# JavaScript Arrays

This workbook provides a basic introduction to using arrays in the JavaScript programming language. It assumes that you have already completed the Introduction to JavaScript workbook. This workbook follows the same conventions as the previous workbook, it includes explanations, code descriptions and code examples, which you should read. It also includes activity sections which ask you to carry out a set of actions, for example to create or modify some code. The activities reinforce what you have read, but also give you vital coding practice. The workbook is design to be read in sequence, if you skip parts it may make it harder to understand later parts.

Whilst the workbook includes everything you need to cover, you may find it useful to refer to other sources of information. The w3Schools website has a course on JavaScript, as does the codeacademy website. Our library also has books on JavaScript.

## Arrays

Often we will want to use data structures that are more sophisticated than simple strings or numbers. An array allows us to store a list of related values, without necessarily knowing how many values we will need to hold. Consider the following example.

|  |  |
| --- | --- |
| var modules;  modules = ['Maths', 'Programming', 'Web Development']; |  |

We have defined a variable called modules and assigned an array to it. The array is identified using the square braces […]. We have put three values into the array, in this case three strings. The values are separated by commas. We can refer to specific values in the array by using their index number. Note: the values are indexed starting at zero. So, modules[0] is the string 'Maths' and modules[2] is the string 'Web Development'. We can use this to access values in an array and also to change them. Consider the following example.

|  |  |
| --- | --- |
| var modules;  var oldModule;  var newModule = 'Graphics';  modules = ['Maths', 'Programming', 'Web Development'];  oldModule = modules[1];  modules[1] = newModule; |  |

If we ran this code then oldmodule would have the value 'Programming' and modules would have the value ['Maths', 'Graphics', 'Web Development'].

Arrays have a number of properties and methods associated with them, including.

.length returns the number of values in the array.

.pop() removes the last value from an array and returns that value.

.push(value/s) adds the value or values to the end of the array and returns the new array length.

.shift() removes the first value from an array and returns that value.

.unshift(value/s) adds the value or values to the start of the array and returns the new array length.

There are others, but you can look those up yourself.

One of the things we will often want to do with arrays is to move through them value by value, performing some operation on each value. One way we can do this is using a for loop.

function showModules() {

var modules = ['Maths', 'Programming', 'Web Development'];

var arrayLength = modules.length;

var modulesMessage = '';

var i;

for (i = 0; i < arrayLength; i++) {

modulesMessage = modulesMessage + modules[i] + '<br>';

}

}

A for loop uses a counter and a condition to control how many times a block of code is executed.

|  |  |
| --- | --- |
| **for (i = 0; i < arrayLength; i++) {** |  |
| **…** |  |
| **}** |  |
| for | for keyword |
| () | Contains the loop parameters |
| i = 0 | Initial counter state |
| i < arrayLength | The condition |
| i++ | Update counter at end of loop |
| {…} | Code block to be executed |
| ; | Note the semi-colons separating the loop parameters |

In a for loop the counter is set to its initial value. The condition is then checked. If the condition is true, then the code block will run. If the condition is false then the code block will not run and the loop will exit. If the code block was run, then the counter is updated and the condition is checked again.

So, in the code above, we have initialised our counter, i, to zero. This is because we are going to use the counter to index the values of the array, and the values of an array are indexed starting from zero. The condition is that i is less than the length of the array (modules.length) – the length of the array in this case is 3, and its values are modules[0], modules[1] and modules[2], so we only want to execute the code block when i is less than the length of the array. Our update, i++, increases i by one, so we can step through the array value by value. Taken as a whole, the for loop steps through the values in the array concatenating the values together into a single string with each value followed by the string '<br>' representing a line break in HTML. So we might expect the result to be each of the modules shown on a separate line when we view the output in an HTML page.

**Activity**

1. Create a new JavaScript file and associated HTML file which implements the showModules() function shown above.
2. Use .getElementById and .textContent to write the resulting string to an HTML page.
3. Consider the resulting output, but not for too long…

…because it wasn't really what we wanted. Our '<br>' strings have been displayed as text content rather than being interpreted as HTML mark-up. The clue is in the name, .textContent – we have specified that the string is to be treated as text content. This is handy, but not what we intended in this case. If you replace .textContent with .innerHTML you will achieve the desired outcome. .innerHTML will treat text that looks like HTML mark-up as HTML mark-up. This is one way we can add HTML using JavaScript. We will look at some more elegant and powerful ways later.

