Arrays

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 already be annoying, 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 browser developer console.

Creating arrays

Arrays consist of square brackets and items that are separated by commas.

1. Suppose we want to store a shopping list in an array. Paste the following code into the console:

```
const shopping = ["bread", "milk", "cheese", "hummus", "noodles"];
console.log(shopping);
```

2. In the above example, each item is a string, but in an array we can store various data types — strings, numbers, objects, and even other arrays. We can also mix data types in a single array — we do not have to limit ourselves to storing only numbers in one array, and in another only strings. For example:

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

3. Before proceeding, create a few example arrays.

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:

```
const shopping = ["bread", "milk", "cheese", "hummus", "noodles"];
console.log(shopping.length); // 5
```

Accessing and modifying array items

Items in an array are numbered, starting from zero. This number is called the item's *index*. So the first item has index 0, the second has index 1, and so on. You can access individual items

in the array using bracket notation and supplying the item's index, in the same way that you accessed the letters in a string.

1. Enter the following into your console:

```
const shopping = ["bread", "milk", "cheese", "hummus", "noodles"];
console.log(shopping[0]);
// returns "bread"
```

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

```
const shopping = ["bread", "milk", "cheese", "hummus", "noodles"];
shopping[0] = "tahini";
console.log(shopping);
// shopping will now return [ "tahini", "milk", "cheese", "hummus", "noodles" ]
```

> **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:

```
const random = ["tree", 795, [0, 1, 2]];
random[2][2];
```

4. Try making some more modifications to your array examples before moving on. Play around a bit, and see what works and what doesn't.

Finding the index of items in an array

If you don't know the index of an item, you can use the `indexOf()` method. The `indexOf()` method takes an item as an argument and will either return the item's index or `-1` if the item is not in the array:

```
const birds = ["Parrot", "Falcon", "Owl"];
console.log(birds.indexOf("Owl")); //  2
console.log(birds.indexOf("Rabbit")); // -1
```

Adding items

To add one or more items to the end of an array we can use `push()`. Note that you need to include one or more items that you want to add to the end of your array.

```
const cities = ["Manchester", "Liverpool"];
cities.push("Cardiff");
console.log(cities); // [ "Manchester", "Liverpool", "Cardiff" ]
cities.push("Bradford", "Brighton");
console.log(cities); // [ "Manchester", "Liverpool", "Cardiff", "Bradford", "Brighton" ]
```

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:

```
const cities = ["Manchester", "Liverpool"];
const newLength = cities.push("Bristol");
console.log(cities); // [ "Manchester", "Liverpool", "Bristol" ]
console.log(newLength); // 3
```

To add an item to the start of the array, use `unshift()`:

```
const cities = ["Manchester", "Liverpool"];
cities.unshift("Edinburgh");
console.log(cities); // [ "Edinburgh", "Manchester", "Liverpool" ]
```

Removing items

To remove the last item from the array, use `pop()`.

```
const cities = ["Manchester", "Liverpool"];
cities.pop();
```

```
console.log(cities); // [ "Manchester" ]
```

The `pop()` method returns the item that was removed. To save that item in a new variable, you could do this:

```
const cities = ["Manchester", "Liverpool"];
const removedCity = cities.pop();
console.log(removedCity); // "Liverpool"
```

To remove the first item from an array, use `shift()`:

```
const cities = ["Manchester", "Liverpool"];
cities.shift();
console.log(cities); // [ "Liverpool" ]
```

If you know the index of an item, you can remove it from the array using `splice()`:

```
const cities = ["Manchester", "Liverpool", "Edinburgh", "Carlisle"];
const index = cities.indexOf("Liverpool");
if (index !== -1) {
  cities.splice(index, 1);
}
console.log(cities); // [ "Manchester", "Edinburgh", "Carlisle" ]
```

In this call to `splice()`, the first argument says where to start removing items, and the second argument says how many items should be removed. So you can remove more than one item:

```
const cities = ["Manchester", "Liverpool", "Edinburgh", "Carlisle"];
const index = cities.indexOf("Liverpool");
if (index !== -1) {
  cities.splice(index, 2);
}
console.log(cities); // [ "Manchester", "Carlisle" ]
```

