

# 2828. Check if a String Is an Acronym of Words

EasyArrayString

## Problem Description

The problem presents us with a list of strings called `words`, and another string `s`. We are asked to determine if `s` is an acronym for the `words`. An acronym in this context is defined as a string that can be formed by taking the first character of each string in `words`, in their respective order. For instance, given `words = ["keep", "it", "simple", "stupid"]`, the string `s = "kiss"` would be an acronym of `words` because taking the first letter of each word in order produces "kiss".

To decide whether `s` is an acronym for `words`, we need to follow this rule and check if the concatenation of the first letter of each word in the `words` array is equal to `s`.

## Intuition

To solve this problem, the intuition is straightforward: we will sequentially take the first character from each string in the `words` array and concatenate them into a new string. Then, all we need to do is to compare this newly formed string to the given string `s`.

Here's the step-by-step intuition:

- Initialize an empty string that will hold the concatenated first characters, or simply prepare to perform a comparison on-the-fly without creating a new string.
- Traverse the `words` list, and for each word, take its first character.
- As we get each first character, either add it to the initialized string or compare it directly if not storing it.
- After processing all words, we either compare the created acronym string to `s` or if we've compared on-the-fly, ensure all characters matched.

If the acronym we assembled matches `s`, we return `true`, indicating that `s` is an acronym of `words`. If not, we return `false`.

## Solution Approach

The implementation of the solution in Python is quite simple and utilizes few advanced concepts, but it demonstrates some of the core principles of programming — iteration and comparison.

- Algorithm:** The algorithm followed here is a linear scan of the `words` list to create a string made up of the first character of each word.
- Data Structures:** In the solution, the data structure used is the list (`List[str]`). This list holds the input strings that are processed.
- Patterns:** The pattern used here is a common Python idiom of using a generator expression within a string `join` method. A generator expression is an efficient way to iterate over data without the need for explicitly writing a loop or using additional memory for a list. It is commonly used for its concise syntax and because it generates elements on the fly; it is memory efficient.

The implementation in Python can be walked through step-by-step:

- String Join and Generator Expression:**

```
return "".join(w[0] for w in words) == s
```

In this line of Python code, we have `"".join(...)`, which is a method that takes an iterable and concatenates its elements separated by the string it is called on—in this case, an empty string. This means no separator between the elements.

- Generator Expression:**

The part inside the `join` method `w[0] for w in words` is a generator expression. It goes through each element `w` in the list `words`. For each element, it takes the first character `w[0]`.

- Comparison:**

Finally, the resulting string after concatenating all first characters is compared to the input string `s`. The operator `==` evaluates to `True` if both strings are equal, else `False`. This is the returned value of the `isAcronym` method.

The elegance of this solution comes from its use of Python's generator expressions and the `join` method, which both play well together to create a compact and efficient line of code. The approach is direct and specific to checking whether the acronym matches the input string without any unnecessary steps or calculations.

## Example Walkthrough

Let's walk through a small example using the solution approach to understand it better. Suppose we have the input list of words `["national", "aeronautics", "space", "administration"]` and we want to verify whether the string `s = "nasa"` is an acronym for this list of words.

Here's how the solution algorithm would process this example:

- We start with an empty string (conceptually, as per the provided solution we do not explicitly create it) to collect the first letters.
- We look at the first word `"national"`, extract its first character `"n"`, and add it to the acronym string. Our acronym string is now `"n"`.
- Next, we take the first character of the second word `"aeronautics"` which is `"a"`, and append it to our acronym string. The acronym string becomes `"na"`.
- We continue with the third word `"space"` and append its first character `"s"` to our acronym string, which now becomes `"nas"`.
- Finally, we extract the first character of the last word `"administration"` which is `"a"`, and append it to our acronym string. The acronym string is now complete and equals `"nasa"`.
- We then compare this string to the given string `s`. Since `"nasa" == "nasa"`, we return `True`.

Using the actual Python solution from the solution approach:

```
words = ["national", "aeronautics", "space", "administration"]
s = "nasa"
result = "".join(w[0] for w in words) == s
```

Upon executing the above, `result` would be `True`, showing that `s` is indeed an acronym for the provided list `words`.

