ATSC 212

HTML

Section Overview

What is HTML?

HTML stands for **HyperText Markup Language**. It was originally designed so that military and academic users could publish electronic documents in a more readable format.

With the changes to the internet came changes to the publishing demands of its users. Nowadays, internet publishing involves more than documentation and includes the use of pictures, animation, and sound. It is no longer just about sharing text documents but a media unto itself. Despite that, HTML still forms the basis for most web development today.

Webpage Basics

HTML documents in their simplest form are simply text documents with some special tags to tell browsers how to interpret and display the text.

All HTML tags surround sections of text or files enclosed in **<>** brackets. Most HTML tags have a beginning tag and an ending tag which will be the same as the beginning tag but include a **/** before the tag name. For example, <html> and </html> are tags that indicate an HTML document.

Webpage Basics

All HTML documents begin with **<html>** and end with **</html>**. These tags tell the browser that it should interpret the document as an HTML document. The only text that can occur outside these tags are comments.

In HTML, single line comments can be declared like this

<! YOUR COMMENT GOES HERE >

Multiple line comments can be declared like this

<!– YOUR COMMENT
GOES HERE -->

Webpage Basics

Once you have a the document tags in place, there are two primary sections to define for your document: the head and the body. The head is specified using the **<head></head>** tags and the body using the **<body></body>** tags.

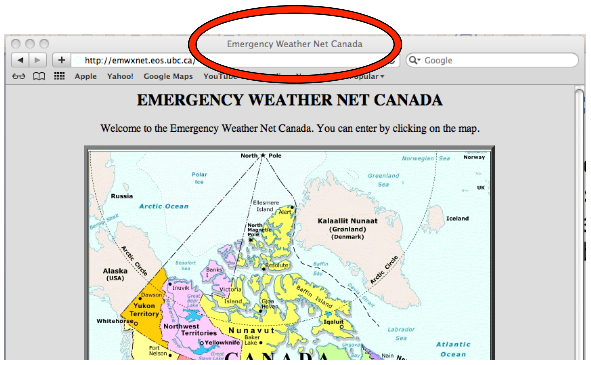
The head of the document normally contains information about document style, imbedded scripts, the title, and other meta information. This is normally where style sheets and javascript go.

The body contains the actual document contents.

Webpage Basics

There are a lot of things that can be specified in the head, but we will only focus on two tags; **<title>** and **<style>**. The title tag allows you to specify the document title which appears in the browser title bar.

<title>Emergency Weather Net Canada</title>


Emergency Weather Net Canada
http://emwxnet.eos.ubc.ca/
Google
Apple
Yahoo!
Google Maps
EMERGENCY WEATHER NET CANADA
Welcome to the Emergency Weather Net Canada. You can enter by clicking on the map.

Webpage Basics

The style tag allows you to set characteristics for how the webpage looks (ie fonts, colors, background). We will examine this in more detail in the Style Sheets section. So far, our standard webpage looks like:

```
<html>
  <head>
    <title>YOUR TITLE</title>
  </head>
  <body>
    STUFF
  </body>
</html>
```

Webpage Basics

Although we have not come across it yet, it is worth noting that many tags can also be supplied with attributes that affect how the tag operates. For example

<font **face="Tahoma" size="+3"**>…stuff…</font>

will specify that everything between the tags should have the **Tahoma typeface** and be **three sizes larger** than the browser default.

Most of the tags we see from now on will have attributes that you will need to set.

Formatting

Anything we add to the body will appear in the webpage itself. We can just type in whatever we want, but if we do not specify the layout, the browser will use its defaults to decide how the page looks. We can give information to the browser about how we would like the content to appear.

Changing fonts is the most common formatting change. We can specify a particular typeface, text size, and color using the font tag.

<font **size="-2" face="Sans" color="#FF0000"**>

Formatting

Bold facing type can be done using the **<b>** flag.

<b>THIS IS BOLD</b>

Italics are done via the **<i>** flag.

<i>Italicized</i>

Underlining is done via the **<u>** flag.

<u>Underlined</u>

Formatting

Keep in mind, these changes are only suggestions to the browser. If the user has the browser override websites, or the browser does not support a particular format (like a typeface), it will ignore the formatting and go with a default.

There are pre-defined header tags which can be used to format headings for different sections of the webpage. These tags are **<h1>**, **<h2>**, ... **<h6>**. <h1> indicates a top level header while subsequent numbers indicated nested section headers. Typically they affect the size and boldness of the test.

Formatting

Different sections of text can be separated into paragraphs using the **<p>** tag. This tag has attributes for justifcation (**align="center|left|right"**) and width in pixels (**width="250"**). By separating sections into paragraphs, we can have text displayed as paragraphs onscreen.

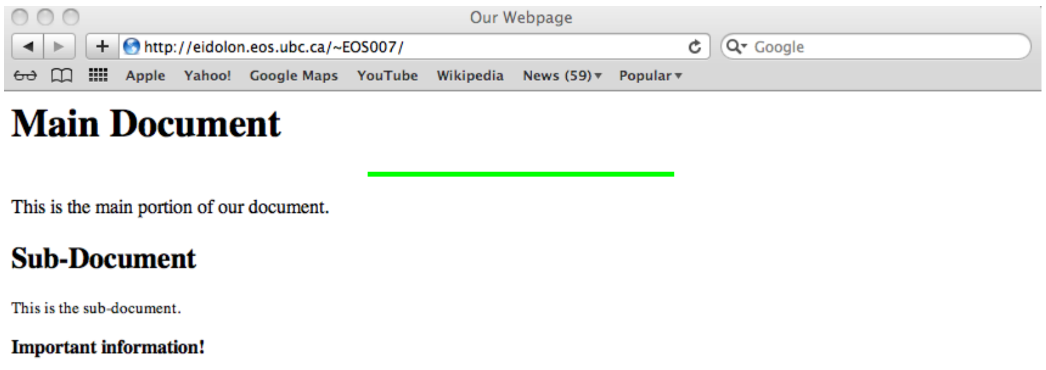
There are lots of other formatting tags, but we will cover only one more of note, the **<hr>** tag. This tag has no closing </hr> tag, does not enclose text, but instead indicates to the browser to draw a horizontal line on the screen. It takes attributes for justification (**align**), color (**color**), thickness (**size**), and width on the screen in pixels (**width**).

