Reading Notes:

HTML Introduction

Switzerland was first to create “HTML", or "Hyper-Text Markup Language

-HTML provide a language for displaying text and pictures SIMULTANEOUSLY

-also devised the notion of "hyper-text", clickable words which could be used to summon OTHER digital resources, such as HTML pages, pictures, and sound files

What is HTML?

HTML (Hyper-Text Markup Language) is a display language, a text-only code which tells a web browser (such as Internet Explorer or Netscape Communicator) how to assemble a web page of text, pictures, and other multimedia content.

HTML pages are stored as ASCII (text-only) files on a web server. Pictures, stored as GIF or JPEG files, are also stored on a web server. When an HTML page is called up in a web browser, the HTML code tells the browser where on the web server the pictures are located, and where on the page those pictures are supposed to be displayed.

HTML formats text (and everything else, as well) using something called a TAG

<tagname>Content</tagname>

Not all tags have to open and close in this manner, surrounding content. Some tags simply insert formatting information into the flow of HTML text. For instance, in the BR (break) tag inserts a carriage return or line break into text without requiring a closing tag, as in the following example:

<p>Here is a paragraph of text.<br /> Notice how the line break occurs just there?</p>

XHTML (ie. HTML 5)

 Extensible Markup Language, or XML

 The primary motivation for creating XML was the need to send data over the Internet in a universal, structured format.

 Because this new version of HTML would not be based on earlier versions of HTML, but on XML, it was named Extensible Hypertext Markup Language, or XHTML.

-follows strict rules: case-sensitive, file extension should be html

XHTML Elements and Rules

At the very beginning of an XHTML document, you should place the following:

//indicates to the browser the version of XML used for the document’s code  
<?xml version="1.0" encoding="UTF-8"?>

//Next two lines make up the DTD (Document Type Definition) – what type of document this is   
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

xmlns stands for XML namespace, which helps the browser keep track of what all the custom XML tags mean, especially if the same name is used for different tags. A namespace lessens the confusion by providing the browser with information on the meaning of the tags in the document.

we are telling the browser that all tags contained within the <html> element belong to the XHTML namespace, as defined by the W3C and located at the given URL

 XHTML documents must be well-formed

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
....  
....  
</head>  
<body>  
....  
....  
</body>  
</html>

NOTE: Exception – frameset page

 XHTML documents must have a <title> element within the header section

<head>  
<title>Steve Rubin's Homepage</title>  
</head>

 Elements must nest and not overlap

<p>The following word is written in italics: <i>italics</i></p> //GOOD

<p>The following word is written in italics: <i>italics</p></i> //BAD p vs i

 All element and attribute names must be in lowercase

In XHTML, although you must write attribute names in lowercase, you can write attribute values in either upper or lowercase. For example, the following is valid in XHTML, even though I don't recommend it.  
  
<body bgcolor="YELLOW">

 All tags must close

In XHTML, you must terminate empty tags such as the <br> tag with a space followed by a forward slash ("/"). For Example: <br />  
  
This requirement to close empty tags also extends to all other tags, which would not normally close in HTML. For example:  
  
<img src="xyz.gif" />  
<meta name="keywords" content="Steve Rubin, web design, critical thinking" />  
<bgsound="xyz.wav" />

 All attributes must have values enclosed within quotes

//GOOD

<body bgcolor="blue">  
<img src="xyz.gif" width="200" height="300" />  
<map name="europe">

//BAD

<body bgcolor=blue>  
<img src=xyz.gif width=200 height=300>  
<map name=europe>

//MORE

<input type="checkbox" name="choice" value="europe" checked>  
  
The above example is incorrect in XHTML. In XHTML, it would be correctly coded as follows:  
  
<input type="checkbox" name="choice" value="europe" checked="checked" />

XHTML Validation

**The Main Reasons are:**

1. Validation helps to ensure your web page looks right, no matter what web browser is being used.
2. Search engines can view and understand the page content. Some errors may prevent your content from being read by the search engines.
3. Pages are more accessible to people with disabilities.

### HTML5:

HTML5 is a standard for structuring and presenting content on the World Wide Web. The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, andGoogle Gears.

**HTML5 basic format**

<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<title></title>  
</head>  
  
<body>  
  
<header>  
</header>  
  
<section>  
</section>  
  
<footer>  
</footer>  
  
</body>  
  
</html>

HTML5 has introduced new tags that actually have real meaning and purpose, much more semantic than simply using divs all over the place.

VIDEO: 10 Things every web dev should know

-semantics are better (more meaningful)

-additional features added

-backwards compatible (you can covert old HTML version to 5 quickly)

-UTF-8 (required for validation) //under meta

-no need for type attribute

Additions to markup:  
-new elements that have better readability (old still work)

-better semantics and structure

-will make IDE’s work better

JavaScript (power behind HTML5)

-geo-location – location awareness for web-apps (user’s coordinates)

-canvas api – 2D drawing surface you can add to your page and add graphics to

without any additional plugins

How does it work with web-pages?

-tons new features that allow pages to behave more like apps

-official language of HTML5

-JavaScript has good performance as it has been improved over the years

Web Storage

-5mbytes

-user can create concurrent threads on web application

3. Should we be using HTML5 now?

-If you have a lot of users using mobile devices that obtain your content then yes.

-If you have a lot of IE users then no (until people move on to new browsers)

4. Care about which browsers implement and API not which specifications it belongs to

-Focus on if browser supports API

NOTE: API – application programming interface (protocol used as an interface by software components to communicate with each other)

NOTE: DOM – Document Object Model; cross-platform and language-independent convenstion for representing and interacting with objects in HTML, XHTML, and XML documents

5. <video> tag

-allows you to put videos right into your page

-add additional controls while video is not playing (etc.. attributes)

-problem: what format of video to use? (encoding world)

.mp4 (iOS)

.ogv

.webm

OR also supply various source tags to allow browser to check what they can or can’t use

6. JSON – King of Content

-used as an alternative to XML to serialize data

-Java Script Object Notation

-takes an object written in JavaScript and turns it into a String

-good because it is more readable than XML and easier to parse

-2 Methods: Stringafy and Parse

-used by many WebServices that use this

-advantage – easy to use and convert / seralizable / unseralizable

-disadvantages: be careful with remote services to not get a malicious object

7. Make it easy on yourself with graceful degradation and feature detection

-largely deals with API

-feature detection – does geolocation exist? do webworkers exist?

-solution: get Modernizr (clean, consistent interface, detect CSS3 features)

8. <canvas>

-more than just for drawing

-alter video source (ie make it black and white)

-JavaScript grabs frames from video and throws it into a canvas then does editing

-use with double buffering techniques

-general purpose display

9. CSS3 supports new features

-CSS replaces common things done with JS

-ie. transition – opacity occurs gradually (fade effect)

-ie. transform – take an element and do something with it (rotate it, scale it)

10. XHTML is dead

-why? people did not like it

-needed to evolve HTML

-However, this is not the end of XHTML 🡪 XHTML5

Why? It does allow you to validate documents and plays well on XML environments (where you need an well structured document)

NOTE: Graphics and Games – SVG (xhtml base extension of the language) – scalable vector graphics

**6 Essential HTML Page Tags**

required tags, their placement, and their usage.

1<html>

2<head>

3<title> </title> \*IMPORTANT\* (most important because search engines use the words in the TITLE as keywords for categorization of your web page in their databases)

HEAD of an HTML page contains mostly invisible, meta, or scripting information.

</head>

4<body>

BODY of an HTML page contains ALL visible content; this includes all text, pictures, headings, lists, animations, sound files, and other multimedia elements.

</body>

</html>

 NO VISIBLE CONTENT may be placed in the HEAD of an HTML page! If it's visible, it goes into the BODY!

**7 Saving and Naming your HTML File**

 must create a file name for your HTML page. This name should always end with the file extension ".htm" or ".html" to indicate that it is an HTML page.

There are four rules governing the naming of HTML pages:

1. NO SPACES IN THE NAMES!
2. Always end the file name with ".htm" or ".html"
3. Only use characters from a-z, A-Z, 0-9, and \_ (underscore). All other characters (including commas, periods (except for the "dot" at the beginning of the dot-extension), tildes, exclamation points, etc) are considered reserved, and are therefore illegal.
4. Always start the file name with a lower-case letter; no numbers or underscore characters are allowed as the FIRST character of the name (although you may use them anywhere else in the name).

**8 Case-Sensitivity and File Names**

myPage.htm" is NOT the same as "myPage.html

URLs are CASE-SENSITIVE

**9 White Space and Other Technical Issues**

Specific HTML coding practices, rules, issues

White Space

Ordinary carriage returns are considered to be "white space". White space is a traditional programming term, used to describe anything which generates blank areas in code. White space includes spaces, carriage returns, tabs, non-breaking-space characters, and anything else which is space-like, carriage-return-like, or tab-like. White space has traditionally been used to break up code into clearer, more easily proof-readable constructions.

-it is used to clarify structure in code

-html does NOT recognize ordinary carriage returns as line breaks for text

-Anywhere that I want a line break I must insert BR tag

<br/> for XHTML

HTML5 is a case-insensitive language, which means that you may use either upper-case or lower-case characters in tag names.

For instance, **<br>** is the same as **<BR>** as far as HTML5 is concerned.

XHTML is a case-sensitive language, with all lower-case tag syntax. In XHTML, all tags must have closing tags.

In XHTML, there is only one acceptable way to code the BR tag: **<br />**

**10 Tag Nesting and the Parent/Child Relationship**

This nesting indicates something called a "parent-child relationship" between the two tags

<p>Here is an <i>italicized</i> word nested within a paragraph.</p>

SIBLINGS - nested tags under the same tag (same parent)

**11 Formatted Lists**

-A series of bullet points on some topic, or a list of steps or tasks to perform, or a sequence of definitions in the dictionary.

NOTE: comments - <!-- end nested list -->

Unordered

Tag: **UL**  
Attribute: **TYPE**  
Values: **disc**, **circle**, **square**

– bullet points (most common)

- UL stands for "Unordered List", while LI stands for "Line Item".

- UL has an attribute: TYPE – used to specify which type of bullets the list will display (3 types: reg, hollow circle, hollow square)

Ordered

Tag: **OL**  
Attribute: **TYPE**  
Values: **1** (default), **A, a, I, I //note: type=”1” is same as default #s**

**A – capital letters, a is lowercase leters, I is uppercase roman numerals,**

**i is lower-case roman numerals**

– numbers / letters

- Ordered Lists are created using the OL and LI tags

You may make your lists start on ANY number (not just 1, A, a, I, or i) by using the START attribute.

Tag: **OL**  
Attribute: **START**  
Value: any number

<ol start="4">

<ol type="I" start="10"> 🡪 starts at X

<ol type="A" start="6"> 🡪 starts at F

Definition – name of word, next line def.

If you wish to support IE5 for Mac, you must NEVER use nested ordered lists. Internet Explorer 5 for Windows does NOT have this problem.

**14 Resources and URLs**

-every digital thing is considered to be a resource

-HTML or text files are resources, pictures are resources, sound files are resources, Flash animation files are resources; any digital media file is a resource. Resource is a catch-all term for digital media or files of any kind.

-The HTML page invokes (or calls up) its various resources using something called a URL

-There are two kinds of URLs: absolute and relative.

-Absolute URLs point to a specific location on the entire Internet. Relative URLs point to a location on a web server which is in relation to the current HTML page.

**15 Absolute URLs**

An absolute URL represents a unique address on the Internet for any piece of digital media (resource).

Absolute URL syntax involves two parts, described in some textbooks as "scheme" and "scheme\_specific\_part", which must be separated by a colon and described in this manner (NO SPACES):

scheme:scheme\_specific\_part

schemes:

**http** (Web address)  
**mailto** (email address)

**ftp** (ftp site address for transfer of files)  
**file** (file address on a hard disk or server)

www.yahoo.com

"com" is the domain name.  
"yahoo" is the sub-domain name (which is commonly called the domain name).  
"www" is the name of the server in the "yahoo.com" domain.

web addresses always begin with the following code:

http://

http://www.yahoo.com/

"http" is the "scheme".  
":" separates the "scheme" from the "scheme\_specific\_part".  
"//" is the required prefix for an absolute web address.  
"www.yahoo.com" represents the server, the sub-domain, and the domain name.

mailto:srubin45@comcast.net

**16 The A (anchor) Tag, Part 1**

The A (anchor) tag allows you to create hyper-text references on an HTML page. Hyper-text, hyper-reference, hot-link, these terms all mean the same thing: a clickable link on a page (usually a word or a picture) which invokes another HTML resource.

Tag: **A**  
Attribute: **HREF**  
Description: The anchor tag allows you to create an anchor point in the flow of HTML text. The HREF attribute assigns a URL to an anchor tag and makes an active hyper-reference out of the word(s) or picture marked by that tag.  
Example:  
**<a href="http://www.yahoo.com/">Link Words</a>**

**17 IMG (image) Tag Basics**

A (anchor) tags are not restricted to invoking only HTML resources.

<a href="http://fog.ccsf.org/~srubin/satthumb.gif">View of Saturn</a>

even though we can invoke an image resource using the A (anchor) tag, we can't INSERT an image into an HTML page this way; we can only create a hot-link or hyper-reference TO an image. To actually insert an image/picture into the flow of an HTML page, we need to use the IMG (image) tag.

Tag: **IMG**  
Attributes: **SRC**, **WIDTH**, **HEIGHT**, **ALT, TITLE**

ALT attribute will allow it's value to pop-up when the user hovers over it, in most browsers except Firefox.

TITLE attribute will pop-up it's value when the user hovers over it in Firefox. I suggest always using both attributes for the IMG tag.

html5

<img src="http://fog.ccsf.org/~srubin/satthumb.gif" width="170" height="129" alt="saturn" title="saturn">

xhtml

<img src="http://fog.ccsf.org/~srubin/satthumb.gif" width="170" height="129" alt="saturn" title="saturn" />

**18 The A (anchor) Tag, Part 2**

**19 Relative URLs**

A relative URL is essentially a set of directions to a resource from your current location

These paths require special prefixes in order to operate correctly, as well as to maneuver through the server's directory structure.

There are three prefixes for relative URLs:

#### Prefix One:

**./**

Conservative syntax for plain relative URL; it means "look for the resource in the same directory."

(NOTE: In most cases the ./ can be omitted and so the above example is virtually the same as)

#### Prefix Two:

**../**

This means: "go up one directory level and start looking for the resource."

