JavaScript Array Methods

« Previous (js_arrays.asp) Next Chapter » (js_booleans.asp)

The strength of JavaScript arrays lies in the array methods.

Converting Arrays to Strings

In JavaScript, all objects have the valueOf() and toString() methods.

The **valueOf()** method is the default behavior for an array. It returns an array as a string:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.valueOf();
```

Try it Yourself » (tryit.asp?filename=tryjs_array_valueof)

For JavaScript arrays, valueOf() and **toString()** are equal.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Try it Yourself » (tryit.asp?filename=tryjs_array_tostring)

The **join()** method also joins all array elements into a string.

It behaves just like toString(), but you can specify the separator:

Example

```
<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange","Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>
```

Try it Yourself » (tryit.asp?filename=tryjs_array_join)

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is: Popping items out of an array, or pushing items into an array.

The **pop()** method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();              // Removes the last element ("Mango") from
fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_pop)

The **push()** method adds a new element to an array (at the end):

Remember: [0] is the first element in an array. [1] is the second. Array **indexes** start with 0.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_push)

The pop() method returns the string that was "popped out".

The push() method returns the new array length.

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The **shift()** method removes the first element of an array, and "shifts" all other elements one place down.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();            // Removes the first element "Banana" from
fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_shift)

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_unshift)

The shift() method returns the string that was "shifted out".

The unshift() method returns the new array length.

Changing Elements

Array elements are accessed using their **index number**:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";        // Changes the first element of fruits to
"Kiwi"
```

Try it Yourself » (tryit.asp?filename=tryjs_array_change)

The length property provides an easy way to append a new element to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruit
```

Try it Yourself » (tryit.asp?filename=tryjs_array_add)

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];           // Changes the first element in fruits to
undefined
```

Try it Yourself » (tryit.asp?filename=tryjs_array_delete)

Using **delete** on array elements leaves undefined holes in the array. Use pop() or splice() instead.

Splicing an Array

The **splice()** method can be used to add new items to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Try it Yourself » (tryit.asp?filename=tryjs_array_splice)

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0,1);         // Removes the first element of fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_remove)

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

Sorting an Array

The **sort()** method sorts an array alphabetically:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();              // Sorts the elements of fruits
```

Try it Yourself » (tryit.asp?filename=tryjs_array_sort)

Reversing an Array

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

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

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();              // Sorts the elements of fruits
fruits.reverse();           // Reverses the order of the elements
```

Try it Yourself » (tryit.asp?filename=tryjs_array_reverse)

Numeric Sort

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**:

Example

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

Try it Yourself » (tryit.asp?filename=tryjs_array_sort2)

Use the same trick to sort an array descending:

Example

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

Try it Yourself » (tryit.asp?filename=tryjs_array_sort3)

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.

Example:

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

The function calculates 40-100, and returns -60 (a negative value).

The sort function will sort 40 as a value lower than 100.

Find the Highest (or Lowest) Value

How to find the highest value in an array?

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b-a});
// now points[0] contains the highest value
```

Try it Yourself » (tryit.asp?filename=tryjs_array_high)

And the lowest:

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
// now points[0] contains the lowest value
```

Try it Yourself » (tryit.asp?filename=tryjs_array_low)

Joining Arrays

The **concat()** method creates a new array by concatenating two arrays:

Example

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias","Linus"];
```

```
var myChildren = myGirls.concat(myBoys);     // Concatenates (joins)
myGirls and myBoys
```

Try it Yourself » (tryit.asp?filename=tryjs_array_concat)

The concat() method can take any number of array arguments:

Example

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias","Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with
arr2 and arr3
```

Try it Yourself » (tryit.asp?filename=tryjs_array_concat2)

Slicing an Array

The **slice()** method slices out a piece of an array:

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1,3);
```

Try it Yourself » (tryit.asp?filename=tryjs_array_slice)

Complete Array Reference

For a complete reference, go to our Complete JavaScript Array Reference (/jsref/jsref_obj_array.asp).

The reference contains descriptions and examples of all Array properties and methods.

« Previous (js_arrays.asp) Next Chapter » (js_booleans.asp)