Week One: XHTML Markup

Content: the collective term for all of the *text, images*, video, sound, animations and downloads

XHTML (eXtensible Hyper Text Markup Language): the *markup code* defines the document's *structure* through *tags* that identify each element of your content

CSS (Cascading Style Sheets): enable you to *define* how each marked up element of your content is *presented* to the user

LEGACY CODE

- ***Tables:*** were designed for laying out grids of data not to divide the page into sections
- ***Font Tags:*** will eat up bandwidth and can be replaced by style sheets

** Whispering Example **

XHTML DEFINED

- XHTML is a reformulation of HTML written with the syntax of XML
- XHTML is based on the free-form structure of XML where tags can be named for the content that they contain
- Correctly written XHTML markup gives you the best chance that your pages will display correctly in a broad variety of devices for years to come

Well Formed Markup: markup code that is *structured properly* according to the rules of XHTML

Valid Markup: means the markup contains only XHTML, with *no meaningless tags*, tags that are *not closed properly* or *depreciated*

XHTML RULES!

1. **DOCTYPE:** the DOCTYPE is written before the opening `<html>` tag and informs the browser whether the page contains HTML, XHTML or a mix of both. Without a DOCTYPE, many browsers will go into *Quirks Mode,* a backwards compatibility mode.

   Strict: all markup is XHTML compliant

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
   ```

   Transitional: a mix of HTML & XHTML

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   ```

2. **XML Namespace:** points the browser to where it can find the DOCTYPE

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

3. **Content Type:** states what character (set) coding was used for the document

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

4. **Lowercase Tags:** all tags in XHTML are written in lowercase

```
<html>
```

5. **Closing Tags:** XHTML requires that you close every tag

```
<p>Close your tags.</p>
```

6. **Nesting Tags:** all tags in XHTML must be nested properly to work correctly

```
<p>Nest your tags <strong>correctly</strong>.</p>
```

7. **Inline Tags:** inline tags cannot contain block level tags.

8. **Attributes:** all attributes must have values and must be quoted.

```
alt="Your Value Needs to be Quoted"
```

9. **Entities:** an entity is a short string of characters that represents a single character

```
<p>Pees &amp; Carrots</p>
```

XHTML ANATOMY

1. **DOCTYPE**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

2. **HTML Tag**

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

3. **Head Tag**

```
<head>
```

4. **Title Tag**

```
<title>Premium Design Works - WEB120 &ndash; Web Authoring II</title>
```

5. **Meta Tag(s)**

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

6. **Link Tag(s)**

```
<link href="WEB120.css" rel="stylesheet" type="text/css" />
```

7. **Script Tag(s)**

```
<script src="http://www.premiumdw.com/validateEmail.js" language="JavaScript"
type="text/JavaScript"></script>
```

8. **Closing Head tag**

```
</head>
```

9. **Body tag**

```
<body>
```

10. **Image Tags**

```
<img src="images/flameLogo.gif" alt="Premium Design Works" />
```

11. **Comment Tags**

```
<!-- Begin Copy -->
```

12. **Division Tag(s)**

```
<div id="copy">
```

13. **Headline Tag(s)**

```
<h1>WEB120 &ndash; Web Authoring II</h1>
```

14. **Paragraph Tag(s)**

```
<p><b>Course Description: </b>This course gives an overview of the basic
principles and practices of professional web site design and production via XHTML
&amp; CSS.</p>
```

15. **Closing Division Tag(s)**

```
</div>
```

16. **Closing Body Tag**

```
</body>
```

17. **Closing HTML Tag**

```
</html>
```

Assignment: XHTML Page

Week Two: CSS Anatomy

Types of Styles

Styles: are what control the *presentation* of your *structured* markup

1) **Inline Styles:** are added to a tag using the XHTML `style` attribute.

```
<p style="font-size: 25pt; font-weight:bold; font-style:italic; color:red;">By
adding inline CSS styling to this paragraph, we can override the default
styles.</p>
```

- an *inline style* only affects the tag to which it is attached
- *inline styles* are another way of adding presentational markup directly into your pages
- *inline styles* should be used only in special circumstances
- *inline styles* will override other styles

2) **Embedded Styles:** can be placed within the `<head>` tag of your document and need to be called out with the `<style>` tag.

```
<head>
<title>Inline Styles example</title>
<meta http-equiv="Content-type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="Content-Language" content="en-us" />

 <style type="text/css">
     p {font-variant: small-caps}
 </style>

</head>
```

- *embedded styles* are limited to the page in which they are contained
- sometimes it is easier to write the styles as *embedded styles* before you move them to be a linked style
- *embedded styles* will override linked styles

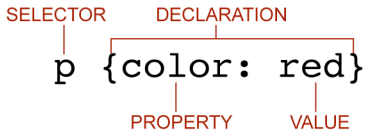
3) **Linked Styles:** are placed in a separate document that links to multiple pages as to control your styles globally

```
<head>
<title>Premium Design Works - WEB120 &ndash; Web Authoring II</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<link href="WEB120.css" rel="stylesheet" type="text/css" />