#### Prefix Three:

**/**

This means: "go to the root directory for the web server and start looking for the resource." A root directory for a web server is the single folder which has been assigned as the "public" web folder; any resource inside this "root directory" may be served to the outside world via the Web; no resources OUTSIDE this root directory may be served via the Web.

**20 The A (anchor) Tag and the NAME Attribute**

A (anchor) tags can also allow you to create hyper-references that jump to individual points WITHIN a particular HTML page.

you need two A (anchor) tags: One A (anchor) tag to mark the place that you want to jump to, and one regular A (anchor) tag to make the hyper-reference.

These "jump-to" points are called "NAME anchors", and are created using the NAME attribute in the A (anchor) tag.

Tag: **A**  
Attribute: **NAME**  
Description: The NAME attribute turns an A (anchor) tag into a "jump-to" point, which can be referenced by other A (anchor) tags used as hyper-references/hot-links.  
Example: **<a name="fred">Word</a>**

**Note:** Each NAME on an HTML page MUST be unique (no repeats).

"jump-to" points; they are NOT clickable hyper-references, nor do they alter the appearance of marked content.

Once marked in the HTML code, any hyper-reference may jump directly to a NAME-anchored point by calling that name as the URL, preceded by the pound (#) prefix. For instance, "**#fred**" would be an appropriate URL for a hyper-reference jumping to the "**fred**" NAME anchor elsewhere on the same page.

<a href="#fred">Link that jumps to "fred" NAME anchor</a>

<a href="#">This link will jump to the top of the page</a> //jump to top

You may also jump to NAME anchors that exist on OTHER HTML pages, using either absolute or relative URL syntax. Simply take the absolute or relative URL, and append the pound (#) sign, followed by the name of the NAME anchor that you want to jump to, to the end of the URL.

Example:

<a href="./page2.html#bill">Link to "bill" NAME anchor on page2.html</a>

**21 Heading Tags**

Tags: **H1, H2, H3, H4, H5, H6**  
Description: Creates headings and sub-headings for a document. Always closes.

Example:

<h2>Important Stuff</h2>

H1 has the largest font size, H6 has the smallest font size.

Heading tags automatically make text bold, and align to the left of an HTML page, just as paragraphs do. Heading tags automatically create line spacing following themselves; this following space grows smaller as the H tag numbers grow larger. Heading tags don't require extra BR tags to set themselves off from surrounding materials; they do this automatically as well. Heading tags ALWAYS close.

**22 Attributes**

We can use attributes to change the BEHAVIOR of a tag,

We can also use attributes to change the APPEARANCE of a tag.

Tag: **BODY**  
Attribute: **BGCOLOR**  
Values: color values (named colors or hex codes)  
Description: The BGCOLOR attribute in the BODY tag can be used to set the background color for the BODY of an HTML page.

Tag: **BODY**  
Attribute: **TEXT**  
Values: color values (named colors or hex codes)  
Description: The TEXT attribute in the BODY tag can be used to set the text color for all content in the BODY of an HTML page.

attributename="value"

<body bgcolor="black" text="yellow">

You may add as many attributes to a tag as you desire; there is no limit. However, you may NOT repeat the same attribute twice in the same tag!

**23 Coding Style and Tag Syntax**

**24 Comment Tag**

In HTML, a comment is just another tag, although it follows special rules. Here is what an HTML comment looks like in code:

<!-- Here is a comment -->

Incorrect (multiple lines):

<!-- Here is

a comment -->

Correct:

<!--

Here is a comment

Here is another comment

-->

//Note: particularly useful for tables

**25 Special Characters**

Certain characters in HTML are reserved, such as the less-than (**<**) and greater-than (**>**) signs, and the ampersand (**&**) character. If you type characters like these directly into HTML code, you will usually (although not always) get either a broken web page, or gibberish characters on the screen.

you must use character entity references, also known as escape sequences or escape characters.

The syntax for escape characters in HTML is always the same: The ampersand (**&**) character, followed by the character code, and ending with a semi-colon (**;**).

The character codes may be represented either by name (named entity) or by number (numeric entity); named entities are special text abbreviations of the character name, while numeric entities are special number codes representing the character preceded by a pound (**#**) sign.

Examples:

Character: **<** (less-than sign)  
Escape Character: **&lt;** or **&#060;**

**26 PRE Tag**

Tag: **PRE**  
Description: Used to set pre-formatted text. Always closes.

The PRE tag always displays content using the browser-default monospace font (usually Courier 10pt). BR tags are not required for line breaks within the PRE tag; ordinary carriage returns are recognized as line breaks, making the BR tag unnecessary. The PRE tag also respects space characters, displaying all spaces individually, rather than compressing a group of spaces together and displaying them as a single space.

Example (supported/standards-compliant -- use this):

<p>Here is a paragraph of text.</p>

<pre>

</pre>

<p>Here is another paragraph of text.</p>

Displayed:

Here is a paragraph of text.

Here is another paragraph of text.

**27 Physical Style Tags and Content-Based Style Tags**

There are a group of tags, used to mark-up text styles, which split into two types: physical and content-based style tags.

Physical style tags include B (bold), I (italic), SUP (superscript), SUB (subscript), and many more; these tags act physically on text exactly as they claim. Content-based style tags include STRONG (strong), EM (emphasis), CITE (cite/citation), etc; these tags represent what something MEANS, rather than what something looks like.

Whereas physical style tags impart a specific appearance to text that they mark-up, content-based style tags impart information regarding what that text is supposed to BE, or what it is used for. The fact that content-based style tags impart appearance characteristics to text is completely secondary.

They ALL must close.

Tag: **TT**Description: Makes text "teletype", aka the default monospace font (ordinarily, Courier 10pt, but this is dependant entirely on the user's browser settings).

Example:

<p>Mr. CodeHead says: <tt>var myGoober = 10 \* myRadius;</tt></p>

Displayed:

Mr. CodeHead says: var myGoober = 10 \* myRadius;

Content-based style tags are more concerned with what a particular piece of text means or represents, rather than what it looks like. Content-based style tags are probably going to disappear, over time, in favor of custom XML tags; XML tags are much more flexible and expansible in terms of the content they can potentially mark up. Of the content-based tags, only three are in occasional use today: EM (emphasis), STRONG (strong), and CITE (cite/citation).

Tag: **EM**Description: Adds emphasis to a piece of text, usually rendered using italics.

Example:

<p>I <em>really</em> want to do something with my life.</p>

Displayed:

I really want to do something with my life.

If you want to give an address on a web page in place of or in addition to your citation, use the ADDRESS tag (this is not a content-based style, but I think I should mention it anyway).

Tag: **ADDRESS**Description: Marks a mailing address, including email contact, if applicable.

**28 BLOCKQUOTE and HR**

The <blockquote> tag specifies a section that is quoted from another source.

Browsers usually indent <blockquote> elements.

note that the hr tag should self-close in XHTML, i.e., <hr />

Tag: **HR**  
Attribute: **SIZE**  
Value: integer representing thickness of horizontal rule in pixels  
Description: sets HR thickness.

Tag: **HR**  
Attribute: **WIDTH**  
Value: integer representing width of horizontal rule, in pixels. May also be set to percentage value (representing percentage of browser window width).  
Description: sets HR width.

Tag: **HR**  
Attribute: **ALIGN**  
Value: **left, right, center**  
Description: sets the alignment of a horizontal rule on a page.

Horizontal rules are usually inserted to indicate section breaks in a long document, or as a divider or decorative element. HR tags used to be used with such frequency that they came to be associated with unimaginative web page design, and should be used sparingly.

**29 Miscellaneous HTML Tags**

Tag: **BR**  
Attribute: **CLEAR**  
Values: **all**, **left**, **right**  
Description: The BR tag, in combination with an IMG tag, can define how text will break in relation to a left-or-right-aligned image. **clear="all"** forces text to break below all images; **clear="left"** forces text to break below the left-aligned images; **clear="right"** forces text to break below the right-aligned images. "all" is the only commonly used value for the CLEAR attribute.

Example: **<br clear="all">**

**30 META Tag**

The **META** tag is used primarily for two purposes today:

1. It helps robotic search engines (such as Google and Yahoo) better categorize your HTML pages.
2. It automatically moves users from one page to another (as seen on splash pages or forwarding-address pages) using its auto-refresh capabilities.

requires two attributes: **NAME** and **CONTENT**. **NAME** identifies the type or variety of **META** tag in question; **CONTENT** sets the necessary information for that version of the **META** tag. The **NAME** attribute, in some cases, is interchangeable with the **HTTP-EQUIV** attribute; in other words, either **NAME** or **HTTP-EQUIV** will be used as the identifier attribute for the **META** tag, depending on the purpose of that particular **META** tag; the second attribute will always be **CONTENT**.

always goes within the HEAD of an HTML page; it is NEVER placed in the BODY. You may place as many different **META** tags in the HEAD as you like.

An HTML page on the Web is catalogued in a search engine's database with one entry for every pertinent keyword on the page. Each individual word in the TITLE of an HTML document receives a separate keyword entry in the database. Each individual keyword in the **KEYWORDS META** tag receives a separate keyword entry in the database.

1. whether the word is in the TITLE (top priority),
2. whether the word is in the **KEYWORDS META** tag, and
3. whether the word appears in the text in the BODY of the HTML page.

In addition to the position of the keyword on the HTML page as a whole, the search engine "grades" the importance of a given HTML page based on how close that page is located to the root directory of its domain; top level (root directory) pages are given top priority, even if a given keyword does not appear in the TITLE of that document, but merely in the **META** tag or in the BODY.

**DESCRIPTION META** tag allows you to create a brief description of your page or website which will display below the TITLE and URL for your website on a robotic search engine's results page.

If the **DESCRIPTION META** tag has NOT been set, then the user sees the first 10-20 words from the BODY of that HTML page in place of the description.

<meta http-equiv="refresh" content="10; url=http://www.yahoo.com/">

Two attributes of the **META** tag are required to utilize the auto-refresh feature: **HTTP-EQUIV** and **CONTENT**. The **HTTP-EQUIV** attribute must be set equal to **"refresh"**. The **CONTENT** attribute must be set equal to a value following this format: **"secondsOfDelay; url=someURL"**

**UTF-8 and HTML5**

UTF-8 stands for **U**nicode **T**ransformation **F**ormat-**8**. It is an octet (8-bit) lossless encoding of Unicode characters. The following meta statement is highly recommended in HTML5 files.

<meta charset="utf-8">

**31 Document Appearance**

#### What is the difference between the Proportional and Monospace Fonts?

Monospace fonts, such as Courier and Courier New, have characters which are all identical in width. A space character is as wide as an "M", which is as wide as an "i"; all characters in a monospace font are the same width.

Proportional fonts are used for laying out most ordinary body text, whether for publishing on the Web or for print purposes. Most text within the BODY of an HTML document is laid out using the browser's default proportional font.

Macintosh and Windows PC machines will display 12 point text in a browser window at different sizes

It is impossible to create HTML pages which display with identical font sizes cross-platform without the use of CSS. Without CSS, in fact, it is impossible to be certain of the font size under any circumstances, since users may reset browser default font sizes at will.

Table Notes:

Remember: the row where the upper left-hand corner of a cell resides is the row into which you would code that particular cell. Always look to the upper left hand corner of a cell, and you'll never have trouble deciding into which row you need to place a particular cell.

Alignment:

Notice how both cells in the top row automatically display their content as valign="middle" by default, since I didn't explicitly specify the VALIGN attribute in either of the cells in the top row. In the bottom row, both cells are align="center" by default because they're both created using TH tags (which automatically align center).

Here's the same example, modified with additional ALIGN and VALIGN attributes; I just wanted you to understand that you can use as many of these attributes together in a tag as you wish or desire.

Fixed Table Width:

Individual table cells, sadly, can NOT be set to precise sizes using the WIDTH attribute of the TH or TD tags. Individual table cells may only be reliably sized using GIFs, not the WIDTH attribute. The WIDTH attribute of the TH and TD tags produces results which may or MAY NOT be approximately the size requested; they are not accurate.

To achieve accurate and consistent table cell widths, tables must have most of their built-in spacing taken out. BORDER, CELLPADDING, and CELLSPACING must all be set to "0", i.e. **<table width="500" border="0"cellpadding="0" cellspacing="0">**. Spacing between columns of cells, then, must be created with table cells between the main columns; these "spacer" columns of cells must be filled with some sort of "spacer" GIFs.

A spacer GIF is a 3 pixel by 3 pixel GIF which is the same color as the background color of your web page. Realistically, only three colors of spacer GIFs are used: white, black, and transparent (the transparent spacer is used with all background colors and designs except white and black).

Do NOT use 1 pixel by 1 pixel spacer GIFs

An alternative to using spacer gifs is to use &nbsp; as the content of a table cell. For example:

Table Nesting:

You may nest as many sibling tables as you like inside of a parent table cell. It is important to note, however, that you must NEVER nest tables more than one level deep (i.e. no grandchild or great-grandchild nested tables). In other words, don't try to nest a table inside of one of your nested table's cells; that is forbidden. Nesting tables more than one level deep creates printing and display difficulties that you will want to avoid.

**Printable Pages**

Never use tables to hold printable content, especially if they have been modified with CSS; you'll get very irregular printed results. Create printable pages with vanilla HTML, no TABLES! Better yet, if you have the time and inclination, create printable and downloadable Adobe Acrobat (PDF) files, which are the standard for web-based documentation, as well as being very consistent in appearance cross-platform (much more so than HTML pages).

**Miscellaneous**

Never put content in a cell which would make that cell BIGGER than your defined column width; you'll get very irregular results if you do.

Variable Table Width:

 At 100%, obviously, the table would fill the entire width of the browser window. At lesser percentages, you might want to set the ALIGN attribute of the TABLE tag equal to "center".

If you want your variable width table to extend right up to the edge of the browser window, of course, you will need to set the WIDTH attribute of the TABLE tag to 100%, as well as to set the MARGINHEIGHT, MARGINWIDTH, LEFTMARGIN, and TOPMARGIN attributes of the BODY tag equal to 0.

As the browser window widens, the columns will expand proportionally to one another based on the size of the graphics in the first row of the table.

Remember: graphics can NOT be resized on the fly as the table expands!

NOTES: If you want consistent appearance cross-platform and cross-browser, the BGCOLOR attribute should ONLY be added to TH and TD tags. Secondly, if you desire to create an empty colored cell, you should place an &ampnbsp inside of the cell. Lastly, you may also add the BACKGROUND attribute to the table, tr, th, and td tags.

