UNIDAD 5

PREPROCESADORES CSS Y FRAMEWORKS DE DISEÑO

PREPROCESADORES

Índice

1 .- Introducción

Las páginas web están formadas por varios elementos separados, por un lado está el código HTML que forma el contenido de la página con las referencias a elementos externos que use como hojas de estilo CSS, imágenes o archivos JavaScript.

Las imágenes son recursos gráficos soportando todos los navegadores los formatos *png*, *jpg* para fotos e imágenes vectoriales como *svg*.

Los archivos JavaScript permite incorporar a la página código ejecutable con el propósito de añadir ciertos comportamientos mediante programación.

Finalmente, están las hojas de estilo CSS que proporcionan los estilos y cambian el aspecto visual y maquetación de los elementos, el contenido HTML puede ser el mismo pero con una hoja de estilos diferente la visualización muy diferente.

Las hojas de estilos cambian propiedades de visualización de diferentes elementos como tipo de letra, tamaño de texto, formato, color, alineación, bordes, etc…

Siendo las páginas web cada vez más complejas, en gran medida las hojas de estilo también se convierten en más complejas con la aparición de los dispositivos móviles un poco más dado que hay que adaptar la página al tamaño de cada uno de los dispositivos. Por esto se suelen usar *toolkits* o librerías como Bootstrap con unos estilos elegantes listos para usar sin tener que crearlos desde cero en cada proyecto, proporciona diferentes componentes como *layouts*, botones, formularios, modales, *popovers*, barras de progreso, desplegables y algunos más. Sin embargo, aún con Bootstrap suelen ser necesarios escribir algunos estilos adicionales propios de la página.

2 .- Preprocesadores CSS

Un preprocesador de CSS se puede definir como una herramienta que nos permite escribir pseudocódigo CSS que luego será compilado de convertir en CSS tal y como lo conocemos de forma habitual. Este pseudocódigo está formado por variables, condiciones, bucles o funciones, elementos habituales de cualquier lenguaje de programación. Por este motivo, podríamos decir que tenemos un lenguaje de programación cuya misión es la de generar el código CSS

Los preprocesadores CSS facilitan la escritura y mantenimiento de las hojas de estilos con funcionalidades que CSS no tiene pero esto no hace del CSS resultante generado mejor que si estuviese escrito directamente sin utilizar un preprocesador. Por ejemplo, no conviene crear muchos niveles de anidación ya que el CSS generado será más grande y más costoso de aplicar al navegador. Para desarrollar estilos aplicables a elementos HTML que sean reutilizables y más fácilmente mantenibles hay que emplear alguna de las metodologías más aceptadas que proponen buenas prácticas para la escritura de CSS, algunas de estas metodologías son OOCSS, BEM y SMACSS.

Puede que al principio sea más complicado al tener que aprender a utilizar preprocesadores, pero cuando lo conozcáis os daréis cuenta cómo su uso nos agiliza el desarrollo de los proyectos.

2.1 .- Principales beneficios de utilizar preprocesadores CSS

El uso de preprocesadores en nuestros proyectos aporta una serie de beneficios entre los que podemos destacar:

- Posibilidad de añadir funcionalidades adicionales que no son posible de utilizar en archivos CSS tradicionales, como son el uso de variables y lógica condicional
- Dispondremos de una hoja de estilos más limpia, consiguiendo a la vez que sea más eficiente y fácil de mantener
- Ofrece la posibilidad de reutilizar mucho código, lo que se traduce en un ahorro de tiempo y trabajo
- Los cambios serán mucho más rápidos, ya que únicamente deberemos cambiar el valor de alguna variable
- Ayuda a crear código compatible con todos los navegadores, lo que supone una gran ayuda para los diseñadores de sitios web
- Se crea una capa de abstracción, donde no trabajaremos directamente sobre nuestro archivo CSS, ofreciendo una mayor seguridad
- Posibilidad de separación absoluta del proceso de desarrollo y producción

2.2 .- Variables

Una de las principales características de los preprocesadores es que permiten el uso de variables. Dentro de estas variables, podremos almacenar valores que luego reutilizar en cualquier parte del código. Gracias a esta reutilización ahorraremos mucho trabajo cuando tengamos que modificar un valor que se repite a lo largo de nuestro código. Por ejemplo, podríamos tener declaradas las siguientes variables:

```
$color_principal: #fff;
$color_secundario: #2BA47D;
$tamano_texto: 30px;
body {
color: $color_principal;
background: $color_secundario;
font-size: $tamano_texto;
}
```

Como vemos, luego esas variables pueden ser utilizadas dentro de bloques de estilos. El código anterior una vez compilado daría como resultado lo siguiente:

```
body {
color: #fff;
background: #2BA47D;
font-size: 30px;
```

```
}
```

2.3 .- Funciones, mixins y prefixing

Como ocurre con cualquier lenguaje de programación, el uso de funciones permite evitar la escritura de código repetido. Esto mismo ocurre con los preprocesadores de CSS donde podemos crear una función que contenga una serie de características que se utilizan en varias partes de la hoja de estilos.

Un ejemplo de esto que estamos diciendo sería el siguiente:

```
mixin box-sizing(valor) {
-webkit-box-sizing: valor;
-moz-box-sizing: valor;
box-sizing: valor;
}
```

Ahora, si debemos utilizar de nuevo esta serie de propiedades, lo que deberíamos hacer sería la llamada a esa función, tal y como se muestra en el siguiente ejemplo:

```
div {
box-sizing(border-box);
}
```

El código anterior sería lo mismo que escribir en CSS tradicional lo siguiente:

```
div {
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
}
```

Aunque pueda parecer cosa de poco, en un proyecto con un elevado número de líneas de código puede significar el ahorro de muchas de ellas.

2.4 .- Modulariza el código

Otra de las grandes características de los procesadores es la posibilidad de importar archivos. En vez de utilizar un único CSS o tener varios en nuestro proyecto, lo que tendremos son varios archivos preprocesables que darán como resultado uno solo. Gracias a esta característica podremos tener organizado de una forma lógica y sencilla el proyecto en nuestro alojamiento web, teniendo distintos módulos para cada elemento de la web.

Por poner un ejemplo, podemos tener un archivo que se encargue de los estilos de la parte pública, otro de la intranet de los usuarios y un último que sea el que contenga todas las instrucciones para dar estilos a la administración del sitio.

Gracias a esta organización podremos hacer los cambios de forma más rápida, ya que tendríamos que buscarlo en el archivo correspondiente, en vez de tener que localizarlo en el interior de un gran archivo.

3 .- Sass y Scss

Cualquier preprocesador es perfectamente válido. Podríamos sin duda elegir otras alternativas como Less o Stylus y estaría estupendo para nosotros y nuestro proyecto, ya que al final todos ofrecen más o menos las mismas utilidades. Pero sin embargo, Sass se ha convertido en el preprocesador más usado y el más demandado.

