FREE eBook

LEARNING less

Free unaffiliated eBook created from
Stack Overflow contributors.

#less

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: less

It is an unofficial and free less ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official less.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with less

Remarks

Less is an open-source pre-processor. It makes writing and maintaining CSS easier by allowing the author to define and use variables, mixins etc. It also has features like Guards using which conditional styles can be written, Loops which help to keep the code DRY and a lot of in-built functions to perform math operations, string and color manipulations etc.

Not to be confused with the Unix tool of the same name.

Versions

Version	Release Date
2.7.1	2016-05-09
2.7.0	2016-05-07
2.6.1	2016-03-04
2.6.0	2016-01-29
2.5.3	2015-09-25
2.5.2	2015-09-24
2.5.1	2015-05-21
2.5.0	2015-04-03
2.4.0	2015-02-07
2.3.1	2015-01-28
2.3.0	2015-01-27
2.2.0	2015-01-04
2.1.2	2014-12-20
2.1.1	2014-11-27
2.1.0	2014-11-23
2.0.0	2014-11-09
1.7.5	2014-09-03

Version	Release Date
1.7.4	2014-07-27
1.7.3	2014-06-22
1.7.2	2014-06-19
1.7.1	2014-06-08
1.7.0	2014-02-27
1.6.3	2014-02-08
1.6.2	2014-02-02
1.6.1	2014-01-12
1.6.0	2014-01-01
1.5.1	2013-11-17
1.5.0	2013-10-21
1.4.2	2013-07-20
1.4.1	2013-07-05
1.4.0	2013-06-05
1.3.3	2012-12-30
1.3.2	2012-12-28
1.3.1	2012-10-18
1.3.0	2012-03-10
1.2.1	2012-01-15
1.2.0	2012-01-07

The change log for all the versions can be found in the official GitHub page.

Examples

Installation or Setup

Less has been one of the most popular CSS Pre-processors, and has also been widely deployed in numerous front-end frameworks like Bootstrap, Foundation, etc. The Less Compiler is a

JavaScript based compiler, which can be obtained from a Content Delivery Network:

```
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.5.1/less.min.js"></script>
```

You need to add your Less document **before** the JavaScript compiler is loaded, using `<link />` tag. The Less stylesheet along with the compiler looks like this:

```
<link rel="stylesheet/less" type="text/css" href="main.less" />
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.5.1/less.min.js"></script>
```

Note: Compiling Less in the client-side (or in the browser) is generally not recommended. It should used only for development or when using dynamic settings that make it not possible to compile server-side.

Sample Less Syntax

The following example is a sample Less file which shows how variables are declared and used, how mixins are defined and called in Less.

```
/* Variables */
@color-base: #87ceeb;

/* Simple mixin to set border */

.set-border(@width; @style; @color) {
  border: @width @style darken(@color, 10%);
}

/* Main CSS */
.class1 {
  background-color: @color-base;
  .set-border(1px; solid; @color-base);
  .class2 {
    background-color: #fff;
    color: @color-base;
    .set-border(1px; solid; #fff);
  }
}
```

The above code when compiled will produce the following CSS: (comments are stripped for brevity)

```
.class1 {
  background-color: #87ceeb;
  border: 1px solid #5bbce4;
}
.class1 .class2 {
  background-color: #fff;
  color: #87ceeb;
  border: 1px solid #e6e6e6;
}
```

Compiling a Less file from the command line

```
lessc [options] <source> [destination]
```

The above command is used to compile Less files in the command line. Options are the various settings that the compiler should use either during compilation or after compilation. Options include `-x` or `--compress` for compressing or minifying the output CSS file, `-sm=on` or `--strict-math=on` for applying math operations only on values enclosed within parenthesis etc. The next comes the path of the source Less file that has to be compiled. Destination is the path and name of the output file. If this is not provided the output is printed out in the command line window itself.