</head>
```

- *linked styles* are linked to the pages via a `<link>` tag within the `<head>` tag
- *linked styles* are applied to each page's markup as the page loads
- *linked styles* are contained in a text file, called a style sheet, and appended with a `.css`

Style Rules

Style Rules: are *how* your styles *present* your markup



1) **Selector:** states which tag the rule selects

2) **Declaration:** the declaration states what happens when the rule is applied **and** is made up of *properties* and *values*

3) **Property:** states what is to be affected, in this case

4) **Value:** states what the property will be set to

A) **Words:** a word such as bold is a type of *value*

```
font-weight: bold;
```

B) **Numbers:** numerical *values* are usually followed by a unit type

```
font-size: 12px;
```

C) **Colors:** color *values* are most commonly written as hexadecimal values

```
color: #336699;
```

5) **Multiple Declarations:** rules can have more than one declaration

```
p {
 font-family: Arial, Helvetica, sans-serif;
 color: #85898A;
 font-size: 12px;
 line-height: 18px;
 letter-spacing: 0px;
 text-align: left;
 margin-top: 0px;
 margin-bottom: 8px;
 }
```

6) **Multiple Selectors:** can be grouped for a single rule

```
h1, h2, h3 {
 font-family: Arial, Helvetica, sans-serif;
 color: #85898A;
 font-size: 12px;
 font-weight: bold;
 line-height: 18px;
 letter-spacing: 0px;
 text-align: left;
 }
```

- in this case `h1, h2, h3` have all be defined with the same font & color by using *multiple selectors*

7) **Multiple Rules:** can be applied to the same selector

```
h1 {
 margin-top: 10px;
 margin-bottom: 8px;
 }

h2 {
 margin-top: 5px;
 margin-bottom: 8px;
 }

h3 {
 margin-top: 3px;
 margin-bottom: 8px;
 }
```

- however, let's say that you want to give `h1, h2, h3` different margin values
- you would create *multiple rules* to do so

8) **Contextual Selectors:** are selectors that use more than one tag name in the selector

```
p em {
 color:#F20017;
 }
```

```
<p>Our mission is to develop and present your brand to a wide range of clientele
via <em>logo design</em>, <em>marketing collateral</em>, <em>advertising</em> and
<em>dynamic publishing</em> to the world wide web.</p>
```

- with this *contextual selector*, I have made the `<em>` tags within my `<p>` tags a different color and style

Inheritance

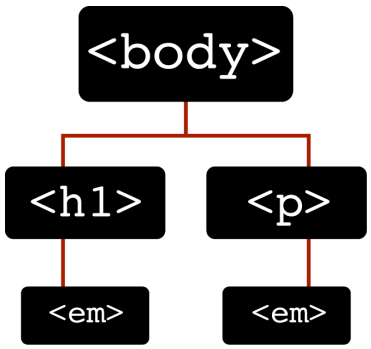
Inheritance: involves passing something down from ancestors to descendants

```
body {
 font-family: Arial, Helvetica, sans-serif;
 color: #85898A;
 background-color: #FFF;
 margin: 0px 0px 0px 0px;
 padding: 0px 0px 0px 0px;
 }
```

- in this example we can see that the `<body>` tag, the top most *ancestor*, will *inherit* the properties listed in the rule

The Cascade

Cascading Rules: cascade down from one level of the hierarchy to the next



- understanding the *cascade* will help you write your *style sheets* in the most economically organized way
- organizing your *style sheets* with the proper *cascade* will also enable them to be edited much more easily

1) **Matching Declarations:** as the page loads, the browser looks at every tag in the page to see if the rule *matches*
2) **Sorting Order:** If a matched property is defined again, the browser will update the value in the *order* it is declared
3) **Specificity:** the rule that is more specific wins

Assignment: CSS Document

Week Three: CSS Text

Fonts & Type Defined

Type: makes the clearest visual statement about the quality of your site's offerings

Fonts: are the digital formats that typefaces come in, each with their own unique characteristics

Font Families: are groups of fonts under a given manufacturers or designer's name; *Times*

Font Faces: are the different type faces within the group or family; *Times Bold*

Text: simply describes a block of type on a page, like a sentence in your body copy

Font Properties: relate to the size and appearance of collections of type; *size, weight*, etc

Text Properties: relate to the font's treatment on your page; *line-height, letter-spacing*, etc

Common Fonts: fonts are read by the browser given that the user has the fonts installed on their computer – p.72

Font Collections: groups of fonts categorized by their general look; sans-serif or decorative – ***p. 71, fig. 3.4***

1) **Serif:** fonts so named because of the added detail to the type's design which makes them easier to read in print
2) **Sans-**Serif: fonts that do not have the added detail to the type's design and have a much more plain appearance making them easier to read on the screen

Font Declarations: it is accepted practice to write a CSS declaration specifying a number of fonts starting with the font you prefer

```
body {
    font-family: Arial, Helvetica, sans-serif;
    color: #85898A;
    }
```

- in this CSS Rule example we are setting a `font-family` for the entire page by way of the `body` selector
- the `font-family` property calls three values to choose from; `Arial, Helvetica, sans-serif`
- the rule will try to load:
  1) Arial
  2) Helvetica (if Arial is not available)
  3) a default sans-serif (if both Arial & Helvetica are not available)
- it is important to use a generic fall-back when specifying fonts because the specified font may not be available

Fonts Properties

Font-Size: you have three types of `font-size` values to choose from

1) **absolute:** values that come in a fixed or absolute size; *pixels, millimeters, inches*
2) **keyword:** values that will be displayed based on their inherit settings; *small, large, xx-large*
3) **relative:** values that are relative to the page or browser settings; *percentages, ems*

Relative Baseline Size: when working with relative units, one must set the baseline size which all relative units will then adhere to

