Sorting an Array

The sort() Method

The sort() method sorts an array alphabetically:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Reversing an Array

The reverse() method reverses the elements in an array.

You can use it to sort an array in descending order:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
fruits.reverse();
```

But how to do a numerical sorting?

For numerical sorting, we need a function to compare values that need to be sorted.

```
let myNumericalSort = function(a, b) {
  return a - b;
}

myNumericalSort(0, 1)
-1 # Here, the output is negative, so no change will be done in the order of the two args.

myNumericalSort(1, 0)
1 # Here, the output is positive, so two incoming numbers would be swapped in their order.

myNumericalSort(10, 2)
8 # Here, the output is positive, so two incoming numbers would be swapped in their order.

myNumericalSort(2, 10)
-8 # Here, the output is negative, so no change in order of the two args.
```

One Point to Note About Functions in JavaScript

One Point to Note About Functions in JavaScript... is that functions can be passed around as arguments to other functions (or methods).

How do we use this fact to do numerical sorting?

```
let arr = [100, 50, 20, 5 , 3]

arr.sort()

Array(5) [ 100, 20, 3, 5, 50 ]
```

This is ‘dictionary order’. 100 comes before 20 because 1 comes before 2.

And 5 comes before 50 because shorter words with same starting characters come before longer words with same starting characters.

```
arr.sort(myNumericalSort)

Array(5) [ 3, 5, 20, 50, 100 ]
```

Even shorter syntax to write the above code

These three following statements are equivalent:

```
arr.sort(myNumericalSort)
Array(5) [ 3, 5, 20, 50, 100 ]
# This was through a named function.

arr.sort(function(a, b) {
  return a - b;
});
Array(5) [ 3, 5, 20, 50, 100 ]
# This was through anonymous function.

arr.sort((a, b) => a - b);
Array(5) [ 3, 5, 20, 50, 100 ]
# This was through arrow function expression.
```

Note About Anonymous Functions in JavaScript

Named function declaration:

function *functionName (parameters) {*
  // code to be executed
}

Things to note about anonymous functions:

1. Definition of an anonymous function comes as part of a broader statement.
2. The way the definition differs from a named function is that it just lacks the ‘functionName’.

```
function myFunction(a, b) {
  return a * b;
} // This is a named function
```

```
const x = function (a, b) {return a * b};
// Here expression on right hand side is anonymous function.
```

Numeric Sort and Compare Function

By default, the sort() function sorts values as strings.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce incorrect result when sorting numbers.

You can fix this by providing a compare function:

```
const points = [40, 100, 1, 5, 25, 10];
points.sort( function (a, b) { return a – b } );
// points: [ 1, 5, 10, 25, 40, 100 ]
```

Note: compare function is an anonymous function.

Use the same trick to sort an array descending:

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b – a});
```

The way it differs from ascending order sorting is that it is returning ‘b-a’ instead of ‘a-b’.

The Compare Function

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:
function(a, b){return a - b}

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative, a is sorted before b.

If the result is positive, b is sorted before a.

If the result is 0, no changes are done with the sort order of the two values.

Example:

The compare function compares all the values in the array, two values at a time (a, b).

When comparing 40 and 100, the sort() method calls the compare function(40, 100).

The function calculates 40 - 100 (a - b), and since the result is negative (-60), the sort function will sort 40 as a value lower than 100.

Shuffling an Array

The Fisher Yates Method

The most popular correct method, is called the Fisher Yates shuffle, and was introduced in data science as early as 1938!

In JavaScript the method can be translated to this:

```
const points = [40, 100, 1, 5, 25, 10];

for (let i = points.length -1; i > 0; i--) {
  let j = Math.floor(Math.random() * (i+1));
  let k = points[i];
  points[i] = points[j];
  points[j] = k;
}
```

Using Math.max() on an Array

You can use Math.max.apply to find the highest number in an array:

```
function myArrayMax(arr) {
  return Math.max.apply(null, arr);
}
```

Math.max.apply(null, [1, 2, 3]) is equivalent to Math.max(1, 2, 3).

OR

Equivalently, using the ‘spread’ operator:

Math.max(...[1,2,3])

Using Math.min() on an Array

You can use Math.min.apply to find the lowest number in an array:

```
function myArrayMin(arr) {
  return Math.min.apply(null, arr);
}
```

Math.min.apply(null, [1, 2, 3]) is equivalent to Math.min(1, 2, 3).

OR

Equivalently, you can use the spread operator on the array to pass it directly.

Why Shuffling, Min and Max are discussed if they are not available out of the box?

This likely because these three functionality are available out of the box in Python.

Sorting Object Arrays

```
const cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];
```

Even if objects have properties of different data types, the sort() method can be used to sort the array.

The solution is to write a compare function to compare the property values:

```
cars.sort(function(a, b){return a.year - b.year});
```

Comparing string properties is a little more complex

```
cars.sort(function(a, b){
  let x = a.type.toLowerCase();
  let y = b.type.toLowerCase();
  if (x < y) {return -1;}
  if (x > y) {return 1;}
  return 0;
});
```