

Introduction to JavaScript Fundamental

What is JavaScript?

HTML controls the content of the webpage and CSS controls the styling of the webpage. JavaScript controls the behaviour/interactivity of a webpage based on user inputs such as clicking, hovering, scrolling etc.

Unlike HTML and CSS which are markup and styling language, JavaScript is a programming language and was initially a front-end language. JavaScript has developed over the years and is also now used in both front-end and back-end development for web applications. There are many JavaScript framework, however, in this tutorial we will only look at vanilla JavaScript to understand the fundamentals first before diving into frameworks as all frameworks uses vanilla JavaScript. A final point — JavaScript is not the same as Java.

How to Include JavaScript on a Website?

To write JavaScript in the HTML we would use the `<script></script>` tags and everything within the tags will be seen as JavaScript.

In the past you needed to put the type in our opening script tag for example `<script type="text/javascript"></script>` however, we no longer need to do this for HTML5.

JavaScript can be embedded in the HTML document and there are two places where we can include our JavaScript code. Where we place the JavaScript in our HTML document will be determined by what the JavaScript is trying to do and the order for loading the code.

JavaScript in the `<head></head>` tags — this will load the JavaScript first before the rest of the website content. You should only do this if it is crucial for the JavaScript to be run before the rest of the website/webapp.

JavaScript in the `<body></body>` tags — this will load the JavaScript after the HTML & CSS content load. The best practice is to place the JavaScript at the bottom of the `<body>` tags because it allows the website to load everything else first before the JavaScript which will improve the loading of your website.

The placing of the JavaScript within the `<head>` or `<body>` of the HTML file will depend on the importance of loading the JavaScript first before the HTML content or after. For example: We will add JavaScript in the `<body>` of the HTML because it needs to modify the behaviour of an element which needs to be loaded first before the JavaScript.

JavaScript can be added in a separate JavaScript file for example `index.js` — the `.js` extension indicates to the browser that the file is a JavaScript file (*and we do not need the `<script>` tag within the `.js` file*). However, we will need to provide a link in the HTML to locate the JavaScript file as we do for any other linked files such as CSS/Weblinks/HTML pages etc. for example `<script src="index.js"></script>`

Again the link can be placed in the `<head>` or `<body>` of the HTML file depending on the importance of the load for the JavaScript file i.e. before the HTML content or after.

How to Output JavaScript in the web browser?

There are three different ways to output JavaScript on a webpage or web browser and we will explore the three methods within this section.

1. Alert Boxes

When we alert something in the browser, what we are doing is writing some text that will appear in an alert box in the browser.

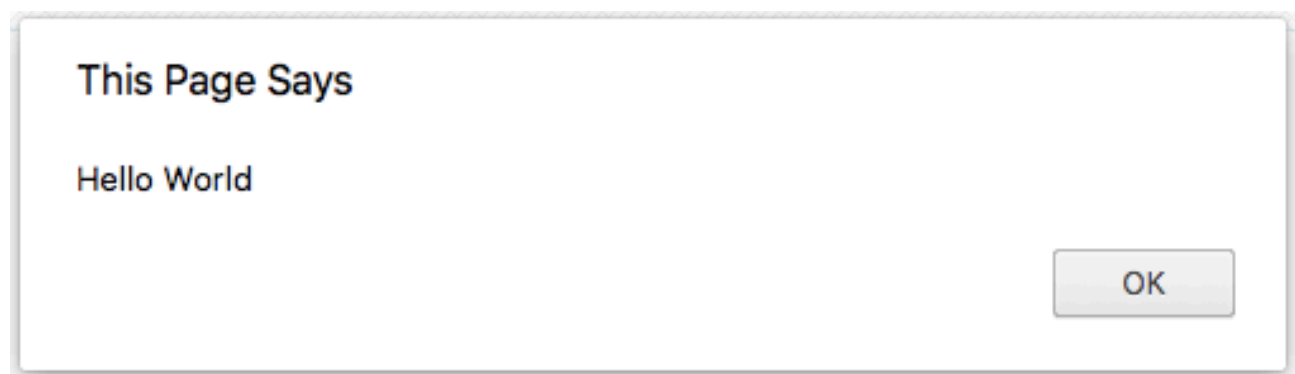
To use the JavaScript alert function we will use the syntax:

```
alert("");
```

Within the quotation marks we will write our text (a text is a string and a string is a data type that is indicated by single/double quotation marks — we will learn more about data types in the later section). This will generate an Alert Box within the browser - see screenshots below.



```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Introduction to Javascript</title>
8    </head>
9
10   <body>
11
12     <script>
13       alert("Hello World");
14     </script>
15   </body>
16 </html>
```



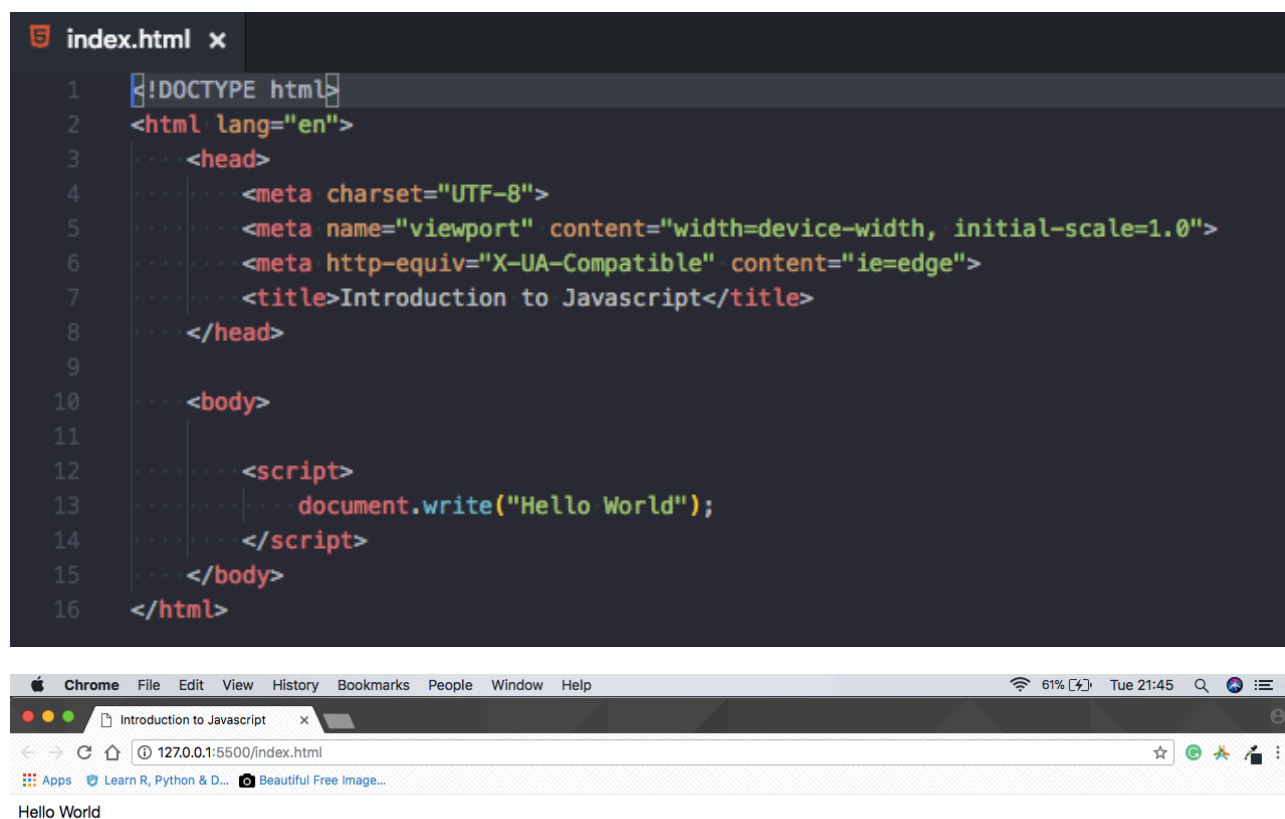
2. Writing to the Browser

To write text directly in the browser without an alert box we can use the `document.write` function to display text within the browser webpage.

