



- Introduction
- Preprocessing
- Variables
- Nesting
- Partials
- Import
- Mixins
- Inheritance
- Operators
- Functions
- Formatting

## Introduction

Sass (Syntactically Awesome Style Sheets) is a CSS preprocessor.

Writing a lot of CSS can be overwhelming that is why learning SASS and LESS can make any web developer and designer's life much easier.

There are two pre-processing methods: SASS and LESS.

## CSS Drawbacks

Using CSS only might work for you but when making big websites with multiple pages, there might be some features you wish CSS has.

Take a look at the following disadvantages of using CSS alone.

- No way to re-use common style rules.
- No way to specify variables that can be defined and re-used all through the style sheet.
- You can't execute computations where you can add numerical values to elements.

# Advantages of Using Pre-Processing Methods

While using CSS alone might give you nuisance, using pre-processing methods can save you a lot of time and effort.

Check out the list of advantages of using pre-processing methods below.

- Allows you to use variables that can be re-used all throughout the style sheet.
- Higher level style syntax that provides advanced CSS features.
- Compiled CSS files are uploaded to the production web server.

## What Is SASS?

**SASS** stands for Syntactically Awesome Style Sheets and was designed and created by **Hampton Catlin**.

SASS manipulates CSS using variables, mixins, and inheritance and nesting rules.

Given the extensions `.sass` and `.scss` respectively, it's translated to well-formatted CSS using a command line tool or web-framework plugin.

SASS makes it easier to write less CSS codes and manipulate them dynamically.

It's a great way to write more functional CSS codes and can speed up the workflow of every web developer and designer.

# .sass VS. .scss Format

Before we begin on how to use SASS, let's compare .sass and .scss extensions of SASS.

First I will provide a simple CSS code and then I will show you how to simplify them on both extensions of SASS.

## CSS Code

For our CSS, I used a header tag and put a zero value for margin and padding then white color for its text color.

```
header {  
    margin: 0;  
    padding: 0;  
    color: #fff;  
}
```

## .scss Extension Format (New Way of Writing SASS)

To format this into .scss extension format, we will use a variable *\$color* and give it a hexadecimal color value of *#fff* for white color. And then under the CSS style, instead of putting a hexadecimal color value of *#fff*, use the variable *\$color* that was set in the beginning of the code.

```
$color: #fff;  
header {  
    margin: 0;
```

```
padding:0;  
color: $color;  
}
```

## .sass Extension Format (Old Way of Writing SASS)

For our .sass extension, we will have the same variable and value just like the .scss extension format, but, this time, without semi-colons and brackets.

Notice that indentions are more reliant. This is the old format in writing SASS.

```
$color: #fff
header
  margin: 0
  padding: 0
  color: $color
```

## How does it work?

There are several ways you can compile Sass:

- The original Ruby Sass binary. Install it with `gem install sass`, and compile it by running `sassc myfile.scss myfile.css`.
- A GUI app such as [Hammer](#), [CodeKit](#), or [Compass](#)
- [libsass](#), which is a blazing fast Sass compiler written in C. You can also install libsass via NPM with [node-sass](#) (`npm install node-sass`).

## What's the deal with .sass vs .scss?

When Sass first came out, the main syntax was noticeably different from CSS.

It used indentation instead of braces, didn't require semi-colons and had shorthand operators. In short, it looked a lot like Haml.

Haml (HTML abstraction markup language) is based on one primary principle: *markup should be beautiful*.

**Haml** functions as a replacement for inline page templating systems such as PHP, ASP, and ERB, the templating language used in most Ruby on Rails applications.

Some folks didn't take too kindly to the new syntax, and in version 3 Sass changed its main syntax to .scss. SCSS is a superset of CSS, and is basically written the exact same, but with all the fun new Sass features.

That said, you can still use the original syntax if you want to.

I personally use .scss

## Set Up

If you want to try some of these concepts while following along, either:

Install your compilation method of choice, and create a `style.scss` file.

Or Follow along on [Sassmeister](#)

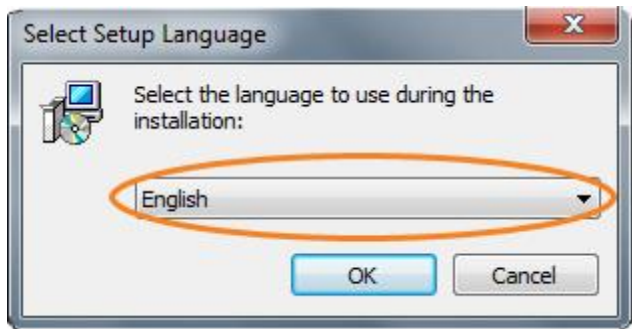
## Resources you need to understand SASS:

- [Ruby Installer](#)
- Text Editor

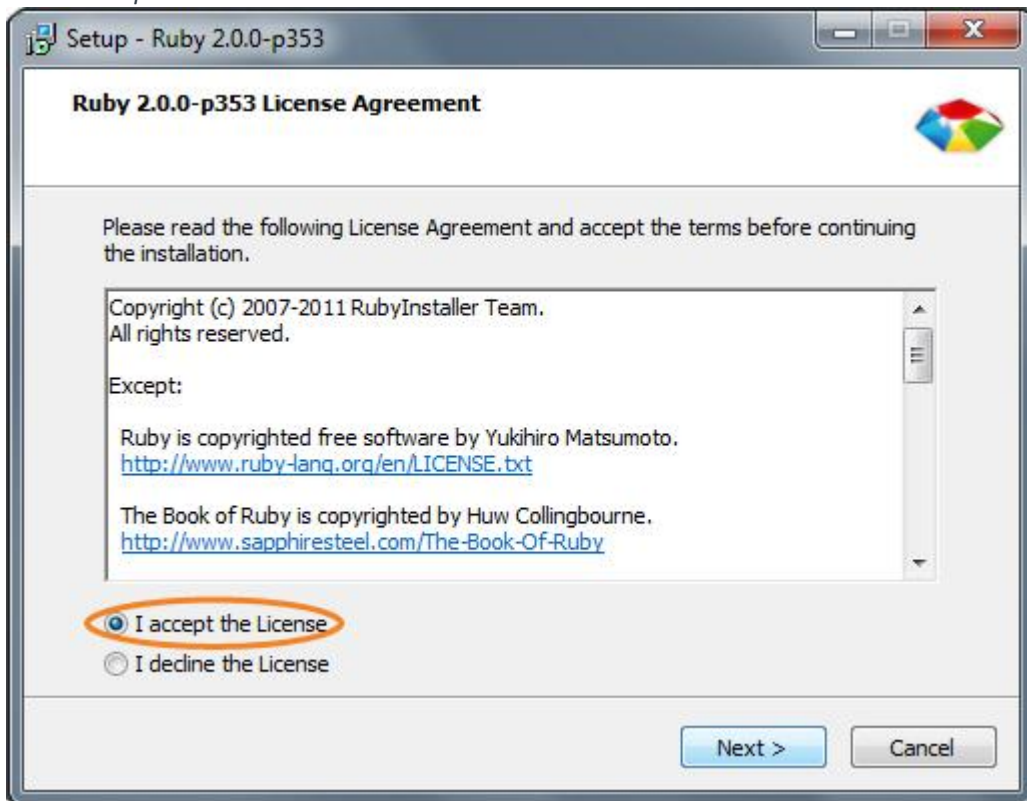


# Installing Ruby

Before you can be able to test how SASS works, you need to download **Ruby** to your computer. Launch the Ruby installer and you will be prompted with the Setup page. Select your preferred language and click OK.



Then click on *I accept the License* radio

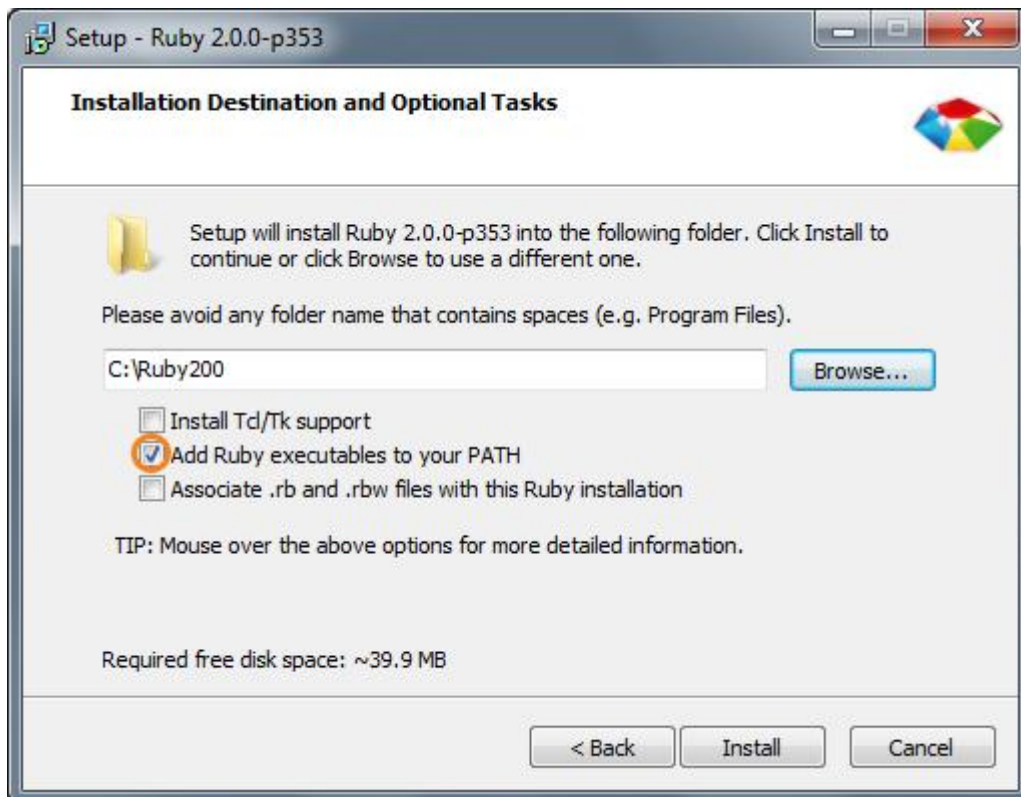


button.

Next, install it to your preferred location and make sure that the radio button for *Add Ruby executables to your PATH* is checked.

Click the *Install* button.

This will install the software and when it's done installing, just click the *Finish* button.



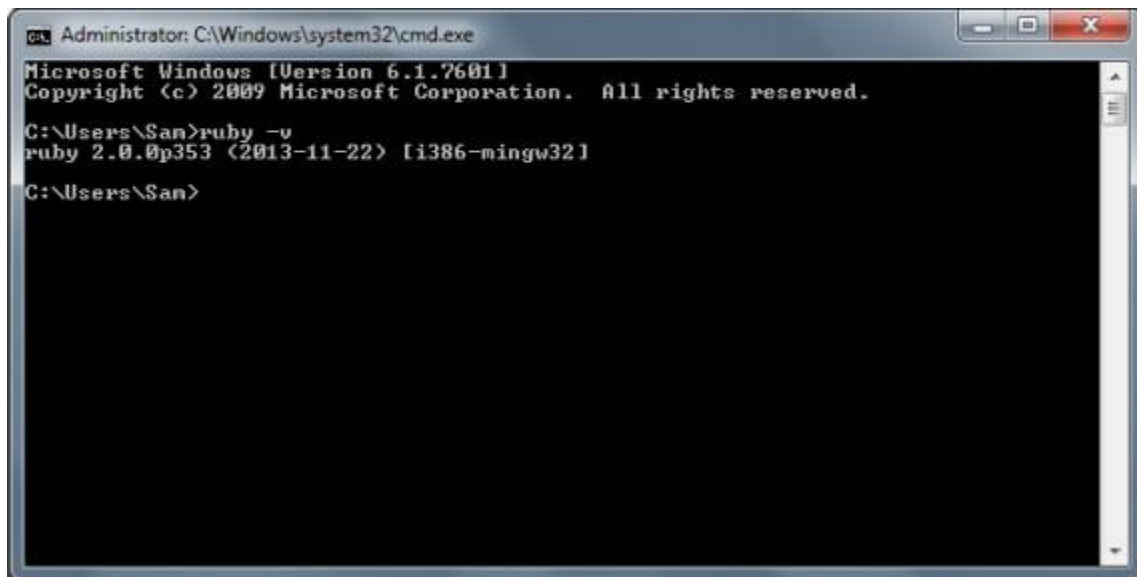
# Checking if Ruby is Running Properly

Now that you have installed Ruby, let's go ahead and check if this is working properly.

Open *Command Prompt* and type the word *ruby -v*.

And you can see, it would return the current version of the Ruby installed along with the date.

If it's returning an error, it could be that Ruby was not installed correctly or you did not put the Ruby executable into your path.

