

A Crash Course in Web Design

Zaeem Maqsood 14.3.2000

CONTENTS	PAGE
1 - Getting Started in Web design	2
2 - HTML Building Blocks	5
3 - Design Principles	8
4 - Interactive Interfaces	12

1 - Getting Started in Web Design

Part 1 will give a brief overview of the field of web design and provide resources for the serious developer. We will start by creating our first, very simple web page - "firstpage.html", then we will introduce the terminology and ideas in current use and learn how to use the Web as an effective tool.

The following is the outline for your first web page. Save it as a file called "firstpage.html" and open it in a browser. Then we will discuss what it all means.

Example 1 - First Web Page

```
<HTML>
  <HEAD>
    <TITLE>      First Page
  </TITLE>
  </HEAD>
  <BODY>
    This is my first web page!!
    Pretty soon, this page will be brimming with cool stuff!
  </BODY>
</HTML>
```

firstpage.html

All web pages, regardless of content, have the same basic framework, as above.

There are a couple of rules to follow:

1. Each start tag has a corresponding end tag (e.g. <BODY> and </BODY>).
2. The web page follows a 'nested' structure, with start tags and end tags following each other immediately.

Even in the simple exercise above, we have encountered many new ideas. We should not continue without understanding what we have just created so below is a brief discussion of some of the most important aspects of web design.

- Architecture

{KEYWORDS: Browser, WebServers, Dynamic Sites, Platforms[Netscape,IE,Windows, Linux, Mac, WebTV, Palmtops]}

The Internet is built up of **web browsers and web servers**. Browsers are used to view information and servers are used to publish it. The information can be as simple as plain writing, or as exciting as **live video broadcasts, animation, music and multi-player gaming**. Examples of browsers are Netscape and Internet Explorer and the internet can be accessed from most computers, ranging from your home PC to the new DigitalTV and handheld devices.

- Languages & Databases

{KEYWORDS: HTML, JavaScript, CSS/DHTML, XML, Java, ASP, Perl, PHP, SQL Server, Access2000, MySQL, mSQL}

HTML (HyperText Markup Language) is the founding language of the Web. It is very simple and popular but it is not very powerful. All other languages are built 'on-top-of' HTML. Other major languages include **JavaScript, Java, ASP, Perl and PHP**. All are more complex than HTML but allow us to make much more powerful use of the Internet, for example, connecting to databases like Access or MySQL.

- Multimedia

{KEYWORDS: .gif, animated gifs, jpeg, .png, shockwave, quicktime, realaudio/video, mpeg, streaming multimedia, plug-ins}

Images, sound, animation and video (multimedia) come in many different formats on the Internet. Some formats require special software to play the files, whereas others such as the image formats '**gif**' and '**jpeg**' are viewable in browsers without special software.

- Services

{KEYWORDS: Web, Email, Telnet, FTP, Usenet, real-time multi-user (games, chat), WAP}

It is important to remember that the Internet is made up of different 'services'. One of them is the World Wide Web (the Web), but others include **email and Usenet**. Email can be thought of as a letter sent electronically to someone and Usenet news groups are like common message boards that anyone can see and add to. They all have their specific uses and are usually used together.

- Tools

{KEYWORDS: HotDogPro, FrontPage, PaintShopPro, Photoshop, Visual Interdev}

Tools are very useful for web design, but they are most powerful when the user has a fair bit of experience with '**raw**' **designing** without the editing tools. Some of the tools are for graphics and some are for programming. The web design novice will not be able to do without graphics tools such as **Paint Shop Pro and Photoshop**, but will certainly be better off in the long run learning HTML with a regular, simple word processor, like Notepad for the time being.

- Design and Planning Principles

{KEYWORDS: Functionality, Interface (Content, Style, Structure), Paper-based, Notation, Open Source}

Every good web designer needs to ask some simple questions **before they start development**. What the web site is for, what it is supposed to do and what it should look like are some of the most important and most frequently overlooked. Essential design and project planning principles will be explained. They will become one of your most important tools since they will save a lot of time and hassle - **measure twice, cut once!**

You now have the conceptual underpinnings of the most important aspects of web design. What is needed is a set of resources you can use to both help apply you're new understanding and help you learn even more. Detailed below are some such resources and coupled with your handle on the new terminology, you should at least be able to know where to start, upon embarking on web design.

