Sass is a preprocessor scripting language that is interpreted or compiled into CSS.

1 CSS COMPATIBLE

Sass is completely compatible with all versions of CSS.

Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of the Compass style library.

Syntax

There are two syntaxes available for Sass. The first, known as SCSS (Sassy CSS) and used throughout this reference, is an extension of the syntax of CSS.

The second and older syntax, known as the indented syntax (or sometimes just "Sass"), provides a more concise way of writing CSS. It uses indentation rather than brackets to indicate nesting of selectors, and newlines rather than semicolons to separate properties.

Variables

Think of variables as a way to store information that you want to reuse throughout your stylesheet. You can store things like colors, font stacks, or any CSS value you think you'll want to reuse. Sass uses the $ symbol to make something a variable. Here's an example:

$body-bg: #cc0000;

$body-color: #ffffff;

$link-color: darkblue;

body {

color: $body-color;

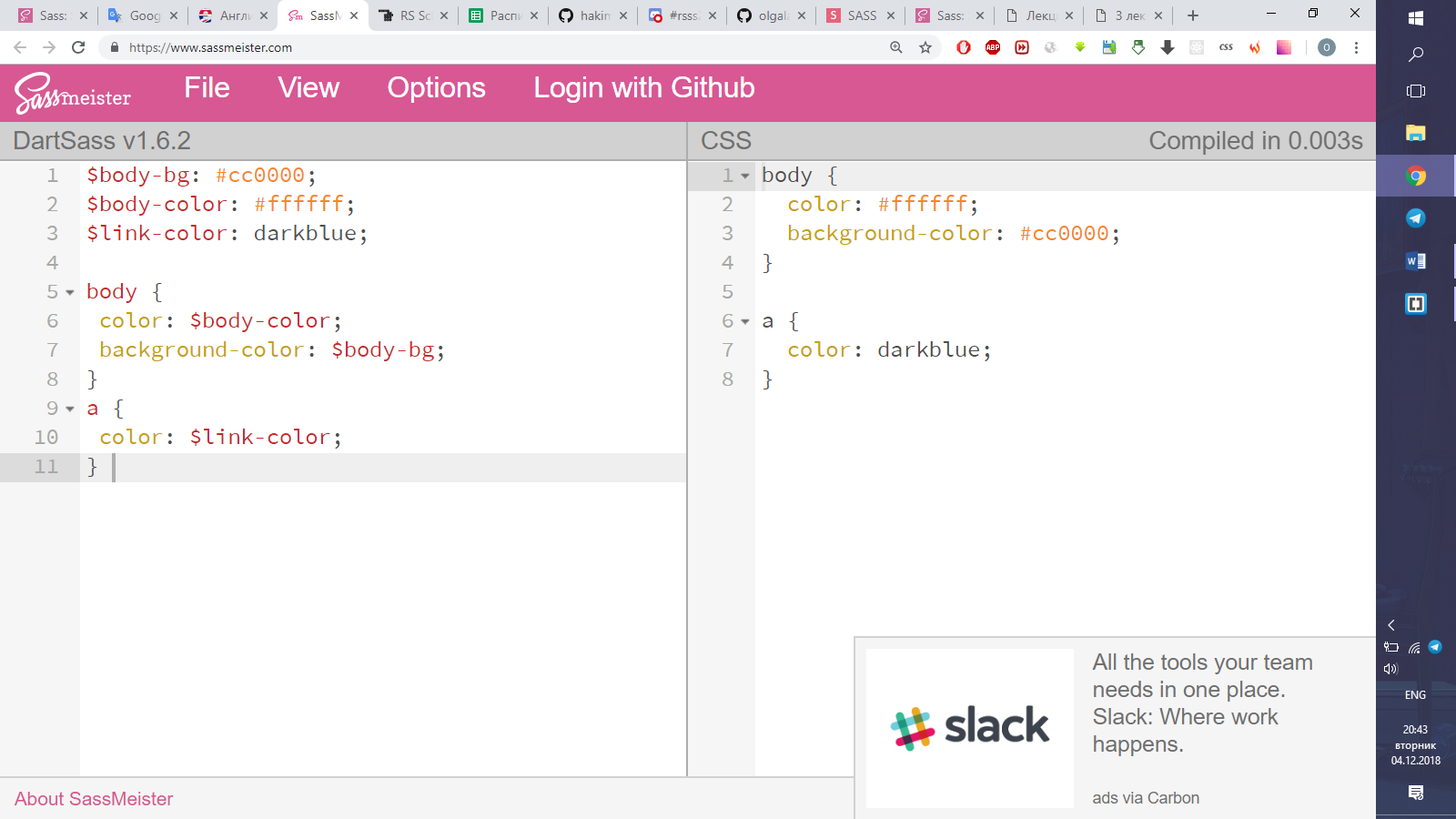
background-color: $body-bg;

