SASS

Syntactically Awesome StyleSheets

Table of Contents

1. Introduction

Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of the Compass style library.

1.1. Features

- Fully CSS-compatible
- Language extensions such as variables, nesting, and mixins
- Many useful functions for manipulating colors and other values
- Advanced features like control directives for libraries
- Well-formatted, customizable output

1.2. Syntax

There are two syntaxes available for Sass:

1. **SCSS** (Sassy CSS) - is an extension of the syntax of CSS. This means that every valid CSS stylesheet is a valid SCSS file with the same meaning. In addition, SCSS understands most CSS hacks and vendor-specific syntax. Files using this syntax have the .scss extension.
2. **SASS** - an 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 nesting of selectors, and newlines rather than semicolons to separate properties. The indented syntax has all the same features, although some of them have slightly different syntax. Files using this syntax have the .sass extension.

Either syntax can import files written in the other.

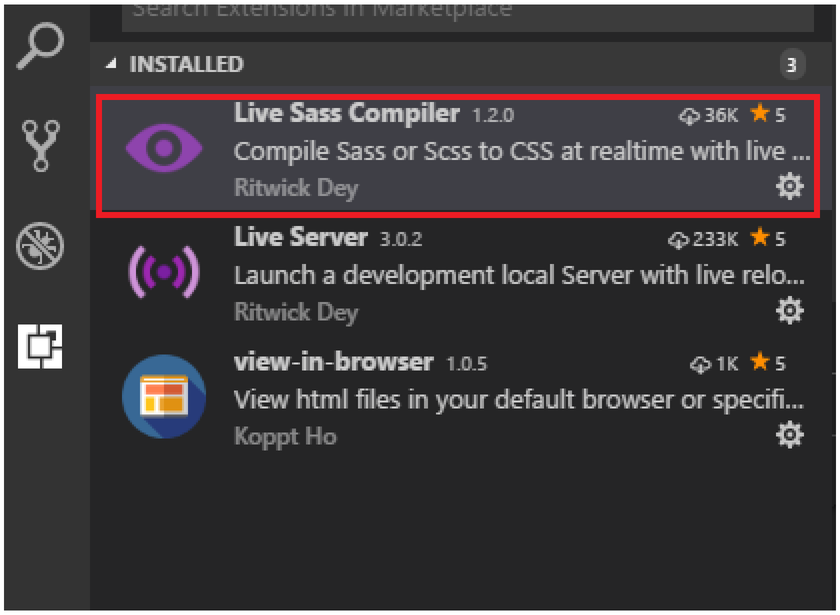
1.3. Using Sass

Sass can be used in three ways:

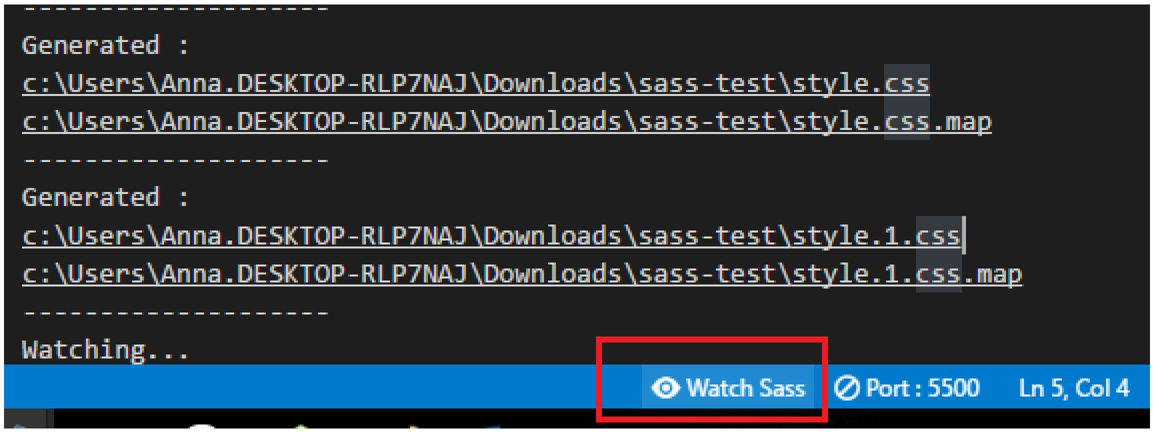
1. as a command-line too
2. as a standalone Ruby module
3. as a plugin for any Rack-enabled framework

1.4. Using Sass with VS code

A VSCode Extension that help you to compile/transpile your SASS/SCSS files to CSS files at realtime with live browser reload is the “Live Sass Compiler”:


Search Extensions in Marketplace
INSTALLED 3
Live Sass Compiler 1.2.0 36K ★5
Compile Sass or Scss to CSS at realtime with live ...
Ritwick Dey
Live Server 3.0.2 233K ★5
Launch a development local Server with live relo...
Ritwick Dey
view-in-browser 1.0.5 1K ★5
View html files in your default browser or specifi...
Koppt Ho

After you added this extension, you will have this icon in your vs code dashboard:



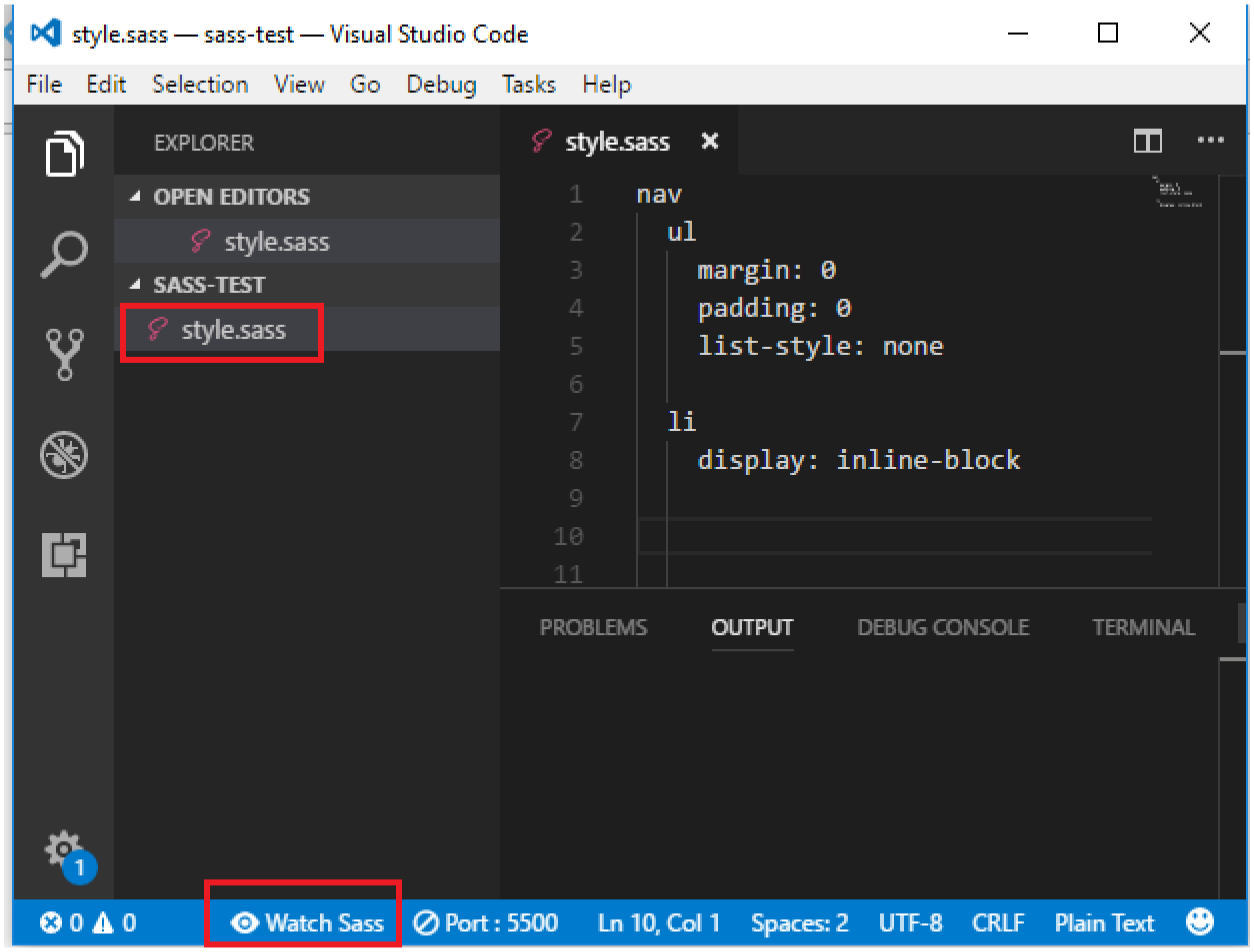
```
--------------------
Generated :
c:\Users\Anna.DESKTOP-RLP7NAJ\Downloads\sass-test\style.css
c:\Users\Anna.DESKTOP-RLP7NAJ\Downloads\sass-test\style.css.map
--------------------
Generated :
c:\Users\Anna.DESKTOP-RLP7NAJ\Downloads\sass-test\style.1.css
c:\Users\Anna.DESKTOP-RLP7NAJ\Downloads\sass-test\style.1.css.map
--------------------
Watching...
```

