Section 5: JavaScript

JavaScript is a client-side programming language that you can embed in your XHTML pages. As a client-side language, scripts run on the browser, not the server.

Including JavaScript in your pages
You can include JavaScript code blocks in your pages by wrapping them in <script> tags and an XML CDATA section. (The CDATA syntax is necessary to prevent XML parsers from confusing characters in the JavaScript code as actual XHTML).

```
<script type="text/javascript">
// <![CDATA[
...
// ]]>
</script>
```

You can also put your JavaScript code in a separate file and reference it from your XHTML, similar in spirit to how you write your CSS in a separate file:

```
<script type="text/javascript" src="myexternalfile.js"></script>
```

Note that <script> blocks may go either in the <head> or <body> section of the document, depending on the nature of the code.

Syntax
JavaScript is very similar in syntax to C, C++, PHP, and Java. (However, JavaScript is *not* Java – the similarity in naming is marketing-based.)

Example for *loop*

```
var i;
for (i = 1; i <= 10; i++) {
	document.write("Hello, your number is " + i + "<br />");
}
```

This code block first defines variable i. Notice that in JavaScript, variables are defined using the var keyword. Also, unlike PHP but similar to C, variable names do not have a "$" sign prepended to them. The for loop shares the same syntax as PHP; in this example, the value of i iterates from 1 to 10.

The third line calls the "write" method of the "document" object. The "document "object is one of the predefined variables in browser-based JavaScript, and represents the current XHTML document. Although you'll see later what other methods and properties it has, know that the "write" method simply writes out arbitrary XHTML to the page. Also notice that in JavaScript, string concatenation is done with the "+" operator, and not the "." from PHP. Further note that JavaScript is, like PHP, weakly typed; although i may hold an integral value, you can concatenate it with a string to form a longer string.

Form validation

One very popular use of JavaScript is for client-side form validation. While you can't get rid of server-side validation (you must still double-check if the data you are receiving from the client is correctly formatted), client-side form validation improves the user's experience as they won't, say, need to submit the form and wait for a response just to find out that they missed out a field, for example.

Let's say we have a form like this:

```
<form method="post" action="dostuff.php" name="myform">
<input type="text" name="yourname" size="20" />
<input type="checkbox" name="student" /> Student
<input type="text" name="schoolname" size="20" />
<input type="submit" value="Do Stuff" />
</form>
```

We don't want the user to submit the form with the name empty; also, if the user has checked the "student" checkbox, we don't want the user to leave the schoolname text box empty either. While we'll be checking this server-side as well (as in previous lectures, sections, and projects), we can improve the user experience by immediately providing some feedback to the user if their form input does not meet our specifications. We can achieve this using Javascript.

First, let's write some Javascript code above the <form> tag:

```
<script type="text/javascript">
//<![CDATA[
function checksubmission() {
	// Did user leave yourname blank?
	if (document.myform.yourname.value == "") {
		// Show an error message
		alert("Please do type your name");
		// Exit out of the function, returning false
		return false;
	}

	// Did user check student and leave schoolname blank?
	if (document.myform.student.checked &&
	(document.myform.schoolname.value == "")) {
		// Show an error message
		alert("Please tell us your school name");
		return false;
	}

	// If all our checks pass, return true
	return true;
}
//]]>
</script>
```

The syntax should look familiar, now that we know PHP. We have defined a function, checksubmission(), that first checks to see if the "yourname" textbox in the form has been left blank. "document.myform" refers to the list of all objects inside the "myform" form; since the "yourname" textbox is contained inside myform, we can refer to it using "document.myform.yourname". The "yourname" textbox has a property named "value",

accessible by simply referring to "document.myform.yourname.value". If the value is blank (i.e., if the user hasn't entered anything), we display an alert message (using the built-in alert() function); then, we exit out of the checksubmission() function, returning "false".

The second portion of the code is similar in nature; we're just checking the status of the checkbox (using the "checked" Boolean property of the "student" checkbox) before checking the value of the "schoolname" textbox.

If you can't remember where to find the "document", "checked", and "value" properties, you can always refer to a JavaScript tree on the Internet[1].