A screenshot of a Windows Command Prompt window. The title bar reads "Administrator: C:\Windows\system32\cmd.exe". The window content shows the following text:

```
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\San>ruby -v  
ruby 2.0.0p353 (2013-11-22) [i386-mingw32]  
  
C:\Users\San>
```

# Installing SASS

To install SASS, open *Command Prompt* and type the word ***gem install sass*** and you can see that the *Installation Prompt* that it was successfully installed.

A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The window shows the output of the command "gem install sass". The output text is: "Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Users\Sam>gem install sass Successfully installed sass-3.2.12 Parsing documentation for sass-3.2.12 1 gem installed C:\Users\Sam>".

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sam>gem install sass
Successfully installed sass-3.2.12
Parsing documentation for sass-3.2.12
1 gem installed

C:\Users\Sam>
```

## Preparing the Necessary Files

Before digging in with SASS, we need to prepare the necessary file you need before you code.

Create a new folder to your preferred location and name it SASS or any name you preferred.

Inside the SASS folder, create an HTML file and name it *index.html*.

For the HTML content put the following code.

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<title>Introduction to SASS</title>
```

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

```
</head>
```

```
<body>
```

```
<div id="container">
```

```
<header>
```

```
<h1>Sass Sample Document</h1>
```

```
<h2>A 1stwebdesigner tutorial</h2>
```

```
</header>
```

```
<div>
```

```
<p id="samplepara">Simple paragraph of text</p>
```

```
<p>Another paragraph of text</p>
```

```
</div>
```

```
<div>
```

```
<ul id="list1">
```

```
<li>List Item 1</li>
```

```
<li>List Item 2</li>
```

```
<li>List Item 3</li>
```

```
</ul>
```

```
</div>
```

```
<footer>
```

```
<h3>This is a cool footer content</h3>
```

```
</footer>
```

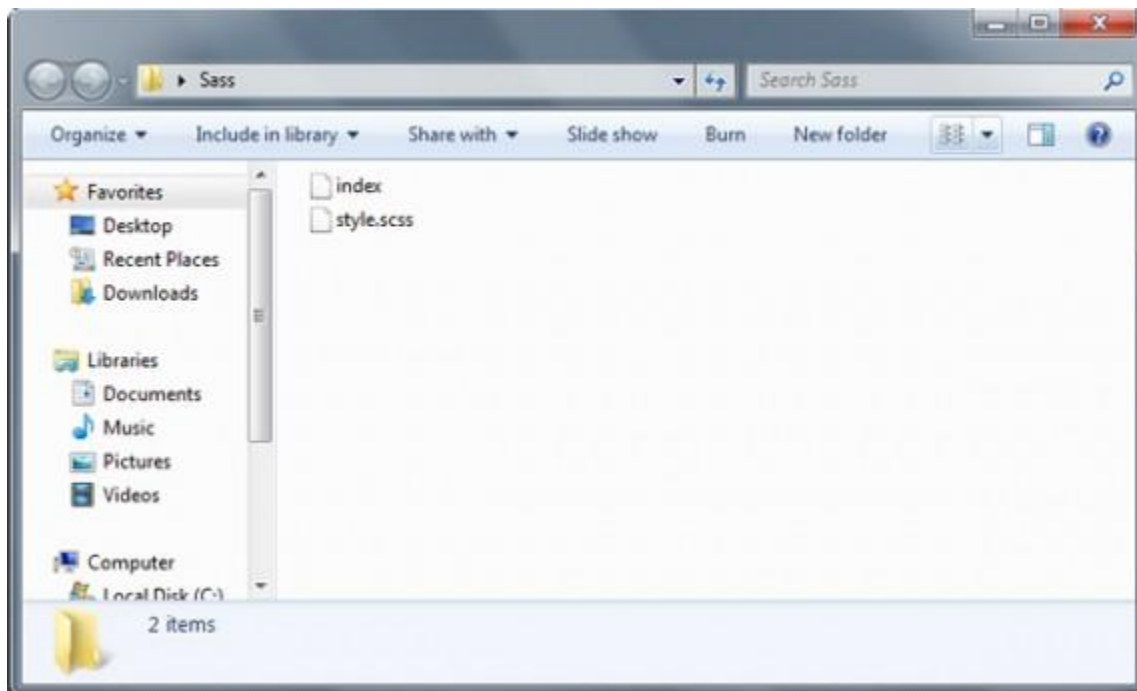
</div>

</body>

</html>

Now for our SASS file, create a new blank file in your text editor and name it *style.scss*.

If you followed the steps, by this time you will have the following file structure.



## Converting SASS code into CSS code

To convert the SASS code into CSS code, we're going to use the *-watch* command in command prompt.

This will compile the SASS codes to CSS.

Additionally, this will also watch the directories for changes or updates.

Let's try to convert the SASS file to CSS file.

Before we start, we need to put a sample code on our *style.scss* to see if this working.

Copy and paste the following sample SASS code on the *style.scss* file you created under SASS folder.

```
$myMargin: 0px auto;
$myColor: red;
$myWidth: 600px;

h1 {
  color: $myColor;
  $myMargin: $margin;
}
```

Next, open your command prompt and then go to the location where you put your files in.

In my case, I put it in my desktop so I will type in *cd "Desktop"* and it will locate the desktop directory.

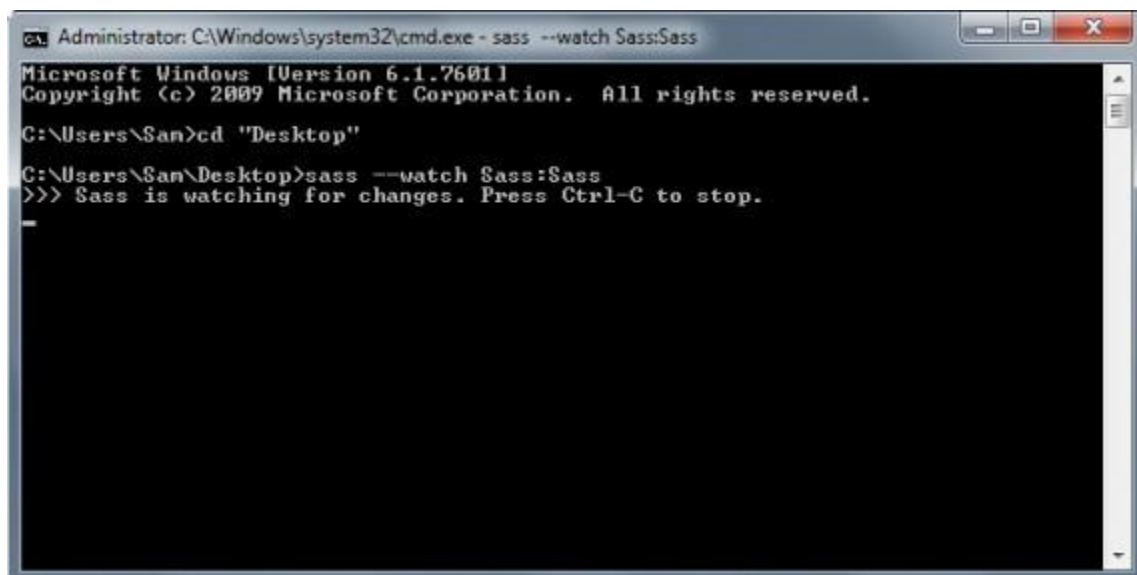




```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\San>cd "Desktop"
C:\Users\San\Desktop>
```