Usage/Shortcuts:

1. Click to Watch Sass from Statusbar to turn on the live compilation and then click to Stop Watching Sass from Statusbar to turn on live compilation .
2. Press F1 or ctrl+shift+P and type Live Sass: Watch Sass to start live compilation or, type Live Sass: Stop Watching Sass to stop a live compilation.
3. Press F1 or ctrl+shift+P and type Live Sass: Compile Sass - Without Watch Mode to compile Sass or Scss for one time.

<u>Full sass compiling operation:</u>

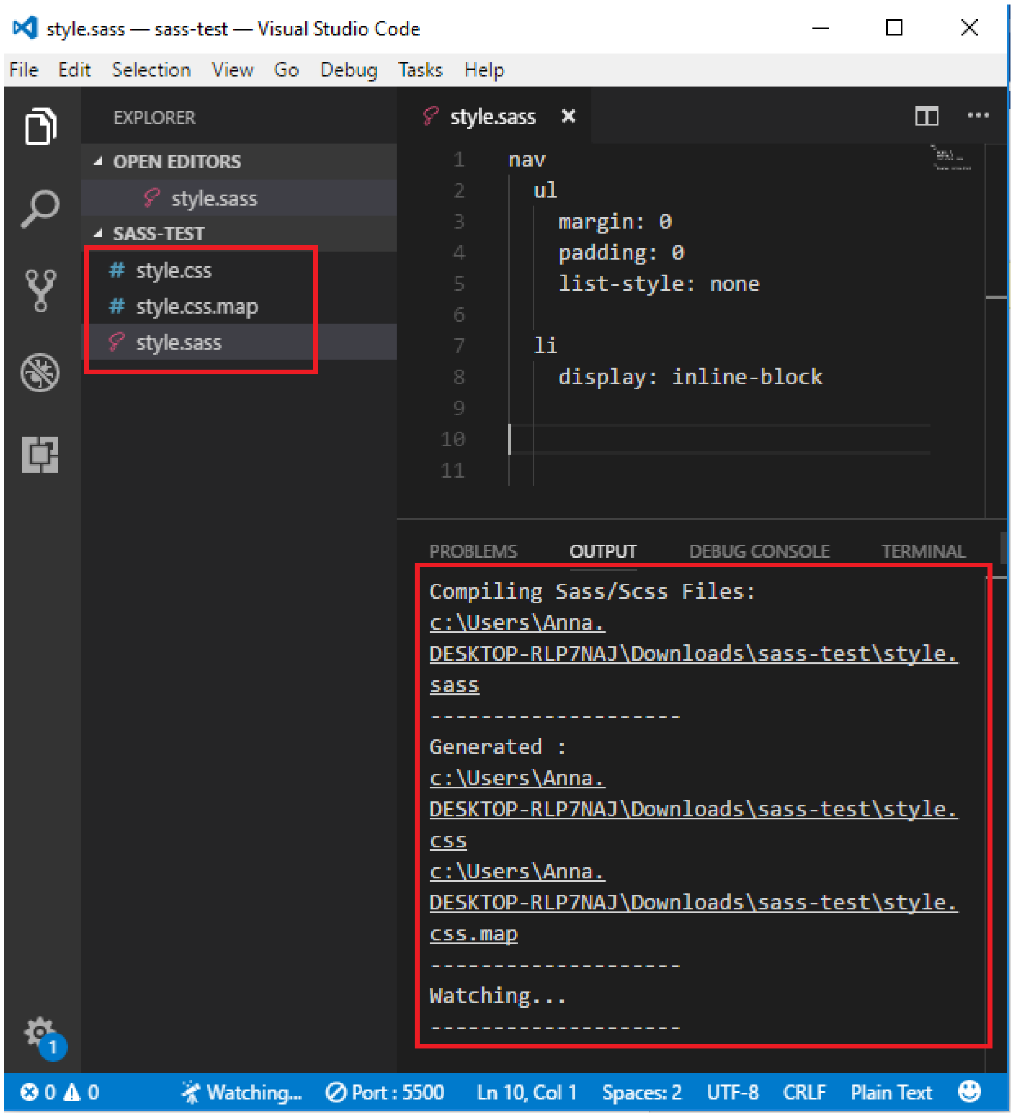
Step 1 -


style.sass — sass-test — Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
style.sass
SASS-TEST
style.sass
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
0 0 Watch Sass Port : 5500 Ln 10, Col 1 Spaces: 2 UTF-8 CRLF Plain Text

```
nav
  ul
    margin: 0
    padding: 0
    list-style: none

  li
    display: inline-block
```

Step 2-


style.sass — sass-test — Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
style.sass
SASS-TEST
style.css
style.css.map
style.sass
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
0 0 Watching... Port : 5500 Ln 10, Col 1 Spaces: 2 UTF-8 CRLF Plain Text

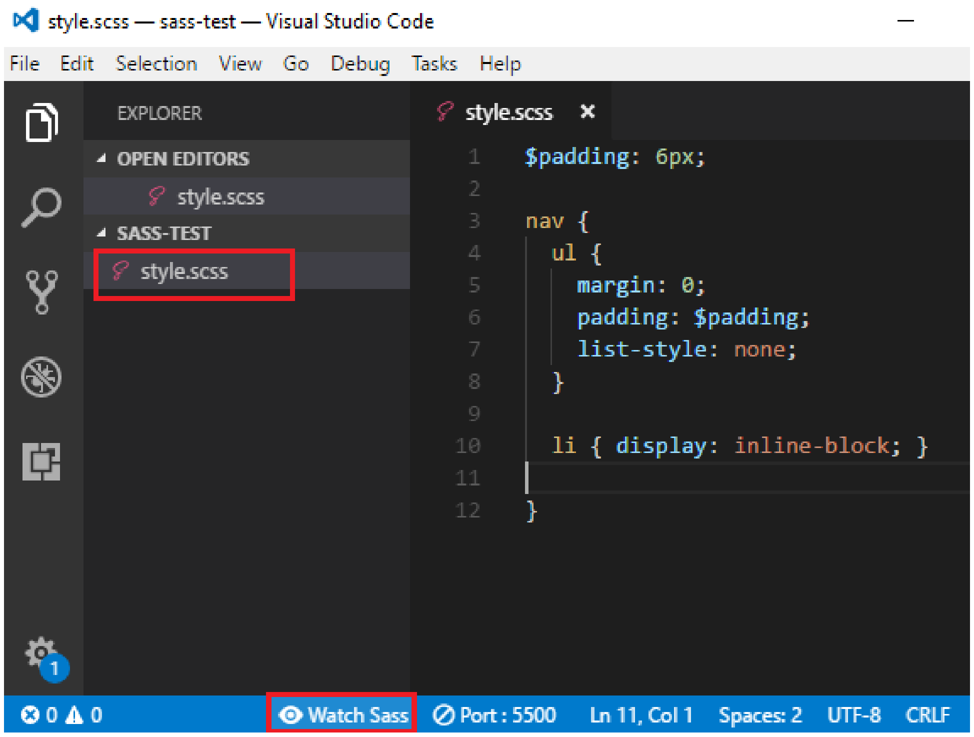
```
nav
  ul
    margin: 0
    padding: 0
    list-style: none

  li
    display: inline-block
```

```
Compiling Sass/Scss Files:
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.
sass
--------------------
Generated :
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.
css
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.
css.map
--------------------
Watching...
--------------------
```

