SASS-101 Cheat Sheet

A cheat sheet for the basics of SASS (Syntactically Awesome StyleSheets) that covers everything from variables and nesting to mixins and inheritance. This is a great resource for anyone who is just getting started with SASS.

Table of Contents

What is Sass?

Sass is a CSS preprocessor. It lets you use features that don't exist in CSS yet like variables, nesting, mixins, inheritance and other nifty goodies that make writing CSS fun again.

- Sass (Syntactically Awesome StyleSheets)
- CSS Preprocessor/Extension
- Features that don't exist in CSS yet
- Variables, nesting, mixins, inheritance, etc.
- Sass (.scss) files are compiled into CSS (.css) files

Allows us to have logic in CSS. Browser doesn't understand Sass, so we need to compile it into CSS. Sass is a superset of CSS, so all valid CSS is valid Sass.

.scss vs .sass

- .scss: Sassy CSS
- .sass: Syntactically Awesome StyleSheets

Sass is the original syntax, but it's not as popular. It's indentation based, so you don't need to use curly braces or semicolons. It's a little bit more concise, but it's not as popular. We'll be using .scss.

```
 // .scss
$primary-color: #f00;

body {
  background-color: $primary-color;
}
```

```
 // .sass
$primary-color: #f00

body
  background-color: $primary-color
```

Variables in Sass

- Variables start with a dollar sign
- Variables can be used to store colors, font stacks, or any CSS value
- Variables must be declared before they are used
- If a variable is declared twice, the last value will be used

Allows us to store values and reuse them throughout our code. So later if we want to change the color, we can just change it in one place.

```
$primary-color: #f00;

body {
  background-color: $primary-color;
}

h1 {
  color: $primary-color;
}
```

Nesting in Sass

- Nesting allows us to nest selectors inside of other selectors
- This helps to keep our code more organized
- It also helps to avoid repetition and saves time

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

  li {
    display: inline-block;
  }

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

Mixins in Sass

- Mixins allow us to define styles that can be re-used throughout the stylesheet

```
@mixin flex-center {
  display: flex;
  justify-content: center;
  align-items: center;
}

.container {
  @include flex-center;
}
```

Mixins are like functions. We can pass in arguments to mixins.

```
@mixin flex-center($justify, $align) {
  display: flex;
  justify-content: $justify;
  align-items: $align;
}

.container {
  @include flex-center(center, center);
}
```

Functions in Sass

- Functions allow us to create reusable logic within our Sass files

- Similar to mixins, but they return a value based on the arguments that are passed in

```
@function divide($a, $b) {
  @return $a / $b;
}

.container {
  width: divide(100, 2) * 1px;
}
```

Inheritance in Sass

- Inheritance allows us to share a set of CSS properties from one selector to another
- This helps to keep our code DRY

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

.btn-primary {
  @extend %btn;
  background-color: #f00;
  color: #fff;
}

.btn-secondary {
  @extend %btn;
  background-color: #000;
  color: #fff;
}
```

Operators in Sass

- Sass supports a handful of operators that can be used when doing calculations

```
.container {
  width: 100% / 2;
}
```

Conditionals in Sass

- Sass supports conditionals like if/else statements

```
@mixin rect($width, $height) {
  @if $width > $height {
    width: $width;
    height: $height;
  } @else {
    width: $height;
    height: $width;
  }
}

.container {
  @include rect(100px, 200px);
}
```

Loops in Sass

- Sass supports loops like for and while

```
@for $i from 1 through 3 {
  .card-#{$i} {
    width: 2em * $i;
  }
}
```

Partials in Sass

- Partials allow us to split our Sass code up into separate files
- Partials are prefixed with an underscore
- Partials are imported into a main Sass file using the @use rule

This is very useful for creating single components like Buttons, Cards, etc. We can create a separate file for each component and import them into our main Sass file.

```
// _variables.scss
$primary-color: #f00;

// main.scss
@use 'variables.scss';

body {
  background-color: variables.$primary-color;
}
```

As long as we use the underscore in the file name, it won't be compiled into a separate css file. So we can just import it into our main Sass file.

> When we import css modules, multiple http requests are made. When we import sass modules, they are all compiled into a single css file.

Compiling Sass

- Sass can be compiled using a build tool like Gulp or Grunt or using the Sass CLI
- If you're using VS Code, you can install the Live Sass Compiler extension

Build Tool

- We can install the Sass CLI using `npm install -g sass`
- Compile Sass using `sass input.scss output.css` or `sass --watch input.scss output.css`

VS Code Extension

- Install the **Live Sass Compiler** extension
- Go to **File > Preferences > Settings** and click **Edit in settings.json**
- Add the following code to the settings.json file:

```
"liveSassCompile.settings.formats": [
  {
    "format": "expanded",
    "extensionName": ".css",
    "savePath": "/css"
  }
],
"liveSassCompile.settings.generateMap": false
```

This will compile our Sass files to CSS and save them in a css folder in the root directory of our project (we need to create this folder manually).

- Open the Sass file you want to compile
- Click **Watch Sass** in the bottom right corner of the window
- Now whenever we save our Sass file, it will automatically compile it to CSS.

If we make any mistakes in our Sass file, it will show us an error in the output CSS file. So we can use this to debug our Sass code.

Outro

Thank you for checking out this cheat sheet!

If you found it helpful please check out more of my work on yodkwtf.com or follow me on twitter. I also run a small youtube channel called Yodkwtf Academy.