```
body {
 font-family: Arial, Helvetica, sans-serif;
 color: #85898A;
```

```
font-size; 100%;
}
```

- by declaring your `font-size` as `100%` you are saying that your baseline size will be 100% of the default baseline size
- since16 points is the default baseline size, 100% of 16 points is what all relative units will adhere to as the default baseline size

```
h1 {
  font-size: .8em;
  }
```

- in this example we are declaring that `.8em` will be 80% of 16 points – you do the math.

Font-Style: determines whether a font is set to be italicized or not

```
h1 {
    font-style: italic;
    }
```

- in this example we have set the headline to display as `italic` (which makes the typeface lean a bit)
- we also can set it to `oblique` (which attempts to use the oblique version of the typeface) or `normal` (which makes the typeface stand up-right)

Font-Weight: determines the weight or boldness of the typeface

```
h1 {
    font-weight: normal;
    }
```

- since headlines are bold by default you can override the bold setting by using `normal`, as in this example
- other possible value include: `100-900, bold, bolder, lighter` – ***p .85, fig. 3.12***

Font-Variant: determines whether the typeface will be displayed as lowercase or small caps

```
h1 {
    font-variant: small-caps;
    }
```

- by using `normal`, the text will be displayed in lowercase (if written that way)

(Font Property) Shorthand: the `font` property will allow one to use shorthand to declare the properties of the font

```
body {
    font-family: verdana, arial, sans-serif;
    font-size: 1em;
    }
```

```
p {
    font: bold italic small-caps 12pt verdana, arial, sans-serif;
    }
```

1) **values** for the `font-size` and `font-family` must always be declared
2) the **sequence** must be - `weight, style, variant, size, family`

Text Properties

1) **text-indent:** numerical values will give the first line of paragraph text an indent – ***fig. 3.15***

```
p   {
    text-indent: 3em;
    }
```

2) **letter-spacing:** numerical values will give space in-between letter pairs of text – ***fig. 3.18***

```
p   {
    letter-spacing: .2em;
    }
```

3) **word-spacing:** numerical values will give space in-between word pairs of text – ***fig. 3.19***

```
p   {
    word-spacing: .2em;
    }
```

4) **text-decoration:** keyword values will give text a decorative style

```
p   {
    text-decoration: underline;
    }
```

values: `underline, overline, strikethrough, blink`

5) **text-align:** keyword values will align the text of any block level tag – ***fig. 3.20***

```
p   {
    text-align: left;
    }
```

values: `left, right, center, justify`

6) **line-height:** numerical values will give space between lines of text

```
p   {
    line-height: 18px;
    }
```

7) **text-transform:** keyword values will change the capitalization of text within an element – ***fig. 3.21***

```
p   {
    text-transform: capitalize;
```

```
    }
```

values: `uppercase, lowercase, capitalize, none`

8) **vertical-align:** length and keyword values will move type up or down with respect to the baseline – ***fig. 3.22***

```
p   {
    vertical-align: 60%;
    }
```

values: any length `value, sub, sup, top, middle, bottom`

Using Text Classes

Classes: are used in special circumstances when the normal hierarchy of the cascade will not do

```
p.footer {
    font-size: 10px;
    line-height: 14px;
    margin-top: 25px;
    }
```

- a *class* selector is preceded by using a `.`
- in this example, the **class** must be within the `<p>` tag

```
p.footer a {
    font-style: italic;
    color:#F20017;
    }
```

- a *contextual selector*, such as the `<a>` tag, may be used in a class

```
<p class="footer"><b>&copy; Premium Design Works</b> &ndash; <a
href="mailto:info@premiumdw.com?subject=Contacting%20Premium%20Design%20Works">info@pr
emiumdw.com</a></p>
```

- to specify the *class* in your markup you must use the `class` attribute and specify its `value`; in this case, `class="footer"`

Assignment: CSS Text

Week Four: CSS Boxes

The Div & the ID

Div tags: where `<div>` is short for division, are the basic tags you will use to group your content into logical components

```
<div id="logo">
    <a href="http://www.premiumdw.com/">
     <img src="images/flameLogo.gif" alt="Premium Design Works" />
    </a>
</div>
```

ID's: are the means to control each `<div>` tag of logical components via the style, selected by the **#**

```
#logo {
    width: 300px;
    height: 60px;
    margin-top: 0px;
    }
```

- only one instance of an ID can be used per page
- used to identify a unique piece of your page's markup; logo, navigation, etc.
- used to identify and enable a specific target for a JavaScript function

The Box Model – *p. 103, fig. 4.2*

- every element you create in your markup produces a "box" on the page
- by default the border of the box is not visible and the background is transparent
- by giving borders and colors to your boxes, you will see the "box model" in effect

Borders: you can set the *thickness*, *style* and *color* of your border – ***p. 104, fig. 4.3***

