COMP 519 Web Programming

A Complex JavaScript Example

Here's a complicated JavaScript example, one that I'll consider working up from a static page to a fully working dynamic page (with some additional "conveniences" for the user). This is a JavaScript application to generate random permutations, where a permutation is an arrangement of the integers $\{1, \ldots, n\}$ for some chosen value of $1 \leq n \leq 40$. (I choose the upper bound of 40 as the browser starts to "hang" on bigger values.) You need not be overly concerned with the method used for generating the permutations, this can be thought of as the typical "black box" routine for which you supply an integer for input, and receive a string that contains the permutation. (I will explain the method a bit, but not the precise mathematical reasoning behind it.)

The examples given below are the files `permutation1.html` - `permutation10.html`, which can all be found in the directory

`http://www.csc.liv.ac.uk/∼martin/teaching/comp519/JS/`.

(So a complete hyperlink is

`http://www.csc.liv.ac.uk/∼martin/teaching/comp519/JS/permutation1.html`

for example.) Keep in mind that you can always look at the source code of the pages by right-clicking on them with the mouse and selecting the "View Source" option.

Take a look at `permutation1.html`. This defines the structure of the web-page. There's a text input box (designed, funnily enough for user input), a textarea which will eventually be used for the output (but the user can also edit this area), and a button which, for now, does absolutely nothing. So at the moment, this page is essentially static. Yes, a client can type in text in the input box and textarea, but there's no way of using this information.

The only difference between the first file and `permutation2.html` is that the button does something when it's clicked on (but something that's not very exciting). This is just a reminder how you can capture such an event as a button being clicked on the page (by setting an event handler to call a function when the button is clicked). The point is that we want to write a function that will use the input to eventually find a permutation and insert the result into the textarea (output) element.

`permutation3.html` demonstrates how you can get the value from the input area, namely by using the JavaScript function document.getElementById. As the name suggests, this returns the DOM element that has the supplied "id" associated with it. (Hence this is why it's useful to assign (unique) ids to input boxes, text areas, and so forth. There are other ways to find these elements, but this is one of the easiest and most direct ways to do it.) All this file does is find the input box area, and retrieve the value that's in it. Recall that it's stored as a string. This string is then output in the alert box. So even if you enter a couple of characters, there's no error (yet). We'll see how to check if it's an integer in a moment.

The next file (`permutation4.html`) uses the function verifyIntInRange, found in the external file `verify.js` in the same directory. You can view that file just by typing its name into the browser navigation bar. As you can see from the function call, this takes three inputs, a variable that is a text box (or textarea), and a lower and upper bound. Looking at the source code, you can see that this is first parsed as a floating point number (using the parseFloat command, and then we check if it's an integer (with the second part of the "if" conditional test) and is in the specified range. If not, an error message is displayed and the text box is cleared.

The next file (`permutation5.html`) includes the Permutation function to generate the permutation, given an integer. The idea behind this function is that you start with the list of numbers $1, 2, \ldots, n$ (in that order) and, for a large number of times, pick a position and swap the two numbers in adjacent positions. For example, you might start with $1, 2, 3, 4, 5$ and perform the following (pseudo)random choices that will swap elements:

1 2 3 4 5	(starting configuration)
1 3 2 4 5	swap numbers at positions 2 and 3
1 3 4 2 5	swap numbers at positions 3 and 4
3 1 4 2 5	swap positions 1 and 2
3 1 2 4 5	swap positions 3 and 4
3 1 2 5 4	swap positions 4 and 5
3 1 5 2 4	positions 3 and 4
3 1 5 4 2	positions 4 and 5
$\vdots$	$\vdots$

Hopefully, you might be convinced that doing this transposition operation a large number of times will lead to a (nearly) random permutation. Showing this is the case is a mathematical task, as well as figuring out what exactly is meant by the word "large". (But I won't talk about that here.) Simply use this function as shown, namely pass it an integer, and you will receive back a string that contains the permutation. In this case the permutation is simply inserted into the alert box output. Note that checking that the input value is an integer assures that the argument passed to the Permutation function is a valid argument (so that function need not do any checking itself).

The next page (`permutation6.html`) changes from the previous one in the display of the output. How do we do this? It's similar to retrieving the input value. We use the `getElementById` function to locate the textarea element, and then assign its `value` property using the string returned from `Permutation`. Voíla! Dynamic insertion of text into an existing element of a webpage!

Okay, so now you have a fully working page that "does exactly what it says on the tin". But I'm lazy (as those who know me well can attest to). And the page doesn't fully behave how I'd *like it* to behave. Namely, I'd like to insert a number in the input area, press the "Enter" key, *and* have it give me a permutation back. I don't want to have to reach for the mouse and click the button if I don't want to do so. How can I do this? Well, you can check for keyboard events, and if the user presses the "Enter" key you can proceed as before (get the input value, check that it's an integer, and, if so, generate a permutation, and output it to the screen).

The file `permutation7.html` implements this functionality. To do this, we assign another event handler to look for keyboard events. Note that this event handler needs to be assigned to the text input element, as that's what will trigger the (important and relevant) keyboard events that we want to capture. The user-defined function checkForEnter does the job. This takes two inputs, the relevant element in which the events occur, and the event itself. The "`this`" keyword refers to the "input" element itself in this case (the JavaScript way of performing a self-reference to an object while "inside" that object). The variable "`evt`" will contain a JavaScript event object variable that is set when an event occurs (any sort of event, like moving or clicking the mouse, or hitting a key). We can capture keyboard events using any of three event handlers, namely keydown, keyup, or keypress. I chose to use the keyup event handler (partly because Internet Explorer handles the keypress events slightly differently from other browsers and I want this to work in all browsers, or so I hope).

