

Web Design (COMP 20030)

Practical 4

JavaScript

With JavaScript, a web page really can react to what you're doing: images can swap when you move a mouse over them, form elements can influence each other on the fly, and calculations can be made. With the use of libraries like jQuery, JavaScript applications can be pretty powerful. [Don't believe me?](#)

JavaScript is a scripting language originally developed by Netscape (Mozilla's predecessor) that works in all web browsers. With the notable exception of Node.js, an important thing to note is that all JavaScript is executed by the browser: that is to say that all the computation work is done on the computer that contains the browser.

Even though it's simple to work with, JavaScript is a complete programming language and includes variables, if statements, loops, arrays, and functions. Remember though that

JavaScript is not Java.

The name was an advertising gimmick: so Java is to JavaScript what butter is to butterflies. Java Applets and Servlets may be based on the web, but they, too, have nothing to do with JavaScript!

Different browsers, and even different versions of the same browser, may deal with JavaScript differently. Also, as both JavaScript and HTML were designed to be easy for developers to use, you may write horrible code in either that may seem to work; but will ultimately prove erratic and buggy. Writing good code, and performing cross platform testing are the only ways to mitigate against this.

Note that the **highlighted** parts of this practical are designed for people without any programming experience.

tl/dr: The following is a **LONG** document detailing basic JavaScript; suitable for people who have never programmed before. For C programmers: some stuff will be obvious, but be aware that other stuff won't.

Download the pdf and use the bookmarks for navigation. Questions are at the end and refer back to earlier parts.

Like HTML, JavaScript is just text that can be typed into a text editor. JavaScript code is not compiled (at least not at our end): the code will sit there waiting to be parsed by the browser in much the same way as HTML and CSS.

JavaScript can go right into the HTML that describes your page; both within the head and body (there are various advantages and disadvantages in where you choose to put it). Equally, it can be referenced from an external .js sheet much in the same way that CSS are used.

The beginning of a bit of JavaScript within a web page starts with

```
<script type="text/javascript">
```

Everything between `//` and the end of a line is a JavaScript comment, and will be ignored. Comment your code! A basic rule of good programming style (in any language) is that you should always think about the next person who has to look at your script. It might be a friend, a co-worker, an employer, or it could be you in three months. The easiest way to make sure you'll understand your own script in three months is to comment freely and often. If you want to comment a huge block of text, you can put it between `/*` and `*/` like this:

```
/* this is a block of text  
that I've commented out */
```

`alert("better, stronger, faster");` calls up a simple dialog box with the words "better, stronger, faster" inside. Here's your first piece of JavaScript: the alert method. You need to know that you should end it with a semicolon, and put the text you want in the alert box between quotes.

JavaScript on a web page ends with the `</script>` tag.

Introduction to Variables

Variables are simply the way JavaScript stores information. For example, if you write `x=2` then `x` is a variable that holds the value 2. If you then say `y =x+3` then `y` will hold the value 5. Here's an example of JavaScript that uses variables.

```
<script type="text/javascript">
```

```
    // make some variables
```

```
    var secs_per_min = 60;  
    var mins_per_hour = 60;  
    var hours_A_day = 24;  
    var days_each_year = 365;
```

```
    /* do some calculations and  
       save the results in new variables */
```

```
    var secs_per_day = secs_per_min * mins_per_hour * hours_A_day;  
    var secs_per_year = secs_per_day * days_each_year;
```

```
</script>
```

The first line here is a comment. This comment states the obvious, but it's nice to block off chunks of variable declarations.

Variable declarations

C programmers: note lack of main()

There are a few things to notice about the variable declarations:

- The first time you use a variable, you should declare it with the word "**var**."

For people who already know how to programme in C or Java: note that Javascript is dynamic and weakly typed; that is to say that variables do not have explicit types (like int and float). The browser will try and infer the type of the variable based on circumstances. The "var" keyword declares the variable in local scope.

