7 Functions and Mix-Ins

Contents

[What are Functions in SASS? 2](#_Toc162677533)

[What is map-get? 2](#_Toc162677534)

[Creating your first SCSS function 2](#_Toc162677535)

[Mix-ins 4](#_Toc162677536)

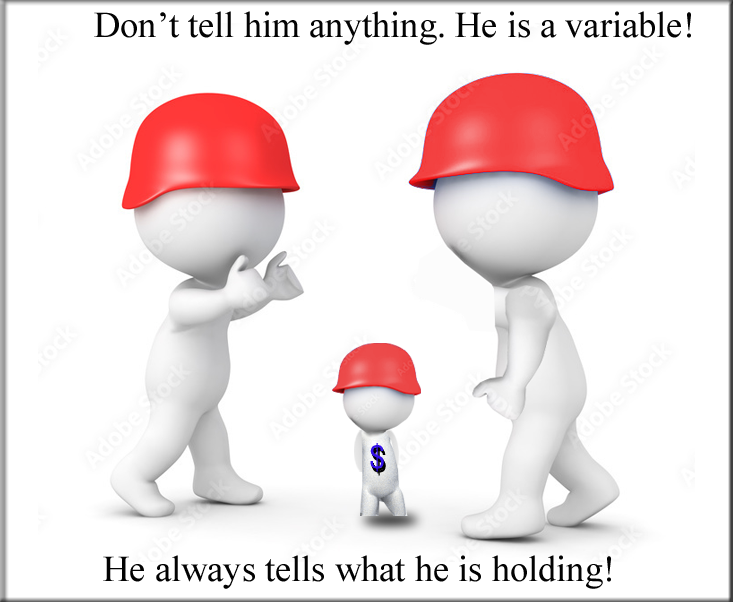
[Creating a Mix-In 7](#_Toc162677537)

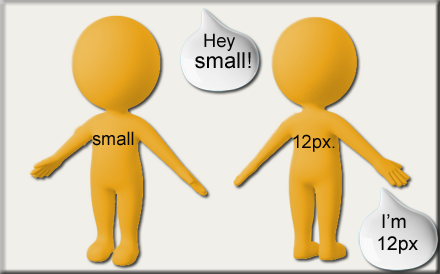
[How to include the mixin in your CSS rule 8](#_Toc162677538)

[Mixins Arguments 9](#_Toc162677539)

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

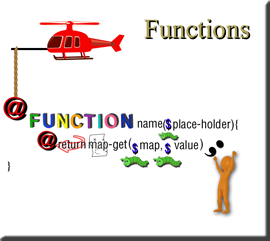
What is a Variable?

****



Basically, if you really think about it, that is what all variables are; they are just place-holders to hold data stuff. So, move over memory, I am pushing in my variables, and now I got some room to put some of my stuff.

# What is a Function?



**\*Note**-In our illustration above, you will see that our place-holder variable, which is in the parenthesis in the first line, is what we will use for our value variable, in the second line of our code. So, the place-holder will be filled in with our value, and both the value, and the place-holder, will be the exact same thing.

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.

A function is something that you would use if you do not want to keep repeating the same code over and over. You can call a function’s name any where in your code, and it is just as if we had written the full code right in that spot to be use, at that moment by any part of your code that wishes to use it.

Filling in the place holder

Filling in the place holder later on in the code



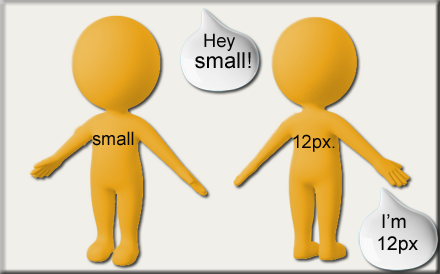
# What is map-get?

By using map-get, in SCSS, we can take several values, and then throw them into a single variable. You can kind of think of these parenthesis, as a way to wrangle in your multiple values, and put them into a group. When the program sees these parenthesis with stuff in them, it knows this stuff goes together in some way.

So, inside of these parenthesis we are using what we want to map, and its value. These are called key: value pairs, at times this is referred to as map,key, which can certainly be confusing, to say the least. But anyways, to continue, the map, key is inside of parenthesis, so we know this stuff goes together. We separate these pairs, inside of those parenthesis with commas.

