SASS

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.

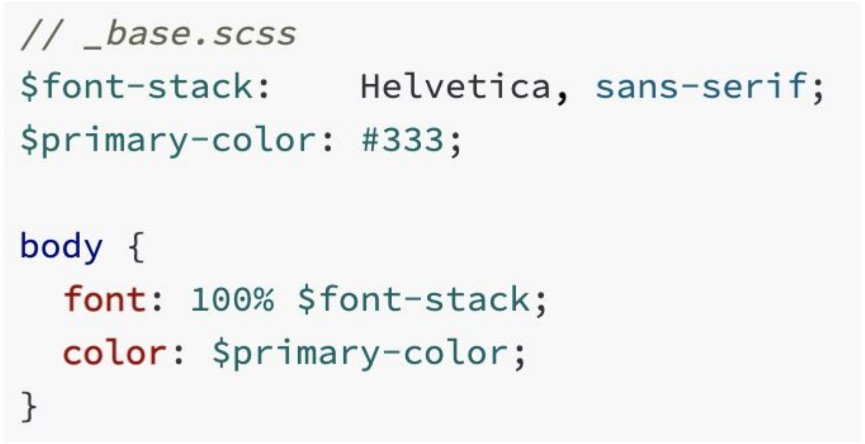
This rule loads another Sass file as a module, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename. Using a file will also include the CSS it generates in your compiled output!

* The Sass team discourages the continued use of the @import rule. Sass will gradually phase it out over the next few years, and eventually remove it from the language entirely.

PARTIALS

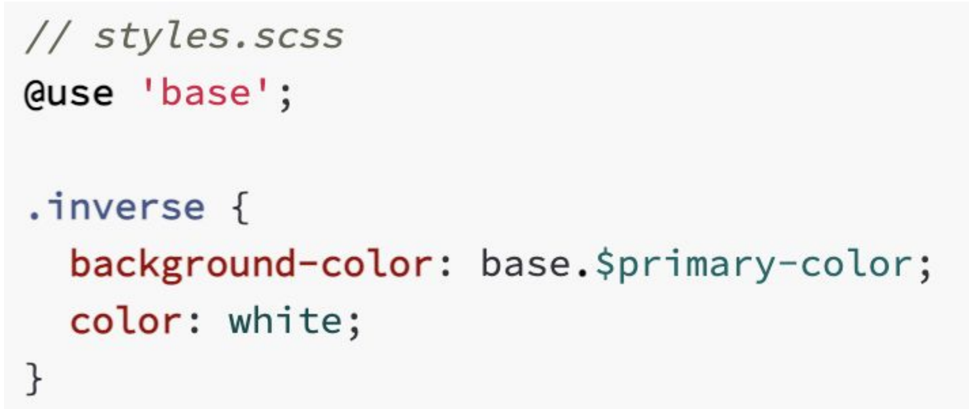
You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. This is a great way to modularize your CSS and help keep things easier to maintain. A partial is a Sass file named with a leading underscore. You might name it something like _partial.scss. 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 @use rule.

EXAMPLE



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

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



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

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

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.

SASS @mixin and @include

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.

@include directive is created to let you use (include) the @mixin.

You can also use arguments which allows you to produce a wide variety of styles with very few mixins.

scss

```scss
@mixin text($size, $lineHeight, $weight) {
  font-size: $size;
  line-height: $lineHeight;
  font-weight: $weight;
}
```

scss

```scss
.MyComponent {
  @include text(18px, 27px, 500);
}

// Compiles to
.MyComponent {
  font-size: 18px;
  line-height: 27px;
  font-weight: 500;
}
```

In this example the @mixin text, takes in three parameters $size, $lineHeight and $weight. Each one is tied to a CSS property. When the mixin is called (with @include), Sass will copy the properties and pass in the argument values.

*In this example we are not using @use because the @mixin is in the same file.

Optional arguments

Normally, every argument a mixin declares must be passed when that mixin is included. However, you can make an argument optional by defining a default value which will be used if that arguments isn’t passed. Default values use the same syntax as variable declarations: the variable name, followed by a colon and the expression.

```scss
@mixin replace-text($image, $x: 50%, $y: 50%) {
  text-indent: -99999em;
  overflow: hidden;
  text-align: left;

  background: {
    image: $image;
    repeat: no-repeat;
    position: $x $y;
  }
}

.mail-icon {
  @include replace-text(url("/images/mail.svg"), 0);
}
```

ANY QUESTIONS?