**FREE eBook**

# LEARNING

# sass

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#sass

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sass

It is an unofficial and free sass ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sass.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with sass

## Remarks

This section provides an overview of what sass is, and why a developer might want to use it.

It should also mention any large subjects within sass, and link out to the related topics. Since the Documentation for sass is new, you may need to create initial versions of those related topics.

**Why SASS?**

- Inheritance feature
- We can use conditional statements
- More functional than traditional `CSS`
- Efficient and clear way to write `CSS`

## Versions

| Version | Release Date |
|---|---|
| 3.4.22 (Current) | 2016-03-28 |
| 3.4.0 | 2014-08-18 |
| 3.3.0 | 2014-03-07 |
| 3.2.0 | 2012-08-10 |

## Examples

### Setup

When it comes to using SASS, there are multiple ways of setting up your workspace. Some people prefer to use command line tools (probably Linux people) and others prefer to use GUI applications. I'll cover both.

## Command Line Tools

The 'Install SASS' page at `sass-lang.com` covers this quite well. You can use SASS with Ruby (which can be installed from a Linux package manager or you can download the installer on Windows). macOS comes with Ruby pre-installed.

Once you've installed Ruby, you need to install SASS (in some cases, `sudo` may not be needed):

```
sudo gem install sass
```

Finally, you can check you've installed SASS with `sass -v`.

# GUI Applications

Whilst there are a number of GUI Applications that you can use, I recommend Scout-App. It auto-builds and compresses your CSS files for you, on file save and supports macOS, Windows and Linux.

## Variables

If you have a value that you use often, you can store it in a variable. You could use this to define color schemes, for example. You would only have to define your scheme once and then you could use it throughout your stylesheets.

To define a variable, you must prefix its name with the $ symbol. (Like you would in PHP.)

You can store any valid CSS property value inside a variable. Such as colors, fonts or URLs.

**Example #1:**

```
$foreground: #FAFAFA;
$background: rgb(0, 0, 0);

body {
    color: $foreground;
    background-color: $background;
}

p {
    color: rgb(25, 25, 20);
    background-color: $background;
}
```

## Importing

Let's assume the following scenario: *You have two stylesheets:* `_variables.scss` *and* `layout.scss`*. Logically, you keep all your variables inside your variable stylesheet but want to access them from your layout stylesheet.*

> **NOTE:** You may notice that the variables stylesheet has an underscore ('_') before it's name. This is because it's a partial - meaning it's going to be imported.
>
> **sass-lang.com** says the following about partials: *You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. This is a great way to modularize your CSS and help keep things easier to maintain. [...] The underscore lets Sass know that the file is only a partial file and that it should not be generated into a CSS file. Sass partials are used with the @import directive.*

SCSS variables are great for this scenario. Let's assume that your `_variables.scss` looks like this:

```
$primary-color: #333;
```

You can import it with `@import` and then the stylesheet's name in quotes. Your layout stylesheet may now look like this (take note of there not being an underscore or file extension in the import):

```
@import 'variables';
body {
  color: $primary-color;
}
```

---

This would output something like the following:

```
body {
  color: #333;
}
```

## Nesting

layout.scss

```
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
        li {
            margin: 0 5px;
           }
       }
}
```

output

```
nav ul {
    margin: 0;
    padding: 0;
    list-style: none;
}
nav ul li {
    margin: 0 5px;
}
```

## Comments

SASS supports two types of comments:

- Inline comments - These only span one line and are usually used to describe a variable or block. The syntax is as follows: `// Your comment here` (you prepend it with a double slash (`//`) and the rest of the line is ignored by the parser.

- Multiline comments - These span multiple lines and are usually used to display a copyright or license at the top of a document. You can open a multiline comment block with `/*` and you can close a multiline comment block with `*/`. Here's an example:

```
/*
   This is a comment
   It's a multiline comment
   Also a hiaku
*/
```

Read Getting started with sass online: https://riptutorial.com/sass/topic/2045/getting-started-with-sass

# Chapter 2: Compass CSS3 Mixins

## Introduction

Getting started guide using Sass exentsion Compass. Compass is very useful when dealing with CSS3 as it provides mixins to write 1 line in order to support every browser using CSS3 features. It is also great to include sprite images.

## Examples

### Set up environment

Open your command line

## Installation using Ruby

```
gem update --system

gem install compass
```

---

## Create a Project

```
compass create <myproject>
```

This will initialize a compass project. It will add a folder called . The folder will look like have the following structure:

| File/Folder | description |
| --- | --- |
| sass/ | Put you sass/scss files in this folder |
| stylesheets/ | In this folder your compiled css will be stored |
| config.rb | Configure compass - e.g. folder path, sass compilation |

---

## Use compass

```
compass watch
```

This will compile your sass files every time you change them. The sass folder path can be changed inside of the config.rb

### Using CSS3 with compass

You can find a complete reference which CSS3 components are supported on this page

In order to use CSS3 in your project Compass provides mixins to support CSS3 features in every browser. On top of your Sass/Scss file you have to specify that you want to use compass

```
@import "compass/css3";
```

---

## Border-radius

Include border-radius with compass in your sass file:

```
div {
    @include border-radius(4px);
}
```

CSS output

```
div {
  -moz-border-radius: 4px;
  -webkit-border-radius: 4px;
  border-radius: 4px;
}
```

As you can see you can use the normal CSS name. Just put @include in front of it and use ( ) to set your value.

---

## Flexbox Example

```
.row {
  @include display-flex;
  @include flex-direction(row);
}
```

CSS Output

```
.row {
  display: -webkit-flex;
  display: flex;
  -webkit-flex-direction: row;
  flex-direction: row;
}
```

---

## Conclusion

This are only two examples. Compass provides much more CSS3 mixins. It is very handy to use Compass and you don't have to worry that you have forgot defining a CSS3 component for a specified browser. If the browser supports the CSS3 feature, compass will define it for you.

