**SASS & BEM Tutorial**

**Title**: Sass and BEM for beginners

**Author Video**: Coder Coder

**Github Link to files**: [Link](https://github.com/thecodercoder) (no uploaded files)

**Duration**: One Youtube Video (3:45:09) **Dated**: 14 Jan 2023

**Link**: [Sass and BEM for Beginners](https://www.youtube.com/watch?v=jfMHA8SqUL4&t=2713s) **Course** **Files**: There are none

**Sites used**: **\***[**https://developer.mozilla.org/en-US/docs/Web/CSS**](https://developer.mozilla.org/en-US/docs/Web/CSS) **CSS Cascading Style Sheets**

[**https://httpbin.org**](https://httpbin.org) **website to POST HTML submissions and ‘GET’ submissions**

[**https://jigsaw.w3.org/css-validator/**](https://jigsaw.w3.org/css-validator/) **HTML validator service which checks HTML content validity**

[**https://emojipedia.org**](https://emojipedia.org)**(to use Windows emojis – at blinking cursor use ‘Windows’ key + ‘.’ dot)**

[**Specificity Calculator**](https://www.youtube.com/watch?v=B0uYbAe4YY8) **Good demo on specificity with 4 ranges (Inline, id, class, type)**

**\***[**http://fonts.google.com/**](http://fonts.google.com/) **For various font-family fonts**

**\***[**https://codepen.io**](https://codepen.io) **Codepen for researching other options (use my Github acct credentials)**

**\***[**https://sass-lang.com/**](https://sass-lang.com/documentation/modules/math/) **Sass:math homepage (see Part 12)**

[**https://caniuse.com**](https://caniuse.com) **Webpage to confirm how to implement background-clip**

**\***[**https://www.fontsquirrel.com/**](https://www.fontsquirrel.com/blog/2009/10/welcome-to-font-squirrel) **Creates useable fonts downloaded from Google Fonts**

**Additional Sites:**

1. Good description of flex-basis: [link](https://www.youtube.com/watch?v=jx4FtPlDXJg) by Kevin Powell (excellent orator)

**Software**:

**IDE**: VS Code, **Live Sass Compiler** (v6.1.2, author: Glenn Marks) – Extension in VS Code

To customize “Live Sass Extension” which is now installed:

Command palette: **Ctrl + Sh + p** in search window: “open settings” then select “Preferences: Open Settings (**JSON**)”

Scroll to the bottom of the file, locate “*liveSassCompile.settings.formats*”: [ { and add/edit as indicated below:

* **“format”: “expanded”,** applies compiled code in standard css layout. “**compressed**” applies minified code into a single line of code (saves memory space)
* **“savePath”: “/dist”,** is included. This will save compiled style.css file into this default directory
* **“savePathSegmentKeys”: null,** was not included (could be a mistake)

**Analysis Software**: Adobe XD (not available but very useful). Measures box height, margins, spacing, etc

# Part 1 – What is Sass (00:22 – 01:17)

Sass stands for Syntactically (or Semantically) Awesome Style Sheets and is a CSS preprocessor. In short, this just takes code and processes it into CSS that browsers understand. It makes writing code easier and is organized in a smarter way instead of rifling through tons of lines of code to find the variable or specific element or class selector.

You can either use the original **Sass** with extension **.sass** and doesn’t use curly braces nor semicolons and uses indentation and new lines to differentiate new style rules. Or you can use **SCSS** with extension **.scss** which looks similar to pure CSS layouts but makes use of many features to help the developer achieve the final state quicker. It also allows you to copy and paste into a CSS file without any issues.

