@verekia's blog

HTML5, CSS3 and JavaScript tutorials

Follow @verekia

DON’T READ this Less CSS tutorial (highly addictive)

Posted on October 12, 2011

- Tweet 102

WARNING: If you like copy-pasting, find-replacing, scrolling in giant messy CSS files, not reusing your code, writing hundreds of times the same thing, you should definitely LEAVE THIS PAGE RIGHT NOW. I’m serious, if you start reading this tutorial, you might like it and could dangerously become way more productive. At the end of this read, you might even think crazy things like *“pure CSS sucks”* ! You’ve been warned…

If you’re a web developer or designer you probably faced this kind of situation before:
“I wonder if we should use a different theme on our website, can we try a blue one instead of the current green?”

2 possible answers:

- A pure CSS developer: “Damn… Can you come back in 15 minutes?”
- A Less CSS developer: “Sure! Wait just a second… Done, look!”

If you’re using pure CSS, even when respecting best practices, chances are that your answer will be the first one. Indeed, CSS is really far from being perfect. Hasn’t it come to your mind that some essential features are missing from a such widely used language? For instance, very basic things like *variables*.

Declaring a simple variable to store a color at the top of your stylesheet and being able to change the entire theme of your site by only changing this single variable would be pretty neat right? And what about all these repetitions each time you use CSS3 properties with vendor prefixes? Wouldn't it be useful to have to declare that just once?

Well guess what, we're gonna fix that! This article will demonstrate the power of Less, an enrichment of CSS using JavaScript. It adds tons of awesome features missing from CSS to make your web developer life so much easier. Trust me, once you've tried it, it changes the way you code your stylesheets forever and you'll never come back to pure CSS!

What is Less?

A "dynamic" CSS

According to the official website, Less is a "dynamic stylesheet language". I have a question for you now: How many stylesheet languages do you know?

Well except if you developed your own language and a dedicated browser to interpret it, chances are that you only know CSS. You might even have never thought about "What if there were another stylesheet language?". Well Less is *kind of* one. "Kind of" because you develop using the Less language, but in the end, it's compiled to pure CSS so that browsers can read it.

It's also called "dynamic" because it enriches CSS with many common features available in most modern dynamic programming languages (like variables for instance). During the development phase, you'll be able to use all those new cool features we'll see in this tutorial and it will help you code way more efficiently.

Less compiler, which is written in JavaScript, will convert all your Less code into CSS to be interpretable by the client. When does this conversion happen? Well you have 2 options: client-side or server-side.

Client-side?

If executed client-side, a simple JavaScript file will convert all your Less files to regular CSS before rendering the page, just like any other JavaScript library. It's very convenient and easy to set up, but you guys may think "What if my JavaScript is disabled or blocked by some strict policy?". Well indeed that's a problem, the page would be rendered without any style. Moreover, we may want to save the client's resource for more interesting tasks. This option is however the fastest way to get started with Less because you have nothing to configure.

Server-side?

You can also execute Less on the server. And since Less is written is JavaScript, it naturally fits with Node.js (there is a npm package to install it). However many developers don't use Node.js and are more familiar with PHP, Java, .Net or Ruby for instance.

Well, you'll be glad to learn that since Less (and its twin brother SASS) are becoming very popular, some implementations have been realized for other programming languages:

- SASS is the equivalent of Less for Ruby,
- LessPHP is the PHP implementation of Less,
- A Java implementation has been made by Asual,
- and Dotless is the .Net implementation.

SASS syntax differs a little bit from Less, so you should read a specific SASS tutorial instead of this one. Please also notice that I haven't tried those other implementations, because I have another very interesting alternative to suggest: using the CSS output for production.

A simple universal solution: Use the CSS output for production!

We can make Less work everywhere, even if JavaScript is deactivated, and without using any server nor client resources with very little extra work: just use the client version of Less for development, render the page in your own modern browser, then just copy the CSS output generated by Less and put it in a regular CSS file for production. That's all.