Formatting

Example:

```
<html>
  <head>
    <title>Our Webpage</title>
  </head>
  <body>
    <h1>Main Document</h1>
    <hr color="#00FF00" size="4" width="250">
    <p>This is the main portion of our document.</p>
    <h2>Sub-Document</h2>
    <p><font size="-1">This is the sub-document.</font></p>
    <p><b>Important information!</b></p>
  </body>
</html>
```

Formatting


Our Webpage
http://eidolon.eos.ubc.ca/~EOS007/
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (59) Popular

Main Document

This is the main portion of our document.

Sub-Document

This is the sub-document.

Important information!

Formatting

The **<br>** tag acts like a carriage return. Browsers do not recognize returns within the document as formatting for the webpage, so you need to specify when you want a line to be separate. With paragraphs, this is most easily done using the <p> tag, but if you just want to put a return between any two lines, use <br>.

Fonts

We mentioned the **<font>** tag earlier, let's take a look at it in more detail. This tag is used to suggest the font that the browser should use to display text contained between the beginning and ending font tags. If the browser does not support some aspect of the font, it will use its default setting.

The main attributes of the <font> tag are **color**, **face**, and **size**. The color attribute and accepts a #RRGGBB string and sets the color of the font. We will look at the color string more closely in the next section.

Fonts

The size attribute determines how large the text should be. You can set it to a number from 1 (smallest) to 7 (largest) or you can specify an offset from the default (ie +1 for one size larger or -2 for two sizes smaller). Most browsers have a default of 3.

The face attribute determines the typeface for the browser to use. This is a string of one or more comma delimited typefaces. Common examples are Helvetica, Sans Serif, Times New Roman, Tahoma, and Arial. The browser will normally attempt to display using the first typeface in a list, going to each subsequent typeface if not supported.

Colors

Many tags in HTML have an attribute which will take a color code and use this to set a background, foreground, border, or text color. The standard color format is #RRGGBB.

The color strings always start with #. Beyond that, you have pairs of hexadecimal numbers representing the amount of red (RR), green (GG), and blue (BB) within the color. Hexadecimal numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F (taking values from 0 to 15). Therefore the number pairs represent a value from 0 (00) to 255 (FF). The higher the value, the more a color appears.

Colors

For example, #FF0000 is pure red, #00FF00 is pure green, and #0000FF is pure blue.

The colors are determined by color adding, and while any combination is valid, most monitors cannot display subtle differences nor can most people see them. (ie #F00000 looks exactly like #F10000).

Most commonly used or 'safe' colors are created through combinations of the digits 0, 3, 6, 9, C, and F.

Colors

Here are some example colors.

#000000 – white	#666600 – moss
#660000 – dark red	#CCCC00 – mustard
#CC0000 – red	#FFFF00 – yellow
#FF0000 – bright red	#006666 – dark aqua
#006600 – dark green	#00CCCC – aqua
#00CC00 – green	#00FFFF – cyan
#00FF00 – bright green	#660066 - dark purple
#000066 – dark blue	#CC00CC – purple
#0000CC – blue	#FF00FF - fuschia
#0000FF – light blue	

Style Sheets (CSS)

Adding all these different formatting tags to your document makes it a bit cluttered and difficult to deal with. For that reason, **style sheets** (usually referred to as cascading style sheets or CSS) were introduced.

Style sheets let you define formatting for various elements of your webpage once so that you can easily apply them later, like a theme in a word or powerpoint document. This removes clutter from your document and ensures consistency in appearance. You can override style sheet definitions using the formatting techniques we have already discussed.

Style Sheets (CSS)

There are four ways you can add style sheet information to your webpage.

1. You can add the style sheet within your <head> section.
2. You can link or import the style sheet from a separate file.
3. You can apply style sheets to specific sections.
4. You can inline style definitions.

Both points 3 and 4 are achieved by using the style attribute within different tags.

Style Sheets (CSS)

You can add a style sheet to your <head> section by using the **<style>** tags. Anything that appears between the <style> and </style> tags forms the style sheet for the webpage. The <style> tag does not take any attributes.

Nowadays, most browsers properly interpret the <style> tag and style sheet information, but for portability to older browsers and to ensure complete compatibility, the style sheet is usually listed as a comment between the <style> tags. (ie <style><!-- ... --></style>)

Style Sheets (CSS)

If you put the style sheet definitions between the style tags, it will apply to the webpage. However, if you are designing a website with many pages sharing the same style or having alternative site styles, storing the style sheet in separate files makes it easier to share and improves site consistency. Style sheet files simply contain the style definitions normally found in the commented section and have the extension .css.

To import a style sheet file, simply add **@import url (‘…’);** between your style tags (in the commented section to be safe). The URL can be absolute or relative to the webpage file.

Style Sheets (CSS)

Alternatively you can link the style sheets into your webpage. Linking can give you the option of choosing between different style sheets for different purposes (ie one style sheet for viewing and another for printing).

Linking is achieved with the **<link>** tag. This tag has three attributes you will need to set: **rel**, **type**, and **href**. rel should be set to ‘StyleSheet’. type should be set to ‘text/css’. Together, these attributes tell the browser what type of file is being linked. href specifies the URL for the file, absolute or relative. Unlike importing, <link> tags do not have to appear between <style> tags but can be used anywhere within the webpage.

Style Sheets (CSS)

However, there is an important caveat to importing or linking multiple style sheets. If the various files contain style definitions for the same element(s) (ie making all paragraph text a particular color), the then last style definition in the webpage will be the one the browser will use.

Now let's look at how we create style definitions. For portability, our style sheet files will have <style> tags and html commenting around our definitions. A **definition** consists of **selectors|classes { declarations }** .

Style Sheets (CSS)

Selectors are HTML elements and often correspond to HTML tags which have a style attribute (ie P is the selector for the <p> tag and would apply to all <p> tags). **Classes** refer to a particular style element, such as a color, and can belong to either a selector (ie P.red) or the general document (ie .red).