Now that we are in the desktop file directory, type in the `sass --watch Sass:Sass`



```
Administrator: C:\Windows\system32\cmd.exe - sass --watch Sass:Sass
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\San>cd "Desktop"
C:\Users\San\Desktop>sass --watch Sass:Sass
>>> Sass is watching for changes. Press Ctrl-C to stop.
_
```

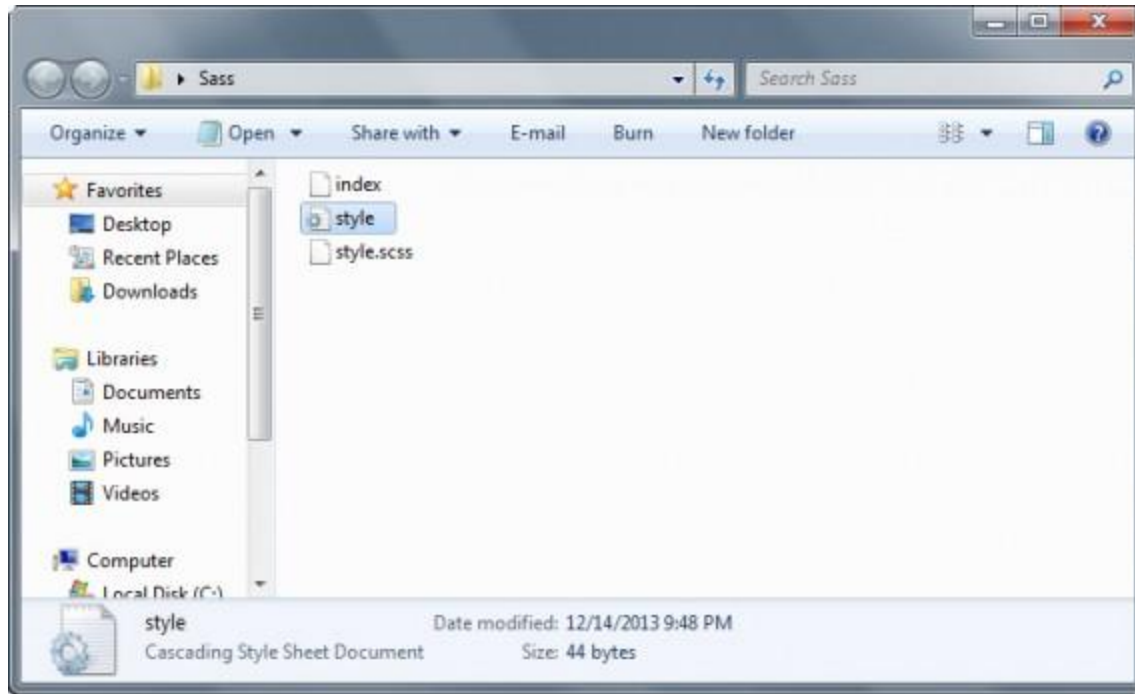
Using `--watch` command, we will convert all of the `.scss` files on the folder SASS.

It will also watch for the changes or updates on the file.

Notice that there are two SASS, divided by a colon.

The first one represents the current location of the .scss file while the second one represents the location of the output of the file.

Make sure you link the converted CSS file to your HTML file to see this working.



## . Preprocessing

CSS on its own can be fun, but stylesheets are getting larger, more complex, and harder to maintain.

This is where a preprocessor can help.

Sass lets you use features that don't exist in CSS yet like variables, nesting, mixins, inheritance and other nifty goodies that make writing CSS fun again.

Once you start tinkering with Sass, it will take your preprocessed Sass file and save it out as a normal CSS file that you can use in your web site.

# Variables

SASS variables are declared using the `$` character and are defined like CSS values.

Using SASS, you can declare variables for styles like font size, margin, padding and so on.

You can store things like colors, font stacks, or any CSS value you think you'll want to reuse.

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

Using variables and giving it a style value makes it easy to reuse a style repeatedly.

There are six different types of variables you can use with SASS.

1. **Strings** (e.g. `$myString: "your text here";`)
2. **Numbers** (e.g. `$myNum: 10px;`)
3. **Colors** (e.g. `$myColor: white;`)
4. **Booleans** (e.g. `$myBool: true;`)
5. **Lists** (e.g. `$myItemList: 1px solid red;`)
6. **Nulls** (e.g. `$myVar: null;`)

## Here's an example:

```
$myColor: #009a82;
```

```
$myString: " some text here ";
```

```
$myFontSize: 13px;
```

```
$myMargin: 0px auto;
```

```
$myWidth: 460px;
```

```
h1 {  
  color: $myColor;  
  margin: 0;  
  padding: 0;  
}
```

```
#container {  
  width: $myWidth;  
  margin: $myMargin;  
}
```

```
$font-stack: Helvetica, sans-serif;
```

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

```
body {
```

```
  font: 100% $font-stack;
```

```
  color: $primary-color;}
```

When the Sass is processed, it takes the variables we define for the `$font-stack` and `$primary-color` and outputs normal CSS with our variable values placed in the CSS.

This can be extremely powerful when working with brand colors and keeping them consistent throughout the site.

```
body {  
  
  font: 100% Helvetica, sans-serif;  
  
  color: #333;  
  
}
```

---

## Nesting

When writing HTML you've probably noticed that it has a clear nested and visual hierarchy. CSS, on the other hand, doesn't.

Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML.

Be aware that overly nested rules will result in over-qualified CSS that could prove hard to maintain and is generally considered bad practice.

