Back to Week 1

JS | Arrays - Sort & Reverse

LESSON

Learning goals

After this lesson, you will be able to:

- use advanced array methods such are `.sort()` and `.reverse()`
- understand different ways to *compare* elements when implementing `.sort()` method.

Introduction

Continuing with our array methods, in this lesson we will review `.sort()` and `.reverse()`, and how they can help us when we work with arrays.

It is critical to truly understand these methods, and how we can make the most of them.

.sort()

JavaScript sort is by no means easy to understand, but it is one of the few array methods that is much more **difficult to do on your own**.

> According to MDN, **.sort()** method sorts the elements of an array **in place** and returns the array.

The default sort order is according to string Unicode code points.

Sorting numbers

Since `sort()` method order the values according to string Unicode, if we try to order an array of numbers from lowest to highest we can not just call the function because we will have something like this:

```
const numbers = [22, 23, 99, 68, 1, 0, 9, 112, 223, 64, 18];

numbers.sort();

console.log(numbers);
// [ 0, 1, 112, 18, 22, 223, 23, 64, 68, 9, 99 ]
```

Explain this code

> The items are **sorted as strings by default** and the reason for that is the method converts elements of the array into strings and then compares them.

👀 That is why the `.sort()` method in the previous example ordered elements according to their **Unicode code points**.

What is behind the scenes

The JavaScript `.sort()` method walks through every element in the array and compares them, based on a default `compare` function, or one that you give it.

```
function compare(a, b) {
  if (a < b) return -1; // a is less than b
  if (a > b) return 1; // a is greater than b
  if (a === b) return 0; // a equals b
}
```

Explain this code

The `compare()` function accepts two elements at a time, often referred to as `a` and `b`.

If `a - b > 0`:

- `a` is *greater than* `b`.
- Switch `b` to be before `a` in the array.

If `a - b < 0`:

- `b` is *greater than* `a`.
- Switch `a` to be before `b` in the array.

If `a - b === 0`:

- `a` and `b` are the same
- Keep `a` and `b` in the same place.

With array `[22, 23, 68, 0, 9, 1, 99]`:

1. `22 < 23`, they are in the correct order!
2. `23 < 68`, they are in the correct order!
3. `68 > 0`, switch places!
4. `68 > 9`, switch places!
5. `68 > 1`, switch places!
6. `68 < 99`, they are in the correct order!

...and repeat until it sorts the array.

As a conclusion: if we want to sort numbers in numerical order, we must include in the sort method one parameter: a compare function. The compare function, when we want it in ascending order, is as simple as switching places when the first number is higher than the second one.

```js
const numbers = [22, 23, 99, 68, 1, 0, 9, 112, 223, 64, 18];

// ES5
numbers.sort(function (a, b) {
  return a - b;
});

// ES6
numbers.sort((a, b) => a - b);

console.log(numbers);
// [ 0, 1, 9, 18, 22, 23, 64, 68, 99, 112, 223 ]
```

Explain this code

And of course, if we want to order our array in reverse numerical order, we just need to change our `compare` function.

```js
const numbers = [22, 23, 99, 68, 1, 0, 9, 112, 223, 64, 18];
// ES5
numbers.sort(function (a, b) {
  return b - a;
});

// ES6
numbers.sort((a, b) => b - a);

console.log(numbers);
// [ 223, 112, 99, 68, 64, 23, 22, 18, 9, 1, 0 ]
```

Explain this code

Sorting strings

Sorting strings is a bit trickier than numbers. Remember that by default the `.sort()` method order is according to **string Unicode** code points.

ASCending order

So, if we want to order by ascending alphabetic order, this is the only case where we don't need to provide a comparison function:

```js
const words = ['Hello', 'Goodbye', 'AA', 'First', 'A', 'a', 'Second', 'b', 'Third'];

words.sort();

console.log(words);
// ["A", "AA", "First", "Goodbye", "Hello", "Second", "Third", "a", "b"]
```

Explain this code

Notice how uppercase letters are sorted before lowercase ones, ie: `"A"` then `"a"`

DESCending order

If we want to sort in **DESC**ending alphabetical order, then we have two options.

Option 1 for DESC order: `.reverse()`

```
const words = ['Hello', 'Goodbye', 'AA', 'First', 'A', 'a', 'Second', 'b', 'Third'];

words.sort().reverse();

console.log(words);
// ["b", "a", "Third", "Second", "Hello", "Goodbye", "First", "AA", "A"]
```

Explain this code

Option 2 for DESC order: a different `compare` function

```
const words = ['Hello', 'Goodbye', 'AA', 'First', 'A', 'a', 'Second', 'b', 'Third'];

words.sort(function (a, b) {
  if (a < b) return 1; //  1 here (instead of -1 for ASC)
  if (a > b) return -1; // -1 here (instead of  1 for ASC)
  if (a === 0) return 0;
});

console.log(words);
// ["b", "a", "Third", "Second", "Hello", "Goodbye", "First", "AA", "A"]
```

Explain this code

We can also sort by different attributes, such as DESC length.

```
const words = ['b', 'a', 'Third', 'Second', 'Hello', 'Goodbye', 'First', 'AA', 'A'];

words.sort(function (a, b) {
  if (a.length < b.length) return 1; //  1 here (instead of -1 for ASC)
  if (a.length > b.length) return -1; // -1 here (instead of  1 for ASC)
  if (a.length === b.length) return 0;
});

console.log(words);
// ["Goodbye", "Second", "Hello", "First", "Third", "AA", "A", "a", "b"]
```