- Resources

Web

- <http://www.webmonkey.com>
Great tutorials an all round starting point
- <http://www.php.net>
The official php website
- <http://www.mysql.com>
Great free database. Very fast and robust.
- <http://www.javasoft.com>
The first place for anything Java
- <http://www.linux.org>
The official Linux starting point
- <http://www.opensource.org>
Open Source evangelising. And why not?
- <http://www.websitesthatsuck.com>
Web design tutorial – cult status surely?
- <http://www.microsoft.com>
Tutorials, downloads, reference for ASP
- <http://www.download.com>
A huge site, with masses of free resources for designers. Will take years to try it all.
- <http://www.zdnet.com>
Online magazine to help keep up to date with developments in the field.
- <http://www.thestandard.com>
E-business news straight from Silicon Valley.
- <http://www.ebay.com>
The biggest and probably the best auction community out there.

Usenet

- <news://ba.php>
The most popular PHP news group
- <news://uk.local.london.info>
Tons of information and a chance to give back.
- <news://alt.activeserverpages>
Heavily trafficked news group for ASP.

Books:

- | | |
|------------------------|---------------------|
| Webmasters Handbook | John Merlin Fischer |
| JavaScript For Dummies | Emily A. VanderVeer |
| Java In A Nutshell | David Flanagan |
| HotSites | Roger Walton |
| HotWired Style | Jeffrey Veen |

Magazines:

- | | |
|-------------------|--------------------------------------|
| Wired | Monthly - cutting edge, cult status. |
| WebTechniques | Monthly - in depth treatment |
| Internet Business | Monthly – fairly competent |
| Revolution! | Monthly – new media |

2 - HTML Building Blocks

By the end of Part 1, we developed our first web page and learnt the important concepts behind the Internet and how to use the Net as a value added tool. Part 2 will take our HTML knowledge further. We will learn the building blocks (or elements) of HTML, which will allow us, by the end of Part 2, to build some pretty impressive web pages. Getting to grips with HTML is 50% of becoming a real web designer, so you can't practice enough! Part 2 can be used later on as an HTML reference.

You can copy and paste the following examples into separate files (with the filenames given). Like the previous example, open this up in your browser. You now have a set of web pages that make use of hyperlinks, frames, tables, meta-tags, fonts and colors - in other words the building blocks of HTML. Getting this example up and running is not cheating. Rather it is showing you how to use the elements of HTML in real code and is a method learning ideally suited to the web. You will be doing a lot of peeking at the source code of other peoples web pages - an entirely legitimate activity (but still quite geeky).

Example 2 Your first web page - with Elements

```
<HTML>
  <HEAD>
    <TITLE>
      First Page
    </TITLE>
  </HEAD>
  <BODY BGCOLOR=RED TEXT=DARKBLUE>

    This is my <b>first</b> web page!! <BR>
    Pretty soon, this page will be <font size=5 face=" Verdana, Arial, Helvetica, sans-serif" color=blue>
    brimming</font> with cool stuff!<BR>
    <A HREF="secondpage.html">Second WebPage</a><BR>
    <FORM >
    <CENTER>
    <INPUT TYPE=TEXT VALUE="HELLO WORLD" SIZE=25><BR>
    </CENTER>
    </FORM>
    <TABLE BORDER=2 ALIGN=RIGHT>
      <TH>FRUIT</TH><TH>COLOR</TH>
      <TR><TD>APPLE</TD><TD>RED</TD></TR>
      <TR><TD>PEAR</TD><TD>GREEN</TD></TR>
    </TABLE>
  </BODY>
</HTML>
```