With that in mind, here's an example of some typical styles for a site's navigation:

```
nav {
```

```
ul {  
  
  margin: 0;  
  
  padding: 0;  
  
  list-style: none;  
  
}  
  
li { display: inline-block; }  
  
  
  
  
  
  
  
  
  
a {  
  
  display: block;  
  
  padding: 6px 12px;  
  
  text-decoration: none;  
  
}  
  
}
```

You'll notice that the `ul`, `li`, and `a` selectors are nested inside the `nav` selector. This is a great way to organize your CSS and make it more readable. When you generate the CSS you'll get something like this:

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

```
nav li {  
  
    display: inline-block;  
  
}
```

```
nav a {  
  
    display: block;  
  
    padding: 6px 12px;  
  
    text-decoration: none;  
  
}
```

For the SASS version, you will have a format like this.

```
$myFontSize1: 13px;
$myFontSize2: 18px;
$myFontSize3: 25px;
$myWidth: 500px;
$myMargin: 0px auto;

#container {
  width: $myWidth;
  margin: $myMargin;

  p {
    font-family: Arial;
    font-size: $myFontSize1;
  }

  h1 {
    font-family: Tahoma;
    font-size: $myFontSize3;
  }

  h2 {

    font-family: Helvetica;
    font-size: $myFontSize2;
  }
}
```



# 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.

A partial is simply a Sass file named with a leading underscore.

You might name it something like `_partial.scss`.

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.

# Import

CSS has an import option that lets you split your CSS into smaller, more maintainable portions.

The only drawback is that each time you use `@import` in CSS it creates another HTTP request.

Sass builds on top of the current CSS `@import` but instead of requiring an HTTP request, Sass will take the file that you want to import and combine it with the file you're importing into so you can serve a single CSS file to the web browser.

Let's say you have a couple of Sass files, `_reset.scss` and `base.scss`.

We want to import `_reset.scss` into `base.scss`.

## SCSS SYNTAX

```
// _reset.scss
```

```
html,
```

```
body,
```

```
ul,
```

```
ol {
```

```
  margin: 0;
```

```
  padding: 0;
```

```
}
```

```
/* base.scss */
```

```
@import 'reset';
```

```
body {
```

```
  font: 100% Helvetica, sans-serif;
```

```
  background-color: #efefef;
```

```
}
```

Notice we're using `@import 'reset';` in the `base.scss` file.

When you import a file you don't need to include the file extension `.scss`.

Sass is smart and will figure it out for you. When you generate the CSS you'll get:

```
html, body, ul, ol {  
  
  margin: 0;  
  
  padding: 0;  
  
}  
  
body {  
  
  font: 100% Helvetica, sans-serif;  
  
  background-color: #efefef;  
  
}
```

# Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist.

Mixins let you define common properties once then re-use them over and over again.

Mixins are defined using *@mixin* directive and contains a block of codes and then reuse them using *@include* directive

A mixin lets you make groups of CSS declarations that you want to reuse throughout your site.

You can even pass in values to make your mixin more flexible.

A good use of a mixin is for vendor prefixes.

Here's an example for `border-radius` .

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

```
.box { @include border-radius(10px); }
```

To create a mixin you use the `@mixin` directive and give it a name.

We've named our mixin `border-radius`.

We're also using the variable `$radius` inside the parentheses so we can pass in a radius of whatever we want.

After you create your mixin, you can then use it as a CSS declaration starting with `@include` followed by the name of the mixin.

When your CSS is generated it'll look like this:

```
.box {  
  
-webkit-border-radius: 10px;  
  
-moz-border-radius: 10px;  
  
-ms-border-radius: 10px;  
  
border-radius: 10px;  
  
}
```

Let's put this into practice. Copy the code below to your *style.scss* file.

```
@mixin border {  
  border: 1px solid red;  
}
```

```
#container {  
  width: 960px;  
  margin: 0 auto;  
  @include border;  
}
```

## Extend/Inheritance

This is one of the most useful features of Sass.

Using `@extend` lets you share a set of CSS properties from one selector to another.

It helps keep your Sass very DRY.

In our example we're going to create a simple series of messaging for errors, warnings and successes.

```
.message {
```

```
  border: 1px solid #ccc;
```

```
  padding: 10px;
```

```
  color: #333;
```

```
}
```

```
.success {
```

```
  @extend .message;
```

```
  border-color: green;
```

```
}
```

```
.error {
```

```
  @extend .message;
```

```
  border-color: red;
```

```
}
```

```
.warning {  
  
  @extend .message;  
  
  border-color: yellow;  
  
}
```

What the above code does is allow you to take the CSS properties in `.message` and apply them to `.success`, `.error`, & `.warning`.

The magic happens with the generated CSS, and this helps you avoid having to write multiple class names on HTML elements.

This is what it looks like:

```
.message, .success, .error, .warning {  
  
  border: 1px solid #cccccc;  
  
  padding: 10px;  
  
  color: #333;  
  
}  
  
.success {
```



```
border-color: green;
```

```
}
```

```
.error {
```

```
border-color: red;
```

```
}
```

```
.warning {
```

```
border-color: yellow;
```

```
}
```

# Operators

Doing math in your CSS is very helpful.

Sass has a handful of standard math operators like `+`, `-`, `*`, `/`, and `%`.

In our example we're going to do some simple math to calculate widths for an `aside` & `article`.

```
.container { width: 100%; }

article[role="main"] {

  float: left;

  width: 600px / 960px * 100%;

}

aside[role="complimentary"] {

  float: right;

  width: 300px / 960px * 100%;

}
```

We've created a very simple fluid grid, based on 960px.

Operations in Sass let us do something like take pixel values and convert them to percentages without much hassle.

The generated CSS will look like:

```
.container {  
  
  width: 100%;  
  
}  
  
article[role="main"] {  
  
  float: left;  
  
  width: 62.5%;  
  
}  
  
aside[role="complimentary"] {  
  
  float: right;  
  
  width: 31.25%;  
  
}
```

## Function Directives

Function directives in Sass are similar to mixins, but instead of returning markup, they return values via the `@return` directive.

