Information technology 3

SASS

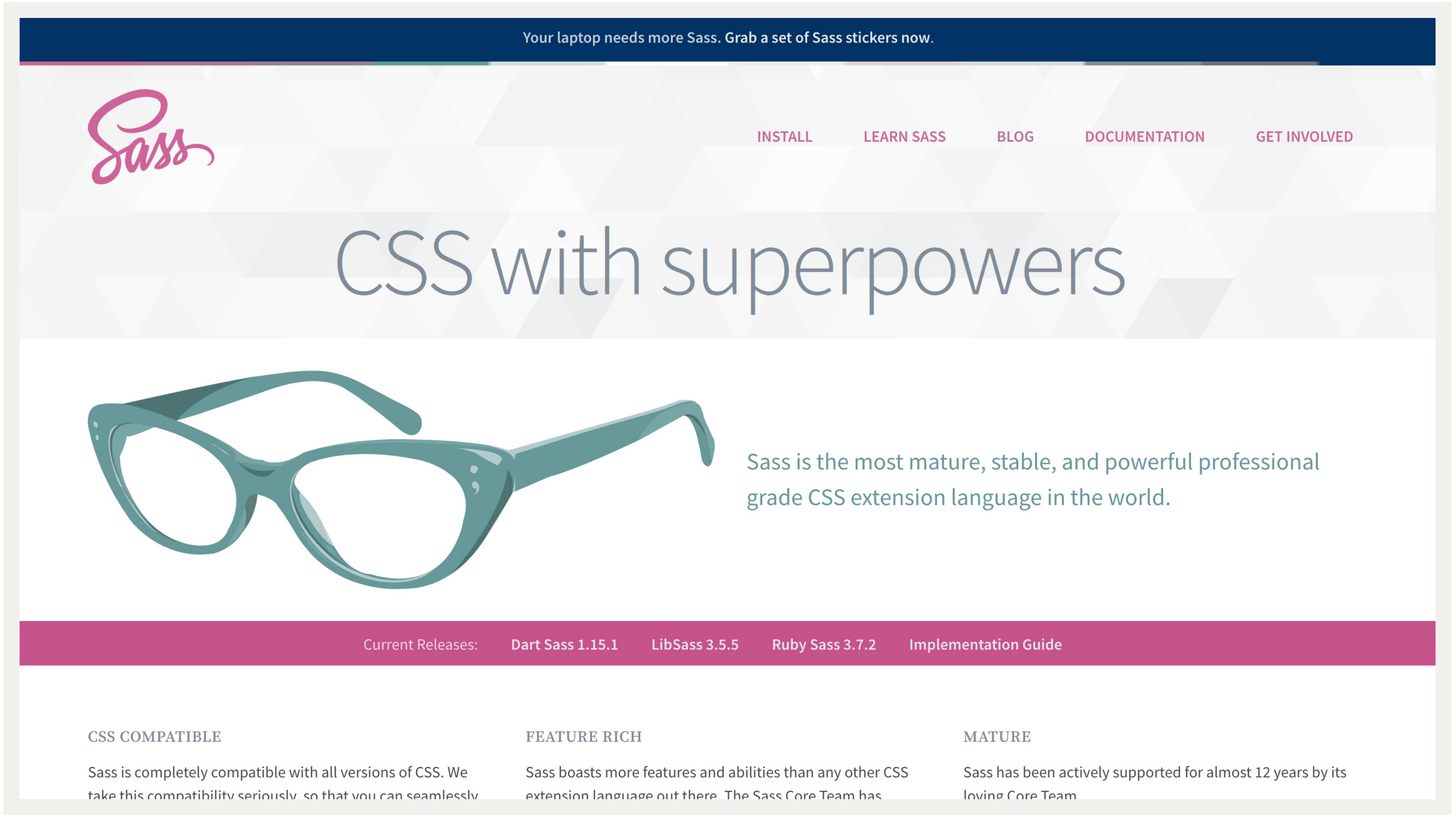
Kristof Michiels

What is SASS?

What is SASS?

- Sass is short for Syntactically Awesome Stylesheets
- Sass a CSS preprocessor
- A layer between your stylesheets and the .css files you serve to the browser
- Allows you to write faster, more efficient, and easier to maintain code

The SASS website


Your laptop needs more Sass. Grab a set of Sass stickers now.

CSS with superpowers

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.

