Computer Science 75
Fall 2009
Scribe Notes

Lecture 7: November 2, 2009
Andrew Sellergren

Contents

1 Announcements (6:00–8:00) 2

2 JavaScript (2:00–105:00) 2
2.1 Introduction . . . 2
2.2 Embedding and Linking to JavaScript . . . 3
2.3 Client-side Form Validation . . . 5
2.4 Syntax . . . 13
2.4.1 Arrays . . . 13
2.4.2 Form Focus . . . 13
2.4.3 Objects . . . 14
2.4.4 Events . . . 15
2.4.5 Manipulating Style . . . 15

Computer Science 75
Fall 2009
Scribe Notes

Lecture 7: November 2, 2009
Andrew Sellergren

1 Announcements (6:00–8:00)

- To help with the learning process for JavaScript as well as for your final projects, feel free to attend one or more seminars held by CS 50's TFs.

2 JavaScript (2:00–105:00)

2.1 Introduction

- In the past few weeks, we've worked with PHP, a server-side, interpreted language that allowed us to generate dynamic XHTML content. We began by using XML to implement a database and then transitioned to MySQL. Now we'll be introducing JavaScript, a client-side, interpreted language. What it means to be "client-side" is that the code is actually executed by the user's browser (the client) rather than the server. What it means to be "interpreted" is that the code is not compiled before being run (like C or C++ or Java), but rather is passed at runtime to a parser (the interpreter), which translates it into the 0's and 1's that the computer actually understands.

- Interpreted languages are generally faster in development but slower in execution. If you make one small change to your code, you don't need to recompile the entire thing in order to deploy it. That being said, compiled code is already written so that the machine can understand it—it doesn't have to be parsed and translated by an interpreter like PHP and JavaScript do. To mitigate the performance costs of interpreted languages, it's possible to use accelerators which cache the results of code being interpreted. This is a bit like storing compiled versions of PHP scripts.

- These days, the speed of JavaScript engines is even a selling point for browsers. Whereas ten years ago, JavaScript wasn't being used for much more than annoying popups, nowadays, JavaScript is being used for interesting UI components, like autocomplete, search filtering, and even simple animations.

- A lot of the fun of JavaScript is the wealth of APIs out there. As you learn how to interact with Google Maps via their API, you'll also be learning some of the basics of JavaScript. The Google Maps API Reference is incredibly thorough (albeit poorly organized).

- Unlike PHP, which has extensive, easily accessible documentation replete with examples, JavaScript has no such centralized resource. However, you'll find a few useful tutorials and references on the Resources page.

- A sidenote: JavaScript has no relation to Java. As the story goes, the name was chosen to piggyback on the success that Java was experiencing at the time.

2.2 Embedding and Linking to JavaScript

- In order to be interpreted client-side, JavaScript is sent to the browser along with XHTML content and images. JavaScript can be embedded directly in XHTML like so:

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

// ]]>
</script>
```

  This `script` tag can be placed either in the `head` or the `body` of a document. The two sets of forward slashes denote comments in JavaScript but what about the `CDATA`? This denotes data that shouldn't be parsed as XML. What we're saying is that the JavaScript code shouldn't be parsed as XML so that characters like less-than or greater-than operators won't be interpreted as misplaced angle brackets which would prevent our XHTML from validating. The forward slashes ensure that the `CDATA` tag isn't interpreted as JavaScript. Although this isn't strictly necessary to ensure that our JavaScript will execute and our page will render properly, it is necessary in the interest of adhering to web standards, which will certainly help with cross-browser compatibility.

- Sidenote: ampersands in URLs can be changed to their HTML entity `&` so that your XHTML will still validate.

- The other method for using JavaScript on a webpage is to include an external file like so:

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

  Note that although the `script` tag is empty in this case, many browsers will choke if you try to self-close the open tag, so it's better to explicitly write out a close tag.

- Including JavaScript code via a separate file is good practice because multiple pages can use the same code without resorting to copy-paste. Moreover, you only have to change your code in one place to update it.

- Interestingly, including JavaScript via a separate file still doesn't protect against your code being stolen. Because the browser has to execute the code, the source must be fully available to it. It is possible, however, to *obfuscate* or *compress* your JavaScript code (by removing whitespace and abbreviating variable names), which makes it much harder to read and decipher. Moreover, compressed code is a lot cheaper to transfer over the wire because it eliminates unnecessary bytes. Ultimately, obfuscating and compressing JavaScript only raises the bar: an astute programmer will

still be able to decipher it. However, if he's astute enough to decipher this garbled code, chances are he's astute enough to write it himself, so he won't bother stealing yours.

- If we navigate to Facebook, we can see that they are including several external JavaScript files which are compressed and obfuscated. If we use the Net tab of Firebug, we can see that loading the homepage takes about 700 milliseconds. Typically, users can tolerate 2 seconds or less for page loads. And herein lies another advantage of including external JavaScript files rather than embedding directly in the XHTML: browsers will cache these external files so that they only need to be loaded from the server once. Of course, this also means that buggy versions of code might persist longer than you'd like. Some sites will attempt to mitigate this by sending out no-cache headers or, as Facebook seems to be doing, creating unique filenames for its JavaScript includes so that the browser will assume it's a new file that needs to be downloaded.[1] Take a look at example 2 on PHP's manual page for the `header()` function for examples of how to use HTTP headers to control caching.

- Let's take a look at a very simple example of JavaScript:

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

    document.write("hello, world");

// ]]>
</script>

<noscript>
    goodbye, world
</noscript>
```