Read Compass CSS3 Mixins online: https://riptutorial.com/sass/topic/10600/compass-css3-mixins

# Chapter 3: Convert units

## Examples

### Convert px to (r)em

To convert px to em or rem you can use the following function:

```
@function rem-calc($size, $font-size : $font-size) {
    $font-size: $font-size + 0px;
    $remSize: $size / $font-size;
    @return #{$remSize}rem;
}

@function em-calc($size, $font-size : $font-size) {
    $font-size: $font-size + 0px;
    $remSize: $size / $font-size;
    @return #{$remSize}em;
}
```

The `$font-size` is the original font size.

For example:

```
$font-size: 14;

body {
  font-size: #{$font-size}px;
  font-size: rem-calc(14px); // returns 1rem
  // font-size: rem-calc(28); // returns 2rem
}
```

Read Convert units online: https://riptutorial.com/sass/topic/6661/convert-units

# Chapter 4: Extend / Inheritance

## Syntax

- `@extend .<className>`
- `@extend .<className>, .<className>`
- `@extend .<className> !optional`
- `@extend .<className>, .<className> !optional`

## Parameters

| Parameter | Details |
| --- | --- |
| className | The class that you want to extend. |

## Remarks

Sass' `@extend` rule allows you to share CSS properties across multiple classes, keeping code DRY and easier to read.

## Examples

### Extend a Class

```
.message
  color: white

.message-important
  @extend .message
  background-color: red
```

This will take all of the styles from `.message` and add them to `.message-important`. It generates the following CSS:

```
.message, .message-important {
  color: white;
}

.message-important {
  background-color: red;
}
```

### Extend from Multiple Classes

```
.message
  color: white
```

```
.important
  background-color: red

.message-important
  @extend .message, .important
```

In the above code `@extend` is used in one line to add multiple classes' code to `.message-important`, however, it is possible to use one extend per line like this:

```
.message-important
    @extend .message
    @extend .important
```

Either one of these methods will generate the following CSS:

```
.message, .message-important {
  color: white;
}

.important, .message-important {
  background-color: red;
}
```

## Chaining Extends

```
.message
  color: white
  background: black

.message-important
  @extend .message
  font-weight: bold

.message-error
  @extend .message-important
  font-style: italic
```

This code causes `.message-error` to extend from `.message-important`, which means that it will contain code from both `.message-important` and `.message`, since `.method-important` extends from `.message`. This results in the following CSS:

```
.message, .message-important, .message-error {
  color: white;
  background: black;
}

.message-important, .message-error {
  font-weight: bold;
}

.message-error {
  font-style: italic;
}
```

**Disclaimer: Make sure that the class(es) you are extending from occur only *once* in the code, otherwise Sass may generate some messy, convoluted CSS.**

## Optional Extends

Sometimes, you may want an `@extend` to be optional, and not require the specified class to exist in your code.

```
.message-important
  @extend .message !optional
  background: red
```

This will result in the following CSS:

```
.message-important {
  background: red;
}
```

**Disclaimer: This is useful during development when you may not have all of your code written yet and don't want errors, but it should probably be removed in production because it could lead to unexpected results.**

## Placeholders

Sometimes you will create classes that won't be used in their own right, rather only be extended inside other rule sets. This means that the compiled CSS file will be larger than it needs to be. Placeholder selectors solve this problem.

Placeholder selectors are similar to class selectors, but they use the percent character (%) instead of the (.) used for classes. They will not show up in the compiled CSS.

```
%button {
    border: 5px solid black;
    border-radius: 5px;
    margin: 0;
}

.error-button {
    @extend %button;
    background-color: #FF0000;
}

.success-button {
    @extend %button;
    background-color: #00FF00;
}
```

This will compile to the following CSS:

```
.error-button, .success-button {
    border: 5px solid black;
    border-radius: 5px;
```

```
    margin: 0;
}

.error-button {
    background-color: #FF0000;
}

.success-button {
    background-color: #00FF00;
}
```

## Extending the parent

Typically trying to extend the parent like so:

```
.parent {
  style: value;

  @extend &;
}
```

Will result in an error, stating that the parent cannot be extended. This makes sense, but there's a workaround. Simply store the parent selector in a variable.

```
.parent {
  $parent: &;
  style: value;
  @extend #{&};
}
```

There's no benefit to doing this in the above example, however this gives you the power to wrap parent styles from within an included mixin.

Read Extend / Inheritance online: https://riptutorial.com/sass/topic/2894/extend---inheritance

# Chapter 5: Functions

## Syntax

- @function function-name(parameter) { /* Function body */ }

## Examples

### Basic Functions

A function is similar in look to a mixin but it doesn't add any styles, it only returns a value. Functions should be used to prevent repeated logic in your styles.

Sass has some built-in functions that are called using the standard CSS function syntax.

```
h1 {
    background: hsl(0, 25%, 50%);
}
```

Functions are declared using the below syntax,

```
@function multiply(x, y) {
    @return x * y;
}

// example use below
h1 {
    margin-top: multiply(10px, 2);
}
```

In the code above, `@function` declares a function, and `@return` signifies the return value.

Read Functions online: https://riptutorial.com/sass/topic/4782/functions

# Chapter 6: Installation

## Remarks

This covers Ruby only, which is the main SASS compiler for many systems but other options exist. A very common one for any node developer would be node-sass which could be easier, and orders of magnitude faster, for many users.

## Examples

### Mac

Ruby comes pre-installed on a Mac computer.

Follow the instructions below to install Sass:

1. Open CMD
2. Run `gem install sass`
3. If you get an error message, try `sudo gem install sass`
4. Check it works using `sass -v`

### Linux

Ruby will need to be installed first before setup. You can install Ruby through the apt package manager, rbenv, or rvm.

