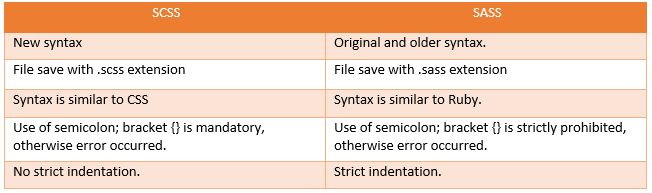
**whats-the-difference-between-scss-and-sass**

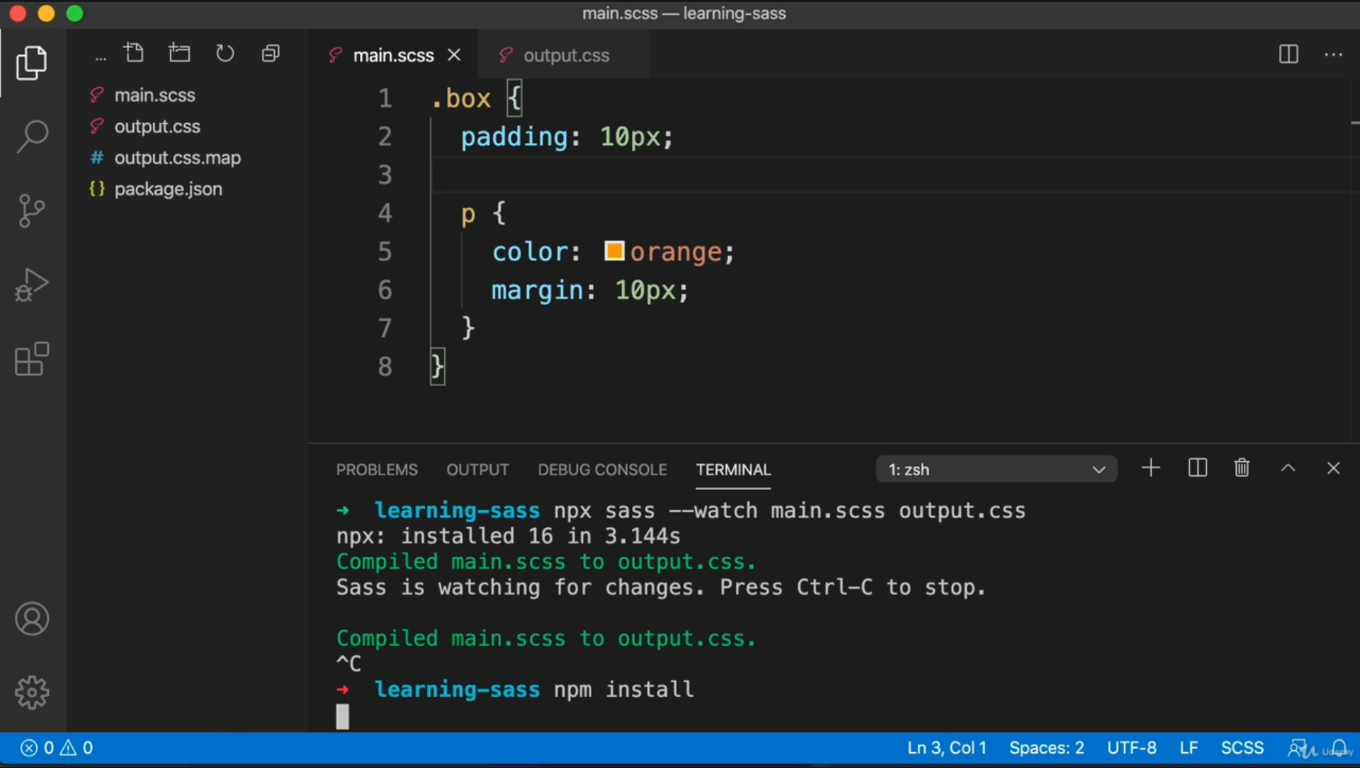
SASS stands for Syntactically Awesome StyleSheets. It is an extension of CSS that adds power and elegance to the basic language. SASS is newly named as SCSS with some chages, but the old one SASS is also there. Before you use SCSS or SASS please see the below difference.

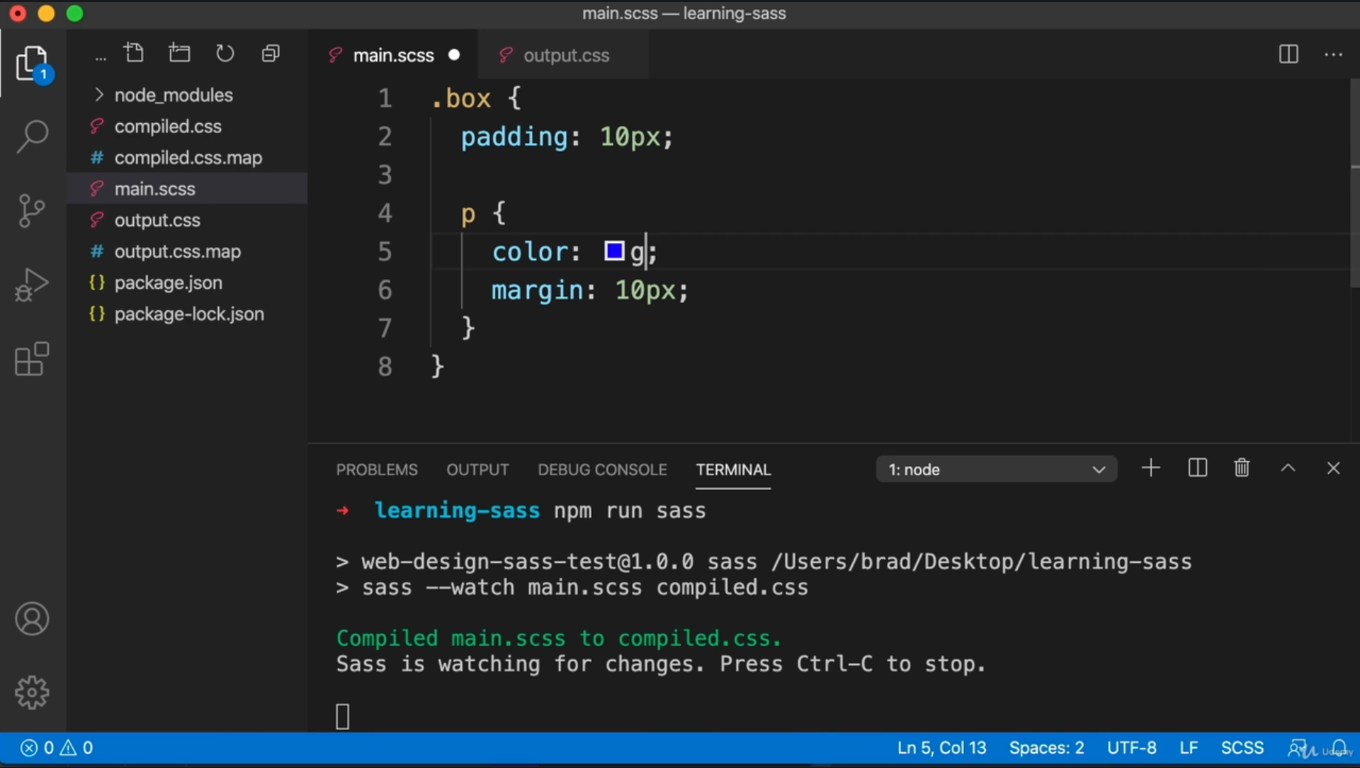
[](https://i.stack.imgur.com/fJUJV.jpg)



