**Less Css:-**

The dynamic stylesheet language.

LESS extends CSS with dynamic behavior such as variables, mixins, operations and functions.

LESS runs on both the server-side (with Node.jsand Rhino) or client-side (modern browsers only).

1. What is LESS
2. Why use LESS?
3. Installation of LESS
4. Variables
5. Mixins
6. Cascading + Nesting
7. &combinator
8. Operations
9. Comments
10. @import
11. String interpolation
12. Escaping
13. Pre-compile
14. Post-compile
15. LESS Elements
16. Minification and compression
17. Alternative to LESS

## What is LESS

A recent [internet survey](http://css-tricks.com/poll-results-popularity-of-css-preprocessors/) from June 2012 showed that almost half (46%) of respondents have not even tried a CSS preprocessor.  Although not a scientific study, this indicates that there is a large gap between those that have tried to use a CSS preprocessor such as LESS or SASS, and those that are not currently employing a CSS preprocessor.  Further, of those that have a preference as to which CSS preprocessor to use, 51% prefer to use LESS and 41% use SASS. I encourage you to try both LESS and SASS, and decide which preprocessor works best for you.  Each has their own advantages and disadvantages, which will not be discussed in this article.

So, what exactly is LESS?

* LESS is a programming
* LESS compiles to CSS3
* LESS is a CSS Preprocessor
* LESS syntax is modelled after traditional CSS
* LESS is often referred to as “dynamic css”

## Why use LESS?

Why should a web developer or front-end UI engineer use LESS over traditional CSS?

* Save time
* Reduce mistakes
* Reduce repetition (DRY)
* It makes logical sense to break out CSS into multiple files, but traditional CSS @import is terrible for performance as it creates a new HTTP request for each file
* LESS is cool!

CSS preprocessors such as LESS do not extend the functionality of CSS; only the browser vendors can change the way CSS is implemented.  Instead, a CSS preprocessor makes the development flow for a developer much easier.  If you are already combining, minimizing and compressing your CSS, then using a CSS preprocessor is a logical next-step.

## Installation of LESS

I covered the installation of LESS in my previous article, entitled [starting with LESS](http://brianflove.com/2012/09/06/starting-with-less/), so I will not be diving into the details here.  For Windows users, I recommend [WinLess](http://winless.org/), which is a simple installation and allows you to easily compile your less files into CSS.  For Macintosh users, I recommend you install[homebrew](http://mxcl.github.com/homebrew/), then:

|  |  |
| --- | --- |
| 1  2  3  4 | $ brew install git  $ brew install node  $ curl https://npmjs.org/install.sh | sh  $ npm install less --global |

## What can LESS do?

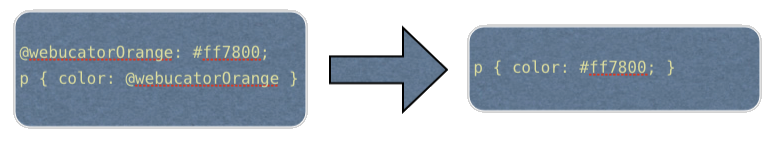
LESS introduces many features that CSS doesn’t provide, such as:

* Variables in your CSS
* Mixins (think functions) that allow you to reuse rulesets
* Nesting of styles to mimic your DOM structure
* Simple mathematical operators: +, -, \*, / of numbers and colors
* Mathematical operations such as floor(), ceiling() and round()
* Color operations such as darken(), lighten(), fadein() and fadeout()

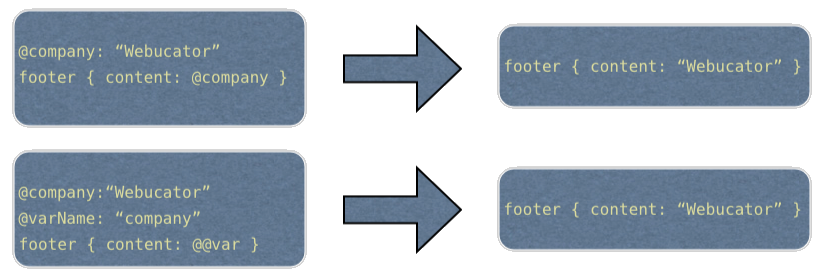
Let’s look at these features in depth.

