**SASS (Syntactically Awesome Stylesheets)**

Contenido

[1. Introducción 3](#_Toc94591909)

[2. Instalación SASS 3](#_Toc94591910)

[3. Variables SASS 6](#_Toc94591911)

[4. SASS Nesting 12](#_Toc94591912)

[5. Mixins 20](#_Toc94591913)

[6. Extend 22](#_Toc94591914)

[7. Functions 24](#_Toc94591915)

[8. Placeholder Selectors 25](#_Toc94591916)

[9. Import and Partials 26](#_Toc94591917)

[10. Práctica guiada: tienda de motos 27](#_Toc94591918)

[10.1. Descripción del proyecto. 27](#_Toc94591919)

[10.2. Preparación del workspace 27](#_Toc94591920)

[10.3. Header 28](#_Toc94591921)

[10.3.1. Estructura HTML 28](#_Toc94591922)

[10.3.2. Estilos 29](#_Toc94591923)

[10.4. About us 33](#_Toc94591924)

[10.4.1. Estructura HTML 33](#_Toc94591925)

[10.4.2. Estilos 34](#_Toc94591926)

[10.5. Galería 36](#_Toc94591927)

[10.5.1. Estructura HTML 36](#_Toc94591928)

[10.5.2. Estilos 38](#_Toc94591929)

[10.6. Footer 40](#_Toc94591930)

[10.6.1. Estructura HTML 40](#_Toc94591931)

[10.6.2. Estilos 41](#_Toc94591932)

[10.7. NavBar 42](#_Toc94591933)

[10.7.1. Estructura HTML 42](#_Toc94591934)

[10.7.2. Estilos 43](#_Toc94591935)

[10.8. Hacer el proyecto responsive 46](#_Toc94591936)

[11. Tipos de datos en SASS 66](#_Toc94591937)

[Numbers 66](#_Toc94591938)

[Strings 66](#_Toc94591939)

[Colors 67](#_Toc94591940)

[Lists 67](#_Toc94591941)

[Maps 67](#_Toc94591942)

[12. Sass Interpolation 68](#_Toc94591943)

[13. For Loop 68](#_Toc94591944)

[14. Each Loop 69](#_Toc94591945)

[15. @content 70](#_Toc94591946)

[16. If directive 73](#_Toc94591947)

[17. Arquitectura 75](#_Toc94591948)

[17.1. Componentes 75](#_Toc94591949)

[17.2. Estructura de un componente 76](#_Toc94591950)

[17.3. El patrón 7-1 76](#_Toc94591951)

[Carpeta base/ 77](#_Toc94591952)

[Carpeta layout/ 77](#_Toc94591953)

[Carpeta components/ 77](#_Toc94591954)

[Carpeta pages/ 77](#_Toc94591955)

[Carpeta themes/ 78](#_Toc94591956)

[Carpeta abstracts/ 78](#_Toc94591957)

[Carpeta vendors/ 78](#_Toc94591958)

[Archivo principal main.scss 78](#_Toc94591959)

[18. Enlaces, recursos y bibliografía: 80](#_Toc94591960)

# Introducción

Como ya sabemos, CSS no es un lenguaje de programación, lo cual implica que a veces escribirlo sea muy laborioso y nos obligue a teclear mucho código para llevar a cabo algunas de sus funcionalidades. Con el objetivo de conseguir un lenguaje de maquetación y estilos algo más dinámico surge SASS (Syntactically Awesome Stylesheets).

SASS implica un lenguaje y un preprocesador ya que requiere una conversión a CSS antes de que pueda utilizarse en una página, es lo que se llama “transpilar”.

# Instalación SASS

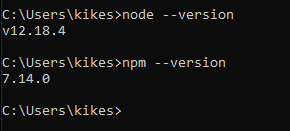
Hay varias formas de instalar SASS en nuestro sistema, algunas de ellas de pago. Para una información detallada de todas las formas de instalación podemos echar un vistazo [aquí](https://sass-lang.com/install).

Básicamente hay dos formas: usando aplicaciones y usando la línea de comandos. Nosotros usaremos la línea de comandos. La ventaja de la instalación que vamos a ver es que es válida para cualquier plataforma (Windows, Mac, Linux, etc.)

Lo primero que haremos es instalar Node.js. Node.js es un Entorno de ejecución de JavaScript (JavaScript runtime enviroment) para servidor. Con SASS no vamos a usar Javascript, pero necesitamos instalar Node.js porque necesitamos utilizar npm, el gestor de paquetes de javascript. Node.js ya contiene npm. Por tanto, vistamos la página nodejs.org, descargamos la última versión y la instalamos.

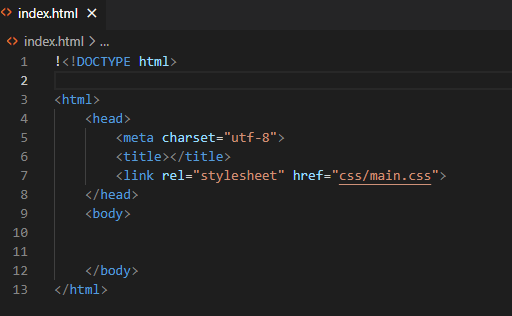
Para ver si hemos instalado correctamente:

tanto node.js como npm en una ventana de comandos ejecutamos



**CREACIÓN DEL ESPACIO DE TRABAJO**

1. Creamos un archivo html



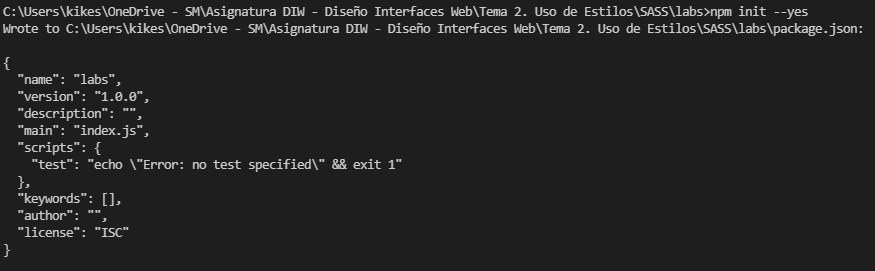
2. Creamos una carpeta css y dentro un archivo main.css

3. Creamos una carpeta scss y un archivo main.scss

4. Creamos el archivo package.json

para ello ejecutamos en una ventana de comandos el comando

npm init –yes



5. Aquí vemos el nombre de la aplicación, la versión, etc.

6. Ahora tenemos que instalar SASS que será la que nos transpile el código que escribamos SASS a CSS.

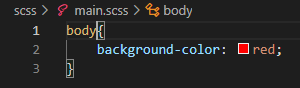
npm i -g node-sass

El modificador -g hace que lo instale globalmente, es decir, instalará el transpilador de tal forma que podamos usarlo en otros proyectos.

7. Compilación de SASS a CSS. Con este comando transpilamos el código fuente de SASS a CSS, como primer parámetro le pasamos la carpeta donde tiene que llevar a cabo el proceso de transpilación y como segundo parámetro el archivo fuente.



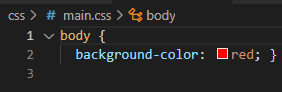
Por ejemplo, si en el archivo main.scss escribimos



Tendríamos que transpilarlo con el comando:



Y nos lo transpila al fichero main.css



Sin embargo, hacerlo de esta forma nos obligaría a compilar manualmente cada vez. Hay una alternativa y es escribiendo el siguiente comando:



El parámetro “w” significa watch, de esta forma nos compilará automáticamente), en cuanto hagamos algún cambio y guardemos. el fichero scss.

Otra forma es modificando el archivo package.json



Ahora ejecutaríamos npm run watch.

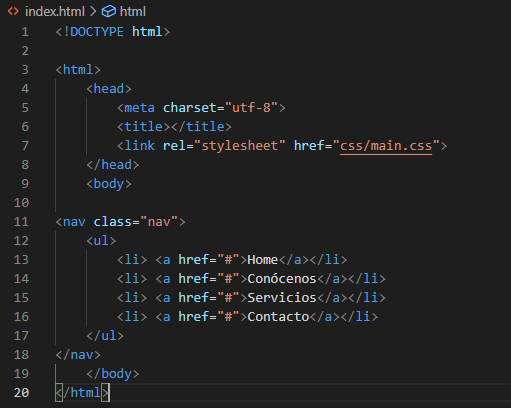


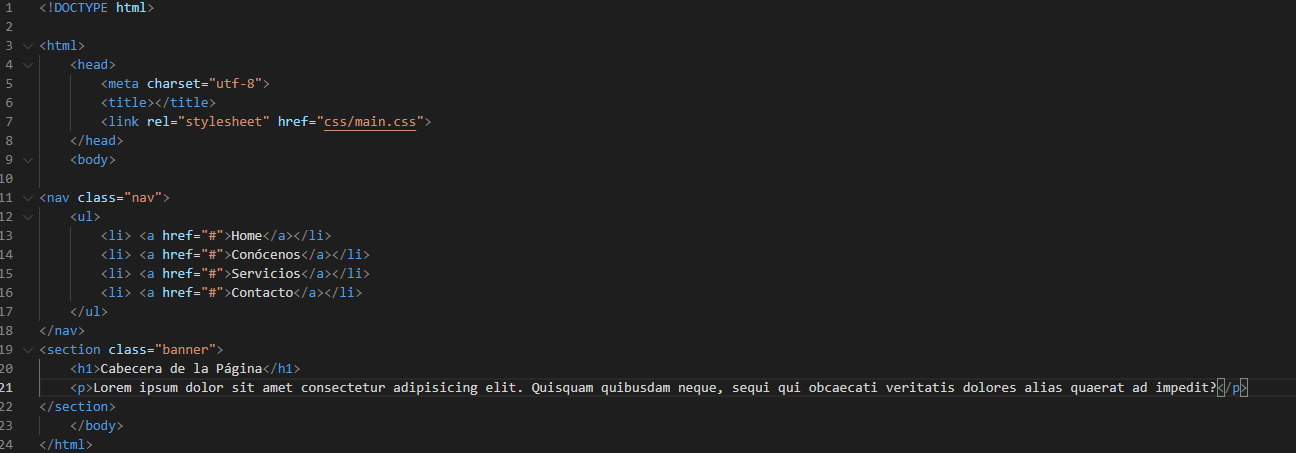
# Variables SASS

SASS nos permite usar variables al modo que lo usamos en programación.

