Class 01: The Behavioural Layer

The web standards model (reminder)
What is JavaScript?
Objects, Properties and Methods
What is the Document Object Model (DOM)?
The building blocks of JavaScript: Values (data types), variables and operators
Using JavaScript to change CSS values (DOM Scripting)
Learning JavaScript

References

JavaScript and jQuery by Jon Duckett
JavaScript for Web Designers by Mat Marquis
Learning Web Design (5th Ed) by Jennifer Robbins
Eloquent JavaScript (3rd Ed) by Marijn Haverbeke

JavaScript

https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
https://www.codecademy.com/learn/introduction-to-javascript
http://eloquentjavascript.net/
https://developer.mozilla.org/en-US/docs/Web/JavaScript
http://net.tutsplus.com/tutorials/javascript-ajax/the-best-way-to-learn-javascript/
http://net.tutsplus.com/tutorials/javascript-ajax/24-javascript-best-practices-for-beginners/
http://alistapart.com/articles/behavioralseparation

Homework

Read: Chapters 1-6 of JavaScript and jQuery
or Chapters 1-5 of JavaScript for Web Designers
or Chapters 21 and 22 of Learning Web Design
or Chapters 1, 2, 3 and 4 of Eloquent JavaScript (if you're familiar with programming)

Follow the video tutorial JavaScript: Getting Started at Pluralsight.
Work through the introduction to JavaScript at Codecademy.

This week's task

Create a webpage and use JavaScript to write a different greeting and display a different style on that page depending on the time of day. We'll check your work next week. See the introduction and worked example below for guidance. Also check out the video tutorials on Moodle, where I demonstrate the development of a time-based script.

JavaScript for time-based greetings

This is a general introduction to JavaScript and working through this exercise will help you understand some of the basic principles of scripting and introduce you to some of the language used to describe the various parts of JavaScript. If you've never done any scripting or programming before, this may feel a little outside your comfort zone but it's really not as difficult as it seems. As with HTML and CSS, it's mostly about working out how the syntax works and, in the case of JavaScript, how a script is constructed to achieve a specific task. The task in this exercise is to build a script that can display a different greeting at different times of the day.

We will work with the JavaScript **Date** *object* and use the **getHours** *method* to find the value of the hour *property* and assign that value to a variable. Once we have the value, we can test it against one or more conditions to determine what part of the day it is (e.g. morning or afternoon).

To test our script, we'll need an HTML file. The first four script examples use the HTML template shown below. To keep things simple, no CSS is used for these examples, but you could add your own.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Introduction to JavaScript</title>
</head>

<body>
    <main>
        <h1>A greeting</h1>

        <script>
        // Script goes here
        </script>

    </main>
</body>
</html>
```

The key to writing scripts is to start simple and build them incrementally, so we might decide to begin with a very simple script that gets the hour of the day and writes it to the document. Once we can see that this works, we can add more to it, one step at a time.

A basic script

Our initial script might look like this:

```
<script>
var dateTime = new Date();
var hours = dateTime.getHours();
document.write ("<p>The hour is: " + hours + "</p>");
</script>
```

This script does 3 things, each represented by a line of code, referred to as a *statement*. A semicolon (;) is used to mark the end of each statement.

The first statement creates a new Date object (using the *new* keyword and the *Date()* object constructor) and assigns it to a variable, which we've called “dateTime”. The empty brackets after Date tell JavaScript that there are no additional parameters and that it should just return the current date and time.

The second statement uses the *getHours* method to find the hour from the information stored in our “dateTime” variable and assigns it to a variable called “hours”. The value returned by *getHours* is an integer (a whole number with no fractions) in the range 0-23. So, if the time is 2.45pm when the script is run, *getHours* will return the value 14 and the variable called “hours” will therefore be assigned the number 14.

The third statement builds a string from 3 elements, the opening paragraph tag and some text, the value contained in the “hours” variable and the closing paragraph tag. This process is usually referred to as *concatenation* and it just means “adding strings together”. A “string” is just any sequence of characters, letters, numbers and symbols. The plus operator (+) is used to concatenate strings. Once the string has been constructed, the *document.write* method is used to write the string to the document so that it appears in the browser window. As with algebra, operations in parenthesis (brackets) take place first, so the string is constructed first even though the *document.write* method appears first in the statement.

Notice that the string of text we built is made of some text, a number and some more text. JavaScript didn't complain that we mixed numbers and letters (as some programming languages might). That's because JavaScript is what is called a "weakly typed" scripting language, meaning that it infers the data type from the context. In this case, we are building a string, so JavaScript correctly assumes that we want to change the data type of our number to a string.

When the webpage is opened and the script runs, the following will be displayed (assuming the time is 2.45pm when the page is opened:

The hour is: 14

This is a nice, easy way to begin. Even though this is a very simple script with just three statements, it does something useful. See pages 136 to 139 of Jon Duckett's JavaScript & jQuery for more information on working with dates and times.

See this script in action.

Let's take our script to the next level and get it to work out if it's morning or afternoon…

Iterating your script, is it morning or afternoon?

Once we have the basic script working, we can develop it by adding more statements that do other things like determining whether it's morning or afternoon. You could do this with an if…else statement to test against a condition. The following script is an evolution of the one above:

```
<script>
var dateTime = new Date();
var hours = dateTime.getHours();
var greeting;
if (hours >= 12) {
    greeting = "Good afternoon";
}else{
    greeting = "Good morning";
}
document.write ("<p>" + greeting + "</p>");
</script>
```

This script is the next logical step. Instead of just writing the number of the hour to the browser, this script checks to see whether the number represents morning or afternoon and writes a different greeting depending on the result. Clearly, a value of 12 or higher means it's the afternoon. While a value of less than 12 means it's the morning.

The first two statements in this script are the same as the first two in the previous script.

The third statement creates a variable called "greeting" but does not assign it a value (in effect, it's an empty box, ready to accept a value at some point in the future).

Following that, we have added an if…else *conditional statement*. This tests to see if it's an afternoon and if not an afternoon it must be a morning. The first line of the statement is the most important because it contains the *condition* and determines what happens next. In this case, we are comparing the value stored in the "hours" variable with the number 12. We are using the *greater than or equal to* (>=) operator to do this. The condition says: if the value assigned to the "hours" variable is greater than or equal to the number 12, do something, otherwise do something else.

If the condition returns *true*, the first statement is run and the "greeting" variable is assigned an afternoon greeting. If the condition returns *false*, the second (else) statement is run and the variable is assigned a morning greeting.

The final statement adds the opening and closing paragraph tags to the greeting and writes the resulting string to the document in the browser. Assuming the webpage is opened at 2.45pm, the user will see the following:

Good afternoon

This is because the number 14 is greater than the number 12 and so the condition is true.

See Chapter 4 of Jon Duckett's JavaScript & jQuery for more details on making decisions with your scripts.

See this script in action.

A smarter script, using multiple conditions

Again, our script could be iterated further so that it becomes even more smart, for example it could let us know that it's lunch time or time to go to bed or whatever we like. To do this, we might use a series of if…else statements nested in what's called an *else if clause*.

Here's what such a script might look like:

```
<script>
var dateTime = new Date();
var hours = dateTime.getHours();
var greeting;
if (hours <= 5) {
    greeting = "You should be in bed!";
}else if (hours <= 11) {
    greeting = "Good morning!";
}else if (hours == 12) {
    greeting = "Almost lunch time!";
}else if (hours == 13) {
    greeting = "It's lunch time!";
}else if (hours <= 16) {
    greeting = "Good afternoon!";
}else if (hours <= 22) {
    greeting = "Good evening!";
}else{
    greeting = "Time to get ready for bed!";
}
document.write ("<p>" + greeting + "</p>");
</script>
```

Look at the conditions for each statement and see if you can work out what the greeting will be for any given time. Understanding the logic of these sorts of decision-making structures in any programming language is an important step in learning how to write beautiful code.

The order in which the conditions occur is important because the conditional statement will complete as soon as it finds a condition that is true. Once that happens, the statement associated with the first true condition is run. In our example, the 5th condition is true (because the time is 2.45pm and the value stored in the "hours" variable is 14) and so the "greeting" variable is assigned the string value "Good afternoon!".

Notice that the condition for lunch time looks like this *(hours == 13)*. In JavaScript we use a double equals symbol to indicate *equality*. That's because a single equals symbol is used for *assignment*, e.g. where a value is assigned to a variable.

See this script in action.

Using your script to change CSS values

Well, once we have this decision-making structure, we could change anything we like within the given time ranges. This next iteration of the script adds a new variable for colour, sets a different hexadecimal value in each condition and then sets the body background colour to the colour for that time of day.

Here's what the script looks like:

