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>