They can be used to DRY (Don't repeat yourself) up your code, and make everything more readable.

```
.scss
```

```
$settings: (  
  maxWidth: 800px,  
  columns: 12,  
  margin: 15px,  
  breakpoints: (  
    xs: "(max-width : 480px)",  
    sm: "(max-width : 768px) and (min-width: 481px)",  
    md: "(max-width : 1024px) and (min-width: 769px)",  
    lg: "(min-width : 1025px)"  
  )  
);
```

```
@mixin renderGridStyles($settings){  
  .container {  
    padding-right: map-get($settings, "margin");  
    padding-left: map-get($settings, "margin");  
    margin-right: auto;  
    margin-left: auto;  
    max-width: map-get($settings, "maxWidth");  
  }  
  
  .row {  
    margin-right: map-get($settings, "margin") * -1;  
    margin-left: map-get($settings, "margin") * -1;  
  }  
  
  $breakpoints: map-get($settings, "breakpoints");  
  @each $key, $breakpoint in $breakpoints {  
    @include media($breakpoint) {  
      @include renderGrid($key, $settings);  
    }  
  }  
}
```

```
@mixin renderGrid($key, $settings) {  
  
  $i: 1;  
  
  @while $i <= map-get($settings, "columns") {  
  
    .col-#{ $key }-#{ $i } {  
  
      float: left;  
  
      width: 100% * $i / map-get($settings, "columns");  
  
    }  
  
    $i: $i+1;  
  
  }  
  
}
```

```
@mixin media($queryString){  
  
  @media #{ $queryString } {  
  
    @content;  
  
  }  
  
}
```

```
@include renderGridStyles($settings);
```

```
p {  
  padding: 20px;  
  color: white;  
  background: #9b59b6;  
  margin: 20px;  
}
```

Css

```
.container {  
  padding-right: 15px;  
  padding-left: 15px;  
  margin-right: auto;  
  margin-left: auto;  
  max-width: 800px;  
}
```

```
.row {  
  margin-right: -15px;  
  margin-left: -15px;
```

```
}
```

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

```
  .col-xs-1 {
```

```
    float: left;
```

```
    width: 8.33333%;
```

```
  }
```

```
  .col-xs-2 {
```

```
    float: left;
```

```
    width: 16.66667%;
```

```
  }
```

```
  .col-xs-3 {
```

```
    float: left;
```

```
    width: 25%;
```

```
  }
```

```
  .col-xs-4 {
```

```
    float: left;
```



```
width: 33.33333%;  
}
```

```
.col-xs-5 {  
  float: left;  
  width: 41.66667%;  
}
```

```
.col-xs-6 {  
  float: left;  
  width: 50%;  
}
```

```
.col-xs-7 {  
  float: left;  
  width: 58.33333%;  
}
```

```
.col-xs-8 {  
  float: left;
```

```
width: 66.66667%;  
}
```

```
.col-xs-9 {  
  float: left;  
  width: 75%;  
}
```

```
.col-xs-10 {  
  float: left;  
  width: 83.33333%;  
}
```

```
.col-xs-11 {  
  float: left;  
  width: 91.66667%;  
}
```

```
.col-xs-12 {  
  float: left;
```

```
width: 100%;  
  
}  
  
}  
  
@media (max-width: 768px) and (min-width: 481px) {  
  
  .col-sm-1 {  
  
    float: left;  
  
    width: 8.33333%;  
  
  }  
  
  
  .col-sm-2 {  
  
    float: left;  
  
    width: 16.66667%;  
  
  }  
  
  
  .col-sm-3 {  
  
    float: left;  
  
    width: 25%;  
  
  }  
  
  
  .col-sm-4 {
```

```
float: left;

width: 33.33333%;

}
```

```
.col-sm-5 {

float: left;

width: 41.66667%;

}
```

```
.col-sm-6 {

float: left;

width: 50%;

}
```

```
.col-sm-7 {

float: left;

width: 58.33333%;

}
```

```
.col-sm-8 {
```

```
float: left;  
width: 66.66667%;  
}
```

```
.col-sm-9 {  
float: left;  
width: 75%;  
}
```

```
.col-sm-10 {  
float: left;  
width: 83.33333%;  
}
```

```
.col-sm-11 {  
float: left;  
width: 91.66667%;  
}
```

```
.col-sm-12 {
```

```
float: left;

width: 100%;

}

}

@media (max-width: 1024px) and (min-width: 769px) {

.col-md-1 {

float: left;

width: 8.33333%;

}

.col-md-2 {

float: left;

width: 16.66667%;

}

.col-md-3 {

float: left;

width: 25%;

}
```

```
.col-md-4 {  
    float: left;  
    width: 33.33333%;  
}
```

```
.col-md-5 {  
    float: left;  
    width: 41.66667%;  
}
```

```
.col-md-6 {  
    float: left;  
    width: 50%;  
}
```

```
.col-md-7 {  
    float: left;  
    width: 58.33333%;  
}
```

```
.col-md-8 {  
    float: left;  
    width: 66.66667%;  
}
```

```
.col-md-9 {  
    float: left;  
    width: 75%;  
}
```

```
.col-md-10 {  
    float: left;  
    width: 83.33333%;  
}
```

```
.col-md-11 {  
    float: left;  
    width: 91.66667%;  
}
```



```
.col-md-12 {  
    float: left;  
    width: 100%;  
}  
  
@media (min-width: 1025px) {  
    .col-lg-1 {  
        float: left;  
        width: 8.33333%;  
    }  
  
    .col-lg-2 {  
        float: left;  
        width: 16.66667%;  
    }  
  
    .col-lg-3 {  
        float: left;  
        width: 25%;  
    }  
}
```

```
.col-lg-4 {  
    float: left;  
    width: 33.33333%;  
}
```

```
.col-lg-5 {  
    float: left;  
    width: 41.66667%;  
}
```

```
.col-lg-6 {  
    float: left;  
    width: 50%;  
}
```

```
.col-lg-7 {  
    float: left;  
    width: 58.33333%;  
}
```

```
.col-lg-8 {  
    float: left;  
    width: 66.66667%;  
}
```

```
.col-lg-9 {  
    float: left;  
    width: 75%;  
}
```

```
.col-lg-10 {  
    float: left;  
    width: 83.33333%;  
}
```

```
.col-lg-11 {  
    float: left;  
    width: 91.66667%;  
}
```

```
.col-lg-12 {  
    float: left;  
    width: 100%;  
}  
  
p {  
    padding: 20px;  
    color: white;  
    background: #9b59b6;  
    margin: 20px;  
}
```

HTML

```
<div class="container">  
  <div class="row">  
    <div class="col-sm-12 col-md-6 col-lg-3"><p>1</p></div>  
    <div class="col-sm-12 col-md-6 col-lg-3"><p>1</p></div>  
    <div class="col-sm-12 col-md-6 col-lg-3"><p>1</p></div>  
    <div class="col-sm-12 col-md-6 col-lg-3"><p>1</p></div>
```

</div>

</div>

# SASS Output Formatting

One of the best features of SASS is it offers several options to control how the `.scss` code is formatted when compiled into CSS.

Using the `-style` option command, we can perform formatting to our compile CSS code.

