What is Sass?

Sass (Syntactically Awesome Stylesheets) is a CSS preprocessor that extends CSS with additional features, making it more powerful and easier to maintain. It provides advanced functionalities like variables, nesting, mixins, inheritance, and modules, which simplify styling for larger projects.

Sass comes in two syntaxes:

SCSS: Uses a syntax similar to CSS but with extra features. Files have the .scss extension.

Indented Syntax: A more concise version of Sass, without braces and semicolons. Files have the .sass extension.

Key Differences Between Sass and Regular CSS

Variables: Sass allows you to define variables for colors, fonts, sizes, etc., which aren't available in vanilla CSS (although CSS now supports custom properties).

Nesting: Sass allows CSS rules to be nested within one another, following the HTML structure, whereas vanilla CSS requires all rules to be written separately.

Partials and Imports: Sass allows splitting code into multiple files (partials) and importing them into one main file.

Mixins and Functions: Sass provides reusable blocks of code and functions, unlike standard CSS.

Example Sass File Demonstrating Nesting, Partials, and Imports

1. Partials: We can break down the styling into smaller, manageable files called partials. These partials can then be imported into a main Sass file. Partials have filenames that start with an underscore (\_).

Let’s create three partials:

\_variables.scss – to store variables

\_navbar.scss – to define styles for the navbar

\_footer.scss – to define styles for the footer

\_variables.scss

scss

Copy code

// Define variables

$primary-color: #3498db;

$font-stack: Helvetica, sans-serif;

\_navbar.scss

scss

Copy code

// Use variables and nesting for the navbar styles

.navbar {

background-color: $primary-color;

font-family: $font-stack;

ul {

list-style: none;

padding: 0;

li {

display: inline-block;

a {

text-decoration: none;

color: white;

&:hover {

color: darken($primary-color, 10%);

}

}

}

}

}

\_footer.scss

scss

Copy code

// Footer styles with variables

.footer {

background-color: lighten($primary-color, 20%);

font-family: $font-stack;

text-align: center;

padding: 10px;

}

2. Main File (styles.scss)

The main Sass file will import the partials.

scss

Copy code

// Import partials

@import 'variables';

@import 'navbar';

@import 'footer';

Compiled CSS Output

After compiling the Sass files (styles.scss) into CSS using a tool like the Sass CLI or a build tool (e.g., Webpack), the compiled CSS will look like this:

css

Copy code

.navbar {

background-color: #3498db;

font-family: Helvetica, sans-serif;

}

.navbar ul {

list-style: none;

padding: 0;

}

.navbar ul li {

display: inline-block;

}

.navbar ul li a {

text-decoration: none;

color: white;

}

.navbar ul li a:hover {

color: #2a7abf;

}

.footer {

background-color: #5dade2;

font-family: Helvetica, sans-serif;

text-align: center;

padding: 10px;

}

Notable Differences from Standard CSS

Nesting: In Sass, nested structures like .navbar ul li a:hover are defined logically within the parent blocks, making the code more readable. In the compiled CSS, these are flattened into standard selectors.

Sass: .navbar ul li a:hover

Compiled CSS: .navbar ul li a:hover

Variables: In Sass, $primary-color and $font-stack are used throughout different files, allowing for easy updates. In the compiled CSS, these variables are replaced by their values, such as #3498db and Helvetica, sans-serif.

Imports: Sass allows the use of @import to break styles into multiple partials, which are combined into one CSS file after compilation. Regular CSS also supports @import, but it loads each file separately, which can slow down performance. In Sass, all styles are compiled into a single CSS file.

Functions and Mixins: The compiled CSS includes color manipulation using the darken() and lighten() functions in Sass, which aren’t available in vanilla CSS. These functions generate darker and lighter shades directly in the output CSS.

Summary

Sass enhances CSS by adding features like variables, nesting, and partials, making the code easier to manage and more maintainable.

Nesting allows for more readable and logical code organization.

Partials and Imports help modularize styles, but after compilation, everything is combined into a single CSS file, unlike standard CSS @import.

CSS compilation turns Sass code into browser-compatible CSS, simplifying the management of complex style rules.

In a nutshell, Sass provides a more organized, efficient way to write CSS, especially for large-scale projects.