`SASS (Syntactically Awesome Style Sheets) :`

- It is a **CSS preprocessor** that extends the capabilities of standard CSS
- It introduces features like variables, nested rules, mixins, functions, and more, making CSS more maintainable, reusable, and easier to write
- **Different Between SASS and SCSS :**
  - .scss (Sassy CSS): Uses a syntax similar to CSS with curly braces and semicolons
  - .sass: An older, indentation-based syntax (no braces or semicolons)

`Partials and Importing :`

- They allow to modularize stylesheets
- **What are Partials ?**
  - files that contain small chunks of CSS code
  - They are meant to be imported into other SASS files
  - By convention, partial filenames begin with an underscore `_` This tells SASS that the file is a partial and should not be compiled into a separate CSS file
- **What is Importing :**
  - `@import` & `@use` directive allows to include the content of one SASS file into another
  - When you import a partial, you don't need to include the underscore or file extension in the import statement
  - SASS combines all imported files into a single CSS file during compilation
- `Example :`
  - **Creating Partial Files :**

```scss
// _variables.scss
$primary-color: red;
$secondary-color: green;
$font-stack: Arial, sans-serif;
```

```scss
// _buttons.scss
.btn {
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;

    &-primary {
        background-color: $primary-color;
        color: white;
    }

    &-secondary {
        background-color: $secondary-color;
        color: white;
    }
}

// _mixins.scss
@mixin flex-center {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

- **Import Partials into a Main File :**

```scss
// main.scss
@use "variables";
@use "mixins"
@use "buttons";

body {
    font-family: $font-stack;
    background-color: lighten($primary-color, 40%);
}

.container {
    @include flex-center;
    height: 100vh;
}
```

- **Compile the Main File :**
    - When you compile main.scss, SASS will combine all the imported partials into a single CSS file
    - **Compiled `main.css` :**

```
.btn {
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
    .btn-primary {
    background-color: #3498db;
    color: white;
}
.btn-secondary {
    background-color: #2ecc71;
    color: white;
}

body {
    font-family: Arial, sans-serif;
    background-color: #d6eaf8;
}

.container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
```

`Variables :`

- powerful feature that allows to store and reuse values throughout stylesheets
- especially useful for values like colors, font stacks, spacing, and other reusable properties
- **Syntax :**
  - Variables in SASS start with a dollar sign `$`

```
$variable-name: value;
```

- **Types of Values You Can Store in Variables :**
  - **Colors :** Store hex, RGB, RGBA, HSL, or named colors

```
$primary-color: #3498db;
$secondary-color: rgba(46, 204, 113, 0.8);
```

- **Numbers :** Store integers, decimals, or units (px - %)

```
$base-spacing: 16px;
$container-width: 1200px;
```

- **Strings :** Store text values, such as font names

```
$font-stack: "Helvetica Neue", sans-serif;
```

- **Lists :** Store multiple values in a list (comma-separated or space-separated)
`scss       $padding-values: 10px 20px 30px 40px; $font-sizes: 12px, 14px, 16px, 18px;`

- **Maps :** Store key-value pairs (similar to JavaScript objects)

```
$theme-colors: (
    "primary": #3498db,
    "secondary": #2ecc71,
    "error": #e74c3c,
);
```

`Nesting & Parent Element :`

- **`What is Nesting :`**

  - Nesting allows to write CSS rules inside other rules, reflecting the relationship between parent and child elements in your HTML

  - **Example One:**

```
// SCSS File
nav {
    background-color: #333;

    ul {
        list-style: none;
        padding: 0;

        li {
            display: inline-block;
            margin-right: 10px;

            a {
                color: white;
                text-decoration: none;
            }
        }
    }
}

// Compiled CSS File
nav {
    background-color: #333;
}

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

nav ul li {
    display: inline-block;
    margin-right: 10px;
}

nav ul li a {
    color: white;
    text-decoration: none;
}
```

- **Example Two :**

```
    // SCSS File
    .parent {
       > .child {
           font-size: 20px;
       }
       .child-two {
           font-weight: bold;
       }
       + p {
           color: #ddd;
       }
    }

    // Compiled CSS File
    .parent > child {
        font-size: 20px;
    }
    .parent .child-two {
        font-weight: bold;
    }
    .parent + p {
        color: #ddd;
    }
```

- **`What is the Parent Selector ?`**
  - The `&` symbol refers to the parent selector. It allows you to create more complex and dynamic styles by combining or modifying the parent selector
  - **Example :**

```
// SCSS File
.button {
    background-color: #3498db;
    color: white;
    padding: 10px 20px;