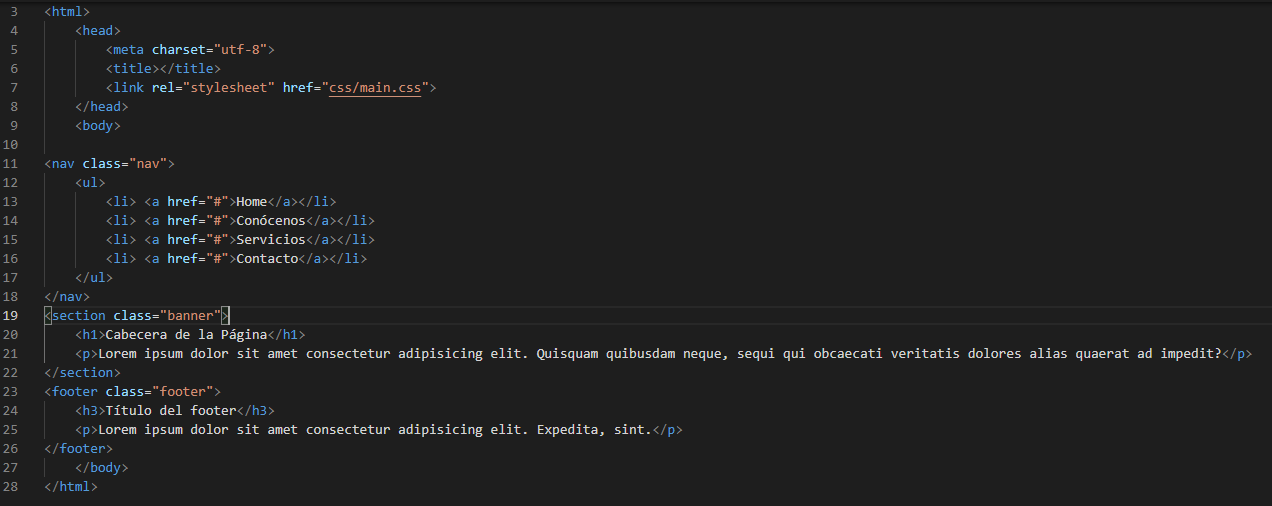
Para demostrar cómo se usa primero crearemos en nuestro index.html varios elementos:

Una sección de nav:

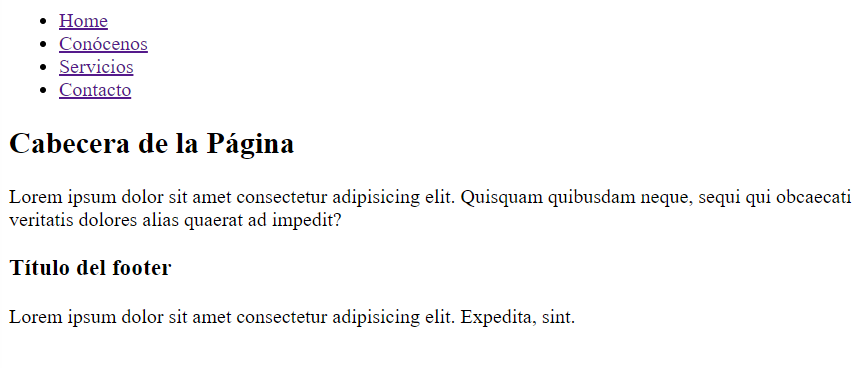


Una sección de banner con un h3 y un paragraph

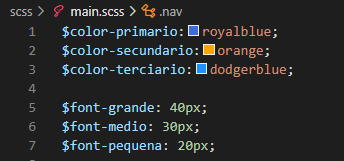
y un footer con un h3 y un paragraph



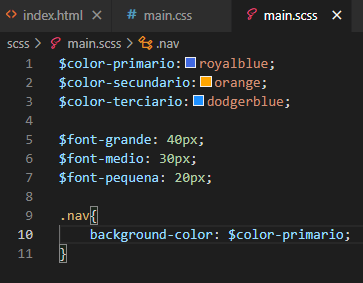
Quedaría algo así:



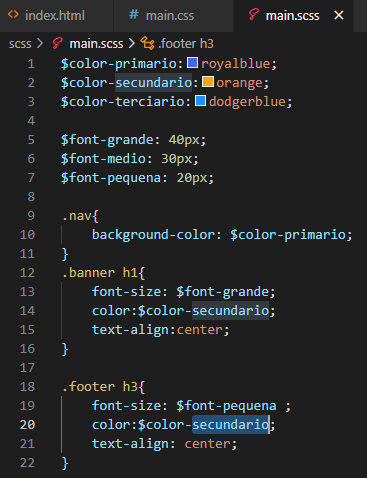
Es muy común usar en las webs una gama de colores para mantener la apariencia del sitio: color primario, secundario y terciario. Estos valores los podemos almacenar en variables para usarlos posteriormente en otras partes del código de tener que escribir el valor del color continuamente.



Vamos a establecer, además, los diferentes tamaños de fuente:



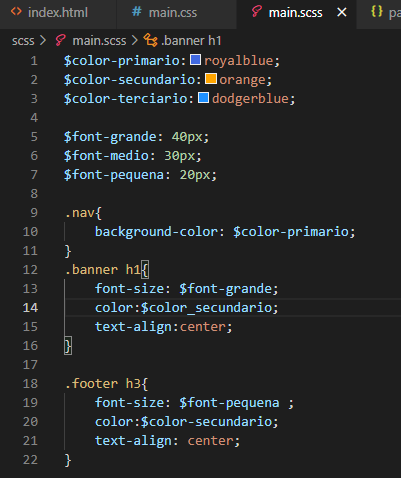
Ahora vamos a usar esas variables recién creadas:



(Nota: si vamos a ver el fichero CSS deberíamos ver que se compila de forma correcta).

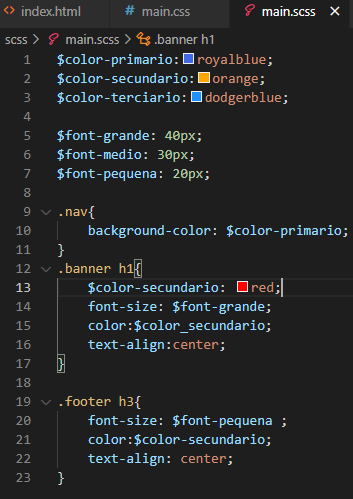
Si ahora cambiamos el valor de una variable veremos cómo se actualiza en todas las partes donde la hemos usado, esto facilita muchísimo la mantenibilidad de nuestro código, por ejemplo, si la empresa decide cambiar el color corporativo secundario, no tendremos que ir cambiándolo en todos los sitios donde hemos usado el color.

**Nota**: Una curiosidad en SASS es que se puede usar las variables de esta forma $color-secondary o de esta: $color\_secondary de forma idéntica. Ambas funcionarán.



Alcance de las variables (variable scope)

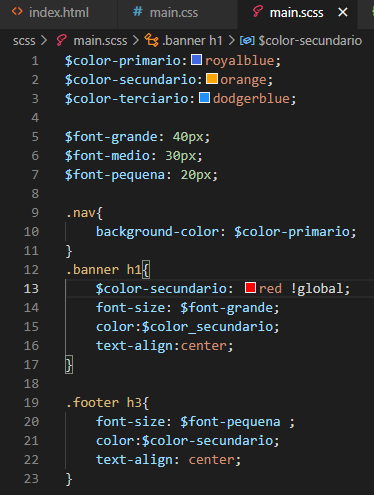
Si declaramos una variable con el mismo nombre dentro de una regla SASS solo tendrá efecto dentro de esa regla, por ejemplo:





Solo cambia aquí

Si quisiéramos que tuviera efecto a nivel global tendríamos que marcarlo con la palabra reservada !global :



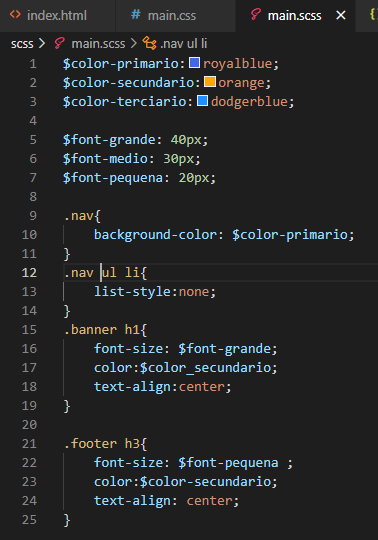


Esto se puede hacer, aunque no es recomendable: no es recomendable ni crear variables locales y marcarlas con el atributo global.

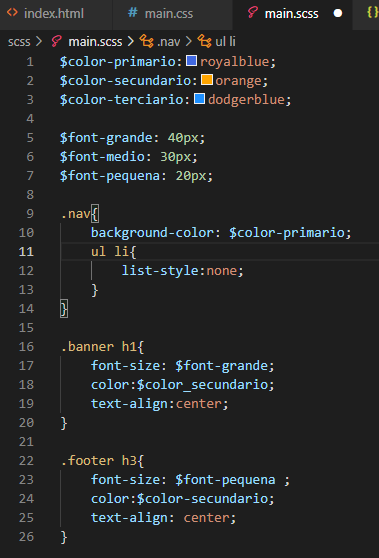
# SASS Nesting

Nos permite ahorrar gran cantidad de líneas de código en CSS.

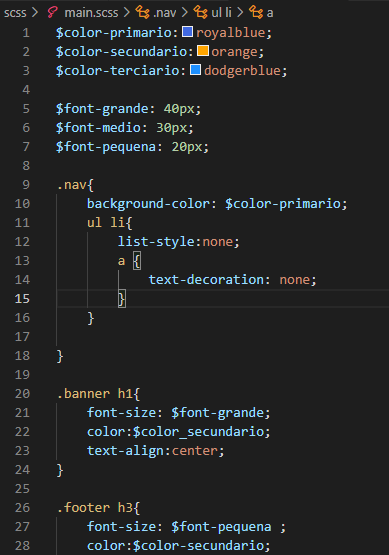
Imaginemos que queremos quitar los bullet points de la lista desordenada del nav. Al modo CSS lo haríamos así:



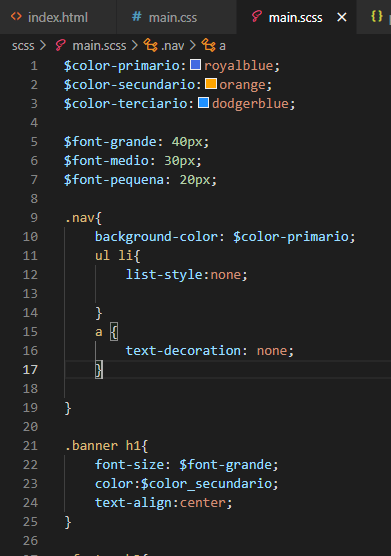
Con SASS tenemos una forma más eficiente de hacerlo, lo podríamos anidar de esta forma:



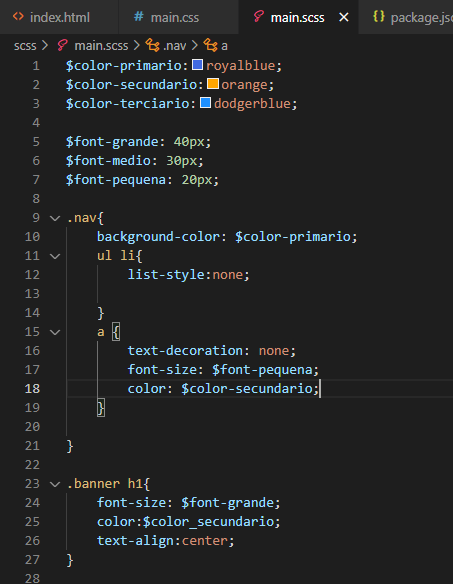
Si quisiéramos quitar el subrayado a los links podríamos poner:



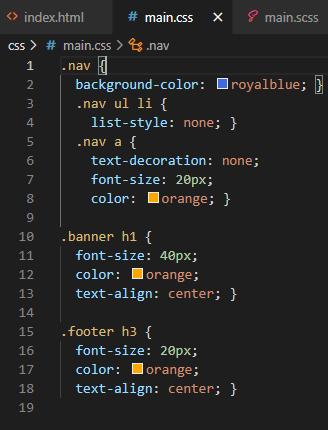
Aunque también se podría poner así, no es conveniente anidar demasiado las propiedades, ya que el debuggeo es mucho más complicado.