Then Run

```
sudo su -c "gem install sass"
```

### Windows

The fastest way to get Ruby on your Windows computer is to use Ruby Installer. It's a single-click installer that will get everything set up for you super fast. After installing Ruby, follow the instructions below to install Sass:

1. Open CMD
2. Run `gem install sass`
3. If you get an error message, try `sudo gem install sass`
4. Check it works using `sass -v`

Read Installation online: https://riptutorial.com/sass/topic/2052/installation

# Chapter 7: Loops and Conditons

## Examples

### While loop

The `@while` directive will loop over a block of code until the condition specified becomes false. In the following example, this loop will run until `$font-size <= 18` while incrementing the value for `$font-size` by 2.

```
$font-size: 12;

@while $font-size <= 18 {
    .font-size-#{$font-size} {
        font-size: ($font-size * 1px);
    }

    $font-size: $font-size + 2;
}
```

Output of above code

```
.font-size-12 {
  font-size: 12px;
}

.font-size-14 {
  font-size: 14px;
}

.font-size-16 {
  font-size: 16px;
}

.font-size-18 {
  font-size: 18px;
}
```

### for loop

The `@for` directive allows you to loop through some code for a set amount of iterations and has two forms:

- `@for <var> from <start> through <end> {}`
- `@for <var> from <start> to <end> {}`

The difference in the two forms is the *through* and the *to*; the *through* keyword will include the `<end>` in the loop where *to* will not; using *through* is the equivalent of using `>=` or `<=` in other languages, such as C++, JavaScript, or PHP.

**Notes**

- Both `<start>` and `<end>` must be integers or functions that return integers.
- When `<start>` is greater than `<end>` the counter will decrement instead of increment.

**SCSS Example**

```
@for $i from 1 through 3 {
  .foo-#{$i} { width: 10px * $i; }
}

// CSS output
.foo-1 { width: 10px; }
.foo-2 { width: 20px; }
.foo-3 { width: 30px; }
```

## Conditional directive (if)

The `@if` control directive evaluates a given expression and if it returns anything other than `false`, it processes its block of styles.

**Sass Example**

```
$test-variable: true !default

=test-mixin
  @if $test-variable
    display: block
  @else
    display: none

.test-selector
  +test-mixin
```

**SCSS Example**

```
$test-variable: true !default

@mixin test-mixin() {
  @if $test-variable {
    display: block;
  } @else {
    display: none;
  }
}

.test-selector {
  @include test-mixin();
}
```

The above examples produces the following CSS:

```
.test-selector {
  display: block;
}
```

### Each loop

The @each directive allows you to iterate through any list or map. It takes the form of `@each $var or <list or map> {}` where `$var` can be any variable name and `<list or map>` can be anything that returns a list or map.

In the following example, the loop will iterate through the `$authors` list assigning each item to `$author`, process its block of styles using that value of `$author`, and proceed to the next item in the list.

#### SCSS Example

```
$authors: "adam", "steve", "john";
@each $author in $authors {
    .photo-#{$author} {
      background: image-url("avatars/#{$author}.png") no-repeat
    }
}
```

#### CSS Output

```
.photo-adam {
  background: image-url("avatars/adam.png") no-repeat;
}
.photo-steve {
  background: image-url("avatars/steve.png") no-repeat;
}
.photo-john {
  background: image-url("avatars/john.png") no-repeat;
}
```

## Multiple Assignment

Multiple assignment allows you to gain easy access to all of the variables by declaring multiple variables in the `@each` directive.

#### Nested Lists

To have easy access to all the nested elements, you may declare separate variables to match each nested element. Be sure you have the correct amount of variables and nested elements. In the following example, an each loop is iterating through a list of three elements each of which contains three elements nested. Having the wrong amount of declared variables will result in a compiler error.

```
@each $animal, $color, $cursor in (puma, black, default),
                                  (sea-slug, blue, pointer),
                                  (egret, white, move) {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
    border: 2px solid $color;
    cursor: $cursor;
```

```
  }
}
```

**Maps**

Multiple assignment works for Maps as well but is limited to only two variables, a variable to access the key and a variable to access the value. The names `$key` and `$value` are arbitary in the following example:

```
@each $key, $value in ('first': 1, 'second': 2, 'third': 3) {
  .order-#{$key} {
    order: $value;
  }
}
```

## Each Loop with maps/ list values

In the below example value in map `$color-array` is treated as list of pairs.

SCSS Input

```
$color-array:(
  black: #4e4e4e,
  blue: #0099cc,
  green: #2ebc78
);
@each $color-name, $color-value in $color-array {
  .bg-#{$color-name} {
    background: $color-value;
  }
}
```

CSS Output

```
.bg-black {
  background: #4e4e4e;
}

.bg-blue {
  background: #0099cc;
}

.bg-green {
  background: #2ebc78;
}
```

Read Loops and Conditons online: https://riptutorial.com/sass/topic/2671/loops-and-conditons

# Chapter 8: Mixins

## Syntax

- `@mixin mixin-name ($argument1, $argument, ...){ ... }`

## Examples

### Create and use a mixin

To create a mixin use the `@mixin` directive.

```
@mixin default-box ($color, $borderColor) {
    color: $color;
    border: 1px solid $borderColor;
    clear: both;
    display: block;
    margin: 5px 0;
    padding: 5px 10px;
}
```

You can specify a list of arguments inside a parenthesis following the mixin's name. Remember to start your variables with `$` and separate them with commas.

To use the mixin in another selector, use the `@include` directive.

```
footer, header{ @include default-box (#ddd, #ccc); }
```

The styles from the mixin will now be used in the `footer` and `header`, with the value `#ccc` for the `$color` variable and `#ddd` for the `$borderColor` variable.

### Mixin with variable argument

There are some cases in mixins where there can be single or multiple arguments while using it. Let's take a case of `border-radius` where there can be single argument like `border-radius:4px;` or multiple arguments like `border-radius:4px 3px 2px 1px;`.

Traditional with Keyword Arguments mixing will be like below:-

```
@mixin radius($rad1, $rad2, $rad3, $rad4){
 -webkit-border-radius: $rad1 $rad2 $rad3 $rad4;
 -moz-border-radius: $rad1 $rad2 $rad3 $rad4;
 -ms-border-radius: $rad1 $rad2 $rad3 $rad4;
 -o-border-radius: $rad1 $rad2 $rad3 $rad4;
 border-radius: $rad1 $rad2 $rad3 $rad4;
}
```

And used as

```
.foo{
    @include radius(2px, 3px, 5px, 6px)
}
```

The above example is complex (to code, read and maintain) and if you can't pass only one value or two values, it will throw an error, and to use **one, two or there** values you have to define three other mixins.

