Intro to CSS

Introduction

CSS3 is the currently used version of CSS, which stands for Cascading Style Sheets. CSS is the code which browsers use to alter the cosmetics and the layout of content rendered on a web page.

Like HTML, CSS is used in almost every web page you'll have ever visited. In this lesson you will learn the commonly used ways to select HTML elements to style in CSS, as well as some of the commonly used CSS properties used when changing the cosmetics of a web page.

Today we will learn how to add some cosmetic styling to a web page. This will enable us to make our web apps attractive to the user and more intuitive to use.

We will use the same index.html file we were working on during the course!

Styling with CSS

In this lesson we are going to use CSS to style the elements within the index.html contained in the start code. To do this, the first thing that we have to do is create a CSS file and reference it from within our HTML file.

Referencing the CSS file is done via a <link> tag added to the <head> tag - the important bit is that we need to make sure the path is matching with where the CSS file is located. If you leave the default autocomplete for the link:css emmet abbreviation, then you get the following:

```
<link rel="stylesheet" href="style.css">
```

Make sure this is in the head tag as seen in the video!

You could hide this file in a folder like public that is created next to the index.html, in which case you'd need to change the path to:

```
<link rel="stylesheet" href="public/style.css">
```

but for our purposes we can leave it the default option.

Remember, use ChatGPT when in doubt with a well formed question!

Traditionally, we call our first/main CSS file either main.css or style.css, however very often multiple CSS files will be created for styling separate parts of our apps - like forms.css, nav.css, etc.

Now, after the HTML file is loaded in our browser it will go and get the CSS file that we have referenced and apply the styling rules contained within it. Currently, the file is empty so the page won't look any different.

Let's add some styling rules.

In order to style an HTML element, we first have to target it using a CSS selector, then we can define a set of rules that will be applied to any elements that will be targeted by that selector.

First, let's change the font. To do this we can target the <body> and set a rule for its font-family property. This rule will cascade downwards and affect all children of <body> too. When setting a font, it's good practice to specify fallback fonts, in case the font that we want to use isn't available for some reason.

```
/* style.css */

body{                          /* NEW */
 font-family: Helvetica, Arial, sans-serif; /* NEW */
}                              /* NEW */
```

This will tell the browser to attempt to use Helvetica, if Helvetica isn't available then the browser will try to use Arial, finally if neither Helvetica or Arial are available, the browser will use the operating system's default sans-serif font.

Note: If the name of the font includes a space then the name must be wrapped in quotes, e.g. "Lucida Grande"

Next, we will style the footer and the header of the web page. Let's add a background color so that it stands out from the rest of the page.

Our first instict might be to select <footer> elements, but we'll run into a problem if we do.

```
header {
   background-color: orange;
}

footer {                  /* NEW */
 background-color: orange; /* NEW */
```

```
}                    /* NEW */
```

This applies the background colour too ALL <footer> and <header> elements on the page, which isn't exactly what we want. We only want to style the main footer at the bottom of the page. We have a few options here. We could add an id, allowing us to target the element directly, but it's best to avoid littering our HTML with unnecessary classes and ids where possible. Instead, we will apply this rule to all <footer>s that are direct children of the <body>. We can do this using the > selector.

```
body > header { /* UPDATED */
  background-color: orange;
}

body > footer { /* UPDATED */
 background-color: orange;
}
```

Since we are working with them, let's talk about selectors!

CSS selectors

There are many ways to select items in CSS.

Universal selector

Selects all elements in a given document. Generally better to avoid it - it will apply to everything. Example: *

Type selector

Selects all element of a given name. Example: div, p

Class selector

Selects all elements with a given class attribute. Example: .input-field

Id selector

Selects all elements with a given id attribute. Example: #login-form

On top of different selectors, you can use combinators to combine multiple selectors.

Descendant combinator

Selects all instances of the second element that are descendants of the first element. Example: div p -> Selects all p tags that are nested inside div tags - at any level.

Child combinator

Selects all instances of the second element that are direct children of the first element. Example: body > footer -> Selects all footer elements that are direct descendants of the body tag.

Finally, you can use pseudo classes and pseudo elements.

Pseudo class

Allows the selection of elements based on state information that is not contained in the document tree. Example: a:visited, this targets anchor tags already visited by the user.

Pseudo elements

Represents entities that are not included in HTML. Example: p::first-line, this targets the first line of every p tag.

