How to Create an Array

To create an empty array, use **[]**.

To create an array with some initial data, use **[** *item***,** *item***,** ... **]**.

Array Operators

There is one operator for JavaScript arrays, the access operator **[]**.

To get the Nth element of an array, use *array***[***n***]**.

- Indexing is zero-based, i.e., the first element is element 0.
- If the index is out of range, **undefined** is returned.

- Examples:
  - `["my", "first", "array"][0]` returns `"my"`
  - `["my", "first", "array"][2]` returns `"array"`
  - `["my", "first", "array"][3]` returns **undefined**

To set an element of the array, use assignment with **[]**.

```
var lst = ["my", "first", "array"];
lst[1] = "second";
```

This code changes the 2nd element of **lst** to `"second"`.

> Unlike many other languages, there is no error if you assign a value to an index beyond the length of the array. JavaScript will simply add value for the index. This is not recommended programming practice.

Array Methods

Arrays come with many useful methods attached.

If you want to know how many elements an array has, use *array***.length**.

If you want to add an item to the end of the array, use *array***.push(***value***)**.

> Note: this changes the array. It returns the length of the modified array.

If you want to see if some item is already in the array, use *array***.indexOf(***value***)**. This scans the array from left to right. If it finds something equal, it returns the index, zero-based. If it doesn't find anything, it returns -1.

If you want to display the elements of an array in a string, use *array***.join(***separator***)**. This concatenates every element into one string, separated by the separator string. A common separator string is `", "`.

Sorting Arrays

Quite often when you have an array of data, you want to sort it before showing it to users. For example, if you have a list of user names, you should sort the names, in ascending order. If you have a list of products, you may want to sort it by price, in descending order.

For arrays of simple values, like numbers and strings, sorting is very easy. Suppose we have this array of names:

```
var names = ["John", "Mary", "Abe", "Kevin"];
```

Then this line of code

```
names.sort();
```

will return `["John", "Mary", "Abe", "Kevin"]`.

sort() does an *in-place* sort. That means it changes the order of elements in the array to be sorted.

Suppose you have an array of student grades for a quiz, like this:

```
var grades = [ { name: "John", grade: 87 }, { name: "Mary", grade: 92 }, { name: "Abe", grade: 73 }, {
name: "Kevin", grade: 96 } ];
```

There's two ways you might want to sort this: by name or by grade. JavaScript doesn't know how to sort students, so you have to give it a **comparison** function to tell it how tell when one item should come before another. A comparison function should take two arguments and return

- a negative number, e.g., -1, if the first argument is "less" than the second, i.e., should appear before the second argument in a sorted list,
- a positive number, e.g., 1, if the first argument is "greater" than the second, i.e., should appear after the second argument in a sorted list,

- zero, if the two arguments are equal, i.e., it's arbitrary which one should appear first

For numbers, this means that simple subtraction can be used for the comparison function. The order of the subtraction determines whether the sort is ascending or descending.

Thus, to sort the list of grades by grades in *ascending* order, do this:

```
grades.sort(function(x, y) { return x.grade - y.grade; })
```

To sort the grades by grade in *descending* order, do this:

```
grades.sort(function(x, y) { return y.grade - x.grade; })
```

To sort strings, you can't subtract, but fortunately there is a function that effectively does the comparison for us, localeCompare(). **localeCompare(x, y)** will return -1 if the string **x** should appear before **y** in an ascending sort, 1 if **x** should appear after **y**, and 0 if **x** and **y** are equal.

So, to sort the grades by name in *ascending* order, do this

```
grades.sort(function(x, y) { return x.name.localeCompare(y.name); })
```

To sort the grades by name in *descending* order, do this

```
grades.sort(function(x, y) { return y.name.localeCompare(x.name); })
```

Explore

When looking for methods to use with arrays, be sure you find out

- Whether the method modifies the array; most methods do not but a few, like **push()**, do.
- What the method returns, if anything.
- Whether the method is supported by most browsers. Many new methods are being added to JavaScript, especially for arrays, but some are not available in older browsers, unless you load additional libraries.

Experiments

Some experiments with arrays are shown here. Try these in your browser's JavaScript console.

The test array deliberately has an array inside it. See how that affects the results of these functions.

```
var lst = [1, [2, 5], 2, 3, 4, 5];

lst.length
lst.push(6)
lst.indexOf(5)
lst.join("-")
```

Readings

- Mozilla Developer Network Array documentation