6.170 Tutorial 5 - HTML & CSS

- Prerequisites
- Goals of this Tutorial
- Useful Resources for HTML & CSS
- Topic 1: HTML
  - What is HTML?
  - HTML Tags
  - HTML Elements
  - Web Browsers
  - HTML Page Structure
  - HTML Versions
  - The <!DOCTYPE> Declaration
  - Declarations
- Topic 2: CSS
  - What do .css files actually do?
  - CSS syntax
  - Most common selectors
- Topic 3: Specificity and the “Cascade” in CSS:
- Topic 4: CSS Box Model
- Topic 5: Linking CSS to Rails Templates

Prerequisites

1. Have Ruby installed on your computer

Note: Having completed Tutorial P1, Ruby should already be installed on your computer.

Goals of this Tutorial

Become familiar with HTML & CSS

Useful Resources for HTML & CSS

1. Mozilla Developer Network (https://developer.mozilla.org/)
   a. HTML: https://developer.mozilla.org/en-US/docs/HTML
   b. CSS: https://developer.mozilla.org/en-US/docs/CSS
2. CSS Basic PDF: http://www.cssbasics.com/full.pdf
3. Layouts and Rendering in Rails: http://guides.rubyonrails.org/layouts_and_rendering.html

Topic 1: HTML

What is HTML?

HTML is a language for describing web pages.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is a **markup** language
- A markup language is a set of markup **tags**
- The tags **describe** document content
- HTML documents contain HTML **tags** and plain **text**
- HTML documents are also called **web pages**

HTML Tags

HTML markup tags are usually called HTML tags

- HTML tags are keywords (tag names) surrounded by **angle brackets** like <html>
- HTML tags normally **come in pairs** like and
    - The first tag in a pair is the **start tag,** the second tag is the **end tag**
    - The end tag is written like the start tag, with a **forward slash** before the tag name
    - Start and end tags are also called **opening tags** and **closing tags**
    - There is often text inside the tags:
      <tagname>content**</tagname>** (i.e. **** content****)
- Certain HTML tags can also appear alone, like <img>

HTML Elements

"HTML tags" and "HTML elements" are often used to describe the same thing.
But strictly speaking, an HTML element is everything between the start tag and the end tag, including the tags:
** **This is a paragraph.** **

Web Browsers

The purpose of a web browser (such as Google Chrome, Internet Explorer, Firefox, Safari) is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page:

Screenshot of browser displaying sample heading and paragraph text removed due to copyright restrictions.
Refer to: Step 4 in HTML Editors.

