Agenda

- **Referencing Stylesheets**
- **Importing**
- **Fonts**
- **Compact CSS**
- **Advanced Selectors**
- **Rule Specificity**
- **To Do**

Referencing CSS

When linking to external style sheets, there are a number of features to be aware of:

- Each stylesheet requires it's own **link** tag
- Multiple stylesheets can be loaded (main.css, fonts.css, layout.css, etc)
- **rel** determines how the HTML will use the file linked to (*REQUIRED)*
- **href** defines the path, relative or absolute, to the css file (*REQUIRED*)
- **type** attribute is has been deprecated (dropped in HTML5)
- **title** attribute allows the user to group stylesheets
- **media** attribute allows you to specify what medium this should be viewed in

```
<link   rel="stylesheet"
        href="http://www.mysite.com/css/styles.css"
        type="text/css"
        title="Main Style"
        media="screen" />
```

Media variations: media attribute

- **media="type"** can be a powerful tool
- Examples of media types:
    - **all** - well supported, commonly used (*DEFAULT)*
    - **aural**
    - **braille**
    - **embossed**
    - **handheld** - not common (smart phones identify as **screen**)
    - **print** - well supported, commonly used
    - **projection**
    - **screen** - well supported, commonly used
    - **tty**
    - **tv**
- Note that most smart phone devices (iPhones, Andriod, etc) do not identify themselves as media type **handheld**. The reasoning was that the devices were more than capable of displaying standard web sites, so they identify as **screen**

Alternate style sheets: rel attribute

You can also have alternate stylesheets supported

- **rel="stylesheet"** defines the default stylesheets to use
- **rel="alternate stylesheet"** allows different styles to be defined
- **Tip:** If you have multiple sheets that belong to one "look" make sure they all have the same **title**
- You can create alternate style sheets for users, allowing them to select which ones they would like to see
- An example would be to create a 'Larger Text' style sheet with everything in a larger font, or providing an option for users to choose an austere, simple page instead of a colorful/image heavy stylesheet
- Many browsers are now giving users the ability to select which sheet they would like to see. (Firefox: View > Page Style)

Alternate style sheets: title attribute

- A stylesheet is "persistent" if it is linked with **rel="stylesheet"** and has no **title** attribute. All persistent stylesheets are always used when rendering
- A stylesheet is "preferred" if it is linked with **rel="stylesheet"** and has a **title** attribute. these styles will be used as the default. preferred stylesheets with the same title are grouped together
- Finally, a stylesheet is "alternate" if it is linked with **rel="alternate stylesheet"** and has a **title**. These stylesheets are supposed to allow the user to choose stylesheets, they are grouped together by **title** and show up in the browser's stylesheet selector if it has one (**View > Page Style** in Firefox). Each group (by **title**) is mutually exclusive.

Persistent Styles

Persistent styles apply in all cases, even when alternate styles are chosen.

```
<!-- rel="stylesheet" with NO title="" -->
<link   rel="stylesheet"
        href="css/persistent_styles.css" />
```

Preferred Styles

Preferred styles will be the default, but will be replaced if alternate styles are chosen.

```
<!-- rel="stylesheet" AND a title="" -->
<link   rel="stylesheet"
        href="css/default_styles.css"
        title="Standard Styling"      />
```

Alternate Styles

Alternate styles will not be applied by default, but may be chosen by the user agent. Each alternate stylesheet must have a **title=""**.

```
<!-- use rel="alternate stylesheet" for alternative stylesheets... -->
<link   rel="alternate stylesheet"
        href="css/large_text.css"
        title="Large Text Styles" />
```

Grouping Stylesheets

If more than one link tag shares the same **title=""** then they will be grouped together.

```
<-- group stylesheets with shared title="" -->
<link    rel="alternate stylesheet"
         href="css/monchrome_layout.css"
         title="Monochrome Styles"   />
<link    rel="alternate stylesheet"
         href="css/monchrome_fonts.css"
         title="Monochrome Styles"   />
```

To enable alternate stylesheet switching, you can either apply some javascript to manage the switch with hyperlinks, or use **Firefox for alternate stylesheets**.

@import: Managing Multiple CSS Files

You can use the **@import** method to import multiple stylesheets from a basic stylesheet that you link to. This will reduce the clutter that can sometimes occur within the **<head>** section.

It is also a great place to be able to manage your CSS. From your main document, create a **<link>** to your **styles.css** page.