firstpage.html

Example 3 Your second web page - with MetaTags

```
<HTML>
  <HEAD>
    <TITLE>
      Second Page
    </TITLE>
    <META HTTP-EQUIV="Refresh" CONTENT="3;URL=secondpage.html">
  </HEAD>
  <BODY BGCOLOR=BLUE TEXT=WHITE VLINK=YELLOW ALINK=GREEN>
    This is my Second web page!!
    It uses the Reload MetaTag to reload itself<BR>
    <IMG SRC="helloworld.gif"><BR>
    <A HREF=firstpage.html>First page</A>
  </BODY>
</HTML>
```

secondpage.html

Example 4 Your third web page - with Frames

```
<FRAMESET ROWS="76,*" FRAMEBORDER=no FRAMESPACING=no BORDER=1 RESIZE="yes">
  <FRAME SRC="firstpage.html">
  <FRAME SRC="secondpage.html" >
</FRAMESET>
```

framepage.html

- Body Attributes

<body bgcolor=?>	Sets the background color, using name or hex value
<body text=?>	Sets the text color, using name or hex value
<body link=?>	Sets the color of links, using name or hex value
<body vlink=?>	Sets the color of followed links, using name or hex value
<body alink=?>	Sets the color of links on click

- Font

<pre></pre>	Creates preformatted text
<h1></h1>	Creates the largest headline
<h6></h6>	Creates the smallest headline
	Creates bold text
<i></i>	Creates italic text
<tt></tt>	Creates teletype, or typewriter-style text
<cite></cite>	Creates a citation, usually italic
	Emphasizes a word (with italic or bold)
	Emphasizes a word (with italic or bold)
	Sets size of font, from 1 to 7
	Sets font color, using name or hex value
	Sets the style of the font.

- Links

	Creates a hyperlink
	Creates a mailto link
	Creates a target location within a document
	Links to that target location from elsewhere in the document

- Form Components

<form name="NAME"></form>	Creates all forms
<select multiple name="NAME" size=?></select> Size sets the number of menu items visible before you need to scroll.	Creates a scrolling menu.
<option> the scrolling menu	Sets off each menu item in
<select name="NAME"></select> the pulltime menu	Creates a pulldown menu
<option>	Sets off each menu item in
<textarea name="NAME" cols=40 rows=8></textarea> Columns set the width; rows set the height.	Creates a text box area.
<input type="checkbox" name="NAME"> follows tag.	Creates a checkbox. Text
<input type="radio" name="NAME" value="x"> follows tag	Creates a radio button. Text
<input type="text" name="foo" size=20> area. Size sets length, in characters.	Creates a one-line text
<input type="submit" value="NAME">	Creates a Submit button
<input type="image" border=0 name="NAME" src="name.gif"> using an image	Creates a Submit button
<input type="reset">	Creates a Reset button

- Frames

<frameset rows="#,#" ></frameset>	Specifies the screen space allocated to each frame
<frameset frameborder=? ></frameset>	Specifies if the frames inner border - thin black line - is visible or not. 0 or "no" for invisible and 1 or "yes" for visible
<frameset framespacing=? </frameset>	
<frameset border=# ></frameset>	Specifies the width of the frames outer border - grey line. If it is set to 0, then it is invisible.
<frameset resize="yes"></frameset>	Allows frames to resize.
<frame src="URL">	Specifies which HTML document should be displayed
<frame name="name">	Names the frame, or region, so it may be targeted by other frames
<frame marginwidth=#>	Defines the left and right margins for the frame; must be equal to or greater than 1
<frame marginheight=#>	Defines the top and bottom margins for the frame; must be equal to or greater than 1
<frame scrolling=VALUE>	Sets whether the frame has a scrollbar; value may equal "yes," "no," or "auto." The default, as in ordinary documents, is auto.
<frame noresize>	Prevents the user from resizing a frame

- Tables

<table width=80% border=0></table>	Creates a table
<tr></tr>	Sets off each row in a table
<td width=25%></td>	Sets off each cell in a row
<th></th>	Sets off the table header (a cell with bold, centered text)

