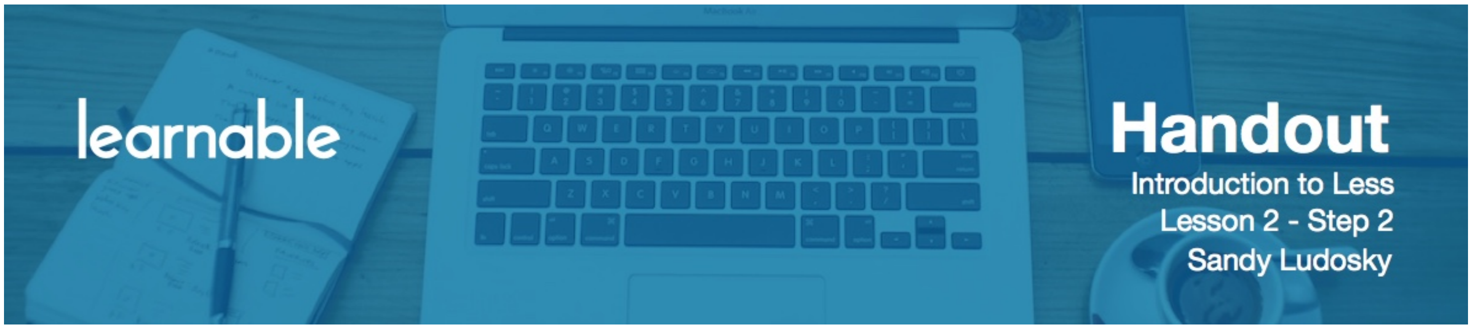

Introduction

So we're going to make our first demonstration of the Less syntax, and for that we'll use the online Less compiler **WinLess**. Navigate to winless.org and click on the second tab on the right, which is Online Less Compiler.

Advantage of Less over CSS

CSS is a great language, although it has some limitations. For example, you cannot define variables or do math calculation. That means that you can end up repeating yourself in different places of your CSS files with the same pieces of styling. That's not very good for best practices.

Also, it doesn't allow you to keep your code **DRY (don't repeat yourself)**. It's very typical for stylesheets to include hundreds of lines of code with a large number of indefinite rules. The longer and more complex a project becomes, the harder it becomes to manage. Lucky for web developers, Less helps you resolve this inconvenience by allowing you to write less code, which is object oriented and more maintainable.

Less syntax demo

This is how you declare Less variables

```
@border-width: 1px;
@red: #842210;
```

and use them

```
div#header {
    border: @border-width solid @red;
}
```

The same for the child elements:

```
div#header h1 {
    color: @red;
}
```

Now suppose we have a footer:

```
div#footer {
    border: @border-width solid @red;
}

div#footer p {
    font-size: 10px;
}
```

Now let me show you nesting to avoid code duplication. You can notice that for the header and footer and the paragraph inside it we're using twice the same selector. To keep the style sheet clean and well structured, we're going to use nesting. That allows you to follow the same pattern of the DOM structure that we find inside the HTML document:

```
div#header {
    border: @border-width solid @red;
    h1 {
        color: @red;
    }
}
```

With nesting we are be able to nest selectors within each other that are part of the same group. And you can see that this is more clean and better structured than before.

Footer's styles can be re-written as well:

```
div#footer {
    border: @border-width solid @red;
    p {
        font-size: 10px;
    }
}
```

Now notice that we're using twice the same CSS property: first with the header and then with the footer. So, I'm going to define a mixin that I'm going to name `commonRules`:

```
.commonRules {
    border: @border-width solid @red;
}
```

Now we can use this mixin as an object. You are defining the rule set that you can reuse multiple times in your stylesheets. Next I'll use this mixin:

```
div#header {
    .commonRules;
    h1 {
        color: @red;
    }
}

div#footer {
    .commonRules;
    p {
        font-size: 10px;
    }
}
```

The purpose of the mixin is to help you cut down on code so you can define one time an object. So, that's a better way to maintain your CSS.

Next we're going to talk about the operators that allow you to do math calculation. I'll define a new variable:

```
@padding: 10px;
```

And now update the mixin:

```
.commonRules {
    border: @border-width solid @red;
    padding: @padding + 10;
}
```

I am using the `+` operator to increase the original padding's value by 10.