Web Fundamentals - Session 2

Chapter 1 - CSS

Recap

- A Website is just a collection of files: HTML, CSS & Javascript.
- We can edit these files locally.
- We can create them locally (using a text editor) and view then in a browser.

Task:

1. Go to the github repository for this session: https://github.com/code61/html2
2. Download the code into your coding_course folder (by clicking 'Download ZIP' in the bottom right).

What is CSS?

CSS (Cascading Style Sheets) was created by the World Wide Web Consortium (W3C) to solve to problem of formatting and styling a document. This allows HTML to do its job - defining the content of a document.

A powerful example of this is CSS Zen Garden - by clicking on the links you completely change how the site looks, but the HTML remains unchanged.

Linking your CSS and HTML code

There are a few ways to include CSS in an HTML file:

1. Put the css inline in the html.
2. Put the css in a <style>..</style> section in the <head>
3. Link to a separate .css file in the <head>

The first way is sometimes useful, but defeats the point of using CSS to separate presentation and information. The second way is a bit better, but is strictly something you should never do in a professional website. The third way is the best, and we'll be doing this.

Link to a separate CSS file in the <head>

By separating these CSS rules into their own file you:

(a) reduce repetition in your code and (b) reduce the amount of information that has to be sent to the browser for each page - if the CSS file applies to the whole site, it only needs to be sent to the visitor once.

To link to an external CSS file you can do the following:

```
<!DOCTYPE html>
<html>
  <head>
        <title>My page</title>
        <link rel='stylesheet' type='text/css' href='path/to/my_css_file.css'>
  </head>

  <!-- body goes here -->

</html>
```

More about links below.

More about the HTML <link> Tag

Linking to other files (stylesheets, javascript files, images) can be done in several ways, just like linking to another page. Say you have the following directory structure:

```
first_site
---- index.html
---- images
|    ---- background.jgp
---- stylesheets
|    ---- main.css
```

and you're going to deploy your site to "http://www.my_first_site.com". Suppose you want to link to main.css from index.html and to background.jpg from main.css. There are three different styles of links you can use:

1. Absolute links

Absolute external links include the complete url to the resource you're linking to. **Absolute links start with either http:// or https://**.

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="http://www.my_site.com/stylesheets/main.css">
/* in main.css */
body {
  background-image: url("http://www.my_site.com/images/background.jpg");
}
```

Absolute external links can be used to link to resources held on different sites, but wouldn't usually be used for links within your own site.

They're a bit fragile - if you change your domain name all the links will break. They also won't work when you're developing locally.

2. Root-relative links

Root-relative links contain the path to the resource *relative to the site's root*. The site's root is (roughly) the folder that contains the site - in this case, first_site. **Root-relative links begin with a /**:

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="/stylesheets/main.css">
/* in main.css */
body {
  background-image: url("/images/background.jpg");
}
```

Root-relative links are a bit more flexible than absolute external links: e.g. if you change your domain name everything will still be fine. They're sometimes useful for your own static sites, but probably won't work when developing locally (because the root will be taken to be the root of your file system and not the folder containing the site!).

3. Document-relative links

Document-relative links contain the path to the resource *relative to the file where the link is written*. **Document-relative links don't begin with /**:

```
<!-- in index.html -->
<link rel="stylesheet" type="text/css" href="stylesheets/main.css">
/* in main.css */
body {
  background-image: url("../images/background.jpg");
}
```

To link to the stylesheet from index.html we use stylesheets/main.css which says "look in the same folder I'm in (first_site) and find a folder called styleheets and a file in it called main.css".

To link to the image from the stylesheet is slightly more complicated: we use ../images/background.jpg. This means "go to the folder above the one I'm in (I'm in stylesheets, so that's first_site) and find a folder called images and a file in it called background.jpg".

Important to know.

In document-relative links (and in many other places e.g. command line navigation):

- . means in the folder **that I'm in**
- .. means in the folder **above the one that I'm in**

Relative links are the most flexible - they will work on your local file system. The only think you have to be careful about is moving your files into different folders, which can cause links to break.

You should be using relative local links in these exercises.

Task:

1. Open exercise1.html in Sublime Text and in Chrome, and make a separate CSS file, called exercise1.css.
2. Link your exercise1.css file to your exercise1.html file within the <head> tag.