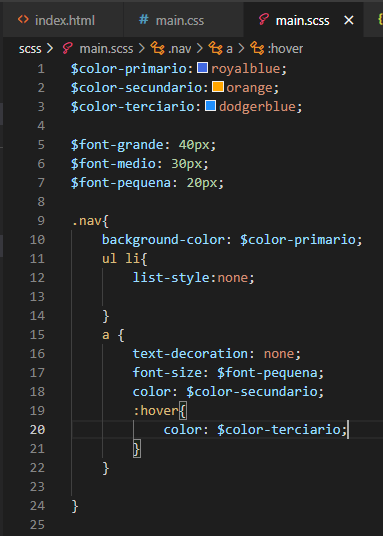
Con algo más de estilos quedaría de esta forma:



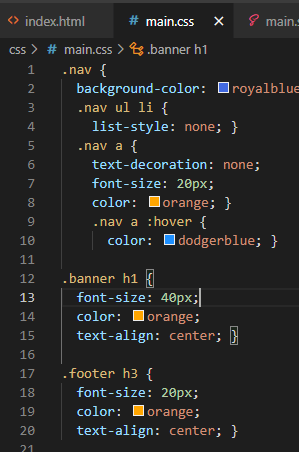
Ahora, si chequeamos el fichero main.css (el css transpilado) veremos lo siguiente:



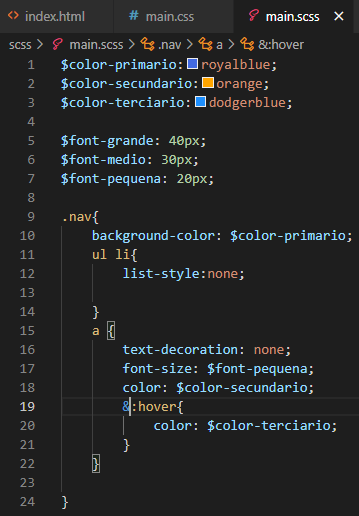
Si quisiéramos poner que los enlaces cambiaran de estilo cuando pasemos el ratón por encima podríamos poner lo siguiente:



Pero vemos que no funciona. Si vamos al fichero css. Vemos que se ha transpilado de la siguiente forma:



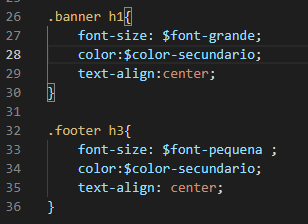
Le ha puesto un espacio entre el a y el :hover. Para unir dos propiedades hay que usar el &. Quedaría de esta forma:



# Mixins

Mixins sirve para lograr el principio de DRY (Don’t Repeat Yourself). Sirve para agrupar código que podamos reutilizar en más sitios.

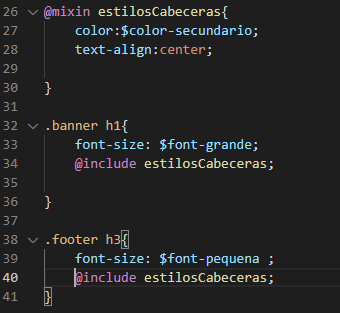
En el ejemplo vemos que hay estilos que se están repitiendo, por ejemplo, el h1 del banner y el h3 del footer tienen estilos comunes:



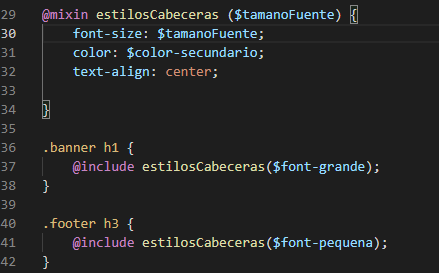
Para crear un mixin se usa la siguiente nomenclatura:

@mixin [nombre-mixin]

Para usarlos posteriormente lo haremos con la clausula @include.

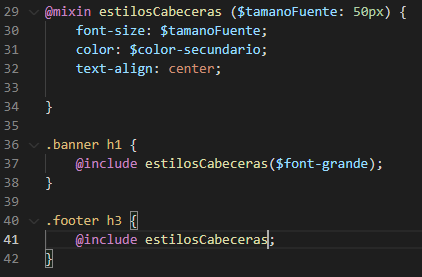


Mixins también nos permite usarlo como funciones introduciéndole parámetros:



Nos permite personalizar algunas propiedades.

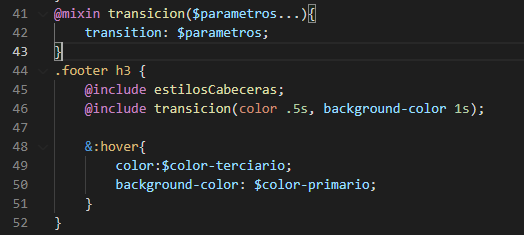
También podemos ponerle un valor por defecto, por ejemplo:



Si no establecemos nada, el tamaño fuente será

de 50px

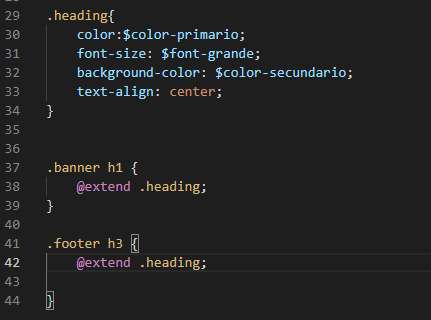
Otra característica de los mixins es que podemos definir parámetros de entrada variables.



# Extend

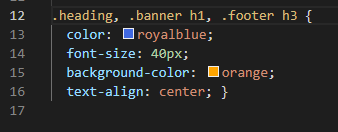
Sirve que un selector herede estilos de otro selector, permite agrupar selectores con estilos comunes.

Para seguir con el ejemplo vamos a borrar los mixins creados y crearemos una clase .heading a la cual le daremos estilos que heredarán otras clases:

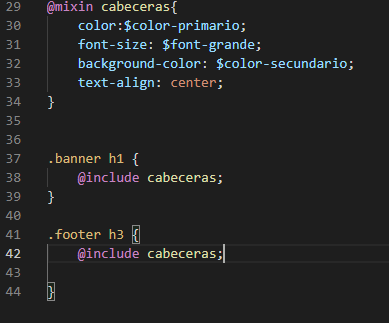


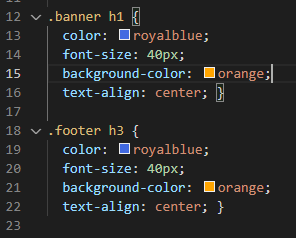
Las diferencias entre extend y mixin son básicamente dos. Por un lado, cómo queda el código CSS resultante:

Con extend:



Con mixins:





Además, mixins nos permite pasar parámetros.

# Functions

Las funciones son similares a los mixins con la diferencia de que un mixin devuelve una porción de código, mientras que las funciones devuelven un valor.

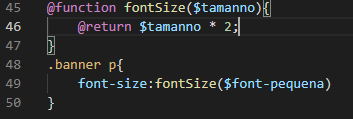
SASS nos proporciona funciones nativas pero, además, nos permite construir nuestras propias funciones:

Para crear una función usaremos la siguiente sintaxis:

@function nombreFuncion($parametro){

….

}



En este caso no se ve la utilidad, pero en proyectos grandes puede ahorrar mucho código.

Funciones del propio lenguaje. Tal y como habíamos dicho SASS tiene funciones construidas en el propio lenguaje, por ejemplo, relativas a los colores:

lighten (color, porcentaje): Aclara el color que se pasa por parámetro en un porcentaje dado.

darken (color, porcentaje): El opuesto al anterior.

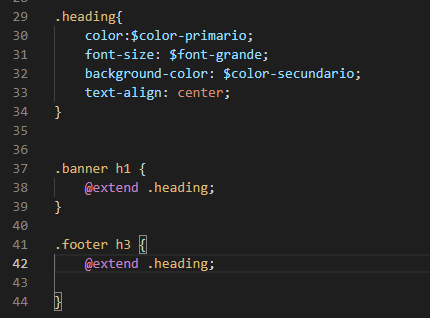
transparentize(color, porcentaje): Permite hacer el color transparente en un porcentaje que le marquemos (el porcentaje irá de 0 a 1)

mix (color, color, porcentaje): Mezcla dos colores con un porcentaje.

Para un catálogo completo ver [aquí](https://sass-lang.com/documentation/modules).

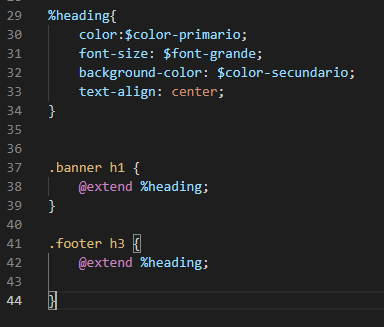
# Placeholder Selectors

Cuando usamos la herencia tenemos que crear una clase que luego será heredada por otras. Esta clase se crea en el CSS resultante aunque luego no se use en alguna otra clase. Para evitar esto (que se cree en el CSS resultante aunque no se esté usando) podemos usar los placeHolder selectors:



Creará en el CSS la clase heading aunque luego no se use

Con placeholders quedaría de esta forma:



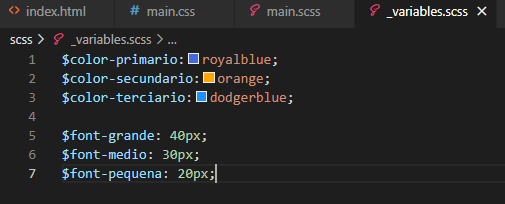
Permite crear código resultante más conciso y legible.

# Import and Partials

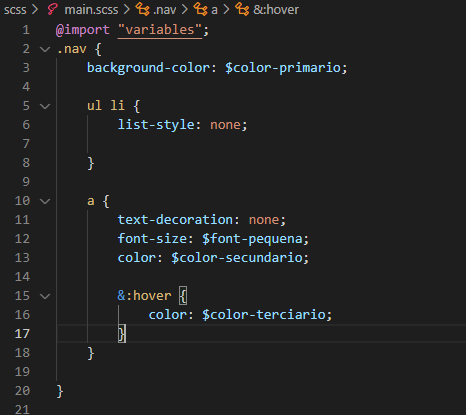
Cuando los proyectos crecen es preferible dividir el código en varios ficheros para hacerlo más mantenible. Por ejemplo, es común poner en un archivo separado todas las variables, o dedicar un archivo para los colores, otro para las fuentes, otro para el reset de estilos, etc. Posteriormente estos archivos se importarán en otro de estilos.

Por defecto, SASS transpila todos los archivos que tengan la extensión .scss, en el caso de que hayamos separado parte del código en un fichero diferente no queremos que esto suceda, ya que este fichero forma parte del archivo que los importa. Para evitar que SASS transpile estos ficheros tenemos que nombrarlos con un guión bajo por delante, por ejemplo “\_variables.scss”. Cuando SASS detecta este guión bajo sabe que es un archivo de tipo “partial” y no tiene que transpilarlo.

En el ejemplo, quitaremos la declaración de variables del archivo main.scss y para ponerlas en el archivo de nueva creación \_variables.scss.



Posteriormente tenemos que importar este archivo en el main.scss.



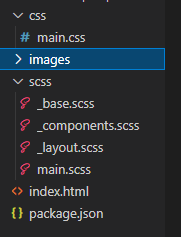
# Práctica guiada: tienda de motos

## Descripción del proyecto.

En esta práctica guiada crearemos una tienda de motos. El objetivo es ver cómo se estructura un proyecto hecho en SASS

## Preparación del workspace

Seguimos los pasos que dimos en el apartado 2. Además, crearemos los siguientes ficheros y estructura de directorios:



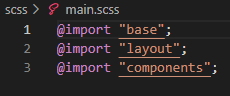
En el archivo main importaremos el resto de los archivos.

En el partial \_base.scss incluiré todas las variables, mixins, reseteo de estilos, etc

En el partial\_components incluiré componentes reusables como botones, headings, subheadings, etc.

En \_layout.scss incluiré los componentes relativos al layout.

El archivo main.scss tendrá el siguiente contenido:



En el archivo html crearé la estructura básica:



Usaré fontawesome y Google fonts.