Consider the below Less code

```
/* Filename: test.less */
#demo {
 color: @color;
 background: beige;
 width: 100% / 4;
}
@color: red;
```

Print compiled CSS in Command window:

When the following command is executed in the command line, the test.less file would be compiled and the output will be printed directly on the command window as no destination path is provided.

```
lessc test.less
```

Output:

```
#demo {
  color: red;
  background: beige;
  width: 25%;
}
```

Create a CSS file and write compiled output to the file:

The same file when compiled with the below statement will create a file named test.css in the same path as the test.less file and print/write the output to that CSS file.

```
lessc test.less > test.css
```

Create a CSS file and minify it:

The below command will print/write the output to a CSS file and also compress it at the end.

```
lessc –x test.less > test.css
```

Output:

```
#demo{color:red;background:beige;width:25%}
```

With Strict Math option enabled:

When the strict match option is enabled, the output will be as follows because the values for `width` is not enclosed within braces.

```
lessc -sm=on test.less > test.css
```

Output:

```
#demo {
  color: red;
  background: beige;
  width: 100% / 4;
}
```

Nesting in Less

In Less you can write much more simple CSS rules and also keep them well formatted, so instead of writing this code:

CSS

```
.item {
  border: 1px solid;
  padding: 4px;
}
.item .content, .item .image {
  float: left;
}
.item .content {
  font-size: 12px;
}
.item .image {
  width: 300px;
}
```

you can just write this:

Less

```
.item {
  border: 1px solid;
  padding: 4px;
  .content, .image {
    float: left;
  }
  .content {
    font-size: 12px;
  }
  .image {
    width: 300px;
```

```
  }
}
```

and Less will compile that code into the normal CSS we all know.

Joining Files - Imports

The `@import` statement allows you to insert CSS/Less code from another file into your own CSS/Less file.

```
.foo {
  background: #900;
}
@import "my-other-css-file.css";
@import "my-other-less-file.less";
```

Read Getting started with less online: https://riptutorial.com/less/topic/4716/getting-started-with-less

Chapter 2: Color Operation Functions

Syntax

- contrast(<reference-color>, <output-for-light-refcolor>, <output-for-dark-refcolor>, <threshold>)
- lighten(<reference-color>, <amount>, <method>)
- darken(<reference-color>, <amount>, <method>)

Parameters

Parameter	Details
reference-color	The color based on which the color operation should be performed.
output-for-light-ref-color	The color value that should be output when reference color is a dark color. This is optional and default value is white.
output-for-dark-ref-color	The color value that should be output when reference color is a light color. This is optional and default value is black.
threshold	This is a percentage value which defines when the reference color is considered as a dark color and when it is considered as a light color. Color comparison is done using gamma-corrected luma values. This is optional and the default value is 43%
amount	This is a percentage value which specifies the amount by which the reference color should be darkened or lightened.
method	This can either be `absolute` or `relative` and defines whether the adjustment should be relative to the current value or not. It is an optional field and the default value is absolute.

Remarks

Contrast

In the contrast function, the output-for-dark-ref-color and output-for-light-ref-color can be provided in any order. The function automatically identifies which is the dark color (to be used when ref color is light) and which is light color (to be used when ref color is dark) based on their own luma values.

Examples

Setting text color depending on the darkness or lightness of background color

```
#demo-element {
  background: @theme-color;
  color: contrast(@theme-color, black, white, 50%);
}

@theme-color: red;
```

The above example will set the text color of the element as `white` if the `background-color` is dark and vice-versa. This is achieved using the `contrast()` color operation function.

The contrast function accepts four parameters where the first one is the reference color based on which the output should be provided. A dark color that should be output when the reference color is light and a light that should be output when reference color is dark . Threshold is a percentage which defines when a color is considered as a light color and when dark.

A demo of this example can be found here.

Set a darker or lighter shade of the background color for another property

Darken

```
#demo {
  @refcolor: #f0b9b8;
  background: @refcolor;
  border: 1px solid darken(@refcolor, 25%);
}
```

The above code makes use of the `darken()` function to set the border color as a shade that is 25% darker than the reference color (which is also the background color).

Less compiler cannot read the value assigned to one property and use it for another and so a separate variable should be defined. This variable will be directly assigned to the background color whereas for the border color, the `darken` function is used to get a darker shade.