Accessing every item

Very often you will want to access every item in the array. You can do this using the `for...of` statement:

```
const birds = ["Parrot", "Falcon", "Owl"];

for (const bird of birds) {
  console.log(bird);
}
```

Sometimes you will want to do the same thing to each item in an array, leaving you with an array containing the changed items. You can do this using `map()`. The code below takes an array of numbers and doubles each number:

```
function double(number) {
  return number * 2;
}
const numbers = [5, 2, 7, 6];
const doubled = numbers.map(double);
console.log(doubled); // [ 10, 4, 14, 12 ]
```

We give a function to the `map()`, and `map()` calls the function once for each item in the array, passing in the item. It then adds the return value from each function call to a new array, and finally returns the new array.

Sometimes you'll want to create a new array containing only the items in the original array that match some test. You can do that using `filter()`. The code below takes an array of strings and returns an array containing just the strings that are greater than 8 characters long:

```
function isLong(city) {
  return city.length > 8;
}
const cities = ["London", "Liverpool", "Totnes", "Edinburgh"];
const longer = cities.filter(isLong);
console.log(longer); // [ "Liverpool", "Edinburgh" ]
```

Like `map()`, we give a function to the `filter()` method, and `filter()` calls this function for every item in the array, passing in the item. If the function returns `true`, then the item is added to a new array. Finally it returns the new array.

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 its simplest form, this takes a single parameter, the character you want to separate the string at, and returns the substrings between the separator as items in an array.

> **Note:** Okay, this is technically a string method, not an array method, but we'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:

```
const data = "Manchester,London,Liverpool,Birmingham,Leeds,Carlisle";
```

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

```
const cities = data.split(",");
cities;
```

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

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

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

```
const commaSeparated = cities.join(",");
commaSeparated;
```

5. Another way of converting an array to a string is to use the `toString()` method. `toString()` is arguably simpler than `join()` as it doesn't take a parameter, but more limiting. With `join()` you can specify different separators, whereas `toString()` always uses a comma. (Try running Step 4 with a different character than a comma.)

```
const dogNames = ["Rocket", "Flash", "Bella", "Slugger"];
dogNames.toString(); // Rocket,Flash,Bella,Slugger
```

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. Below the `// number 2` comment, start a `for...of()` loop to go through every item in the `products` array.
3. 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, which should be familiar to you.

7. Finally, below the `// number 6` comment, you'll need to add a `}` to mark the end of the `for...of()` loop.

Live output

- 0

Total: $0.00

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
const totalBox = document.querySelector('.output p');
let total = 0;
list.innerHTML = '';
totalBox.textContent = '';
// number 1
                'Underpants:6.99'
                'Socks:5.99'
                'T-shirt:14.99'
                'Trousers:31.99'
                'Shoes:23.99';

// number 2

  // number 3

  // number 4

  // number 5
  let itemText = 0;

  const listItem = document.createElement('li');
  listItem.textContent = itemText;
  list.appendChild(listItem);

// number 6

totalBox.textContent = 'Total: $' + total.toFixed(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 goes 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.

Live output

Search

Editable code

Press Esc to move focus away from the code area (Tab inserts a tab character).

```
const list = document.querySelector('.output ul');
const searchInput = document.querySelector('.output input');
const searchBtn = document.querySelector('.output button');

list.innerHTML = '';

const myHistory = [];
const MAX_HISTORY = 5;

searchBtn.onclick = () => {
  // we will only allow a term to be entered if the search input isn't empty
  if (searchInput.value !== '') {
    // number 1

    // empty the list so that we don't display duplicate entries
    // the display is regenerated every time a search term is entered.
    list.innerHTML = '';

    // loop through the array, and display all the search terms in the list
    for (const itemText of myHistory) {
      const listItem = document.createElement('li');
      listItem.textContent = itemText;
      list.appendChild(listItem);
    }
```

Reset Show solution

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: Arrays.

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 in order 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.

mdn web docs

This page was last modified on May 9, 2023 by MDN contributors.