CSS Intro Notes

 CSS allow for much greater control over HTML appearance, it also provides a mechanism for making global appearance changes to a group of HTML pages, or even an entire website.

allows you to create external ".css" files, which contain all of the HTML tag appearance information. These ".css" external style sheet documents can then be applied to as many HTML pages as you like.

When you redefine an HTML tag's appearance using CSS properties, you override the web browser's default settings; conversely, any property which you do NOT define reverts to the browser's default setting.

Style sheet properties are divided into two parts:

1. property name
2. value

The property name and the value are separated by a colon (**:**). The entire property ends with a semi-colon (**;**).

property-name:value;

Note: A property name NEVER has any spaces in it. An individual value NEVER has any spaces in it. There MAY be a space after the colon.

The following example is acceptable:

font-size: 24pt;

You will also notice that there are NO equals signs (**=**) or quote marks (**""**) in CSS. ..

An HTML tag's appearance properties are defined in CSS using the following syntax:

tagname { }

 This is called a "declaration"..

You will place the properties that you want to define for the tag inside the curly braces; you may define as many properties in a declaration for a given tag as you want.

agname { property-name:value; property-name1:value1; property-name2:value2; }

The tagname portion of the declarations shown above is called the "selector". The "selector" is whatever tag you're currently defining in CSS.

tagname = selector

grouping selectors (tags) that share properties:

h1, h2, h3 { text-align:center; color:#000000; }

h1 { font-size:36pt; }

h2 { font-size:24pt; }

h3 { font-size:18pt; }

You must be VERY careful NOT to repeat properties for the same tag within separate declarations. The following example would be WRONG:

h2, p { color:#000000; }

p { font-size:24pt; color:#330000; }

CSS comments do NOT look like HTML comments; they follow traditional C/C++ comment syntax.

Single line comments are prefixed by slash-slash (//) and end when you type a carriage return. Multiple line comments are prefixed by slash-asterisk (/\*) and end with an asterisk-slash (\*/).

Having said this, single-line comments are broken in CSS; do NOT use single line comment syntax!

A contextual selector for a declaration states the parent selector first, then a space, then the child selector, followed by a space and the curly-braces containing the properties.

Generic Example:

parenttagname childtagn

ame { property:value; etc... }

Actual Example (defining the **B** tag when it is a child of the **P** tag):

p b { color:#660066; font-weight:bold; }

**Remember**: you can define the main properties for a tag in a general declaration (like the one for the **B** tag above), then override one or more of the properties defined for the tag in different CONTEXTS (situations) using contextual selectors in CSS declarations, as you can also see above.

When creating contextual selectors, you MUST separate the different selectors with spaces. The following example is WRONG:

pb { color:#660066; }

There are three main types of cascading style sheets: linked, embedded, and inline.

Linked style sheets are the preferred form of style sheet.

This external text-only document is linked to the HEAD of an HTML page using the LINK tag. –many pages

Embedded style sheets use CSS declarations placed within a STYLE tag in the HEAD of a particular HTML page. Because this embedded style information is inside of the HEAD of one HTML page, it affects only the appearance of tags on that HTML page, and can not be applied to any other HTML pages.

Inline style sheets can only use a limited portion of basic CSS syntax; they do not affect HTML tags on a page as a whole, they merely affect whatever is marked by their particular **SPAN**tag.

used in much the same way as the old **FONT** tag.

Inline style sheets are only useful if you do not have access to the HEAD of the HTML page in question. In other words, if you are adding information to the BODY of an HTML page through some web-based mechanism but cannot affect the HEAD of that page, you might find inline style sheets helpful.

When two or more style sheets are in conflict (referring to the same properties of the same tag), this conflict must be resolved. Conflict resolution in style sheets is called the "cascade".

cascade is resolved by the specificity of the style sheet in question.

In order of specificity (most specific to least specific):

1. Inline
2. Embedded
3. Linked
4. Web Browser Default Settings

An embedded style sheet is style information placed within a **STYLE** tag in the HEAD section of a single HTML page.n

<head>

<style type="text/css">

/\* style information here \*/

</style>

</head>

eg:

<head>

<style type="text/css">

p { font-size:14pt; color:#000000; }

b { font-weight:bold; }

</style>

</head>

An external CSS document is attached to an HTML page using the **LINK** tag. The **LINK** tag, like **IMG**and **BR**, doesn't ordinarily close in HTML, but if you are using XHTML, then you must use the "space-close-slash" before the closing bracket syntax.

The **LINK** tag always goes in the HEAD of an XHTML page. **LINK** requires three attributes: **REL**,**TYPE**, and **HREF**.

Inline style sheets mainly use the **SPAN** tag within the BODY of an HTML page to add style to selected text.

Example (abbreviated):

<span>Here is some text.</span>

SPAN requires one attribute: **STYLE**. The **STYLE** attribute of the **SPAN** tag will be set to some number of properties which would then apply to that one **SPAN** tag.

 inline style sheets do NOT redefine tags; they merely mark text in HTML using the**SPAN** tag. Again, inline style sheets override both embedded style sheets and linked style sheets applying to the same HTML page.

To define more than one version of an HTML tag, you need to use a CLASS. Class declarations in CSS are prefixed with a dot (**.**), followed by the name of the class, a space, and the curly-braces containing the style properties; as with any CSS declaration, you may define as many properties for a class declaration as you wish.

Example:

.warning { color:#FF0000; }

In an HTML page connected to this CSS declaration, then, I would add the **CLASS** attribute to any HTML tag; the value of the **CLASS** attribute would be the name of the desired class in the style sheet, **"warning"**.

Example:

<p class="warning">This paragraph has had the warning class applied

to it.</p>

if you want to restrict usage of a class declaration to ONE kind of HTML tag, you can specify that it in the class declaration, as well, by adding the desired tag selector before the dot (**.**) character.

Example:

p.warning { color:#FF0000; }

By adding the **P** tag selector before the dot (**.**) character in the above class declaration, I have restricted usage of this class to the **P** tag.

Class names must NOT be repeated in a style sheet. You can NOT, for instance, create two classes of the same name for different tags.

Example (WRONG):

p.warning { color:#FF0000; }

p b.warning { color:#FFFF00; }

Since every class name must be unique, a common practice in this circumstance is to append the name of the tag to the end of the class name, giving you unique names.

Example (CORRECT):

p.warningP { color:#FF0000; }

p b.warningPB { color:#FFFF00; }

the **SPAN** tag would NOT be considered an inline style sheet, because its CLASS style information would be defined in a linked or embedded style sheet; the **SPAN** tag would merely be a blank HTML tag which you could apply CSS properties to.

Example:

<p>Here is some text with <span class="warning">special words</span>

in it.</p>

When setting font sizes in style sheets, you MUST use PIXEL-based values if you want consistent appearance. In your style sheets, then, instead of saying **"24pt"** for 24 point font size, you would need to say **"24px"** for 24 PIXEL font size.

A point is 1/72 of an inch. A 12 point font, therefore, should be 12/72 of an inch tall (roughly). The Macintosh displays everything at 72 pixels per inch.

On a PC, fonts are displayed by the Windows OS at 96 pixels per inch. A 12 point font, for instance, is 12/72 of an inch, 12/72 of 96 pixels is 16 pixels, therefore a 12 point font is technically 16 pixels tall on the PC. A web browser displays everything at 72 pixels per inch. When a 12 point font on the PC is displayed in a web browser, the pixels are all preserved, so the font becomes 16 pixels tall, or 16/72 of an inch (which is really 16 points). If you want to preserve what you see at "12 point font size" in a web browser on a PC, then, you must set the CSS property font-size to 16px (16 pixels).

The conversion formula for the PC:

point-based-font-size / 72 \* 96 = pixel-based-font-size

Example Conversion for a 30 point font as viewed in a web browser on the PC:

30 / 72 \* 96 = 40px

If you just do all of your design using pixel-based font sizes, you will avoid having to do these conversions

when setting fixed font-size in CSS: users will NOT be able to resize the fonts in their web browser to make them larger; this can be a problem for users whose monitors display small pixels or for users who are visually-challenged. – fix with javascript

The pseudo-class was developed to define tags with more than one state. The only tag in HTML that has more than one state is the **A** (anchor) tag, which has three official states: link, visited, and active.

To define the **A** (anchor) tag, then, you need to use the pseudo-class, which is marked in a CSS declaration using the colon (**:**) character to separate the selector and the name of the pseudo-class.

Example:

a:link { color:#0000FF; }

a:visited { color:#FF00FF; }

a:active { color:#FF0000; }

Due to flaws in early web browser CSS implementations, you must ALWAYS set the pseudo-class for the **A**(anchor) tag when using CSS (otherwise, early Internet Explorer versions will make your hyper-references the same color as your main text color, making hyper-references and text indistinguishable from one another).

You may also use contextual selectors to define different versions of the **A** (anchor) tag.

At this time, classes of the pseudo-class are not fully supported,

Example:

a:link { color:#0000FF; text-decoration:underline; }

a:visited { color:#FF00FF; text-decoration:underline; }

a:active { color:#FF0000; text-decoration:underline; }

To turn OFF the underline, set the text-decoration property to the none value.

Example (abbreviated):

text-decoration:none;

Lastly, a very useful pseudo-class involves the changing of a link's color when the mouse is placed or hovers over it.

Example:

a:hover { color:#000000; }

If you want only certain links to display some style when the mouse is placed or hovers over them then you should use a class.

Example:

<head>  
  
<style type="text/css">  
  
a.x:hover {color: #000000; background-color: yellow; font-weight: bold;}  
  
</style>  
  
</head>  
  
<body>  
  
<h2>a:hover and class</h2>  
  
<a href="#">Hover does nothing</a><br /><br />  
  
<a class="x" href="#">Hover with class="x"</a>

Individual values in CSS NEVER have spaces in them. The only exception to this rule is in regard to regular font names like "Times New Roman" and "Courier New", which must be enclosed in quote marks.

There are three types of measurement units: percentage, absolute, and relative.

Percentage units are represented using integers and the percent (**%**) sign in the usual manner; negative values are also allowed (when applicable).

Example:

10%

200%

-130%

Absolute units include **in** (inches), **cm** (centimeters), **mm** (millimeters), **pt** (points; 1 point equals 1/72nd of an inch), and **pc** (picas; 1 pica equals 12 points).

With the absolute units, you may use any number to define your value, including numbers with decimal places and negative numbers.

Relative units include **em** (ems), **ex** (ex's), and **px** (pixels).

Pixels are considered to be relative units because monitors display pixels at different sizes. Despite this, pixels are an absolute quantity in CSS, because 1 pixel is 1/72nd of an inch as far as web pages are concerned.

Pixel values are always stated using integers.

**Text Properties**

**Property: font-family**

Example:

font-family:"Times New Roman", Times, serif;

As with the deprecated **FONT** tag with its **FACE** property, you need to give users more than one font choice, in case the first choice font is not loaded on their system. The first font in the list is the first choice, the second font in the list is the fallback choice if the first font is not available, etc.

 You should always end your list with one of the five generic font names; the generic font name should NOT be in quotes.

There are also relative values available for font-size, but they are not consistently implemented. Theoretically, they are each supposed to be 50% smaller or larger than their neighbor,

"medium" is supposed to be the browser default size.

Values:

xx-small, x-small, small, medium, large, x-large, xx-large

There are two other relative values which DO work:

* smaller (50% smaller than the parent element's font)
* larger (50% larger than the parent element's font)

#### Property: font-style

Example:

font-style:italic;

Values: **normal, oblique, italic**  the oblique value ONLY works if the user has an oblique version of that font loaded into the browser

#### Property: text-decoration

Example:

text-decoration:underline;

Values: none, underline, line-through

#### Property: text-indent

Example:

text-indent:.5in;

p { text-indent:-.5in; margin-left:.5in; }

#### Property: line-height

line-height:120%;

line-height:24px;

line-height:normal;

The line-height property sets the "leading" for text, creating space between lines in a paragraph. Set to percentage values, pixel values, or "normal".

#### Property: text-align

Example:

text-align:left;

Values: left, center, right, justify

You must always set text properties for the tags individually!

1. conceptual parts of EACH element The element itself.
2. The padding, which is the space between the element and its border.
3. The border.
4. The margin, which is the space between an element's border, and the next element's margin.

Not only can you set these properties, you can set the four sides of any of these properties individually or collectively, as well. For instance, you can set a margin of 1 inch all the way around something, or you can individually set the left, right, top, and bottom margins at different sizes.

don't use these properties with the IMG tag (in fact, do NOT define the IMG tag in CSS at all, PERIOD, or you'll be sorry!). Do NOT set these properties for individual words or hyper-references within an element, or you will get very inconsistent behavior cross-browser.

border property requires three values: the width of the border, the style of the border, and the color of the border, each value separated by a space.

You can use the background property for setting the background color and/or image for the BODY of an HTML page. One of the interesting features of the background property is that it may now be set for individual elements, such as paragraphs, as well!

ID

An ID is identical, in many ways, to the **NAME** attribute in HTML; in fact, in future browsers, ID will completely replace the **NAME** attribute! An ID is used to identify an element for manipulation in a scripting language, such as JavaScript.

An ID is declared in CSS much like a class, only using the pound (**#**) sign rather than a dot (**.**) character. Unlike a CLASS, an ID may NOT be prefaced by a selector or contextual selector; it must stand alone in the CSS code.

Example:

#banana { color:yellow; }

 may ONLY be applied to ONE element on a given HTML page.

IDs are mostly used with CSS-P, for positioning elements absolutely on an HTML page, and with Dynamic HTML.

Some properties are inherited from a parent object, and some properties are NOT inherited. Color and font-size, for instance, ARE inherited, while background, margin, and border (among others) are NOT inherited.

NOTE:  
selectors – tags b { font-weight:bold; color:#000000; }

paramaters – { parameters go here }

contextual selectors – allow you to define the appearance for HTML tags in parent-child relationshiops with other HTML tags

parenttagname childtagname { property:value; etc... }

p b { color:#660066; font-weight:bold; }

To define more than one version of an HTML tag, you need to use a CLASS. Class declarations in CSS are prefixed with a dot (**.**), followed by the name of the class, a space, and the curly-braces containing the style properties; as with any CSS declaration, you may define as many properties for a class declaration as you wish.

.warning { color:#FF0000; }

<p class="warning">This paragraph has had the warning class applied

to it.</p>

Now, if you want to restrict usage of a class declaration to ONE kind of HTML tag, you can specify that it in the class declaration, as well, by adding the desired tag selector before the dot (**.**) character.

