Edinburgh Napier University

SET08101/SET08401 Web Tech

Lab 4 - JavaScript

Dr Simon Wells

1 Aims

At the end of the practical portion of this topic you will:

- Update HTML & CSS via the DOM
- Interact with the Browser window

NOTICE: Just as with our other core web languages, HTML & CSS, there is a lot of useful material online. In addition to reading the relevent chapters of the module texts, you should also avail yourself of the following which document JavaScript:

- `https://developer.mozilla.org/bm/docs/Web/JavaScript`
- `https://www.w3schools.com/jsref/default.asp`

2 Activities

We now have all of our web technologies, HTML, CSS, and JavaScript, in place. So we can combine them and exploit them to build more interesting and dynamic web sites. Last week we merely explored JavaScript as "yet another programming language", discovering that it had all of the parts that we would expect a programming langue to provide us with. Now however we want to use it for what it was created for, to enable increased functionality on our web pages, beyond what mere HTML and CSS can support.

2.1 Preliminaries

This week we shall mostly use a single, simple, external javascript file and an associated simple host HTML document to bootstrap our JavaScript into the browser so that it can fulfil its purpose. Whilst some of our exercises might demonstrate things in different ways for clarity, such as some of the interaction with the DOM examples, we can always use the following as a starting place to develop some new functionality.

```
<!DOCTYPE html>
<html>
    <head>
         <script src="code.js"></script>
        <title>SET08101 - JavaScript</title>
    </head>
    <body >
        <p><a href="#" onClick="helloFunction();">Click Me</a></p>
    </body>
</html>
```

Now create a second text file in the same folder as external.html called code.js and place the folowing code in it:

```
function helloFunction(){
    alert('Hello Napier');
}
```

2.2 Interacting with the DOM

One useful thing to do from JavaScript is to update our HTML from JavaScript code, or *programatically*. Let's try that out:

```
<!DOCTYPE html>
<html>
    <head>
        <title>SET08101 - Interacting with the DOM</title>
    </head>
    <body >
```

```
7         <p><a href="#" onClick="addMessage();">Click Me</a></p>
8         <h1>OUTPUT:</h1>
9         <p id="outputDemo"></p>
10
11        <script>
12            function addMessage()
13            {
14                document.getElementById("outputDemo").innerHTML = "HELLO WORLD"
15            }
16        </script>
17    </body>
18 </html>
```

Rather than a specific HTML typographical element like <p> we can add new content to a more general element like a , e.g.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>SET08101 - Interacting with the DOM</title>
5      </head>
6      <body >
7          <p><a href="#" onClick="addMessage();">Click Me</a></p>
8          <h1>OUTPUT:</h1>
9          <div id="output"></div>
10
11         <script>
12             function addMessage()
13             {
14                 document.getElementById("output").innerHTML = "<p>HELLO WORLD</p>"
15             }
16         </script>
17     </body>
18 </html>
```

It gets even more interesting than this. If we use += instead of just the "=" when assigning our new HTML then we can add additional lines rather than setting things just once.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>SET08101 - Interacting with the DOM</title>
5      </head>
6      <body >
7          <p><a href="#" onClick="addMessage();">Click Me</a></p>
8          <h1>OUTPUT:</h1>
9          <div id="output"></div>
10
11         <script>
12             function addMessage()
13             {
14                 document.getElementById("output").innerHTML += "<p>HELLO WORLD</p>"
15             }
16         </script>
17     </body>
18 </html>
```

Try this out and experiment a little with different tags. Try setting things up so that each time you click the link a new element will be added to an ordered list.

We can also mix HTML, CSS, and JavaScript, for example:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <style type="text/css">
5              p {color:red}
6          </style>
7          <title>SET08101 - Interacting with the DOM</title>
8      </head>
9      <body >
```

```
10          <p><a href="#" onClick="addMessage();">Click Me</a></p>
11          <h1>OUTPUT:</h1>
12          <p id="redPara">I am RED</p>
13
14          <script>
15              function addMessage()
16              {
17                  document.getElementById("redPara").style.color = "blue";
18              }
19          </script>
20      </body>
21  </html>
```

What about toggling the colour of our paragraph each time the link is clicked. To do this we'll need a variable to store the current colour in. We'll then need to check the current state and switch colours to the other on each click, for example:

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <style type="text/css">
5               p {color:red}
6           </style>
7           <title>SET08101 - Interacting with the DOM</title>
8       </head>
9       <body >
10          <p><a href="#" onClick="colourChanger();">Click Me</a></p>
11          <h1>OUTPUT:</h1>
12          <p id="redPara">I am RED</p>
13
14          <script>
15              var current="red";
16
17              function colourChanger()
18              {
19                  if(current === "red")
20                      current = "blue";
21                  else
22                      current = "red";
23
24                  document.getElementById("redPara").style.color = current;
25              }
26          </script>
27      </body>
28  </html>
```

Notice how we've stored the current colour in a variable called current. Also notice that this variable is within the script tags but not within the colourChanger() function. This is because we want our variable to be initialised once when the script is loaded and then updated each time the function is called.

