COMP 2015

Introduction to JavaScript and jQuery

Contents

Schedule

Day 1

Lesson 1
Lab 1 (due before the beginning of Lesson 2)
Assignment 1 (due before the beginning of Lesson 4)
Assignment 2 (due before the beginning of Lesson 6)

Day 2

Quiz 1
Lesson 2
Lab 2 (due before the beginning of Lesson 3)

Day 3

Quiz 2
Lesson 3
Lab 3 (due before the beginning of Lesson 4)

Day 4

Quiz 3
Lesson 4
Lab 4 (due before the beginning of Lesson 4)
Assignment 1 due

Day 5

Quiz 4
Lesson 5
Lab 5 (due before the beginning of Lesson 4)

Day 6

Lesson 6
Assignment 2 due
Final Exam

Grades

Grades will be available on http://my.bcit.ca

Labs

Lab 1	6%
Lab 2	6%
Lab 3	6%
Lab 4	6%
Lab 5	6%
Labs total	**30%**

Assignments

Assignment 1	10%
Assignment 2	10%
Assignments total	**20%**

Quizzes

Quiz 1	7%
Quiz 2	7%
Quiz 3	8%
Quiz 4	8%
Quizzes total	**30%**

Final Exam

Final Exam	**20%**

Submitting Homework

Submit homework to BCIT's D2L (Desire to Learn) dropbox folder for this course.

It is located in our course section of COMP2015 at http://learn.bcit.ca (COMP2015 > Activities > dropbox).

Instructor Contact Information

Your instructor can be reached at chrisharris@sent.com

Please do not use D2L or my.bcit.ca to contact the instructor. BCIT's mail server strips any JavaScript out of emails and email attachments.

If you have code-related questions, put JavaScript without the script tags into the body of the email instead.

Introducing JavaScript

- JavaScript was originally called LiveScript
- JavaScript has nothing to do with Java
- JavaScript was invented by Netscape in 1995
- JavaScript was the first client-side scripting language
- JavaScript runs in the browser
- There are essentially millions of types of browsers
- Each browser has a different way of running JavaScript
- One of our most difficult challenges is getting our scripts to behave similarly in every browser
- ECMAScript is the standardized version of JavaScript

What JavaScript Is Allowed To Do

- AJAX
- DOM
- HTML
- CSS
- JSON
- Timers and event handling

What JavaScript Is Not Allowed To Do

- Read (or write to) the client's file system
- Cannot hide JavaScript
- Send emails without the client's knowledge/consent
- Close browser windows it did not open
- Execute programs on the client's computer
- Open tiny windows (less than 100px x 100px)
- Establish connections to other computers
- Set file-upload inputs
- Read the browser's history (links, yes; read, no)

Useful Sites/Resources

Mozilla Developer Network – https://developer.mozilla.org/en-US/docs/Web/JavaScript

ECMA Standards – http://www.ecmascript.org

JavaScript Lint Tool – http://www.jslint.com

Online Help – http://stackoverflow.com

Atom Editor – https://atom.io

Emmet – https://emmet.io

Our First JavaScript Scripts

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <script>
        alert('Hello World');
    </script>
</body>
</html>
```

In between the <script> and </script> tags, only JavaScript can appear. Do not put HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <script>
        <h1> Error! </h1>
    </script>
</body>
</html>
```

Where To Write JavaScript

We can write JavaScript in the `<head>` section, or the `<body>` section, or in an external `<script>` section (its own file).

Problem

Although the head section is very often a great place to put JavaScript code, some changes need to be considered first.

The `<head>` section of a document gets fully loaded before any of the `<body>` section gets loaded. Because of this, JavaScript running in the `<head>` may not be able to target elements in the body.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        document.getElementById('myH1');
        // ERROR! Because the head is loaded before
        // the body, the h1 tag has not been created yet
    </script>
</head>
<body>
    <h1 id="myH1">My Heading</h1>
</body>
</html>
```

Solution

We can move the script *after* the h1 tag, or we can *delay* the script in the head using `window.onload`

Moving the script after the tag:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h1 id="myH1">My Heading</h1>
    <script>
        document.getElementById('myH1'); // Works!
    </script>
</body>
</html>
```

Using `window.onload`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        window.onload = function() {
            document.getElementById('myH1'); // Works!
        }
    </script>
</head>
<body>
    <h1 id="myH1">My Heading</h1>
</body>
</html>
```

External Scripts

We can also put our JavaScript into an external file and run it that way.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="myJSfile.js"></script>
</head>
<body>
    <h1 id="myH1">My Heading</h1>
</body>
</html>
```

Similarly, external JavaScript files can be delayed using the *defer* attribute. Deferred scripts will begin downloading right away, but will not execute until the entire document is rendered. After the page is finished, all deferred external JavaScripts will execute in the same order in which they were declared.

