Lab 8 – JavaScript: Client-side storage and DOM

Aims:

- To practice using LocalStorage and Cookies to maintain 'state'.
- Use HTML5 client-side localStorage to store and retrieve user information
- Use client-side cookies to store and retrieve client-side data
- To gain experience setting breakpoints and watches with Firebug
- Use alternative methods to access elements in the DOM document
- Use some simple string manipulation functions

Task 1 in this lab need to be demonstrated to your tutor.

Task 1: Using HTML5 localStorage to store data on the client

In this task we will adapt the registration form for the Web site we developed in Labs 5 & 7.

HTML5 introduces new methods to store data locally within the user's browser.
In the past this has been done with cookies. However, HTML5 Web Storage is faster and more secure (see http://www.w3.org/TR/webstorage/). The data is not included with every server request, but used only when asked for. It is also possible to store large amounts of data, without affecting the website's performance.
If the client does not support LocalStorage, we may need to use Cookies to maintain 'state'.

Using Web Storage, data is stored in key/value pairs, and a web page can only access data stored by itself.
There are two new objects for storing data on the client:

- **localStorage** - stores data with no expiration date
- **sessionStorage** - stores data for one session

In this Task we will use **localStorage**.

In Lab 7 you developed a JavaScript file to check/validate data being submitted to a server from a form.
In this task you will extend the JavaScript to record submitted user data so that if the user revisits the site the personal data will automatically be filled in for them.

Step 1: Copy the Web form and associated files from Lab 7.

1. You need to have at least some JavaScript data checking/validation working for this exercise.
   If not complete Lab 7 first.
2. Copy the files you created in Lab 7 to a new folder called `lab08`.
3. Validate the HTML and CSS. Fix any errors.
4. Rename the file '`validate.js`' to '`validate_prefill.js`'
5. Update the script reference in the HTML file '`register.html`' to the new JavaScript file.

Step 2: Storing user data

In this step will we write a function that stores the values the user has entered into the form in **`localStorage`**. We will call this function at the end of the 'validate' function we wrote in Lab07.

1. Create a function called store_user() as shown below.
   This function simply reads the values of the input controls and writes them to localStorage.

```
function store_user(){
       //get values and assign them to a localStorage attribute.
       //we use the same name for the attribute and the element id to avoid confusion
       localStorage.username = document.getElementById("username").value;
       localStorage.age = document.getElementById("age").value;

       //add other elements here…

       alert ("Name stored: " +localStorage.username); //added for testing – remove later
}
```

2. Extend the function to store the rest of the user information input – that is **`species`**, **`age`**, **`beard`** and **`email`**.

> ***Hint***: Getting the species requires checking the value of each of the radio buttons. However, we have already determined what the **`age`** and the **`species`** are in the function **`validate()`** which will call **`store_user()`**. Why not just pass these values as parameters so the formal declaration looks like:
>
> ```
> function store_user(age, species){
>       …
>  }
> ```

3. Invoke the **`store_user`** function from the validate function.
   We only want to do this if the validation returns **`result`** returns true.

```
function validate(){

       var errMsg = "";                  /* stores the error message */
       var result = true;                /* assumes no errors */
       /*get values from the form
       ...    */
       var age = document.getElementById("age").value;
       var species = getSpecies();
       /*-----------------------------------------------------
       Other code from validate() implementation is here ...
       result is set to false if it does not validate
       ----------------------------------------------------- */

       if (result){      //if the form validates ok.
       //As result is a Boolean variable saying (result) is the same as saying (result == true)
              store_user(age, species);
       }
       return result;    //if false the information will not be sent to the server
}
```

4. Load **`register.html`** and run.

   As shown in Lab06, use ***JavaScript Error Console*** or ***Firebug*** to help debug any errors in your JavaScript code

Step 3: Prefill the form with stored data

If the user has previously visited the site with their browser we want the browser to automatically fill in their personal details (let's assume no one else is using the same browser – probably a bad assumption!). To do this when the page loads we will check if data exists in localStorage, and if so we will load it into the form.
This is just the reverse process of storing the data.