To use the JavaScript `document.write` method we will use the syntax:

```
document.write("");
```

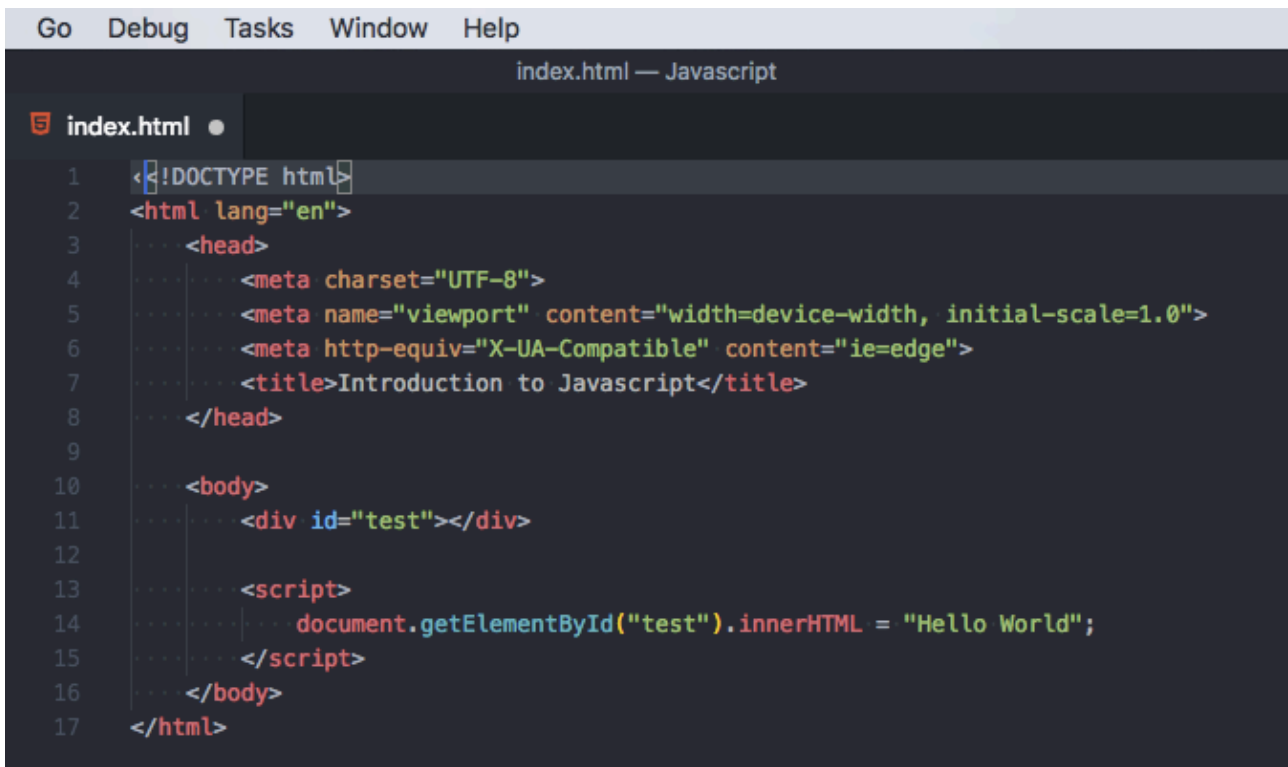
This will write the text string directly in the webpage - see screenshots below.



The document refers to the html page and we are using the write method to write to the document. We can write to specific elements within our documents by referring to the id of the element using the `getElementById` — for example:

```
<div id="test"></div>
<script>
  document.getElementById("test").innerHTML = "";
</script>
```

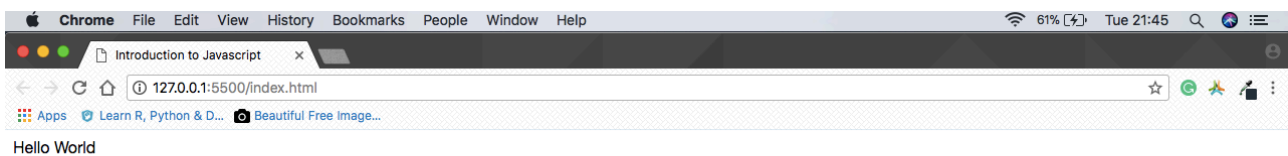
The inner HTML will add the text within the opening and closing `<div>` tags — see screenshot below.



```
Go  Debug  Tasks  Window  Help

index.html — Javascript

index.html
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1.0">
6          <meta http-equiv="X-UA-Compatible" content="ie=edge">
7          <title>Introduction to Javascript</title>
8      </head>
9
10     <body>
11         <div id="test"></div>
12
13         <script>
14             document.getElementById("test").innerHTML = "Hello World";
15         </script>
16     </body>
17 </html>
```

