CodeFirst:Girls Beginners Coding course – Front end Web development

Week 2 - Cascading Style Sheets or "CSS"

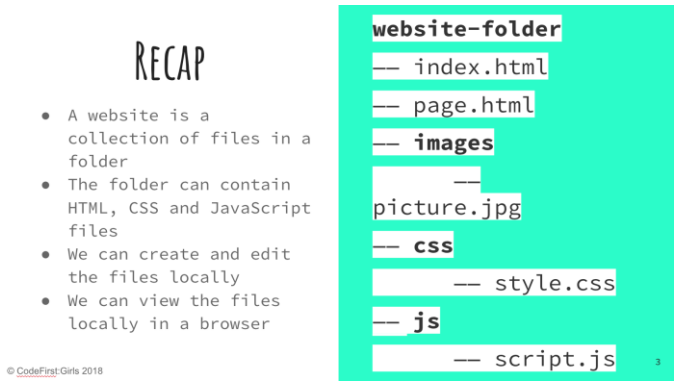
Note to instructors!

The majority of this lesson is a 'code-along'. The purpose of this is the ensure that by the end of the session (or the beginning of next session if you overrun), all students have a completed static website. This is really useful for some students to use as a reference to use when starting their personal projects. We will also go back to this website in the github tutorial, and will use this one to demo 'publishing to github pages'. Then, by the end of the course, all students should have two published websites, their personal landing page, which they developed in this session, and their group project website. It's also a great opportunity to walk through development practices and concepts that some may be struggling with.

Important things to note:

- **Make sure everyone starts with finished solution for week one, this was the homework so hopefully everyone will have done (there are instructions below if they haven't)**
- **Make sure you have read through the session thoroughly before the lesson starts.**

- **Try and avoid switching windows too much, try and have your text editor and web browser, or text editor and the slides on the same page.**
- **Explain each step thoroughly and allow for questions**
- **Don't panic if you don't finish - but make sure that everyone finishes it for homework**
- **Encourage students to personalise and change the website as they feel fit for homework (there are some alternative CSS solutions for different designs on github)**

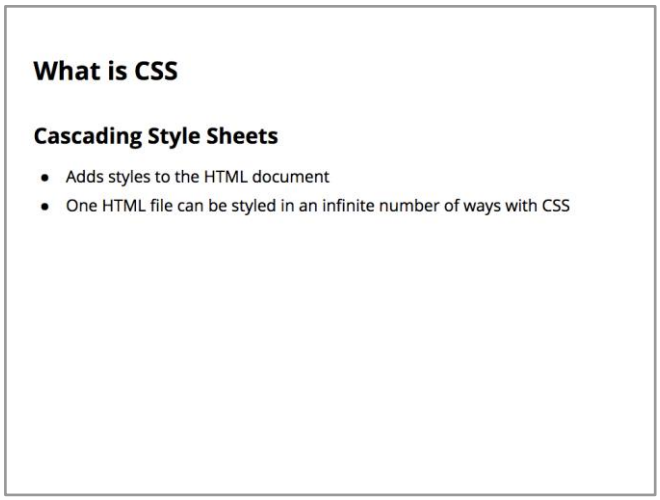


What is CSS?

Let's recap what we have learned so far.

- A website is a collection of files in a folder. The folder can contain HTML files, CSS files and JavaScript files.
- All these files can be created and edited locally using a text editor on our machine.
- We can view these files locally too using a browser.

The HTML we have written up until now doesn't look very exciting when viewed in a browser. That's because every browser will add basic styling to all HTML elements.



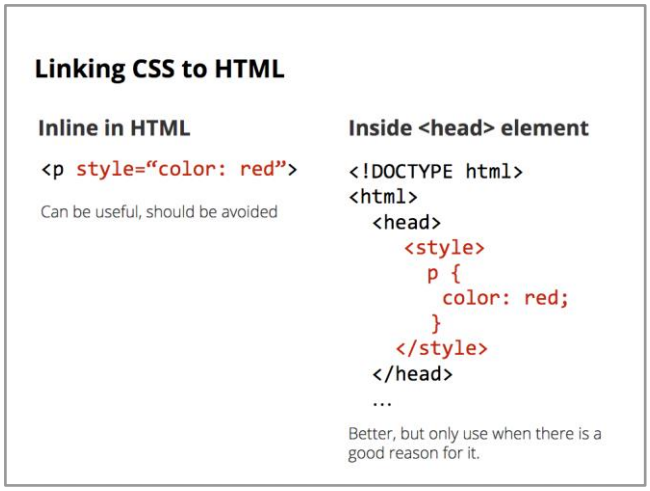
To improve the styling of HTML is where CSS comes in. It stands for **Cascading Style Sheets** and was created by the World Wide Web Consortium, or W3C for short, to solve the problem of formatting and styling a document. This way HTML documents can concentrate on defining the content of the web page and CSS files can concentrate on the styling of the web page.

The same HTML document can be styled in an infinite number of ways by changing the CSS. A powerful example of this is the CSS Zen Garden. It's a website which was created in 2003 and showcases what is possible with CSS based design. It contains hundreds of different designs, each one made up of exactly the same HTML, but for each one the CSS is different. If you browse through the collection you can see what is possible with CSS. Use the web inspector to find out all the clever CSS.