Full scss compiling operation:

Step 1:



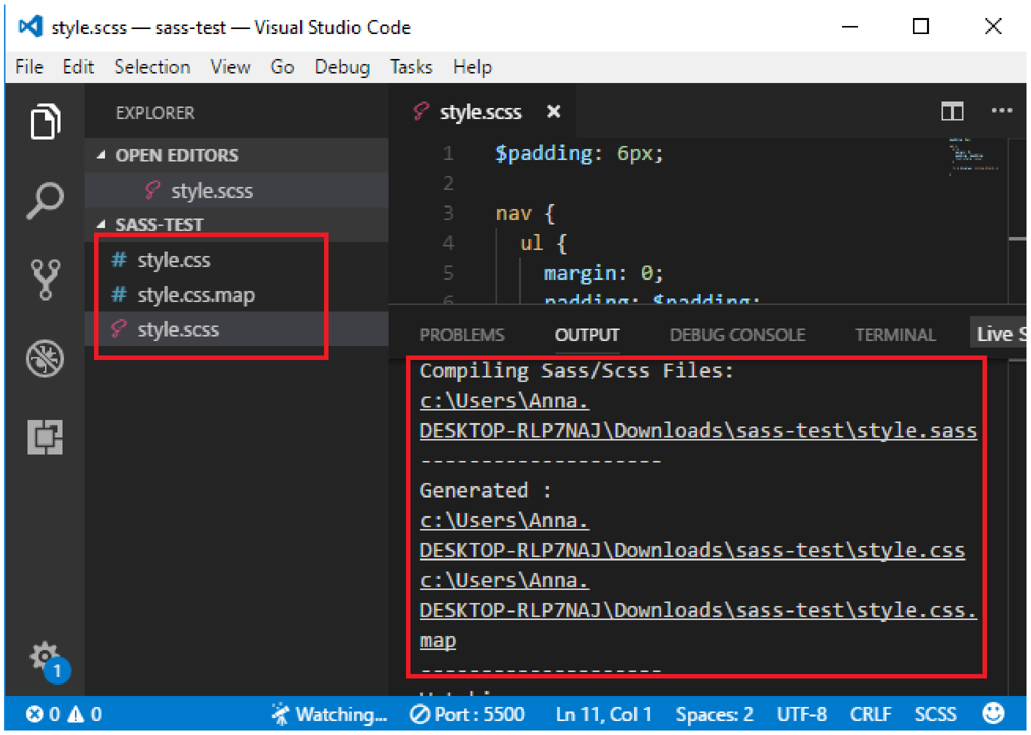
```
$padding: 6px;

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

  li { display: inline-block; }

}
```

Step 2:



```
$padding: 6px;

nav {
  ul {
    margin: 0;
```

```
Compiling Sass/Scss Files:
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.sass
--------------------
Generated :
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.css
c:\Users\Anna.
DESKTOP-RLP7NAJ\Downloads\sass-test\style.css.
map
--------------------
```

2. CSS Extensions

2.1. Nested Rules

Sass allows CSS rules to be nested within one another. The inner rule then only applies within the outer rule's selector. For example:

SCSS

```scss
#main {
  width: 97%;

  p, div {
    font-size: 2em;
    a { font-weight: bold; }
  }

  span { font-size: 3em; }
}
```

Compiled CSS

```css
#main {
  width: 97%;
}

#main p, #main div {
  font-size: 2em;
}

#main p a, #main div a {
  font-weight: bold;
}

#main span {
  font-size: 3em;
}
/*# sourceMappingURL=style.css.map */
```

This helps avoid repetition of parent selectors, and makes complex CSS layouts with lots of nested selectors much simpler.

2.2. Referencing Parent Selectors: &

Sometimes it's useful to use a nested rule's parent selector in other ways than the default. For instance, you might want to have special styles for when that selector is hovered over or for when the body element has a certain class. In these cases, you can explicitly specify where the parent selector should be inserted using the & character.

& will be replaced with the parent selector as it appears in the CSS. This means that if you have a deeply nested rule, the parent selector will be fully resolved before the & is replaced. For example:

SCSS

```scss
#main {
  color: blue;
  a{
    font-weight: bold;
    &:hover { color: red; }
  }
}
```

Compiled CSS

```css
#main {
  color: blue;
}

#main a {
  font-weight: bold;
}

#main a:hover {
  color: red;
}
/*# sourceMappingURL=style.css.map */
```

& must appear at the beginning of a compound selector, but it can be followed by a suffix that will be added to the parent selector. For example:

SCSS

```scss
#main {
  color: black;
  &-sidebar { border: 1px solid; }
}
```

Compiled CSS

```css
#main {
  color: black;
}

#main-sidebar {
  border: 1px solid;
}
/*# sourceMappingURL=style.css.map */
```

2.3. Nested Properties

CSS has quite a few properties that are in "namespaces;" for instance, font-family, font-size, and font-weight are all in the font namespace.

In CSS, if you want to set a bunch of properties in the same namespace, you have to type it out each time. Sass provides a shortcut for this: just write the namespace once, then nest each of the sub-properties within it. For example:

SCSS

```scss
.funky1 {
  font: {
    family: fantasy;
    size: 30em;
    weight: bold;
  }
}


.funky2 {
  font: 20px/24px fantasy {
    weight: bold;
  }
}
```

Compiled CSS

```css
.funky1 {
  font-family: fantasy;
  font-size: 30em;
  font-weight: bold;
}

.funky2 {
  font: 20px/24px fantasy;
  font-weight: bold;
}
/*# sourceMappingURL=style.css.map */
```

2.4. Comments: /* */ and //

Sass supports standard multiline CSS comments with /* */, as well as single-line comments with //. The multiline comments are preserved in the CSS output where possible, while the single-line comments are removed. For example:

SCSS

```scss
/* several lines comment
 * since it uses the CSS comment
syntax,
 * it will appear in the CSS output.
 */
 body { color: black; }

 //single-line comment syntax.
 //won't appear in the CSS output
 a { color: green; }
```

Compiled CSS

```css
/* several lines comment
 * since it uses the CSS comment
syntax,
 * it will appear in the CSS output.
 */
body {
  color: black;
}

a {
  color: green;
}
/*# sourceMappingURL=style.css.map */
```

3. SassScript

In addition to the plain CSS property syntax, Sass supports a small set of extensions called SassScript.

SassScript allows properties to use variables, arithmetic, and extra functions.

3.1. Variables: $

The most straightforward way to use SassScript is to use variables.

Variables begin with dollar signs, and are set like CSS properties.

For historical reasons, variable names (and all other Sass identifiers) can use hyphens and underscores interchangeably. For example, if you define a variable called $main-width, you can access it as $main_width, and vice versa.

Variables are only available within the level of nested selectors where they're defined. If they're defined outside of any nested selectors, they're available everywhere. They can also be defined with the !global flag, in which case they're also available everywhere. For example:

SCSS

```scss
$width: 5em;
#main {
  width: $width;
}
```

Compiled CSS

```css
#main {
  height: 5em;
}

#menu {
  width: 8em;
  color: red;
}

#sidebar {
  width: 8em;
}
/*# sourceMappingURL=style.css.map */
```

3.2. SassScript data types

Data Type & Description	Example
Numbers It represents numeric types / size.	1.2 13 10px
Strings It is sequence of characters defined within single or double quotes, or without quotes	‘hello’ "hello" hello
Colors It is used for defining color value.	Red #008000, rgb(25,255,204)
Booleans Conditions that returns true or false.	True false 10 > 9
Nulls It specifies null value (is used as unknown value)	null
lists of values Represents a list of values which are separated by spaces or commas.	1px solid #eeeeee, 0 0 0 1px
Mapping It maps from one value to another value.	(key1: value1, key2: value2)
other types of CSS property value such as Unicode ranges and !important declarations. However, it has no special handling for these types. They're treated just like unquoted strings.	!important

SassScript allows properties to use variables, arithmetic, and extra functions.