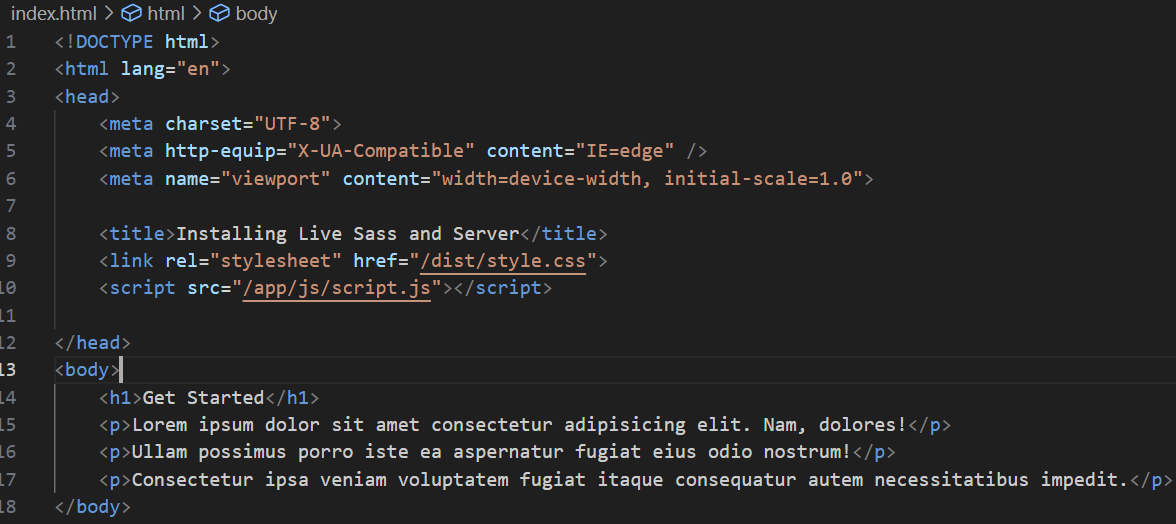
# Part 2 – Compiling Sass (01:17 – 12:49)

Need to install:

* **Live Sass Compiler**: in VS Code (VSC), use Extensions icon, search for **live sass compiler** (Compiled by: Glenn Marks)
  + Already installed, but for posterity
    - Remarks above under **Software**, will do the trick, however …
    - To start the inclusion of the customized properties: on a new line, include
    - “livesassCompile.settings.formats” click this option
    - in the line ‘savePath”: “/dist”, can name this anything you wish. This is the folder where the final compiled style.css file will be placed
* **Live Server**: in VSC, use Extensions icon and search for **live server** (developed by Ritwick Dey)
  + Already installed, but for posterity
    - Just install it – may need to restart VSC or use **Ctrl + Sh + p** and type “*reload*”, then select the “**Developer: Reload Window**”
* Once reloaded, you should see at the bottom of VSC window this lil gem:

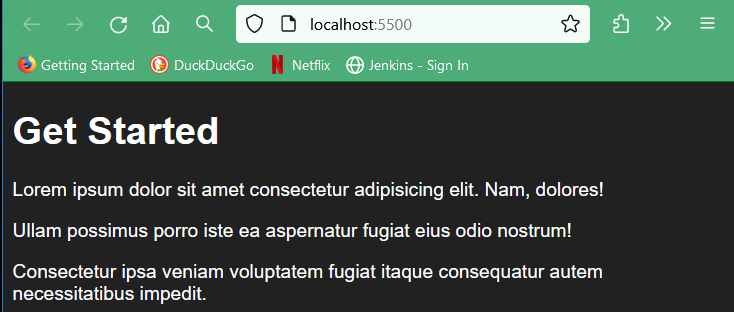


With the ‘starter code’ below:



‘**Click’** the “**Watch Sass**” which will then change to “**Watching …**”

* Notice in VSC Explorer window the ‘dist’ folder is created and 2 files created:
  + **style.css** // this will be the final css file used by the html
  + **style.css.map** // maps where the files are being extracted from
* make changes to the style.css and notice when the file is ‘saved’, “**Watching …**” will show ‘**Success’** if successful.

‘**Click’** the “**Go Live**” (if the active page is the index.html) or right-click the index.html and select “**Open with Live Server**”

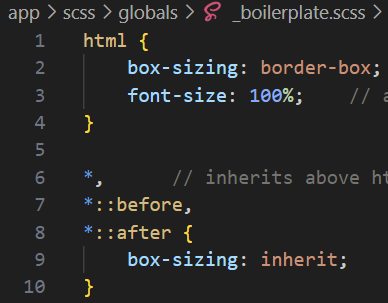
* Browser will open displaying our page

# Part 3 – Sass Partials (12:49 – 26:17)

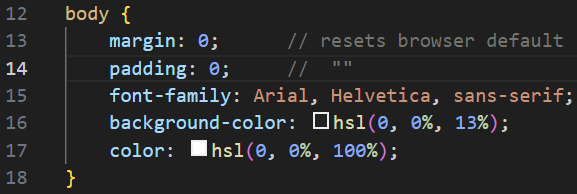
You can split up files, instead of having one huge file, into multiple files and even keep these organized using sub-folders. This allows different parts of the website separated into different files to make it easier to locate especially true when it comes to debugging the website.

