***Notas del Curso.***

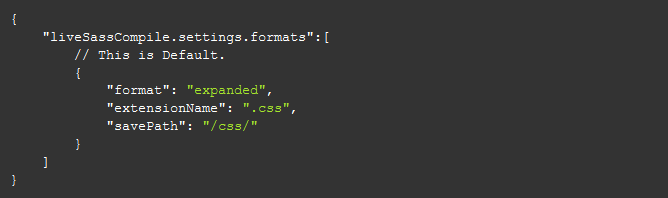
**Compilación de código en archivos CSS.**

Una de las primeras cosas que tenemos que aprender cuando estamos trabajando con Preprocesadores es la compilación de código que se compilará en archivos de CSS. En esta clase, te quiero mostrar cómo funcionan estos compiladores. En Mac utilizamos Codekit, y para Windows podemos usar Preposs.

**Configuración de VSCode para Sass.**  
  
Las extensiones que van a ser necesarias para el uso de Sass son:

* [**Live Server**](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer): Esta crea un servidor que se actualiza cuando guardamos, lo que nos permite ver los cambios sin necesidad de recargar manualmente. Es necesaria para la siguiente extensión.
* [**Live Sass Compiler**](https://marketplace.visualstudio.com/items?itemName=ritwickdey.live-sass): Esta transforma nuestros archivos .sccs o .sass a .css y crea un servidor basado en **Live Server**
* [**Sass**](https://marketplace.visualstudio.com/items?itemName=syler.sass-indented): Sirve para obtener sugerencias, autocompletado y formateo al escribir el código.  
  .  
  Para asegurarte que los archivos se guarde en la carpeta /project\_folder/css/.debes configurarlo con un archivo json de la siguiente forma:

1. Crea una carpeta en la raíz de tu proyecto con el nombre .vscode
2. Adentro crea un archivo con el nombre settings.json
3. En el archivo json copia lo siguiente:



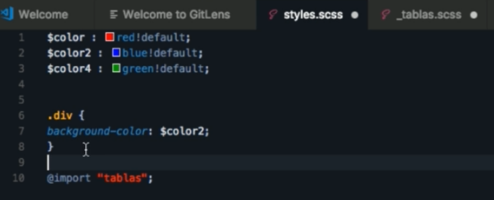
# Estructura e Importación de CSS

La primera ventaja que se nos ocurre es que podemos organizar nuestro Sass. Lo clave es que podemos separar nuestro código en archivos. Ya no tenemos que revisar un archivo muy amplio, sino que podemos hacer modificaciones a un solo archivo, lo que nos hace el trabajo mucho más fácil.

Para organizar e importar archivos usamos @import



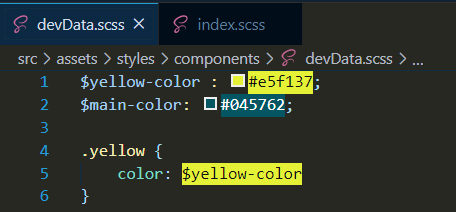
La forma de nombrar los archivos que van a ser importados luego, se les coloca un \_ al inicio.

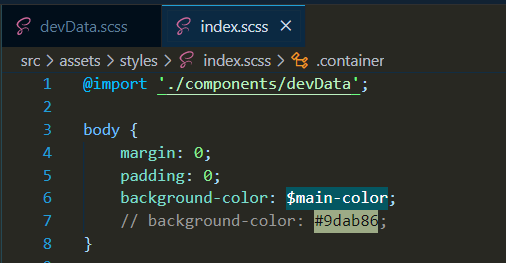


# Variables

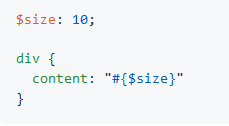
Las variables son una forma de almacenar la información que se desea reutilizar a lo largo de la hoja de estilo.

Se puede almacenar cosas como colores, pilas de fuentes o cualquier valor de CSS que se desea reutilizar. Sass usa el símbolo $ para hacer que algo sea una variable.





### **Escapar una variable.**

Para escapar una variable se usa el comodín #.

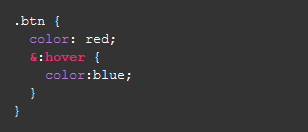
Esto es necesario en casos como, por ejemplo, cuando la variable está rodeada por comillas y de no ponerse el escape la variable pasaría como una cadena de caracteres.

## **Anidaciones.**



El comodín & se usa para hacer referencia al padre. Es decir que en este caso **&\_\_icon** es lo mismo que **.btn\_\_icon.**

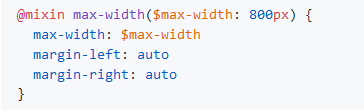
Pueden ser muy útiles para las pseudoclases. Por ejemplo:



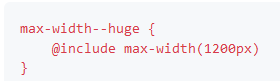
## **Mixins.**

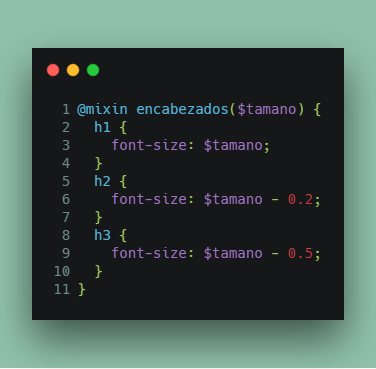
Los mixins nos ayudan a reciclar declaraciones para evitar mucho trabajo. Para esto vamos a usar @mixin.

Cuando se define un mixin, los argumentos se definen como una serie de variables separadas por comas, y todo ello encerrado entre paréntesis.