  `write()` is a method which comes built into the `document` object. This code has the same effect as simply inserting "hello, world" in our XHTML, but it also demonstrates the use of the `noscript` tag, which executes when a user has JavaScript disabled in his browser. It's actually a fun exercise, in fact, to disable JavaScript in your browser and to see which of your favorite websites actually break as a result. You'll be surprised at how many actually depend on it!

- When you're developing a site that uses JavaScript, you'll have to ask yourself if it's worth the extra time and effort to develop a separate version that doesn't require JavaScript. You may find that it's not worth it given

[1] Facebook used to accomplish this by appending a timestamp after a question mark in the JS filename so that the browser would see a new URL that needed to be downloaded.

that you'll be spending 50% or more of your time to accommodate 1-3% of users.[2]

- Another concern is that JavaScript interferes with many screen readers which improve access to sites for the visually impaired. One of the reasons we preach the use of YUI is that they've been more sensitive than other libraries in accommodating these users by adhering to accessibility guidelines.
- JavaScript supports a number of different statements, many of which will be familiar to you from PHP:
  - `break`
  - `const`
  - `continue`
  - `do ... while`
  - `for`
  - `for ... in`
  - `for each ... in`
  - `function`
  - `if ... else`
  - `return`
  - `switch`
  - `throw`
  - `try ... catch`
  - `var`
  - `while`
  - `with`

2.3 Client-side Form Validation

- The downside of using only server-side validation for forms is that errors in the user's input will only be discovered after the user has submitted the form. This means that most of the user's input will be wiped out when he is bounced back to the form to resubmit it. We've already seen that using a form which submits to itself helps solve that problem, as does creating a Back link that will pre-populate the form with his previous input, but another technique is the use of client-side validation. With JavaScript, we can detect errors in the user's inputs before the form is actually submitted.
- Let's take a look at `form1.html`:

[2]Source: Visual Revenue.

```
<!--

form1.html

A form without client-side validation.

David J. Malan
Computer Science E-75
Harvard Extension School

-->

<!DOCTYPE html PUBLIC
     "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title></title>
  </head>
  <body>
    <form action="process.php" method="get">
      Email: <input name="email" type="text" />

      <br />
      Password: <input name="password1" type="password" />
      <br />
      Password (again): <input name="password2" type="password" />
      <br />
      I agree to the terms and conditions:
      <input name="agreement" type="checkbox" />
      <br /><br />

      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

We notice that this form submits to `process.php`, which actually just spits out the contents of the `$_REQUEST` superglobal. There's actually no JavaScript that's performing client-side validation, so if we leave one of the fields empty, we still get through. `form2.html` takes care of this along with a few other sanity checks:

```
<!--
```

```
form2.html

A form with client-side validation.

David J. Malan
Computer Science E-75
Harvard Extension School

-->

<!DOCTYPE html PUBLIC
     "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script type="text/javascript">
    // <![CDATA[

        function validate()
        {
            if (document.forms.registration.email.value == "")
            {
                alert("You must provide an email adddress.");
                return false;
            }
            else if (document.forms.registration.password1.value == "")
            {
                alert("You must provide a password.");
                return false;
            }
            else if (document.forms.registration.password1.value !=
                document.forms.registration.password2.value)
            {
                alert("You must provide the same password twice.");
                return false;
            }
            else if (!document.forms.registration.agreement.checked)
            {
                alert("You must agree to our terms and conditions.");
                return false;
            }
            return true;
        }

    // ]]>
```

```
    </script>
    <title></title>
  </head>
  <body>

    <form action="process.php" id="registration" method="get"
        onsubmit="return validate();">
      Email: <input name="email" type="text" />
      <br />
      Password: <input name="password1" type="password" />
      <br />
      Password (again): <input name="password2" type="password" />
      <br />

      I agree to the terms and conditions:
      <input name="agreement" type="checkbox" />
      <br /><br />
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

Now when we leave the `email` field blank and click Submit, an alert window pops up telling us that we must provide an e-mail address. This alert window has been annoyingly overused in the past and it doesn't look exactly the same in all browsers, but it is a very quick and easy way of delivering a message to the user.

