SASS

Syntactically Awesome Stylesheets

1

What is SASS?

SASS is a CSS preprocessor

Runs on Ruby

Extends the functionality of CSS

Eliminates (or, at least mitigates) many of the traditional CSS pain points

Source files are created by the user (more, later) and 'compiled' by SASS into the output CSS file

2

Why Use SASS?

It is a pre-processing language which provides indented syntax (its own syntax) for CSS

It provides some features, which are used for creating stylesheets that allows writing code more efficiently and is easy to maintain

3

Why Use SASS?

It is a super set of CSS, which means it contains all the features of CSS and is an open source pre-processor, coded in Ruby

It provides the document style in a good, structured format than flat CSS

It uses re-usable methods, logic statements and some of the built-in functions such as color manipulation, mathematics and parameter lists

4

Features of SASS

It is more stable, powerful, and compatible with versions of CSS

It is known as syntactic sugar for CSS, which means it makes easier way for user to read or express the things more clearly

It uses its own syntax and compiles to readable CSS

You can easily write CSS in less code within less time

It is an open source pre-processor, which is interpreted into CS

5

Advantages of SASS

It allows writing clean CSS in a programming construct

It helps in writing CSS quickly

As Sass is compatible with all versions of CSS, we can use any available CSS libraries

It is possible to use nested syntax and useful functions such as color manipulation, mathematics and other values

6

Disadvantages of SASS

It takes time for a developer to learn new features present in this pre-processor

If many people are working on the same site, then should use the same preprocessor

Some people use Sass and some people use CSS to edit the files directly. Therefore, it becomes difficult to work on the site

There are chances of losing benefits of browser's built-in element inspector

7

SASS Installation

8

Installing SASS

SASS is s Ruby Gem, so you have to have Ruby installed first

Ruby Installer is probably the easiest way for Windows

Get the latest version that your PCs architecture will support

https://rubyinstaller.org/

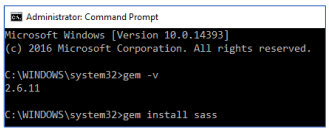
9

Installing SASS

Once Ruby is installed, open a command prompt

Enter **`gem -v`** to ensure that Ruby has been successfully installed

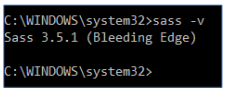
Enter **`gem install sass`**



10

Installing SASS

Enter **`sass -v`** to confirm successful installation



11

Installing SASS

It isn't required that you install SASS on your own PC

I've confirmed that it does work on the lab computers, but is a little more convoluted

If you install to your laptop, the **`sass`** command will be installed on the system's PATH and can be run from anywhere

If you use a lab PC, you'll have to navigate to the **`C:\ruby24-x64\bin`** folder

12

Installing SASS

So, if your project files are on your E:\ drive, for example, let's say

```
E:\csci1720\lab6
```

On a lab machine, you'd have to navigate to

```
C:\ruby24-x64\bin
```

and run

```
sass --watch E:\csci1720\lab6\scss:E:\csci1720\lab6\css
```


13

Installing SASS

If you're working from your own machine, however, all you'd have to do (from the command prompt) is

```
cd E:\csci1720\lab6
```

```
sass --watch scss:css
```

Then minimize the command prompt

While SASS is running, anytime you change your source code, SASS will detect it and recompile your CSS

14

Installing SASS

IMPORTANT POINT:

In a SASS environment, you never touch the CSS file!

15

SASS Source Code

There are two ways (languages, if you will) to create SASS source code

SASS (e.g., main.sass)

- Uses indentation as delimiters
- Instead of braces {}

SCSS (e.g., main.scss)

- Superset of CSS
- Uses CSS syntax we're familiar with

16

SASS Source Code

There are two ways (languages, if you will) to create SASS source code

SCSS seems to be more popular

Makes sense, since any valid CSS is also valid SCSS

Syntax is much the same

17

SASS Variables

18

SASS Variables

One of the (many) neat things about SASS is variables

Like variables in other languages, they allow easy global changes

Instead of having to find each instance, for example, of

```
color: #333;
```

In a long CSS file, you can change

```
$fontColor: #333;
```
to
```
$fontColor: #444;
```

in one place, and it'll propagate throughout the entire resulting CSS file

19

Sass Variables

Using the $ (dollar) symbol

Can be used to store colors, size, etc…

Usable to set default background-color, font-color, font-size, etc…

```
$link-color: #ffffff;
$v-link-color: #646363;

a {
  color: $link-color;
}
a:visited {
    color: $v-link-color;
}
```

```
body a {
  color: white;
}
a:visited {
    color: #646363;
}
```