Linking to a separate CSS file in the <head>

We put the CSS in a completely separate file, and them link that to the head of the HTML file, so that it's loaded before the rest of the content of the page. Though there are other ways to do this, this is best practice. There are many benefits of doing it this way:

- Making updates to your CSS is much easier. All the CSS sits together in one file (or several sometimes) and is uncluttered by other code.
- It reduces the need to repeat your code. For example if every page on your website contains a header section you want to look the same on every page, then using a separate CSS file means you only need to write the code for this once. It reduces the amount of information that has to be sent to a browser for each page. If the CSS applies to a whole site, the CSS file will only be sent once .

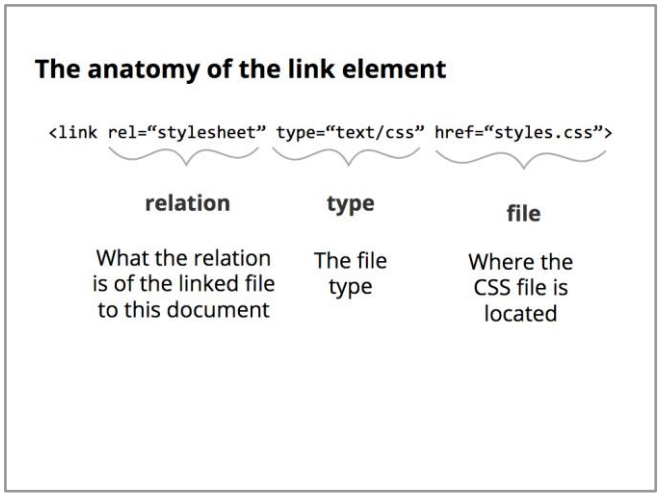


This is the bad way to do it

Of course, the other example isn't the only way to add css. This is the old school way to do it, and should only be done in an emergency!

- You can add the CSS directly to an HTML element as an attribute. This can be useful in certain cases, but should be avoided.
- You can also add the CSS in your HTML file in the <head> section of the page. The CSS would then be contained inside the style element between <style> and </style> tags. This can be useful in more cases, but also should be avoided.

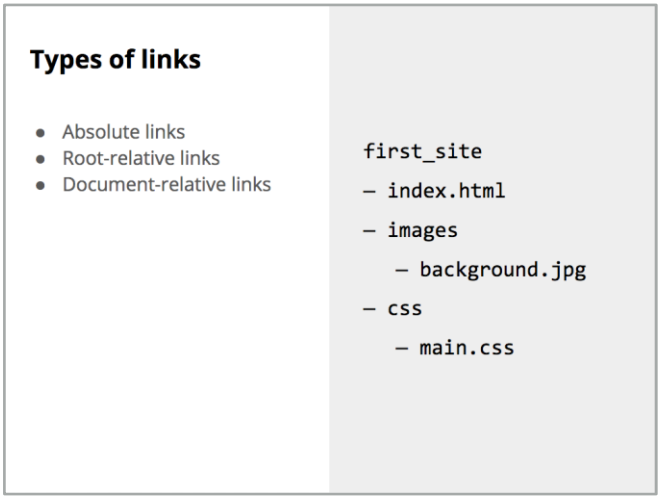
For the exercises you will do in this course you should write your CSS in a separate file.



The anatomy of the link element

A separate CSS file is always linked in an HTML file in the **<head>** section, using the **<link>** element which is an all-in-one element. Therefore it contains several attributes to give the browser the information it requires.

- The **rel attribute** stands for Relation and indicates what the relation of this linked file is to the document in which it is linked. In this case the relationship is stylesheet.
- The **type attribute** indicates what type of file is being linked.
- The **href attribute** we saw earlier and this contains the filename of the CSS document.

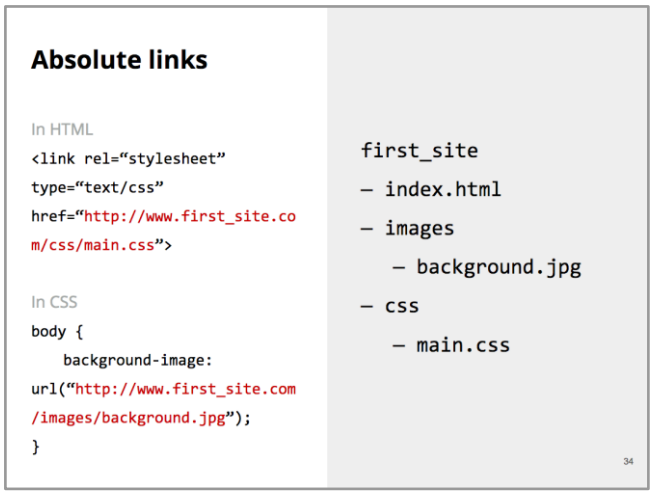


Types of links

The value of the href attribute not only needs to contain the filename of the CSS document, but it also needs to contain the correct path for the file. The file path depends on your website folder structure.

There are three different ways you can link your main.css file to your index.html file

- Absolute links
- Root-relative links
- Document-relative links



Absolute links

An absolute link is a link that contains the full url to a resource you are linking to. They will start with either http:// or https://

Imagine that you are hosting your first_site on a domain name with the url http://www.first_site.com and you want to link your main.css file to your index.html file using an absolute link.

