Intro to JavaScript

ACM Webmonkeys @ UIUC, 2010

What is JavaScript?

- Javascript is a scripting language used primarily on the web for client-side scripting
  - Also used on Palm's WebOS for app development
- Has nothing to do with Java
- Loosely-typed, dynamically bound
- Supports objects and closures, making it suitable for larger-scale projects

"Client-side scripting"

- In the context of web apps, Javascript is a client-side language.
  - This means it runs **in the web browser**, NOT on the server
- You **cannot** directly access your database or any files on your web server from JavaScript the way that PHP can
  - You *can* communicate with the server via separate HTTP requests (this is how AJAX, etc. work)
- Always keep in mind that the code is running on the client's computer! Do not expect it to be secure!
  - Browsers can execute arbitrary JS on your page
- JS is meant to improve usability and add fancy effects to your web pages' interfaces.

One last note before we get started

- Not all browsers will have JavaScript enabled.
    - Plug-ins like NoScript will block your scripts from running by default
    - Bots (from Google et al.) do not have JS enabled.
- So, if you're using JavaScript on your site, make sure you have a degradation strategy for users without it
    - If needed, this can be "Please enable JavaScript to view this page"

Adding JavaScript to your webpage

- To include a script on your webpage, you need to add a <script> tag into the HTML. The syntax of the script tag is:

```
<script type="text/javascript">
  // JavaScript goes here.
</script>
```

- Or, if you have an external JS file to include:

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

- The script is evaluated by the browser when it is first encountered.

Triggering JavaScript from your page

- To call a JavaScript function (you cannot call a script tag explicitly, you must wrap your code in a function), use one of two methods:
  1. A link (**<a>** tag) with an href of javascript:myFunc()
     - <a href="javascript:doThisStuff();">Click Here!</a>
  2. An **event** in any tag:
     - <div onclick="doThisStuff();"></div>
     - <button onclick="callMyFunction('blah');">
- Note that events include onclick, onmouseover, onmouseout, onload, etc., and can be added as attributes to any HTML tag.
  - Use of these events is often referred to as DHTML (dynamic HTML), but that doesn't really mean much.

The basics of JavaScript

- Variables in JavaScript are loosely-typed, so you don't need to define what type they are.
  - var x; // defines x as a variable. Can be omitted.
  - x = 4; // sets x to 4.
  - x = "str"; // sets x to a string value, even if it was an int.
  - y = x + 4; // operations work as in C
- You can concatenate strings and variables with the + operator, so the above example would've set y to "str4".
- Arrays in Javascript are implemented as objects, so to make an array you do:
  - myArr = new Array(); // (next line overwrites)
  - myArr = [4, 2]; myArr.push(3);
  - myArr is now [4, 2, 3];
- Can do associative arrays as well (i.e., string keys)

Control structures, etc.

- Javascript has the same **if** statements and **for** and **while** loops that you'd expect (syntax is the same as C/PHP/etc)
- There's also the for-each structure, which iterates over each object in a collection (e.g., an array):
    - for (**obj** in **myArray**) {
    -     alert("now looking at " + obj);
    - }

