Computer Science 75 | Lecture 7: March 30, 2009
Spring 2009 | Andrew Sellergren
Scribe Notes

Contents

1 David's Extracurricular Exploits (0:00–14:00) 2

2 Introduction to JavaScript(14:00–23:00) 2

3 JavaScript Syntax (23:00–53:00) 3

4 Form Validation and More (53:00–10:00) 6

1 David's Extracurricular Exploits (0:00–14:00)

- Recently, David undertook a project to improve Harvard's online campus map located at map.harvard.edu. If you check out the old site, you can see that it's fairly useful up to a point. The buildings are all demarcated by polygon outlines and labeled with their names. It's searchable, too.
- Still, there's obviously room for improvement. As you'll learn for Project 3, Google Maps has a fantastic API. Given this, as well as an XML file from Harvard containing the names of buildings on campus as well their latitude and longitude coordinates and polygon outlines, David was able to create a mashup which displayed Harvard's campus using Google Maps.
- One of the problems that David encountered has to do with the way Google Maps is implemented. It's actually just a series of PNG tiles which are requested from the server as needed. This presents a problem, though, when trying to add your own objects to the map. You need to cut and scale them just as the map tiles are cut and scaled. Fortunately, others out there have encountered this problem and written their own scripts to handle it.
- A few of the features still need some tweaking. The rendering of all campus building outlines takes so long on some computers that it crashes the browser. At low enough magnification, the display of the building labels renders as a huge black splotch because they all overlap. Still, there are some cool tricks like auto-complete which David implemented using Ajax. And they'll only get better (we hope)!
- Another pet project of David's is the Harvard event aggregator. What this does is download data from a number of different event calendars and compiles them into a nice readable format for people to peruse. Using a bit of JavaScript and Ajax on the front end, users can toggle more information about a particular event.
- One of the biggest issues that David has encountered (and you will encounter whenever you're doing web development) is that of *cross-browser compliance*. Basically, it ain't gonna be easy. Ahem, excuse me. It's difficult is what I mean.

2 Introduction to JavaScript(14:00–23:00)

- Unlike all the languages we've looked at so far, JavaScript is entirely client-side. It's an interpreted language, like PHP, meaning that it must be passed to an interpreter at runtime. In the case of JavaScript, however, that interpreter is packaged with the user's browser.
- The first major gotcha regarding JavaScript is that the user can disable it. This is less common these days, but in the case of browsers on mobile

phones, for example, it might be a problem if your site absolutely depends on JavaScript in order to function properly. The overarching goal is for *graceful degradation*, which is to say that if the user's browser is fully supported, then he'll have access to all of your site's features. If his browser is only partially supported, then your site should still function even if not to its fullest.

- What's the major advantage of using JavaScript? As we'll see, it can be used not only for stylistic details, but also for parsing and dynamically displaying data. If you can do some or all of your data manipulation client-side, then you need only transfer the raw data from server to client, which amounts to fewer bytes than formatted data would entail. JSON (JavaScript Object Notation), for example, is a more lightweight way of transmitting data than XML because it doesn't require a whole lot of metadata to be coupled with it.
- One of the Ajax-driven sites that you're probably familiar with is facebook. As you may have noticed, you can click on links on facebook without having to reload the chat bar. This is a neat trick that they and Gmail have pulled off using Ajax, which dynamically requests content on-the-fly. To you, the user, the data loading appears seamless.
- The JavaScript references out there aren't as complete and reliable as the PHP references you're already familiar with, but a little Googling never hurt anybody.

3 JavaScript Syntax (23:00–53:00)

- JavaScript is embedded directly in the webpage using the `script` tag like so:

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

  Note that you can also specify what version of JavaScript is required for your webpage using the `language` attribute, but this is not generally done.
- Why are we enclosing our JavaScript within CDATA tags? This way, we don't have to worry about our JavaScript not being valid XHTML. Certain characters like the left-angle bracket (`<`) meaning "less than," might be interpreted as the opening of an XHTML tag, for example.
- The `script` tag can be embedded at the beginning of the webpage in the `head` element or else anywhere in the `body` element. In some cases, (e.g. Google Analytics) it's recommended that the JavaScript be inserted just

before the closing HTML tag. This is because asynchronicity introduces the problem of race conditions. What if your script loads and begins executing before the page has fully rendered? You may encounter runtime errors if, for example, you begin searching for a node in the DOM before that node has actually been loaded. Thankfully, there are JavaScript libraries available which solve this problem by listening and waiting for the DOM to be fully loaded before executing your script.

- Another way of embedding JavaScript is by removing it to a source file like so:

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

  While it's tempting in this case to include the forward slash within the open tag so that `script` is an empty element, this will introduce all sorts of problems. Just bite the bullet and add that closing tag!

- The `noscript` tag are used to display, at the very least, some kind of message to users who have JavaScript disabled.

- Question: does `noscript` apply only to JavaScript? Most likely, no. It probably applies also to VB script which can be embedded in webpages.

- All variables in JavaScript are objects, primitives, or arrays. Even strings are implemented as objects such that they have built-in methods associated with them. Primitives are the data types which come native to JavaScript. Among them are string, number, and boolean.

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

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

  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:

  ```
  x[0] = ‘a’;
  x[1] = ‘b’;
  x[2] = ‘c’;
  ```

  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:

  ```
  x[x.length] = ‘a’;
  x[x.length] = ‘b’;
  ```

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

- 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 associative arrays in PHP. 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 };
```

  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.

- 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();
```

```
            document.forms.login.username.value =
                document.forms.login.username.value;
        }
        // else put cursor in password field
        else
        {
            document.forms.login.password.focus();
            document.forms.login.password.value =
                 document.forms.login.password.value;
        }
    // ]]>
```

As you can see, with the dot notation, we're following a hierarchy of page elements. This hierarchy is the DOM itself, the root being the `document` element. The `forms` element contains all of the HTML forms on the page. From there, `login` refers to the name of the particular form and the `username` refers to the field of that form that we're interested in. The `focus()` method is a built-in method which places the cursor on that element.

