Sass is an extension language for CSS but it's popularity has seen the birth of a number of extensions to Sass itself in the form of mixin libraries and plugins.

Neat and Bourbon are both popular Sass extensions and we'll look at each of them in this video.

Both libraries were created by the team at Thoughtbot, a global design and development consultancy. They can both be installed as Ruby gems and are fairly simple to set up as long as you have Ruby and Rubygems installed. Let's set things up.

On your terminal run the following command

```
gem install bourbon neat
```

Once installed you can start a bourbon project by running

```
bourbon install
```

Or a neat project by running

```
neat install
```

For more details about installation and details of what these libraries can do, head to bourbon.io or neat.bourbon.io.

Bourbon

Bourbon is predominantly a mixin library that features a number of useful helper functions and mixins for performing common tasks like converting pixels to ems, pixels to rems, clearfix, using popular easing functions and much much more.

Let's take a look at a couple of these in action.

I've got the beginnings of a contact form component here which we'll turn into a two column floated layout with a little banner positioned in the top right corner.

The HTML is fairly typical and I've used BEM for the class naming

- for more info on BEM and using it with Sass check out Episode 2.

I've set up some default styling and we're going to use make a few tweaks and changes to illustrate some of Bourbon's features.

Firstly, to start using Bourbon we have to `@import` it into our project (having gone through the installation steps).

{% highlight scss %}
@import 'boubon/bourbon';
{% endhighlight %}

We now have access to all of Bourbon's features which mostly come in the shape of mixins. For a full list of what Bourbon can do for you, head to bourbon.io/docs.

Let's start by creating a two-column layout. We'll look at Bourbon's grid framework, Neat, a bit later on but for now I'm just going to create two equal sized columns and float them side by side.

When we float things, the parent container collapses so we need to use clearfix. Instead of bouncing out to the web and copying and pasting a clearfix snippet, we can use Bourbon's clearfix mixin.

```
.contact__container {
    @include clearfix;
}
```

Pretty handy. Another useful mixin is the positioning shorthand mixin. This position mixin takes two arguments: one for the type of position and one as a shorthand declaration for the `top`, `right`, `bottom` and `left` offset properties.

This allows our absolutely positioned banner to be placed in the corner of its relatively positioned parent container in a single line.

One final area we can look at is when styling the form inputs. This contact form only has 4 types of inputs: `text`, `email`, `tel` for the phone number and `textarea`. Instead of having to type these out (and all the others if you were using them), Bourbon has a handy variable for `$all-text-inputs` which allows them to be styled all at once.

There's also variables for `$all-text-inputs-hover` and `$all-text-inputs-focus` but I'd just use a bit of nesting and the Sass ampersand to reference the parent here as it's quicker to type.

{% highlight scss %}

{$all-text-inputs} {

```
&:hover,
&:focus {
    background:#fff;
}
```

}
{% endhighlight %}

Bourbon originally contained a lot of mixins that output vendor prefixes for fancy new CSS3 properties like animations, transitions, border-radius and transforms. These properties are now mostly available unprefixed and any for properties that do need prefixing, using a tool like Autoprefixer is more robust approach.

Many of the old CSS3 mixins have been removed from Bourbon in a recent update. Now it provides a great base for anyone that wants to use a consistent set of functions and add-on features without having to rely on using a whole front-end framework like Bootstrap or Foundation.

Neat

Neat is another part of the Bourbon family offering a lightweight, semantic grid system.

The library bundles another handful of mixins for building flexible and responsive grid systems. It used to depend on the Bourbon library but has been updated to remove this as a dependency which is ideal if you only want a grid system and don't need anything that Bourbon provides.

Let's take a look at an example of building a simple page layout with a Neat grid.

Once Neat is installed, the first step is to `@import` the neat library into your Sass files.

{% highlight scss %}
@import 'neat/neat';
{% endhighlight %}

With the libraries imported, we're able to start building a grid.

The first thing to configure is how many columns our grid should have and how much spacing there should be between the columns. We create a map variable called `$neat-grid` with two keys: `columns` and `gutter`.

