Lesson 9

In addition to developing your portfolio, today's lesson will cover the use of CSS pre-processors. Key points of today's lesson include:

- Purpose of pre-processor variables.
- Pros and cons of mixins.
- Using nesting.
- Functions for data, maths and logic.

Lesson Recordings:

1. https://vimeo.com/715928558/053aaca746
2. https://vimeo.com/715965920/e8ca451af4
3. https://vimeo.com/716006445/f3171d04e4

Tasks:

- Create a new LESS variable and CSS element to create an element that has a content colour of highlight and a darker highlight colour for the background.
- Define a set of elements that are nested inside a parent element.
- Create a list of data items in LESS.
- Use the each() function of LESS to create a CSS style for each of the items in the previously created data list.

Questions and Answers:

What's the difference between LESS variables and native CSS variables?

The LESS variables can be used in LESS logic that controls how the CSS file is built. In comparison, native CSS variables allow can be used in realtime execution of the of the website. There are some/many situations where you can use either LESS or native CSS variables.

How can I use CSS to reference items on the web page?

The nth-child selector allows you to specify the position of an element within its container. For example, the following rule will select the third list item of a list and set its background as red:

```
li:nth-child(3){
  background: red;
}
```

What's the purpose of LESS mixins?

They allow you to reuse the styling of a class or ID element as part of another element. This is an advantage for maintainability because any changes to the original definition will automatically update any using it as a mixin. Think of mixins as being the same as Javascript functions in the sense that they are containers for instructions that can be called into throughout the code.

Are there any disadvantages to using LESS mixins?

Yes. They increase the size of the CSS file because every mixin placement creates a repetition of the element class or ID they are created from. This isn't much of a problem where mixins are are

limited in their use, but it can lead to hundreds or thousands of extra lines of code being generated when mixins are used too much.

Is there a native CSS alternative to LESS mixins?

Yes, but it's not as convenient. CSS allows you to write styles for multiple elements by writing their selection rules separated with commas. Here's an example of the same styling can be applied to h1 and h2 elements at the same time:

```
h1,
h2{
  text-decoration: underline;
  border: 12px solid red;
}
```

Example: LESS Variables

This example shows how to use variables in LESS to apply a consistent background colour and font size to h1 and h2 elements:

```
@colour_highlight: yellow;
@font_big: 40px;

h1{
  color: red;
  background: @colour_highlight;
  font-size: @font_big;
  .title();
  .extra();
}


h2{
  color: blue;
  background: @colour_highlight;
  font-size: @font_big - 10px;
  .title();
}
```

Example: LESS Mixins

This example shows how mixins can be used to repeat a set of styles across multiple elements - i.e the title and extra classes being called into use within the h1 and h2 definitions:

```
.title{
  text-decoration: underline;
  border: 12px solid red;
}


.extra{
  font-weight: bold;
}


h1{
  color: red;
  .title();
  .extra();
}
```

```
h2{
  color: blue;
  .title();
}
```

Example: LESS Nesting

The following example shows how nesting can be used to define rules that only affect elements that are within a specified parent container. This example defines the border and background for for the footer element and uses nesting to specify the font-size for h1 and h3 elements inside the footer:

```
footer{
  border: 2px solid blue;
  background: transparent;
  .extra();


  h1{
    font-size: 2em;
  }


  h3{
    font-size: 1.3em;
  }
}
```

Example: LESS List and Each

This example shows a list called @chocolates being created to store multiple items and how the each() function can be used to create a CSS definition for each item in the list:

```
@chocolates: Mars, Twix, Galaxy;

each(@chocolates, {
  [data-chocolate="@{value}"]{
    colour: blue;
    background: silver;
    border: 2px solid;
    font-weight: bold;
    z-index: @index;
    content: "@{value}";
    .title();
  }
});
```

Example: CSS nth-child()

The following example shows how the CSS nth-child selector can be used to set the background colour of the third child of any element using the slideshow class:

```
<style>
.slideshow > *:nth-child(3){
  background: red;
}
</style>

<ul class="slideshow">
```

```
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ul>

<section class="slideshow">
  <div>
    <h3>Content 1</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 2</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 3</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 4</h3>
    <p>Content...</p>
  </div>
</section>
```

The following example shows how the nth-child select can be used to specifically reference li items inside the slideshow container. This results in any third child that is **not** an li element being ignored.

```
<style>
.slideshow > li:nth-child(3){
  background: red;
}
</style>

<ul class="slideshow">
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ul>

<section class="slideshow">
  <div>
    <h3>Content 1</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 2</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 3</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 4</h3>
    <p>Content...</p>
  </div>
</section>
```

Example: Javascript querySelectorAll()

The following example shows how Javascript's querySelectorAll can create a list that references all elements that have the slideshow class:

```
<ul class="slideshow">
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ul>

<section class="slideshow">
  <div>
    <h3>Content 1</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 2</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 3</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 4</h3>
    <p>Content...</p>
  </div>
</section>

<script>
document.querySelectorAll(".slideshow").forEach( function(item){
  console.log(item);
} );
</script>
```

You can extend the above example with the following code to use **forEach** to reference every item in the list produced by **querySelectorAll**. The **i** variable define the number to place into the **nth-child** query selector used with **item.querySelector** - resulting in an **active** class being added to the third child of every slideshow element.

```
<script>
let i = 3;
document.querySelectorAll(".slideshow").forEach( function(item){
 item.style.border = "2px solid";
 item.querySelector(`:nth-child(${i})`).classList.add("active");
} );
</script>
```

The following CSS will need to be applied to make the **active** elements of each slideshow to appear with the background highlight colour:

```
<style>
.slideshow > .active{
  background: red;
}
</style>
```

An alternative version of this CSS definition can make use of the **:not** selector to define any child element of the slideshow that doesn't have the **active** class as invisible:

```
<style>
.slideshow > *:not(.active){
  display: none;
}
</style>
```

Example: Simple Slideshow

The following example is a simple interactive slideshow that can be controlled by buttons placed on the web page. The main functionality of the slideshow is defined by functions stored within the **slideshow** object. Slideshows items can be activated become visible by providing their position number to the **slideshow.activate**() function - as demonstrated by the HTML buttons.

```
<ol class="slideshow">
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ol>

<section class="slideshow">
  <div>
    <h3>Content 1</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 2</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 3</h3>
    <p>Content...</p>
  </div>
  <div>
    <h3>Content 4</h3>
    <p>Content...</p>
  </div>
</section>

<button onClick="slideshow.open(1)">1</button>
<button onClick="slideshow.open(2)">2</button>
<button onClick="slideshow.open(3)">3</button>
<button onClick="slideshow.open(4)">4</button>

<style>
.slideshow > *:not(.active){
  display: none;
}
</style>

<script>

let slideshow = {};
slideshow.activate = function(i){
  document.querySelectorAll(".slideshow").forEach( function(item){
    item.style.border = "2px solid";
    item.querySelector(`:nth-child(${i})`).classList.add("active");
  } );
}

slideshow.open = function(i){
  this.reset();
  this.activate(i);
  // above is the same as the following
  //slideshow.reset();
  //slideshow.activate(i);
}

slideshow.reset = function(){
  document.querySelectorAll(".slideshow > .active").forEach( function(item){
    item.classList.remove("active");
  } );
}
```

```
slideshow.activate(1);

</script>
```