{less}

It's CSS, with just a little more.

Use with Node.js:

```
npm install -g less
> lessc styles.less styles.css
```

Or the browser:

```
<link rel="stylesheet/less" type="text/css" href="styles.less" />
<script src="https://cdn.jsdelivr.net/npm/less@4
(https://cdn.jsdelivr.net/npm/less@4)" ></script>
```

```
(https://cdn.jsdelivr.net/npm/less@4)" ></script>
```

View the Less.js changelog (https://github.com/less/less.js/blob/master/CHANGELOG.md)

Overview

Less (which stands for Leaner Style Sheets) is a backwards-compatible language extension for CSS. This is the official documentation for Less, the language and Less.js, the JavaScript tool that converts your Less styles to CSS styles.

Because Less looks just like CSS, learning it is a breeze. Less only makes a few convenient additions to the CSS language, which is one of the reasons it can be learned so quickly.

- *For detailed documentation on Less language features, see Features (../features/)*
- *For a list of Less Built-in functions, see Functions (../functions/)*
- *For detailed usage instructions, see Using Less.js (../usage/)*
- *For third-party tools for Less, see Tools (../tools/)*

What does Less add to CSS? Here's a quick overview of features.

Variables

These are pretty self-explanatory:

```
@width: 10px;
@height: @width + 10px;

#header {
  width: @width;
  height: @height;
}
```

Outputs:

```
#header {
  width: 10px;
  height: 20px;
}
```

Learn More About Variables (../features/#variables-feature)

Mixins

Mixins are a way of including ("mixing in") a bunch of properties from one rule-set into another rule-set. So say we have the following class:

```
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
```

And we want to use these properties inside other rule-sets. Well, we just have to drop in the name of the class where we want the properties, like so:

```
#menu a {
  color: #111;
  .bordered();
}

.post a {
  color: red;
  .bordered();
}
```

The properties of the `.bordered` class will now appear in both `#menu a` and `.post a`. (Note that you can also use `#ids` as mixins.)

Learn More About Mixins (../features/#mixins-feature)

Nesting

Less gives you the ability to use nesting instead of, or in combination with cascading. Let's say we have the following CSS:

```
#header {
  color: black;
}
#header .navigation {
  font-size: 12px;
}
#header .logo {
  width: 300px;
}
```

In Less, we can also write it this way:

```
#header {
  color: black;
  .navigation {
    font-size: 12px;
  }
  .logo {
    width: 300px;
  }
}
```

The resulting code is more concise, and mimics the structure of your HTML.

You can also bundle pseudo-selectors with your mixins using this method. Here's the classic clearfix hack, rewritten as a mixin (`&` represents the current selector parent):

```
.clearfix {
  display: block;
  zoom: 1;

  &:after {
    content: " ";
    display: block;
    font-size: 0;
    height: 0;
    clear: both;
    visibility: hidden;
  }
}
```

Learn More About Parent Selectors (../features/#parent-selectors-feature)

Nested At-Rules and Bubbling

At-rules such as `@media` or `@supports` can be nested in the same way as selectors. The at-rule is placed on top and relative order against other elements inside the same ruleset remains unchanged. This is called bubbling.

```
.component {
  width: 300px;
  @media (min-width: 768px) {
    width: 600px;
    @media  (min-resolution: 192dpi) {
      background-image: url(/img/retina2x.png);
    }
  }
  @media (min-width: 1280px) {
    width: 800px;
  }
}
```

outputs:

```
.component {
  width: 300px;
}
@media (min-width: 768px) {
  .component {
    width: 600px;
  }
}
@media (min-width: 768px) and (min-resolution: 192dpi) {
  .component {
    background-image: url(/img/retina2x.png);
  }
}
@media (min-width: 1280px) {
  .component {
    width: 800px;
  }
}
```

Operations

Arithmetical operations `+`, `-`, `*`, `/` can operate on any number, color or variable. If it is possible, mathematical operations take units into account and convert numbers before adding, subtracting or comparing them. The result has leftmost explicitly stated unit type. If the conversion is impossible or not meaningful, units are ignored. Example of impossible conversion: px to cm or rad to %.

```
// numbers are converted into the same units
@conversion-1: 5cm + 10mm; // result is 6cm
@conversion-2: 2 - 3cm - 5mm; // result is -1.5cm

// conversion is impossible
@incompatible-units: 2 + 5px - 3cm; // result is 4px

// example with variables
@base: 5%;
@filler: @base * 2; // result is 10%
@other: @base + @filler; // result is 15%
```

Multiplication and division do not convert numbers. It would not be meaningful in most cases - a length multiplied by a length gives an area and css does not support specifying areas. Less will operate on numbers as they are and assign explicitly stated unit type to the result.

```
@base: 2cm * 3mm; // result is 6cm
```

You can also do arithmetic on colors:

```
@color: (#224488 / 2); // result is #112244
background-color: #112244 + #111; // result is #223355
```

However, you may find Less's Color Functions (../functions/#color-operations) more useful.