```
#borders {
    border-width: 2px;
    border-style: solid;
    border-color: #85898A;
    }
```

1) **Border-Width:** you can set the thickness of your border via: *thin*, *medium*, *thick* or any unit
2) **Border-Style:** you can set the style of the border via: *none*, *hidden*, *dotted*, *dashed*, *solid*, *double groove*, *ridge*, *inset* and *outset*
3) **Border-Color:** you can set the color of each border via: *RGB*, *hexadecimal* or *keyword*

Border-Shorthand: you can write the border declaration with shorthand; *width*, *style*, *color*

```
#copy img {
    border:1px solid #F20017;
    }
```

Padding: will allow you to set the space between the box's content and the border of the box – ***p. 106, fig. 4.4***

```
#padding {
    padding-top: 10px;
    padding-right: auto;
    padding-bottom: 10px;
    padding-left: auto;
    }
```

1) **Padding -Top:** allows you to set the distance from the ***top*** of your "box" to adjacent elements in units
2) **Padding -Right:** allows you to set the distance from the ***right*** of your "box" to adjacent elements in units
3) **Padding -Bottom:** allows you to set the distance from the ***bottom*** of your "box" to adjacent elements in units
4) **Padding -Left:** allows you to set the distance from the ***left*** of your "box" to adjacent elements in units

Margins: will allow you to set the distance between this "box" and the adjacent elements – ***p. 107, fig. 4.6***

```
#margins {
    margin-top: 10px;
    margin-right: auto;
    margin-bottom: 10px;
    margin-left: auto;
    }
```

1) **Margin-Top:** allows you to set the distance from the ***top*** of your "box" to adjacent elements in numerical and percentage units
2) **Margin-Right:** allows you to set the distance from the ***right*** of your "box" to adjacent elements in numerical and percentage units or auto units
3) **Margin-Bottom:** allows you to set the distance from the ***bottom*** of your "box" to adjacent elements in numerical and percentage units
4) **Margin-Left:** allows you to set the distance from the ***left*** of your "box" to adjacent elements in numerical and percentage units or auto units

Margins & Padding Defaults: each browser has a default margin and padding to the page and its elements, so it is a good idea to set your own to zero

```
* {
    margin: 0;
    padding: 0;
    }
```

- by using the * selector, we are specifying "all" elements

Margins & Padding Shorthand: by using the shorthand method you can combine all margin and padding properties into one declaration

```
#shorthand {
    margin: 5px 10px 5px 10px;
    padding: 10px 5px 10px 5px;
    }
```

- when using shorthand, you will write your values clockwise; *top*, *right*, *bottom*, *left*

Collapsing Margins: vertical margins will "collapse" when there are both top and bottom margins horizontal margins do not

- when top and bottom margins meet, they overlap until one of the margins touches the border of the other element
- the larger of the two will override – ***p. 108, fig. 4.8***

Block & Inline Properties

Block: elements, such as `<p>` , `<h1>` , and `<div>` sit one above another when displayed in the browser window

Inline: elements such as `<span>` , `<img>` sit side by side when displayed in the browser window

Positioning Elements

Static Positioning: with *static* positioning, each element is simply laid out on the page one-by-one as it is written in the code – ***p. 112, fig. 4.13***

- *static* positioning is the default type of positioning
- to override *static* positioning, you will need to specify either *relative*, absolute or *fixed* positioning

Float & Clear Properties

Float: the *float* property is primarily used to flow text around images – ***p.117, fig. 4.18***

```
img {
 float:left;
 margin:0 4px 4px 0;
 }
```

- in order for the float property to work properly, you must write, in you code, the element to be floated before the element that wraps around it

```
<img src="../ch1_xhtml/stylin_logo1.gif" width="150" height="80" />
<p>Here is a paragraph of text. It wraps around the image because the image's float
property is set to left. </p>
```

Clear: the *clear* property is used to override the *float* property – ***p.119, fig. 4.20*** & ***p.121, fig. 4.21***

```
p {
   margin:0 0 10px 0;
   }

img {
   float:left;
   margin:0 4px 4px 0;
   }

.clearthefloats {
   clear:both;
   }
```

Assignment: CSS Boxes

Week Five: CSS Page Layout

Absolute Layouts

Absolute Positioning: will allow you to put your boxes anywhere on the page by means of a *top* and *left* coordinate

```
#logo {
      width: 500px;
      position: absolute;
      top: 25px;
      left: 50px;
      }
```

Position: by giving the *position* property a value of *absolute*, you are specifying that the rule will adhere to a position that you specify on the page

Top: the *top* property will say how many pixels down from the *top* of the page that your box will sit

Left: the *left* property will say how many pixels across from the *left* of the page that your box will sit

Static Layouts

Centered Layouts: one of the more popular techniques of today is to center your "Live Area" page layout within the browser frame, which we will do in the following steps

Background Images: one popular idea is to give the entire page a different background than the background that the "main content" sits within

```
body {
 background-image:  url(images/bodyBG.gif);
 font-family: Arial, Helvetica, sans-serif;
 color: #85898A;
 }
```

- in this example I have used a one pixel shim for the background color as to ensure that it will match with the main content background

Main Content: to center the entire layout in the page, one would use a static layout with a "main" content box that is centered in the page using auto margins

