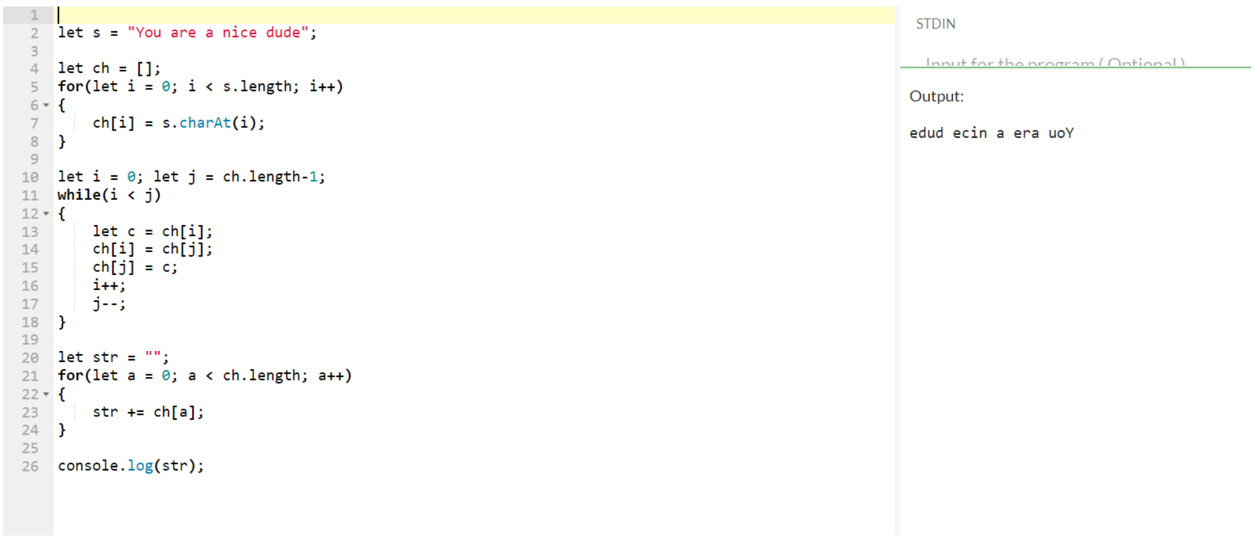
- **String Reversal:** Write a function to reverse a given string in JavaScript without using built-in reverse functions.



```javascript

let s = "You are a nice dude";

let ch = [];
for(let i = 0; i < s.length; i++)
{
    ch[i] = s.charAt(i);
}

let i = 0; let j = ch.length-1;
while(i < j)
{
    let c = ch[i];
    ch[i] = ch[j];
    ch[j] = c;
    i++;
    j--;
}

let str = "";
for(let a = 0; a < ch.length; a++)
{
    str += ch[a];
}

console.log(str);
```