Lighten

```
#demo {
  @refcolor: #f0b9b8;
  background: @refcolor;
  box-shadow: 2px 2px 0px lighten(@refcolor, 5%);
}
```

The above code makes use of the `lighten()` function to set the shadow color as a shade that is 5% lighter than the reference color (which is also the background color).

Changing opacity

It is possible to change the opacity of a color with `fade()` function.

`fade()` takes 2 parameters:

- a color
- opacity (in %)

Example:

```
@elegant: #eeffgg;

.light-elegant {
    background-color: fade(@elegant, 20%);
}
```

```
<div class="light-elegant">
    I have a 20% elegant background!
</div>
```

Read Color Operation Functions online: https://riptutorial.com/less/topic/4718/color-operation-functions

Chapter 3: Extend

Introduction

This is related to the extend functionality of less, which was introduced in v1.4.0.

"Extend is a Less pseudo-class which merges the selector it is put on with ones that match what it references." [ref]

Syntax

1. selector1:extend(css selector){ //other styles go here}
2. selector1{ &:extend(css selector)); //other styles go here }

Parameters

Parameter	Details
css selector	This is any generic CSS selector, and may include `.class`, `#id`, `::pseudoElements`, etc

Examples

Basic Example

The following Less:

```
.paragraph{
  font-size: 12px;
  color: blue;
  background: white;
}
.parent{
  font-size: 14px;
  color: black;
  background: green;
  .nestedParagraph:extend(.paragraph){

  }
}
```

will compile into the following css:

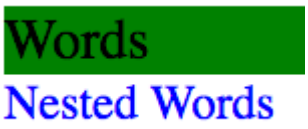
```
.paragraph,
.parent .nestedParagraph {
  font-size: 12px;
  color: blue;
```

```
  background: white;
}
.parent {
  font-size: 14px;
  color: black;
  background: green;
}
```

We have applied the styles for `.paragraph` to the `.parent .nestedParagraph` element! Assuming our HTML is:

```
<div class="parent">
  Words
  <div class="nestedParagraph">
    Nested Words
  </div>
</div>
```

Our output will be



Nested Words

This is one way to easily apply many pre-configured styles to deeply nested components.

Extend may additionally be used with &, the parent select feature, the below compiles to the same as above.

```
.paragraph{
  font-size: 12px;
  color: blue;
  background: white;
}
.parent{
  font-size: 14px;
  color: black;
  background: green;
  .nestedParagraph{
    &:extend(.paragraph);
  }
}
```

Multiple extends on a single selector

The following Less

```
.paragraph{
  font-size: 12px;
  color: darkgrey;
  background: white;
}
```

```
.special-paragraph{
  font-size: 24px;
  font-weight: bold;
  color: black;
}

.parent{
  background: lightgrey;
  .nestedParagraph{
    &:extend(.paragraph);
    &:extend(.special-paragraph);
  }
}
```

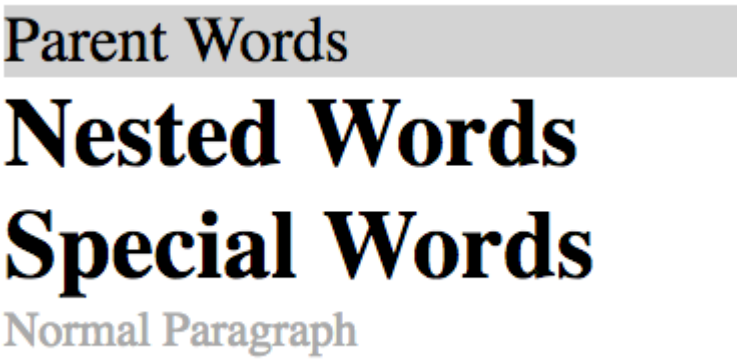
Will compile to

```
.paragraph,
.parent .nestedParagraph {
  font-size: 12px;
  color: darkgrey;
  background: white;
}
.special-paragraph,
.parent .nestedParagraph {
  font-size: 24px;
  font-weight: bold;
  color: black;
}
.parent {
  background: lightgrey;
}
```

With the provided html:

```
<div class="parent">
  Parent Words
  <div class="nestedParagraph">
    Nested Words
  </div>
</div>

<div class="special-paragraph">
  Special Words
</div>

<div class="paragraph">
  Normal Paragraph
</div>
```

We see the following result:



In this particular example, `nestedParagraph` would like to use `paragraph`'s styles, with the overrides from `special-paragraph`. Styles may easily be overridden by paying attention to the order elements are extended in.