Declarations are specifications of the properties of that selector or class (ie P { color : red } would make the text of all <p> tagged sections red). Declarations are always pairs of properties and specifications separated by a colon. Multiple declarations can be made within the {} provided the are separated by semicolons.

Style Sheets (CSS)

Before delving too deep into types of declarations, it is worth noting that some declarations require a measure, some distance on the screen. In these cases, you can supply set measures (ie 2px) or percentages (ie 90%).

The standard measures are **em**, **ex**, **in**, **cm**, **mm**, **px**, **pt**, and **pc**. An em is the width of a capital M in the used typeface. ex is the height of a lowercase x in the used typeface. in is inches. cm is centimeters. mm is millimeters. px is pixels. pt is points which is a typographic measure equal to 1/72 of an inch. pc is picas which is also a typographic measure, equal to 1/6 of an inch.

Style Sheets (CSS)

It is worth noting that real measures (in, cm, mm, pt, pc) do not relate well to screen space, but are very useful in specifying how the webpage will print. Vice-versa, screen measures (em, ex, px) do not relate well to real space, but are very useful in specifying how a webpage will appear in a browser.

Colors also can be specified in multiple ways.

1. #RRGGBB format specified earlier.
2. Standard color (ie red, green, blue)
3. rgb(R, G, B) where R, G, B are numbers 0-255
4. rgb(R, G, B) where R, G, B are percentages 0%-100%

Style Sheets (CSS)

Time to take a closer look at selectors. You have already seen tags like <body>, <p>, and <h1>. Soon you will see anchor tags (<a>), table tags (<table>), and list tags (<ol>, <ul>). All of these tags can be selectors. Typically they are listed as selectors using the tag name in uppercase. For example:

```
BODY { font-size : x-large ; color : blue}
```

would make the font size of all text in the document extra large (about 20 pixels) and the color of all text blue. Most any tag we discuss can be a selector.

Style Sheets (CSS)

Now that we know about selectors, let's look at declarations. A declaration is a combination of a property/attribute and a value (most all tag attributes can be in declarations). One of the most common things to set is the font. Font properties include **font-family**, **font-style**, **font-variant**, **font-weight**, and **font-size**.

font-family is like a typeface (ie Sans Serif, Times New Roman). font-style can be normal, italic, or oblique. font-variant is rarely used but can specify small-caps. font-weight specifies the boldness (normal, bold, bolder, lighter, or 100, 200, ... 800, 900). Finally font-size goes from xx-small, x-small, small, medium, large, x-large, xx-large or a number in pixels.

Style Sheets (CSS)

Although using font declarations will do a lot to change the appearance of text, there are more text properties you can declare: **word-spacing**, **letter-spacing**, **text-decoration**, **vertical-align**, **text-transform**, **text-align**, and **text-indent**.

Word-spacing and letter-spacing are declared as measurements and do what you would expect, space out words or letters by the specified measurement.

Text-decoration has the options underline (like <u>), overline, line-through, and blink. All the options are self-explanatory, but blink should be used sparingly.

Style Sheets (CSS)

Text-indent is a measurement or percentage that specifies the amount to indent.

Text-align can be right, left, or center justified.

Vertical-align can be baseline, super, sub, top, text-top, middle-bottom, or text-bottom. Super and sub are the most useful a they allow for superscripts and subscripts.

Finally, text-transform can be capitalize, uppercase, or lowercase.

Style Sheets (CSS)

Outside of text, color and background are the most commonly set properties. The possible declarations are **color**, **background-color**, **background-image**, **background-repeat**, and **background**.

Color lets you set the color of something, usually text, and can be specified in any of the ways mentioned previously (ie P { color : rgb(255, 0, 255) }).

Background can be used to set background-color, background-image, and background-repeat, although they can also be done separately.

Style Sheets (CSS)

Background-color is specified in the same way as color and will alter the background color of whatever selector you have chosen.

Background-image allows the background to show an image, instead of a single color, specified by **url('…')**. If the background area is smaller than the image, only part of the image will be shown. If the background area is larger, the entire image will be shown and may be tiled to cover the area (depending on the browser default). Background-repeat allows you to specify that the image be tiled to cover the entire background (or up to a set number of times).

Style Sheets (CSS)

We will be looking at anchors and links shortly which are created using the <a> tag. You can style links using three different selectors; **A:link**, **A:active**, and **A:visited**. A:link is used to alter the appearance of typical, untaken links. A:active is used to alter the appearance of a link currently in use. A:visited changes the appearance of a link that has been clicked.

You can alter the link appearance in any ways we discuss later in the links section, but the most common use is to change the color of the link to make it stand out. Most browsers have a default of blue for links and active links, and a default of purple for visited links.

Style Sheets (CSS)

The last thing we will cover are declarations for lists. Later on we will see that we can create ordered lists (<ol>) and unordered lists (<ul>). There are four properties we can set specific to lists: **display**, **line-height**, **list-style-type**, and **list-style-image**.

Display can be set to inline, block, or list-item and controls how the list displays. Inline makes each element display on its own line without an index marker. Block and list-item organize the elements vertically and on some browsers do nothing.

Line-height takes a measurement and sets each bullet to be that high.

Style Sheets (CSS)

Finally, list-style-type and list-style-image control how list markers appear.

List-style-image allows you to specify an image via **url (‘...’)** that acts as the list marker.

List-style-type allows you to choose a pre-set marker from disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, and none. Some types are for bullets (ie disc, circle, square), while the others are for ordered list markers.

Images

Adding images or video to your webpage adds flair and is simple and you can do it via the **<img>** tag. The tag has several important attributes: **align**, **alt**, **border**, **controls**, **height**, **hspace**, **ismap**, **loop**, **src**, **start**, **usemap**, **vspace**, and **width**.

Height and width specify the size the image appears as in pixels. If height and/or width are not specified, the browser will default to the actual size of the image.

Hspace and vspace are used to produce a margin of whitespace in pixels around the image (horizontally and/or vertically).

Images

The align attribute allows the image to be inlined (for text to wrap around it). align can be LEFT, RIGHT, TOP, MIDDLE, and BOTTOM (where the image appears on the screen relative to the text).

The alt attribute allows you to give some text description that appears in lieu of the image in browsers that do not support displaying images.

