

# 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 them in a browser.

### Task:

1. Go to the github repository for this session: <https://github.com/CodeFirstGirls/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 the 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:

## Web Fundamentals – Session 2

(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='exercise1.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.jpg
---- 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");
```

## Web Fundamentals – Session 2

}

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 `styleheets`, 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 thing 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

DEMO by forking [this pen](#) and typing in the CSS (below) yourself during class. This demo follows the exercise they have downloaded from GitHub. Students will finish it according to the picture they have downloaded as well.

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 {
```

## Web Fundamentals – Session 2

```
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

[DEMO, type in CSS](#) 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 – particularly, the **ID** & **class** selectors. You can also combine 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">  

```

**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">Raspberry 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.

## Web Fundamentals – Session 2

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 `<div>` and `<span>`

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

`<div>` 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.

`<span>` 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, using it in a combination of selectors is redundant..

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`

## Competition Guidelines

The CF:G competition is a chance for students to put into practice the skills learnt on the course. Students can work individually or in groups of 2–3 to create a landing page for a website. **Aim to form teams by Week 3, and collaborate on HowCloud or Facebook.**

The criteria for the competition are here:

- A visually appealing design – good use of CSS, HTML elements, and Twitter Bootstrap
- Good formatting
  - Code split into the appropriate files
  - Files indented properly
- A live website (Github page, Heroku or own domain)
- Extras e.g:
  - A contact form (for example name and email)
  - Social buttons
  - Widgets
  - As many different HTML elements you can manage

Interactive elements (like forms) on the website don't need to be functional, but should be present if they need to be for the visual aspect of the design.

- (optional) Good organisation
  - Version control using GitHub
  - Sensible commit messages

**Some of the winning entries from last term's competition can be found [here](#).**

There'll be prizes of Amazon vouchers for the winning team and a chance to be pitched alongside the winning entries from courses around the U.K. for a grand prize.



## Session 2 Homework

### Finishing off

#### Task:

1. Read [this guide](#) and [this guide](#) on how to use GitHub and version control – **This is vital.**
2. Watch this more [in-depth video](#) about how the internet works, for Front-End Devs
3. Finish off both CSS exercises from class. Check your solutions online:
  - Find the HTML2 repository on CodeFirstGirls' 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

### More HTML/CSS & Preparation

#### Task:

1. Complete the whole of **Project 3** on the [General Assembly Dash](#) site.
2. **Find out** what these development concepts are:
  - A framework
  - An API
  - A Library
  - A Toolkit
3. **(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](#)

[This article](#) which explains about CSS Specificity (and more).

[This article](#) has more information on CSS selectors.

[A CSS Validator](#)