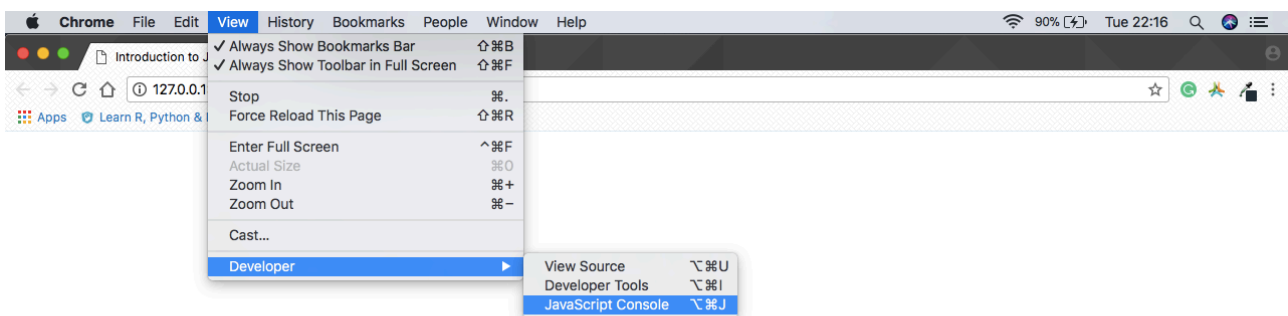


3. Logging to the Console

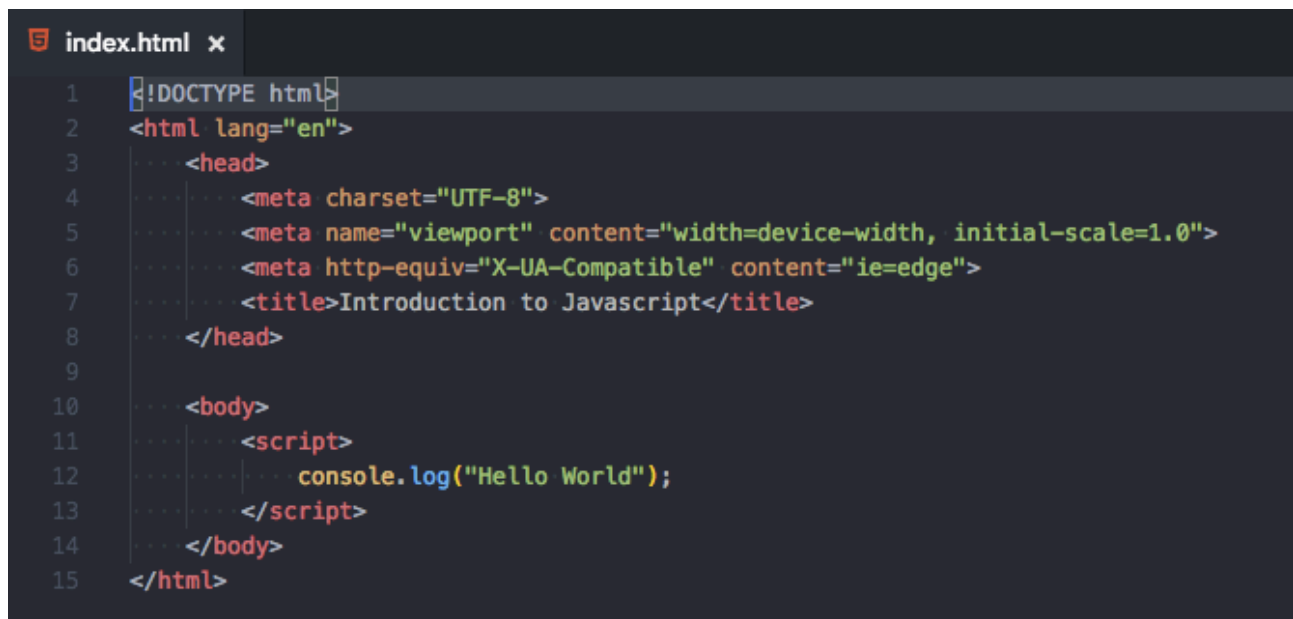
Finally we can output JavaScript code to the JavaScript console of the web browser. To do this we simply use the syntax:

```
Console.log(“”)
```

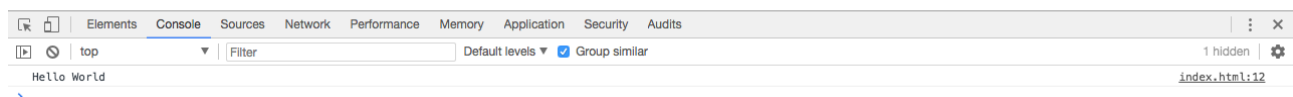
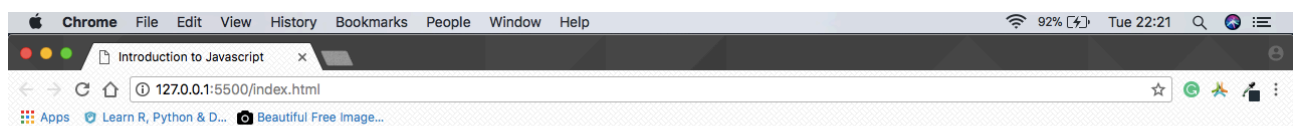
This will write to the console of the browser. To view the console in Google Chrome go to View > Developer > Javascript Console



The console will log the string. We can use the console to debug JavaScript and make sure the JavaScript is functioning properly by testing and using the console log to log the results of the JavaScript — see screenshot of the JavaScript console.



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Introduction to Javascript</title>
8   </head>
9
10  <body>
11    <script>
12      console.log("Hello World");
13    </script>
14  </body>
15 </html>
```



These are the three different methods we can use inside the browser to output JavaScript to the browser (please refer to **Appendix 1 - Output Java.html** as reference for the examples above). We will use the JavaScript console and look at it in more detail in the sections to follow.

How to add Comments in JavaScript?

Adding comments to your code is useful because it allows other developers to read and understand your code. It also helps to write comments as notes for yourself. To write a single line comment in JavaScript we will use double forward slashes — for example:

```
//This is a single line comment.
```

To comment out multi-line we use the same method as CSS — for example:

```
/*Everything within this is commented out.*/
```

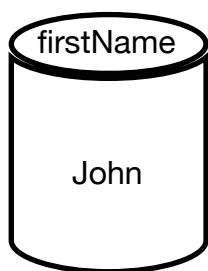
How to create JavaScript Variables?

In this section we will be looking at variables as we will be using variables frequently within JavaScript and therefore it is essential to understand what variables are and how they operate in JavaScript.

A variable is like a container that holds a data. To use a variable we first need to declare a variable in JavaScript in order to use the variable. To declare a variable we will use:

```
var firstName;
```

This will create a variable called firstName; however, this variable does not contain any data/value. We can assign data/value in two ways using the = sign followed by the data:



Method 1 - declare the variable first and then assign the data to the variable:

```
var firstName;  
firstName = "John";
```

Method 2 - declare the variable and assign data at the same time:

```
var firstName = "John";
```

We are storing information in the variables and we can use these values later on in our code by calling/referencing the specific variable name for example:

```
document.write(firstName);
```

Note that we did not need to use var as we only use var when declaring a new variable.

Data Types in JavaScript.

A data type refers to the type of information. There are seven data types:

- Boolean
- Null
- Undefined
- Number
- String
- Symbol
- Object

1. Boolean

Standard across all programming language, booleans are true and false and are often used for conditional statements.

2. Null and Undefined

Undefined means a variable has been declared but has not yet been assigned a value. On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value.

3. Number

The number data type covers integers and floats. The number data type can handle positive, negative and decimal numbers (*compare this with many other languages that have multiple data type to support different type of numbers*).

4. String

A string represents a grouping of characters. Each element in the String occupies a position in the String. The first element is at index 0, the next at index 1, and so on. The length of a String is the number of elements in it.

To write a string we simply use single or double quotation. Numbers within quotations will be treated as string and cannot be computed using math operators.

5. Arrays

We can create an array of data types within a variable by using the opening and closing square brackets for example:

```
var people = ["Andy", "Bella", "Cathy", 1, 5, 7];
```

Arrays allows for different types of data to be stored in a single variable.

6. Objects

Objects are essentially variables but contain many values (*note an array is an object*). The values are written as name:value pairs within curly braces — for example:

```
var car = {carMake:"BMW", carModel:"4 series",  
colour:"Black", carYear: 2018};
```

The car is the object and objects have properties such as the make, model, year, colour and year. Essentially objects are containers for named values.

Objects are more complicated and we will revisit this in the later chapters.

What are Arithmetic Operators in JavaScript?

Arithmetic Operators allows us to perform basic maths on number data types.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Percentage

Loop Arithmetic Operators

x++	Adds 1 to the number x. When we loop the first number will be x and then the next number will be x + 1
x--	Subtracts 1 to the number x. When we loop the first number will be x and then the next number will be x - 1
++x	When we loop the first number will be 1+x.
--x	When we loop the first number will be x-1.

Loop examples:

5++ in a loop will = 5, 6, 7, 8 etc.

5-- in a loop will = 5, 4, 3, 2 etc.

++5 in a loop will = 6, 7, 8, 9 etc.

--5 in a loop will = 4, 3, 2, 1 etc.

What are Assignment Operators in JavaScript?

Assignment Operators allows us to assign a value to an existing value for example:

```
var x = 10;  
x = 10 + 5;
```

This will add 5 to the existing value of x. This can be written in an alternative (shorter syntax):

```
var x = 10;  
x += 5;
```

This adds 5 to the existing value of x which is 10, essentially doing the same as the above code but in a shorter syntax form.

We can use the assignment operators with any arithmetic operator such as =, +=, -=, *= etc. and this will assign the new value to the existing variable value. Note this will only work with number data types.

What are Functions in JavaScript?

In Javascript there are two types of functions:

1. Predetermined functions

```
document.write()
```

Document is the object and write is the function. The write function is a predetermined function because it already exists within the JavaScript language and we do not need to create the write function.

2. User-defined functions.