Al haber recibido un mayor apoyo de la comunidad es más factible que encuentres ofertas de trabajo con Sass o que heredes proyectos que usan Sass, por lo que generalmente te será más útil aprender este preprocesador.

Además, varios frameworks usan Sass, como el caso de Bootstrap. Por ello, si tienes que trabajar con ellos te vendrá mejor aprender Sass.

3.1 .- Sintaxis

Como decimos, la tarea de aprender Sass se divide en dos bloques principales. Primero aprender a trabajar con el código de Sass y conocer su sintaxis especial. La segunda tarea sería aplicar Sass en tu flujo de trabajo, de modo que puedas usar el preprocesador fácilmente y no te quite tiempo el proceso de compilación a CSS.

Lo primero que debes conocer de Sass es que existen dos tipos de sintaxis para escribir su código:

- **Sintaxis Sass**: esta sintaxis es un poco diferente de la sintaxis de CSS estándar. No difiere mucho. Por ejemplo, te evita colocar puntos y coma al final de los valores de propiedades. Además, las llaves no se usan y en su lugar se realizan indentados.
- **Sintaxis SCSS**: Es una sintaxis bastante similar a la sintaxis del propio CSS. De hecho, el código CSS es código SCSS válido. Podríamos decir que SCSS es código CSS con algunas cosas extra.

En la práctica, aunque podría ser más rápido escribir con sintaxis Sass, es menos recomendable, porque te aleja más del propio lenguaje CSS. No solo es que te obligue a aprender más cosas, sino que tu código se parecerá menos al de CSS estándar y por lo tanto es normal que como desarrollador te sientas menos "en casa" al usar sintaxis Sass. En cambio, al usar SCSS tienes la ventaja de que tu código CSS de toda la vida será válido para el preprocesador, pudiendo reutilizar más tus conocimientos y posibles códigos de estilo con los que vengas trabajando en los proyectos.

Como entendemos que al usar un preprocesador es preferible mantenerse más cerca del lenguaje CSS, en este caso usaremos siempre sintaxis SCSS.

3.2 .- Instalación

La instalación de Sass depende del sistema operativo con el que estás trabajando. Aunque todos requieren comenzar instalando el lenguaje Ruby, ya que el compilador de Sass está escrito en Ruby.

Veamos, para cada sistema, cómo hacernos con Ruby.

- **Windows**: Existe un instalador de Ruby para Windows. https://rubyinstaller.org/ Puedes usarlo para faciltar las cosas.
- **Linux**: tendrás que instalar Ruby usando tu gestor de paquetes de la distribución con la que trabajes, apt-get, yum, etc.
- **Mac**: Ruby está instalado en los sistemas Mac, por lo que no necesitarías hacer ningún paso adicional.

Una vez instalado el lenguaje Ruby tendremos el comando "gem", que instala paquetes (software) basados en este lenguaje. Así que usaremos ese comando para instalar Sass. De nuevo, puede haber diferencias entre los distintos sistemas operativos.

Windows

Tendrás que abrir un terminal, ya sea Power Shell, "cmd" o cualquiera que te guste usar. Si no usas ninguno simplemente escribe "cmd" después de pulsar el botón de inicio. Luego tendrás que lanzar el comando.

```
gem install sass
```

Linux

Para instalar Sass, una vez tienes Ruby anteriormente, podrás hacerlo con el siguiente comando en tu terminal.

```
sudo gem install sass --no-user-install
```

Mac OS X

En Mac usarás generalmente "sudo", igual que en Linux. Aunque puedes probar antes sin sudo y usar sudo si realmente te lo pide la consola por no tener los suficientes permisos.

```
sudo gem install sass
```

Nota importante para usuarios de Mac: En Mac OS X es posible que tengas que hacer algunas otras operaciones. En una instalación limpia de Mac quizás tengas que instalar las extensiones de Xcode:
xcode-select --install
O bien instalar el propio Xcode desde el App Store (no te preocupes, porque es gratuito). Luego además tendrás que abrir el Xcode, una vez instalado, para completar la instalación de todos los componentes necesarios.

Para cualquier sistema, una vez instalado Sass, podemos ver si realmente está disponible, haciendo el siguiente comando:

```
sass -v
```

Eso nos debería devolver la versión de Sass instalada en nuestro sistema.

```
❯ sass -v
Sass 3.5.5 (Bleeding Edge)
```

Figura 1: Verificar la versión de Sass

3.2.1 .- Live Sass Compiler. Integración con Visual Studio Code

Esta extensión nos ayuda a complilar nuestros archivos sass/scss a código css en tiempo real sin necesidad de depender de otras aplicaciones que nos limiten el tiempo de uso o requieran de una licencia.

Para instalar Live Sass es muy sencillo, hay que seguir los mismos pasos que realizamos anteriormente con Live Server:

1. Ubicar el icono de extensiones de Visual Studio Code.
2. Escribir en el buscador de extensiones Live Sass.
3. Dar click en “instalar”.

