Preprocessors: sass, scss, Stylus
Preprocessors: sass, scss

Now that you have learned about different animation effects, let's explore the topic of preprocessors which can make the process of creating them easier. CSS preprocesses are special compilers used to create a CSS file that can be referenced by an HTML document. They are generally used to reduce the amount of CSS you need to write and allow you to re-use values across multiple rules. This will make re-using animations and effects much easier. And because preprocessors are an extension of CSS they'll help not just in animation but any CSS code. Let's learn a little more about them.

Preprocessors provide audit functionality on top of the CSS features already present. Some of the features of preprocessors include the option to create variables, loops, and if else statements. Different preprocessors each have their own syntax and configurations for adding these features. Some of the most commonly used preprocessors include Sass, LESS, Stylus and PostCSS. The use of these preprocessors requires the installation of a compiler on top of your web server.

In the early days of CSS, the main problem developers faced was the difficulty of managing the code. The way CSS was designed made the code very long, messy and complex. It also made it difficult to troubleshoot. Preprocessors have their own scripting language that adds logical structures, automation properties, reusability and bloating of the code. You'll now explore some of the different preprocessors available.

SASS and SCSS

Syntactically Awesome Style Sheets (SASS) is a scripting language that CSS compiles and interprets into CSS. SCSS, which stands for Sassy CSS is the syntax for SASS and can be seen as an advanced version of both SASS and CSS. The difference between SASS and SCSS is best explained by the SASS documentation, which states:

"There are two syntaxes available for Sass. The first, known as SCSS (Sassy CSS) and used throughout this reference, is an extension of the syntax of CSS. This means that every valid CSS stylesheet is a valid SCSS file with the same meaning. This syntax is enhanced with the Sass features described below. Files using this syntax have the .scss extension.

The second and older syntax, known as the indented syntax (or sometimes just "Sass"), provides a more concise way of writing CSS. It uses indentation rather than brackets to indicate the nesting of selectors and newlines rather than semicolons to separate properties. Files using this syntax have the .sass extension."

This example highlights these differences.

Regular CSS:

```
body {
    font: 100% Arial;
    color: lightblue;
}
```

This is the SCSS:

```
$font-stack: Arial;

$primary-color: lightblue;

body {

  font: 100% $font-stack;

  color: $primary-color;

}
```

SASS for the same block:

```
$font-stack: Arial

$primary-color: lightblue

body

  font: 100% $font-stack

  color: $primary-color
```

The variables have been defined at the top with labels such as '$font-stack' and '$primary-color'. This is done with the '$' suffix. The result for both will be the same, and it is not hard to imagine how much time this can save for the developer in complex code blocks where there are a number of occurrences of 'lightblue' color. These variables are placed at the top of the SCSS page.

In the case of SASS, the variation has mainly removed the curly brackets and semi-colons from the code.

The nesting of selectors and separation of properties here is done by means of indentation. You should note that all this syntax is valid and will produce the same output.

For someone familiar with programming concepts, these preprocessors also allow the usage of math and other functions that can be utilized for adding rules conditionally.

Another important functionality in SASS is the use of directives. Let us explore a couple of directives called @mixin and @include.

Syntax

@mixin *name*{ *property*: *value*; *property*: *value*; ... }

```
@mixin some-rules {

   color: lightblue;

   font-size: 25px;

   font-weight: bold;

}
```

```
div {
    @include some-rules;
}
```

There are two directives @mixin and @include, that are used here.

In the first step, you will add properties that you want to reuse inside @mixin.

In the second step, you use the second directive @include and add the mixin identifier that you have created using the @mixin directive.

Similar to these, there are a couple of other directives that are also used. @import allows the import of rules from another file, and @extend allows all the rules from a specific selector to be added inside another selector.

Stylus CSS

Now that you know how preprocessors behave let us explore one more of their type, called Stylus. If you continue to use the example above, the code for Stylus will look like this:

```
body
  font 100% Arial
  color lightblue
```

It is not hard to miss the simplicity of the code without the colons, brackets or semicolons. But you should note that it is still allowed to use all of them in Stylus without any error. Similarly, you can also use '$' or any other symbol before variables, but you are not 'required' to do so.

For someone unfamiliar with programming, functions are a block of self-contained code that consists of steps designed to accomplish and obtain the desired output. The preprocessors, as mentioned, allow the use of functions. Here is an example of this using Stylus.

```
add(a, b)
  a + b
div
   margin add(10px, 20px)
```

What is evident in the code above is that first, you have defined a function called 'add' and passed the variables 'a' and 'b' inside it. You added some functionality inside the function. In this case, you add the two values a and b with the '+' or addition operator. Once you've done that, instead of assigning a value to the 'margin' property, you pass the function add with numeric pixel values passed to it. The output of this code will yield to a form 'margin 30px' after compilation.

These functions are useful when adding color gradation or creating advanced geometric shapes on your web page. There are other features available for preprocessors too. And, just like any programming language, the space of CSS preprocessors is also competitive, and by no means are these the only options available. Once you have gained an understanding of regular CSS, the usage of preprocessors should be explored. The use of preprocessors today is almost inescapable given the number of advanced features they provide which are not available in conventional CSS.