Computer Science E-75: Building Dynamic Websites
Lecture 5: JavaScript
March 10, 2008

- Grading projects:
  - We evaluate projects (proj 1 and onwards) with the following things in mind: correctness, design, style, configuration, and prose (from your readme).
- Question: my MySQL install on my computer gives me a strange error message
  - Either connect to cs75.net's database from your local machine or email the staff
-
- This lecture we're going to go a bit client-side: UI tricks
- Some references we recommend are found in tonight's lecture slides.
- JavaScript used to be limited to gratuitous use of pop up menus, not really that elegant. It is now more mature and sophisticated.
  - Google makes brilliant use of Javascript in their web applications, including Google Maps and Gmail.
- Javascript is **client-side**. This means it is not executed on the server and run on the end user's machine. This means:
  - You can do data verification and checks on the client's side before any data is ever sent to the server.
- How to insert Javascript code in your web page:
-
```
<script type="text/javascript">
   // <![CDATA[
   ....
   // ]]>
   </script>
```
    - Why the comments?
      - For the purposes of validation, you wrap the javascript code in the CDATA tag so that validators and web browsers do not parse the data as XHTML.
- This is tradtionally inserted into the <head> tag in a page, but you can put it anywhere in the body as well.
- You can also insert Javascript into the page with either of these:
  - `<script src="file.js" type="text/javascript"></script>`
  -
```
<script language="Javascript1.5">
// <![CDATA[
....
// ]]>
</script>
```
  - The code directly above is generally not used, but it can be done.
  - `<noscript>` tag is used for a web browser to execute if it does not recognize (or cannot execute) the code within <script> tags. This is decreasingly useful. It cannot have any code inside of it and the web browser will display what is in between <noscript> tags verbatim.
- **Statements**
  - Syntactically, it's very similar to C/C++, PHP, even Perl.

    - break, const, continue, do...while, for, for...in, for each...in, function, if...else, return, switch, throw, try...catch, var, while, with
    - **Var** lets you define a variable. Javascript, like PHP, is weakly typed.
    - It is not at all like Java, so don't get it confused!
- Examples
    - On the course website, when you visit the login page, you do not have to click on the username field to begin typing your username. This is thanks to a javascript trick:
        - At the very end of the login page we have:

```
   <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.log
in.username.value;
     }

     // else put cursor in password field
     else
     {
         document.forms.login.password.focus();
         document.forms.login.password.value = document.forms.log
in.password.value;
     }

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

    - What could this mean?
        - It's saying that for the username field (in the login form within the document), we test to see if it's empty, and if it is, it uses the `focus()` method to put the focus of the keyboard on the field itself.
        - The next line (setting the value equal to itself) is basically a hack that ensures that the cursor is at the end of any text that is in it, rather than (as is the case with some browsers), at the beginning before any data in the field.
    - You can think of "document" as being similar to a "superglobal variable" from PHP. From there, we can access all of the objects in the actual document plus associated methods and functions.
    - Document.forms, then, references all of the <forms> in the document. If you were to peruse the code (visible from http://cs75.net/lectures/5/src/login/index.phps), you see that we name one of our forms "login", and the username field is named "username". In this way, we are able to access it via the document object.
    - We could access it in some other way. For example, by using an ID:

    - `<input name="username" id="loginform"...`
    - Then you can access it by ID, rather than by name, by using:
    - `document.getElementById("loginform")`
    - Note, however, that you can only have ONE element in the document with a specific ID (that is, no two elements can have the same ID), but you can have different elements in the same document with different names.
  - Notice that, right now, the code is at the end of the document. We can assume that once the entire document has been loaded into memory, we can access all of the objects within it. What if we instead did:
  - `<script type="text/javascript">`

```
// <![CDATA[
     function init()
{
    // 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.log
in.username.value;
    }

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

  - In this case, we've wrapped the code into a function. It would not run by default, then. We would have to run the function at some specified time. It is possible to call the function after the page has loaded by using 'onload':
    - `<body onload="init()">`
- **Client-Side Validation**
    - This can be done entirely in Javascript. But! You still have to check on the server because javascript can be disabled, or some malicious user can still send bad data via raw HTTP headers.
    - It's, unfortunately, more work for you to have to write code to perform both server- and client-side validation, but it helps ensure better user experience (for example, so that a user does not have to reload the entire page just to find out that they have not filled out the form correctly)
    - form1.html (found in: http://cs75.net/lectures/5/src/forms/) is just a simple form that submits to process.php.
        - What are some things we could check on the client side in this form?
            - Verify that the email is valid

