WEB INTERFACE DESIGN TOPIC 2: CSS PREPROCESSORS

CFGS - DIW 2

Índice

1. What are CSS preprocessors?

- CSS preprocessors are tools that extend the functionality of CSS (Cascading Style Sheets) by adding features such as variables, mixins, functions, and nesting. They allow developers to write CSS in a more efficient and maintainable way by providing advanced features that are not available in standard CSS.
- Some popular CSS preprocessors include:
  - Sass (Syntactically Awesome Stylesheets)
  - Less (Leaner CSS)
  - Stylus
  - PostCSS
- These preprocessors help streamline the development process, improve code organization, and make it easier to manage and reuse CSS styles across a project. They typically require a compilation step to convert the preprocessed code into standard CSS, which is then used by web browsers to style the web pages.

2. Less

- Less is a CSS preprocessor, which means it extends the functionality of CSS by adding additional features and capabilities. Less stands for "Leaner CSS" and is a popular choice among web developers for writing more maintainable and modular CSS code.
- Less provides a set of features that simplify and enhance the process of writing CSS stylesheets.

2.1 Resources

- The first site I always recommend is lesscss.org, where you can find excellent information and very rich content tutorials.
- If you are comfortable with English, you might be interested in checking out the introduction to LESS tutorial by Smashing Magazine.
- We are going to work with an extension called "Easy Less" available in the Visual Studio Code marketplace.

2.2. Variables

- In Less, variables are used to store and reuse values throughout your stylesheets. They make it easier to manage and update styles by allowing you to define values in one place and use them across multiple rules.
- To declare a variable in Less, you use the @ symbol followed by the variable name and the value assigned to it. Here's an example:

```
@blue: #00214D;
@red: #CF142B;
@white: #FFF;
@black: #000;
@baseFontSize: 15px;
@baseLineHeight: 22px;
```

2.2. Variables

```
@blue: #00214D;
@red: #CF142B;
@white: #FFF;
@black: #000;
@baseFontSize: 15px;
@baseLineHeight: 22px;
```

- To give you an idea, the following LESS example (created with the variables above) compiles as shown on the right side:

```
h1 { color: @red; }
h2 { color: @blue; }
h3 { color: @black; }
p { color: @black;
font-size: @baseFontSize;
line-height: @baseLineHeight;}
```

```
h1 { color: #CF142B; }
h2 { color: #00214D; }
h3 { color: #000; }
p { color: #000;
font-size: 15px;
line-height: 22px; }
```

2.2. Variables

```
@primary-color: #ff0000;
@font-size: 16px;
@background-color: #f0f0f0;
```

```
body {
  color: @primary-color;
  font-size: @font-size;
  background-color:
@background-color;
}

h1 {
  font-size: @font-size * 2;
/* Variables can also be used
in calculations */
}
```

2.2. Variables

- By using variables, you can easily update the values assigned to them, and the changes will propagate throughout your stylesheets. This allows for better maintainability and flexibility in your Less code

2.2. Variables

Try the previous examples using Easy Less in Visual Studio Code.

2.3. Mixins

- Un mixin en LESS es un conjunto de propiedades CSS agrupadas. Esto nos permite utilizar el grupo en sí y no repetir mil veces las mismas propiedades.
- Si estuviésemos hablando de programación (como tal), esto sería lo más parecido a un array. Otra forma de entenderlo podría ser una variable con múltiples valores.
- Los mixins son, junto a las variables y funciones, uno de los tres principales pilares (y quizá de las razones de ser) de LESS.
- Para entender cómo funciona un mixin vamos a ver un ejemplo real, que es como se aprende.

2.3. Mixins

- A mixin in LESS is a reusable block of CSS properties and styles that can be included in other selectors. It allows you to group a set of CSS declarations together and use them in multiple places without duplicating code.
- For example, when it comes to applying a border-radius, it can be quite cumbersome to have to write the prefixes for different browsers each time, right?

```
.border-radius(@radius) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius(4px); }
```

2.3. Mixins

```
.border-radius(@radius) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }
.sidebar { .border-radius(4px); }
```

- If you look closely, in our mixin, we included all the CSS3 prefixes that make the rounded border visible in webkit, mozilla, etc. Additionally, since we want to reuse this mixin, we assigned it the parameter @radius, which allows us to assign different values each time we call the mixin. The result of the above code, once compiled, is:

```
.sidebar {
-webkit-border-radius: 4px;
-moz-border-radius: 4px;
border-radius: 4px; }
```

2.3. Mixins

- Furthermore, to avoid getting overwhelmed with default values, we can make the parametric mixin have a default value.

```
.border-radius(@radius: 6px) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius; }
.sidebar2 { .border-radius(12px); }
```

- The idea is the same as the previous example, but now our .border-radius mixin has a default value (applied to .sidebar) that can be easily modified (applied to .sidebar2).

