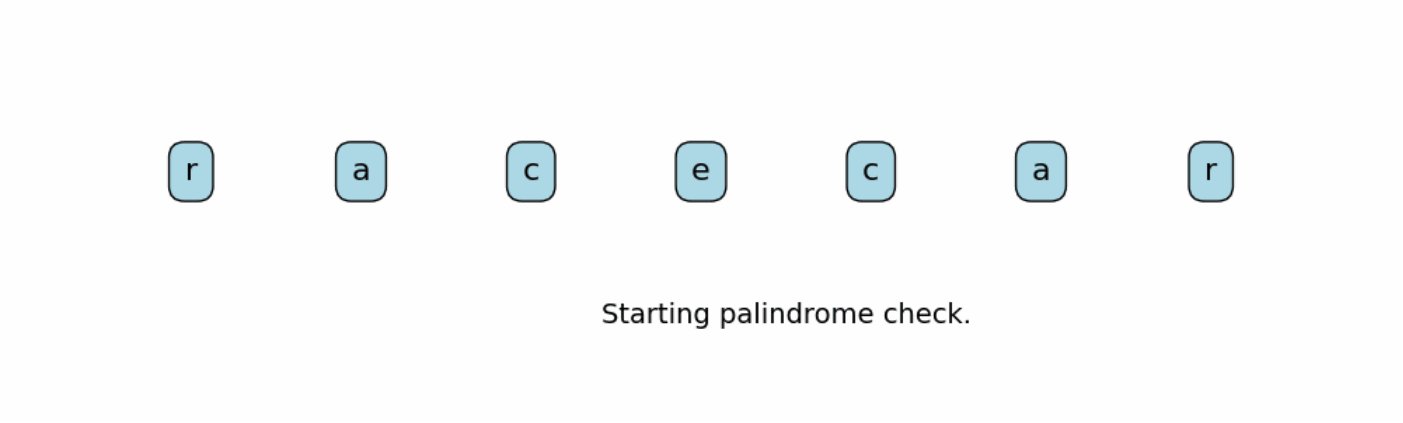
Palindrome

Palindrome

Worse approaches

- **Reversing the String:**
  You can reverse the string (using slicing like `s[::-1]` in Python) and then compare it to the original. This is very concise but uses O(n) additional space.
- **Recursion:**
  A recursive approach can check the first and last characters and then recurse on the substring that excludes them. However, this method uses extra space for the call stack and is generally less efficient.

Overall, the two-pointer approach is typically considered the best way to solve the palindrome problem due to its efficiency in both time and space.



A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as `madam` or `racecar`.

Here are ways to determine if a string is a palindrome:

- Reverse the string and it should be equal to itself.
- Have two pointers at the start and end of the string. Move the pointers inward till they meet. At every point in time, the characters at both pointers should match.

The order of characters within the string matters, so hash tables are usually not helpful.

When a question is about counting the number of palindromes, a common trick is to have two pointers that move outward, away from the middle.

- Note that palindromes can be even or odd length. For each middle pivot position, you need to check it twice - once that includes the character and once without the character.
- For substrings, you can terminate early once there is no match

```
def is_palindrome(s: str) -> bool:
    left, right = 0, len(s) - 1
    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True

# Example usage:
input_string = "racecar"
if is_palindrome(input_string):
    print(f"'{input_string}' is a palindrome.")
else:
    print(f"'{input_string}' is not a palindrome.")
```

Revision #3
Created 28 December 2024 22:46:12 by victor
Updated 17 March 2025 00:29:01 by victor