Example:

p.warning { color:#FF0000; }

By adding the **P** tag selector before the dot (**.**) character in the above class declaration, I have restricted usage of this class to the **P** tag.

Example (WRONG):

p.warning { color:#FF0000; }

p b.warning { color:#FFFF00; }

Since every class name must be unique, a common practice in this circumstance is to append the name of the tag to the end of the class name, giving you unique names.

Example (CORRECT):

p.warningP { color:#FF0000; }

p b.warningPB { color:#FFFF00; }

The **SPAN** tag may also be used in combination with the **CLASS** attribute. The **SPAN** tag itself does not have any special formatting associated with it, and lends itself well to acting as a generic shell HTML tag. In this kind of situation, the **SPAN** tag would NOT be considered an inline style sheet, because its CLASS style information would be defined in a linked or embedded style sheet; the **SPAN**tag would merely be a blank HTML tag which you could apply CSS properties to.

Example:

<p>Here is some text with <span class="warning">special words</span>

in it.</p>

<span> will allow you to style text, but it adds no semantic content.

As you're emphasizing some text, it sounds like you'd be better served by wrapping the text in <em></em> and using CSS to change the color of the <em> element. For example:

**CSS**

.description {

color: #fff;

}

.description em {

color: #ffa500;

}

**Markup**

<p class="description">Lorem ipsum dolor sit amet, consectetur

adipiscing elit. Sed hendrerit mollis varius. Etiam ornare placerat

massa, <em>eget vulputate tellus fermentum.</em></p>

When a web browser loads an HTML page into a browser window, it reads the HTML code, finds the image resources marked by the IMG tags, and downloads them; it then assembles the text and pictures together and displays them in the browser window. Wherever you place an IMG tag in the flow of HTML text, a picture will be inserted.

Tag: **IMG**  
Attribute: **SRC**  
Value: a URL (relative or absolute) which gives the location of the image resource.  
Description: The SRC attribute of the IMG tag states exactly where the desired picture resource is located on the Internet.

<img src="./graphics/capitalY.gif" /> (relative URL)

<img src="./capitalY.gif" /> (relative URL)

<img src="capitalY.gif" /> (relative URL)

<img src="http:/fog.ccsf.org/capitalA.gif" /> (absolute URL)

The IMG tag has two additional attributes which should ALWAYS be used: WIDTH and HEIGHT. These attributes state the width (in pixels) and the height (in pixels) of the digital image in question.

 If you forget to put WIDTH and HEIGHT into your IMG tag, the web browser may prevent the display of anything on the HTML page following that IMG tag until the entire picture is finished downloading.

You may resize an image in the browser window, on the fly, by setting the WIDTH and HEIGHT attributes of the IMG tag to values other than the actual width and height of the image. This is not recommended for most pictures, however, as browsers are extremely poor at resizing images on the fly, introducing ugly pixelations, distortions, empty gaps, or other visual flaws into the picture. The only time you'll resize pictures in this manner is when you are putting what are called "spacer" GIFs into an HTML page; because these "spacer" GIFs are just one color (white, black, some other color, or transparent),

In conclusion, the minimum complement of attributes for the IMG tag must include SRC, WIDTH, and HEIGHT.

A pixel is a square or rectangular dot of colored light. Computers use pixels to display visual content to a user; you put enough pixels together, and you have a picture. Pixels are always formed in a square grid, just like graph paper, in rows and columns.

For the Web, digital images must be created with a resolution of 72 pixels per inch. Remember this when making images in Photoshop for web-based delivery. For print, of course, 300 pixels per inch, 600 pixels per inch, and even 1200 pixels per inch resolutions are not uncommon, but images for the Web are always 72 pixels per inch.

The ALT attribute sets a text equivalent for a picture, so that users with their images turned off will still see something in the browser window where the picture should be. The ALT attribute should be always be included within the IMG tag to accomodate those with sight disabilities. Users with images turned on (normal) will also see the ALT text before the picture downloads completely. On some browsers, the ALT attribute text also pops-up as a "rollover" comment as the user mouses over the picture; this feature, however, is not universal among browsers and platforms and cannot be relied upon. Note that Firefox does not display the ALT attribute text as a pop-up "rollover" comment as the user mouses over the picture. Alternatively, in Firefox you can use the TITLE attribute to pop-up the "rollover" comment.

Tag: **IMG**  
Attribute: **ALIGN**  
Values: **left**, **right**, **top**, **middle**, **bottom**  
Description: sets the alignment of text in relation to an image.

The two most common values for ALIGN are "left" and "right", which allow content to flow around your picture very handily.

If you wish to center a graphic on a page, use the DIV tag with its ALIGN attribute set to CENTER, placed around the IMG tag.

VSPACE allows us to set the vertical space (space at the top and bottom of the IMG), in pixels, between the edge of the IMG and the beginning of text or other content. HSPACE allows us to set the horizontal space (space at the left and right of the IMG), in pixels, between the edge of the IMG and the beginning of other content.

Tag: **IMG**  
Attributes:  
**SRC** (states URL where the image resource is located)  
**WIDTH** (states the width of the image, in pixels)  
**HEIGHT** (states the height of the image, in pixels)  
**ALT** (states the alternative text for the image when images are disabled)  
**TITLE** (pops-up as a "rollover" comment as the user mouses over the picture)  
**BORDER** (states the border for the image, usually set to 0 (zero) in combination with the A (anchor) tag)  
**ALIGN** (allows text wrap around a picture when set to "left" or "right")  
**VSPACE** (creates space, in pixels, above and below the picture)  
**HSPACE** (creates space, in pixels, to the left and right of the picture)

There are two primary compression schemes for image files used on the Web today: GIF and JPEG.

GIF compression works best with images that have lots of flat color space (no gradations or continuous tones), such as drawings and flat-color graphics.

JPEG compression works best with pictures that are photographic in nature, without sharp edges, and containing lots of graduated, continuous tone color.

PNG is a bitmapped image format that employs lossless data compression. PNG was created to improve and replace the GIF format, as an image-file format not requiring a patent license. PNG is pronounced "ping".

There are two ways to express color in HTML: 1) named colors, and 2) hex codes.

 There are 16 "official" named colors: aqua, gray, navy, silver, black, green, olive, teal, blue, lime, purple, yellow, fuchsia, maroon, red, and white.

named colors have not traditionally been consistent in hue from platform to platform and browser to browser, despite official standards. Secondly, these named colors are mostly not "web-safe";

Hex codes express color precisely using the 24-bit RGB color palette, the standard for full-color reproduction on computers. Hex codes are expressed in seven digits, the first digit always being the # (pound) symbol, followed by six numbers representing the desired color.

Why are there LETTERS in some of these hex codes, and what do these numbers and letters represent?

Hex codes are written in Base 16, the hexadecimal counting system.

Base 16 uses:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10

Because Base 16, or hex, rolls over to "10" at 16 rather than 10, we need to add some letters into our numbers in order to make up the difference. This means that "A" is really 10, "B" is really 11, C is really 12, etc, until you get to "10" which is equivalent to our 16.

 24-bit color, there is one 8-bit palette with 256 shades of red, one 8-bit palette with 256 shades of green, and one 8-bit palette with 256 shades of blue; the three palettes are combined to make millions of possible colors.

Web-safe colors may only use the following digits (in hex): **00, 33, 66, 99, CC, FF**. The following randomly selected codes, then, WOULD be web-safe: **#3399CC, #6666FF, #9900FF**, etc. The following randomly selected hex codes would NOT be web-safe: **#3399A5, #1234AB, #DDDDDD**, etc.

Testing your designs in grayscale will give you a much clearer (although not completely accurate) idea of what your design will look like to the color-blind.

Don't overload your front page with content;

maintain a clear and obvious focal point for the page.

don't succumb to clutter

Maintain a clear navigational structure. There is nothing more annoying than losing your way in a website of any size, small or gargantuan, and having no clear idea of where you are or how to get anyplace else.

Use designs with strong focal areas which don't require pinpoint positioning.

complete design considers the entire user experience, from ease of access to key content and information, to arrangement of content, to navigational structures and site maps, to proper management of expectations regarding wait and delay (a primary component of most web experience)

img turned into hyper link

<a href="http://www.yahoo.com/"><img src="capitalA.gif" width="54" height="54" border="0" alt="Capital A" /></a>

an imagemap allows you to take an ordinary image and make multiple clickable hyper-references available to the user from that one image.

A shareware program can be downloaded from the Web called MapEdit, which can handle imagemap construction for you automatically. MapEdit is offered for Mac, PC, and UNIX platforms.

On the computer, locations on the coordinate plane are mapped in a slightly different manner. The origin point of the coordinate plane on the computer is always in the upper-left-hand corner of whatever is being scrutinized. If you are determining locations on the computer screen, the origin point is at the upper-left-hand corner of the screen. If you are determining locations on a web browser window, the origin point is at the upper-left-hand corner of the browser window. If you are determining locations on a digital image, the origin point is at the upper-left-hand-corner of the image itself.

On the computer, values on the y-axis increase as you move DOWN, not up; this is because the origin point is at the upper-left-hand corner of things.

When you are deriving the position and shape of clickable areas for an image, you will be doing so using these pixel-based x,y coordinates.

There are only three types of shapes which may be defined as clickable areas in an imagemap: **RECT**,**CIRCLE**, and **POLY**.

The **RECT**shape includes both rectangles and squares. It is defined by two points on the digital image: the position of the upper-left-hand corner of the rectangle, and the position of the lower-right-hand corner of the rectangle.

x,y,x1,y1

The **CIRCLE**shape includes perfect circles only; no ellipses, ovoids, or other irregular shapes are permitted. A CIRCLE shape is defined on an image by the position of the center point of the circle, followed by the radius value of the circle, in pixels;

x,y,r

The **POLY**shape includes all polygons, beginning with triangles and increasing in complexity to the most peculiar and irregular shape you can imagine. A POLY shape must have at least three x,y coordinate points defining it (a triangle), but may have as many x,y coordinate points as desired, in any shape, no matter how squiggly or peculiar.

 It doesn't matter whether you move clockwise or counter-clockwise around the polygon, just as long as you present the x,y coordinates in order. If you don't do this, you will not get the shape you want, but some ugly mess, as the computer draws the shape from first point to second point to third point, etc. You don't need to repeat the x,y coordinate for the starting point once you get to the last x,y coordinate of the polygon.

x,y,x1,y1,... xn,yn

All that you need for deriving x,y coordinates is image editing software, such as Adobe Photoshop or Paint Shop Pro.

the **IMG**tag which displays the image on the XHTML page, and the **MAP**and **AREA**tags (invisible to the user) which contain all of the imagemap information.

Because the actual imagemap information is coded only into **MAP**and **AREA**tags, the **IMG**tag needs to link to the **MAP**tag and its imagemap information using a special attribute, **USEMAP**

**MAP**opens and closes around the imagemap information, with one **AREA**tag for each clickable area on the imagemap. Here is an example of the basic structure (without attributes):

<map>

<area />

<area />

<area />

</map>

The **MAP**tag ALWAYS closes after the **AREA**statements.

The **MAP**tag has one attribute, **ID**, which is required for the imagemap to operate properly.

Tag: **MAP**  
Attribute: **NAME**  
Value: any valid name (based on the naming rules discussed in earlier modules)  
Description: the **NAME**attribute of the **MAP**tag provides an identifier which the **USEMAP**attribute of the **IMG**tag will use to connect itself to the imagemap information in the **MAP**.  
Example (abbreviated): **<map name="fred"></map>**

Note: In order for your XHTML pages to validate with the W3C standards, you need to insert an id attribute into your map statement. For example:  
  
<map name="fred" id="fred">

Tag: **AREA**  
Attribute: **SHAPE**  
Value: **rect**, **circle**, **poly**  
Description: sets the shape of the clickable area which the individual **AREA**tag will define. Each **AREA**tag in a**MAP**may be a different **SHAPE**, or they may all be the same **SHAPE**.

Tag: **AREA**  
Attribute: **COORDS**  
Value: integer **x,y**(and **r**) coordinates separated by commas; the number of integers required depends entirely on the value of the **SHAPE**attribute for that **AREA**tag.  
Description: the **COORDS**attribute of the **AREA**tag defines the **x,y**(and **r**) coordinates for the shape defined by the **SHAPE**attribute of that particular **AREA**tag.

Tag: **AREA**  
Attribute: **HREF**  
Value: any URL, relative or absolute  
Description: the **HREF**attribute of the **AREA**tag defines the hyper-reference for that clickable area in the imagemap. Note that HTML5 does not require the area tag to self-close.

Example:

<img src="demoMap.gif" width="200" height="100" alt="Demo Map" border="0" usemap="#ethel" />

<map name="ethel" id="ethel">

<area shape="rect" coords="7,7,71,71" href="destination1.html" />

<area shape="circle" coords="127,202,25" href="destination2.html" />

<area shape="poly" coords="80,247,41,187,39,131,45,111,84,124" href="destination3.html" />

</map>

**BORDER**attribute of the **IMG**tag must be set equal to **"0"**when that IMG is being used for an imagemap. Because an **IMG**tag used as an imagemap is considered to be a hyper-reference by a web browser, a blue hot-link border will appear around the image (in many browsers) unless the **BORDER**attribute is set equal to **"0"**.

 DO NOT have TWO **MAP**tags of the SAME NAME on a single XHTML page; this is not only pointless, it is forbidden.

Multimedia Notes:

Midi Files:

-Musical Instrument Digital Interface (MIDI) protocol

-files are small

-not a specification for sampled digital audio; it is a bank of digitized sounds and control information for replaying the file (similar to electronic synthesizer)

Wav Files:  
-proprietary to Microsoft and IBM (windows-based pcs)

Mp3 and Mp4 Files:

MP3 files are CD quality songs that are compressed down from huge files to much smaller files without any noticeable decrease in sound quality.

MP3 stands for MPEG 1 (Motion Picture Experts Group) Layer 3.

MP3 squishes down sound files 10-15 times smaller than a parent file. And yet that tiny .mp3 file retains most of the perceived audio fidelity by stripping out the waveforms that the human ear doesn't process.

MP4 is short for Moving Picture Expert Group-4.

considered the gold standard for all types of streaming and broadcast applications online

You should generally indicate the size of the mp3 or mp4 file on your web page, next to the link for it.

Embeding audio-clips:

Embedding an audio file on a web page means that the viewer's sound controller will be loaded on the page, at the time that the browser loads the page.

Midi, wav, and mp3 files can all be embedded on a web page.

