Introduction to Web Engineering SENG2050/6050

Lecture 1d
JavaScript

Lecture 1d: JavaScript

- What is JavaScript?
- The JavaScript Language
- Browser Object Model (BOM)
- Document Object Model (DOM)
  - ✓Objects
  - ✓Events
- Help Functions

JavaScript

- JavaScript ≠ Java
  - It may sometimes “look” like Java, but it is not Java
  - It is Object-Oriented
  - It is much simpler than Java
  - JavaScript is fully interpreted (no “byte code”)
    - Risky!

JavaScript

JavaScript is a client-side scripting language

- It is embedded inside HTML and run by the browser
- It does not (generally) have access to data on the Web server
- It does have access to data input into HTML forms **BEFORE** this data is submitted to the server (this is the main use to which we will put JavaScript in this course)

JavaScript and HTML

JavaScript is embedded in HTML using a `<script>` tag, which should be placed in the `<head>`

```
<script type="text/javascript“>
   JavaScript statements
</script>
<noscript>
   Please go to this
   <a href="noscript.html">non-scripted
   page</a>
</noscript>
```

JavaScript and HTML

- The block of JavaScript is evaluated when the page is loaded by the browser
  - It should be used to define functions which can be invoked later
- The `<noscript>` tag's content is displayed by browsers which do not support JavaScript

JavaScript and HTML

JavaScript can be triggered in response to events in the Web page

- `<input type="button" value="Back" onclick="javascript:backfunc()" />`
- This example will call the function `backfunc` when the button is clicked
- It could also invoke any JavaScript expression e.g.,`onclick="javascript:history.back()"`

JavaScript Language

- `function` *name* (*parameters*) `{` *body* `}`
- Note that it has no return type
- *parameters* is a comma-separated list of parameter **names** – again, no types

JavaScript Language

- JavaScript is weakly (loosely) typed
  - The “type” of a variable is determined by the value assigned to it…
  - ```
    var myint=20;
    myint="Ok so now it’s a string";
    myint=20e10; myint=true;
    ```
  - **DO NOT DO THIS** – it leads to **VERY** messy, hard-to-follow code
  - Treat JavaScript as if it were strongly typed!
    ```

JavaScript Language

- JavaScript supports booleans (constants `true` and `false`), integers, floating-point numbers, strings (single- or double-quotes) and various objects
- JavaScript has the usual operators: `+ - * / % == != < <= > >= && || ! = += -= /= *= ++ --`
- JavaScript also has `===` `!==` which allow for a strict comparison
  - i.e. `0==false` vs. `0!==false`
- JavaScript has control structures: `if`, `if-else`, `while`, and `for`

JavaScript Language

Variables are declared with the keyword `var`

- The scope of a variable declared within a function is the body of the function
- The scope of a parameter is the body of its function
- The scope of a variable declared outside of a function, or a function name, is the entire Web page (including frames!) – effectively global

JavaScript Language

- JavaScript supports a simple object system
  - `var` *object* `= new` *Class*`(`*args*`)`
  - Many objects are predefined by the browser
- ```
  with (document) {
    title = "A Simple E.g.";
  }
  ```
  is the same as
  ```
  document.title = "A Simple E.g."
  ```
  - Useful if you need to set many properties of one object
  - Can be confusing
  ```

JavaScript Language

JavaScript also supports arrays

```
coffees = new Array("French Roast",
  "Colombian", "Kona");
message = "I like " + coffees[1] +
  "coffee.";
```

- `Array` is a predefined class
- It contains zero or more values of *any* type
- You access the a value by *array_name*`[`*index*`]`
- *index* ranges from `0` to *array_name*`.length-1`

JavaScript Language

- You can have arrays of arrays

```
myArray = new Array();
myArray[0] = new Array(1,2,3);
myArray[1] = new Array(4,5,6);
myArray[0][0] == 1
myArray[1][2] == 6
```

