Less is a CSS pre-processor, that adds features such as variables, nesting, mixins and more. Less (and other preprocessors, such as Sass help developers to write maintainable and DRY (Don't Repeat Yourself) code.

```
//Single line comments are removed when Less is compiled to CSS.

/*Multi line comments are preserved. */


/*Variables
==============================*/



/* You can store a CSS value (such as a color) in a variable.
Use the '@' symbol to create a variable. */

@primary-color: #A3A4FF;
@secondary-color: #51527F;
@body-font: 'Roboto', sans-serif;

/* You can use the variables throughout your stylesheet.
Now if you want to change a color, you only have to make the change once.*/

body {
    background-color: @primary-color;
    color: @secondary-color;
    font-family: @body-font;
}

/* This would compile to: */
body {
    background-color: #A3A4FF;
    color: #51527F;
    font-family: 'Roboto', sans-serif;
}


/* This is much more maintainable than having to change the color
each time it appears throughout your stylesheet. */


/*Mixins
==============================*/



/* If you find you are writing the same code for more than one
element, you might want to reuse that easily.*/

.center {
    display: block;
    margin-left: auto;
    margin-right: auto;
    left: 0;
    right: 0;
}

/* You can use the mixin by simply adding the selector as a style */

div {
    .center;
    background-color: @primary-color;
}

/*Which would compile to: */
.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
  left: 0;
  right: 0;
}
div {
    display: block;
    margin-left: auto;
    margin-right: auto;
    left: 0;
    right: 0;
    background-color: #A3A4FF;
}

/* You can omit the mixin code from being compiled by adding paranthesis
   after the selector */

.center() {
  display: block;
  margin-left: auto;
  margin-right: auto;
  left: 0;
  right: 0;
}

div {
  .center;
  background-color: @primary-color;
}

/*Which would compile to: */
div {
  display: block;
  margin-left: auto;
  margin-right: auto;
  left: 0;
  right: 0;
  background-color: #A3A4FF;
}
```

```
/*Functions
==============================*/



/* Less provides functions that can be used to accomplish a variety of
   tasks. Consider the following */

/* Functions can be invoked by using their name and passing in the
   required arguments */
body {
  width: round(10.25px);
}

.footer {
  background-color: fadeout(#000000, 0.25)
}

/* Compiles to: */

body {
  width: 10px;
}

.footer {
  background-color: rgba(0, 0, 0, 0.75);
}

/* You may also define your own functions. Functions are very similar to
   mixins. When trying to choose between a function or a mixin, remember
   that mixins are best for generating CSS while functions are better for
   logic that might be used throughout your Less code. The examples in
   the Math Operators' section are ideal candidates for becoming a reusable
   function. */

/* This function will take a target size and the parent size and calculate
   and return the percentage */

.average(@x, @y) {
  @average_result: ((@x + @y) / 2);
}

div {
  .average(16px, 50px); // "call" the mixin
  padding: @average_result;    // use its "return" value
}

/* Compiles to: */

div {
  padding: 33px;
}
```

```
/*Extend (Inheritance)
==============================*/



/*Extend is a way to share the properties of one selector with another. */

.display {
  height: 50px;
}

.display-success {
  &:extend(.display);
    border-color: #22df56;
}

/* Compiles to: */
.display,
.display-success {
  height: 50px;
}
.display-success {
  border-color: #22df56;
}

/* Extending a CSS statement is preferable to creating a mixin
   because of the way it groups together the classes that all share
   the same base styling. If this was done with a mixin, the properties
   would be duplicated for each statement that
   called the mixin. While it won't affect your workflow, it will
   add unnecessary bloat to the files created by the Less compiler. */



/*Nesting
==============================*/



/*Less allows you to nest selectors within selectors */

ul {
    list-style-type: none;
    margin-top: 2em;

    li {
        background-color: #FF0000;
    }
}

/* '&' will be replaced by the parent selector. */
/* You can also nest pseudo-classes. */
/* Keep in mind that over-nesting will make your code less maintainable.
```

```
Best practices recommend going no more than 3 levels deep when nesting.
For example: */

ul {
    list-style-type: none;
    margin-top: 2em;

    li {
        background-color: red;

        &:hover {
          background-color: blue;
        }

        a {
          color: white;
        }
    }
}

/* Compiles to: */

ul {
  list-style-type: none;
  margin-top: 2em;
}

ul li {
  background-color: red;
}

ul li:hover {
  background-color: blue;
}

ul li a {
  color: white;
}



/*Partials and Imports
==============================*/



/* Less allows you to create partial files. This can help keep your Less
   code modularized. Partial files conventionally begin with an '_',
   e.g. _reset.less. and are imported into a main less file that gets
   compiled into CSS */

/* Consider the following CSS which we'll put in a file called _reset.less */

html,
body,
ul,
ol {
  margin: 0;
  padding: 0;
}

/* Less offers @import which can be used to import partials into a file.
   This differs from the traditional CSS @import statement which makes
   another HTTP request to fetch the imported file. Less takes the
   imported file and combines it with the compiled code. */

@import 'reset';

body {
  font-size: 16px;
  font-family: Helvetica, Arial, Sans-serif;
}

/* Compiles to: */

html, body, ul, ol {
  margin: 0;
  padding: 0;
}

body {
  font-size: 16px;
  font-family: Helvetica, Arial, Sans-serif;
}


/*Math Operations
==============================*/



/* Less provides the following operators: +, -, *, /, and %. These can
   be useful for calculating values directly in your Less files instead
   of using values that you've already calculated by hand. Below is an example
   of a setting up a simple two column design. */

@content-area: 960px;
@main-content: 600px;
@sidebar-content: 300px;

@main-size: @main-content / @content-area * 100%;
@sidebar-size: @sidebar-content / @content-area * 100%;
@gutter: 100% - (@main-size + @sidebar-size);

body {
  width: 100%;
}
```

```
.main-content {
  width: @main-size;
}

.sidebar {
  width: @sidebar-size;
}

.gutter {
  width: @gutter;
}

/* Compiles to: */

body {
  width: 100%;
}

.main-content {
  width: 62.5%;
}

.sidebar {
  width: 31.25%;
}

.gutter {
  width: 6.25%;
}
```

Practice Less

If you want to play with Less in your browser, check out LESS2CSS.

Compatibility

Less can be used in any project as long as you have a program to compile it into CSS. You'll want to verify that the CSS you're using is compatible with your target browsers.

QuirksMode CSS and CanIUse are great resources for checking compatibility.

Further reading

- Official Documentation