Will not validate for XHTML:

<embed src="beethoven.mid" autostart="false" loop="false" width="20%" height="20" />

autostart="false" - means that the midi file will NOT start playing automatically right after the page is loaded. This is generally a good idea. The viewer must click the start button on the controller for the file to play.

loop="false" - means that the midi file will not keep playing continuously after the viewer clicks start.

width="20%" - means that the controller will occupy 20% of the viewer's screen width.

heigth="20" -means the controller will be 20 pixels high.

Will validate for XHTML:

You can also embed an audio file using just the **object tag**. The object tag **will validate** for XHTML. See the first example on this [page](http://fog.ccsf.edu/~srubin/h4Object.html). Here is the coding:

<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"codebase="http://www.apple.com/qtactivex/qtplugin.cab" width="200" height="16">

<param name="src" value="jungle.mp3" />

<param name="controller" value="true" />

<param name="autoplay" value="false" />

<param name="autostart" value="0" /> <param name="pluginspage"value="http://www.apple.com/quicktime/download/" />

<object type="audio/x-mpeg" data="jungle.mp3" width="200" height="16"> <param name="src" value="jungle.mp3"/>

<param name="controller" value="true" />

<param name="autoplay" value="false" />

<param name="autostart" value="0" /> <param name="pluginurl"value="http://www.apple.com/quicktime/download/" />

</object>

</object>

**Note:** classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B" is necessary for the audio file to play in an IE controller. IE needs a non-standard value to the valid classid attribute, an identifier to load an associated activeX.

1. If you embed an audio file with the embed tag, I suggest that you also provide a link to it. Here is an example of a[linked midi file](http://fog.ccsf.edu/~srubin/beethoven.mid). This will bring up an audio controller outside of the web page, and the midi file will play automatically.

3. For HTML5, the classid tag won't validate. This example's coding will validate for HTML5.

<object type="audio/x-mpeg" data="jungle.mp3" width="200" height="16"> <param name="src" value="jungle.mp3">  
<param name="controller" value="true">  
<param name="autoplay" value="false">  
<param name="autostart" value="0">   
<param name="pluginurl" value="http://www.apple.com/quicktime/download/">  
</object>

4. For HTML5, you can use the **audio tag**, for example:

<audio controls="controls">  
<source src="jungle.mp3" type="audio/mpeg" />  
<source src="jungle.ogg" type="audio/ogg" />  
Your browser does not support this audio  
</audio>

I needed to convert jungle.mp3 into an ogg file for use with this tag. The OGG Converter (allows a Free Trial, but costs 19.95 to buy it).  
  
Converts MP3, WAV, WMA, OGG from one to another  
Supports Resampling of MP3, WAV, WMA, OGG Vorbis

You can also download a [free audio converter](http://www.freemake.com/free_audio_converter/).

5. For HTML5, the embed tag will not validate. You should use the audio tag or the object tag.

Mpeg Files:

MPEG, which stands for Moving Picture Experts Group, is the name of family of standards used for coding audio-visual information (e.g., movies, video, music) in a digital compressed format. The major advantage of MPEG compared to other video and audio coding formats is that MPEG files are much smaller for the same quality. This is because MPEG uses very sophisticated compression techniques. MPEG file extensions can be .mpg, .mpeg, or .mpe.

MP4 is a multimedia container format standard specified as a part of MPEG-4. It is most commonly used to store digital video and digital audio streams, especially those defined by MPEG, but can also be used to store other data such assubtitles and still images.  MPEG-4 allows streaming over the Internet. A separate hint track is used to include streaming information in the file. The only official filename extension for MPEG-4 files is**.mp4**.

Avi Files:

AVI stands for Audio Video Interface. AVI is used in Windows operating systems to provide sound and video.

The file extension for AVI files is .AVI.

Quicktime Files:

-Apple software for creating, playing and streaming digital media for Mac OS and Windows.

Ext: .qt or .mov

WMV Files:

-Windows Media Video (WMV) is a generic name for the set of video codec technologies developed by Microsoft.

Video version 9 codec and submitted it to SMPTE for standardization.

The video stream is often combined with an audio stream of Windows Media Audio and encapsulated in Advanced Systems Format files, carrying the .wmv or .asf file extensions. WMV files are played by players such as MPlayer or Windows Media Player, the latter being only available for Microsoft Windows and Macintosh systems.

Embedding Video Clips:

Embedding a video file on a web page means that the viewer's associated video controller will be loaded onto the page, at the time that the browser loads the page.

<embed src="julia.qt" height="140" width="160" autostart="false" loop="false" controller="true" />

autostart="false" means that the quicktime file will NOT start playing automatically right after the page is loaded. This is generally a good idea. The viewer must click the start button on the video controller for the file to play.

height="140" - means the controller will be 140 pixels high.

width="160" - means that the controller will be 160 pixels wide.

loop="false" - means that the quicktime file will not keep playing continuously after the viewer clicks start.

controller="true" - This adds user controls to the movie.

 A very common error in the failure to display the controller for a video file is not declaring the width and height to be large enough.

 If you are going to embed a video file I suggest that you also link to it.

 You can embed a video file using just the object tag. The object tag will validate for XHTML

 For HTML5, you can use the video tag, for example:

<video width="200" height="150" controls="controls">  
<source src="chimp.mpeg" type="video/mpeg">  
<source src="chimp.ogg" type="video/ogg">  
Your browser does not support the video tag.  
</video>

I needed to convert chimp.mpeg into an ogg file for use with this tag. The OGG Converter (allows a Free Trial, but costs 19.95 to buy it).  
  
Converts MP3, WAV, WMA, OGG from one to another  
Supports Resampling of MP3, WAV, WMA, OGG Vorbis

 The HTML5 <video> element also has methods, properties, and events.

There are methods for playing, pausing, and loading, for example. There are properties (e.g. duration, volume, seeking) that you can read or set. There are also DOM events that can notify you, for example, when the <video> element begins to play, is paused, is ended, etc.

 For HTML5, the embed tag will not validate. You should use the video tag or the object tag. If you use the object tag, you must use the type attribute.  
  
For example:  
  
<object height="136" width="155" type="video/qt">  
<param name="src" value="julia.qt" />  
<param name="controller" value="true" />  
<param name="autostart" value="false" />  
  
<object type="video/quicktime" data="julia.qt" height="136" width="155" class="qt">  
<param name="controller" value="true" />  
<param name="autostart" value="false" />  
Error text.  
</object>

You can also use the Quick Time Object (QTObject) for embedding video files. It works cross browser and cross platform.

Java vs JavaScript

Actually, the 2 languages have almost nothing in common except for the name. Although Java is technically an interpreted programming language, it is coded in a similar fashion to C++, with seperate header and class files, compiled together prior to execution.

Java has been generating a lot of excitment because of its unique ability to run the same program on IBM, Mac, and Unix computers.

A Java applet is a program written in the JavaTM programming language that can be included in an HTML page, much in the same way an image is included. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM).

Java Applet Tag

For a java applet to work on a web page, it must already have been compiled. A compiled java applet normally has a file extension of class.

Here is an example of a simple applet tag:

<applet code="myApplet.class" width="100" height="140">

</applet>

Here's a more complex example of an APPLET tag:

<applet code="flame.class" width="250" height="110">

<param name="text" value="Bungle+in+the+Jungle" />

<param name="link" value="http://javaboutique.com" />

</applet>

This tells the viewer or browser to load the applet whose compiled code is flame.class. The first param statement sets the applet's "text" attribute (which customizes the text this applet displays) to be "Bungle+in+the+Jungle". A second param statement will allow the applet to also act as a link, if one clicks the text at the bottom of the applet. If the page is viewed by a browser that can't execute Java applets, then the browser will ignore the APPLET and PARAM tags, displaying only the HTML between the <param /> and </applet> tags (the alternate HTML).

Object Tag

Note that the applet tag is valid for Transitional XHTML, but not for Strict XHTML because the applet tag has been deprecated. Instead you can use the object tag, which is valid for Transitional and Strict XHTML

<!-- first object tag is for Firefox -->  
<object classid="java:flame.class" type="application/x-java-applet" height="200" width="200">  
<param name="text" value="Bungle+in+the+Jungle" />

<!-- 2nd object tag is for IE -->

<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" height="200" width="200">   
<param name="code" value="flame" />  
<param name="text" value="Bungle+in+the+Jungle" />  
</object>   
</object>

The **classid** attribute identifies which version of Java Plug-in to use.

For Firefox - classid="java:classname.class"  
  
(where classname.class should be replaced by the actual name, e.g.,   
classid="java:flame.class"  
  
  
For IE - classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"  
(always use this classid for IE, but note that it may not display the applet in some versions/platforms for IE)  
 **Note: I suggest that you upload your class file and all associated folders and files directly into the directory that contains your hw5.html file.**

For HTML5, the applet tag and the classid attribute are not valid. Instead, you should use the object tag with a param statement that has name="code" and value equaling the class name of the applet.

Here's a valid HTML5 example:

<object type="application/x-java-applet" height="200" width="200">  
<param name="text" value="Bungle+in+the+Jungle">  
<param name="code" value="flame.class" >  
</object>

Frames:

A frame is a division of a browser window which displays a distinct HTML page. A browser window can be divided into multiple frames, with each frame displaying a different HTML page.

Frames are used to help users easily find their way through tangled or difficult-to-navigate content (i.e. multiple chapters in an online book with numerous hyper-referenced footnotes). Frames provide a means for a persistent navigation strip to be available to a user at all times, while providing additional areas for display of content, whether primary or ancillary.

Downside:

Sub-pages within a frameset may not be bookmarked,

Due to the multiplicity of pages involved in a single frame site, the number of simultaneous hits received by a server for the first frame page is greatly increased, causing problems for sites with heavy usage

With a three-frame frames page, for instance, the server receives four hits, instead of one hit as with a regular HTML page, quadrupling traffic for that first page;

Search engines tend to catalog sub-pages within a frame site, bringing up sub-pages for users without a frameset or any context. There is a workaround for this problem, but it is an awkward one.

Many inexperienced web programmers (and some experienced ones) use frames to "keep" or "trap" their users within the confines of their frameset; this has become one of the biggest evils in contemporary web design. Marketing people, especially, wish to "brand" the web within a frame containing their company logo, forcing users to surf within this restricted web browser window space from which there is no escape.

Any link which leads out of your web site MUST erase any framesets in use.

Conclusion:

One should use frames only when absolutely necessary and avoid using them whenever possible.

Inline Frames:

An inline frame (iframe) cannot be viewed on all browsers. Internet Explorer and Firefox are two of the browsers that handle inline frames correctly.

<iframe> tag allows you to create a frame that can appear anywhere within a standard html document

used:

utilizing page sidebars

highlighting related external elements

you can target links to open other pages in the iframe

You must create a separate html document to contain the info tha will be placed in the inline frame

<iframe src="ifra.html" width="20%" height="70" align="left" name="ifra"></iframe>

The align tag allow you to set the justification for the frame within the core html page, just as if you were aligning an image.

The name attribute allows you to name the frame. This name allows you to target elements, such as a new page or an image, in the <a href....> statement on the current page.

How to put html code to appear in a frame? ie an image:

<a href="croc.jpg" target="ifr">clicking here</a>

frameborder attribute determines if a border of the frame will be visible and if so what size (“0” is invisible)

Note: For browsers that do not support inline frames, you should place text after the iframe tag, but before the /iframe tag, to tell the viewer that their browser does not support inline frames.

Example:

<iframe src="ifra1.html" width="300" height="150" align="right" name="ifr">  
<!-- Here's what would show using a browser that does not support inline frames -->  
Your browser does not support inline frames  
</iframe>

CSS 2 Notes:

CSS-P (or Cascading Style Sheets Positioning, a part of the unevenly implemented CSS2 standards) allows the web

programmer to position HTML-formatted resources anywhere they like on a web page.

 replaces tables as the primary means for page layout in HTML.

 CSS-P provides programmers with a means for creating application-like screens from positioned "layers" which, in combination with JavaScript, can be used to create simple "dynamic" HTML content (or DHTML).

NOTE: Where to place it?

you must use CSS-P only within the "splash" region of your HTML page (within the first 300-400 pixels in height, and 500-700 pixels in width) so that the user won't have to scroll to see the CSS-P-based content

Only one type of HTML tag should normally be positioned: the DIV tag.

DIV tag was originally created to mark a section division within a larger HTML page

ANY type of HTML content may be placed WITHIN the DIV tag which we're positioning.

 the positioned DIV tag will ONLY have CSS-P properties assigned to it; you won't be assigning regular text-formatting CSS properties to a DIV tag.

Text-formatting properties will only be assigned to the P tag, the heading tags, etc, as usual.

In the future, you will be able to cut out the middleman and assign CSS-P properties directly to ANY HTML tag, positioning that tag.

The DIV tag requires one attribute: ID. This ID will be set to a unique identifying name, and assigned appropriate CSS-P properties in EITHER a linked or embedded style sheet. Note: To ensure cross-browser compatibility, you MUST use an ID to assign CSS-P properties to a DIV tag.

<div id="fred">

<img src="capitalA.gif" width="54" height="54" />

</div>

Positioning:

elements are positioned from THEIR upper-left-hand corner in relation to the upper-left-hand corner of the web page.

#fred {

position:absolute;

left:100px;

top:30px;

}

Example (in context using an embedded style sheet):

<html>

<head>

<title>Example CSS-P Page</title>

<style type="text/css">

#fred {

position:absolute;

left:100px;

top:30px;

}

</style>

</head>

<body>

<div id="fred">

<img src="capitalA.gif" width="54" height="54" />

</div>

</body>

</html>

Minimum of THREE ( 3 ) properties:

The left property represents the x-coordinate value, and the top property represents the y-coordinate value; these properties are ALWAYS set using pixel values.  The position property will always be set to absolute.

ALWAYS define left and top properties using integer pixel-based values.

Position:

Relative is unreliable

Fixed positioning "fixes" that element in the web browser window so that other elements on an HTML page scroll up and down while the "fixed" element remains stationary, stuck in place.

absolutely positioned element is removed from the regular flow of HTML code and positioned in relation to the upper-left-hand corner of the web page

How to get text to wrap in CSS P?

Use width property

Overlap / Overlapping

you need to use the z-index property to set the z-axis (forward and back) value in this flattened three-dimensional space; z-index tells the browser which element goes in front and which element goes behind.

Property: **z-index**  
Values: 0 (zero) or any integer; (negative values are permitted).  
Example: **z-index:10;**

The higher the z-index value is, the farther in front an element becomes; the lower the z-index value is, the farther in back an element becomes. You may not use numbers with decimal places.