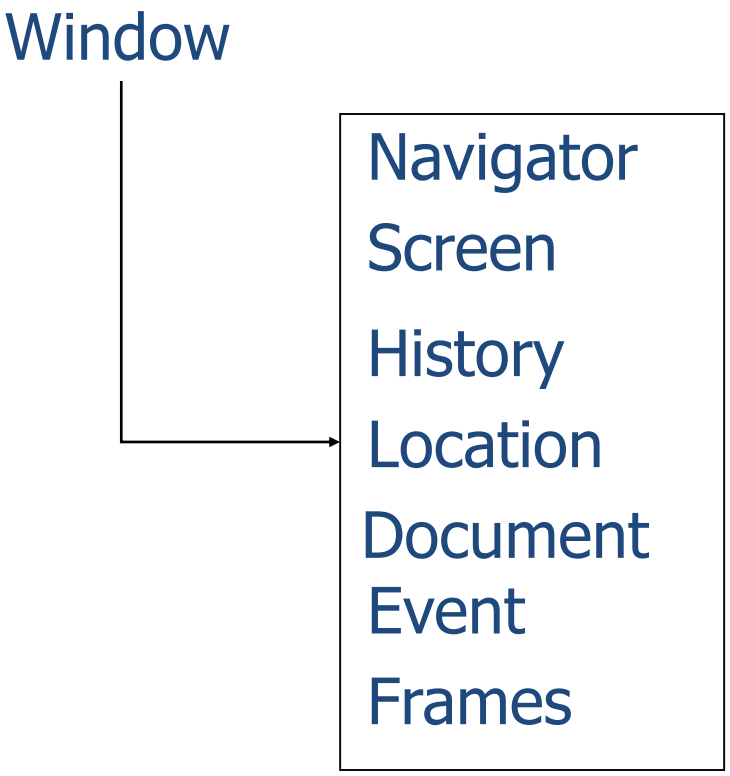
- Arrays can also be indexed by strings

```
wantCar["make"] = "Nissan";
wantCar["model"] = "Skyline GTR";
```

BOM

- BOM = *Browser* Object Model
  - A hierarchy of objects used to represent the Web browser for JavaScript to access
  - Has objects for Web browser, each of its "windows", the current URL, the browsing history, and the structure of the document being browsed
  - Some of this is browser dependent

Summary of Object Model



BOM - Window

Properties	Methods	Event Handlers
closed defaultStatus frames name opener parent self status top window	alert() blur() close() confirm() focus() moveBy() moveTo() open() print() prompt() resizeBy() resizeTo() scrollBy() scrollTo() setInterval() clearInterval() setTimeout() clearTimeout()	onLoad onUnload onFocus onBlur onError onResize

BOM

- `window`
  - A window of the browser – could be a full window, a tab, or a frame
  - Other BOM objects are properties of `window`
  - `name` – the name of the window
  - `close()` – close the window
  - `closed` – has the window been closed?
  - `status` – the string displayed as the browsers status line – you can set this, but you shouldn't

BOM

- `window`
  - `alert(`*message*`)` – pop up a dialog with *message* and an “OK” button
  - `confirm(`*message*`)` – pop up a dialog with *message*, an “Accept” button and a “Cancel” button – returns true if the user clicks on “OK”, and false if the user clicks on “Cancel”

BOM

- `window`
  - `prompt(`*message*`, `*default*`)` – pop up a dialog with *message*, a text input with the given *default* value, and an “Accept” button and a “Cancel” button – returns the text typed into the input if the user clicks on “OK”, or `null` if the user clicks on “Cancel”

BOM

- `navigator`
  - `appName` – which browser?
  - `appVersion` – which version?
  - This is **NOT** always a reliable way of determining which browser you have

Properties

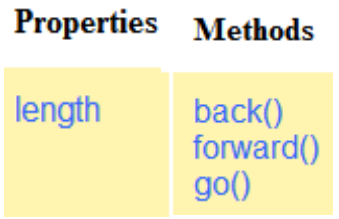
appCodeName
appName
appVersion
cookieEnabled
platform
userAgent

- `screen`
  - `availHeight` – actual displayable area ?
  - `availWidth` – actual displayable area ?
  - Opposed to the height and width properties, which give the actual size of the entire screen, in pixels.

