SCSS

MAIN GOAL

SASS is meant to give us a different way to write our stylesheets. The SCSS we write will be converted into CSS by the SCSS **pre-processor.**

So at the end of the day there is still a normal CSS file produced by the pre-processor, SCSS just allows us to write our style rules with a different **syntax.**

You can almost think of the SCSS pre-processor as a translator. It will take valid SCSS code and turn it into valid CSS for you. The SCSS language has a much more feature rich syntax that allows for a more compact and cleaner looking style sheet.

BENEFITS OF PREPROCESSORS

When you write larger and larger websites you will start to see that the CSS sheets get very large and repetitive. You will have rules that have the exact same declaration block but a different selector.

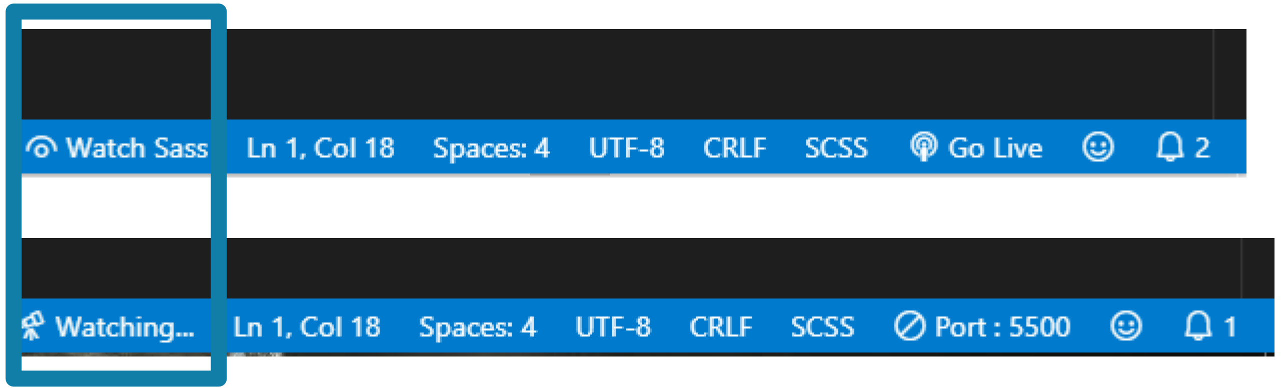
Preprocessing allows us to no longer repeat ourselves in our CSS. We can write a rule or property in one place and use it multiple times.

LIVE SASS COMPILER

This is an extension that we can install in Visual Studio Code, and it will handle all of the busy work for us!

All you have to do is download the "**Live SASS Compiler**" extension from the VS Code marketplace. Make sure the author of the extension is **Glen Marks.** Be careful, there are two and the wrong one is more popular (it is no longer maintained).

Once you have installed the extension, you might have to restart your editor.



WHAT DID WE JUST DO

We have just started up a local SASS preprocessor! The SASS preprocessor will now watch for any saves in a SCSS file and automatically generate our CSS.

Now, every time we save our HTML, CSS, or SCSS our live server will show the changes right away!

A word of caution, with a SASS preprocessor **please do not work in CSS**. Your SASS preprocessor will simply overwrite anything in that file and any changes will be gone. Make sure you write in your SCSS files.

KNOWLEDGE CHECK

Let's see if we can get our SASS pre-processor working for us:

1. Install the **Live SASS Compiler** extension onto VsCode
2. Create a new folder called ScssIntro in your Scratch folder
3. Connect this folder to Git/GitHub
4. Create an index.html and a **style.scss** file
    1. Remember, you still link index.html to **style.css** as this file will exist soon!
5. Turn on the Live SASS Compiler
6. Add in at least 3 content tags to your page
7. In style.scss, write some simple CSS
8. Verify that the style.css file is created when you save the file
9. Add, commit and push your code.