        - Make sure that the two password fields match
        - make sure that all of the fields are filled out
- form2.html:
    - checks to make sure that:
        - Something has been entered in the email address and first password field.
        - First and second password field match
        - The 'I agree to terms and conditions' box has been checked.
    - Note that there is a new XHTML attribute in the form element: `onsubmit`
        - `onsubmit` tells the browser to run a function when the form is submitted. In this case, the browser will execute the validate function that ensures the above is fine. If the function returns true, the browser will submit the form. If the function returns false, the browser does NOT submit the form.
    - If you were to try submitting the blank form, it gives you a popup telling you to enter an email address.
        - Using an alert popup is sometimes annoying, but we used it because it's easiest to implement. We'll show you other methods in the future.
    - Note that the email check is not very smart since it just makes sure that the email is not null. You could therefore get around it by simply inserting a a space in the field. We'll fix this in just a minute..
    - Note the code says this:
        - `document.forms.registration.email.value == ""`
        - It would NOT be correct to test for a null value this way:
        - `document.forms.registration.email == ""`
        - Because "`document.forms.registration.email`" is an OBJECT and will not return the value that is stored within that object. You must, then, use the ".value" method to get the value returned to you.
- form3.html:
    - Notice the use of "with". This allows us to test all of the values more easily than having to re-type 'document.forms.registration' in every if statement. Since the with statement encloses lines of code with { and }, you can define where the "with" statement ends.
- form4.html: another approach to the same idea
    - In this case, we pass the form object to the function by defining the function to accept an argument "f".
    - Since Javascript is loosely-typed, we can pass objects into functions and use them.
- form5.html: disable the submit button (temporarily)
    - Some websites disable the submit button (or change the text in it) to imply to the user not to click again. For example: when ordering something online, some sites disable the submit button after the first click so as to discourage the user from submitting an order twice.
    - The code requires that a user check the "Accept" box before enabling the submit button.
    - To disable the submit button, our XHTML code looks like this:

    - `<input disabled="disabled" name="button" type="submit" value="Submit" />`
    - Notice the disabled="disabled" attribute/value. This means that the button itself is disabled.
    - Also notice the agreement checkbox:
    - `<input name="agreement" onchange="toggle();" type="checkbox" />`
    - The "onchange" attribute (and its toggle() value) tells the browser to run the "toggle()" function when the value of the checkbox changes.
    - Take a look at the toggle() function:

    -
      ```
      function toggle()
      {
            if (document.forms.registration.button.disabled)
                  document.forms.registration.button.disabled = false;
            else
                  document.forms.registration.button.disabled = true;
      }
      ```
    - This means that, if the registration button is disabled, it will set the .disabled parameter to "false" (which enables it). Conversely, if the button is enabled, it will disable it.
    - Note that IE7 seems to have a problem with the "onchange" attribute, where it does not run the function unless the checkbox changes AND the focus moves elsewhere. Better, in this case, seems to be "onclick"
- form6.html: Simplifies this a little bit.

    -
      ```
      <input name="agreement"
      onclick="document.forms.registration.button.disabled = !
      document.forms.registration.button.disabled;" type="checkbox" />
      ```
    - This is a long line, but it simplifies the above process a bit. In it, when the agreement checkbox is clicked, it sets the disabled value to the opposite of what it currently is. That is, it sets it to "false" if it is "true", or "true" if it is "false".
- form7.html: finally! we do email validation (with regex)

    -
      ```
        function validate()
      {
         if (!document.forms.registration.email.value.match(/.+@.+\.edu$/))
         {
             alert("You must provide a .edu email adddress.");
             return false;
         }
         [..]
      }
      ```
        - `document.forms.registration.email.value.match` - this is an example of Javascript's loose data type, where objects can have value or methods. "Email" in this case, is an object in the document. "Value" is a method that returns a string of email. Match is a method for strings that test the regular expression against the string.