Properties

availHeight
availWidth
colorDepth
height
pixelDepth
width

BOM



- `history`
  - `length` – how many items in the browser history
  - `back()` – go back one in the history
  - `forward()` – go forward one in the history
  - `go(`*n*`)` – go to the *n*th item in the history
  - Note: security restrictions on the browser might stop you using some of the history functionality

BOM

- `location`
  - `href` – the current URL
  - You can assign to it to load a new URL

    `location.href = absoluteURL;`
  - Also `protocol`, `host`, `port`, `pathname`, `hash`, `search`

Properties	Methods
hash host hostname href pathname port protocol search	reload() replace()

Document Object Model

- W3C Standard
- The Document Object Model is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents.
- The document can be further processed and the results of that processing can be incorporated back into (or even replace) the presented page.

DOM

- `document`
  - An object of the BOM that describes the document (XML or HTML) being displayed by a browser window
  - Defined by the Document Object Model
  - Netscape4 and IE5 defined not-quite compatible DOMs
  - W3C has attempted to standardise DOM (and extend it to handle arbitrary XML documents)

DOM

DOM's Property	Returns	Comments
`firstChild`	The first child node.	An `[object]`. All children are included in `childNodes` collection.
`lastChild`	The last child node.	An `[object]`. All children are included in `childNodes` collection.
`nextSibling`	The next child of the node's parent.	An `[object]`.
`nodeName`	The HTML Tag.	Examples: P, FONT, UL.
`nodeType`	Whether the node is a tag, text, or attribute.	Returns 1 for tag, 2 for attribute, and 3 for text.
`parentNode`	A reference to the parent node.	An `[object]`.
`previousSibling`	A reference to the previous child of the node's parent.	An `[object]`.
`specified`	Whether an attribute value is set.	Boolean

The following table shows the read-write properties:

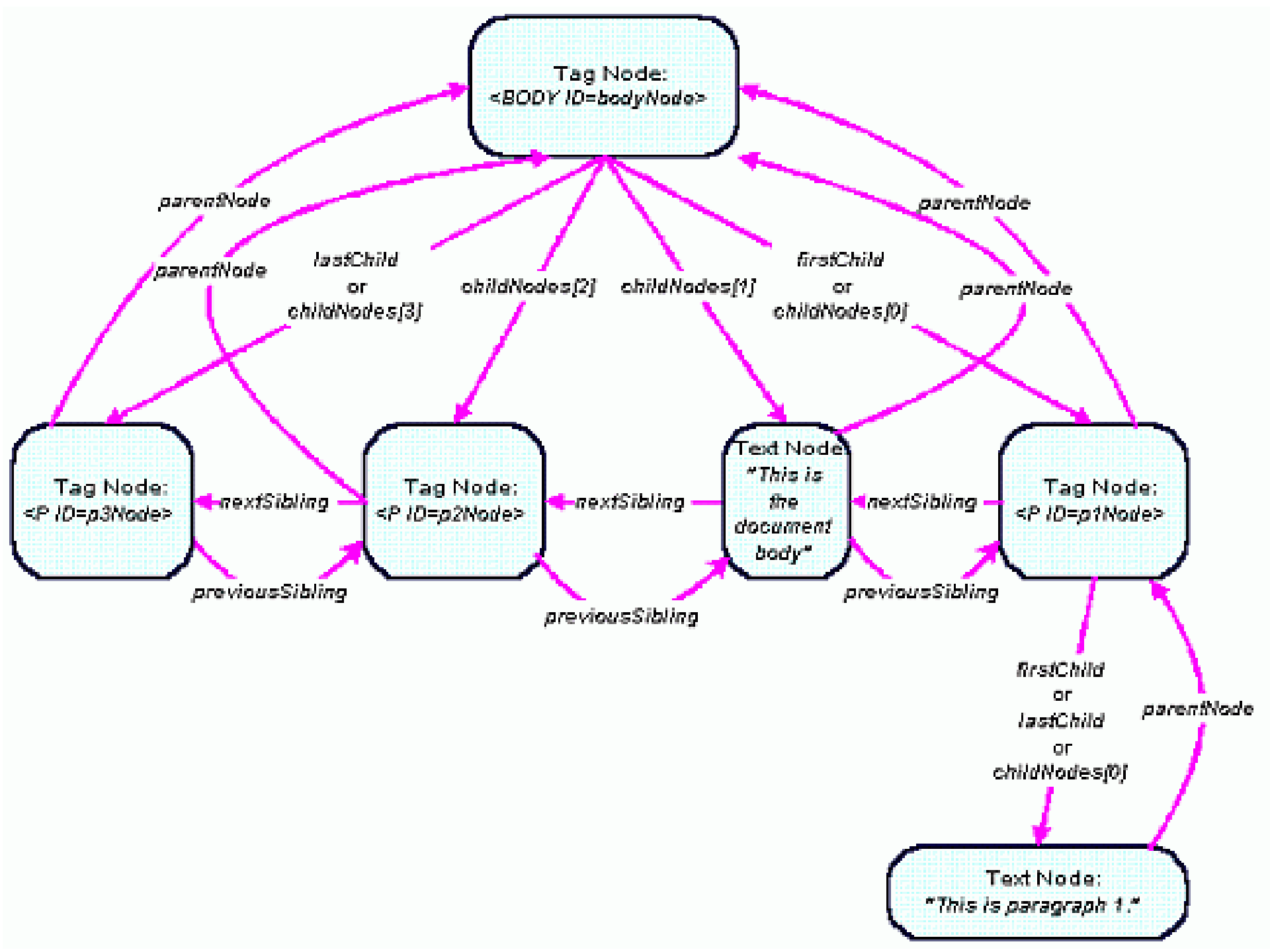
DOM's Property	Returns	Comments
`data`	The value of a text node.	A string. Returns `undefined` for all other nodes. Can be set as well.
`nodeValue`	The value of a text node.	A string. Returns `null` for all other nodes. Can be set as well.

And finally, here are the collections defined for the DOM:

DOM's Property	Returns	Comments
`attributes`	A collection of the node's attributes.	Access by name. Example:`attributes.id`.
`childNodes`	A collection of the node's children.	Access by index. Example:`childNodes[2]`.

DOM Example

```
<HTML>
    <HEAD>
        <TITLE> Simple DOM Demo </TITLE>
    </HEAD>
    <BODY ID="bodyNode">
        <P ID = "p1Node">
            This is paragraph 1.
        </P>
        This is the document body
        <P ID = "p2Node"> </P>
        <P ID = "p3Node"> </P>
    </BODY>
</HTML>
```


Tag Node:
<BODY ID=bodyNode>
parentNode
parentNode
lastChild
or
childNodes[3]
childNodes[2]
childNodes[1]
firstChild
or
childNodes[0]
parentNode
parentNode
Tag Node:
<P ID=p3Node>
nextSibling
Tag Node:
<P ID=p2Node>
nextSibling
Text Node
"This is
the
document
body"
nextSibling
Tag Node:
<P ID=p1Node>
previousSibling
previousSibling
previousSibling
firstChild
or
lastChild
or
childNodes[0]
parentNode
Text Node:
"This is paragraph 1."