In the body of the **styles.css** file, import as many files as you need:

- **@import url("css/type.css");**
- **@import url("css/nav.css");**
- **@import url("css/template.css");**

NOTE: when using the **@import** directive in an external CSS file, ensure it comes before your CSS selector rules. It might be best to not include any CSS rules in a page that uses **@import**.

@charset: Assigning CSS Character Set

If your CSS files only contain standard keyboard characters, assigning the character set is not required. The CSS language itself does not use any exotic characters, so assigning a charset is often omitted. If, however, your CSS needs to use special characters, ensuring the correct character set can be very important. Eg: When using a pseudoclass to add content to HTML **content: "Äwesome!"**

If needed, add the **@charset** rule to the top of the CSS document.

```
/* assign charset at top of .css file */
@charset "utf-8";

/* CSS rules follow... */
```

@font-face And Font Servers

Fonts that aren't available on the client machine can be provided by a third-party font server or by your website's server.

- **Google Fonts**
- **Adobe Typekit**
- **Font Squirrel**
- Plenty more available online...

Third Party Font Server

Browse the font server site, select the fonts you like and they wil provide you with code

for a **<link />** tag to add to your HTML. You can then use the font in your CSS.

```
<!-- add the link tag to the HTML -->
<link href="https://fonts.googleapis.com/css?family=Bungee" rel="stylesheet" />

/* use the font in the CSS */
selector{
        font-family: 'Bungee', cursive;
}
```

No longer stuck with 'Arial', 'Helvetica', 'Georgia' and the rest.

Serve Fonts From Your Website

If the font(s) are copyright free, or if you have license to distribute a font, you can serve them yourself using the CSS **@font-face** directive.

Different client systems will need fonts in varying formats, so if you are distributing the font yourself, be sure to have the font available in most common formats.

- Obtain the font(s) you wish to use. You must either have license to distribute the font or the font must be copyright free.
- Upload your font(s) to a **Font Generator** to obtain a font kit including various formats for the font(s)
- Uncompress the font kit and locate it somewhere with your website files
- Use **@font-face** to load the fonts. The font-kit generated by Squirrel Font will include some same CSS to help you with the sometimes volumous coding
- Apply the new font in CSS

```
/* load the font in CSS */
@font-face {
  font-family: 'alex_brushregular';
  src: url('/fonts/alexbrush-regular-webfont.woff2') format('woff2'),
     url('/fonts/alexbrush-regular-webfont.woff') format('woff'),
        url('/fonts/alexbrush-regular-webfont.ttf') format('tff');
}

/* use the font in CSS */
selector{
        font: bold 20px alex_brushregular ;
        color:#555;
}
```

Don't get carried away with wild font choices. Some fonts have very poor readability, reducing accessibility.

CSS Measurements

In CSS, there are a number of measurement values that can be used in declarations.

Size measurements

- **px** - Pixels is one of the most common measuring types. It is specific to computers, as it renders based on the pixel-display of all monitors. Great for

specific measurements on boxes and borders, but for fonts, it can lead to difficulty as some browsers don't allow scaling and font can end up too small

- **%** - Percentage measurements are also very common.
- **em** - A relative measurement of the letter "m" in the chosen font. This is a popular one for spacing around text, as it inherits the font dimensions
- **rem** - Similar to **em**, this will inherit only the Root value, helpful for avoiding 'compound inheritance' values
- **pt** - Points is an aold system related to type-setting that had a standard size for points. This is a relative size based on the users' system
- **pc** - Picas is a very old system that was frequently related to typewritten documents. Rarely used.
- **ex** - Similar to **em**, this is a relative measurement of the letter "x". However, it is rarely used, and not fully supported
- **vh, vw, vmin, vmax** - size relative to viewport widths and heights. very useful for scaling, though may not enjoy full support yet
- **named** - This refers to fonts and a few other elements. The named sizes are xx-small, x-small, small, medium, large, x-large, xx-large. They are relative to the browsers base font size

Color Measurement

Colors can be specified in a number of ways, corresponding to the system that your organization or visual element requires. Almost all color systems use a method for expressing a combination of the three color spaces used for displaying colors on computer monitors: Red Green and Blue.

