What is Less?

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themable and extendable.

Less runs inside Node, in the browser and inside Rhino.

There are also many 3rd party tools that allow you to compile your files and watch for changes

A "dynamic" CSS

According to the official website, Less is a "dynamic stylesheet language"

It's also called "dynamic" because it enriches CSS with many common features available in most modern dynamic programming languages (like variables for instance).

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 3 options:

- On the fly in the browser with `less.js`,
- By using the lessc compiler via the command-line,
- With some fancy app like CodeKit or LiveReload.

Compiling in the browser with less.js

`Less.js` will perform AJAX requests to grab your Less files, will then process those files to convert them into CSS, and finally inject the resulting CSS in the head of your document.

It is extremely inefficient in terms of performance and you should never use it in production.

Plus, if the user's JavaScript is deactivated, well… it will just not work at all.

But if you just want to do a quick Less prototype or if you don't have access to your usual development environment, `less.js` is perfectly fine!

Lessc and the command-line

You can also execute Less in your terminal.

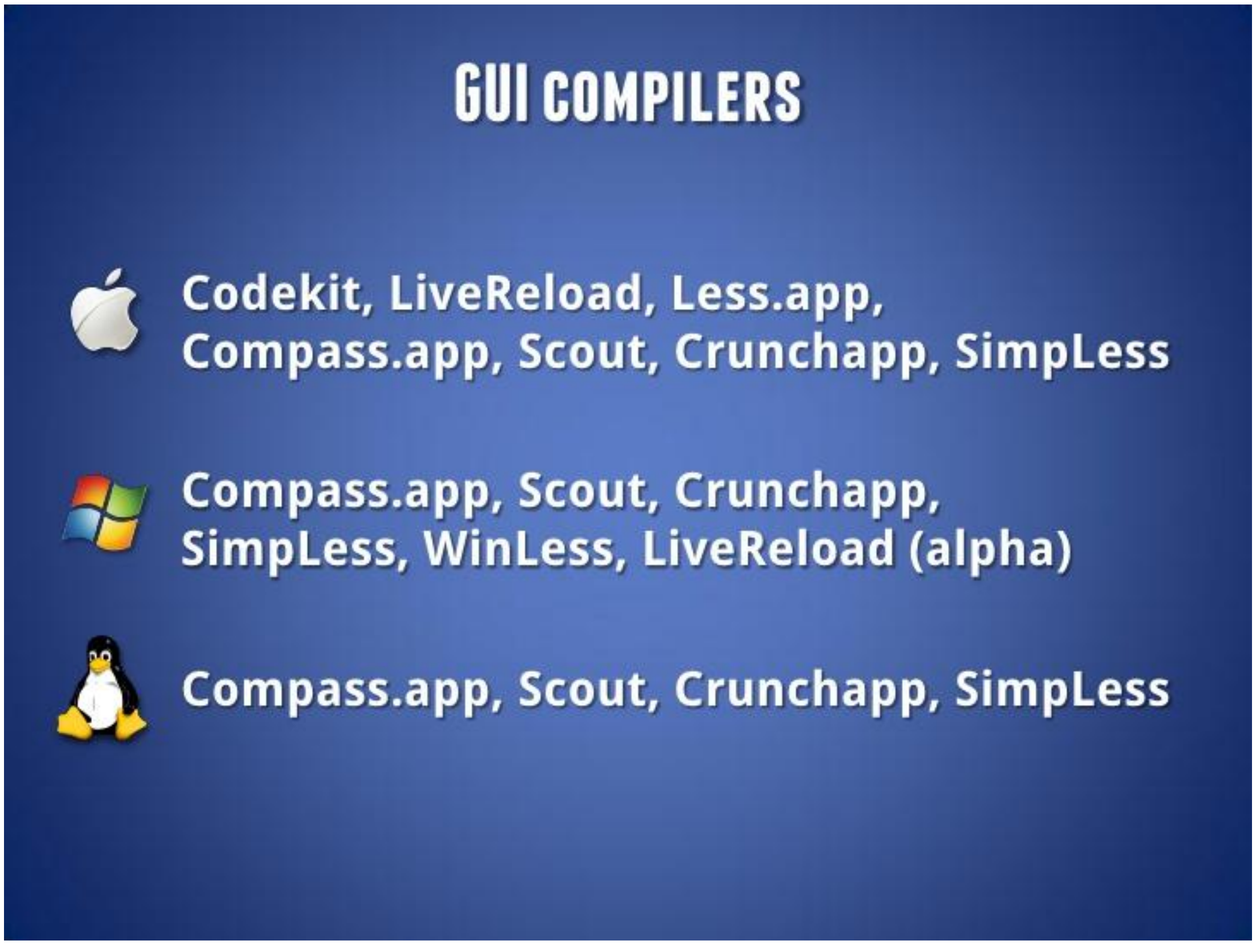
In order to do this, you will need to install Node.js on your development machine, and then install the Less module by running:

```
npm install less -g
```

You can then simply run the following command to compile a `style.less` file to `style.css`:

```
lessc style.less > style.css
```

Using an application to compile for you



GUI COMPILERS

Codekit, LiveReload, Less.app, Compass.app, Scout, Crunchapp, SimpLess

Compass.app, Scout, Crunchapp, SimpLess, WinLess, LiveReload (alpha)

Compass.app, Scout, Crunchapp, SimpLess

```
<!doctype html>
<head>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="shape" id="shape1"></div>
    <div class="shape" id="shape2"></div>
    <div class="shape" id="shape3"></div>
</body>
</html>
```

A little bit of basic CSS styling

First, create and open a `style.less` file.

Then we 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;
}
```

Run `lessc style.less > style.css` to compile it and this is what your browser should display:

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;
}
```

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

```
@defaultThemeColor:@lightGreen;

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

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:

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

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;
}
```

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;
}
```

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;
}




.Round{
    .RoundedShape(9999px);
}

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




#shape1{
    .Round;
}

#shape2{
    .RoundedSquare;
}
```

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:

```
@defaultShapesWidth:200px;
@borderSize:@defaultShapesWidth * 0.1;
@defaultThemeColor:@lightBlue;
@borderColor:@defaultThemeColor - #222;
border:@borderSize solid @borderColor;
```

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.

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

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

```
@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:

```
#header h1{
}

#main h1{
}
```

LESS:

```
#header{
    /* #header styles */
    h1{
        /* #header h1 styles */
    }
}
```

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

This `&` symbol represents the current selected elements.

It’s the equivalent of “`this`” in most programming languages

Importing

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”;
```

```
/**********************
       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) }
```