Week 6 Lab exercises – JavaScript Basics

Aims:

- Understand the structure of a JavaScript file and the purpose of each part.
- Link an external JavaScript file to an HTML page.
- Create function that responds to browser events.
- Create an interactive Web page with JavaScript prompts and message alerts.
- Use JavaScript to read user input and write back to an HTML page.
- Debug JavaScript using both the Firefox Error Console and Firebug.

Task 1: Dynamic User Interaction using JavaScript

Dynamic **behaviour** can be added to presentation of a Web page using **CSS**, e.g. with **pseudo classes** like `hover`, or through **JavaScript**. Here we will use JavaScript.

Step 1: Create an HTML file with user interaction

First we will create a simple HTML page that can take some user input. In a new folder **`lab06`**, create a Web page called **`clickme.htm`** with the appropriate **metatags**, **title**, **heading** and **paragraph** as follows:

Typically to display a button we would use a form with an `<input type="submit">`, but here we will use a `<button>` element instead, because we are not sending anything to the server.

Load the page into Firefox and validate the HTML before you proceed.

Step 2: Create a link to a JavaScript file in the HTML file

To provide separate behavior to a Web page using a JavaScript file, the Web page must have a reference to the JavaScript file (in a similar way it needs references to CSS files). Add the following line in the <head> section to create a link to the (as yet non-existent) JavaScript file called 'io.js' in the same folder as the HTML page:

```
<head>
      …
      <script src="io.js"></script>
</head>
```

Step 3: Create an external JavaScript file

Create the JavaScript file **`io.js`** using NotePad++ or similar text editor

1. Create a JavaScript template

Create a copy of the following JavaScript code template replacing the text in *italics* as appropriate:

```
/**
* Author: Your name and student Id
* Target: What html file are reference by the JS file
* Purpose: This file is for …
* Created: ??
* Last updated: ??
* Credits: (Any guidance/help/code? Credit it here.)
*/

"use strict";     //prevents creation of global variables in functions

// this function is called when the browser window loads
// it will register functions that will respond to browser events
function init() {

}

window.onload = init;
```

object — event — function name

2. Create a function that does something:

The pattern for a function definition is as follows:

```
/* Short comment on what the function does goes here*/
function myFunction(parameters) {
      // ...your JavaScript code to handle the event goes here
}
```

We will now use this pattern to display a prompt box. This box will take an input from the user then assign to a string. We will then display that string in an alert box.

Just above the `init()` function in your JavaScript file, create a function with no parameters called `promptName()` and insert the following lines between its braces:

```
var sName = prompt("Enter your name.\nThis prompt should show up when
        the\nClick Me button is clicked.", "Your name");
alert("Hi there " + sName +". Alert boxes are a quick way to check the
        state\n of your variables when you are developing code.");
```

What does this last parameter do?

Notice the use of the **`\n`** escape characters in the above strings to force the text onto a new line.

3. Register the function to respond to events in the browser:

The function you have written does not do anything yet because nothing is calling it. We will get our JavaScript to respond to an *onclick* event when the HTML button element with attribute `id="clickme"` is clicked.

When the HTML loads in the browser window, we need to register the elements on the HTML page that will generate events to which the JavaScript will then respond. There are many types of HTML *events* we can get JavaScript to respond to. Window object events like: *onload* or *onunload*; form object events: *onblur*, *onfocus*, *onchange*, *onsubmit*; mouse events like *onclick*, *ondblclick*, *ondrag* and so on. (Notice events do *not* use camelCase.)

We can register events inside the `init()` function. First we need to get a reference to the HTML object. One way to do this is to use the `getElementById()` method. A reference to the HTML object is then stored in a local JavaScript variable. The code pattern for this is:

```
var myVariable = document.getElementById("id_of_an_HTML_element");
```

We use can then use this variable to register a 'listener' function that will respond to an action of a user on the HTML page:

```
myVariable.eventType = myFunction;
```

Using the above pattern, register a function to respond to a click event on the button by adding the following lines to the init() function you created in your template above:

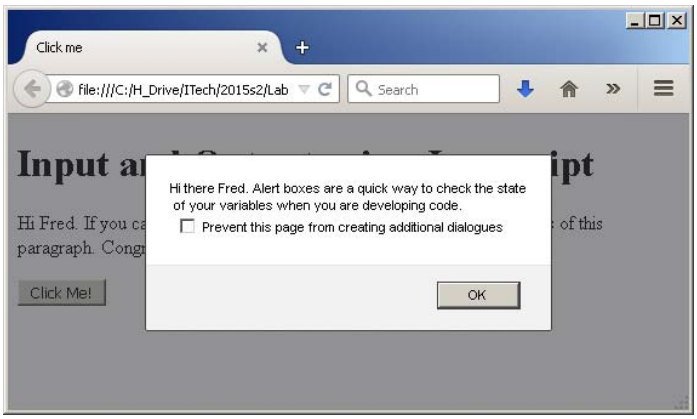
clickMe is a variable local to the `init` function. **Careful**: JavaScript is case sensitive!

```
var clickMe = document.getElementById("clickme");
clickMe.onclick = promptName;
```