The elegant part of this algorithm is that we don't even need to store the acronym string. The comparison happens on-the-fly within the generator expression as it generates each first character and is immediately followed by the string concatenation and comparison with `s`. Thus, the process is both memory- and time-efficient.

## Solution Implementation

Python

```
# Import the typing module to use the List type annotation
from typing import List

# Define the Solution class
class Solution:

    # Define the isAcronym method which takes a list of words and a string 's'.
    # It checks if the acronym formed by the first letters of the words
    # in the list matches string 's'.
    def isAcronym(self, words: List[str], s: str) -> bool:
        # Create an acronym by joining the first letter of each word in the list 'words'
        acronym = "".join(word[0] for word in words)

        # Compare the created acronym with the string 's' and return the result.
        # This will return True if they match, False otherwise.
        return acronym == s
```

Java

```
class Solution {
    // Method to check if the given string 's' is an acronym of the list of words 'words'
    public boolean isAcronym(List<String> words, String s) {
        // Construct the acronym from the first letter of each word
        StringBuilder acronym = new StringBuilder();

        // Loop through each word in the list
        for (String word : words) {
            // Append the first character of each word to the acronym
            acronym.append(word.charAt(0));
        }

        // Compare the constructed acronym to the input string 's'
        // Return true if they are equal, otherwise return false
        return acronym.toString().equalsIgnoreCase(s);
    }
}
```

C++

```
#include <vector>
#include <string>

class Solution {
public:
    // This method checks if the first characters of the words in the vector
    // form the string 's', effectively checking if 's' is an acronym of 'words'.
    bool isAcronym(std::vector<std::string>& words, std::string s) {
        std::string acronym; // Build the acronym from the first letters of 'words'
        // Iterate over each word in 'words'
        for (const auto& word : words) {
            // Ensure the word is not empty to avoid accessing out of range
            if (!word.empty()) {
                acronym += word[0]; // Append the first character of the current word
            }
        }
        // Compare the build acronym with the string 's' and return the result
        return acronym == s;
    }
};
```

TypeScript

```
/**
 * Checks if a provided string `s` is an acronym of the first letters of an array of words.
 *
 * @param {string[]} words - An array of words to derive the acronym from.
 * @param {string} s - The string to compare with the acronym.
 * @returns {boolean} - Returns true if `s` is an acronym of the first letters of `words`, otherwise returns false.
 */
function isAcronym(words: string[], s: string): boolean {
    // Map each word to its first character and join them to form the acronym
    const acronym = words.map(word => word[0]).join('');

    // Compare the formed acronym with the string `s`
    return acronym.toUpperCase() === s.toUpperCase();
}

# Import the typing module to use the List type annotation
from typing import List

# Define the Solution class
class Solution:

    # Define the isAcronym method which takes a list of words and a string 's'.
    # It checks if the acronym formed by the first letters of the words
    # in the list matches string 's'.
    def isAcronym(self, words: List[str], s: str) -> bool:
        # Create an acronym by joining the first letter of each word in the list 'words'
        acronym = "".join(word[0] for word in words)

        # Compare the created acronym with the string 's' and return the result.
        # This will return True if they match, False otherwise.
        return acronym == s
```

## Time and Space Complexity

### Time Complexity

The time complexity of the given code snippet can be analyzed as follows:

- There is a list comprehension that iterates through each word in the list `words`. This operation takes  $O(n)$  time, where `n` is the number of words in the list.
- For each word, we are accessing the first character `w[0]`, which is an  $O(1)$  operation.
- The `join` operation itself takes  $O(m)$ , where `m` is the total number of characters in the final acronym string, which in this case is the same as the number of words `n` since we are only considering the first character of each word.

Therefore, the total time complexity of the given function is  $O(n)$ , since both iterating through the words and joining the characters are linear operations with respect to the number of words.

### Space Complexity

The space complexity of the given code snippet can be analyzed as follows:

- The list comprehension creates a temporary list that holds the first character of each word, which requires  $O(n)$  space, where `n` is the number of words.
- The acronym string created by the `join` operation also requires  $O(n)$  space.

Considering this, the total space complexity of the function is  $O(n)$ , the space needed to create the temporary list and the acronym string.