3.3. Lists

Lists are how Sass represents the values of CSS declarations like

- margin: 10px 15px 0 0
- font-face: Helvetica, Arial, sans-serif
- etc.

Lists are just a series of other values, separated by either spaces or commas. In fact, individual values count as lists, too: they're just lists with one item.

On their own, lists don't do much, but the SassScript list functions make them useful:

- The nth function can access items in a list
- The join function can join multiple lists together
- The append function can add items to lists
- The @each directive can also add styles for each item in a list.

In addition to containing simple values, lists can contain other lists.

For example, 1px 2px, 5px 6px is a two-item list containing the list 1px 2px and the list 5px 6px.

If the inner lists have the same separator as the outer list, you'll need to use parentheses to make it clear where the inner lists start and stop. For example, (1px 2px) (5px 6px) is also a two-item list containing the list 1px 2px and the list 5px 6px. The difference is that the outer list is space-separated, where before it was comma-separated.

When lists are turned into plain CSS, Sass doesn't add any parentheses, since CSS doesn't understand them. That means that (1px 2px) (5px 6px) and 1px 2px 5px 6px will look the same when they become CSS. However, they aren't the same when they're Sass: the first is a list containing two lists, while the second is a list containing four numbers.

Lists can also have no items in them at all. These lists are represented as () ,They can't be output directly to CSS; if you try to do e.g. font-family: (), Sass will raise an error. If a list contains empty lists or null values, as in 1px 2px () 3px or 1px 2px null 3px, the empty lists and null values will be removed before the containing list is turned into CSS.

Comma-separated lists may have a trailing comma. This is especially useful because it allows you to represent a single-element list. For example, (1,) is a list containing 1 and (1 2 3,) is a comma-separated list containing a space-separated list containing 1, 2, and 3.

3.4. Maps

Maps represent an association between keys and values, where keys are used to look up values. They make it easy to collect values into named groups and access those groups dynamically. They have no direct parallel in CSS, although they're syntactically similar to media query expressions:

$map: (key1: value1, key2: value2, key3: value3);

Unlike lists, maps must always be surrounded by parentheses and must always be comma-separated. Both the keys and values in maps can be any SassScript object.

A map may only have one value associated with a given key (although that value may be a list). A given value may be associated with many keys, though.

Like lists, maps are mostly manipulated using SassScript functions.

Maps can also be used anywhere lists can. When used by a list function, a map is treated as a list of pairs. For example, (key1: value1, key2: value2) would be treated as the nested list key1 value1, key2 value2 by list functions. Lists cannot be treated as maps, though, with the exception of the empty list. () represents both a map with no key/value pairs and a list with no elements.

Note that map keys can be any Sass data type (even another map) and the syntax for declaring a map allows arbitrary SassScript expressions that will be evaluated to determine the key.

Maps cannot be converted to plain CSS. Using one as the value of a variable or an argument to a CSS function will cause an error.

Use the inspect($value) function to produce an output string useful for debugging maps.

3.5. Operations

All types support equality operations (== and !=).

In addition, each type has its own operations that it has special support for.

3.6. Number Operations

SassScript supports the standard arithmetic operations on numbers:

- addition +,
- subtraction -,
- multiplication *
- division /
- modulo %
- <
- >
- >=
- <=

Sass math functions preserve units during arithmetic operations. This means that, just like in real life, you cannot work on numbers with incompatible units (such as adding a number with px and em) and two numbers with the same unit that are multiplied together will produce square units (10px * 10px == 100px * px). Be Aware that px * px is an invalid CSS unit and you will get an error from Sass for attempting to use invalid units in CSS.

Division and /

CSS allows / to appear in property values as a way of separating numbers. Since SassScript is an extension of the CSS property syntax, it must support this, while also allowing / to be used for division. This means that by default, if two numbers are separated by / in SassScript, then they will appear that way in the resulting CSS.

However, there are three situations where the / will be interpreted as division. These cover the vast majority of cases where division is actually used. They are:

If the value, or any part of it, is stored in a variable or returned by a function.

- If the value is surrounded by parentheses, unless those parentheses are outside a list and the value is inside.
- If the value is used as part of another arithmetic expression.

For example:

SCSS

```scss
p {
// Plain CSS, no division
  font: 10px/8px;

  $width: 1000px;

// Uses a variable, does division
  width: $width/2;

// Uses a function, does division
  width: round(1.5)/2;

// Uses parentheses, does division
  height: (500px/2);

// Uses +, does division
  margin-left: 5px + 8px/2px;

// In a list, parentheses don't count
  font: (italic bold 10px/8px);
}
```

Compiled CSS

```css
p {
  font: 10px/8px;
  width: 500px;
  width: 1;
  height: 250px;
  margin-left: 9px;
  font: italic bold 10px/8px;
}
/*# sourceMappingURL=style.css.map
*/
```

If you want to use variables along with a plain CSS /, you can use #{} to insert them. For example:

SCSS

```scss
p {
  $font-size: 12px;
  $line-height: 30px;
  font: #{$font-size}/#{$line-height};
}
```

Compiled CSS

```css
p {
  font: 12px/30px;
}
/*# sourceMappingURL=style.css.map */
```

Subtraction, Negative Numbers, and -

There are a number of different things - can mean in CSS and in Sass. It can be a subtraction operator (as in 5px - 3px), the beginning of a negative number (as in -3px), a unary negation operator (as in -$var), or part of an identifier (as in font-weight). Most of the time, it's clear which is which, but there are some tricky cases. As a general rule, you're safest if:

- You always include spaces on both sides of - when subtracting.
- You include a space before - but not after for a negative number or a unary negation.
- You wrap a unary negation in parentheses if it's in a space-separated list, as in 10px (-$var).

The different meanings of - take precedence in the following order:

- A - as part of an identifier. This means that a-1 is an unquoted string with value "a-1". The only exception are units; Sass normally allows any valid identifier to be used as an identifier, but identifiers may not contain a hyphen followed by a digit. This means that 5px-3px is the same as 5px - 3px.
- A - between two numbers with no whitespace. This indicates subtraction, so 1-2 is the same as 1 - 2.
- A - at the beginning of a literal number. This indicates a negative number, so 1 -2 is a list containing 1 and -2.
- A - between two numbers regardless of whitespace. This indicates subtraction, so 1 -$var are the same as 1 - $var.
- A - before a value. This indicates the unary negation operator; that is, the operator that takes a number and returns its negative.

3.7. Color Operations

All arithmetic operations are supported for color values, where they work piecewise. This means that the operation is performed on the red, green, and blue components in turn. For example:

SCSS	Compiled CSS
`p {`	`p {`
`  color: #010203 + #040506;`	`  color: #050709;`
`}`	`}`
	`/*# sourceMappingURL=style.css.map */`

Often it's more useful to use color functions than to try to use color arithmetic to achieve the same effect.

Arithmetic operations also work between numbers and colors, also piecewise. For example:

SCSS	Compiled CSS
`p {`	`p {`
`  color: #010203 * 2;`	`  color: #020406;`
`}`	`}`
	`/*# sourceMappingURL=style.css.map */`

Note that colors with an alpha channel (those created with the rgba or hsla functions) must have the same alpha value in order for color arithmetic to be done with them. The arithmetic doesn't affect the alpha value.