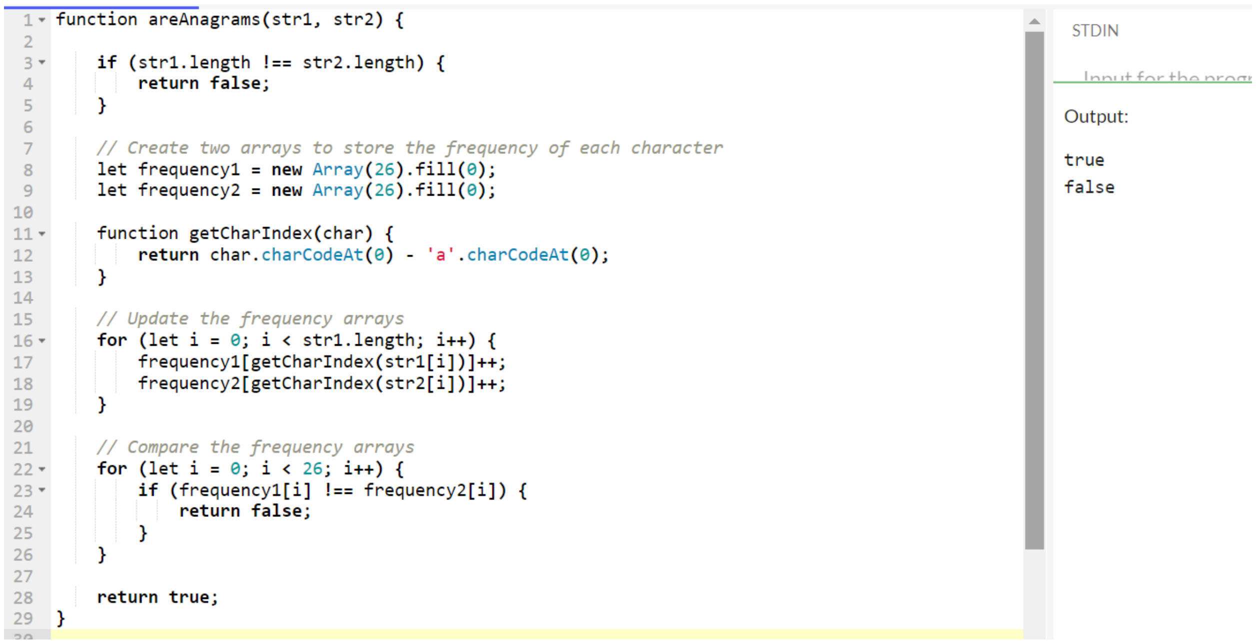
STDIN

Input for the program (Optional)

Output:

```
edud ecin a era uoY
```

- **Anagram Check:** Implement an algorithm to check if two strings are anagrams of each other (contain the same characters with the same frequency)



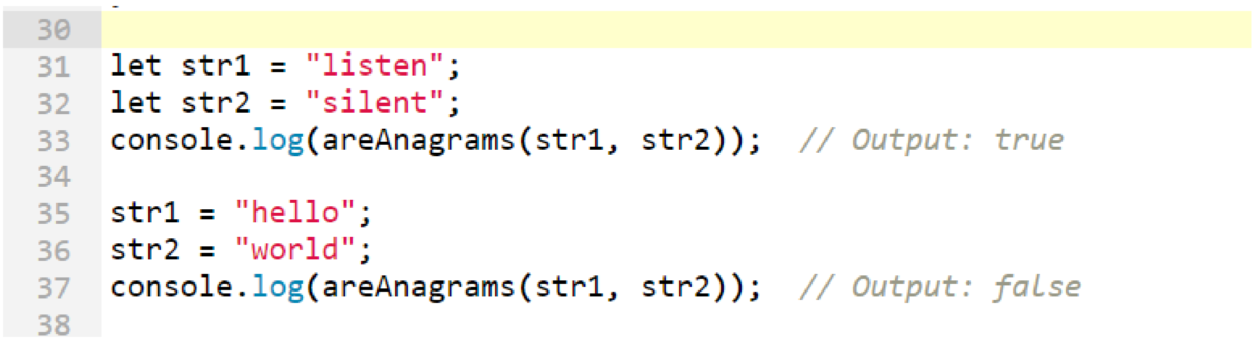
```javascript
function areAnagrams(str1, str2) {

    if (str1.length !== str2.length) {
        return false;
    }

    // Create two arrays to store the frequency of each character
    let frequency1 = new Array(26).fill(0);
    let frequency2 = new Array(26).fill(0);

    function getCharIndex(char) {
        return char.charCodeAt(0) - 'a'.charCodeAt(0);
    }

    // Update the frequency arrays
    for (let i = 0; i < str1.length; i++) {
        frequency1[getCharIndex(str1[i])]++;
        frequency2[getCharIndex(str2[i])]++;
    }

    // Compare the frequency arrays
    for (let i = 0; i < 26; i++) {
        if (frequency1[i] !== frequency2[i]) {
            return false;
        }
    }

    return true;
}
```