- **Hex: #003366** - One of the most common systems, it combines 2 digits each for Red Green and Blue. Each hex pair stand stands for digits between 00 and FF in hexadecimal
- **Compact Hex: #f30** - This is almost identical to the Hex version, except that when three numbers are used, most browsers will just double-up each digit. So, **#f30** will become **#ff3300**
- **Named: cornflowerblue** - Although the official W3C specification only lists 16 named colors, almost every browser supports the various **named colors**
- **RGB: rgb(255, 0, 255)** - This allows the user to specify the same thing as Hex, but in Decimal format
- **RGB %: rgb(100%, 0%, 50%)** - Almost identical to RGB, but with percentages of each color space instead

Compact CSS Code

When building your CSS rules, there are some easy ways to compact your code, reducing code bloat and filesize.

Grouping selectors

- If you would like to apply the same style to a multitude of different selectors, you can group them with the comma

- **h1, h2, h3, h4, h5 { color: red; }**
- This can be done with Elements, classes and IDs
- Specific rules for H3 (for example) can come later

Grouping comes to Classes

- As well as grouping Elements, you can also group classes
- Define several classes: **.cool{ color:red; } .wicked { border:1px solid black; }**
- Assign several classes at once: **<p class="cool wicked">** will select both the **cool** and **wicked** classes
- The order of the attribute values is irrelevant
- Case is important! Remember to use a single rule for CSS naming and stick with it

Rule Compacting

There are a lot of rules that will accept a compacted ruleset:

- **margin: top right bottom left;**
- **padding: top right bottom left;**
- **border: width style color;**
- **font: font-style font-variant font-weight font-size/line-height font-family;**
- **background: color image position repeat size attachment;**

Box model properties support 1, 2, 3 and 4 values. For example:

- **border-width:10px;**
  - all four borders are 10px
- **border-width:10px 20px;**
  - top and bottom borders are 10px
  - left and right borders are 20px
- **border-width:10px 20px 30px;**
  - top border is 10px
  - left and right borders are 20px
  - bottom border is 30px
- **border-width:10px 20px 30px 40px;**
  - top border is 10px
  - right border is 20px
  - bottom border is 30px
  - left border is 40px

Note you do not need to specify values for all fields in a compact rule. Just keep the space delimited list in the correct order. For example, you could specify several font values all at once:

font: italic small-caps bold 1.3em "helvetica", sans-serif;

or, specify only a select couple.

font: 1.3em "helvetica", sans-serif;

CSS Selectors

There are several **advanced CSS selector techniques** that provide plenty of options for the application of style.

Universal Selector

- This is a fairly widely supported selector: the asterisk *****
- Use it to specify a global rule
- *** { color: red; }** will make all font color red on your page
- The universal selector can be very powerful, as it can command a lot of changes to your styles,

Pseudo-classes

A Pseudo-class is a selector that allows you to reference a specific action or attribute of an element. This is most often used when specifying **a** attributes, such as **:visited** and **:link**.

There are other uses for pseudo elements, that will allow for greater detail in styling content, such as **:first-letter** and **:first-line**. These allow the browser to apply a style based on an unselected, but displayed element.

Best part: No **span** or **div** required to access these pseudo-elements.

- Within **a** pseudo-elements in particular, it's important to follow a certain order:
    - **a:link**
    - **a:visited**
    - **a:hover**
    - **a:active**

Easy mnemonic: LVHA - **L**o**V**e - **HA**te. Because of the way that CSS applies rules, it looks at the LAST rule for a given element and applies more weight to it. For example, if you are hovering over a link, you don't want to see the link state

- There are several additional **pseudo-classes** and **pseudo-elements**
    - **selector:before, selector:after** - add text content before or after the selector's content
    - **selector:first-line, selector:first-letter** - style the first line or letter of the selector
    - **selector:first-of-type**, **selector:last-of-type** - style the first or last tag of the selector type
    - **selector:nth-of-type(n)** - select the nth() tag of the same type as selector
    - **selector:nth-child(n)** - select the nth() child
    - **selector:nth-child(2n)** - select every even numbered child
    - **selector:nth-child(2n+1)** - select every odd numbered child

Descendant Selectors

- Separated by a space, the last element is the one being styled
- **h1 em** will apply rules to all em's within an h1
- You can have as many as you want: **ul ol ul li**
- Great for specifying different styles when they apply only in certain areas

- **#nav a { ... }** will only apply these rules to Anchors within the div called nav
- Degree of separation can be infinite, so be careful
- **p em** will match ANY em that descends from a paragraph, no matter how nested