For example:

SCSS	Compiled CSS
`p {`	`p {`
`  color: rgba(255, 0, 0, 0.75) +`	`  color: rgba(255, 255, 0, 0.75);`
`rgba(0, 255, 0, 0.75);`	`}`
`}`	`/*# sourceMappingURL=style.css.map */`

The alpha channel of a color can be adjusted using the opacify and transparentize functions.

For example:

SCSS

```scss
$translucent-red: rgba(255, 0,0, 0.5);
p {
  color: opacify($translucent-red,
0.3);
  background-color:
transparentize($translucent-red,
0.25);
}
```

Compiled CSS

```css
p {
  color: rgba(255, 0, 0, 0.8);
  background-color: rgba(255,0,0, 0.25);
}
/*# sourceMappingURL=style.css.map */
```

3.8. String Operations

The + operation can be used to concatenate strings.

Note that if a quoted string is added to an unquoted string (that is, the quoted string is to the left of the +), the result is a quoted string. Likewise, if an unquoted string is added to a quoted string (the unquoted string is to the left of the +), the result is an unquoted string. For example:

SCSS

```scss
p:before {
  content: "Foo " + Bar;
  font-family: sans- + "serif";
}
```

Compiled CSS

```css
p:before {
  content: "Foo Bar";
  font-family: sans-serif;
}
/*# sourceMappingURL=style.css.map */
```

Within a string of text, #{} style interpolation can be used to place dynamic values within the string:

SCSS

```
p:before {
  content: "I ate #{5 + 10} pies!";
}
```

Compiled CSS

```
p:before {
  content: "I ate 15 pies!";
}
/*# sourceMappingURL=style.css.map */
```

Null values are treated as empty strings for string interpolation:

SCSS

```
$value: null;
p:before {
  content: "I ate #{$value} pies!";
}
```

Compiled CSS

```
p:before {
  content: "I ate  pies!";
}
/*# sourceMappingURL=style.css.map */
```

3.9. Boolean Operations

SassScript supports the following operators for boolean values:

- and
- or
- not

3.10. Parentheses

Parentheses can be used to affect the order of operations:

SCSS

```
p {
  width: 1em + (2em * 3);
}
```

Compiled CSS

```
p {
  width: 7em;
}
/*# sourceMappingURL=style.css.map */
```

3.11. Functions

SassScript defines some useful functions that are called using the normal CSS function syntax:

SCSS	Compiled CSS
`p {` `color: hsl(0, 100%, 50%);` `}`	`p {` `color: red;` `}` `/*# sourceMappingURL=style.css.map */`

See this page for a full list of available functions.

3.12. Keyword Arguments

Sass functions can also be called using explicit keyword arguments. The above example can also be written as:

SCSS	Compiled CSS
`p {` `color: hsl($hue: 0,` `$saturation: 100%,` `$lightness: 50%);` `}`	`p {` `color: red;` `}` `/*# sourceMappingURL=style.css.map */`

While this is less concise, it can make the stylesheet easier to read. It also allows functions to present more flexible interfaces, providing many arguments without becoming difficult to call.

Named arguments can be passed in any order, and arguments with default values can be omitted. Since the named arguments are variable names, underscores and dashes can be used interchangeably.

3.13. Interpolation: #{}

You can also use SassScript variables in selectors and property names using #{} interpolation syntax.

It's also possible to use #{} to put SassScript into property values. In most cases this isn't any better than using a variable, but using #{} does mean that any operations near it will be treated as plain CSS. For example:

SCSS

```scss
$name: foo;
$attr: border;
p.#{$name} {
  #{$attr}-color: blue;
}

p {
  $font-size: 12px;
  $line-height: 30px;
  font: #{$font-size}/#{$line-height};
}
```

Compiled CSS

```css
p.foo {
  border-color: blue;
}

p {
  font: 12px/30px;
}
/*# sourceMappingURL=style.css.map */
```

3.14. & in SassScript

Just like when it's used in selectors, **&** in SassScript refers to the current parent selector.

It's a comma-separated list of space-separated lists.

If there is no parent selector, the value of & will be null.

For example:

SCSS

```scss
.foo.bar .baz.bang, .bip.qux {
  $selector: &;
  /* $selector is: #{$selector};*/
}
```

Compiled CSS

```css
.foo.bar .baz.bang, .bip.qux {
  /* $selector is:.foo.bar .baz.bang, .bip.qux;*/
}
/*# sourceMappingURL=style.css.map */
```

3.15. Variable Defaults: !default

You can assign to variables if they aren't already assigned by adding the !default flag to the end of the value. This means that if the variable has already been assigned to, it won't be re-assigned, but if it doesn't have a value yet, it will be given one.

Variables with null values are treated as unassigned by !default.

For example:

SCSS

```scss
$content1: "First content";
$content1: "Second content?" !default;

$content2: null;
$content2: "Non-null content" !default;

$content3: "First time reference"
!default;

#main1 {
  content: $content1;
}

#main2 {
  content: $content2;
}

#main3 {
  content: $content3;
}
```

Compiled CSS

```css
#main1 {
  content: "First content";
}

#main2 {
  content: "Non-null content";
}

#main3 {
  content: "First time reference";
}
/*# sourceMappingURL=style.css.map
*/
```

4. @-Rules and Directives

Sass supports all CSS3 @-rules, as well as some additional Sass-specific ones known as "directives." These have various effects in Sass, detailed below.

@import It imports the SASS or SCSS files, it directly takes the filename to import.
@media It sets the style rule to different media types.
@extend @extend directive is used to share rules and relationships between selectors.
@at-root @at-root directive is a collection of nested rules, which is able to make style block at root of the document.
@debug @debug directive detects the errors and displays the SassScript expression values to the standard error output stream.
@warn @warn directive is used to give cautionary advice about the problem; it displays the SassScript expression values to the standard error output stream.
@error @error directive displays the SassScript expression value as fatal error.

4.1. @import

Sass extends the CSS @import rule to allow it to import SCSS and Sass files. All imported SCSS and Sass files will be merged together into a single CSS output file. In addition, any variables or mixins defined in imported files can be used in the main file.

Sass looks for other Sass files in the current directory, and the Sass file directory under Rack, Rails, or Merb. Additional search directories may be specified using the :load_paths option, or the --load-path option on the command line.

@import takes a filename to import. By default, it looks for a Sass file to import directly, but there are a few circumstances under which it will compile to a CSS @import rule:

- If the file's extension is .css.
- If the filename begins with http://.
- If the filename is a url().
- If the @import has any media queries.
- If none of the above conditions are met and the extension is .scss or .sass, then the named Sass or SCSS file will be imported. If there is no extension, Sass will try to find a file with that name and the .scss or .sass extension and import it.

For example:

```scss
//import the file foo.scss

@import "foo.scss";

//import the file foo.scss

@import "foo";

//import the file foo.css

@import "foo.css";

@import "foo" screen;

@import url(foo);

//import multiple files in one @import.

//would import the a and the b files.

@import "a", "b";
```

Imports may contain #{} interpolation, but only with certain restrictions. It's not possible to dynamically import a Sass file based on a variable; interpolation is only for CSS imports. As such, it only works with url() imports. For example:

SCSS	Compiled CSS

`$family: unquote("Droid+Sans");`
`@import`
`url("http://fonts.googleapis.com/css?fam`
`ily=#{$family}");`
	`@import`
	`url("http://fonts.googleapis.com/css?`
	`family=Droid+Sans");`
	`/*# sourceMappingURL=style.css.map */`

Partials

If you have a SCSS or Sass file that you want to import but don't want to compile to a CSS file, you can add an underscore to the beginning of the filename. This will tell Sass not to compile it to a normal CSS file. You can then import these files without using the underscore.

For example, you might have _colors.scss. Then no _colors.css file would be created, and you can do

@import "colors";

and _colors.scss would be imported.

Note that you may not include a partial and a non-partial with the same name in the same directory. For example, _colors.scss may not exist alongside colors.scss.

Nested @import

Although most of the time it's most useful to just have @imports at the top level of the document, it is possible to include them within CSS rules and @media rules. Like a base-level @import, this includes the contents of the @imported file. However, the imported rules will be nested in the same place as the original @import.

