Detect Capital Use Documentation

1. **Problem Statement**

   Given a word (a string consisting of only uppercase and lowercase English letters), determine if it uses capital letters correctly. A word is said to use capital letters correctly if it satisfies one of the following conditions:

   - All letters are uppercase (e.g., "USA").
   - All letters are lowercase (e.g., "leetcode").
   - Only the first letter is uppercase, and the rest are lowercase (e.g., "Google").

   Return True if the word follows these capitalization rules; otherwise, return False.

2. **Intuition**

   The problem requires checking simple conditions related to the casing of characters in a string. Instead of analyzing each character individually, we can utilize Python's built-in string methods to quickly evaluate each rule.

3. **Key Observations**

   - Python provides helpful methods like .isupper(), .islower(), and checking string slices, which simplify case checking.
   - The problem is not about modifying the string, just validating patterns.
   - The length of the word is up to 100, so performance isn't a major concern.

4. **Approach**

   Check for the three valid cases:

   - Case 1: word.isupper() returns True if all letters are uppercase.

- Case 2: word.islower() returns True if all letters are lowercase.
- Case 3: word[0].isupper() and word[1:].islower() checks if only the first letter is capital and the rest are lowercase.

If any of the above cases is True, return True; otherwise, return False.

5. **Edge Cases**

- A one-letter word (e.g., "A", "a") should return True in both cases.
- Words with mixed casing in incorrect positions (e.g., "FlaG") should return False.
- An empty string is not allowed by constraints (1 <= word.length <= 100), so no need to handle it.

6. **Complexity Analysis**

Time Complexity

- O(1) in practice because built-in string methods like .isupper() and .islower() are optimized and scan the string once. Given the constraint (max 100 characters), this is effectively constant time.

Space Complexity

- O(1) since no extra data structures are used—only built-in string operations.

7. **Alternative Approaches**

- Character-by-character check: Loop through the word and manually count uppercase/lowercase letters. This would be more verbose and less Pythonic.
- Regular Expressions: Use regex patterns to match valid capitalization rules. This is powerful but overkill for such a simple problem.

8. Test Cases

Test Case	Input	Output	Explanation
Test Case 1	"USA"	True	All letters are uppercase
Test Case 2	"leetcode"	True	All letters are lowercase
Test Case 3	"Google"	True	First letter uppercase, rest lowercase
Test Case 4	"FlaG"	False	Incorrect capitalization
Test Case 5	"g"	True	Single lowercase letter
Test Case 6	"G"	True	Single uppercase letter
Test Case 7	"gOOGLE"	False	Starts lowercase, then all uppercase

9. Final Thoughts

This problem is a great example of how built-in string methods in Python can make validation tasks simple and readable. While regular expressions or manual character checks can solve the problem too, they are unnecessary in this case. Always prefer the cleanest and most efficient approach when constraints are small and the language provides strong abstractions.