By using **variable Argument** you don't have to worry about how many arguments can you pass. Variable arguments can be declared by defining a variable name followed by **three dots(...)**. Following is an example of a variable argument.

```
@mixin radius($radius...)
{
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    -o-border-radius: $radius;
    border-radius: $radius;
}
```

And used as

```
.foo{
    @include radius(2px 3px 5px 6px)
}
.foo2{
    @include radius(2px 3px)
}
.foo3{
    @include radius(2px)
}
```

The above example is much simpler (to code, maintain and read), you need not worry about how many arguments are about to come - is it **one or more than one**.

If there is more than one argument and in any case you want to access the second argument, you can do it by writing *`propertyname : nth(variable_name, 2)`*.

## Sensible defaults

SASS gives you the ability to omit any parameter except the ones you want to overwrite of course. Let's take again the `default-box` example:

```
@mixin default-box ($color: red, $borderColor: blue) {
    color: $color;
    border: 1px solid $borderColor;
    clear: both;
    display: block;
    margin: 5px 0;
    padding: 5px 10px;
}
```

Here we'll now call the mixin having overwritten the second parameter

```
footer, header{ @include default-box ($borderColor: #ccc); }
```

the value of `$borderColor` is `#ccc`, while `$color` stays `red`

## Optional arguments

SASS's optional arguments let you use a parameter only if you specify its value; otherwise, it will be ignored. Let's take an example of the following mixin:

```
@mixin galerie-thumbnail ($img-height:14em, $img-width: null) {
    width: $img-width;
    height: $img-height;
    outline: 1px solid lightgray;
    outline-offset: 5px;
}
```

So a call to

```
.default {
  @include galerie-thumbnail;
}
.with-width {
  @include galerie-thumbnail($img-width: 12em);
}
.without-height {
  @include galerie-thumbnail($img-height: null);
}
```

will simply output the following in the CSS file:

```
.default {
  height: 14em;
  outline: 1px solid lightgray;
  outline-offset: 5px;
}

.with-width {
  width: 12em;
  height: 14em;
  outline: 1px solid lightgray;
  outline-offset: 5px;
}

.without-height {
  outline: 1px solid lightgray;
  outline-offset: 5px;
}
```

SASS doesn't output properties with `null` as their value, which is very helpful when we need to include an optional argument in our call or not.

### @content directive

Mixins can be passed a block of SASS compliant code, which then becomes available within the mixin as the `@content` directive.

```
@mixin small-screen {
  @media screen and (min-width: 800px;) {
    @content;
  }
}

@include small-screen {
  .container {
    width: 600px;
  }
}
```

And this would output:

```
  @media screen and (min-width: 800px;) {
    .container {
      width: 600px;
    }
  }
```

Mixins can use the `@content` directive and still accept parameters.

```
@mixin small-screen($offset) {...
```

Read Mixins online: https://riptutorial.com/sass/topic/2131/mixins

# Chapter 9: Nesting

## Examples

### Basic nesting

Whenever you declare a new rule *inside* another rule it is called nesting. With basic nesting, as shown below, the nested selector will be compiled as a new CSS selector with all its parents prepended, separated by a space.

```
// SCSS
.parent {
  margin: 1rem;

  .child {
    float: left;
  }
}

// CSS output
.parent {
  margin: 1rem;
}

.parent .child {
  float: left;
}
```

### Nesting depth

Nesting is a very powerful feature, but should be used with caution. It can happen quite easily and quickly, that you start nesting and carry on including all children in a nest, of a nest, of a nest. Let me demonstrate:

```
// SCSS
header {
  // [css-rules]

  .holder {
    // [css-rules]

    .dropdown-list {
      // [css-rules]

      ul {
        // [css-rules]

        li {
          margin: 1rem 0 0 1rem;
        }
      }
    }
  }
```

```
}

// CSS output of the last rule
header .holder .dropdown-list ul li {
  margin: 1rem 0 0 1rem;
}
```

# Problems

## Specificity

The `li` from the example above has a `margin` set. Let's say we want to override this in a media-query later on.

```
@media (max-width: 480) {

  // will not work
  .dropdown-list ul li {
    margin: 0;
  }

  // will work
  header .holder .dropdown-list ul li {
    margin: 0;
  }
}
```

So by nesting too deep consequently you'll have to nest deep again whenever you want to overwrite a certain value. Even worse, this is often where the rule `!important` comes to use.

```
@media (max-width: 480) {

  // BIG NO-NO, don't do this
  .dropdown-list ul li {
    margin: 0 !important;
  }
```

*Why is the `!important`-rule is a bad idea*

You should write your SCSS in a good fashion that these workarounds aren't even necessary in the first place. Using `!important` on such a minor issue already will lead you down a rabbit hole!

## Reusability

This is fairly similar to the specificity problem, but worth pointing out separately. If you style something like a button or even a dropdown, you might want to reuse those styles somewhere else on your page.

By nesting too deeply your styles are only bound to the elements sitting inside the most outer parent (the element at the top of your SCSS). This leads you to copy styles and paste them somewhere else again. Possibly in an other nested rule.

Your stylesheets will become larger and larger and more difficult to maintain.

# How deep should you nest?

Most styleguides set the maximum depth to 2. This is good advice in general, as there are only very few occasions where you'd want to nest deeper. Most of the time, 2 is enough.

### Nesting with @at-root

Nesting is probably most often used to create more specific selectors, but it can also be used simply for code organization. Using the `@at-root` directive, you can 'jump out' of where you nest it in your Sass, bringing you back at the top level. Doing this allows you to keep styles grouped without creating more specificity than you need.