When an keyup event is triggered, a property called `keyCode` is set in the event that indicates which key was pressed (stored as a number in the ASCII standard). The relevant key we want, the "Enter" key, has value 13. So what we do is relatively straightforward, that is we check the `keyCode` property (of the event object) to see if it's the "Enter" key and return `true` or `false` as appropriate. Then, if `true`, we start up the sequence of events as before (exactly as if the button had been clicked). Again, see `permutation7.html` for this version.

So I'm happier now since my laziness is (somewhat) satisfied. But what happens if I do choose to use the mouse to click the button? If I then want to insert a different value, I have to either use the mouse to select the input box or use the "Tab" key to move all the way around the webpage to get to the input box again. My laziness again rebels against this. How can I alter this behavior of the page? Here's where I use a function to change the "focus". What is this? For such a page like this where there's input boxes and buttons (and similar pages containing checkboxes, radio buttons, and the like), one element is said to have the "focus". For example, when you're inputting text in the input box, it's that box that has the focus (or is "in focus"). When you click the button with the mouse, that button receives the focus (and whatever other element that had the focus is "blurred"). Clicking the button transfers the focus to the button, and after the output is displayed, the button still has the focus. So to get back to the input box, you have to put the focus back on that are by selecting it with the mouse, or using the "Tab" key to get there. I don't want to do this.

A fix for this problem is easy. In this case, after the button is clicked, the permutation generated and displayed, call the focus function of the text input box to transfer focus back to the input box, so then I can change the value there without having to first use the mouse (say) to get back to the input box. (See `permutation8.html` and compare that to `permutation7.html`, and what happens when you click the button, to see the difference.) Hooray! My laziness is once again (more) satisfied. This behavior isn't necessarily ideal, as in some browsers the focus is transferred to the input text box, but the cursor appears at the beginning of the input field instead of at the end. (We can work to get the cursor at the end of the input field, but this is surprisingly complicated to do so in a cross-compatible fashion, i.e. so that it will work in all (or most) browsers. So I have left the behavior as it is.)

Well, it's almost satisfied. What happens when you *first* load the page? What has the focus? It's not the input box. :(I want it to be the input box!! Let's use the load event handler that can appear as a property of the `body` element of the page. I'll call the focus function to transfer focus to the text input box when the page loads. As you can see you use the `document.getElementById` function to locate the input area and then call the function to transfer focus there. Hooray (again)! Laziness prevails! When I load (or reload) the page, the input box has the focus and I can go to change the input immediately instead of having to use the mouse to get to the input box. (Note: This has the same problematic behavior as before, namely that the cursor can appear at the start of the text input, rather than at the end. Again, I haven't implemented the surprisingly complicated fix to get the cursor at the end of the input field.)

The final version (`permutation10.html`) doesn't use a ¡textarea¿ element on the page, but I have instead used the JavaScript methods for creating a new HTML element and inserting it into the page. To do this, I have replaced the ¡textarea¿ element with a simple ¡span¿ element called "Output". Even though there seems to be nothing in the element, in the Document Object Model, this ¡span¿ area has a child. So the idea is that when a new permutation is made, I create a new "text node" that consists of the string for the permutation (which I get back from the "Permutation" function), and replace the existing child of the "Output" ¡span¿ element. Just for illustrative purposes, I've made the background of that ¡span¿ element to be light blue in color. As I said, this uses some of the built-in JavaScript methods for creating new elements and manipulating the DOM.

Annoyingly (but not surprisingly), Internet Explorer and Firefox implement the DOM in different fashions. In particular, Firefox creates a child node of the "Output" ¡span¿ element when it first loads the page, but Internet Explorer does not. Hence, this is why the code for the "onload" event handler is written the way it is. For Firefox, we generate a permutation and replace the (already existing) child of the "Output" ¡span¿ element. In Internet Explorer (apparently) this ¡span¿ element has no children, so we instead append the newly created text node as a child to that element. Having written code and tested it on those two browsers, I have already tried it out on Google's Chrome browser and it appears to work correctly. You might also notice that the colored background box appears differently in Firefox and Internet Explorer. (Chrome shows it similarly to Firefox.) I believe this might stem from how Internet Explorer implemented the so-called W3C "box model" in its Cascading Style Sheet implementation differently than the other browsers. I haven't fully explored this issue, nor considered/found ways to correct that.

This is as far as I've taken this page. You might have other ideas. If you don't want to start with a default value in the text input box, you can simply change the `value` property of the `<input>` element to be an empty string (or some other value if you wanted it). As you can see, you can add as much, or as little, as you want to such a page. Part of what you want to consider is what will the user(s) want, what will be useful to them, or just extraneous.

P.S. In terms of JavaScript programming, sometimes "debugging" your programs might be more difficult than if you're programming in Java or C. If you use Firefox, then (depending upon the version of Firefox) there is an extension that can be installed called FireBug, that can prove useful in debugging JavaScript code. There's also a JavaScript error console that can give *some* error messages (or warnings). Select the "Error Console" under the "Tools" menu.

Internet Explorer and Chrome (and likely other browsers) have some tools for debugging JavaScript, but I am less familiar with them. You generally want to look in the menu options for phrases like "Developer tools" or something similar. It is worthwhile checking for such tools to help debug JavaScript code if you intend to write large and complex scripts.

My general advice is to proceed somewhat slowly with your coding, and test your programs incrementally, i.e. often so that you can detect and correct errors as they occur instead of having a large mess of code that might have multiple errors that can be hard to fix.