Child Selectors

- Uses the > combinator to specify children
- Will select an element that is directly descended from another element
- **p > strong** will only select a strong tag that is inside a paragraph
- Will not select a strong tag embedded in any other tag (even if it is within a paragraph)

Adjacent Sibling Selectors

- The + combinator specifies the first sibling tag following the root
- It will not include the root for styling
- **div + h2** will select the first sibling h2 following a div tag

General Sibling Selectors

- The ~ combinator specifies all siblings tags following the root
- It will not select sibling tags that precede the root, nor will it include the root
- **h1 ~ h2** will select all sibling h2 tags that follow an h1

Attribute Selectors

- You can also select elements by what the element contains
- The method is: **element[attribute=value]**
- If you specify an attribute with no value, you will match any element that simply has the attribute present (no matter what it's value is)
- **p[align] {color: red;}** will turn red any paragraph that contains an align attribute
- Similarly, **img[alt] {border: 1pt red solid; }** will draw a red line around all images with an alt attribute
- You can further direct the attribute selector to find content that is at the beginning (using the carat: ^), somewhere inside (using the asterisk: *) or end of a value (using the dollar sign: $)
- Attribute selector examples:
    - **img[alt]** - select all img tags that have alt attributes
    - **img[src="images/pic.jpg"]** - select img tags whose src is 'images/pic.jpg'
    - **img[src^="gallery/"]** - select all img tags whose src BEGINS with 'gallery/'
    - **img[src*="logo"]** - select all img tags whose src CONTAINS 'logo'
    - **img[src$=".png"]** - select all img tags whose src ENDS with '.png'

Rule Order & Specificity

When a style declaration is found to be in conflict with a previously declared style, there is a method for determining which one will win out. The process is built into every web browser, and almost all of them calculate the values identically (for once!).
Here is how the W3C defines the process by which style order is determined:

- **Discover all declarations.** Find all declarations that apply to the element and

property in question, for the target media type. Declarations apply if the associated selector matches the element in question.

- **Sheet Order.** The primary sort of the declarations is by weight and origin: for normal declarations, author style sheets override user style sheets which override the default style sheet.
    - *Note:* For "!important" declarations, user style sheets override author style sheets which override the default style sheet. "!important" declaration override normal declarations. An imported style sheet has the same origin as the style sheet that imported it.
- **Specificity:** The secondary sort is by specificity of selector: more specific selectors will override more general ones. Pseudo-elements and pseudo-classes are counted as normal elements and classes, respectively.
- **Sort by Order:** Finally, sort by order specified: if two rules have the same weight, origin and specificity, the latter specified wins. Rules in imported style sheets are considered to be before any rules in the style sheet itself.

All of the rules are pretty straightforward, except for Specificity, which bears some explanation.

Specificity is a method for determining what rules are applied when a conflict between two selectors occurs. Through a method of adding values when certain elements and selectors are present, a specificity number can be applied to any CSS rule. The one with highest specificity wins and the associated rule is applied to the element(s) in question.

Ruleset for Specificity

Style sheets can also override conflicting style sheets based on their level of specificity, where a more specific style will always win out over a less specific one. It is simply a counting game to calculate the specificity of a selector.

- Count the number of IDs in the selector.
- Count the number of CLASS, pseudoclass, and attribute selectors.
- Count the number of HTML tag names in the selector.

Next, write the three numbers in exact order with no spaces or commas to obtain a three digit number. (Note, you may need to convert the numbers to a larger base to end up with three digits.) The final list of numbers corresponding to selectors will easily determine specificity with the higher numbers winning out over lower numbers. Following is a list of selectors sorted by specificity:

- #id1
  /* **id**=1 **class**=0 **HTML**=0 --> specificity = 100 */
- UL UL LI.red
  /* **id**=0 **class**=1 **HTML**=3 --> specificity = 013 */
- LI.red
  /* **id**=0 **class**=1 **HTML**=1 --> specificity = 011 */
- LI
  /* **id**=0 **class**=0 **HTML**=1 --> specificity = 001 */

!important rule

- This declaration, when added to a rule, will override almost all other declarations, with no respect for specificity.
- It indicates that this rule should take over any previous rules that might govern a specific element.
- Example **.selector{ color:blue !important; }**
- This rule can cause havoc if overused, as it will overrule any other declaration with the same selector. **Be careful with !important**

To Do

- Download, review, and complete the homework assignment from **D2L**
- Find a partner for your homework.
- Practice using **advanced CSS selectors**