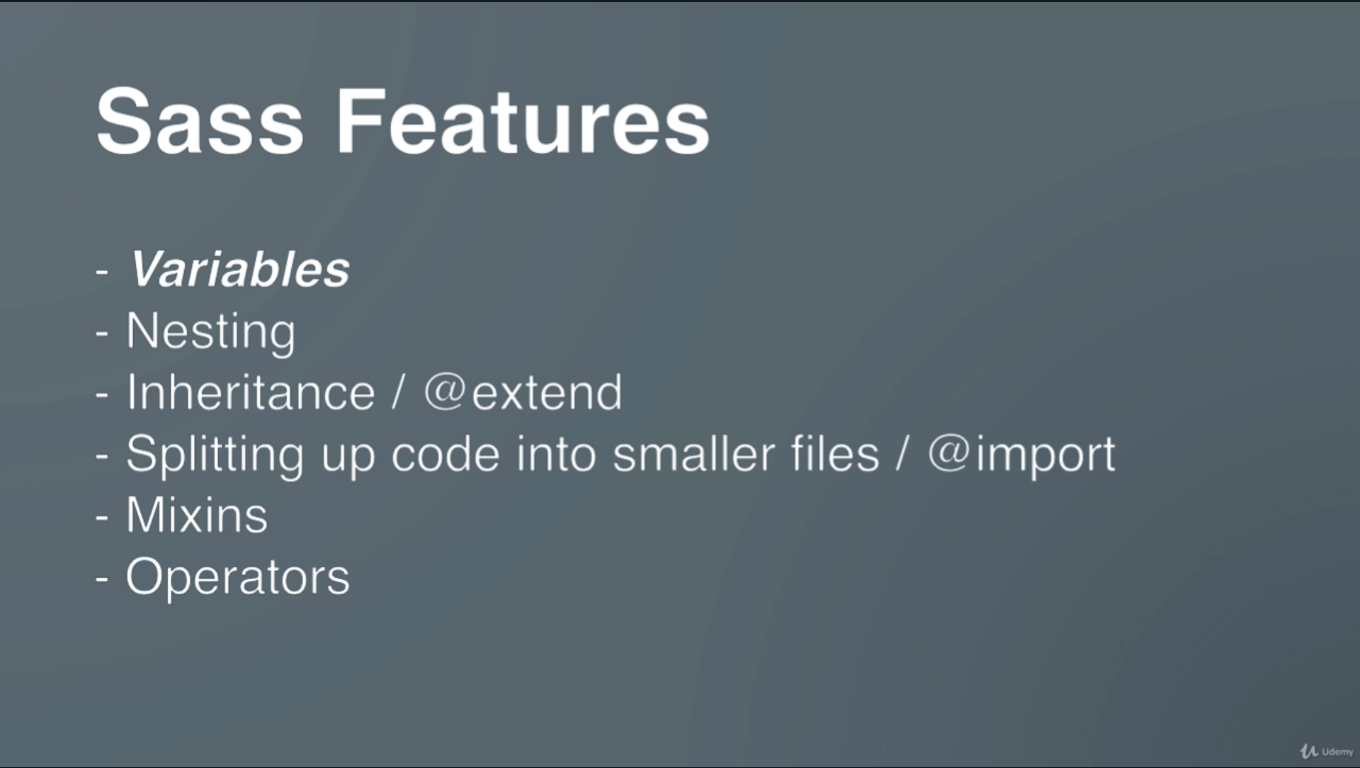
sass --watch source.scss compiled.css

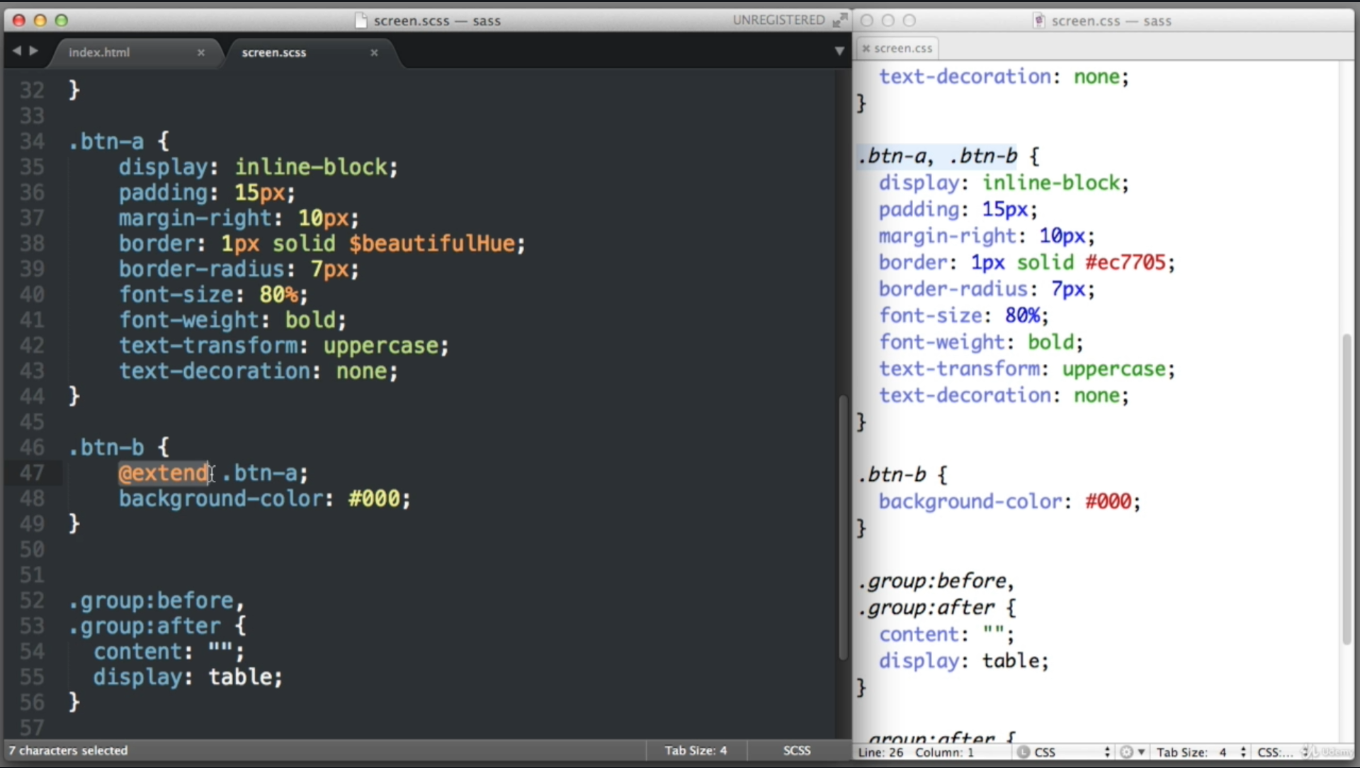


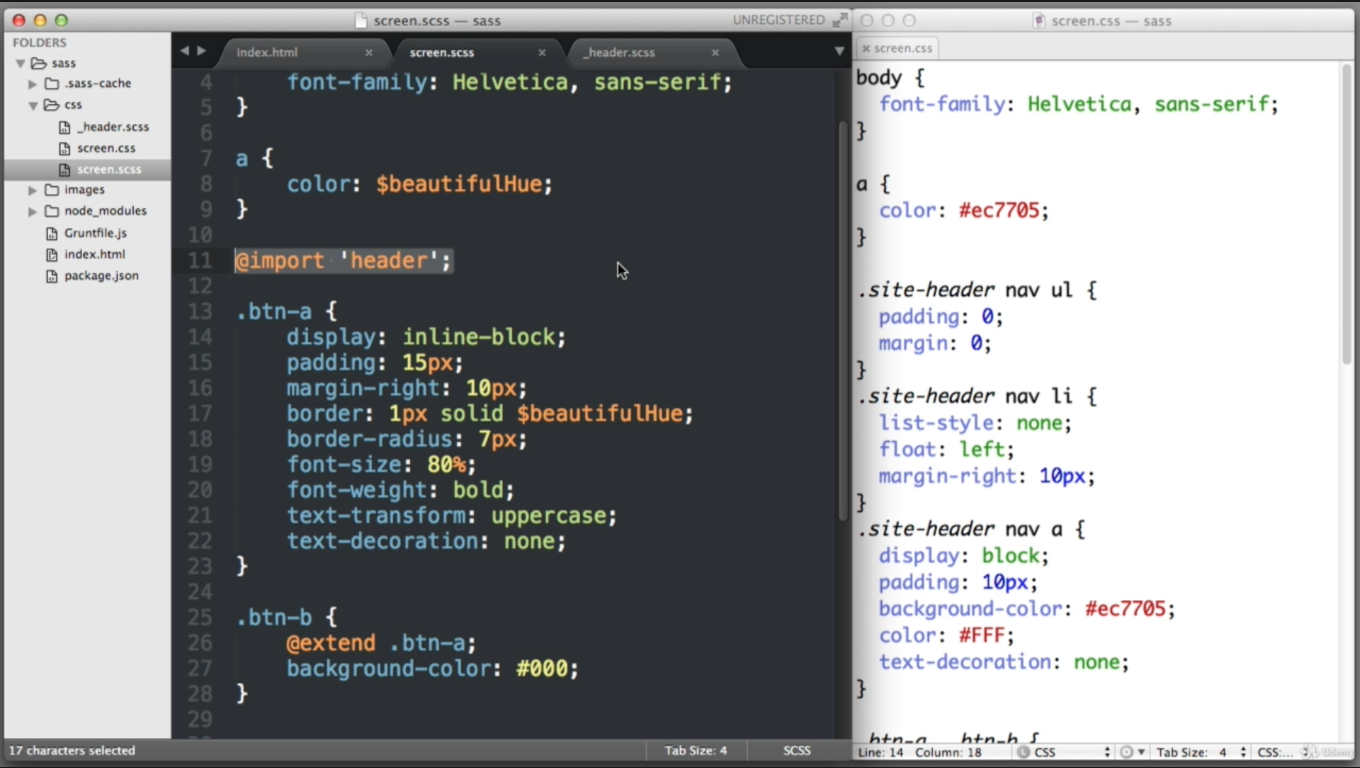


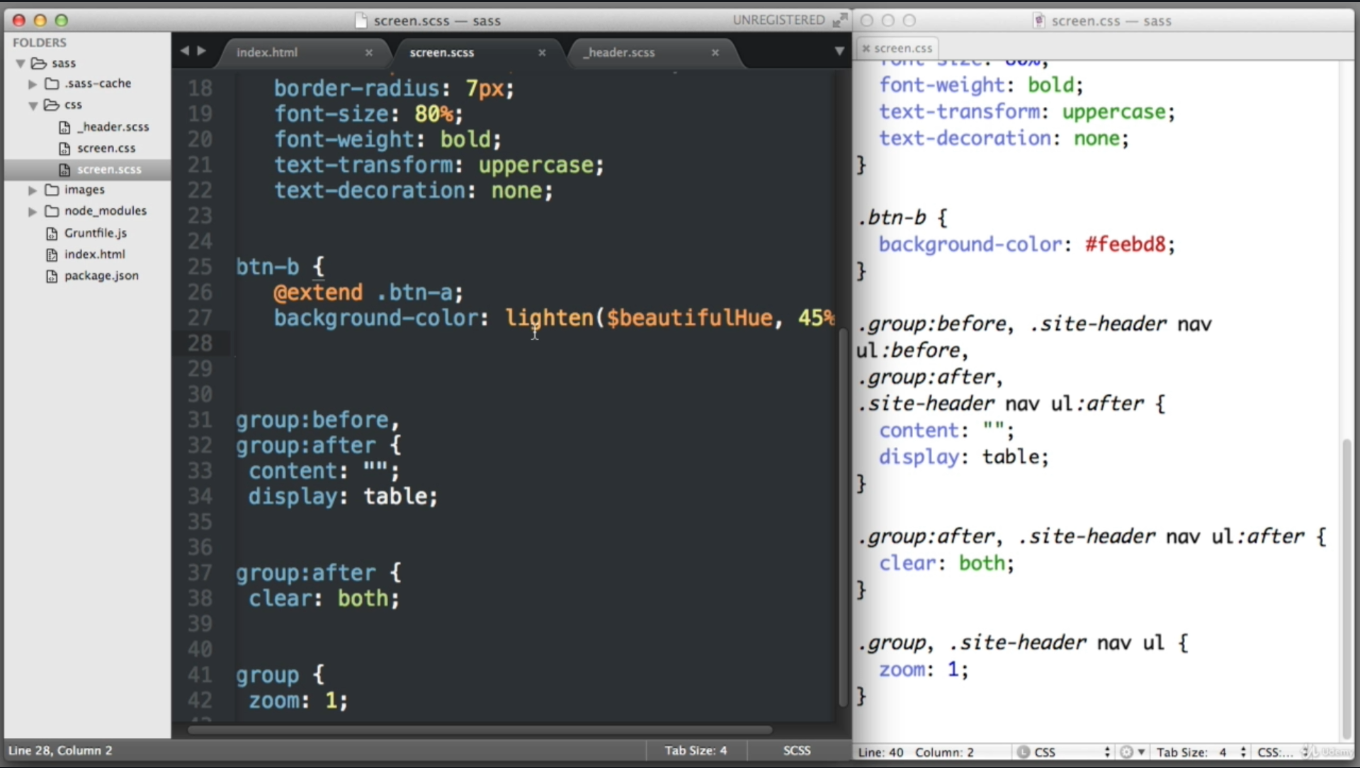


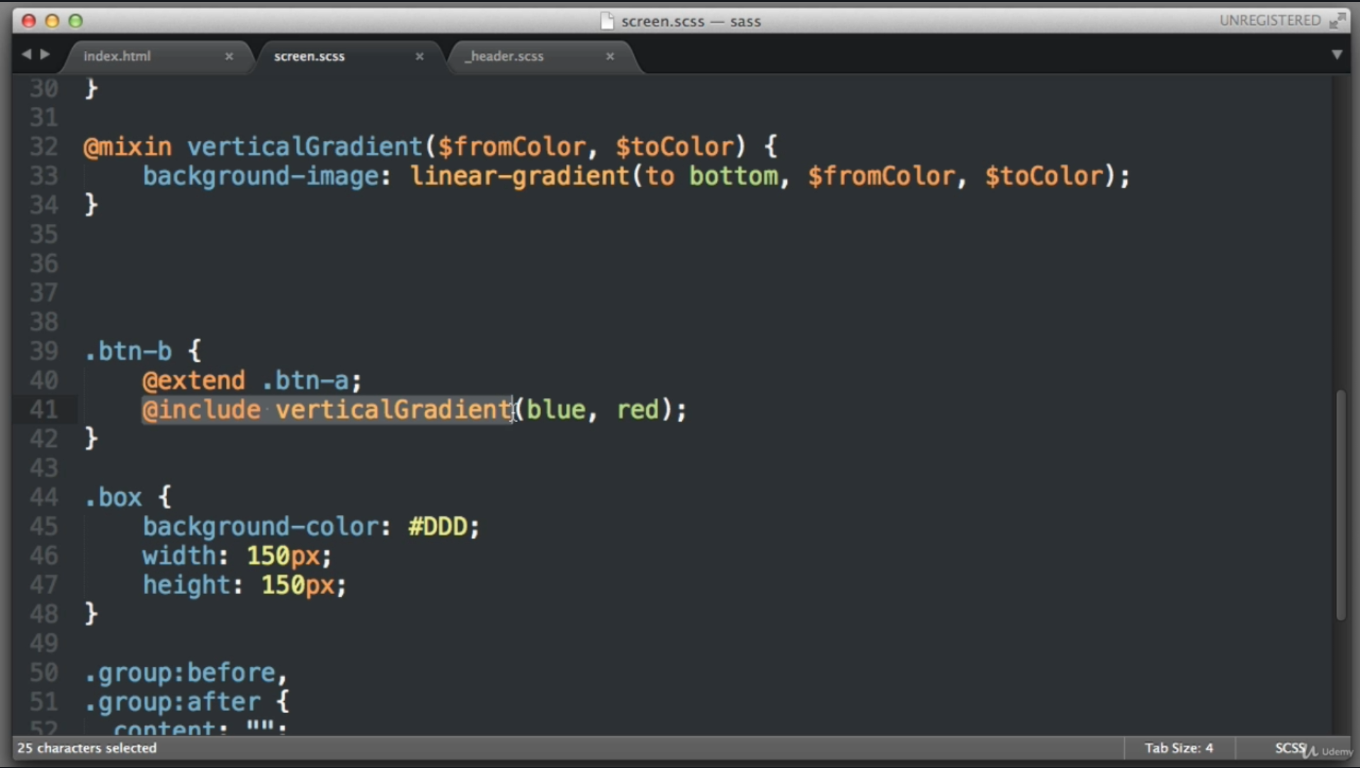
Refer : sass-continued.zip

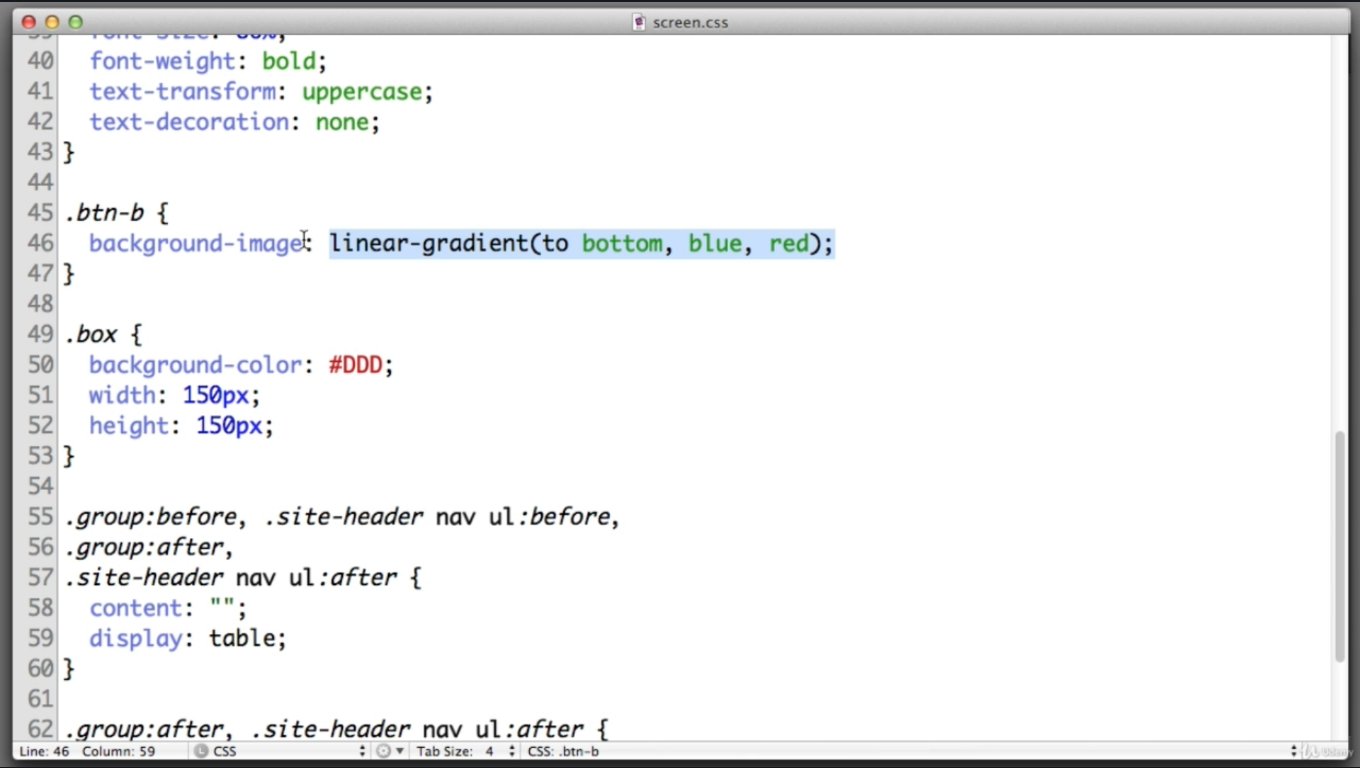


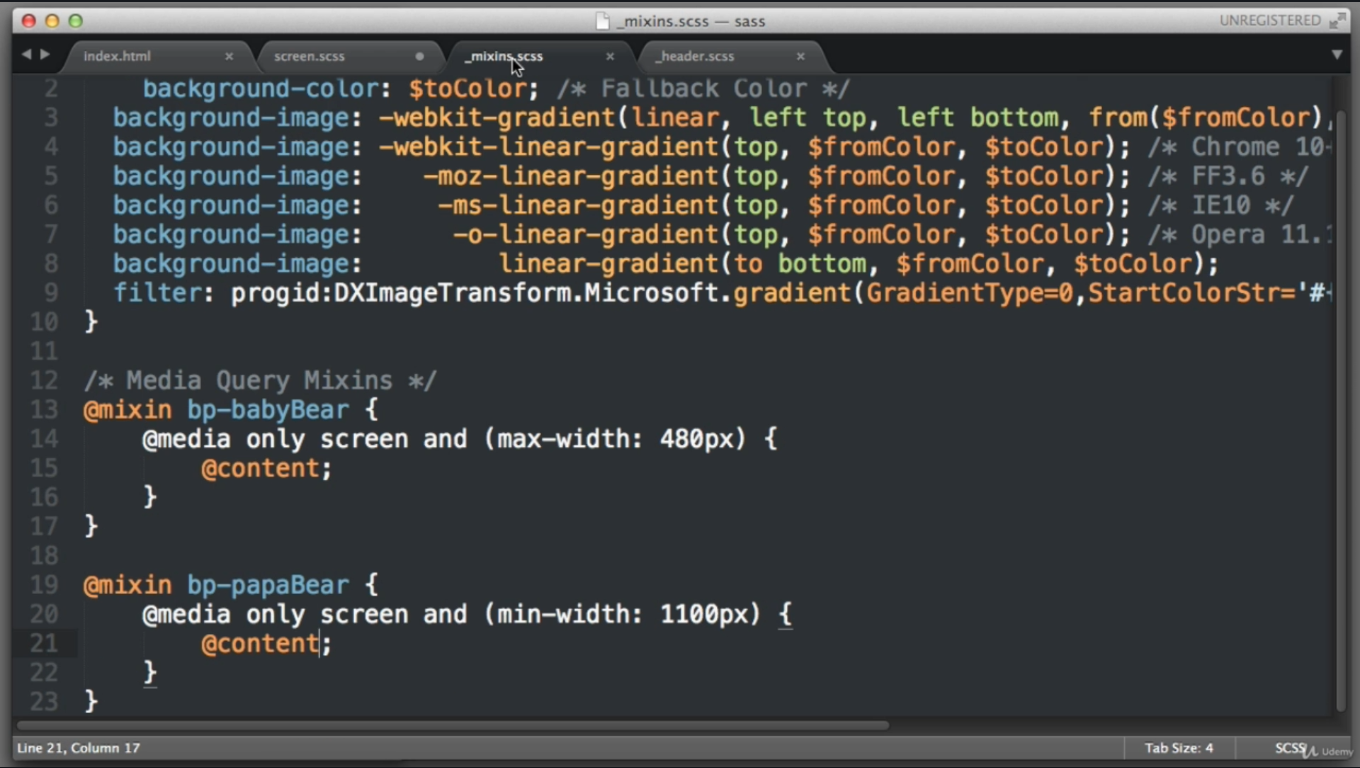


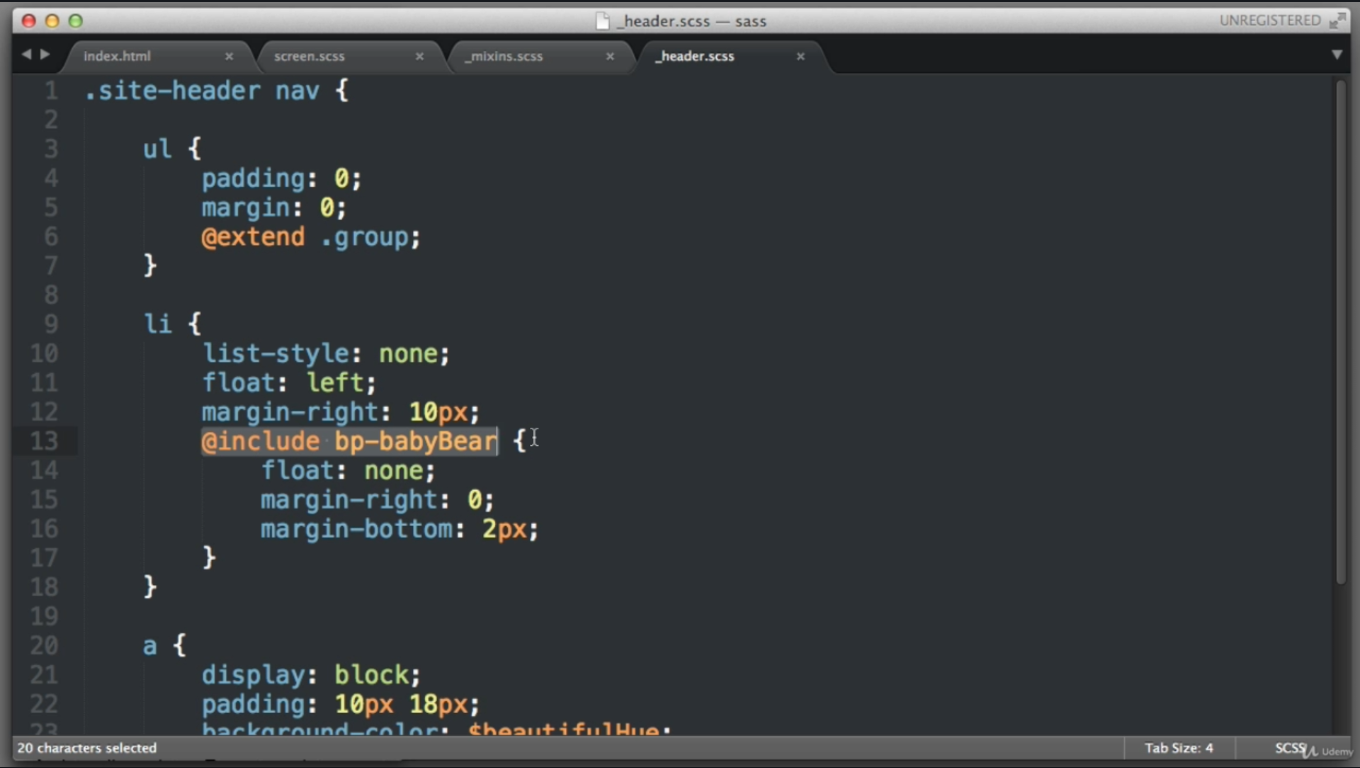


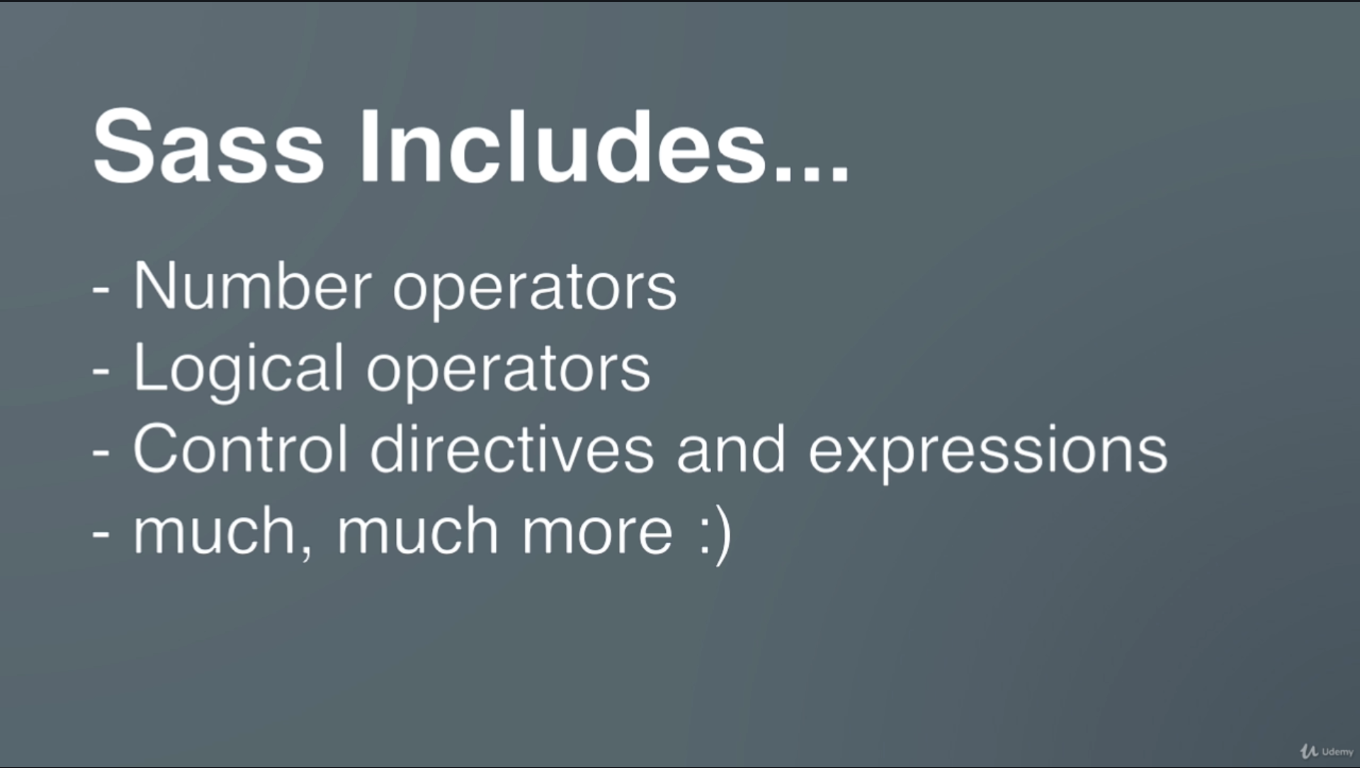












An example of some SCSS and SASS syntax:

**SCSS**

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

font: 100% $font-stack;

color: $primary-color;

}

//Mixins

@mixin transform($property) {

-webkit-transform: $property;

-ms-transform: $property;

transform: $property;

}

.box { @include transform(rotate(30deg)); }

**SASS**

$font-stack: Helvetica, sans-serif

$primary-color: #333

body

font: 100% $font-stack

color: $primary-color

//Mixins

=transform($property)

-webkit-transform: $property