- If we pull up Live HTTP Headers, we notice that in the case above when the user leaves a field blank, the form is never actually submitted to the server. In addition to improving the user's experience, this has the effect of reducing load on our servers.

- If we take a look at the bottom portion of the XHTML source for `form2.html`, we see that it's nearly identical to that of `form1.html` except that we've defined two additional attributes for the `form` element: `id` and `onsubmit`. The `onsubmit` attribute allows us to specify some JavaScript code when the user submits the form. Hence, `onsubmit`. In this case, we've assigned `return validate();` as its value. What this means it that the function `validate` will be called and if it returns true, then the form will be submitted. If it returns false, the form will not be submitted. Let's look at the code for `validate` to see when it returns true and when it returns false.

- `validate` is defined in the `head` of our page, which is fine for our purposes here, but may cause problems as our code gets more complicated. The `head` of the page is going to be loaded before the `body`. so if in the `head`,

we execute some JavaScript code which references an XHTML element that hasn't been loaded into memory yet because the `body` hasn't finished loading, then we're going to get a JavaScript error. There are ways to prevent this using event handlers and library code, but for now just be aware of the issue.

- Withint `validate`, we begin by checking the `email` field. We access this field by moving down the hierarchy of page elements (the DOM) beginning with the root object, `document`. The `forms` object, which belongs to `document`, contains all the `form` elements on our page. One of these `form` elements is `registration`, which contains the `email` field. Finally, this field has a `value` attribute. Altogether, we access the `value` of the `email` field like so:

  ```
  document.forms.registration.email.value
  ```

  Another way of accessing the same value, whether or not we had given the form a unique name, would be:

  ```
  document.forms[0].email.value
  ```

- Note that if our `onsubmit` attribute had a value of `validate()` rather than `return validate()`, then the form would go through regardless of what `validate` returned. It would assume the default value of true.

- By defining the `id` rather than the `name` attribute of the `form`, we also enable the use of a special method that belongs to `document`. So, to retrieve the same value of the `email` field, we could also write:

  ```
  document.getElementById('registration').email.value
  ```

- Question: these days, JavaScript is bundled with the browser, so it comes with it when you install. You don't need to worry so much about what version is installed.

- For the other fields in our form, we're checking if they're empty and, in the case of the password fields, if they match. Finally, we check if the checkbox has been checked by accessing its `checked` property.

- So was it worth it to write these lines of JavaScript code given that they can so easily be circumvented simply by disabling JavaScript in the browser? As we mentioned before, it saves us hits on our servers and also improves a user's experience by providing him more immediate feedback. Given the slow speed of the internet in some places, forcing a user to go back and forth between pages will waste a good deal of his time.

- Clearly, however, client-side validation **must** be backed up by server-side validation given that the former is so easily thwarted.

- There exist various frameworks, particularly in PHP, which allow you to very quickly create the logic for both server-side and client-side form validation. CakePHP and CodeIgniter are two very popular ones.
- In `form3.html`, we've cleaned up our code a little bit using the `with` syntax. To avoid having to access each DOM element beginning with with the root element `document`, we can write the following:

  ```
  with (document.forms.registration) {...}
  ```

  What this says is that for every DOM path between the curly braces that we haven't explicitly defined beginning with `document`, assume that it begins with this path. Saves us some extra typing, at least.
- In `form4.html`, we actually pass an argument to the `validate()` function:

  ```
  <form action="process.php" method="get" name="registration"
        onsubmit="return validate(this);">
  ```

  Now we redefine the `validate()` function so that it takes one argument:

  ```
  function validate(f)
  {
      if (f.email.value == "")
      {
          alert("You must provide an email adddress.");
          return false;
      }
      else if (f.password1.value == "")
      {
          alert("You must provide a password.");
          return false;
      }
      else if (f.password1.value != f.password2.value)
      {
          alert("You must provide the same password twice.");
          return false;
      }
      else if (!f.agreement.checked)
      {
          alert("You must agree to our terms and conditions.");
          return false;
      }
      return true;
  }
  ```

Thus, we can access the registration form as `f`, because we've passed it by reference to the `validate` using the `this` keyword, which references the object that called it. This function is now more useful because it allows us to vary the form which it validates (although the names of the fields are still hardcoded inside the function). It's a step in the right direction at least.