The following are the SASS formatting styles.

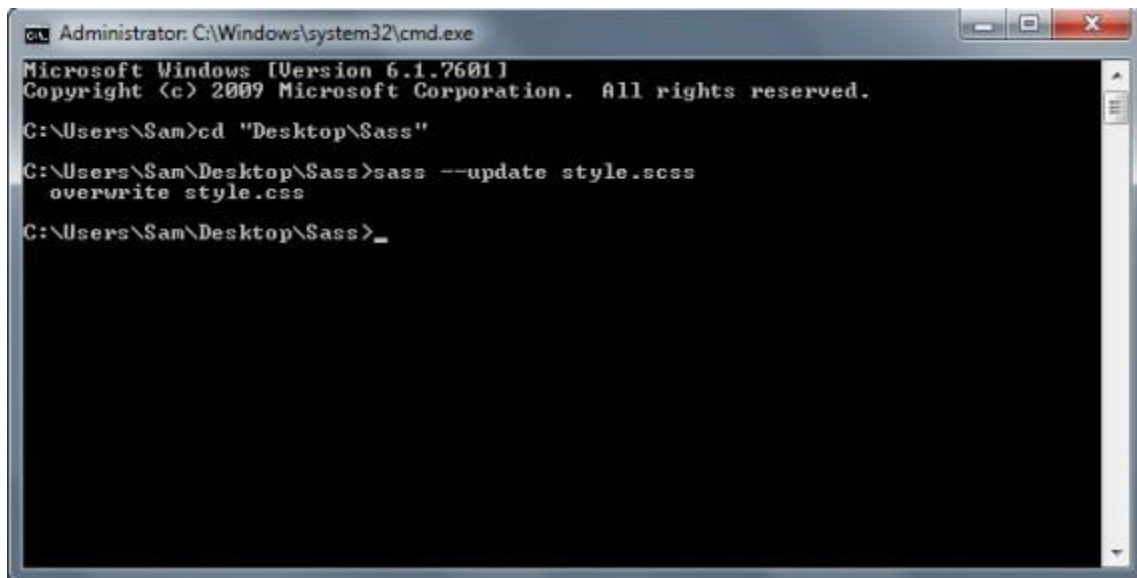
## Nested Format

Nested style is the default format of SASS.

This format gives indention to all of the styles in your compiled CSS file.

To see this in action, copy and paste the code below into your `style.scss` and then open your *Command Prompt* (make sure you are inside the directory of SASS) and type in `SASS -update style.scss`.

This command will update the formatting for the compiled CSS file using the update command.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sam>cd "Desktop\Sass"
C:\Users\Sam\Desktop\Sass>sass --update style.scss
      overwrite style.css
C:\Users\Sam\Desktop\Sass>_
```

Now go ahead and open your compiled style.css file.

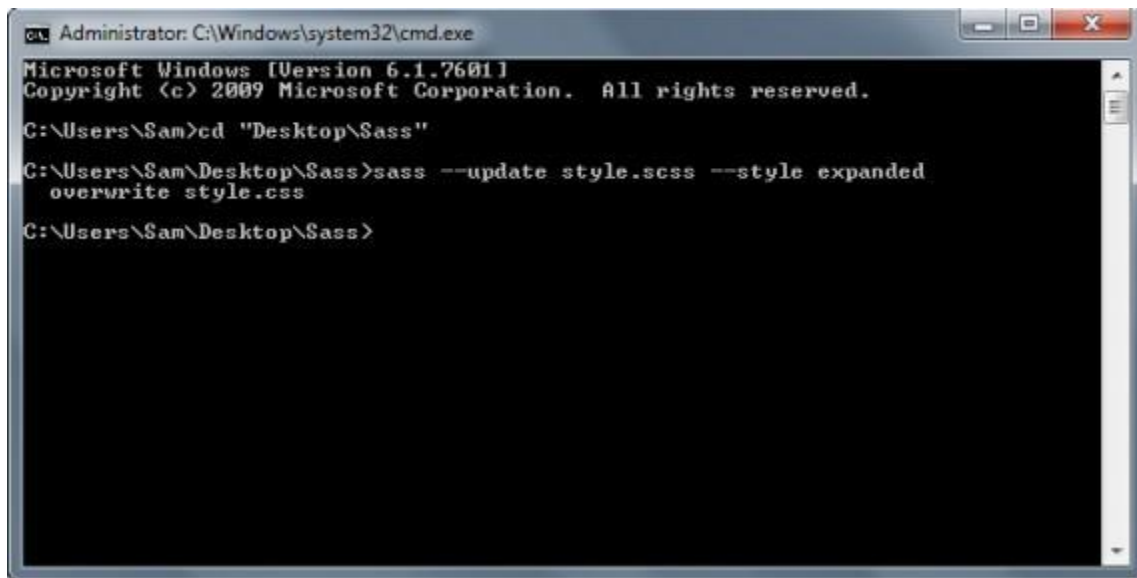
As I've said above all of the styles will have proper indentation.



```
1
2 #samplepara {
3     color: #aa30ff;
4 }
5
6 #list1 {
7     color: #e33939;
8 }
9
```

## Expanded Format

This is the most user-friendly and readable format as the braces are properly expanded and each property will have its own line. Let's see this in action. Using the same code above, open your command prompt and type `sass -update style.scss -style expanded`. Notice that we added `-style` command this is used to format compiled CSS file to a particular formatting.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sam>cd "Desktop\Sass"
C:\Users\Sam\Desktop\Sass>sass --update style.scss --style expanded
                        overwrite style.css
C:\Users\Sam\Desktop\Sass>
```

So if you are going to look at the compiled CSS file on the SASS file, you can see the code formatting is similar to the image below. Notice that each property has its own line. The braces are also fully expanded.