-ms-transform: $property

transform: $property

.box

+transform(rotate(30deg))

**Output CSS after Compilation(Same for Both)**

body {

font: 100% Helvetica, sans-serif;

color: #333;

}

//Mixins

.box {

-webkit-transform: rotate(30deg);

-ms-transform: rotate(30deg);

transform: rotate(30deg);

}

For more guide you can see the official [website](https://sass-lang.com/guide).

Sass has two syntaxes. **The new main syntax (as of Sass 3) is known as “SCSS” (for “Sassy CSS”)**, and is a superset of CSS3’s syntax. This means that every valid CSS3 stylesheet is valid SCSS as well. SCSS files use the extension .scss.

**The second, older syntax is known as the indented syntax (or just “Sass”).** Inspired by Haml’s terseness, it’s intended for people who prefer conciseness over similarity to CSS. Instead of brackets and semicolons, it uses the indentation of lines to specify blocks. Although no longer the primary syntax, the indented syntax will continue to be supported. Files in the indented syntax use the extension .sass.

SASS is an *interpreted* language that spits out CSS. The structure of Sass *looks* like CSS (remotely), but it seems to me that the description is a bit misleading; it's *not* a replacement for CSS, or an extension. It's an interpreter which spits out CSS in the end, so Sass still has the limitations of normal CSS, but it masks them with simple code.

The difference is syntax. Underneath the textual exterior they are identical. This is why sass and scss files can import each other. Actually, Sass has four syntax parsers: scss, sass, CSS, and less. All of these convert a different syntax into an [Abstract Syntax Tree](http://en.wikipedia.org/wiki/Abstract_syntax_tree) which is further processed into CSS output or even onto one of the other formats via the sass-convert tool.

Use the syntax you like the best, both are fully supported and you can change between them later if you change your mind.