The border attribute specifies the width of the border around the image in pixels. Normally, no border is displayed around images (so border=0 is not necessary). The color of the border is the same as the surrounding text.

Images

Ismap and usemap indicate that the image is to be treated as a clickable map object. We will not cover map objects, but essentially they are places on the webpage which can be moused over and clicked with results depending where on the area the click occurred. Ismap requires that map object be handled by the server (so an anchor will need to wrap the image). Usemap specifies a URL that the browser uses to determine how to handle the map.

Images

Controls, loop, and start are all used for video. Controls ensures that playback controls are made available for the video (otherwise the user will have no control over playback).

Loop indicates the number of times the video will playback before stopping. Loop=INFINITE will continually play the video over and over. By default, most browsers will play a video once.

Start indicates when the browser should start playing the video. The options are FILEOPEN (which begins play once the file is downloaded) or MOUSEOVER (which plays when the user mouses over the video).

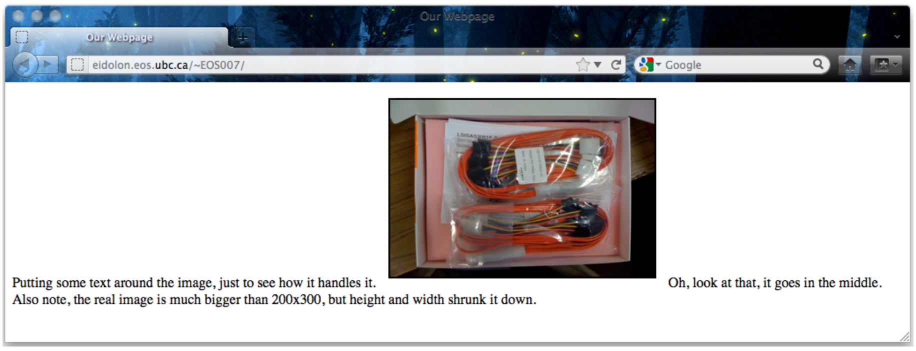
Images

The final attribute, src, is also the most important. This is the URL where the image or video file is located. The URL can be relative to the webpage containing the <img> tag or absolute (the complete path to the file on the website). Let’s take a look at a couple examples...

ATSC 212 – HTML

Images

Putting some text around the image, just to see how it handles it.
<img src="IMG_0014.JPG" border=2 hspace=10 vspace=10 height=200 width=300>
Oh, look at that, it goes in the middle.<br>
Also note, the real image is much bigger than 200x300, but height and width shrunk it down.

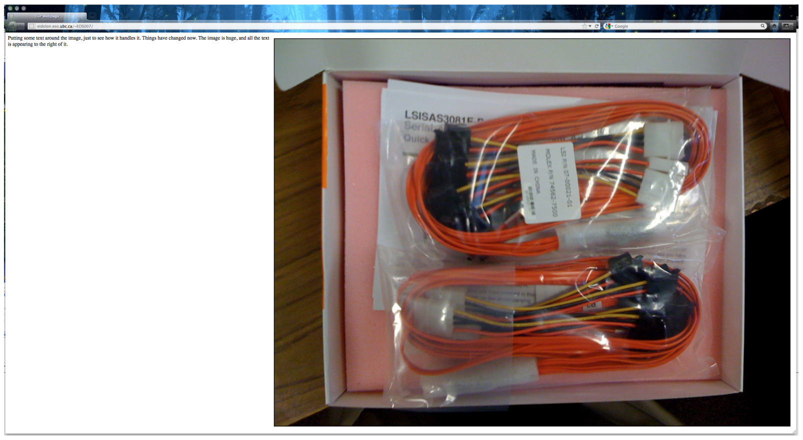


Putting some text around the image, just to see how it handles it. Oh, look at that, it goes in the middle.
Also note, the real image is much bigger than 200x300, but height and width shrunk it down.

ATSC 212 – HTML

Images

Putting some text around the image, just to see how it handles it.
<img src="IMG_0014.JPG" border=2 hspace=10 vspace=10 align=RIGHT>
Things have changed now. The image is huge and all the text is appearing to the right of it.



Putting some text around the image, just to see how it handles it. Things have changed now. The image is huge, and all the text is appearing to the right of it.

Anchors and Links

One of the primary elements of webpages is the ability to go directly to another spot within the same document or to a different webpage altogether. When going to a spot within the same document, the element is called an **anchor**. When going to a different webpage, it is called a **link**. For simplicity, and because the approach to both behaviours is achieved with the same tag, the **<a>** tag, we will hereafter refer to them as links.

The tag has several important attributes you will use: **href**, **name**, **title**, and **coords**. There are others, but these will give you the key functionality.

Anchors and Links

The **title** attribute allows you to specify text that will appear as a tooltip when someone mouses over the link.

The **name** attribute gives you a way to identify the link within a document. This is the key to creating links to other places within the same document.

The **coords** attribute allows you to specify a clickable area for the link. Normally, it would be text in the tags, but you can create a link over an image or area of the document using coords.

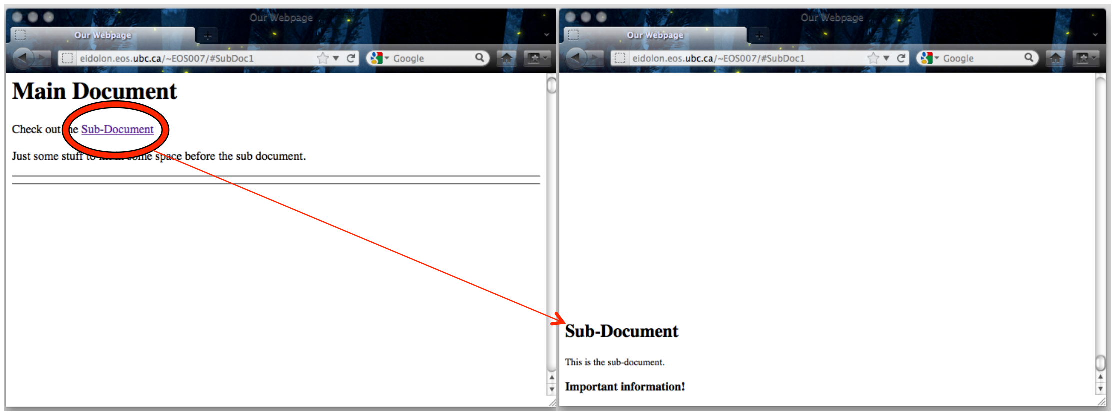
Finally, **href** allows you to specify where to go.