## Variables

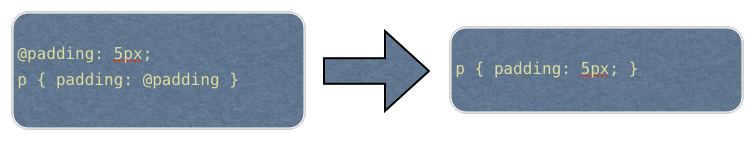
* Start with @ symbol
* Can store hexadecimal colors, e.g. #333 or #fefefe
* Can store stings, e.g. “Webucator, Inc.”
* Can store sizes, e.g. 10px

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/color-variables-with-less.png)

In the example above, we are defining a variable named @webucatorOrange, which contains the specific orange color used on our web site.  We can then use this variable anywhere throughout our LESS code.  This example also shows the CSS that is output when compiling the LESS to CSS.

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/string-variables-in-less.png)

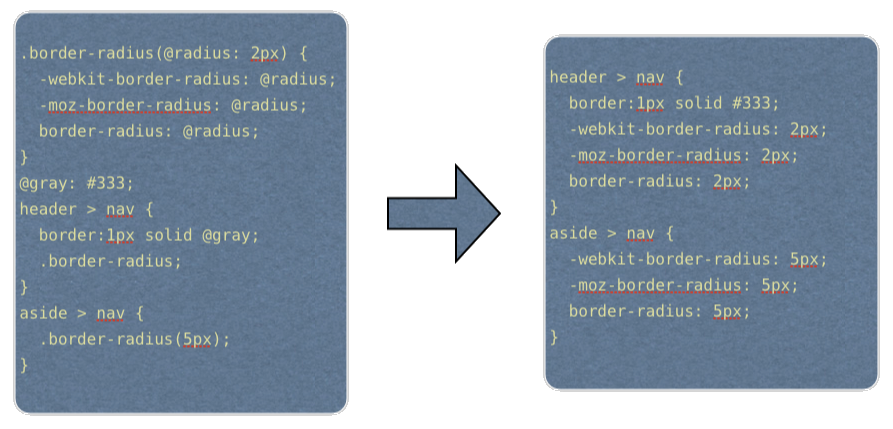
In this example, we are defining a variable named @company, which stores a string.  On the second line we are defining a variable @varName, which stores the string “company”.  We can evaluate this variable name, and then evaluate the result of that.  The CSS that is created is shown on the right.

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/size-variables-in-less.png)

In this example, we are defining a variable named @padding, which stores a size of 5 pixels.  We can then use this @padding size throughout our LESS code.  And, as we will see later, we can use this variable in mathematical operations or functions.

## Mixins

* Include properties from one ruleset to another
* Reuse code
* Can accept parameters
* Can define default value for parameters
* @arguments is a special variable that contains the ordered value stored in all parameters

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/mixins-in-less.png)

In this example above we are defining a mixin called “border-radius”, which is used to replicate the cross-browser functionality of the CSS property border-radius.  I find it good practice to name the mixins as close as possible to the traditional CSS properties.  Mixins can accept parameters, which can also have a default value.  Mixins do not need to have parameters.  In this example, the mixin has a single parameter named @radius with the default value of 2 pixels.  We can then use this parameter in our resulting CSS rules.  I am also defining a variable named @gray that stores a color.  On the right you can see the CSS that is output when we compile the LESS.  Wherever we call the mixin, the resulting CSS is output where we called it; with the appropriate value for the parameter as specified.

## Cascading + Nesting

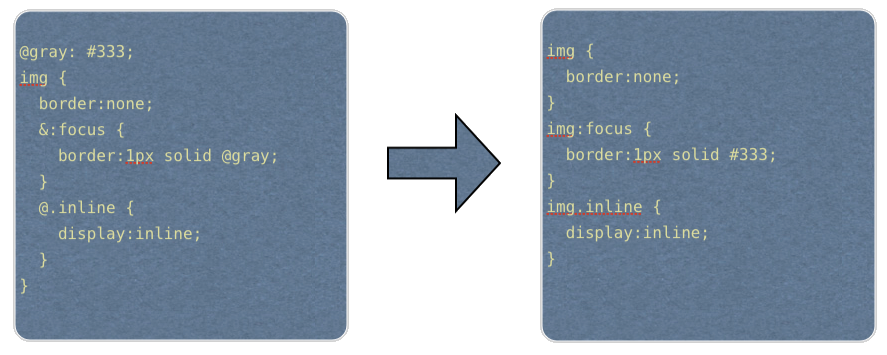
* Nest rulesets in place of cascading
* Can be used in combination with traditional cascading approach
* Mimics your DOM structure
* Outputs to cascading rulesets

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/nested-styles-in-less.png)

In this example, we are using the traditional cascading approach on the left, and the nested approach on the right.  As you can see, the nested approach reduces duplication of selectors and closely mimics the structure of the HTML or DOM.  It is also much cleaner and easier to read — making our lives easier as we have to maintain the code in the future.  We are also using the &combinator for the :focus pseudo selector, which we will discuss in more detail next.