Also, having the styles separated into these different files will allow multiple developers working on the code at the same time with a lower likelihood of developers working on the same file at the same time.

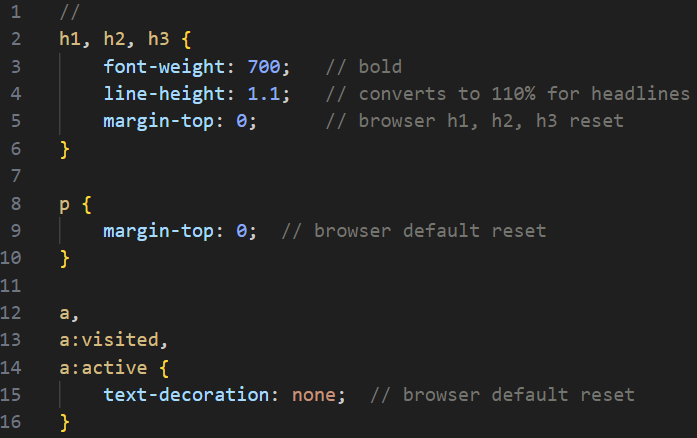
The first partial we’ll create is for some boilerplate styles. These are some general styles that you can use on pretty much every website. These boilerplate styles can be considered as ‘global’ styles. To make this simpler, we can create a ‘global’ folder that will hold the ‘global’ scss file.

In the ‘**app/scss’** folder create a **new** folder called:

* **globals**
* inside this folder, create a **Scss partial** file (**Note**: the **underscore** preceding the name. This is how the compile acknowledges this file as a partial thereby using it to create the style.css file named:
  + **’\_boilerplate.scss’**
* Add this starter code into **’\_boilerplate.scss’** file
* Cut/paste **all** code from **’style.scss’** into bottom of **‘\_boilerplate.scss’**

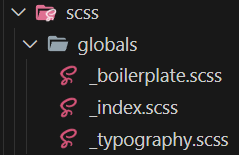


**Done!**

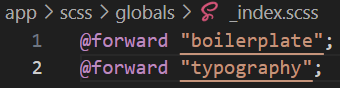


Create another scss partial file, named:

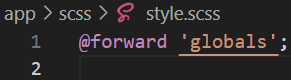
* **‘\_typography.scss’**
* Insert this code

You may be wondering how these ‘**partial’** files are loaded into our ‘**style.scss’** file? I was wondering about this myself! This is done by creating another ‘**partial’** named **‘\_index.scss’** which is what the Sass compiler looks for to consolidate all the code into one ‘main’ file.

**Note:** insert another **‘\_index.scss’** file into each sub-folder to keep code organized and allow the Sass compiler migrate through the maze of folders to find the particular partial.

In the **‘\_index.scss’** file, we can use the **@forward** (**@** is the rule, **forward** is feature) to let the compiler know there’s additional code in the named **‘.scss’** file.

* No need to include the ‘**underscore’** nor the **file name extension** (compiler understands it’s a Sass file – so no need to include)

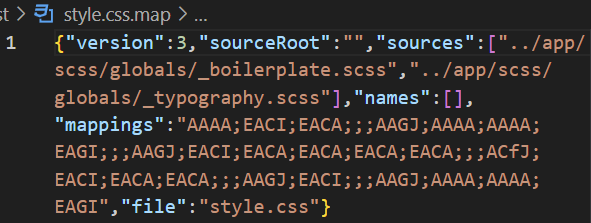
The **@forward** rule tells the compiler to forward the identified style to get loaded. The **‘\_index.scss**’ file is taking these styles from the ‘**globals**’ folder and forwarding them to our main Sass file which is the ‘**style.scss**’ file in the /scss **root** folder. In our main ‘**style.scss**’ file, identify which folders need to be forward (this is like hierarchy):

* The compiler will look inside the ‘**globals’** folder and locate the file named ‘**style.scss**’ which is the main file.

**With all the files created/saved** …

* check the bottom of VSC’s Terminal window to confirm no ‘**error’** messages
* check the layout in the webpage!
* **SUCCESS! ✓**

**Figure 1:** style.css.map

Reviewing the ‘**style.css.map’** you can see all the **scss** rules that were forwarded to the main css file ‘**style.css’**. This map is beneficial in sourcing where the css styles originated from.