Anchors and Links

Example:

```
<h1>Main Document</h1>
<p>Check out the <a href="#SubDoc1">Sub-Document</a></p>
<p>Just some stuff to fill in some space before the sub
    document.</p>
<hr>
<hr>
...add 500 <p>&nbsp;</p>...
<h2><a name="SubDoc1" title="Sub-Document">Sub-Document
      </a></h2>
```

Anchors and Links


Our Webpage
Our Webpage
eidolon.eos.ubc.ca/~EOS007/#SubDoc1
Google
Our Webpage
Our Webpage
eidolon.eos.ubc.ca/~EOS007/#SubDoc1
Google

Main Document

Check out the Sub-Document

Just some stuff to fill in some space before the sub document.

Sub-Document

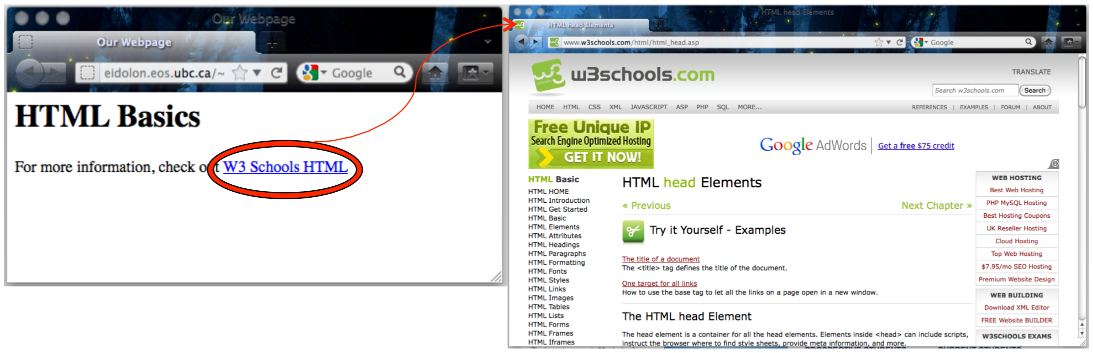
This is the sub-document.

Important information!

Anchors and Links

Example:

```
<h1>HTML Basics</h1>
<p>For more information, check out
    <a href=www.w3schools.com/html>W3 Schools HTML</a></p>
```



HTML Basics

For more information, check out W3 Schools HTML

HTML head Elements

Try it Yourself - Examples

The title of a document
The <title> tag defines the title of the document.

One target for all links
How to use the base tag to let all the links on a page open in a new window.

The HTML head Element

The head element is a container for all the head elements. Elements inside <head> can include scripts, instruct the browser where to find style sheets, provide meta information, and more.

Tables

Organizing data on your webpage is key to making it readable and recognizable. A useful element for this is the table.

Tables are rows and columns of data built up cell by cell within row by row. There are three tags used for building tables: **<table>**, **<tr>**, and **<td>**.

The **<table>** tag indicates a table is being started. The key attributes we will consider for this tag are **align**, **background**, **bgcolor**, **border**, **bordercolor**, **cellpadding**, **cellspacing**, **cols**, and **width**.

Tables

Align and width work as they do with other tags, allowing you to specify how the table aligns itself within the document and how wide it appears.

The **background** attribute allows you to specify a URL to an image that will appear as the background to the table. If you prefer for the table to have a solid color instead, you can use the **bgcolor** attribute which takes a “#RRGGBB” color. If neither attribute is specified, the table will have the same background color or image as the rest of the document.

Tables

The **border** attribute allows you to specify the thickness of the cell borders in pixels and **bordercolor** lets you specify the color of the borders. If you set border=0, then no borders will appear, which is a nice trick for organizing data within a document using a table without it appearing as a table.

Cellpadding determines how much space, in pixels, should go between a cell’s contents and its border. **Cellspacing** specifies the number of buffer pixels between cell borders. These two attributes can be used to make a table appear less crowded.

Tables

Finally, **cols** allows you to specify the number of columns in a table. Normally, tables will have a number of columns equal to the maximum number of cells in any of its rows. This attribute ensures cells appear, particularly where there is no data.

Tables

The **<tr>** flag is used to add a row to the table. It has **align**, **bgcolor**, and **bordercolor** which override the same attributes within the table for that row. It also has a couple other useful attributes: **nowrap** and **valign**.

Nowrap prevents text from wrapping within cells to form new lines. This forces each cells' contents to be displayed as a single line.

Valign allows you to specify the vertical alignment within cells of that row to be either at the 'top', in the 'middle', or at the 'bottom'.

Tables

Finally, the **<td>** flag is used to create individual cells within the table. It is between the <td> and </td> flags that table content goes. This flag also has **align**, **background**, **bgcolor**, **bordercolor**, **nowrap**, **valign**, and **width** attributes that work as described for the other tags you have seen. These override the same flags as specified in either <tr> or <table>. In addition, it has two other useful attributes: **colspan** and **rowspan**.

Colspan allows you to create a cell which covers multiple columns within the row (appearing as a single cell). **Rowspan** allows you to create a cell which will cover multiple, subsequent rows within a single column.