    &:hover {
        background-color: darken(#3498db, 10%);
    }
}

// Compiled CSS File
.button {
    background-color: #3498db;
    color: white;
    padding: 10px 20px;
}

.button:hover {
    background-color: #217dbb;
}
```

  - **Example Three :**

```
//SCSS File
.box {
    font-size: 20px;

    &-holder {
        width: 1000px;

        &-content {
            margin: auto;
        }
    }
}

// Compiled CSS File
.box {
    font-size: 20px;
}
.box-holder {
    width: 1000px;
}
.box-holder-content {
    margin: auto;
}
```

Placeholder Selectors:

- Placeholder selectors are special SASS selectors that start with `%`
- They are not compiled into CSS unless they are extended using the `@extend` directive
- useful for creating reusable styles without adding extra classes to your HTML
- **Example :**

```
    // SCSS File
    %center-align {
        display: flex;
        justify-content: center;
        align-items: center;
    }

    .header {
        @extend %center-align;
        background-color: #3498db;
    }

    .footer {
        @extend %center-align;
        background-color: #2ecc71;
    }

    // Compiled CSS File
    .header, .footer {
        display: flex;
        justify-content: center;
        align-items: center;
    }

    .header {
        background-color: #3498db;
    }

    .footer {
        background-color: #2ecc71;
    }
```

Control Flow (conditional logic) :

- **`@if` Directive :**

    - evaluates a condition. If the condition is true, the associated block of styles is applied

    ```
    @if condition {
        // Styles to apply if the condition is true
    }
    ```

- **`@else` Directive :**

    - used with `@if` to provide an alternative block of styles if the condition is false

    ```
    @if condition {
        // Styles if condition is true
    } @else {
        // Styles if condition is false
    }
    ```

- **`@else if` Directive :**

    - chaining Multiple Conditions

    ```
    @if condition1 {
        // Styles if condition1 is true
    } @else if condition2 {
        // Styles if condition2 is true
    } @else {
        // Styles if all conditions are false
    }
    ```

- **Example :**

```
    $theme: "blue";

    body {
        @if $theme == "dark" {
            background-color: black;
            color: white;
```

```
        } @else if $theme == "blue" {
            background-color: blue;
            color: white;
        } @else {
            background-color: white;
            color: black;
        }
    }
```

`Interpolation :`

- allows to dynamically generate selectors, property names, values, or even entire blocks of CSS
- Interpolation is done using the `#{}` syntax. You can use it to insert dynamic values into :
  - Selectors
  - Property Names
  - Values
  - Media Queries
  - Comments
- **Example :**

```
// SCSS File
$prefix: "my";
$type: "button";
$side: "top";
$value: 20px;
$version: "1.0.0";

.#{$prefix}-#{$type} {
    background-color: #3498db;
    color: white;
}

.box {
    margin-#{$side}: $value;
}

/* This is version #{$version} of the stylesheet. */

// Compiled CSS File
.my-button {
    background-color: #3498db;
```

```
        color: white;
    }

    .box {
        margin-top: 20px;
    }

    /* This is version 1.0.0 of the stylesheet. */
```

`Comments :`

- used to add notes, explanations, or documentation to code
- **Types Of Comments :**

  1) **Single-Line Comments :**

     – start with `//` . They are not included in the compiled CSS

  2) **Multi-Line Comments :**

     – wrapped in `/* */` . They are included in the compiled CSS by default.

  3) **Silent Comments :**

     – To prevent multi-line comments from appearing in the compiled CSS, add an exclamation mark `!` after the opening `/*`

`Mixin :`

- allows to reuse blocks of styles across stylesheet, making code DRY (Don't Repeat Yourself)
- To Define a Mixin We Use `@mixin` directive

```
    @mixin mixin-name {
        // Styles
    }

    // Including a Mixin
    selector {
       @include mixin-name;
    }
```

- **Example :**

```
// SCSS Code
@mixin centering {
   display: flex;
   justify-content: center;
   align-center: center;
}

.container {
    @include centering;
    background-color: #3498db;
}

// Compiled CSS File
.container {
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: #3498db;
}
```

- Arguments allow to pass values into a mixin

```
@mixin mixin-name($arg1, $arg2, ...) {
    // Styles using $arg1, $arg2, etc.
}
```

  – **Example :**

```
// SCSS File
@mixin button($bg-color, $text-color) {
    background-color: $bg-color;
    color: $text-color;
    padding: 10px 20px;
    border-radius: 5px;
}

