Palindrome: Algorithm 3

Overview

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

1. Compare the first character to the last, the second to the second last, and so on.
2. Stop when the middle of the string is reached.

For example, for string "noon", we compare the first character ("n") to the last character ("n"), and the second character ("o") to the second last ("o"). Since both pairs of characters that were compared are equal, "noon" is a palindrome.

For a string with an odd length, let's consider "racecar". We compare the first character ("r") to the last character ("r"), the second character ("a") to the second last ("a"), and the third character ("c") to the third last character ("c"). The middle character, "e" does not need to be compared with anything. Since all pairs of characters that were compared are equal, "racecar" is a palindrome.

Finally, for a string that is not a palindrome, let's consider "dented". We compare the first character ("d") to the last character ("d"), the second character ("e") to the second last ("e"), and the third character ("n") to the third last character ("t"). Since *not* all pairs of characters that were compared are equal, "dented" is a *not* palindrome.

Code

```python
def is_palindrome_v3(s):      """ (str) -> bool

    Return True if and only if s is a palindrome.

    >>> is_palindrome_v3('noon')
    True
    >>> is_palindrome_v3('racecar')
    True
    >>> is_palindrome_v3('dented')
    False
    """

    # s[i] and s[j] are the next pair of characters to compare.
    i = 0
    j = len(s) - 1

    # The characters in s[:i] have been successfully compared to those in s[j:].
    while i < j and s[i] == s[j]:
        i = i + 1
        j = j - 1

    # If we exited because we successfully compared all pairs of characters,
    # then j <= i.
    return j <= i
```

Jennifer Campbell • Paul Gries
University of Toronto