20

Sass Variables

SASS variables can be inserted as CSS properties using #{}

```
$border-side:top;
$border-color:blue;
$border-style:ridge;
$border-width:15px;
…
border-#{$border-side} :
  $border-width $border-style $border-
color;
```

```
border-top : 15px ridge blue
```


21

SASS Nesting

When writing HTML you've probably noticed that it has a clear nested and visual hierarchy

CSS, on the other hand, doesn't

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

Be aware that overly nested rules will result in over-qualified CSS that could prove hard to maintain and is generally considered bad practice

22

SASS Nesting

```
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

When you generate the CSS you'll get something like this

23

SASS Nesting

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li { display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav li {
  display: inline-block;
}

nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```


24

SASS Nesting

```
img {
     width: $pic-width;

     &.img-l {
          float: left;
          margin-right: 2em;
     }

     &.img-r {
          float: right;
          margin-left:
     2em;
     }
}
```

You can generate selector-specific classes by nesting the code inside the selector

Using the ampersand is similar to the 'this.' directive in OO languages

So this code, when processed, produces this:

25

SASS Nesting - Classes

```
img {
     width: $pic-width;

     &.img-l {
          float: left;
          margin-right:
2em;
     }

     &.img-r {
          float: right;
          margin-left:
          2em;
     }
}
```

```
img {
  width: 15em;
}
img.img-l {
  float: left;
  margin-right: 2em;
}
img.img-r {
  float: right;
  margin-left: 2em;
}
```


26

SASS Partials

27

Sass Partials

You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files

This is a great way to modularize your CSS and help keep things easier to maintain

A partial is simply a SCSS file named with a leading underscore. You might name it something like _variables.scss

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

28

SASS Partials

File name: _variables.scss

```
//Variables

// Colors
$text-color: #333;
$bg: #ddd;
$text-color-neg: #ddd;
$bg-neg: #333;
$font-blue: #005;
$font-red: #500;
$a-color: #f00;
$a-hover: #eee;

// Dimensions
$pic-width: 15em;
$min-w: 328px;
$max-w: 1024px;
$p-w: 40em;

// Borders
$main-border: 1px solid $font-blue;
$pic-border: 1px solid $font-red;
```


29

SASS Partials

Resulting code

```
/* main.scss */
@import 'variables';

body {
    background: $bg;
    color: $text-color;
    font-family: "Quattrocento Sans";

    a {
        text-decoration: none;
        font-weight: bold;
        color: $a-color;
    }

    a:hover {
        color: $a-hover;
    }
}
```


30

SASS Partials

Resulting code (main.css)

```
body {
  background: #ddd;
  color: #333;
  font-family: "Quattrocento Sans"; }
  body a {
    text-decoration: none;
    font-weight: bold;
    color: #f00; }
  body a:hover {
    color: #eee; }

h1, h2, h3 {
  color: #005;
  text-align: center; }
```


SASS Mixins

Sass Mixins

Mixins are kind of developer defined functions
The developer can make them for clear SASS
Two kind of mixins
Parameterless
- Get default styles every time

With parameters
- Get style based on some parameters
- Gradient, borders, etc…

Sass

// Use `@mixin mixin-name`

// Then the styles are normal SASS

```
@mixin clearfix{
    zoom:1;
    content:"";
    height:0;
    clear:both;
}
```

```
ul#main-nav{
  @include clearfix;
  …
}
```


34

SASS Extend/Inheritance

35

Sass Extend/Inheritance

This is one of the most useful features of Sass

Using @extend lets you share a set of CSS properties from one selector to another

It helps keep your Sass very DRY (“Don’t Repeat Yourself”)

In our example we're going to create a simple series of messaging for errors, warnings and successes

36

Sass Extend/Inheritance

```
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: green;
}
```

What the above code does is allow you to take the CSS properties in .message and apply them to .success

The magic happens with the generated CSS, and this helps you avoid having to write multiple class names on HTML elements

This is what it looks like:

37

Sass Extend/Inheritance

```
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: green;
}
```

```
.message, .success {
  border: 1px solid #cccccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}
```


38

SASS Operators

39

Sass Operators

Doing math in your CSS is very helpful

Sass has a handful of standard math operators like +, -, *, /, and %

In this example we're going to do some simple math to calculate widths for an aside & article

40

Sass Operators

```
.container { width: 100%; }

article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}

aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

We've created a very simple fluid grid, based on 960px

Operations in Sass let us do something like take pixel values and convert them to percentages without much hassle

The generated CSS will look like:

41

Sass Operators

```
.container {
  width: 100%;
}
article[role="main"] {
  float: left;
  width: 62.5%;
}
aside[role="complementary"] {
  float: right;
  width: 31.25%;
}
```


42

Questions?

43

Copyrights

44