## &combinator

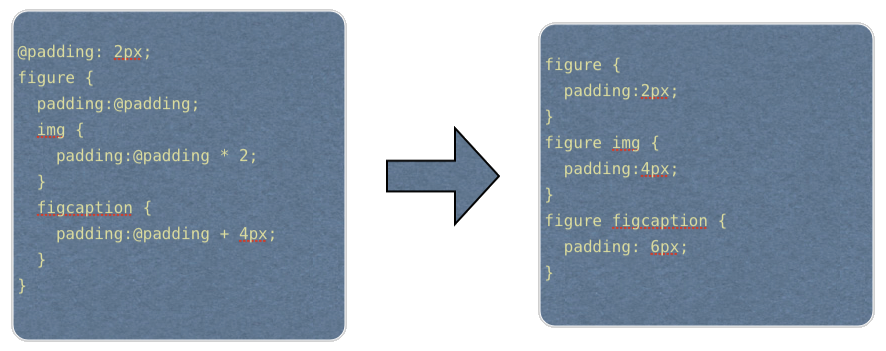
* Nested selector is concatenated with the parent selector when using the &combinator
* This is very useful for Pseudo classes, such as :focus and :hover

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/combinator-in-less.png)

In this example we are using the &:focus and &.inline combinators on the img selector.  You can see the resulting CSS that is output on the right.  The :focus pseudo selector is concatenated with the parent selector, resulting in img:focus.  The second combinator shows that we can use this with classes and other selectors besides pseudo classes.  Here, I am specifying images that have the class “inline” applied to them, and then setting the display property to inline.

## Operations

* Any size or color variable can be operated upon
* Simple Mathematical operators: +, -, \*, /
* Color functions
* Math functions

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/math-operators-in-less.png)

This example shows some simple math operations.  Here I am just multiplying and adding to an existing variable that is storing a length of 2 pixels.  You can do much more than just add and multiply simple pixels.  Here are some more examples:

* padding: @padding \* 10% // outputs “padding: 20px;”
* padding: ((@padding \* 10%) / 2) + 5px; //outputs “padding: 15px;”
* adding: @padding + (2 \* @padding); //outputs “padding:6px;”

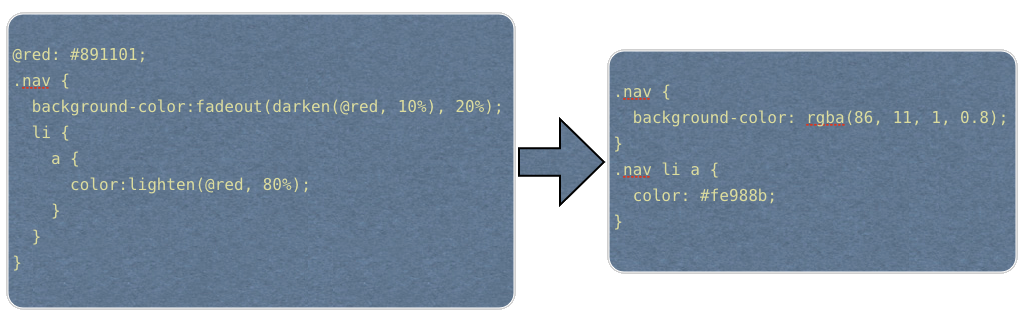
There are also several math functions that LESS provides:

* round()
* ceiling()
* floot()
* percentage()

## Color functions

enable you to apply various functions to create new colors or obtain information about existing colors.  These include:

* darken(@color, 20%); //returns color 20% darker
* lighten(@color, 20%); //returns color 20% lighter
* fadein(@color, 20%); //returns color 20% more transparent
* fadeout(@color, 20%); //returns color 20% less transparent
* fade(@color, 80%); //returns color with 80% transparency
* hue(@color); //hue channel of color
* saturation(@color); //saturation channel of color
* lightness(@color); //lightness channel of color
* alpha(@color); //alpha channel of color

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/color-functions-in-less.png)

In this example I have a variable named “red” that contains a specific shade of red for my web site.  I can then slightly alter this color using the darken() and lighten() functions.  In the first instance I am creating a new color that is 10% darker, and then use this as the color argument to the fadeout() function, so create an image that has 80% transparency.  In the second instance I am creating a new color that is 80% lighter than the original @red color.

