The process of making a design responsive is complex and often requires a whole load of new styling rules for different sized devices to wrangle a design into shape.

When working with a responsive project, Sass has a number of useful features that can make our lives easier and make our code more maintainable.

Nesting media queries

Media queries are one of the three pillars of responsive design:

- Fluid containers (sized in percentages)
- Flexible media (videos and images that scale and don't break out of their containers)
- Media Queries

We covered media queries in a previous AtoZ CSS video but in essence they are a way to create conditional CSS that only renders when certain characteristics about the device being used to view the content are true.

```
.box { background: blue; }

@media screen and ( min-width:500px ) {
    .box { background: red; }
}
```

In this example, my boxes will have a blue background by default but if the width of the screen is greater than a minimum width of 500px, they will be red.

Even in this very short example there is some duplication which Sass can help us reduce. Using Sass we can nest the media query declaration inside the braces of the `.box` class and avoid having to redeclare the class a second time.

```
.box {
    background:blue;

    @media screen and ( min-width:500px ) {
```

```
        background:red;
    }
}
```

Nesting is a contentious issue among developers due to the way it can mask specificity issues but in this case, the nested media query doesn't add any additional specificity and I personally find it a great way to make the code I write shorter and more readable as it groups selectors and any related media queries together.

Using Variables in Media Queries

When crafting responsive styles across a whole project it's common to have a handful of major breakpoints where a number of media queries handle styles for broad categories of different sized devices.

These might be described as small, medium and large screens or though of as the size of device: mobile, tablet, desktop. Whichever you prefer, it's likely you'll need to reference the same breakpoint sizes a number of times. This is a perfect opportunity for using a variable.

Let's have a look at an example.

I've got a simple page design here with a couple of media queries to handle the breakpoints between small and medium screens. The value 500px and 800px have been used in a couple of places.

Now, I could have used any length unit here such as `em`, `rem` or `px` but I've chosen `px` for the sake of simplicitiy as the numbers are often more relatable. We'll look at converting to relative units in the next section.

To make the pixel values more meaningful and easy to reuse, let's create a variable to reference our breakpoints.

```
$media-small: 500px;
$media-medium: 800px;
```

I like to prefix my variables with a naming convention to add extra meaning but, as will all variables, you can name these however you prefer.

To use the variables, we might attempt to replace the pixel values as we would with other variables such as my `$brand-color` used here.

```
@media screen and ( min-width: $media-small ) {
    background:$brand-color;
}
```

But this doesn't work and throws an error.

Because the variable is being used within another string of characters for the media query declaration, we need to use Sass interpolation instead.

```
@media screen and ( min-width: #{ $media-small } ) {
    // styles
}
```

Using interpolation (which you can learn more about in Episode 9 of this series) the variable expression is evaluated within the surrounding media query declaration. To complete this example, we can replace all other instances of our pixel breakpoints with variables.

This is an improvement on the raw CSS approach and we could now adjust the point at which our design responds to small or medium screens by adjusting the variable value.

However, there's still a lot of duplication in the media queries themselves which we can address in the final section.

Responsive Mixins

To streamline the process of working with media queries (and to reduce the the amount you have to type) it's possible to create a mixin to handle outputting the media query.

So we can create a very flexible mixin that allows us to craft all sorts of different breakpoints, we can create a mixin that responds to different properties and values.

I'll create a `respond-to` mixin that takes two arguments: one for the `$value` of the breakpoint - which could be something like `500px` or `800px` and one for the CSS property that the query should check about the device - such as `min-width` or `max-width`.

```
@mixin respond-to( $value, $property ) {
    @media screen and ( #{ $property }: #{ $value } ) {
        @content
    }
}
```

Calling the `@content` block within the body of the media query means that any styles passed into the mixin block will only be output when the media query is true.

To use the mixin we use the `@include` directive, specify the mixin name and pass in the two variable values. For example

```
@include respond-to( 600px, 'min-width' ) {
    width:100%;
}
```

If you find that you most commonly use `min-width` or `max-width` queries, you can create a default value for the property argument:

```
@mixin respond-to( $value, $property:'min-width' ) {
    @media screen and ( #{ $property }: #{ $value } ) {
        @content
    }
}
```

By adding the default value, if the mixin is called without the second argument, the default will be used.

```
@include respond-to( 600px ) {
    width:100%;
}
```

If we look at the compild CSS, you can see the resulting output.

```
@media screen and ( min-width: 600px ) {
    width:100%;
}
```

When working with responsive design there's always a lot to think about so if Sass can help us reduce the amount of code we have to write, it can certainly help speed up the process.