6 Functions and Mix-Ins

Make sure you have [watch SASS](file:///C:\xampp\htdocs\Enlightenment\Articles\2024\6-SASS-Or-SCSS\2%20Creating%20the%20Project.docx) on

# What are Functions in SASS?

Functions in Sass are very similar to functions that are in JavaScript. So, if you are familiar with JavaScript, you should find this function stuff to be much easier to get the hang of.

# Creating your first SCSS function

Go to your **main.scss** file

We are going to be placing the functions right under our import statements on the page. Notice that what is in the parenthesis is $weight-name, this was a made up variable name, but it will represent the font -weights, font-weights is the official CSS name, and inside that is the weight of bold. so now our own variable that we threw in the parenthesis will take the place of that official name.

map-get (official name, our name)

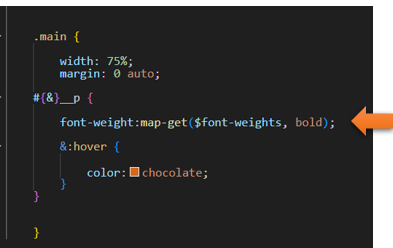
@function weight($weight-name) {

    @return map-get($font-weights, $weight-name);

}

This will be simplifying some of our code, so it does not look so clucky.

Scroll down to where you wrote that nested code. We will be targeting this line of code.



…and with our new function, we can change it to this: We also changed the code from bold to regular, but it is much simpler to write now.

 .main {

        width: 75%;

        margin: 0 auto;

    #{&}\_\_p {

        font-weight:weight(regular);

        &:hover {

            color:chocolate;

        }

    }

    }

bold



regular

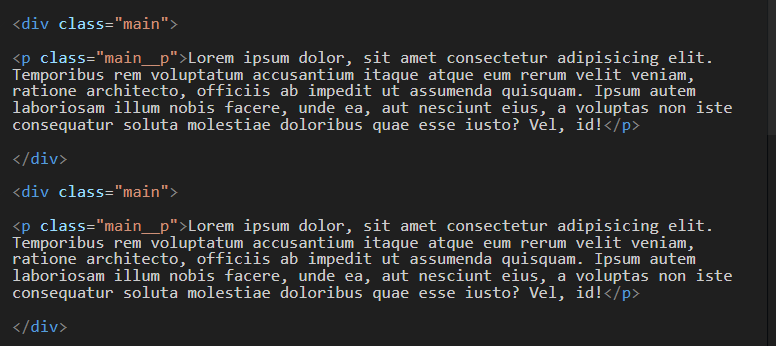


# Mix-ins

Mix-ins are similar to functions.

Go to the **index.html** file and add another paragraph

highlight the entire paragraph on the page, and **alt- shift -down arrow**.



Back in the **main.SCSS**.

Add the top 3 lines to your **.main** rule

.main {

        display:flex;

        justify-content: center;

align-items: center;

padding-top: 5%;

        width: 75%;

        margin: 0 auto;

    #{&}\_\_p {

        font-weight:weight(regular);

        &:hover {

            color:chocolate;

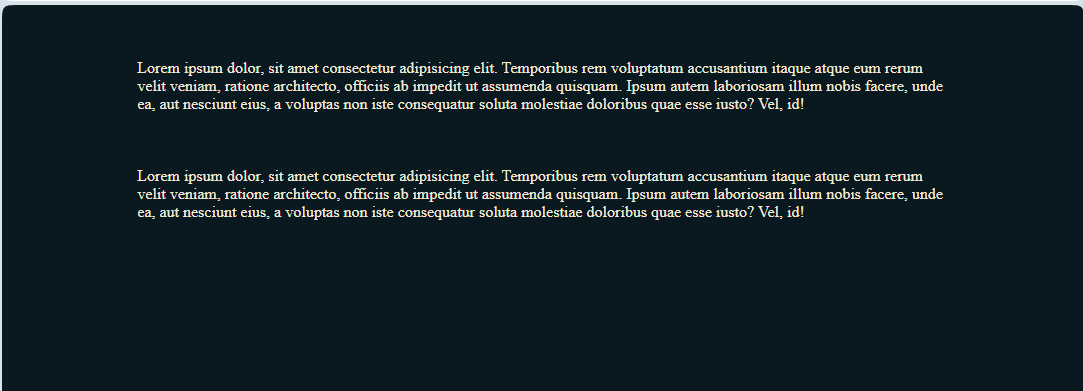
        }

    }

    }

Watch it, remember if you do not have [Watch SASS](file:///C:\xampp\htdocs\Enlightenment\Articles\2024\6-SASS-Or-SCSS\2%20Creating%20the%20Project.docx) on, you will not be able to view this

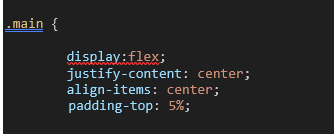
**Save** that SASS file and then switch back to the index.html file. And it now is flexible with size of viewport.



