- Syntactically Awesome Style Sheets," is a preprocessor scripting language that is interpreted or compiled into CSS.
- It extends CSS with features like variables, nesting, mixins, and more, allowing for easier and more efficient styling of web pages

■ SASS VS SCSS

Sass (Syntactically Awesome Style Sheets)

- Original syntax, using indentation for nesting and omitting semicolons and braces.
- Uses the `.sass` file extension.

```
$primary-color: #FF6347

body
  font-family: Arial, sans-serif
  color: $primary-color;
```

SCSS (Sassy CSS)

- Newer syntax, closer to standard CSS with curly braces, semicolons, and indentation for readability.
- Uses the `.scss` file extension.

```
$primary-color: #FF6347;

body {
  font-family: Arial, sans-serif;
  color: $primary-color;
}
```

Key Differences

1. **Syntax**: SCSS syntax is closer to CSS, making it easier for developers familiar with CSS to transition to SCSS. Sass syntax, on the other hand, uses indentation and omits certain characters like semicolons and braces.
2. **File Extension**: The file extension `.sass` is used for Sass files, while `.scss` is used for SCSS files.
3. **Interchangeability**: Both Sass and SCSS files can be imported into each other, so you can mix and match them in your project.

■ Key features of Sass

1. **Variables**: Assign names to values like colors or font families, making it simple to update them universally throughout your stylesheets.
2. **Nesting**: Organize CSS selectors within one another, following the HTML structure, which improves readability by clearly indicating hierarchy and relationships between elements.
3. **Mixins**: Group sets of CSS declarations into reusable modules, allowing you to apply the same styles across multiple elements without repeating code, enhancing maintainability and reducing redundancy.
4. **Partials and Imports**: Break down your stylesheets into smaller, more manageable files called partials, then bring them together using imports, promoting code organization and making it easier to collaborate on larger projects.
5. **Mathematical Operations**: Perform arithmetic directly within your stylesheets, enabling dynamic adjustments to values like sizes or colors based on calculations, enhancing flexibility and efficiency in styling.
6. **Control Directives**: Introduce conditional statements and looping constructs into your stylesheets using directives like `@if`, `@for`, and `@each`, providing greater control over the application of styles and facilitating more complex styling logic.

■ Sass Installation

```
npm install -g sass
sass style.scss style.css
```

■ Sass Variables

Typed Variables: Define variables with specific types, such as numbers, strings, colors, or lists.

```
$primary-color: #FF6347; // Color
$font-size: 16px; // Number
$font-stack: "Arial", sans-serif; // String
$font-size-responsive: (sm: 16px, md: 26px, lg: 36px); // Map
```

Usage: Utilize typed variables just like regular ones, ensuring consistency and type safety throughout your stylesheets.

```
// Sass Variables with Types

// Sass Variables with Types
$primary-color: #FF6347; // Color
$font-size: 16px; // Number
```

```scss
$font-stack: "Arial", sans-serif; // String
$font-size-responsive: (sm: 16px, md: 26px, lg: 36px); // Map

// Example Usage
.button {
  background-color: $primary-color;
  font-size: $font-size;
  font-family: $font-stack;
}

// Responsive Font Size Usage
@media screen and (min-width: 576px) {
  .button {
    font-size: map-get($font-size-responsive, sm);
  }
}

@media screen and (min-width: 768px) {
  .button {
    font-size: map-get($font-size-responsive, md);
  }
}

@media screen and (min-width: 992px) {
  .button {
    font-size: map-get($font-size-responsive, lg);
  }
}
```

■ Sass Nesting

```html
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Services</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>
```

```scss
// Define styles for the navigation bar
nav {
```

```
  background-color: #333;
  padding: 10px;

  // Style the unordered list inside the navigation bar
  ul {
    list-style: none;
    margin: 0;
    padding: 0;

    // Style list items
    li {
      display: inline-block;
      margin-right: 10px;

      // Style links inside list items
      a {
        color: #fff;
        text-decoration: none;
        padding: 5px 10px;

        // Change link color on hover
        &:hover {
          background-color: #555;
        }
      }
    }
  }
}
```

- The `nav` selector applies styles to the navigation bar.
- Inside `nav`, the `ul` selector styles the unordered list, removing default list styles and adjusting margins and padding.
- Nested within `ul`, the `li` selector styles list items as inline blocks with a margin to separate them.
- Inside `li`, the `a` selector styles links within list items, setting text color, padding, and removing underlines.
- The nested `&:hover` selector changes the background color of links when hovered over.