This simple task will allow you to use Less on any kind of server. The only drawback is that you can't use some specific advanced features like using JavaScript in your Less files for detecting the screen size on the fly for example. So if you don't have this kind of very specific need, you can use this technique.

Okay sounds cool, let's start!

In this entire scenario, my goal will be to draw CSS shapes with different colors and decorations (cause it's visual and I know you like it). Of course you can, and should, imagine how you can apply this to a real project. I highly

recommend that you implement this example for real, because it's the best way to learn. Honestly, you should **really** do it. I know you're lazy but trust me, it's worth it. So download less.js, create an empty HTML file and here we go!

Our HTML file

First, here is the HTML we'll be using:

```
<!doctype html>
<head>
    <link rel="stylesheet/less" type="text/css" href="style.less">
    <script src="less-1.1.3.min.js" type="text/javascript"></script>
</head>
<body>
    <div class="shape" id="shape1"></div>
    <div class="shape" id="shape2"></div>
    <div class="shape" id="shape3"></div>
</body>
</html>
```

As you can see, we declared a stylesheet with a `rel` attribute set to `stylesheet/less` and created a file called `style.less`. All our Less and CSS code will go to this file from now. We also declared the Less JavaScript file that will compile our Less code into CSS. This JavaScript file will perform some AJAX calls to read your Less files, so you need to use a web server. If you don't have one, just download WAMP (Windows), MAMP (Mac) or LAMP (Linux) depending on your operating system and you're ready to go.

Syntax highlighting

You can now open the `style.less` file in your favorite editor. I personally use Eclipse. You'll soon notice that since you're editing a *.less file, you don't get any syntax highlighting by default. To fix this, just find the place in your editor's preferences where you can choose which files should be associated with CSS highlighting. Less and CSS syntaxes are very close, and even if they differ a little bit, for most cases it will work fine. In Eclipse go to Window > Preferences > General > Content Types > Text > CSS, click Add, and enter `.less`.

A little bit of basic CSS styling

Okay, we can finally start! Let's add some CSS styles to our shapes in order to see them. We'll set the default shape to a simple light blue square:

```
.shape{
    display:inline-block;
    width:200px;
    height:200px;
    background:#ddf;
    margin:20px;
}
```

Nothing surprising here, just pure and simple CSS for the moment. But what if our example gets mucher bigger and we declared the `#ddf` light blue color in several different places in the file? Even with very good CSS best practices, we would still have to replace every single declaration of this color in order to try a new theme for example. That's not convenient at all. Let's use a Less variable to handle this.

Variables

Less variables can be declared and used with the `@` symbol. You give them a name and a value, and then you can refer to the name of the variable anywhere in your Less file:

```
@lightBlue:#ddf;

.shape{
    display:inline-block;
    width:200px;
    height:200px;
    background:@lightBlue;
    margin:20px;
}
```

Pretty cool right? But it won't be very helpful if we have to change the references to `@lightBlue` everywhere in the code. So we can for instance declare an other variable for our "default theme":

```
@lightRed:   #fdd;
@lightGreen: #dfd;
@lightBlue:  #ddf;

@defaultThemeColor:@lightGreen;

.shape{
    display:inline-block;
    width:200px;
    height:200px;
    background:@defaultThemeColor;
    margin:20px;
}
```

With this example you can see our code becoming more modular and reusable. We just have to change the `@defaultThemeColor` to `@lightGreen` to apply the new green theme:

You can see a Less best practice here as well: declaring all the dumb constants at the top of the file, and then adding some logic.Talking about constants, be aware that Less "variables" are actually just constants, and their value cannot be changed once they're set. So like Indiana Jones, "choose wisely"!

> **Personal tip**: I like to use the usual Hungarian notation (lowercase for the first word, then uppercase for the first letter of the following words) for variables. You could also use the CSS-style naming convention with dashes-between-lowercase-words, it's up to you. It personally helps me to differentiate Less from pure CSS.