# Creating a Mix-In



Inside of your main.SCSS file



display:flex;

        justify-content: center;

        align-items: center;

Cut those top 3 lines from the rule properties that we just added to .main, and put them into our new mixin

@mixin flexCenter {

    display:flex;

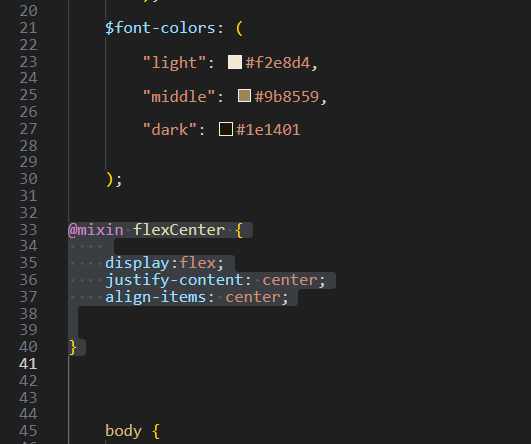
    justify-content: center;

    align-items: center;

}

You are going to place the mixin before your first tag, which for us is the body tag, and after The maps

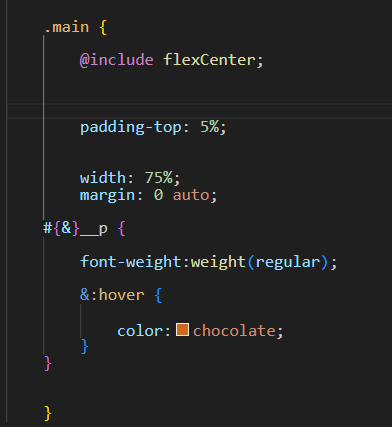
z



# How to include the mixin in your CSS rule

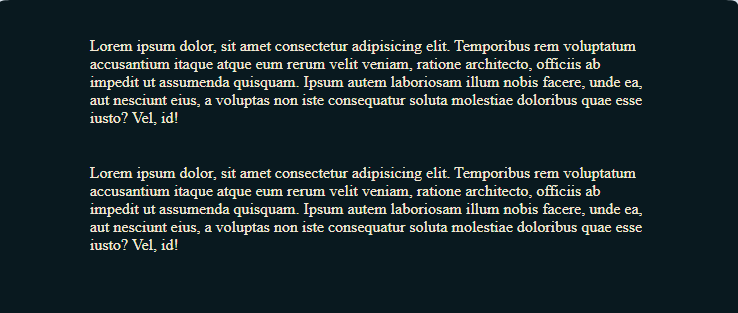
Now that you have defined the mixin, you need to know how to place it into your rule so that our .main can use it.

@include flexCenter;



Now, Save it and go back to index.html, to test the code

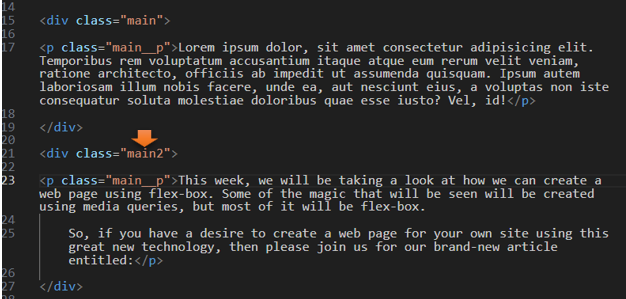
… and it is still reading the code correctly!



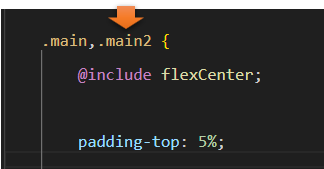
# Mixins Arguments

Remember to [turn on SASS](file:///C:\xampp\htdocs\Enlightenment\Articles\2024\6-SASS-Or-SCSS\2%20Creating%20the%20Project.docx)

