Sass

Sass is a CSS extension language that adds features such as variables, nesting, mixins and more. Sass (and other preprocessors, such as Less) help developers write maintainable and DRY (Don't Repeat Yourself) code.

Sass has two different syntax options to choose from. SCSS, which has the same syntax as CSS but with the added features of Sass. Or Sass (the original syntax), which uses indentation rather than curly braces and semicolons. This tutorial is written using SCSS.

If you're already familiar with CSS3, you'll be able to pick up Sass relatively quickly. It does not provide any new styling properties but rather the tools to write your CSS more efficiently and make maintenance much easier.

```
//Single line comments are removed when Sass is compiled to CSS.

/* Multi line comments are preserved. */



/* Variables
============================== */



/* You can store a CSS value (such as a color) in a variable.
Use the '$' symbol to create a variable. */

$primary-color: #A3A4FF;
$secondary-color: #51527F;
$body-font: 'Roboto', sans-serif;

/* You can use the variables throughout your stylesheet.
Now if you want to change a color, you only have to make the change once. */

body {
    background-color: $primary-color;
    color: $secondary-color;
    font-family: $body-font;
}

/* This would compile to: */
body {
    background-color: #A3A4FF;
    color: #51527F;
    font-family: 'Roboto', sans-serif;
}


/* This is much more maintainable than having to change the color
each time it appears throughout your stylesheet. */
```

```
/* Mixins
============================== */



/* If you find you are writing the same code for more than one
element, you might want to store that code in a mixin.

Use the '@mixin' directive, plus a name for your mixin. */

@mixin center {
    display: block;
    margin-left: auto;
    margin-right: auto;
    left: 0;
    right: 0;
}

/* You can use the mixin with '@include' and the mixin name. */

div {
    @include center;
    background-color: $primary-color;
}

/* Which would compile to: */
div {
    display: block;
    margin-left: auto;
    margin-right: auto;
    left: 0;
    right: 0;
    background-color: #A3A4FF;
}


/* You can use mixins to create a shorthand property. */

@mixin size($width, $height) {
    width: $width;
    height: $height;
}

/* Which you can invoke by passing width and height arguments. */

.rectangle {
    @include size(100px, 60px);
}

.square {
    @include size(40px, 40px);
}
```

```
/* Compiles to: */
.rectangle {
  width: 100px;
  height: 60px;
}

.square {
  width: 40px;
  height: 40px;
}



/* Functions
============================== */



/* Sass provides functions that can be used to accomplish a variety of
   tasks. Consider the following */

/* Functions can be invoked by using their name and passing in the
   required arguments */
body {
  width: round(10.25px);
}

.footer {
  background-color: fade_out(#000000, 0.25)
}

/* Compiles to: */

body {
  width: 10px;
}

.footer {
  background-color: rgba(0, 0, 0, 0.75);
}

/* You may also define your own functions. Functions are very similar to
   mixins. When trying to choose between a function or a mixin, remember
   that mixins are best for generating CSS while functions are better for
   logic that might be used throughout your Sass code. The examples in
   the Math Operators' section are ideal candidates for becoming a reusable
   function. */

/* This function will take a target size and the parent size and calculate
   and return the percentage */

@function calculate-percentage($target-size, $parent-size) {
  @return $target-size / $parent-size * 100%;
}
```

```
$main-content: calculate-percentage(600px, 960px);

.main-content {
  width: $main-content;
}

.sidebar {
  width: calculate-percentage(300px, 960px);
}

/* Compiles to: */

.main-content {
  width: 62.5%;
}

.sidebar {
  width: 31.25%;
}



/* Extend (Inheritance)
============================== */



/* Extend is a way to share the properties of one selector with another. */

.display {
    @include size(5em, 5em);
    border: 5px solid $secondary-color;
}

.display-success {
    @extend .display;
    border-color: #22df56;
}

/* Compiles to: */
.display, .display-success {
  width: 5em;
  height: 5em;
  border: 5px solid #51527F;
}

.display-success {
  border-color: #22df56;
}

/* Extending a CSS statement is preferable to creating a mixin
   because of the way Sass groups together the classes that all share