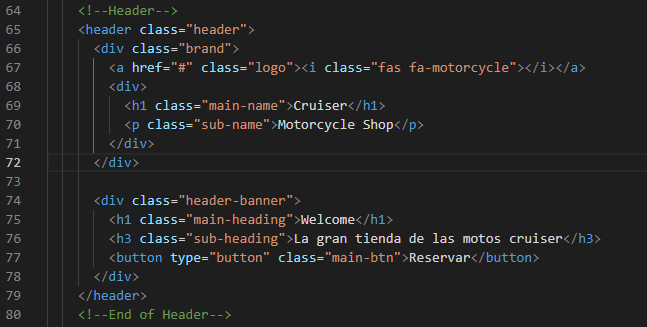
## Header

### Estructura HTML

Comenzaré con la estructura del header que se compondrá de dos partes:

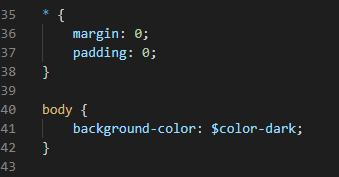
* Un logo un título, un sutítulo (en naranja)
* Un banner con un heading y un subheading (en rojo)



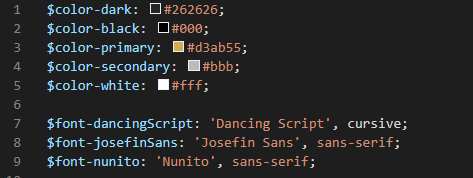


### Estilos

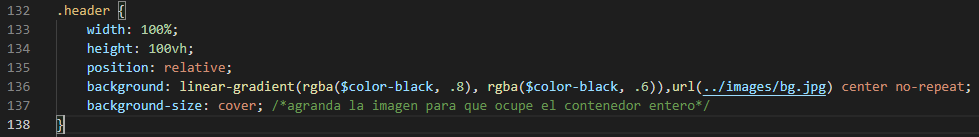
Antes de nada, vamos resetear los estilos que los navegadores dan por defecto, además pondremos un background negro, esto lo haremos en el archivo \_base.scss:



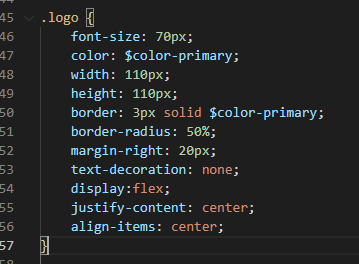
En el archivo \_base.scss definiré las variables que voy a usar:



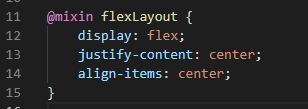
Los estilos del background los definiré en el fichero \_layout.scss. Vamos a necesitar una imagen de background que ocupe el 100%, que sea negro, y que tenga una foto que ocupe el 100%:



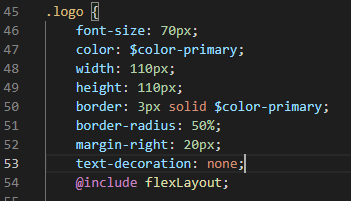
En el archivo \_components.scss voy a dar estilos al logo. Al logo le pondré el color primario del sitio y lo centraré:



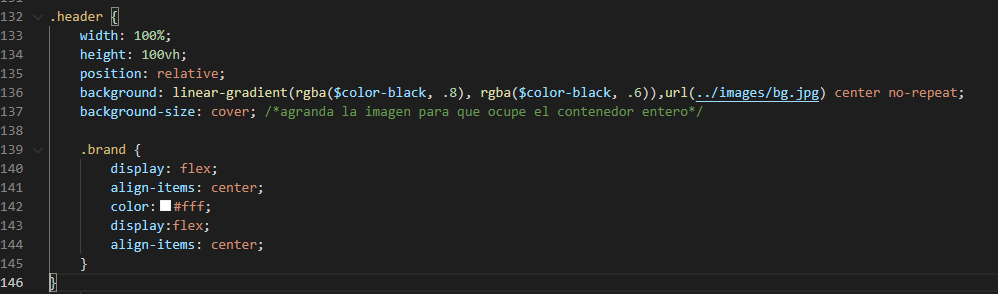
El código de flexbox es algo que voy a usar mucho en el sitio, así que lo voy a extraer a un mixin. Lo crearé en el archivo \_base.scss:



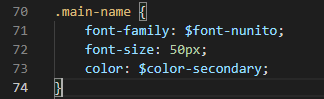
Y en el archivo \_components.scss quedará así:



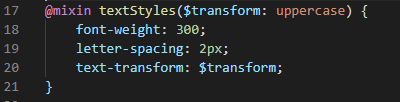
Ahora tengo que poner el logo y la marca uno junto al otro. Para ello, en el archivo \_layout.scss:



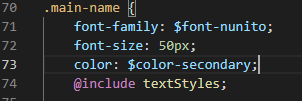
Ahora en el archivo \_components.scss daré estilos al nombre principal:



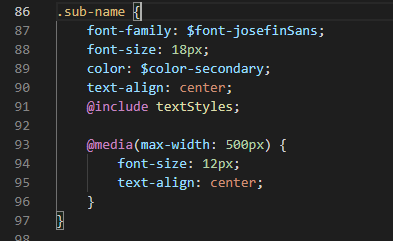
Voy a crear otro mixin para los estilos de texto que voy a repetir en varias partes. Lo crearé en el archivo \_base.scss:



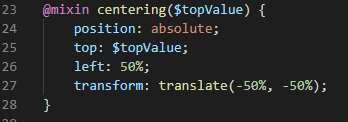
El estilo de la clase .main-name (fichero \_components.scss) quedará así:



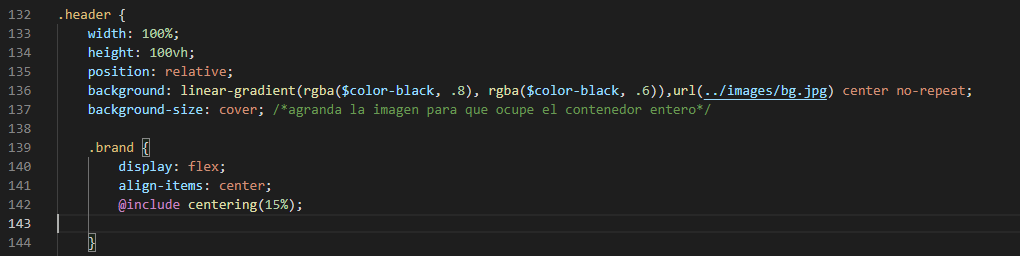
Ahora daré estilos a la clase sub-name:



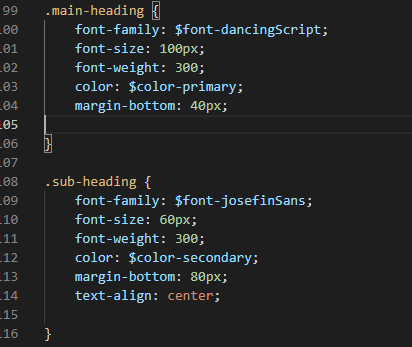
Vamos ahora a trabajar con el banner. En el archivo \_layout.scss lo anidaré en la clase .header. Para acceder a él aprovechar la posibilidad de anidación que nos proporciona SASS. El banner tiene que estar centrado tanto verticalmente como horizontalmente, para ello crearé un mixin en el archivo \_base.scss y poder así reutilizarlo:



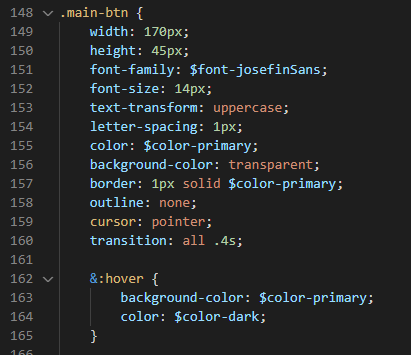
ahora lo usaré en la clase banner (archivo \_layout.scss):



Ahora en el archivo \_components.scss daré estilos al main-heading y al sub-heading:



Ahora daré estilos al botón principal:



## About us

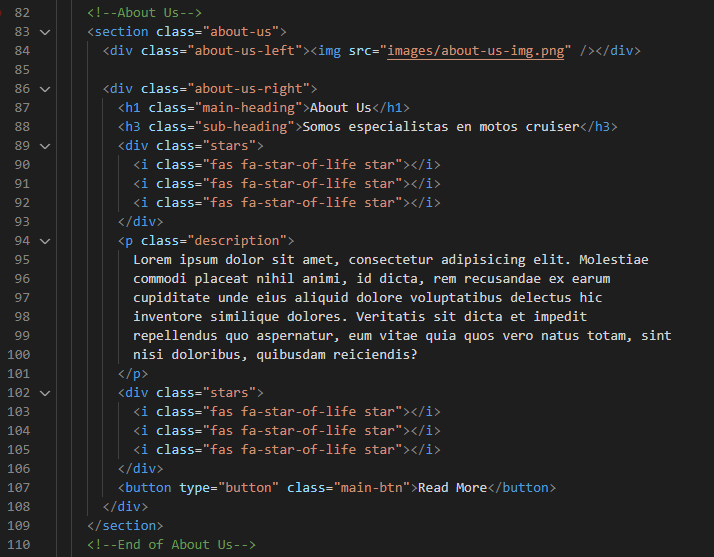
### Estructura HTML

Consta de dos partes:

* Una parte a la izquierda (la imagen).
* Otra parte con el texto e iconos (estrellas a modo de separador).



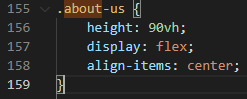
Lo primero de todo creamos la estructura html:



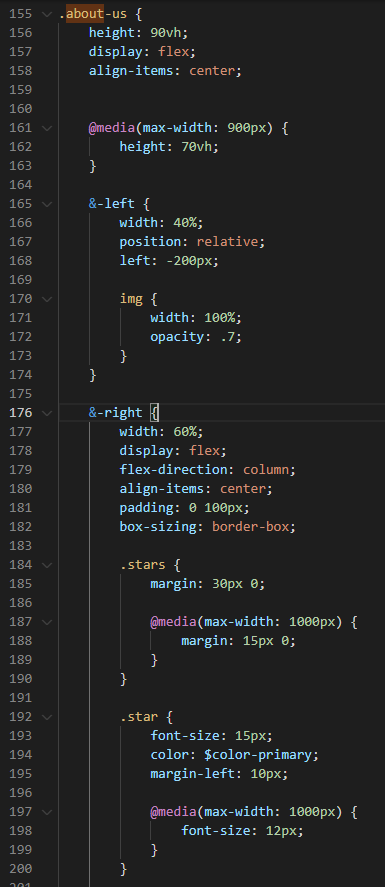
### Estilos

Estilos de la sección about Us

Definimos el layout en el fichero \_layout.scss

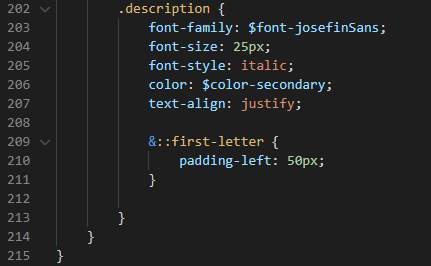


Ahora damos estilos a la parte izquierda y a la parte derecha aprovechando el anidamiento que nos permite SCSS:



(continúa….)

Continuación de clase .about-us:



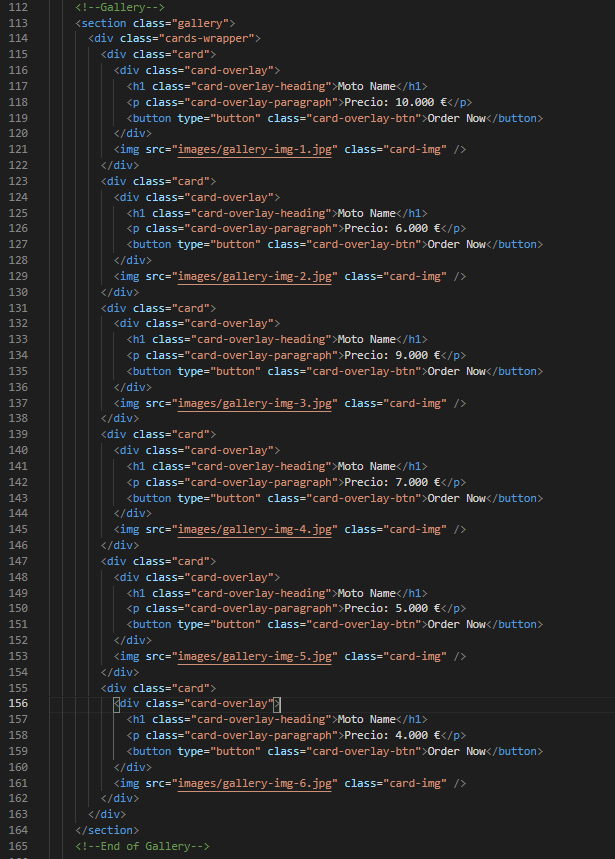
## Galería

Ahora haremos la galería de fotos.