Positioning

You'll notice that our footer doesn't extend all the way to the edges of the page. This is because there is some margin on the body that pushes everything away from the edges of the window. We can remove this margin, which will cause elements such as the page's header and footer to extend all the way to the edge of the window.

```
body{
 margin: 0; /* NEW */
 font-family: Helvetica, Arial, sans-serif;
}
```

The orange background on the <footer> is also quite tight around the text. We can use padding to create more space within the element. While we're there, we can also change the colour of the text to white and align it in the centre of the element.

```
/* ... */

body > footer > p {  /* NEW */
  padding: 16px;     /* NEW */
  color: white;      /* NEW */
  text-align: center; /* NEW */
}
```

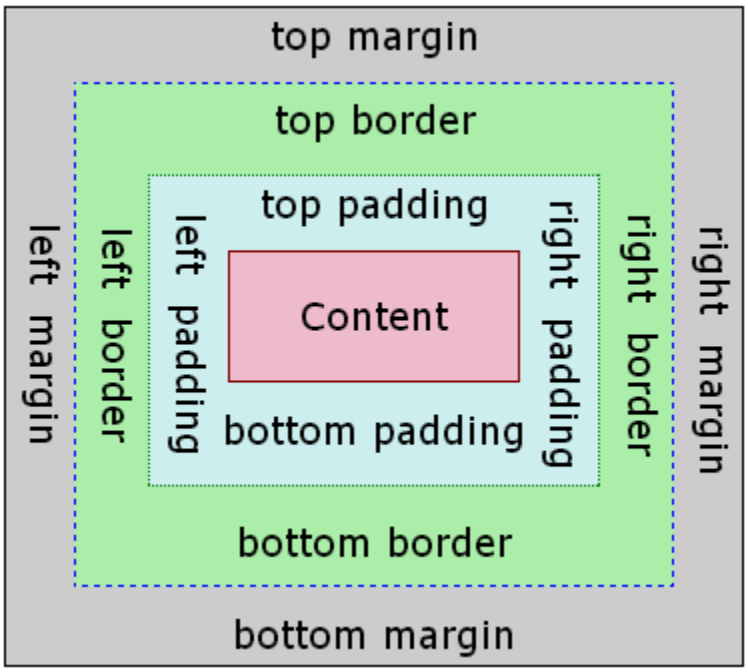
Note that margin creates space around the element, whle padding creates space within the element.

Since we are talking about margins and padding, it's time to discuss the box model.

The box model

Each single HTML element is enclosed in the so-called 'box model': https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_box_model/Introduction_to_the_CSS_box_model

This illustration from the CodeProject explains it all:



The properties used here are:

```
div{
  width: value;
  height: value;
  margin-top: value;
  margin-right: value;
  margin-bottom: value;
  margin-left: value;
  padding-top: value;
  padding-right: value;
  padding-bottom: value;
  padding-left: value;
}
```

This can be shortened to:

```
div{
  width: value;
  height: value;
  margin: value-top value-right value-bottom value-left;
  padding: value-top value-right value-bottom value-left;
}
```

Display Types

The three most common display types are block, inline, and inline-block.

All elements have a default type in HTML of either block or inline, and we can change the types using CSS if we want to. A block element takes up the whole width of its container, regardless of how wide you set the element to be.

No other elements will wrap around it. Most of the 'container' type elements have a default display of block, like <p>, , <section>, <ul> as well as headings like <h1>, <h2> etc.

An inline element only takes up the space that its content needs and other elements wrap around it. Most inline elements are text-based elements like links (<a>), , <cite> and outdated styling tags like <b> or <u>.

One of the most common ways that a display style is altered is to change the type to inline-block - this combines parts of both of those types.

Inline-block elements can have a height and width set on them and push other elements away when they have padding and margin set, but allow other elements to wrap around them on the same line if there is space.

Another thing to note is that block elements are designed to be 'bigger' than inline elements - because of this you can nest inline elements inside a block, but you may have issues when trying to nest block elements inside an inline element.

You can also set display to be none, or visibility: hidden. The former makes it look like that the element was never there in the first place, the latter just makes it not display content - but it still takes up the same space as it otherwise would.

Styling the header

Next we will style the header similarly to the footer. We can use a similar approach that we used to style the footer. The element that we want to style is a <header>, but it isn't the only <header> element on the page. We specifically want to style the <header> element that is a direct child of the <body>.

```
body > header {            /* NEW */
  background-color: orange; /* NEW */
  color: white;            /* NEW */
}                          /* NEW */
```

Our web page is starting to take shape, but the header doesn't look quite right. There's some margin pushing it down from the top of the page and the links haven't taken the white font colour that we wanted.

We can use the "select element" feature of Chrome's devtools to figure out where the margin is coming from.

Open Chrome's devtools using Option + ⌘ + J and select the "select element" tool by clicking on the mouse cursor icon in the top left. You can use this tool to hover over elements, highlighting them and displaying their margin and padding.

Using Chrome's devtools we should be able to see that the margin is coming from the <h1> element within the header. We could select this <h1> element by selecting h1 elements which are children of a header which is a child of the body. We may also wish to apply a specific font to the logo of our site, whether that's in the header or elsewhere though, so we'll use an id.