- As we mentioned earlier, alert windows are generally discouraged because they have different appearances in different browsers. For an alternative approach, take a look at HarvardNews and click on the RSS icon next to the Reset button. You'll notice a small window pops up in the center of the page while the rest of the page is dimmed out (via a semi-opaque overlay). Within this window, the font style is a little more consistent with the rest of the site and there's even a button which can be clicked to generate a URL that suits the user's choice. This is all done very easily using the Dialog module from the YUI Library. Note that version 3 of YUI has been released at least in part, but you should probably stick to version 2.

- Sidenote: one other useful feature of Firebug is the ability to highlight any part of a webpage, right click it, and select Inspect Element to jump directly to its place in the DOM. When an element is highlighted in the DOM like this (i.e. selected in blue in the left panel of Firebug), you can click the Styles tab in the right panel of Firebug to display all the CSS properties associated with that element.

- One small feature we can add is for the Submit button to be disabled unless the Terms and Conditions checkbox is checked. We do that in `form5.html` by writing a new function called `toggle()` which we assign as an event handler for the `onclick` attribute of the checkbox. The `toggle()` function is written like so:

```
function toggle()
{
    if (document.forms.registration.button.disabled)
        document.forms.registration.button.disabled = false;
    else
        document.forms.registration.button.disabled = true;
}
```

  All this logic does is to enable or disable the Submit button depending on whether it was disabled or enabled when the checkbox was clicked. As a result, a user is prevented from submitting his form if he hasn't agreed to our terms.

- The only enhancement we've made in `form6.html` is to put the `toggle()` function's logic inline:

```
<input name="agreement" onclick="document.forms.registration.button.disabled =
 !document.forms.registration.button.disabled;" type="checkbox" />
```

- In PHP, strings are primitives, but in JavaScript, strings are objects. They have, in fact, several built-in methods, one of which is `match`, which allows us to do some regular-expression matching. We leverage this in `form7.html` in order to actually check more than just whether or not the `email` field is empty:

  ```
  if (!document.forms.registration.email.value.match(/.+@.+\.edu$/))
  {
      alert("You must provide a .edu email address.");
      return false;
  }
  ```

  In this case, we're saying if the string doesn't contain `.edu` as a suffix, then spit out an error accordingly. Note that you don't have to quote the regular expression when you pass it to `match()`. This regular expression matches any character one or more times (`.+`), followed by a literal `@` sign, followed by any character one or more times (`.+`) followed by a literal `.edu` at the end of the string. We're also requiring that the `.edu` be in lowercase. We could take care of this by calling the `toLowerCase()` method of the string before calling its `match()` method. In fact, because even literal strings in JavaScript are actually objects, we can still access their methods using dot notation like so:

  ```
  "malan@post.harvard.edu".match(/.+@.+\.edu$/)
  ```

  This expression would return true, for example. This kind of syntax is not necessarily recommended though, if only for readability purposes.

- Realize that this regular expression is not very rigorous. It will allow through all valid e-mail addresses, but also strings like this:

  ```
  m@l@n@post.harvard.edu
  ```

  Generally speaking, most validations are somewhat lax with regard to e-mail addresses, but however stringent you are, be sure to include subdomains, or you'll thoroughly anger David with his `post.harvard.edu` address. Of course, there are other ways to validate e-mail addresses besides using regular expressions, so you don't necessarily need to do it this way unless perhaps a course specifically asks you to do so as a pedagogical exercise.

2.4 Syntax

2.4.1 Arrays

- To initialize an array in JavaScript, you can use **either** of the following lines of code:

```
var a = new Array();
var a = [];
```

The first line is actually invoking the constructor for an array while the second line is more common and recommended. Once you've initialized an array, you can add values to it using the following syntax:

```
a[0] = 'foo';
a[1] = 'bar';
a[2] = 'baz';
```

Hardcoding the indices of the array isn't always a good idea, however. What you can do instead is to invoke the built-in `length` method of the array:

```
a[a.length] = 'foo';
a[a.length] = 'bar';
a[a.length] = 'baz';
```

In the first line, the `length` method will return 0 since the array `a` has just been initialized. In the second line, however, the `length` method will return 1 because it has been updated since the addition of `'foo'` as an element.