# Part 4 – Sass Variables and CSS Custom Properties (26:20 – 43:45)

Variables are ways to store bits of information and then give them a label with a name. The name identifying the variable is typically descriptive enough to understand what it’s used for. The term ‘variable’ is used since you can call one ‘variable’ some descriptive name and call another variable something totally different. On a similar note, when you call a ‘variable’ something, the data in this variable may change as it’s travelling through the code being assigned different value contained in that variable – hence it’s ‘varied’ or better yet ‘variable’. Ouch, that was a play on words!

Now Sass and css both have variables. In CSS they are called ‘custom properties’, in Sass they are called ‘variables’. Some people just refer to both as ‘variables’. They are used for a number of reasons:

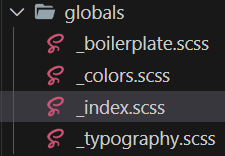
* Color properties (red, fuscia, rebeccapurple, etc)
* Font size or styles (large could be 90px and small 12px, etc)

When implementing variables, it makes changing the code that much easier – in one place, one name changes the entire layout.

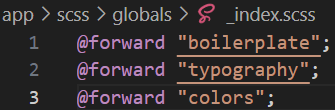
For example: in our **boilerplate.scss** file, we have:

* **background-color:** *hsl(0, 0%, 11%);*
* **color:** *hsl(0, 0%, 100%);*

If we don’t use variables, we would need to copy/paste each line above into the code where we want this to show up. That’s neither convenient nor productive as the code could be copied incorrectly or you may miss a location that needs the change when updating the page in the future.

We can move these values into a location that easily identified/located for consistency and ease of altering. In the:

* **‘\_globals’** folder, create a new partial file named **‘\_colors.scss’**
* As soon as the file is created – head over to the **‘\_index.scss**’ to include this new file with the **@forward** rule



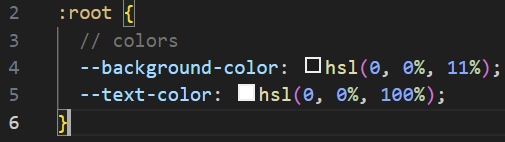
Now we can create some **CSS Custom Properties** (variables) to make coding easier:

As a general rule, you want to apply these custom properties to the **root** directory which will make them ‘**global’**:

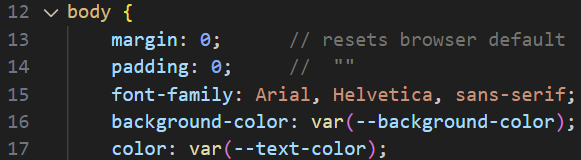
* **:root {** //insert rules here **}**

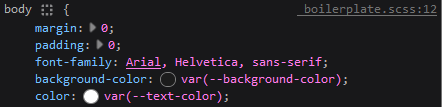
This can also be applied to the **html** or **body** selectors if that meets the use-case scenario.

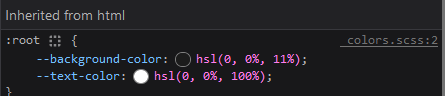
**Noteworthy**, is that any rule applied within the **:root** selector will override those same named properties in any other selector due to ‘**specificity’**.

Since the **‘\_colors’** variable will be **global**, lets set them up in **‘:root’** directory using the **CSS Custom Properties** **syntax** which does use ‘**double-hyphens’** prefixed to the *variable name*.

* example: **--background-color**:

We can now replace the color properties we included in the **body** selector which was in the **‘\_boilerplate.scss’**

If you look at the “**Inspector’** in the ‘**dev tools**’ it will show the **body** selector is picking up the variable colors we assigned:

As well, the **:root** selector is picking up our defined **global** variables using the ‘**--**’ prefix for all global variables.

The browser will ultimately change any ‘**hsl’** values to ‘**hex’**. ‘**hsl**’ values are just simpler to work with and modify the color.

Moving ahead! **Sass** also uses **variables** and are similar to (in some ways) but different from “**CSS Custom Properties”**. With **Sass** variables, the *compiler* will convert any **Sass** variables into their **final CSS values**.