I have set fred's z-index to 10, and martha's z-index to 20; martha, therefore, is in front of fred.

If NO z-index value is set, or BOTH z-index values are identical, then whichever DIV element comes LATEST on the HTML page goes in front.

You can turn a positioned DIV tag into a colored box. This feature is particularly useful when creating application "screens" for DHTML purposes. Or

creating low-weight design elements without GIFs.

Width, Height, background-color

#fred {

position:absolute;

left:100px;

top:30px;

width:100px;

height:100px;

background-color:#CC99CC;

border:solid 1px #996699;

}

Last, but not least, you MUST set the border property for the positioned element.

The border MUST be the same color as the background color; this is due to flaws in CSS implementation in older Netscape browsers.

CHILD DIV TAGS

To produce consistent appearance and to control properly the distance between the edge of positioned content and the edge of a DIV box, one must put HTML content into positioned DIV tags which are NESTED INSIDE the positioned, colored DIV box.

<div id="parentDiv">

<div id="childDiv">

<p>Here is some text. And some more text. And so on and so on and so on.</p>

</div>

</div>

"parentDiv" which is 200x200

"childDiv" has been set to be 180 pixels wide.

 "childDiv" DIV tag's positioning is set at 10,10. Child nested elements are positioned in relation to the PARENT element, NOT to the web page as a whole. Therefore, "childDiv" is 10 pixels from the left and 10 pixels from the top of the PARENT element's upper-left-hand corner.

"childDiv" DIV tag has been set to be 180 pixels wide. Since it has been positioned 10 pixels from the left of the parent element, and the parent element is 200 pixels wide, I want to create an even "margin"

MAIN POINT:

because padding, border, and margin properties do not work consistently cross-browser in CSS-P, I have had to create a fake margin by positioning a child element absolutely within an absolutely positioned parent element.

When positioning parent/child elements, there are only a few rules to keep in mind:

1

The parent DIV box MUST be LARGER than the child elements positioned within it. In other words, all of the child elements MUST fit COMPLETELY within the parent DIV box. Again, you must do this in order to ensure cross-browser compatibility; the browsers don't handle overflow of content in a consistent manner.

2

You can NOT nest DIV tags more than ONE level deep!

NOTE

If you use the CSS3 elements of text-shadow and border-radius on a page then you should validate the page for CSS3.

The code for the validation icon:  
  
<a href="http://jigsaw.w3.org/css-validator/check/referer?profile=css3">  
<img style="border:0;width:88px;height:31px"  
src="http://jigsaw.w3.org/css-validator/images/vcss-blue"  
alt="Valid CSS!" />

Reading Notes – Forms

Forms are the means by which users may input information into a web page.

HTML forms are helpless without some sort of script processing them in some manner, whether CGI/PerlPHP, or JavaScript; or by using a mailto action which sends an unformatted email.

Two type of scripts that HTML forms get processed by:

1. CGI/Perl or PHP scripts – process info and write it to database or send it via email to someone (store info)

2. JavaScript scripts – used for calculations or browser-specific activities (ie. changing page locations, etc). (calculate info)

Form tag and Form attributes

Description: this tag marks the boundaries of an HTML form. This tag MUST close. There may be more than one form on an HTML page. Each FORM tag will almost always have just ONE submit button (or simulated submit button) associated with it.  There will NEVER be any FORM tag WITHOUT a submit button (or simulated submit button).

<form>

<!-- one or more form elements, with their accompanying HTML context -->

<!-- a submit button -->

</form>

Two attributes for the FORM tag are used when connecting a form for unformatted email or to a CGI script: METHOD and ACTION.

Method - Description: this attribute defines the manner in which the form information is conveyed to a URL for providing unformatted email or conveyed to a URL containing a CGI script.

<form method="post">

<!-- all form element statements -->

</form>

Action - Description: the ACTION attribute of the FORM tag connects that form to a URL for providing "insecure" email, which means that the form data is NOT encrypted, but it does NOT mean that any harm will come to your computer or to the person's computer receiving the email of the form data.

<form method="post" action="mailto:srubin45@comcast.net"

enctype="text/plain">

<!-- all form element statements -->

</form>

Note: enctype="text/plain" should be included in your form statement when sending unformatted email. It specifies the MIME type of the data to be sent by the post method.

Input tag:

Description: the INPUT tag creates form elements within the confines of a FORM tag. Note: INPUT tags may ONLY be inserted as child elements of a FORM tag; they won't work outside of a FORM tag.

Form elements, such as radio buttons, checkboxes, and submit buttons, are created and inserted into a FORM using the INPUT tag. The INPUT tag doesn't require a closing tag.

Every INPUT tag, however, REQUIRES the TYPE attribute, which tells the browser what type of form element you want.

Attribute: **TYPE**  
Value: **text** (default), **radio**, **checkbox**, **password**, **submit**, **image**, **reset**, **button**, **hidden**, **file**  
Description: the TYPE attribute sets the INPUT tag to become the desired type of form element.  
Example (abbreviated): **<input type="submit" />** (makes a submit button)

Attribute: **NAME**  
Value: any name, NO SPACES IN THE NAME (and following the naming convention rules laid out in previous modules)  
Description: the NAME attribute of the INPUT tag identifies that INPUT for analysis in a script. Most INPUT tags require the NAME attribute.

Attribute: **VALUE**  
Value: any plain text (alphanumeric) information; spaces are permissible although not always recommended.  
Description: the VALUE attribute of the INPUT tag allows you to assign some piece of information to a named INPUT; this information can be retrieved by a script upon submission of the form. (Note that HTML5 does not require the ending / for the input statement).

Example (INPUT tag only):

<input type="checkbox" name="pets" value="dog" />

Checkboxes:

 The INPUT tag creates ONLY the form element itself; it does not create any context for the form element. Context for a form element must be created using ordinary HTML code and text.

<form>

<p>Choose one or more of your favorite pets: <br />

<input type="checkbox" name="pets" value="dog" /> Dog<br />

<input type="checkbox" name="pets" value="cat" /> Cat<br />

<input type="checkbox" name="pets" value="bird" /> Bird

</p>

</form>

Notice how all of the checkboxes in this group have the same NAME, while the VALUEs are all different?

An exception to this rule is when using a php script with checkboxes. There must be NOTHING else on the HTML page which uses that NAME, however; each named element on an HTML page must have a UNIQUE name (a named element being anything using the NAME attribute, regardless of the tag).

Submit Buttons:

<input type="submit" value="Submit Me, Please" />

The VALUE of a submit button is the text which appears on the button itself. Here is the above submit button in context with the set of checkboxes from the previous example:

Radio Buttons:

Radio buttons are coded identically to checkboxes, with one exception: the TYPE attribute must be set equal to **"radio"**.

Radio buttons behave differently from checkboxes in one important respect: the user may check only ONE from a group of radio buttons, whereas they may check as many as they like from a group of checkboxes. That is the essential nature of radio buttons.

It is important that all radio buttons in a group have the same NAME!

Once you've checked all the buttons once, you can't UNCHECK them, nor can you get them to behave like proper radio buttons.

Method Attribute:

You will notice that a question mark (?) has been added to the end of the URL for the current page, and see so-called "name-value pairs" appended onto the end of the URL following the question mark (in the example above, the name-value pairs are **"pets=cat"** and **"colors=red"**).

When a FORM is submitted, the various form elements in the FORM may be converted into name-value pairs derived from the values of their NAME and VALUE attributes, e.g. **NAME=VALUE**. When there is more than one

name-value pair in a FORM, the ampersand (&) character is inserted between the pairs. These name-value pairs may be looked at by a script analyzing a FORM.

The text comprised of the question mark followed by name-value pairs in the web browser's location bar is called a "query string".

get and post:

When a FORM has its METHOD attribute set equal to **"get"** (the default value), clicking on the submit button causes the browser to extract all of the chosen name-value pairs from the FORM; these name-value pairs are then converted into a query string which is appended onto the end of the URL in the web browser's location bar.

When a FORM has its METHOD attribute set equal to **"post"**, the form information is submitted in a manner which does not need further explanation at this time. With **method="post"**, the submission process is invisible to the user. –secure: use with mailto and sensitive data

Input tag: Text and Password

type=”text”

the NAME attribute, and the SIZE attribute; the VALUE attribute is not necessary with a text INPUT field.

Attribute: **SIZE**  
Value: an integer representing the number of characters wide the TEXT INPUT field will be (using the browser's default monospace font, usually Courier or Courier New 10 point).  
Description: The SIZE attribute of the INPUT tag determines the width of a text input field.

<input type="text" size="30" name="fred" />

If you submit this form, you will notice that the name-value pair in the query string is**"firstName=somename"** (where **"somename"** is whatever you have typed). The VALUE of a text INPUT field is whatever the user types into the text field; this is why you don't need to set a VALUE attribute for a text INPUT field.

Attribute: **MAXLENGTH**  
Value: an integer representing the maximum number of characters the user may type into this particular text INPUT field.  
Description: the MAXLENGTH attribute restricts the number of characters a user may type into a text INPUT field.

<form>

<p>Login Name:<br>

<input type="password" size="30" maxlength="30" name="thePassword" /></p>

<input type="submit" value="Submit Me, Please" />

</form>

Remember, the password INPUT field doesn't actually have any real security attached to it beyond obscuring the letters with bullets. Real security requires a secure server connection using the "https" scheme and an appropriate server. HTML forms themselves have NO security built into them.

Checked and Reset

Radio buttons and checkboxes may be pre-checked by using the CHECKED attribute of the INPUT tag.

<input type="checkbox" name="pets" value="cat" checked="checked" />

**Remember**: only ONE radio button may be pre-checked, while as many checkboxes as you like may be pre-checked.

Reset Input:

<input type="reset" value="Clear Me, Please" />

clears info on page (resets to default)

Image and Button:

The IMAGE INPUT allows us to substitute a GIF or JPEG for a regular submit button, while the BUTTON INPUT allows us to create a submit-like button that does nothing at all. The uses for the IMAGE INPUT seem obvious, but the uses for the BUTTON INPUT are only open to those who use JavaScript scripts to add functionality to plain HTML elements.

The image INPUT requires that the TYPE attribute for the INPUT tag be set equal to **"image"**. In addition to the TYPE attribute, you also need to use the SRC, WIDTH, and HEIGHT attributes of the IMG tag.

<input type="image" src="mod8ggo.gif" width="50" height="50" />

To prevent the appearance of the border, you must set the BORDER attribute of the INPUT tag equal to **"0"** (zero).

The image INPUT may NOT be used as a substitute for anything other than the submit button, however.

Button input:

The button INPUT is coded identically to the submit and reset buttons; the only difference is that the button INPUT has no specific function, but acts as a generic button for use in conjunction with a script.

<form>

<input type="button" value="Do-Nothing Button" />

</form>

In addition to the button INPUT, there are two final variations of the INPUT tag, **FILE** and **HIDDEN**. The file INPUT allows the programmer to create a text field which may be used to find and submit files to a server. The hidden INPUT is invisible to the user entirely and has many uses. We will see a good use of the INPUT tag and **HIDDEN** for PHP in assignment 8.

Select and Option tags:

The SELECT and OPTION tags are used to create pulldown menus and multiple-choice scrollboxes for forms

To create a pulldown menu, one must first lay a SELECT tag within a FORM.

The SELECT tag has one important attribute, NAME, which is just like the NAME attribute of the INPUT tag.

<form>

<select name="colors">

</select>

</form>

ou must create one OPTION tag for every desired option in the pulldown menu.

<form>

<select name="colors">

<option>Red</option>

</select>

</form>

If the value attribute is NOT specified, the value of the option is set to the content of the <option> tag

 You may pre-select any one of the other options in the pulldown menu by adding the SELECTED attribute to the appropriate OPTION tag.

Attribute: **SELECTED**  
Value: **SELECTED**  
Description: whichever OPTION tag contains the SELECTED attribute becomes the default choice in the pulldown menu. Note that XHTML requires a value of "selected", whereas HTML requires no value.

<option value="blue" selected="selected">

The VALUE attribute of the OPTION tag does NOT have to be the same value as the text appearing in the pulldown menu itself. This is a particularly useful feature when creating pulldown menus which will be used for navigation on a website.

Scrollbox

 two attributes to the SELECT tag: SIZE and MULTIPLE. The SIZE attribute converts the SELECT tag into a scrollbox, while the MULTIPLE attribute allows the user to choose more than one item at a time.

Attribute: **SIZE**  
Value: an integer representing the number of lines tall the scrollbox will be; this integer should always be at least **"4"** (it may be greater)

<select name="fred" size="4">

Attribute: **MULTIPLE**  
Value: none  
Description: allows a scrollbox to record multiple choices by the user; this functionality requires that the user hold down the "Command" key (Mac) or the "Control" key (PC) while clicking in order to make non-contiguous multiple choices. Holding down the "shift" key while clicking allows the user to choose a contiguous range of multiple choices.

<select name="fred" size="4" multiple="multiple">

Just as with the regular pulldown menu, the SELECTED attribute may be added to any OPTION tag which you wish to make pre-selected; unlike the pulldown menu, more than one OPTION tag may be pre-selected.

<option value="violet" selected="selected">Violet</option>

TextArea tag:

The TEXTAREA tag creates a multiple-line text input field within a FORM; here is an example.

TEXTAREA requires three attributes: NAME, COLS, and ROWS. NAME names the TEXTAREA so that it can be accessed by the script, COLS determines the width of the TEXTAREA, and ROWS determines the height of the TEXTAREA. The TEXTAREA tag always closes.

<form>

<p>Tell me your life story:<br>

<textarea name="lifeStory" cols="40" rows="5">

Type your life

story here…

</textarea></p>

<input type="submit" value="Submit This Puppy" />

</form>

 The VALUE of a TEXTAREA is whatever text is placed between the opening and closing TEXTAREA tags.

HTML 5

The new input types:

* email
* url
* number
* range
* Date pickers (date, month, week, time, datetime, datetime-local)
* search
* color

The new form elements:

* datalist
* keygen
* output

New form attributes:

* autocomplete
* novalidate

New input attributes:

* autocomplete
* autofocus
* form
* form overrides (formaction, formenctype, formmethod, formnovalidate, formtarget)
* height and width
* list
* min, max and step
* multiple
* pattern (regexp)
* placeholder
* required

**NOTE: In XHTML all form attributes are required to have values. In HTML5, attributes that are either 'on' or 'off' can be specified with no value.**  
  
Example:   
  
XHTML: <input type="checkbox" name="email" checked="checked" />  
HTML5: <input type="checkbox" name="email" checked>

CGI NOTES

CGI stands for Common Gateway Interface. All web servers have CGI capabilities. The CGI is the means for web pages to communicate with the server directly, with power to read and rewrite information on the server hard drive, whether in a database or in a more ordinary directory or file.