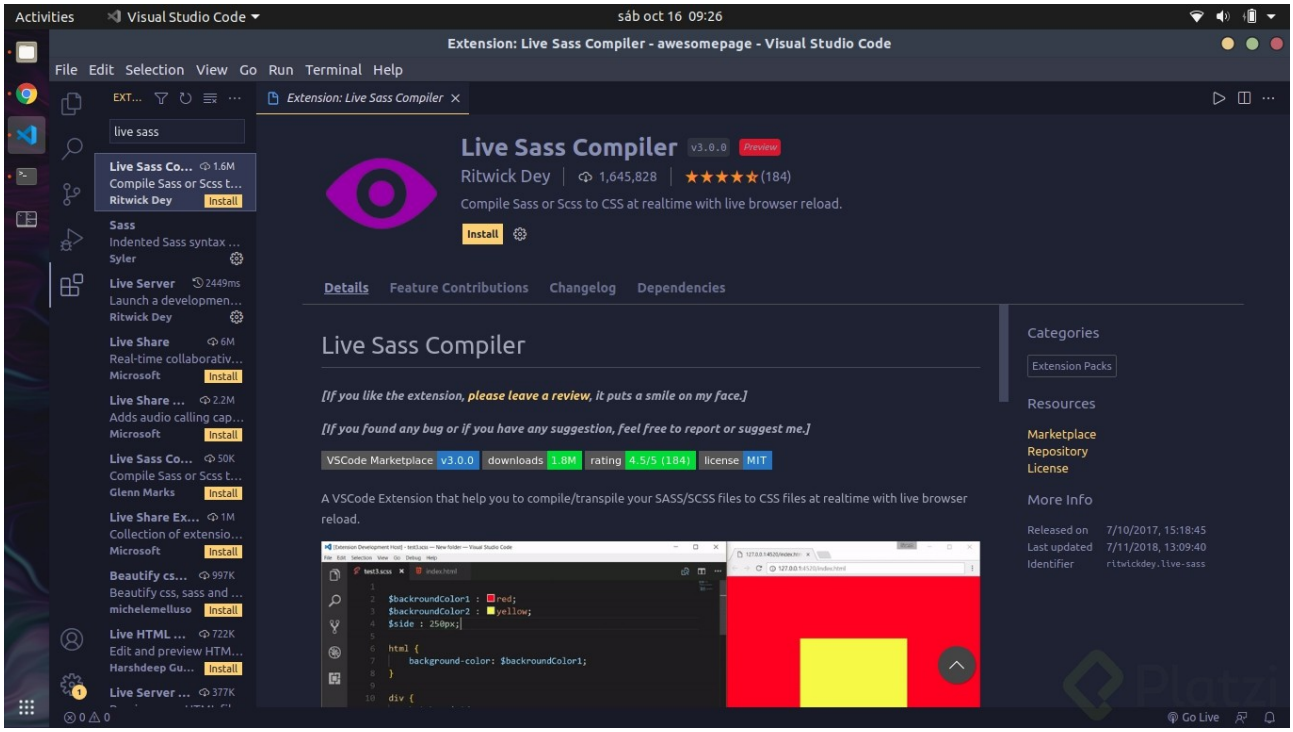


Figure 2: Instalación de Live Sass Compiler

Al igual que con Live Server, al abrir un archivo de Sass, notarás un icono en la parte inferior derecha que dice “Watch Sass” el cual activa la extensión.

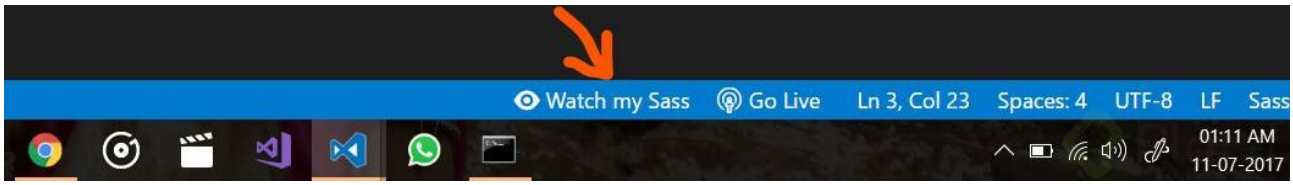


Figure 3: "Watch Sass"

Al activarla se desplegará una terminal que te mostrara el log de lo que compila y estara atenta a los cambios que realizes en tus estilos.

Al estar atenta a los cambios, la extensión te notificará si todo estuvo correcto y compiló satisfactoriamente , de lo contrario te mostrará un error .

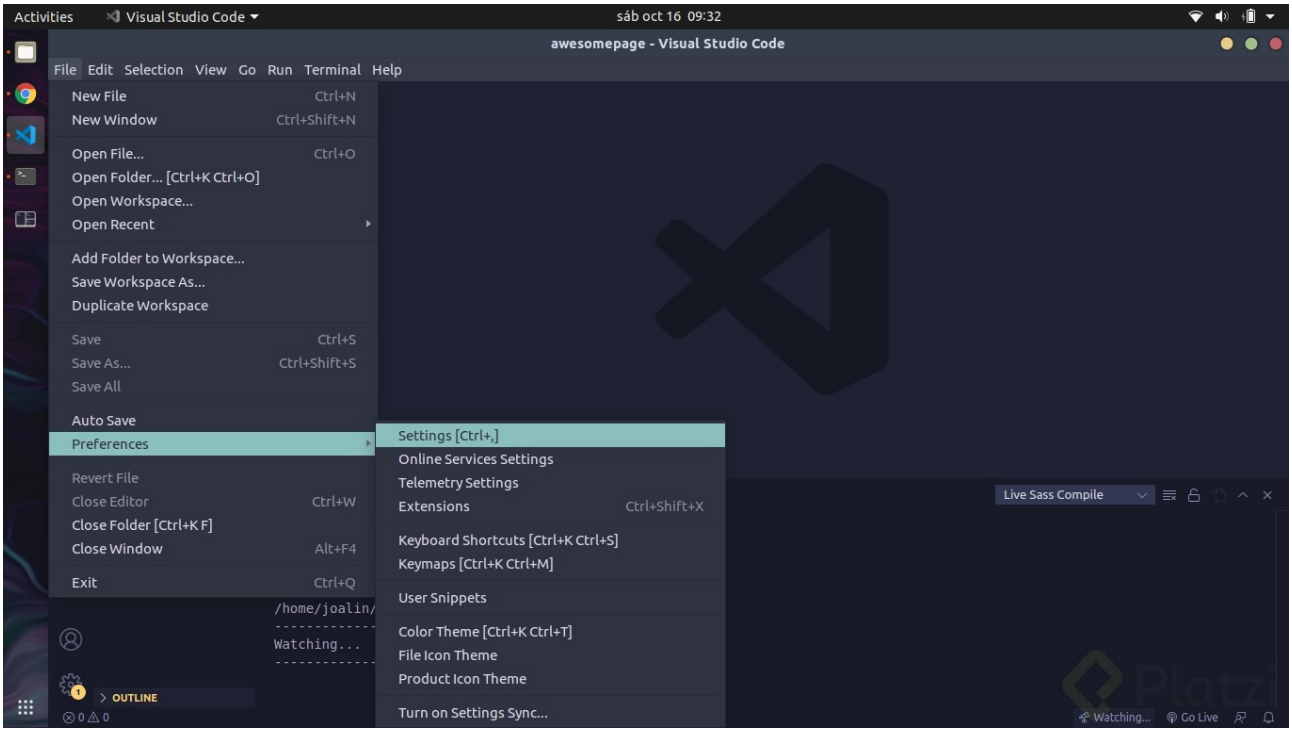
No te preocupes y recuerda que los errores son nuestros amigos, pues nos muestran en que nos equivocamos, Live Sass te mostrará donde esta el error.

