# SASS documentation

## Variables

In SASS we define a variable by adding the $ in front and then we call it normally as we would in a programming language COOL! So if the variable value changes it changes everywhere. No need for repeating. Of course we can assign a variable value to a variable.

//Variable :

$text-color : white;

We now can use this variable :

body {

font-family: sans-serif;

color: $text-color;

}

## Mixin

Mixin is a very important feature of SASS. We combine a lot of stuff together. We name it like this :

@mixin warning{

background-color: orange;

color: white ;

}

And we call it :

.warning-button{

@include warning;

}

or to get multiple properties by this syntax

@mixin large-text{

font:{

size: 40px;

weight: bold ;

}

}

## Nested mixins

We can nest a mixin or multiple mixing inside other mixins.

@mixin rounded{

border-radius:6px;

}

@mixin box{

@include rounded;

background-color: #000088;

}

## Mixin Arguments

We can have mixins with arguments like in a programming language.

@mixin rounded($radius) {

border-radius:$radius;

}

And we take it by passing the parameter we want at parenthesis :

@mixin box{

@include rounded(5px);

background-color: #000088;

}

Or a little more complicated with default values so you don't have to pass arguments :

@mixin rounded($radius : 6px) {

border-radius:$radius;

}

@mixin box($radius : 6px, $border :1px solid black){

@include rounded($radius);

background-color: #000088;

border: $border;

}

And if we call it we can only pass the argument we want and the other will be our default :

#header{

@include box($border :1px solid pink);

}

## Nested mixins and classes/tags

We can nest a mixin or multiple mixing inside other mixins.

@mixin rounded{

border-radius:6px;

}

@mixin box{

@include rounded;

background-color: #000088;

}

Another powerful feature is that you nest a class/tag inside class instead of :

.navigation h2{

color:white;

}

You write :

.navigation {

h2 {

color:white;

}

}

## SASS functions

In CSS we already have some functions which you probably know like rgb, rgba, hsl and hsla. SASS gives us more power with more functions and the ability to create our own. 4 of the most used ones are darken, lighten, transparentize and opacify.

With the code below we are using the darken function and we are saying make the $link-color darker by 15% when we hover on it.

a{

color: $link-color;

**&**:hover{

color: darken($link-color,15%);

}

}

Lighten function:

@mixin warning{

background-color: orange;

color: white ;

**&**:hover{

background-color: lighten(orange,30%);

}

}

Also we have transparentize and opacify functions.Opacity is just the opposite of transperency. If something is 100% opaque,then it's 0% transperent.

#main-menu {

a{

color:$menu-item-color;

padding: 6px 8px;

border-bottom: 1px solid transparentize(#fefefe,1);

transition: border-bottom 0.5s;

transition-timing-function: ease-in-out;

**&**:hover{

border-bottom: 1px solid opacify(#fefefe,0.5);

}

}

}

## SASS Maps(they are like associative arrays/dictionaries)

### Maps with each loop

After version 3 SASS was added map data structure. In simple words map is a box, container whatever you want to call it that you have keys in it with values.

Imagine you have a box that you store your keys. Inside we have 3 keys with names for 3 different purposes.

//Name of the box

$map : (

//Key / Value

garage : blue,

house : red,

cellar : green

);

So let's say your house is ultra futuristic and by writing HTML/CSS you can open doors when you add the correct color to a class :

.garage{

color : map-get($map,garage);

}

.house{

color : map-get($map,house);

}

.cellar{

color : map-get($map,cellar);

}

If we wanted to open all your doors in your amazing house we will do :

@each $key, $keyColor in $map{

.#($key){

color: $keyColor;

}

}

And at CSS with the final loop the result would be :

.garage {

color: blue;

}

.house {

color: red;

}

.cellar {

color: green;

}

## Partial(separating content into smaller files)

Also we can use partial and get content from another underscored scss file .We just put @import rule what to get.

We create a file that will hold all variables called : \_variables.scss

SCSS @import works like javascript require and PHP include.We call the file where we will use it like that:

@import "variables" ;

## Inheritance with @extend

Sometimes we need to have a class that also has something similar with another class. We need to to remember using both classes or more to achieve that.SASS makes that easy with inheritance by using @extend inheritFrom.

.error{

color: red;

}

.critical-error{

@extend .error;

border: 1px solid red;

font-weight: bold;

}

Or we can use any combination we want for example @extend + @include, inheritance and mixin :

.cta-button{

@extend .warning-button;

@include rounded;

}

extend+include

#### Side note for extend :

You can't have extend inside a media query if the class is NOT in the same media query.

Wrong

.cta-button{

@extend .warning-button;

@include rounded;

}

@media screen and (min-width: 960px) {

.super-cta-button{

@extend .cta-button;

}

}

Correct

@media screen and (min-width: 960px) {

.foo{

border-bottom-color: blue;

}

.super-cta-button{

@extend .foo;

}

}