```
function myFunction(){  
    document.write("Hello World");  
}
```

This is a function that we have create ourselves and does not exist within the normal JavaScript language i.e. not predetermined. To execute this function we will write

```
myFunction();
```


To recognise a function we normally see the syntax of the name of the function followed by the opening and closing parentheses/brackets after the name e.g. `functionName()`

What are Scopes in JavaScript?

Scopes are an important concept to understand within JavaScript. Scopes affect variables and whether they can be used within and outside of functions. There are two types of scopes, Global and Local scopes.

1. Global Scopes

A global scope variable is a variable that is declared outside a function and therefore the variable is outside in the global scope. The variable can be called multiple times outside and inside different functions.

2. Local Scopes

A local scope variable is a variable that is declared within a function and therefore the variable is inside the local scope of the function. This means that the variable can only be used within that specific function and cannot be used outside the function or within another different function.

If you want to create variables that you wish to use in multiple functions, the variable must be outside in the global scope. If you want a variable or data that you want to use within a certain function then the variable can be created within the local scope of the function.

Refer to Appendix 3 - Global and Local Scope.html.

What are Events in JavaScript?

JavaScript events are things that happen to a HTML element i.e. JavaScript code is triggered and interacts/changes the behaviour of a HTML element. Event examples include:

```
onclick - user clicks on a HTML element.  
onchange - An HTML element has changed.  
onload - The browser has finished loading.  
onkeydown - the user pushes a keyboard key.  
onmouseover - the user moves the mouse over an HTML element.  
onmouseout - the user moves the mouse away from an the HTML  
element.
```

There are many events and you should google the different types of JavaScript events to understand what they do and how to write the syntax for the event.

Important Note: unload only works inside specific HTML elements for example in the body tag, image tag and script tag but it will not work in a div tag as an example.

To call an event on a HTML elements we have to write the event in the opening HTML element tag followed by the function we wish to run — for example:

```
<body onload="myFunction()">

    <script>
        function myFunction() {
            alert("Hello World")
        }
    </script>
</body>
```

When everything within the body script has been loaded in the browser this will trigger the event and call on the function to alert the user with an alert message box with the text Hello World.

Objects and Properties in JavaScript?

In the previous section we saw how to create a variable. Variables are essentially an object. The variable `var person = "John";` is an example of a very basic object of person where do not know of any properties.

To create an advanced object by assigning properties we simply use the curly braces — for example:

```
var person = {firstname: "John", lastname: "Doe", age: 35};
```

Another way of writing the above that is more readable is:

```
var person = {
    firstname: "John",
    lastname: "Doe",
    age: 35
};
```

To access the properties from an object within our JavaScript functions we would reference the object followed by the property name within that object — for example:

```
Document.write(person.firstname);
```

Objects and Methods in JavaScript?

A method is simply a function within an object. Taking the above example we can create a method within the object that will write the full name of the object person.

```
var person = {
  firstname: "John",
  lastname: "Doe",
  age: 35,
  fullName: function(){
    document.write(this.firstname + " " + this.lastname);
  }
};
```

When creating the method in the object we do not need to write the name of the function after the `function()` keyword this is because we have already named the function within the object at the beginning i.e. `fullName`.

The keyword `this` refers to the name of the object i.e. this is the same as saying `person`. — because the method is within the object JavaScript understands which object `this` is referring to.

To call the method we can simply write the object followed by the method — example:

```
person.fullName()
```

Objects allows us to record different information/data using properties and this is all located within one place. However, we can call on these properties & methods as many times as we want throughout our website using a simple short code. Therefore, if we would want to change a data of an object, we simply change it in the Object once and it will automatically apply the changes everywhere that makes reference to the object. This makes the code flexible and easy to update/change.

The preferred way of writing the above method is to use the `return` keyword within the method rather than `document.write` keyword. This will return the value but will not show in the browser if the method was called upon. Instead we would use the `document.write` function to call the method — see syntax below:

```
var person = {
  firstname: "John",
  lastname: "Doe",
  age: 35,
  fullName: function(){
    return this.firstname + " " + this.lastname;
  }
};

document.write(person.fullName());
```

Switching between Code and Strings in JavaScript?

In this section we will be looking at how to combine text with JavaScript code inside JavaScript Code and how to include other coding languages inside JavaScript.

To combine two variables we use the + to concatenate the codes. We can also concatenate strings with JavaScript codes. For example:

```
var firstname: "John",  
var lastname: "Doe",  
  
function fullName(){  
    return firstname + " " + lastname;  
}  
  
document.write(person.fullName());
```

We separated the two JavaScript code (variables) by concatenating an empty string that has a space. This will return John Doe and not JohnDoe.

We can use the concatenation to add HTML tags and styling etc. — fo example

```
function fullName(){  
    return "<p class = 'test'>" + firstname + " " + lastname  
    + "</p>";  
}
```

This will create JavaScript code within a paragraph tag with a class of test. Similarly we could have added styling within the paragraph tag (`style = ' '`). Note that we need to use single quotes within the double quotes.

You can also use other coding languages such as PHP but this must be all on one line — for example:

```
function fullName(){  
    return "<?php echo 'Good Day'; ?>" + firstname;  
}
```

This is how we can combine JavaScript with other coding languages or JavaScript.

String Methods in JavaScript?

In JavaScript primitive value is where we have a variable that does not have any properties or methods inside of it for example:

```
var dog = "Husky"
```

In JavaScript there are some existing properties and methods that we can use in the JavaScript library for our primitive values for example:

```
document.write(dog.length);
```

Length is not something that we have created inside of dog but it is something that already exists in the JavaScript library. This property looks at our variable value and returns the number of characters within our string i.e. returns the number 5.

We also have methods within JavaScript that we can use for our primitive values for example:

```
document.write(dog.charAt(3));
```

What this does it looks at our primitive value and finds the character at the third position i.e. returns the character k (*JavaScript uses zero indexing - the first character starts at index of 0, second character at index 1, etc.*).

Number Methods and Math Objects in JavaScript?

Number Methods

The basic idea of a number method is exactly the same as a string method i.e. we can have a primitive value and we are able to perform methods on it even though it is not an object — for example:

```
var x = 9.656;  
x.toFixed(0);  
(100 + 23).valueOf();
```

The toFixed method allows you to specify the number of decimals to display in a number i.e. 0 means none therefore 9.656 will be rounded up to 10.

The valueOf method calculates the sum of the two numbers within the parentheses before it i.e. $100 + 23 = 123$.

There are a whole bunch of number methods we can use - use Google to find different number methods available in JavaScript.

Math Objects

The math object allows us to do more complicated things using maths. Example of Math Object:

