Max Consecutive Ones - Documentation

1. Problem Statement

You are given a binary array nums (containing only 0 and 1). Return the maximum number of consecutive 1s in the array.

Function Signature:

```
def findMaxConsecutiveOnes(self, nums: List[int]) -> int
```

2. Intuition

We want to count the longest streak of 1s in the array. To do this efficiently:

- Iterate through the array.
- Track the current number of consecutive 1s.
- Reset the count when a 0 is encountered.
- Keep track of the maximum streak throughout.

3. Key Observations

- The count only resets when a 0 is encountered.
- The final result is the highest value of count before it resets.
- We can solve this in one pass using just two variables: one for current streak and one for the maximum.

4. Approach

- Initialize max_count and count as 0.
- Traverse through each element in the array:
    - If the element is 1, increment count and update max_count if needed.
    - If the element is 0, reset count to 0.
- Return max_count as the final answer.

5. Edge Cases

- All 1s: e.g., [1,1,1,1] → Output: 4
- All 0s: e.g., [0,0,0] → Output: 0
- Alternating: e.g., [1,0,1,0,1] → Output: 1
- Single element: [1] → Output: 1, [0] → Output: 0

6. Complexity Analysis

□ Time Complexity:

- O(n) — where n is the length of the array nums.
- Only one pass is required to compute the result.

💾 Space Complexity:

- O(1) — constant space used (only two integer variables).

7. Alternative Approaches

1. Using String Join (not optimal for large arrays):

```
def findMaxConsecutiveOnes(nums):
    return max(map(len, ''.join(map(str, nums)).split('0')))
```

- Time Complexity: O(n)
- Drawback: Less readable, involves string conversion (not ideal for interviews).

8. Test Cases

Input	Expected Output	Description
[1,1,0,1,1,1]	3	Longest streak is 3
[1,0,1,1,0,1]	2	Two 1s in the middle
[0,0,0]	0	No 1s in the array
[1,1,1,1,1]	5	Entire array is 1s
[1]	1	Single 1
[0]	0	Single 0
[1,0,1,1,1,0,1,1]	3	Multiple streaks

9. Final Thoughts

- This is a great example of a linear scan problem with an efficient constant-space solution.
- Ideal for practicing sliding windows or counters in arrays.
- Mastering this pattern helps with more advanced array-based problems.