}

a {

color: $link-color;

}



 This can be extremely powerful when working with brand colors and keeping them consistent throughout the site.

### Referencing Parent Selectors: &

Sometimes it's useful to use a nested rule's parent selector in other ways than the default. For instance, you might want to have special styles for when that selector is hovered over or for when the body element has a certain class. In these cases, you can explicitly specify where the parent selector should be inserted using the &character. For example:

a {

font-weight: bold;

text-decoration: none;

&:hover { text-decoration: underline; }

body.firefox & { font-weight: normal; }

}

is compiled to:

a {

font-weight: bold;

text-decoration: none; }

a:hover {

text-decoration: underline; }

body.firefox a {

font-weight: normal; }

& will be replaced with the parent selector as it appears in the CSS. This means that if you have a deeply nested rule, the parent selector will be fully resolved before the & is replaced. For example:

#main {

color: black;

a {

font-weight: bold;

&:hover { color: red; }

}

}

is compiled to:

#main {

color: black; }

#main a {

font-weight: bold; }

#main a:hover {

color: red; }

& must appear at the beginning of a compound selector, but it can be followed by a suffix that will be added to the parent selector. For example:

#main {

color: black;

&-sidebar { border: 1px solid; }

}

is compiled to:

#main {

color: black; }

#main-sidebar {

border: 1px solid; }

If the parent selector can't have a suffix applied, Sass will throw an error.

## Nesting

Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML.

