**1. Introduction**

- Hello, my name is Victor, I’m an Application Developer for Atos and I’m going to show you SASS and how to work with it to make some CSS style.

- We will start with a SASS overview, what is it and why you should use it.

- Then we are going through the SASS features like nesting, inheritance, variables, modules, mixins, operators and functions.

**1.1. What is SASS? | CSS preprocessors**

- So, what is SASS?

- SASS is described as CSS with superpowers, SASS is a CSS preprocessor, it allows us to write CSS code in an easier way which is later compiled into the usual CSS we know.

**1.2. Indentation syntax**

- First it comes with an indentation syntax, it gets rid of the braces and semicolons making you able to write more fluid.

- SASS also comes with some features that, against CSS, makes possible the code reuse, something I’m sure you are going to find very helpful.

- Let’s get into it.

**3. Requirements**

- Ok, from this point I’m going through some examples using Visual Studio Code.

- If you want you can go to code.visualstudio.com, click in download and follow the installer’s instructions, or you can use any code editor you like.

- Previously I mention that SASS code gets compiled to normal CSS, so, we are going to use two extensions.

- Go to extensions, look for Sass, so visual studio code can recognize the syntax, click on install, look for Live Sass Compiler and again click on install, later you will see that using Live Sass Compiler we can get our SASS compiled each time we save a file.

- Once you have installed the extensions and restarted your VS Code you are ready to go.

**4. Nesting**

- So, the first feature I’m talking about is nesting.

- Here I have my index.html, let’s just create a folder called ‘css’, and then a file called ‘style.css’.

- When you are defining a style you write selectors one after another, and if you want to apply some CSS to an element which lays inside another element, you write the parent element, space, the child element, you open braces and define some attributes.

- Well, with SASS, you can nest your selectors.

- Let’s rename our style file to ‘style.sass’, now we can ignore those braces and semicolons.

*note*

- So, here I have the body and section selectors, you can tell both are at the same level, now I want to define the properties of the square and the p tag inside.

- In SASS, we can write the parent selector, go to the next line, define some properties, then we can add the child selector, jump to next line, and define its properties as you can see.

- Now our code is more readable, nesting gives us some structure so we can easily notice which selectors are applied to each element.

- But there is one thing, nesting is cool, but if it goes too deep, the code can become difficult to read, something we do not want, so let’s just keep in mind that too much nesting probably isn’t the best idea.

- Now that we have a SASS file, Live SASS Compiler can compile the code automatically, if you go to the bottom of your VS Code, just click on ‘Watch SASS’, it shows a terminal, you can close it if you want to, and you are going to see in explorer that there is a new CSS file, it is the compiled version, make sure you are referencing this CSS file in your index and not the SASS one, because web browsers cannot read SASS yet.

**5. Inheritance**

- Ok, let’s talk about inheritance.

- If you are familiar with object-oriented programming, which is not the subject, you probably know inheritance.

- What is inheritance? basically, it means that a child element inherits all the parent’s properties, and also has properties of its own, inheritance allows you to extend functionality, or in this case, extend CSS properties.

- Let’s see a simple example, here in index.html, I’m going to change the second’s div class to ‘square-border’.

- Now in the SASS file I’m adding that class, so, what I want here is to have a square with the same properties of the others and a white border, but I don’t want to repeat what I already have neither to combine selectors, I want square-border to extend from square.

- To extend we must write @extend, space, square, and then just define the border property.

- There we go, the second square now has a white border.

- Save changes and look in the CSS file to watch the difference.

**6. Variables**

- So, the next feature is variables, SASS has a way to set CSS properties as variables.

- To define a variable, you must write the name of the variable starting with a ‘$’ sign, and then assign its value with a colon.

- Let’s set the colors as variables.

- The first color we have here is the body background, let me go to the top, and let’s say dollar, body-color, white, we also have the square color, so, square-color, and finally the border color.

- Let’s just replace the values with the variables, and that’s it, we have our color variables.

- Ok, you may be thinking, ‘why would I use SASS variables if CSS already has variables?’, well yes, CSS 3 came with a way to define variables, but let me show you the difference.

- Here we have the usual CSS code which has some variables, this is the file that will be included in the page, and now, this is the compiled CSS version of the SASS code, which does not have any variable definition, why? Because in SASS only the values are compiled to CSS code, the values you define in SASS variables will simply pass to the CSS elements, this means that you can separate the values and end up with less lines of CSS code.

**7. Modules**

- We just separated a small piece of our code using variables, but we can also separate a bigger piece or section using modules.

- On CSS you can include CSS files inside other CSS files with the ‘import’ rule, well, you can apply the same principle in SASS using something called ‘partials’.

- A partial is an additional file you can import in your main file, by convention, a partial name should start with an underscore, so, let’s get our variables out of here.

- Create a new file called ‘colors’, underscore + colors.

- Then cut the variables, paste them inside the ‘colors’ file, and now we can type ‘@import’ and just write ‘colors’ with no underscore or extension. This is how we can separate CSS code in files and include them in the main file, you can see that our page is displaying in the same way.

- Right now, we are doing this with just a couple of variables, but sometimes the CSS code can be very big making it hard to read, imagine if we had a hundred of CSS lines with color related properties, with this, we could separate related CSS content into files making it easier to handle, which is something we all should do as a good practice.

**8. Mixins**

- Now, there is another SASS feature we can use for segmentation and code reuse, mixins.

- Sometimes while creating some CSS code, you notice that you are in fact repeating some CSS properties between several selectors, and here is where mixins come to the picture. With SASS mixins you can just say goodbye to repeating properties and start giving an even better structure to your CSS.

- A mixin is basically a set of properties you want to include in several selectors, let’s see this in code.

- For this example, I’m going to create another partial called ‘mixins’, underscore mixins.

- In order to define a mixin, you need to type @ mixin name, let’s say ‘square-border’.

- You can see I added another 2 classes, ‘square-transparent’ and ‘square-reverse’, this is how the page looks now.

- So, inside ‘square-border’ mixin I want to set some properties, a border, a transition for an animation, and I want to increment the border width on hover.

- First let’s import our mixins partial, @include mixins.

- And let’s include our mixin in ‘square-transparent’ and ‘square-reverse’.

- Now both squares are showing with a bluish border and a smooth animation. As you can see, with mixins we do not need to repeat the same code, we just have to include the code.

- But what if now I wanted to increment the border width on hover by 5 pixels for ‘square-transparent’ but 10 pixels for ‘square-reverse’? well, mixins support parameters.

- Let’s open parenthesis after the mixin’s name, and add a parameter, $ sign border-width, we can pass the parameters when we include our mixin, 5px, and 10px, everything behaves the same.

- We can also set default values for parameters, let’s go back to ‘square-border’, and I’m going to set ‘border-width’ default value as 5px, so now we can actually just remove this parenthesis, and it stills behaving in the exact same way.

**9. Operators**

**10. Functions**