


# MANUAL SCSS

## Modules

Compatibility: Dart Sass since 1.23.0 | LibSass X | Ruby Sass X | 

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. Using a file will also include the CSS it generates in your compiled output!

SCSS Sass

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

CSS

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

.inverse {
  background-color: #333;
  color: white;
}
```

Tenemos dos documentos scss, uno base y otro styles, en el base tenemos el estilo body, muchos asignados con las variables declaradas arriba.

Por otro lado en el styles.scss, declara `@use 'base'` (Especifica que usará info declara en el documento base), y así es como a algunos estilos del inverse asigna base. (Seguido del nombre de la variable declara en `_base.scss`)

## Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. It helps keep your Sass very DRY. You can even pass in values to make your mixin more flexible. Here's an example for `theme`.

SCSS   Sass

```
@mixin theme($theme: DarkGray) {  
  background: $theme;  
  box-shadow: 0 0 1px rgba($theme, .25);  
  color: #fff;  
}  
  
.info {  
  @include theme;  
}  
.alert {  
  @include theme($theme: DarkRed);  
}  
.success {  
  @include theme($theme: DarkGreen);  
}
```

CSS

```
.info {  
  background: DarkGray;  
  box-shadow: 0 0 1px rgba(169, 169, 169, 0.25);  
  color: #fff;  
}  
  
.alert {  
  background: DarkRed;  
  box-shadow: 0 0 1px rgba(139, 0, 0, 0.25);  
  color: #fff;  
}  
  
.success {  
  background: DarkGreen;  
  box-shadow: 0 0 1px rgba(0, 100, 0, 0.25);  
  color: #fff;  
}
```

To create a mixin you use the `@mixin` directive and give it a name. We've named our mixin `theme`. We're also using the variable `$theme` inside the parentheses so we can pass in a theme 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.

Crea un mixin con nombre 'theme', recibe como parámetro, la variable `$theme`: cuyo valor es 'DarkGray', este mixin tiene diferentes propiedades CSS, entre ellas las de background, cuyo valor es el que le hemos asignado a la variable `$theme`.

Por otro lado, creamos otras clases, que las escribimos el `@include theme`, lo que significa que llevarán ese mismo mixin, salvo que las pasemos un parámetro distinto, que hay ya cambiará

## Extend/Inheritance

Using `@extend` lets you share a set of CSS properties from one selector to another. In our example we're going to create a simple series of messaging for errors, warnings and successes using another feature which goes hand in hand with `extend`, placeholder classes. A placeholder class is a special type of class that only prints when it is extended, and can help keep your compiled CSS neat and clean.

SCSS   Sass   ⇒   CSS

```
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

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

Variables que engloban paquetes enteros de estilos