- Images & animated gifs

	Adds an image
	Aligns an image: left, right, center; bottom, top, middle
	Sets size of border around an image
<hr>	Inserts a horizontal rule
<hr size=?>	Sets size (height) of rule
<hr width=?>	Sets width of rule, in percentage or absolute value
<hr noshade>	Creates a rule without a shadow

- Metatags

<META HTTP-EQUIV="expires" CONTENT="Wed, 26 Feb 1997 08:21:57 GMT">
 This tells the browser the date and time when the document will be considered "expired." If a user is using Netscape Navigator, a request for a document whose time has "expired" will initiate a new network request for the document. An illegal Expires date such as "0" is interpreted by the browser as "immediately." Dates must be in the RFC850 format, (GMT format):

<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
 This is another way to control browser caching. To use this tag, the value must be "no-cache". When this is included in a document, it prevents Netscape Navigator from caching a page locally.

<META HTTP-EQUIV="Refresh" CONTENT="0;URL=http://www.newurl.com">
 This tag specifies the time in seconds before the Web browser reloads the document automatically. Alternatively, it can specify a different URL for the browser to load.

<META HTTP-EQUIV="Set-Cookie" CONTENT="cookievalue=xxx;expires=Wednesday, 21-Oct-98 16:14:21 GMT; path=/">
 This is one method of setting a "cookie" in the user's Web browser. If you use an expiration date, the cookie is considered permanent and will be saved to disk (until it expires), otherwise it will be considered valid only for the current session and will be erased upon closing the Web browser.

<META HTTP-EQUIV="Window-target" CONTENT="_top">
 This one specifies the "named window" of the current page, and can be used to prevent a page from appearing inside another framed page. Usually this means that the Web browser will force the page to go to the top frameset.

<META NAME="keywords" CONTENT="life, universe, mankind, plants, relationships, the meaning of life, science">

You use the keywords attribute to tell the search engines which keywords to use.

<META NAME="description" CONTENT="This page is about the meaning of life, the universe, mankind and plants.">

Using the META description attribute, you add your own description for your page

3 - Design Principles

We now know how to put the Internet to work for us (Part 1) and how to establish a simple, yet powerful presence on the Web (Part 2). In a sense we have learnt an important first lesson in web design - *creating a web page is easy!*. But take a look at the web pages we have created - they wouldn't really cut it out on the internet, in fact they are pretty ghastly and practice will make us only *so much better* as developers. What we need to help us produce professional looking web sites is a set of design principles for the internet, which is the topic of Part 3. Time to learn the important second lesson in web design - *creating a web page may be easy, but developing a professional web site isn't*.

It would be impossible to present web design principles in any definitive sense. There are many approaches to web design and many ways to analyse and present those same approaches. One reason is because different criteria matter more to different people. For example, an analysis may concentrate more on structure and another on metaphor. Another reason is because the Web has attracted developers from so many different backgrounds, like programmers, designers, animators, advertisers & regular entrepreneurs. But the main reason there are so many different approaches to web design is because the field is still so young – not much older than five years. The multiplicity of design criteria should be seen as a strength, reflecting the diverse appeal the Internet offers. I will not present one approach to web design, or even one approach to analysing web design. Rather, I'll survey a few of the more popular approaches and allow you to decide which is more appropriate. The approaches outline below are proto-typical. There are many 'pure' examples of each, but successful web-site design is often a balanced fusion of elements of all.

Regardless of which design model you chose, there are a number of considerations that need to be addressed before setting out to design you site.

The overall framework within which to consider should be the following:

- **Users;** Who are they? What do they know? What do they want to know? What is their computing environment?
- **Function;** Very dependent on Users. What is your site about? What is it exactly that your site will do? Will it statically present information or seek to respond to the user in some way?
- **Content;** Very dependant on Function. What will your site display? Will it be text or images or multimedia?
- **Style;** Very dependent on Content. A useful metaphor for thinking about style is of the library and the gallery. The library seeks to present a mass of data in an orderly and accessible way. A gallery displays media according to a visually appealing theme.
- **Structure;** Very dependent on Style. Will the site be deep and narrow or shallow and wide? What kind of navigation will you use – frames, toolbars? Will the user be able to jump from one page to every other? Will there be links to other sites, or even intra-page links?