To link the CSS file to your index.html file you need to include the full path to where your CSS file is located. In this case the main.css file is inside a folder called css, which sits inside your first_site folder. The full path would therefore be http://www.first_site.com/css/main.css

In the **<head>** element of your HTML file you would put this:

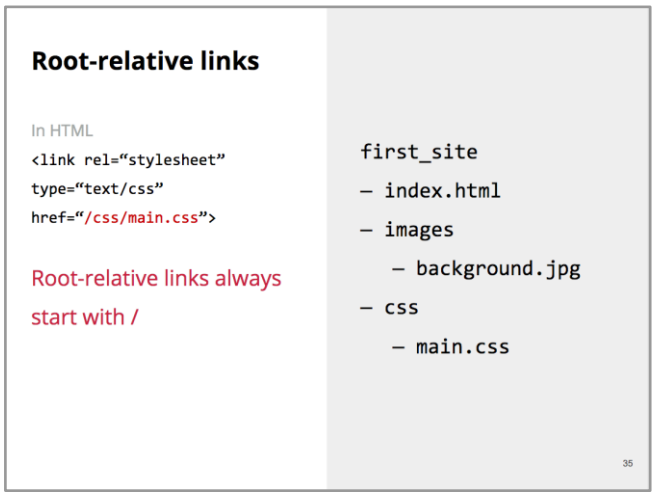
```
<link rel="stylesheet" type="text/css"
href="http://www.first_site.com/css/main.css">
```

You can use absolute links in your CSS as well. If you want to add your background image in CSS you can use the same method to create an absolute link to there.

```
body {
      background-image:
url('http://www.first_site.com/images/background.jpg');
}
```

Absolute links can be used to link to resources on your own website, but they are usually used when you are linking to external resources on other websites. They can be a bit fragile. If the file you are linking to is moved to a different location, or deleted, then your link will break. Or if you change your domain name you need to update all your absolute links.

Absolute links to resources on your own website also don't work when you are developing locally because you don't have a local URL to link to.



Root-relative links

Root-relative links are links that contain the path to the resource relative to the website root. The root of a website is the folder which contains the site. In our example that is first_site.

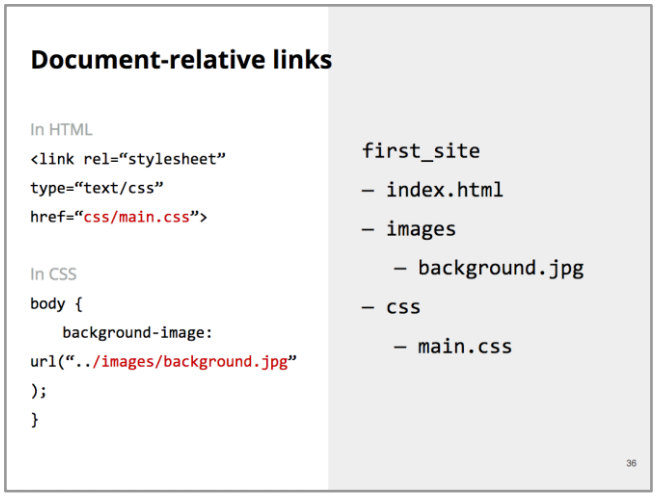
A root-relative link always begins with a forward slash: /

Using a root-relative link to link your CSS file to your HTML file your link element will look like this:

```
<link rel="stylesheet" type="text/css" href="/css/main.css">
```

You would use root-relative links to link to files on your own domain. They are more flexible than absolute links. If you change your domain name everything will still be fine.

On static websites root-relative links can be useful, but they probably won't work when you are developing locally. And that's because your operating system considers the root folder to be the root of your file system and not your website folder. This is usually the user folder of your operating system.



Document-relative links

Document-relative links contain the path to the resource you are linking to, relative to the file where the link is written.

In our example if you want to link to main.css from index.html you would write this:

```
<link rel="stylesheet" type="text/css" href="css/main.css">
```

What your are telling the browser here is to look for a folder called css/, in the same folder index.html is in, and then inside the css/ folder to find a file called main.css
If you want to use document-relative links in CSS it's a little more complicated. In our example the background image is in a folder called images and main.css is in a folder called css. Both folders are located in the root of our first_site folder.

Remember that document-relative links are relative to the file where the link is written. To link the background image from your main.css file you need to tell the browser to:

- go to the folder above the one I am in;
- find another folder called images;
- go in there and find a file called background.jpg

To go up a folder you use ../ to indicate the folder above.

Your CSS would now look like this:

```
body {
      background-image: url('../images/background.jpg');
}
```

../ means the folder above the one I am in now.

If you are two folders deep to the one you need you would repeat ../ You will use the ../ notation a lot in CSS.

When you learn about command line navigation you come across ./ as well. This means the same folder I am in now. You don't normally use this in CSS or HTML.

Document-relative links are the most flexible. They will work on your local file system and they will work on a live website. The only thing you need to be careful of is when you move a file to a different location. In that case don't forget to update all links to that file as well.

In the exercise we will do in this course you should be using document-relative links.

TASK :

Note to instructors:

It is very important everyone starts from the same point for this code along.

Demo opening the whole folder in atom, either by opening atom and selecting 'open folder' or by dragging and dropping the folder onto the icon on the taskbar (mac only) - this is particularly important as it is much easier to work on a project when you can see the whole file structure, rather than individual files.

