LAB 2: Working with the DOM and JavaScript events

Contents

Introduction

In this lab you will practice using JavaScript to explore and manipulate the DOM of a web page interactively.

The lab consists of three tasks.

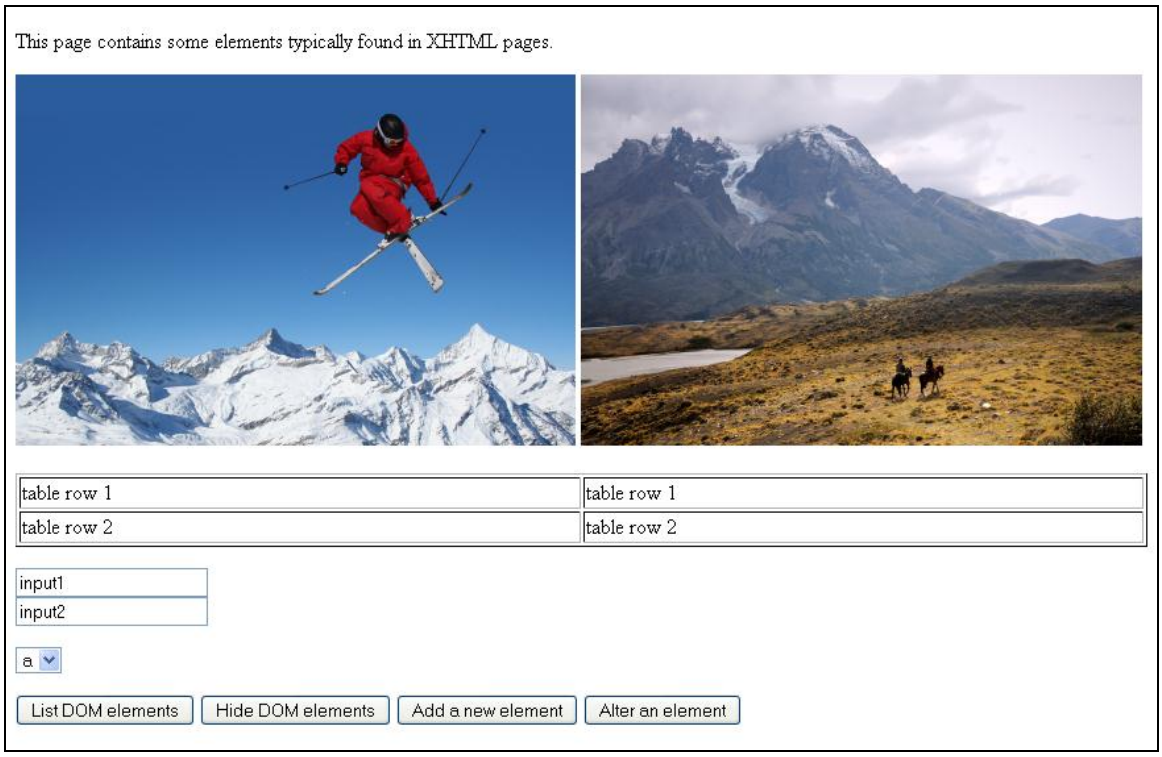
Resources

- Lecture notes and sample code from Module Website
- Visual Studio and Firefox/Firebug (or any combination of editor/browser)
- JavaScript reference, e.g. **https://developer.mozilla.org/en/JavaScript/Reference**

Task 1. Working with the DOM

1. Download *LAB_ClientJavaScript.zip* from the course website and extract the contents, which consist of a VS2010 web application project.

2. View the page *dom.html* in Firefox. The page contains a selection of page elements: paragraphs, images, a table, inputs



3. Open the DOM Inspector in Firefox (**Tools > DOM Inspector**) and explore the DOM nodes in the page and their attributes. You may need to install the DOM Inspector add-on within Firefox if the menu item is not present.

4. Open the file *dom.html* for editing in VS2010 and complete the following three exercises:

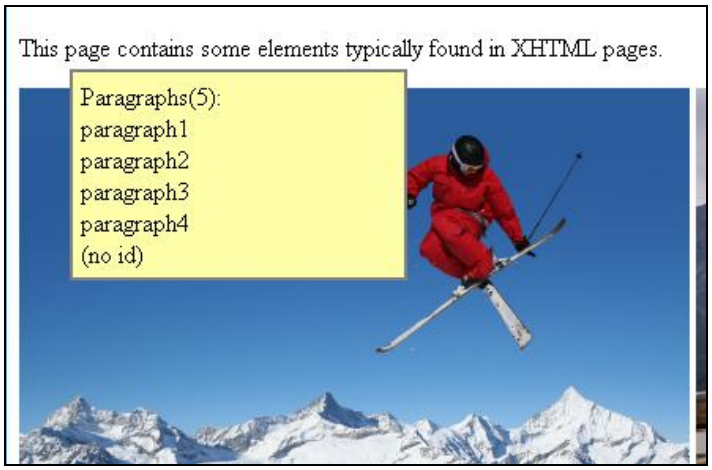
Exercise 1.1: Getting DOM information

The function *domView()* in *dom.html* is intended to create a list of the ids of all the paragraph elements in the page and display this list in a pop-up box (actually a DIV element which can be made visible when required). The box is shown and hidden when the **List** and **Hide** buttons respectively are clicked. The line:

```
nodeList = document.getElementsByTagName("p");
```

gets a list of paragraph elements and stores the list in the variable *nodelist*.