1. Create a function called **`prefill_form()`** as below:

```
/* check if localdata on user exists and if so prefill the form*/
function prefill_form(){
     if(localStorage.username != undefined){    //if localStorage for username is not empty
          document.getElementById("username").value = localStorage.username;
          document.getElementById("beard").value = localStorage.beard;
          document.getElementById("email").value = localStorage.email;
          document.getElementById("age").value =localStorage.age;
          switch(localStorage.species){
               case "Human":
                    document.getElementById("human").checked = true;
                    break;
               case "Dwarf":
                    document.getElementById("dwarf").checked = true;
                    break;
               case "Hobbit":
                    document.getElementById("hobbit").checked = true;
                    break;
               case "Elf":
                    document.getElementById("elf").checked = true;
                    break;
          }
     }
}
```

2. Call the **`prefill_form()`** function from within the **`init()`** function which is triggered when the window.onload event fires.
3. Run and test that the output is as expected.
4. 'Reset' the page and then reload it. Do the values preload from localStorage?
5. As localStorage is persistent between page views and sessions, the data should reappear on refresh or after the browser has been closed and reopened.
6. If your code does not work, use ***JavaScript Error Console*** or ***Firebug*** to help debug the errors.

Once you have completed the above task, show it to your tutor.

Task 2: Debugging JavaScript - using Firebug

Up until now we have relied on ***JavaScript Error Console*** or ***Firebug*** to help identify ***errors*** in our JavaScript. This checking will only pick up ***syntax errors*** (e.g. missing semi-colons or misplaced brackets).
It will not pickup ***logic errors*** in the code.

One way to track the logic of the program and identify errors is to examine the state of variables as the program executes, or to watch step by step the sequence of execution of the lines of code.

In Task 1 Step 2 we added an alert box to test whether or not data had been correctly stored in localStorage.

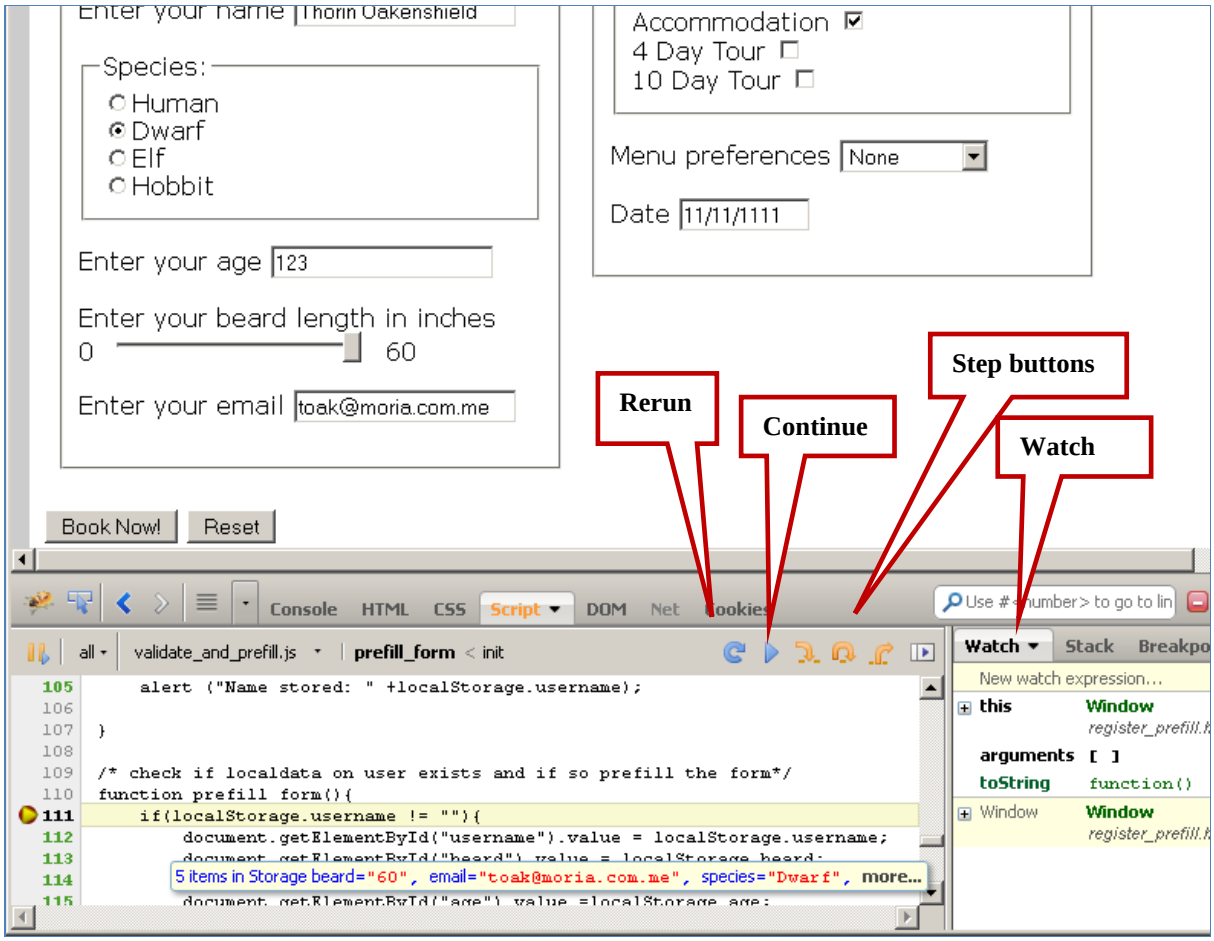
```
alert ("Name stored: " +localStorage.username); //added for testing – remove later
```