### Estructura HTML

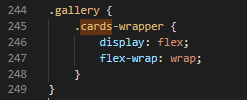
Primero de todo es crear la estructura html.

Crearemos 6 imágenes y queremos que al hacer hover haga un efecto overlay. A cada elemento le daremos una clase llamada card y anidado en la card tendremos por un lado la foto y por otro el overlay.



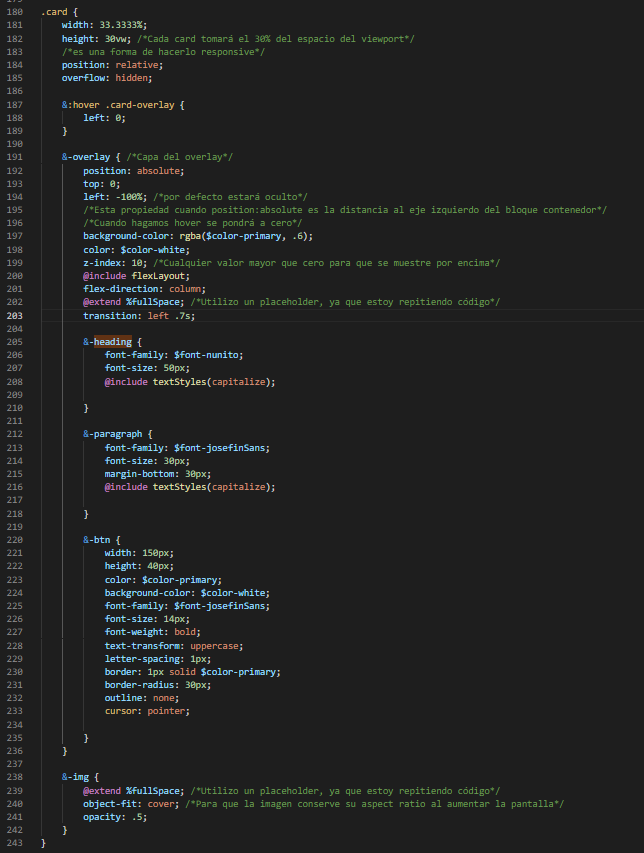
### Estilos

Lo primero que haremos es el layout de las fotos y para ello usaremos flexbox (esto lo haremos en el fichero \_layout.scss).

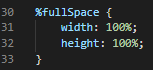


Con este código nos pone las cards horizontalmente. Ahora tenemos que dar estilos a cada una de las cards.

Ahora vamos a colocar las cards, definiremos la clase en el fichero \_components.scss. Queremos que haya tres fotos por línea:



He utilizado un placeholder para evitar repetir código. El placeholder lo implemento en el fichero \_base.scss:



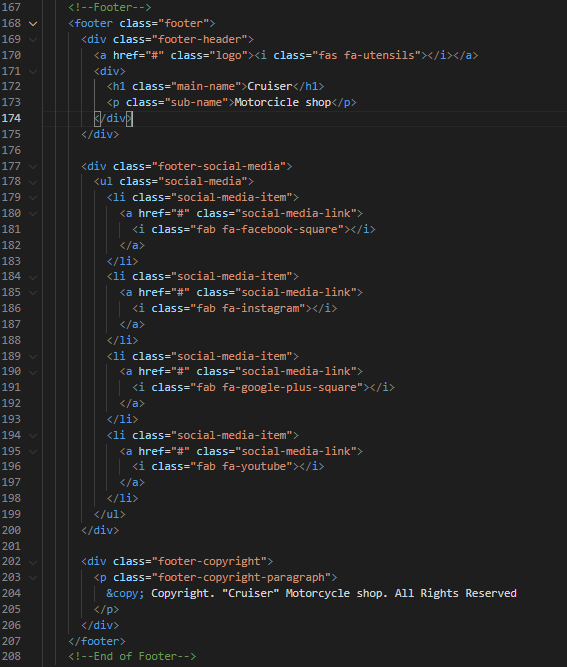
## Footer

### Estructura HTML

La estructura consta de tres partes:

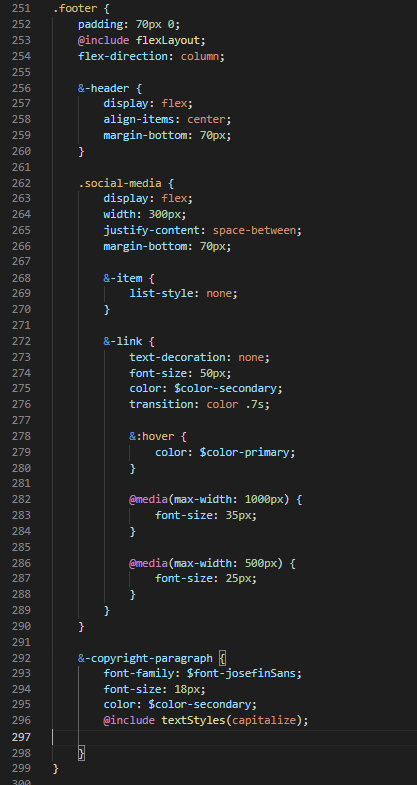
* Logo, name y subname.
* Iconos de redes sociales.
* copyright





### Estilos

El estilo del footer es sencillo y hay poco que comentar. En el fichero \_layout.scss escribiremos:



## NavBar

Por defecto el NavBar estará oculto, en su lugar se mostrará un hamburger que cuando se presione se mostrará.

### Estructura HTML

Lo situaré al principio del documento.

Un menú hamburger se puede hacer de muchas formas. Lo más sencillo sería usando javascript. De todas formas, como se trata de hacerlo con CSS puro utilizaremos la técnica de usar un checkbox para detectar un evento de click sobre el menú hamburger:



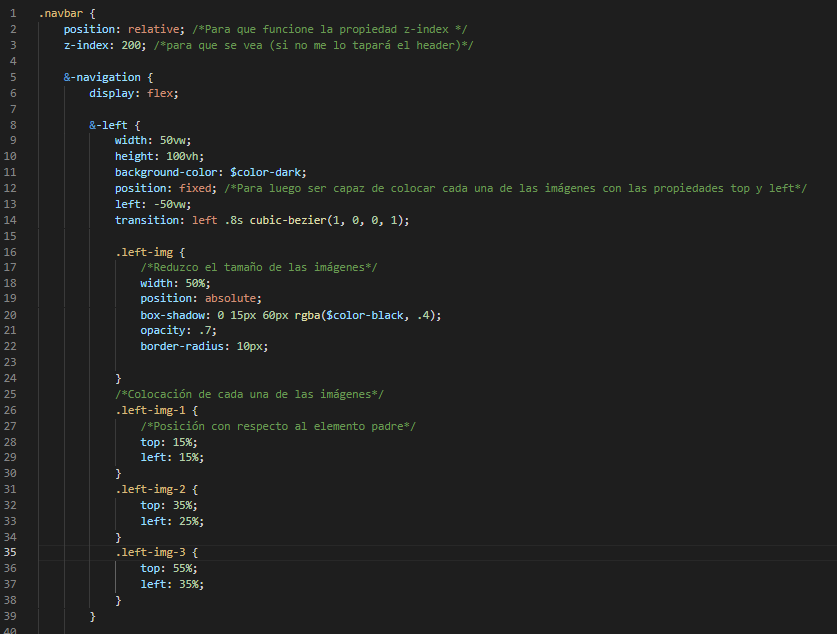
Este es el menú de navegación propiamente dicho que está dividido en dos partes (left y right)

El checkbox está oculto

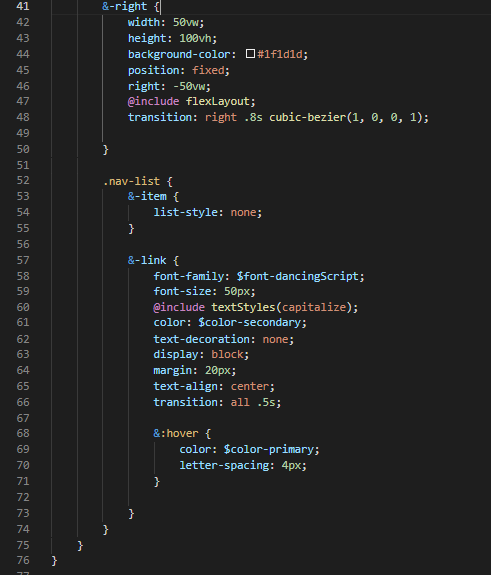
Tres líneas del menú hamburger

### Estilos

Primero empezaré con los estilos de la parte izquierda (imágenes). En el archivo \_layout.scss:



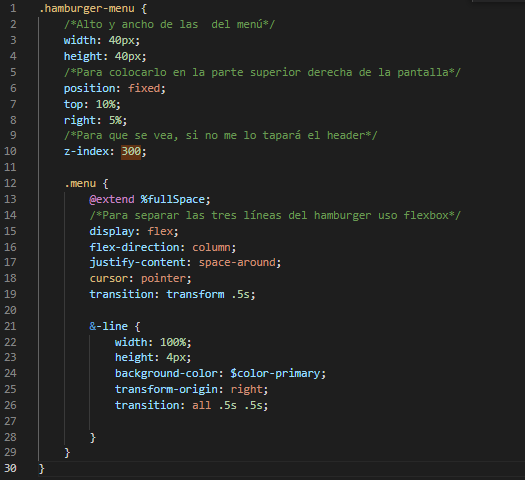
Parte izquierda (anidada al mismo nivel que &-left):



Menu Hamburger

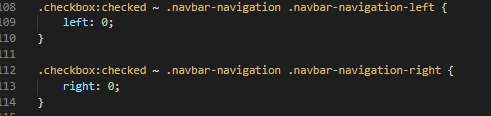
Ahora tenemos que crear el menú hamburger y hacerlo funcionar.

Para la creación del menú:



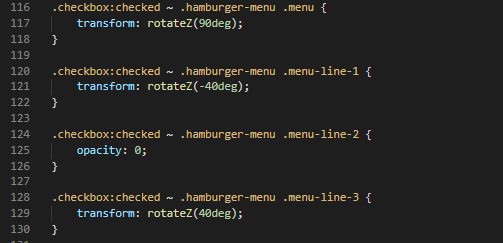
Tamaño de las líneas del hamburger

Ahora tengo que hacer funcionar el menú, es decir, cuando se haga click sobre él se debe abrir el navbar. Para ello, añadiré en el fichero \_layout.scss:



Selector de hermano

Para que el menú hamburger gire cuando se selecciona tendremos que añadir el siguiente código al fichero \_layout.scss:



## Hacer el proyecto responsive

Para hacer el proyecto hemos llevado a cabo lo que se llamaría una estrategia Desktop first. Ahora lo vamos a adaptar a dispositivos móviles.