```
<script>
var dateTime = new Date();
var hours = dateTime.getHours();
var greeting;
var colour;
if (hours <= 5) {
    greeting = "You should be in bed!";
    colour = "#56adff";
}else if (hours <= 11) {
    greeting = "Good morning!";
    colour = "#5ee4ec";
}else if (hours == 12) {
    greeting = "Almost lunch time!";
    colour = "#6cf1b7";
}else if (hours == 13) {
    greeting = "It's lunch time!";
    colour = "#5fd067";
}else if (hours <= 16) {
    greeting = "Good afternoon!";
    colour = "#bfd859";
}else if (hours <= 22) {
    greeting = "Good evening!";
    colour = "#ecd34b";
}else{
    greeting = "Time to get ready for bed!";
    colour = "#fba035";
}
var body = document.querySelector("body");
body.style.backgroundColor = colour;
document.write ("<p>" + greeting + "</p>");
</script>
```

All we've done here is to add a new variable for colour (statement 4) and then added a different hex value to each of the conditions in our else if clause.

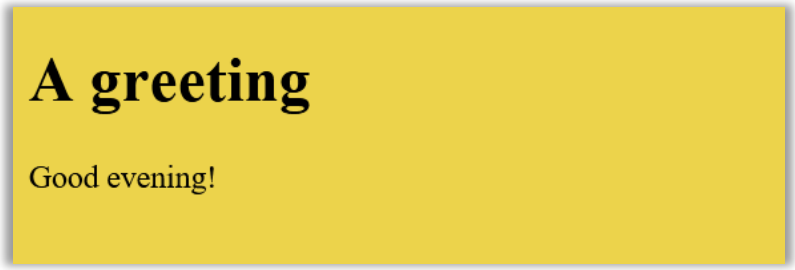
Notice that we've used the *document.querySelector* method to select the body element. This is a lovely way to select elements in JavaScript because you can use a CSS selector as a parameter, meaning you can select elements in JavaScript just as you would in CSS. It's a relatively new method and won't work with older browsers (before IE8). If you need to support older browsers, it's probably better to use the *document.getElementByTagName* method, which was first supported by IE6.

Once the body element is stored in a variable that we've called "body", we can change the *backgroundColor* property of body by assigning it the hex value contained in the "colour"

variable. This happens in the penultimate statement of our script. Notice that we change or set the value of a CSS property on an element simply by assigning it. In the penultimate statement, we use dot notation to identify the property we want. We begin by identifying the HTML element (the element previously selected and stored in the "body" variable). Then we tell JavaScript to get the style for that element and finally, the background colour. Once we have that property selected, we use an equals symbol to assign the value contained in the "colour" variable. The value is updated "on the fly", no page refresh is required and the change is displayed in the browser window as soon as the script is run.

The final statement writes the greeting as before.

If the time is half past six in the evening, the script will display the following:



The greeting "Good evening!" and the colour *#ecd34b* are both specific to that time of day and will change at other times.

See this script in action.

Essentially, JavaScript enables you to change whatever you like (both HTML and CSS). Have a go at changing some other things based on the time of the day. You can select any HTML element and change any CSS property of that element.

A note on the placement of JavaScript (doing it the right way)

In all of the examples above, for simplicity, we've placed the script directly in the HTML file and used *document.write* to display the result. One of the downsides of *document.write* is that we need to place the script *inline* with the HTML because it can only write to the document at the point where the script appears.

Obviously, this runs contrary to the web standards model, which suggests that JavaScript should be kept separate from HTML. Ideally, our script should be in its own file and we should link to that JavaScript file just as we link to our CSS file. In that way, we keep everything separate, just as the web standards model suggests we should.

Our script can still work if separated from the HTML, but we need to change some of the methods we've used. For example, we can remotely select and change HTML elements by using something like the *innerHTML* property of the *getElementById* method to inject our greeting rather than writing it in place. This will work whether the script is in the HTML file or not.

The other (potentially more important) issue is that if JavaScript were to fail or if the user has turned it off, no greeting will be displayed because our greeting is created by JavaScript. It would be far better to include a default greeting that was changed (not created) using

JavaScript so that at least some greeting is seen, irrespective of whether JavaScript is available or not. This is a key principle of working with JavaScript; always make sure there is a fall-back.

The next iteration of our script will resolve both of the above issues (placement and fallback)…

Improving the script for best practice

This version of our script uses a slightly different HTML template.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Introduction to JavaScript</title>
</head>

<body>
    <main>
        <h1>A greeting</h1>
        <p id="greeting">Welcome</p>
    </main>
<script>
// Script goes here
</script>
</body>
</html>
```

There are two key differences between the HTML template we were using for the previous examples and this one. First, we have moved the script so that it is placed just before the closing body tag. This is important because our new script will select an element, so we need to make sure that the DOM (Document Object Model) is complete when the script runs. Second, we have added a default greeting below h1. This means that if JavaScript is unavailable, the page will still display the greeting “Welcome”.

