Lab 2: CSS

Objectives

1. See how CSS rules are created.
2. Consider different CSS selectors – HTML, class, id and Pseudo-class selectors.
3. Consider different locations to store styles.
4. Add an external CSS files to multiple files to see styles ‘cascade’.

View the files

Your file structure should appear as follows:

```
f:\public_html\year1\ppd\lab2\
f:\public_html\year1\ppd\lab2\images
f:\public_html\year1\ppd\lab2\styles
```

There will be four HTML files created for you: *index.html*, *qualifications.html*, *skill-set.html* and *work-experience.html*.

Open the *index.html* file you will see it has a series of `<div>` tags used to divide the page content up.

Grouping Elements <div> and <span>

Two tags which you will find very useful when we look at CSS are `<div>` and `<span>`. Prior to working with CSS these tags don’t do anything particularly exciting. Their role is to group content such that styles can be applied to that grouped content via CSS. The `<div>` identifies blocks of content, the `<span>` tag inline content.

The <div> Tag

The `<div>` tag is short for (logical) division. It is used to place the content of HTML documents in blocks. This is useful when you identify a group of tags which together produce a distinct part of the document such as heading, footer or menu.

Like other HTML tags a `<div>` tag can have both `class` and `id` attributes such that classes and ID selectors can be applied to their content. We'll see what `class` and `id` do when we consider CSS.

As a 'block' element the `<div>` is like other block elements such as `<h1>`, `<h2>`, `<li>` in that it creates a line break. The following code places each section on a new line.

```
<div>Heading</div>
<div>Main Content</div>
<div>Footer</div>
```

However, unlike the `<p>` and heading elements, you **can** place other block elements inside the `<div>`, including other `<div>` tags.

This is a very simple example, but the idea could be extended such that groups of tags that make up the document heading, main content and footer are blocked together by `<div>` tags. An example would be as follows:

```
<div id="container">
<div id="header">
     <h1>Sheffield Winter Gardens</h1>
     <h2>Part of Heart of the City Project</h2>
</div>
<div id="content">
     <p> Sheffield's Winter Garden … </p>
     <p> The Winter Garden is part of the city's 'Heart of
the City' project …………..</p>
     <p>The garden contains more than 2,500 plants from all
around the world……….</p>
     <p>The Winter Garden will be open daily to … </p>
</div>
<div id="footer">
     <p>Page Maintained by Martin Cooper</p>
</div>
</div>
```

In *index.html* review the `<div>` elements used to divide the document.
How many `<div>` tags are there?
Why do you think the document has been divided in this fashion?

The <span> Tag

The `<span>` tag provides an inline container around document content. It is mainly used to allow styles to be applied to text that has no obvious HTML wrapper. Most commonly the `<span>` is used in association with a CSS selector known as a class (more later).

For example, consider the text below:

```
<p>Sheffield's Winter Garden is one of the largest temperate
glasshouses to be built in the UK.</p>
```

Assume we want to style the word largest. There is no natural container that would allow us to do so. Therefore we would need to add a `<span>` tag. We can place the `<span>` tag around the text. The `<span>` tag on its own will have no visual effect on the document. However, later on we can associate CSS with the `<span>` to style its content.

```
<p>Sheffield's Winter Garden is one of the
<span class="highlight">largest</span> temperate glasshouses
to be built in the UK.</p>
```

Whereas the `<div>` is a block level tag, `<span>` is inline. As such, `<span>` does not create a line break and works in the same fashion as familiar tags such as `<strong>` and `<em>` that we saw earlier.

Inline Styles

Preview the page *index.html* in your web browser.

You should notice the text ‘dynamic web pages’ appears red.

This is because it is enclosed in the following:

```
<span style="color:#ff0000;">dynamic web pages</span>
```

The use of the `style` attribute is known as an inline style. In this case the style is used to apply a red foreground colour to the text.

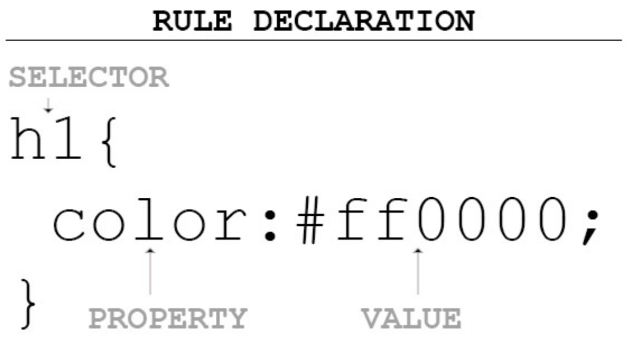
The `color` value is a CSS property used to set the hexadecimal colour code.

Although this technique works we'll see two other ways of adding styles, namely document level stylesheets and then external stylesheet.

In both techniques, instead of using the `style` attribute, we work with something called rules.

