# Documentation

## 1. Problem Statement

We are given a string s. We need to check whether the string can be constructed by taking a substring of it and appending multiple copies of the substring together.

## 2. Intuition

To determine if the string s is formed by repeating a substring, we can utilize the idea of string manipulation. By concatenating the string with itself, we create a situation where the repeated substring must appear in the middle of the concatenated string, but not at the very beginning or end. This provides an efficient way to check for repeated patterns.

## 3. Key Observations

- String Doubling Trick: If a string s can be formed by repeating a substring, then concatenating s with itself (s + s) will contain the string s in the middle (excluding the first and last characters). This is a key observation to solve the problem efficiently.
- String Lengths: If the length of s is less than 2, it cannot be formed by repeating any substring, so we can directly return False in such cases.

## 4. Approach

- Concatenate the String: Concatenate the string s with itself to create a new string doubled_s = s + s.
- Exclude the First and Last Characters: Remove the first and last characters from the doubled string. This is done to ensure that we check for the repetition of substrings within the string rather than trivially finding s as it appears in the doubled string.
- Check for Substring: Check if the original string s appears in the modified doubled string. If it does, return True; otherwise, return False.

5. **Edge Cases**

- Single Character String: If s has only one character, it cannot be constructed by repeating a substring. Return False.
- Empty String: If the string is empty, return False, as there are no substrings to repeat.
- No Repeating Pattern: Strings like "aba" will not have a repeating substring, so the result will be False.

6. **Complexity Analysis**

Time Complexity

- Concatenation: Concatenating the string s with itself takes O(n) time, where n is the length of the string.
- Substring Check: Checking if s is a substring of the modified string (doubled_s[1:-1]) also takes O(n) time in the worst case. Thus, the total time complexity is O(n).

Space Complexity

- Space for doubled_s: We create a new string doubled_s, which requires O(n) additional space. Thus, the space complexity is O(n).

7. **Alternative Approaches**

- Brute Force Approach: You could iterate through every possible substring and check if repeating it multiple times results in the original string. However, this approach would have a time complexity of $O(n^2)$, which is much less efficient than the concatenation trick.
- KMP (Knuth-Morris-Pratt) Algorithm: This string matching algorithm could be used to find repeated patterns efficiently, though it might involve more complex implementation compared to the doubling trick.
- Dynamic Programming: You could also use dynamic programming to store results of substrings and check for repeated patterns, but this too would be more complicated than the straightforward approach provided here.

8. **Test Cases**

Test Case 1

Input: "abab" Output: True Explanation: The string "abab" can be formed by repeating the substring "ab".

Test Case 2

Input: "aba" Output: False Explanation: The string "aba" cannot be formed by repeating any substring.

Test Case 3

Input: "abcabcabcabc" Output: True Explanation: The string "abcabcabcabc" can be formed by repeating the substring "abc" four times.

Test Case 4

Input: "xyzxyzxyzxyzxyz" Output: True Explanation: The string "xyzxyzxyzxyzxyz" can be formed by repeating the substring "xyz" five times.

Test Case 5

Input: "z" Output: False Explanation: The string "z" cannot be formed by repeating any substring.

9. **Final Thoughts**

The approach of concatenating the string with itself and checking for the existence of the original string in the middle of the doubled string provides an efficient solution to the problem. This method avoids unnecessary complexity and works within linear time and space constraints, making it an optimal choice for the problem.