Notice, in the table below, how we take the value, on the right side of the equation, and throw it into the map on the left side. So, now anytime we say “small” it will mean 12px.

small is 12px; that is what I am; call my name, and I will give you just that-- 12px.



Take this for instance:

|  |  |
| --- | --- |
| Function | Description & Example |
| map-get(map, key) | Returns the value for the specified *key*in the map. |
| Example: | **$font-sizes:** ("small": 12px, "normal": 18px, "large": 24px) **map-get**($**font-sizes**, "small") Result: 12px |

# Creating your first SCSS function

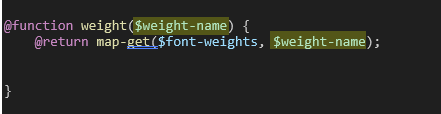
Go to your **main.scss** file

We are going to be placing the functions right under our import statements on the page. In our example, the $weight-name is where we are going to place the property, which we are grabbing out of $font-weights. So, our function name is weight. the word weight is now what we can use to represent this entire block of code thing, when we call it.

$font-weights and $weight-name are just place holders in the map-get function. We will be filling in what we want to really use here when we call it later in the code. By putting the name $weight-name in the parenthesis, it also initializes it (so-to-speak) and enables us now to use our new variable inside of the function block.

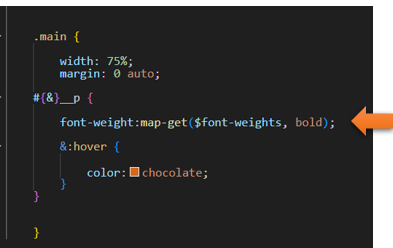
syntax

**map-get** (**map, value**)



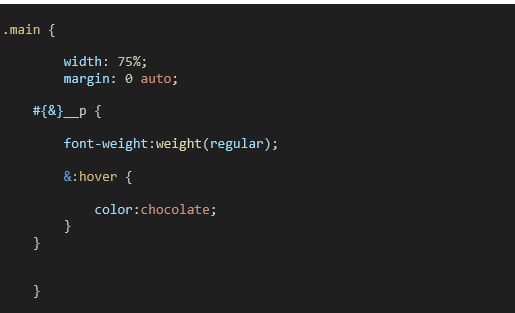
This will be simplifying some of our code, so it does not look so clucky.

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.

**Notice** how in this code we are replacing $font-weights with the actual CSS name of font-weight. Then we use a colon, and then we use the name of our function. Inside of the parenthesis is where we fill in our place-holder, with the property of either bold, or regular. You will see from this next example that our function holds all the code now to tell it that the property of regular is what we want to be pulling from font-weight.



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**.

Watch it, do not duplicate the entire div with the class of main, we are only duplicating the paragraph with in it.

<div class="main">

<p class="main\_\_p">First paragraph in main.Lorem ipsum dolor, sit amet consectetur adipisicing elit. Temporibus rem voluptatum accusantium itaque atque eum rerum velit veniam, ratione architecto, officiis ab impedit ut assumenda quisquam. Ipsum autem laboriosam illum nobis facere, unde ea, aut nesciunt eius, a voluptas non iste consequatur soluta molestiae doloribus quae esse iusto? Vel, id!</p>

<p class="main\_\_p">First paragraph in main.Lorem ipsum dolor, sit amet consectetur adipisicing elit. Temporibus rem voluptatum accusantium itaque atque eum rerum velit veniam, ratione architecto, officiis ab impedit ut assumenda quisquam. Ipsum autem laboriosam illum nobis facere, unde ea, aut nesciunt eius, a voluptas non iste consequatur soluta molestiae doloribus quae esse iusto? Vel, id!</p>

</div>

Back in the **main.SCSS**.