You can see that each element of the framework is dependent on the preceding one, but in real development we cannot assume we will have full knowledge of the preceding elements. Therefore, a prototyping approach is generally adopted – go as far as you can with what knowledge you have, use what you develop to elicit further knowledge and go back to refine your previous understanding and so on. For

example, we may have a rough idea of our intended users, but it is only when we create an initial web site and get the users to play with it do we realise where our original understanding was perhaps slightly off the mark. In this way, we update our model of the user which effects how the site will function, what the content will be etc.

'Hacker' Design - usable structure

By 'hacker' design, I refer to a set of rules of thumb, rather than clear principles and was used by the original creators of the web. 'Hacker' web developers have a different and coherent enough set of guidelines to other developers that they deserve to have it presented.

Characteristics:

- ease of use/learning design principles
- flexibility to adapt to rapid changes in technology
- highlight web functionality
- accommodating different types of users
- adopted the freewheeling spirit of the early web days

Proponents:

- Vincent Flanders <http://www.websitesthatsuck.com>
- John Merlin Fisher Webmaster's Handbook

Analytical Approach:

Has a very specific view of the user. The user is an average Joe, surfing at home on a mid-range PC. Joe knows what he is after (roughly), but maybe not how to get it. It is the developer's job to be as useful to Joe as possible. This is a very utilitarian approach, but it sees the imbalance of function and limiting a sites utility, hence generally dislikes the typical 'student home page' and overproduced 'arty' sites. In their view, it doesn't take a genius, or a highly trained graphic designer to produce a good web site – all it takes is consideration of the user, good knowledge of HTML and a bit of common sense.

The 'hacker' designer highlights the following elements of web site organization:

- Balanced document trees and navigational aid
- Links to related sites
- Keeping response times low
- Encouraging the user to revisit.
- Developing a consistent style
- Developing for the lowest common denominator (640*480 screen, 256 colors, 33.3K modem, no unique browser features)
- Avoid annoying designs, like blinking text & too many colors.

Advantages:

- Very quick to learn and apply.
- Services the needs of most developers and web users.
- Quickly incorporates advances in web technology.

Disadvantages

- Does not acknowledge the importance of graphic design essentials like font, colors, spacing & creativity generally.
- Does not offer a structured enough approach to design.
- Does not have an in-depth enough analysis of the system within which the website operates, especially for web applications.
- Does not accommodate the needs of artists, using the web itself as the subject of art.
- View of the web and its users not mature enough for today's market realities

Graphic Design - appealing style

Characteristics

- Concentration on graphical building blocks like font, images, spacing, balance and impact
- Frequent use of professional tools such as Photoshop and QuarkExpress and Shockwave and technologies such as Dynamic HTML and Cascading Style Sheets.
- Fusion of artistic creativity with structural design
- Use of the web as another medium to display or create art

Proponents

- Wired

Analytical Approach

Whereas the 'hacker' designer's main priority was utility to the user, the graphic designer emphasises the visual experience of the web site. Often, the site requires special capabilities to fully appreciate the graphic designers work, such special plug-ins, high bandwidth, high resolution screens and up-to-date browsers. It is an over-simplification with an element of truth that this is the graphic designer's view of the user and the environment – a view that is increasingly becoming a reality. The graphic designer's analysis would highlight the following characteristics:

- The whole visual experience, from images, font and color to clarity and simplicity.
- Originality and theme of the design.
- Attention paid to the users 'eye' and flow of attention

Advantages

- High interest and novelty factor – the holy grail, to many, of web design.
- Highly professional appearance – very important for the corporate web presence.
- Web content taken in new directions

Disadvantages

- Usability and utility low priority

Web Application Design - powerful content

Characteristics

- Formal analysis undertaken
- Dynamic content
- Interactive interfaces
- Often in teams and on intra-nets

Proponents

- Yahoo! And other search engines.
- <http://www.zdnet.com>
- Human Computer Interaction and Human Factors Engineers