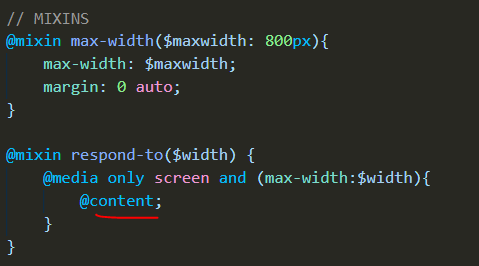
En este caso para llamarlo vamos a usar un **@include** y le estamos definiendo un valor por defecto. Si deseamos cambiar ese valor, cuando lo llamemos se lo podemos cambiar de esta forma:



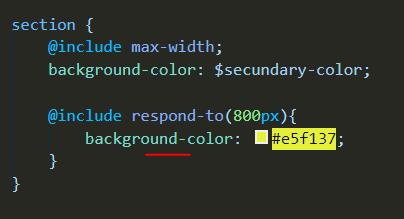
**Uso de Mixins con MediaQuerys Ejemplo 1.**

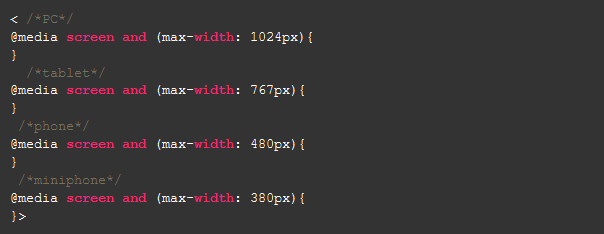
**Content y Uso de Mixins con MediaQuerys Ejemplo 2.**

Content es una de las características que tienen los mixins. Esta nos permite incluir un bloque de contenido dentro de un mixin.



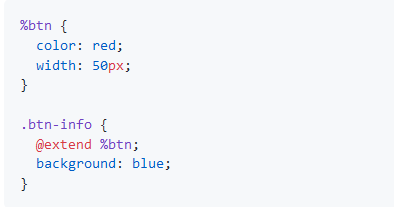
Se usa de esta forma:





## **Extend**

Una muy excelente manera para ahorrar código es con Extend que permiten que una declaración herede estilos declarados por otra regla o placeholder. Los extend se declaran con el símbolo de porcentaje %.



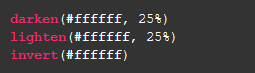
Todo lo que declaremos con **%** va a ser un extend. Es decir que estas propiedades solo se van a llamar cuando usemos @extend **%....** Es decir que mientras no se llame con el extend, este código no se llama al css.

Esto a final de cuentas va a ser igual que:

<button class=”btn btn-info”><button/>

# Funciones.

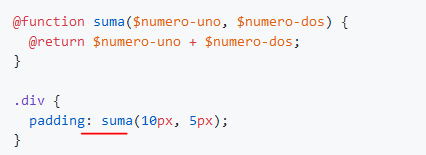
Sass tiene muchas funciones que podemos usar cuando estamos modificando CSS. Muchas de estas funciones son muy útiles como por ejemplo aclarar un color u oscurecerlo. Así hay muchísimas más, sin embargo hay algunas que no nos permiten hacer cambios visuales y que pueden parecer no tan útiles.



La lista completa de funciones se pueden ver aquí:  
<https://sass-lang.com/documentation/file.SASS_REFERENCE.html#functions>

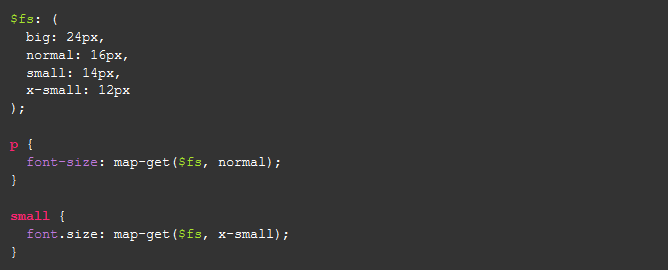
**Crear Funciones.**

A diferencia de los Mixins que nos permiten generar nuevas propiedades, estas funciones que vamos a poder crear van a funcionar más que todo en la forma en que podríamos manipular los datos en la asignación del valor de alguna propiedad de css:



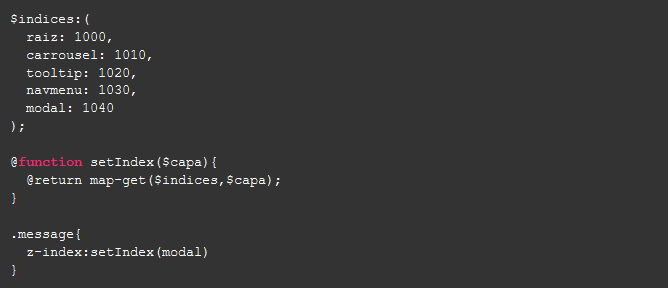
# Ejemplos de funciones de Sass.

Las funciones también puede ser extraer valores de un array. Una función puede utilizar arrays que son más parecidas a listas. Para Sass es más un mapa. En esta clase vamos a aprender a usar mapas con los posibles tipos de fuentes que hayan.



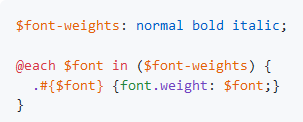
En este caso tenemos un arreglo nombrado **$fs.** Yparaacceder a uno de sus valores vamos a usar la funcion de Sass **map-get.** Como primer argumento será el arreglo a acceder y el segundo argumento será el nombre de esa propiedad del arreglo, que nos devolverá su valor.

Un ejemplo de uso de esto es usar estos arreglos para organizar los z-index de un proyecto.



## **Controles de Flujo con Sass.**

### **each**



y esto da como resultado:

### 

### **for**



Resultado:

### 

### **if**



**Documentación de Sass.**

[**https://sass-lang.com/documentation**](https://sass-lang.com/documentation)

**Important Questions and Answers.**