First, let's add an id to the HTML.

```
<!-- index.html -->

<header>
  <h1 id="logo">CSS</h1> <!-- UPDATED -->
  <!-- ... -->
</header>
```

Now we can use this id to select this h1 from within our CSS and remove the margin from this element. We can use the # selector to select an element by id.

```
/* style.css */

#logo {     /* NEW */
  margin: 0; /* NEW */
}         /* NEW */
```

Next we will change the font-family of our logo, so that it uses a custom font from Google fonts: https://fonts.google.com/. Navigate to Google fonts and choose a font that you would like to use. We will then need to reference it from our HTML to load it, because the user likely won't have the font installed.

```
<!-- index.html -->

<head>
  <!-- ... -->
  <link href="https://fonts.googleapis.com/css2?family=Fredoka+One&display=swap"
rel="stylesheet">
</head>
```

Just like our stylesheet, the browser will now load our custom font when it loads our HTML file. Now that we have loaded the font, we can use it to set the font-family of our logo. Remember to wrap the name of your font in quotes if it contains a space and supply a fallback font in case the font can't be loaded for some reason.

```
#logo {                       /* NEW */
  margin: 0;                    /* NEW */
  font-family: "Fredoka One", sans-serif; /* NEW */
```

```
}                          /* NEW */
```

We won't worry about positioning the elements inside of our header today. That will become much easier using some tools that we will learn later. But we will do a little bit more styling.

Let's remove the bullet points from the list of links and also make them white, as blue on orange isn't particularly easy to read.

```
nav > ul {       /* NEW */
  list-style: none; /* NEW */
}              /* NEW */

nav > ul > a {     /* NEW */
  color: white;    /* NEW */
}              /* NEW */
```

Styling the <main>

Now things are really starting to take shape. The main content looks a bit strange as it spans the whole page. It might be easier to read if it were more restricted to the centre of the page. We can do this by setting the width of the <article> that contains the content.

```
body > main > article { /* NEW */
  width: 70%;       /* NEW */
}                /* NEW */
```

Note that it's always better to use a relative value when possible. Absolute values (px, for example) should only be used for things like margin, padding or, in some cases, images.

This looks a little better, but it would be nice if our content sat in the middle of the page, rather than the left-hand side. We can achieve this by setting the margin property of this element to auto. This will apply a horizontal margin that fills the rest of the vertical space that is not occupied by the article.

```
body > main > article { /* NEW */
  margin: auto;      /* NEW */
  width: 70%;       /* NEW */
```

```
}                /* NEW */
```

Great, this almost looks like an actual website, save for some dodgy positioning that we'll learn how to deal with later.

Font sizing

It is important to understand that there are many ways to set the size of our fonts, images and different elements.

Pixels

One of the most common ways to set the size of a font is to use pixels.

Pixels can be used when precision is important. A pixel on one screen is a pixel on another. The pixel value you specify will roughly appear the same way across different browsers. However, it will be much harder to create a responsive website that looks good on any display.

```
h1 {
  font-size: 28px;
}
p {
  font-size: 12px;
}
```

EM/REM sizing

Another common way of setting the size of a font in CSS is to use em sizes. The em unit of measurement refers to the font size of a parent element. If your element's base font size is set to be 24 pixels, then every child element of it will use that as a reference - 2em will be 48 pixels, 0.5em will be 12 pixels, and so on.

The default is usually 16 pixels, but it can depend on the browser.

```
p {
  font-size: 1em;
}
h1 {
  font-size: 1.5em;
```

```
}
```

REM is the same as EM, the only difference is that REM will rely on the root element's size rather than the parent element - this can help us being consistent.

Viewport units - vh/vw

This is best used when creating a responsive webpage is important. These are always calculated based on the browser's viewport size. 1vh is 1% of the viewport's height, and 1vw is 1% of the viewport's width. Regardless of display type, this will make sure that sizes stay the same in relation of the display.

```
p {
  font-size: 1vw;
}
h1 {
  font-size: 4vw;
}
```

Finally - a word on specificity

Specificity states which CSS rules apply and/or overwrite other rules stated by us. As a general rule of thumb, the more specific you are, the stronger the rule is.

Conclusion

CSS at its core is very simple. Select an element that you would like to style, and apply some rules to it. Like most other technologies, the hard part is learning to use it well. In this lesson we covered some of the best practices when using CSS to change the appearance of a web page. Once we're comfortable with this we will be in an excellent position to learn some of the more advanced CSS features that can be used to position elements on a page.

Further Resources

- MDN docs on CSS: https://developer.mozilla.org/en-US/docs/Web/CSS