SASS Documentation

You can create Sass files as you create for CSS, but now make sure to put the right extension **.css => .scss**

SASS Comments

You can create Sass comments as we do in JavaScript.

```
// This is a Sass comment.
```

SASS variable

- Variables in Sass are just like properties where you store values to use them again somewhere on your code by calling that variable name.
- Variables start with a dollar sign $ followed by the variable name.

E.g.

```
$main-fonts: Arial, sans-serif;
$headings-color: green;
//To use variables:
h1 {
    font-family: $main-fonts;
    color: $headings-color;
}
```

Note: *CSS has variables of its own, which are totally different from Sass variables.*

CSS Variable

- Property names that are prefixed with --, like --example-name, represent *custom properties* that contain a value that can be used in other declarations using the var() function.

E.g.

```
:root {
  --first-color: #16f;
  --second-color: #ff7;
}

#firstParagraph {
```

```
  background-color: var(--first-color);
  color: var(--second-color);
}
```

Differences:

- CSS variables can have different values for different elements, but Sass variables only have one value at a time.
- Sass variables are *imperative*, which means if you use a variable and then change its value, the earlier use will stay the same. CSS variables are *declarative*, which means if you change the value, it'll affect both earlier uses and later use

To know more about the SASS variable, refer to this link.

SASS Nesting:

- Sass allows the nesting of CSS rules, which is a useful way of organizing a style sheet. It lets you nest your elements inside other elements in CSS.

  E.g.

```
nav {
        background-color: red;

        ul {
                list-style: none;
                li {
                        display: inline-block;
                }
        }
}
```

  Notice that the style will apply to only the elements that are inside that specific nav in your HTML.

SASS Mixin

- In Sass, a mixin is a group of CSS declarations that can be reused throughout the style sheet. This helps not to repeat the same code over and over again.

- You can use mixins in Sass with the keyword mixin. It acts like a Javascript function, where you can put a piece of code and use it everywhere in your file.

E.g.

```
@mixin reset-list {
  margin: 0;
  padding: 0;
  list-style: none;
}
@mixin horizontal-list {
  @include reset-list;
  li {
    display: inline-block;
    margin: {
      left: -2px;
      right: 2em;
    }
  }
}
```

Now, we can call this mixin by using the include directive.

```
nav ul {
  @include horizontal-list;
}
```

- Mixins can also take arguments, which allows their behaviour to be customized each time they're called. The arguments are specified in the @mixin rule after the mixin's name, as a list of variable names surrounded by parentheses.

  E.g.

```
@mixin box-shadow($x, $y, $blur, $c){
        box-shadow: $x $y $blur $c;
}
```

  Have a look at the example below where we called our mixin inside a div element:

```
div {
  @include box-shadow(0px, 0px, 4px, #fff);
}
```

  As you can see, we gave the values for our variables when we called the mixin function.

Use if and else to add logic to your mixin.

- Sass allows the use of if and else statements as we do with normal JavaScript functions. The directive @ if in Sass is useful to test for a specific case.

  E.g.

```
@mixin text-effect($val) {
  @if $val == danger {
    color: red;
  }
  @else if $val == alert {
    color: yellow;
  }
  @else if $val == success {
    color: green;
  }
  @else {
    color: black;
  }
}
```

  Now you can call that mixin inside an element p as an example:

```
p {
 font-size: 18px;
 @include text-effect(danger);
}
// The color will be red.
```

Note: We have to try to cover important concepts for further understanding. You can refer SASS official doc.