<https://dev.to/finallynero/scss-cheatsheet-7g6>

[](https://dev.to/finallynero)

[Nero Adaware](https://dev.to/finallynero)

Posted on Mar 11, 2019

# SCSS Cheatsheet

[#webdev](https://dev.to/t/webdev)[#beginners](https://dev.to/t/beginners)[#cheatsheet](https://dev.to/t/cheatsheet)[#css](https://dev.to/t/css)

### What is SCSS

SCSS simply means Sassy CSS. Sassy CSS is a CSS preprocessor that gives you access to use features that are not available in Vanilla(normal) CSS.  
In this article I will list, explain and give Examples of some of those features.

#### Variables

In SCSS you can declare variables easily. To create a variable just add a $sign to the variable name and set them like a normal CSS property.

// Font-weight for a project

$font-light : 400

$font-thick: 500

$font-heavy: 600

.element{

Font-weight: $font-light;

}

//color palette for a project to maintain consistency

$product-dark-blue: #324e85

$product-light-blue:#4c7396

$product-lighter-blue:#9bb7cf

.element{

color: $product-dark-blue;

}

#### Nesting

SCSS allows you to nest CSS rules

// vanilla CSS

.container {

Width: 100%;

Color: grey;

Background-color: green;

}

.container div {

Border: 1px solid black;

}

.container div a {

text-decoration: none;

color: #f2f2f2;

}

.container div a::hover {

color: #b2b2b2;

}

// SCSS

.container {

width: 100%;

color: grey;

background-color: green;

div {

border: 1px solid black;

a {

text-decoration: none;

color: #f2f2f2;

&::hover {

color: #b2b2b2;

}

}

}

}

Sassy isn't It? SCSS helps you write cleaner and concise CSS.

#### Inheritance(Using @extend or @mixin)

###### @extend

@extends helps you inherit the properties of another class.

// SCSS

.header {

color: grey;

}

.sub-header {

@extend .header;

font-size: 40px;

}

Below is what the SCSS compiles to

// Compiled CSS

.header, .sub-header {

color: grey;

}

.sub-header {

font-size: 40px;

}

##### Mixin

Mixin is another way SCSS implement inheritance using @mixin. You can achieve the same effects of @extend using mixin. You first create the mixin using @mixin then add it to any class that needs that property using @include

// SCSS

//create mixin

@mixin red-color{

color: grey;

}

.header{

@include red-color; /\* add mixin \*/

}

.sub-header{

@include red-color;

font-size: 40px;

}

.header {

color: grey;

}

.sub-header {

color: grey;

font-size: 40px;

}

Can you see the different between the compiled css of @extend and @mixin, @mixin puts the style in both selectors(header and sub-header) while @extend seperates both seletors with a comma and then apply this style to them.

There is one thing though @mixin can do that @extend cannot, That is pass parameters and use it. @mixin can also take default values for the parameter.

// SCSS

@mixin fontSize ($params: 10px){

font-size: $params;

}

.header{

@include fontSize(20px);

}

.sub-header{

@include fontSize(20px);

}

// Compiled CSS

.header {

font-size: 20px;

}

.sub-header {

font-size: 20px;

}

You can use any of them if you need to inherit a class but the best practice according to [CSS tricks](https://css-tricks.com/the-extend-concept/) is to use @extend when you are not passing parameters.

#### Import

SCSS allow you to import other SCSS stylesheet into a SCSS file using @import

@import “button.scss” or @import “button”

#### Partials

Partials are SCSS files you don’t what to be compiled to CSS but you want to import them(using @import) into another file. To create a partial you just need to add an underscore to the beginning of the file name \_font.scss, then you can import them with or without the underscore.  
Partials helps to modularize your code and separate concerns.  
For example in my projects I partials for colors, fonts, buttons e.t.c then I import them into a main.scss file.