HTML Page Structure

Below is an example of HTML page structure:

```
<html>
<body>
<h1>This a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML+	1993
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML 1.0	2000
HTML5 / XHTML5	2012

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration helps the browser to display a web page correctly.
There are many different documents on the web, and a browser can only display an HTML page 100% correctly if it knows the HTML type and version used.

Declarations

Before HTML5, it was common to declare the HTML version at the start with something messy:

XHTML 1.0

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

HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

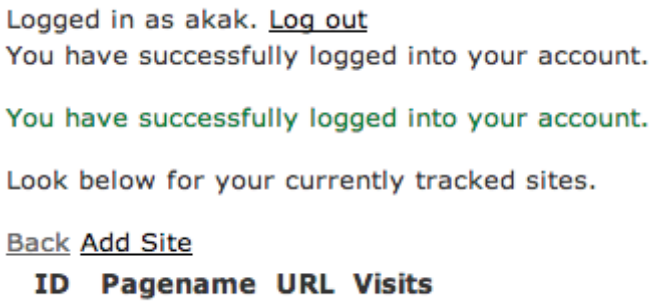
HTML5 is much simpler, and is what we recommend you use:

HTML5

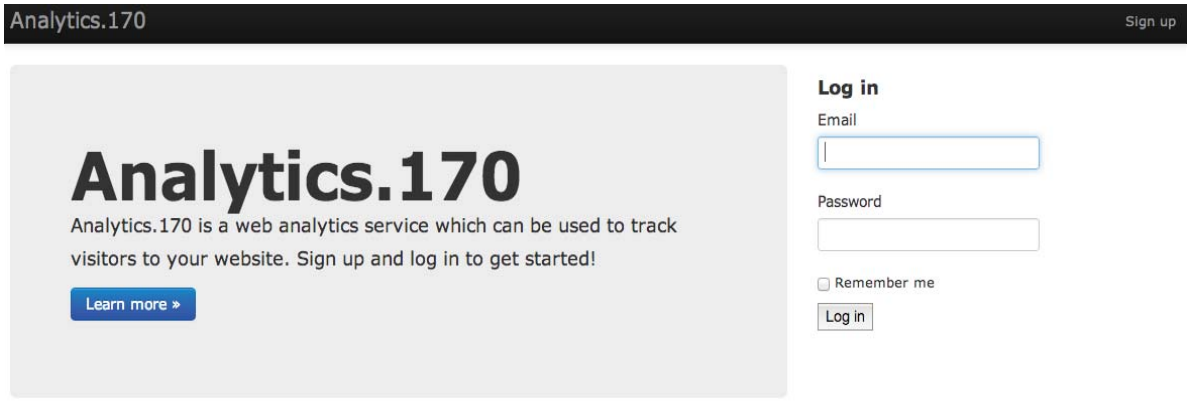
```
<!DOCTYPE html>
```

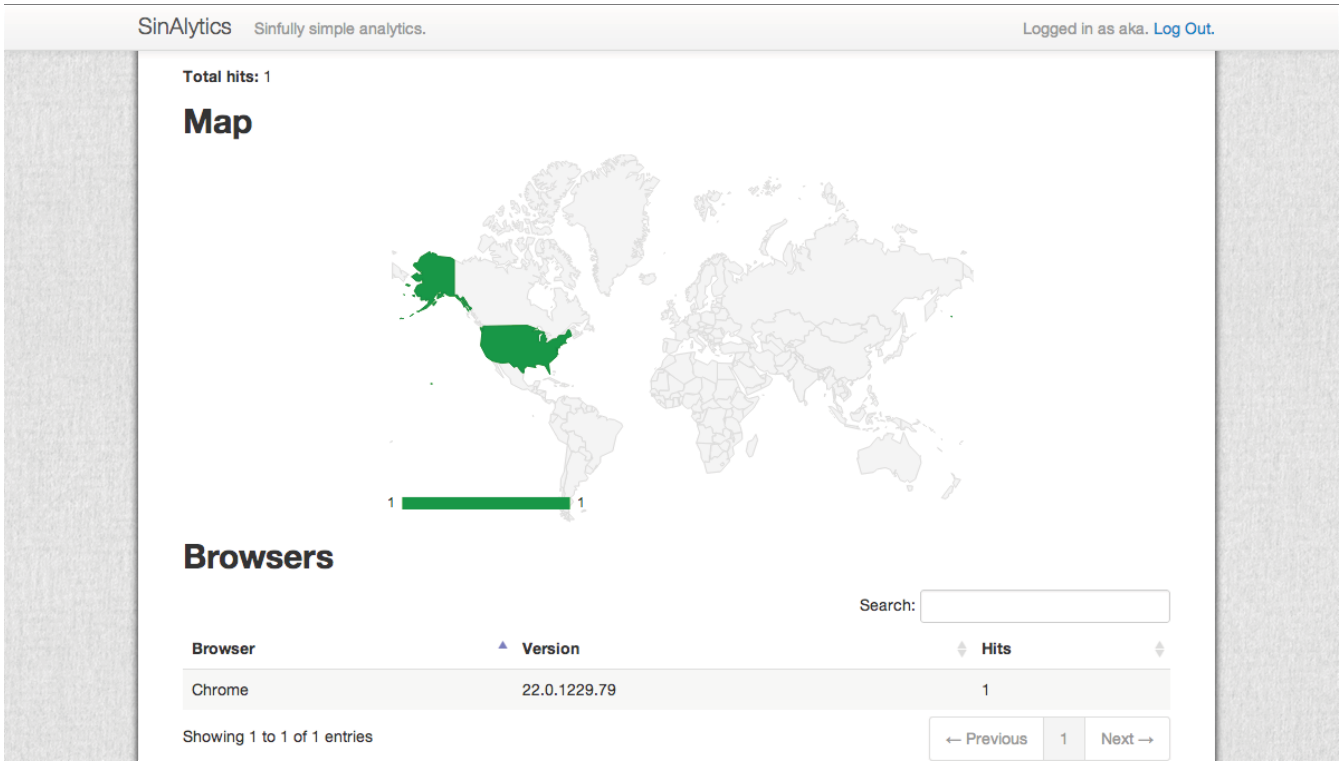
Topic 2: CSS

Here are screenshots of some sample P1.2's



The functionality is present in all these pages, but the sites themselves are visually unappealing. (In fact, perhaps good visual design is essential to your site's functionality - e.g. people abandoned MySpace as a social networking tool since they couldn't tolerate looking at the cluttered, eye-soring profiles vs. Facebook's incredibly clean [c. 2007] layout.) Maybe we can find a way to style the pages so that our sites could look... like this!


Analytics.170
Sign up
Analytics.170
Analytics.170 is a web analytics service which can be used to track visitors to your website. Sign up and log in to get started!
Learn more »
Log in
Email
Password
Remember me
Log in


SinAlytics
Sinfully simple analytics.
Logged in as aka. Log Out.
Total hits: 1
Map
1
1
Browsers
Search:
Browser
Version
Hits
Chrome
22.0.1229.79
1
Showing 1 to 1 of 1 entries
← Previous
1
Next →

Let's look at an HTML page about whales:

Welcome to my awesome page on whales!

Here are some super cool whale facts!

- Whales are mammals.
- Scientists think the blue whale is the largest animal ever!
- Blue whales uses special bones in its mouth called baleen plates to eat super small organisms like krill!
- I like whales.

But this isn't the best looking page on whales. One way to style it could be by manually editing each HTML element with HTML attributes. However, now you're mixing concerns about *what* data should be presented and *how* that data should be presented in the same file. The better way is to create a separate, external file called a .css file which specifies how certain data on the whale page should be presented.

What do .css files actually do?

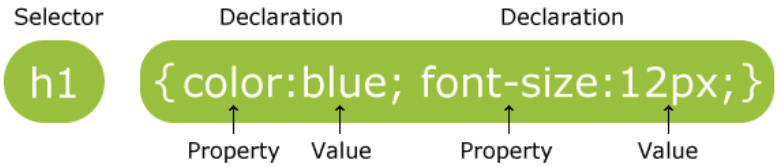
CSS stands for **C**ascading **S**tyle **S**heets. And .css files are just that: stylesheets. Your browser has a default way of rendering certain HTML elements. But if you link an HTML page to a .css file, the browser can parse that .css file and then override its native way of rendering HTML data.

CSS syntax

CSS files as a regular grammar:

```
CSS FILE ::= (RULE)*
RULE ::= (SELECTOR)+ (DECLARATION)*
DECLARATION ::= PROPERTY (VALUE)+
```

CSS files are a series of zero or more presentation rules. Here's a sample rule:



(from *http://w3schools.com/css/css_syntax.asp*)

This rule demands that content between <h1> tags be blue and of font-size 12 pixels.

How does the browser know which HTML elements a rule targets? Notice that a rule consists of 1 or more selectors. Selectors specify which elements the presentation rule is targeting. In this example, the presentation rule is targeting all <h1> HTML elements.

Most common selectors

`htmlElementType`	targets all HTML elements of type `htmlElementType` (e.g. h1, span, div, etc.)
`.className`	targets elements of class `className`
`#idName`	targets one element with id `idName` (assuming HTML is well-formed - ids should be unique per document!)
`*`	targets all elements
`selector1, selector2`	targets all elements specified by **either** `selector1` or `selector2`
`selector1 selector2`	targets all elements specified by `selector2` that are children (both direct and indirect) of elements specified by `selector1`
`selector1[attr=value]`	targets all elements specified by selector1 with a specified attribute, e.g. `input[type=password]`

Selectors can also be combined! For example,
`img.classA.classB#idName[alt="Profile image"]`

There are a lot more ways to select elements; for a complete list check out the W3C specs: http://www.w3.org/TR/selectors/.

How does the browser know how to display particular selected elements? Notice that a rule also consists of declarations, which are simply property:value pairs. The browser renders targeted elements based on these property:value pairs. There are lots of properties - no need to memorize all of them. Check a reference like the Mozilla Developer Network (MDN:

https://developer.mozilla.org/en-US/docs/CSS/CSS_Reference) to find the properties (and the possible values that can be associated with those properties) you need in order to implement a particular styling.

CSS let's you change almost everything about a page's appearance. Properties range from typographical (font-size, letter-spacing, line-height, font-weight, font-style) to appearance (color, opacity, backgrounds, border) to positional ("position", top/left/right/bottom) to size (height, width, padding, margin) and more.

Let's now style the whale page with what we know!

1. Create a new file called whale.css.
2. Link to it whale.html by including <link rel="stylesheet" href="/path/to/whale.css" /> in whale.html's header.
3. Give the "Welcome..." <p> element the id "welcome". Add a presentation rule to whale.css that makes this element's content larger, Arial font, centered, and of dark blue color. For example:

```
#welcome {
      font-family:Arial;
      font-size:36px;
      color: #3B6AA0;
      text-align: center;
}
```

Now, whale page looks like:

4. Give the whale picture <img> element the class “whale_pic”. Then, using CSS, center the whale picture and give it rounded corners! For example:

```
.whale_pic {
      display:block;
      -webkit-border-radius: 20px;
      -moz-border-radius: 20px;
      border-radius: 20px;
      margin-left: auto;
      margin-right: auto;
}
```

Note: Margin-left and margin-right are equal to ‘auto’ so the browser calculates the left and right margins of the .whale_pic element on the fly as the browser is resized.

Now whale page looks like...

Welcome to my awesome page on whales!

Here are some super cool whale facts!

- Whales are mammals.
- Scientists think the blue whale is the largest animal ever!
- Blue whales uses special bones in its mouth called baleen plates to eat super small organisms like krill!
- I like whales.

5. Cool. Now let's give the page some good background color!

```
body {
       background-color: #E0EEEE;
}
```

6. Cool. Now let's style the actual text of the page.

```
p, ul {
	font-family: Arial;
	color: #3B6AA0;
}
```

Whale page now looks like:

7. Now our whale page is marginally better looking. But wait! You now realize there's some redundancy in the CSS. We can get rid of the color and font-family declarations in the #welcome CSS rule since they're covered by the p presentation rule. Identifying and getting rid of these redundancies can allow us to write cleaner CSS.

Topic 3: Specificity and the “Cascade” in CSS:

So we’ve just seen CSS in action, but...
What happens if there are rule conflicts (like if there are multiple stylesheets that target the same elements)?

This is the **C**ascading part of the style sheets. There’s a particular order of precedence for applying stylesheets (information obtained from: http://bit.ly/5uu3iX [vanseo design], http://bit.ly/bisLP [w3c css2 spec]). When applying this order of precedence, the stylesheets cascade into each other, functionally creating a new stylesheet.

1. Get all CSS declarations from all stylesheets that apply to a particular property for a particular element.

2. Break ties by origin and weight.

What do you mean an origin?

Your *browser* has a default stylesheet. A *user* can specify his/her own stylesheet for his/her browser (for example, if s/he needs a particular stylesheet to see the content better due to medical concerns). And then of course there are the style sheets the *author* of an HTML page explicitly links to. So there are three possible stylesheet origins: browser, user, and author.

What do you mean by weight?

When you write a declaration like

h1 {color: blue;}

you can add **!important** after the value in the declaration to specify that that particular declaration should have some overriding power. For example:

h1 {color: blue !important;}.

So if your HTML page links to one stylesheet that says

h1 {color: red;}

and another that says

h1 {color: blue ! important;}

then the latter rule trumps over the former.

Note: !important is considered bad practice - it's usually a sign that your original classes, IDs, and styles were not well thought out and you're brute-forcing things.

The order of precedence in terms of origin and weight is: user important overrides author important which overrides author unimportant which overrides user unimportant which overrides any browser rules.

U Important >> A Important >> A Unimportant >> U Unimportant >> Browser

3. Break further ties by specificity.

What do you mean by specificity?

Intuition test - Assume we had an <h1> element with class "intro". Assume the following rules were in the same stylesheet; which trumps which?

```
h1 {color: red;}
h1.intro {color: blue;}
```

The rules are from the same kind of origin and are both unimportant. We break the tie with specificity and decide that the second rule wins out. But what about the following case?