.primary-button {
    @include button(#3498db, white);
}

.secondary-button {
    @include button(#2ecc71, black);
}

// Compiled CSS File
.primary-button {
    background-color: #3498db;
```

```
        color: white;
        padding: 10px 20px;
        border-radius: 5px;
    }

    .secondary-button {
        background-color: #2ecc71;
        color: black;
        padding: 10px 20px;
        border-radius: 5px;
    }
```

- We Can Combine multiple mixins for more complex styles

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

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

.container {
    @include flex-center;
    @include box(200px, 200px);
    background-color: #3498db;
}

// Compiled CSS File
.container {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 200px;
    height: 200px;
    background-color: #3498db;
}
```

- **Content Blocks :**
  - We Use `@content` to pass additional styles into a mixin

```
// SCSS File
@mixin hover-effect {
    &:hover {
        @content;
    }
}

.button {
    @include hover-effect {
        background-color: darken(#3498db, 10%);
        transform: scale(1.05);
    }
}

// Compiled CSS File
.button:hover {
    background-color: #217dbb;
    transform: scale(1.05);
}
```

- **Responsive Mixin :**

```
    @mixin respond-to($breakpoint) {
        @media (min-width: $breakpoint) {
                @content;
            }
       }

    .container {
        width: 100%;

        @include respond-to(768px) {
            width: 750px;
        }

        @include respond-to(992px) {
            width: 970px;
        }
    }

    // Compiled CSS File
    .container {
        width: 100%;
    }

    @media (min-width: 768px) {
     .container {
        width: 750px;
     }
    }

    @media (min-width: 992px) {
     .container {
        width: 970px;
     }
    }
```

Loops :

- SASS supports three types of loops :
  - **`@for` Loop :**
    - iterates over a range of values. It's useful for generating styles based on a sequence of numbers
    - **Syntax :**

```
@for $var from <start> through <end> {
    // Styles to repeat
}

// $var: The loop variable
// <start>: The starting value
// <end>: The ending value.
// through: Includes the end value
// to: Excludes the end value
```

- **Example :**

```
// SCSS File
@for $i from 1 through 4 {
    .col-#{$i} {
        width: percentage($i / 12);
    }
}

// Compiled CSS File
.col-1 {
    width: 8.33333%;
}

.col-2 {
    width: 16.66667%;
}
.col-3 {
    width: 25%;
}
```

```
.col-4 {
    width: 33.333333%;
}
```

– `@each` **Loop :**

- Iterates over a list or map. It's useful for generating styles based on a collection of values

- **Syntax :**

```
@each $var in <list> {
    // Styles to repeat
}

// $var: The loop variable.
// <list>: A list or map of values
```

- **Example One :**

```
// SCSS File
$colors: red, green, blue;

@each $color in $colors {
    .bg-#{$color} {
        background-color: $color;
    }
}

// Compiled CSS File
.bg-red {
    background-color: red;
}

.bg-green {
    background-color: green;
}

.bg-blue {
    background-color: blue;
}
```

- **Example Two :**

```
$theme-colors: (
    "primary": #3498db,
    "secondary": #2ecc71,
    "error": #e74c3c,
);

@each $name, $color in $theme-colors {
    .btn-#{$name} {
        background-color: $color;
        color: white;
    }
}

// Compiled CSS File
btn-primary {
    background-color: #3498db;
    color: white;
}

.btn-secondary {
    background-color: #2ecc71;
    color: white;
}

.btn-error {
    background-color: #e74c3c;
    color: white;
}
```

- **`@while` Loop :**
    - repeats styles as long as a condition is true
    - **Syntax :**

```
@while <condition> {
    // Styles to repeat
}
```

- **Example :**

```
// SCSS File
$i: 1;

@while $i <= 3 {
    .spacing-#{$i} {
        margin: #{$i * 10}px;
    }
    $i: $i + 1;
}

// Compiled CSS File
.spacing-1 {
        margin: 10px;
    }

    .spacing-2 {
        margin: 20px;
    }

    .spacing-3 {
        margin: 30px;
    }
```

Function :

- reusable blocks of code that return a value , allow to perform calculations, manipulate data, and generate dynamic styles
- **Syntax :**

```
@function function-name($arg1, $arg2, ...) {
    // Logic goes here
    @return value;
}
```

  - Call a function by its name and pass arguments (if required)

```
selector {
    property: function-name(arg1, arg2);
}
```

- **Example :**

```
// SCSS File
@function calculate-width($columns, $total-columns) {
    @return percentage($columns / $total-columns);
}

.container {
    width: calculate-width(3, 12);
}

.container {
    width: 25%;
}
```