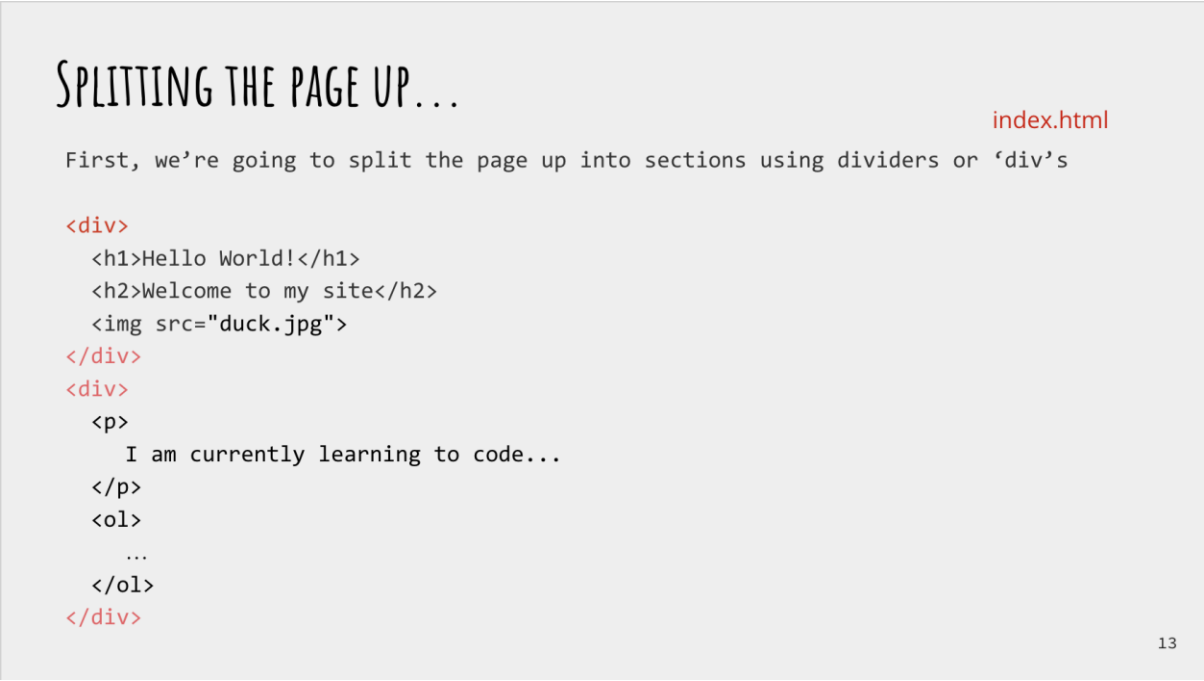
For anyone who has not finished their homework, get them to copy and paste their solution found here, https://github.com/CodeFirstGirls/beginners-week-one/blob/solution/index.html, into the index.html page.

Do not proceed with the session until everyone has the project open, and the completed index.html file

Once everyone has the project open in atom, with the completed version of index.html

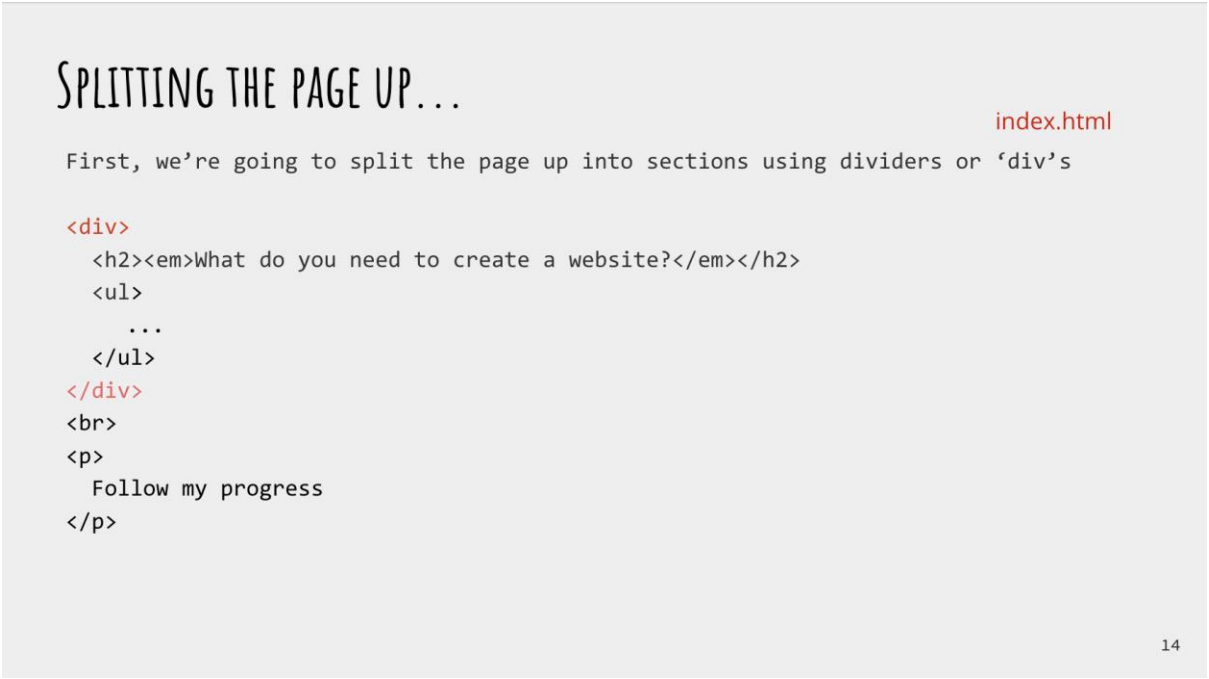
1. On the right hand side of atom, in the folder structure, left click and select ‘new folder’ and call it ‘css’
2. In this folder, create a new file called ‘styles.css’
3. It’s absolutely fine to copy and paste from the slides for this class (as it is with any), but please don’t skip ahead too far, as we will be covering important aspects of css in most slides which you may miss. Also, don’t hesitate to ask questions!