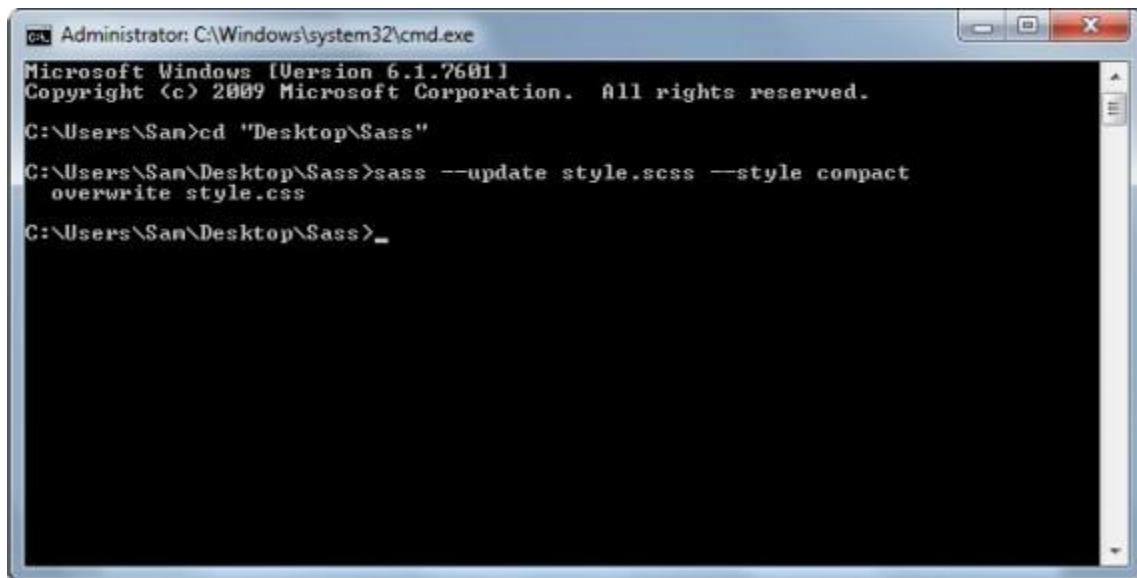
```
1
2  #samplepara {
3      color: #aa30ff;
4  }
5
6  #list1 {
7      color: #e33939;
8  }
9
```

## Compact Format

This is the compact format output CSS code in a condensed but still readable format.

It adds spaces between braces but all in one line. To see how this works using the same code above, open your command prompt and type *sass --update style.scss --style compact*.

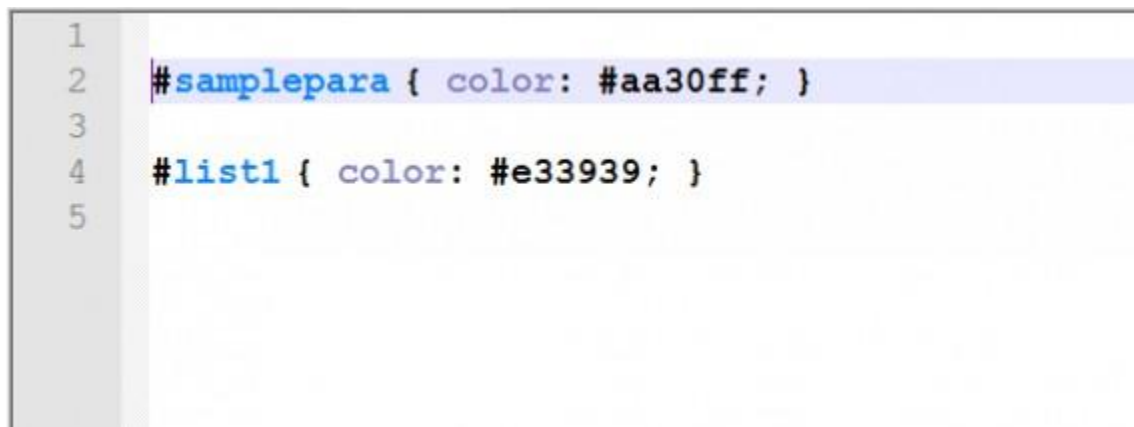




```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\San>cd "Desktop\Sass"
C:\Users\San\Desktop\Sass>sass --update style.scss --style compact
overwrite style.css
C:\Users\San\Desktop\Sass>_
```

If you are going to check the compiled CSS file, you can see something similar to the image below. It is condensed. Each property and style is in one line.



```
1
2 #samplepara { color: #aa30ff; }
3
4 #list1 { color: #e33939; }
5
```

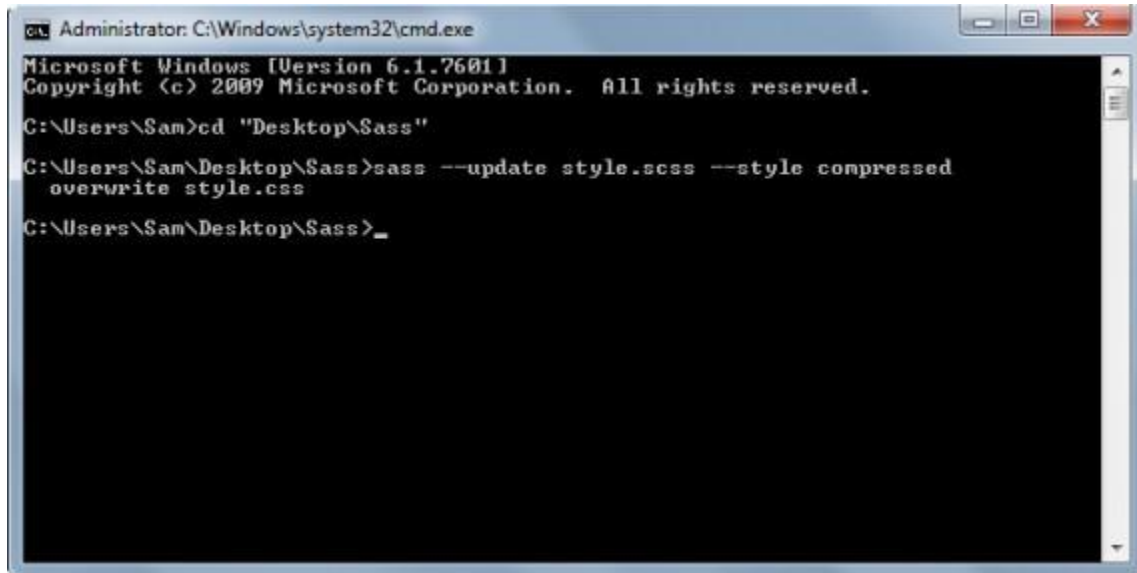
## Compressed Format

The compressed format has minimized output.

This is suitable for the production environment.

This format has more condensed formatting.

Using the same code, open your command prompt and type in `sass --update style.scss --style compressed`.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sam>cd "Desktop\Sass"

C:\Users\Sam\Desktop\Sass>sass --update style.scss --style compressed
overwrite style.css

C:\Users\Sam\Desktop\Sass>_
```

So if you are going to look at the compiled CSS file on the SASS file, you can see code formatting similar to the image below.

As you can see, all codes are in one line with no spaces at all.



```
1
2 #samplepara{color:#aa30ff}#list1{color:#e33939}
3
```