For example, you could to something like this :

```
.item {
    color: #333;

    @at-root {
        .item-wrapper {
            color: #666;

            img {
                width: 100%;
            }
        }
    }

    .item-child {
        background-color: #555;
    }
}
```

That would compile to this :

```
.item {
  color: #333;
}
.item-wrapper {
  color: #666;
}
.item-wrapper img {
  width: 100%;
}
.item .item-child {
  background-color: #555;
}
```

By doing this, all of our styles related to the `.item` class are together in the SCSS, but we don't necessarily need that class in every selector.

### Excluding contexts

By default declarations inside `@at-root` will appear in any context. This means that rules inside a `@media` block for instance will remain there.

```
@media print {
  .item-wrapper {
    @at-root {
      .item {
        background: white;
      }
    }
  }
}

// Will compile to
@media print {
  .item {
    background: white;
  }
}
```

This is not always desired behavior, so you can exclude the media context, by passing `media` to the the `without` option of the `@at-root` directive.

```
@at-root (without: media) {..
```

For more information, see the **official documentation**

## The parent selector (&)

Nesting is great for keeping related selectors together to make it easier for future developers to understand your code. The parent selector, represented by an ampersand ("&") can help do that in more complex situations. There are several ways its can be used.

Create a new selector that requires both the parent selector and another on the same element by placing the new selector directly after a parent selector.

```
// SCSS
.parent {

  &.skin {
    background: pink;
  }
}
```

```
// CSS output
.parent.skin {
  background: pink;
}
```

Have the parent appear after a nested selector in the compiled CSS by placing the parent selector *after* the nested selector.

```
// SCSS
.parent {

  .wrapper & {
    border: 1px solid black;
  }
}
```

```
// CSS output
.wrapper .parent {
  border: 1px solid black;
}
```

## States and pseudo-elements

Besides using nesting for classes and children, nesting with the parent selector is also commonly used to combine the states of `:active`, `:hover` and `:focus` for links.

```
// SCSS
a {
  color: blue;

  &:active,
  &:focus {
    color: red;
  }

  &:visited {
    color: purple;
  }
}
```

```
// CSS output
a {
  color: blue;
}

a:active,
a:focus {
  color: red;
}

a:visited {
  color: purple;
}
```

Similarly, you can style pseudo-elements by nesting with the parent selector.

```
// SCSS
.parent {

  &::after {
    display: table;
    clear: both;
    content: '';
```

```
  }

  &::only-child {
    font-weight: bold;
  }
}
```

```
// CSS output
.parent::after {
  display: table;
  clear: both;
  content: '';
}

.parent::only-child {
  font-weight: bold;
}
```

## Nesting properties

Some CSS properties belong to a namespace, for instance `border-right` belongs to the `border` namespace. To write less, we can utilize property nesting, and skip these prefixes, even on multiple levels.

If we needed to create a border on the right and left of a class named `.borders` we could write this:

```
//SCSS
.borders {
   border: 2px dashed blue;
   border: {
       left: 1px solid black;
       right: 1px solid red;
   }
 }

// CSS output
.borders {
   border: 2px dashed blue;
   border-left: 1px solid black;
   border-right: 1px solid red;
 }
```

This saves us having to write `border-right` and `border-left`, however we are still writing repetitive code with the lines `1px solid black` and `1px solid red`. We can write still less repetitive CSS with the folowing:

```
// SCSS
.borders {
  border: 2px dashed blue {
    left: 1px solid black;
    right: {
      color: red;
    }
  }
```

```
}

// CSS output
.borders {
   border: 2px dashed blue;
   border-left: 1px solid black;
   border-right-color: red;
}
```

Read Nesting online: https://riptutorial.com/sass/topic/2178/nesting

# Chapter 10: Operators

## Examples

### Assignment Operator

Sass uses the colon (`:`) operator to assign values to variables.

**Example**

```
$foreColor: red;

p {
    color: $foreColor;
}
```

### Arithmetic Operators

Sass supports the following standard arithmetic operators:

| Operator | Description |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| / | Division |
| % | Remainder |

**Examples**

```
p {
    font-size: 16px + 4px; // 20px
}
```

```
h3 {
    width: 2px * 5 + 12px; // 22px
}
```

```
h2 {
    width: 8px + (12px / 2) * 3; // 26px
}
```

Normal order of operations applies as usual.

## Comparison Operators

Sass supports all the usual comparison operators: `<`,`>`,`==`,`!=`,`<=`,`>=`.

### Examples

```
(10px == 10) // Returns true

("3" == 3) // Returns false
```

```
$padding: 10px;
$padding <= 8px; // Returns false
```

Read Operators online: https://riptutorial.com/sass/topic/3047/operators

# Chapter 11: Partials and Import

## Examples

### Importing

Using `@import` allows you to split up your files into multiple smaller files. This makes sense, as you are able to keep better structure for your stylesheets and avoid very large files.

### Example

Let's say you have a few files.

```
- application.scss
- header.scss
- content
  |-- article.scss
  '-- list.scss
- footer.scss
```

Your main stylesheet `application.scss` can import all files as well as define its own styles.

```
// application.scss
// Import files:
@import 'header.scss';
@import 'content/article.scss';
@import 'content/list.scss';
@import 'footer.scss';

// other styles in application.scss
body {
  margin: 0;
}
```

Note that you can also omit the `.scss` extension so you could write `@import 'header';` instead of `@import 'header.scss'`.