Analytical Approach

Web application designers give emphasis to functionality, rather than usability or visual experience. This is achieved by generating dynamic content in an attempt to 'personalise' the user's experience, or give it a real-time effect. Functionality is also achieved by adding interactivity with the user to the interface. Often, functionality entails server side databases and programs, (CGI) or client side programming (JavaScript, Java). The user is seen as demanding in both their requirements and their abilities. They normally have definite information requirements and the resources to access the information, but may not be the most web-savvy of users. A web app designer's analysis would highlight the following characteristics:

- Design process and methodology (hidden from the user)
- Functionality of the site, given by dynamism and interactivity
- Information architecture
- Performance and integrity of the system

Advantages

- Professional process allows for well integrated teamwork and hence commercialisation of web design.
- Very powerful systems running off the web, offering alternative strategies for software engineering.
- Highly usable web sites, catering to most users

Disadvantages

- Overly formal methodology for building web sites – not suited to most developers.
- Lengthy design and implementation process.
- Sometimes suffers from functionality over form, leading to visually dry sites.

4 - Interactive Interfaces

JavaScript Overview

In Part 3 we discussed different approaches to web design. Their emphasis of their methods ranged from organising structure to creating visual appeal to generating dynamic content. However their goals were common – to understand and service the user. HTML can take each approach much of the way, but sometimes the designer may need to present an interface with a bit more functionality. JavaScript was developed for precisely this job – not as a substitute but rather a compliment to HTML. JavaScript is a simple, object oriented, browser embedded scripting language, developed by Netscape a few years ago. Each of these terms needs explaining, but first it should be noted that JavaScript is not related to Java in any real way. In fact, Netscape renamed LiveScript to JavaScript when they realised how popular Java, (developed by Sun Microsystems), was becoming.

Architecture

JavaScript is simple because although it is a lot more functional than HTML, it is not a fully functioning programming language like C++, Visual Basic or Java. Its syntax allows the developer to create standard programming structures like functions, variables, arrays, loops and conditions and importantly, to easily integrate these structures with the HTML elements we have already learnt about. It achieves this integration because it is an embedded language – it sits within and is triggered by the actual HTML as follows:

Example 5 Your fourth web page - with JavaScript

```
<html>
  <head>
    <title> My JavaScript Page </title>
    <script language="JavaScript">
      <!--
        function alertbox(){
          window.name="JavaScript"
          window.alert(window.name)
        }
      //-->
    </script>
  </head>
  <body onLoad=alertbox()>
    This page tests my JavaScript
  </body>
</html>
```

jsfirstpage.html

Try the example out. It introduces some important ideas in JavaScript. You can see how the JavaScript is *embedded* with the `<script language="JavaScript">` and `</script>` tags and how it is *hidden* from the user with `<!--` and `//-->` tags. We can also see how functions are created with the *function* `alertbox(){` and closing `}` statements and how our functions can use JavaScript *methods* such as `alert()` and refer to *properties* like `name`. This function needs to be called from somewhere and in this example it is called from the HTML `<body>` tag using what is called an *event handler* – in this case, `onLoad`.

Most HTML tags have event handlers, which are not part of the HTML language as such. Rather they are an innovation by companies such as Netscape and Microsoft (who *just* happen to sell the two most popular browsers) to allow their scripting languages (JavaScript and VBScript) to interact with the HTML elements. When something happens to the HTML element the tag refers to, a particular *event* is fired.

How the event is *handled* is up to us – in our case we called a function called `alertbox`.

`OnLoad` is one event handler for `<body>` tag. It is fired when the web page has finished loading. Another is `onUnload`, which, of course, is fired when the web page has finished unloading. If we changed `<body onLoad=alertbox()>` to `<body onUnload=alertbox()>`, what would happen? There are lots of event handlers – too many to list here, so having a JavaScript reference book around will be handy.

So we can see already that JavaScript affords us a far greater degree of interactivity with web pages than just HTML on its own. With HTML, all we really have for interaction are hyperlinks. In JavaScript we can interact with almost any HTML element, and how we do so will be explained...