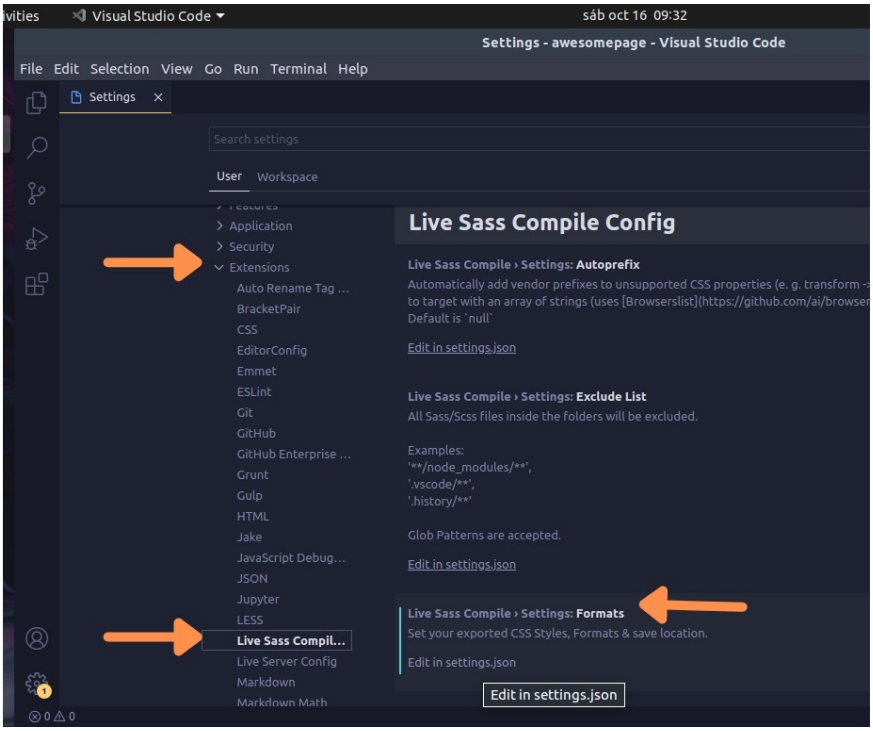
De esta forma no volverás a preocuparte por compilar tus archvos, Live Sass lo hará por ti!

3.2.1.1 Mantén ordenado tu código

Si quieres mantener el orden en tu código separado por carpetas, Live Sass nos permite configurar el path o ruta donde queremos que coloque los archivos de css que generó, para ello seguiremos los siguientes pasos:

1. Vamos a los Settings de Visual Studio Code
2. En Settings buscamos extensiones y dentro ubicamos Live Sass Compiler
3. Una vez ubicada la extensión, buscamos la parte de formats y damos click en editar json .



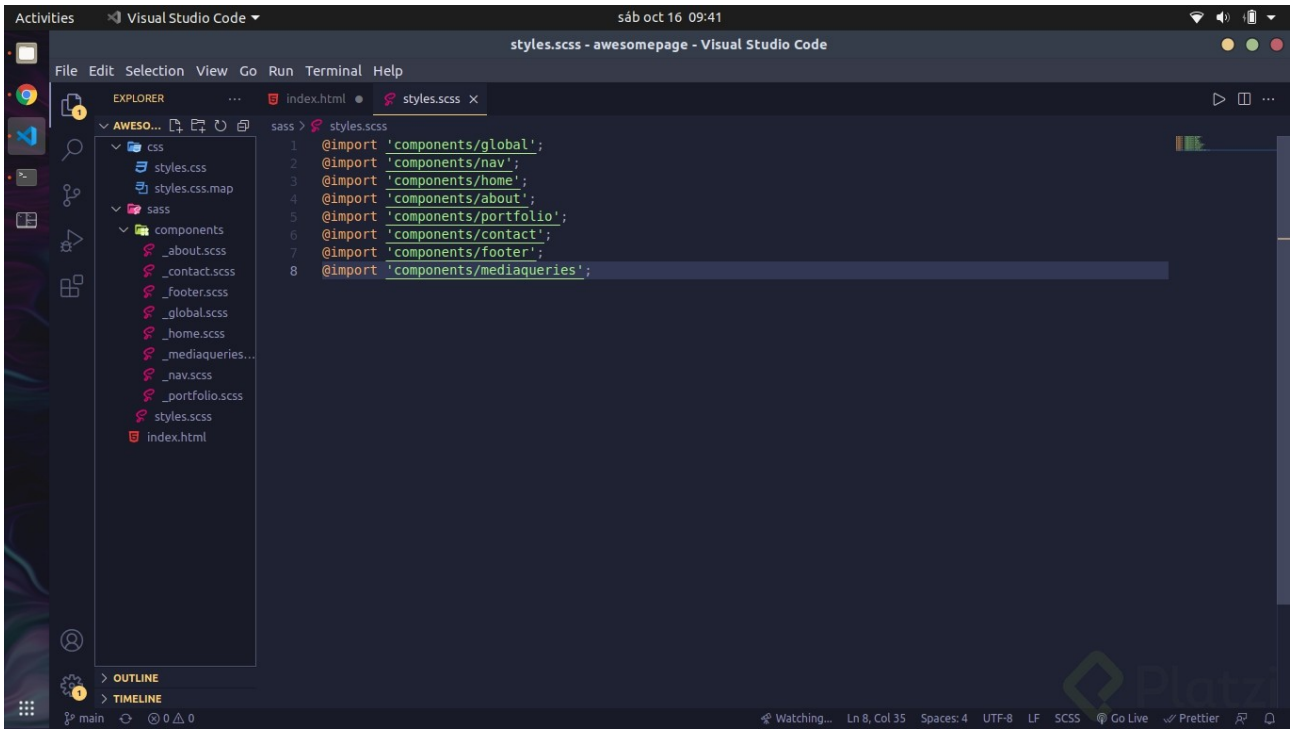


4. Esto nos abrirá un archivo donde estan las configuraciones de formato, extensión y ruta.
5. No te preocupes, solo realizaremos una configuración sencilla.

6. En "save path" colocaremos la ruta donde queremos que nos guarde los archivos css generados, si no existe esta carpeta en tu proyeto no te preocupes, la extensión la creará por ti, en mi caso quiero que se guarde en una carpeta llamada css, así que solo reemplazaré lo que viene por defecto a "./css", quedando de esta forma:

```
{
        "format": "expanded",
        "extensionName": ".css",
        "savePath": "./css"
  }
```

7. Guardamos los cambios y de esta manera, cuando trabajemos con la extensión, nos creará una carpeta aparte que contendrá los archivos de `css`, permitiendonos tener mejor ordenados nuestros archivos en nuestro proyecto.



8. Ahora solo recuerda que en tu `html` debes indicar la ruta exacta de donde se encuentra tu hoja de estilos.

3.3 .- Variables

Al igual que en la programación, puede crear e implementar variables en Sass en toda su hoja de estilo. Esto es extremadamente conveniente porque si hay un cierto estilo que desea usar en su sitio web (por ejemplo, un tipo de paquete de fuente, color, etc.), simplemente puede almacenarlo en una variable y usarlo en toda su hoja de estilo. Declarar una variable en Sass es especialmente sencillo para los programadores porque

es conceptualmente idéntico a Javascript. Por ejemplo, si declaras $primary-color: #525; ($ variable: value;) al comienzo de su hoja de estilo, entonces puede simplemente ingresar $primary-color para cualquier valor de color.