Code along:



Adding divs

'Div' stands for 'divider' and we use them to literally divide the page up into parts. This will make it easier to style. A divider is a 'block level' element. This means that it forces the next html element onto a new line



Now we have split the page up into three different parts (plus the last paragraph) which we can now easily style differently!

Linking the stylesheet and first css definition

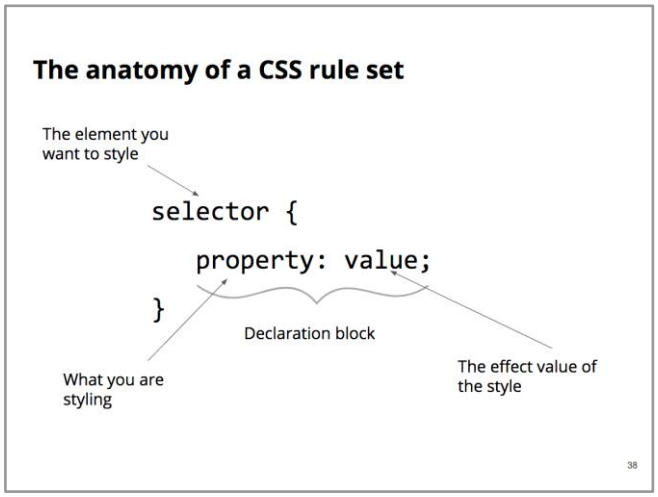
First things first before we can add all our styles, is we have to link the stylesheet to our index.html! As we have already discussed, we need to do this by adding a link in the head of the index.html.

Now let's add our first CSS rule!

Just to play around, (we will change this in a minute) let's change the colour of the whole page. We do this by changing the 'color' - spelling the american way unfortunately. (watch out for this as it's a common gotcha).

We are going to use RGB values which stands for Red Green Blue. All colours in computers are composed of different amounts of red, green and blue from 0 to 255. So If I wanted purely red I would put in rgb(255,0,0). It may be a bit confusing at first but luckily, In the developer tools, chrome has a really handy tool to select colour, and it will give you RGB values for the exact colour you want to select.

Instructor task: open developer tools and demo the colour selector



The anatomy of a CSS rule set

Ok, before we get carried away, let's have a look at the anatomy of a CSS rule. The CSS we just added is called a rule set:

```
selector {
      property: value;
}
```

Each rule set is made up like this:

- The **selector** is the thing you want to style.
- After the selector comes an opening curly bracket {
- Inside the curly brackets you add a declaration block each style change you want to add to the selector.
- A declaration block is made up of a property and value separated by a : and ended with a ;
- The **property** is what you want to change. The **value** is what you are changing the property into.
- At the end you close the rule set with a closing curly bracket }

Dont forget the semi colons!! It's such a 'gotcha'

Basic definitions

Note to instructor - the following script is just a guidance. Please go into the various different CSS rules as in depth as you want, according to how much you feel appropriate. Try not to spend too long on each slide, as it's important to get the exercise finished.

WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
body {
    color: rgb(65,75,86);
    font-family: arial;
    margin: 0px;
}
```


Replace the previous colour declaration with rgb(65,75,86)

We've already talked about colour so we can skip that for now. Font-family changes (you guess it) the font for the selected element. Handily, this can also hold several font names as a "fallback" system, if the browser does not support the first.

Now, let's talk about Margin. This is one of the primary positioning css rules which you can use to space elements out. Margin adds space around the outside of the element, so it pushes other elements away.

The browser adds a margin by default, and by setting it to 0px, we are removing it. 'Px' stands for pixel, which is a fixed length. (1 dot of pixel of the viewing device). This is the most commonly used css measurement value, but we will be covering others in the following slides.

Now we've added some styles to the whole body, lets select the divs.

Again here, we are selecting the three divs we added to the HTML earlier, so they will al be styles the same.

Min-height specifies the minimum height the element can be. This is important for responsible sites as they now should get larger and bigger according to the different size of screen. However, by specifying min height, it will never get below this height no matter how small.

Similarly, the width is defining the width of these divs.

So, what is this 'vh' and 'vw'? These however stand for 'viewport height' and 'viewport with'. 100 viewport width means that the element needs to take up 100% of the viewport width. This is a really useful little rule!

There are several different size systems that you can use in css. The one you are probably most familiar with is **pixel**, which is a standard length. Then **viewport width/height** (there is also **window height and width**) and then the **%**. That will be the **%** width of height of the parent element.

Now lets target some other things!

Here we are centering the text of the h1 and h2 elements to the center, and then making the font size of the h1.

Here we have introduced another unit, '**em**', just to jazz things up! Em means it is relative to the font size of the element, so **5em** is 5 times the relative font size of the element. Again, this is a useful property for responsive design.

WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
h1 {
     font-size: 5em
}
h2 {
     font-size: 2.5em;
}
img {
     display: flex;
     margin: 0 auto;
     padding: 20px;
     border-radius: 130px;
}
```

21

Now we are targeting the h2, to make it smaller, and the image.