Object Navigation

Objects in programming are similar to objects in real life. A glass of water has certain properties (it is full/empty, it is a clear/green/blue/red glass) and certain uses (you can drink from it, fill it up, drop it, examine it etc). A list of properties and uses describes objects – things – very well.

JavaScript is an object-oriented (OO) language. JavaScript treats the web page, and the elements within it, as *objects*.

The characteristics of the object are called *properties* and the uses are called *methods* in JavaScript, and every object has certain properties and methods. An example is the `alert()` method used in the previous example. We say this method *belongs* to the window object. Methods may take *parameters* like a string or a number. The `alert()` method can take any of these parameters. The window object has other methods like `close()`, `blur()` and `focus()` that behave differently and take their own parameters. Try replacing `alert()` with them and see what happens.

Object properties like `window.name` can have their values set and read, whilst other properties like `window.self` and `window.parent` actually refer to other objects, upon which we can call methods and whose properties we can read & write. We say the object's `window.self` and `window.parent` *belong* to the window object.

Properties like `window.length` are read only. It is a bit awkward, but you will have to keep looking up object methods and properties in reference books until you remember them – there aren't enough to scare you off though.

The window object is part of what we called JavaScript's object hierarchy. This object tree describes how we can refer to objects, that is, what objects belongs to which other objects. Understanding this tree will help us a lot when figuring out how to refer to a particular object.

JavaScript Object Model

Object

window
parent, frame, self, top
location
history
document
 form
 button
 checkbox
 fileUpload
 hidden
 image
 password
 radio
 reset
 select
 submit
 text
 textarea
 hyperlink
 anchor
 plugin
 applet

Date
Math
string
navigator
this

JavaScript syntax

window [optional]
parent, frame, self, top
location
history
document
document.someForm
document.someForm.someButton
document.someForm.someCheckbox
document.someForm.somefileUpload
document.someForm.someHidden
document.someForm.someImage
document.someForm.somePassword
document.someForm.someRadio
document.someForm.someReset
document.someForm.someSelect
document.someForm.someSubmit
document.someForm.someText
document.someForm.someTextArea
document.someHyperlink
document.someAnchor
document.somePlugin
document.someapplet

Date
Math
someString
navigator
this

The example below shows you how to use the syntax associated with the Object Model:

Example 6 Your fourth web page - showing the JavaScript Object Model

```
<html>  
  <head>  
    <title> My JavaScript Page </title>  
    <script language="JavaScript">  
      <!--  
        function alertbox(){  
          window.name="JavaScript"  
          alert(document.forms["jsform"].elements["jsselect"].name)  
        }  
  
        function showVal(menu){  
          var menuValue = menu.options[menu.selectedIndex].value  
          alert(menuValue)  
        }  
      //-->  
    </script>  
  </head>  
  <body >  
    This page tests my JavaScript  
    <form name="jsform">  
      <select name="jsselect" onChange=showVal(this)>  
        <option value="I like">This select  
        <option value="to be">list makes  
        <option value="clicked ;-)">a sentence  
      </select>  
  
      <input type=button name="showName" value="show the name of the select list" onClick=alertbox()>  
    </form>  
  </body>  
</html>  
jsfirstpage.html
```

We have created a form called "jsform" with two HTML elements – "jsselect" and "showName".

The select list fires an onChange event when an item is selected, which calls our JavaScript function showVal() *and* passes a reference, of the list element, to the function. The showVal() function call uses the 'this' keyword, which is short for saying document.forms["jsform"].elements["jsselect"] . Of course, a reference to this select list from any other place on the web page would have to use the fully qualified object reference.

The select list element is now seen as a JavaScript object (called "menu" by us, in the showVal() function, for short) and has the property selectedIndex, which holds the actual list item we selected. Menu.options[...] is also a property, but it is a property that holds a list of objects – the actual menu items in the select list. If we wanted to find the value of only the first item in the list, we could write: menu.options[0].value, and the second item's value would be: menu.options[1].value, but we have made our function intelligent enough to know which list item we actually selected – by returning the value of menu.selectedIndex. We have made our function *dynamic*, responding appropriately to real-time user interaction.

