Interactive webpages using JavaScript and the DOM

Dr Jon Bird

Overview

This lecture is about how we can use JavaScript (JS) to develop *interactive* web pages.

The focus is on the client side and builds on the last lecture materials that covered HTML5 and CSS.

I'm going to introduce JavaScript and the Document Object Model (DOM) and JSON. In order to appreciate the full power and capabilities of JavaScript and the DOM, I recommend you read the extensive resources at:

www.w3c.com/js/

https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript (JS)

What is JavaScript? 1

In the previous session we focused on using HTML and CSS to produce *static* webpages – once they were rendered by a browser they did not change.

JavaScript (JS) is a programming language that enables us to add *dynamic* features to website pages such as: updating content in response to users interacting with a page e.g. clicking a button, animated graphics and image rollover effects.

JS can do this while running in the browser and our focus in this lecture is on using it as a client-side technology. However, JS is also used to do server-side programming i.e. responding to requests from clients, and we'll look at this next week.

Overview of key JS syntax

Some JS syntax will be familiar to you as you have learnt C. However, some JS syntax is quite different. I’ll give a quick overview of the syntax for following programming elements:

- Variables and constants
- Comparisons and equality
- If conditions
- Loops
- Objects
- Arrays
- Functions

In the workshop you’ll try out these code snippets using jsbin, which is an online collaborative client side coding tool which lets you try out HTML, CSS and JS.

Variables and constants

Similar to Python, JavaScript is a *dynamically typed* programming language.
This means that the user does not have to define the type of a variable when declaring it and a variable can contain data of any type.
There are two types of variable, declared with ‘let' and 'const'

```
let x = 5;  //declare a variable

x = 10; //we can assign another value to x

const y = 8; //declare a constant

y = 13; // throws error as y is a constant
```

Comparisons and equality

```
let x = 5;
x  ===  5  //true
x  ===  8  //false
x  !==  8  //true
```

if statements

```
if(x>=5) {
     console.log(“x >= 5”);
}
else{
     console.log(“x < 5”);
}
```

for loops

```
for(firstExpression; condition; incrementExpression) {
    //code per iteration
}
```

```
for(let i = 0; i<5; i++) {
    console.log(i);
}
```

JS Objects

JS objects can contain multiple values

```
let x = {
          firstName: “Bernie”,
          lastName: “Sanders”,
          colour: “blue”,
          active: true,
          sayHello: () => {
            console.log(“Hello”)
          }
};

console.log(x.firstName) //”Bernie”
console.log(x.sayHello()) //”Hello”
```

Arrays

Arrays are collections of objects and are 0 indexed

```
let colours = [‘blue’, ‘red’, ‘black’, ‘purple’];

colours[0] //’blue’
colours[0] = ‘pink’
colours[0] //’pink’
colours[3] //’purple’
```

Functions

A function is a block of code which can take input and produces output.
Input to functions are known as arguments, and are passed in through a pair of brackets:

```
const addTen = (x) => {
    return x+10;
}

addTen(20) //30
```

There is an alternative function syntax that was used in the earlier versions of JS and still works and will often be seen in code examples:

```
const function addTen(x){
    return x+10;
}
```

jsbin for online testing of client-side code

We can test code by using jsbin – an open source, free online tool primarily used for testing HTML, CSS and JavaScript.

You don’t have to register to use it but if you do then you can share your code with others.

You can test and modify the code snippets in these slides in jsbin during the workshop.

Adding JS to an HTML page 1

JS code is added to an HTML page using a <script> tag.

```
<script>
//JS code goes here
</script>
```

It is similar to the way we add CSS to an HTML page, which uses the <style> tag. Like CSS, we can add code directly in the HTML file or we can link to an external JS file:

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

Adding JS to an HTML page 2

One issue that can arise is that the JS code runs before the HTML page has fully loaded. This can cause issues if the JS code refers to HTML elements that have not yet loaded in the browser.

There are a number of ways to manage this issue.