See if you can come up with a way to change the colour of your paragraph and also update the message to match the colour. Consider how you might switch between three, or even four, colours instead of just two.

Revisit the examples from above and attempt to separate out the HTML, CSS, and JS into different source files to get a better structure and organisation.

2.2.1 Interact with the Browser Window

Remember that all the components of our web page are represented internally within our web browsers in terms of a data structure called the DOM. The DOM is a tree structure, a hierarchical organisation of the elements that make up our page. Everything within our page can be accessed from a single root object, called the *window*.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>SET08101 - Interacting with the Window</title>
```

```
    </head>
    <body >
        <script>
            for (property in window)
            {
                if(property) {console.log(property)}
            }
        </script>
    </body>
</html>
```

This should give you an idea of the range of possible parameters that the window object gives you access to. It will be a reasonably long list, and will differ across browsers. You should also note that many of the properties also have additional parameters, such as screen, which has things like screen.height and screen.width. So our exploration of things that we can access doesn't stop here. Note that you can also explore things directly in the web console as a more interactive way of investigating. This can be useful as the web console provides suggestions and auto-complete (if you hit tab after highlighting the line to complete from the suggestions).

Why not try adapting the above example to update the HTML page itself with the output of the properties available from the window object[1]

We can interact with our window to control the part of the page that is displayed. For example, using the scrollBy method of the window object. Let's look at a simple example, try to work out exactly what is happening all the way through this one:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the Window</title>
    </head>
    <body >
        <div id="output"></div>

        <script>
            var out = document.getElementById("output");
            for(i=1; i<2000; i++)
            {
                out.innerHTML += i+ " ";
            }
            window.scrollBy(0, 100);

        </script>
    </body>
</html>
```

Note that you should probably make sure that your browser window is reasonably small as the scrollBy method only works when the content of the page is bigger than the available window resulting in scroll bars. Our scrollBy method lets us choose the x and y dimensions to scroll by.

We can go as far as creating new windows if we want to. Note that this example might get blocked by our browsers pop-up control functionality, mainly because pop-ups were heavily misused by web designers to the point where people began installing pop-up blockers then similar functionality was eventually built in. You'll also probably want to change the URL of the window that you open to something more interesting, either a local html file or a site you might want to visit instead of "test.html" in the example code...

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the Window</title>
    </head>
    <body >
        <div id="output"></div>

        <script>
```

[1] *Hint:* You can use a similar loop but probably want to append an HTML element to the screen, perhaps using '+=' on each iteration.

```
            window.open("test.html", "new window", "top=100, left=100, width=640,
                height=480, status=yes");
        </script>
    </body>
</html>
```

However, because the only person with direct access to the web console is the user, you should be able to directly type the window.open line into your console and it should bypass the pop-up blocker.

We can query our browser to find out more about it by using the 'navigator' property. For example, let's retrieve the browser's name, codename, and version number (amongst a whole heap of other information).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the Window</title>
    </head>
    <body >
        <div id="output"></div>

        <script>
            var out = document.getElementById("output");
            out.innerHTML += window.navigator.appName
                + " " + window.navigator.appVersion
                + " " + window.navigator.appCodeName;
        </script>
    </body>
</html>
```

Our browser is usually displaying a specific location, and we can access that information from our code like so:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the Window</title>
    </head>
    <body >
        <div id="output"></div>

        <script>
            var out = document.getElementById("output");
            out.innerHTML += "Href: " + window.location.href
                + "<br>Protocol: " + window.location.protocol
                + "<br>Host: " + window.location.host
                + "<br>Path:" + window.location.pathname
                + "<br>Hash: " + window.location.hash;
        </script>
    </body>
</html>
```

In the above listing we've broken down the content of our address bar into manageable chunks that we could, if necessary use in our code.

There is frequently also a history of previous pages that have been visited, again, information that we can access and manipulate. Open a new browser window and navigate to a web site of your choice[2], then follow a few links. Open the browser console and access the window.history.length data then try out the window.history.back(), and window.history.forward() functions. The number of elements in the history also means that you can jump forwards and backwards using the history.go() function and passing it a positive or negative number as an offset from the current page, i.e. to move forward 2 pages then history.go(2) and to move to the previous page then history.go(-1). Note that this browser API doesn't give us access to the specific pages through JavaScript as this would be a big security risk.

[2]such as, perhaps, wikipedia

2.3 Interacting with the User

In the preliminary above, and in the last topic, we used an alertbox to interact with our user like so:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the user</title>
    </head>
    <body >
        <script>

        window.alert('hello');
        </script>
    </body>
</html>
```

It turns out that there are two other dialogs that we can get by default. After the alertbox, the second is the confirmation box and the third is the prompt box. Let's try them out. First the confirmation:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the user</title>
    </head>
    <body >
        <script>
        window.confirm('are you sure?');
        </script>
    </body>
</html>
```

We can also get the result of choosing one or the other button and use that in our code, for example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the user</title>
    </head>
    <body >
        <script>
        var result;
        result = window.confirm('are you sure?');
        console.log(result);
        </script>
    </body>
</html>
```