For example:

foo.scss:

SCSS	Compiled CSS

`.example {`
`    color: red;`
`  }`
	`.example {`
	`  color: red;`
	`}`
	`/*# sourceMappingURL=foo.css.map */`

style.scss:

SCSS	Compiled CSS

`#main {`
`  @import "foo";`
`}`
	`#main .example {`
	`  color: red;`
	`}`
	`/*# sourceMappingURL=style.css.map */`

Directives that are only allowed at the base level of a document, like @mixin or @charset, are not allowed in files that are @imported in a nested context.

It's not possible to nest @import within mixins or control directives.

4.2. @media

@media directives in Sass behave just like they do in plain CSS, with one extra capability: they can be nested in CSS rules. If a @media directive appears within a CSS rule, it will be bubbled up to the top level of the stylesheet, putting all the selectors on the way inside the rule. This makes it easy to add media-specific styles without having to repeat selectors or break the flow of the stylesheet.

@media queries can also be nested within one another. The queries will then be combined using the and operator.

Finally, @media queries can contain SassScript expressions (including variables, functions, and operators) in place of the feature names and feature values.

For example:

SCSS

```scss
.sidebar1 {
  width: 300px;
  @media screen and (orientation: landscape) {
    width: 500px;
  }
}


@media screen {
  .sidebar2 {
    @media (orientation: landscape) {
      width: 500px;
    }
  }
}

$media: screen;
$feature: -webkit-min-device-pixel-ratio;
$value: 1.5;

@media #{$media} and ($feature: $value) {
  .sidebar3 {
    width: 500px;
  }
}
```

Compiled CSS

```css
.sidebar1 {
  width: 300px;
}

@media screen and (orientation: landscape) {
  .sidebar1 {
    width: 500px;
  }
}

@media screen and (orientation: landscape) {
  .sidebar2 {
    width: 500px;
  }
}

@media screen and (-webkit-min-device-pixel-
ratio: 1.5) {
  .sidebar3 {
    width: 500px;
  }
}
/*# sourceMappingURL=style.css.map */
```

4.3. @extend

There are often cases when designing a page when one class should have all the styles of another class, as well as its own specific styles. The most common way of handling this is to use both the more general class and the more specific class in the HTML. For example, suppose we have a design for a normal error and also for a serious error. We might write our markup like so:

```
<!DOCTIPE html>
<html>

<head>
    <style>
        .error {
            border: 1px #f00;
            background-color: #fdd;
        }

        .seriousError {
            border-width: 3px;
        }
    </style>
</head>

<body>

    <div class="error">
        error
    </div>

    <div class="error seriousError">
        seriousError
    </div>
</body>

</html>
```

Unfortunately, this means that we have to always remember to use .error with .seriousError. This is a maintenance burden, leads to tricky bugs, and can bring non-semantic style concerns into the markup.

The @extend directive avoids these problems by telling Sass that one selector should inherit the styles of another selector. For example:

This means that all styles defined for .error are also applied to .seriousError, in addition to the styles specific to .seriousError.

In effect, every element with class .seriousError also has class .error.

@extend works by inserting the extending selector (e.g. .seriousError) anywhere in the stylesheet that the extended selector (.e.g .error) appears. Thus the example above:

SCSS

```scss
.error {
  border: 1px #f00;
  background-color: #fdd;
}

.seriousError {
  @extend .error;
  border-width: 3px;
}
```

Compiled CSS

```css
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
}

.seriousError {
  border-width: 3px;
}
/*# sourceMappingURL=style.css.map */
```

.

Extending Complex Selectors

Class selectors aren't the only things that can be extended. It's possible to extend any selector involving only a single element, such as . a:hover, For example:

SCSS

```scss
div, a:hover{
  color:green;
}

.hoverlink {
  @extend a:hover;
}
```

Compiled CSS

```css
div, a:hover, .hoverlink {
  color: green;
}
/*# sourceMappingURL=style.css.map */
```

Multiple Extends

A single selector can extend more than one selector. This means that it inherits the styles of all the extended selectors. For example:

SCSS

```scss
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.attention {
  font-size: 3em;
  background-color: #ff0;
}
.seriousError {
  @extend .error;
  @extend .attention;
  border-width: 3px;
}
```

Compiled CSS

```css
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
}

.attention, .seriousError {
  font-size: 3em;
  background-color: #ff0;
}

.seriousError {
  border-width: 3px;
}
/*# sourceMappingURL=style.css.map */
```

In effect, every element with class .seriousError also has class .error and class .attention. Thus, the styles defined later in the document take precedence: .seriousError has background color #ff0 rather than #fdd, since .attention is defined later than .error.

Multiple extends can also be written using a comma-separated list of selectors. For example:

```scss
.seriousError {
  @extend .error, .attention;
  border-width: 3px;
}
```

is the same as

```scss
.seriousError {
  @extend .error;
  @extend .attention;
  border-width: 3px;
}
```

Chaining Extends

It's possible for one selector to extend another selector that in turn extends a third. For example:

SCSS

```scss
.error {
  border: 1px #f00;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
.criticalError {
  @extend .seriousError;
  top: 10%;
}
```

Compiled CSS

```css
.error, .seriousError, .criticalError
{
  border: 1px #f00;
}

.seriousError, .criticalError {
  border-width: 3px;
}

.criticalError {
  top: 10%;
}
/*# sourceMappingURL=style.css.map */
```

Now everything with class .seriousError also has class .error, and everything with class .criticalError has class .seriousError and class .error.

Selector Sequences

Selector sequences, such as .foo .bar or .foo + .bar, currently can't be extended.

However, it is possible for nested selectors themselves to use @extend. For example:

SCSS

```scss
#fake-links .link {
  @extend a;
}

a {
  color: blue;
  &:hover {
    text-decoration: underline;
  }
```

Compiled CSS

```css
a, #fake-links .link {
  color: blue;
}

a:hover, #fake-links .link:hover {
  text-decoration: underline;
}
/*# sourceMappingURL=style.css.map */
```

```
}
```

Merging Selector Sequences

Sometimes a selector sequence extends another selector that appears in another sequence. In this case, the two sequences need to be merged.

If the two sequences do share some selectors, then those selectors will be merged together and only the differences (if any still exist) will alternate. In this example, both sequences contain the id #admin, so the resulting selectors will merge those two ids:

SCSS

```scss
#admin .tabbar a {
  font-weight: bold;
}
#demo .overview #admin {
  @extend a;
}
```

Compiled CSS

```css
#admin .tabbar a, #admin .tabbar #demo
.overview #admin, #demo .overview
#admin .tabbar #admin {
  font-weight: bold;
}
```

4.4. @extend-Only Selectors

Sometimes you'll write styles for a class that you only ever want to @extend, and never want to use directly in your HTML.

This is especially true when writing a Sass library, where you may provide styles for users to @extend if they need and ignore if they don't.

If you use normal classes for this, you end up creating a lot of extra CSS when the stylesheets are generated, and run the risk of colliding with other classes that are being used in the HTML. That's why Sass supports "placeholder selectors" (for example, %foo).

Placeholder selectors look like class and id selectors, except the # or . is replaced by %. They can be used anywhere a class or id could, and on their own they prevent rulesets from being rendered to CSS.