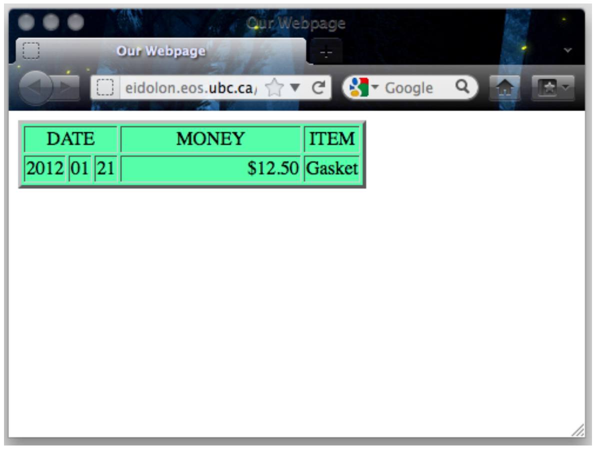
Tables

Example:

```
<table bgcolor="#55FFAA" border=3 cols=5>
  <tr bordercolor="#FF0000" align="center" valign="bottom">
    <td colspan=3>DATE</td>
    <td width=150>MONEY</td>
    <td>ITEM</td>
  </tr>
  <tr bordercolor="#00FF00" valign="middle">
    <td>2012</td>
    <td>01</td>
    <td>21</td>
    <td align="right">$12.50</td>
    <td>Gasket</td>
  </tr>
</table>
```

Tables

Notice how the table only extends as large as it needs to be for the contents (not across the entire document) with the exception of the MONEY column where we specified the width of the header cell (which adjusted the entire column).



DATE			MONEY	ITEM
2012	01	21	$12.50	Gasket

Tables

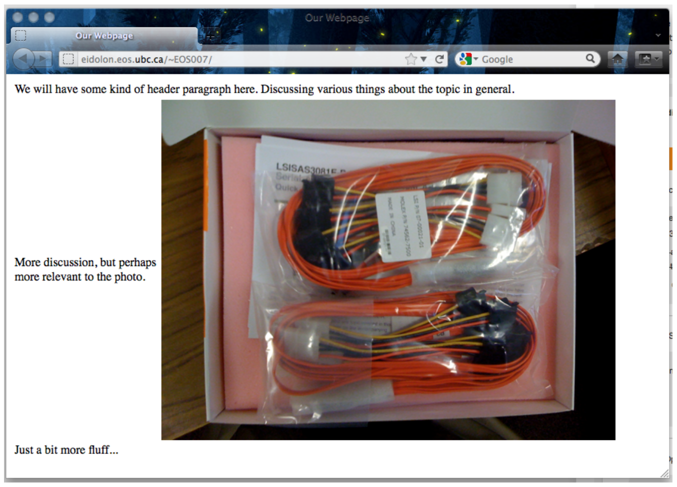
Example:

```
<table border=0 width=800>
  <tr>
    <td colspan=2>We will have some kind of header paragraph here.
                  Discussing various things about the topic in
                  general.</td>
  </tr>
  <tr>
    <td>More discussion, but perhaps more relevant to the
         photo.</td>
    <td><img src="./IMG_014.JPG" width=600></td>
  </tr>
  <tr>
    <td colspan=2>Just a bit more fluff...</td>
  </tr>
</table>
```

Tables

We will see the <img> tag later, but as you can see, the table invisibly organizes the page when the border=0.

Also notice that the table has a set width of 800 pixels, so it covers that much of the browser window. With the width of the image set to 600, it limits the text to the left.



We will have some kind of header paragraph here. Discussing various things about the topic in general.

More discussion, but perhaps more relevant to the photo.

Just a bit more fluff...

Lists

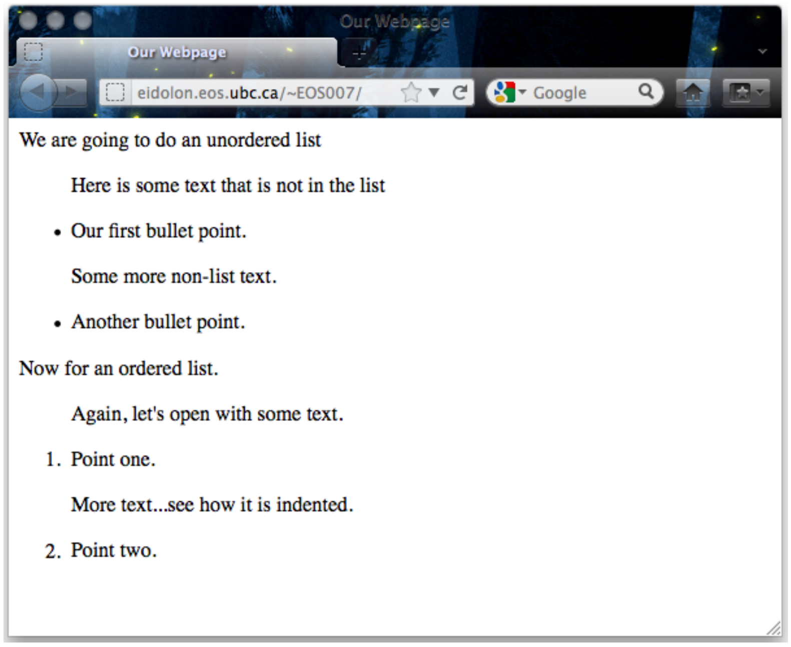
Although we could use a table and some fancy formatting to make a nicely appearing list in a webpage, there is an easier way. We can create ordered lists (ie 1. ..., 2. ...) using the **<ol>** tag and unordered lists (ie bullet points) using the **<ul>** tag.

We add elements to the list between the opening and closing tags using the **<li>** tag. Anything that appears between <li> tags will be added to the list. Anything between <ol> or <ul> tags that is not between <li> tags, will not be added to the list, but will be indented as much as the list elements. Let's see an example.

Lists

```
<p>We are going to do an unordered list</p>
<ul>
  <p>Here is some text that is not in the list</p>
  <li>Our first bullet point.</li>
  <p>Some more non-list text.</p>
  <li>Another bullet point.</li>
</ul>
<p>Now for an ordered list.</p>
<ol>
  <p>Again, let's open with some text.</p>
  <li>Point one.</li>
  <p>More text...see how it is indented.</p>
  <li>Point two.</li>
</ol>
```

Lists



We are going to do an unordered list

Here is some text that is not in the list

- Our first bullet point.

Some more non-list text.

- Another bullet point.

Now for an ordered list.

Again, let's open with some text.

1. Point one.

More text...see how it is indented.

2. Point two.

Lists

Most of the properties for lists can be set with style sheets as we have seen. However, for ordered lists, if you wish the list to begin at something other than the default (ie 1, A, a), then you need to use the **start** attribute. For example

```
<ol start=4>
```

will begin an ordered list whose first element is 4 (subsequent elements are 5, 6, ...).