Note: you can only declare external scripts by using the *src* attribute with a filename and path.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="myJSfile.js" defer></script>
            <!--  OR  -->
    <script src="myJSfile.js" defer='defer'></script>
</head>
<body>
    <h1 id="myH1">My Heading</h1>
</body>
</html>
```

Comments

Comments are “notes” that are ignored by JavaScript and are intended for humans to read instead.

There are three types of comment tags used in JavaScript:
1. // two forward slashes comments out everything until the end of the line
2. <!-- this acts exactly the same as //
3. /* comments out everything until */ is found. This can span multiple lines

Variables

Variables are like containers that can store data (values) for us. We can declare variables using the `var` keyword, and in JavaScript we can freely assign and change the data type.

In addition, we can use the `typeof()` method to check what kind of data type our variable currently holds.

```
var x = 10;  // Declaring a variable named x, equal to 10
typeof(x);   // Returns 'number'

x = 'BCIT';
typeof(x);   // Returns 'string', i.e. a sequence of characters
```

We will see later in the class that variables can also contain *arrays* (lists of things) and references to *objects*

Getting and Setting Values

When writing JavaScript, there are two fundamental concepts that we will need to use and understand: *getting* and *setting*.

We may need, for example, to ask the browser how wide the current window is. To accomplish this, we can *get* the value by calling one of JavaScripts built-in properties, `window.innerWidth`. We can choose to store this value in a variable if we need to use it later in the script, but this is optional – it is perfectly acceptable to just ask for, i.e. *get*, this value without storing it.

Example:

```
// 'Getting' the current window width
console.log(window.innerWidth);

// Optionally storing the value in a variable
var currentWidth = window.innerWidth;
alert(currentWidth); // an example of how we might later use this value
```

In contrast to *getting*, *setting* a value always implies that we are using '=' (called the 'assignment operator') to set a property.

If, for example, we wanted to *set* the background color of the first paragraph on the page, we can use the assignment operator to either store the value directly, e.g.:

```
document.getElementsByTagName('p')[0].style.backgroundColor = 'red';
```

or we can optionally create a variable that references the first paragraph and use a combination of *getting* and *setting* to set the property on our variable, e.g.:

```
// Using document.getElementsByTagName() to get a reference
// to the HTML element and storing it in a variable
var firstParagraph = document.getElementsByTagName('p')[0];
// We can now use firstParagraph to set properties on our element,
// rather than having to repeatedly write document.getElementsByTagName('p')[0]

// Setting the backgroundColor property of our variable,
// (a reference to the element on the page) to red
firstParagraph.style.backgroundColor = 'red';
```

We can also use a combination of getting and setting in one line if we do not need to create variables for use later in the script. To change the color of the first paragraph to match the color of the second paragraph, we could use:

```
<p id="p1">First Paragraph</p>
<p id="p2">Second Paragraph</p>

<script>
document.getElementById('p1').style.color = document.getElementById('p2').style.color;
</script>
```

Objects

Objects are essentially variables that can hold multiple values, called **properties**. They also have **methods**, which allow us to *use* the objects for various purposes.

Note: you will be able to tell you are interacting with an object when you see what is called 'dot notation'.

An example of dot notation:

```
document.getElementById('myH1');
// say it aloud as document DOT get element by ID...
```

There are four important objects we need to familiar ourselves with:

document

The HTML page

```
document.getElementById('myH1');
```

window

The current containing window

```
window.alert('Hello World');
```

style

The CSS properties of an element

```
document.getElementById('myH1').style.backgroundColor = red;
```

navigator

The browser

```
navigator.appName;
```

Console

Another useful object is the console, which can (and should!) be used to:

- Try small bits of JavaScript code quickly in the browser without having to create files
- Debug your code by testing values and types of variables
- Checking messages generated by the browser when your code contains errors

The console can be activated using control-shift-j (Windows) or option-command-i (Mac), depending on which browser you are using.

The `console.log()` method can also be used in your scripts to print out values that you would like to test.

Mathematical Operators

```
var x = 20;
var y = 7;

*   multiplication  e.g. x * y = 140
/   division        e.g. x / y = 2.85714...
%   modulus         e.g. x % y = 6 (x and y must be integers)
+   addition        e.g. x + y = 27
-   subtraction     e.g. x - y = 13
++  increment       e.g. x++ (x equals 21 now)
--  decrement       e.g. y-- (y equals 6 now)
```

The + operator can also be used to 'concatenate' (i.e. join together) two strings.

example:

```
var firstName = 'Jason';
var lastName = 'Harrison';

alert(firstname + lastName);  // alerts JasonHarrison

alert(firstname + ' ' + lastName); // alerts Jason Harrison
```

Next Week

Lab 1 due before the beginning of Lesson 2.

Download it from our course section of COMP2015 on D2L http://learn.bcit.ca (COMP2015 > Content > Lesson 1)

Quiz 1 at the beginning of Lesson 2.

Quiz topics can also be found on D2L under COMP2015 > Content > Lesson 1