Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska Europejski Fundusz Społeczny

"ZPR PWr - Integrated Development ProgramWrocław University of Technology"

Wrocław University of Science and Technology

Advanced Web Technologies

Lab

Subject: SASS and SCSS, CSS preprocessors

Prepared by: mgr Piotr Jozwiak

Date: July 2020

Number of hours: 2 hours

Table of Contents

Entry

Thanks to CSS, we can design the appearance of the website exactly as we want it. It is a very simple language that identifies desired HTML elements and changes them accordingly. Compared to direct styling in HTML, CSS is great and approachable. It allows you to separate the visual part from the data. However, the CSS style sheet language has its severe limitations from a development point of view. Many web designers are looking for a more convenient and efficient way to work on implementing style sheets. CSS preprocessors are the answer to this need. By using a style sheet language that is an extension of basic CSS, design work can be greatly facilitated.

Using preprocessors,programmers can access a range of features that CSS might have if it wasn't such a simple language: functions, variables, mixins, and more. Preprocessors also save a significant amount of time by allowing you to reuse predefined properties instead of saving them in multiple copies.

Let's look at the main idea between pure CSS and the definition for the preprocessor:

```
/* WITHOUT A PREPROCESSOR */
.header-large{
   font-
   family:Tahoma;font-
   weight:bold;font-
   size:48px;
   font-variant:small-
   caps;text-
   decoration:underline;color
   :#ff0000;
}
.header-medium{
   font-
   family:Tahoma;font-
   weight:bold;font-
   size:36px;
   font-variant:small-
   caps;text-
   decoration:underline;color
   :#ff0000;
}
.header-small{
   font-
   family:Tahoma;font-
   weight:bold;font-
```

```
/* WITH A PREPROCESSOR */
.header-large{
   font-
   family:Tahoma;font-
   weight:bold;font-
   size:48px;
   font-variant:small-
   caps;text-
   decoration:underline;color
   :#ff0000;
}
.header-medium{
   .large-
   heading;font-
   size:36px;
}
.header-small{
```

In the above example, it is clear that the readability of the definitions increases because we have eliminated redundant repetitions. This is just a small example of what the CSS preprocessor can do. More secrets will be revealed in today's lab.

The most famous of these extension languages is SASS. What exactly is "**Syntactically Awesome Style Sheet**"?

What is SASS?

To understand what SASS is and why we need it, we must first understand CSS. Websites are based on HTML. However, HTML does not provide good solutions for presenting data. Cascading Style Sheets (CSS) are an extension of HTML pages that provide a layer of appearance definition. CSS sits on top of the HTML code, just like a template, and defines the design of each element on the page: font, font color, background. Web designers can set all these design elements with CSS.

For example, to set all HTML headers to 22pt with the Calibri font, we define this in a style sheet where all the guidelines for the appearance of the site are given. These rules are usually in a separate file that is then referenced by HTML pages. This dramatically reduces the amount of information on the HTML page and allows you to reduce the transfer by storing the style definitions once downloaded in the browser's cache.

But CSS has its serious limitations, which is especially clear when we have been working with the language for many years. The advantage of CSS is also its disadvantage: the language is designed very simply. SASS makes everything a bit more sophisticated and greatly simplifies the work of creating a project.

Unfortunately, not all browsers can interpret SASS definitions the way they can today with CSS definitions. Younger web developers may not remember this anymore, but the beginnings of CSS also looked very hard. Each browser introduced some differences in understanding the definition of style sheets. Today, this problem is rather marginal and CSS works flawlessly in every browser. However, despite the fact that SASS has been around for 10 years, it has not been so unified. That is why today it is used as a preprocessor that generates CSS files based on its definitions.

The language used to implement SASS is called SassScript. The preprocessor was designed on the basis of YAML language. Therefore, SASS syntax is largely different from CSS.

Why use SASS?

When you start working on web design, it seems that CSS may be enough. After all, pure CSS makes it possible to create very attractive websites. Therefore, if you want to use SASS, you cannot omit the use of CSS. The old-fashioned style sheet language will continue to emerge as the basis for presentation in the future. SASS and other languages are based on CSS framework.

However, by enabling SASS, we get many additional features not available in CSS:

- **Variables**: With SASS we can save information in variables to use later. For example, it is possible to centrally store a color value under a variable.