Forms

Thusfar, we have seen how we can organize and format data in webpages, but we cannot interact with someone viewing the webpage. They have no means of input. Forms provide a framework for interactive content. The form instructs the browser on what information to provide and how to provide it, as well as telling the server that serves the webpage what to do with the information it receives back.

Forms are begun with the **<form>** tag. Important attributes for this tag are **action** and **method**. It will also take attributes like name which function as they do for other tags.

Forms

The **action** attribute specifies a URL that processes the data sent by the form. This typically will be a script on the server which may return a webpage once the data is processed. This attribute is what allows data to be passed from forms.

The **method** attribute can be either POST or GET and tells the browser how to send the data to the server. Unless you want to control the specific method for data transfer (which can be useful at times), then it is not important to specify this attribute (the browser will simply choose its default).

Forms

Between the form tags, we need ways of collecting information. We use the **<input>** tag to do this. This tag does not have an end tag (</input>) and encapsulates one type of data. Important attributes include **checked**, **disabled**, **maxlength**, **notab**, **readonly**, **src**, **tabindex**, **type**, and **value**. It also takes attributes like name, align, and size which works as defined for other tags. However, we will say more about name shortly.

The **type** attribute is the most important. It defines the the kind of input and how it will appear. Acceptable types are **text**, **password**, **radio**, **checkbox**, **submit**, **reset**, **image**, **file**, **hidden**, and **button**.

Forms

Most of these types are obvious. **Text** defines a text box that people can type anything into. **Password** queries for an encrypted password. **File** allows for any generic file to be uploaded. **Radio** and **checkbox** are types of pre-defined choices. **Submit**, **image**, and **reset** are the buttons you normally see that allow you submit your data or reset the form to defaults. **Image** works like submit except you can specify how the button appears. **Button** allows you to define a selectable button that indicates data should be sent (pressed) or not (not pressed). Finally the **hidden** type allows you to send data of which the user is unaware. This is a useful way to carry data from form to form or page to page without the user having to constantly re-send it.

Forms

The **checked** attribute is only relevant to radio and checkbox inputs and specifies that the particular input is selected by default.

The **disabled** attribute prevents that input from being shown or used (depending on the browser). This can be useful when going through multiple inputs which rely on previous entries which may make particular inputs invalid.

The **maxlength** attribute defines the number of characters you can enter into a text or password type. This is not the same as the size attribute which affects how big the text box will appear on the screen.

Forms

The **notab** and **tabindex** attributes control how you can tab through input selections. notab prevents that input from being selected with the tab key while tabindex determines the order in which inputs will be selected by the tab key.

Readonly is similar to disabled except that the input is shown, a value can be submitted and selected, but it cannot be changed by the user. This can be useful for showing locked-in selections.

Forms

The **value** attribute allows you to specify a default value that is submitted if the input is chosen.

Finally, the **src** attribute works with the type="image" to indicate the image file that should be used to replace the submit button. This attribute is only applicable if type is image, otherwise it does nothing.

The <input> tag, by itself, does not necessarily display information in the form. Anything else we type between the <form> tags will appear as text within the form (and we can format this text using the tags seen so far).

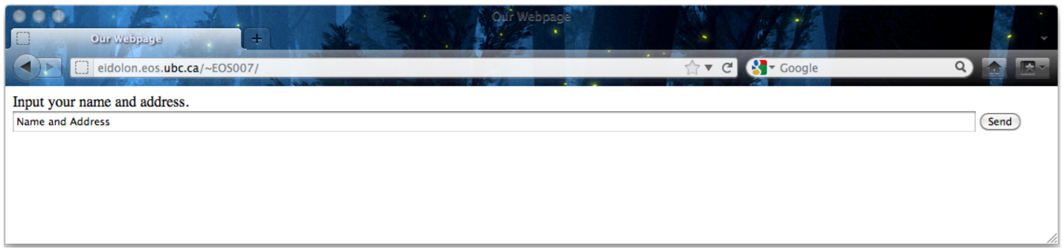
Forms

There is one last caveat and that is the name attribute. Normally we would use this attribute to help define a tag within a document (particularly for linking), however, for forms, the name attribute also defines the name (or parameter) the data will be passed under when submitted. So every input tag should have a name attribute, even though some may be the same name (as in the case of radio buttons or checkboxes).

Let's take a look at an example...

Forms

```
<form method="POST" action="../get_text.pl">
  Input your name and address.<br>
  <input type="TEXT" name="NMA" size="150" maxlength="125"
value="Name and Address">
  <input type="Submit" value="Send">
</form>
```



Input your name and address.

Name and Address

Send

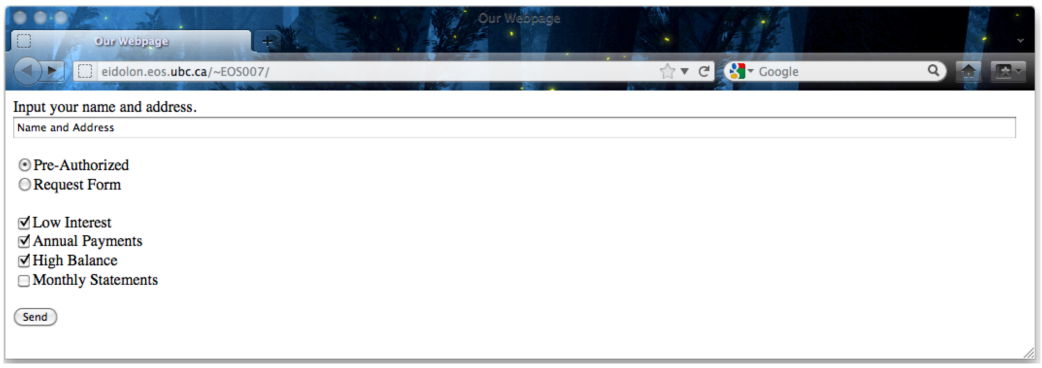
Forms

Here is another example:

```
<form method="POST" action="../get_text.pl">
  Input your name and address.<br>
  <input type="TEXT" name="NMA" size="150" maxlength="125" value="NMA"><br>
  <br>
  <input type="RADIO" name="GetForm" value="PreAuth" checked>Pre-
Authorized<br>
  <input type="RADIO" name="GetForm" value="ReqForm">Request Form<br>
  <br>
  <input type="CHECKBOX" name="Options" value="LowInt">Low Interest<br>
  <input type="CHECKBOX" name="Options" value="AnnualPay">Annual
Payments<br>
  <input type="CHECKBOX" name="Options" value="HighBal">High Balance<br>
  <input type="CHECKBOX" name="Options" value="Monthly">Monthly
Statements<br>
  <br>
  <input type="Submit" value="Send">
</form>
```

Forms

Notice that only one radio button can be clicked but multiple checkboxes can be selected. This is the primary difference between these two types of input.