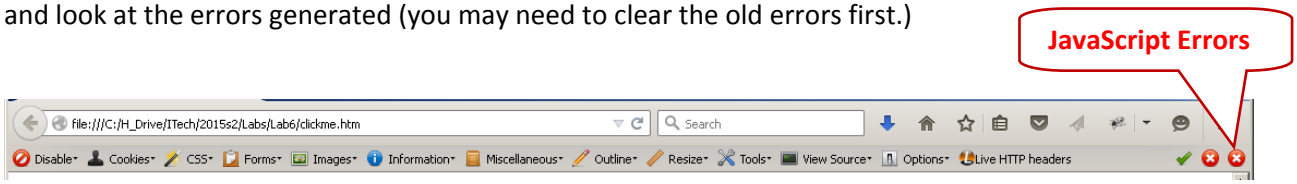
Name of the function you wrote. Remember: no brackets

Step 4. Test your program, debug, deploy to mercury and retest.

Save your JavaScript file then refresh the HTML file. The output should appear something like this:



If not, click on the JavaScript error icon ⊗ on the far right of the Web Development toolbar. Refresh the page and look at the errors generated (you may need to clear the old errors first.)



If you still can't fix all the errors, complete **Task 3 Finding Syntax Errors in JavaScript** then try again.

Upload your HTML and JavaScript files to a new **lab06** directory under the unit folder on the Mercury server **~/{your unit code}/www/htdocs**.

Open the page in Firefox, revalidate the HTML and test that the JavaScript works.

Step 5. Writing output to the Web page.

We will now write some JavaScript code that will change the content of the paragraph element on our Web page. We will do this in a new function.

1. Create an id for an HTML element

First we need to create an id attribute for our paragraph element in the HTML. Modify your file **`clickme.htm`** to add the attribute **`id="message"`** to the first <p> element. Resave your file and check that the HTML is valid.

2. Create a new function to write content into the HTML element

Create a new function called **`rewriteParagraph(userName)`**. Notice that unlike the function we created in Step 3.2, this function has a parameter called `username`.

Function definition

```
function rewriteParagraph(userName){
        //1. get a reference to the element with id "message" and assign it to a local variable
        // 2. write text to the html page using the innerHTML property
}
```

Write the implementation of this function. In the first line, get a reference to the paragraph element with id attribute "message" and assign this reference to a local variable called **`message`** (as you did in Step 3.3).

The **`innerHTML`** property sets or returns the HTML content (inner HTML) of an element. In the second line of your function write text to the html page using the innerHTML property as follows:

```
message.innerHTML = "Hi " + userName + ". If you can see this you have
    successfully overwritten the contents of this paragraph. Congratulations!!";
```

3. Invoke the new function

Unlike the **`init()`** and **`promptName()`** functions above, we will not directly register a browser event to invoke our new function. Instead we will make a function call.

Within the **`promptName()`** function add the following function call at the end:

```
rewriteParagraph(sName);
```

Function call

This will pass a copy of the value in the **`sName`** variable that we got from the user to the **`rewriteParagraph`** function (where it become the local variable **`userName`**.

4. Test your program, debug, deploy to mercury and retest.

The result should now look something like this:

Step 6. Another way to write output to the Web page.

From the example file of the first JavaScript lecture (lect5_html_io.html), observe how the **`textContent`** property can be used inside a **`<span>`** element to write into a placeholder on the HTML page.

1. Modify your **`clickme.htm`** file to include a with an appropriately named id attribute just above the button element. The span should sit inside a <p> element.

2. Create a new function called **`writeNewMessage()`**that will the display the message "You have now finished Task 1" into the span using the **`textContent`** property.

3. Modify your code so that when the user clicks on the <h1> element the **`writeNewMessage`** function is invoked.

4. Test your program, debug, deploy to mercury and retest.

The output should now look something like this:

Something optional to try:

We can use an alternative method to reference an element on an HTML page without having to create an id on the HTML page. The **`document.getElementsByTagName`**`("tagName")` method returns an array of elements that have the element tag name specified in the parameter. For example:

```
var click2 = document.getElementsByTagName("h1");
```

would return an array of references to all of the <h1> elements in the document. To get access to the first element in the array (with index = 0) we would write:

```
click2[0].onclick = writeNewMessage;
```

Note that as there is only one <h1> in our document, if we were to write **`click2[1]`** we would get an error as the second element does not exist.

