Is Palindrome

In this lesson, you will learn how to determine if a string is a palindrome or not.

WE'LL COVER THE FOLLOWING

In this lesson, we will be considering how to test whether a string is a palindrome in Python. At first, we'll come up with a concise solution that takes extra space, but we'll eventually code a solution that takes a linear amount of time and a constant amount of space.

> A palindrome is a word, number, phrase, or any other sequence of characters that reads the same forward as it does backward.

Here are some examples of a palindrome:

Examples of palindromes

Was it a cat I saw

Never odd or even

radar

Live on time, emit no evil

Implementation

Let's get started with the implementations.

Solution 1

```
s = "Was it a cat I saw?"

# Solution uses extra space proportional to size of string "s"
s = ''.join([i for i in s if i.isalnum()]).replace(" ", "").lower()
print(s == s[::-1])
```

Let's discuss the above one-line solution in parts. So the `[i for i in s if i.isalnum()]` statement returns a list of all the characters which are alphanumeric. This helps us get rid of the question marks, the spaces and every character which is not alphanumeric. The `join` function in the same line then joins the elements from this list into a string which is turned into lowercase via `lower` function.

Finally, on **line 5**, `s` after modifications on **line 4**, is compared with its reversed form. If they are both equal, `True` is printed on the console

otherwise `False` will be printed on to the console. The above solution is very

concise, but as it requires extra space, we prefer another solution which will solve the problem in linear time. Let's go!

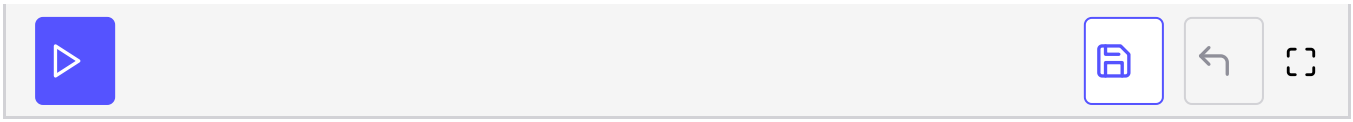
Solution 2: The Preferred Solution

```
def is_palindrome(s):
    i = 0
    j = len(s) - 1

    while i < j:
        while not s[i].isalnum() and i < j:
            i += 1
        while not s[j].isalnum() and i < j:
            j -= 1

        if s[i].lower() != s[j].lower():
            return False
        i += 1
        j -= 1
    return True

s = "Was it a cat I saw?"
print(is_palindrome(s))
```



is_palindrome(s)

`i` and `j` are two iterators which point to the first and the last character of `s` (**lines 2-3**). Next, we have a `while` loop on **line 5** which runs as long as `i` is less than `j`. `i` is supposed to go forward from the beginning of `s` while `j` has to go backward from the end of `s`. So, on **line 6** and on **line 8**, we have two `while` loops which increment or decrement the iterators accordingly. So `i` is incremented on **line** 7 if `s[i]` is not an alphanumeric and `i` is less than `j`. Similarly, `j` is decremented on **line 9** if `s[j]` is not alphanumeric and `i` is less than `j`. This will help us ignore all the characters in `s` which are not alphanumeric.

On **line 11**, we check for the palindrome property by comparing `s[i]` and `s[j]`. They have to be the same for `s` to be a palindrome. However, if `s[i]` and `s[j]` do not match, `False` is returned from the function on **line 12** to indicate an instance where the character pointed to by `i` is not the same as the character pointed to by `j`. If `s[i]` equals `s[j]`, the execution proceeds to

lines 13-14 where `i` is incremented and `j` is decremented to move to the next characters. If the condition on **line 11** stays `False` for all possible iterations of the `while` loop, `True` is returned on **line 15** to indicate that `s` is indeed a palindrome.

That sums up everything! In this lesson, you learned how to check if a string is a palindrome or not. In the next lesson, we'll explore how to find if a string is an anagram or not. See you there!