   the same base styling. If this was done with a mixin, the width,
```

```
   height, and border would be duplicated for each statement that
   called the mixin. While it won't affect your workflow, it will
   add unnecessary bloat to the files created by the Sass compiler. */



/* Nesting
============================== */



/* Sass allows you to nest selectors within selectors */

ul {
    list-style-type: none;
    margin-top: 2em;

    li {
        background-color: #FF0000;
    }
}

/* '&' will be replaced by the parent selector. */
/* You can also nest pseudo-classes. */
/* Keep in mind that over-nesting will make your code less maintainable.
Best practices recommend going no more than 3 levels deep when nesting.
For example: */

ul {
    list-style-type: none;
    margin-top: 2em;

    li {
        background-color: red;

        &:hover {
          background-color: blue;
        }

        a {
          color: white;
        }
    }
}

/* Compiles to: */

ul {
  list-style-type: none;
  margin-top: 2em;
}

ul li {
  background-color: red;
```

```
}

ul li:hover {
  background-color: blue;
}

ul li a {
  color: white;
}



/* Partials and Imports
============================== */



/* Sass allows you to create partial files. This can help keep your Sass
   code modularized. Partial files should begin with an '_', e.g. _reset.css.
   Partials are not generated into CSS. */

/* Consider the following CSS which we'll put in a file called _reset.css */

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

/* Sass offers @import which can be used to import partials into a file.
   This differs from the traditional CSS @import statement which makes
   another HTTP request to fetch the imported file. Sass takes the
   imported file and combines it with the compiled code. */

@import 'reset';

body {
  font-size: 16px;
  font-family: Helvetica, Arial, Sans-serif;
}

/* Compiles to: */

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

body {
  font-size: 16px;
  font-family: Helvetica, Arial, Sans-serif;
}
```

```
/* Placeholder Selectors
============================== */



/* Placeholders are useful when creating a CSS statement to extend. If you
   wanted to create a CSS statement that was exclusively used with @extend,
   you can do so using a placeholder. Placeholders begin with a '%' instead
   of '.' or '#'. Placeholders will not appear in the compiled CSS. */

%content-window {
  font-size: 14px;
  padding: 10px;
  color: #000;
  border-radius: 4px;
}

.message-window {
  @extend %content-window;
  background-color: #0000ff;
}

/* Compiles to: */

.message-window {
  font-size: 14px;
  padding: 10px;
  color: #000;
  border-radius: 4px;
}

.message-window {
  background-color: #0000ff;
}



/* Math Operations
============================== */



/* Sass provides the following operators: +, -, *, /, and %. These can
   be useful for calculating values directly in your Sass files instead
   of using values that you've already calculated by hand. Below is an example
   of a setting up a simple two column design. */

$content-area: 960px;
$main-content: 600px;
$sidebar-content: 300px;
```

```
$main-size: $main-content / $content-area * 100%;
$sidebar-size: $sidebar-content / $content-area * 100%;
$gutter: 100% - ($main-size + $sidebar-size);

body {
  width: 100%;
}

.main-content {
  width: $main-size;
}

.sidebar {
  width: $sidebar-size;
}

.gutter {
  width: $gutter;
}

/* Compiles to: */

body {
  width: 100%;
}

.main-content {
  width: 62.5%;
}

.sidebar {
  width: 31.25%;
}

.gutter {
  width: 6.25%;
}
```

SASS or Sass?

Have you ever wondered whether Sass is an acronym or not? You probably haven't, but I'll tell you anyway. The name of the language is a word, "Sass", and not an acronym. Because people were constantly writing it as "SASS", the creator of the language jokingly called it "Syntactically Awesome StyleSheets".

Practice Sass

If you want to play with Sass in your browser, check out SassMeister. You can use either syntax, just go into the settings and select either Sass or SCSS.

Compatibility

Sass can be used in any project as long as you have a program to compile it into CSS. You'll want to verify that the CSS you're using is compatible with your target browsers.

QuirksMode CSS and CanIUse are great resources for checking compatibility.

Further reading

- Official Documentation
- The Sass Way provides tutorials (beginner-advanced) and articles.