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

        display:flex;

        justify-content: center;

        align-items: center;

.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;

        }

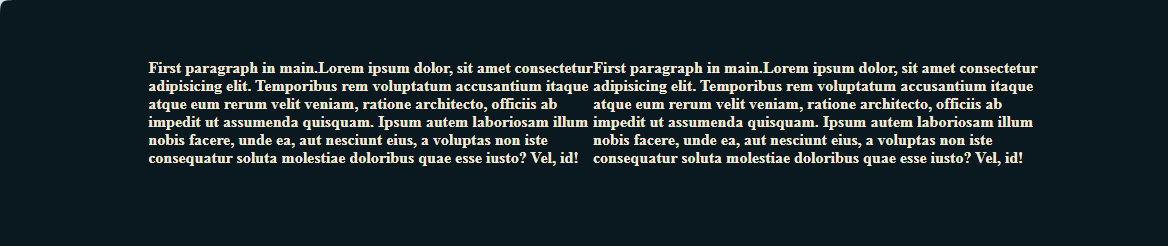
    }

    }

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.

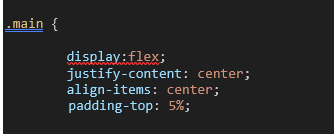
Yes it looks bad, with out a gap between columns but we will fix that. And you will find that by default if you have more than one paragraph inside of one div it will default to columns.



# 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. We are also adding a gap to make the inbetween sections of the paragraphs look better

@mixin flexCenter() {

    display:flex;

    justify-content: center;

    align-items: center;

    gap: 35px;

}

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

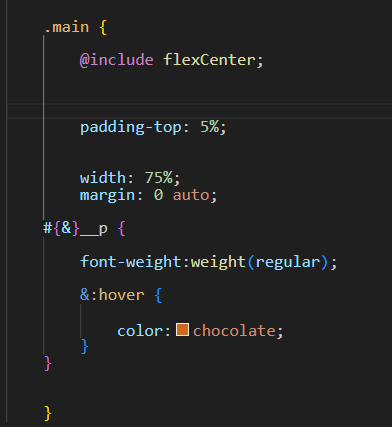
z



# How to include the mixin in your CSS rule

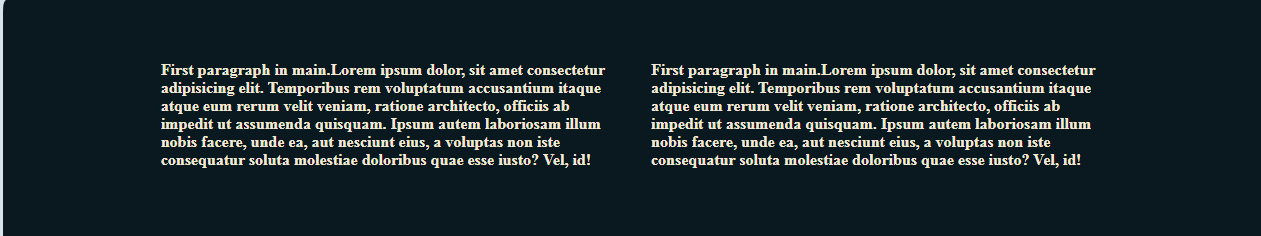
Now that you have defined the mixin, you need to know how to place it into your rule so that our .main can use it.

@include flexCenter;

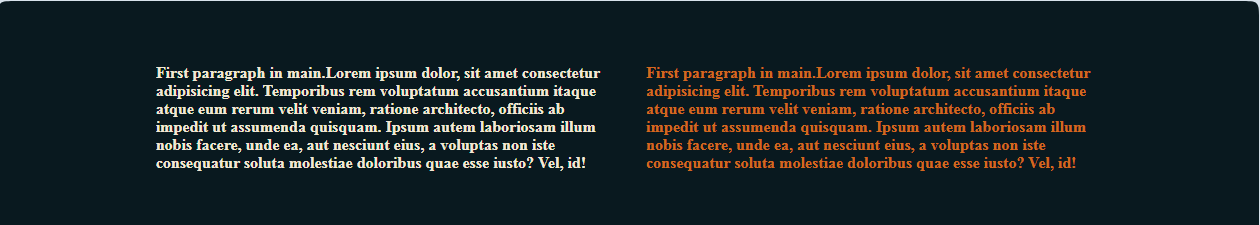


Now, Save it and go back to index.html, to test the code

… and it is still reading the code correctly!



