

Swipe Left

# SASS : Ultimate Beginner's Guide

Part - 01



Ariba M.  
@frontendcharm

Sass



## First of all, what is SASS?

SASS is a very popular CSS Preprocessor. The acronym stands for Syntactically Awesome Style Sheets.

It is the most mature, stable, and powerful professional grade CSS extension language in the world.

Overall, it is basically CSS with extended capabilities. In fact, if you already know CSS, learning SASS should be shockingly easy to pick up.



Ariba M.  
@frontendcharm

Keep Swiping...

When we link CSS stylesheets to our HTML, we always do something like:

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

When we use SASS, we cannot simply write `href="style.scss"`. We cannot directly link SASS files.

This isn't possible because SASS is a scripting language. It is not CSS. With the help of extensions or NodeJS modules, the SASS files can be transformed into usable CSS. And we can link those generated CSS files to our HTML.

Why not just use CSS then? What's the point? SASS provides additional functionality that enhances what CSS can achieve. And you'll see that very soon.



Ariba M.  
@frontendcharm

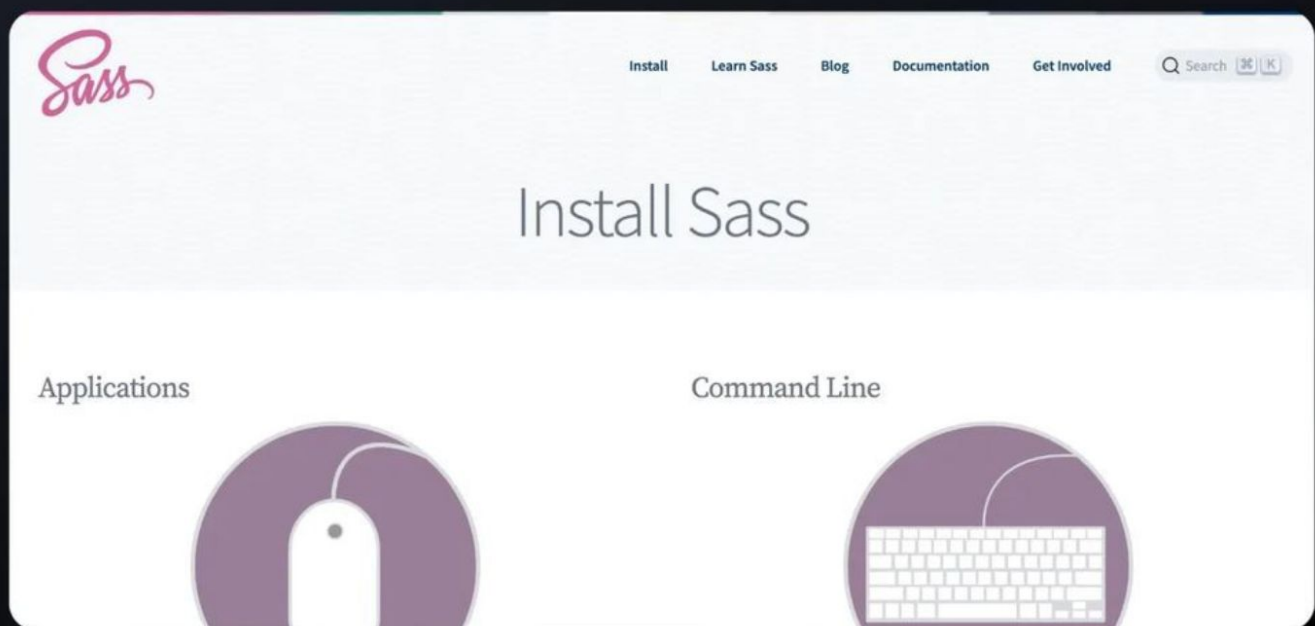
Keep Swiping...

# Installation :

So, with all that said, we have to install a tool that will make our SASS files usable. There are several ways to set up Sass on your computer.

1. Install anywhere ( Standalone )
2. Install using NPM
3. Install using Homebrew ( Mac OSX )

You can read more about them at the official Sass web site.



[sass-lang.com/install](https://sass-lang.com/install)





Ariba M.  
@frontendcharm

Keep Swiping...





## Extensions :

First, we're going to open Visual Studio Code. VS Code is probably the most popular text editor out right now, and you will find many tutorials using this beautiful piece of software.

A major reason is there are many useful extensions available on this text editor that makes coding easier.

Once VS Code is opened, you want to look at the side panel to the left and click the extensions tab. In the search bar, search for "Live Sass Compiler" and "Sass"



	<b>Sass</b> Indented Sass syntax Highlighting, Aut... Syler 
	<b>Live Sass Compiler</b> Compile Sass or Scss to CSS at realtim... Ritwick Dey 

After installing the extension, we are ready to go.



Ariba M.  
@frontendcharm

Keep Swiping...

## Create SASS file

SASS files can end with `.sass` or `.scss`. In this guide, we're singularly working with `.scss`. And yes, there is a difference. The syntax is different.

*When you create a `style.scss` file :*

Two files will be created: a `css` file and a `css.map` file.  
Don't touch either of them

Every time you write new stylings in your `scss` file and hit save, it will **convert those sass instructions into css**. And that conversion will be saved in the `css` file. **The CSS file will constantly be overwritten**. It is not adding to the CSS file. It is overwriting it every time.



## Variables :

In SASS, we can create variables. These variables can be strings, colors, numbers, booleans, lists, and nulls.