La estrategia a seguir es ir reduciendo la pantalla e ir viendo en qué tamaño tengo que establecer breakpoints

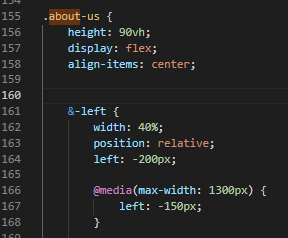
El primer breakpoint he decidido que sea 1600px. En ese punto debería de trabajar el tamaño del texto en las sección About us. Para ello le cambiaré el tamaño de letra (en el fichero \_components.scss). Utilizaré el nesting que me proporciona SASS.:



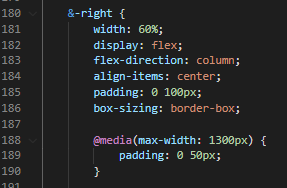
Ahora tengo que introducir también una media query en la clase description (fichero \_layout.scss):



El siguiente punto de corte para el tamaño de la fuente lo pondré en 1300px. En este punto voy a mover la moto que aparece a la izquierda de la sección “About us” a la derecha (fichero \_layout.scss):

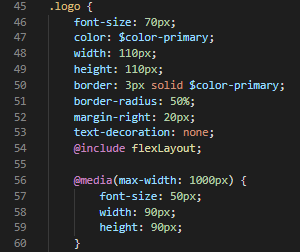


A la parte derecha le voy a reducir el padding arriba y abajo, a la derecha e izquierda le pondré un padding de 50px.

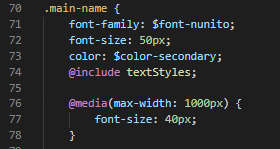


El siguiente breakpoint lo voy a dejar en 1000px:

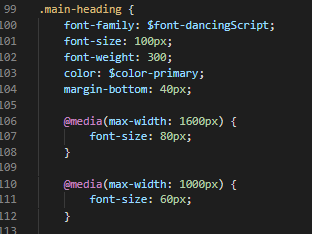
En este punto modificaré el logo

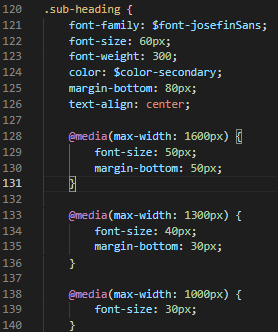


También modificaré el tamaño del main-name:

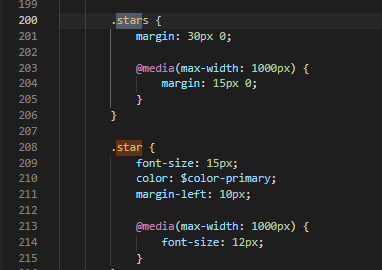


En el main-heading y sub-heading también cambiaré el tamaño del texto:

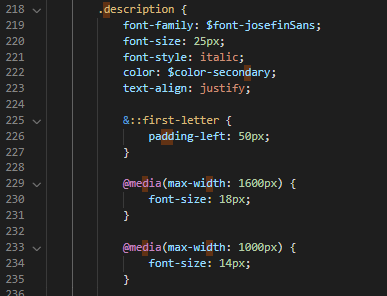




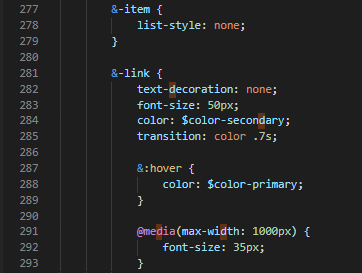
Ahora cambiaré el tamaño de las estrellas:



Cambiaré también el tamaño de la fuente de la descripción:

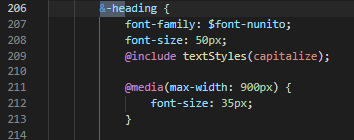


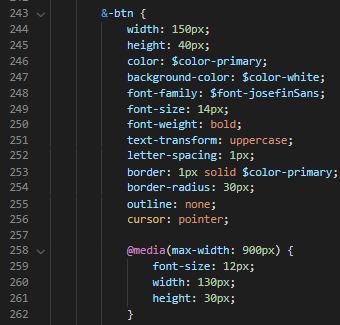
Y de las redes sociales en el footer:

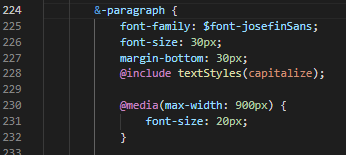


El siguiente punto de corte lo voy a establecer en 900 ya que el overlay de las imágenes no queda muy bien. Modificaré el heading del card, el botón y el paragraph.

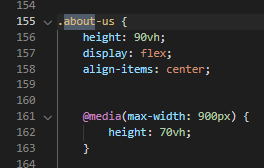
En el fichero \_components.scss:



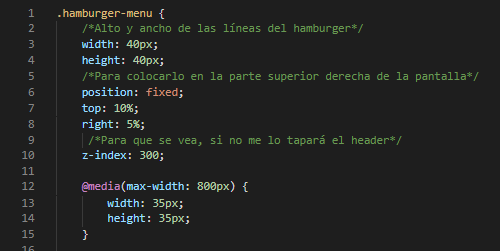




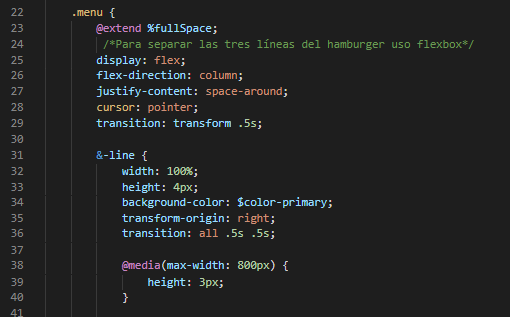
Por último, modificaré también la sección About us (fichero \_layout.scss):



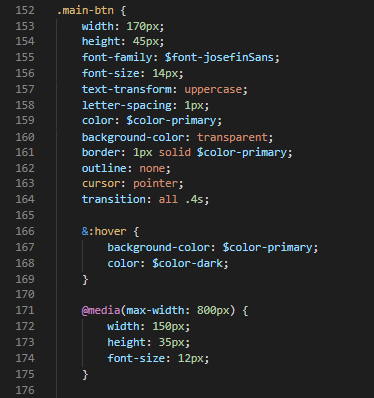
El siguiente punto de corte que encuentro es 800px, aquí voy a cambiar el hamburger menú:



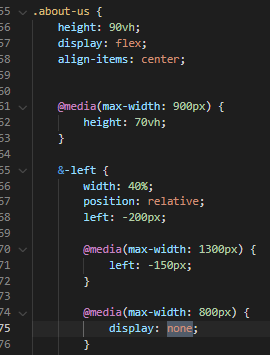
y la altura de las líneas:



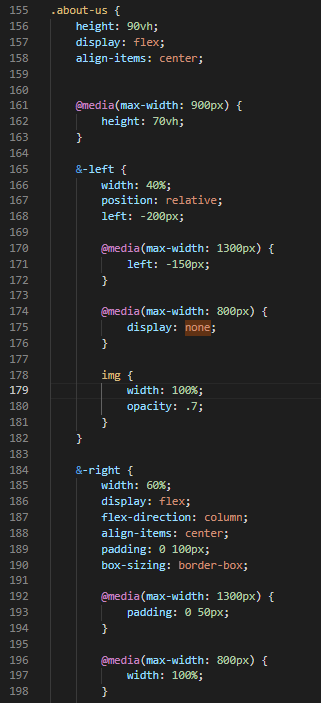
Voy a cambiar también el tamaño del botón:



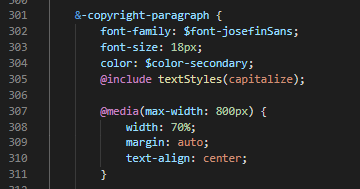
En la sección About us voy a esconder la imagen que aparece a la izquierda:

ç

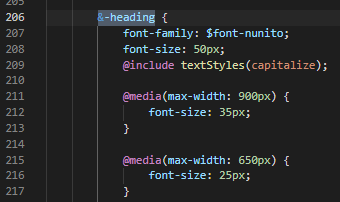
Además, quiero que la parte derecha tome el 100%:

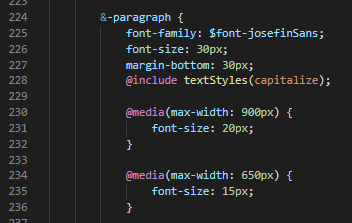


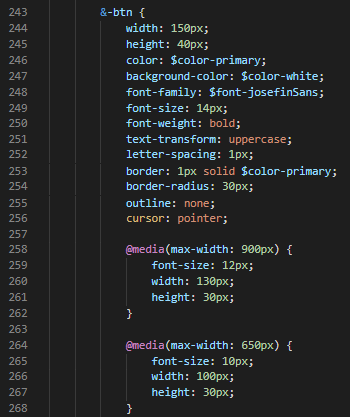
El paragraph del footer lo voy a centrar y hacerlo más pequeño (Fichero \_layout.scss):



El siguiente punto de corte que veo adecuado establecer es 650px en el overlay de las imágenes (fichero \_components.scss):

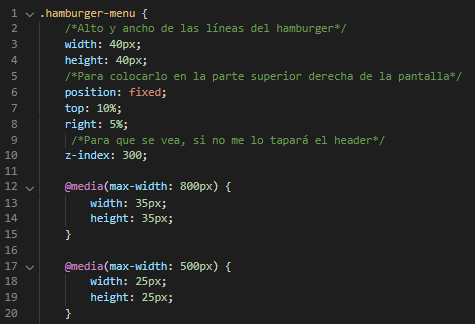




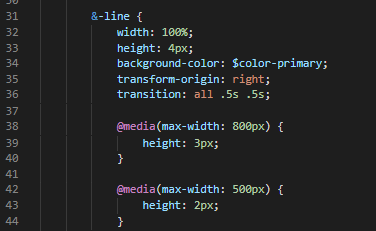


Ahora busco el siguiente punto de corte que será 500px.

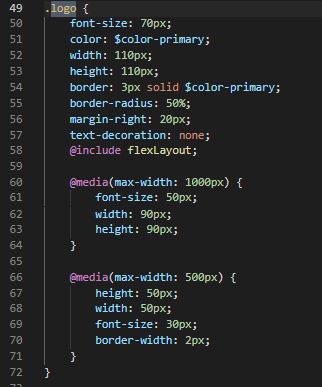
Comenzaré con el icono del menú hamburger:



También voy a cambiar el tamaño de las líneas:



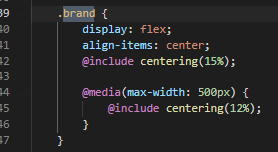
El tamaño del logo también se cambiará\_



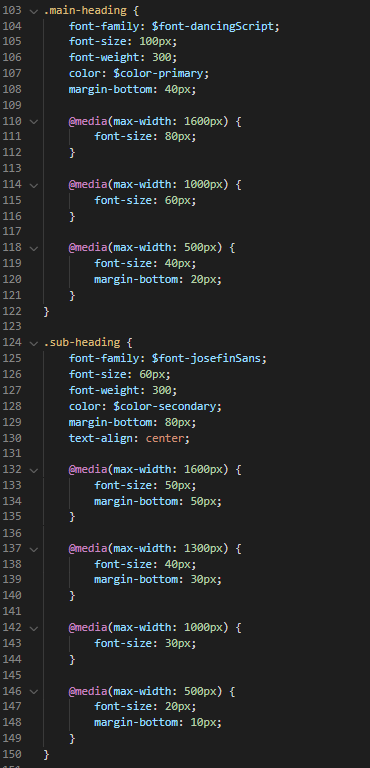
Ahora cambiaré el main-name y el sub-name:



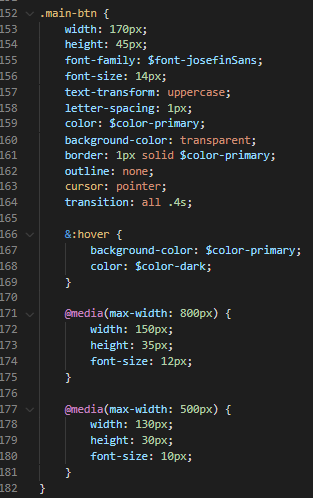
Voy a mover, además, esos elementos un poco más arriba (fichero \_layout.scss):