```
Math.random();
Math.round(9.6);
Math.ceil(4.4);
Math.floor(4.7);
Math.pow(8,2);
Math.sqrt(64);
Math.PI;
```

Math is a JavaScript object and we can perform a function called random to select a random number between 0 and 1.

The round function will round the math object i.e. 9.6 will be rounded to 10.

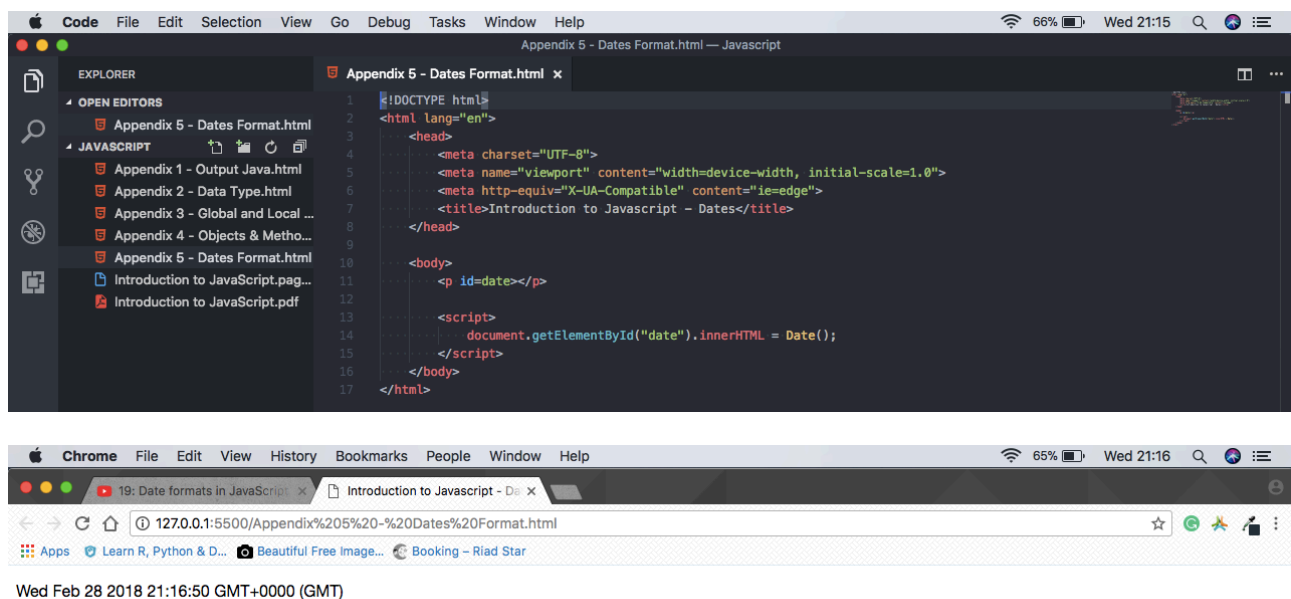
The ceil function allows us to force the rounding of 4.4 to 5 even though this should round down (note: ceil is short for ceiling). Floor will force the value to be rounded down.

Note: the Math object has properties (*PI is a property as it does not have parentheses at the end*) and methods that we can use for more complex maths operations using the built in JavaScript object.

Date Formats in JavaScript?

In this chapter we will look at how to write dates in the browser and change the date to a specific date as well as how to change the format of the date using JavaScript.

In JavaScript there is a method called `date()` which will write the current date and time — see example below:



The above screenshot demonstrates how the date is formatted using the `date()` method. This is known as the “full format”.

To write a specific date we would create a constructor (the concept of constructors are familiar to object oriented programming).

We would create an object and based on that object we will manipulate the information using the date function. To create a new constructor we would write:

```
var dateSelector = new Date();
```

We can now use this dateSelector variable as it is a constructor of `Date()` and we can manipulate the date method within the constructor by adding arguments within the parentheses. We can therefore select a specific date as well as manipulate the format of the date.

We can now reference the constructor within our JavaScript code and manipulate the constructor — for example:

```
document.getElementById("Date") = dateSelector;
```

Selecting a Specific Date

To select a specific date using our constructor it is important to understand the `Date()` method. This method was originally created in 1970 and the default date and time that was set for this method was 01 January 1970 00:00:00.

1. Milliseconds

We can change the date by using the default milliseconds within the parentheses for example 1000 millisecond = 1 second and so we can write `Date(1000)` which will give us the output of "Thu Jan 01 1970 00:00:01 GMT+0000 (GMT)" within our browser. The greater the number the greater the date and time.

2. dateString

We can use a date string to write out the date within the parentheses for example:

```
Date("2018-02-28");  
Browser Result = Wed Feb 28 2018 00:00:00 GMT+0000 (GMT)
```

3. Comma Separated

We can also write the date and time using the comma separator for example:

```
Date(2018, 01, 28, 22, 05, 00, 00);  
Browser Result = Wed Feb 28 2018 22:05:00 GMT+0000 (GMT)
```

Note:

`(year, month, day, hour, minutes, seconds, milliseconds)`

The month uses 0 indexing i.e. 00 = January, 01 = February, 03 = March etc.

4. Date and Time

We can write the date separating the a T for time and ending it with a Z to indicate UTC string format for example:

```
Date("2018-02-28T12:30:00Z");  
Browser Result = Wed Jan 31 2018 12:30:00 GMT+0000 (GMT)
```

5. Short Date

We can write the short-date within the constructor parentheses using the month/day/year for example:

```
Date("02/23/2018");  
Browser Result = Fri Feb 23 2018 00:00:00 GMT+0000 (GMT)
```

6. Long Date

We can select the date using the long-date within the constructor parentheses for example:

```
Date("Jan 10 2018");  
Date("10 Jan 2018");  
Date("10 January 2018");  
Browser Result = Wed Jan 10 2018 00:00:00 GMT+0000 (GMT)
```

7. Full Date Format

Finally we can select the date using the full date within the constructor parentheses for example:

```
Date("Wed Feb 28 2018 23:01:05 GMT+0000 (GMT)");  
Browser Result = Wed Feb 28 2018 23:01:05 GMT+0000 (GMT)
```

Formatting the Date

Note a constructor turns the `Date()` method into a string (i.e. it is the same as doing `date.toString()`). To change the date format we can use the `.toUTCString()` to change the format of our constructor object — for example:

```
document.getElementById("Date") = dateSelector.toUTCString();  
Browser Result = Wed, 28 Feb 2018 00:00:00 GMT
```

Alternatively we can also use `toDateString()` to change the format of our constructor object — for example:

```
document.getElementById("Date") =  
dateSelector.toDateString();  
Browser Result = Wed Feb 28 2018
```

Date Methods

A method is technically a function within an object. So when we create the date object using a constructor we are able to perform some JavaScript methods for the date object for example:

```
var date = new Date()  
document.getElementById("test").innerHTML = date.getDate();
```

The method is called on the date using the punctuation mark followed by the method i.e. in the above example we used the method of `.getDate()` which returns the day value in your browser i.e. number between 1 and 31.

A list of Date methods can be found at:

https://www.w3schools.com/js/js_date_methods.asp

How to Create Arrays in JavaScript?

Arrays allows us to save multiple data within a single variable within JavaScript. When would we use a variable? If we have 100 names and we need to print them out inside our website, instead of creating 100 variables and assigning a name to each one, we can create something called an array.

The benefits of an array is that it takes up less code and is a lot less complicated to create because we are assigning a bunch of data to a single variable.

Variables vs Arrays (Creating Arrays)

We can see below the contrast of creating three names using three different variables and creating three names using an array.