DOM

- `document`
  - `title` – change the title of the document (the browser decides how to use this)
  - `URL` – the URL of the document
  - `referrer` – the URL of the document that linked to this one
  - `body` – an object representing the `<body>` tag

DOM

- `document`
  - `getElementById(`*id*`)` – returns the object representing the tag with attribute `id="`*id*`"` (there can only be one such tag)
  - `getElementsByName(`*name*`)` – returns the array object representing the tags with attribute `name="`*name*`"`
  - `getElementsByTagName(`*tag*`)` – returns the array object representing the set of all *tag*s

DOM

- Once you have a tag object…
  - `node.getNodeType()` – `ELEMENT_NODE`, `ATTRIBUTE_NODE`, `TEXT_NODE`, …
  - `node.getNodeName()` – tag name
  - `node.getNodeValue()` – text or attribute
  - `node.childNodes` – an array of all children
  - `node.firstChild`, `node.lastChild`
  - `child.nextSibling`, `child.previousSibling`

DOM

- Manipulating attributes
  - `node.getAttribute(`*attrName*`)` – the value of the named attribute
  - `node.setAttribute(`*attrName*`, `*newValue*`)` – change the value of the named attribute
  - `node.removeAttribute(`*attrName*`)` – remove the named attribute
- The objects for different tags have different additional properties and methods.

Forms

The objects for different tags have different additional properties and methods.

Form Object Collections

Collection	Description	IE	F	O	W3C
elements[]	Returns an array containing each element in the form	5	1	9	Yes

Form Object Properties

Property	Description	IE	F	O	W3C
acceptCharset	Sets or returns a list of possible character-sets for the form data	No	No	No	Yes
action	Sets or returns the action attribute of a form	5	1	9	Yes
enctype	Sets or returns the MIME type used to encode the content of a form	6	1	9	Yes
id	Sets or returns the id of a form	5	1	9	Yes
length	Returns the number of elements in a form	5	1	9	Yes
method	Sets or returns the HTTP method for sending data to the server	5	1	9	Yes
name	Sets or returns the name of a form	5	1	9	Yes
target	Sets or returns where to open the action-URL in a form	5	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Form Object Methods

Method	Description	IE	F	O	W3C
reset()	Resets the values of all elements in a form	5	1	9	Yes
submit()	Submits a form	5	1	9	Yes

Textarea

Similarly, the Password, Text and Hidden objects.

Textarea Object Properties

Property	Description	IE	F	O	W3C
accessKey	Sets or returns the keyboard key to access a textarea	4	1	9	Yes
cols	Sets or returns the width of a textarea	4	1	9	Yes
defaultValue	Sets or returns the default text in a textarea	4	1	9	Yes
disabled	Sets or returns whether or not a textarea should be disabled	5	1	9	Yes
form	Returns a reference to the form that contains the textarea	4	1	9	Yes
id	Sets or returns the id of a textarea	4	1	9	Yes
name	Sets or returns the name of a textarea	4	1	9	Yes
readOnly	Sets or returns whether or not a textarea should be read-only	4	1	9	Yes
rows	Sets or returns the height of a textarea	4	1	9	Yes
tabIndex	Sets or returns the tab order for the textarea	4	1	9	Yes
type	Returns the type of the form element	4	1	9	Yes
value	Sets or returns the text in a textarea	4	1	9	Yes

Standard Properties

Property	Description	IE	F	O	W3C
className	Sets or returns the class attribute of an element	5	1	9	Yes
dir	Sets or returns the direction of text	5	1	9	Yes
lang	Sets or returns the language code for an element	5	1	9	Yes
title	Sets or returns an element's advisory title	5	1	9	Yes

Textarea Object Methods

Method	Description	IE	F	O	W3C
blur()	Removes focus from a textarea	4	1	9	Yes
focus()	Sets focus on a textarea	4	1	9	Yes
select()	Selects the text in a textarea	4	1	9	Yes

Submit

```
<html>
    <head>
        <script type="text/javascript"> function validate(){ … }  </script>
    </head>
    <body>
        <form action="tryjs_submitpage.htm" onsubmit="return validate()">
          Name (max 10 char.): <input type="text" id="fname" size="20"><br />
          Age (from 1-100): <input type="text" id="age" size="20"><br />
          E-mail: <input type="text" id="email" size="20"><br /><br />
          <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

DOM Elements

DOM Anchor
DOM Area
DOM Base
DOM Body
DOM Button
DOM Event
DOM Form
DOM Frame
DOM Frameset
DOM IFrame
DOM Image
DOM Input Button
DOM Input Checkbox
DOM Input File
DOM Input Hidden
DOM Input Password

DOM Input Radio
DOM Input Reset
DOM Input Submit
DOM Input Text
DOM Link
DOM Meta
DOM Object
DOM Option
DOM Select
DOM Style
DOM Table
DOM TableCell
DOM TableRow
DOM Textarea

Events

Each DOM event can trigger a JavaScript method

- Which can then *replace* the DOM object's method!