'Flex' is a very useful display setting which allows for 'flexible placing' around the window. There will be a homework exercise to help you practise using flex that you should complete, as it is very useful (and you can always revisit it to see how you ordered things)

Padding here is similar to the margin property, but instead of adding space around the outside of the element, it adds space inside of the element. It sets the padding area on all four sides of an element. It is a shorthand for padding-top, padding-right, padding-bottom, and padding-left. **Demo how to change padding values for different sides**

Border radius adds corners to the element, so here we are making it a circle by cutting off the corners!

WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
img {
    ...
}
p {
    width: 40%;
    margin: 10vh 10vw;
    font-size: 2em;
}
a {
    color: rgb(0,0,0);
}
```


Now we are styling all the paragraphs. We only want them to be 40% of the width of their parent elements, and a responsive margin.

Let's also change the colour of the links (anchor tags) to make them look nicer than just blue!

WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
a {
     color: rgb(0,0,0);
}
ol {
     list-style: none;
     padding: 0;
}
```


Lists are very useful things, but almost never do we use their default styling, as it's a bit ugly. So let's just remove that like so!

WRITING CSS & SOME BASIC DEFINITIONS.

styles.css

```
ol {
    list-style: none;
    padding: 0;
    margin-bottom: 10%;
    font-size: 3em;
}
```

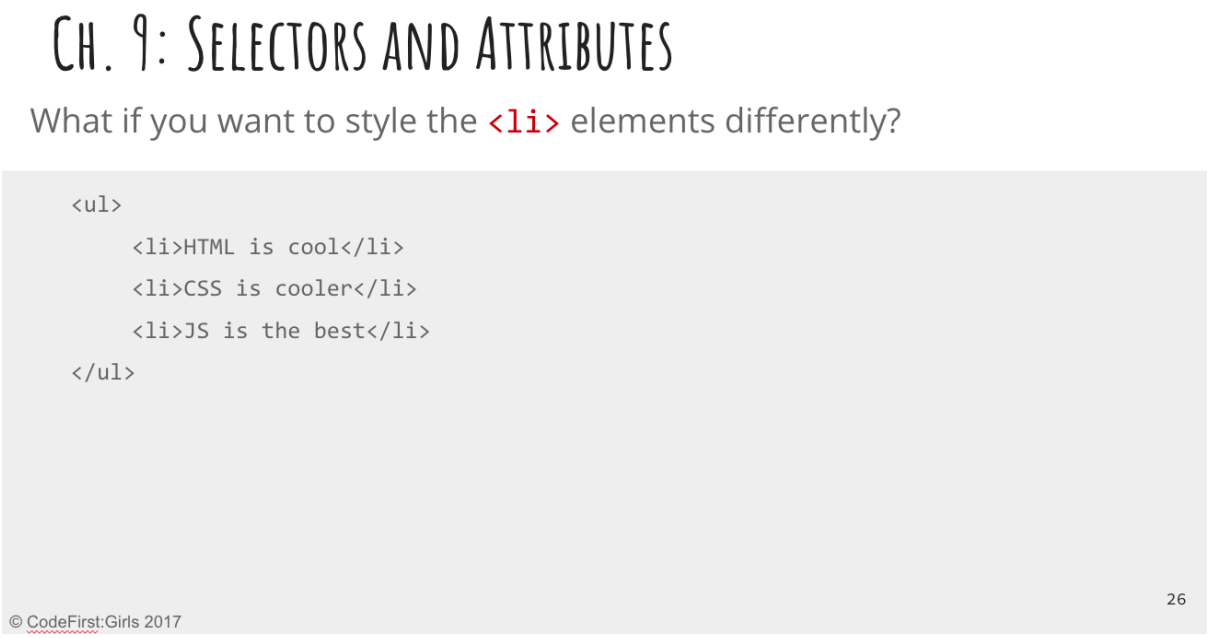

Let's also add a wee bit of margin on the bottom and pump the font size up, to make it slightly more in your face.

writing CSS & some basic Definitions.

styles.css

```
ol {
    list-style: none;
    padding: 0;
    margin-bottom: 10%;
    font-size: 3em;
    display: flex;
    justify-content: space-evenly;
}
```


And here we can use Mr Flexbox again. Again this is a really useful way to getting elements into the right places. Justify-content defines how the browser distributes space between and around content items along the main-axis of a flex container, and the inline axis of a grid container.