2.3. Mixins

```
.border-radius(@radius: 6px) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius; }
.sidebar2 { .border-radius(12px); }
```

- The compiled code would be as follows:

```
.sidebar { -webkit-border-radius:
6px; -moz-border-radius: 6px;
border-radius: 6px; }

.sidebar2 { -webkit-border-radius:
12px; -moz-border-radius: 12px;
border-radius: 12px; }
```

2.4. Examples

- In the case of CSS, let's consider the following scenario: we are styling a website, and we have a paragraph with its CSS rules. Additionally, for the standard paragraph tag, we can have several classes for an introduction paragraph and a highlighted paragraph. Let's say we define the standard paragraph with a sans-serif font, normal size, line-height, etc. Here's how our CSS would look (this is just an example, no LESS involved):

```
p {
color: #232323;
font-family: Helvetica,
Arial, sans-serif;
font-size: 14px;
line-height: 21px; }
```

2.4. Examples

- We make the font size a little larger and make all the text uppercase. Finally, for the highlighted paragraph, we'll make it bold and blue.

```
p {
color: #232323;
font-family: Helvetica, Arial, sans-serif;
font-size: 14px;
line-height: 21px; }

p .intro {
font-variant: small-caps;
font-size: 16px;
line-height: 24px; }

p .highlight {
color: #00214D;
font-weight: bold; }
```

2.4. Examples

- Now with less

```
// Variables
@textColor: #232323;
@textHighlight: #00214D;
@fontFamily: Helvetica, Arial, sans-serif;
@fontSize: 14px;
@lineHeight: 21px;
@introSize: 16px;
@introLineHeight: 24px;
@introFontVariant: small-caps;
// LESS for Paragraph

 p { color: @textColor;
font-family: @fontFamily;
font-size: @fontSize;
line-height: @lineHeight; }
```

2.4. Examples

- Now with less

```
// Variables //
…
// LESS for Paragraph // -------------------
p { color: @textColor;
font-family: @fontFamily;
font-size: @fontSize;
line-height: @lineHeight;

.intro { font-variant: @introFontVariant;
font-size: @introSize;
line-height: @introLineHeight; }
// End of .intro class
.highlight { color: @textHighlight;
font-weight: bold; }
// End of .highlight class    }
// End of paragraph rule
```

2.4. Examples

Despite the fact that the code with Less is longer, there are several advantages of using it:

2.5. Functions

- Less provides a variety of built-in functions that allow you to perform operations and manipulate values within your stylesheets. Here are some commonly used Less functions:
  - Color Functions: Less provides functions for working with colors, such as adjusting brightness, changing opacity, blending colors, and more. Examples include lighten(), darken(), fade(), saturate(), desaturate(), mix(), etc.
  - Math Functions: Less includes mathematical functions for performing calculations on numeric values, such as add(), subtract(), multiply(), divide(), percentage(), etc.
  - String Functions: Less offers functions for manipulating and formatting strings, such as quote(), replace(), length(), extract(), to-upper-case(), to-lower-case(), etc.

2.5. Functions

- List Functions: Less provides functions for working with lists, allowing you to manipulate and access individual elements of a list. Examples include length(), append(), join(), extract(), etc.
- URL Functions: Less offers functions for working with URLs, such as url-encode(), data-uri(), url().
- Miscellaneous Functions: Less includes additional functions for various purposes, such as unit(), round(), ceil(), floor(), e() (escaping strings), default() (providing fallback values), and more.

- These functions can be used within your Less stylesheets to create dynamic and flexible styles based on calculations, color transformations, string manipulations, and more. They enhance the capabilities of Less and help streamline the development process.

2.5. Functions

- LESS can identify the assigned unit and then add a number to that unit. In this example, LESS should add 4px (pixels) to the font size of the blockquote tag.

```
// Variables para el ejemplo //
@baseFontSize: 16px;
@baseFontFamily: Helvetica, sans-serif;
@quoteFontFamily: Georgia, serif;

// Damos estilos al párrafo y blockquote con
LESS //
p { font-family: @baseFontFamily;
font-size: @baseFontSize; }

blockquote { font-family: @quoteFontFamily;
font-size: @baseFontSize + 4; }
```

2.5. Functions

```
// Variables for the example//
@baseFontSize: 16px;
@baseFontFamily: Helvetica,
sans-serif; @quoteFontFamily:
Georgia, serif;

// We apply styles to the paragraph and
blockquote using LESS. //
p { font-family:
@baseFontFamily;
font-size: @baseFontSize; }

blockquote { font-family:
@quoteFontFamily; font-size:
@baseFontSize + 4; }
```

```
p { font-family:
Helvetica, sans-
serif; font-size:
16px; }

blockquote { font-
family: Georgia,
serif; font-size:
20px; }
```