Mixins

Do you know how to draw rounds in CSS? You just have to set a very high CSS3 border-radius value. Let's apply that to the first shape:

```
1 #shape1{
2     border-radius:9999px;
3 }
```

Since we used CSS3, if we want it to work for as many browsers as possible, we must provide the alternative `-webkit-` and `-moz-` vendor prefixes versions as well. So you would do something like this is regular CSS:

```
#shape1{
    -webkit-border-radius:9999px;
       -moz-border-radius:9999px;
            border-radius:9999px;
}
```

We see something interesting here: three properties with the same value. Pretty boring to write right? No problem, Less to the rescue!

With Less we can define "mixins", which are something comparable to functions in other programming languages. In Less they're used to group CSS instructions. We'll use a mixin to handle the boring repetition of border-radius declarations. Just like variables, you have to declare a mixin before calling it, with the `.` symbol this time:

```
.Round{
    -webkit-border-radius:9999px;
       -moz-border-radius:9999px;
            border-radius:9999px;
}

#shape1{
    .Round;
}
```

You can then apply this `.Round` mixin wherever you want to make something round without having to declare the `border-radius` instructions again. That's already cool, but it gets even cooler with parameters.

> **Personal best practice**: You probably noticed that I used a capitalized letter for the mixin name. Well since CSS class selectors also use the `.` symbol, there is no way to differentiate it from a Less mixin name. That's why I always use capitalized letters. That also mimics Object Oriented Programming, which uses capitalized letters for classes.

Parametric mixins

I now want to be able to create rounded squares, not just rounds. Let's modify our `.Round` mixin to something more generic: `.RoundedShape`. Our `RoundedShape` mixin should be able to handle rounds and rounded squares, depending on the parameters used to call it. To declare parameters, use parentheses after the mixin name:

```
@defaultRadius:30px;

.RoundedShape(@radius:@defaultRadius){
    -webkit-border-radius:@radius;
       -moz-border-radius:@radius;
            border-radius:@radius;
}
```

A parameter can be used similarly to a variable inside the mixin, and you can specify a default value for the parameter if none is given when the mixin is called. I also used a variable for this default value to keep the code reusable and generic.

So now that we have our cool RoundedShape mixin, let's create 2 other mixins which will call it to create rounds and rounded squares:

```
.Round{
    .RoundedShape(9999px);
}

.RoundedSquare(@radius:@defaultRadius){
    .RoundedShape(@radius);
}
```

And then, all we have to do to is applying those `Round` and `RoundedSquare` to our shapes:

```
#shape1{
    .Round;
}

#shape2{
    .RoundedSquare;
}
```

Now refresh your page and… BOOM!

Since we haven't passed any parameter to `RoundedSquare`, the 30px `@defaultRadius` variable is used, but we could also call it with parameters, on `#shape3` for instance:

```
#shape3{
    .RoundedSquare(60px)
}
```

With just 3 lines to call our mixins, we have a whole logic handling behind the scene that makes our code super reusable and easily maintainable. How cool is that? We can go way further with mixins. In my opinion, variables and mixins are so powerful that it's a sufficient reason to move to Less! But since Less offers tons of other amazing features, let's take a look at it.

Operations

One other powerful feature of Less is the ability to use mathematical operations in your stylesheets (I agree it sounds boring, but it's actually very cool). Let's declare a `@defaultShapesWidth` variable to set the default shapes widths to `200px` instead of hard-coding it like we used to:

```
1 @defaultShapesWidth:200px;
```

Now I want to add some borders to our shapes. Borders, whose size will always be 10% of the shape width. How would you do that in pure CSS? Yeah, exactly, you can't at all! So thanks to Less, we can now do this kind of operation:

```
1 @borderSize:@defaultShapesWidth * 0.1;
```

You can add, subtract, multiply and divide values. So you might think *"Yeah, but it only works for pixels right?"*. Nope sir, it also works for colors!

```
1 @defaultThemeColor:@lightBlue;
2 @borderColor:@defaultThemeColor - #222;
```

We darkened the light blue color by subtracting 2 hexadecimal units to each RGB value. It can be confusing the first time. Just keep in mind that adding or subtracting `#000` has no effect and `#fff` has a maximal effect.