```
#main {
 width: 780px;
 background-image: url(images/mainBG.gif);
 margin-right: auto;
 margin-left: auto;
 }
```

- by using the value of *auto* for the properties of *right* and *left* margins, the main box will appear to be centered in the page
- by using a main background image to match the body background image, I have created an illusion of depth

Headers: within the main content box, one should start out with a “branded” header for the page layout

```
#header {
      border-bottom: 1px solid #85898A;
      margin: 0px 75px 0px 75px;
      padding: 25px 0px 25px 0px;
      }
```

- notice that this *header* box has no width property, so it will extend the width of the *main* box it is within
- the box also uses the margin properties to set the border of the *header* box within the *main* box
- and uses padding to set the logo away from the top and bottom

Columns: to give my layout a *menu* and a *copy* block I will use a two column layout

```
#menu {
      width: 100px;
      height: 500px;
      float:left;
      margin: 25px 0px 0px 75px;
      }

#copy {
      width: 500px;
      float: right;
      margin: 25px 0px 25px 0px;
      padding: 0px 75px 0px 25px;
      border-left: 1px dotted #85898A;
      }
```

- by using two boxes, *menu* and *copy*, and floating them *left* and *right*, they will sit side by side; thus you will be able to build a two column layout
- by giving a *border* to the *left* side of the copy box, I have reinforced the “look” of columns

Footers: often contain navigation items, the copyright notice and links to email

```
#footer {
      clear: both;
      border-top: 1px solid #85898A;
      margin: 0px 75px 0px 75px;
      padding:  25px 0px 25px 0px;
      }
```

- when using a box, such as this *footer*, after two floating columns you must use the *clear* property; in this case I have cleared *both*
- notice that I have also used the same *margin* properties
- I have also used the same *border* values to make the footer look consistent with the *header*, however the *border* is now on the *top* of the footer box

Assignment: Absolute & Static Page Layouts

Week Six: Lists & Menus

Understanding Lists

Lists: are the basis for navigation and menus and come in three styles; *unordered*, *ordered* and *definition* ***– p. 175, fig. 7.1***

Unordered: lists that are bulleted by default
Ordered: lists that are numbered by default
Definition: lists that contain sub-items

```
<div id="listcontainer">
 <ul>
  <li>Gibson</li>
  <li>Fender</li>
  <li>Rickenbacker</li>
  <li>Washburn</li>
 </ul>
```

Menus

Styling Menus: by giving style properties to the *list item*, you will be able to create some stylish looking menus – let's take a look at a boring but useful example…

```
div#listcontainer {
   border:1px solid #000;
   width:160px;
   font-size:.75em;
   margin:20px;
   }
```

- this example has an unordered list inside a `<div>` with the border turned on – ***p. 177, fig. 7.2***
- *notice that the list item still has the default bullet*

```
ul {
 border:1px solid red;
 }

li {
 border:1px solid green;
 }
```

- now we have turned the `<ul>` and `<li>` borders on – ***p. 177, fig. 7.3***

```
ul {
 border:0px solid red;
 margin:0 0 0 1.25em;
 padding:0;
 }
```

```
li {
 border-bottom:2px solid #069;
 margin:0;
 padding:.3em 0;
 }
```

- by giving the <li> components a *border-bottom*, we have started giving the menu a more designed look to it – ***p. 180, fig. 7.6***

```
ul {
  border:0;
  margin:10px 30px 10px 1.25em;
  padding:0;
  list-style-type:none;
}

li {
  border-bottom:2px solid #069;
  margin:0;
  padding:.3em 0;
  text-indent:.5em
}
```

- to get rid of the bullet and make it look more like a menu we must use the *list-style-type* property and set it to *none* – ***p. 181, fig. 7.7***

```
li:first-child {
 border-top:2px solid #069;
 }
```

- lastly, if we add a *border-top* to the *first-child* element in the list, we get the final look of the menu – ***p. 182, fig. 7.8***
- unfortunately this does not work in IE for the PC ***– p. 185***

Menus & States

```
#menu {
     width: 75px;
     float:left;
     margin-top: 25px;
     padding-left: 75px;
     }

     #menu ul {
         border-top: 1px solid #85898A;
         }
```

- here is my work around to the *border-top* issue

```
     #menu li {
         color: #F20017;
         font-size: 12px;
         font-weight: bold;
         line-height: 18px;
```

```
letter-spacing: 0px;
text-align: left;
border-bottom: 1px solid #85898A;
list-style-type: none;
padding-top: 2px;
padding-bottom: 2px;
}
```

- again, I give the menu a *border-bottom*

States: there are four basic states that we must cover when creating our link styles; *link*, *visited*, *hover* and *active* *(you must write them in this order)*

a:link: is the state of the default link

```
#menu li a:link {
    color:#85898A;
    text-decoration: none;
    }
```

a:visited: is the state when you have previously viewed that link's page

```
#menu li a:visited {
    color:#85898A;
    text-decoration: none;
    }
```

a:hover: is the state of the rollover effect for that link

```
#menu li a:hover {
    color:#F20017;
    text-decoration: underline;
    }
```

a:active: is the state of the link when it is being clicked

