CSS PREPROCESSORS - SASS

Tehran Institute of Technology

Zahra Mansoori
z.mansoori@gmail.com
http://tarsimm.com

Contents

1. Sass Introduction

Sass stands for Syntactically Awesome Stylesheet. It is an extension to CSS, a CSS pre-processor. It is completely compatible with all versions of CSS.

Sass reduces repetition of CSS and therefore saves time.

1.1. Sass — History

Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006. It is an open-source.

1.2. Sass — Features

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.

Read more about Sass at the official Sass web site: https://sass-lang.com/

1.3. 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 uses the $ symbol, followed by a name, to declare variables:

```
$variablename: value;
```

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:

Example. 1: Variables

```
<!DOCTYPE html>
<html>
      <link rel="stylesheet" href="mystyle.css">
      <body>
            <h1>Hello World</h1>
            <p>This is a paragraph.</p>
            <div id="container">This is some text inside a
            container.</div>
      </body>
</html>
```

myStyle.scss

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

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

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

myStyle.css

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

#container {
  width: 680px;
}
```

1.4. Sass Variable Scope

Sass variables are only available at the level of nesting where they are defined.

Example. 2: Variable Scope

```
<!DOCTYPE html>
<html>
      <link rel="stylesheet" href="mystyle.css">
      <body>
            <h1>Hello World</h1>
            <p>This is a paragraph.</p>
      </body>
</html>
```

myStyle.scss

```
$myColor: red;
h1 {
  $myColor: green;
  color: $myColor;
}
p {
  color: $myColor;
}
```

myStyle.css

```
h1 {
  color: green;
}

p {
  color: red;
}
```

1.5. Using Sass !global

The default behavior for variable scope can be overridden by using the `!global` switch. `!global` indicates that a variable is global, which means that it is accessible on all levels.

Example. 3: !global usage

```
<!DOCTYPE html>
<html>
      <link rel="stylesheet" href="mystyle.css">
      <body>
            <h1>Hello World</h1>
            <p>This is a paragraph.</p>
      </body>
</html>
```

myStyle.scss

```
$myColor: red;

h1 {
  $myColor: green !global;
  color: $myColor;
}
p {
  color: $myColor;
}
```

myStyle.css

```
h1 {
  color: green;
}

p {
  color: green;
}
```

1.6. Sass Nested Rules

Sass lets you nest CSS selectors in the same way as HTML.

Example. 4: Nested Rules

```
<!DOCTYPE html>
<html>
      <link rel="stylesheet" href="mystyle.css">
      <body>
            <nav>
              <ul>
                <li><a href="#">HTML</a></li>
                <li><a href="#">CSS</a></li>
                <li><a href="#">SASS</a></li>
              </ul>
            </nav>
      </body>
</html>
```

myStyle.scss

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
```

```
    li {
      display: inline-block;
    }
    a {
      display: block;
      padding: 6px 12px;
      text-decoration: none;
    }
  }
```

myStyle.css

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

1.7. Sass @import

Sass keeps the CSS code DRY (Don't Repeat Yourself). One way to write DRY code is to keep related code in separate files.

You can create small files with CSS snippets to include in other Sass files. Examples of such files can be: reset file, variables, colors, fonts, font-sizes, etc.

Example. 5: @import

reset.scss

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

standard.scss

```
@import "reset";

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

standard.css

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

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

1.8. Sass Partials

By default, Sass transpiles all the .scss files directly. However, when you want to import a file, you do not need the file to be transpiled directly.

Sass has a mechanism for this: If you start the filename with an underscore, Sass will not transpile it. Files named this way are called partials in Sass.

So, a partial Sass file is named with a leading underscore:

```
_filename;
```

Example. 6: Partials

_colors.scss

```
$myPink: #EE82EE;
$myBlue: #4169E1;
$myGreen: #8FBC8F;
```

standard.scss

```
@import "colors";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: $myBlue;
}
```

1.9. 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.

1.9.1. Defining a Mixin

Sass @mixin Syntax:

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

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

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

1.9.2. Using Mixins

The @include directive is used to include a mixin.

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

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

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

Example. 7: Mixins

```
<!DOCTYPE html>
<html>
      <link rel="stylesheet" href="mystyle.css">
      <body>
            <h1>Hello World</h1>
            <p class="danger">Warning! This is some text.</p>
      </body>
</html>
```

mystyle.scss

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

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

mystyle.scss

```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
  background-color: green;
}
```