And this is what we get when applying these new variables to a border rule:

```
1 border:@borderSize solid @borderColor;
```

The result is exactly as expected. That's awesome. But the color operation we used was pretty simple. Adding a little bit of dark is easy, performing saturation operations is way more complicated and unintuitive. Fortunately, we can use some handy color functions to do that!

> **Side note**: Remember that all this funky Less code will be converted to CSS in the end, so this will be static once generated. If you programmatically resize the width of a shape, the border won't scale with it on the fly.

Color functions

Less provides the following color functions:

- `darken()` and `lighten()`, which add some black or white,
- `saturate()` and `desaturate()`, to make a color more "colorful" or more "grayscale",
- `fadein()` and `fadeout()`, to increase or reduce transparency,
- and `spin()`, which modifies the hue of the color.

If, for instance, we want to make the border completely desaturated to get it's equivalent on the grayscale, we would use the desaturate function like this:

```
1 @borderColor:desaturate(@defaultThemeColor, 100%);
```

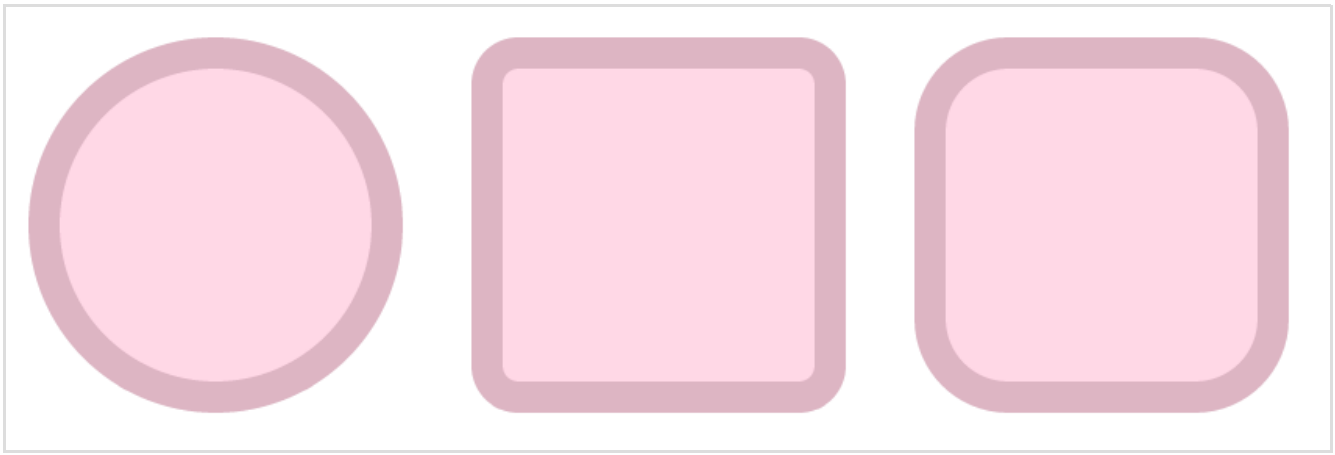
And we can even use the output of a color function as the input of another:

```
1 @borderColor:darken(desaturate(@defaultThemeColor, 100%), 20%);
```

Which has the following result:

All color operations take a color and a percentage as parameters, except spin, which uses integers between 0 and 255 instead of percentages to modify the hue:

```
1 @defaultThemeColor:spin(@lightBlue, 100);
```



Nested rules

When designing a complex page using CSS, you often have to chain selectors to aim a particular element in the DOM, like this:

```
1 #header h1{
2 }
3
4 #main h1{
5 }
```

Depending on the container of the `h1` elements, their style will be different if they're in the `#header` or the `#main` section. This syntax is fine when you have a very simple DOM, but if you have to chain 4 or 5 selectors it gets messy and visually hard to represent the hierarchy of your styles. With Less you can nest rules inside parent rules to mimic the DOM structure:

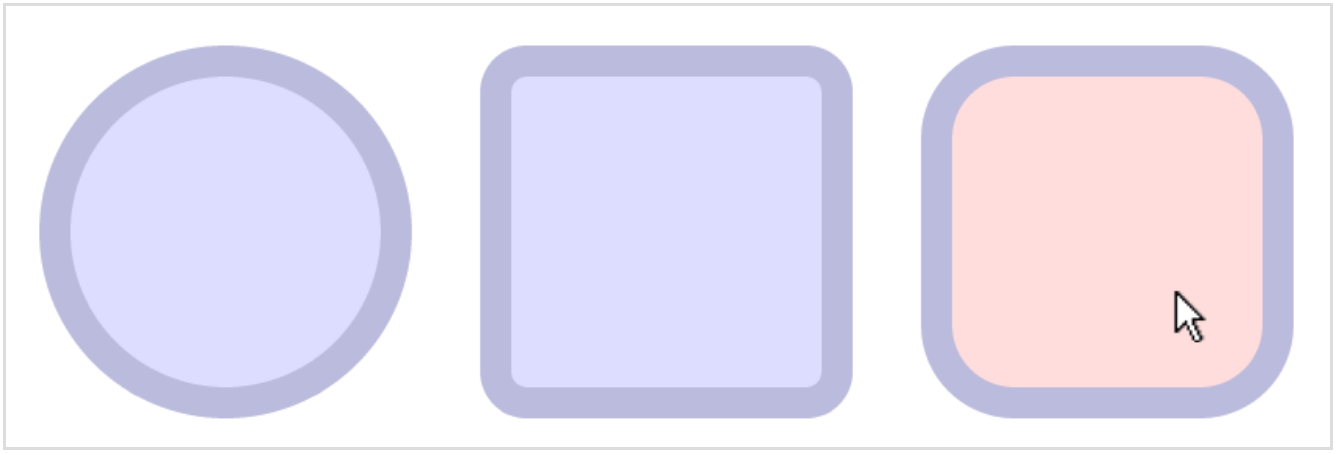
```
1 #header{
2     /* #header styles */
3     h1{
4         /* #header h1 styles */
5     }
6 }
```

This can be a little bit confusing at the beginning but when you get used to it, this syntax helps a lot to visualise where is located the element you're working on.

There is also a super handy way to use pseudo-classes (you know, those special selectors like `:hover`). Let's say we want to change our shapes' color when hovering them with the mouse, we can use nested rules combined with the & selector:

```
.shape{
    &:hover{
        background:@lightRed;
    }
}
```

This & symbol represents the current selected elements. It's the equivalent of "`this`" in most programming languages. Very convenient! And here is the result:



Nested rules are pretty awesome, but there is a little drawback: since we use CSS syntax highlighting for our `*.less` files, many editors (including Eclipse) will go crazy and do some very bad syntax coloration with nested rules. If you can deal with broken syntax highlighting it will be fine, but for most of us, it's really awful. The only thing we can do is wait for a proper plugin to bring a real Less support to our editors.

Personally, I only use nested rules for very simple one level nesting with few properties or for the `&:hover` thing. But it really depends on you.

Importing

Finally, the last thing I wanted to show you is how to import Less files. It's a very good practice to separate your rules into different files instead of having a 1000 lines giant file.

Importing a file into another in Less is as simple as that:

```
@import "colors.less";
```

You can even omit the .less extension and just declare:

```
@import "colors";
```

So depending on your project, you can split your files in a relevant way. Personally, I like having separate files for:

- CSS Reset or normalization,
- Colors,
- Fonts and typography,
- UI Elements,
- The main styles of the page,
- …and any other relevant group of rules depending on your project.

You can take a look at the very good Twitter Bootstrap Github repository to see how those guys splitted up their files for a professional usage.

You may think that imports result in many unnecessary HTTP requests that will slow down the loading time of your page, and that's true! But remember we'll use the Less output as a new CSS. If we do so, we really don't care about using multiple files, since at the end, everything will be concatenated into just one! So don't be afraid of imports. They will really help you to structure your code.

