BEM is a system of naming classes that encourages the good practice of writing modular code. It can really help to bring structure and meaning to the way that we name things, which is often quite a difficult practice. And BEM is something that I use consistently in my day to day client work.

It can be used with or without Sass, but BEM and Sass are a great combination. In this episode, you'll learn what BEM is and what it stands for, how to leverage the Sass ampersand with BEM, and some BEM best practices.

BEM is a naming convention to help add structure and meaning to class names. It stands for **Block Element Modifier**, and it's a great system for building flexible and modular code by creating a series of components. It's a smart way of naming your CSS classes to give them more transparency and meaning to other developers. They are far more strict and informative, which makes the BEM naming convention ideal for teams of developers on larger projects that might last a while.

Thinking about your code as a series of components rather than on a page by page basis can help you work faster by building a system of reusable pieces. This is easier to maintain and helps the design of a project look and feel consistent.

The BEM syntax looks a little strange when you see it for the first time because it uses a notation of double underscores and double dashes to demonstrate whether a particular class is a Block, Element or Modifier. The syntax looks as follows:

```
{% highlight css %}
.block { }
.block__element { }
.block–modifier { }
{% endhighlight %}
```

It might look a bit ugly but it can really help to make your code more readable so bear with me.

Let's take the example of a common design pattern of an image floated beside some text. This is often referred to as a "media object" which was popularised by Nicole Sullivan of OOCSS fame.

We can refer to this whole thing as a BEM "block" and we might give it the class name of `.media`.

```
{% highlight css %}
.media { }
{% endhighlight %}
```

This media block has two parts to it: the media part (which is often an image) and the text content.

We could refer to the image and text as elements within the media block and with BEM syntax, this would be written as follows:

```
{% highlight css %}
.media { }
.media__image { }
.media__content { }
{% endhighlight %}
```

Perhaps we'd also like the flexibility of a media block with the image on the right and the content on the left. For this we could use a modifier class which is added to the parent block. This class can be used to modify the styles of the elements within and could be written as follows:

```
{% highlight css %}
.media–flipped {
.media__image {
float:right;
margin-left:1em;
}
}
{% endhighlight %}
```

This method of naming classes works well and I use it a lot in my day to day work. But one complaint of this system is that the class names are long which can get tedious to type. Well, Sass may just be able to help us out...

The Sass Ampersand and BEM

In the previous episode we looked at the Sass ampersand character and how it can be used to represent the parent selector when when nesting.

The ampersand has another use and can be particularly handy when working with BEM.

Going back to the previous example, here's the full code for our media block in "normal" CSS.

```
{% highlight css %}
.media {
overflow:hidden; // or alternative clearfix technique
}
.media__image {
float:left;
```

```
margin-right:1em;
}
.media__image img {
display:block;
}
.media–flipped .media__image {
float:right;
margin-left:1em;
}
.media__content {
display:table-cell;
width:10000px;
}
```
{% endhighlight %}

`media` is repeated in each selector and repetition is something we often want to reduce in our code. Instead, this syntax can be re-written with nesting and the Sass ampersand:

{% highlight css %}
```
.media {
overflow:hidden;

&__image {
    float:left;
    margin-right:1em;

    img {
        display:block;
    }
    .media--flipped & {
        float:right;
        margin-left:1em;
    }
}
&__content {
    display:table-cell;
    width:10000px;
}

}
```
{% endhighlight %}

Normally, nesting selectors like this produces descendent selectors in the compiled CSS but using the ampersand as the first character in the selector does not. And this Sass snippet will output exactly the same CSS as we looked at before.

Nesting in this way provides all the benefits of nesting (grouping code together and demonstrating hierarchy) without the downside of generating

overly specific selectors.

So, this may sound like the perfect solution? But, unfortunately there are a couple of downsides.

BEM best practices

One of the biggest issues with using BEM with the Sass ampersand is that your codebase becomes less searchable. Instead of being able to find and replace (or just find) a selector by name, you can only search for the block or the element or the modifier part of the class name.

Another downside of this syntax is that if you have a lot of CSS that spans across many lines, you may find the parent selector isn't visible when working on part of the code further down the file. This could lead to confusion and make the code less easy to read.

If you like the nested BEM syntax but still want a searchable codebase you could always add a comment before each selector which could be searched for.

```
{% highlight css %}
.media {
// .media__image
&__image { }

// .media__content
&__content { }

}
{% endhighlight %}
```

Another tip for working with BEM is to really think about how you create your modules so they can be kept as compact as possible to improve readability. Each block should be a distinct component rather than using the block as a naming prefix for all elements on a given page.

Components are things like buttons, registration forms, items in a grid of products or portfolio pieces, pagination or social icons; anything distinct that's made up of a number of elements that could be reused throughout a site.

In the past I've made the mistake of taking a parent class like `.home` or `.products` and using that as the base "block" for all my BEM classes. This leads to a lot of really long and complex class names which is less than ideal and very tedious to type.

```
{% highlight css %}
.home { }
.home__title { }
.home__features { }
```

.home__features-item { }
.home__features-item-title { }
.home__features-item-title–center { }
{% endhighlight %}

Instead, the `.home__features-item` should be its own component

- perhaps just called `.feature` - with its own elements and modifiers.

{% highlight css %}
.home { }
.home__title { }

.features { }
.feature { }
.feature__title { }
.feature__title–center { }
{% endhighlight %}

Naming things is hard but using a system like BEM can help focus your thinking and add meaning and structure to your code which is important for maintainability and readability and will be a great help to you and your fellow team mates in the future.