CSS Rules

CSS is based on rules. A rule declaration is broken down as follows:



The selector can be an HTML element such as `<h1>` above. This is known as an HTML selector. The curly brackets enclose a 'property/value' pair. More than one 'property/value' pair can be added. These are separated by semi-colons and usually, for ease of editing placed, on a new line. For example:

```
h1 {
    font-size: 16px;
    color: #000099;
    font-family: Arial, Helvetica, sans-serif;
}
```

There is a number of ways in which selectors can be declared. The above is an HTML selector. There are also a large number of properties. These 20 properties provide a good starting point:

http://www.mustbebuilt.co.uk/css-basics-20-properties-to-teach/

HTML Selectors

HTML selectors use HTML tags to target their content. Any HTML element can be targeted. Commonly used HTML selectors include `<body>`, `<h1>`, `<p>`, `<ul>` but any can be used.

In *index.html* add a `<style>` between the `<head>` tags. This is a document level stylesheet.

```
<head>
<meta charset="utf-8">
<title>Your Name :: About Me</title>
<style>
</style>
</head>
```

It is inside the `<style>` tag that we'll now add a number of CSS rules.

A stylesheet will often have a rule that uses the `<body>` tag. This will set defaults like font style, font color and background colour.

Add the following CSS rule inside the `<style>` tag.

```
body{
     font-family: helvetica, arial, sans-serif;
     color:#000099;
     background-color:#FFFFCC;
}
```

Create HTML selectors to target the document's headings. Have an experiment with some of the 20 CSS properties such as `text-transform`, `background-color` and `text-align`.

```
h1{
     font-size: 18px;
     text-transform: uppercase;
     font-family:"Times New Roman";
     background-color:#cccccc;
     text-align: center;
}
```

Class Selectors

With class selectors, it is up to you to give the selector a name. This should be unique and should begin with a dot (`.`). An example of a class selector based rule is as follows:

```
.maintext {
        font-size: 12px;
        color: #000;
}
```

In the above example, a class of name `maintext` is declared.

To apply the class selector use the class attribute that can be added to any HTML element that you want to assume the style. For example the following applies the rule `.maintext` to the first and third paragraphs but not the second paragraph.

```
<p class="maintext">Paragraph One</p>
<p>Paragraph Two </p>
<p class="maintext">Paragraph Three</p>
```

This gives you more flexibility in page design, as the `class` attribute can be added to any tag.

We can use a class selector to replace our inline selector from earlier.

Find the inline selector:

```
<span style="color:#ff0000;">dynamic web pages</span>
```

and replace it with:

```
<span class="highlight"> dynamic web pages</span>
```

Then inside `<style>` create a CSS rule for the class *highlight* ie:

```
.highlight{
     color:#FF0000;
}
```

This is now more flexible that the previous inline style. We can easily change the colour via the code in the `<style>` and because this is now a class selector we could easily apply the class to some other content in the page. For example, place a `<span>` around the text 'web developers' in *index.html* and assign the *highlight* class to the `<span>`.

Save and review your file. Both pieces of text in the `span.highlight` tags appear red.

ID Selectors

ID selectors are very similar to class selectors in that they can be applied to any element in an HTML file. ID selectors are flagged with a hash (`#`) and can be given any user-defined name as deemed appropriate.

Create an ID-based rule with the other rules in `index.html` and place it inside the `<style>` tag.

```
#footer {
    color: #0066CC;
    background-color: #CCCCCC;
    font-size: 10px;
    text-align: center;
}
```

This will look in the HTML for an element with an *id* value of *footer*. It will find:

```
<div id="footer">Page Maintained by mustbebuilt.co.uk</p>
```

ID vs Class Selectors

The main difference between ID and class selectors is that ID selectors are only supposed to be used once per document. They can be used on any HTML element in the document but should only appear once. In the example above a rule is declared for a section of the document that will be identified as the footer of the file and styled accordingly.

The document will have only one footer. When we consider CSS positioning we will see how this ability to single out an element for styling lends itself to page layout with CSS.

A Container with an ID Selector

Create an ID selector to target the `<div>` element that is acting as the document container. Assign the ID selector to the `<div>` as follows:

```
<body>
<div id="container">
```

Add the following ID selector to your style rules:

```
#container{
     width:900px;
     margin: 50px auto;
     background: #FFFFFF;
}
```

These properties will set the container's width to 900 pixels. By setting the margin to 'auto' this will tell the browser to allocate margin equally to both the left and right sides.

Multiple Selectors

Sometimes we want rules to apply in more than one circumstance. This is achieved by separating the selectors with a comma. The following can be used to apply the same font-family property to all `<h1>`, `<h2>` and `<h3>` elements.

Add the following to apply the same font styling to multiple elements.

```
h1,h2,h3 {
     font-family: Arial, Helvetica, sans-serif;
}
```

Contextual Selectors

Class selectors can be used with HTML selectors to define specific rules that will only be applied to a tag with the class applied. For example, the following rule will only apply to `<p>` elements that have the class attribute set to 'maintext'.

```
p.maintext{
     color: #FF0000;
}
```

Any other combination of HTML element with the 'maintext' class will not recognize the rule.
Thus the `<li>` tag below, even though it has the class attribute, will not pick up the `p.maintext` rule.

```
<li class="maintext">Paragraph Two</li>
```

HTML selectors can also be made contextual. The following rule is only applied when a `<strong>` tag is a child of a `<p>` tag.

```
p strong{
     color: #FF0000;
}
```

White spacing is used to separate the elements. In the above example `<p>` is the parent of `<strong>`.