Functions in JavaScript

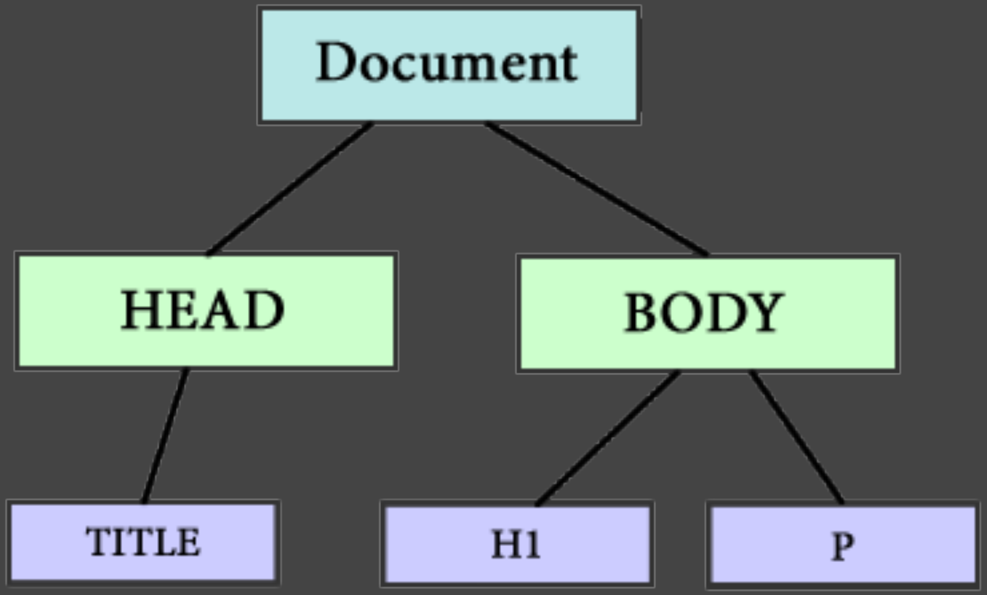
- Functions in JavaScript can be defined straightforwardly:
  - function myFunction(x, y, z) {
  -     return x + y + z;
  - }
- JavaScript is also a functional language, however, so you can even assign functions to variables (closures):
  - foo = function(x, y, z) { return x + y + z; }
  - alert(foo(1,2,3)); // alerts 6
- This latter property will come in handy later, but don't worry if you don't understand it immediately.
- A useful built-in function is the **alert** function, which brings up a dialog box containing the passed message.
  - Good for debugging on a tight schedule.

But how do we get anything done?

- In general, just adding numbers and concatenating strings in JavaScript is pointless unless we somehow output our result, or cause some sort of change in the page.
- To do this, we need to interact with the page itself.
  - We could use the function document.write() to write onto the end of the page, but that's only so useful.
  - The best way to do this is via the DOM model, which will let us manipulate individual elements on the page.

The Document Object Model (DOM)

- The Document Object Model, or DOM, is an abstract model of a web page. Rather than looking at a page as a string of HTML, we consider the HTML as the elements it defines and create a **tree**:



What's in a DOM element?

- Any DOM element has the following properties:
  - **innerHTML** - returns the HTML text inside this element.
    - e.g., document.body.innerHTML will return the text inside the body tag
  - **outerHTML** - if possible, returns the inner HTML text plus the element's tag
    - Doesn't work for body, but if you have some paragraph tag, myparagraph.outerHTML would return <p>blah blah blah</p>
  - **children** - List of the element's child elements.
- You can also access any HTML or CSS attribute, like:
  - mydiv.align = "center";
  - myparagraph.style.width = "400px";

Getting around the DOM

- Within any piece of JavaScript, you can use the global variable **document** to refer to the root node of the DOM tree. From the document element, you can access the properties **head** and **body**; for further levels of detail, however, we'll need to look up elements in particular.
- How can we look up an element?
  - Option 1: by name (e.g. find all divs)
    - use document.getElementsByName("div")
      - Returns an array.
  - Option 2: by ID (e.g. find div with ID "mydiv")
    - First, make sure the element has an ID:
      - <div id="mydiv">Blah</div>
    - Then, use document.getElementById("mydiv");
      - Returns the div itself, or null if not found.

Let's try it out: rewriting text

- Suppose we have a text field, and when the user clicks a link, we want the text field's value to be overwritten to the word "hello". How can we do this?
  1. Give the text field an ID
  2. Create a link, and trigger a JavaScript function from it
  3. Create a JavaScript function to change the text field's value property

The code

```
<html>
<head>
<script type='text/javascript'>
function changeTextField() {
    document.getElementById("myfield").value = "hello";
}
</script>
</head>
<body>

Text field: <input type="text" id="myfield" />
<a href='javascript:changeTextField();'>Click here!</a>

</body>
</html>
```

Recap

- So, all that's really going on is that we created a function in Javascript which used a little bit of DOM manipulation (finding an element within the document by a specific ID) and set its value property to the string "hello".
- Then, we bound that function to be called when the user clicked on the text link.
- Let's try something a little bit more complex next.