```
var name1 = "Willian";
var name2 = "Anna";
var name3 = "Jack";

var names = ["Willian", "Anna", "Jack"];
```

As you can see, the array does the same as the three separate variables but is less complex and uses less code. Alternatively we can write arrays using the JavaScript array function (*however, this is not the preferred way for most developers as it requires more code*).

```
var names = new Array("Willian", "Anna", "Jack");
```

In order to select a data from the array we will need to select the index number. JavaScript uses zero indexing for arrays to refer to a data therefore 0 refers to the first data, 1 refers to the second data, 2 the third data and so on.

```
document.getElementById("test").innerHTML = names[0];
```

The code above will return the data Willian from the array. If we chose [1] this will return Anna because [1] refers to the second data within the array and not the first.

Creating Array Objects

To create an array object we would create a variable and within the curly braces { } we will then refer to specific properties within the array for the array object — for example:

```
var persons = {firstName:"Willian", lastName:"Banks",
               age:29};
```

In other programming languages they have what is technically known as associated array; this is where there is an array that has names for the data. However, in JavaScript we do not have associated arrays, instead what we have created above is known as an object. Within our object we have properties. To refer to a data within our array object we would write:

```
document.getElementById("test").innerHTML =
persons.firstName;
```

Alternatively we can treat the array as a associated array (although this is an object) and write:

```
document.getElementById("test").innerHTML =  
persons["firstName"];
```

Both will return the firstName data i.e. Willian from the persons array object.
We can also write the array object in a way to make it easier to read for example:

```
var persons = {  
    firstName: "Willian",  
    lastName: "Banks",  
    age: 29  
};
```

Note: the last data within the array object does not require a comma separator.
Furthermore, we can also add arrays within an array object.

Methods within an Array Object

To create a method within an array object we simply add it within the array — for example:

```
var persons = {  
    firstName: "Willian",  
    lastName: "Banks",  
    age: 29,  
    fullName: function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

We have created a function called fullName which concatenates the first name and last name of a person object and we can now call on the function within our code. To call on our array function we would write:

```
document.getElementById("test").innerHTML =  
persons.fullName();
```

This is how we use functions within our objects which then gets renamed as methods.

Arrays in Array Object

We can create arrays within our array object by using the square brackets [] — for example:

```
var persons = {  
    firstName: "Willian",  
    lastName: "Banks",  
    age: 29,  
    pets: ["Briggy", "Snow", "Tyga"]  
};
```

To print out data within the array object from the pets array, we simply refer to the index number of the array item — for example:

```
document.getElementById("test").innerHTML =  
persons.pets[1];
```

This will return the pet name of “Snow” within the persons object.

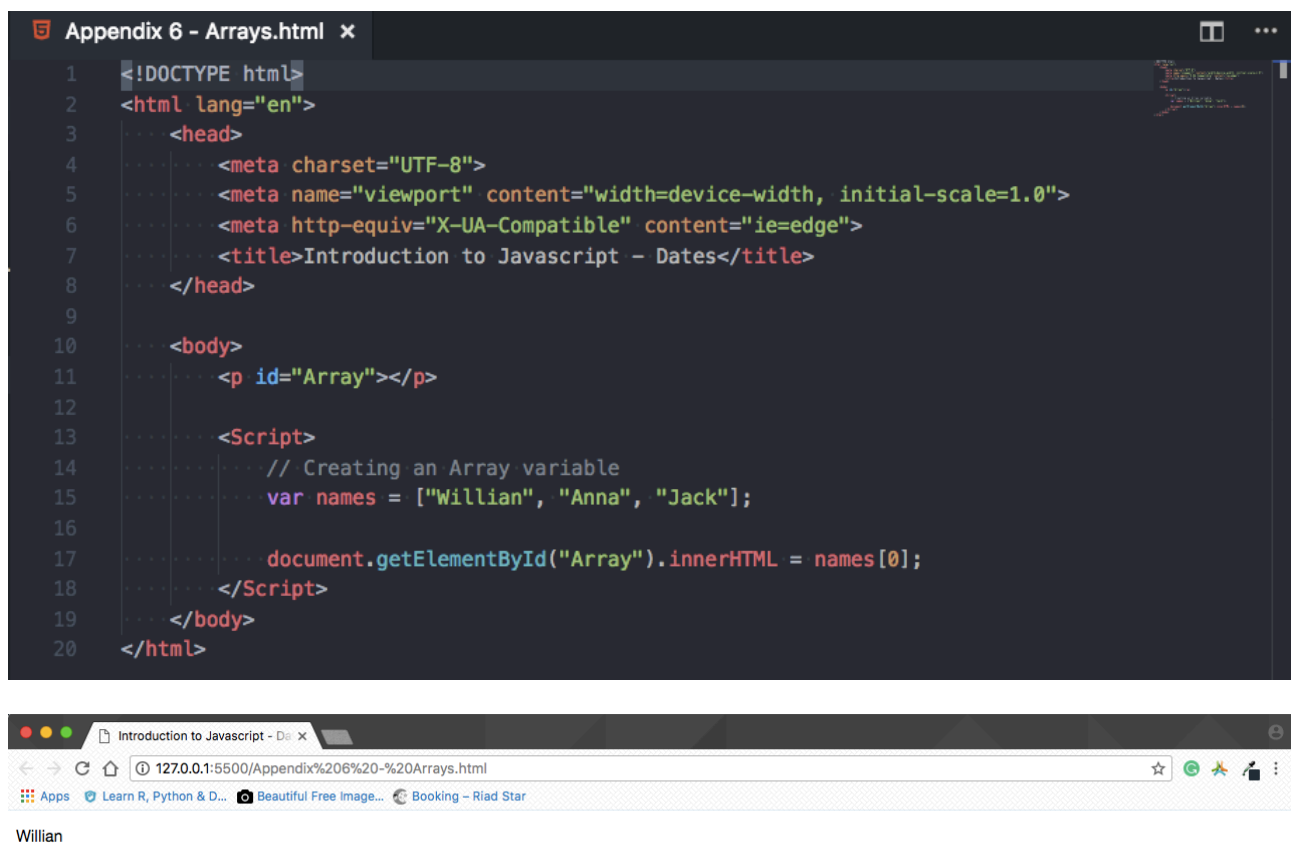
How to Change Data in an Array in JavaScript?

In this section we will learn how to change data within a JavaScript array using functions. We will learn how to delete, insert and replace data within the example array below.

```
var names = ["Willian", "Anna", "Jack"];
```

Deleting Data within an Array

Currently if we were to print out the first name [0] from the array within a paragraph tag we will in our browser, this will return Willian.



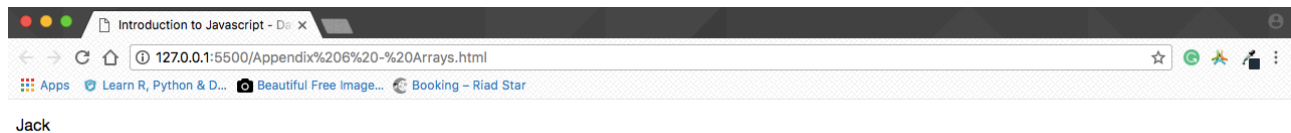
We can use the pop function to delete the last data from the array — for example:

```
names.pop();
```

This will delete the last date from the names array i.e. Jack will no longer exist and if we were to print index [2], the browser will return undefined.