**Activity**

1. Update the showModules() function to use .innerHTML instead of .textContent.
2. Confirm that it works as expected.
3. Rewrite the showModules() function so that it uses two arrays, modules = ['Maths', 'Programming', 'Web Development'] and marks = [45, 98, 65]. The output displayed in the HTML page should show each module and its mark, as a percent, on a separate line, e.g. Maths 45%.
4. Rewrite the showModules() function again so that it uses a single array, modules = ['Maths', 45, 'Programming', 98,'Web Development', 65]. The output displayed in the HTML page should show each module and its mark, as a percent, on a separate line, e.g. Maths 45%.

for loops are pretty versatile, but there are other loops which we can use.

If this is our code as a for loop…

|  |  |
| --- | --- |
| **for (i = 0; i < arrayLength; i++) {…}** |  |
| i = 0 | Initial counter state |
| i < arrayLength | The condition |
| i++ | Update counter at end of loop |
| {…} | Code block |
|  |  |

…this is the same code as a while loop…

|  |  |
| --- | --- |
| **var i = 0;** |  |
| **while (i < arrayLength) {** |  |
| **…** |  |
| **i++;** |  |
| **…** |  |
| **}** |  |
| var i = 0 | Initial counter state |
| i < arrayLength | The condition |
| i++ | Update counter during the code block |
| {…} | Code block |
|  |  |

Note that the counter is initialised before the while statement and it is updated as part of the code block.

In this case the for loop and the while loop do exactly the same thing. Generally we would use a for loop when we want to step through a set of values and a while loop when we want to repeat a block of code until some condition is true.

The do while loop looks similar, but has an important difference… the condition is checked at the end of the loop not the beginning, so the code block will always execute at least once.

|  |  |
| --- | --- |
| **var i = 0;** |  |
| **do {** |  |
| **…** |  |
| **i++;** |  |
| **…** |  |
| **} while (i < arrayLength);** |  |
| var i = 0 | Initial counter state |
| i < arrayLength | The condition |
| i++ | Update counter during the code block |
| {…} | Code block |
|  |  |

This do while loop looks like it will do exactly the same as the for and while loops shown previously, but it won't in all cases! Suppose there are no items in the array, so arrayLength is zero, then the for loop and the while loop will not run the code block because they check the condition first and i is not less than zero, it is equal to zero. This means that the value of modulesMessage is still an empty string, ''.

The do while loop will run the code block once. The array is empty, so modules[0] has the value undefined. modulesMessage is concatenated with this to produce the string 'undefined'. The condition is then checked and is false, so it will not run it a second time. In this case, the value of modulesMessage is the string 'undefined'.

Typically we will use for or while loops, do while is less common.

**Activity**

1. Write a new version of the showModules() function so that it uses a while loop instead of a for loop.
2. Write a new version of the showModules() function so that it uses a do while loop instead of a for loop (even though we know it won't work correctly on empty arrays).
3. Test the for, while and do while versions of the showModules() function to verify that they behave in the same way on non-empty arrays, but differently on empty arrays.
4. Modify the do while version of the showModules() function to include an if or switch statement such that it behaves the same way as the for and while versions in the case of an empty array.

Earlier on we used an array that contained two different types of data, module names and module marks, ['Maths', 45, 'Programming', 98,'Web Development', 65]. We will often want to create and manipulate complex data structures which hold a number of different pieces of data. In JavaScript we can use objects which contain properties (variables) which can have values. These objects can also contain methods (functions) which allow the object to be manipulated or interrogated.

|  |  |
| --- | --- |
| **var student = new Object();** |  |
| **student.firstName = 'Chris';**  **student.lastName = 'Jones';**  **student.course = 'BSc Computing';** |  |
| **student.moduleMarks =** **['Maths', 45, 'Programming', 98,'Web Development', 65];** |  |
| new | Keyword |
| Object() | Constructor function for objects |

This code create a new object called student. It adds a property called firstName to student and assigns the value 'Chris' to it. It adds a property called lastName to student and assigns the value 'Jones' to it. It adds a property called course to student and assigns the value 'BSc Computing' to it. It adds a property called moduleMarks to student and assigns the array ['Maths', 45, 'Programming', 98,'Web Development', 65] to it.

Once we have created our object we can retrieve its property values, for example:

studentFullName = student.firstName + ' ' + student.lastName;

We can also change its property values, for example:

student.firstName = 'Christopher';

student.moduleMarks[1] = 77;

If we assign a value to a property that doesn't exist, that property will be added to the object. For example:

student.enrollmentNumber = 948464;

would add the property enrollmentNumber to student and assign the value 948464 to it.

We can remove a property from an object using the delete keyword, for example:

delete student.enrollmentNumber;

Note that this deletes the property and its value, it does not delete the entire object.

If we just want to clear the value of a property, rather than delete it, we can just set its value to undefined, for example:

student.course = undefined;

Alternatively we could set it to an empty string, so long as an empty string is not a valid value for the property.