NESTING

The biggest difference you will see when writing scss is you can now nest rules inside of other rules to mimic your HTML code much more closely.

For example, if you had the following HTML:

```
<div id="card_container">
    <h1>This is a card</h1>
</div>
```

You can now write a rule like:

```
#card_container {
    display: grid;
    >h1 {
         font-weight: bold;
     }
}
```

NESTING CONT.

You can also add in pseudo selectors into these nested rules

```
#card_container {
    display: grid;
    >h1 {
         font-weight: bold;
     }
    &:hover {
         box-shadow: 5px 5px 7px grey;
    }
}
```

KNOWLEDGE CHECK

Let's see if we can get our SASS pre-processor working for us:

1. Ensure your index.html has at least 1 level of nesting (for example, a p tag inside of an article)
2. Use nesting to style the containing tag and the content tag
3. Use a pseudo selector to style the container when it is **hovered**
4. Add, commit and push your code

VARIABLES

One of the most useful features of SASS is the allowance of variables. Now, CSS does support variables but this is relatively new and not widely supported.

Variables allow us to define **key-value pairs** that we can reference anywhere in our SCSS file. This is actually a fundamental principle of programming languages, so it is good to be exposed to them now.

Let's look at creating and referencing a variable:

```
$main-font-color: purple;

h1 {
    color: $main-font-color
}
```

KNOWLEDGE CHECK

Let’s see if we can get our SASS pre-processor working for us:

1. Create a variable that defines a font color of your choosing
2. Use this variable to change the font color of at least 3 different text based tags
3. Change the value of the variable to a different color, and prove to yourself that it changes all rules that used the variable.
4. Add, commit and push your code

PARTIALS

One of the coolest features of SCSS is the ability to share common code among multiple different SCSS files.

For example, let's imagine that you have written a full SCSS file for your home page and a second SCSS file for your about page. Now imagine that both of these pages have the same code for both the header and footer (it is very common for all pages on a site to have the same header/footer on all pages).

With normal CSS, you might end up copy-pasting the style for the header and footer in both CSS files you made. With SCSS, we can make a new file for the header/footer SCSS and **use** that SCSS in the files for our home page and about page.

PARTIALS CONT.

While this might seem a little bit messy at first, partials actually allow you to organize your code in a much more maintainable way.

A typical setup I like to start with is as follows for my SCSS files:

1. _header.scss
   1. This is where I style my header with normal scss rules
2. _footer.scss
   1. This is where I style my footer with normal scss rules
3. _general.scss
   1. This is where I add in anything that is used on all pages like animations, variables, scss that is common (like the *, body rules), etc.
4. page_name.scss
   1. For every page my site has, I make an individual scss file. It is in these files I **use** the other partial files.

Notice that _header, _footer and _general all started with an _ character. This tells SCSS that these files are designed to be used in other files and are not full complete scss documents themselves.

PARTIALS CONT.

Once you have a good setup, you can **use** partials with the following syntax in any SCSS file:

```
@use “file_name";
```

For example, at the top of style.scss:

```
@use "general";
@use "header";
@use "footer";
```

Notice that we did not need:

- The .scss at the end of the filename
- The starting _ before general, header and footer

SCSS is smart enough to figure this stuff out for us.

KNOWLEDGE CHECK

Let's see if we can get our SASS pre-processor working for us:

1. Create a file called _general.scss
2. Add in rules for *, body and img tags (the standard starting rules we usually put into our sites) into the _general.scss file
3. In style.scss, remove the *, body and img tag rules if you have them
4. Add in the **use** line in style.scss that will use the _general.scss file
5. Verify it works by opening the style.css file and seeing the rules still showing up after being compiled.
6. Add, commit and push your code

SASS

There is lots to discover when it comes to the extra power provided by SASS!

Take your time with the things covered in this deck as they will take some time to get used to. As we explore more with SASS, we will see that it is actually a very good introduction to programming as many of the concepts carry over.