Although declaring variables with var is not strictly necessary, it's usually a good idea (most things in JavaScript are "not strictly necessary" since JavaScript will try to run *no matter what*: but it's a good idea to keep code as correct as possible to prevent inconsistent results and cope with the varying versions of JavaScript. Variables must start with either a letter or the underscore character. After the first character, variables can have numbers. So monkey_23 is a fine variable name.

- Variable names are case-sensitive.

- Variables should describe what they are.

Variable names such as x, y, or blahblahblah aren't very useful to a person who's trying to figure out your script. Don't make your variables so long that they take forever to type, but make them long enough to be descriptive.

- You can give a variable a value when you declare it, or wait until later.

In the example, each variable was given a value the first time it was declared. You don't have to do this, and we'll see examples later on where it's nice to declare a variable even though we don't know the value right away.

- Statements end with a semicolon.

Statements are the sentences of JavaScript and semicolons are the end punctuation marks. Spaces and line breaks are ignored by the JavaScript interpreter, so the layout of the script serves only to make it more legible for people. This entire example could have been written in one really long line if you take out the comments. But that would be impossible to read.

At the end of the code above we see some basic maths. After JavaScript executes these statements, the variable secs_per_year will contain whatever you get when you multiply 60, 60, 24, and 365. From this point on, whenever JavaScript sees the variable secs_per_year, it will substitute in that huge number.

Now that we've defined the variables, let's do something with them. Here's how we can use JavaScript to write variables and HTML to a Web page:

```
<script type="text/javascript">
    // here's how to use JavaScript to write HTML

    var some_number = 87587;
    document.writeln("<p><b>The monkey dances ");
    document.writeln(some_number);
    document.writeln(" times before collapsing.</b></p>");
</script>
```

Here are the interesting points about these three lines:

document.writeln() writes whatever's between the parentheses to a web page.

Characters inside quotes are considered to be strings (see below); characters not inside quotes are considered to be variable names.

This example uses double quotes ("), but it could also use single quotes ('). The two are interchangeable (unless you're working with JSON, which you won't be, here).

You can write HTML with the `document.writeln()` command.

Note the `` and `` tags in the first and third lines. This writes to the page; meaning that the end result will be parsed as HTML. Writing HTML to a page in this manner is not exactly good practice, in general, but is nonetheless an interesting exercise here.

A frequent question is, "Where should JavaScript go in a page?"

You cannot go far wrong by putting all of your JavaScript at the bottom of your web page (just before the closing of the body tag), as this ensures that the web page is loaded faster. When loading a script, the browser can't continue on until the entire file has been loaded. Thus, the user will have to wait longer before noticing any progress. Note for future reference that it is generally preferable to keep javascript in external sheets, which are imported in the `<head></head>` like external CSS. However, that won't be necessary for this practical. The important thing to ensure though is that, regardless of where it is, all of your JavaScript is kept together for the sake of organisation, maintainability and readability.

The Magic of Strings

Any group of characters between quotes is called a string. Either single or double quotes will do. Just as variables can hold numbers, they can also hold strings. So it's legal to say:

```
var bad_monkey = "The monkey scowls at you and burps.";
```

Sticking this statement into a JavaScript declares the variable `bad_monkey` and assigns the string of characters, describing the monkey to it. Once you've done this, you can write

```
document.writeln(bad_monkey);
```

whenever you want JavaScript to write out the message about what bad monkeys do. [What's with this practical's obsession with monkeys?](#)

Here's an example of what you can do with strings.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Practical 4</title>
</head>
<body>
  <h1>JavaScript Test</h1>
  <script type="text/javascript">
    /* get a name */

    var monkey = prompt("What's the monkey's name?", "The monkey");

    /* declare some short strings*/

    var demanding = " demands, no, insists, upon receiving ";
    var requesting = " nicely asks for the benefit of all ";
    var tech = " a computer that won't crash, and a homemade browser!";
    var peace = " love for everyone and peace on earth.";

    /* construct some longer strings */

    var tetchy_monkey = monkey + demanding + tech;
    var hippy_monkey = monkey + requesting + peace;

    /* make a fancy string */
    var shouting_hippy = hippy_monkey.toUpperCase();
    document.writeln(shouting_hippy + "<br />");