To create variables, we simply write a dollar-sign followed by the variable name. And then we can assign these variables a value as shown:

```
1 $primary-color: #333;  
2  
3 p {  
4   color: $primary-color;  
5 }
```

And then to use these variables, we simply just call upon them.





## Nesting :

In plain CSS, we have CSS combinators. Using descendant selectors (as shown below), child selectors, and adjacent selectors, the CSS sample shown here is completely valid.

```
1 nav ul {
2   margin: 0;
3   padding: 0;
4   list-style: none;
5 }
6
7 nav li {
8   display: inline-block;
9 }
10
11 nav a {
12   display: block;
13   padding: 6px 12px;
14   text-decoration: none;
15 }
```



However, the problem with this is it can quickly become chaotic and unorganized. This is true for projects of all sizes.

In SASS, we can tame the chaos with nesting. It's fairly straight forward. We are simply nesting descendant selectors within parent selectors.

```
1 nav {
2   ul {
3     margin: 0;
4     padding: 0;
5     list-style: none;
6   }
7
8   li { display: inline-block; }
9
10  a {
11    display: block;
12    padding: 6px 12px;
13    text-decoration: none;
14  }
15 }
```





Ariba M.  
@frontendcharm

Keep Swiping...

## Mixins :

Moving on, we have **Mixins**. Mixins are very cool. Mixins are like variables, but instead of holding values, **they can hold CSS instructions**.



```
1 @mixin basetext {  
2   font-size: 1em;  
3   line-height: 1.2;  
4   color: #fff;  
5 }  
6  
7 .p {  
8   @include basetext;  
9 }
```

To create a mixin, start with **@mixin** followed by the **mixin's name**. (The name is just how you will reference the mixin.)

Then within **curly braces {}**, you define the CSS instructions.

And when you actually want to use these instructions( aka the **mixin**), include it as shown. Start with **@include** followed by the mixin's name.

So now as per the example, the **p** selector will adopt all the CSS stylings offered by the mixin. "**basetext**"



## Extend :

Another way to avoid writing the same code again and again is SASS's `@extend`. It's basically the same thing as mixin. Using `@extend` lets you share a set of CSS properties from one selector to another.

Let's take an example...So what the code does here is it creates a basic style for buttons ( this style will be used for most buttons ).

Then, we create one style for a "login" button and one style for a "register" button.

Both "login" and "register" button inherit all the CSS properties from the `.button-base` class, through the `@extend` directive.

And if you want overwrite any stylings just write it below the `@extend`.

```
style.scss

1  .button-base {
2    padding: 10px;
3    font-size: 1em;
4    color: #fff;
5  }
6
7  .login {
8    @extend .button-base;
9    background: #151515;
10 }
11
12 .register {
13   @extend .button-base;
14   background: #2101a1;
15 }
```



Ariba M.  
@frontendcharm

Keep Swiping...

# Modules :

You don't have to write all your Sass in a single file. You can split it up however you want with the `@use` rule.

This rule loads another Sass file as a `module`, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename.



```
1 // _base.scss
2 $font-stack: Helvetica, sans-serif;
3 $primary-color: #333;
4
5 body {
6   font: 100% $font-stack;
7   color: $primary-color;
8 }
```

Notice we're using `@use` 'base' in the `styles.scss` file.

When you use a file you don't need to include the file extension. **Sass** is smart and will figure it out for you.



```
1 // styles.scss
2 @use 'base';
3
4 .inverse {
5   background-color: base.$primary-color;
6   color: white;
7 }
```





Ariba M.  
@frontendcharm

Keep Swiping...

# Operations :

Remember when we said **SASS is a scripting language** ?  
And we mentioned it's not CSS ?

With SASS, **you can perform mathematical operations**. You can add, subtract, multiply, divide, or find the modulus. You can perform these mathematical operations with SASS variables or plain numbers.



```
1 font-size: 20px - 15px;  
2 width: 100% - $base-width;
```

Here is an example of that. Also, you don't have to use the same units of measurements. You can subtract percentages with pixels. Or rem units with view-height units. **Go crazy.**





Ariba M.  
@frontendcharm

Keep Swiping...

## Conditionals :

Another very logical feature of SASS is conditionals. There are: if, else-if, and else. It looks like this:

The possibilities are endless.

```
1 @if $valid == "true" {  
2   ...  
3 } @else {  
4   ...  
5 }
```

## Functions :

And lastly, SASS offers functions. You can add parameters, if you want.

Simply use  
@function.

```
1 @function isShort($password) {  
2   ...  
3   @return true;  
4 }
```

You can pass values, and you can also return values.



## Partials :

- SASS allows you to **import SASS stylesheets** and use any of the SASS instructions ( functions, mixins, variables ) found in those imported SASS stylesheets.
- SASS files that begin with an underscore are called **Partials**.
- We can create partials for SASS files that we are merely importing. In other words, if you want to import SASS files, those imported files need to start with an **underscore**.
- Also notice how when we are using **@import** we are not writing the file extension **.scss** nor are we writing the **underscore**. That's very important.
- If you want to learn more about that, you can check out the **official SASS documentation**.

Turn ON notification 

# Did you Find it Useful?



**@frontendcharm**  
**Ariba**

## Follow For More !



THANK  
YOU

FOLLOW FOR MORE SUCH  
INTERESTING POST



**Linked in**

<http://www.linkedin.com/in/depaksharma/>