SASS (slaying asinine styling standards. jk)

SASS is an extension of CSS3 that adds nesting, variables, functions, basic programming logic, and selector inheritance, then compiles to well-formatted CSS. It helps you keep your styles organized and modular. Instead of copying and pasting colors and dimensions everywhere, you can change macros and variables in one place and have them change across the app.

SASS is especially helpful for cross browser development, as even simple CSS properties like border-radius are often implemented differently in each browser, forcing you to duplicate parameters for border-radius, -moz-border-radius, -webkit-border-radius, and so on. We'll cover this in an example later.

Syntaxes

SASS eliminates curly braces and semicolons, determining nesting by tabs. We'll be using SASS syntax for examples.
SCSS has all the functionality of SASS, but is more verbose. As with CSS, curly braces and semicolons are mandatory. There are a few other syntactic differences for SASS keywords.
Official site, full documentation
(note that the official documentation uses SCSS instead of SASS syntax)

Getting started

Sass is built on Ruby, so you must have that installed to use it. (If you don't already have Ruby set up for Rails or whatnot, get the **rvm** package to manage Ruby installations; you'll thank yourself later.)

The *sass* Ruby gem comes with this compilation command:

```
sass –watch style.sass:style.css
```

If you are as disgustingly lazy as I am, use my bash macro.

```
function sasswatch() {
 if [ "$2" ]; then
   sass --watch $1:$2;
 else
   ARG2=`echo $1 | sed -e 's/\.sass$/\.css/g;s/\.scss$/\.css/g'`;
   sass --watch $1:$ARG2;
 fi;
}
```

You can also use the *compass* gem to watch a directory and autocompile its sass files. Documentation: https://github.com/Compass/compass-rails

Examples: here are a few that demonstrate the power and robustness of SASS.

Nesting – obvious benefits

```
//CSS
table tr.foo {
  padding: 0;
}
table .h1 {
  margin: 2em 0;
}

li {
  font-family: serif;
  font-weight: bold;
  font-size: 15px;
}
```

```
//SASS
table
  tr.foo
    padding: 0
  h1
    margin: 2em 0

li
  font
    family: serif
    weight: bold
    size: 15px
```

Variables – coherence

```
//CSS
.wrapper {
  border-color: #3bbfce;
  padding: 8px
}

.nav {
  color: #3bbfce;
  margin-bottom: 16px;
}
```

```
//SASS
$accent: #3bbfce
$spacing: 16px

.wrapper
  border-color: $accent
  padding: $spacing/2

.nav
  color: $accent
  margin-bottom: $spacing
```

Mixins (reusable styling) – eliminate repetition and cross browser frustration!

```
//CSS
.nav
  box-shadow: 1px 1px 4px 2px #ccc
  -webkit-box-shadow: 1px 1px 4px 2px #ccc
  -moz-shadow: 1px 1px 4px 2px #ccc
  -o-shadow: 1px 1px 4px 2px #ccc

.wrapper
  box-shadow: -1px -1px 2px 0 #ccc
  -webkit-box-shadow: -1px -1px 2px 0 #ccc
  -moz-box-shadow: -1px -1px 2px 0 #ccc
  -o-box-shadow: -1px -1px 2px 0 #ccc
```

```
//SASS
=box-shadow($color, $blur:0, $spread:0, $h:0, $v:0)
  -moz-box-shadow: $h $v $blur $spread $color
  -webkit-box-shadow: $h $v $blur $spread $color
  -o-box-shadow: $h $v $blur $spread $color
  box-shadow: $h $v $blur $spread $color

.nav
  +box-shadow(1px, 1px, 4px, 2px, #ccc)

.wrapper
  +box-shadow(-1px, -1px, 2px, 0, #ccc)
```

Standard programming stuff

Mixins can take variable numbers of arguments, have default argument values, and pass blocks of styles as arguments. If you do the latter, read up on SASS's variable scoping first.

Mixins aren't functions, though – just reusable, variable styles. You can also write real functions in SASS if you need to do calculations. A writeup on mixins vs. functions.

Control directives – although you shouldn't need to use them often, SASS implements *if, for, each,* and *while*.

Another great feature is the ability to import stylesheets (*@import*). You can separate out styles intended for different widgets or pages into separate files, or perhaps keep all of your variables, mixins, and global app styling in separate files.

Interpolation

You can interpolate SASS variables in selectors and property names using #{$var}. So for example, instead of writing separate browser-agnostic mixins for each of border-top-left-radius, border-top-right-radius, border-bottom-right-radius, and border-bottom-left radius, you can cover them all with one mixin:

```
=border-rad($dir, $r)
  -moz-border-#{$dir}-radius: $r
  -webkit-border-#{$dir}-radius: $r
  -khtml-border-#{$dir}-radius: $r
  border-#{$dir}-radius: $r
```

and then call `+border-rad('top-left', 5px)`, WLOG. (You could also be slightly more clever and only conditionally add the dash after $dir, in case you want this mixin to handle plain border-radius.)

stdlib

Of course, many common (cross-browser) mixins have already been implemented, and you shouldn't have to redo the work. The compass and bourbon gems give you a large repertoire of useful mixins.