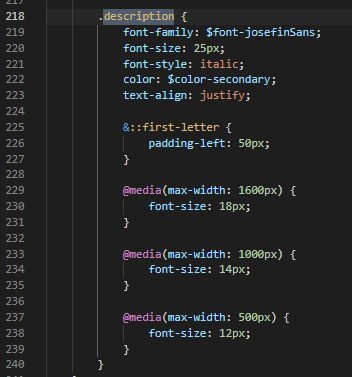
Ahora cambiaré el main-heading y el sub-heading:



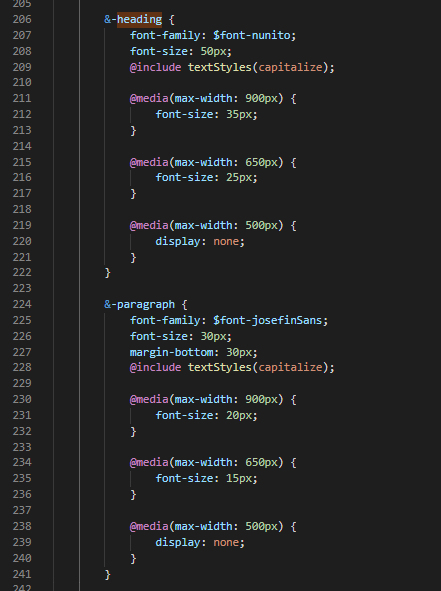
Cambiaré también el tamaño del botón (\_component.scss):



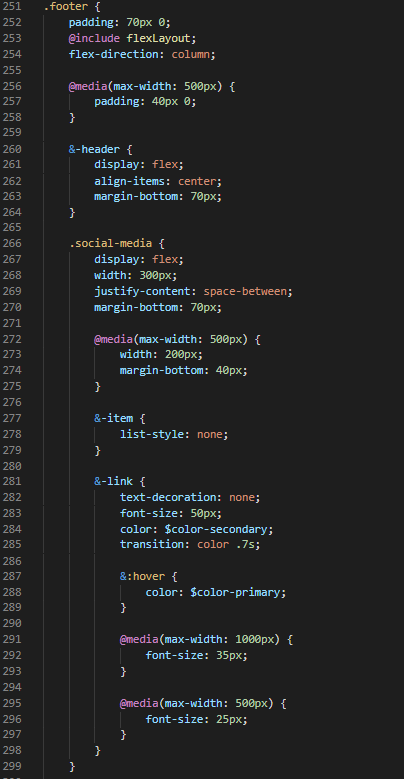
Y por último, el tamaño de la letra de la descripción (\_layout.scss):

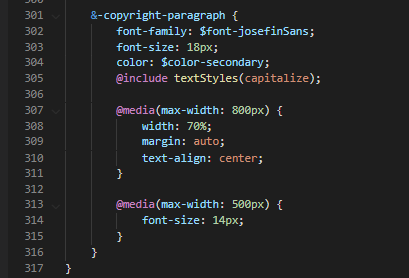


En el tamaño de 500px voy a ocultar el heading y el paragraph del overlay, ya que ocupa demasiado (\_components.scss):

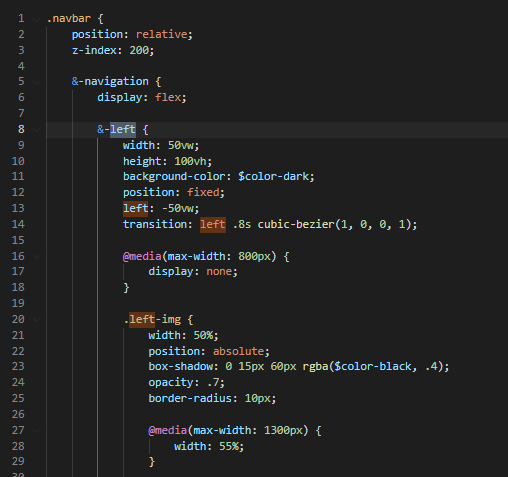


En el footer modificaré el padding, además en los iconos de las redes sociales le modificaré el tamaño y, por último, el tamaño del paragraph de los enlaces:





Ahora nos quedaría hacer el navbar responsive. Vamos a empezar por tamaños de pantalla grandes y vamos a ir reduciendo. (fichero \_layout.scss):



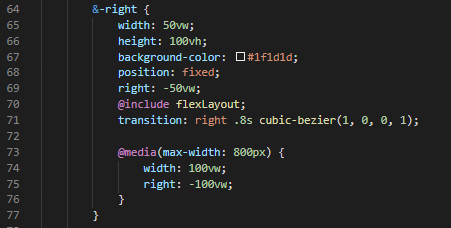
El siguiente punto de corte veo que es 1000px, punto en el que debería de aumentar un poco más el tamaño de las imágenes:



Ahora las voy a mover un poco a la izquierda:

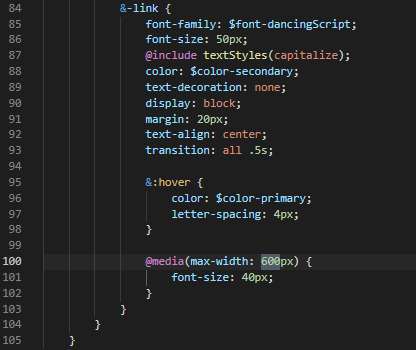


El siguiente punto de corte sería 800px. En ese punto quiero que solo se vea el menú y las imágenes desparezcan:



Lo último que voy a hacer es disminuir el tamaño de la letra del menú de navegación.

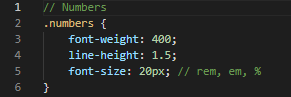
Estableceré el siguiente punto de corte el 600px:

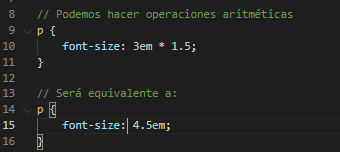


# Tipos de datos en SASS

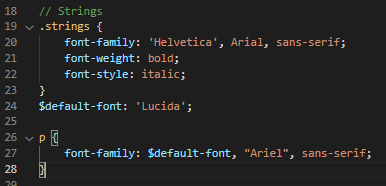
Tenemos 7 tipos de datos en Sass

Numbers: SASS soporta tanto números enteros como con decimales:

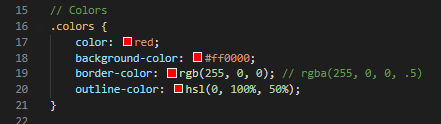




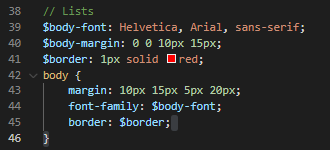
Strings: Se pueden crear con comillas dobles o simples



Colors: Sass soporta las mismas expresiones que CSS: valores hexadecimales como #0000, nombres de colores, expresiones rgb del tipo rgb(255, 0, 0), expresiones rgba del tipo rgba(100, 39, 153, 75), expresiones hsl y hsla.



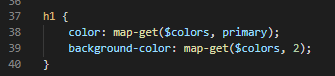
Lists: Son el equivalente a los arrays:



Maps: Son pares clave-valor:



Los maps no son traducidos inmediatamente a CSS, sino que se utilizan con las funciones map para recuperar valores:

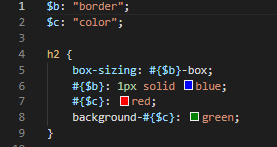


Booleans: Son valores que se utilizan para hacer comparaciones. Lo veremos cuando lleguemos a la directiva if.

Null: Se usa sobre todo para inicializar variables en mixins.

# Sass Interpolation

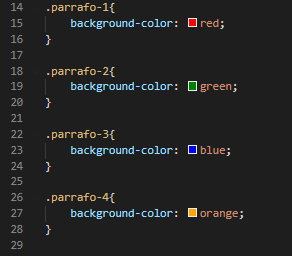
Se trata de sustituir el resultado de una expresión o el contenido de una variable y embeberlo en CSS.



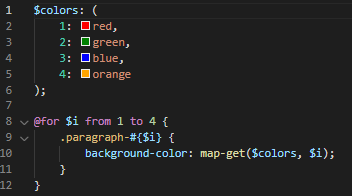
Es muy útil cuando se usa con bucles (loops).

# For Loop

Un código así en CSS:

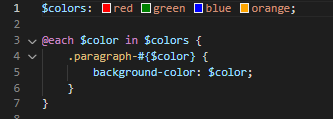


Se podría expresar el SASS de la siguiente forma:

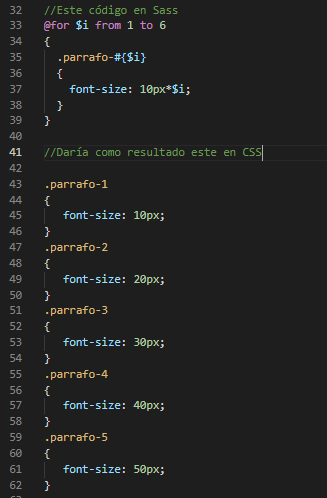


# Each Loop

Es un bucle que recorre todos los elementos de una lista:



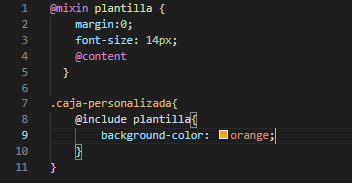
Este otro ejemplo:



# @content

La directiva @content nos permite inyectar reglas dentro de un mixin, de esta forma conseguimos reutilizar código y personalizarlo añadiendo otras reglas.

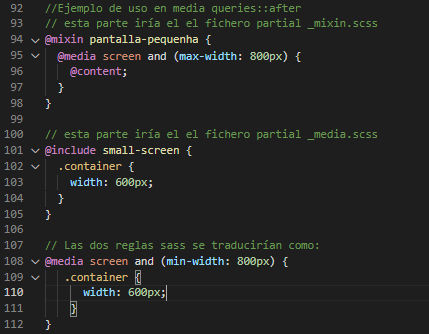
Ejemplo:



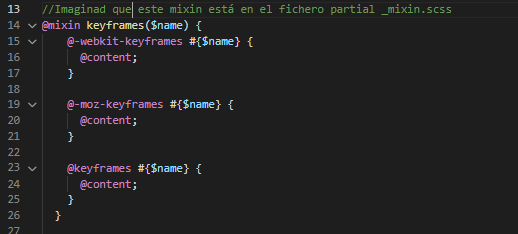
Nueva regla que se añade (se inyecta) al mixin plantilla

Usos más comunes de @content:

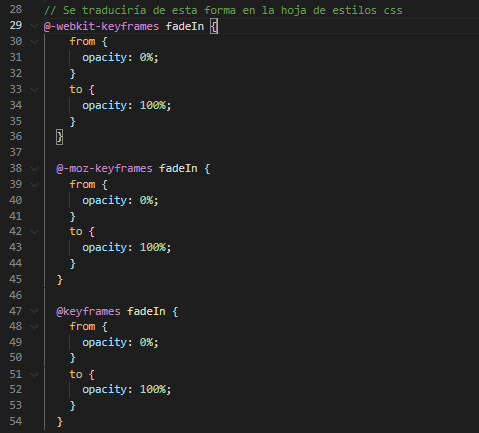
* En media-queries:



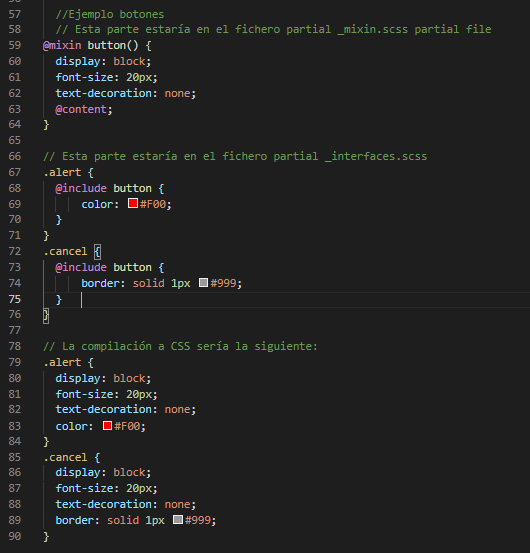
* Cuando tengamos que usar prefijos para que algunas propiedades nos funcionen en todos los navegadores, por ejemplo, en el caso de keyframes:



Se traduciría a CSS de esta forma:



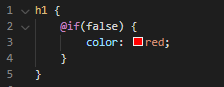
Cuando tengamos unos estilos base y queramos añadir otros, por ejemplo, tenemos los estilos base de un botón y queremos añadir otro en función de que el botón sea de tipo alert o de tipo cancel:



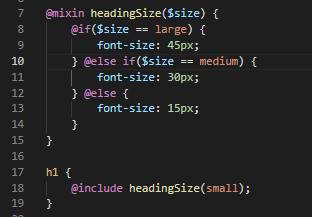
reglas que añadimos o inyectamos a los estilos base usando @content

# If directive

Permite aplicar unos estilos u otros en función de que se cumplan determinadas condiciones:

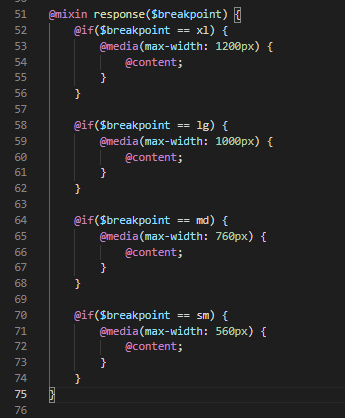


Usadas en un mixin:



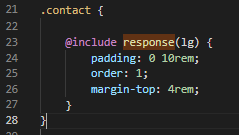
Se puede usar, por ejemplo, para crear media-queries.

En el fichero partial \_mixins.scss tendríamos este mixin:

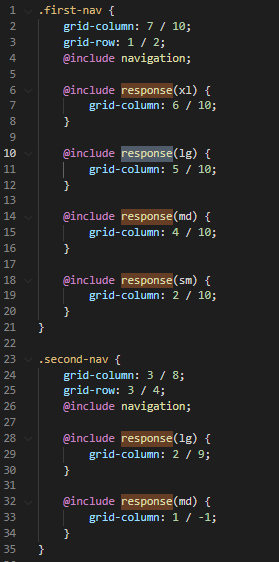


**Ventaja**: Se trata de tener las media-queries parametrizadas, de tal forma que las tenemos centralizadas en este fichero. Así, si modificamos el tamaño de una media query, no tenemos que ir cambiando el breakpoint en todas y cada una de las reglas donde la usamos.

Que lo usaríamos, por ejemplo, en el partial \_footer.scss:



O en el partial \_navigation.scss:



# Arquitectura

La arquitectura de un proyecto es uno de los aspectos cruciales a la hora de garantizar la mantenibilidad y escalabilidad del mismo. Uno de los beneficios de SASS es el ser capaz de dividir el proyecto en varios archivos y usar la directiva @import para importarlo en otros. Hay [diferentes arquitecturas](https://www.sitepoint.com/look-different-sass-architectures/).

No se puede establecer que una arquitectura es mejor que otra, la elección dependerá del tipo de proyecto y de nuestras preferencias, lo que sí que debemos hacer es ser consistentes con la elección de la misma.

La que aquí seguiré es la propone Jonathan Snook en su libro Scalable and Modular Architecture for CSS.

## Componentes

Debemos pensar nuestro proyecto como una colección de componentes. Los componentes pueden ser cualquier cosa que queramos pero deben de cumplir las siguientes condiciones:

* Que haga solo una cosa.
* Que sea reutilizable y de hecho se reutilice en el proyecto
* Que sea independiente.

Un componente puede ser un formulario, una barra de menú, un botón, una sección, etc.

### Estructura de un componente

Los componentes deben de ir en su propia carpeta components/

Si quieres que puedan ser personalizados externamente debes limitarte a estilos estructurales tales como tamaños (ancho/alto), padding, margins, alineación, etc. evitando propiedades tales como colores, sombras, fuentes, etc.

Debes evitar importar componentes en otros componentes para evitar que tengas demasiadas dependencias entre los mismos y el mantenimiento termine siendo imposible.

## El patrón 7-1

Este patrón establece que debes tener todas las partes almacenadas en 7 carpetas diferentes y un único archivo raíz (normalmente se llama main.scss) que importa el resto de los componentes. Las carpetas serían las siguientes:

* base/
* components/
* layout/
* pages/
* themes/
* abstracts/
* vendors/

Para usar este patrón puedes reutilizar [esta plantilla](https://github.com/KittyGiraudel/sass-boilerplate).

Un ejemplo de este patrón podría ser:

sass/

|

|– base/

| |– \_reset.scss # Reset/normalize

| |– \_typography.scss # Reglas tipográficas

| … # Etc.

|

|– components/

| |– \_buttons.scss # Botones

| |– \_carousel.scss # Carousel

| |– \_cover.scss # Cubierta

| |– \_dropdown.scss # Dropdown

| … # Etc.

|

|– layout/

| |– \_navigation.scss # Navegación

| |– \_grid.scss # Sistema de retícula

| |– \_header.scss # Encabezamiento

| |– \_footer.scss # Pie de página

| |– \_sidebar.scss # Barra lateral

| |– \_forms.scss # Formularios

| … # Etc.

|

|– pages/

| |– \_home.scss # Estilos específicos para la página de inicio

| |– \_contact.scss # Estilos específicos para la página de contacto

| … # Etc.

|

|– themes/

| |– \_theme.scss # Tema por defecto

| |– \_admin.scss # Tema del administrador

| … # Etc.

|

|– utils/

| |– \_variables.scss # Variables Sass

| |– \_functions.scss # Funciones Sass

| |– \_mixins.scss # Mixins Sass

| |– \_helpers.scss # Clases & placeholders

|

|– vendors/

| |– \_bootstrap.scss # Bootstrap

| |– \_jquery-ui.scss # jQuery UI

| … # Etc.

|

|

`– main.scss # Archivo principal de Sass

### Carpeta base/

La carpeta base/ contiene lo que podríamos llamar los estilos base del proyecto. Aquí deberíamos meter cosas como el archivo reset para reiniciar los estilos CSS, algunas reglas tipográficas y probablemente un archivo CSS que define algunos estilos estándares para los elementos HTML más comunes (que normalmente se suele llamar \_base.scss):

* \_base.scss
* \_reset.scss
* \_typography.scss

### Carpeta layout/

Esta carpeta contendrá todo lo relativo a la distribución de los componentes en el sitio (header, footer, sidebar, barras de navegación, asides etc.), el sistema CSS grid. Ejemplo de archivos:

* \_grid.scss
* \_header.scss
* \_footer.scss
* \_sidebar.scss
* \_forms.scss
* \_navigation.scss

### Carpeta components/

Aquí guardaríamos los archivos con estilos relacionados con los widgets (un carousel, un loader, etc.). Normalmente esta carpeta tendrá muchos ficheros, ya que una aplicación con una buena arquitectura debería estar compuesta por muchos widgets:

* \_media.scss
* \_carousel.scss
* \_thumbnails.scss

### Carpeta pages/

Aquí irían archivos con estilos para páginas que lleven estilos diferentes al resto del sitio, por ejemplo, la página “home” puede tener estilos diferentes al resto de las páginas del sitio. Por tanto, nos podremos encontrar archivos con los siguientes nombres:

* \_home.scss
* \_contact.scss

### Carpeta themes/

En sitios grandes es habitual encontrarse con diferentes temas, aquí irían los archivos dedicados a los mismos:

* \_theme.scss
* \_dashboard.scss

Para sitios pequeños es posible que no nos encontremos nada en esta carpeta o que la misma sea inexistente.

### Carpeta abstracts/

En esta carpeta encontraremos todas las herramientas, utilidades y funciones helpers de Sass que se usan a lo largo del proyecto:

* \_variables.scss
* \_mixins.scss
* \_functions.scss
* \_placeholders.scss

### Carpeta vendors/

Si estamos usando algún tipo de librería externa (Boostrap, Normalize, Font Awesome, etc.) este sería el sitio donde deberían ir los archivos CSS de las mismas:

* \_normalize.scss
* \_bootstrap.scss

Si quisiéramos sobreescribir algunas de estas librerías con estilos propios se suele crear otra carpeta con el nombre vendors-extensions/ donde ponen los mismos nombres de archivos que el desarrollador original les dio. Esto se hace para evitar modificar los archivos originales, lo cual es considerado una mala práctica.

### Archivo principal main.scss

Este archivo no debería de contener más que @import y comentarios.

Los archivos deben de importarse según al carpeta que los contiene, una tras otra en este orden:

1. abstracts/
2. vendors/
3. base/
4. layout/
5. components/
6. pages/
7. themes/

Las reglas que debe seguir este archivo deben ser las siguientes:

* Un archivo para cada import.
* Un @import por línea.
* Los archivos que pertenecen a la misma carpeta no deben tener entre ellos ninguna línea en blanco.
* Dejar una línea en blanco después del último archivo de una carpeta.
* No se deben poner las extensiones ni los guiones bajos.
* @import 'abstracts/variables';
* @import 'abstracts/functions';
* @import 'abstracts/mixins';
* @import 'abstracts/placeholders';
* @import 'vendors/bootstrap';
* @import 'vendors/jquery-ui';
* @import 'base/reset';
* @import 'base/typography';
* @import 'layout/navigation';
* @import 'layout/grid';
* @import 'layout/header';
* @import 'layout/footer';
* @import 'layout/sidebar';
* @import 'layout/forms';
* @import 'components/buttons';
* @import 'components/carousel';
* @import 'components/cover';
* @import 'components/dropdown';
* @import 'pages/home';
* @import 'pages/contact';
* @import 'themes/theme';
* @import 'themes/admin';

Breakpoints

Nombrar los puntos de ruptura

Es una buena costumbre poner nombre a las media queries, pero los nombres no deberían de ir vinculados a dispositivos concretos, sino a tamaños de pantalla, por ejemplo:

$breakpoints: (

'medium': (min-width: 800px),

'large': (min-width: 1000px),

'huge': (min-width: 1200px),

);

Breakpoint manager

Se trata de usar un mapa de breakpoints asociado a nombres:

$breakpoints: (

'small': 800px,

'medium': 1000px,

'large': 1200px,

);

Posteriormente el mixin usaría ese map:

/// Mixin to manage responsive breakpoints

/// @author Kike Sainz

/// @param {String} $breakpoint – nombre Breakpoint

/// @require $breakpoints

@mixin respond-to($breakpoint) {

  // Si la clave existe en el mapeo

  @if map-has-key($breakpoints, $breakpoint) {

    // Imprime una media querie basada en el valor

    @media (min-width: map-get($breakpoints, $breakpoint)) {

      @content;

    }

  }

  // Si la clave no existe en el mapeo

  @else {

    @warn "Error, no se ha podido recuperar el valor para `#{$breakpoint}`. "

        + "Los valores disponibles son: #{map-keys($breakpoints)}.";

  }

}

¿Dónde situar las media queries?

En esto hay diferentes opiniones. Hay desarrolladores que son partidarios de colocarlos dentro de los selectores y otros que prefieren colocarlos fuera. Un ejemplo de cómo colocarlos dentro sería:

.selector {

  color: red;

  @include respond-to('medium') {

    color: blue;

  }

}

# Enlaces, recursos y bibliografía:

Raghu, V. (2014, 19 agosto). A Look at Different Sass Architectures. Sitepoint. Recuperado 24 de enero de 2014, de <https://www.sitepoint.com/look-different-sass-architectures/>

Snook, J. (2019). Scalable and Modular Architecture for CSS: http://smacss.com/