```
function even() {
  alert("That was an even click");
  document.getElementById("count").onclick=odd;
}
function odd() {
  alert("That was an odd click");
  document.getElementById("count").onclick=even;
}
<input type="button" value="Click Me!" id="count"
  onclick="javascript:odd()" />
```

Helpers

JavaScript has some useful built-in functions

- `parseInt(`*string*`)` – converts as much of *string* as it can into an integer; if the string doesn't start with a valid integer then returns NaN (not a number)
- `parseFloat(`*string*`)` – similarly, converts as much of *string* as it can into a floating-point number
- `isNaN(`*value*`)` – test whether an above conversion resulted in "not a number"
- `eval(`*string*`)` – evaluates an arbitrary string as if it was JavaScript, then tries to execute it in the current context – be **VERY** careful when using this.

Helpers

JavaScript has some useful built-in objects

- `Math` – contains many mathematical constants and functions
- `Date` – for the manipulations of dates and times
- `String` – for the manipulations of strings
- `RegExp` – for applying regular expressions to strings

Cool stuff - AJAX

- Ajax (or AJAX) stands for **A**synchronous **J**avaScript **A**nd **X**ML
- Ajax can help to make your web applications more responsive

Ajax

- Use of the non-standard XMLHttpRequest object to communicate with server-side scripts
- Can send as well as receive information in a variety of formats, including XML, HTML, and even text files
- Most appealing characteristic?
  - "asynchronous" nature
    - which means you don't block and wait for the function to return
    - it can fetch new content without having to reload the entire page
    - allows you to update portions of a page based upon user events.
- But they still aren't as responsive as desktop applications
- The technology has been available for some time.

How Ajax works

- Browser displays a html page.
- JavaScript on the HTML page sends an HTTP request to the server
- The server responds with a *small amount* of data, rather than a complete web page
- JavaScript uses this data to modify the page

- This is faster because less data is transmitted and because the browser has less work to do

How Ajax works – Client side

- Events triggers an Ajax application– mouse clicks, mouse movement, keys, Dom objects.
- Usually from an HTML form -- providing data to the server
- JavaScript has to handle events from the form, create an XMLHttpRequest object, and send it (via HTTP) to the server
  - Despite the name, the XMLHttpRequest object does not require XML

The XMLHttpRequest object

- JavaScript has to create an XMLHttpRequest object
- There are three ways of doing this
  - For most browsers, just do
    `var request = new XMLHttpRequest();`
  - For some versions of Internet Explorer, do
    `var request = new ActiveXObject("Microsoft.XMLHTTP");`
  - For other versions of Internet Explorer, do
    `var request = new ActiveXObject("Msxml2.XMLHTTP");`
- Doing it incorrectly will cause an Exception

The XMLHttpRequest object

