Chapter 1
Introduction to JavaScript

Adapted from JavaScript: The Complete Reference 2nd and 3rd Editions
by Thomas Powell & Fritz Schneider
© 2004 - 2012 Thomas Powell, Fritz Schneider

Intro

• JavaScript is premier client-side scripting language used in Web development
  – Note especially in my definition
    • Client side (changed), Focus on web development, Scripting
    • Never limitations other than those self-imposed

• Highly misunderstood though increasingly popular

• Part of the client-side ‘triangle’ consisting of (X)HTML, CSS and of course JavaScript
  – Manipulation of mark-up and style via the document object model or DOM

First Look at JavaScript - Hello World

```html
<!DOCTYPE html>
<head>
<title>JavaScript Hello World</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h1>First JavaScript</h1>
<script>
    document.write("Hello World from JavaScript!");
</script>
</body>
</html>
```
Helloworld Deconstructed

• `<script>` tag used to delimit the script code from the HTML
  – The script tag causes the browser’s JavaScript interpreter to be invoked, the script run and any output produced
  – The browser is considered the “host” environment
    • There are other hosts for JavaScript and its variants

• The demo also shows how the script can write back out to the document in this case using the `document.write()` method

Being Aware of JavaScript’s Silent Failures

• Most browsers will give minimal feedback that a JavaScript failure is occurring
  – Look in the lower left corner of the status bar in IE to double click on the warning icon
  – You may see in Mozilla browsers a status bar message like JavaScript errors occurred or similar

• Make sure you can turn on your browser’s error reporting
  – IE (Tools > Internet Options > Advanced)
  – Mozilla (use javascript: URL or Tools > JavaScript Console)

• A little homework: Browse the Web with JavaScript error reporting on
Adding Script to (X)HTML Documents

- There are four standard ways to include script in an (X)HTML document:
  1. Within the `<script>` element
  2. As a linked file via the `src` attribute of the `<script>` element
  3. Within an (X)HTML event handler attribute such as `onclick`
  4. Via the pseudo-URL javascript: syntax referenced by a link

*Note: There may be other approaches but they are non-standard*

The `<script>` tag

- The `<script>` tag (`<script>...</script>`) in all major browsers interprets contents as JavaScript unless one of the following occurs:
  - Inclusion of language attribute
    `-- <script language="VBS">...</script>`
  - Inclusion of type attribute
    `-- <script type="text/javascript">...</script>`
  - The `type` attribute is W3C recommended, `language` more common and in many ways more useful
  - *Be careful of Mime types like application/javascript*

*Note: A `<meta>` tag can also be used to set the script language document wide or even by a Web server.
  - `<meta http-equiv="Content-Script-Type" content="text/javascript"/>`

Using the `<script>` Tag

- You can use as many `<script>` tags as you like in both the `<head>` and `<body>` and they are executed sequentially though network and threading issues can occur – consider the environment!

```html
<h1>Ready start</h1>
<script>
    alert("First Script Ran");
</script>
<h2>Running</h2>
<script>
    alert("Second Script Ran");
</script>
<h2>Keep running!</h2>
<script>
    alert("Third Script Ran");
</script>
<h1>Stop!</h1>
```
<script> Tag in the <head>

- Given top-down read (and execution) often script is found in the `<head>` of an (X)HTML document

```html
<!DOCTYPE html>
<html>
<head>
<title>JavaScript in the Head</title>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<script>
function alertTest() {
    alert("Danger! Danger! JavaScript Ahead");
}
</script>
</head>
<body>
<h2>Script in the Head</h2>
<hr>
<script>
alertTest();
</script>
</body>
</html>
```

Script masking and <noscript>

- Script Hiding using HTML and JavaScript comments

```html
<script>
/*
 * put your JavaScript here
 */
</script>
```

- Avoids printing script onscreen in non-script aware browsers

- `<noscript>` Element

  - Useful to provide alternative rendering in browsers that have script off or don't support script

```html`
<noscript>
<em>your browser does not support JavaScript or it is currently disabled.</em>
</noscript>
```

- The next example shows a great way to keep non-JavaScript aware users out of your site

```html
<!DOCTYPE html>
<head>
<title>JavaScript Masked with noscript Too!</title>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
</head>
<body>
<script>
<!--
document.write("Congratulations! If you see this you have JavaScript.");
//-->
</script>
<noscript>
<h1 class="errorMsg">JavaScript required</h1>
<p>Read how to <a href="/errors/noscript.html">rectify this problem</a>.</p>
</noscript>
</body>
</html>
```
Meta Refresh Trick with `<noscript>`

- Change the `<head>` to contain a meta refresh to automatically redirect the user to an error page if the script is off.
- Copy this into every page into your site and you can improve the chances users have script on.

```html
<!DOCTYPE html>
<html>
<head>
<title>Needs JavaScript</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<noscript>
<meta http-equiv="refresh" content="0;URL=/errors/noscript.html">
</noscript>
</head>

- Downsides
  - Consider non-script aware bots
  - Likely won't validate
```

