Web Technologies – Exercises: Week 6

This week's exercise uses JavaScript, arrays, DOM1 properties, methods and events.

Outcomes:

By the end of these exercises you should be able to:

- Use an array to store and manipulate a list.
- Use JavaScript to respond to events in a web page.
- Make a web page *dynamic*.

Task 1: Do the quiz.

Task 2: Display information from click events:

Develop your understanding of how to use events and event handlers as well as visualising the default size of elements in the browser. A picture showing an *example* of the layout can be found on StudySpace.

1) Create a valid XHTML page in a file called week6onclick.htm entitled "Week 6: global onclick" whose **<body>** section contains:
   a) A 1st level heading element **<h1>**.
   b) A paragraph **<p>** with some text in it and one link to any other page.
   c) A 2nd level heading **<h2>** containing some text.
   d) An unordered list **<ul>** with three list items **<li>**, each containing some text.
   e) A 3x3 table (with 3 rows and 3 columns in each) containing some random text (feel-free to copy/paste something appropriate – the content isn't important!)
   f) A final 3rd level heading with some more text.
   g) A right-aligned **<address>** block containing your name, k-number and course title (use CSS to right-align the text.)

   NB: Use your own text in the page, don't copy mine!
2) Define an `onclick` event handler on the **<body>** element that passes the `event` object to a function, *e.g.*

```
<body onclick="return clickHandler(event)">
```

   - **NB#1:** Because the handler is attached to the **<body>** *via* its `onclick` attribute and the **body** element is at the top of the node tree, it responds to clicks from *anywhere* in the page.
   - **NB#2:** To cancel default actions, the event handler must return false...
3) The function should be defined in a **<script>** block in the **<head>** that:

a) Takes `e` as its argument – in W3C DOM compliant browsers, `e` will be the `event` object passed by the **`onclick`** event handler from step 2) e.g.

```
function clickHandler(e) { … }
```

b) Displays the **name** of the *node* on which the click occurred in an `alert` box.

- Every node has a `nodeName` property.
- Every browser event generates an event object (called `e` in the function) which has a `target` or `srcElement` property which is a reference to the *node*.
- Make your code compatible with Internet Explorer *and* W3C-compatible browsers by using an `if` test to decide whether to use `target` or `srcElement` depending on which exists as a property of the `event` object.

c) Cancels the default action.

4) Add some CSS that draws a border around every element on the page (1 rule?!)
   a) Make the border a different colour (not green!) around the list items only.
      - Different browsers do padding/margin differently on list items so don't be surprised if your borders don't match my screenshot as that was done in Firefox.
   b) Personalise your page by changing at least the body font, paragraph text colour, list bullet plus anything else you fancy.
5) Click around the page to make sure it does something on every element ...
6) Make sure it's valid XHTML & CSS.
7) Finally, modify the event handler function to display the contents of the node (*e.g.* the text or the HTML inside the tags) on a separate line after the node name in the `alert` from step 3. (If it works try clicking on an empty bit of the document <grin>)
   - **Hint this is lecture 7 stuff**: You could display the contents of the `innerHTML` property of a node, or use the fact that the text inside an element is in *child* text node(s) beneath that node in the node tree (see the DOM slides or do the reading or leave it until **week 7**…)

Task 3: Create a clickable sequence of images within a page:

The aim of this task is to create a page in which you can click on an image to have the image change to the next image in a pre-defined sequence with wrap-around (*e.g.* 1→2→3→1→2… *etc.*). The simplest way to do this is to store the list of images to

display in an array and modify the array when the image is clicked… (but you could ignore the instructions below and create your own solution if you like!)

1) Pick a sequence of at least three images that you like, that are all approximately the same size and related in some way
   - Find your own images, *e.g.* visit flickr and search for a sequence of photos…
2) Create a valid XHTML page in a file called week6gallery.htm entitled "Week 6: image gallery" whose **<body>** section contains:
   a) A 1st level heading element **<h1>** describing the subject.
   b) An **<img>** element centred in the page using CSS whose **src** attribute uses the 1st image in your sequence as a default.
      - Find out how to use CSS to make the cursor become the "pointer" as you hover over the image (nothing to do with JavaScript or **<a>** tags!) This tells the user that the image is "clickable" so they expect something to happen when they click…
   c) What's the licensing policy for the images you have chosen? Add a paragraph of text to the page giving the policy and a short discussion of whether or not using the images violates the policy.

There are many ways you can make a "marquee" work with JavaScript – the following is one approach but feel-free to follow your own path provided it (a) uses JavaScript and (b) you fully understand the code...

3) In the **<head>** add a **<script>** containing an array holding the file names or URLs for the images you have chosen.
   a) If you're using downloaded images, make sure the file name matches the relative path to the file from the web page.
   b) If you're using URLs (referencing images on a web site) make sure the URLs work!
   c) Make sure the *first* element of the array matches the *next* image in your sequence and the *last* element matches the default **<img src…>** from above.

- In the **<img>** element, add an **onclick** event handler like this:
  ****
  (using whatever name you like for the function but always passing "this" as the argument.)

- **NB:** "`this`" is a key concept in object-oriented programming – it usually refers to the current object, so in the event handler attribute it means "the object on which the event happened" which is the image element.
- (So why pass it as an argument? Because in the *function* "`this`" might not mean the image – it might [depending on the browser and DOM/JavaScript] mean the function or it might mean the image or it might mean the event object … so it's easiest to pass it explicitly by name!)

4) In the **`<script>`** create the event handling function.
   a) Test it like this:

```
function nextImage(imgObject) {
      alert(imgObject);
}
```

   so that when you click on the image, an **`alert`** should popup.
   NB: Firefox is the best browser here – IE just says "Object".
   b) Change `alert(imgObject)` to `alert(imgObject.src)` and see what happens…
5) Use an array to store the list of images:
   - *e.g.* `imgs = [image1, image2, image3, image4]`
6) The function needs to cycle through the images & the easiest way is probably:
   a) Store the current image number in a global variable `var ii=0;` at the start.
   b) Increment it at each click `ii++`
   c) Reset to zero when the end is reached `if (ii == imgs.length) ii=0;`
7) Write the new image (`imgs[ii]` or the "popped" element) into the **`<img> src`** element using something like `imgObject.src = imgs[ii];`
8) Test it – the alerts should show you what's going on and the images should change in sequence no matter how many times you click.

<u>That's an image marquee – well done!</u>

9) Make sure it's valid XHTML & CSS ☺

Alternative marquee code approach: You could use array `pop`, `push` and `sort` methods to sort the image array at each step, something like this:

- Start: `[image2, image3, image4, image1]`
- Reverse: `[image1, image3, image4, image2]`
- Pop: `[image1, image3, image4]` & `image2` is in a variable.
- Reverse: `[image3, image4, image1]`
- Push: `[image3, image4, image1, image2]`

so that the last element in the array is always the image we've just dealt with…

(1) *I.e.* you can mix `array.reverse()`, `array.pop()` and `array.push()` to take the next image in the sequence off the *front* of the array and push it back onto the *back* of the array.

(2) Try it! Modify the function to do `alert(array);` (insert whatever you called your array, *e.g.* `images`).

(3) Add to the function the `reverse`/`pop`/`reverse`/`push` sequence, where the `pop()` saves the "popped" element in a temporary variable which `push()` uses to push it back onto the array.

(4) Do another `alert` afterwards to verify that the sequence changed…

Don't forget to do some reading!

Dr James Denholm-Price