Pseudo Classes

A pseudo-class is a way to targets an element that is in a particular state. It is most commonly associated with the `<a>` tag and the different states of a link.

The pseudo-classes associated with the `<a>` are:

:link	The link in its native state.
:visited	The link when it has been visited.
:hover	The link whilst the cursor hovers over it.
:active	The link whilst it is been pressed or selected.

Tip: It is popular to use the `text-decoration` property to remove the underline from links. However, make sure that users are still aware that the text is a link. For example, by grouping your links in navigation lists.

Thus to style up links you could use the following:

```
a:link{
     text-decoration:none;
     color:#000000;
}
a:visited{
     text-decoration:none;
     color:#CCCCCC;
}
a:hover{
     text-decoration:underline;
     color:#66CC00;
}
a:active{
     text-decoration:underline;
     color:#CC6600;
}
```

However, the above would restyle all `<a>` on the page and that may not be the desired effect.

Therefore, you are more likely to create specific rules that add more context. For example, we may only want to style the bulleted list inside the `<div id="nav">`.

```
<div id="nav">
<ul>
  <li><a href="index.html">Home</a></li>
  <li><a href="qualifications.html">Qualifications</a></li>
  <li><a href="skill-set.html">Skill Set</a></li>
  <li><a href="work-experience.html">Work
Experience</a></li>
</ul>
</div>
```

Add the following contextual rules (with their pseudo-classes) to your styles inside the `<style>` tag.

```
#nav li a:link{
     text-decoration:none;
     color:#000000;
}
#nav li a:visited{
     text-decoration:none;
     color:#CCCCCC;
}
#nav li a:hover{
     text-decoration:underline;
     color:#66CC00;
}
#nav li a:active{
     text-decoration:underline;
     color:#CC6600;
}
```

Tip: There are other pseudo-classes such as first-child and :focus that you could investigate.

Where to create Rules

Styles can be added to a document in a number of ways – inline styles, document level styles or as external stylesheets. We have now seen inline and document level styles.

Inline styles that use the `style` attribute are usually too specific and are difficult to maintain.

The document level styles that we have in `<style>` work well but they only apply to *index.html*. To use the same styling rules on all four of our HTML files we need to use an external stylesheet.

External Stylesheets

Currently, our rules are inside the `<style>` tag and only apply to *index.html*.

So that we can have a consistent style across our four files we'll move to an external stylesheet.

To do so, copy the rules you have created into a new stylesheet file, saved with a CSS extension. For this purpose, there is an empty file set up for you in the *styles* folder called *main.css*.

Remove the rules from the `<style>` tag in the `<head>` of *index.html*.

Move all the rules to the *main.css* stylesheet.

Then, in the `<head>`, add the `<link>` to attach this external stylesheet.

Your HTML should appear as follows:

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

Try attaching this stylesheet to the other files in the sequence with the `<link>` tag. Again place the `<link>` in the `<head>` of the documents.

An alternative way to attach a stylesheet with <style> and @import

An alternative technique used to attach a stylesheet is the `@import` construct.

```
<style>
@import url(styles/main.css);
</style>
```

As the `@import` is CSS syntax it can be placed inside a `<style>` or an external CSS file. This means that this technique can be used to import one stylesheet into another

Loading a Google Font

A web browser, by default, is limited to the fonts loaded on a user's machine. As such, if you would like all visitors to your site to use a specific font and if the font is not one of the 'web-safe' fonts such as Arial, Verdana, Times New Roman, then you'll need to provide visitors with the font. This can involve various font files and gets complicated. Fortunately, a number of services have arisen to make this a relatively painless process. Google Fonts is one popular service.

Visit https://fonts.google.com/ to see the range of fonts available. Once you find a font, look for the `@import` command to load it into your stylesheet. Here is an example:

```
@import
url(https://fonts.googleapis.com/css?family=Raleway);
```

If this is placed at the top of our stylesheet then we can use the font (in this case Raleway) in the font-family property ie:

```
h1, h2{
    font-family: 'Raleway', sans-serif;
}
```

Styling the Table

In the *qualifications.html* page there is an HTML table. Add the following rules to your stylesheet in *main.css* to style the table. You'll also need to add the `class` attribute of `quals` to the `<table>`.

```
.quals{
    border-collapse: collapse;
}
.quals th{
    border-bottom:3px solid #ccc;
    padding:5px;
}
.quals td{
    border-bottom:1px solid #ccc;
    padding:5px;
}
```

The `border-collapse` property when set to `collapse` will make the table cells share borders. This allows use to style the table using the `border` property.

The border property forms part of the box model, a key element of CSS which we'll investigate in the next lab.