- **Mathematical functions**: In SASS you can also use math operations like +, -
, *, / or%. This allows, for example, to influence size specifications in a very dynamic way.
- **Functions**: make it easier to work on a project. They allow you to modify color values or parse lists, among other things.
- **Loops**: Another advantage of SASS is the ability to configure loops. They are repeated until they reach a certain state.
- **Mixins**: simply put, these are templates. We can create them ourselves or simply integrate them with our own code when using the framework.
- **Indentations**: the original SASS (unlike SCSS) works with indentation and line breaks to structure your code. We don't need parentheses to display nesting or semicolons to mark the end of a line.
- **Nesting**: CSS does not allow you to work with nesting in your code. However, SASS gives users the ability to visually represent dependencies to reduce redundancy and simplify the writing process.
- **Inheritance**: It is possible to inherit properties from one selector to another. This saves you writing code and makes your code more readable.
- **Partial Files**: Definitions can be divided into more smaller files, which are included with the appropriate commands.,

SCSS vs. SASS

SASS is not one syntax but two. SASS as the original form is otherwise called "indented syntax" due to the fact that it uses YAML as a way to define rules. But there is another variant that is more CSS syntax oriented and is therefore called sassy CSS (SCSS). As if that wasn't enough, this syntax appears in the very definition of CSS in version 3. Therefore, it can be assumed that in some time browsers will correctly interpret SCSS without the need to compile it first. The question arises - why two syntaxes for the same thing? Before answering that question, let's take a deeper look at the differences. Then the answer will reveal itself.

The original SASS syntax works with indentation and line breaks as it uses a process adapted from YAML. To end a line of code, just insert a line break. Indentation works simply with the tab button. Groups, called declaration blocks, are created by changing positions in a typeface. This is not possible with CSS alone. In this case, use parentheses for grouping and semicolons for property declarations. This is exactly what is needed for SCSS. This makes the code more concise and easier to copy and paste. We don't need to care about pairs of parentheses, etc. On the other hand, many users complained that you have to learn two different syntaxes to complete one task. Therefore, a second variant called SCSS was created.

The primary advantage of SCSS is that it is a superset of CSS. In other words, every CSS file is also SCSS – but not the other way around. That's a lot. If a developer learns CSS, then while learning SCSS, he develops only those elements that introduce new functionality to style sheets.

Now you can see the answer to the question posed earlier. The new syntax is intended to bring the preprocessor closer to its resulting CSS file to facilitate the learning process. There is one more advantage. When the developer

decides that he wants to start using SCSS, all he has to do is change the extension of the CSS files to SCSS and preprocess it. The change itself will be incremental. This is not possible with the SASS syntax.

Installing the SASS preprocessor

It's time for examples in practice. Before we start, we need to install the necessary software. I will show below how to install SASS on both Linux and Windows.

Installation on Linux

On Linux, the easiest way to install SASS is using the npm package manager. This can be done with the command:

```
npm install -gsass
```

This solution has its drawbacks. Unfortunately, suchapproach will install the SASS engine in pure JavaScript, which is slower than other available builds. However, it retains full functionality.

Another method is to install Ruby and then install SASS using the gem manager. This can be done as follows:

```
apt-get install ruby-full build-essential rubygems
gem install sass
```

Windows installation

On Windows, it's best to use the chocolatey package manager. Of course, it is not available on every Redmont system, but I encourage you to start working with it. It significantly facilitates work with installers - like the manager known from Debian -aptitude. You can read more about the manager here:https://chocolatey.org/.

If choco is already installedon our system, then installing SASS will be as easy as in the case of Linux. Just execute the following command:

```
choco installsass
```

If, however, for some reason we do not want to install the manager on the system, we will be forced to install Ruby, and then use the gem installer, as in the example with Linux installation.

You can read more about installing SASS here:https://sass-lang.com/install

SASS installation verification

To verify the operation of the preprocessor, the easiest way is to execute the command displaying its version. The following command is used for this:

```
sass --version
1.26.10 compiled with dart2js 2.8.4
```

If the result is the number of the installed SASS version, it means that we are ready to work with the preprocessor.

Starting the preprocessor

We have the software ready. So it's time for some examples. Before we discuss them, we need to learn how to use the preprocessor.

The easiest way to get started is by defining two folders:

- css
- scss

As the name suggests, the first one will be used to store CSS files. It is to this folder that the preprocessor will generate style files. The second folder is the container for our source files.

In our example, we will use SCSS files that have the .scss extension. However, before we write the first file, we will start the preprocessor to work in continuous mode. This means that it will monitor the input folder for any modifications appearing there, and if any, it will automatically compile the scss files to css files.

To do this, we execute the command:

```
sass --watchscss:css
```

The message “Sass is watching for changes. Press Ctrl-C to stop." It tells us that the preprocessor is running and listening for a change. The resulting files will automatically appear in the css folder. It's time for examples.

SASS basics

Let's move on to discussing the basic capabilities of SASS. In our examples, we will use the SCSS syntax, because it is easier to compare with CSS - you can see what has changed right away. Nothing stands in the way of writing examples in SASS.

Variables

The concept of a variable does not need to be explained to anyone. This is perhaps the most missing element in CSS. Due to the lack of this concept, e.g. the definition of the main color on the page had to be written in hundreds of places. After changing it to a different value, tracking all the places using it was not an easy task. SASS introduces the concept of variables by using the $ symbol in front of a name to indicate that an item is a variable definition. The following example best illustrates the idea:

SCSS	CSS

```
$font stack:    Helvetica,sans-serif;
$primary-color:#333;

bodysuit{
  font:100%$font
  stack;color:$primary-
  color;
}
```

```
bodysuit{
  font:100%Helvetica,sans-
  serif;color:#333;
}
```

At the time of processing, SCSS fetches the variables we define for $font-stack i **$primary-color**and generates standard CSS with variable values placed in style definitions. This solution is extremely efficient when working with, for example, colors and maintaining their consistency throughout the site.