**Activity**

1. Write a new version of the showModules() function that holds the student data in an object as shown above.
2. The output displayed in the HTML page should show the following:

**Student Name:** Chris Jones

**Course:** BSc Computing

**Results:**

Maths 45%

Programming 98%

Web Development 65%

At the moment we can only use our student object to hold a single student's details which is rather limiting. It might be more useful to create a reusable template for the student object which we could then use to create a new student object each time we needed one. First we create a function for producing a student object:

|  |  |
| --- | --- |
| **function Student(firstName, lastName, course, moduleMarks) {** |  |
| **this.firstName = firstName;**  **this.lastName = lastName;**  **this.course = course;** |  |
| **this.moduleMarks = moduleMarks;**  **}** |  |

Note the use of the this keyword to indicate that the property belongs to the object that *this* function creates. Typically we will use a capital letter at the beginning of object constructor function names to differentiate them from other functions.

Now that we have our function we can create new objects from it, using the new keyword, for example:

var student1 = new Student('Tom', 'Jenkins', 'BSc Computing', ['Maths', 55, 'Programming', 87,'Web Development', 65]);

var student2 = new Student('Mary', 'Evans', 'BSc Computer Networking', []);

We can then access an objects values in the usual way, for example:

student2FullName = student2.firstName + ' ' + student2.lastName;

Objects can also contain methods (functions) which allow the object to be manipulated or interrogated. This is useful when there are actions which we will want to perform many times on an object or set of objects. In our example, we might want a method which returns the students full name. We can include a method to do this our object template:

|  |  |
| --- | --- |
| **function Student(firstName, lastName, course, moduleMarks) {** |  |
| **this.firstName = firstName;**  **this.lastName = lastName;**  **this.course = course;** |  |
| **this.moduleMarks = moduleMarks;**  **this.fullName = function() {**  **return this.firstName + ' ' + this.lastName;**  **}**  **}** |  |

So, our Student object now has a method we can call, for example:

student2FullName = student2.fullName();

**Activity**

1. Write a new version of the showModules() function that uses the student object constructor as shown above. Create objects for Chris Jones, Tom Jenkins and Mary Evans using the data shown above.
2. The output displayed in the HTML page should show the following:

**Student Name:** Chris Jones

**Course:** BSc Computing

**Results:**

Maths 45%

Programming 98%

Web Development 65%

**Student Name:** Tom Jenkins

**Course:** BSc Computing

**Results:**

Maths 55%

Programming 87%

Web Development 65%

**Student Name:** Mary Evans

**Course:** BSc Computer Networking

**Results:**

No results found

1. Add a new function fullResults() to your student object which returns a string containing the correctly formatted results for the student. Rewrite your showModules() function accordingly.

As we have seen earlier, we can store arrays in objects. We can also store objects in arrays.

At the moment each of our student objects is stored in a variable, but it is rather untidy to have to create a new variable every time we want to add a student, especially if we don't know how many students we will need to add. If we hold them in an array it is easy to add, delete, and modify them.

Adding objects to an array is simple, for example:

var studentArray = [];

studentArray[0] = new Student('Tom', 'Jenkins', 'BSc Computing', ['Maths', 55, 'Programming', 87,'Web Development', 65]);

We can then access properties, for example:

el.innerHTML = studentArray[0].course;

or:

el.innerHTML = studentArray[0].moduleMarks[2];

We can also access methods, for example:

el.innerHTML = studentArray[0].fullName();

Now that we have our students in an array we can use a loop to go through all the student records in turn.

**Activity**

1. Write a new version of the showModules() function that uses the student object constructor but stores the objects in an array, as shown above. Create objects for Chris Jones, Tom Jenkins and Mary Evans using the data shown earlier. Use a loop and the fullName() and fullResults() functions to generate the output.
2. The output displayed in the HTML page should show the following:

**Student Name:** Chris Jones

**Course:** BSc Computing

**Results:**

Maths 45%

Programming 98%

Web Development 65%

**Student Name:** Tom Jenkins

**Course:** BSc Computing

**Results:**

Maths 55%

Programming 87%

Web Development 65%

**Student Name:** Mary Evans

**Course:** BSc Computer Networking

**Results:**

No results found

1. Add a new function fullRecord() to your student object which returns a string containing the complete correctly formatted student record (such as for Mary Evans above). Rewrite your showModules() function accordingly.

This workbook has provided a basic introduction to Arrays in the JavaScript programming language. It has not covered all the details, so you may well need to do some additional reading or look up specific details when you need them.

This workbook has focussed mainly on the Javascript language itself, the relationship to HTML pages has been rather limited. In the next workbook, we will look in more detail at the Document Object Model (the DOM), and how we can manipulate it using JavaScript.