```
var request = null;
    function getXMLHttpRequest( ) {
        try { request = new XMLHttpRequest(); }
        catch(err1) {
            try { request = new ActiveXObject("Microsoft.XMLHTTP"); }
            catch(err2) {
                try { request = new ActiveXObject("Msxml2.XMLHTTP"); }
                catch(err3) {
                    request = null;
                }
            }
        }
        if (request == null) alert("Error creating request object!");
    }
```

The XMLHttpRequest object

- Once you have an XMLHttpRequest object, you have to prepare it with the open method
- ***request***.open(***method***, ***URL***, ***asynchronous***)
  - The ***method*** is usually 'GET' or 'POST'
  - The ***URL*** is where you are sending the data
    - If using a 'GET', append the data to the URL
    - If using a 'POST', add the data in a later step
  - If ***asynchronous*** is true, the browser does not wait for a response (this is what you usually want)

The XMLHttpRequest object

- Once the XMLHttpRequest object has been prepared, you have to send it
- ***request***.send(null);
  - This is the version you use with a GET request
- ***request***.send(***content***);
  - This is the version you use with a POST request
  - The content has the same syntax as the suffix to a GET request
    - request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
  - request.send('var1=' + ***value1*** + '&var2=' + ***value2***);
  - POST requests are used less frequently than GET requests

Ajax Server Side

- The server gets a standard HTTP request
- The response is a completely standard HTTP response, but instead of returning a complete HTML page as a response, the server returns an arbitrary text string (possibly XML, possibly something else)

Ajax Server Side

- Ajax uses asynchronous calls—you don't wait for the response
- Instead, you have to handle an event
  - request.onreadystatechange = someFunction;
    - This is a function assignment, *not* a function call
      - Hence, there are ***no parentheses*** after the function name
    - When the function is called, it will be called with no parameters
  - Function someFunction() {

```
if(request.readyState == 4){
    var response = request.responseText;
        if (http_request.status == 200) {
            // Do something with the response
        }
    }
}
```

request.readyState == 4

- The callback function you supply is called not just once, but (usually) four times
- request.readystate tells you why it is being called
- Here are the states:
  - 0 -- The connection is uninitialized
    - This is the state before you make the request, so your callback function should not actually see this number
  - 1 -- The connection has been initialized
  - 2 -- The request has been sent, and the server is (presumably) working on it
  - 3 -- The client is receiving the data
  - 4 -- The data has been received and is ready for use

request.readyState == 4

- A ready state of 4 tells you that you got a response--it doesn't tell you whether it was a *good* response
- The http_request.status tells you what the server thought of your request
  - 404 Not found is a status we are all familiar with
  - 200 OK is the response we hope to get

Ajax Server Side

- ```
  http_request.onreadystatechange =
  function() { showContentsAsAlert(http_request); };
  http_request.open('GET', url, true);
  http_request.send(null);
  ```

- ```
  function showContentsAsAlert(http_request) {
   if (http_request.readyState == 4) { /* 4 means got the response */
            if (http_request.status == 200) {
                 alert(http_request.responseText);
            } else {
                 alert('There was a problem with the request.')
            }
       }
  }
  ```

From: http://developer.mozilla.org/en/docs/AJAX:Getting_Started

Summary

- Create an XMLHttpRequest object (call it ***request***)
- Build a suitable URL
- ***request***.open('GET', ***URL***)
- ***request***.onreadystatechange = ***handlerMethod***;
- ***request***.send(content);
- function ***handlerMethod***() {//function content}

Drawbacks of using Ajax

- The back button doesn’t go “back”
- You cannot bookmark a particular state of the page
- JavaScript must be enabled
- Network delays can cause problems
- Search Engines cannot record the whole page properly
- Browser caching is an issue.

JavaScript Resources

- Lots and lots of JavaScript resources
  - http://www.pageresource.com/jscript/
  - Note: some of these use older versions of JavaScript; be sure to note the new conversions where they differ
  - http://www.webreference.com/programming/javascript/diaries/
- ECMAScript ("standardised" JavaScript)
  - http://www.ecma-international.org/publications/ standards/Ecma-262.htm
- W3C DOM 1 and 2 (and 3)
  - http://www.w3.org/DOM/DOMTR
- Another useful DOM page
  - http://xml.coverpages.org/dom.html

A Word on Commenting

- Comments in HTML
  - Between <!-- and -->
- Comments in CSS
  - Between /* and */
- Comments in Java and JavaScript
  - From // to end of line
  - Between /* and */
- **USE THEM!**
  - A well-commented document is **MUCH** easier to understand and debug (and to assess for marking)

THE END

QUESTIONS??

THANKS!!