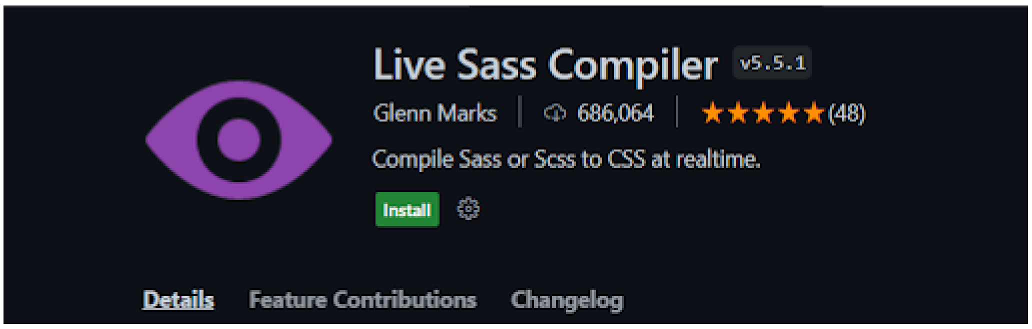
SAAS

SASS (Syntactically Awesome Style Sheets) is a **CSS preprocessor** that extends the capabilities of CSS by adding features such as variables, nested rules, and mixins. It is a scripting language that is then compiled into CSS. This allows for more maintainable and organized CSS code, as well as the ability to use programming concepts in styling a website. SASS was first released in 2006 and has since become one of the most popular CSS preprocessors, along with its main alternative, Less. One of the main advantages of using SASS is that it can make your CSS more modular, reusable, and easy to maintain. Additionally, SASS supports mathematical operations and functions, making it easier to handle responsive design and perform calculations in the styling.

In order to write and run SASS code, we need to install a VSCode extension called Live SASS compiler. This is because the browser only reads CSS code and cannot read CSS code hence, we need to install this extension because it is responsible for transforming SASS code into CSS code. In order to write SASS code the extension of our file needs to be .scss.



Live Sass Compiler v5.5.1

Glenn Marks | 686,064 | ★★★★★ (48)

Compile Sass or Scss to CSS at realtime.

SASS Variables

One of the most common problems in CSS is redundancy to choose colors for all elements, and changing a specific color will lead to changing this color multiple times for other elements, that is why SASS variables were made, where you could define a specific color for example and us this variable for different elements, rather than choosing the color over and over again. In the following example, whenever I want to change the paragraphs text color I can change only the variable $textColor, rather than changing the color in the p tag.

SASS Variables

```
$primaryColor: #58e3e3;
$textColor:black;
#btn1{
    background: $primaryColor;
}
#btn2{
    background: $primaryColor;
}
p{
    color: $textColor;
}
```

Nesting in SASS

The second use for SASS is **nesting**, and this is done by writing CSS-like code but nested inside each other, just like HTML. For example:

Nesting in SASS

```
$hoverColor:rgb(48, 202, 219);
.header{
    // CSS for elements in the header class
    display: flex;
    justify-content: center;
    align-items: center;
    // CSS for buttons inside the header class
    button{
        background-color: $primaryColor;
        text-align: center;
        &:hover{
            background:$hoverColor;
        }
    }
}
```

Separate Code

Another powerful use for SASS is **separating code**, where you don't have to include all your CSS code in one file, you could have separate files for each component or class, and you could import it into other .scss stylesheets. For example we want to write our code for the header class in a separate file, so we call the file _header.scss, and then importing it by writing @import 'header'.

_header.scss file

```
$hoverColor:rgb(48, 202, 219);
.header{
    // CSS for elements in the header class
    display: flex;
    justify-content: center;
    align-items: center;
    button{
        background-color: $primaryColor;
        text-align: center;
        &:hover{
            background:$hoverColor;
        }
    }
}
```

_variables.scss

```
$primaryColor: #58e3e3;

$textColor:black;
```

styles.scss

```
@import 'header';

@import 'variables';
```

Mixins

Mixins are the functions for SASS in other words and are reusable and called anywhere in the code, for example we can make a mixin that centralizes a flexbox, the mixin is going to look something like this:

Mixin code

```
@mixin flexCenter(){

    display: flex;

    justify-content: center;

    align-items: center;

}

// Including the mixin in code

.container{

    @include flexCenter();

}
```

In this example, the only difference is that we included an argument in the mixin, which is the $direction, to allow the user to specify the flex-direction whenever he is using the mixin to make things more flexible.

Mixin code with arguments

```
@mixin flexCenter($direction){

    display: flex;

    justify-content: center;
    align-items: center;
    flex-direction: $direction;
}

// Including the mixin in code
.container{
    @include flexCenter(column);
}
```

To develop more your Idea about SAAS, have a look at this video.

‹ Previous next ›