Extending nested selectors

You may also extend nested selectors. The below Less

```
.otherChild{
  color: blue;
}

.otherParent{
  color: red;
  .otherChild{
    font-size: 12px;
    color: green;
  }
}

.parent{
  .nestedParagraph{
    &:extend(.otherParent .otherChild);
  }
}
```

Will compile to

```
.otherChild {
  color: blue;
}
.otherParent {
  color: red;
}
.otherParent .otherChild,
.parent .nestedParagraph {
  font-size: 12px;
  color: green;
}
```

With the following html

```
<div class="otherParent">
  Other Parent Words
  <div class="otherChild">
    Other Nested Words
  </div>
</div>

<div class="parent">
  Parent Words
  <div class="nestedParagraph">
    Nested Words
  </div>
</div>
```

The result is

Other Parent Words
Other Nested Words
Parent Words
Nested Words

The font color for the nested paragraph is green, not blue! This shows we can extend nested selectors!

Less Extend only supports exact matching

The following Less

```
div.paragraph{
  color: blue;
}

*.paragraph{
  color: green;
}

.otherClass.paragraph{
  color: red;
}

.paragraph.otherClass{
  color: darkgrey;
}

.parent{
  .nestedParagraph{
    &:extend(.paragraph);
  }
}
```

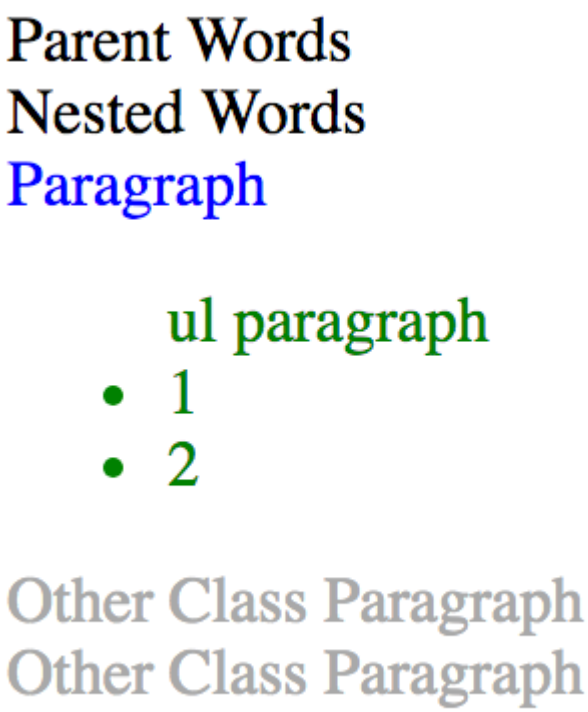
Will compile into

```
div.paragraph {
  color: blue;
}
*.paragraph {
  color: green;
}
.otherClass.paragraph {
  color: red;
}
.paragraph.otherClass {
  color: darkgrey;
}
```

Using the following HTML

```
<div class="parent">
  Parent Words
  <div class="nestedParagraph">
    Nested Words
  </div>
</div>

<div class="paragraph">
  Paragraph
</div>

<ul class="paragraph">
  ul paragraph
  <li>1</li>
  <li>2</li>
</ul>

<div class="otherClass paragraph">
  Other Class Paragraph
</div>

<div class="paragraph otherClass">
  Other Class Paragraph
</div>
```

Our result is



We can see that Less Extend only supports exact matching, as the Nested Words do not have styled applied to them.