This leads to `application.scss` having all imported `.scss` included in the compiled file you serve to the client (browser). In this case your compiled file would be `application.css` which you include in your html.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="/application.css?v=18c9ed25ea60">
  </head>
  <body>
    ...
  </body>
</html>
```

Although you are working with multiple files you only serve one file, eliminating the need for multiple requests (one for each file) and speeding up the load time for your visitors.

# Main benefits

- Better structure for development using folder and multiple files
- Serving only one file to the client (browser)

### Partials

You can create partial files that contain smaller snippets of your stylesheets. This allows you to modularize your CSS and allows for better structure of your stylesheets. A partial is a Sass file named with a leading underscore, i.e: `_partial.scss`. The underscore lets Sass know that the specific file is a partial and it will not be generated into a CSS file.

Sass partials are meant to be used with the `@import` directive. When using `@import`, you can omit the leading underscore.

### Example

Supposing you have a file structure with partials like this

```
- main.scss
- _variables.scss
- content
  |-- _buttons.scss
  '-- _otherelements.scss
```

You can include those partials in your `main.scss` file as follows (leading underscores and file extensions are omitted in this example):

```
// main.scss - Imports:
@import 'variables';
@import 'content/buttons';
@import 'content/otherelements';
```

Read Partials and Import online: https://riptutorial.com/sass/topic/2893/partials-and-import

# Chapter 12: Scss useful mixins

## Examples

### Pure css3 pointer arrows with outline border

**!!! Container should be positioned relatively or absolutely**

**$direction** - top, bottom, left, right

**$margin** - margin by the edge in **$direction**. For top and bottom direction - it's from left to right. For left and right - it's from top to bottom.

**$colors** - first is a border color, second - is a background color (maybe it's better to inherit background color from a parent)

**$arrowSide** - is a relative size of an arrow

**$isInset** - arrow is inside (true) or outside of it's container

Here is a working Plunker https://plnkr.co/edit/PRF9eLwmOg8OcUoGb22Y?p=preview

```
%pointer-core {
    content: " ";
    position: absolute;
    border: solid transparent;
    z-index: 9999;
}

@mixin pointer($direction, $margin: 10px, $colors: (#999, $gray), $arrowSide: 8px, $isInset:
false){

    $opposites: (
        top: bottom,
        bottom: top,
        left: right,
        right: left
    );

    $margin-direction: (
        top: left,
        bottom: left,
        left: top,
        right: top
    );

    &:before {
        @extend %pointer-core;
        border-width: $arrowSide;

        @if $isInset {
            border-#{$direction}-color: nth($colors, 1);
            #{$direction}: -1px;
        }
```

```
        @else
        {
            border-#{map-get($opposites, $direction)}-color: nth($colors, 1);
            #{map-get($opposites, $direction)}: 100%;
        }

        #{map-get($margin-direction, $direction)}: 0;

        margin-#{map-get($margin-direction, $direction)}: $margin - 1;
    }

    &:after {
        @extend %pointer-core;
        border-width: $arrowSide - 1;

        @if $isInset {
            border-#{$direction}-color: nth($colors, 2);
            #{$direction}: -1px;
        }
        @else
        {
            border-#{map-get($opposites, $direction)}-color: nth($colors, 2);
            #{map-get($opposites, $direction)}: 100%;
        }

        #{map-get($margin-direction, $direction)}: 0;

        margin-#{map-get($margin-direction, $direction)}: $margin;
    }
}
```

## Tooltip pointer example

```
$color-purple-bg: #AF6EC4;
$color-purple-border: #5D0C66;


$color-yellow-bg: #E8CB48;
$color-yellow-border: #757526;

.tooltip {
    position: relative;

    &--arrow-down {
        @include pointer('bottom', 30px, ($color-purple-border, $color-purple-bg), 15px);
    }

    &--arrow-right {
        @include pointer('right', 60px, ($color-yellow-border, $color-yellow-bg), 15px);
    }
}
```

Read Scss useful mixins online: https://riptutorial.com/sass/topic/6605/scss-useful-mixins

# Chapter 13: SCSS vs Sass

## Examples

### Main Differences

Although people often say `Sass` as the name of this CSS-preprocessor, they often mean the `SCSS`-syntax. `Sass` uses the `.sass` file extension, while `SCSS`-`Sass` uses the `.scss` extension. They are both referred to as "Sass".

Speaking generally, the `SCSS`-syntax is more commonly used. `SCSS` looks like regular CSS with more capabilities, whereas `Sass` looks quite different to regular CSS. Both syntaxes have the same abilities.

# Syntax

The main differences are that `Sass` doesn't use curly brackets or semicolons, where `SCSS` does. `Sass` is also whitespace-sensitive, meaning you have to indent correctly. In `SCSS`, you can format and indent your rules as you please.

# SCSS:

```
// nesting in SCSS
.parent {
  margin-top: 1rem;

  .child {
    float: left;
    background: blue;
  }
}
```

# SASS:

```
// nesting in Sass
.parent
  margin-top: 1rem

  .child
    float: left
    background: blue
```

After compilation, both will produce the same following CSS:

```
.parent {
  margin-top: 1rem;
}
.parent .child {
  float: left;
  background: blue;
}
```

### Mixins

`Sass` tends to be the more "lazy" syntax. Nothing illustrates this nicer than how you define and include mixins.

## Defining a mixin

`=` is how you define a mixin in `Sass`, `@mixin` in `SCSS`.

```
// SCSS
@mixin size($x: 10rem, $y: 20rem) {
  width: $x;
  height: $y;
}

// Sass
=size($x: 10rem, $y: 20rem)
  width: $x
  height: $y
```

## Including a mixin

`+` is how you include in `Sass`, `@include` in `SCSS`.

```
// SCSS
.element {
  @include size(20rem);
}

// Sass
.element
  +size(20rem)
```

### Maps

When it comes to maps, usually `SCSS` is the easier syntax. Because `Sass` is indent-based, your maps have to be saved in one line.

```
// in Sass maps are "unreadable"
$white: ( white-50: rgba(255, 255, 255, .1), white-100: rgba(255, 255, 255, .2), white-200:
rgba(255, 255, 255, .3), white-300: rgba(255, 255, 255, .4), white-400: rgba(255, 255, 255,
.5), white-500: rgba(255, 255, 255, .6), white-600: rgba(255, 255, 255, .7), white-700:
```

```
rgba(255, 255, 255, .8), white-800: rgba(255, 255, 255, .9), white-900: rgba(255, 255, 255, 1
)
```

Because you can format your code on multiple lines with `scss`, you can format your maps to be more readable.