Esto es útil por al menos dos razones: 1. Siempre que desee reutilizar el mismo color u otro valor, simplemente se necesita reutilizar la variable, sin tener que recordar nunca estilos comunes. 2. Si se alguna vez necesita cambiar el valor de varios selectores, todo lo que se tiene que hacer es cambiar el valor de la variable.

```
$primary-color: #333;

.navbar {
  background-color: $primary-color;
}
.footer {
  border: 1px solid $primary-color;
}
```

Sass va a leer las variables y las va a sustituir por su respectivo valor para crear el archivo .css. Esto de las variables viene genial cuando quieres mantener una consistencia en el aspecto de la web que estás desarrollando. Una práctica muy habitual consiste en definir todos los colores de la aplicación en forma de variables sass para tenerlos bien localizados.

Aparte de poder poner colores puedes poner cualquier propiedad css que se te ocurra (distancias en píxeles, transiciones, números, etc)

3.4 .- Nesting. Anidación de estilos

Otra característica fundamental de sass es el nesting de variables, es decir, poder anidar propiedades unas dentro de otras ahorrando código. Veamos un ejemplo:

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

El código que ves arriba, al compilarse en código css dará como resultado:

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav a {
  display: block;
  padding: 6px 12px;
```

```
  text-decoration: none;
}
```

En este caso no te ahorras, muchas líneas, pero cuando tienes muchas anidaciones si que puede llegar a ser útil. Además si por ejemplo quieres quitar el selector del nav y quieres poner una clase, por ejemplo, te va a resultar mucho más fácil cambiarlo en un solo sitio.

Otra ventaja es que puedes hacer anidaciones unas dentro de otras, por ejemplo:

```
nav {
  .container {
    margin: 0;
    padding: 0;
    list-style: none;
    ul {
      display: block;
      padding: 6px 12px;
      text-decoration: none;
    }
  }
}
```

Además, si pones **&** también puedes hacer nesting de propiedades del selector en el que lo apliques como por ejemplo:

```
a {
  font-weight: bold;
  text-decoration: none;
  &:hover {
    text-decoration: underline;
  }
}
```

En este caso accedes al selector padre, es decir la etiqueta **a** en el estado **hover**

Que será compilado a:

```
a {
  font-weight: bold;
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
```

Como ves, con **&** referencias al objeto sobre el que lo aplicas.

Nota: La anidación es una herramienta muy útil para mantener el código fuente de la hoja de estilo ligero y eficiente. Con todo, la tentación de hacer un uso excesivo de ella es muy fuerte, llevando así a invertir su efecto y crear estructuras muy complejas que pueden acarrear problemas a la hora de realizar cambios o labores de mantenimiento. Así, conviene no pasar de un tercer nivel.

3.5 .- Importación de estilos

Como sabrás, CSS también tiene una manera de importar ficheros css unos dentro de otros, y sass hace lo mismo, la diferencia es que con el import de sass no se generan nuevas peticiones HTTP. Cada vez que importes un fichero, sass va a coger el contenido del fichero importado y lo va a combinar con el otro fichero:

```
@import 'base';

body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

Así de sencillo se importan ficheros con sass. Con esto vas a poder modularizar mucho el código de tus aplicaciones web. Puedes combinar esta importación de ficheros con las variables vistas anteriormente para tener un fichero localizado con todas las variables de la aplicación.

Extender

La regla extender permite ahorrar mucho trabajo. La directiva garantiza que todas las propiedades de una clase se transfieran a las de otra. Usando *@extend* se evita tener que redefinirlo todo. La directiva también funciona como una cadena. Una clase definida por *@extend* puede a su vez formar parte de una tercera clase:

```
.button-scope {
  margin: 5px;
  border-radius: 2px;
}
.home-button {
  @extend .button-scope;
  background-color: $black;
}
.back-button {
  @extend .home-button;
}
```

El compilador resuelve el código CSS de la siguiente manera:

```
.button-scope, .home-button, .back-button {
margin: 5px;
border-radius: 2px;
}

.home-button, .back-button {
background-color: #000000;
}
```

3.6 .- Mixins

La mayoría de las personas con experiencia en programación están familiarizadas con la escritura de funciones o métodos en sus programas; sin embargo, los archivos CSS

tradicionales no admiten esta función. Afortunadamente, Sass permite a los desarrolladores escribir mixins o un grupo de declaraciones CSS que se agrupan y se pueden usar repetidamente en cualquier lugar de su hoja de estilo. Un mixin es esencialmente el equivalente a una función; puede aceptar un valor de entrada y conectarlo a las declaraciones CSS agrupadas dentro del mixin. Para crear un mixin, simplemente escriba @mixin mixinName(value) seguido de las declaraciones CSS que utilizan el valor introducido. Para invocar este mixin en su hoja de estilo, escriba la palabra clave @include seguido del nombre y el valor del mixin (si corresponde). Sin duda, Mixins en Sass allana el camino para un diseño de sitios web potente y eficiente al crear un código menos repetitivo.

Los mixins ahorran mucho código, y además puedes meter parámetros a estas funciones para que se puedan personalizar para caso de uso, por ejemplo:

```
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}

.box { @include transform(rotate(30deg)); }
```

En este caso el mixin recibe un parámetro llamado property que indica la cantidad de rotación a aplicar. A continuación se llama a la función con el **@include** indicando el nombre de la función y pasando el parámetro. En este caso nos evita tener que escribir las propiedades para los distintos navegadores cada vez que queramos hacer una transformación.

Otro ejemplo: Con diferentes niveles de compatibilidad del navegador para diferentes propiedades CSS, es común tener que proporcionar diferentes prefijos de proveedores a las propiedades, a fin de garantizar la compatibilidad entre navegadores (especialmente cuando es necesario el soporte para Internet Explorer)

```
pags {
  -webkit-transform: escala (2);
      -ms-transform: escala (2);
          transformar: escala (2);
}
```

Esto puede ser una molestia tener que hacerlo repetidamente para cada propiedad con diferentes niveles de soporte. Usando un descaro **Mixin** puede simplificar el proceso siempre que tenga que repetir grandes secciones de código

Usando el código para el transform propiedad anterior como ejemplo, podemos convertirlo fácilmente en un mixin reutilizable siempre que queramos utilizar un transform: scale() regla:

```
@mixin transform-scale ($ x) {
    -webkit-transform: escala ($ x);
        -ms-transform: escala ($ x);
            transformar: escala ($ x);
}
```

Con el mixin anterior, definimos la cantidad que queremos escalar cada transform con $x argumento. Este mixin se puede utilizar en cualquier bloque de código Sass, utilizando el @include declaración de antemano

```
pags {
    @include transform-scale (2);
}
```

3.7 .- Interpolación

La interpolación se puede utilizar casi en cualquier sitio de una hoja de estilo Sass para embeber el resultado de una expresión sass en el código CSS. Solo envolviendo una expresión en #{} en alguno de los siguientes lugares:

- selectores
- nombres de propiedades en declaraciones
- valores de propiedades personalizadas
- Reglas arroba
- @extends
- CSS @imports
- cadenas de caracteres entrecomilladas o no
- funciones especiales
- nombres de funciones
- Comentarios

```
@mixin corner-icon($name, $top-or-bottom, $left-or-right) {
  .icon-#{$name} {
    background-image: url("/icons/#{$name}.svg");
    position: absolute;
    #{$top-or-bottom}: 0;
    #{$left-or-right}: 0;
  }
}

@include corner-icon("mail", top, left);
```

Genera el siguiente código CSS:

```
.icon-mail {
  background-image: url("/icons/mail.svg");
  position: absolute;
  top: 0;
  left: 0;
}
```

3.8 .- Bucles

Los bucles proporcionan al lenguaje de hojas de estilo el atractivo de un lenguaje de programación real. En SASS se utilizan para crear bloques de instrucciones que se repiten **hasta que tiene lugar una condición determinada**. Existen tres directivas diferentes para crear bucles:

- *@for*
- *@while*
- *@each*

3.8.1 .- For

@for es el bucle estándar en programación: comienza al inicio y repite la orden hasta que se alcanza un estado de salida y con ello el final. En SASS, esta directiva se presenta en dos variantes diferentes: o bien el último ciclo se ejecuta una vez más cuando se alcanza el objetivo o se abandona el bucle antes.

```
@for $i from 1 through 4 {
      .width-#{$i} { width: 10em + $i; }
}
@for $i from 1 to 4 {
      .height-#{$i} { height: 25em * $i; }
}
```

Tras la directiva primero se especifica una variable cualquiera ($i) y luego se define el punto de inicio (1) y el de destino (4). Con through se indica que la cuarta repetición también debe realizarse, mientras que con to el bucle se detiene después de la tercera vuelta. Si para el inicio se define un valor superior al elegido para el final, SASS cuenta hacia atrás. Hay dos elementos integrados en el bucle: primero se elige la notación de CSS que obtendrá una cifra superior con #{$i}: la variable –y, por lo tanto, también la notación– se incrementará en 1 con cada vuelta.

Recuerda: En SASS #{} es una interpolación. Con ella se puede combinar a una variable con un identificador que se ha asignado a sí mismo.

Seguidamente se escribe, bien entre corchetes o con sangría, lo que debe suceder. En nuestro ejemplo, cada vuelta hace aumentar la información de tamaño. En CSS aparecerá un valor para cada pasada:

```
.width-1 {
width: 11em;
}
.width-2 {
width: 12em;
}
.width-3 {
width: 13em;
```

```
}
.width-4 {
width: 14em;
}
.height-1 {
height: 25em;
}
.height-2 {
height: 50em;
}
.height-3 {
height: 75em;
}
```

3.8.2 .- While

La directiva **@while** tiene una mecánica muy similar a la de @for pero, mientras que este último tiene puntos fijos de inicio y final, un bucle @while contiene una consulta de tipo lógico: mientras un estado siga siendo verdadero, las órdenes deberán repetirse. Como se verá a continuación, con la función @while podemos lograr exactamente el mismo resultado:

```
$i: 1;
@while $i < 5 {
      .width-#{$i} { width: 10em + $i; }
      $i: $i + 1;
}
```

Con este tipo de bucle, primero se debe asignar un valor a las variables, ya que la directiva en sí no necesita un valor de inicio. En el ciclo se ha de especificar el estado hasta el que se realizan las repeticiones. En nuestro ejemplo, el bucle se ejecuta siempre que la variable sea inferior a 5. La instrucción dentro del bucle es inicialmente la misma que en el ejemplo de @for. De nuevo, haremos que se ajuste la designación del elemento de las variables y aumente el tamaño. También se ha de integrar un comando en el bucle que haga que $i aumente con cada vuelta, porque, de lo contrario, el bucle se ejecutaría hasta que el compilador SASS se detuviera. Al final se obtiene el mismo código CSS que en @for.

3.8.3 .- Each

La directiva **@each** funciona de manera diferente. Este bucle se basa en una lista de datos especificada por el usuario que el bucle recorrerá en cada vuelta. Para cada entrada, @each hace una repetición diferente. De este modo, indicando una lista con los valores 1, 2, 3 y 4 sería posible generar el mismo resultado que con los otros bucles. La verdadera ventaja de este bucle, sin embargo, es que también puede introducirse otra información en la lista exceptuando valores numéricos (con @each, por ejemplo, se insertan varias imágenes diferentes en el diseño). Puedes introducir los datos directamente en la directiva o introducir la lista en una variable y luego llamarla.

```
$list: dog cat bird dolphin;
@each $i in $list {
      .image-#{$i} { background-image: url('/images/#{$i}.png'); }
}
```

Aquí, de nuevo, necesitas una variable. Esta asume el nombre de una de las entradas de la lista en cada vuelta. El nombre está integrado tanto en la denominación del bloque de código como en el nombre de archivo de la imagen. Para que el diseño funcione bien, las imágenes deben haber sido guardadas en la ruta especificada. El bucle se asegura de que la variable se haga cargo de la siguiente entrada.

```
.image-dog {
background-image: url("/images/dog.png");
}
.image-cat {
background-image: url("/images/cat.png");
}
.image-bird {
background-image: url("/images/bird.png");
}
.image-dolphin {
background-image: url("/images/dolphin.png");
}
```

3.8.4 .- If

Además de los bucles, SASS ofrece aún otra herramienta más conocida en el entorno de la programación: las ramificaciones según el principio “si-entonces-si no”. Con la directiva @if una orden solo se ejecuta si tiene lugar un cierto evento; en caso contrario, se ejecuta otro comando. Además de esta directiva, también hay una función if(). No guardan relación, pero pueden aparecer juntas. La función es fácil de explicar. Contiene tres parámetros: la condición y dos salidas diferentes. La primera salida se emite si el primer parámetro es verdadero, de lo contrario la función reproduce el tercer parámetro.

```
$black: #000000;
$white: #ffffff;
$text-color: $black;
body {
      background-color: if($text-color == $black, $white, $black);
}
```

En nuestro ejemplo, la función verifica si la variable $text-color está configurada en negro (black). Si este es el caso (como en el ejemplo), el fondo se mostrará blanco. En cualquier otro caso, el CSS mostraría el fondo en negro. Como puede verse en este ejemplo, esta técnica no es quizá la más adecuada para el diseño de un sitio web completo. Tanto la directiva como la función son útiles en primera línea en mixins o partials. Esto permite a la plantilla reaccionar mejor a los valores que aparecen en el diseño final. A la inversa, si ya sabes que el color del texto es negro, no hace falta escribir una ramificación compleja para que el fondo sea blanco.