2.4.2 Form Focus

- What are some of the tricks we can pull off using JavaScript? If we take a look at the CS 75 login page, we can see that the cursor is automatically placed in the username field when the page is loaded. We accomplish this using a few lines of JavaScript:

```
<script type="text/javascript">
// <![CDATA[
     // put cursor in username field if empty
     if (document.forms.login.username.value == "")
     {
          document.forms.login.username.focus();
     }
     // else put cursor in password field
     else
```

```
    {
        document.forms.login.password.focus();
    }
// ]]>
```

Turns out that form fields have a built-in method named `focus` which places the cursor within them. The lesson here is that just as strings are objects and have their own methods, so too are form elements. How would you know this method exists? Well, Google is your friend, for starters. But W3Schools and Mozilla have some decent documentation, as well.

2.4.3 Objects

- JavaScript comes packaged with multiple global objects, including the following:
  - Array
  - Boolean
  - Date
  - Function
  - Math
  - Number
  - Object
  - RegExp
  - String
- What exactly *is* an object, though? It's a collection of key-value pairs which amounts to a hash table, much like an associative array in PHP.[3] Ajax once stood for *asynchronous JavaScript and XML* because its requests generally returned data in XML format. Now, it has become popular to return data in JSON (JavaScript object notation) format, which frankly is much more convenient. Whereas traversing through an XML DOM can be quite tedious in JavaScript, traversing through an object's properties is generally quite painless in JavaScript.
- To initialize or *instantiate* an object, use the following syntax:

```
var obj = new Object();
var obj = {};

obj.key = value;
obj["key"] = value;

var obj = { key: value };
```

[3] Although, arrays in PHP can't have functions associated with them.

Note that the first two lines are equivalent to each other, the second two lines are equivalent to each other, and the last line is a combination of the first two groups. Using the dot notation is generally recommended.

2.4.4 Events

- Here's a small sample of the events for which we can assign handlers:
  - `onblur`
  - `onchange`
  - `onclick`
  - `onfocus`
  - `onkeydown`
  - `onkeyup`
  - `onload`
  - `onmousedown`
  - `onmouseup`
  - `onmouseout`
  - `onmouseover`
  - `onmouseup`
  - `onresize`
  - `onselect`
  - `onsubmit`

  In years past, the most popular event to leverage was `onmouseover`, which could be used to enlarge an image when a user moved the pointer over it. Now, however, many other tricks are implemented using these events. Take the autocomplete feature of HarvardMaps, for example. This is implemented using a listener for the `onkeyup` event. When the user has entered a few characters and then paused typing briefly, an HTTP request is sent to the server to find matches for this half-completed query. Google does this in a similar fashion, although instead returning a JavaScript object, it seems to return an actual function call with a JavaScript array as its argument.

2.4.5 Manipulating Style

- JavaScript can be used to manipulate the style of a webpage, mainly by accessing the `style` and `className` attributes of elements.
- David used JavaScript to manipulate style in his HarvardEvents page. If you click on any of the events, a description of it will open underneath it. To do this, a `div` containing the description of the event is by default loaded with `display` set to `none`. Then when the event is clicked, this property is toggled to `block`.

- Although the blink feature was arguably annoying, it's a shame that it was removed because it can still prove useful in some scenarios. To re-implement this feature, as David did for CS 50's site, in which office hours that were going on would blink to catch the user's attention, try the following function:

```
function blinker()
{
    var blinks = document.getElementsByName("blink");
    for (var i = 0; i < blinks.length; i++)
    {
        if (blinks[i].style.visibility == "visible")
            blinks[i].style.visibility = "hidden";
    else
        blinks[i].style.visibility = "visible";
    }
}
```

  The method `getElementsByName` returns an array which we loop through, toggling the `visibility` property. The `visibility` property holds space for the element on the page, unlike the `display` property, so that the page won't shrink and grow depending on if the element is visible or not.

- The problem we'll run into next, though, is that the `div` will only blink once if we don't find some way to call the function over and over again. Turns out we can do this by using a method called `setInterval`:

```
YAHOO.util.Event.addListener(window, "load", function() {
    window.setInterval("blinker()", 500);
});
```

  Here, we're calling the `blinker()` function every 500 milliseconds as soon as the page loads. The function is actually *anonymous* or *lambda* in that it has no name. This is useful if we think that we're never going to call this function again, so there's no need to store a pointer for it.

- YUI's role here is to standardize the way we add listeners for DOM events. In this case, we're adding a listener for the `window` object to be completely loaded.

- The YUI library provides a great way to interface with JavaScript and the DOM across multiple browsers. Check out this list of other JavaScript libraries:
  - Dojo
  - Ext JS
  - jQuery

  - MooTools
  - Prototype
  - script.aculo.us
  - YUI

- We'll talk in weeks to come more about static code analysis, cross-browser quirks, as well as debugging and compression.