



Code4SP Training Materials Subchapter 3: CSS

WP3: Code4SP Training Materials

Prepared by: Social Hackers Academy



CITIZENS
IN POWER



Center for Social
Innovation





WHAT IS SASS?

- Sass stands for Syntactically Awesome Stylesheet.
- Sass is an extension to CSS. Sass is a CSS pre-processor.
- Sass is completely compatible with all versions of CSS.
- Sass reduces repetition of CSS and therefore saves time.
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006.
- Sass is free to download and use.





Why use SASS?

Stylesheets are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help. Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.





A Simple Example why Sass is Useful

Let's say we have a website with three main colors:

#a2b9bc

#b2ad7f

#878f99





A Simple Example why Sass is Useful

So, how many times do you need to type those HEX values? A LOT of times. And what about variations of the same colors? Instead of typing the above values a lot of times, you can use Sass and write this:

```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```





How Does Sass Work?

A browser does not understand Sass code. Therefore, you will need a Sass pre-processor to convert Sass code into standard CSS. This process is called transpiling. So, you need to give a transpiler (some kind of program) some Sass code and then get some CSS code back.





Sass Comments

Sass supports standard CSS comments `/* comment */`, and in addition it supports inline comments `// comment`:

```
/* define primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;

/* use the variables */
.main-header {
  background-color: $primary_1; // here you can put an inline comment
}
```





Sass Variables

Variables are a way to store information that you can re-use later.

With Sass, you can store information in variables, like:

- Strings
- Numbers
- Colors
- Booleans
- Lists
- nulls





Sass Variables

Sass uses the \$ symbol, followed by a name, to declare variables:

Sass Variable Syntax:

```
$variablename: value;
```





Sass Variables

The following example declares 4 variables named myFont, myColor, myFontSize, and myWidth. After the variables are declared, you can use the variables wherever you want:

```
$myFont: Helvetica, sans-serif;
$myColor: red;
$myFontSize: 18px;
$myWidth: 680px;

body {
  font-family: $myFont;
  font-size: $myFontSize;
  color: $myColor;
}

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





Sass Nested Rules

Sass lets you nest CSS selectors in the same way as HTML. Look at an example of some Sass code 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;  
  }  
}
```





Sass Importing Files

Just like CSS, Sass also supports the `@import` directive. The `@import` directive allows you to include the content of one file in another.

The CSS `@import` directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it.

However, the Sass `@import` directive includes the file in the CSS; so no extra HTTP call is required at runtime!

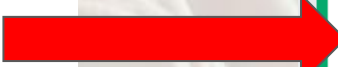


Sass Importing Files

SCSS Syntax (reset.scss):

```
html,  
body,  
ul,  
ol {  
  margin: 0;  
  padding: 0;  
}
```

SCSS Syntax (standard.scss):



```
@import "reset";  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: red;  
}
```



Sass Mixins

- The `@mixin` directive lets you create CSS code that is to be reused throughout the website.
- The `@include` directive is created to let you use (include) the mixin.



Defining a Mixin

The following example creates a mixin named "important-text":

```
@mixin name {  
  property: value;  
  property: value;  
  ...  
}
```



```
@mixin important-text {  
  color: red;  
  font-size: 25px;  
  font-weight: bold;  
  border: 1px solid blue;  
}
```



Using a Mixin

The `@include` directive is used to include a mixin.

So, to include the `important-text` mixin created:

```
selector {  
  @include mixin-name;  
}
```



```
.danger {  
  @include important-text;  
  background-color: green;  
}
```





Sass @extend Directive

- The @extend directive lets you share a set of CSS properties from one selector to another.
- The @extend directive is useful if you have almost identically styled elements that only differ in some small details.





Sass *@extend* Directive

The following Sass example first creates a basic style for buttons (this style will be used for most buttons). Then, we create one style for a "Report" button and one style for a "Submit" button. Both "Report" and "Submit" button inherit all the CSS properties from the `.button-basic` class, through the `@extend` directive. In addition, they have their own colors defined:

```
.button-basic {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report {
  @extend .button-basic;
  background-color: red;
}

.button-submit {
  @extend .button-basic;
  background-color: green;
  color: white;
}
```