■ Sass @import and Partials

Sass `@import` directive and partials allow you to break your stylesheets into smaller, more manageable files

1. **Create Partial Files**: Name your partial Sass files with a leading underscore, like `_variables.scss`, `_mixins.scss`, etc. This convention tells Sass not to compile them into separate CSS files.
2. **Define Styles**: Write your CSS rules or Sass variables, mixins, or functions inside these partial files.

```
@import 'variables';

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

Benefits:

- **Modularity**: Break your code into smaller, reusable chunks for better organization and maintainability.
- **Scalability**: Easily add or remove features by managing smaller files.
- **Readability**: Improve code readability by separating concerns into logical units.

■ Sass Mixins

Sass mixins allow you to define reusable sets of CSS declarations that can be included anywhere in your stylesheets

Define Mixins: Use the `@mixin` directive followed by a name and a set of CSS declarations within curly braces.

```
@mixin button-styles {
  border: 1px solid #ccc;
  padding: 10px 20px;
  background-color: #f0f0f0;
  color: #333;
}
.button {
  @include button-styles;
}
```

■ Passing Variables to a Mixin

```scss
// Define a mixin with parameters
@mixin button-styles($border-color, $background-color, $text-color) {
  border: 1px solid $border-color;
  padding: 10px 20px;
  background-color: $background-color;
  color: $text-color;
}

// Use the mixin and pass variables
.button {
  @include button-styles(#ccc, #f0f0f0, #333);
}
```

■ Sass @extend and Inheritance

The `@extend` directive in Sass allows you to share a set of CSS properties from one selector to another, effectively inheriting styles.

```scss
// Define a base button style
.btn-base {
  border: 1px solid #ccc;
  padding: 10px 20px;
  background-color: #f0f0f0;
  color: #333;
}

// Apply the base style to specific buttons
.btn-primary {
  @extend .btn-base;
  background-color: #FF6347; // Customize specific properties for primary button
}

.btn-secondary {
  @extend .btn-base;
  background-color: #6495ED; // Customize specific properties for secondary button
}
```

■ Mathematical Operations

Basic Operations

You can use basic arithmetic operators like `+`, `-`, `*`, and `/` to perform calculations.

```scss
$base-font-size: 16px;
$multiplier: 1.5;

h1 {
  font-size: $base-font-size * $multiplier;
}
```

Unit Conversion

Sass automatically converts units when necessary.

```scss
$base-padding: 20px;
$padding-percent: 0.1;

.container {
  padding: $base-padding + $padding-percent * 100%;
}
```

Functions

```scss
// Define a variable for a decimal value
$decimal: 0.75;

// Usage of percentage() function
$percentage_value: percentage($decimal); // Converts decimal to percentage

// Usage of round(), ceil(), and floor() functions
$rounded_value: round(3.14); // Rounds to the nearest integer
$ceiled_value: ceil(3.14);   // Rounds up to the nearest integer
$floored_value: floor(3.14); // Rounds down to the nearest integer

// Output the results
.result {
  percentage-value: $percentage_value;
  rounded-value: $rounded_value;
  ceiled-value: $ceiled_value;
  floored-value: $floored_value;
}
```

■ Control Directives `@if @else`

The `@if` and `@else` directives in Sass provide conditional logic for applying styles based on specified conditions

```
$use-dark-theme: true;
.navbar {
  @if $use-dark-theme {
    background-color: #333;
    color: #fff;
  }
  @else {
    background-color: #fff;
    color: #333;
  }
}
```

■ Control Directives `@for`

The `@for` directive in Sass allows you to loop over a range of values and generate styles accordingly

```
@for $i from 1 through 3 {
  .item-#{$i} {
    width: 100px * $i;
  }
}
```

■ Control Directives `@each

The `@each` directive in Sass allows you to loop over each item in a list or map and apply styles accordingly

```
$colors: red, green, blue;

@each $color in $colors {
  .text-#{$color} {
    color: $color;
  }
}
```

■ Sass Function

```
@function double($number) {
        @return $number * 2;
```

```
}

.box {
        width: double(100px);
        height: double(50px);
}
```