We can put this into a variable to print out the data that was deleted — for example:

```
20 // Delete data from an Array
21 var x = names.pop();
22 document.getElementById("Array2").innerHTML = x;
23 </Script>
24 </body>
25 </html>
```



We can use the shift function to delete the first data from the array — for example:

```
names.shift();
```

This does the same affect as the pop function, however, this will delete the first data and shift the others across. Therefore, index [0] will now become Anna and not Willian/ Undefined because Anna has shifted position as the first data within the array.

Inserting Data within an Array

To insert a new data within an array we simply use the push function and within the parentheses we add the data. This will add the new data at the end of the array increasing the array size — for example:

```
var names = ["Willian", "Anna", "Jack"];
names.push("Emily");
document.getElementById("Array").innerHTML = names[3];
```

The browser will now return Emily at index [3] rather than undefined.

If we were to put this into a variable and printed out the variable, this will return the number of items within the array and not the inserted item i.e. 4 in the above scenario.

We can insert data at the beginning of the array using the unshift function — for example:

```
names.unshift("Emily");
document.getElementById("Array").innerHTML = names[0];
```

This will return Emily and not Willian because we have inserted a new data at the beginning of the array.

Replacing/Changing Data within an Array

We can change any data within an array. To do this we will need to make reference to the array variable, reference the index number of the data we wish to change and provide a new value — for example:

```
var names = ["Willian", "Anna", "Jack", "Emily"];
names[1] = "Panda";
document.getElementById("Array").innerHTML = names[1];
```

This will change/replace Anna with Panda and print out Panda as the new variable data at index [1] of the array.

Splice Data within an Array

In JavaScript we can use a function called splice. This method allows us to select a position within an array and decide whether we would want to delete a number of data from the index position, add data before the index data or change the data of the index data. We can also do this with an unknown amount of data within an array. For example:

```
var names = ["Willian", "Anna", "Jack", "Emily", "Kate"];
names.splice(1, 0, "Panda")
```

The arguments we passed through the splice function parentheses:

- First argument: the index position to start from? i.e. [1] = start from Anna
- Second argument: how many items should be deleted from this index position? i.e. 0 = none, 1 = delete Anna, 2 = delete Anna and Jack etc.
- Third argument: the data to insert after the index position i.e. "Panda" will insert Panda after Anna.

The array will now become names = Willian, Anna, Panda, Jack, Emily, Kate.

Note: the more data we add in the third argument the more will be inserted after Anna but this will not delete/replace the existing data after Anna. For example:

```
names.splice(1, 0, "Panda", "Peter", "Beth");
```

The array will now become names = Willian, Anna, Panda, Peter, Beth, Jack, Emily, Kate.

We are also able to delete data before inserting the new data to the array for example if the second argument was 2, this will delete Anna and Jack before entering Panda, Peter and Beth data into the array.

```
names.splice(1, 2, "Panda", "Peter", "Beth");
```

The array will now become names = Willian, Panda, Peter, Beth, Emily, Kate.

How to write Conditional Statements in JavaScript?

Conditional statements in JavaScript, like any other programming language, requires a condition to be met in order for a block of code to run. In this chapter we will examine the 4 different conditional statements that we can use within JavaScript (*if statements, else if statements, else statements and switch statements*).

If Statements

A if statement is a conditional statement that looks for the value of true and false before a block of code can be run. Below is an example of how we can write an if statement.

```
var x = 5;

if(x==5){
    console.log("True");
};
```

We write the conditional statement within the if parentheses (*brackets*) and if the statement is true the the block of code within the curly braces { } will execute i.e. True will be logged within the browser JavaScript console.

This is the basic idea behind a conditional statement.

Else If Statements

This builds on the If Statement demonstrated above, however, else if allows you to check more than one conditional statement — for example:

```
var x = 2;

if(x==1){
    console.log("X is 1");
}
else if (x==2){
    console.log("X is 2");
};
```

In the above example, the code will go through the first statement to see if it is true, because it is not true it will ignore the block code and move onto the else if statement to check if it is true. Since 2 does equal to 2 it will run the second block of code and log X is 2 in the JavaScript console.

Else if statements allows us to add more if statements to our code (known as nesting if statements). JavaScript will run through each statement until it can find a statement that is true. The statement that returns true, will have its code block executed.

What happens if all statements are false - then nothing will happen as no code block will be executed. However, we can have a conditional statement that executes when all if statements returns false, using what is called an else statement.

Else Statement

Following on from the previous example, if we allowed the user to select a number for the value x, we do not want to write every possible answer as if and else if statements. We can use what is called an else statement.

```
var x = 8;

if(x==1){
    console.log("x is 1");
}
else if (x==2){
    console.log("x is 2");
}
else{
    console.log("x is " + x);
};
```

The else statement acts as an all encompassing conditional statement that will run only if the above if statements return false. In the above example both the if statement were false and so the else statement executed its block of code which will return x is 8.

Note: we do not necessarily need an else if statement for the else statement and we can write the above as:

```
if(x==1){
    console.log("x is 1");
}
else{
    console.log("x is " + x);
};
```

This will check for the first statement to be true if not the else statement will execute.

Switch Condition Statement

The switch statement is similar to the if statements above; however, it operates slightly different. The switch statement requires a expression/condition that it evaluates once and the value of the expression is compared with the value of each case. If there is a match then associated block of code is executed. The syntax for a switch statement:

```
switch(expression){
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        code block
}
```

The **break** keyword lets JavaScript know to break out of the switch statement once the code block is run. This will stop more code running and testing each case block within the switch statement.

The **default** keyword lets JavaScript know to run the default block code when there are no case match.

Example Switch Statement:

```
var day;

switch(new Date().getDate()) {
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
    break;
  case 0:
    day = "Sunday";
}
```

Note: the last case does not require a break at the end because it would be obsolete. We could have made Case 0 as the Default: condition/statement because if none of the above matches then the only day left must equate to Sunday - syntax below:

```
...
  case 6:
    day = "Saturday";
    break;
  default:
    day = "Sunday";
}
```


Case statements can look for any data type such as numbers, strings, booleans etc. as expressions/expressions - for example:

```
var name = "Tom";

switch(name) {
  case "Tom":
    name = "Its Tom!";
    break;
  default:
    name = "Its not Tom";
}

var name = "Tom";

switch(name=="Tom") {
  case true:
    name = "Its Tom!";
    break;
  case false:
    name = "Its not Tom";
}
```

The switch statement is an extension to the if, else if and else statement, it allows us to write the conditional statement in an easy and readable code.

Comparison Operators in JavaScript?

In this chapter we are going to look at comparison operators which builds on the last chapter. Comparison operators allows us to compare statements inside a conditional statement. We use comparison operators to compare one data with another data to return true or false answers which we can use in our conditional statements.

Is Equal Comparison Operator (==)

This comparison operator checks to see if two values are equal to each other regardless of the datatype and will return true or false. For example:

```
var x = 10

if (x == 10){
}
```

The is equal operator will return true because x is equal to 10.

```
var x = "10"

if (x == 10){
}
```

The comparison operator will return true because x is equal to 10 regardless of datatype.