```
#menu li a:active {
    color:#F20017;
    text-decoration: underline;
    }
```

Menus & Backgrounds

```
#menu {
     margin-left: 75px;
     margin-top: 25px;
     width: 100%;
     }

     #menu li {
         float: left;
         color: #F20017;
         font-size: 12px;
```

```
font-weight: bold;
letter-spacing: 0px;
border-right: 1px solid #85898A;
list-style-type: none;
padding: 2px 0px 2px 0px;
}
```

- to give the menu a ***horizontal*** appearance, we need to *float* the *list items* of the menu *left*

```
#menu li:first-child {
    border-left: 1px solid #85898A;
    }
```

- again we must take care of the *first-child* element in the menu

```
#menu li a:link {
    color:#85898A;
    text-decoration: none;
    padding: 2px 15px 2px 15px;
    }
```

- we must now put the padding in the *a* states in order to achieve a background that will extend to our borders

```
#menu li a:visited {
    color:#85898A;
    text-decoration: none;
    padding-left: 15px;
    padding: 2px 15px 2px 15px;
    }

#menu li a:hover {
    color:#F20017;
    text-decoration: underline;
    padding: 2px 15px 2px 15px;
    background-color: #CCCCCC;
    }
```

- I have chosen to put the background color in the *hover* state via the *background-color* property

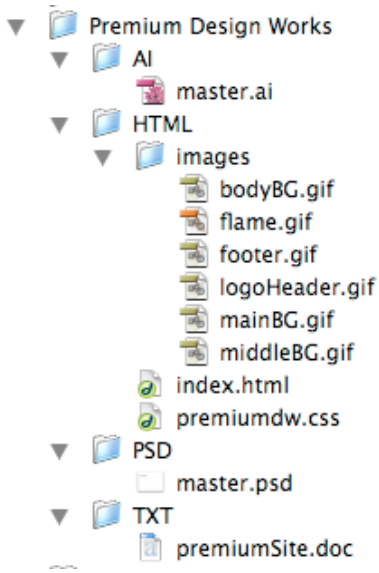
```
#menu li a:active {
    color:#F20017;
    text-decoration: underline;
    padding: 2px 15px 2px 15px;
    background-color: #CCCCCC;
    }
```

Week Seven: Building Websites

Getting Started

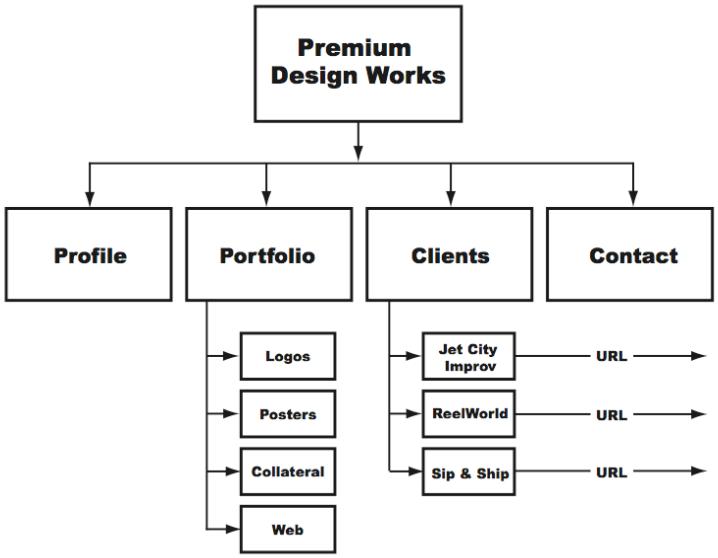
Templates: the interface components and layout you have already built in the assignment phase can be reused to speed development time