Nesting

Nesting is very visible in pure HTML. This is because HTML is a kind of XML. However, CSS, although it relates directly to HTML, does not have such a concept.

SASS allows you to nest CSS selectors in the same way as HTML hierarchy. This makes it easier and more intuitive to define rules for complex rules. Below is an example of using this functionality:

```
nav{
    st{
      margin:0;
      padding:0;
      letter-style:none;
    }

    li{display:inline block; }

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

```
nav
  st{margin:
  0;
  padding:0;
  letter-style:none;
}
navli{
  display:inline block;
}
nav a{
  display:block;padding:
  6px 12px;text-
  decoration:none;
}
```

Mixins

Some things in CSS are tedious to write, especially in CSS3 in the context of browser related prefixes. Mixin lets you create groups of CSS declarations that can be reused on your site. You can even pass values to make your mixin more flexible. This is best illustrated by an example from property transform:

```
@mixintransform($properties) {
    -webkit-transform:$properties;
    -ms-
    transform:$properties;tra
    nsform:$properties;
  }

.box{
```

CSS

```
x{
  -webkit-transform:rotate(30deg);
  -ms-
  transform:rotate(30deg);trans
  form:rotate(30deg);
}
```

```
    @includetransform(rotate(30deg));
}
```

To define a mixin you needuse the @mixin directive and give it a name. In our example, the mixin was named transform. We also used the $property variable, which acts as a parameter. After creating the mixin, below is how to use it to define the .box style. To use a mixin, use the @include directive in the place of use and then indicate the name of the mixin.

Extend/Inheritance

Extend is one of the most usefulSASS functionality. Using the @extend expression, we can share style sheet properties between different selectors. This functionality allows you to keep your code clean. This is similar to class inheritance. This is well illustrated by the following example, in which I present simple selectors for styling messages about errors, warnings and success of some action.

SCSSCSS

```
/* This CSS will print because
   %message-shared is extended. */
%message-shared{
    border:1pxsolid
    #cc;padding:10px;
    color:#333;
  }

  /* This CSS won't print because
     %equal-
heights is never extended. */
  %equal-
    heights{display:
    flex;flex
    wrap:wrap;
  }

  .message{
    @extend%message-shared;
  }

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

  .error{
```

```
/* This CSS will print because
   %message-shared is extended. */
.warning,.error,.success,.message{bord
  er:1pxsolid #cc;
  padding:10px;c
  olor:#333;
}

.success{
  border-color:green;
}

.error{
  border-color:red;
}

.warning{
  border-color:yellow;
}
```

```
}

.warning{
  @extend%message-
  shared;border-
  color:yellow;
}
```

The above code makes the .message selectors,**.success**, .error and .warning behave as %message-shared. This means that wherever @extend%message-shared, the definition from the referenced selector will be inserted at the given position. The magic happens in the generated CSS, where each of these classes will get the same CSS properties as%message-shared. This helps to avoid having to write multiple class names on HTML elements.

Note that the selector%equal-heightsit is not generated in the output file. This is because it has not been used anywhere - so the compiler will skip its definition.

Operators

Sometimes doing basic mathematical expressions is very desirable when defining styles. This is where operators such as +, -, *, / and % come in handy. They allow you to build simple expressions.

SCSSCSS

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

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

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

In the example above, we have created a simple fluid grid based on the size of 960px. Operators in SASS make it easier for us to work with converting pixel values to percentages.

Functions

In SASS, we also find many functions that support complex calculations. Their list is available e.g. here:https://www.w3schools.com/sass/sass_functions_string.asp.In addition, you can use @if, @else control expressions (https://sass-lang.com/documentation/at-

rules/control/if) Whether@for and @while loops. There are a lot of possibilities, the more that we can add our own functions. Let's look at the following example:

SCSS

```
@functionarea($base,$exponent) {
    $result:1;
    @for$_from1through$exponent{
      $result:$result*$base;
    }
    @return$result;
  }

  .sidebar{
    float:left;
    margin-left:area(4,3) *1px;
  }
```

CSS

```
.sidebar{float
  :left;
  margin-left:64px;
}
```

As you can see in the example above, we have defined the pow function, whose task is to calculate the power. This is not possible in pure CSS, however, with SASS, complex functionalities can be wrapped into functions and made available as libraries to perform repetitive tasks.

Summary

We discussed the basic principles of operation together with examples of CSS preprocessors. They perform a very useful function in the world of web design. The more we start using this technology, the more we will see its benefits. More detailed information about how SASS works can be read in the documentation:https://sass-lang.com/documentation/syntax.