elements in the list shift one position to the left.
- **Skipping Elements:** The internal iterator moves to the next index based on the original sequence. Because the elements shifted left, the element that was originally at the "next" index is now at the current index, causing the loop to skip it entirely.
- **Residual Evens:** This behavior frequently allows even numbers (like 10 in the example) to remain in the list if they follow another removed item.

2 Recommended Solutions

To ensure the function works correctly for any list, one of the following two patterns should be used:

2.1 Solution A: List Comprehension (Recommended)

Creating a new list is the most "Pythonic" and safest approach. It avoids mutation issues entirely by building a filtered result.

```
def remove_even(numbers):
    # Returns a new list containing only odd numbers
    return [n for n in numbers if n % 2 != 0]
```

2.2 Solution B: Iterating over a Copy

If you must modify the original list object, you should iterate over a shallow copy of the list (using slice notation `[:]`).

```
def remove_even(numbers):
    for n in numbers[:]:   # Iterate over a copy
        if n % 2 == 0:
            numbers.remove(n)
    return numbers
```

3 Final Result

Using either of the solutions above yields the correct output:

```
[1, 3, 5, 7, 9]
```

By adopting these methods, all even numbers are reliably removed, and the logic remains robust regardless of the input list's contents.