Final Less code

Here is what the final code looks like (no import for such a little example):

```
/**********************
       CONSTANTS
**********************/

@lightRed:   #fdd;
@lightGreen: #dfd;
@lightBlue:  #ddf;
@defaultShapesWidth:200px;
@defaultRadius:30px;

/*********************************
   OPERATIONS & COLOR FUNCTIONS
*********************************/

@darkBlue: @lightBlue - #555;

@defaultThemeColor:@lightBlue;
//@defaultThemeColor:spin(@lightBlue, 100);

@borderSize:@defaultShapesWidth * 0.1;
@borderColor:@defaultThemeColor - #222;
//@borderColor:darken(desaturate(@defaultThemeColor, 100%), 20%);

/**********************
        MIXINS
**********************/

.RoundedShape(@radius:@defaultRadius){
    -webkit-border-radius:@radius;
       -moz-border-radius:@radius;
            border-radius:@radius;
}

.Round{
    .RoundedShape(9999px);
}

.RoundedSquare(@radius:@defaultRadius){
```

```
    .RoundedShape(@radius);
}

/**********************
        STYLES
**********************/

.shape{
    display:inline-block;
    width:@defaultShapesWidth;
    height:200px;
    background:@defaultThemeColor;
    margin:20px;
    border:@borderSize solid @borderColor;
}

.shape{
    &:hover{ background:@lightRed   }
}

#shape1{ .Round }
#shape2{ .RoundedSquare }
#shape3{ .RoundedSquare(60px) }
```

You will also notice that you can use single line comments, which is a very simple but clever addition to CSS!

And here is the demonstration of this Less CSS example.

Time for production: get the CSS output

Okay, enough coding, we've worked hard on our little shapes example, and it's time to deploy it on the production server! Less was super convenient for the development phase, but we now want to grab its precious pure CSS output. To do that, just open a browser (I'll use Chrome), open the developer tool, and check the bottom of the `head` section:

```
<!DOCTYPE html>
▼<html>
  ▼<head>
    <link rel="stylesheet/less" type="text/css" href="style.less">
    <script src="less-1.1.3.min.js" type="text/javascript"></script>
   ▼<style type="text/css" media="screen" id="less:lesstuto-style">
      .shape {
        display: inline-block;
        width: 200px;
        height: 200px;
        background: #ddddff;
        margin: 20px;
        border: 20px solid #bbbbdd;
      }
      .shape:hover {
        background: #ffdddd;
      }
      .Round {
        -webkit-border-radius: 9999px;
        -moz-border-radius: 9999px;
        border-radius: 9999px;
```

```
}
#shape1 {
  -webkit-border-radius: 9999px;
  -moz-border-radius: 9999px;
  border-radius: 9999px;
}
```

All your final CSS is here. So just click on it and Copy / Paste it to a new CSS file. You should also optimize and minify it using your favorite CSS compression tools for better performance.

> **Note**: Some very specific advanced features of Less allow you to use JavaScript in your Less files, to get the screen size for instance. Remember that if you use this "CSS output" technique, the final code will obviously become completely static, so any code depending on the context of JavaScript execution won't be functional. The rest is fine!

Conclusion

We've seen a lot of amazing features in this tutorial, and I hope it made you want to start using Less. I personally think Less is a real game changer for web development, and I cannot even consider going back to pure CSS. Less is like CSS5 in 2011!

There are still very interesting things I haven't covered in this tutorial so if you're interested in learning more about Less, you should definitely check the official documentation. Oh and by the way, since Less is also open source and hosted on Github you can easily get involved, report issues, or fork it. This great comment also gives some very interesting tips you should be aware of.

That's all folks! Thanks for reading and feel free to share your thoughts and previous experience about Less in the comments section. I'd love to know if you enjoy it as much as I do!