Is Equal Comparison Operator (===)

This comparison operator checks to see if two values are equal to each other but also checks if the two datatypes are the exact same type and will return true or false. For example:

```
var x = 10

if (x === 10){
}
```

The is equal operator will return true because x is equal to 10 and the datatypes are the same i.e. both are numbers.

```
var x = "10"

if (x === 10){
}
```

The comparison operator will return false because although x is equal to 10, they are not the same datatype because one is a string and the other is a number.

Not Equal Comparison Operator (!=)

This comparison operator checks to see if the two data are not equal to one another for example:

```
var x = 10

if (x != 5){
}
```

This will return true because x is not equal to 5.

Not Equal Comparison Operator (!==)

This comparison operator will check both the comparison as well as the datatype to see if the statement is true or false for example:

```
var x = 10

if (x !== 10){
}
```

This will return false because x is equal to 10.

```
var x = "10"

if (x !== 10){
}
```

This will return true because although x is equal to 10, however, they are not the same datatype and so x (a string) is not equal to the number 10.

Greater Than Comparison Operator (>)

This comparison operator will check to see if one data is larger than the other data for example:

```
var x = "10"  
  
if (x > 10){  
}
```

This will return false because x is not greater than 10.

```
var x = "10"  
  
if (x >= 10){  
}
```

This will return true because although x is not greater than 10 but it is equal to 10.

Lesser Than Comparison Operator (<)

This comparison operator will check to see if one data is smaller/lesser than the other data for example:

```
var x = "10"  
  
if (x < 10){  
}
```

This will return false because x is not lesser than 10.

```
var x = "10"  
  
if (x <= 10){  
}
```

This will return true because although x is not lesser than 10 but it is equal to 10.

Logical Operators in JavaScript?

Logical Operators build on from comparison operators and goes hand in hand as it allows you to add additional/multiple conditions inside a statement. There are three logical operators which are AND and OR and NOT logical operators.

AND Logical Operator (&&)

This logical operator adds an additional condition where both conditions must be met in order for the conditional statement to return true — for example:

```
var x = 10

if (x == 10 && x == 5 ){
}
```

This will return false because both conditions are not true (only one condition is true but the other is false which will return false).

OR Logical Operator (||)

This logical operator adds an additional condition where one of the two conditions must be met in order for the conditional statement to return true — for example:

```
var x = 10

if (x == 10 || x == 5 ){
}
```

This will return true because one of the conditions is true.

Joining AND and OR logical operators

All AND conditions must be met in order to return true, however, the OR statement allows for flexibility — for example:

```
var x = 10
var y = 5

if (y == 5 && x == 10 || x == 5 ){
}
```

The above statement will return true because y is equal to 5 and x is equal to 10. The AND operator makes sure at least 2 conditions must be true in order to satisfy the statement. The y data must equal to 5. However, due to the OR operator, x can equal to 10 or 5 but it must satisfy one of these conditions to meet all the conditions of the statement.

NOT Logical Operator (! ())

This logical operator checks to see if a condition is false (*this is the opposite of the above which by default checks to see if the conditions are true*). For example:

```
var x = 10
var y = 5

if (!(y == x)){
}
```

The above will return true because y is not equal to x. We use the exclamation mark before the parentheses to say is not whatever condition is within the parentheses.

Loops in JavaScript?

In this chapter we will look at loops within JavaScript. There are 4 different types of loops that we will look at in details providing examples. It is important to understand loops as they are powerful and useful when you want to run the same code over and over again, each time with a different value. This is often when working with arrays.

The For Loop

The for loop, loops through a block of code a number of times. The syntax for a for loop:

```
for(statement 1, statement 2, statement 3){
    code block
}
```

Within the for loop parentheses we will need to provide 3 arguments/parameters (two are optional, however, it must be noted that you should avoid creating an infinite loop which will crash the browser). Below is an example of the 3 arguments/statements for a for loop:

```
var text = ""

for(var i = 0; i < 5; i = i++ ) {
    text = text + i;
}
```

Statement 1 = this is executed before the loop starts i.e. the starting variable which will control the loop.

Statement 2 = defined the condition for running the loop.

Statement 3 = runs each time after the loop (code block) has been executed.

As long as the condition is true, the for loop will continue to loop through the block code until the condition of statement 2 is false. Statement 1 and 3 control the loop and prevent a infinite loop which will crash the browser.

This will run through the loop 5 times and print out 01234.

We can use arrays within our for loops and allow the block code run through the data within our array for example:

```
var names = ["Daniel", "Mark", "Amy", "Salma", "Dave"];
var text = "";

for(var i = 0; i < name.length; i = i++ ) {
    var text += i;
}
```

The `.length` property returns the number of data within the array object. In this example there are 5 data in the array and so this function will return 5.

Note: the `+=` in javascript is the same as saying `text = text + i` (same as the first example).

To print out the names of the array we can switch out `i` and use the array for example:

```
var names = ["Daniel", "Mark", "Amy", "Salma", "Dave"];
var text = "";

for(var i = 0; i < name.length; i = i++ ) {
    var text += name[i] + "<br>";
}
```

The above for loop will increase the index number with `i` which will select each object within the array to print out.

Statement 1 and Statement 3 are optional and do not have to be within the for loop parameters, however, they must be present in order to prevent an infinite loop - for example:

```
var names = ["Daniel", "Mark", "Amy", "Salma", "Dave"];
var text = "";
var i = 0

for(; i < name.length;) {
    var text += name[i] + "<br>";
    i = i++;
}
```

We must still include the semicolon within the for loop method to indicate statement 2 condition, however, statement 1 and statement 3 are outside the parentheses - but they help control/prevent an infinite loop.

The For/In Loop

The for in loop, loops through properties of an object. This means that if we have an object with properties, we can spit out the data of the object using the for/in loop — for example:

```
var persons = {  
    name: "Bob",  
    gender: "Male",  
    hairColor: "Brown",  
    eyeColor: "Brown"  
}  
  
var text = ""  
  
for (var x in persons) {  
    text += persons[x] + "<br>"  
}
```

As we can see this is similar to the for loop however within the parentheses we require a variable (*note: the variable can be created outside the for/in loop and so we can write x in persons as the argument*) and the object name.

Statement 1 = variable

Statement 2 = object name

The variable x will increment through the loop based on the number of properties within the object.

The While Loop

The while loop, loops through a block of code as long as the specified condition is true. The syntax for the while loop:

```
while (condition) {  
    code block to be executed if condition is true;  
}
```

This is similar to an if statement where we would need to write some condition within the while loop parentheses and the code block will only run while the condition is true.

```
var text = "";  
var x = 0;  
  
while (x < 10) {  
    text += x + "<br>";  
    x++;  
}
```

While the above condition is always true this loop will keep looping until the statement is false. It is important to increment x to prevent an infinite loop.

We can also use the while loop in an array to print out all the data from the array — for example:

```
var personList = ["Piper", "Ellen", "Sasha", "Amber"];
var text = "";
var x = 0;

while (x < personList.length) {
    text += nameList[x] + "<br>";
    x++;
}
```

This will loop through the number of data within the array because .length property will return the number of data within an object. As long as x is smaller than the number of data in an object then this loop through the code block and print out each data. This demonstrates that we can use while loops rather than for loops to get data out of our objects.

The Do/While Loop

The do while loop is a variant of the while loop. The loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the while condition is true. The syntax for a do while loop:

```
do{
    code block to be executed;
}
while (condition)
```

The loop will always execute at least once, even if the condition is false, because the code block is executed first before the condition is tested.