This however is very kludgy. A better more flexible way to do this is to use breakpoints, watch variables and stepping using tools such as Firebug.

Step 1. Setting up a breakpoint using Firebug

Breakpoints allow us to select lines in the program where program execution will pause when it reaches that point. We can then look at the state of variables and progressively execute the code line by line to check the control flow is as we expect.

1. Load the Firebug add-on as in Lab 6.
2. Load **`register.html`** developed in the previous task.
3. Press F12 to view/toggle the Firebug window.
4. Select the Script tab. You should be able to see the JavaScript associated with the HTML page – i.e. **`validate_prefill.js`** (you may need to enable JavaScript viewing and refresh the page).



5. Set a breakpoint on the first line of the function **`prefill_form()`** by clicking on the line number next to the line as shown in the snapshot below. The line will be highlighted.
6. Refresh the Web page. The execution of the script should pause at the breakpoint.
7. Scroll your mouse over the variable names in the code. The values of the variables will show as tooltips.
8. You then have a choice to continue program execution (the 'play' continue button as shown below), stepping through the code or rerunning the program.

Step 2: Create a Watch on a variable

1. Set a breakpoint on the line **`var result = true;`** in the **`validate()`** function
2. Select 'New watch expression' in Firebug as shown in the above screenshot and type **`result`**.
3. Refresh the Web page. Click 'Book Now'.
   The execution of the script should pause at the breakpoint.
   Step though the code and watch the value of **`result`** change.

Step 3: Finding out more

If you are familiar with debugging tools from other IDEs the Firebug tool should be easy to master.
If not, visit the Firebug JavaScript help page https://getfirebug.com/javascript and practice using the tool.
Being familiar with the tool will help make it ***much*** easier to debug and test your code, and is likely to save you ***lots of time*** when doing your assignment.

Task 3: Using Client-side Cookies with JavaScript

In this task we will use JavaScript to manipulate cookie data and examine various ways to access elements in a HTML DOM document. We will also use some string functions to extract the parts of a cookie string.

Exercise 1: Storing Cookie Data

Although *cookies* may be replaced to some extent by localStorage and sessionStorage in HTML5, they are still widely used, and needed if the client does not support HTML5 Web Storage.

A *cookie* is a "token" of information that can be exchanged between a client and a server.
A server normally sends a cookie to a client, so that the client can identify itself each visit.
However, for this task we are only going to explore the use of cookies as a way to store information for the client only (and share data between pages).

Cookies can be divided into five parameters, where each parameter uses a **`key=value`** pair.
Only one parameter is required (the **`name=value`** data); other parameters are optional.

```
name=value; expires=dateValue; domain=URL; path=pathName; secure=false
```

1. Unzip Lab08_Cookie.zip from Blackboard and open the existing file called "`cookie.html`".

2. The page contains a form and a section in the head for our JavaScript code. Create a new function called "`storeCookieData()`" (or similar) in the head script section and add a simple `alert("Testing!")` method call (or similar) to your function for initial testing.