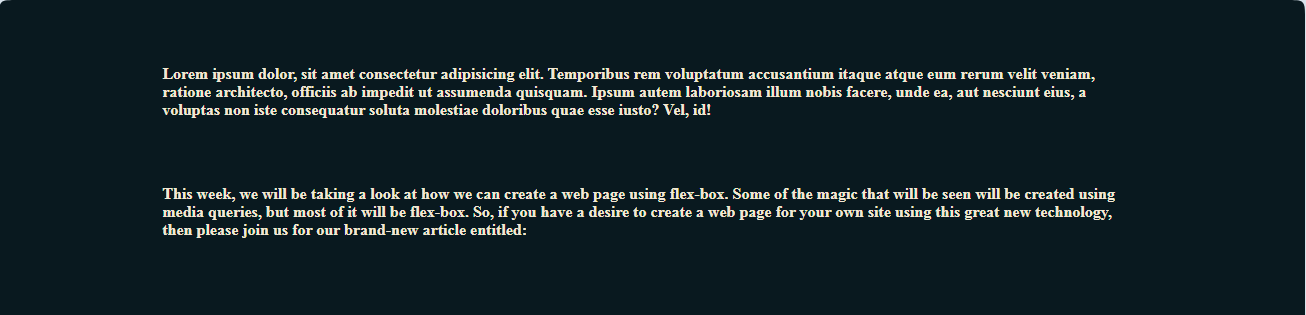
Ok, we are going to start out by giving our second paragraph a different class name in the index.html page.



We can easily give this div the same initial properties, by going to the class definition for .main, adding a coma, and adding the name of the new class. Rember to put the dot in front of .main2, it is a class and it will need it to work

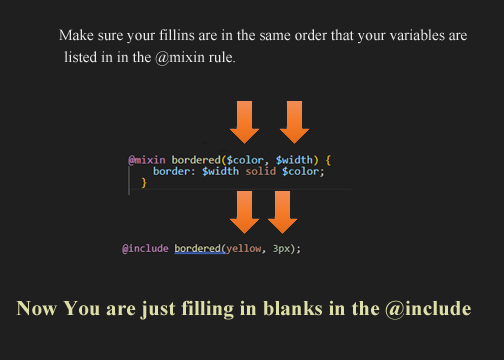


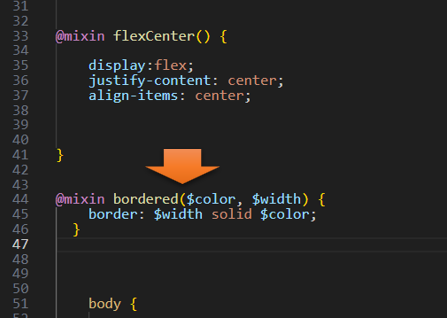
Now when you go back to the index.html file and run it, both paragraph’s settings should look alike.



Now let’s just add another mixin at the top of our code. Under your other mixin rule, and before the body rule. See how we added the two arguments in the parenthesis after the mixin name of bordered.

Notice that inside of the block of code, that we have something that we sort off recognize from CSS rule creation, except that instead of values for border, we are throwing our made-up variables into border. This is what makes our variables…well, variable. We can fill in the blanks when we are inside of any rule. The rule we chose to throw it into is the .main… this time. But we can throw it into any rule, just remember to fill in those blanks, and they need to be filled in in the same order.





Now go down to the code of .main, .main2, and then add another include

.main,.main2 {

        @include flexCenter;

        @include bordered(yellow, 3px);  // Call mixin with two values

        padding-top: 5%;

        width: 75%;

        margin: 0 auto;

    #{&}\_\_p {

        font-weight:weight(regular);

        &:hover {

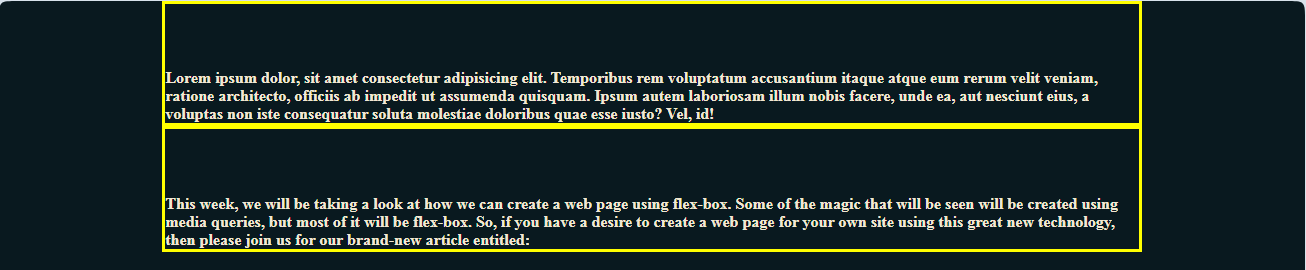
            color:chocolate;

        }

    }

    }

You can play with those parameters in the @include inside of the .main, .main2 rule. You can see by the web page below that the project is listening to the new @mixin rules.



Check out your CSS regular file, and you can see how it was added as normal CSS there.