Las funciones tienen la particularidad de devolver un solo valor. Para requisitos más complejos, conviene emplear la directiva @if, que cuenta con la ventaja de poder distinguir más de dos casos entre sí:

```
$black: #000000;
$white: #ffffff;
$lightgrey: #d3d3d3;
$darkgrey: #545454;
@mixin text-color($color) {
@if ($color == $black) {
background-color: $white;
      }
@else if ($color == $white) {
background-color: $black;
      }
@else if ($color == $lightgrey) {
background-color: $black;
      }
@else {
background-color: $white;
      }
}
p {
      @include text-color($lightgrey);
}
```

La sintaxis de la directiva permite teóricamente crear un caso para cada valor previsto, aunque es necesario colocar la directiva @else (que, en combinación con if puede invocarse tantas veces como se desee) detrás de la @if inicial. Solo el último @else permanece libre, cubriendo así a todos los demás casos.

3.9 .- Parciales

Cuando se trabaja con Sass, a menudo es una buena idea organizar su código en secciones o archivos reutilizables según el tipo de estilos que aplican (por ejemplo, tener un archivo para su diseño de cuadrícula, otro archivo para mixins, otro para esquemas de color, etc. .). Sass fomenta este flujo de trabajo modular mediante el uso de lo que se conoce como **parciales**. Un parcial sería cualquier archivo Sass que comience con un guión bajo (_), por ejemplo _grid.scss. Luego puede tener un directorio de todos sus parciales, luego un archivo Sass principal para importarlos, como main.scss

Para importar estos parciales a su archivo principal, simplemente usaría un @import declaración en la parte superior del archivo:

```
// archivo main.scss

@importar 'cuadrícula'
@importar 'mixins'
```

Cuando se trabaje con parciales, no es necesario que se proporcione el guión bajo o la extensión del nombre de archivo. Sass los detectará automáticamente. Esto facilita el uso de parciales para organizar mejor el código.

3.10 .- Funciones

SASS conoce numerosas funciones que facilitan el trabajo en la hoja de estilo. Se trata de workflows predefinidos que evitan tener que ejecutarlos manualmente. Las funciones pueden asignarse a diferentes categorías:

- Colores: con estas funciones pueden ajustarse los valores de color, la saturación, la transparencia y muchas otras propiedades. Con mix(), p.ej., pueden mezclarse dos colores y crear uno nuevo.
- Listas: en las listas (series de valores de propiedad CSS) las funciones sirven para leer el número de entradas o fusionar varias listas en una sola.
- Cadenas: las strings son cadenas fijas de caracteres, como las que se utilizan en los textos. Las funciones de esta categoría pueden colocar una cadena de caracteres entre comillas o convertir un texto completo a mayúsculas.
- Selectores: con las funciones de esta categoría se manipulan selectores completos. Con selector-unify() puedes fusionar dos selectores en uno solo (ahorrando mucha escritura).
- Números: dentro del tema de los números, valores o unidades, encontrarás funciones que pueden, entre otras cosas, redondear hacia arriba y hacia abajo, buscar el número más grande de un conjunto o entregar un número aleatorio.
- Mapas: los mapas en SASS son estructuras de datos compuestas de claves y propiedades. Con las funciones adecuadas, estos conjuntos de datos pueden manipularse fusionando dos mapas o borrando una clave de un mapa.
- Introspección: estas funciones permiten revisar el contenido de la hoja de estilo completa para comprobar, entre otras cosas, si en el código se encuentra una característica determinada, un mixin o una función en concreto.
- Miscellaneous: en el punto miscelánea SASS también incluye la útil función if() – que no debe confundirse con la directiva del mismo nombre. La diferencia se explica más abajo en el punto “Ramificación”.

Consejo: Una lista completa de las funciones SASS ya incluidas en el paquete de instalación se puede encontrar en la documentación oficial del lenguaje de la hoja de estilos. Allí también encontrarás una breve explicación de cada función.

Las funciones se insertan en el código siguiendo siempre el mismo patrón: cada función posee un nombre propio y ciertos parámetros que se encierran entre paréntesis separados por comas. Como resultado, la función entrega un valor individual. Explicaremos esta sintaxis de SASS usando el ejemplo de la simple pero útil función mix():

```
***CODE***
$color-1: #ffff00;
$color-2: #0000ff;
body {
```

```
background-color: mix($color-1, $color-2, 30%);
}
```

En esta función se mezclan los dos valores de color que se han definido previamente en las variables $color-1 y $color-2 (no es necesario guardar antes los valores de color en las variables; si se prefiere, los valores hexadecimales pueden incluirse directamente en la función). Como tercer parámetro, se especifica la proporción de la mezcla: en nuestro ejemplo, el resultado final contendrá un 30 por ciento de $color-1. Si dejas este último parámetro vacío, SASS interpreta que la mezcla ha de resultar 50/50. En el código CSS aparecerá un valor único (el valor hexadecimal del color resultante):

```
body {
  background-color: #4d4db3;
}
```

Las funciones explicadas hasta ahora están incluidas por defecto en SASS, pero el lenguaje de hojas de estilo también permite **definir funciones específicas para un proyecto**, lo que agiliza las fases del trabajo que más se repiten. Con ello se asemejan a las mixins, con la diferencia de que, mientras estas entregan líneas de código como output, las funciones solo entregan un valor. Las funciones se crean con la directiva *@function*, aunque en realidad necesitan siempre dos directivas: además de la inicial *@function*, se necesita un *@return* anidado con el cual se define el valor de salida:

```
$column-count: 12;
@function column-width($num) {
@return $num * 100% / $column-count;
}

.three-columns {
width: column-width(3);
```

En este ejemplo de arriba hemos calculado el ancho de columna para una cuadrícula de diseño. Primero definimos 12 columnas. En el siguiente paso, se da nombre a la función y se define cuántos parámetros contiene (en nuestro ejemplo, un número). Además, determinamos qué debe hacer la función y, por lo tanto, qué valor entrega. En este caso, *column-width* multiplica el número que se introduce como parámetro por 100% y divide el resultado por el número de columnas. Una vez definida la función, **se puede utilizar una y otra vez con parámetros distintos**. En el código final solo aparece el valor resultante:

```
.three-columns {
  width: 25%;
}
```

Consejo: En la creación de funciones pueden utilizarse guiones o guiones bajos en los nombres. Cuando se llama a la función más tarde, esta distinción es irrelevante. Tanto *function-name* como *function_name* llaman a la misma función.

3.11 .- Comentarios

Con SASS también es útil añadir comentarios al código fuente. Los comentarios contribuyen a que el código sea comprensible para cualquiera que lo lea en otro momento. Esto ocurre así especialmente cuando se crean plantillas para otros usuarios, porque los comentarios pueden serles útiles en la edición. Muchos diseñadores web también usan comentarios para estructurar el código con más claridad. La mayoría de los lenguajes de programación y marcado tienen la capacidad de insertar en el código texto que en la compilación o el análisis se ignora. Este texto está dirigido a las personas y no a las máquinas.