This approach means we don't have to modify the HTML element, but rather references the position of the element in the HTML DOM. What is the *disadvantage* of this approach?

Task 2: Finding Syntax Errors in JavaScript

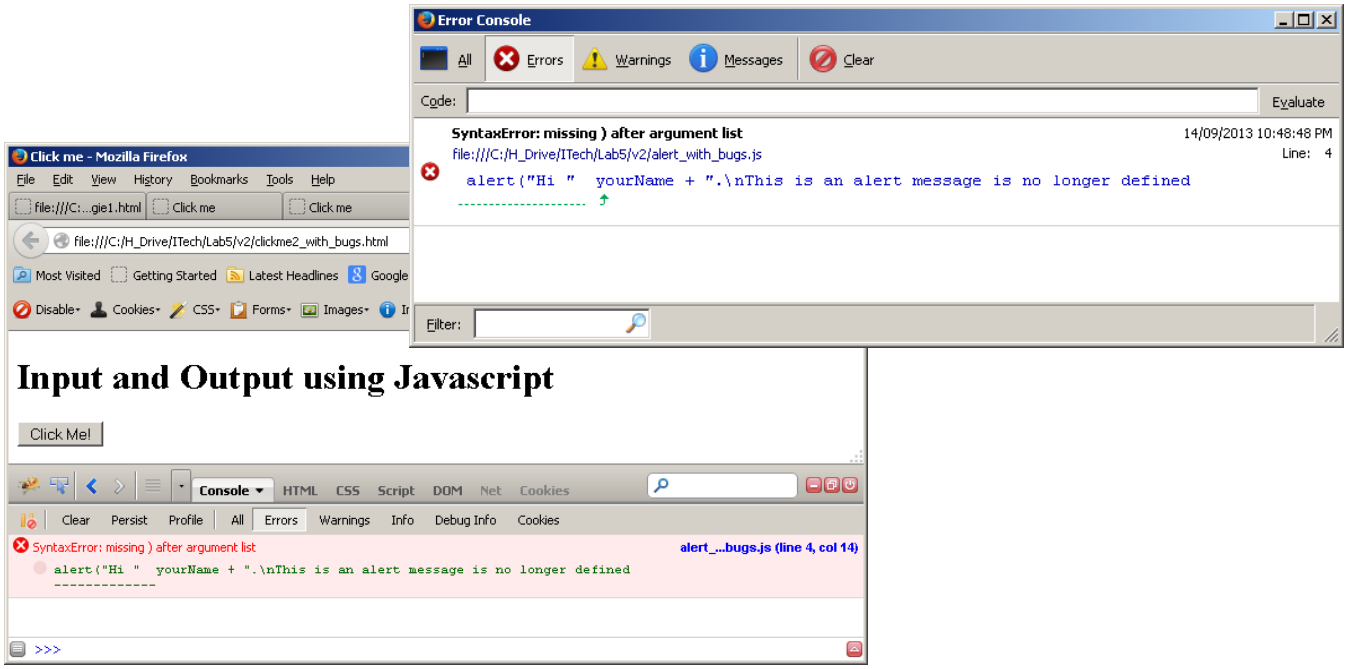
Using an HTML or CSS validator will not pick up errors in your JavaScript.
In this task you will use the Firefox *JavaScript Error Console* and a Firefox add-in called *Firebug*.

Step 1: Install FireBug

1. Using the Tools|Add-ons find Firebug (v1.12.1 or greater) and install it.
2. Press F12 (or click View|Firebug) to see the Firebug window.
3. Load the HTML file **clickme.htm** you created in Task1 and use the tabs to view the HTML and JavaScript (CSS can also be viewed when it exists).

Step 2: Bug Hunt

1. Download the files **alert_with_bugs.html** and **alert_with_bugs.js** (in lab06.zip) into your **lab06** folder. Load the file **alert_with_bugs.html** into Firefox. This file references the JavaScript file **alert_with_bugs.js** which has 4 syntax errors in it.
2. Use Firefox to identify error information, such as the line number and the nature of error.
   - Firefox: Select "Tools"-> "Web Developer" -> "Error Console",
     (Note: Error Console can be enabled in newer versions of Firefox with Ctrl-Shift-J
     See https://developer.mozilla.org/en/Error_Console)
   - Also use 'Console' in 'Firebug' Add-On to view the Error.



3. Note that not all bugs necessarily will be found at once, as one error might make the rest of the file difficult/impossible to parse.
4. Correct any errors you identify.

Hints:

- Be careful with quotation marks when concatenating string literals and variables
- Single quotation marks must match with single, and double with double.
- JavaScript is CaSe-sEnSiTiVe!
- The error may not always be on the line identified by the error checker but may cascade. This is the case with the last error in the JS file.

Show the web pages from Task 1 and 2 to your tutor so your work can be marked off as complete.