Current Releases: Dart Sass 1.15.1 LibSass 3.5.5 Ruby Sass 3.7.2 Implementation Guide

CSS COMPATIBLE

Sass is completely compatible with all versions of CSS. We take this compatibility seriously, so that you can seamlessly

FEATURE RICH

Sass boasts more features and abilities than any other CSS extension language out there. The Sass Core Team has

MATURE

Sass has been actively supported for almost 12 years by its loving Core Team.

http://sass-lang.com/

Why SASS?

- Enabling “super functionality” in addition to your normal CSS. It then translates (or compiles) that syntax into regular CSS files via a command-line program or web-framework plugin
- To be honest: I was for a long time a reluctant believer in Sass. I write stylesheets by hand!
- But: Sass can be a powerful tool that any style-crafter can easily insert into their daily work

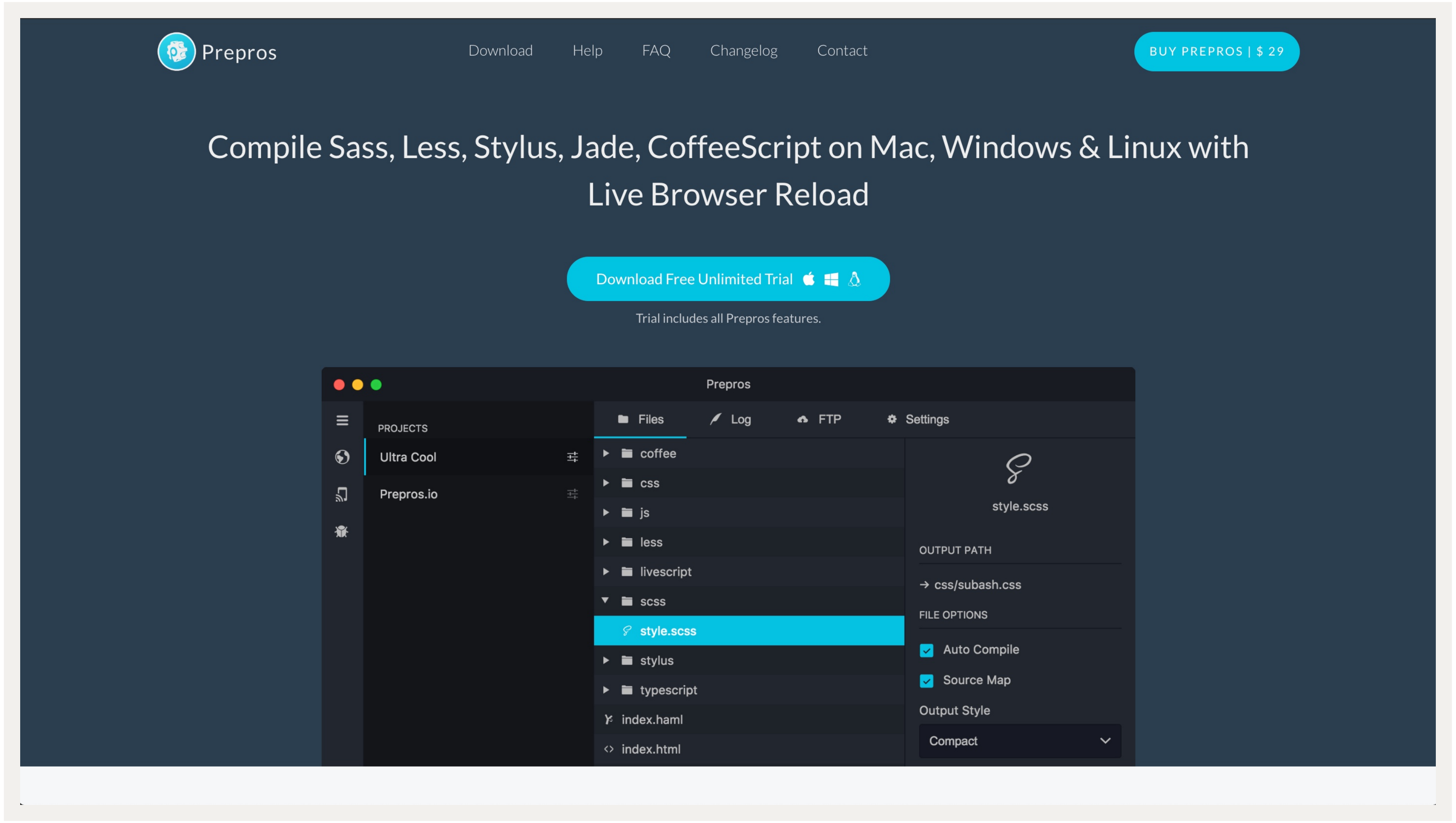
Why SASS?