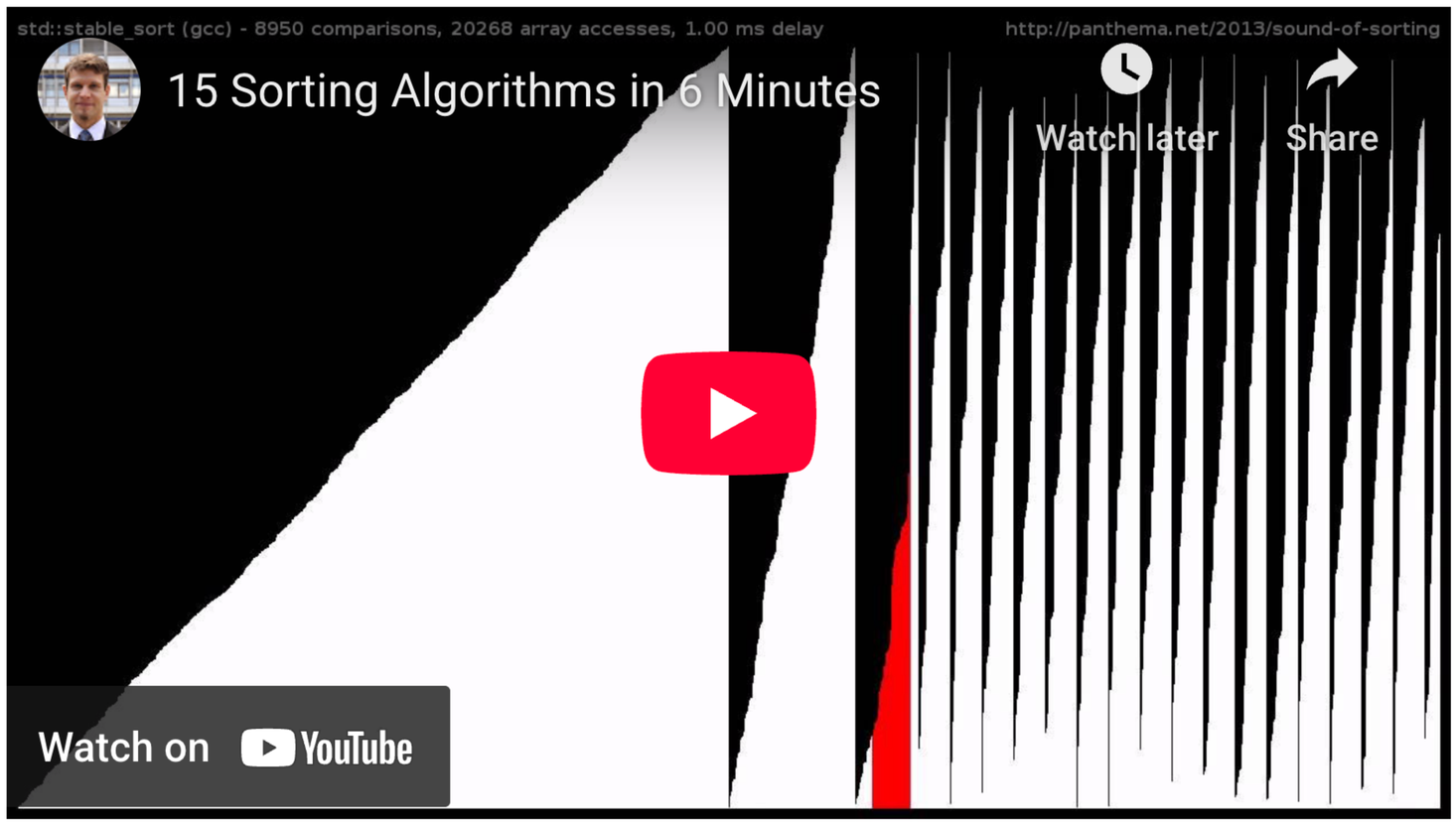
Explain this code

Sorting algorithms

There are a lot of algorithms to sort an array. Here we can see how 15 of them perform to sort some elements.

> It is important to notice that there is no ideal algorithm because it all depends on the data you are sorting.

Check this **Sorting Algorithm Animation Web** to know more about them.



.reverse()

Reverse method reverses an array in place. The first array element becomes the last, and the last array element becomes the first.

✔ **The `reverse` method transposes the elements of the calling array in place, mutating the array, and returning a reference to it.**

```
array.reverse();
```

Explain this code

Example

```
const arr1 = ['one', 'two', 'three'];
const arr2 = arr1.reverse();

console.log(arr1); // ['three', 'two', 'one'] // --> original array is mutated
console.log(arr2); // ['three', 'two', 'one']
```

Explain this code

This method is useful when you are retrieving data ordered in one way, but you want to show to the users oppositely.

Summary

In this lesson, we learned that `.sort()` method sorts the elements of an array in place. When it comes to sorting numbers, an additional *compare* function needs to be passed since sort uses *string Unicode code points*. `.reverse()` method reverses an array in place.

Manipulating arrays to get the data we want may just be the most common task you perform in programming. If you need to review these topics, do so because the stronger you are with these data structures, the more efficient you will be as a developer.

Extra Resources

- MDN Sort
- Sophisticated Sorting in JavaScript

- Array Methods - super useful

Mark as completed

PREVIOUS
← JS | Arrays - Map, Reduce & Filter

NEXT
LAB | Greatest movies of all time →