    /*givemecolor function */

    function givemecolor(thecolor,thetext)
    {
      return '<span style="color:'+thecolor+'\">'+thetext+'</span>';
    }

    document.writeln(givemecolor('red', tetchy_monkey));
    document.writeln(givemecolor('pink', hippy_monkey));

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

- The script starts with something new:
var monkey = prompt("What's the monkey's name?", "The monkey");

Here we're calling the prompt method to get input from a user. When the prompt method is called, it brings up a dialog box that asks for a user's input. When the user hits OK, the prompt returns whatever is in the input box. In the above line, this returned value gets put into the monkey variable.

Notice that the prompt method takes two parameters, both of which are strings. The first parameter is what gets printed above the input field in the dialog box. In this case it's, "What's the monkey's name?" The second parameter, "The monkey" in this example, is set as the default value of the input box. If you don't want a default value, just put two quote marks as the second parameter. Like this:

```
var monkey = prompt("What's the monkey's name?", "");
```

The next lines are a few straightforward variable assignments, like we've seen before. After these lines we see:

var tetchy_monkey = monkey + demanding + tech;

This line introduces a string operator: the plus sign. When the plus sign appears between two strings, or two variables that contain strings as above, it means "concatenate." So the above line creates a new variable called tetchy_monkey that contains a string made of whatever the three variables contain. In this case, it's "The monkey" + " demands, no, insists upon receiving" + " a computer that won't crash, and a homemade browser!" In other words,

var tetchy_monkey = monkey + demanding + tech;
is the same as saying

var tetchy_monkey = "The monkey demands, no, insists, upon receiving a computer that won't crash, and a homemade browser!";

Beneath the "make a fancy string" comment you can see that JavaScript has inbuilt ways to manipulate strings.

var shouting_hippy= hippy_monkey.toUpperCase();

This particular var is given the value of whatever is in hippy_monkey, but with all the letters uppercase.

User Defined Function

Following the writing of shouting_hippy we can see an example of a JavaScript function called givemecolor. This function takes in a string (which will be the name of a HTML colour) and a second string called thetext, and applies the specified colour to thetext.

We could just write

```
document.writeln(<span style="color:'red'\">>'+tetchy_monkey+'</span>');  
and that would be exactly the same as
```

```
document.writeln(givemecolor('red', tetchy_monkey));  
but that's not as easy to read or use.
```

While we have document.writeln() call givemecolor() above, it is also very easy to allow users to call functions by adding an onclick attribute to some html element (or onsubmit in the case of forms)

<button onclick="someFunction()">Click me to run my function!</button>

Branching

Branching that involves "if-else" allows your script to behave very differently based on certain conditions. For instance, you could make the code that runs in your script be dependent upon what a user inputs. Here's the basic form of an if-else statement:

```
if (a particular condition is true)
{
    //do something
}
else if (the condition above isn't true, but this new condition is true)
{
    //do something different
}
```

The important parts of this structure are:

- It starts with the word "if" (if must be lowercase).
- There is a condition in parentheses that is either true or false.
- There is a set of statements that should be executed if the condition is true. These statements go between curly brackets.
- Remember, spacing is only there to make the script more legible. You can put the entire if-else statement on one line if you want, but that would be difficult to read.

Here's an example of an if-else statement in action.

If you type “yes” in the prompt box, you should receive a warm greeting before seeing the rest of the page. If you don’t type “yes”, you should get nothing.

```
var monkey_love = prompt("Do you love the monkey?", "Type yes or no");
```

```
if (monkey_love == "yes")
{
    alert("Welcome! I'm so glad you came! Please, read on!");
}
```

Note that the condition is two equal signs “==”. This is one of those things that everyone messes up initially. If you put one equal sign instead of two, you're telling JavaScript that monkey love should be given the value "yes" instead of testing whether or not it actually does equal "yes." Luckily, most browsers look for this sort of mistake and will warn you about it when you try to run your script. However, it's best not to make the mistake in the first place.

