Variables and Functions in Sass

1. Variables in Sass

Variables in Sass allow you to store reusable values such as colors, fonts, sizes, or any other CSS value. This helps maintain consistency across your stylesheets and simplifies making global changes, as you only need to update the variable once.

Syntax:

scss

Copy code

$variable-name: value;

2. Functions in Sass

Sass functions allow you to create reusable blocks of code that return a value. Sass includes many built-in functions (e.g., for manipulating colors, numbers, strings), but you can also define custom functions. This is helpful when you need to apply calculations or logic to your styles, keeping the code DRY (Don’t Repeat Yourself).

Syntax:

scss

Copy code

@function function-name($parameter) {

@return some-value;

}

Example: Using Variables and Functions in Sass

Let’s create a Sass file that demonstrates the use of variables and a custom function.

Example Sass Code:

scss

Copy code

// Variables

$primary-color: #3498db;

$secondary-color: #2ecc71;

$base-font-size: 16px;

// Function to calculate rem values

@function calculate-rem($size-in-px) {

@return $size-in-px / $base-font-size \* 1rem;

}

// Example usage of variables and functions

body {

font-size: $base-font-size;

color: $primary-color;

}

h1 {

font-size: calculate-rem(32px);

color: $primary-color;

}

p {

font-size: calculate-rem(14px);

color: $secondary-color;

}

Compiled CSS Output:

css

Copy code

body {

font-size: 16px;

color: #3498db;

}

h1 {

font-size: 2rem;

color: #3498db;

}

p {

font-size: 0.875rem;

color: #2ecc71;

}

Explanation:

Variables: The variables $primary-color, $secondary-color, and $base-font-size store commonly used values. By using variables, you can change these values in one place and update the styles across the entire site.

Custom Function (calculate-rem): The calculate-rem() function converts pixel values into rem units. This function simplifies calculating rem values based on the base font size, ensuring consistent typography across the site.

In the example, the h1 element uses calculate-rem(32px) to convert 32px to 2rem (since 32px / 16px = 2), and p uses calculate-rem(14px) to convert 14px to 0.875rem.

Inheritance in Sass

Inheritance in Sass works similarly to inheritance in object-oriented programming, allowing a class or selector to inherit styles from another selector. This is done using the @extend directive. Inheritance is useful when different elements share common styles but need individual modifications.

Syntax:

scss

Copy code

.selector1 {

// Common styles

}

.selector2 {

@extend .selector1;

// Additional specific styles

}

Example of Inheritance in Sass:

Let’s say we have two button types (.button and .primary-button) that share some common styles, but the primary button has extra styles.

Sass Code:

scss

Copy code

// Base button styles

.button {

padding: 10px 20px;

font-size: 16px;

background-color: lightgray;

border: none;

cursor: pointer;

}

// Inherit base styles and customize for primary button

.primary-button {

@extend .button;

background-color: $primary-color;

color: white;

}

Compiled CSS Output:

css

Copy code

.button, .primary-button {

padding: 10px 20px;

font-size: 16px;

background-color: lightgray;

border: none;

cursor: pointer;

}

.primary-button {

background-color: #3498db;

color: white;

}

Explanation:

Inheritance (@extend): The .primary-button inherits all the styles from .button (such as padding, font-size, and cursor styles). Then, additional styles specific to .primary-button are applied (like the $primary-color background and white text).

Efficiency: Instead of duplicating the common button styles, Sass compiles the CSS in a way that minimizes repetition by merging selectors where possible (i.e., .button, .primary-button {}).

Advantages of Inheritance in Sass

Code Reusability: Common styles can be defined once and reused across multiple elements or components, reducing redundancy.

Maintenance: If common styles need to be updated, you only need to change them in one place.

Consistency: Inheritance ensures that all components that extend a base class maintain consistent styles, making design elements uniform across the site.

Summary

Variables in Sass store reusable values like colors and font sizes, making it easier to maintain consistency across a project.

Functions in Sass allow for reusable logic, such as calculations, to be applied to styles. They help automate tasks like converting pixel values to rem.

Inheritance in Sass, through the @extend directive, allows selectors to inherit styles from other selectors, reducing code duplication and improving maintainability.

These features make Sass a powerful tool for creating organized, scalable, and maintainable stylesheets.