**Pros to CSS Custom Properties:**

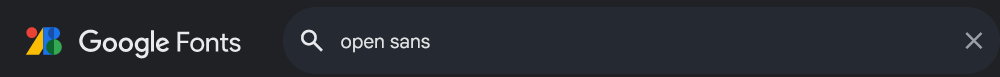
* **CSS Custom Properties** are newer and being native to CSS, you can use these variables without using **Sass**.
* **Big Benefit:** **CSS Custom Properties** allow javascript to change the values of the **Custom Properties** once the website is loaded, basically on the fly – **Sass** variables can’t do this.

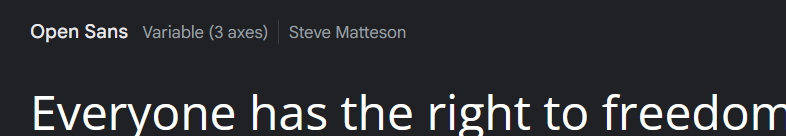
**Sass is an older technology:**

* Although a shift has occurred where more developers are using CSS Custom Properties, legacy frameworks out there in the wild still use Sass – it’s advantageous to know how these work for future considerations.

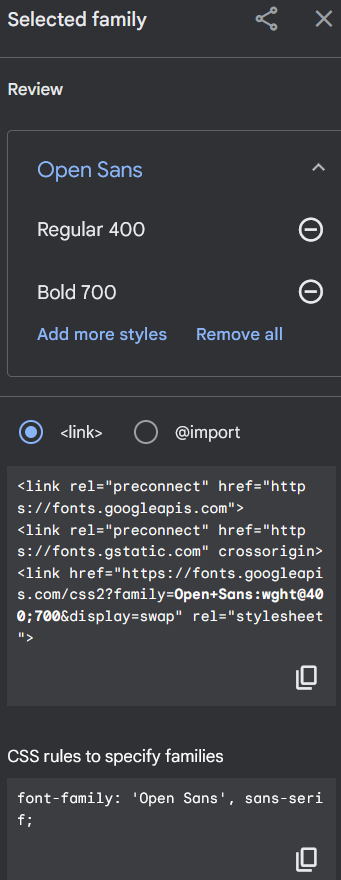
We have stored our ‘**color’** using **CSS Custom Properties**. We will go ahead and use **Sass variables** to store the **font** information (**Note**: we could use CSS Custom Properties but for this exercise it’s Sass variables).

**1st Step**: head over to [Google Fonts](http://fonts.google.com/)

* Looking for ‘open sans’ in the search bar



* Select this:

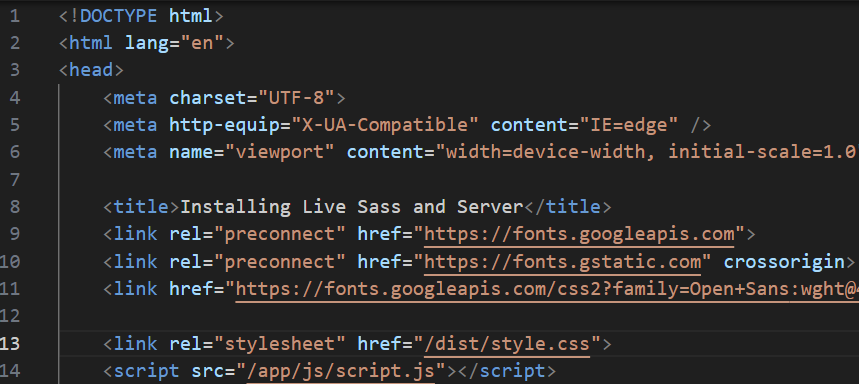


* Now need to select the font-weights we need:
  + Regular 400 (select it)
  + Bold 700 (select it)

They are both selected and presented in the ‘**Review’**

With the fonts we need added to the review, we need to get the code to *copy* and get this into our **<head>** tag of our **html** using the **<link>** tag. Place this above our own ‘**styles’** to prevent any issues or overwrites of code. Go ahead and ‘click’ this

* Paste in <head> tag above our style.css (to reduce any unwanted behavior)



* It does have 2 link tags to google.fonts and then loads this CSS file from ‘googleapis.com’

**Particular Note:** google.fonts has been found by court in the European Union to have violated the GDPR regulations because they can track the ip address of users of the website when they load your website if you’re using google fonts which is the way we have linked the files here.

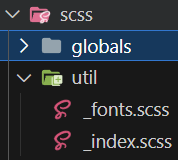
**Workaround:** download the needed font files from google fonts and then host the website server instead of loading from google servers.