Input your name and address.
Name and Address

● Pre-Authorized
○ Request Form

☑ Low Interest
☑ Annual Payments
☑ High Balance
☐ Monthly Statements

Send

Dynamic Webpages

Now that you have seen how to add forms to webpages, how do you deal with that data? How do you create an interactive experience for the user?

Thusfar we have been creating .html files, static content that the browser interprets directly. However, it is possible for a web server to send html content to a browser based on a script. Using forms, the browser can send data to the web server which can be used in a script to dynamically generate new web content which can be sent to the browser.

Dynamic Webpages

There are many approaches to dynamically generating web content. Pre-compiled code (such as code in C) or scripted code in languages like PERL or PHP can be executed by the web server when instructed by the browser (via a link). This pre-compiled or scripted code can parse form data and then generate new web content on the fly and send it back to the browser.

Learning how to script dynamic web content could be a course unto itself, but we will take a look at some basic elements using PHP.

Dynamic Webpages

PHP can be inserted directly into HTML or can be scripted entirely on its own. Whether inserting a section of script into a webpage or creating a stand alone file, we enclose PHP code with

```
<?php
...
?>
```

Typically, files containing PHP scripts are given the extension .php to let the server know that it needs to use the PHP interpreter to handle the script.

Dynamic Webpages

PHP is a functional language with a lot of predefined data structures, so most of what you do is make function calls. Two very useful functions are **header(...)** and **echo(...)**.

Both functions take a string as an argument. header() takes the string and embeds that string into the <head> section of your document. Thus, whatever string you put into header should match the usual things you would see in that section.

echo() simply sends the string to the browser (like a print to browser command). You can use echo to compose HTML directly.

Dynamic Webpages

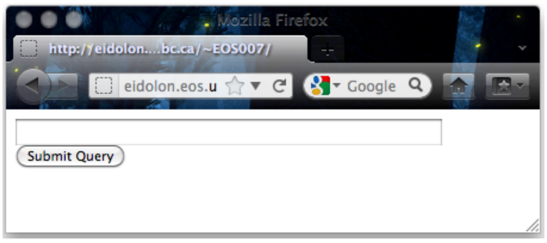
Getting at the data from forms is handled through some predefined data structures called **$_GET**, **$_POST**, and **$_REQUEST**. All variables and data names begin with a **$**.

Recall that in forms you can specify the method that the data is sent from the browser. Data sent using method GET ends up in the $_GET data structure, while data sent using the method POST ends up in the $_POST data structure. These two data structures are dictionaries, or hashes. They are indexed using the <input> name attribute. For example, $_POST[“NMA”] will get the value sent by the input named NMA.

Dynamic Webpages

Let’s take a look at an example. Here is a webpage with a simple textbox form that will accept typed data.

```
<html>
 <body>
  <form method=POST action=“handleform.php”>
   <input type=TEXT name=“info” size=50><br>
   <input type=SUBMIT>
  </form>
 </body>
</html>
```



Dynamic Webpages

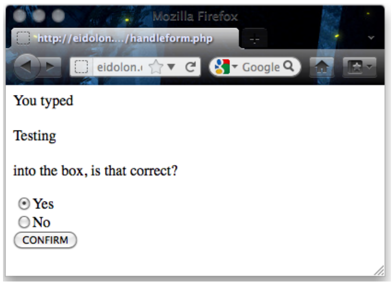
Now let’s create a PHP script (called “handleform.php”) that reads that data and echoes it on a new webpage.

```
<?php
  $textbox = $_POST[“info”];
  echo(“<html>\n <body>\n”);
  echo(“  <p>You typed<br><br>$textbox<br><br>into the box, is that
correct?</p>\n”);
  echo(“  <form method=POST action=\”confirmed.php\”>\n”);
  echo(“   <input type=HIDDEN name=\”info\” value=\”$textbox\”>\n”);
  echo(“   <input type=RADIO name=\”confirm\” value=\”YES\”
checked>Yes<br>\n”);
  echo(“   <input type=RADIO name=\”confirm\” value=\”NO\”>No<br>
\n”);
  echo(“   <input type=SUBMIT value=\”CONFIRM\”>\n”);
  echo(“  </form>\n”);
  echo(“ </body>\n</html>\n”);
?>
```

Dynamic Webpages

Here is what our script is doing. First it is getting the data from our textbox and storing it in the variable $textbox. It does this by going to the $_POST data structure (since we used the POST method in our form) and looking up ‘info’ which was the name of our text box.

After that, it creates an HTML page using echo that includes a radio button form for confirming the text entered.



You typed

Testing

into the box, is that correct?

● Yes
○ No
CONFIRM

Dynamic Webpages

A couple things of note regarding handleform.php. First, notice the radio button form in this new page is designed to call a different PHP script called confirmed.php. The form also passes the contents of the previous textbox along in a hidden input still called 'info'.

Every argument supplied to echo() was a string which meant that if we wanted quotes to appear within our string we had to escape them using the backslash.

Finally, newlines were added to the end of every echoed line, but they are not required. Browsers disregard newlines, so they are only included to make it easier to view the source for testing purposes.

AND THAT IS IT…

This barely scratches the surface. A quick search online will yield a multitude of information regarding HTML and PHP. HTML has many great pages, although a good way to get an idea how HTML looks is to view the source of pages you visit on the web. This can usually be done by right clicking within a page and then looking for an option like 'View Source'.

For PHP, I recommend the developers site, www.php.net.