- The Sass syntax is a superset of CSS3, you don’t have to change anything about the way you write CSS
- All your formatting preferences can remain the same
- The only thing that changes is that you work in .scss files now
- When used properly and intelligently, Sass can be a huge help

Is it different than CSS?

- Getting started with Sass syntax is painless, and you can use as little or as much as you’d like :-)
- The only thing is that your browser can't read Sass
- So we will need a tool that transforms our sass files to regular css
- In this class we will use Prepros. To familarize yourself you could also use Codepen.io

Prepros



Compile Sass, Less, Stylus, Jade, CoffeeScript on Mac, Windows & Linux with Live Browser Reload

Trial includes all Prepros features.

OUTPUT PATH
→ css/subash.css
FILE OPTIONS
☑ Auto Compile
☑ Source Map
Output Style: Compact

http://prepros.io

We will use the SCSS-syntax

- There are 2 flavors of SASS
- We will use SCSS with its closer alignment to normal CSS
- You also need to know that there are other CSS-preprocessors: less, stylus, postcss, compass...
- The original sass-syntax works like this (a bit far off from regular CSS):

```
$pink: #ea4c89

p   font-size: 12px
    color: $pink

p   strong
    text-transform: uppercase
```

What is the DRY principle?

- Every piece of knowledge must have a single, unambiguous, authoritative representation within a system
- The idea is that duplicating code can cause failure and confusion for developers
- CSS is anything but DRY
- SASS is a great tool to make your css dry :-) A few examples will prove this...

Simple SASS example (1)

```scss
$brand-color: #fc3;

a {
    color: $brand-color; }
nav {
    background-color: $brand-color; }
```

Will compile to:

```css
a { color: #fc3; }
nav { background-color: #fc3; }
```

Simple SASS example (2)

```
p { margin-bottom: 20px; font-size: 14px; line-height: 1.5; }
footer { margin-bottom: 20px; font-size: 14px; line-height: 1.5; }
```

Can be written in SASS like this:

```
@mixin default-type {
    margin-bottom: 20px;
    font-size: 14px;
    line-height: 1.5;}

p { @include default-type; }
footer {@include default-type;}
```

Don't edit your .css file!

- Important note: when you’re using Sass, you’ll no longer be editing any .css files
- The .scss files are where it will happen from now on
- The reason being, any changes you make to the .css file will eventually get overridden as soon as you update the .scss and Sass compiles the output
- A typical beginner's mistake :-)

Using SASS

Using Sass

- Focus today: is on quick wins => most easy to add, most productive things
- This will get you started and hopefully make you fall in love with Sass ;-)
- The true capabilities of Sass are overwhelming

1. Variables

- THE reason to start using SASS
- We repeat ourselves so often in a stylesheet. Colors, fonts, background images, widths, etc.—there are patterns that require an epic battle with find-and-replace should any of those patterns be changed. Variables make all of that much simpler and easier to maintain.

```
$color-main: #333;
$font-sans: Helvetica;
body {
    padding: 0 8%;
    font-family: $font-sans;
    font-size: 100%;
    color: $color-main;
    background: #fff url(../img/bg.jpg) repeat-x -80% 0;
}
```

1. Variables: using them for style guides

At the top of your stylesheet:

```scss
$color-main: #333; // black
$color-light: #999; // grey
$color-accent: #ea4c89; // pink
```

Next, using the darken or lighten color function in Sass, we can generate different shades of color that will always be based on the brand palette

1. Variables: using them for style guides

Let’s darken the pink (#ea4c89) by 30%:

```scss
section.secondary {
    background: darken($color-accent, 30%);
}
```

We can also lighten colors:

```scss
section.secondary {
    background: lighten($color-accent, 30%);
}
```

Lots of functionality! http://sass-lang.com/

2. Mixins

- Mixins allow you to define and reuse blocks of styles
- Use mixins to define a group of styles just once and refer to it anytime those styles are needed
- Shared styles can be abstracted into mixins, and you'll still have the ability to override or augment those styles with additional rules

2. Mixins

```scss
@mixin title-style {
    margin: 0 0 20px 0;
    font-family: $font-serif;
    font-size: 20px;
    font-weight: bold;
    text-transform: uppercase;
}
section.main h2 {
    @include title-style;
}
section.secondary h3 {
    @include title-style;
    color: #999;
}
```