## Comments

Comments are pretty straight-forward in LESS.  Multiline comments are preserved and included in the CSS output when you compile the LESS, and single line comments are not preserved.  Therefore, you should generally use single line comments, unless you really need the comment to be included in the generated CSS.

* /\* These comments are preserved into your compiled CSS \*/
* // These comments are silent

## @import

The use of @import() in your CSS is without-a-doubt a terrible idea.  This causes extra HTTP requests to be made from the client (browser) to your server, for each file you are imported.  If you have broken your CSS up into 5 different files, this is 5 requests, rather than a single request.  For site performance, you should have a single CSS file that is requested by the client.  Also, a general rule of thumb is to not use inline styles.  The CSS file that your client requests is cached by the browser, and therefore is not downloaded on each subsequent page visit.  The HTML that you return to your user is generally not cached, so all that inline CSS code is shipped to your user’s browser on each and every page load.

What solution does LESS provide?

* @import will compile and copy result into a single file
* All variables and mixins are available to main file or files imported after declarations
* Order matters
* Can include/ignore .less extension
* Can import “classic” css using .css extension

I recommend you use a single styles.less file that includes all of the necessary CSS files you are using.  Here is an example:

|  |  |
| --- | --- |
|  | // import normalize for CSS resets  @import "normalize";  // same as @import “normalize.less”;  // import mixins for all of my "global" variables and mixins  @import "mixins";  // base for mobile devices  @import "base";  //tablets  @media only screen and (min-width: 768px) {  @import "768";  }  //desktops  @media only screen and (min-width: 1030px) {  @import "1030";  } |

## String interpolation

* Use @{name} construct
* For embedding variable values within declarations

[](http://uploads.brianflove.com/wp-content/uploads/2012/10/string-interpolation-in-less.png)

The example shows that we might want to store our site’s base URL as a string variable, along with another variable to hold references to the folder where we store our images.  I can then use both of these variables to output the full URL to a background image.  The resulting CSS is displayed at the bottom.

## Escaping

* If you need to output CSS that is not valid CSS syntax
* Proprietary syntax not recognized by LESS
* If not used, LESS compiler will throw an error
* Simple prefix with ~ symbol and put in quotes (string)

For example, you might be using IE specific code that is not valid CSS to create a gradient:

|  |
| --- |
| filter: ~”progid:DXImageTransform.Microsoft.gradient(startColorstr='#dfdfdf', endColorstr='#f8f8f8')”; |

We put the invalid CSS in double-quotes, and then prefix it with the ~ symbol.  This tells the LESS compiler to ignore the code string, and to just output whatever is in there.

## Pre-compile

* Compile LESS to CSS3, so only the single CSS file is loaded by your web application
* This should be used for production
* Is less convenient during development
* You can set up the LESS compiler to “watch” a directory and to re-compile every time a file is saved in that directory — much easier than manually compiling to test your resulting CSS

On the mac, you can compile using the lessc command:

|  |  |
| --- | --- |
| 1 | $ lessc style.less > ../css/style.css |

## Post-compile

* You can use a less.js file, and include your less files in your HTML.
* This should NOT be used in production
* Easiest for development

|  |  |
| --- | --- |
|  | <!--LESS stylesheets first-->  <!---Post-compile LESS to CSS3--->  <script type="text/javascript" src="less.js"></script><script type="text/javascript">// <![CDATA[  less.watch();  // ]]></script> |

## LESS Elements

A LESS file that contains some useful mixins that are cross-browser, including:

* .gradient
* .rounded
* .opacity
* .box-shadow
* .inner-shadow

## Minification and compression

For production use, it is best to minimize and compress your CSS code.  This makes your code lightweight, and therefore, your web site will be faster.  In the end, your users will love you.  :-)  You can use the -x argument to the LESS compiler to minify your resulting CSS code, and you can use the –yui-compress flag to also have your resulting CSS code compressed using the YUI CSS Compressor.

|  |  |
| --- | --- |
| 1  2 | $ lessc -x styles.less > ../css/styles.css  $ lessc -x --yui-compress styles.less > ../css/styles.css |

Link :- <http://lesscss.org/>

http://www.hongkiat.com/blog/less-basic/

Video link :- <http://www.youtube.com/watch?v=vCdczWhhToo>

Css to less Convert tools online:- <http://css2less.cc/>

Compiler less css free download : <http://wearekiss.com/simpless>