Los programadores y diseñadores web también usan esta función para “comentar” el código correcto, lo que se hace colocando un bloque de código, que no se necesita, pero que no se desea eliminar del código fuente, entre las etiquetas de comentarios.

Cada lenguaje tiene un método específico para comentar texto. En SASS, esto se hace de dos maneras diferentes. Por un lado, como en CSS, en SASS puedes utilizar /* */. Con este método puedes comentar varias filas al mismo tiempo. A menudo se encuentran comentarios en CSS o SASS donde cada línea en el bloque de comentarios comienza con un asterisco. Sin embargo, esto es solo una convención y no una necesidad.

```
/* Esto es un comentario.
Todo lo escrito entre estas marcas
no se tendrá en cuenta. */
```

Recuerda: Cuando se comenta, no importa si se escribe el código en SCSS o en sintaxis sangrada. Los comentarios funcionan igual en ambas sintaxis.

Además de este conocido método de CSS, con // también puedes comentar líneas aisladas:

```
// Esta línea es un comentario.
// Y esta línea también.
```

La diferencia entre los dos métodos también radica en que, si con la configuración por defecto la primera variante se copia en el CSS compilado, la segunda se pierde. De una forma u otra, un documento CSS con comentarios no puede lanzarse online como una versión productiva. Para ello se utiliza una versión minimizada que los navegadores pueden cargar más rápido.

3.12 .- Operadores

Por último me gustaría echar un vistazo a los operadores, una funcionalidad indispensable en cualquier lenguaje de programación y que nos pueden venir muy bien para escribir código css:

```
.container {
  width: 100%;
}

article {
  float: left;
  width: 600px / 960px * 100%;
}

aside {
  float: right;
  width: 300px / 960px * 100%;
}
```

No hace falta ni explicar que hace el código de arriba, es tan sencillo de usar que es incluso natural. Los operadores que tienes disponibles son:

- + : Para sumar cantidades
- - : Para restar números
- * : Para multiplicar
- / : Para dividir
- %: Para sacar el resto de las divisiones

4 .- Stylus

Este preprocesador fue creado por Learn Boost en el año 2010. Está escrito en JavaScript y tiene las mayorías de características contenidas en Sass y Less, pero con una sintaxis más sencilla y flexible. Stylus es el menos utilizado de los tres.

Lo que distingue a Stylus ante todo es su flexibilidad : los dos puntos, los puntos y las comas son opcionales. Además, no se necesitan llaves rizadas para definir bloques de código: en lugar de símbolos, Stylus utiliza indentaciones para ello. En cuanto a las variables, puedes usar el signo $ como en Sass... Pero también puedes omitirlo.

Todo esto te permite escribir menos y tener un código más limpio. Desafortunadamente, algunos desarrolladores encuentran esta flexibilidad como un defecto. El no tener identificadores claros hace que el código sea difícil de leer y entender, especialmente en proyectos grandes. Lo que obtienes es una anarquía causada por la libertad, por así decirlo. Si comparamos Stylus vs. Sass, este último tiene claramente una ventaja con su código de fácil lectura.

5 .- Frameworks de diseño web

Los **frameworks de diseño** ofrecen un esqueleto basado en columnas que facilita mucho la tarea de construir una web que se adapte a cualquier dispositivo. Además, ofrecen una serie de clases *CSS* y funciones de *JavaScript* que se pueden utilizar directamente y que nos ahorran tiempo a la hora de crear código.

Hoy en día existe en el mercado una amplia variedad de **frameworks de diseño web** *responsive* que nos ayudan a implementar nuestras interfaces gráficas. Algunos de los **frameworks más utilizados** son los siguientes.

- **Bootstrap** (18.9% de uso): es un *framework* de código abierto que facilita el diseño de sitios web y aplicaciones móviles. Ofrece una amplia variedad de componentes y estilos predefinidos para acelerar el desarrollo.
- **Animate** (9.4% de uso): es una biblioteca de animaciones CSS que permite agregar animaciones atractivas a los elementos de la página web.
- **Foundation** (0.5% de uso): es un *framework* de diseño responsivo que permite crear sitios web y aplicaciones móviles que se adaptan a diferentes tamaños de pantalla.
- **UIkit** (0.2% de uso): es un *framework* ligero y modular que ofrece una amplia variedad de componentes y estilos predefinidos para acelerar el desarrollo.
- **Skeleton** (0.1% de uso): es un *framework* CSS minimalista que proporciona una base sólida para el diseño de sitios web responsivos.
- **Tailwind** (0.1% de uso): es un *framework* de utilidades CSS que permite crear diseños personalizados sin tener que escribir CSS desde cero.
- **Materialize** (0.1% de uso): es un *framework* CSS basado en la guía de estilo Material Design, desarrollada por Google.
- **Material Design Lite** (0.1% de uso): es una versión ligera de Material Design que permite agregar estilos y componentes de Material Design a sitios web sin depender de ningún *framework*.
- **Semantic UI** (0.1% de uso): es un *framework* de diseño que utiliza lenguaje natural para nombrar sus clases y componentes, lo que facilita su uso y comprensión.
- **Bulma** (0.1% de uso): es un *framework* CSS basado en *Flexbox* que ofrece una amplia variedad de componentes y estilos predefinidos para acelerar el desarrollo.
- **MetroUI** (menos del 0.1% de uso): es un *framework* CSS que permite crear interfaces de usuario con el estilo de la interfaz Metro de Microsoft.

6 .- Referencias y recursos

- Página oficial desde donde poder consultar toda la documentación y poder descargar la librería:
    - getbootstrap.com
- Blog oficial de Bootstrap donde consultar las últimas novedades:
    - blog.getbootstrap.com
- Ejemplos y trozos de código útiles para Bootstrap:
    - bootsnipp.com
- Ejemplos de uso:
    - expo.getbootstrap.com
- Listado de los recursos disponibles:
    - startbootstrap.com/bootstrap-resources/

- Temas y plantillas gratuitas para Bootstrap:
    - startbootstrap.com
    - bootstrapzero.com
    - bootswatch.com
    - bootbundle.com
- Listado de clases de Bootstrap:
    - bootstrapcreative.com/resources/bootstrap-4-css-classes-index/
- Frameworks de diseño web:
    - Materialize CSS (materializecss.com)
    - Zurb Foundation (foundation.zurb.com)
    - Skeleton (getskeleton.com)
    - HTML5 Boilerplate (html5boilerplate.com)
- Cursos:
    - W3schools: w3schools.com/bootstrap4/
- Tutoriales:
    - Tutorial en PDF de W3schools traducido: tutorialesenpdf.com/bootstrap/