/\* \_colors.scss(partial) \*/

$light-gray: #F2F2F2

$dark-gray: #737373

/\*EOF colors.scss\*/

/\* \_buttons.scss(partial) \*/

.button-primary{

color: #4c7396;

background-color: #ffffff;

}

.button-secondary{

background: #4c7396;

color: #FFFFFF;

}

/\*EOF buttons.scss\*/

/\* main.scss \*/

@import "\_buttons.scss"

@import "\_colors.scss"

#### Operators

Scss offers you different kind of operators that you can use in your CSS. Arithmetic operators like :

* Addition(+)
* Subtraction(-)
* Division(/)
* Multiplication(\*) e.t.c

// SCSS

@mixin top-margin ($margin){

margin-top: 30px + $margin;

}

.container{

width: 800px - 80px;

@include top-margin(10px);

}

//Compiled CSS

.container {

width: 720px;

margin-top: 40px;

}

Note that arithmetic operators only work when both values use the same unit i.e rem, em, px

They also support comparison operators like ==, !=, <, >, >=, <= and logical operators and, or, not.

You can check out this article for more on Operators

## [**SASS Operators**](https://dev.to/sarah_chima/sass-operators-56f)

### [**Sarah Chima ・ Nov 24 '17 ・ 5 min read**](https://dev.to/sarah_chima/sass-operators-56f)

[#sass #css #beginners](https://dev.to/sarah_chima/sass-operators-56f)

#### Color Functions

Scss provides some function that can be used to manipulate colors. Some of them include:

* mix($colorX, $colorY, weight) : This function is used to mix two color together. First argument is the first color, second is the second color and the third argument is the percentage of first color you want to mix.

mix(blue, grey, 30%) /\*results 30% blue and 70% grey\*/

* lighten($color, $amount): this function is used to return a lighter color. The first argument is the color and the second is the percentage of how much you want to lighten it.

lighten(#ff0000, 30 ) /\*results #ff9999/\*

* darken($color, $amount): Takes similar arguments as lighten function but this returns a darker color specified.

darken(#ff0000, 30 ) /\*results #660000\*/

* opacify($color, $amount): This function returns a color with the opacity increase. The first argument is the color and the second is value between 0 and 1.
* transparentize($color, $amount): This function makes a color more transparent, it takes similar arguments to opacify. It returns a color with the opacity reduced. You can say it is the opposite of opacify.

These are just some of the color functions available in Scss, you can check the [docs](https://sass-lang.com/documentation/file.SASS_REFERENCE.html) for more.

#### Other functions

Scss still offers other functions that might be useful to you

* str-length($string): This function returns the number of characters in a string.
* percentage($number): This function converts number without unit to a percentage.
* round($number): This function rounds a number to the nearest whole number.
* min($number1, $number2, $number3, …..): this function returns the minimum value from a set of numbers.
* random(): This returns a random number and takes no arguments.
* quote($string): This function adds quotes to a string.
* unquote($string): This function removes quotes from a string.
* to-lower-case($string): This function converts a string to lowercase.
* to-upper-case($string): This function converts a string to uppercase.

#### Loops

We can use for loops in our css thanks again to Scss. This can be really useful in creating utility class. You can create classes for your color, font-size , margin , padding and a lot of other properties.

Below are some examples of loops I used to create utility classes in my projects

/\* Generate utility classes for font-size \*/

@for $x from 1 through 70 {

.font-size-#{$x} {

font-size: 0px + $x;

}

}

So I can easily have a class of font-size-20 if I need that on an element.

/\* Generate utility classes for margin \*/

@for $i from 0 through 500 {

.m#{$i} {

margin: 0px + $i;

}

.mt#{$i} {

margin-top: 0px + $i;

}

.mb#{$i} {

margin-bottom: 0px + $i;

}

.ml#{$i} {

margin-left: 0px + $i;

}

.mr#{$i} {

margin-right: 0px + $i;

}

}

If you have used loops to create utility classes in your projects please share in the comments so others can adopt them. Loops are like my favorite scss feature

#### Conditionals

Another awesome feature of scss is the ability to use If/else statements in css. I have not really used this feature in any of my projects but I think they are awesome.

$bg: pink;

$bg-mobile: red;

p {

@if $bg == pink {

color: blue;

} @else if $bg-mobile == red {

color: green;

} @else {

color: grey;

}

}

This is just a simple use case for if else statements, if you have used it in a project please share a code snippet in the comment.

Scss has a lot more features than what is covered in this article but these are the commonly used ones. You can check out their [docs](https://sass-lang.com/documentation/file.SASS_REFERENCE.html).