Now we have a JavaScript function that can check our submission; at the moment, though, there's nothing that runs the function. Since we want to execute the function when the user clicks on the submit button, we can add checksubmission() to our form's "onsubmit" attribute. We do this by modifying the <form> tag like so:

```
<form method="post" action="dostuff.php" name="myform"
       onsubmit="return checksubmission();">
```

This makes the web browser run checksubmission() once the user clicks on the submit button, but before the form data is actually submitted. Since we used "return", if the return value of checksubmission() is true, then the data will be submitted as usual; if we return false, the form submission will be canceled.

Trimming, prototypes, and regular expressions

The validation example above was very simple; the validation failed only if the input boxes were empty. But as we saw with PHP, that's not enough to ensure that the user types in valid data. For example, what if the user types in " " (space) for their name? In PHP we would have used the trim() function to remove any spaces before and after the input, and checked that against the empty string to check if a user had simply typed in one or more spaces.

Unfortunately, JavaScript does not come with a built-in trim function; instead, we can use the power of regular expressions and JavaScript's notion of "prototypes" to define our own trim() function.

Our own trim() function

```
String.prototype.trim = function() {
   return this.replace(/^\s+|\s+$/g, "");
}
```

These three lines do several things at once. Let's look at them by parts:

- `this` – in this case this is a reference to the string that is being referred to. Notice that strings (of type String) are objects in JavaScript; therefore, they possess methods (some built-in methods include toLowerCase() and italics()[2]) that change the contents of their data. This is different from PHP, where strings are strictly data types.

[1] http://www.howtocreate.co.uk/tutorials/javascript/javascriptobject

[2] See http://www.w3schools.com/jsref/jsref_obj_string.asp for a list

- `replace(/^\s+|\s+$/g, "")` – The replace() method goes through its parent (in this case, "this") and replaces text that matches a regular expression with some other text. It takes two arguments – the regular expression, and the text to replace with. In this case our regular expression matches 1 or more spaces at the beginning of the string, *or* 1 or more spaces at the end of the string. The "g" is a regular expression modifier that makes the regex match *all* instances of text that match the expression; since we may have spaces both at the beginning and end of the string, we want it to match multiple times. The second argument is an empty string because we want to replace all spaces with nothing (i.e., we want to delete them).
- `function() { }` – defines an unnamed (or *lambda*) function that takes no arguments but when called executes the code within the curly braces.
- `String.prototype.trim = function() { }` – defines and attaches a new function called "trim()" to the *prototype* of the String object, with the contents of the lambda function as the code to be executed when String.trim() is called. What this means is that **all** String objects get a new function, "trim". Essentially we're modifying what a String is (i.e., its prototype) for this page.

Having defined this, we can simply call, for example, document.myform.myinputbox.value.trim() to get the trimmed version of the contents of "myinputbox" in "myform". This is because "value" is of type String, which now has our trim() function.

Using the regular expression object
Using regular expressions, we can achieve other, more complex validation techniques. We can, for example match all e-mail addresses:

```
var regex = /.+@.+/;
var myemailaddress = "example@keitr.com";
if (regex.test(myemailaddress)) {
	alert("Your address is valid!");
} else {
	alert("Your address isn't valid!");
}
```

The first line defines a new regular expression, "regex", with a regular expression that matches the form "anything@something". (In real life, this is probably not a good-enough regular expression). The third line then calls "test" method on the regular expression. That's right, "regex" is an object – of type regular expression – that, like any other object, has methods and properties.[3] The test() function returns true when the provided string matches the regular expression, false when it doesn't.

Other methods of the regular expression object you might find useful are "match" (to do regular expression matching, similar to using the "matches" argument of preg_match() in PHP), "replace" (although the String object already has a replace() method, as we saw above) and "split" (to split a String into an array by boundaries defined as a regular expression).

[3] See http://www.w3schools.com/jsref/jsref_obj_regexp.asp for a list

Other event handlers

What if we have a combo box (a dropdown list) that, upon selection, should automatically submit its enclosing form for us? This is common on websites where the combo box is being used for navigation purposes.