Start writing some CSS & Some Basic Definitions

Follow along with the demo and start writing your css in exercise1.css (option 3) it should look something like this:

```
h1 {
color: red;
}
```

This is called a **rule set** - a single section of CSS including a *selector*, *curly braces*, and a *declaration* consisting of *properties* and *values*.

The h1 bit specifies the tag that will be styled, this is a CSS **selector**.

The bit inside the { ... } specifies the styles that will be applied, this is the **declaration**. Here we change the colour of the h1 text red. If there is more than one declaration within the curly bracket (e.g. make the size of the h1 text 30px), they are all collectively called a **declaration block**.

The **property** of HTML being changed in the example above is "color", and the **value** it is being changed to is "red".

If you save the changes to the css file, then open (or refresh) the page in your browser you should see the changes. By opening the developer tools, and hovering over the h1 you should be able to see your css rule at the side.

A few more properties

In the example above we changed the color of the h1 element. Here are a few examples of other simple properties you can try out while you're getting the hang of css:

```
p {
        font-family: 'Arial';
}
h2 {
        font-size: 20px;
}
```

```
h3 {
    background-color: green;
    font-size: 2px;
}
```

The last example, of the h3 rule set, is a demonstration of a **declaration block** - when you specify more than one declaration for a selector.

Task:

1. Open exercise1.css in Sublime Text and in Chrome.
2. Make the h1 turn red.
3. Continue with the exercise until exercise1.html looks like exercise1_solution.png.

Chapter 2 - Selectors and Attributes

Corresponding DEMO. So far you've used html tags to specify CSS rules. For example,

```
h2 {
  font-size: 40px;
  color: pink;
}
```

will turn all your <h2> massive and pink. This is called an **element type** selector; when a selector targets an HTML element directly by the **tag name**.

It is often useful to be able to make CSS rules more specific, so you can apply different styles to different parts of the page. One common way to do this is to target specific HTML attributes - in particular, the ID and class selectors.

Using the ID and class selectors

HTML tags can have *attributes* which provide additional information about the element. (This is **completely different** from a CSS attribute!) You've already seen some examples of this:

```
<a href="http://www.facebook.com">
<img src="cat_pic.png">
```

Recap: Both href and src are examples of html attributes. They are written in pairs with their values: attribute="value".

There are some attributes that can be added to any tag. Two examples of these are id and class:

```
<h2 id="products_title">Our scrumptious puddings</h2>
<ul id="products_list">
  <li class="product_item">Black forrest gateau</li>
  <li class="product_item">Rasberry lemon swirl cheesecake</li>
  <li class="product_item">Sticky toffee pudding</li>
  <li class="product_item">Death-by-chocolate cake</li>
</ul>
```

IMPORTANT!

Both ID and class are used to add some information to the HTML tags. The key difference is that ID should specify a ***unique*** *element on the page*, whereas multiple elements can **share** the same class. This is useful when you have loads of code to sift through.

CSS lets you target the id and class attributes in special ways:

```
/* make the h2 with id="products_title" purple */

h2#products_title { color: purple; }

/* remove the bullets from all li with class="product_item" */

li.product_item { list-style-type: none; }
```

The id is targeted by adding #id_value to the tag and the class is targeted by adding a .class_value to the tag.

It is also possible to target any items with a given class or id by leaving out the HTML tag:

```
/* make any element with id="products_title" purple */

#products_title { color: purple; }

/* make any element with class="product_item blue" */

.product_item { color: blue; }
```

HTML <divs> and <spans>

There are two important HTML tags, that we didn't use last week: and . Both are really useful when it comes to using HTML attributes to target CSS classes.

 stands for *division* and is used to break the page up into different parts. It is a 'block-level' element, which means that it will start a new line before and after it.

 can be used to apply classes and ids to certain bits of text. It is an 'inline' element, which won't start a new paragraph before or after.

```
<div id='info_section'>

  <p>This is a paragraph in the info section. We can use a span to target <span
class='important'>certain bits of important text</span>.<p>

</div>
```

The Universal Selector

The universal selector matches any element within the context in which it's placed in a selector. The * character is the universal selector:

```
* {

        font-family: 'Helvetica', sans-serif;

}
```

This should only be used for declaring things on the entire document, otherwise it slows performance.