Pseudo Elements

The following Less

```
.addDivider::before{
  content: "";
  height: 80%;
  background: white;
  width: 1px;
  position: absolute;
  top: 10%;
  left: 0;
}

.nav-bar{
  background: black;
  display: flex;
  flex-direction: row;
  width: 400px;
  .nav-item{
    color: white;
    width: 100px;
    list-style-type: none;
    position: relative;
    text-align: center;
    padding: 0;
    &:not(:first-child){
      &::before{
        &:extend(.addDivider::before);
      }
    }
```

```
  }
}
```

Will compile into the following CSS

```
.addDivider::before,
.nav-bar .nav-item:not(:first-child)::before {
  content: "";
  height: 80%;
  background: white;
  width: 1px;
  position: absolute;
  top: 10%;
  left: 0;
}
.nav-bar {
  background: black;
  display: flex;
  flex-direction: row;
  width: 400px;
}
.nav-bar .nav-item {
  color: white;
  width: 100px;
  list-style-type: none;
  position: relative;
  text-align: center;
  padding: 0;
}
```

Using the following HTML

```
<div class="nav-bar">
  <div class="nav-item">one</div>
  <div class="nav-item">two</div>
  <div class="nav-item">three</div>
  <div class="nav-item">four</div>
</div>
```

Our result is

We have defined a default divider pseudoclass which we have added into a nested element! The white borders can now be added to other elements using `extend`.

Read Extend online: https://riptutorial.com/less/topic/9706/extend

Chapter 4: Guards (for writing conditional mixins)

Syntax

- **Mixin with Guards**
- .mixin-name(<arguments>) when <is-negation> (<ref-variable> <operator> <value>)
- **CSS Guards**
- <selector> when <is-negation> (<ref-variable> <operator> <value>)

Parameters

Parameter	Details
arguments	The variables that are passed to the parametric mixin. Arguments are optional.
is-negation	This indicates whether the guard condition is a `not` condition or not. For example `when not (@type = error)` means the mixin will be used whenever the value of `@type` is not *error*.
ref-variable	This is the variable whose value determines which mixin's properties should be applied to the element. This is mandatory.
operator	This is the operator that is used for evaluating the condition. It can be `=`, `>`, `<`, `>=`, `<=`. This is not mandatory. When the operator and value is not given, the compiler assumes the condition to be `<ref-variable> = true`.
value	The value that is used for evaluating the condition. This is optional but becomes mandatory when an operator is provided.

Examples

Style an element based on a variable's value

```
.set-colors(@type) when (@type = error) {
  @base-color: #d9534f;
  background: @base-color;
  color: contrast(@base-color, lighten(@base-color, 25%), darken(@base-color, 25%));
  border: 1px solid contrast(@base-color, lighten(@base-color, 25%), darken(@base-color,
25%));
}
.set-colors(@type) when (@type = info) {
  @base-color: #5bc0de;
  background: @base-color;
  color: contrast(@base-color, lighten(@base-color, 5%), darken(@base-color, 5%));
```

```
  border: 1px solid contrast(@base-color, lighten(@base-color, 5%), darken(@base-color, 5%));
}
.set-colors() {
  background: white;
  color: black;
  border: 1px solid black;
}

.error-message {
  .set-colors(error);
}
.info-message {
  .set-colors(info);
}
.default-div {
  .set-colors;
}
```

In the above example, the `background`, `border` and `color` are assigned based on the type of element. If the element is a default text `div` then the background will be white whereas the text and border would be black. If it is an "error" message display `div` or an "info" message display `div` then the colors are assigned based on the type.

The compiled CSS output would be as follows:

```
.error-message {
  background: #d9534f;
  color: #f0b9b8;
  border: 1px solid #f0b9b8;
}
.info-message {
  background: #5bc0de;
  color: #46b8da;
  border: 1px solid #46b8da;
}
.default-div {
  background: white;
  color: black;
  border: 1px solid black;
}
```

Read Guards (for writing conditional mixins) online: https://riptutorial.com/less/topic/4795/guards--for-writing-conditional-mixins-

Chapter 5: Loops

Examples

Writing a simple for loop

Usage of loops is an excellent way to keep the code DRY and avoid repetition. Unlike in Sass, there are no built-in `@for` or `@each` directive in Less for writing loops but it can still be written using recursive mixins. A recursive mixin is nothing but a mixin which keeps calling itself.