Let's now look at the prompt box. It works fairly similarly to the confirm box:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the user</title>
    </head>
    <body >
        <script>
        window.prompt('What is your name?');
        </script>
    </body>
</html>
```

Similar to the confirmation box we can also access the value that our user typed into the prompt so that we can use it in our code. For example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Interacting with the user</title>
    </head>
    <body >
        <script>
        var result;
        result = window.prompt('What is your name?');
        if(result != null)
        {
            console.log(result);
        }
        </script>
    </body>
</html>
```

2.4 Challenges

As in previous weeks, consider how the challenges we've already addressed can be enhanced using this weeks topic. What would JavaScript enable you to do on your set of personal pages, on your world factbook site, or on your choose your own adventure game? You'll notice that some types of site lend themselves to useful JavaScript that really enhances the user experience whilst for other sites including JavaScript can be a little gimicky.

1. Create a simple dungeon crawler game that uses JavaScript to generate what is found in each room of a dungeon as you explore through it. Use a set of pages to implement the "rooms" of the dungeon, with each page having a link to the next location, or room, perhaps with links to a variety of rooms. In each room use JavaScript to generate the contents of the room. Start with whether the room contains treasure, perhaps of varying types. Then think about a monster to defend the treasure. Perhaps you will want to battle the monster? You can elaborate on this to a great degree. Research how you might store information about your player and the treasure that's been collected or the health of your player[3]. If the player dies then you might want to direct your player towards a suitable ending page. Similarly if they get through all of the rooms and out of the dungeon safely then they might get a reward page of some sort.

2. Consider how you might integrate JavaScript into your solutions to the challenges in previous weeks. You might find that some sites don't lend themselves well to the additional dynamicism of JavaScript whilst for other, JavaScript offers an opportunity to develop features that just couldn't be achieved with HTML & CSS alone. For example, your personal pages might not require much JavaScript, although I could imagine a utility that calculated your overall year and degree grades based on the individual grades for assignments and their relative weightings. Similarly, The world factbook based challenge might not lend itself to supporting a lot of JavaScript functionality, however, perhaps a function on each page that checked whether data had been updated in the official factbook and updated your local copy might be useful as much of the data associated with countries, for example, population and GDP data, is dynamic on annual timescales. Finally, the choose your own adventure challenge perhaps offers the greatest opportunities for JavaScript functionality; you could incorporate some more dynamic story aspects to the game, perhaps selecting different sections of prose depending, or enabling a random encounter of some sort that alters the available story paths. Remember that JavaScript gives you access to many APIs within the browser that can enable you to incorporate things like generated graphics (using the canvas element) or sound (using the web audio API). Whilst these are not right for many sights, sound for example is downright annoying when intrusive, unexpected, and uncontrollable, it is very effective when used correctly and with sufficient user control.

Two things to note are that JavaScript isn't always necessary, and often there isn't a good use case for it, but when there is, JavaScript enables you to build some fantastic functionality that just isn't practical in any other way right now. As a reward, treat yourself to a little game of MonsterBreeder[4].

[3] *Hint:* there are various ways to store data in the user's browser, e.g. cookies, web storage, and indexed DB

[4] This is actually a good example of the kind of innovative game that you can develop using core web technologies. Notice how the use of simple graphics and sound, provide a simple but atmospheric user interface, and that the game,

2.5 Finally

If you want additional practise then the W3Schools JavaScript exercises are a great place to start:

- JavsScript Exercises: `https://www.w3schools.com/js/`

A good way to practice a new language or to improve your existing abilities, not just in JavaScript, but in any language you might be trying to learn, is to try to solve a set of problems using the language. I often use Project Euler when starting out with a new language but there are also many others:

1. Project Euler: `https://projecteuler.net/`
2. Stack Exchange Code Golf: `http://codegolf.stackexchange.com/`
3. Code kata: `http://codekata.com/`
4. Reddit Daily Programmer: `https://www.reddit.com/r/dailyprogrammer`
5. Programming Praxis: `http://programmingpraxis.com/`
6. Rosetta Code: `http://rosettacode.org/wiki/Main_Page`
7. International Collegiate Programming Contest Problems Index: `http://acm.hit.edu.cn/judge/ProblemIndex.php`
8. Algorithmist: `http://www.algorithmist.com/index.php/Main_Page`

Another trick I have is to have a short list of small coding projects that I redo whenever I learn a new language. My personal favourites are to write programs that deal with the following topics (but you should put together your own list if my interests don't match yours):

1. Writing 1D and 2D Cellular Automata
2. Conway's Game of Life
3. Simple text based games, e.g. story-telling or adventure games
4. Quines
5. Simple codes & ciphers

By repeating exercises that you're already familiar with in a new language you can quickly get a feel for a new language does things differently or offers features that make a given problem easier or more difficult to tackle.

This isn't the end of our JavaScript journey, but just the beginning, we're going to spend time getting used to client side JavaScript, then we'll look at server-side JavaScript later on.

whilst generating random encounters, has a neat story-arc and well-formed core mechanic. `http://monsterbreeder.com/`