STDIN

Input for the progr

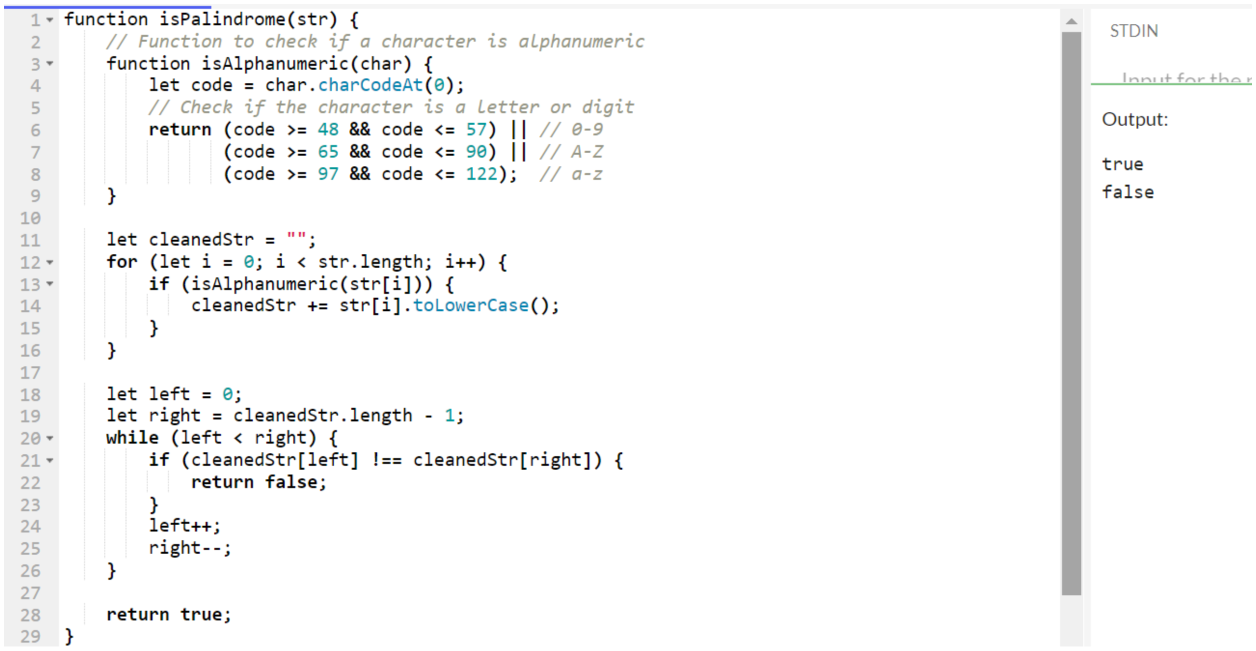
Output:

```
true
false
```



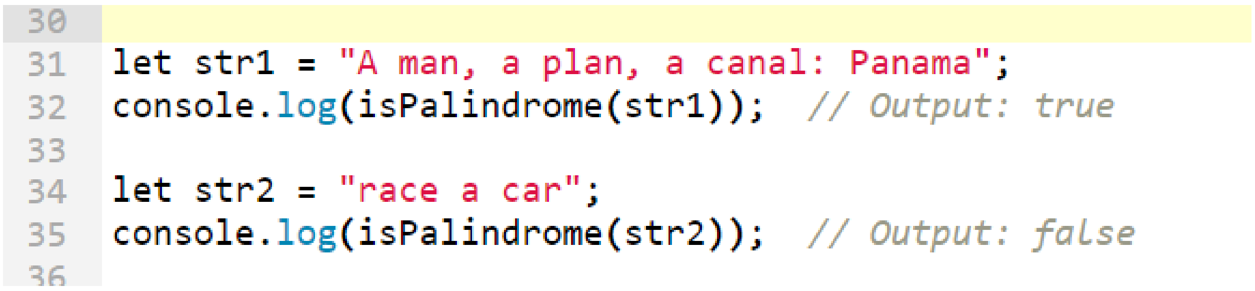
```javascript

let str1 = "listen";
let str2 = "silent";
console.log(areAnagrams(str1, str2));  // Output: true

str1 = "hello";
str2 = "world";
console.log(areAnagrams(str1, str2));  // Output: false

```

- **String Palindrome:** Create a function to check if a given string is a palindrome (reads the same forwards and backwards) while ignoring non-alphanumeric characters.