There are four key components to a loop written using Less and they are as follows:

- A mixin with guard expressions. The guard is used to terminate the loop when the loop's exit criteria is met. In terms of a JavaScript for loop (`for([initialization]; [condition]; [final-expression])`), the guard is the `[condition]`.
- A primary call to the mixin to execute the first iteration. This primary call to the mixin can be made from within a selector block (if the mixin doesn't have a selector wrapping all its contents) or from outside a selector block (if the mixin has a selector wrapping its contents). In terms of a JavaScript for loop, this primary call serves as the `[initialization]` as it sets the base value for the counter-like variable.
- A call to the mixin from within itself to make it recursive. This call typically passes an incremented or a decremented value of the counter variable as the argument. Thus it invokes the subsequent iterations. In terms of a JS for loop, this does the `[final-expression]` along with the next call.
- Last but not the least, the other contents of the mixin which is equivalent to the `statement` in a typical for loop syntax.

Below is a simple for loop written in Less that creates multiple `#img*` selectors (where * is a number) and also sets the `background-image` property as `image*.png`.

```
.for-loop(@index) when (@index > 0) { /* recursive mixin with guard expression - condition */

  /* the statement */
  #img@{index} {
    background-image: url("http://mysite.com/image@{index}.png");
  }
  /* end of the statement */

  .for-loop(@index - 1); /* the next iteration's call - final-expression*/
}
.for-loop(3); /* the primary call - initialization */
```

Compiled CSS:

```
#img3 {
  background-image: url("http://mysite.com/image3.png");
}
#img2 {
  background-image: url("http://mysite.com/image2.png");
```

```
}
#img1 {
  background-image: url("http://mysite.com/image1.png");
}
```

Writing a for-each loop

A for-each loop in Less has the same key components as a for loop except for the following differences:

- A variable which contains the list of items that has to be iterated over.
- An `extract()` function to extract each item in the variable based on the loop's index.
- A `length()` function to calculate the length of the array (that is, the no of items in the list) and use it in the primary mixin call (for `[initialization]`).

Below is a sample for-each loop written in Less that iterates through each item in the `@images` variable, creates a `#id` selector where the `id` is the same as the item/image name and also sets the background image property for it.

```
@images: cat, dog, apple, orange; /* the array list of items */

.for-each-loop(@index) when (@index > 0) { /* recursive mixin call with guard - condition */

  @image: extract(@images, @index); /* extract function to fetch each item from the list */

 /* the statement */
  #@{image} {
    background-image: url("http://mysite.com/@{image}.png");
  }
  /* end of the statement */

  .for-each-loop(@index - 1); /* the next iteration's call - final-expression */
}

.for-loop(length(@images)); /* the primary call with length() function - initialization */
```

Compiled CSS:

```
#orange {
  background-image: url("http://mysite.com/orange.png");
}
#apple {
  background-image: url("http://mysite.com/apple.png");
}
#dog {
  background-image: url("http://mysite.com/dog.png");
}
#cat {
  background-image: url("http://mysite.com/cat.png");
}
```

Read Loops online: https://riptutorial.com/less/topic/5424/loops

Chapter 6: Mixins

Examples

Introduction

Mixins are similar to defining and calling functions. Say, if we need to create a repetitive style, mixins are handy. Mixins may or may not take parameters. For e.g.:

```
.default-round-borders {
          border-radius: 5px;
     -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
}

.round-borders (@radius) {
          border-radius: @radius;
     -moz-border-radius: @radius;
  -webkit-border-radius: @radius;
}
```

We have two types of declarations above. One takes in a parameter and the other doesn't. Let's see how this is being used somewhere:

```
@sky-blue: #87ceeb;
@dark-sky-blue: #baffff;

#header {
  background: @sky-blue;
  .default-round-borders;
}

.btn {
  background: @dark-sky-blue;
  .round-borders(3px);
}
```