Script Hiding Notes

- Markup aficionados are concerned about script hiding using HTML comments. See [http://www.w3.org/TR/xhtml1/#C_4](http://www.w3.org/TR/xhtml1/#C_4)
- If you care about this don't do

```html
<script>
</![CDATA[

document.write("Congratulations! You have JavaScript.");
]]>
</script>
```
- Instead try

```html
<script>
/* <![CDATA[ */

document.write("Congratulations! You have JavaScript.");
/* ]]> */
</script>
```

Event Handlers

- [XHTML](http://www.w3.org/TR/xhtml1/) defines a set of event handler attributes related to JavaScript events such as `onclick`, `onmouseover`, etc. which you can bind JavaScript statements to.

```html
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Events</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
</head>
<body>
<form action="#" method="get">
<div id="formfields">
<input type="button" value="press me"
onclick="alert('You pressed my button!');">
</div>
</form>
<p><a href="http://www.yahoo.com"
onmouseover="alert('hi');">
Yahoo!</a></p>
</body>
</html>
```
Linked Scripts

- Like linked style sheets you can store JavaScript code in a separate file and reference it
  - Use a .js file
  - Contains only JavaScript
  - Store these files like images in a common directory in your site (e.g. /scripts)
  - Linked scripts can be cached and “clean up” (X)HTML documents
  - Linked scripts can have problems in certain network or browser situations

Linked Script Example

```html
<!DOCTYPE html>
<head>
<title>Linked Script</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script src="danger.js"></script>
</head>
<body>
<form action="#" method="get" id="form1">
<div id="formfields">
<input type="button" name="button1" id="button1" value="press me" onclick="alertTest();">
</div>
</form>
</body>
</html>
```

Linked Script Example Contd.

- In file danger.js you would have simply have code like

```javascript
function alertTest() {
    alert("Danger! Danger!");
}
```
Fully Decoupled Script Example 1

```html
<!DOCTYPE html>
<html>
<head>
<title>Linked Script</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script src="danger.js"></script>
</head>
<body>
<form action="#" method="get">
<input type="button" id="button1" value="press me">
</form>
<script src="events.js"></script>
</body>
</html>
```

• In the file events.js we have:
```javascript
document.getElementById('button1').onclick = function () {
    alertTest();
};
```

Multiple Linked <script> Tags

• Commonly developers reference multiple JS files separately:
```html
<script src="lib1.js"></script>
<script src="lib2.js"></script>
```

• It is questionable the value of this practice:
  – Round trip times
  – Load order concerns
  – Sharing same name space

• Idea:
```html
<script src="alllibs.js"></script>
```
  – Code organization and caching is cited instead but analysis of both claims is specious at best
  – “Code for yourself – prep for delivery”

Fully Decoupled Script Example 2

```html
<!DOCTYPE html>
<html>
<head>
<title>Linked Script</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script src="combined.js"></script>
</head>
<body>
<form action="#" method="get">
<input type="button" id="button1" value="press me">
</form>
</body>
</html>
```

• Since it would be better to combine the two scripts together we need to address load order issues.
Fully Decoupled Script Example 2 Contd.

```javascript
/*combined.js*/
function alertTest() {
    alert("Danger! Danger!");
}
window.onload = function () {
    document.getElementById('button1').onclick = function () { alertTest(); }
}
```

JavaScript Pseudo-URLs

- You can use the JavaScript pseudo-URL to trigger a script statement.
  - For example
    ```html
    <a href="javascript: alert('hi');">Click me</a>
    ```
- You can also type such a URL directly in the browser's location box, for example
  ```javascript
  javascript:alert(5*5)
  ```
- Be aware that JavaScript pseudo-URLs do not degrade well in non-JavaScript aware situations.
  - Question: What happens with script off here?

Other JavaScript Inclusion Methods

- There are a few other ways (some sneaky) to include JavaScript in a Web page, the most notable being the JavaScript entity supported by Netscape 4.x generation browsers.
  - This method uses a standard HTML character entity in a macro style manner
    ```html
    &{script};
    ```
- You shouldn't use any other forms of script inclusion since they are likely not supported or may have other concerns.
Defensive Coding 101

• If our script is going to play nicely on the Web we must be very defensive
  – Don’t bash and watch out for being bashed!

• Encapsulate code and assume the worst is a good idea

• Potential Concerns
  – Variable and Function name conflicts
  – Load order and network concerns
  – Catastrophic errors thrown without handling
  – Event rebinding
  – Browser quirks!

