
University of Toronto - Computer Science - Image

Palindrome: Algorithm 2

Overview

To determine whether a string is a palindrome, the second algorithm we explored was:

1. Split the string into two halves.
2. Reverse the second half.
3. Compare the first half to the reversed second half.

For example, the first half of the string `"noon"` is `"no"` and the second half is `"on"`. The reverse of the second half is `"no"`. Since the first half is equal to the reverse of the second half, `"noon"` is a palindrome.

For a string with an odd length, let's consider `"racecar"`. When splitting the string into two halves, we omit the middle character, `"e"`. The first half of the string is `"rac"` and the second half of the string is `"car"`. The reverse of the second half is `"rac"`. Since the first half is equal to the reverse of the second half, `"racecar"` is a palindrome.

Finally, for a string that is not a palindrome, let's consider `"dented"`. The first half of the string is `"den"` and the second half of the string is `"ted"`. The reverse of the second half is `"det"`. Since the first half is *not* equal to the reverse of the second half, `"dented"` is *not* a palindrome.

Code

```
def is_palindrome_v2(s):
    """ (str) -> bool

    Return True if and only if s is a palindrome.

    >>> is_palindrome_v2('noon')
    True
    >>> is_palindrome_v2('racecar')
    True
    >>> is_palindrome_v2('dented')
    False
    """

    # The number of chars in s.
    n = len(s)

    # Compare the first half of s to the reverse of the second half.
    # Omit the middle character of an odd-length string.
    return s[:n // 2] == reverse(s[n - n // 2:])

def reverse(s):
    """ (str) -> str

    Return a reversed version of s.

    >>> reverse('hello')
    'olleh'
    >>> reverse('a')
    'a'
    """

    rev = ''

    # For each character in s, add that char to the beginning of rev.
```

```
    for ch in s:
        rev = ch + rev

    return rev
```

Jennifer Campbell • Paul Gries
University of Toronto