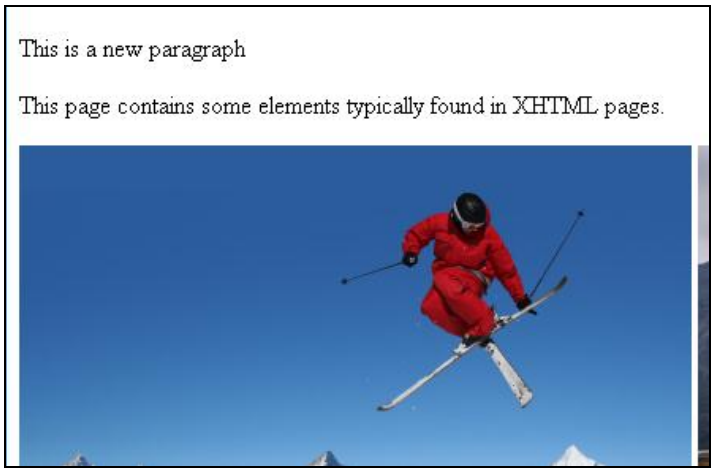
Complete the code for the function so that it produces the following result when the List button is clicked. You will need to set the initial value of *output*, and for each node in *nodelist* you will need to add the appropriate information to *output*.



Now **modify the code** so that it lists ids of (a) inputs and (b) table cells.

Exercise 1.2: Adding an element to the DOM

5. The **Add a new element** button in the page calls the function *addNewParagraph()*. **Complete this method** so that it adds a new paragraph before the current first paragraph in the page.



6. Now **add a new *textarea* element** to the page and modify *addNewParagraph()* so that the new paragraph contents are copied from the text entered by the user in the textarea.

7. Use the DOM Inspector to check that the new elements appear as DOM nodes.

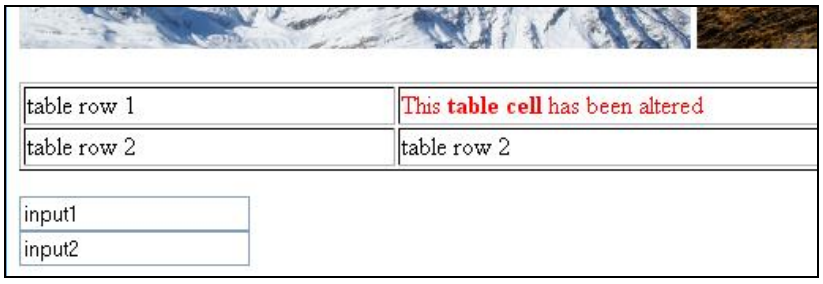
Exercise 1.3: Modifying a DOM element

8. **Add a new function** to your page to change the text of the second column in the first row of the table. You will need to find the correct element in the DOM and set its *innerHTML* property to change its contents.

   If you set the content of the cell to be a span element like this:

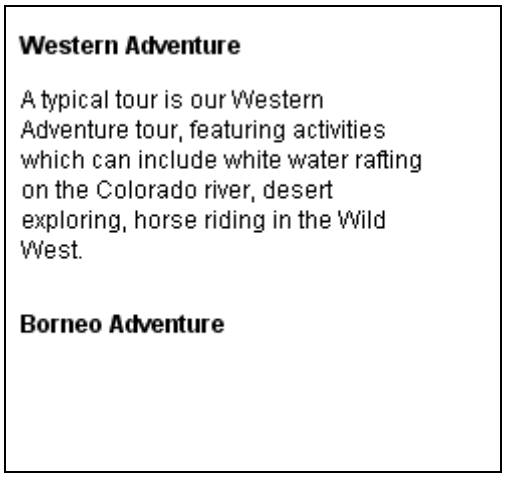
   ```
   <span id="redcell">...</span>
   ```

   then it will be styled as red text to make the change obvious.