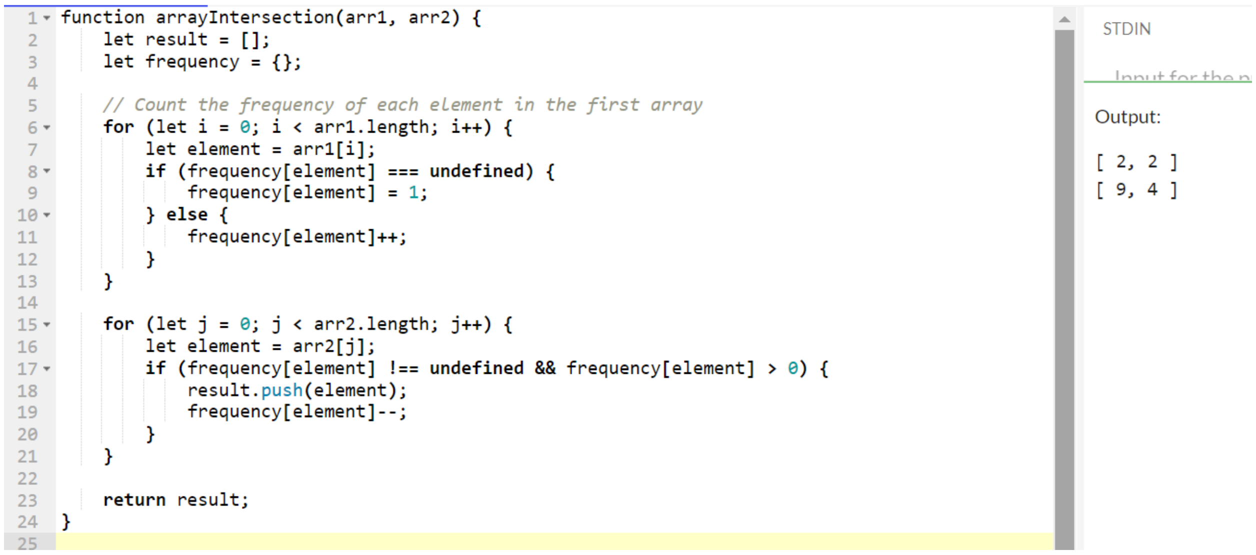
```javascript
function isPalindrome(str) {
    // Function to check if a character is alphanumeric
    function isAlphanumeric(char) {
        let code = char.charCodeAt(0);
        // Check if the character is a letter or digit
        return (code >= 48 && code <= 57) || // 0-9
               (code >= 65 && code <= 90) || // A-Z
               (code >= 97 && code <= 122);  // a-z
    }

    let cleanedStr = "";
    for (let i = 0; i < str.length; i++) {
        if (isAlphanumeric(str[i])) {
            cleanedStr += str[i].toLowerCase();
        }
    }

    let left = 0;
    let right = cleanedStr.length - 1;
    while (left < right) {
        if (cleanedStr[left] !== cleanedStr[right]) {
            return false;
        }
        left++;
        right--;
    }

    return true;
}
```

```
true
false
```



```javascript

let str1 = "A man, a plan, a canal: Panama";
console.log(isPalindrome(str1));  // Output: true

let str2 = "race a car";
console.log(isPalindrome(str2));  // Output: false

```

- **Array Intersection: Given two arrays, write a function to find their intersection (common elements).**

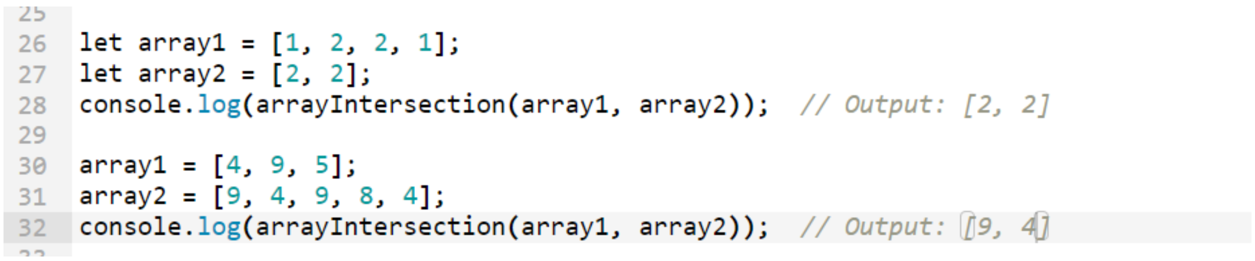


```javascript
function arrayIntersection(arr1, arr2) {
    let result = [];
    let frequency = {};

    // Count the frequency of each element in the first array
    for (let i = 0; i < arr1.length; i++) {
        let element = arr1[i];
        if (frequency[element] === undefined) {
            frequency[element] = 1;
        } else {
            frequency[element]++;
        }
    }

    for (let j = 0; j < arr2.length; j++) {
        let element = arr2[j];
        if (frequency[element] !== undefined && frequency[element] > 0) {
            result.push(element);
            frequency[element]--;
        }
    }

    return result;
}
```