Note: The form buttons used are type="button" (not type="submit") because we don't need to send (submit) the form, but this would not make sense for a normal form.

3. When the user clicks the "Store…" button we want to call our `storeCookieData()` function. To do this, set the button "`onclick`" event attribute to call your new store cookie function.

4. Test! If everything is fine, you should get your test `alert()` message when you click the "Store…" button.

5. Now let's add some serious code in the `storeCookieData()` function and actually create and store cookie data. Start by deleting or "commenting out" (using // or /* */) the test alert() messages.

6. Add the following code (or similar if you are confident).
The "key" we use is here is "firstname" and the "value" is taken from the form.

```
tmp = document.forms[0].firstname.value;
document.cookie = "firstname="+tmp;
```

Notes:

- *Don't forget to "refresh" your HTML file in the browser so that your changes to the JavaScript code are ready to be used.*
- *If strange cookie values show, you might need to clear your browsers cookies:*
  Firefox: Tools=>'Options' =>'Privacy'
  IE: Tools=>'Internet Options' => 'Advanced Privacy Settings'
- *We have used the "`tmp`" variable to hold the field value, but this is not essential.*

- *We must get the "value" of the "firstname" element not just the "firstname" element by itself. (This is a common mistake). Refer to a DOM reference for input elements.*

7. **Test!** Fill in a value, click "Store…" and then click "Show…" and see the result.
What happens to the cookie string each time we store a new value? What happens if there is no value?

Explanation:
We can access form elements several ways. For example, we can access the form elements through the form object with the `document.forms[]` array (our single form is stored in position 0 of the array) or `document.forms['myForm']` or the `document.getElementById('myForm')` method.
We can also directly access the form control elements, using say `getElementById('firstname')`.

Exercise 2: Multiple Cookies

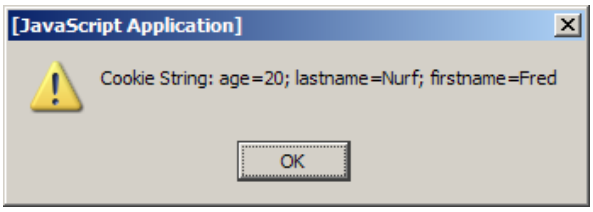
Now let's extend our cookie knowledge further and store multiple cookie `key=value` pairs.

1. Save the "`cookie.html`" file as "`cookie2.html`" and add two extra form elements for "`lastname`" and "`age`". (Remember to set the "`name`" and "`id`" attributes, and set the label as required).

2. We need to modify the `storeCookieData()` function to store the new form element values as well. Assign the `key=value` pair needed for each element one line at a time.
Use the code listed below (or similar) in your function.

```
...
document.cookie =
"firstname="+document.getElementById('firstname').value;
document.cookie =
"lastname="+document.getElementById('lastname').value;
document.cookie = "age="+document.getElementById('age').value;
```

3. Use the "Show…" button to see the alert results. It should look something like the alert box shown on the left below



Notes:

- *In some older browsers, the alert showed the last cookie `key=value` stored (the age) first, as in the alert box shown above on the right.*
- *Does using single ' or double " quotes make a difference to the code?*
Make sure you understand when it is important and why.
- *If you look at the cookie string in more detail, you will see that between each `key=value` pair that we stored are two important characters added by the browsers – a semicolon ; and a single space. We use this important aspect later to break up the cookie string!*

4. If you enter a form value that contains a semicolon '**;**' what happens to the cookie string?

5. If you enter a character like a '**=**' what happens to the cookie string?

So far we have only used `key=value` cookie information. We can also store other information with our cookie data such as an expiry date, which you can apply to each `key=value` pair that you set.