* Go to end of document under “**Google Fonts Setup**” if this is what you wish to do.

## Sass Modules

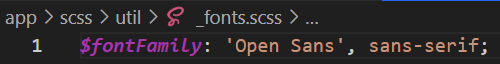
Since we are using the fonts with Sass variables, we also want to be able to load that font value into other Sass files so we need to use something called **Sass Modules** to make the **Sass variables accessible** from any of the Sass files. (**Me: similar to other languages, you can use modules in Sass)**

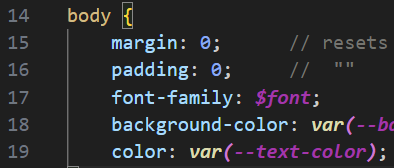
**\*\*** Instead of adding another **Sass** **partial** file in the ‘globals’ directory, we’ll need to create a sub-folder in our ‘scss’ folder:

* **util**  // in this folder, create **2 other partial files**
  + **\_fonts.scss** // will hold all the fonts
  + **\_index.scss** // as mentioned earlier, every folder **MUST** have an

‘**\_index.scss**’ which is what the SCSS compiler will be

looking for.

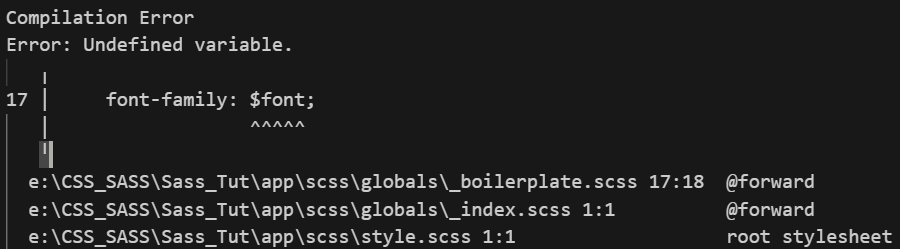
* In the Sass partial file, **‘\_index.scss’**:
  + **@forward “fonts”;** // compiler knows to pick up this file
* In the **‘\_fonts.scss’**, create a Sass variable to store the **font-family** from google.fonts:
  + For all Sass variables, need to include the **$** to denote this as a **Sass variable**
  + **$fontFamily**: need to go to **google.fonts** and pick up the remaining code (see previous page – green box)
    - Only need the **‘Open Sans’, sans-serif;** text for this

Now we can replace the **font-family** in our **body** selector in the **‘\_boilerplate.scss**’ partial file:

* When we did this – got an ‘**error**’:

Displayed by Sass **“Watching…”**

And in the VSC ‘**Output’** window pane, this was displayed:



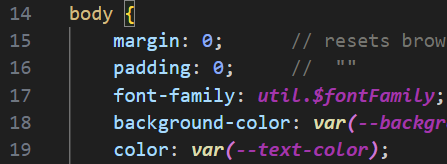
**It states:** Undefined variable on line 17 for the **$font;**

**Problem**: What the Sass compiler is telling us is that this **$font** has not been defined!

**Reason**: this font is buried in a new folder, **util,** in a file called **‘\_fonts’**. Compiler doesn’t know how to get there since the routing has not been defined.

**Solution**: need to load the **Sass** **partial** from our ‘**util’** directory as a **Sass** **Module** in our **‘\_boilerplate.scss’** file. This is similar to loading other modules in other languages to provide the code in that file, the location of where to find the missing links.

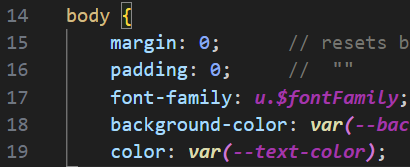
The way to incorporate this is:

* At the top of the **‘\_boilerplate.scss**’ file, **add**:
  + This is relative from the **‘\_boilerplate.scss**’ file’s location. Hence, go up 2 levels, into the ‘util’ folder.
  + The **‘@use’** tells the **‘\_boilerplate.scss’** file to use this folder to find the ‘**fontFamily’**, which then looks for the ‘\_index.scss’ to find the ‘\_fonts.scss’ file.
* ****Now that we know where the file is located, we need to use the ‘**namespace’** whenever we are using a **Sass Module**. In other words, the Sass compiler will give it a ‘**namespace’** that matches the ‘**name’**:



Now when we **Save it** … **Success**!