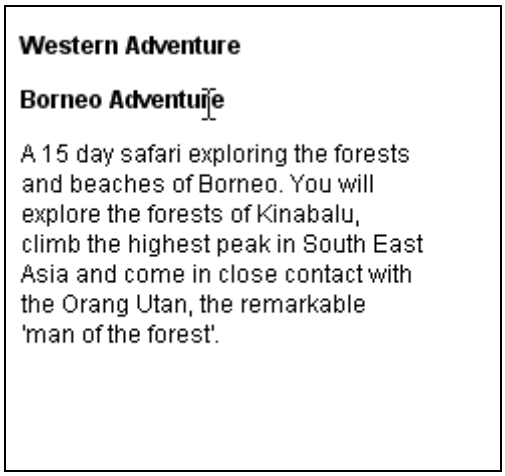
Task 2. Working with JavaScript events

1. View the page *events.html* in your browser. The page contains two headings and two DIV elements with text, one of which is initially hidden.



2. Open the page for editing in VS2010. You will see that there are two functions which expand and collapse (hide) a DIV respectively. The id of the DIV is passed to each function as a parameter.

3. Add **inline event handlers** so that when the mouse is moved over either title the DIV immediately below it is expanded and the other DIV is collapsed. Test in Firefox and Internet Explorer.



4. Modify your page to use the **traditional model for registering event handlers**. You should add a new function *registerHandlers()* and add a new button to the page to call *registerHandlers()*. (You can use the inline model to set up the handler for this button).

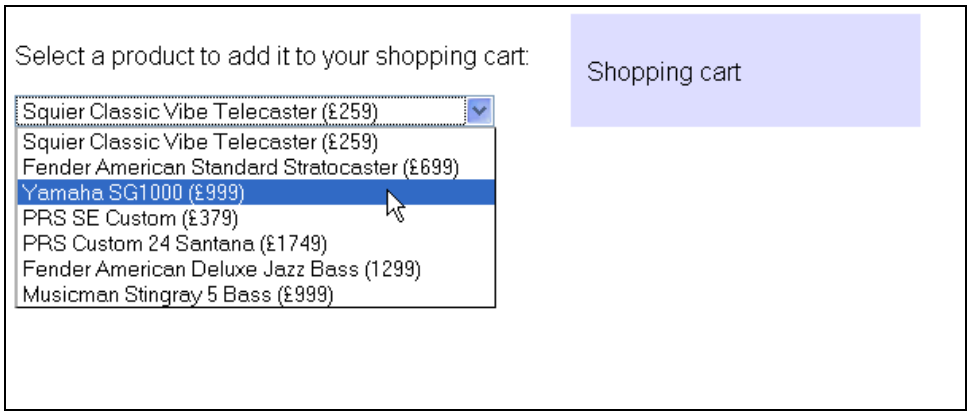
5. View the page in your browser and test its behaviour. There should be no response initially. After the button has been clicked the page should behave exactly like the inline version in step 3 above. Test in Firefox and Internet Explorer.

6. Modify your page to use the **W3C model for registering event handlers**. This will require modification to *registerHandlers().*View the page in your browser and test its behaviour. The page should behave exactly like the traditional version in step 4 above. Test in Firefox and Internet Explorer. Does it work in both?

7. Add a function to remove the event handlers and a button to call this function. View the page in your browser and test that the responses to *mouseover* events can be turned on and off with the appropriate buttons.

Task 3. A simple JavaScript shopping cart

1. View the page *shopping.html* in your browser. The page contains a SELECT element with a list of products which can be added to the shopping, and a DIV element on the right which is intended to display the current cart contents and total price.



2. Open the page for editing in VS2010. Note that the OPTION elements within the select are defined so that the product name is stored as the text property of each OPTION and the price is stored as the value property. There is an incomplete function *addToCart()* in the code.

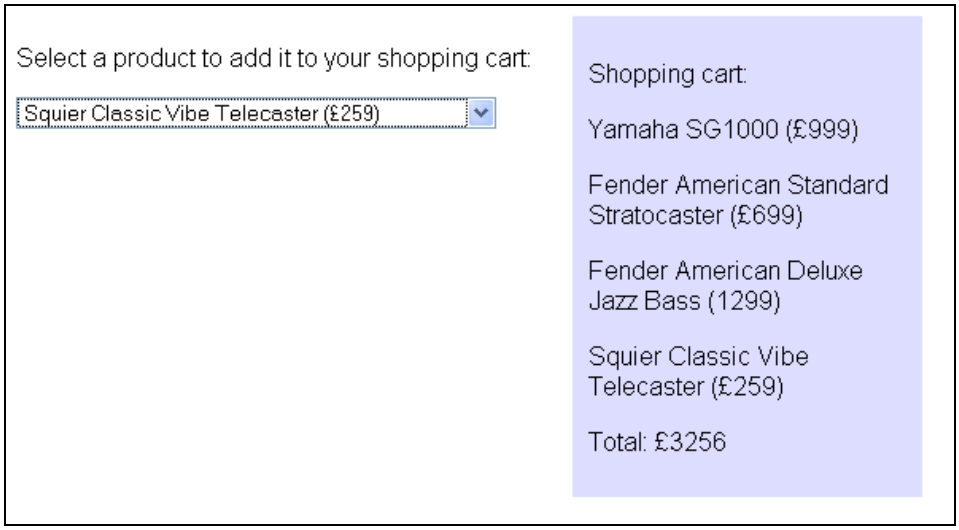
3. Complete the page so that selecting an item causes it to be added to the shopping cart, and the current contents and total price of the items in the cart are displayed. Your code will need to:

   - Get the index of the selected item in the SELECT's array of OPTIONs.
   - Get the *text* and *value* properties of the selected item and use these to create a new *ProductInfo* object. The constructor for *ProductInfo* is provided for you – you create a new object like this:

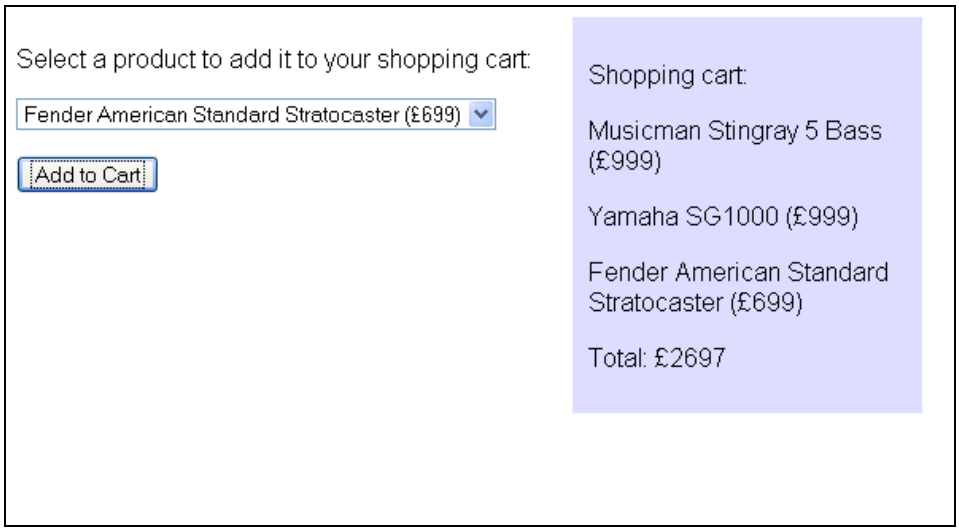
```
var product = new ProductInfo(productname, productprice);
```

   - Add the new *ProductInfo* object to the array *products* which is already declared for you as a global variable.
   - Set up a string variable to contain the output for the shopping cart
   - Add the name of each item in the products array to the output
   - Add up the prices of each item in the products array to get the total price and add this to the output.
   - Set the *innerHTML* property of the shopping cart DIV to display the output.
   - Add an event handler for the onchange event of the SELECT

4. Test your page in your browser. Select items and check that the information in the shopping cart display is updated correctly.



5. Finally, modify your page so that items are added using an **Add to Cart** button rather than when the item is selected.



Note: Debugging

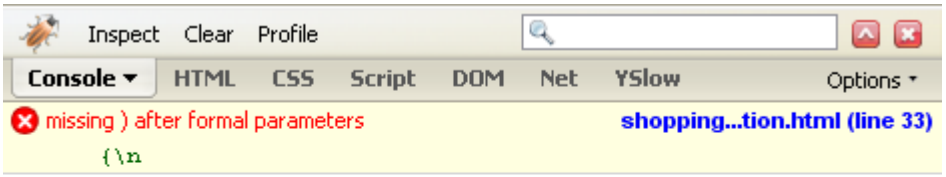
It can be difficult to work out what is wrong when a page with JavaScript does not behave the way you want it to. One reason for this is that you develop in one environment (VS2010 or any other editor) and run the code in another (the browser).

Often if there is an error then the only thing you can observe is that nothing happens, with no obvious error message being displayed.

The Firebug extension to Firefox can be helpful in the following ways:

JavaScript syntax errors:

Errors may be flagged up in the browser status bar and reported in more detail in the Firebug console. You can then click on **filename(line number)** to view the relevant code. Note that you have to go **back to the editor** to change the code, then reload the page in the browser to test again. VS2010 also may highlight syntax errors while you are editing.



Runtime errors:

Problems which occur when syntactically correct JavaScript is executing can be harder to track down. For example, if you use *document.getElementById* and the id you specify does not exist on the page, then the JavaScript execution simply stops at that point with no error message. The Firebug Script tag lets you set breakpoints, step through code and inspect variable values to help you find where in the code the problem is occurring and whether the variables are being assigned values correctly.