For example:

SCSS

```scss
#context a%extreme {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
.notice {
  @extend %extreme;
}
```

Compiled CSS

```css
#context a.notice {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

4.5. The !optional Flag

Normally when you extend a selector, it's an error if that @extend doesn't work. For example, if you write

a.important {@extend .notice}

It's an error if there are no selectors that contain .notice.

It's also an error if the only selector containing .notice is h1.notice, since h1 conflicts with a and so no new selector would be generated.

Sometimes, though, you want to allow an @extend not to produce any new selectors. To do so, just add the !optional flag after the selector. For example:

SCSS

```scss
a.important {
  @extend .notice !optional;
  color:green;
}
```

Compiled CSS

```css
a.important {
  color: green;
}
/*# sourceMappingURL=style.css.map */
```

4.6. @extend in Directives

There are some restrictions on the use of @extend within directives such as @media. Sass is unable to make CSS rules outside of the @media block apply to selectors inside it without creating a huge amount of stylesheet bloat by copying styles all over the place. This means that if you use @extend within @media (or other CSS directives), you may only extend selectors that appear within the same directive block.
For example, the following works fine:

SCSS

```scss
@media print {
  .error {
    border: 1px #f00;
    background-color: #fdd;
  }
  .seriousError {
    @extend .error;
    border-width: 3px;
  }
}
```

Compiled CSS

```css
@media print {
  .error, .seriousError {
    border: 1px #f00;
    background-color: #fdd;
  }
  .seriousError {
    border-width: 3px;
  }
}
/*# sourceMappingURL=style.css.map */
```

But this is an error:

```scss
.error {
  border: 1px #f00;
  background-color: #fdd;
}

@media print {
  .seriousError {
    // INVALID EXTEND: .error is used outside of the "@media print" directive
    @extend .error;
    border-width: 3px;
  }
}
```

```
Error: You may not @extend an outer selector from within @media.
       You may only @extend selectors within the same directive.
       From "@extend .error" on line 9 of sass/c:\Users\Anna\sass-test\style.scss
        on line 1 of sass/c:\Users\Anna\sass-test\style.scss
```

Someday we hope to have @extend supported natively in the browser, which will allow it to be used within @media and other directives.

5. Control Directives & Expressions

SassScript supports basic control directives and expressions for including styles only under some conditions or including the same style several times with variations.

Note: Control directives are an advanced feature, and are uncommon in day-to-day styling. They exist mainly for use in mixins, particularly those that are part of libraries like Compass, and so require substantial flexibility.

SassScript basic control directives and expressions:

if() Based on the condition, *if()* function returns only one result from two possible outcomes.
@if The *@if* directive accepts SassScript expressions and uses the nested styles whenever the result of the expression is anything other than *false* or *null*.
@for The *@for* directive allows you to generate styles in a loop.
@each In *@each* directive, a variable is defined which contains the value of each item in a list.
@while It takes SassScript expressions and untill the statement evaluates to false it iteratively outputs nested styles.

5.1. if()

The built-in if() function allows you to branch on a condition and returns only one of two possible outcomes. It can be used in any script context. The if function only evaluates the argument corresponding to the one that it will return -- this allows you to refer to variables that may not be defined or to have calculations that would otherwise cause an error (E.g. divide by zero).

SCSS

```scss
div{
  height: if(true, 1px, 2px);
  width: if(false, 1px, 2px);
}
```

Compiled CSS

```css
div {
  height: 1px;
  width: 2px;
}
/*# sourceMappingURL=style.css.map */
```

5.2. @if

The @if directive takes a SassScript expression and uses the styles nested beneath it if the expression returns anything other than false or null:

SCSS

```scss
div{
  @if 1 + 1 == 2 { border: 1px solid;  }
  @if 5 < 3      { border: 2px dotted; }
  @if null       { border: 3px double; }

}
```

Compiled CSS

```css
div {
  border: 1px solid;
}
```

You can explicitly test for $var == false or $var == null if you want to distinguish between these.

The @if statement can be followed by several @else if statements and one @else statement.

If the @if statement fails, the @else if statements are tried in order until one succeeds or the @elseis reached.

For example:

SCSS

```scss
$type: monster;
p {
  @if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
    color: green;
  } @else {
    color: black;
  }
}
```

Compiled CSS

```css
p {
  color: green;
}
```

5.3. @for

The @for directive repeatedly outputs a set of styles.

For each repetition, a counter variable is used to adjust the output. The directive has two forms:

- @for $var from <start> through <end>
- @for $var from <start> to <end>

$var can be any variable name, like $i; <start> and <end> are SassScript expressions that should return integers. When <start> is greater than <end> the counter will decrement instead of increment.

The @for statement sets $var to each successive number in the specified range and each time outputs the nested styles using that value of $var. For the form from ... through, the range includes the values of <start> and <end>, but the form from ... to runs up to but not including the value of <end>. Using the through syntax, SCSS

```scss
@for $i from 1 through 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

Compiled CSS

```css
.item-1 {
  width: 2em;
}

.item-2 {
  width: 4em;
}

.item-3 {
  width: 6em;
}
/*# sourceMappingURL=style.css.map */
```

SCSS

```scss
@for $i from 1 to 3 {
  .item-#{$i} { width: 2em * $i; }
}
```

Compiled CSS

```css
.item-1 {
  width: 2em;
}

.item-2 {
  width: 4em;
}
/*# sourceMappingURL=style.css.map */
```

5.4. @each

The @each directive usually has the form @each $var in <list or map>.

$var can be any variable name, like $length or $name, and <list or map> is a SassScript expression that returns a list or a map.

The @each rule sets $var to each item in the list or map, then outputs the styles it contains using that value of $var.

For example:

SCSS	Compiled CSS

`@each $color in green, red, blue, black`
`{`
`  .#{$color}-div {`
`    background-color: $color;`
`  }`
`}`
	`.green-div {`
	`  background-color: green;`
	`}`
	
	`.red-div {`
	`  background-color: red;`
	`}`
	
	`.blue-div {`
	`  background-color: blue;`
	`}`
	
	`.black-div {`
	`  background-color: black;`
	`}`
	`/*# sourceMappingURL=style.css.map */`

Multiple Assignment

The @each directive can also use multiple variables, as in @each $var1, $var2, ... in <list>. If <list> is a list of lists, each element of the sub-lists is assigned to the respective variable. For example:

SCSS

```scss
@each $animal, $color, $cursor in
(puma, black, default),
(sea-slug, blue, pointer),
(egret, white, move) {
.#{$animal}-icon {
background-image:
url('/images/#{$animal}.png');
border: 2px solid $color;
cursor: $cursor;
}
}
```

Compiled CSS

```css
.puma-icon {
  background-image:
url("/images/puma.png");
  border: 2px solid black;
  cursor: default;
}

.sea-slug-icon {
  background-image: url("/images/sea-
slug.png");
  border: 2px solid blue;
  cursor: pointer;
}

.egret-icon {
  background-image:
url("/images/egret.png");
  border: 2px solid white;
  cursor: move;
}
/*# sourceMappingURL=style.css.map */
```

Since maps are treated as lists of pairs, multiple assignment works with them as well. For example:

SCSS	Compiled CSS
`@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) { #{$header} { font-size: $size; } }`	`h1 { font-size: 2em; } h2 { font-size: 1.5em; } h3 { font-size: 1.2em; } /*# sourceMappingURL=style.css.map */`

SCSS:

```
@each $header, $size in (h1: 2em, h2:
1.5em, h3: 1.2em) {
  #{$header} {
    font-size: $size;
  }
}
```

Compiled CSS:

```
h1 {
  font-size: 2em;
}

h2 {
  font-size: 1.5em;
}

h3 {
  font-size: 1.2em;
}
/*# sourceMappingURL=style.css.map */
```

5.5. @while

The @while directive takes a SassScript expression and repeatedly outputs the nested styles until the statement evaluates to false. This can be used to achieve more complex looping than the @for statement is capable of, although this is rarely necessary. For example:

SCSS:

```
$i: 6;
@while $i > 0 {
  .item-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

Compiled CSS:

```
.item-6 {
  width: 12em;
}

.item-4 {
  width: 8em;
}

.item-2 {
  width: 4em;
}
/*# sourceMappingURL=style.css.map */
```

6. Mixin Directive

Mixins allow you to define styles that can be re-used throughout the stylesheet without needing to resort to non-semantic classes like .float-left.

Mixins can also contain full CSS rules, and anything else allowed elsewhere in a Sass document. They can even take arguments which allows you to produce a wide variety of styles with very few mixins.

6.1. Defining a Mixin: @mixin

Mixins are defined with the @mixin directive. It's followed by the name of the mixin and optionally the arguments, and a block containing the contents of the mixin.

Mixins may also contain selectors, possibly mixed with properties. The selectors can even contain parent references. For example:

SCSS

