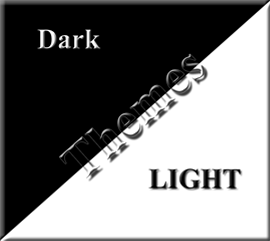
8 Switching Between Themes

Contents

[Creating a theme @mixin 2](#_Toc163906012)

[Using a Boolean value for the theme mixin 2](#_Toc163906013)

[Why does this work? 7](#_Toc163906014)



Turn on [Watch Sass](2%20Creating%20the%20Project.docx)

In your main.scss file



Creating this theme will actually be 3 parts

* Creating a Mixin definition called theme
* creating a .light class to call the mixin farther down in code
* Adding your class to the body tag in the html

# Creating a theme @mixin

We are going to be using a @mixin to accomplish this.

Add another @mixin right above your body rule, and below your other @mixin.

# Using a Boolean value for the theme mixin

**Write Boolean @mixin**

Write the definition above body rule in main. SCSS file. We are using the official SCSS function of lighten here; but both lighten and darken are provided by SASS. This means that you can say their name, and they just work.

This will take a key/value as an argument The property, you want to lighten, and then a percentage of HSL color format. This parameter will range from 0% to 100%. It is working with the third letter in HSL, which represents lightness. The first two letters represent Hue, and Saturation respectfully. But we are not using Hue and Saturation here.



@mixin theme($light-theme: true) {

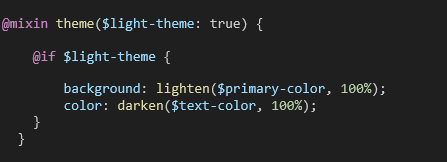
    @if $light-theme {

        background: lighten($primary-color, 100%);

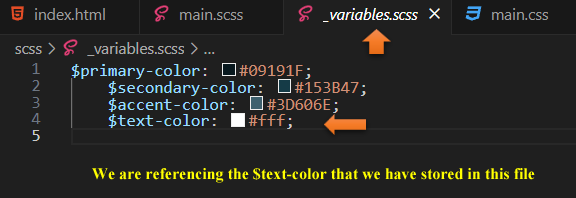
        color: darken($text-color, 100%);

    }

}



Watch it, do not get confused here when we are trying to darken our variable of $text-color. We are not trying to reference the font-colors that we have mapped at the top of our SCSS file. Instead we are referencing the color of our text. If you remember, these are the variables that we have stored externally in our **\_variable** partial file.



**Writing the light class to go into Boolean**

Under your main rule in your SCSS file, write your light class and your dark class. look at how we went about writing the light theme, we used the @include, and then we passed in the variable of light-theme: true. How cool is that.

Make sure you still have those 3 curly braces closing out that rule above it, which was a nested rule for .main.

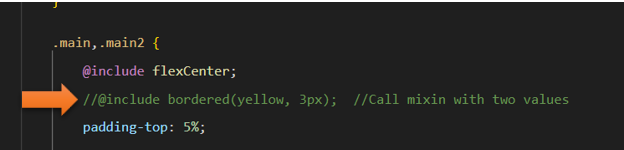
.light {

@include theme($light-theme: true);

}



Remove your border from your .main rules. It is quite ugly.

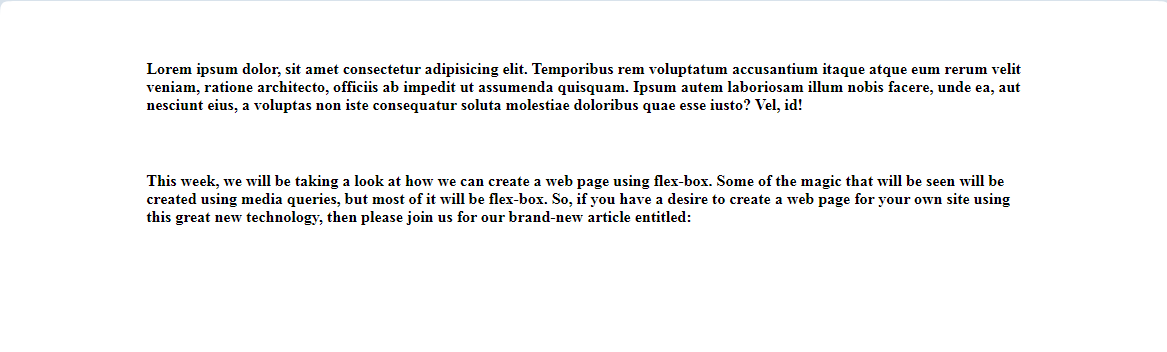


Go to your index.html to add the class to the body so we can use it.

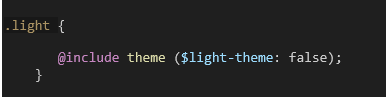
<body class="light">



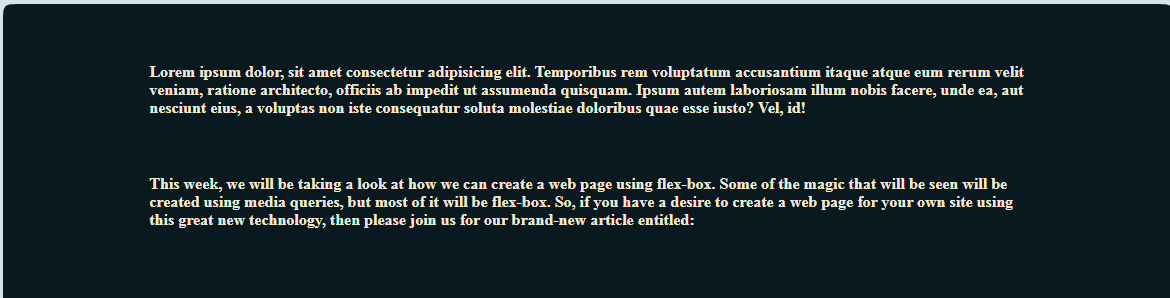
And now your theme is light with dark text.



Now if you change that Boolean value to be false you will revert back to your dark theme. You make that change in the @include for the .light class



Go to your main.html to test it



# Why does this work?

When we wrote the @mixin, we used a Boolean value of true. So, if we want the theme to be light, and it is true, it will include the light theme style. But if we set it to false when we call it in the .light class, it is not going to add these styles, and will just ignore that we even have them. So, if it is false, we will get our original dark theme. This keeps us from having to write 2 mixins, and we are able to do every thing that we need it to in just one mixin code.