(/learn)

Remove Dupes

In this lesson, we'll discuss how to remove duplicates from a string. We'll cover two different solutions and see how ES2015 constructs can greatly simplify our code.

Remove Dupes

Instructions

Write a function that takes in a string and returns a new string. The new string should be the same as the original with every duplicate character removed.

Input: String

Output: String

Examples

```
'abcd' -> 'abcd'
'aabbccdd' -> 'abcd'
'abcddbca' -> 'abcd'
'abababcdcdcd' -> 'abcd'
```

```
function removeDupes(str) {
    // Your code here
}
```

Solution 1

```
function removeDupes(str) {
    const characters = {};
    const uniqueCharacters = [];

    for(let i = 0; i < str.length; i++) {
        const thisChar = str[i];

        if(!characters[thisChar]) {
            characters[thisChar] = true;
            uniqueCharacters.push(thisChar);
        }
    }

    return uniqueCharacters.join('');
}
```

How it Works

We create an object and an array, both of which will hold unique characters.

The for-loop goes through every character and checks if it's present in our object. If so, do nothing. It's a duplicate letter.

If not present in our object, we need to insert it into both the object and the array.

We need to maintain both the object and the array because they perform different functions for us. The object will allow us to check if we've already seen the letter instantaneously. However, objects don't keep track of the order that items are inserted in. For that, we also need to push the characters into an array.

At the end, we join all the characters in the array together and return the string.

Time

We have to process every character, so our time complexity is:

O(n)

Space

We store every character twice, in an array and in an object. This gives us `O(2n)` which simplifies to:

O(n).

As mentioned in *"Is Unique"*, JavaScript received a new data structure that allows items to be inserted only once *and* keeps track of insertion order. Using this construct, the `Set`, we can greatly simplify our code.

Solution 2

```
function removeDupes(str) {
    const uniqueCharacters = new Set(str);
    return Array.from(uniqueCharacters).join('');
}
```

That's really it. Let's unpack these two lines.

```
const uniqueCharacters = new Set(str);
```

This line creates a new set and provides the string as input. In JavaScript, a string is an iterable item, meaning that the Set breaks it up into its individual characters and inserts them one by one, in order.

A set cannot receive the same value more than once. Therefore, it'll skip over repeats in the string. The set will maintain a list of unique characters in order.

On the next line, we get an array out of the set and join the characters back together.

Time and space complexities remain the same.

← Back
Flatten Array

Next →
Highest Frequency

✓ Completed

Report an Issue