Folder Structures: setting up the local folder on your drive in a clean and organized manner is the first step to a good website



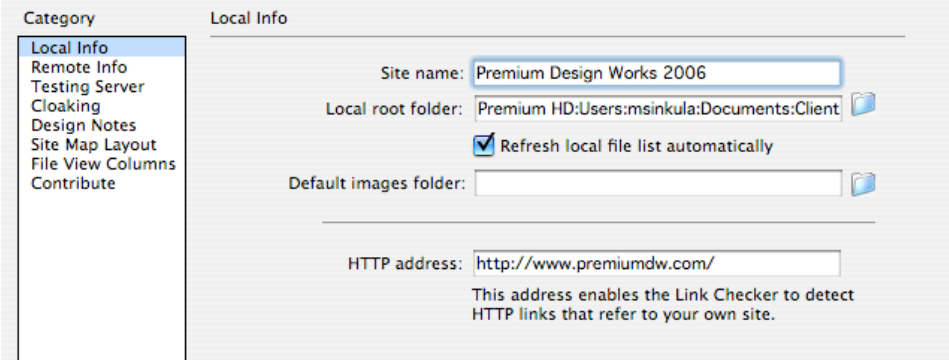
Root Folder: the HTML folder is the ***root*** folder of our website - we will eventually upload the contents of this folder to our server

Site Architecture: any good site will have a good hierarchical structure to the site

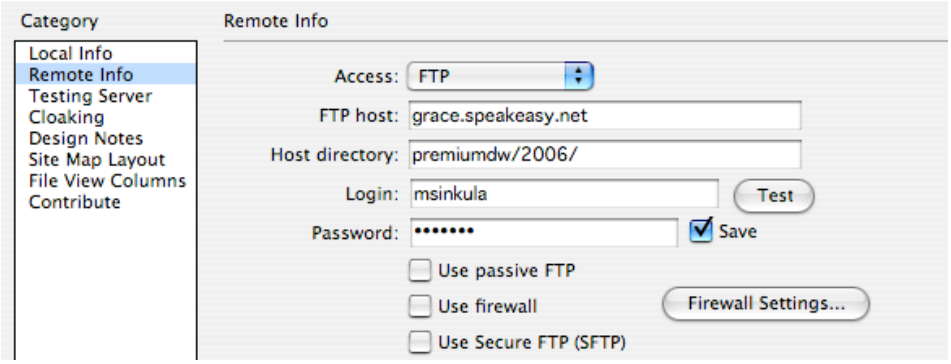


Defining the Site

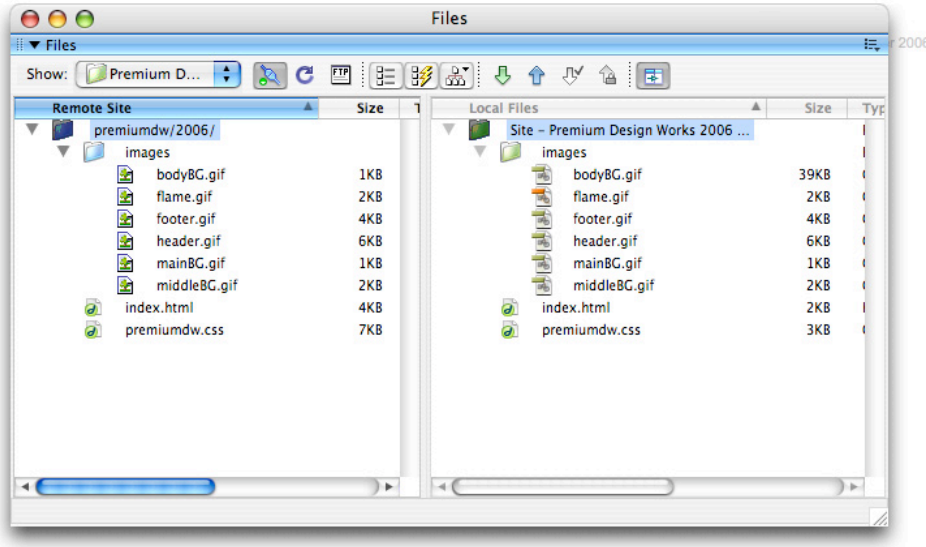
Local Info: the HTML folder within your project folder will act as your ***local*** folder



Remote Info: is the folder on your server where you will upload your files – needs a host, directory, login, password



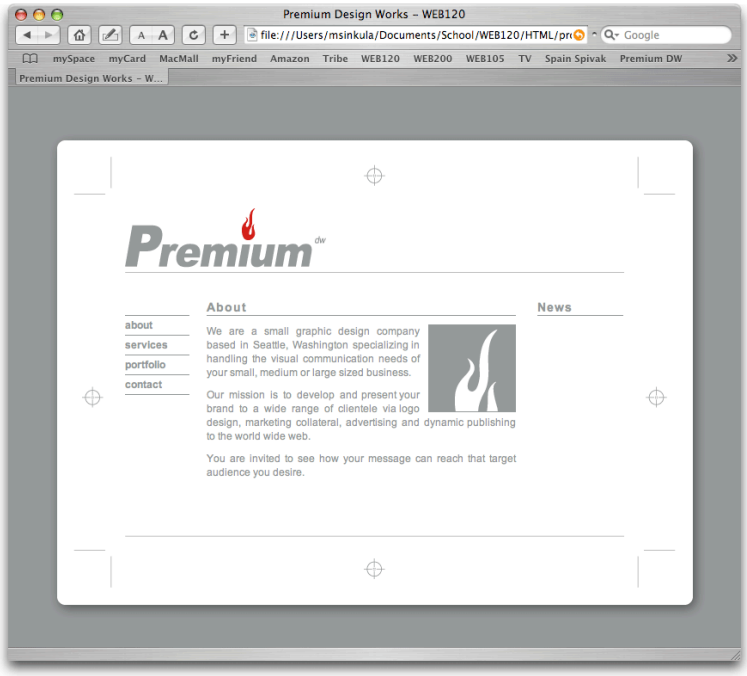
Uploading: Drag and drop



Master Art Files

Specifications

Assignment: Final Project



Project Requirements:

1) **Pages:** must have at least four main pages with one page having at least three sub-pages
2) **Navigation:** must adhere to the four states discussed in assignment phase
3) **Code:** all code and style sheets must adhere to standards & accessibility as we have and will discuss(ed)
4) **Layout:** must adhere to one of the two or three column layouts we discussed in the assignment phase with:
    a. **Header:** must have a header with a name or logo
    b. **Menu:** must have a menu and sub menu
    c. **Middle:** must have a copy section with images
    d. **Footer:** must have a footer with copyright

Week Eight: Advanced Page Layout

Using Backgrounds

Background-Image: property has only one value – the URL of the image – ***p. 255***

