Color is hugely important for adding life and visual interest to a project and is an invaluable tool for highlighting parts of the interface that a user can interact with. In this practical episode we're going to talk all about manipulating colors programatically with Sass.

It's quite common for a project to use a palette of colours for headings, body copy, links and buttons.

When designing the visual style of a site, tools like Photoshop or Sketch may be used but sometimes, working in the browser is the fastest way to experiment as all the colours or fonts or type sizes can be changed at once by leveraging Sass variables.

Often when experimenting with colour, a front-end designer may want to tweak colours to get the just the right tone or contrast. Sass has a number of colour functions built into it that can help do just that.

Each colour function takes a base colour - which can be any valid colour format such as a keyword, hex code, rgb value, hsl value or a variable

- and the modify it by a certain amount (often as a percentage).

```
function( $color, $amount );
```

Let's illustrate some of the most useful colour functions with an example. Here I have a box with a background colour which is defined by the variable `$base-color`.

```
$base-color: #cc3f85;

.box {
    background: $base-color;
}
```

I could lighten the colour slightly with the `lighten()` function.

```
.box {
    background: lighten( $base-color, 20% );
}
```

Or darken the base colour by a certain percentage:

```
.box {
    background: darken( $base-color, 20% );
}
```

The colour could be more saturated or desaturated too:

```
.box {
    background: saturate( $base-color, 20% );
    background: desaturate( $base-color, 20% );
}
```

The colour could be inverted, made grayscale or turned into its complimentary colour (which is its opposite colour on the colour wheel).

```
.box {
    background: invert( $base-color );
    background: grayscale( $base-color );
    background: complement( $base-color );
}
```

There are a number of functions available for making a colour more transparent or more opaque which take a decimal rather than a percentage

- just like when setting the alpha channel in `rgba` or `opacity`.

```
.box {
    background: transparentize( $base-color, 0.5 );
    background: opacify( $base-color, 0.3 );
    background: rgba( $base-color, 0.4 );
}
```

Multiple Colour Functions

It's not uncommon to want to make multiple modifications to a colour such as lightening and increasing saturation; or transparentizing and darkening. This can be achieved by putting functions inside of other functions. Let's take a look at lightening and saturating a colour together:

First let's lighten the `$base-color`

```
.box {
    background: lighten( $base-color, 20% );
}
```

This can the be saturated by wrapping the saturate function around the lighten function. Think of the colour being passed to `saturate` is the result of running the `lighten` function.

```
.box {
    background: saturate( lighten( $base-color, 20% ), 10% );
}
```

If you find this a bit awkward to read, you could create a variable for the lightened colour and just pass that to `saturate`:

```
$lightened-color: lighten( $base-color, 20% );

.box {
    background: saturate( $lightened-color, 10% );
}
```

I personally prefer the first, one line option because it keeps everything together and reduces the need for additional global variables.

Practical use case for colour functions

You may be thinking that the designs you work with are much stricter about colours than letting them be arbitrarily darkened or lightened in the code - and that's a fair point.

But there are some occasions when modifying a colour is much quicker and much more flexible than endlessly using a eyedropper tool in your favourite graphics package.

A great example of this is when working with gradients, borders and shadows.

I've got a button here with a solid background colour and a bit of padding to give the text some breathing room. To give the button a bit more detailing, we could perhaps add a border and a bit of a shine with a gradient and some subtle shadows. This might sound like some kind of dated web 2.0 horror show but we can handle this tastefully - I'm convinced that gradients will be back in fashion soon, just you wait!

```
.button {
    padding:0.5em 2em;

    color:#222;
    background:lightblue;
}
```

So to add these features, instead of picking colours out of a design file, let's use Sass.

I'll start by making the existing background colour a variable and then add a thin border which is a darker version of the original blue.

```
$color-button: lightblue;
.button {
    background:$color-button;
    border:1px solid darken( $color-button, 15% );
}
```

To give a bit more depth to the button, I'll add a light blue inset shadow at the top of button:

```
$color-button: lightblue;
.button {
    background:$color-button;
    border:1px solid darken( $color-button, 15% );
    box-shadow: inset 0 1px lighten( $color-button, 20% );
}
```

And then wrap everything up with a subtle gradient:

```
$color-button: lightblue;
.button {
    background: linear-gradient( $color-button, darken( $color-button, 15% ) );
    border:1px solid darken( $color-button, 15% );
    box-shadow: inset 0 1px 1px lighten( $color-button, 20% );
}
```

Now imagine we want to change the button colour. Because we've created a system for styling buttons where all the colours are modified from a single base colour, we have nothing else to do other than change the value of our variable.

To make this code even more flexible and allow us to generate all sorts of different coloured buttons, we could turn our button class into a button mixin.

```
@mixin button( $color-button ) {
    padding:1em;
    background:linear-gradient( $color-button, darken( $color-button, 15% ));
    border:1px solid darken( $color-button, 15% );
    box-shadow: inset 0 1px 1px lighten( $color-button, 20% );
}
```

Now this mixin can be used to generate buttons of all different colours by including them as needed.

```
.button {
    @include button( lightblue );
}
.button--pink {
    @include button( hotpink );
}
```

If you're new to Sass and want more info about variables and mixins and other Sass features, do check out my Up and Running with Sass ebook which will give you a practical guide to the Sass fundamentals along with exercises, actionable advice and best practices for working with Sass.