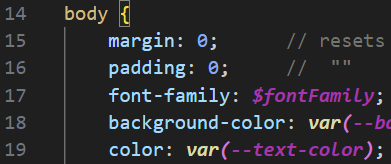
With this in mind, you can now load different font variable names as long as the folder or file name are different than other file names to reduce ‘namespace’ or any other conflicts in naming conventions.

Like other languages, if you don’t want to type out the ‘**util’** every time you need to reference the **namespace**, what we can do is rename it referencing it as some other character(s) or word:

Just remember to **re-reference** any Sass variables with the new **namespace**.

(Similar in nature to Python, Java, C++)

If you don’t want to use ‘**namespaces’** at all, you can go the other route by providing the **\*** (**wildcard**) and then you won’t need to prefix the Sass variable with anything!



This could potentially leave room for ‘**conflicts’**, so it’s recommended to use some form of ‘**namespace’** instead of leaving **THIS UP TO CHANCE**.

**Final**: changed the **namespace** back to the ‘**…as u;**

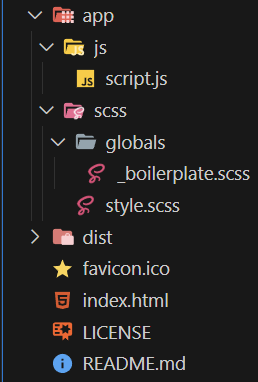
Let’s check our website and using the ‘**Inspector’**, we select the **<body>** tag and now check the rules:

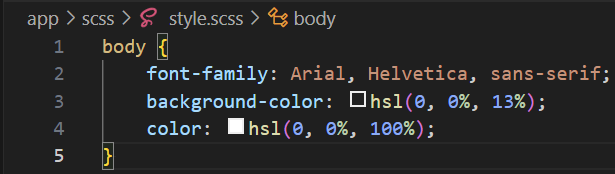
# Starter Code

**Note:** ‘dist’ folder at start doesn’t exist. It’s created at 1st compilation of VSC’s extension using Live Sass Compiler.

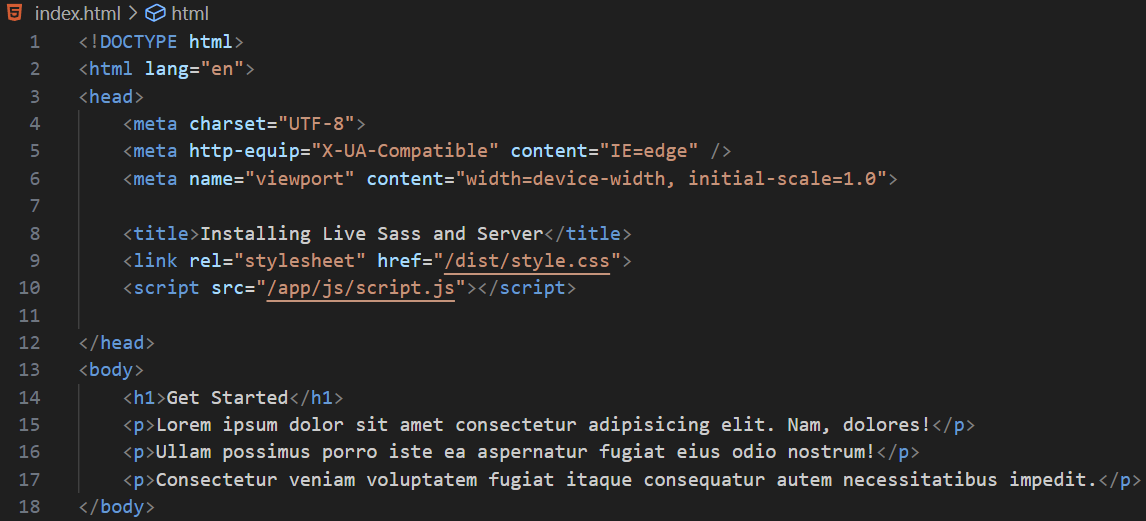
Layout:

1. folders: as listed here
2. files:

* **script.js** is empty
* **style.scss** manually created with basic element selector



* **boilerplate.scss** is empty
* **LICENSE** is empty
* **READEME.md** is empty
* **index.html** has basic template with added lorem text



1. ‘click’ “Watch …” in VSC’s bottom status bar (this will compile and add the “dist’ folder)
2. ‘click’ “Open Live Server” in VSC’s bottom status bar (opens browser with above webpage layout)