CGI scripts can be written in any programming language, including C, C++, Java, Perl and PHP.

Perl and PHP preferred.

Security: CGI scripts have the power to rewrite or erase a web server's hard drive,

bin: Web servers have something called the "cgi-bin", which is simply a directory on the server which contains all of the CGI scripts for that server. These CGI scripts may be accessed if you know the URL for the script and its "cgi-bin" directory.

CGI scripts usually come with one of two dot-extensions on the file name, ".cgi" (for a generic CGI script) and ".pl" (for a Perl script specifically).

Static vs Dynamic HTML

 A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

CGI – creates a gateway that can allow people to obtain html pages on your os

Key: CGI program should not take too long to process

How to connect to a script:

Use a FORM tag

Two attributes: METHOD and ACTION

Attribute: **METHOD**  
Value: **get** (default) or **post**  
Description: this attribute defines the manner in which the form information is conveyed to the script.

Attribute: **ACTION**  
Value: a URL (to the CGI script in the cgi-bin in question)  
Description: the ACTION attribute of the FORM tag connects that form to a CGI script; when the submit button for the form is pressed, the web browser seeks out the CGI script at the URL indicated by the value of ACTION and causes the web server to execute that CGI script.

<form method="post" action="myScript.pl">

<!-- one or more form elements, with their accompanying HTML context -->

<!-- a submit button -->

</form>

BOTH method AND action attributes are REQUIRED to connect from a form to CGI script

Perl

Practical Extraction and Report Language

It's original purpose was to monitor large software projects and generate reports. It was initially developed on Unix.

most of the information that users send to servers is text, such as usernames, passwords, and email addresses.

Rules:

All statements must end with semi-colon

perl is case-sensitive

first line: #!/user/local/bin/perl This tells hills to execute the program that follows use Perl (this line is machine dependent)

Comments: Use # for comments

Variables: start with $ example: $in { ‘choice’ }

Print Statement:

print “<H2> dd </h2”;

print <<”\_END\_”; #output the following info until the statement \_END\_; is reached

Debugging:

note: cgiwrap – allows ordinary users to run their own CGI scripts a gateway that allows more secure user access to CGI programs on an HTTPd server than is provided by the http server itself.

Goal: make certain that any CGI script runs with the permissions of the user who installed it and not those of the server

 /cgi-bin/cgiwrap/ (or phpwrap) for urls ending in ".php" or ".pl".

How to run perl interpreter:

 Go to your cgi-bin subdirectory by entering:   
  
cd public\_html/cgi-bin  
perl testerr4.pl (where testerr4.pl is the name of your Perl script)  
  
You should now be able to locate the error in your perl script. In this example, the Perl script had

PHP:

It's official name is PHP: Hypertext Preprocessor, and it is a server-side scripting language. When your Web browser accesses a URL, it is making a request to a web server. When you request a PHP page, for example, http://fog.ccsf.edu/~srubin/formA.php, the Web server wakes up the PHP parsing engine and says, "Hey! You've got to do something before I send a result back to this person's Web browser."  
  
Then the PHP parsing engine runs through the PHP code found in formA.php, and returns the resulting output. This output is passed back to the Web server as part of the HTML code in the document, which in turn is passed to your browser, which displays it to you.

What does PHP do?

Here are some common uses of PHP:

Perform system functions: create, open, read from, write to, and close files on your system; execute system commands; create directories; and modify permissions.

Gather data from forms: save the data to a file, send data via e-mail, return manipulated data to the user.

Access databases and generate content on-the-fly, or create a web interface for adding, deleting, and modifying elements within your database.

Set cookies and access cookie variables.

Start sessions and use session variables and objects.

Use PHP user authentication to restrict access to sections of your web page.

Create images on-the-fly.

Encrypt Data.

HTML + PHP

<form action="formA.php" method="post">

Regarding the form statement, notice that the action's value does not refer to your php directory, because the server will know to automatically look in the php directory for any files having a file extension of .php. You must NOT include the reference to the php directory in the path; the special cgiwrap that Apache uses when it sees the .php extension presumes that directory implicitly. Note that other servers may not require your php files to be in a special sub-directory called php.

If your hw8b.html has as its URL:   
  
http://hills.ccsf.edu/~USERID/cnit132/homework/hw8/hw8b.html  
  
then the php directory is 3 directories up from hw8b.html and your form statement would be:

Your php file must reside in a directory, called php, which is a sub-directory of your public\_html directory.

Be sure to upload your php file in ascii transfer mode.

PERL EXAMPLE:

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

<title>forms page</title>

</head>

<body>

<form method="post" action="test.pl">

<h3>A Simple Form</h3>

<p>Which course was your favorite?</p>

<p>  
<select name="fav">  
<option>CNIT 131</option>  
<option selected="selected">CNIT 132</option>   
<option>CNIT 133</option>   
</select>  
</p>

<p>Which course was your least favorite?</p>

<p><input type="radio" name="least" value="cnit131" checked="checked" />CNIT 131

<input type="radio" name="least" value="cnit132" />CNIT 132

<input type="radio" name="least" value="cnit133" />CNIT 133</p>

<p>Comments/Questions<br />

<textarea name="comments" rows="4" cols="40"></textarea></p>

<p><input type="submit"> <input type="reset" /></p>

</form>

</body>

</html>

Here is the perl script, named test.pl, that actually processes the submitted information from the form and returns the processed information as a web page.

#!/usr/local/bin/perl

do "cgi-lib.pl" ||die "Fatal Error: Can't load cgi library";

#calls subroutine in cgi-lib.pl library

&ReadParse;

# The above line reads the stream of name=value pairs from the form   
# and puts the values in a perl table called in

print "Content-type: text/html\n\n";

# in perl a \n is a "carriage return, next line"   
# NOTE: the text/html indicates the MIME type of  
# information that the   
# script is sending back to the browser thru the Web server

# The items within the braces below refer to the name contents found within the form.

# The next line says to output the following information, until   
# the statement \_END\_; is reached

print<<"\_END\_";

<html>

<head>

<title>Form Page</title>

</head>

<body>

<p>You indicated that your favorite class is $in{'fav'}</p>

<p>You indicated that your least favorite class is $in{'least'}</p>

<p>The value of comments is $in{'comments'}</p>

</body>

</html>

\_END\_  
;

PHP

Your PHP script must have a file extension of .php. Here is the content of my formA.php.

<?php $msg = "E-MAIL SENT FROM FOG\n";

$msg .= "Sender's Name:

$\_POST[sender\_name]\n";

$msg .= "Sender's E-mail:

$\_POST[sender\_email]\n";

$msg .= "Sender's Age:

$\_POST[age]\n"; $msg .= "Message:

$\_POST[message]\n\n";

$to = "srubin45@comcast.net";

$subject = "Form Feedback from Fog";

$mailheaders = "From:

$\_POST[sender\_email]\n";

$mailheaders .= "Reply-To:

$\_POST[sender\_email]\n\n";

mail($to, $subject, $msg, $mailheaders); ?>

<html>

<head>

<title>The following email has been sent to Steve Rubin</title>

</head>

<body bgcolor="#f5f5dc">

<p><b>Your Name: </b>

<?php echo

"$\_POST[sender\_name]"; ?> <br /><br />

<b>Your E-mail Address: </b>

<?php echo "$\_POST[sender\_email]"; ?> <br /><br />

<b>Your Age: </b> <?php echo

"$\_POST[age]"; ?> <br /><br />

<b>Your Message:</b>

<?php echo "$\_POST[message]"; ?>

</p>

</body>

</html>

Explanation of the above code:

The code in formA.php will send an email back to you and and return the form information back to the user's screen.  
  
A. Begin a PHP block and start building a message string:

<?php  
$msg = "E-MAIL SENT FROM FOG\n";   
  
NOTE: The message E-MAIL SENT FROM FOG will be the first line in the message text of the email that is sent and can be changed to anything you like. \n is a carriage return.   
  
  
B. Continue building the message string by adding an entry for the sender's name:

$msg .= "Sender's Name: $\_POST[sender\_name]\n";

NOTE: The .= indicates concatenation. Your variable after $\_POST must match exactly the variable name in your form input text box for sender's name. For example in my HTML form page, I have:  
  
<input type="text" name="sender\_name" size="40">   
  
NOTE: You don't have to use the variable sender\_name in your form or in this php file. What matters is that the names must MATCH.   
  
C. Continue building the message string by adding an entry for the sender's e-mail address, an entry for age, and an entry for the message:   
  
$msg .= "Sender's E-mail:   
$\_POST[sender\_email]\n";   
$msg .= "Sender's Age:  
$\_POST[age]\n";   
$msg .= "Message:   
$\_POST[message]\n\n";   
  
NOTE: The extra spaces before $\_POST are there so that the messages will be better aligned vertically when you are reading the email. The variable names after $\_POST (in this case, sender\_email, age, and message) must match exactly the variable name in your form input text box, radio buttons, and textarea. For Example, in my HTML form page, I have:  
  
<input type="text" size="25" name="sender\_email" />   
<input type="radio" name="age" value="0-20" />Under 21   
<textarea rows="5" cols="20" name="message"></textarea>   
  
D. Create a variable to hold the recipient's e-mail address (substitute your own):  
  
$to = "srubin45@comcast.net";   
  
E. Create a variable to hold the subject of the e-mail:  
  
$subject = "Form Feedback from Fog";   
  
NOTE: You can change the string inside the quotes to anything you like.  
  
  
F. Create a variable to hold additional mailheaders:   
  
$mailheaders = "From: $\_POST[sender\_email]\n";   
  
NOTE: This will put the sender's email address in the from field of the sent email that comes to you. Note also that the variable name inside of the square brackets must match exactly the variable name in your form input text box for sender's email address.  
  
  
G. Add to the $mailheaders variable:  
  
$mailheaders .= "Reply-To: $\_POST[sender\_email]\n\n";   
  
NOTE: the variable name inside of the square brackets must match exactly the variable name in your form input text box for sender's email address.  
  
  
H. Add the mail() function:  
  
mail($to, $subject, $msg, $mailheaders);   
  
I. Close your PHP block:

?>

NOTE: You're not done yet. Although this code will send the email, you should return something to the user's screen so that they know the form has been sent. Otherwise, they might sit there and continually click the Send button.  
  
J. The next 6 lines are normal HTML statements.  
  
K. Add the text label for the Your Name field and display the user's input:  
  
<p><b>Your Name: </b>   
<?php echo "$\_POST[sender\_name]"; ?>   
  
NOTE: sender\_name must match exactly the variable name in your form input text box for sender's name.   
  
<?php echo "$\_POST[sender\_name]"; ?>   
returns to the screen the value of that form name variable.   
  
  
L. Add the text label for the Your E-mail Address field and display the user's input:  
  
<b>Your E-mail Address: </b>   
  
<?php echo "$\_POST[sender\_email]"; ?>   
  
NOTE: sender\_email must match exactly the variable name in your form input text box for sender's email.  
  
  
M. Add the text label for the Age field and display the user's input:  
  
<b>Your Age: </b>   
<?php echo "$\_POST[age]"; ?>   
  
NOTE: age must match exactly the variable name in your form input text box for sender's age.  
  
  
N. Add the text label for the Message field and display the user's input:  
  
<b>Your Message:</b>   
<?php echo "$\_POST[message]"; ?>  
  
NOTE: message must match exactly the variable name in your form textarea for message.  
  
  
O. Add the closing HTML statements.

JavaScript Notes

AKA: ECMAScript

HTML and CSS are DECLARATIVE languages; they declare what something is (HTML) and what that something is supposed to look like (CSS).

 If you actually want something specific to happen on a web page (such as opening a new window or processing a form), you will need to use an IMPERATIVE language, a language that says "do this!"

JavaScript is an "event-driven" programming language, which means that functions in JavaScript are triggered by events.

 What is an EVENT? An event, in JavaScript, is any user interaction with the computer.

 Clicking the mouse is an event.

Pressing a key on the computer keyboard is an event.

Moving the mouse around on the screen is an event.

Loading an HTML page in the browser is an event.

Any action initiated by the user constitutes an event.

RESOURCE is: a resource is any piece of digital media, whether text, sound, graphic, animation, form element, or CSS document, among others.

Resources can include individual pieces of code, functions, windows, dialog boxes, scrollbars, interface elements, etc. Almost anything in the computer, anything digital, can potentially be a resource which you may then access and/or manipulate in JavaScript.

User interacts with the computer 🡪 generates an event 🡪 events are really functions that, when triggered, access and manipulate a resource (ie. User interaction triggers JavaScript program that uses resources to provide user with some FEEDBACK)

USER INTERFACE, the user interface being that portion of the program which the user will interact with that includes both interface elements (such as buttons and text fields) as well as feedback (such as dialog boxes, screens, sounds, etc).

Imperative languages use LOGIC to make simple decisions and perform repetitive actions in a program.

OPERATIONS are used to perform mathematical calculations, comparisons, transfers of data, and other activities within the computer itself.

Data – stored information in VARIABLES

NOTE: JavaScript is case sensitive

Declaration and Initialization

When creating variables in an imperative programming language, there is a two-step process: 1) declaration (declaring a variable), and 2) initialization (initializing a variable).

When you DECLARE a variable, you tell the computer that that variable exists; you also tell it what that variable's name is.

When you INITIALIZE a variable, you fill that variable with data for the FIRST time (initial = first), setting it to its initial, or beginning, state.

How to declare variables in JavaScript?

var george;

Initialize a variable?

George = 1;

Gets Operator is “=”

COMMAND: instruction or set of instructions

NOTE: JavaScript is derived from the programming language called "C".

DATA TYPES

"primitive" data types: numbers, strings, and booleans.

NOTE: JavaScript DOES recognize a difference between integers and floats, and it has a few built-in features which allow you to manipulate numbers as either one or the other.

boolean: false and true, 0 and 1, off and on.

The values of false, 0, and off are all equivalent, while the values of true, 1, and on are also equivalent.

Beyond the primitive data types, and objects, there are two more data types you should be aware of: null and undefined.