STDIN

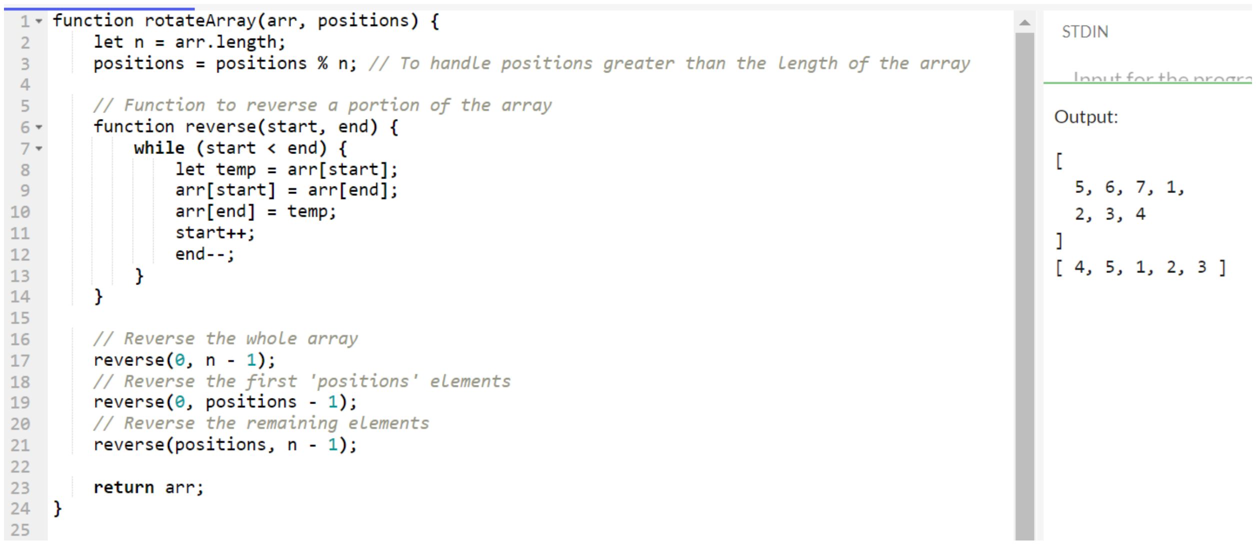
Output:

```
[ 2, 2 ]
[ 9, 4 ]
```



```javascript
let array1 = [1, 2, 2, 1];
let array2 = [2, 2];
console.log(arrayIntersection(array1, array2));  // Output: [2, 2]

array1 = [4, 9, 5];
array2 = [9, 4, 9, 8, 4];
console.log(arrayIntersection(array1, array2));  // Output: [9, 4]
```

- **Array Rotation: Implement a function to rotate an array to the right by a specified number of positions.**

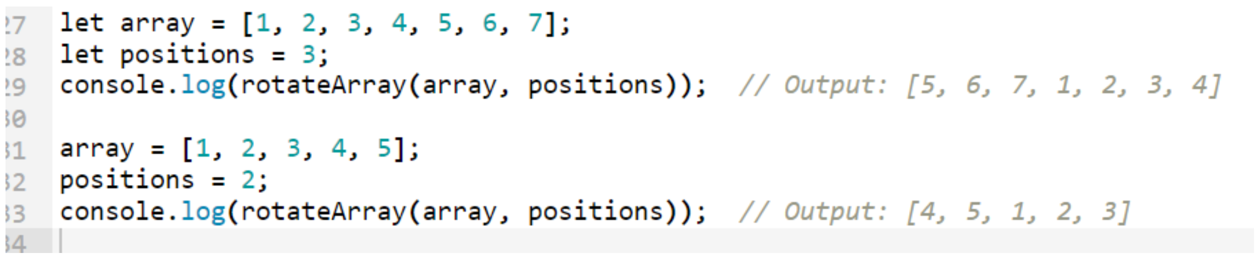


```javascript
function rotateArray(arr, positions) {
    let n = arr.length;
    positions = positions % n; // To handle positions greater than the length of the array

    // Function to reverse a portion of the array
    function reverse(start, end) {
        while (start < end) {
            let temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }

    // Reverse the whole array
    reverse(0, n - 1);
    // Reverse the first 'positions' elements
    reverse(0, positions - 1);
    // Reverse the remaining elements
    reverse(positions, n - 1);

    return arr;
}
```