        - The regular expression is set between slashes rather than double quotes (as it would be in PHP).
        - This regular expression only tests to make sure there are some number of characters, followed by an @ sign, followed by some number of characters, followed by ".edu". So, a very loose check that makes sure that it is a .edu email address.
        - You can use a "RegEx" method to have the browser "compile" a regular expression that you might use frequently within one document. Unless you are using this regular expression very frequently within one page, this would not be useful for you.
- **Global Objects**
    - array, boolean, date, function, math, number, object, RegExp, string ..
    - Many of these should be familiar! But each of these objects have certain methods associated with them that may be useful.
- **Objects**
    - Javascript is object oriented.
    - An object in javascript is little more than a container of a name/method pair. The name/method may be a function/method, or name/value.
        - `var obj = new Object();`
    - define obj as an empty Object. The below is essentially the same:
        - `var obj = {};`
    - Now you can assign a key to the object:
        - `obj.key = value;`
    - An object is also represented as an associative array (and equivalent to the line above):
        - `obj["key"] = value;`
    - The line below does the same as the lines above (first creating the object, and then inserting some value as its key):
        - `var obj = { key: value };`
    - Note that if you did something like this:
        - `var obj = 5;`
        - You're actually creating a variable of a number type.
- **Arrays**
    - Is another type of object, but it has special properties and methods associated with it (specifically, .length)\
    - To create an array, use one of the following ways (they are equivalent):
        - `var a = new Array();`
        - `var a = [];`
    - To insert new data you can do:
        - `a[] = "foo";`
        - `a[] = "bar";`
        - `a[] = "baz";`
    - Which is equivalent to:
        - `a[a.length] = "foo";`
        - `a[a.length] = "bar";`
        - `a[a.length] = "baz";`

        - This works because, as soon as an assignment has been made, a.length goes up by 1. a.length returns the length of the array.
- OOP - Prototype-based
    - Even though there are not classes, you can instantiate objects and, through the use of something similar to lambda-functions (such as those used in Scheme, LISP, or even PHP's create_function()), create methods within them.
    - Don't worry, you won't need to use this when writing javascript for this course, but it is useful to know if you are looking through libraries.
    - Example:
        - http://cs75.net/lectures/5/src/oop/oop1.html
- **Blink**
    - Blink was an HTML element that caused text to blink on and off.
    -
```
function blinker() {
    var blinks = document.getElementsByName("blink");
    for (int i=0; i < blinks.length; i++) {
            if(blinks[i].style.visibility == "visible")
                    blinks[i].style.visibility == "hidden";
            else
                    blinks[i].style.visibility == "visible";
    }
```
    - This function will toggle the visibility (by using CSS's visibility attribute, note that we set this by using style.visibility) of any objects that have the name "blink" associated with them.
    - But, how does this function run?
        - With YUI!
        -
```
YAHOO.util.Event.addListener(window, "load", function() {
        window.setInterval("blinker()", 500);
}
```
        - We use YUI to manage running the blinker() function every 500 milliseconds (every .5 second), rather than implementing a handler ourselves that runs code every 500 ms.
- **Frameworks**
    - YUI, script.aculo.us, Dojo, Prototype
        - All are open-source and takes a lot of the guesswork out of creating neat animations or UI features that can enhance your page.
        - YUI, for example, has library utilities that include: Animation, drag and drop, imageloader (pre-load images in the background before showing them). AutoComplete, button, calendar, color picker, container, menu, slider, tabview, treeview
- **Quirks**
    - If you're trying to debug some strange corner case, or trying to debug some method that one browser would have that another doesn't, try googling the problem and appending "quirksmode" to it.
    - A person wrote a book called "QuirksMode" and has answered many questions regarding strange javascript quirks and issues.
- **Debugging**

    - FireBug and JavaScript Debugger are FireFox extensions that help you debug javascript.
- **Compressor**
    - The worry about client-side code (such as javascript) is that anybody can read it and use it.
    - A compressor is to save space, it may replace white space and lengthy variable names with shorter ones.
    - Some compressors might obfuscate code, but it is not really 'encrypted'. A malicious user can still reverse engineer the code.