In addition, the greeting paragraph has an ID attribute, this will be used for selecting that element from our script using the *document.getElementById* method.

Here's what the improved script looks like:

```
<script>
var dateTime = new Date();
var hours = dateTime.getHours();
var greeting;
var colour;
if (hours <= 5) {
    greeting = "You should be in bed!";
    colour = "#56adff";
}else if (hours <= 11) {
    greeting = "Good morning!";
    colour = "#5ee4ec";
}else if (hours == 12) {
    greeting = "Almost lunch time!";
    colour = "#6cf1b7";
}else if (hours == 13) {
    greeting = "It's lunch time!";
    colour = "#5fd067";
}else if (hours <= 16) {
    greeting = "Good afternoon!";
    colour = "#bfd859";
}else if (hours <= 22) {
    greeting = "Good evening!";
    colour = "#ecd34b";
}else{
    greeting = "Time to get ready for bed!";
    colour = "#fba035";
}
var body = document.querySelector("body");
body.style.backgroundColor = colour;
var el = document.getElementById('greeting');
el.innerHTML = greeting;
</script>
```

See this script in action.

Notice that the new script is very similar to the previous version. The last line of the previous script that wrote the greeting using the *document.write* method is replaced by two new statements. The first selects the element with the id "greeting" and the second replaces any existing text with the contents of the greeting variable. That is the only change, but the script is now much more portable and robust than before.

The first new line declares a variable called “el” (short for element) and uses the *document.getElementById* method to assign our greeting paragraph to that variable using the id attribute value as a parameter. Remember, ids are unique, so there is no doubt about what we are selecting.

The second new line sets the *innerHTML* property of our element to the value contained in the greeting variable. Any value we assign to this property will replace any existing text with the text provided. In this case, the text contained in the “greeting” variable will replace the word “Welcome”.

This final version has improved our script in two important ways:

First, the script is more portable, it will still work as expected wherever it is. In fact, we can now place this script in an external .js file and call it from the HTML file like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Introduction to JavaScript</title>
</head>

<body>
    <main>
        <h1>A greeting</h1>
        <p id="greeting">Welcome</p>
    </main>
<script src="greeting.js"></script>
</body>
</html>
```

The HTML above, assumes our script is saved in a file called *greeting.js*. Now we have succeeded in separating the JavaScript from the HTML. Using an external JavaScript file would be sensible if the same script were being used on many pages but if it is unique to this page, it may be more efficient to include it in the HTML file, which avoids an additional request to the server. Either way, we have avoided using any *inline* JavaScript.

The greeting.js file contains only JavaScript, the script tags are not included because they are HTML, so the beginning of the greeting.js file might look something like this:

```
// A simple script to add a greeting and
// background colour based on time of day
var dateTime = new Date();
var hours = dateTime.getHours();
var greeting;
var colour;
if (hours <= 5) {
    greeting = "You should be in bed!";
    colour = "#56adff";
}else if (hours <= 11) {
    greeting = "Good morning!";
    colour = "#5ee4ec";
```

See this script in action.

Note the location of the script tag, just before the closing body tag and remember what we said about ensuring that the DOM is complete before the script runs. This still holds true when the script runs from an external file. However, if you'd rather see all your script references in the head of your HTML document, you can do this too, providing that you use the *defer* attribute as in:

```
<script src="greeting.js" defer></script>
```

The defer attribute ensures that all the HTML is parsed, and the complete DOM created before the script runs. See Efficiently Run JavaScript for more information.
Secondly, the script is much more robust, and our page will still work even if JavaScript is unavailable…

Under normal circumstances, the bespoke greeting and an appropriate colour background will be displayed:

But if JavaScript is unavailable, we still see the default greeting on a white background (or whatever the default background colour is):

Of course, the two versions of our page don't look the same, but the important point is that we have maintained the functionality of the page (it still provides a greeting). This is one of the core principles of *progressive enhancement*. As designers, we can provide an improved experience for those users who have access to it (in this case, the ability to run JavaScript), while still providing a useable experience for those who don't.

As with HTML and CSS, you'll need to play about with JavaScript to get a feeling for how it works. There's a lot to learn but don't let that put you off, just start by keeping things simple and taking small steps.

In the first instance, you may want to modify the final version of our script to make some other changes to the page or to display a different greeting.

Happy scripting!

December 2020