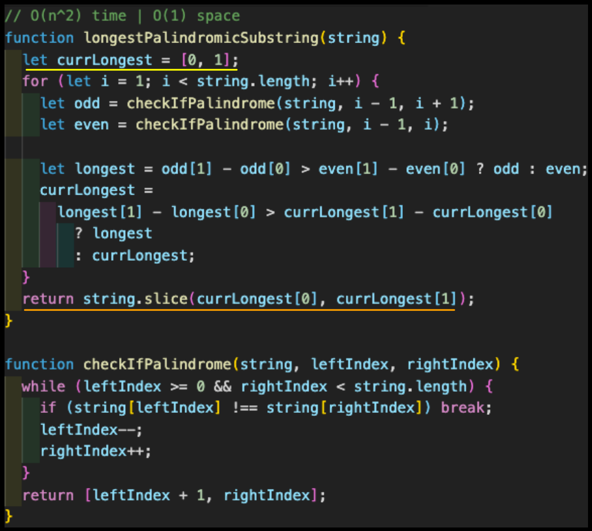
Input: "abaxyzzyxf"
Output: "xyzzyx"

Input: A string
Output: Longest palindromic substring

A palindrome is defined as a string that's written the same forward and backward

A single character string is a palindrome



```javascript
// O(n^2) time | O(1) space
function longestPalindromicSubstring(string) {
  let currLongest = [0, 1];
  for (let i = 1; i < string.length; i++) {
    let odd = checkIfPalindrome(string, i - 1, i + 1);
    let even = checkIfPalindrome(string, i - 1, i);

    let longest = odd[1] - odd[0] > even[1] - even[0] ? odd : even;
    currLongest =
      longest[1] - longest[0] > currLongest[1] - currLongest[0]
        ? longest
        : currLongest;
  }
  return string.slice(currLongest[0], currLongest[1]);
}

function checkIfPalindrome(string, leftIndex, rightIndex) {
  while (leftIndex >= 0 && rightIndex < string.length) {
    if (string[leftIndex] !== string[rightIndex]) break;
    leftIndex--;
    rightIndex++;
  }
  return [leftIndex + 1, rightIndex];
}
```

Time: $O(n^2)$ (where n is the # of elements in our string because we are iterating through the whole array and then iterating out (left and right) at each value

Space: $O(1)$ since we are not using any more space as input grows and bc we are slicing the index at the end

Note: As mentioned above, a single character string is a palindrome. So we set the first letter of the string as our current Longest palindrome. We store it as [0, 1] bc when we return the palindrome using slice, the second parameter is non-inclusive so: "abc". slice (0, 1) returns "a"

Idea: Every palindrome has a center (either odd: "aba" or even: "abba") so we iterate through each element and treat it as if its the center of a palindrome and then we store the indices of the longest palindrome

Rough work :

"aba x y z z y x f"

↑

We iterate thru each value and check if its a palindrome

Palindrome possibilities:

① abba even (4)
↑↑

② aba odd (3)
↑

currlongest = [0, 1] // first letter

ODD or Even Palindrome :

```
odd = checkIfPalindrom(string, i-1, i+1)

even = checkIfPalindrom(string, i-1, i)
```

Get longest of two

```
longest = (odd[1] - odd[0]) > (even[1] - even[0]) ? odd : even

currlongest = (longest[1] - longest[0]) > (currlongest[1] - currlong[0]) ? longest : currlongest
```

Return currlongest after iteration is done

```
return String.slice(currlongest[0], currlongest[1])
```

```
checkIfPalindrome(string, leftIdx, rightIdx)
    while (            ) {
```

z a b b a c
↑   ↑