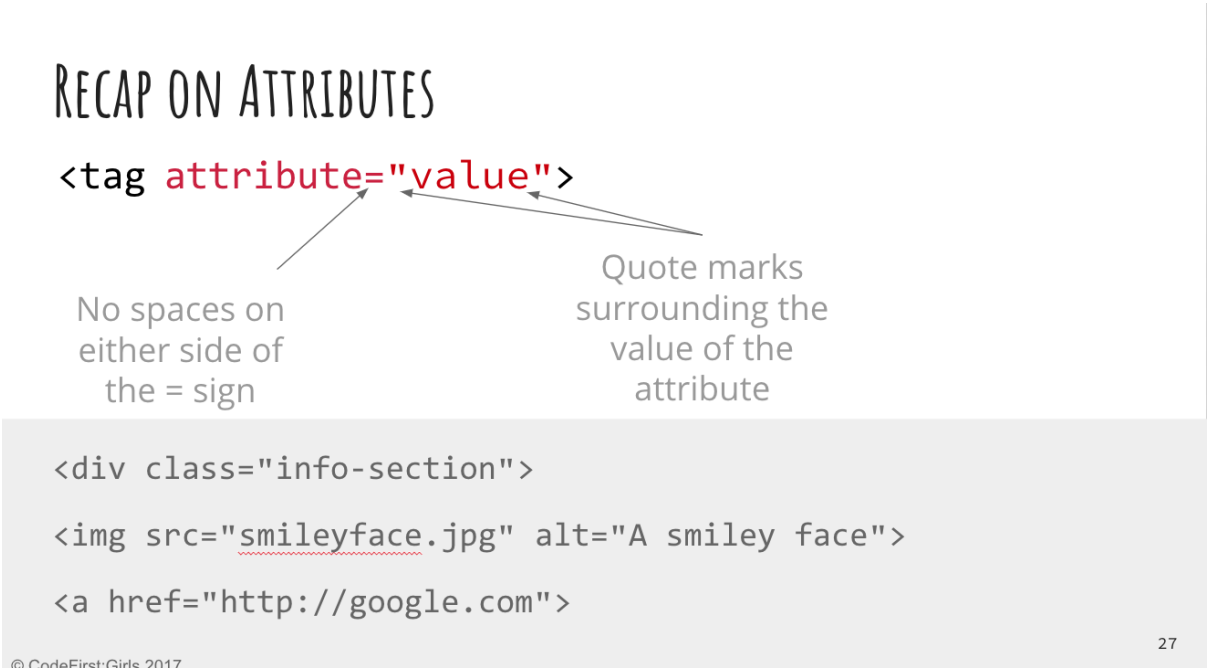


So that's the basic styles out of the way, now let's get a little more complex. What is we want to style the individual 'li' elements differently?

We now know all about what selectors are in CSS. So far we have used the names of HTML tag names as selectors. We call these element type selectors.

Even though you can string up several tag names divided by spaces to reach a deeply nested HTML element with CSS, sometimes you may need to be more specific.

For example imagine if you have a webpage with two **<h2>** elements and you want each one to look different. Simply using element type selectors will not be possible. To achieve this you can use specific HTML attributes.

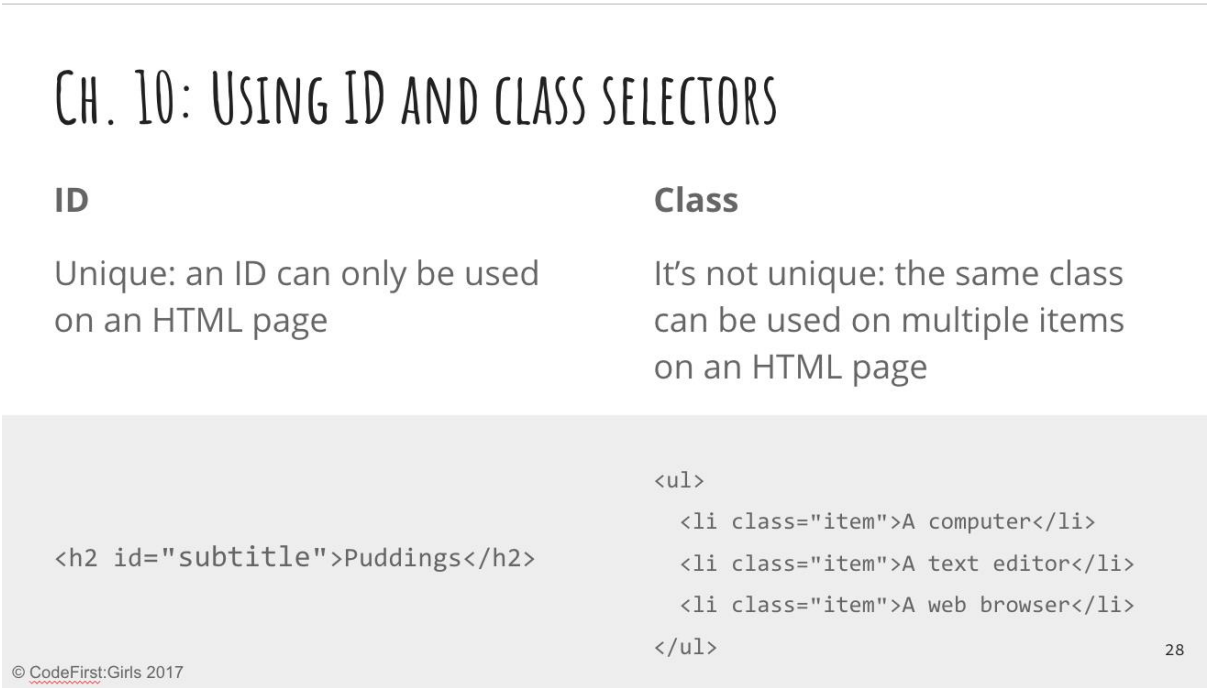


So what is an attribute?