```scss
@mixin clearfix {
  display: inline-block;
  &:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
  }
  * html & { height: 1px }
}
```

Compiled CSS

```css
/* No CSS *//*#
sourceMappingURL=style.css.map */
```

6.2. Including a Mixin: @include

Mixins are included in the document with the @include directive. This takes the name of a mixin and optionally arguments to pass to it, and includes the styles defined by that mixin into the current rule. For example:

SCSS

```scss
@mixin links {
  a {
    color: blue;
    background-color: red;
  }
}

@include links;
```

Compiled CSS

```css
a {
  color: blue;
  background-color: red;
}
/*# sourceMappingURL=style.css.map */
```

Mixins may also be included outside of any rule (that is, at the root of the document) as long as they don't directly define any properties or use any parent references. For example:

Mixin definitions can also include other mixins. For example:

SCSS

```scss
@mixin compound {
  @include highlighted-background;
  @include header-text;
}

@mixin highlighted-background {
background-color: #fc0;
}

@mixin header-text {
font-size: 20px;
}


div {
  @include compound;
}
```

Compiled CSS

```css
div {
  background-color: #fc0;
  font-size: 20px;
}
/*# sourceMappingURL=style.css.map
*/
```

Mixins may include themselves. This is different than the behavior of Sass versions prior to 3.3, where mixin recursion was forbidden.

Mixins that only define descendent selectors can be safely mixed into the top most level of a document.

6.3. Arguments

Mixins can take SassScript values as arguments, which are given when the mixin is included and made available within the mixin as variables.

When defining a mixin, the arguments are written as variable names separated by commas, all in parentheses after the name. Then when including the mixin, values can be passed in in the same manner. For example:

SCSS

```scss
@mixin borders($color, $width) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}

p { @include borders(blue, 1in); }
```

Compiled CSS

```css
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed;
}
/*# sourceMappingURL=style.css.map */
```

Mixins can also specify default values for their arguments using the normal variable-setting syntax. Then when the mixin is included, if it doesn't pass in that argument, the default value will be used instead.

For example:

SCSS

```scss
@mixin borders($color, $width: 1in) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
p { @include borders(blue); }
h1 { @include borders(blue, 2in); }
```

Compiled CSS

```css
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed;
}

h1 {
  border-color: blue;
  border-width: 2in;
  border-style: dashed;
}
/*# sourceMappingURL=style.css.map */
```

6.4. Keyword Arguments

Mixins can also be included using explicit keyword arguments:

SCSS

```scss
@mixin borders($color, $width: 1in) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
p { @include borders($color: blue); }
h1 { @include borders($color: blue,
$width: 2in); }
```

Compiled CSS

```css
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed;
}

h1 {
  border-color: blue;
  border-width: 2in;
  border-style: dashed;
}
/*# sourceMappingURL=style.css.map */
```

While this is less concise, it can make the stylesheet easier to read. It also allows functions to present more flexible interfaces, providing many arguments without becoming difficult to call.

Named arguments can be passed in any order, and arguments with default values can be omitted. Since the named arguments are variable names, underscores and dashes can be used interchangeably.

6.5. Variable Arguments

Sometimes it makes sense for a mixin or function to take an unknown number of arguments. For example, a mixin for creating box shadows might take any number of shadows as arguments. For these situations, Sass supports "variable arguments," which are arguments at the end of a mixin or function declaration that take all leftover arguments and package them up as a list. These arguments look just like normal arguments, but are followed by ....

For example:

SCSS	Compiled CSS

`@mixin box-shadow($shadows...) {`
`  -moz-box-shadow: $shadows;`
`  -webkit-box-shadow: $shadows;`
`  box-shadow: $shadows;`
`}`
 	
`.shadows {`
`  @include box-shadow(0px 4px 5px #666, 2px 6px 10px #999);`
`}`
	`.shadows {`
	`  -moz-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;`
	`  -webkit-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;`
	`  box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;`
	`}`
	`/*# sourceMappingURL=style.css.map */`

Variable arguments also contain any keyword arguments passed to the mixin or function. These can be accessed using the keywords($args) function, which returns them as a map from strings (without $) to values.

Variable arguments can also be used when calling a mixin. Using the same syntax, you can expand a list of values so that each value is passed as a separate argument, or expand a map of values so that each pair is treated as a keyword argument.

For example:

SCSS

```scss
@mixin colors($text, $background, $border) {
  color: $text;
  background-color: $background;
  border-color: $border;
}



$values: #ff0000, #00ff00, #0000ff;
.primary {
  @include colors($values...);
}



$value-map: (text: #00ff00, background:
#0000ff, border: #ff0000);

.secondary {
  @include colors($value-map...);
}
```

Compiled CSS

```css
.primary {
  color: #ff0000;
  background-color: #00ff00;
  border-color: #0000ff;
}

.secondary {
  color: #00ff00;
  background-color: #0000ff;
  border-color: #ff0000;
}
/*# sourceMappingURL=style.css.map */
```

You can pass both an argument list and a map as long as the list comes before the map, as in @include colors($values..., $map...).

You can use variable arguments to wrap a mixin and add additional styles without changing the argument signature of the mixin. If you do, keyword arguments will get directly passed through to the wrapped mixin.

For example:

SCSS

```scss
@mixin wrapped-stylish-mixin($args...) {
  font-weight: bold;
  @include stylish-mixin($args...);
}

@mixin stylish-mixin($color,$width) {
  color: $color;
  width:$width;
}
.stylish {
  // The $width argument will get passed on to "stylish-mixin" as a keyword
  @include wrapped-stylish-mixin(#00ff00, $width: 100px);
}
```

Compiled CSS

```css
.stylish {
  font-weight: bold;
  color: #00ff00;
  width: 100px;
}
/*# sourceMappingURL=style.css.map */
```

6.6. Passing Content Blocks to a Mixin

It is possible to pass a block of styles to the mixin for placement within the styles included by the mixin. The styles will appear at the location of any @content directives found within the mixin. This makes it possible to define abstractions relating to the construction of selectors and directives.

For example:

SCSS

```scss
@mixin mix1{
  * html {
    @content;
  }
}
@include mix1 {
  #logo {
    color: red;
  }
}
```

Compiled CSS

```css
* html #logo {
  color: red;
}
/*# sourceMappingURL=style.css.map */
```

Note: when the @content directive is specified more than once or in a loop, the style block will be duplicated with each invocation.

6.7. Variable Scope and Content Blocks

The block of content passed to a mixin are evaluated in the scope where the block is defined, not in the scope of the mixin. This means that variables local to the mixin cannot be used within the passed style block and variables will resolve to the global value:

SCSS

```scss
$color: white;
@mixin colors($color: blue) {
  background-color: $color;
  @content;
  border-color: $color;
}
.colors {
  @include colors { color: $color; }
}
```

Compiled CSS

```css
.colors {
  background-color: blue;
  color: white;
  border-color: blue;
}
/*# sourceMappingURL=style.css.map */
```

7. Function Directive

It is possible to define your own functions in sass and use them in any value or script context. For example:

SCSS

```scss
$grid-width: 40px;
$gutter-width: 10px;

@function grid-width($n) {
  @return $n * $grid-width + ($n - 1)
* $gutter-width;
}

#sidebar1 { width: grid-width(5); }


#sidebar2 { width: grid-width($n: 5);
}
```

Compiled CSS

```css
#sidebar1 {
  width: 240px;
}

#sidebar2 {
  width: 240px;
}
/*# sourceMappingURL=style.css.map */
```

As you can see functions can access any globally defined variables as well as accept arguments just like a mixin. A function may have several statements contained within it, and you must call @return to set the return value of the function.

It is recommended that you prefix your functions to avoid naming conflicts and so that readers of your stylesheets know they are not part of Sass or CSS

User-defined functions also support variable arguments in the same way as mixins.

For historical reasons, function names (and all other Sass identifiers) can use hyphens and underscores interchangeably. For example, if you define a function called grid-width, you can use it as grid_width, and vice versa.