The above code, compiled all together will give an output like this:

```
#header {
  background: #87ceeb;
          border-radius: 5px;
     -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
}
.btn {
  background: #baffff;
          border-radius: 3px;
     -moz-border-radius: 3px;
  -webkit-border-radius: 3px;
}
```

Prevent a mixin definition from appearing in the compiled CSS file

```
.default-settings() {
  padding: 4px;
  margin: 4px;
  font-size: 16px;
  border: 1px solid gray;
}

#demo {
  .default-settings;
}
```

The above example when compiled would only produce the following output. The `.default-settings()` mixin definition would not be output in the compiled CSS file because parenthesis is added after it.

```
#demo {
  padding: 4px;
  margin: 4px;
  font-size: 16px;
  border: 1px solid gray;
}
```

If these parenthesis are not attached, the Less compiler will treat the mixin definition also as CSS selector and so will print it also in the output CSS file.

Add !important to every property in a mixin without manually typing it

When we attach the `!important` keyword to a mixin call, the Less compiler will automatically add the `!important` to all properties that are present within the mixin.

For example, consider the below mixin:

```
.set-default-props() {
  margin: 4px;
  padding: 4px;
  border: 1px solid black;
  font-weight: normal;
}
```

When the `!important` is attached to the mixin call like below,

```
#demo {
  .set-default-props() !important;
}
```

the compiled CSS would look as follows:

```
#demo {
  margin: 4px !important;
  padding: 4px !important;
```

```
  border: 1px solid black !important;
  font-weight: normal !important;
}
```

> **Note:** Usage of `!important` is considered as bad practice and must be used only as the last resort.

Read Mixins online: https://riptutorial.com/less/topic/4748/mixins

Chapter 7: Parent selectors

Remarks

As at the time of writing (Aug '16), parent selector (`&`) always refers to the full parent selector chain right till the top most level. It cannot be used to select just the immediate parent or the root most ancestor alone.

That is, in the below code `&#type1` would resolve to `#demo-container .content#type1` and not just `.content` or just `#demo-container`.

```
 #demo-container {
  padding: 4px;
  border: 1px solid gray;
  #heading {
    padding: 4px;
    font-size: 20px;
  }
  .content {
    padding: 2px;
    font-size: 18px;
    &#type1 {
      color: chocolate;
    }
  }
}
```

Examples

Basic parent selector

The `&` operator is the parent selector. When used in or as a selector, it is replaced with the full parent selectors (entire sequence of selectors right upto to the topmost level of a nested block) in the final CSS output.

It is useful when creating nested rules that require using the parent selector in a different way than default, like changing the order of the parent selector placement or to concatenate it with other selectors.

```
a {
  text-decoration: none;
  &:hover {
    text-decoration: underline;
  }
}
```

Results in the following CSS where the parent selector `a` was concatenated with the `:hover` rule:

```
a {
```

```
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
```

One big advantage of using parent selectors wherever possible is the reduction of repetition of selectors.

Changing the selector order within a nested block

Less allows the usage of the parent selector (`&`) anywhere in a complex selector and thus allows changing styles when the current element is within another element which gives it a different context:

For example, in the below code the parent selector is placed at the end and thus it actually becomes the child's selector in the compiled CSS.

```
a {
  color: blue;
  .disabled-section & {
    color: grey;
  }
}
```

Compiled CSS:

```
a {
  color: blue;
}
.disabled-section a {
  color: grey;
}
```

Select sibling elements that have the same class without repeating selector

Less doesn't put any restrictions on the number of times the parent selector (`&`) can be used in a complex selector and so, we can use it more than once like in the below examples to select sibling elements without the need to repeat the selector.

```
.demo {
  border: 1px solid black; /* add border to all elements with demo class */
  & + & { /* select all .demo that have another .demo sibling immediately prior */
    background: red;
  }
  & + & + & { /* select all .demo that have two .demo sibling immediately prior */
    background: chocolate;
  }
  & ~ & { /* select all .demo elements that have another .demo sibling prior */
    color: beige;
  }
}
```