STDIN

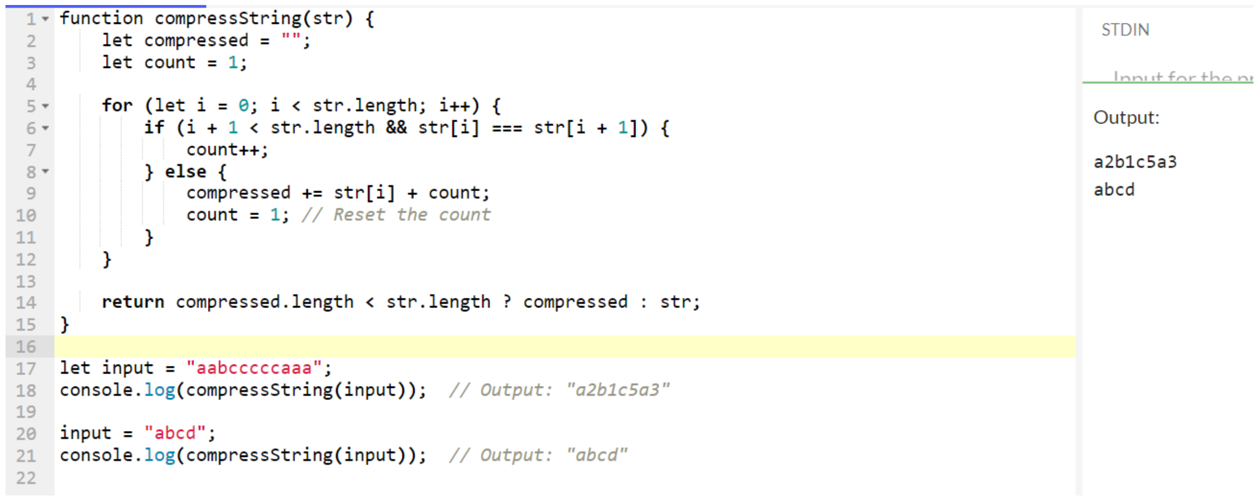
Output:

```
[
  5, 6, 7, 1,
  2, 3, 4
]
[ 4, 5, 1, 2, 3 ]
```



```
let array = [1, 2, 3, 4, 5, 6, 7];
let positions = 3;
console.log(rotateArray(array, positions));  // Output: [5, 6, 7, 1, 2, 3, 4]

array = [1, 2, 3, 4, 5];
positions = 2;
console.log(rotateArray(array, positions));  // Output: [4, 5, 1, 2, 3]
```

- **String Compression: Write a function to perform basic string compression using the counts of repeated characters. For example, "aabcccccaaa" would become "a2b1c5a3."**