6. Create a date object, and use it to get our desired expiration date/time in GMT format (using the `Date.toGMTString()` method). The expiration date is set to 3 days from now.

```
expireDate = new Date();
expireDate.setDate(expireDate.getDate()+3); // 3 days from now
tmp = document.getElementById('firstname').value;
document.cookie = "firstname="+tmp+";
expires="+expireDate.toGMTString();
```

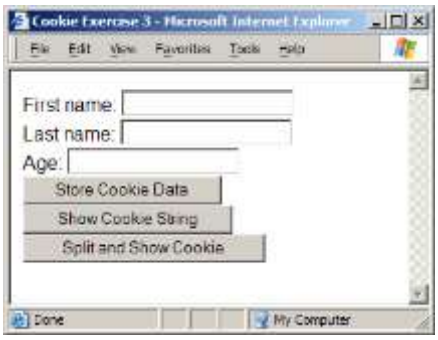
7. Modify the code to set the `expireDate` to a few seconds in the future using the Date object `getSeconds()/setSeconds()` and then test to see when the cookie is removed.

Notes:

- *If you set the expiration date to a value less than the current date, the cookie will expire!*
- *See online references for more details about the methods/properties of the Date object.*
- *Some browsers (eg. Chrome) will not store cookies when the page is run locally, only when it is loaded from a server.*
- *Some browsers (eg. Firefox) will show other (hidden) cookie values that are set when the page is loaded from a server. eg. load your pages onto mercury and see the difference.*

Exercise 3: Reading and Splitting Cookies

Now that we have multiple cookie values stored let's create some better code to break up the cookie string and the key or value we need. The `document.cookie` property is a **string**, and so there are several string methods that can help us including `substring()` and `split()`. We will only investigate using `split()` in this lab.



1. Save your "`cookie2.html`" file as "`cookie3.html`" and make sure that the expiration value of any cookies is okay still.

2. Add a new button to the form that will "Split and Show Cookie".

3. Create a new function called "`splitShowCookie()`" (or similar).

4. Use the `split()` method of a string object to divide a string into an array.
   Remember that we can use `"; "` as the separator.
   For example, use this code with a simple pretend cookie string of `"name=Fred; age=20"` as a way to start testing the `split()` method:

```
tmpStr = "name=Fred; age=20"; // test string
tmpArray = tmpStr.split('; '); // creates an array with two
positions
alert('key=value (0) is '+tmpArray[0]); // zero index, first
pair
alert('key=value (1) is '+tmpArray[1]); // second pair
```

5. That works, but it would be better if we didn't have to know how many key=value pairs we had to start with. So try this `for` loop around the `tmpArray` instead (or a `for..in` loop etc.):

```
tmpStr = "name=Fred; age=20"; // test string
tmpArray = tmpStr.split('; '); // creates an array with two
positions
for (i = 0; i < tmpArray.length; i++) {
  alert('key=value ('+i+') is '+tmpArray[i]);
}
```

6. Instead of the test string, now use the current `document.cookie` string.
   (You may have to set a cookie first though!).
   Test it. (Note that you might need to change the 'expires' for your cookie.)
7. So far we still have complete `key=value` pair strings, and so we need to split them again if we want only the `key` or only the `value`. Replace the previous `alert()` line in the `for` loop with the code listed below (or similar) which will split the `key=value` pair.

```
tmpKeyValue = tmpArray[i].split('=');
alert('('+i+') Key='+tmpKeyValue[0]+' Value='+tmpKeyValue[1]);
```

Notes:

- *First we split the `key=value` pair by the '=' character, then we show each part in the alert.*
- *You may be aware that we can directly split() an existing split result (using dot syntax) but this might not be useful is you are trying to find a specific key=value (you need to match). Example:*

```
tmpStr.split('; ')[0].split('=')[1]; // the value of the first pair
```

Optional:

If you only wanted a specific `key=value`, you could use a conditional test like:

```
if (tmpKeyValue[0] == 'age') { alert('Your age is '+tmpKeyValue[1]); }
```

See the Lecture Notes for another 'template' for Cookie code.

Optional Extras:

Instead of using alert dialogs (okay for testing but probably annoying after a while) create a paragraph with a specific id and try writing results to the `innerHTML` of the paragraph, using the `getElementById()` method to get the element we want. ***An even more challenging extra*** - try adding / replacing a `textnode`.

Optional Extras

Repeat Task 1, but instead of using localStorage, use Cookes instead.

Modify Task 1 again, merging the two solutions, so that if localStorage is not available then Cookies are used instead.