The variable that is null contains NO information. You should know, however, that null is NOT 0; Zero (0) is an integer, a number. The null variable is empty.

null variable is differentiated from the undefined variable. An undefined variable has not been declared, and therefore does not exist, or has not been initialized, and so contains no data.

Typed and Untyped Languages

In JavaScript, variables may contain ANY type of data.

You can change the datatype of a variable at will. This is possible because JavaScript is an untyped language

In an untyped language, a variable may be of any data type, and an individual variable's data type may be changed at any time.

In a typed language, a given variable may only ever be ONE data type.

Every object in JavaScript has two main components: a set of PROPERTIES, and a set of METHODS.

Properties are generally passive in nature, describing the state of some feature of the object. Methods, on the other hand, are generally active, performing some sort of task.

constructor merely defines the basic structure of the object;

To get an object that you can use, you need to create an INSTANCE of the object (also known as INSTANTIATING an object).

When you create an instance of an object, you are creating ONE copy of the object which you have control over.

new SomeObject();

these parentheses are called the "function call" operator (which calls/invokes a function), and are an essential part of the object instantiation process.

A FUNCTION, in JavaScript, is a collection of lines of code, a program, which does something (like switch an image, or pop up a new window, or BUILD AN OBJECT, etc).

FUNCTIONS are also OBJECTS themselves; this is because almost everything in JavaScript is built out of objects.

Dot syntax is a way of accessing child elements of a parent object (remember parents and children?) by using period/dot (.) characters between parent and child elements to indicate a path through a complex tree of relationships, much as the slash (/) character is used in URL syntax to indicate a path through a complex directory structure on a web server.

Dot syntax always creates a path from the most general to the most specific element, reading from left-to-right, and, as you can see from the above example, there are NO SPACES IN THE NAMES of the elements or between the dots!

A METHOD is really a FUNCTION which is attached to an object, just as a PROPERTY is really a VARIABLE attached to an object (again, a function is a program which performs some action)!

I could access the image, fred, on my web page using something called the "simplified document object model", which allows me to use dot syntax to point to an element on an HTML page.

window.document.fred.src = picture0.src;

window is the instance of the Window object which represents the current web browser window.

document is the instance of the Document object which represents the current HTML page in the web browser window.

fred is the name of the IMG tag on the HTML page

src is the SRC attribute of the fred IMG tag.

NOTE: image switching / img switching / switch

 all images involved in a switch MUST have the same dimensions, the same width and height!

A VARIABLE may hold an INSTANCE of an OBJECT. Each object has PROPERTIES and METHODS. Properties are essentially variables attached to an object, while methods are essentially FUNCTIONS attached to an object.

JavaScript is an event-driven programming language, where JavaScript commands are triggered by events caused by user interaction.

JavaScript uses special EVENT HANDLERS inserted as ATTRIBUTES into ordinary HTML tags to access user events, such as clicking the mouse.

most event handlers MUST be placed into tags which can ordinarily accept mouse events, such as the A (anchor) tag, or the INPUT tag in a FORM.

<a href="#" onclick="var george = 1;">Link Word</a>

More than one command can be added

<a href="#" onclick="var george = 1; var fred = 2; var answer = george + fred;">Link Word</a>

Window Object

-Autodeclared

-instance refers to current window

<a href="#" onclick="alert('Hi');">Link Word</a>

Attributes may NOT have double-quote marks inside of the double-quote marks beginning and ending the value for the attribute, as this will break your HTML code. Because we NEED quote-marks of some kind to delimit the string value inside the function call operator for the alert method, we MUST use single-quote marks within the bounds of the double-quote marks marking the attribute value. When we write commands within ordinary JavaScript scripts, however, we will be able to go back to using double-quote marks to delineate string values; we'll talk about this later.

You don't have to use the onclick event handler from within an A (anchor) tag. Instead, you can use the javascript absolute URL as the value for the HREF attribute in order to trigger JavaScript commands.

Example:

<a href="javascript:alert('Hi');">Link Word</a>

**Note:** When using the javascript absolute URL, you must NEVER have any spaces in your JavaScript code; otherwise, the javascript URL will fail to operate.

The following example is WRONG because there are spaces in the URL:

<a href="javascript:var fred = 1; alert(fred);">Link Word</a>

If your JavaScript command requires spaces (as in the above example), you MUST then use the onClick event handler rather than the javascript absolute URL.

As I mentioned earlier, you do NOT have to put your onclick event handlers ONLY in A (anchor) tags; you may place the onclick event handler in ANY clickable tag, such as the INPUT tag in a form.

Example:

<form>  
<input type="button" value="Alert Me" onclick="alert('Hi There');" />   
</form>

The main SCRIPT tag for a page usually goes in the HEAD section of your HTML document, although it can also be placed within the BODY section. Between the opening and closing SCRIPT tags, you will write out your lines of JavaScript code.

The SCRIPT tag has ONE attribute, TYPE. The TYPE attribute tells the browser which programming language is being coded inside of the SCRIPT tag.

Note: There are currently 7 (seven) versions of JavaScript

<script type="text/javascript"> // Here's a couple of lines of JavaScript code: var george = 1; alert("Howdy"); </script>

Note: LANGUAGE is an old attribute that is replaced with TYPE

Note: The <!-- right after the script tag and the //--> right before </script> are used to comply with older browsers that do not have javascript enabled.

code inside the SCRIPT tag is executed as the HTML page loads.

An HTML page may, in fact, have several SCRIPT tags on it, both in the HEAD and in the BODY of the document.

JavaScript code may ALSO be coded into an external text file, called a ".js" file

<script src="abc.js"></script>

Note that some scripts won't validate for XHTML unless you include  
  
/\* <![CDATA[ \*/ right after the script statement and /\*]]>\*/ right before the closing script statement.

<script type="text/javascript">  
**/\* <![CDATA[ \*/**  
*// content of your Javascript goes here*  
**/\*]]>**\*/  
</script>

XHTML is subject to the same syntactical rules as XML. Because of this, an XHTML processor treats the characters < and & as markup, even if they reside inside a <script> block.

Follow following format:

<script type="text/javascript">  
**/\* <![CDATA[ \*/**  
*// content of your Javascript goes here*  
**/\*]]>**\*/  
</script>

NOTE: Ensure there is NO SPACE before text/javascript

OPERATORS

+ - = \ \* are operators

1 – n are operands

operators act on operands

WRITE METHOD

You use the write method of the document object to write to an XHTML page.

document.writeln(“msg”);

document.write("<span style = \"color: purple; font-size: 22px;\">XYZ</span>")

NOTE: \ is an escape character

PROMPT METHOD

You can obtain user input by using the prompt method of the window object.

IE DOES NOT LIKE PROMPTS: The problem with using prompts is that IE7 has basically disabled prompts for so-called security reasons. For this reason, I suggest avoiding prompts. If you do use IE7 and get a message to allow scripts and accept it, then you should refresh the page and the prompt will be normally be displayed.

number = window.prompt( "Enter a number", "" );

This will open a popup dialog box that allows the user to enter a string.

This will assign the user input to the variable 'number'.

To repeat, note that the text displayed by prompt dialog boxes can only be plain text, not XHTML formatted text.

CONVERT STRING TO INTEGER

var number = “12”;

var numb = parseInt( number );  //parseFloat

FORM: INPUT OUTPUT

You can obtain user input by using a form input text box, a form button, and the onclick event handler. This is the preferred way for handling multiple input.

<head>  
<script type="text/javascript">  
/\* <![CDATA[ \*/  
function process()  
{  
var number1, number2, n1, n2, sum;  
number1 = document.myform.num1.value;  
number2 = document.myform.num2.value;  
n1 = parseInt(number1);  
n2 = parseInt(number2);  
sum = n1 + n2;  
alert("The sum of the numbers is " + sum); }  
/\*]]>\*/  
</script>  
</head>  
<body>  
<div align="center">  
<form name="myform" action="">  
First Number: <input type="text" name="num1" size="10" />  
<br /><br />  
Second Number: <input type="text" name="num2" size="10" />  
<br /><br />  
<input type="button" onclick="process()" value="SUBMIT" />  
<input type="reset" value="RESET" />  
</form>  
</div>  
</body>

It should be noted that:  
number1 = myform.num1.value;  
works in IE but NOT in Firefox. Firefox REQUIRES it to be  
number1 = document.myform.num1.value;

Another way to obtain a value from a form input text box is to use the getElementById method. Assume that the form input text box is:

<input type="text" id="num1" size="10" />

number1 = document.getElementById("num1").value

PROBLEM WITH document.write() 🡺 usually puts output to a NEW page

SOLUTION: to output the results, using either form input text boxes or a form textarea. For example:

document.myform.result.value = sum;

Sum is: <input type="text" name="result" size="10" />

LOGIC

Computer logic is not like human logic, it's really more of a routing process, or a mechanization process. In fact, it is comprised of only TWO forms in JavaScript: BRANCH logic, and LOOP logic.

Branch logic allows for simple decisions to be made, based on a CONDITION. A condition can be anything, and is TESTED, returning a value of either true or false (a boolean!). IF STATEMENTS

There are really only three kinds of loops, with many, many variations.

Loop Type 1: Repeat an action WHILE, or for as long as, a condition is true.  
Loop Type 2: Repeat an action FOR a particular length of time, or FOR a given number of repetitions.  
Loop Type 3: Repeat an action until something occurs to stop it

for (myCounter = 0; myCounter < 3; myCounter++) {

sortShrimpIntoBasket();

}

Note: the local variable being declared here does NOT require the var keyword.

BASIC FUNCTION SYNTAX

function doStuff() { }

function

name of the function

list of parameters

NOTE: watch out for carriage returns. They could be interpreted as ;

Call functions from Event Handlers or javascript absolute url:

<a href="#" onClick="doStuff();">Link One</a>

<a href="javascript:doSomethingElse();">Link Two</a>

**Local variables** are temporary. A local variable is declared, initialized, and used for a short period of time, then thrown away. A local variable in JavaScript can only be used within a single function; no other functions have access to that particular variable; when a function is finished executing, the local variables for that function are thrown away.

**Global variables** are persistent. A global variable is created when your web page is loaded, can be accessed by many different processes and functions, and lasts until your web page is unloaded from the browser window.

**PROTECTED NAMESPACE – local variables can have same names in diff methods**

Global variable names start with the

Local variable names start with my

Functions PARAMETERS

function doStuff(fred) {

alert(fred);

}

function doStuff(fred, george) {

alert(fred);

alert(george);

}

window.open("http://www.yahoo.com/", "marian", "width=500,height=400");

1. The first argument is the URL (absolute or relative) that will appear in the new window.

2. The second argument is the NAME of the new window (which will NOT be visible to the user, but rather identifies the window in JavaScript).

3. The third argument is a series of parameters, describing the appearance of the new window; this third argument may NOT contain any spaces, nor will it contain any additional quotation marks.

To get a new browser window with more features, I have to state ALL of the features that I want, explicitly, in the third argument.

window.open("http://www.yahoo.com/", "marian","top=100,left=300,width=500,height=400,resizable,scrollbars,menubar,status,location,toolbar");

The window.open() method RETURNS a reference to the new browser window ITSELF.

JavaScript has a long list of event handlers which it understands. Some of the most commonly used event handlers are:

onclick

onmouseover

onmouseout

onchange

onload

onfocus

onblur

event handlers would be added to an HTML tag, and would be triggered upon that event occuring to the tag.

In order to support older browsers, most event handlers must be placed within clickable tags, such as A (anchor) and INPUT.

onClick, onMouseOver … will not validate for XHTML

Note:

When a page is completely loaded, when all of its graphics and text and sound files and movies are fully downloaded, the onLoad event is triggered. The onLoad event handler ALWAYS goes in the BODY tag of the HTML page.

When you SELECT something on the computer, you are giving that thing FOCUS. When you DESELECT something, you are giving it BLUR.

CREATING METHODS:

 The location property of the window instance of the Window object gives us access to the location bar, and will advance the page in the web browser when assigned to the desired URL.

window.location = "http://www.mozilla.org/";

There is a special keyword of the SELECT tag, selectedIndex, which returns the number of the option which the user has selected.  
  
Example:  
  
myForm.destList.selectedIndex

myForm.destList.options[myForm.destList.selectedIndex].value

var mydest = myForm.destList.options[myForm.destList.selectedIndex].value;

**Load-time errors**: An error that violates the grammatical rules of the language when the script is loaded.

**Run-time errors:** A run-time error occurs when the JavaScript interpreter encounters a problem during the execution of a program.

The most common type of run-time error occurs when you try to access an object or variable that doesn't exist.

Firefox:

Click Tools/Error Console/Errors to find JavaScript errors in Firefox. The line number and approximate position of the error is displayed. Note that you can also check for CSS errors by clicking Tools/Error Console/Warnings.

IE:

status bar in the lower left corner of the browser shows an error icon or says 'error on page' or 'done but with errors', the gray error box may appear or you may need to click on the error icon at the lower left of the page to see the JavaScript error and what line contains the error. For testing purposes, I suggest you have "Disable script debugging" as UNCHECKED, under Tools/Internet Options/Advanced on your browser.

Chrome:

In Chrome, click the "Control this page" button to the right of the address bar and select Tools/Javascript console. Then click on the error link at the right of the page to see which line has the error along with the corresponding error message.

Safari:

In Safari, you must enable the Develop menu. To do so, choose Edit/Preferences and then click the Advanced tab. There is a check box entitled "Show develop menu in menubar" that should be checked. Once the setting is enabled, a menu named "Develop" appears in the Safari menu bar. The Develop menu provides several options for debugging and otherwise working with the page that is currently loaded. You can click Show Error Console to display a list of JavaScript and other errors.

jQuery is a JavaScript library intended to make JavaScript programming easier and more fun.

A JavaScript library is a complex JavaScript program that both simplifies difficult tasks and solves cross-browser problems. In other words, jQuery solves the two biggest headaches with JavaScript—complexity and the finicky nature of different web browsers.

**jQuery Syntax**  
  
The jQuery syntax is tailor made for selecting HTML elements and perform some action on the element(s).  
  
Basic syntax is: $(selector).action()  
  
A dollar sign to define jQuery  
A (selector) to "query (or find)" HTML elements  
A jQuery action() to be performed on the element(s)

**The Document Ready Function**  
  
All jQuery methods are inside a document.ready() function:  
$(document).ready(function(){  
  
// jQuery functions go here...  
  
});  
  
This is to prevent any jQuery code from running before the document is finished loading (is ready).

Event handlers are method that are called when "something happens" in HTML. The term "triggered (or "fired") by an event" is often used.