To confuse matters slightly, in JavaScript there is also the triple equals operator === for comparison of values. So how is this different from “==” ? In many ways you can consider triple equals to be “super true”: you can be certain that two values that are compared using this are genuinely equal, whereas using double equals they may not. This is because double equals will perform type conversion... so 0 == ‘’ is true! Due to the possibility of inconsistent and unexpected results, not to mention likelihood of worse performance, it is consequently recommended to always use triple equals for equality testing (or !== for the converse).

If you need more than one condition to be true you can use double ampersands “&&”:

```
if ((age > 18) && (age < 21))
{
    document.writeln(name + ": can vote, but can't drink.");
}
```

“&&” is the way in which you logically say ‘and’ in JavaScript. Notice also that the whole clause, including the two sub-parts and the ampersands must be enclosed in (parentheses). The > sign means ‘more than’ and < means ‘less than’, so the code will only run if age is between 18 and 21


If you want either one of two things to be true before running the statements in between the braces (in this case, the variable ‘something’) you need to use need to use ||

```
if ((something === "bananas") || (something === "JavaScript"))
{
    document.writeln("The monkey is happy because it has " + something);
}
```

Looping

If you want some piece of code to run multiple times you can put it into a loop. There are various types of loops, such as “while”, “for”, and “for in”, which operate slightly differently, but the for loop is sufficient for day-to-day usage.

The for loop requires three elements: a **counter**; a **condition**; and **something** that happens after every loop



```
for (var my_brilliantCounter = 0; my_brilliantCounter < 12; my_brilliantCounter++;)
{
    alert("this message will be printed 12 times");
}
```


Extra HTML: noscript, canvas

While JavaScript can be used in relation to any tag in a web page, two tags other than `<script>` are designed exclusively with JavaScript in mind. While it is safe to assume that any browser today will be able to understand what JavaScript is (unless it is crafted out of a potato), that doesn't necessarily mean that it is turned on in a user's web browser.

If JavaScript is disabled, anything written between `<noscript></noscript>` is displayed on a webpage. Generally this boils down to asking people to enable their JavaScript! As a caveat, do bear in mind that JavaScript may occasionally not work, and that you should therefore try not to have any critical aspect of your site (such as navigation) entirely dependent upon it.

`<canvas> ... </canvas>` is a tag, new to HTML5, which allows images and shapes to be drawn using JavaScript. HTML5 games are based upon the canvas element.

Canvases can be specified both height and width using the respective attributes

`<canvas height="100" width="10"> This text is displayed if your browser does not support HTML5 Canvas.</canvas>`

While canvases can be assigned to variables using `getElementById()` like other parts of the web page, an important aspect to note is that the drawing context must be separately specified (2d for two dimensional graphics, and `webgl1` or `webgl2` for three dimensional)

```
<script type="text/javascript">
    var canvasStepOne = document.getElementById("aParticularCanvas");
    var canvasStepTwo = canvasStepOne.getContext("2d");
    // and so on
</script>
```

The console

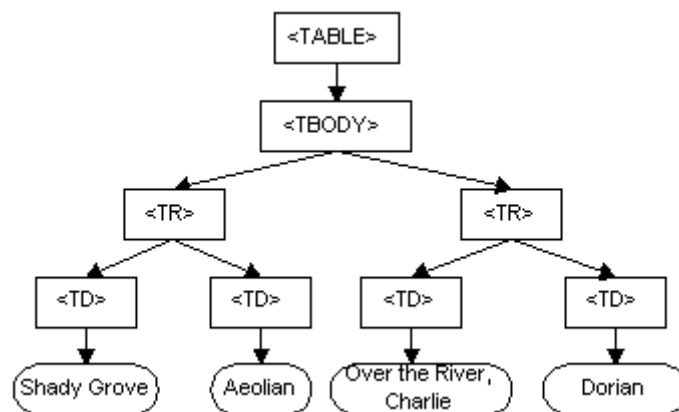
When dealing with HTML, CSS and JavaScript it is important to stick to recognised standards when writing code, as the developer has no control over how his code is interpreted or executed (as browsers do all of this work). Unfortunately, because every browser is developed differently, there is no way to produce a standard when dealing with inbuilt browser tools. In particular, using a browser's console (which, depending on the browser you are using, [can be accessed with the F11 key](#)), while a very valuable tool for developers, is entirely non-standard. Some browsers do not have consoles at all (very old versions of Internet Explorer can crash if you even try to use one). It goes without saying that, while fine in the development of websites; the console shouldn't be used as a method for trying to communicate with actual end-users. Of course, alerts are [generally to be avoided](#) in production websites.