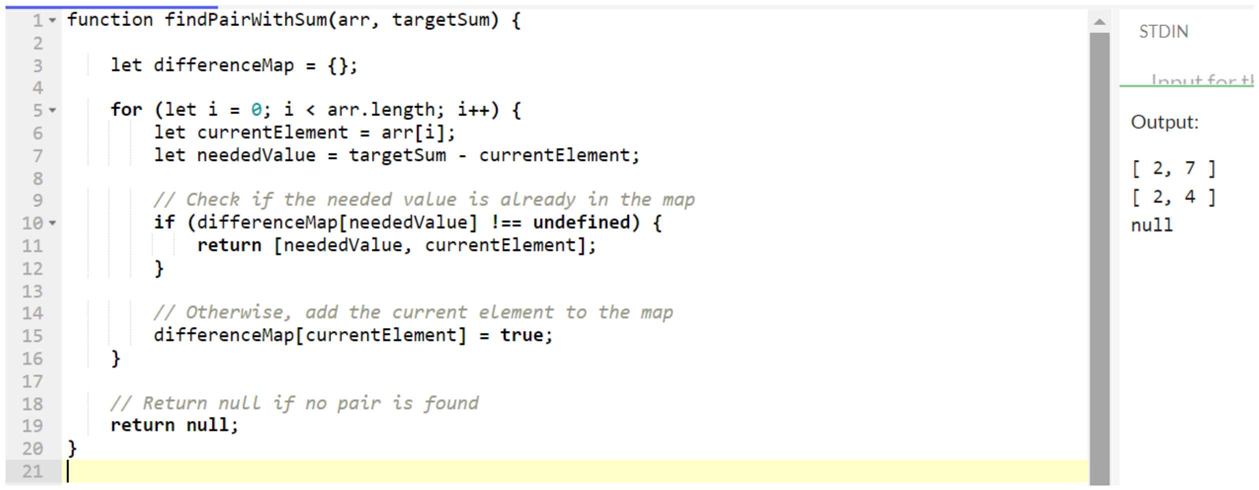
```
function compressString(str) {
    let compressed = "";
    let count = 1;

    for (let i = 0; i < str.length; i++) {
        if (i + 1 < str.length && str[i] === str[i + 1]) {
            count++;
        } else {
            compressed += str[i] + count;
            count = 1; // Reset the count
        }
    }

    return compressed.length < str.length ? compressed : str;
}

let input = "aabcccccaaa";
console.log(compressString(input));  // Output: "a2b1c5a3"

input = "abcd";
console.log(compressString(input));  // Output: "abcd"
```

STDIN

Output:

```
a2b1c5a3
abcd
```

- **Array Sum: Write an algorithm to find the pair of elements in an array that adds up to a specific target sum.**

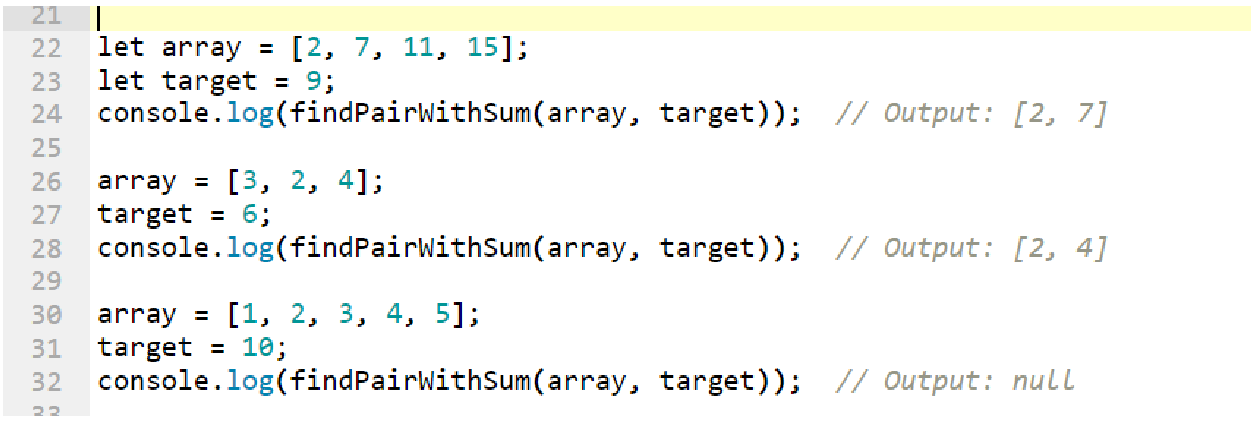


```
function findPairWithSum(arr, targetSum) {

    let differenceMap = {};

    for (let i = 0; i < arr.length; i++) {
        let currentElement = arr[i];
        let neededValue = targetSum - currentElement;

        // Check if the needed value is already in the map
        if (differenceMap[neededValue] !== undefined) {
            return [neededValue, currentElement];
        }

        // Otherwise, add the current element to the map
        differenceMap[currentElement] = true;
    }

    // Return null if no pair is found
    return null;
}
```

