Learn web development

Arrays

Previous Overview: First steps Next

In the final article of this module, we'll look at arrays — a neat way of storing a list of data **items under a single variable name. Here we look at why this is useful, then explore how to create an array, retrieve, add, and remove items stored in an array, and more besides.**

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, an understanding of what JavaScript is.
Objective:	To understand what arrays are and how to manipulate them in JavaScript.

What is an Array?

Arrays are generally described as "list-like objects"; they are basically single objects that contain multiple values stored in a list. Array objects can be stored in variables and dealt with in much the same way as any other type of value, the difference being that we can access each value inside the list individually, and do super useful and efficient things with the list, like loop through it and do the same thing to every value. Maybe we've got a series of product items and their prices stored in an array, and we want to loop through them all and print them out on an invoice, while totaling all the prices together and printing out the total price at the bottom.

If we didn't have arrays, we'd have to store every item in a separate variable, then call the code that does the printing and adding separately for each item. This would be much longer to write out, less efficient, and more error-prone. If we had 10 items to add to the invoice it would be bad enough, but what about 100 items, or 1000? We'll return to this example later on in the article.

As in previous articles, let's learn about the real basics of arrays by entering some examples into a

JavaScript console. We've provided one below (you can also open this console in a separate tab or window, or use the browser developer console if you prefer).

```
>
```

Creating an array

Arrays are constructed of square brackets, which contain a list of items separated by commas.

1. Let's say we wanted to store a shopping list in an array — we'd do something like the following. Enter the following lines into your console:

```
var shopping = ['bread', 'milk', 'cheese', 'hummous', 'noodles'];
shopping;
```

2. In this case, each item in the array is a string, but bear in mind that you can store any item in an array — string, number, object, another variable, even another array. You can also mix and match item types — they don't all have to be numbers, strings, etc. Try these:

```
var sequence = [1, 1, 2, 3, 5, 8, 13];
var random = ['tree', 795, [0, 1, 2]];
```

3. Try creating a couple of arrays of your own, before you move on.

Accessing and modifying array items

You can then access individual items in the array using bracket notation, in the same way that you accessed the letters in a string.

1. Enter the following into your console:

```
shopping[0];
// returns "bread"
```

2. You can also modify an item in an array by simply giving a single array item a new value. Try this:

```
shopping[0] = 'tahini';
shopping;
// shopping will now return [ "tahini", "milk", "cheese", "hummous", "no
```

> **Note**: We've said it before, but just as a reminder — computers start counting from 0!

3. Note that an array inside an array is called a multidimensional array. You can access an item inside an array that is itself inside another array by chaining two sets of square brackets together. For example, to access one of the items inside the array that is the third item inside the `random` array (see previous section), we could do something like this:

```
random[2][2];
```

4. Try further playing with, and making some more modifications to, your array examples before moving on.

Finding the length of an array

You can find out the length of an array (how many items are in it) in exactly the same way as you find out the length (in characters) of a string — by using the `length` property. Try the following:

```
sequence.length;
// should return 7
```

This has other uses, but it is most commonly used to tell a loop to keep going until it has looped through all the items in an array. So for example:

```
var sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++) {
  console.log(sequence[i]);
}
```

You'll learn about loops properly in a future article, but briefly, this code is saying:

1. Start looping at item number 0 in the array.

2. Stop looping at the item number equal to the length of the array. This will work for an array of any length, but in this case it will stop looping at item number 7 (this is good, as the last item — which we want the loop to cover — is 6.
3. For each item, print it out to the browser console with `console.log()`.

Some useful array methods

In this section we'll look at some rather useful array-related methods that allow us to split strings into array items and vice versa, and add new items into arrays.

Converting between strings and arrays

Often you'll be presented with some raw data contained in a big long string, and you might want to separate the useful items out into a more useful form and then do things to them, like display them in a data table. To do this, we can use the `split()` method. In it's simplest form, this takes a single parameter, the character you want to separate the string at, and returns the substrings inbetween the separator as items in an array.

Note: OK, this is technically a string method, not an array method, but I've put it in with arrays as it goes well here.

1. Let's play with this, to see how it works. First, create a string in your console:

```
var myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
```

2. Now let's split it at each comma:

```
var myArray = myData.split(',');
myArray;
```

3. Finally, try finding the length of your new array, and retrieving some items from it:

```
myArray.length;
myArray[0]; // the first item in the array
myArray[1]; // the second item in the array
myArray[myArray.length-1]; // the last item in the array
```

4. You can also go the opposite way using the `join()` method. Try the following:

```
var myNewString = myArray.join(',');
myNewString;
```

Adding and removing array items

We've not yet covered adding and removing array items — lets look at this now. We'll use the `myArray` array we ended up with in the last section. If you've not already followed that section, create the array first in your console:

```
var myArray = ['Manchester', 'London', 'Liverpool', 'Birmingham', 'Leeds', 'C
```

First of all, to add or remove an item at the end of an array we can use `push()` and `pop()` respectively.