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

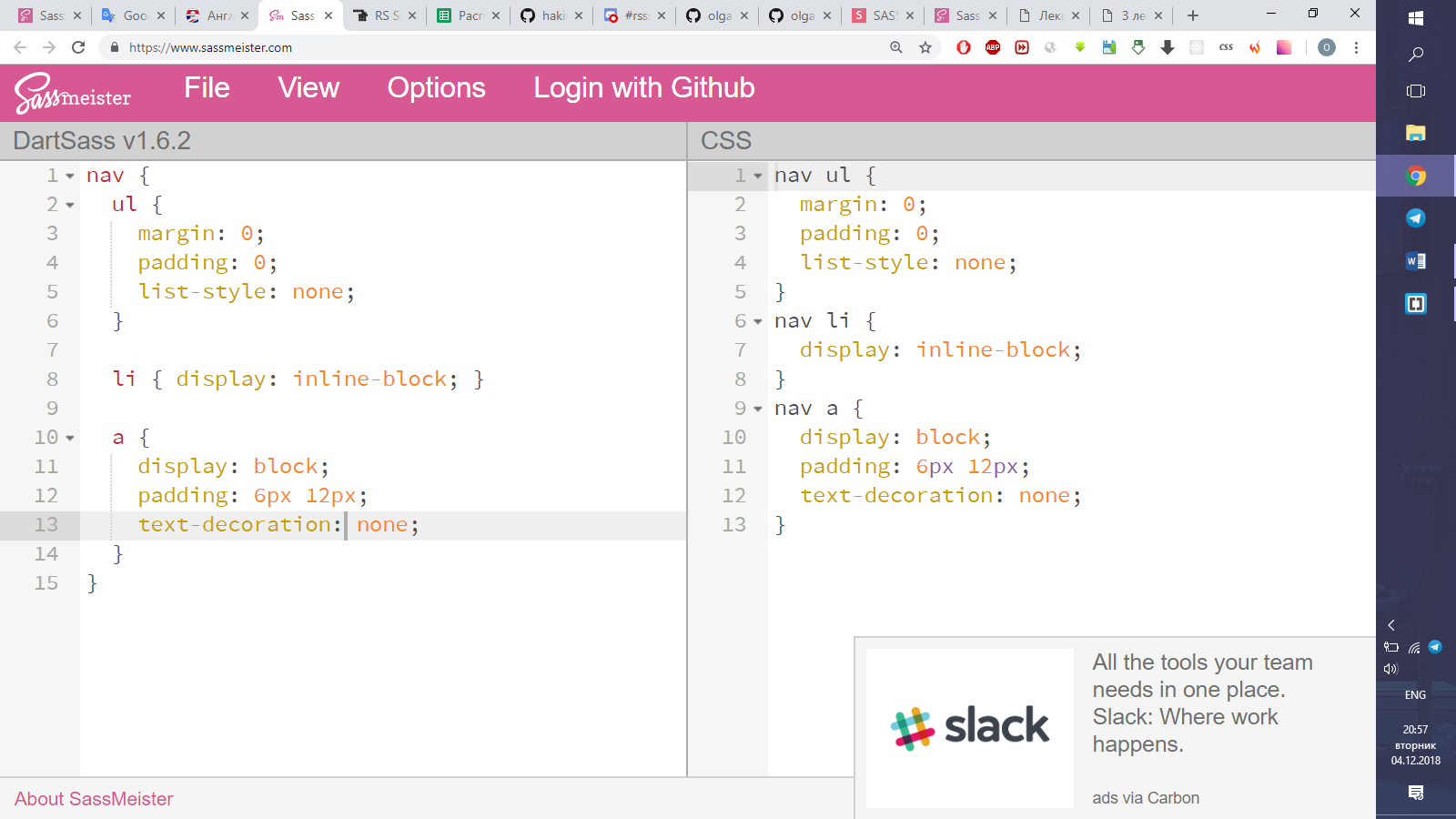
padding: 6px 12px;

text-decoration: none;

}

}

You'll notice that the ul, li, and a selectors are nested inside the nav selector. This is a great way to organize your CSS and make it more readable.



### Nested Properties

CSS has quite a few properties that are in "namespaces;" for instance, font-family, font-size, and font-weight are all in the font namespace. In CSS, if you want to set a bunch of properties in the same namespace, you have to type it out each time. Sass provides a shortcut for this: just write the namespace once, then nest each of the sub-properties within it. For example:

.funky {

font: {

family: fantasy;

size: 30em;

weight: bold;

}

}

is compiled to:

.funky {

font-family: fantasy;

font-size: 30em;

font-weight: bold; }

The property namespace itself can also have a value. For example:

.funky {

font: 20px/24px fantasy {

weight: bold;

}

}

is compiled to:

.funky {

font: 20px/24px fantasy;

font-weight: bold;

}

## Partials

You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. A partial is simply a Sass file named with a leading underscore. The underscore lets Sass know that the file is only a partial file and that it should not be generated into a CSS file. Sass partials are used with the @import directive.

## Import

CSS has an import option that lets you split your CSS into smaller parts.

Let's say you have a couple of Sass files, \_reset.scss and base.scss. We want to import \_reset.scss into base.scss.

Notice we're using @import 'reset'; in the base.scss file. When you import a file you don't need to include the file extension .scss. Sass is smart and will figure it out for you

## Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible. A good use of a mixin is for vendor prefixes. Here's an example for transform.

To create a mixin you use the @mixin directive and give it a name. We've named our mixin transform. We're also using the variable $property inside the parentheses so we can pass in a transform of whatever we want. After you create your mixin, you can then use it as a CSS declaration starting with @include followed by the name of the mixin.