The Sass .sass file is visually different from .scss file, e.g.

Example.sass - sass is the older syntax

$color: red

=my-border($color)

border: 1px solid $color

body

background: $color

+my-border(green)

Example.scss - sassy css is the new syntax as of Sass 3

$color: red;

@mixin my-border($color) {

border: 1px solid $color;

}

body {

background: $color;

@include my-border(green);

}

Any valid CSS document can be converted to Sassy CSS (SCSS) simply by changing the extension from .css to .scss.

*Sass* (*Syntactically Awesome StyleSheets*) have two syntaxes:

* a newer: *SCSS* (*Sassy CSS*)
* and an older, original: indent syntax, which is the original *Sass* and is also called *Sass*.

So they are both part of *Sass* preprocessor with two different possible syntaxes.

The most important difference between *SCSS* and original *Sass*:

***SCSS***:

* Syntax is similar to *CSS* (so much that every regular valid *CSS3* is also valid *SCSS*, but the relationship in the other direction obviously does not happen)
* Uses braces {}
* Uses semi-colons ;
* Assignment sign is :
* To create a *mixin* it uses the @mixin directive
* To use *mixin* it precedes it with the @include directive
* Files have the *.scss* extension.

**Original *Sass***:

* Syntax is similar to *Ruby*
* No braces
* No strict indentation
* No semi-colons
* Assignment sign is = instead of :
* To create a mixin it uses the = sign
* To use *mixin* it precedes it with the + sign
* Files have the *.sass* extension.

Some prefer *Sass*, the original syntax - while others prefer *SCSS*. Either way, but it is worth noting that [*Sass’s* indented syntax has not been and will never be deprecated](http://thesassway.com/news/sass-is-here-to-stay).

**Conversions with *sass-convert***:

# Convert Sass to SCSS

$ sass-convert style.sass style.scss

# Convert SCSS to Sass

$ sass-convert style.scss style.sass

[The Sass and SCSS documentation](https://sass-lang.com/documentation)

Sass @import and Partials

Sass keeps the CSS code DRY (Don't Repeat Yourself). One way to write DRY code is to keep related code in separate files.

You can create small files with CSS snippets to include in other Sass files. Examples of such files can be: reset file, variables, colors, fonts, font-sizes, etc.

## **Sass Importing Files**

Just like CSS, Sass also supports the @import directive.

The @import directive allows you to include the content of one file in another.

The CSS @import directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it. However, the Sass @import directive includes the file in the CSS; so no extra HTTP call is required at runtime!

Sass Import Syntax:

@import *filename*;

**Tip:** You do not need to specify a file extension, Sass automatically assumes that you mean a .sass or .scss file. You can also import CSS files. The @import directive imports the file and any variables or mixins defined in the imported file can then be used in the main file.