2. Mixins (with arguments)

- Sass mixins can also take arguments that we pass to the mixin when we call it
- You can pass multiple arguments by separating the values with commas in the mixin definition

```scss
@mixin title-style($color) {
    margin: 0 0 20px 0;
    font-family: $font-serif;
    font-size: 20px;
    font-weight: bold;
    color: $color;
}

section.main h2 {
    @include title-style(#c63);
}
```

2. Mixins (with arguments and their defaults)

When you use mixin arguments, it's often convenient to define defaults. That way, you simply call the mixin with no arguments, if that's the norm, but can still pass in overrides.

```
@mixin title-style($color, $background: #eee) {
    margin: 0 0 20px 0;
    font-family: $font-serif;
    font-size: 20px;
    color: $color;
    background: $background;}

section.main h2 {
    @include title-style(#c63);}
section.secondary h3 {
    @include title-style(#39c, #333);}
```

2. Mixins (with arguments and their defaults)

Additionally, when you have multiple, default arguments defined for a mixin, you can override those selectively without having to redefine them all.

```scss
@mixin title-style($color: blue, $background: green) {
        font-family: $font-serif;
        font-size: 20px;
        color: $color;
        background: $background;
}

section.main h2 {
        @include title-style($background: pink);
}
```

3. @import

- The @import rule: import other stylesheets, and create a mixins library while you are at it ;-)
- Sass merges them into a single CSS file
- Handy: a single CSS means fewer HTTP connections. Performance!

```scss
// Import other files
@import "reset.scss";
@import "variables.scss";
@import "mixins.scss";
// Site-specific styles
body {...}
```

4. @extend

- Ever find yourself writing a CSS class that has the same styles as another class, except for just a few other rules?
- Using @extend also allows us to be terser in our semantics, defining class names based on meaning rather than appearance

```html
<h2 class="alert-positive">This is a positive alert!</h2>
```

```css
.alert-positive {
        @extend .alert;
        background: #9c3;
}
```

4. Multiple @extend-s

```scss
.alert {
    padding: 15px;
    font-size: 1.2em;
    text-align: center;
    background: $color-accent;
}
.important {
    font-size: 4em;
}
.alert-positive {
    @extend .alert;
    @extend .important;
    background: #9c3;
}
```

5. Placeholder selectors (+ @extend)

- What if the class you're extending exists solely for the purpose of extending other styles? In other words, you might create a class that's not used on its own
- Enter placeholder selectors, which allow you to define "phantom" classes that won't appear in the outputted CSS on their own. You can reference placeholders using @extend

```
%button {
    padding: 10px;
    font-weight: bold;
    background: blue;
    border-radius: 6px;
}
.buy {
    @extend %button; }
```

6. Nesting rules

- With Sass, you can nest CSS rules inside each other instead of repeating selectors in separate declarations
- The nesting also reflects the markup structure
- Nesting in Sass means less typing, using indentation to reflect the selector (and property) formation

```
ol {
    margin: 0;

    li {
    color: red;
    border: 1px solid black;
    margin-bottom: 5px;
    }
}
```

6. Nesting namespaced properties

In addition to nesting rules, you can nest properties that share a namespace in Sass (e.g., font-family, font-size, font-weight, etc.) like so:

```
header h1 {
padding: 15px 0;
font: {
    size: 54px;
    family: Jubilat, Georgia, serif;
    weight: bold;
}
text: {
    transform: uppercase;
    decoration: underline;
    align: center;
}
line-height: 1;}
```

7. Referencing parent selectors with &

- Sass adds the ability to reference the current parent selector using the ampersand as a special placeholder
- The ampersand inserts the parent selector, in this case a

```
a {
    font-weight: bold;
    text-decoration: none;
    color: red;
    border-bottom: 2px solid red;

    &amp;:hover {
        color: maroon; border-color: maroon;
    }
}
```

7. Referencing parent selectors with &

The ampersand is also useful in inserting overrides that happen in the presence of a specific class

```
section.main p {
    margin: 0 0 20px 0; font-size: 18px; line-height: 1.5;

    body.store &amp; {
        font-size: 16px; line-height: 1.4;
    }
}
```

8. Commenting in Sass

You can ensure important comments (copyright info, attribution, notes on hacks, etc.) appear in the compressed style output by inserting an ! as the first character of the comment:

```
/*! This is a multi-line comment that will appear in the final .css file */
```

Single-line comments use the // prefix at the beginning of each line and aren't included in the final output, so you can safely use them for private comments:

```
// This comment will not make it to the final CSS file
```

IT3: SASS

Questions?

kristof.michiels01@ap.be