From 4.0, No division is performed outside of parens using `/` operator.

```
@color: #222 / 2; // results in `#222 / 2`, not #111
background-color: (#FFFFFF / 16); //results is #101010
```

You can change Math (../usage/#less-options-math) setting, if you want to make it always work, but not recommended.

calc() exception

Released v3.0.0 (https://github.com/less/less.js/blob/master/CHANGELOG.md)

For CSS compatibility, `calc()` does not evaluate math expressions, but will evaluate variables and math in nested functions.

```
@var: 50vh/2;
width: calc(50% + (@var - 20px));  // result is calc(50% + (25vh - 20px))
```

Escaping

Escaping allows you to use any arbitrary string as property or variable value. Anything inside `~"anything"` or `~'anything'` is used as is with no changes except interpolation (../features/#variables-feature-variable-interpolation).

```
@min768: ~"(min-width: 768px)";
.element {
  @media @min768 {
    font-size: 1.2rem;
  }
}
```

results in:

```
@media (min-width: 768px) {
  .element {
    font-size: 1.2rem;
  }
}
```

Note, as of Less 3.5, you can simply write:

```
@min768: (min-width: 768px);
.element {
  @media @min768 {
    font-size: 1.2rem;
  }
}
```

In 3.5+, many cases previously requiring "quote-escaping" are not needed.

Functions

Less provides a variety of functions which transform colors, manipulate strings and do maths. They are documented fully in the function reference (../functions/).

Using them is pretty straightforward. The following example uses percentage to convert 0.5 to 50%, increases the saturation of a base color by 5% and then sets the background color to one that is lightened by 25% and spun by 8 degrees:

```
@base: #f04615;
@width: 0.5;

.class {
  width: percentage(@width); // returns `50%`
  color: saturate(@base, 5%);
  background-color: spin(lighten(@base, 25%), 8);
}
```

See: Function Reference (../functions/)

Namespaces and Accessors

(Not to be confused with CSS `@namespace` (http://www.w3.org/TR/css3-namespace/) or namespace selectors (http://www.w3.org/TR/css3-selectors/#typenmsp)).

Sometimes, you may want to group your mixins, for organizational purposes, or just to offer some encapsulation. You can do this pretty intuitively in Less. Say you want to bundle some mixins and variables under `#bundle`, for later reuse or distributing:

```
#bundle() {
  .button {
    display: block;
    border: 1px solid black;
    background-color: grey;
    &:hover {
      background-color: white;
    }
  }
  .tab { ... }
  .citation { ... }
}
```

Now if we want to mixin the `.button` class in our `#header a`, we can do:

```
#header a {
  color: orange;
  #bundle.button();  // can also be written as #bundle > .button
}
```

Note: append `()` to your namespace (e.g. `#bundle()`) if you don't want it to appear in your CSS output i.e. `#bundle .tab`.

Maps

As of Less 3.5, you can also use mixins and rulesets as maps of values.

```
#colors() {
  primary: blue;
  secondary: green;
}

.button {
  color: #colors[primary];
  border: 1px solid #colors[secondary];
}
```

This outputs, as expected:

```
.button {
  color: blue;
  border: 1px solid green;
}
```

See also: Maps (../features/#maps-feature)

Scope

Scope in Less is very similar to that of CSS. Variables and mixins are first looked for locally, and if they aren't found, it's inherited from the "parent" scope.

```
@var: red;

#page {
  @var: white;
  #header {
    color: @var; // white
  }
}
```

Like CSS custom properties, mixin and variable definitions do not have to be placed before a line where they are referenced. So the following Less code is identical to the previous example:

```
@var: red;

#page {
  #header {
    color: @var; // white
  }
  @var: white;
}
```

See also: Lazy Loading (../features/#variables-feature-lazy-loading)

Comments

Both block-style and inline comments may be used:

```
/* One heck of a block
 * style comment! */
@var: red;

// Get in line!
@var: white;
```

Importing

Importing works pretty much as expected. You can import a `.less` file, and all the variables in it will be available. The extension is optionally specified for `.less` files.

```
@import "library"; // library.less
@import "typo.css";
```

Learn More About Imports (../features/#imports-feature)