Messages can be written to the browser's console typically by using [console.log](#)

JavaScript and the Document Object Model (DOM)

The DOM is a means of representing a page in a tree-like structure that allows programmatic access to the document elements. HTML elements that are nested within others are represented as *children* to the enclosing element's node in the DOM tree. As an example, the table defined by the following HTML code is represented as a DOM tree below it:

```
<table>
  <tbody>
    <tr><td>Shady Grove</td><td>Aeolian</td></tr>
    <tr><td>Over the River, Charlie</td><td>Dorian</td></tr>
  </tbody>
</table>
```



To access an HTML element using the DOM, we can use the name attribute of the element if it has one. For example, if a document has an image defined by ``, we can access that image by referring to it as `window.document.myimage` and access or change its properties (height, width, etc.) if we wish.

Your lecture notes have more comprehensive information on the DOM. For a list of all DOM objects and their associated properties, see www.w3schools.com/jsref/dom_obj_document.asp. Examples of DOM use can be found at the link below (note that some of these examples can be used to finish the exercises in this practical): developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Practical Exercises

Exercise 1:

Reproduce the 3 “[branching](#)” examples above (the ones in relation to both the monkey and age). Use a prompt to set the value of each variable you need. Save as q1.html

Exercise 2:

Create a function called my_checker:

This should take in two string variables, var first_name and var second_name

This should have the following rules:

- If the first_name is equal to “Donald” and second_name is “Trump”
 - An alert with the message "I love the poorly educated" should display
- If first_name is “Edward” and second_name is “Snowden”
 - Produce a prompt that says "would you mind collecting your cake from our office in Pennsylvania?"
 - If user responds "yes" say: "we'll even reimburse your plane tickets!"
 - Otherwise say "perhaps next time..."
- If the user has entered neither of the above names, display a dialog box with the message “carry on”

Ensure that my_checker(first_name, second_name) is called. The values of first_name and second_name can be set by using a prompt at the beginning of your JavaScript code.

Save as q2.html

Exercise 3:

Make a canvas 300*300 called “thisCanvas”. Add a button anywhere on the page. When the button is clicked a function called drawTheLine() will cause a line to appear in the canvas, going from the bottom left corner to the top right. You’ll have to make this function.

Add a second button. Similarly, when this button is clicked a function called drawTheRect() should draw a rectangle on the canvas. The position of this rectangle should be defined using canvas.width and canvas.height.

Add a third button. This should clear anything that is currently on the canvas using the [clearRect\(\)](#) function (this is a library function, you don’t have to make it).

Save as q3.html

Exercise 4:

HTML:

Create a web page. Have a html paragraph with the id **mySpecialParagraph**. This paragraph should contain the sentence “Well hello there, user”.

JAVASCRIPT:

Using the JavaScript function `getElementById()`, access the text of **mySpecialParagraph** using `nodeValue`. Create a paragraph element using `createElement()` and, using `createTextNode()`, append a string to this new element that reads “ See: I can use JavaScript”.

Using `insertBefore()`, insert this text before **mySpecialParagraph**.

Using the `textContent` property, overwrite the contents of **mySpecialParagraph** with “ which is obvious, as this was dynamically generated ”.

The webpage should render as:

See: I can use JavaScript

which is obvious, as this was dynamically generated

Now, using a loop, have the above var containing “ See: I can use JavaScript” print out 3 times in the browser’s console, using `console.log`. This time, however, the string should all be in uppercase (use `toUpperCase()`).

Save as q4.html