The above code when compiled will result in the following CSS:

```
.demo {
  border-left: 1px solid black;
}
.demo + .demo {
  background: red;
}
.demo + .demo + .demo {
  background: chocolate;
}
.demo ~ .demo {
  color: beige;
}
```

Read Parent selectors online: https://riptutorial.com/less/topic/5159/parent-selectors

Chapter 8: Variables

Examples

Introduction

In Less, unlike Sass or Shell, the variables are declared by having names starting with a `@` symbol. For example:

```
@sky-blue: #87ceeb;

body {
  background-color: @sky-blue;
}
```

The above example gives you:

```
body {
  background-color: #87ceeb;
}
```

Here it explains how to declare a variable and make use of them.

Operations in Colour

Consider the following example:

```
@sky-blue: #87ceeb;
@dark-sky-blue: @sky-blue + #333;

body {
  background-color: @dark-sky-blue;
}
```

The above example gives you:

```
body {
  background-color: #baffff;
}
```

Here it explains how to declare a variable and also make operations on a particular variable as well.

Concatenate value of two or more variables

To concatenate the value of two or more variables into a single string and print it as the output, we need to make use of interpolation.

The following Less code,

```
#demo:after {
  @var1: Hello;
  @var2: World!!!;
  content: "@{var1} @{var2}";
}
```

when compiled would set **"Hello World!!!"** as value to the `content` property. Below is the compiled CSS:

```
#demo:after {
  content: "Hello World!!!";
}
```

If the value of two or more variables just need to be placed next to each other in space separated manner then interpolation is not required.

```
#demo {
  @top: 4px;
  @right: 2px;
  @bottom: 6px;
  @left: 4px;
  padding: @top @right @bottom @left;
}
```

When the above code is compiled, it would produce the following CSS.

```
#demo {
  padding: 4px 2px 6px 4px;
}
```

This approach will not work when there should be no space between the variable values (or) when the resultant string needs to be within quotes. For those cases, usage of interpolation would be mandatory.

Referencing a Variable Within a CSS Function

By default, LESS will use its own `calc()` unless told otherwise. So:

```
@column-count: 2;

.class-example {
    width: calc(100% / @column-count);
}
```

Would compile to this:

```
.class-example {
    width: 50%;
}
```

While it is our desired width, LESS has used it's own `calc()` function to calculate the `width`. The `calc()` function never makes it to our CSS. If you would like LESS to not use its `calc()` function, you need to escape your values like this:

```
width: calc(~"100% - @{column-count}");
```

Here we've prepended our values with a `~` and wrapped them in quotation marks. Variables can be referenced as well, but you must wrap the variable name in `{ }` brackets. This allows you use the CSS `calc()` function for more complex calculations like this:

```
@column-count: 2;
@column-margin: 24px;

.class-example {
    width: calc("~(100% / @{column-count}) - @{column-margin}");
}
```

This compiles to:

```
.class-example {
    width: calc((100/2) - 24px);
}
```

Variables can make your recursive work easy

Variable declaration

```
@buttonColor: #FF0000;
```

/* Now you can use @buttonColor variable with css Functions. */

```
.product.into.detailed {
    additional-attributes{
        .lib-table-button(
         background-color: @buttonColor;
         );
    }
}
```

Here function `lib-tabgle-button` used variable to set background color

Read Variables online: https://riptutorial.com/less/topic/4749/variables

Credits

S. No	Chapters	Contributors
1	Getting started with less	celtschk, Community, Harry, Praveen Kumar, Shlomi Haver, Stephen Leppik
2	Color Operation Functions	Harry, Mistalis
3	Extend	Hodrobond
4	Guards (for writing conditional mixins)	Harry
5	Loops	Harry
6	Mixins	Harry, Praveen Kumar
7	Parent selectors	Harry, Jens
8	Variables	Harry, Hynes, Praveen Kumar, Ronak Chauhan