Collapse/Expand Text

- So we're writing an FAQ page, and we don't want each question's text to be displayed right away. We want it to be hidden until the user clicks on the question. How can we do this?
- First, we need to know about the **display** property in CSS. There are a few important display types that we care about:
  - **block**
  - **inline**
  - **none**
- "Block" refers to an element which takes up its own line, like a table or a div.
- "Inline" refers to elements which go on the same line as each other, like anchor tags or images.
- "None" means the element doesn't show up at all.

So, our plan is...

- If we put the text for each answer in a div, name each div, and then have clicking on the question trigger some showAnswer() function, we just need that showAnswer() function to change the display of that div from "none" to "block".
- Note that we'll have more than one div, so we'll need to pass in the ID of each answer as an argument to showAnswer().

The HTML

```
<html>
<head>
(on next page)
</head>
<body>
<h1>FAQ</h1>
<a href='javascript:showAnswer(1)'>1: Why use JS?</a>
<div id="answer1" style="display:none">
Why not?
</div>
<br/><br/>
<a href='javascript:showAnswer(2)'>2: Why not VBScript?</a>
<div id="answer2" style="display:none">
Because no one likes VBScript.
</div>
</body>
```

The JavaScript

```
<script type="text/javascript">
function showAnswer(num) {
    var theId = "answer" + num;
    var theDiv = document.getElementById(theId);
    theDiv.style.display = "block";
}
</script>
```

More advanced uses

- Okay, so we can do some simple stuff. But what about <insert fancy effect here>?
- A lot of the time, writing custom JavaScript is unnecessary. You can find dozens of solutions to most common problems out there.
    - Granted, a lot of those so-called solutions are crap. Make sure to look around for a good one before shoving it into your webpage.
- If you do want to write a custom script, you might find it helpful to start with a framework like **Prototype** or **jQuery**
    - They make it easier to do stuff like moving elements, timed events, and locating elements within the DOM.

jQuery

- jQuery is a library for JS that you can include on your page just by adding a script tag pointing to the jquery.js file you can download and put on your server.
- It is, essentially, a way to find objects more easily and act on them more intuitively.
- For example, if we want to change the color of every link on the page to gray, in normal JS we might do:
  - for (link in document.getElementsByName("a")) {
  -    link.style.color = "gray";
  - }
- In jQuery, we can just do:
  - $("a").css("color", "gray");

The Selector

- What was that $("a") stuff?
- In JavaScript, $ is a perfectly valid function name. So, $("a") is just calling the $ function with a parameter of "a".
- jQuery defines the $ function to be what's called the **selector**
  - As its name implies, it selects a set of elements.
- The selector doesn't return a DOM element, however: instead, it returns a special jQuery element which can actually be multiple DOM elements at once.
  - This is why we could change all link colors in one like -- the .css("color", "gray") call was being applied to every element returned by the selector.

Other ways to select things

- $("#mydiv") will select only elements with the id "mydiv"
- $("p:last") will select the last <p> element on the page
- $("/html/body/p/a") will select all anchor tags inside paragraph tags in the body element.
- There are lots more ways, too -- check out the API for a full listing.

Adding effects with jQuery

- One thing jQuery can do well is effects -- that is, making things fade in, move around, etc.
    - If you want even more effects, get scriptaculous (http://script.aculo.us/)
- If we wanted our <a> tags to all fade in, for example (I don't know why) we could do:
    - `$("a").fadeIn('slow');`
- You can attach a callback function as a second argument, to be executed for when the animation finishes.
    - `$("a").fadeIn('slow', function() { alert("Now you can see the links!"); });`

Summary

- JavaScript is a fairly rich language with extensive support.
- Most likely, your experience with JavaScript will be along the lines of "I want this to happen, and straight HTML can't do it. How can I do it in JavaScript?"
  - Start by Googling the problem (e.g. "js lightbox effect")
  - Look at the existing solutions and see if they fit your needs (remember to attribute as needed)
  - If they don't, or if the script is simple enough, write it yourself
- If you plan on using a lot of JavaScript (e.g. an AJAX app), definitely use a framework like Prototype or jQuery to save time and keep yourself sane.