For this, we can use another commonly used event handler attribute, “onchange”.

Example using onchange

```
<form method="get" action="goto.php" name="myform">
<select name="category" onchange="this.form.submit();">
<option value="lectures">Lectures</option>
<option value="sections">Sections</option>
<option value="phorum">Phorum</option>
</select>
</form>
```

Here, the select box has an “onchange” attribute, which is set to “this.form.submit();”. “this” refers to the current object (i.e., the select box); the “form” object refers back to the form that the item is contained in; and “submit()” is a method of the form object that submits the form.

The “onchange” event fires when the selection in the select box changes; therefore, if we click on the box and select “Sections”, the form will be submitted, and we will be taken to goto.php.

Notice that this form has no “submit” button – the only way it can possibly be submitted is through JavaScript. While the merits of this design approach are debatable (users without JavaScript will be unable to use this feature, including those using special web browsers for accessibility reasons), it is a common technique often seen around the web.

Another set of event handlers you may find useful are onmouseover, onmouseout, onmousedown, and onmouseup. As the names suggest, these events are fired when the mouse is over the object, the mouse is no longer over the object, the mouse button is down on the object, and when the mouse button has been released over the object, respectively.

Without a doubt, you’ve seen images on the web that change to another image when you put your mouse cursor over them (image rollovers). Many of these are implemented using JavaScript’s onmouseover and onmouseout event handlers.

A simple example of a rollover effect

```
<script type="text/javascript">
function changetosecondimage(myImage) {
	myImage.src = "secondimage.png";
}
function changeback(myImage) {
	myImage.src = "firstimage.png";
}
</script>
```

```
<img src="firstimage.png"
       onmouseover="changetosecondimage(this)"
       onmouseout="changeback(this)"
/>
```

When the mouse cursor rolls over the image, the onmouseover event fires, which calls our changetosecondimage function with "this" (in this case, referring to the image object) as its sole argument. changetosecondimage then sets the "src" attribute of the image object to refer to the second image. The same thing happens when the mouse cursor rolls out of the image; changeback is called, and the "src" attribute is changed back.

If you test this code out on a remote server, you'll notice that this implementation has a slight problem; the first time you mouse over the image, there is a delay while the second image (secondimage.png) is fetched from the server. This is because the second image is not in the browser's cache (unless it is used elsewhere in the page), so it must be downloaded when it is required.

A simple workaround would be to ensure that the image was already in the browser's cache by using it explicitly somewhere else in the page (i.e., simply insert somewhere in the page). However, this is inelegant, especially when JavaScript provides a better way – you can create an internal representation of an image object that won't be displayed. Even though the internal representation of the image is not displayed, it is still downloaded and cached by the browser, so subsequent attempts by the changetosecondimage() function to use it will not incur a fetching delay.

This we can do by adding the following code in a place where it will be executed when the page loads:

```
var tempImage = Image();
tempImage.src = "secondimage.png";
```

This code simply instantiates a new Image object (a JavaScript object representing an image) called tempImage, and sets its source to the second image. This has the effect of forcing the browser to download the image from the server, and maintaining a copy of it in its cache. We don't actually do anything with the tempImage object; its sole purpose in life is to download the image.

One good place to put this code is in a function that is then invoked by the "onload" event handler of the <body> tag. The "onload" event is a special event that gets fired when the page finishes loading in the browser. JavaScript initialization code is often placed there to ensure that the initialization code runs only when all the elements of the page have been constructed.

Using the Yahoo! UI (YUI) library

Given all the standardized documentation Yahoo! has provided with the YUI library, it's very easy to use all the controls and utilities that Yahoo! provides on its website. Let's take the Calendar control as an example.