You can import as many files as you need in the main file:

### Example

@import "variables";  
@import "colors";  
@import "reset";

Let's look at an example: Let's assume we have a reset file called "reset.scss", that looks like this:

### Example

SCSS Syntax (reset.scss):

html,  
body,  
ul,  
ol {  
  margin: 0;  
  padding: 0;  
}

and now we want to import the "reset.scss" file into another file called "standard.scss".

Here is how we do it: It is normal to add the @import directive at the top of a file; this way its content will have a global scope:

SCSS Syntax (standard.scss):

@import "reset";  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: red;  
}

So, when the "standard.css" file is transpiled, the CSS will look like this:

CSS output:

html, body, ul, ol {  
  margin: 0;  
  padding: 0;  
}  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: red;  
}

## **Sass Partials**

By default, Sass transpiles all the .scss files directly. However, when you want to import a file, you do not need the file to be transpiled directly.

Sass has a mechanism for this: If you start the filename with an underscore, Sass will not transpile it. Files named this way are called partials in Sass.

So, a partial Sass file is named with a leading underscore:

Sass Partial Syntax:

 \_*filename*;

The following example shows a partial Sass file named "\_colors.scss". (This file will not be transpiled directly to "colors.css"):

### Example

"\_colors.scss":

$myPink: #EE82EE;  
$myBlue: #4169E1;  
$myGreen: #8FBC8F;

Now, if you import the partial file, omit the underscore. Sass understands that it should import the file "\_colors.scss":

### Example

@import "colors";  
  
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 18px;  
  color: $myBlue;  
}

# **Sass @mixin and @include**

## **Sass Mixins**

The @mixin directive lets you create CSS code that is to be reused throughout the website.

The @include directive is created to let you use (include) the mixin.

## **Defining a Mixin**

A mixin is defined with the @mixin directive.

Sass @mixin Syntax:

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

The following example creates a mixin named "important-text":

SCSS Syntax:

@mixin important-text {  
  color: red;  
  font-size: 25px;  
  font-weight: bold;  
  border: 1px solid blue;  
}

**Tip:** A tip on hyphens and underscore in Sass: Hyphens and underscores are considered to be the same. This means that @mixin important-text { } and @mixin important\_text { } are considered as the same mixin!

## **Using a Mixin**

The @include directive is used to include a mixin.

Sass @include mixin Syntax:

selector {  
  @include mixin-name;}

So, to include the important-text mixin created above:

SCSS Syntax:

.danger {  
  @include important-text;  
  background-color: green;  
}

The Sass transpiler will convert the above to normal CSS:

CSS output:

.danger {  
  color: red;  
  font-size: 25px;  
  font-weight: bold;  
  border: 1px solid blue;  
  background-color: green;  
}

[Run Example »](https://www.w3schools.com/sass/showsass.php?filename=demo_sass_mixin1)

A mixin can also include other mixins:

SCSS Syntax:

@mixin special-text {  
  @include important-text;  
  @include link;  
  @include special-border;  
}

## **Passing Variables to a Mixin**

Mixins accept arguments. This way you can pass variables to a mixin.

Here is how to define a mixin with arguments:

SCSS Syntax:

/\* Define mixin with two arguments \*/  
@mixin bordered($color, $width) {  
  border: $width solid $color;  
}  
  
.myArticle {  
  @include bordered(blue, 1px);  // Call mixin with two values  
}  
  
.myNotes {  
  @include bordered(red, 2px); // Call mixin with two values  
}

Notice that the arguments are set as variables and then used as the values (color and width) of the border property.

After compilation, the CSS will look like this:

CSS Output:

.myArticle {  
  border: 1px solid blue;  
}  
  
.myNotes {  
  border: 2px solid red;  
}

[Run Example »](https://www.w3schools.com/sass/showsass.php?filename=demo_sass_mixin2)

## **Default Values for a Mixin**

It is also possible to define default values for mixin variables:

SCSS Syntax:

@mixin bordered($color: blue, $width: 1px) {  
  border: $width solid $color;  
}

Then, you only need to specify the values that change when you include the mixin:

SCSS Syntax:

.myTips {  
  @include bordered($color: orange);  
}

## **Using a Mixin For Vendor Prefixes**

Another good use of a mixin is for vendor prefixes.

Here is an example for transform:

SCSS Syntax:

@mixin transform($property) {  
  -webkit-transform: $property;  
  -ms-transform: $property;  
  transform: $property;  
}  
  
.myBox {  
  @include transform(rotate(20deg));  
}

After compilation, the CSS will look like this:

CSS Output:

.myBox {  
  -webkit-transform: rotate(20deg);  
  -ms-transform: rotate(20deg);  
  transform: rotate(20deg);  
}

# **Sass @extend and Inheritance**

## **Sass @extend Directive**

The @extend directive lets you share a set of CSS properties from one selector to another.

The @extend directive is useful if you have almost identically styled elements that only differ in some small details.

The following Sass example first creates a basic style for buttons (this style will be used for most buttons). Then, we create one style for a "Report" button and one style for a "Submit" button. Both "Report" and "Submit" button inherit all the CSS properties from the .button-basic class, through the @extend directive. In addition, they have their own colors defined:

SCSS Syntax:

.button-basic  {  
  border: none;  
  padding: 15px 30px;  
  text-align: center;  
  font-size: 16px;  
  cursor: pointer;  
}  
  
.button-report  {  
  @extend .button-basic;  
  background-color: red;  
}  
  
.button-submit  {  
  @extend .button-basic;  
  background-color: green;  
  color: white;  
}

After compilation, the CSS will look like this:

CSS Output:

.button-basic, .button-report, .button-submit {  
  border: none;  
  padding: 15px 30px;  
  text-align: center;  
  font-size: 16px;  
  cursor: pointer;  
}  
  
.button-report  {  
  background-color: red;  
}  
  
.button-submit  {  
  background-color: green;  
  color: white;  
}

By using the @extend directive, you do not need to specify several classes for an element in your HTML code, like this: <button class="button-basic button-report">Report this</button>. You just need to specify .button-report to get both sets of styles.

The @extend directive helps keep your Sass code very DRY.