and when we hover…

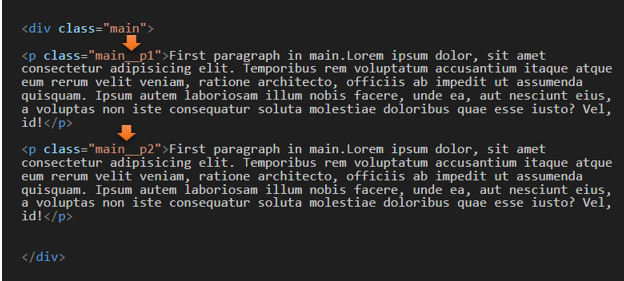


# Mixins Arguments

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.

In our index.html file



In Our main. SCSS file

go to our @mixin flexCenter and pass in the variable of $direction

Then inside the rule, add the line that says flex-direction: $direction;

@mixin flexCenter($direction) {

    display:flex;

    justify-content: center;

    align-items: center;

    gap: 35px;

    flex-direction: $direction;

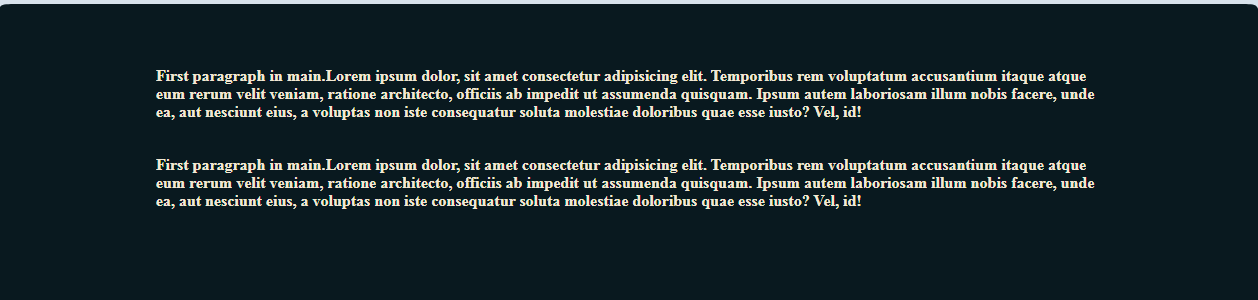
}

Now go to the rule for .main, and make changes to use this new argument of direction. We add parenthesis and inside, we fill in the blank for $direction. We now want our paragraphs to display in a row, instead of a direction.

.main {

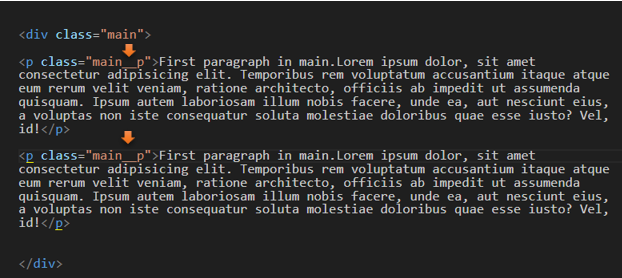
        @include flexCenter(column);

So, now the display is in a column instead of the rows that we had before



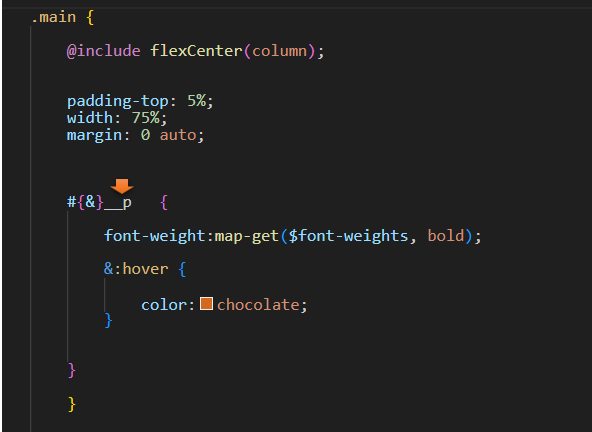
To get the hovers to work again just change those classes on the paragraph tag back to what it was

In the index.html file



In the main. SCSS file

Now our hover will work again



All Better