The default number of columns is `12` and the default gutter is `20px`.

```
$neat-grid: (
    columns: 12,
```

```
    gutter: 20px
);
```

Once we have our grid setting configured, we're ready to start building.

Throughout the following examples, I've set a semi transparent backgroudn colour on each of the grid rows and columns so you can visualise them. The darker the colour, the more deeply nested the layout.

The whole concept of Neat grids is to do away with adding hundreds of grid classes to the HTML. Instead of creating loads of `divs` with classes like `column`, `small-12` or `col-md-4`, we add a meaningful classname to describe the purpose of the element, select this in our Sass and call a mixin to define it as a `grid-container` or a `grid-column`.

```
.latest-posts {
    @include grid-container;
}
.latest-video,
.latest-blog {
    @include grid-column( 6 );
}
```

This keeps all the styling logic in the CSS and keeps the HTML nice and neat and tidy.

To create our grid container, we use the mixin `grid-container`.

To create a column within, we use the `grid-column` mixin, passing a numeric value for the number of columns to span.

Nested Grid

Working with nested grid - or grids within grids - works in exactly the same way. In this example I have a sidebar on the left which spans 4 of the available 12 columns in the first grid container. The main content spans 8 columns.

To create a grid within the main content column, we need an additional `grid-container` element which here I've called `.nested-container`.

In the Sass, this nested container element uses the `grid-container` and `grid-collapse` mixins. Grid collapse removes the gutters from the outside edges of the first and last columns.

```
.nested-grid {
    @include grid-container;

    .sidebar {
        @include grid-column( 4 );
```

```
        min-height:100px;
    }
    .main-content {
        @include grid-column( 8 );
        overflow:hidden;
    }
    .nested-container {
        @include grid-container;
        @include grid-collapse;
    }
    .nested-column {
        @include grid-column( 6 );
    }
}
```

Responsive Grid

You can also create responsive grids in neat using custom grids and a media property in the grid config.

To create a custom grid, first create a new grid config map. This example is taken from the neat documentation:

```
$article-layout-grid: (
    columns: 1,
    gutter: 2rem
);
```

This grid config is for the a narrow device as the Neat grid system works mobile first. Next we create another grid config for the tablet sized grid, adding more columns and a media property that determins the breakpoint at which the grid changes from 1 column to 5.

```
$article-layout-grid--tablet-up: (
    columns: 5,
    gutter: 2rem,
    media: ”( min-width: 800px )”
);
```

Once you have the grid config variables, we can use the grid-container and grid-column mixins as before. To make the grid responsive, we use a new mixin `grid-media` and pass in the config variable for our tablet-sized grid.

Within this mixin we call `grid-column` again, setting the new column widths for the sidebar and main columns - remember our tablet sized grid has 5 columns so the sidebar will span 2 of 5 columns and the main column spans 3 of 5.

```
.article-layout {
    @include grid-container;
}

.article-layout__hero {
    @include grid-column(1, $article-layout-grid);
    min-height:150px;
    margin-bottom: 2rem;
}

.article-layout__sidebar {
    @include grid-column(1, $article-layout-grid);
    min-height:300px;
    margin-bottom:2rem;

    @include grid-media($article-layout-grid--tablet-up) {
        @include grid-column(2);
    }
}

.article-layout__main {
    @include grid-column(1, $article-layout-grid);
    min-height:400px;
    margin-bottom:2rem;

    @include grid-media($article-layout-grid--tablet-up) {
        @include grid-column(3);
    }
}
```

This responsive grid takes a bit of getting used to but for more information and more examples go and check out the Neat website where you'll also find all their excellent documentation. neat.bourbon.io.

While this grid system is quite different to many others I've experimented with, personally I like it a lot. It's a bit more work to constantly `@include` the grid container and grid columns into all the necessary selectors but it does help to keep the HTML lean and keeps all the presentation together in the CSS.

For prototyping, I'd probably still add grid class names to HTML elements (to help me work quickly) but for refining a site ready for final delivery, I can see this approach working really well.

As both Bourbon and Neat are pure Sass libraries, I'd also highly recommend having a read through their code. You'll be able to get a sense of what each mixin is doing behind the scenes and you'll likely learn a few new Sass tips along the way.