Array Methods to Change Element Ordering

In this lesson we will learn how to manipulate element ordering in JavaScript using array methods.

WE'LL COVER THE FOLLOWING

Two such methods to change element ordering are discussed below:

The `reverse()` method

You can change the order of elements in an array with the `reverse()` and `sort()` methods. Use `reverse()` to reverse the order of the elements. Invoking this method, the first item becomes the last, and the last item becomes the first and so forth for the rest of the items as the example below shows:

```
// original array
[1,3,5,7,9]
//after calling reverse() method
[9,7,5,3,1]
```

The `sort()` `compare()` and `compareDesc()` methods

The `sort()` method, as its name indicates, sorts the elements of an array in place and returns the array.

Listing 7-20 demonstrates using these methods.

Listing 7-20: Using the `reverse()`, `sort`, `compare()` and `compareDesc()` methods

```
<!DOCTYPE html>
<html>
<head>
  <title>Splice</title>
  <script>
    var word = ["h", "e", "l", "l", "o"];
    word.reverse();
    console.log(word.toString());
    var nums = [13, 4, 2, 21, 8, 31, 17];
    nums.sort();
    console.log(nums.toString());
    nums.sort(compare);
    console.log(nums.toString());
    nums.sort(compareDesc);
    console.log(nums.toString());

    function compare(v1, v2) {
      return v1 - v2;
    }

    function compareDesc(v1, v2) {
      return v2 - v1;
    }
  </script>
</head>
<body>
  Listing 7-20: View the console output
</body>
</html>
```

The output of the code is this:

console

```
o,l,l,e,h
13,17,2,21,31,4,8
2,4,8,13,17,21,31
31,21,17,13,8,4,2
```

The first line shows that `reverse()` works exactly as expected. You can use the `sort()` method in two ways. If you do not specify an argument, the elements in the array are sorted lexicographically in the ascending order according to the string conversion of each element.

Alternatively, you can pass a compare function as the argument; in this case the function is used to establish the sort order. This compare function takes

the function is used to establish the sort order. This compare function takes two arguments, let's say `v1` and `v2`, and it should return a value less than

zero if `v1` should come first, a value greater than zero if `v2` should come first, and zero if `v1` equals `v2`.

The **second line** of the output shows the array elements in dictionary order, because `sort()` was used without an argument.

The **third** and **fourth lines** show the elements in ascending, and descending order. The order of the elements in the last two invocations of `sort()` were defined by the `compare()` and `compareDesc()` functions.

The `indexOf()` and `lastIndexOf()` methods

You can search for element positions in the array with the `indexOf()` and `lastIndexOf()` methods. Both methods accept an element to search (first argument), and an optional start index.

While `indexOf()` advances from the start index toward the higher indexes, `lastIndexOf()` advances toward lower indexes.

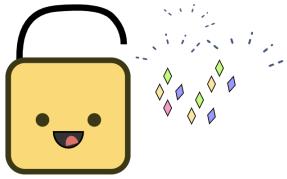
Both methods return the index of the element if found, or a value less than zero if the specified element is not in the array.

Listing 7-21 demonstrates using these methods.

Listing 7-21: Using the `indexOf()` and `lastIndexOf()` methods

```
<!DOCTYPE html>
<html>
<head>
  <title>indexOf, lastIndexOf</title>
  <script>
    var nums = [13, 4, 2, 21, 8, 2, 17];
    console.log(nums.indexOf(2));      // 2
    console.log(nums.indexOf(2, 3));   // 5
    console.log(nums.lastIndexOf(2));  // 5
    console.log(nums.lastIndexOf(11)); // -1
  </script>
</head>
<body>
  Listing 7-21: View the console output
</body>
</html>
```

Achievement unlocked! 🎉



Congratulations! You've learned how to manipulate element ordering using array methods in JavaScript.

Great work! Give yourself a round of applause! :)

In the *next lesson*, we will dive into some more advanced array operations in JavaScript.