When we click the showName button, we fire the buttons onClick event, which is handled by the alertbox() function. All this function does is refer to (and displays) the name property of the jsselect object. It has to use a fully qualified object path. Note also that this function uses variables – containers for temporary data that we set up ourselves. How could we rewrite the function so it doesn't use variables?

Advanced: New Objects, Images and Beyond..

We will take a look at some fairly advanced JavaScript now. There are some nice features that, once mastered, will open up JavaScript as a highly functional programming language. Although advanced, the JavaScript is still not too difficult. An example will again serve to illustrate the key concepts, namely dynamic image manipulation, user-defined objects, multiple windows and programming structures like loops.

Example 7 Your fifth web page - Advanced JavaScript

```
<html>
  <head>
    <script language=JavaScript>
      <!--
        function over(){
          document.images[0].src = "sidenav_home_planttop.gif"
        }
        function out(){
          document.images[0].src = "invisigif.gif"
        }
        function writeimgsrc(){
          var j=0
          <!--***NOTE** :no whitespace in features property-->
          var x= window.open("jsnewwin.html", "newwin", "width=200,height=200")
          <!--***NOTE: need focus()***-->
          x.focus()
          for (var i=1; i<=13; i++){
            j=j+"\n"+i
          }
          x.document.forms[0].elements[0].value=myObj.imgname + ": \n"+j
        }
        function myImage(name){
          this.imgname = name
          <!--***NOTE:no '()' in writeimgsrc, else immediate execution***-->
          this.writeimgsrcloop = writeimgsrc
        }
      //-->
    </script>
  </head>
  <body link=white vlink=white onLoad="myObj=new myImage('the unlucky counting image!')">
    <form>
      <!--***NOTE:method name needs '()'***-->
      <input type=button value="write src" onClick=myObj.writeimgsrcloop() >
    </form>
    <a href="test.html" onMouseOver=over() onMouseOut=out()><img src=invisigif.gif width=200
height=200></a>
  </body>
</html>
jsadvanced.html
```

Example 8 Your eighth web page - the new window created by JavaScript

```
<html>
  <body bgcolor=orange>
    <form>
      <textarea></textarea>
      <center><input type=button value="close me" onClick=window.close() ></close>
    </form>
  </body>
</html>
jsnewwin.html
```

Where do we start? As always with JavaScript written as a set of functions, we start with the HTML and just follow what is going on line by line, looking out for event handlers.

Let's start with the <body> tag in jsadvanced.html. The onLoad event handler creates a user-defined object 'myObj' and this object is an object of type 'myImage'. But what is 'myImage'? Well, the definition of 'myImage' is given in the function 'myImage(name) {...}'. We call this type of function that allows us to create our own objects, 'constructor functions'. Now object-oriented means we can describe a theoretical object in terms of properties and methods and our 'myImage' object 'myObj' is no different. It has a property - 'imgname' and a method - 'writeimgsrcloop'. The 'this' keyword is just fiddly syntax to make sure the property and method is referring to 'this' current object, i.e. 'myObj'. 'imgname' is simply a variable that holds the value I passed in with the constructor function call, but it can hold anything and constructor functions don't need to have parameters. 'writeimgsrcloop' calls another function, 'writeimgsrc' to implement its functionality. The method 'writeimgsrcloop' and hence the function 'writeimgsrc' is called by the onClick event handler for the HTML button. 'writeimgsrc' dynamically creates another browser window with the HTML source file 'jsnewwin.html' and brings it into focus. Then it creates a string of numbers 0-13 and writes this, along with the value of myObj's imgname property.

The image manipulation is similar but implemented quite separately. The web page loads up the image 'invisigif.gif' with the 200*200 dimensions, (even though the image is really only 1 pixel square). When the mouse enters this 200*200 space the 'over()' function is called by the 'onMouseOver' event handler, and similarly, 'out()' when exiting. The 'over()' function swaps images by assigning a new image filename to the 'src' property of the JavaScript object images[0], (where images[7] would be the 8th image on the web page, say).