We learned about attributes in HTML.

Attributes are always attached to the opening tag of an HTML element or inside an all-in-one element. They provide additional information about the HTML element.

An HTML attribute is made up of an attribute name and a value. Both are separated by an equal sign and the value is enclosed in double quotes.



Using the ID and class selectors

There are two attributes you can add to any HTML element. And they are the **id** and **class** attributes. Both are used to add information to the HTML element to which they are added. Both can be used by CSS and JavaScript to target that element for styling or scripting purposes.

The key difference between ID and class is that an ID is unique and can only be added to one HTML element on a page. The same class can be added to multiple elements on a page.

A little analogy: If the world was one bit web page then every person would be an ID, because there is only one of each of us and we are all unique. But our gender would be a class, because there are lots of women and men.

Using the ID attribute on a HTML element will allow you to write CSS that will only affect that particular element. But equally adding the same class name to multiple HTML elements allows you to write the same styles only once for all the elements with that same class name.

Using ID selectors in CSS

In your CSS you need to target your ID and class attributes in a special way.

To include an **ID** in your CSS you first write a hashtag # and then the name of the ID.

```
#lower { }
```

You can also include the HTML tag name before the # if you want. In most cases you don't need to do this, but sometimes it can help with being extremely specific.

```
li#lowest{}
```

If you add the HTML tag name to the ID selector make sure there is no space between the tag name and the hash tag. If you leave a space a browser will look for an element with the ID name nested inside an HTML with that tag name.

Using ID selectors in CSS

```
<li id="lowest">HTML IS COOL</li>
```

```
#lowest { .. }

li#lowest { .. }
```

Both of these are valid

30

Both of these are valid, but the browser will always give preference to the most specific one.

Using class selectors in CSS

To include a **class** name in your CSS you first write a full stop . and then the name of your class.

```
.product_item {}
```

Just like the ID you can include the HTML tag name in front of the class name. But again make sure you don't leave a space.

```
li.product_item {}
```

Adding the tag name creates a more specific selector. If you are using a class name on several HTML elements but not all the elements have the same tag name, then adding the tag name to the class will only target the HTML elements with that tag name.

It's up to you how specific you want to make your CSS selectors. Element type selectors using only single HTML tag names can be useful for styling things globally.

Specificity of selectors

This is a slide to demo css specificity, and is not part of the code along

Similar to the slide with the ID, these are all valid, however the bottom ones are more specific. Specificity is a weight that is applied to a given CSS declaration, determined by the number of each selector type in the matching selector. When multiple declarations have equal specificity, the last declaration found in the CSS is applied to the element. Specificity only applies when the same element is targeted by multiple declarations. As per CSS rules, directly targeted elements will always take precedence over rules which an element inherits from its ancestor.

Finishing up

styles.css

```
div.contrast {
     background-color: rgb(0,0,0);
     color: rgb(255,255,255);
     display: flex;
     justify-content: center;
     align-items: center;
     flex-direction: column;
}
```


Finishing up

To finish up, let's just add a few more things on to get to the finished design.

Let's make the contrast divs a little bit different, by getting all the child elements into columns .

Finishing up

styles.css

```
div.contrast {
     ...
}
ul {
     list-style: none;
     display: flex
}
li {
     margin: 0 20px
}
```


Now we can style the unordered list in a different way, and separate out each list element.

Getting snazzy...

styles.css

```
li {
     margin: 0 20px;
     transition: 0.6s ease;
}
li:hover {
     font-size: 1.2em;
     transition: 0.6s ease;
}
```


35

Now let's add some snazzy stuff. This is just and introduction, and please play around with different transitions at home.

The :hover selector specifies that it should apply the style only when the

mouse is hovering over it.

CSS transitions allows you to change property values smoothly (from one value to another), over a given duration. Thus, we can make the font larger when we hover over each list element, but include a transition so that it changes from the original to the new font size slowly.

Homework for week 2

Finishing off

Task:

1. Finish off the code-along from this week, you can check the solution here (image of how the page should look here)
2. Get to know your flexbox with flexbox froggy!
3. Read this guide and this guide on how to use GitHub and version control - **This is vital.**
4. Watch this more in-depth video about how the internet works, for Front-End Devs

Homework!

Quite simple for today, just finish off the code along. Work through the slides at your own pace or check out the solution on github.

Also on the github repo there are several other designs, completely different to the one we did today, that use all the same HTML, but different css. Have a look at them to see how powerful css can be.

Also, have a look at flexbox froggy, it's a very good tool to use as a reference for all things flexbox.

Use the following slides for reference in future

CSS properties

```
h1 { font-family: .. }
img { width: .. }
div { height: .. }
div h1 { color: .. }
ul { list-style-type: .. }
ul li { padding-left: .. }
li { margin-bottom: .. }
a { text-decoration: .. }
p a { border: .. }
```

40

CSS values

```
h1 { font-family: 'Helvetica', sans-serif; }
img { width: 300px; }
div { height: 595px; }
div h1 { color: blue; }
ul { list-style-type: none; }
ul li { padding-left: 30px; }
li { margin-bottom: 50px; }
a { text-decoration: underline; }
p a { border: 1px solid #000; }
```

41