This entry was posted in **Less CSS** and tagged **-moz-**, **-webkit-**, **.Net**, **Addictive**, **Best practices**, **Color functions**, **CSS**, **CSS Output**, **CSS3**, **Dangerous**, **darken**, **demo**, **desaturate**, **Dotless**, **Dynamic stylesheet language**, **example**, **fadein**, **fadeout**, **Import**, **Importing**, **JavaScript**, **LessPHP**, **lighten**, **Mixins**, **Nested rules**, **Operations**, **Parametric Mixins**, **PHP**, **Round**, **Ruby**, **SASS**, **saturate**, **Shapes**, **spin**, **Square**, **Syntax highlighting**, **this**, **tutorial**, **Variables** by **Jonathan**. Bookmark the **permalink [http://verekia.com/less-css/dont-read-less-css-tutorial-highly-addictive]** .

12 THOUGHTS ON "DON'T READ THIS LESS CSS TUTORIAL (HIGHLY ADDICTIVE)"

Ryan Ransford

on **October 13, 2011 at 7:08 AM** said:

It should also be noted that Less comes with a command-line compiler, lessc, that can be used to compile .less files into .css files.

Jonathan
on **October 14, 2011 at 5:25 PM** said:

You're right! And here is the binary file. Thanks Ryan.

Elliott
on **October 13, 2011 at 10:41 PM** said:

That is very cool, how would you compare this to SASS and SCSS? It's difficult to know what's best to use for future proofing and general standards

Jonathan
on **October 14, 2011 at 5:23 PM** said:

Hi Elliott,
Both embrace the same philosophy, and since Less and SASS syntax are really close, you can easily move from one to another if you have (or want) to. In my opinion it really just depends on your development environment. If I had to use Ruby I would prefer SASS.

Chris Jacob
on **November 13, 2011 at 9:58 PM** said:

Excellent post. P.S. You might also like Stylus as a CSS pre-processor:

Docs: http://learnboost.github.com/stylus/
Demo: http://learnboost.github.com/stylus/try.html

Anas
on **December 4, 2011 at 7:13 AM** said:

WAW !! I always dreamt of something like that !! Thanks a lot !!

Anderson Juhasc
on **January 4, 2012 at 6:34 AM** said:

Nice article! I've been using the "lesscss" since 2010 else or I am mistaken …
The solution to compile and the following:

Less.js I install on my server and I Node.js compile the file when I saved.
Use the editor came up with this code:

```
autocmd FileWritePost,BufWritePost *.less :call
LessCSSCompress()
function! LessCSSCompress()
let cwd = expand(':p:h')
let name = expand(':t:r')
if (executable('lessc'))
cal system('lessc '.cwd.'/'.name.'.less >
'.cwd.'/'.name.'.css &')
endif
endfunction
```

Anderson Juhasc
on **January 4, 2012 at 6:40 AM** said:

I also set up the file .less for use with the Zen Coding, the editors got it using notepad++ and VIM.

Edwin
on **January 7, 2012 at 2:37 PM** said:

You were right !

I'll never get back to pure css again !

This is pure pleasure :)

Thanx !

Thorarinn Stefansson
on **January 15, 2012 at 10:17 AM** said:

Two more hints / tricks:
For mixins that you define to reference later (like `.Round` in your example) you can add empty parentheses when you define them: `.Round(){definition}`
- This way the initial definition isn't included in the rendered CSS, only the code where the mixin is being used, like `#shape1` in your example. This both shortens your output code and allows you to keep a library of definitions that don't add to the weight of the output until they are actually used.

It also helps with naming; `.someName` is a CSS class I use as such, `.someName()` is a mixin.

Also tools like less.app (OS X) and SimpLESS (OSX, Win, Linux) can watch the .less files (on your development computer) and compile them to .css on your every save. Eliminating the need to copy from the browser or install Node.js.

Jonathan
on **January 16, 2012 at 9:01 AM** said:

Thank you very much Thorarinn, awesome tips! :)

Lana
on **January 18, 2012 at 3:34 PM** said:

Thanks for the tutorial! Just what I’ve been looking for!