STDIN

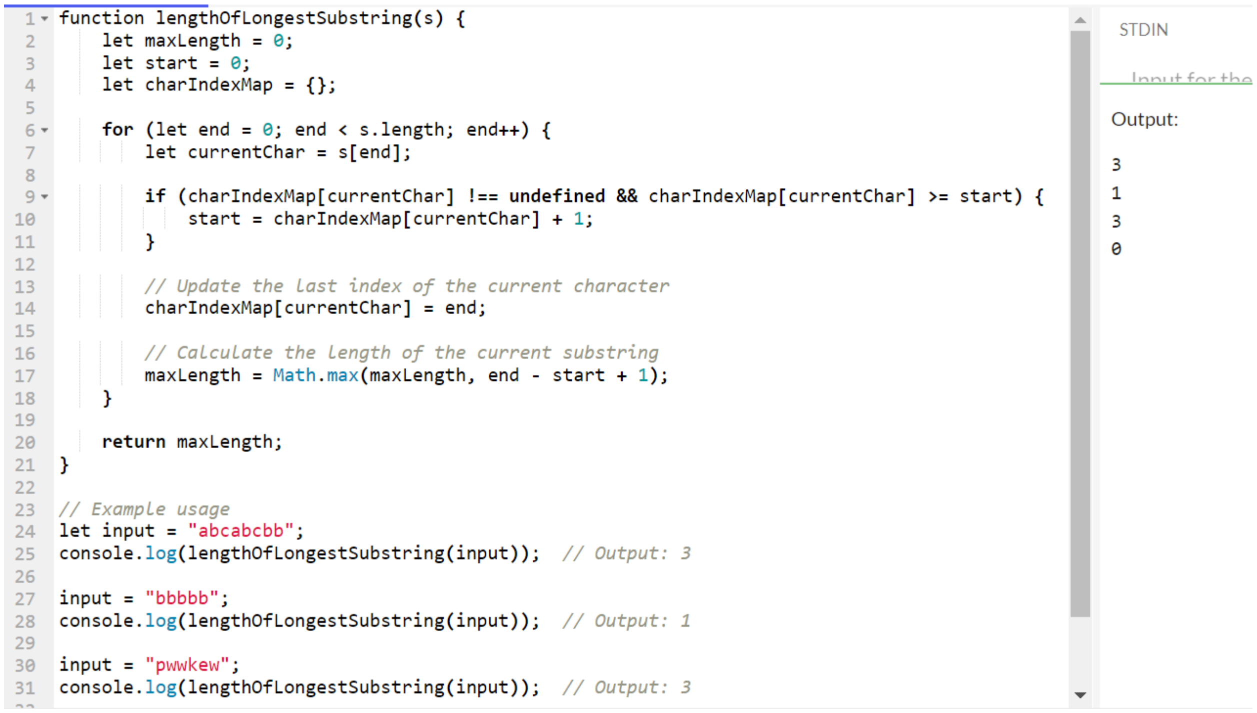
Output:

```
[ 2, 7 ]
[ 2, 4 ]
null
```



```
let array = [2, 7, 11, 15];
let target = 9;
console.log(findPairWithSum(array, target));  // Output: [2, 7]

array = [3, 2, 4];
target = 6;
console.log(findPairWithSum(array, target));  // Output: [2, 4]

array = [1, 2, 3, 4, 5];
target = 10;
console.log(findPairWithSum(array, target));  // Output: null
```

- **Longest Substring Without Repeating Characters: Write an algorithm to find the length of the longest substring without repeating characters in a given string.**



```
function lengthOfLongestSubstring(s) {
    let maxLength = 0;
    let start = 0;
    let charIndexMap = {};

    for (let end = 0; end < s.length; end++) {
        let currentChar = s[end];

        if (charIndexMap[currentChar] !== undefined && charIndexMap[currentChar] >= start) {
            start = charIndexMap[currentChar] + 1;
        }

        // Update the last index of the current character
        charIndexMap[currentChar] = end;

        // Calculate the length of the current substring
        maxLength = Math.max(maxLength, end - start + 1);
    }

    return maxLength;
}

// Example usage
let input = "abcabcbb";
console.log(lengthOfLongestSubstring(input));  // Output: 3

input = "bbbbb";
console.log(lengthOfLongestSubstring(input));  // Output: 1

input = "pwwkew";
console.log(lengthOfLongestSubstring(input));  // Output: 3
```

STDIN

Input for the

Output:

```
3
1
3
0
```