```
// in SCSS maps are more readable
$white: (
  white-50: rgba(255, 255, 255, .1),
  white-100: rgba(255, 255, 255, .2),
  white-200: rgba(255, 255, 255, .3),
  white-300: rgba(255, 255, 255, .4),
  white-400: rgba(255, 255, 255, .5),
  white-500: rgba(255, 255, 255, .6),
  white-600: rgba(255, 255, 255, .7),
  white-700: rgba(255, 255, 255, .8),
  white-800: rgba(255, 255, 255, .9),
  white-900: rgba(255, 255, 255, 1)
);
```

### Comments

Comments in Sass vs. Scss are largely similar, except when multi-lines are concerned. `sass` multi-lines are indentation-sensitive, while `scss` relies on comment terminators.

---

# Single-Line Comment

**style.scss**

```
// Just this line will be commented!
h1 { color: red; }
```

**style.sass**

```
// Exactly the same as the SCSS Syntax!
h1
  color: red
```

---

# Multi-Line Comment

**style.scss**

Initiator: `/*`

Terminator: `*/`

```
/* This comment takes up
```

```
 * two lines.
 */
h1 {
   color: red;
}
```

This will style `h1` elements with the color **red**.

**style.sass**

Now, `SASS` has *two* initiators, but no respective terminators. Multiline comments in `SASS` are sensitive to **indentation levels**.

Initiators: `//` and `/*`

```
// This is starts a comment,
   and will persist until you
   return to the original indentaton level.
h1
  color: red
```

This will style `h1` elements with the color **red**.

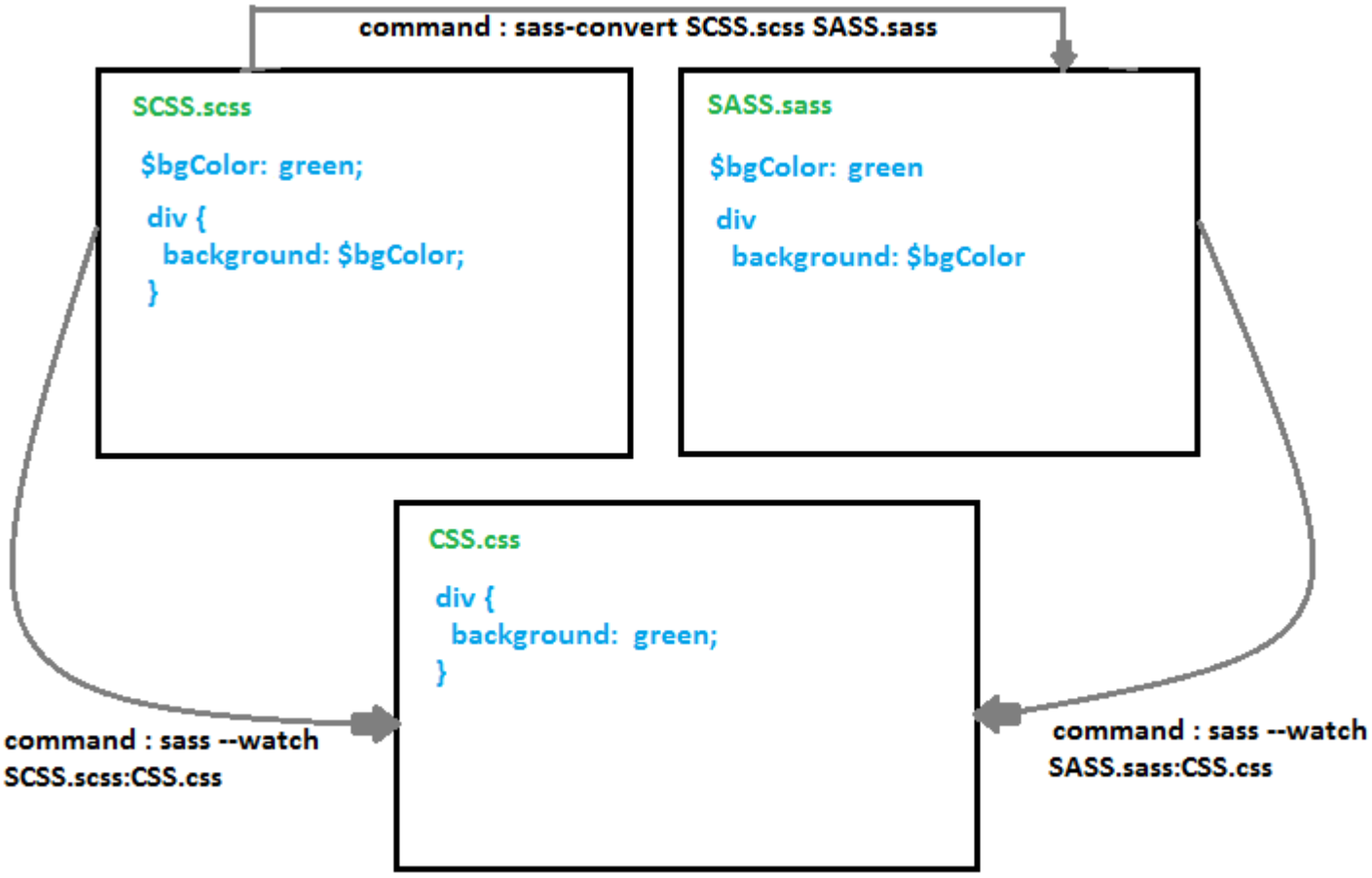
The same can be done with the `/*` Initiator:

```
/* This is starts a comment,
   and will persist until you
   return to the original indentaton level.
h1
  color: red
```

So there you have it! The main differences between comments in `SCSS` and `SASS`!

## Comparision between SCSS & SASS

- `SCSS` syntax resembles more like a `CSS` syntax but `SASS` syntax is little bit different from `SCSS` but both produces exactly the same `CSS` code.
- In `SASS` we are not ending the style properties with semicolon(`;`) but in SCSS we are ending the style properties with (`;`).
- In `SCSS` we used paranthesis `{}` to close the style properties but in `SASS` we don't use `paranthesis`.
- `Indentation` is very important in `SASS`. It will define the nested properties in the `class` or `id` of the element.
- In `scss` we can define multiple variables in single line but in `SASS` we can't do.




## for loop syntax

With the release of sass 3.3 and plus version the @if and else conditions syntax got same. we can now use expressions with not only **scss** but also **sass**.

### sass syntax

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

Compiled to