Go to the page for the Calendar control (http://developer.yahoo.com/yui/calendar/); you'll find a list of links, including one to a page of Examples (this is incredibly useful), and API Documentation (useful when you want to try something not explicitly used in the examples). Under "Getting Started" on the same page, you'll see a code box with some code you're supposed to paste into the <head> sections of any pages where you use the Calendar control:

```
<!--CSS file (default YUI Sam Skin) -->
<link rel="stylesheet" type="text/css"
href="http://yui.yahooapis.com/2.5.0/build/calendar/assets/skins/sam/calendar.css" />

<!-- Dependencies -->
<script type="text/javascript" src="http://yui.yahooapis.com/2.5.0/build/yahoo-dom-
event/yahoo-dom-event.js"></script>

<!-- Source file -->
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.0/build/calendar/calendar-min.js"></script>
```

(Notice that Yahoo!'s sample code is HTML, not XHTML; you'll need to XHTML-ify their samples, for example by closing the "link" tag properly.)

If you follow their example, you'll be told to paste the following code into your page where you want the calendar:

```
<div id="cal1Container"></div>
```

This acts as a placeholder for the calendar. Although it's empty at the moment, the Yahoo! UI library code populates it with the days of the month, buttons, etc. using JavaScript.

Once we have this code, we need to actually call the YUI Calendar library to generate the calendar for us. We can do this with the following code, again provided by Yahoo!:

```
<script type="text/javascript">
//<![CDATA[
var cal1 = new YAHOO.widget.Calendar("cal1Container");
cal1.render();
//]]>
</script>
```

The first line of the JavaScript code initializes a new object of type "YAHOO.widget.Calendar", passing to the constructor (a function that constructs an object) the id of the calendar placeholder. A new variable, "cal1", is then assigned a reference to this new Calendar object. The next line simply calls the "render()" method on the Calendar object, which we presume is a YUI method to render a calendar.

The last thing we need to do is to give our <body> tag a class of "yui-skin-sam". This is a requirement of the visual YUI libraries, so that the Yahoo! CSS files work properly.

Voilà – those are all the steps you need to take to place a fully functioning calendar on your page. Or so it would seem; after all, at the moment, your calendar can do nothing useful beyond telling you which days of the year fall on which days of the week. Let's try something useful.

Let's say we want to tell the user which year, month, and day they clicked on when they click on a day in the calendar. Following from the event handling examples above, we need to somehow "subscribe" to the event handler provided by the calendar that fires when a day is clicked on.

We can subscribe to the event with the following code (placed just before the call to render()):

```
cal1.selectEvent.subscribe(mySelectHandler, cal1, true);
```

This causes the Calendar (or more precisely, the YUI Event utility) to associate the "selectEvent" of the Calendar with a new function we'll create in just a moment, mySelectHandler. The arguments of the subscribe() function are described in detail in the API Documentation for the Event utility[4].

We must define mySelectHandler before we define it as the handler for a particular event; therefore, let's go back to the top of our <script> block, and define the following code:

```
var mySelectHandler = function(type, args, obj) {
       var selected = args[0][0];
       alert("Year:" + selected[0] + " Month:" + selected[1] + " Day:" + selected[2]);
};
```

This code defines mySelectHandler. Notice that this, again, is a lambda function (a function without a name). YUI tells us that it must takes three arguments; for now, we're just concerned with the second, args, which stores the arguments that our handler was called with. In this case, args is an array whose first element is an array of arrays (in the form Year, Month, Day) that tells us which dates were selected[5]. Therefore, we get to the date we want (the first date in the first argument) by referring to "args[0][0]". By setting "selected" equal to this, we can use "selected" to mean "args[0][0]". We can then proceed to get the year, month, and day by referring to selected[0], selected[1], and selected[2], respectively.

With all this code put together, you should be able to click on days in your calendar to get a prompt telling you which day you selected.

Naturally, as you do more and more things with the YUI controls and utilities, you'll want to refer to the API Documentation[6] to find the correct methods, properties, and event handler arguments. Also of potential help is the Yahoo! User Interface Library Group, ydn-javascript[7]. And of course, the course forum is always there to help you!

[4] See http://developer.yahoo.com/yui/docs/YAHOO.util.CustomEvent.html#subscribe

[5] See http://developer.yahoo.com/yui/docs/YAHOO.widget.Calendar.html#selectEvent

[6] See http://developer.yahoo.com/yui/docs/index.html

[7] See http://tech.groups.yahoo.com/group/ydn-javascript/