• Variable Collision
  – Code in browser based JavaScript shares the same namespace.
    • If you define a variable say `num` and later on a script goes and does the same then `num` will overwrite yours. The reverse can also happen.
  – You must avoid global variables that may be bashed
    ```javascript
    var num = 5; // bad idea!
    ```
  – Stemming
    ```javascript
    var JSREF_num = 5;
    ```
  – Object Wrapper
    ```javascript
    var JSREF = { 
    JSREF.num = 5;
    ```

• Event Collision
  – Depending on how events are added it is also possible to overwrite and existing event handler.
    ```javascript
    window.onload = function () {
    /* going to bash an existing one */
    }
    ```

• Safe Loader Code
  ```javascript
  var JSREF = { 
  addLoadEvent = function(newFunction) { 
  var oldFunction = window.onload;
  if (typeof window.onload != "function") { 
    window.onload = newFunction;
  } else {
    window.onload = function () {
    if (oldFunction) {
      oldFunction();
    }
    newFunction();
    };
  ```

```
Safe and Sane (and Wordy)

```javascript
/* safecombined.js */
var JSREF = {};
JSREF.addLoadEvent =
    function(newFunction) { /* see other slide */
        JSREF.alertTest = function () { alert("Danger! Danger!");
    }
JSREF.bindEvents = function () {
    (document.getElementById('button1').onclick=
    function () { JSREF.alertTest(); } );
    /* still trouble above */
    JSREF.addLoadEvent(JSREF.bindEvents);
```

History of JavaScript

- JavaScript first introduced in 1995
  - Invented by Netscape
  - Originally called LiveScript
  - Renamed JavaScript when first beta in Netscape 2
  - Not really related to Java
  - The ideas of DHTML and Ajax add even more confusion
  - Used both client and server-side and within and outside of browsers
- Microsoft supports clone of JavaScript called JScript
  - First introduced in Internet Explorer 3
- Standards oriented JavaScript called ECMAScript

<table>
<thead>
<tr>
<th>Browser Version</th>
<th>JavaScript Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netscape 2.x</td>
<td>1.0</td>
</tr>
<tr>
<td>Netscape 3.x</td>
<td>1.1</td>
</tr>
<tr>
<td>Netscape 4.0-4.05</td>
<td>1.2</td>
</tr>
<tr>
<td>Netscape 4.06-4.7x</td>
<td>1.3</td>
</tr>
<tr>
<td>Firefox 1.5</td>
<td>1.4</td>
</tr>
<tr>
<td>Firefox 2.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Firefox 3.0</td>
<td>1.7</td>
</tr>
<tr>
<td>Firefox 3.5</td>
<td>1.8</td>
</tr>
<tr>
<td>Internet Explorer 4.x</td>
<td>1.0</td>
</tr>
<tr>
<td>Internet Explorer 5.x</td>
<td>1.5</td>
</tr>
<tr>
<td>Internet Explorer 6.x</td>
<td>1.8</td>
</tr>
<tr>
<td>Internet Explorer 7.x</td>
<td>1.9</td>
</tr>
<tr>
<td>Internet Explorer 8.x</td>
<td>2.0</td>
</tr>
<tr>
<td>Internet Explorer 8.5</td>
<td>2.1</td>
</tr>
<tr>
<td>Internet Explorer 9.x</td>
<td>2.5</td>
</tr>
<tr>
<td>Internet Explorer 10.x</td>
<td>3.0</td>
</tr>
</tbody>
</table>
JavaScript Applications

- Common uses of JavaScript include:
  - Form validation
  - Page embellishments and special effects
  - Navigation systems
  - Basic Math calculations
  - Dynamic content manipulation
- Really isn’t any particular limit to what it can do, it’s a regular PL
  - Demos
- The interplay between JavaScript other Web techs can produce powerful results

JavaScript, (X)HTML, and CSS Link

- JavaScript very much relies on markup and CSS in browsers, in fact it manipulates objects that are created by the correct use of tags and style properties
- For example, the document object contains objects and collections corresponding to many of the tags in the (X)HTML document.
  - document.forms[], document.images[], document.links[], etc.
  - We can always jump directly to the object using something like document.getElementById() under a DOM compliant browser

Simple Example 1 of Interplay

```html
<!DOCTYPE html>
<html>
<head>
  <title>Simple DOM Example</title>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <script>
    function showField() {
      alert(document.form1.field1.value);
    }
  </script>
</head>
<body>
  <form action="#" method="get" id="form1" name="form1">
    <input type="text" name="field1" id="field1">
    <input type="button" name="button1" id="button1" value="press me" onclick="showField();">
  </form>
</body>
</html>
```
Simple Example 2 of Interplay

```html
<!DOCTYPE html>
<html>
<head>
<title>Simple DOM Example #2</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
</head>
<body>
<p id="p1" style="color: red;">Hello there</p>
<form>
<input type="button" value="left" onclick="document.getElementById('p1').align='left';">
<input type="button" value="center" onclick="document.getElementById('p1').align='center';">
<input type="button" value="right" onclick="document.getElementById('p1').align='right';">
<br>
<br>
<input type="button" value="red" onclick="document.getElementById('p1').style.color='red';">
<input type="button" value="blue" onclick="document.getElementById('p1').style.color='blue';">
<br>
<br>
<input type="button" value="Big" onclick="document.getElementById('p1').style.fontSize='xx-large';">
<input type="button" value="Small" onclick="document.getElementById('p1').style.fontSize='xx-small';">
</form>
</body>
</html>
```

Conclusions

- JavaScript is a full blown Web programming language
- It is not related to Java in more than name
- It intersects with XHTML through `<script>` linked scripts (.js files), and attributes for event handling (onclick)
- It has evolved over time
- It has many browser compatibility issues to worry about