|  |  |
| --- | --- |
|  | **CE212 Lab 2: JavaScript**  **Introduction**  The purpose of this lab is to gain greater familiarity with some of the language features of JavaScript, and how it may be used to interact with a web browser.  Before working through this lab you should have studied the solution to last week's lab exercise.  JavaScript is not just for making great dynamic web pages; it's also a very powerful and concise programming language.  The first part of the lab works through some examples.  This involves either copying a file or typing in some code and then running it by loading an HTML page into a web browser.  In some cases, the code has been deliberately broken, and needs fixing; fixing it will help you learn.  The lab finishes with an exercise; you should aim to complete it before the start of the lab next week.  It is recommended that you use IntelliJ for editing the HTML, JavaScript and CSS involved with these examples, and the Firefox or Chrome browser for testing them.  **JavaScript from a Hyperlink**  This example demonstrates how to change the appearance of part of a document by invoking JavaScript code from a hyperlink.  The file can be found here: [**Lists.html**](https://moodle.essex.ac.uk/pluginfile.php/857855/mod_resource/content/1/lab2/Lists.html).  An excerpt is shown below.  Note that the excerpt is missing the CSS classes which you need, but these are in the Lists.html file.  Note how **<a href="javascript: ...">Link Text</a>** is used to invoke the code from the link.  The **listStyle()** function is very simple: it gets  **div** tag (identified by its **id** attribute) and then sets the value of its **class** attribute to the be the value passed as an argument called**style**.  The list comes up in the style shown below, and clicking **Default** changes it back to the default style.  **Mini-Exercise:** The 'No-Bullet' style is broken however; see if you can fix it.  *Hint: study the CSS classes.*  **Evaluating JavaScript Code**  In this section we'll learn how to build a JavaScript test page.  This will have a text area in which to type JavaScript code, a button to run the code, and a button to clear the code.  This enables you to very rapidly try out simple program fragments; this is useful when learning a new language.  A screenshot is shown below:  The code to do this is remarkably straightforward - see the snippet below.  Note again the use of **document.getElementById()**, but in addition the property **textContent**.  This is a property of the JavaScript objects that model HTML elements (in other words, the DOM).  Getting or setting a node's **textContent** does what you expect.  On the other hand, for the text area, which is an input control, observe the use of **ipNode.value** to get the text.  A try/catch block is used to allow useful reporting of any errors that the system detects (any error messages will be copied to the output area).  The eval function expects as an argument a string containing Javascript code; it interprets the Javascript and returns the value of the last expression in the code fragment.  The code for this page is available [**here**](https://moodle.essex.ac.uk/pluginfile.php/857855/mod_resource/content/1/lab2/TestJS.html).  Verify the operation of this page by typing some expressions to be evaluated.  Try some examples such as 13<3, "13"<3, 13<"3" and "13"<"3" - try to explain the results!  **Popup Windows**  The next example we shall study is how to pop up a window from an existing browser window, and then to communicate with the opener.  The effect achieved should be this: an initial window with a hyperlink, which can be clicked to open a new window:  Clicking **Yes** or **No** then sends the value back to the calling window i.e. the opener:  This invokes the **reply** function of the opener, which just pops up an alert.  This example also illustrates the use of invisible blocks of HTML; note how the code copied from the div tag is displayed in the popup window, but was invisible in the displayed window.  This code contains many important ideas, including:   * use of **window.open()** to open a new window, which is assigned to a variable **w** * use of **w.document** to get a reference to the document object of the new window * **d.open()** to open the document for writing * **d.close()** to finish writing and display the new window * use of an invisible style to stop content from being displayed * the innerHTML property of an HTML element, which gets all the children (including text content) of that node in the document tree.   Type it in and get it working for yourself.  Above is the head, below is the body (you'll need to add the closing **</body>** tag).  There's a deliberate error in the screenshot - try to fix it. (Hint: look for differences between the atrributes of the two a tags.) Experiment with the HTML; what happens if you remove the **class="invis"** declaration?  Does that block remain invisible or not? If not, why not?  **Exercise: a date selection control**  The aim of this exercise is to produce a date selection control, that allows the selection of a day from a month.  The month is specified by two parameters: the day of the week on which the first day falls, and the number of days in the month.  The popup control should be invoked with a call to a user-defined function called **getDay(startDay, nDays)**.  The control should only show this information; no year or name of month should be shown (this is partly to keep the problem simple, and partly to prevent the direct reuse of an existing calendar control!). When the user has selected a date from the popup the chosen date should be displayed on the parent window.  A suitable solution is shown below (this was cropped from a screenshot of the date control used by Expedia).  **Some hints:**  You may wish to store the initial letters days of the week in an array - the syntax for the declaration of such an array is  var days = ["S", "M", "T", ...];  The easiest way to generate the calendar display is to use a table. Start by creating a popup window as in the earlier example. Use a **div** tag with an **id** attribute in the call to **d.write** and then after calling **d.close()**, use **d.getElementById** to retrieve the **div** element - add a table to it in a similar way to the temperature conversion exercise from lab 1.  To test everything so far you'll need to create an HTML page similar to the one seen earlier - for the moment simply make a call to **getDay** with fixed arguments, e.g. **getDay(4,31)**.To be able to detect a mouse click on a table element it will be necessary to add a mouse event to each table data element. This is done by adding an HTML **onclick** attribute to the element. The following shows how to do this to generate an alert  td.setAttribute("onclick", "alert('"+i+" selected');");  [ The value of the attribute needs to be of the form **alert('*7* selected');** with the number inside the string being the current value of the loop counter being used to generate the table data (JavaScript strings can use single quotes, this can be useful when we're generating a string within a string). ]  Try this and check that it works.  Instead of displaying the selected value on an alert you now need to display it on the parent window.  One way to do this is to create (in the HTML file) a paragraph tag with an **id** attribute containing some initial text (e.g. none) then write JavaScript to retrieve the paragraph element using its id, access its child text node and change the contents of that node using something like  var p = window.opener.document.getElementById("myPara");  var n = p.childNodes[0];  n.data = i;  We now need to allow the user to select the arguments to **getDay**. This can be done by creating a form with two input fields and a submit button. The action associated with the form should be something like  action = "javascript:getDay(parseInt(document.getElementById('if1').value),  parseInt(document.getElementById('if2').value));"  Having got this working there are several improvements that could be made. It's probably preferable to use an "OK" button on the popup window instead of making the selection as soon as the user clicks on a table data item. Hence when the data item is clicked we should store the current selection in an attribute of the table and let the button click action retrieve the value of that attribute and use it to update the display on the parent window.  end of page |