- What's the point of the second line of code in each of these blocks? Actually, in this case, nothing. What it was *supposed* to do was make sure that the cursor would be placed at the end, rather than the beginning, of any input that was already in the field. In this case, however, we've checked that the field is empty to begin with, so it doesn't matter.
- One of other useful application of JavaScript is client-side form validation. A question that has arisen on the Bulletin Board is why is server-side validation necessary if client-side validation is implemented? Well, it's very easy for a user to disable JavaScript and thereby circumvent all client-side validation.
- Why do it at all, then? Arguably, it improves the user's experience because it saves them an HTTP request and provides immediate feedback. Additionally, it saves your servers some work!

4 Form Validation and More (53:00–10:00)

- If up until now, you haven't been using Firefox for development, you should really consider doing so. The wealth of tools and plugins available to help you peel back the layers of the DOM and look underneath the hood of various libraries is more than enough reason to use it.
- If we take a look at our first code sample, `form1.html`, we see that it's a basic HTML form that submits to a page which prints out the `$_REQUEST` superglobal. As a sidenote, if you see variables in this array that look like `__utma`, they most likely belong to Google Analytics. If you clear your cookies, they'll go away.

- The next version, `form2.html`, implements the most basic form validation by checking that the user has provided the same password twice. Let's take a look at the JavaScript which achieves this:

  ```
  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;
  }
  ```

  If we break this down, it's not too different from what we looked at before. One thing that we should note, however, is that we've abstracted this JavaScript into a function. We call this function when the user has clicked the Submit button by using the following line of code:

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

  Here, we see that the `form` tag has an attribute called `onsubmit`, which, as you might've guessed, is given as its value a snippet of code that is to be executed when the user submits the form. Note that because we're using `onsubmit` and not `onclick`, this code will be executed if the user hits Enter or clicks the Submit button himself.

- One convention of JavaScript to be aware of is that semicolons are not explicitly necessary after lines of code. Newlines characters will be interpreted as the end of a line of code. Don't be surprised, then, if you see semicolons left out of code samples online. We encourage you to use semicolons, however, as a matter of good practice.

- We return false from our `validate()` function because the `onsubmit` attribute expects a boolean. If it returns true, then the form will be submitted normally. If it returns false, then the user we'll be bounced back to the page. Here, you can see that we're returning false in all cases where the user has supplied invalid input.
- In accessing the DOM elements, we can see that text fields have a property called `value` which holds their input and checkboxes have a property called `checked` which is a boolean.
- 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);">
```

  This, of course, necessitated that we redefine the `validate()` function so that it would take 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;
}
```

Now we can access the registration form as `f`, because we've passed it to the function 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.

- 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 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.

- 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 `form7.html`, we finally get down to the business of form validation using regular expressions:

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

What we're leveraging here is the fact that in JavaScript, every string has a built-in method called `match()` that takes as its input a regular expression. In this case, we're saying if the string doesn't contain `.edu` as a suffix, then spit out an error accordingly.

- If you know you're going to be using a regular expression a number of times, you can define it as an object of type RegExp and compile it so that it can be quickly invoked.
- Let's say we wanted to put our error message inline rather than in an annoying alert box. We could create a `div` with ID `message`, then add the following lines of code to our if statement:

```
if (!document.forms.registration.email.value.match(/.+@.+\.edu$/))
{
    var el = document.getElementById('message');
    el.InnerHTML = "Invalid e-mail!";
    return false;
}
```

  This is actually the unofficial way of inserting content into a webpage, but it happens to be faster than the official way, which is to use the `createElement` and `addAttribute` methods.
- This is a good opportunity to extol the virtues of Firebug, an amazingly powerful tool for web developers. Not only does it allow you to debug JavaScript, but it also allows you to rifle through the DOM. Another useful tool is JavaScript Debugger, which tends to catch some of the errors which Firebug doesn't.
- David, for example, used Firebug just the other day to figure out how Google Maps positions its buttons on its map interface using CSS since he wanted to replicate this for his HarvardMaps project.
- 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`

- Consider David's autocomplete feature on the HarvardMaps project. Why wouldn't it be a good idea to show autocomplete options the very moment that the user begins typing? If we take a look at Live HTTP Headers when we're typing into the search box, we see that a half-finished query is actually sent to the server to provide suggestions. For performance reasons, then, it makes sense only to query when the user has paused in his typing to avoid unnecessary requests to the server.

- Of course, there's the option to preload all the search suggestions on the client-side to avoid requests to the server at all. With a small amount of data, such as the 600 or so buildings on Harvard's campus, there's no real performance gain for doing this even though it would be easy to load it up when the page loads. For very large datasets, such as Google searches, for example, this isn't really a practical possibility.

- With JavaScript, you can also change the `style` attribute of your DOM nodes. If you wanted to hide an element, for example, you'd change its `display` property under `style` from `block`, the default, to `none`. This is what's going on underneath the hood for the HarvardEvents page when you toggle one of the events. Another style element, `visible`, allows you to hide an element but still display the amount of space that it would normally take up if it weren't hidden.

- You should know that there's a conversion you need to make between CSS style element names and their counterparts in JavaScript. Some CSS style elements have names with hyphens in them, which would equate to minus signs on JavaScript if they weren't translated. The general rule is that if you have a style element named `font-size` in CSS, the corresponding JavaScript name is `fontSize`.

- 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 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.

- 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