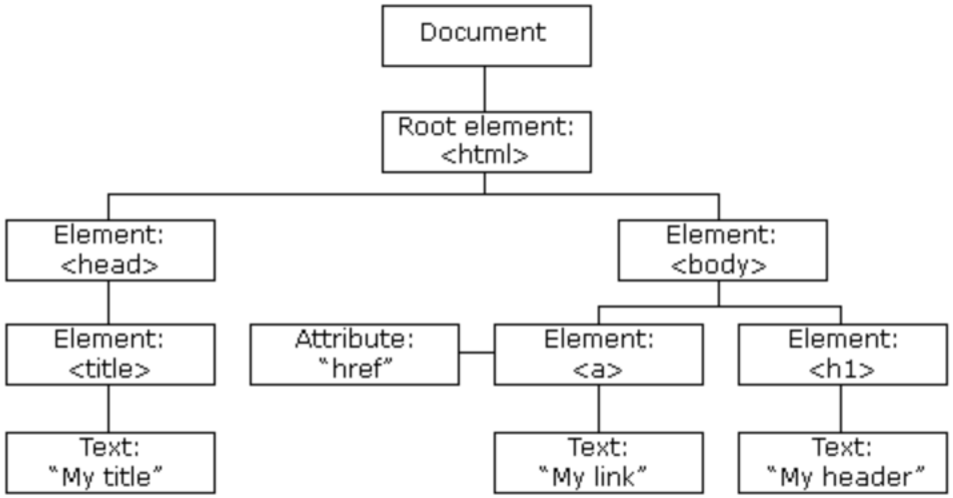
If the JS code is in an external file then load it with this code:

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

If the JS code is defined in the HTML page then we can put the <script></script> tags at the bottom of the body tag so that all of the HTML should have loaded before the JS code is reached and executed.

HTML Document Object Model (DOM)

HTML Document Object Model



When an HTML page is loaded, the browser represents it as a tree of objects. JS can access and manipulate all of these objects. This means JS can:
change and remove all the HTML elements and attributes in the page;
add new HTML elements and attributes;
change all CSS styles;
react to HTML events.

DOM: document.getElementById

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML
element.</p>

<button type="button"
onclick="document.getElementById('demo').style.fontSize=
'35px'">Click Me!</button>

</body>
</html>
```

DOM: Displaying hidden elements

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h2>What happens if you click the button?</h2>
<p>JavaScript can show hidden HTML elements.</p>

<p id="demo" style="display:none">Hello JavaScript!</p>

<button type="button"
onclick="document.getElementById('demo').style.display='
block'">Click Me!</button>

</body>
</html>
```

DOM: getElementsByTagName

```
<!DOCTYPE html>
<html>
<body>

<h2>Finding HTML Elements by Tag Name</h2>

<div id="main">
<p>The DOM is very useful.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
</div>

<p id="demo"></p>

<script>
let x = document.getElementById("main"); //selects the <div> with id="main”
let y = x.getElementsByTagName("p"); //selects all the paragraphs in the div
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;
</script> //updates the contents of the paragraph with id="demo”

</body>
</html>
```

JavaScript Object Notation (JSON)

JavaScript Object Notation

JSON (JavaScript Object Notation) is a widely used data format for web applications.

JSON is text so it can be used to exchange data between a server and a client browser.

It is easy to read and write for humans.

As it is text it can be parsed and generated by most programming languages.

You'll be parsing some JSON data in the workshop.

JSON

JSON syntax is derived from JavaScript object notation syntax:

Data is in name/value pairs

Data is separated by commas

Curly braces hold objects

Square brackets hold arrays

JSON

RFC 7159: JavaScript Object Notation

(MIME: application/json)

```
START  =  {   [
END    =  }   ]
SEP    =  :   ,
QUOTE  =  "   \
```

```
{ "name": "David",
  "id": 101,
  "units":
  ["COMS10010", "COMS2XXXX"]
}
```

JSON versus XML

```
{"employees":[
 { "firstName":"John", "lastName":"Doe" },
 { "firstName":"Anna", "lastName":"Smith" },
 { "firstName":"Peter", "lastName":"Jones" }
]}
```

```
<employees>
 <employee>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
 </employee>
 <employee>
  <firstName>Anna</firstName>
  <lastName>Smith</lastName>
 </employee>
 <employee>
  <firstName>Peter</firstName>
  <lastName>Jones</lastName>
 </employee>
</employees>
```

JSON versus XML

JSON doesn't use end tags

JSON is shorter

JSON is quicker to read and write

JSON can use arrays

The biggest difference is:

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function or in Java by the Gson library.

You'll try this out in the workshop exercises.