```
.item-1 {
  width: 2em;
}
.item-2 {
  width: 4em;
}
.item-3 {
  width: 6em;
}
```

### scss syntax

```
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

compiled to

```
.item-1 {
  width: 2em;
}
.item-2 {
  width: 4em;
}
.item-3 {
  width: 6em;
}
```

Read SCSS vs Sass online: https://riptutorial.com/sass/topic/2428/scss-vs-sass

# Chapter 14: Update Sass version

## Introduction

Update your Sass version using gem / ruby

## Examples

### Windows

You can check the version of Sass using `sass -v`

Update all ruby gems `gem update`

Update only Sass `gem update sass`

### Linux

You can check the version of Sass using `sass -v`

Update all ruby gems `sudo gem update`

Update only Sass `sudo gem update sass`

Read Update Sass version online: https://riptutorial.com/sass/topic/10599/update-sass-version

# Chapter 15: Variables

## Syntax

- $*variable_name*: *value*;

## Examples

### Sass

Variables are used to store a value once which will be used multiple times throughout a Sass document.

They are mostly used for controlling things such as fonts and colors but can be used for any value of any property.

Sass uses the `$` symbol to make something a variable.

```
$font-stack: Helvetica, sans-serif
$primary-color: #000000

body
  font-family: $font-stack
  color: $primary-color
```

### SCSS

Just as in Sass, SCSS variables are used to store a value which will be used multiple times throughout a SCSS document.

Variables are mostly used to store frequently-used property values (such as fonts and colors), but can be used for any value of any property.

SCSS uses the `$` symbol to declare a variable.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #000000;

body {
  font-family: $font-stack;
  color: $primary-color;
}
```

You can use `!default` when declaring a variable if you want to assign a new value to this variable only if it hasn't been assigned yet:

```
$primary-color: blue;
$primary-color: red !default; // $primary-color is still "blue"
```

```
$primary-color: green;        // And now it's green.
```

## Variable Scope

Variables exist within a specific scope, much like in in JavaScript.

If you declare a variable outside of a block, it can be used throughout the sheet.

```
$blue: dodgerblue;

.main {
    background: $blue;

    p {
        background: #ffffff;
        color: $blue;
    }
}

.header {
    color: $blue;
}
```

If you declare a variable within a block, it can only be used in that block.

```
.main {
    $blue: dodgerblue;

    background: $blue;

    p {
        background: #ffffff;
        color: $blue;
    }
}

.header {
    color: $blue; // throws a variable not defined error in SASS compiler
}
```

Variables declared at the sheet level (outside of a block) can also be used in other sheets if they are imported.

## Localize Variables with @at-root directive

@at-root directive can be used to localize variables.

```
$color: blue;

@at-root {
  $color: red;

  .a {
    color: $color;
```

```
  }
  .b {
    color: $color;
  }
}

.c {
  color: $color;
}
```

is compiled to:

```
.a {
  color: red;
}

.b {
  color: red;
}

.c {
  color: blue;
}
```

## Interpolation

Variables can be used in string interpolation. This allows you to dynamically generate selectors, properties and values. And the syntax for doing so a variable is `#{$variable}`.

```
$className: widget;
$content: 'a widget';
$prop: content;

.#{$className}-class {
  #{content}: 'This is #{$content}';
}
// Compiles to

.widget-class {
  content: "This is a widget";
}
```

You cannot, however use it to dynamically generate names of mixins or functions.

## Variables in SCSS

In SCSS variables begin with `$` sign, and are set like CSS properties.

```
$label-color: #eee;
```

They are only available within nested selectors where they’re defined.

```
#menu {
    $basic-color: #eee;
    color: $basic-color;
```

```
}
```

If they’re defined outside of any nested selectors, then they can be used everywhere.

```
$width: 5em;

#menu {
  width: $width;
}

#sidebar {
  width: $width;
}
```

They can also be defined with the `!global` flag, in which case they’re also available everywhere.

```
#menu {
  $width: 5em !global;
  width: $width;
}

#sidebar {
  width: $width;
}
```

It is important to note that variable names can use hyphens and underscores interchangeably. For example, if you define a variable called `$label-width`, you can access it as `$label_width`, and vice versa.

Read Variables online: https://riptutorial.com/sass/topic/2180/variables

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with sass | Angelos Chalaris, Benolot, Christopher, Community, Kartik Prasad, Rohit Jindal, SamJakob, Stewartside |
| 2 | Compass CSS3 Mixins | Schlumpf |
| 3 | Convert units | SuperDJ |
| 4 | Extend / Inheritance | Euan Williams, GMchris, user2367593 |
| 5 | Functions | Euan Williams, GMchris, Hudson Taylor, Pyloid |
| 6 | Installation | Angelos Chalaris, Pyloid, Stewartside |
| 7 | Loops and Conditons | Akash Kodesia, allejo, Angelos Chalaris, GMchris, MMachinegun, ScottL |
| 8 | Mixins | Akash Kodesia, Angelos Chalaris, GMchris, Hudson Taylor, Ninda, Roxy Walsh |
| 9 | Nesting | aisflat439, alexbea, Amy, Christopher, Devid Farinelli, GMchris, Hudson Taylor, John Slegers, MMachinegun |
| 10 | Operators | Angelos Chalaris, Hudson Taylor, Pyloid |
| 11 | Partials and Import | Angelos Chalaris, Hudson Taylor, MMachinegun |
| 12 | Scss useful mixins | Kindzoku |
| 13 | SCSS vs Sass | 75th Trombone, Everettss, Jared Hooper, MMachinegun, Muzamil301, Pyloid, Robotnicka, Rohit Jindal |
| 14 | Update Sass version | Schlumpf |
| 15 | Variables | Daniyal Basit Khan, evuez, GMchris, Hudson Taylor, jaredsk, Pyloid, Stewartside, yassh |