2.5. Functions

- Color functions

```
@color: #888 - #222;
h2 { color: @color; }
```

```
h2 { color: #666666; }
```

```
lighten(@color, 10%); // Devuelve un color 10% más claro que @color
darken(@color, 10%); // Devuelve un color 10% más oscuro que @color
saturate(@color, 10%); // Devuelve un color 10% más saturado que @color
desaturate(@color, 10%); // Devuelve un color 10% menos saturado que @color
fadein(@color, 10%); // Devuelve un color 10% menos transparente que @color
fadeout(@color, 10%); // Devuelve un color 10% más transparente que @color
fade(@color, 50%); // Devuelve un color con 50% de transparencia
spin(@color, 10); // Devuelve un color con 10 grados más en el tono de
@color
spin(@color, -10); // Devuelve un color con 10 grados menos en el tono de
@color
mix(@color1, @color2); // Devuelve una mezcla entre los colores @color1 y
@color2
```

2.6. Inheritance

- In the case of LESS, you can also use inheritance, but at the time of compilation, instead of applying the same styles to multiple classes, it duplicates each individual element.

```
.bloque {
margin: 10px 5px;
padding: 2px;
}

p {
. bloque; /* Hereda estilos de
'.bloque' */
border: 1px solid #EEE;
}
ul, ol {
.bloque; /* Hereda estilos de
'.bloque' */
color: #333;
text-transform: uppercase;
}
```

```
.bloque {
margin: 10px 5px;
padding: 2px;
}
p {
margin: 10px 5px;
padding: 2px;
border: 1px solid #EEE;
}
ul,
ol {
margin: 10px 5px;
padding: 2px;
color: #333;
text-transform: uppercase;
}
```

2.7. Advantages of using less

- Despite the fact that the code with Less is longer, there are several advantages of using it:
  - **Variables**: Less allows you to define variables, making it easier to manage and reuse values throughout your stylesheets. This promotes consistency and makes it simple to update values in one place.
  - **Mixins**: With Less, you can create mixins to group a set of CSS properties and reuse them in multiple selectors. This reduces code repetition and promotes code modularity and maintainability.
  - **Nesting**: Less supports nesting of selectors, allowing you to write more structured and readable code. You can nest child selectors within parent selectors, avoiding the need to repeat the parent selector multiple times.

2.7. Advantages of using less

- **Functions and Operations**: Less provides functions and arithmetic operations that can be used to perform calculations and manipulate values within your stylesheets. This adds flexibility and dynamic capabilities to your styles.
- **Importing and Modularity**: Less allows you to import external files into your main stylesheet, making it easy to organize and modularize your code. You can split your styles into smaller files and import them as needed, improving organization and maintainability.
- **Built-in Functions and Mixins**: Less comes with a set of built-in functions and mixins that provide additional functionality. These can be used to accomplish tasks such as generating gradients, handling vendor prefixes, working with colors, and more.
- **Easy Integration**: Less can be easily integrated into your existing CSS workflow and is supported by many popular text editors and build tools. It can be compiled into standard CSS, which can then be used by web browsers.

Overall, the benefits of using Less include improved code organization, code reuse, faster development, easier maintenance, and more powerful styling capabilities.

3. Saas

- Sass (Syntactically Awesome StyleSheets) is a preprocessor scripting language that is used to enhance the capabilities of CSS (Cascading Style Sheets). It extends the functionality of CSS by introducing features like variables, nesting, mixins, inheritance, and more. Sass files are compiled into regular CSS files that can be interpreted by web browsers.
- Here are some key features of Sass:
  - **Variables**: Sass allows you to define variables to store and reuse values throughout your stylesheets, making it easier to manage and update styles.
  - **Nesting**: You can nest CSS selectors within one another, providing a more structured and organized way to write your stylesheets.
  - **Mixins**: Sass supports mixins, which are reusable blocks of styles that can be included in multiple selectors, reducing code duplication.

3. Saas

- **Inheritance:** You can use inheritance to define a base style and extend it to other selectors, creating a more modular and efficient codebase.
- **Functions:** Sass provides a set of built-in functions and allows you to create your own functions, enabling you to perform calculations, manipulate values, and create dynamic styles.
- **Importing:** You can import other Sass files into your main stylesheet, allowing for better organization and modularization of your code.

Sass offers a more powerful and flexible way of writing CSS, making it easier to maintain and scale stylesheets. It provides a range of features that improve code reusability, maintainability, and readability.

3. Saas

Indicate the main differences between Less and Sass regarding:

Syntax
Variables
Nesting code
Mixins
Import
Functions
Enheritance

SassGuidelines: https://sass-guidelin.es/ and common sense to adapt those rules to the project.
And of course, the official website: https://sass-lang.com/documentation