```
h1 p.intro {color: red;}
#title p {color: blue;}
```

Assume we had an element that satisfied both selectors (e.g. a paragraph element of class "intro" that was embedded within an h1 element with id "title"). Which rules wins out? Not so obvious! Let's see how browsers (roughly) calculate CSS specificity. For more details, check out a CSS book. :)

Each declaration is assigned a tuple (*i*, *c*, *e*) [ice is easy to remember :)] where *i* is the number of ids specified by the selector, *c* is the number of classes specified by the selector, and *e* is the number of HTML element types specified by the selector. For example,

p → (0, 0, 1)
h1 → (0, 0, 1)
#title → (1, 0, 0)
h1#title → (1, 0, 1)
h1#title p → (1, 0, 2)
h1#title p.intro → (1, 1, 2)

So given two selectors *s1* and *s2* with corresponding ice tuples *t1* and *t2*. To determine which selector is more specific, iterate through *t1* and *t2* one index at a time. If at any point, *t1*'s value at that index is greater than *t2*'s, then return: *s1* is the more specific selector. If *t1*'s value at that index is equal to *t2*'s value, then iterate to the next index.

For example, given the selectors above (p, h1, #title, h1#title, h1#title p, and h1#title p.intro), first we can weed out p and h1 since they have no ids specified in their selector, but all the other selectors have 1 id specified. But then of the remaining selectors, only h1#title p.intro has a class specified, so it is the most specific selector.

So who wins here?

h1 p.intro {color: red;} (0, 1, 2)
#title p {color: blue;} (1, 0, 1)

Yeah. #title p. IDs are very "powerful" when it comes to specificity - **use them sparingly**. You should rarely have a reason to use an ID, and you should use classes almost always.

4. Break any *remaining* ties with the declaration declared most "recently" (*bottom of the CSS file*).

So if in a particular stylesheet you have the following rules:

h1 {color: blue;}
h1 {color: red;}

The second rule trumps over the 1st since it was declared more "recently". It's analogous to doing

x = 1
x = 2

in a programming language.

Asset tag helpers provide methods for generating HTML that link views to stylesheets. Stylesheets is one of the six asset tag helpers available in Rails:

- `auto_discovery_link_tag`
- `javascript_include_tag`
- `stylesheet_link_tag`
- `image_tag`
- `video_tag`
- `audio_tag`

If you have two separate author stylesheets that both have a rule targeting the same element...like:

<link rel="stylesheet" type="text/css" href="css/whale1.css" />
<link rel="stylesheet" type="text/css" href="css/whale2.css" />

Then the rule declared in whale2.css wins.

So within a stylesheet, the last declared rule wins. Among different stylesheets of the same origin, the rule in the last linked stylesheet wins. **Although these rules hold true, do not rely on them in your CSS. It leads to brittle code where re-ordering files or compressing them together can break your styles.**

Finally, inline CSS inside HTML attributes win over CSS declared in the <head> which wins over CSS linked to in external stylesheets.

Topic 4: CSS Box Model

So we now have a relatively good understanding of how CSS is used and how it's processed. Let's focus on a specific, often-used case of applying CSS: positioning, aligning elements, or giving elements certain degrees of margin/padding. But these properties can be tricky to work with. Trying to set these properties with CSS without an understanding of the model underlying these properties can lead to problems when debugging your layout (e.g. why won't this element center? why can't I align all these input elements? etc). So let's learn about this model!

In terms of rendering, every HTML element is constrained within a rectangular box. Here, inspect the elements of any webpage with a browser's element inspector to show students the rectangular boxes.

Here's the box model in a nutshell:

Margin vs padding: margin and padding seem to be very similar - they add "space" around an HTML element. There are some important differences, though. As you can see from above, a larger padding will push the border out. Additionally, the background will still appear in the "padding" of the box. So think of padding as part of the element, whereas margin is space around it.

The width/height attributes of an HTML element only correspond to the content box, which contains text/images/etc. To get the full dimensions of an HTML element, you'll have to add the width/height of the content plus the padding plus the border plus the appropriate margins.

You can add margins in several ways:

```
margin: 5px; // Adds margin on all sides
```

```
margin-top: 7px; // Only top
margin-[bottom|left|right]: 3px; // Similar for other sides
margin: 3px 5px; // Sets top and bottom to 3px, left and right to 5px
margin: 3px 5px 6px 7px; // Sets top right bottom left (clockwise order)
margin: 3px 5px 6px; // Sets top to 3px, left and right to 5px, and bottom to 6px
```

Modify the #welcome presentation rule in whale.css to:

```
#welcome {
    font-size: 36px;
    text-align: center;
    padding: 10px;
    border: solid;
}
```

Then inspect the “Welcome to my page...” element in the whale page in the browser. Note how all four components of the box model for the #welcome element are highlighted.

Topic 5: Linking CSS to Rails Templates

Cool. So that’s a good amount of CSS background. Before we do some practice exercises, let’s go over how you’d actually link a Rails template to a CSS file.

1) Place the appropriate CSS file in your app/assets/stylesheets directory.
2) In your layout file, use the following Rails code:

```
<%= stylesheet_link_tag "main", "/photos/columns" %>
```

This will link your layout file to app/assets/stylesheets/main.css and app/assets/stylesheets/photos/columns.css.

And that’s it!

Topic 6: Additional Notes

- In Rails, SCSS/SASS provide simple but useful extensions like variables and mixins to CSS which help “DRY”
- There are many useful stylesheets available online. http://cssgrid.net/ is a nice one which is available for free!

Next Tutorial: JQuery

6.170 Software Studio
Spring 2013