To find out more about the different kind of selectors, read this article.

Task:

1. Open the file html2/exercise2.html in Sublime Text and Chrome.
2. Separate the CSS in the <head> tags into a linked CSS file.
3. Continue until your page looks like html2/exercise2_solution.png

Web Fundamentals - Session 2

Version Control & Using GitHub

You saw this guide and this guide last week.

Taken from this guide: **Version Control** (aka Revision Control aka Source Control) lets you track your files over time. Why do you care? So when you mess up you can easily get back to a previous working version.

You've probably cooked up your own version control system. Got any files like this?

- ResumeOct2006.doc
- ResumeMar2007.doc
- logo3.png
- logo4.png
- logo-old.png

It's why we use "Save As". You want the new file without obliterating the old one. It's a common problem, and solutions are usually like this:

- Make a **single backup copy** (Document.old.txt).
- If we're clever, we add a **version number or date**: Document_V1.txt, DocumentMarch2007.txt
- We may even use a **shared folder** so other people can see and edit files without sending them over email. Hopefully they relabel the file after they save it.

So Why Do We Need A Version Control System (VCS)?

Shared folder/naming system is fine for class projects or one-time papers. But for software projects? Not a chance.

Do you think the Windows source code sits in a shared folder like "Windows2007-Latest-UPDATED!!", for anyone to edit? That every programmer just works in a different subfolder? No way.

Large, fast-changing projects with many authors need a **Version Control System** (aka for "**file database**") to track changes and avoid general chaos. A good VCS does the following things:

- **Backup and Restore.** Files are saved as they are edited, and you can jump to any moment in time. Need that file as it was on Feb 23, 2007? No problem.
- **Synchronization.** Lets people share files and stay up-to-date with the latest version.
- **Short-term undo.** Monkeying with a file and messed it up? (That's just like you, isn't it?). Throw away your changes and go back to the "last known good" version in the database.

- **Long-term undo.** Sometimes we mess up bad. Suppose you made a change a year ago, and it had a bug. Jump back to the old version, and see what change was made that day.
- **Track Changes**. As files are updated, you can leave messages explaining why the change happened (stored in the VCS, not the file). This makes it easy to see how a file is evolving over time, and why.
- **Track Ownership.** A VCS tags every change with the name of the person who made it. Helpful for blamestorming giving credit.
- **Sandboxing**, or insurance against yourself. Making a big change? You can make temporary changes in an isolated area, test and work out the kinks before "checking in" your changes.
- **Branching and merging**. A larger sandbox. You can **branch** a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can **merge** your work back into the common area.

Shared folders are quick and simple, but can't beat these features.

Task: Complete the guides and finish them if you haven't already.

Putting your site online

We're going to use GitHub Pages to host our site - it will provide the server where the files for our site is stored. This will be in the Homework.

GitHub provides a developer pack from students as well, here, including discounts on DNS management, a GitHub account upgrade (for free), Microsoft Visual Studio, and one year free on a domain name from namecheap.

Web Fundamentals - Session 2

Session 2 Homework

Finishing off

Task:

Finish off both CSS exercises from class. Check your solutions online:

- Find the HTML2 repository on Code61's github page.
- In the branch drop-down (just above the list of files) select the solution branch.
- Click on the files in the branch to see the solution

Putting your site online

Task: Use your GitHub account to follow the guides here and here.

Your site will currently show up with a the url of [your-github-username].github.io. In order to use your own domain name, you have to buy one from a domain name registrar, as we covered in the previous section.

More HTML/CSS & Preparation

Task:

1. Complete the whole of **Project 3** on the General Assembly Dash site.
2. Make some **changes** to your first_site based on what you've learnt. Update the online version of your site, by re-uploading to GitHub Pages.
3. **Find out** what these development concepts are:
   - A framework
   - An API
   - A Library
   - A Toolkit
4. (**Optional**) Do the projects from the Codecademy Web Track Sections 7, 8 & 9.
5. (**Optional**) Read this article about absolute vs. relative links.

Resources

Definitions of CSS Terms , W3 Schools CSS Tutorial , Shay Howe's HTML & CSS Tutorial/Guide

How-To Geek explains GitHub , GitHub Guides

An introduction to Version Control

Another (Visual) Introduction to Version Control