1. Let's use `push()` first — note that you need to include one or more items that you want to add to the end of your array. Try this:

```
myArray.push('Cardiff');
myArray;
myArray.push('Bradford', 'Brighton');
myArray;
```

2. The new length of the array is returned when the method call completes. If you wanted to store the new array length in a variable, you could do something like this:

```
var newLength = myArray.push('Bristol');
myArray;
newLength;
```

3. Removing the last item from the array is as simple as running `pop()` on it. Try this:

```
myArray.pop();
```

4. The item that was removed is returned when the method call completes. You could also do this:

```
var removedItem = myArray.pop();
myArray;
removedItem;
```

`unshift()` and `shift()` work in exactly the same way, except that they work on the beginning of the array, not the end.

1. First `unshift()` — try the following commands:

```
myArray.unshift('Edinburgh');
myArray;
```

2. Now `shift()`; try these!

```
var removedItem = myArray.shift();
myArray;
removedItem;
```

Active learning: Printing those products!

Let's return to the example we described earlier — printing out product names and prices on an invoice, then totaling the prices and printing them at the bottom. In the editable example below there are comments containing numbers — Each of these marks a place where you have to add something to the code. They are as follows:

1. Below the `// number 1` comment are a number of strings, each one containing a product name and price separated by a colon. We'd like you to turn this into an array and store it in an array called `products`.
2. On the same line as the `// number 2` comment is the beginning of a for loop. In this line we currently have `i <= 0`, which is a conditional test that causes the for loop to stop immediately, because it is saying "stop when `i` is no longer less than or equal to 0", and `i` starts at 0. We'd like you to replace this with a conditional test that stops the loop when `i` is no longer less than the `products` array's length.
3. Just below the `// number 3` comment we want you to write a line of code that splits the current array item (`name:price`) into two separate items, one containing just the name and one containing just the price. If you are not sure how to do this, consult the Useful string methods article for some help, or even better, look at the Converting between strings and arrays section of this article.
4. As part of the above line of code, you'll also want to convert the price from a string to a number. If you can't remember how to do this, check out the first strings article.
5. There is a variable called `total` that is created and given a value of 0 at the top of the code. Inside the loop (below `// number 4`) we want you to add a line that adds the current item price to that total in each iteration of the loop, so that at the end of the code the correct total is printed onto the invoice. You might need an assignment operator to do this.
6. We want you to change the line just below `// number 5` so that the `itemText` variable is made equal to "current item name — $current item price", for example "Shoes — $23.99" in each case, so the correct information for each item is printed on the invoice. This is just simple string concatenation, so you should be OK with this.

- 0

Total: $0.00

```
var list =
document.querySelect
or('.output ul');
var totalBox =
document.querySelect
or('.output p');
var total = 0;
list.innerHTML = '';
totalBox.textContent
 = '';
// number 1

'Underpants:6.99'

'Socks:5.99'

'T-shirt:14.99'

'Trousers:31.99'

'Shoes:23.99';

for (var i = 0; i
<= 0; i++) { //
number 2
```

Reset Show solution

Active learning: Top 5 searches

A good use for array methods like `push()` and `pop()` is when you are maintaining a record of currently active items in a web app. In an animated scene for example, you might have an array of objects representing the background graphics currently displayed, and you might only want 50 displayed at once, for performance or clutter reasons. As new objects are created and added to the array, older ones can be deleted from the array to maintain the desired number.

In this example we're going to show a much simpler use — here we're giving you a fake search site, with a search box. The idea is that when terms are entered in the search box, The top 5 previous search terms are displayed in the list. When the number of terms gets to over 5, the last term starts being deleted each time a new term is added to the top, so the 5 previous terms are always displayed.

> **Note**: In a real search app, you'd probably be able to click the previous search terms to return to previous searches, and it would display actual search results! We are just keeping it simple for now.

To complete the app, we need you to:

1. Add a line below the `// number 1` comment that adds the current value entered into the search input to the start of the array. This can be retrieved using `searchInput.value`.
2. Add a line below the `// number 2` comment that removes the value currently at the end of the array.

Search

```
var list =
document.querySelect
or('.output ul');
var searchInput =
document.querySelect
or('.output input');
var searchBtn =
document.querySelect
or('.output
button');

list.innerHTML = '';

var myHistory = [];

searchBtn.onclick =
function() {
  // we will only
allow a term to be
entered if the
search input isn't
empty
  if
(searchInput.value
!== '') {
```

Reset Show solution

Conclusion

After reading through this article we are sure you will agree that arrays seem pretty darn useful; you'll see them crop up everywhere in JavaScript, often in association with loops being used to do the same

thing to every item in an array. We'll be teaching you all the useful basics there are to know about loops in the next module, but for now you should give yourself a clap and take a well-deserved break; you've worked through all the articles in this module!

The only thing left to do is work through this module's assessment, which will test your understanding of the articles that came before it.

See also

- Indexed collections — an advanced level guide to arrays and their cousins, typed arrays.
- `Array` — the `Array` object reference page — for a detailed reference guide to the features discussed in this page, and many more.

Previous Overview: First steps Next