```
body {
    background-image: url(images/bodyBG.gif);
    font-family: Arial, Helvetica, sans-serif;
    color: #85898A;
    }
```

- this background image is a one pixel shim that will repeat both horizontally and vertically

Background-Repeat: this property will allow you to control the repeat of your background image – ***p. 255***

```
#main {
    width: 780px;
    background-image: url(images/mainBG.gif);
    background-repeat: repeat-y;
    margin-top: 5%;
    margin-right: auto;
    margin-left: auto;
    }
```

- this background image will only repeat vertically

Background-Position: this property will allow you to set the starting point of your background image – ***p. 255***

```
#middle {
         width: 780px;
         background-image:  url(images/middleBG.gif);
         background-position: center;
         background-repeat: no-repeat;
         }
```

- this background image will always start from the center and not repeat giving the illusion of moving up and down as the page resizes

Full-Length Columns

Alsett Clear-Fix: the *Alsett* clearing method uses the CSS `:after` pseudo-class insert a hidden bit of non-floated content (instead of a div) at the appropriate place in the markup – ***p. 145, fig 5.17***

```
.clearfix:after {
    content: "."; /* the period is placed as the last (page) element  before the div closes */
    display: block; /* inline elements don't respond to the clear property */
     height: 0;  /* ensure the period is not visible */
     clear: both; /* make the container clear the period */
     visibility: hidden;   /* further ensures the period is not visible */
    }
```

```
.clearfix {display: inline-block;}    /* a fix for IE Mac */

/* next a fix for the dreaded Guillotine bug in IE6 */
/* Hides from IE-mac \*/

* html .clearfix {height: 1%;}
.clearfix {display: block;}
/* End hide from IE-mac */
/* end of "no-extra-markup" clearing method */
```

Faux Columns: since the columns will only extend to the content contained within them, you will need to use a background-image to give them a color - ***p. 156, fig 6.7***

```
#contentarea {
    width:774px;
    background-color:#FFF;
    background-image:url(images_pres/faux_columns.gif);
    background-repeat: repeat-y;
    }
```

*** Photoshop Demonstration ***

Week Ten: Accessibility

Accessibility and Standards have much in common – they both are about ensuring that our work will be usable and available to the largest possible number of readers, visitors and customers.

Accessibility Theory

Implementing Accessibility

- Implementing Accessibility takes place in the markup code
- So… Designer dude, "your design will still look good"

Web Crawlers

- Web crawlers are in essence the worlds biggest "blind user" of the web
- This "blind user" gives out recommendations in the form of search results to a bazillion other users every day

Not Limited to the Visually Impaired

- Many access enhancements are targeted for the motor impaired population
- Access enhancements are also geared for the user on his/her palm pilot or web enabled cell phone as well

Section 508

- Section 508 became U.S. law as part of the Rehabilitation Act of 1973 – intended to end discrimination against people with disabilities
- Public Law 105-220 (Amendments of 1998) expanded Section 508 to include computers and other equipment used for transmitting, receiving and/or storing information
- This requires all websites under its jurisdiction to provide "equal or equivalent access to everyone"
- Section 508 does not forbid the use of CSS, JavaScript, images, table-based layouts, Flash or QuickTime

Accessibility Elements

Images

1) Leaving out the `alt` attribute and text for images will cause screen readers to read nothing when coming across an image and will be flagged as WAI access errors and cause validation errors
2) When using `alt` attributes, make sure the text is logical to the image and conveys the proper meaning:

```
<div id="logo">
   <a href="http://www.premiumdw.com/"><img src="images/flameLogo.gif"
   alt="Premium Design Works Home" /></a>
</div>
```

   - to a visually impaired user, this will signify that this image is the link back to the home page

3) For images that are meaningless such as spacer GIFs, one should use a `null alt` attribute:

```
<img src="images/spacer.gif" alt=" " />
```

4) Background images do not get an alt attribute

Media

1) when displaying media that requires a plug-in, include one clear link to the required plug-in
2) If you are using an image as a link to the plug-in, use the proper alt attribute with text

Color

1) if you use color to denote information (such as clickability), reinforce it with other methods
2) make the difference between linked and ordinary text obvious
3) avoid referring to color in your text – “visit the yellow box for help”

CSS

1) test your pages with and without style sheets to make sure you structure is correct (remember week one?)
2) your structured markup is what will convey the hierarchy of meaning when viewed without CSS
3) test you CSS in multiple browsers and across multiple platforms – NOW!!

Scripting

1) code your pages to work even when JavaScript is disabled
2) provide alternatives for non-mouse users:

```
<li><a href="#" onclick="slidingMenu('assignmentLinks')"
onkeypress="slidingMenu('assignmentLinks')">Assignments&gt;&gt;</a></li>
```

- if a non-mouse user is trying to access this sliding menu they can access it thru a “key-press”

Image Maps

1) Avoid image maps if you can
2) If you need to use them, use client-side image maps with the `alt` attribute
3) just say no to server-side image maps

Frames

1) NO!
2) BAD!
3) ICK!

Assignment: Go make your site accessible.