



Preprocesamiento CSS

Diseño de intefaces web

14 de febrero de 2023

CIFP Carlos III - Cartagena

Santiago Francisco San Pablo Raposo

2º curso DAW

Contenido

# Índice de ilustraciones.

# Índice de tablas.

Preprocesamiento CSS.

# 1.- Introducción a los pre-procesadores.

## 1.1.- Definición de pre-procesador.

A medida que los sitios se van complicando, empezamos a encontrar limitaciones al CSS puro, encontrándonos con hojas de estilo de miles de líneas realmente difíciles de mantener, modificar o refactorizar.

Un pre-procesador de CSS es un lenguaje de hoja de estilos, esto es, una extensión de CSS que nos permite añadir nuevas funcionalidades para hacer el trabajo de diseño y maquetación del sitio web mucho más sencillo.

Sin embargo, los navegadores no entienden los lenguajes de hojas de estilos, por lo que previamente hay que procesarlos, esto es, convertirlos a CSS interpretable por los navegadores.

## 1.2.- SAAS (Syntactically Awesome Style Sheet).

Actualmente, **con las novedades de HTML5, con Javascript**, que da gran parte de la vida a las millones de websites que se visitan cada día enriqueciendo la experiencia a los usuarios y facilitando el trabajo a los desarrolladores, como **con el framework angularjs, los CSS necesitan de alguna forma reinventarse, y es verdad que lo está haciendo actualmente, vía CSS3**, cuya propagación depende de los navegadores que interpretan y dan soporte a nuevas propiedades y posibilidades. **En todo este estado de efervescencia de las tecnologías web es en la que se enmarca el preprocesador SASS**.

Con estos websites más complejos de la actualidad, el trabajo en CSS tiene el peligro de crearse un código con muchos trozos repetidos e inabarcable por el número de líneas que se generan. **La función de SAAS y otros preprocesadores, como Less o Stylus, te ayudan justamente a evitar eso**.

### 1.2.1.- ¿Qué hace potente a SAAS?

Permite trabajar mucho más rápido en la creación de código con la **posibilidad de crear funciones que realicen ciertas operaciones matemáticas y reutilizar código gracias a los mixins**, variables que nos permiten guardar valores.

Y algo siempre importante cuando te decantas por una herramienta con alternativas, dispone de una gran comunidad que la hace progresar, por lo que se le augura un gran futuro por delante. El formato #SASS hace que los #CSS sean mucho más divertidos, potentes y reusables.

### 1.2.2.- ¿Cómo funciona?

**SASS dispone de dos formatos diferentes para la sintaxis**, lo que hace se traduce en **dos extensiones de fichero diferentes: .sass y .scss** El primero en salir fue .sass y se caracteriza, al igual que Stylus y Coffeescript para el lenguaje de programación Javascript, en no hacer uso de llaves ni punto y coma final, en busca de la simplicidad y eliminación de ruido. **Ejemplo**:

body

background: #000

font-size: 62.5%

**Los .scss salieron con la versión 3 de preprocesador y permiten utilizar llaves e incorporar código de CSS clásico**.

body {

background: #000;

font-size: 62.5%;

}

**Tanto la sintaxis de .sass como la de .scss no puede ser interpretada directamente por los navegadores**, por lo se debe compilar para traducir nuestro archivo SASS en un clásico fichero CSS.

El uso SASS o cualquier otro pre-procesador no es obligatorio, pero nos proporcionará funcionalidades no incluidas en CSS puro:

* Variables
* Funciones matemáticas
* Bucles
* Condicionales
* Estructuras lógicas y de control
* Mixins o plantillas
* Indentaciones
* Anidamientos
* Herencias
* Ficheros parciales
* Etc.

**SASS es un lenguaje que trabaja mediante indentaciones y retornos de carro**, en vez de utilizar llaves y puntos y comas. **Existe otra forma de utilizar SASS más orientada a la compatibilidad con el estándar CSS llamada SCSS**, que evita dichas indentaciones y retornos de carro y las sustituye por el estándar CSS.

Podemos utilizar cualquiera de las dos, pero no podemos mezclarlas en el mismo proyecto, ya que los ficheros tendrán extensión \*.sass o \*.scss..

**Ejemplo de sintaxis en SAAS**:

$fuente: Helvetica, sans-serif

$color: #F00

$color\_menu: #FFF

$color\_enlaces: #000

$fondo: #CCC

html

background-color: $fondo

body

font: 100%

$fuente color: $color

nav

Ul

margin: 0

padding: 0

list-style: none

color: $color\_menu

li

display: inline-block

a

display: block

padding: 6px 12px

text-decoration: none

color: $color\_enlaces

**Sintaxis en SCSS y el equivalente a CSS**:

Texto

Descripción generada automáticamente

# 2.- Instalación.

## 2.1.- Mediante el uso de un plug-in “en vivo” para Visual Studio Code.

Para conseguirlo, debemos accedr a la página del plug-in de Live Sass Compiler y pulsar el botón Install.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

### 2.1.1.- Configuración en Visual Studio Code.

En este caso, una vez instalado el plug-in, tan solo debemos abrir Visual Studio Code y, desde la barra de estado (View > Appearance > Show Status Bar) y, con un archivo con extensión \*.sass o \*.scss, hacer clic en el botón Watch Sass, de modo que empiece a procesar los cambios en nuestro archivo y volcarlos en otro de css puro que tendrá el mismo nombre que el original, pero con extensión \*.css:



**Para dejar de procesar, tan solo debemos volver a pulsar el botón**.

**Otra opción para conseguir el mismo resultado**, es abrir una Command Palette (View > Command Palette... o Ctrl + Shft + P) y teclear Live Sass: Watch Sass para empezar a procesar en tiempo real, o Live Sass: Stop Watching Sass para dejar de hacerlo.

**Por último, también es posible teclear Live Sass: Compile Sass - Without Watch Mode**, de modo que solo haga la compilación en ese momento (primero se debe guardar el archivo).

## 2.2.- Mediante el uso de frameworks de desarrollo.

Algunos frameworks de desarrollo, incorporan funcionalidades para optimizar el desarrollo con SASS, como es el caso de Webpack en el caso de Laravel.

# 3.- Sintaxis y reglas.

## 3.1.- Anidamiento (Nesting) de estilos y propiedades.

Las reglas de estilo son la base de CSS, y lo mismo ocurre con Sass y SCSS, de modo que la declaración siguiente es válida tanto en CSS como en SCSS:

Texto

Descripción generada automáticamente

Sin embargo, con Sass/SCSS podemos usar el anidamiento o nesting, lo que nos permitirá anidar los estilos que correspondan a elementos hijos dentro de los estilos de sus padres:

Texto

Descripción generada automáticamente

}

Correspondería al CSS:

Texto

Descripción generada automáticamente

No obstante **debemos tener en cuenta que el anidamiento excesivo de estilos se convertirá en hojas de estilo cada vez más complejas y poco manejables para el navegador**, por lo que debemos realizar el anidamiento con precaución.

También es posible utilizar el anidamiento para las listas de selectores, esto es, selectores separados por comas, ya que cada uno de estos selectores será anidado de manera separada y luego combinado de nuevo en la lista de selectores resultante:

Texto

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

## 3.2.- Referencia a selectores padre.

Otra utilidad interesante para combinar con el anidamiento es el uso del **selector de padre o parent selector, &, utilizado para referenciar al selector exterior del anidamiento**.

**SCSS**:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 3.3.- Comentarios.

* //: Comentarios de línea. No son procesados en el CSS de salida.
* /\*: Comentarios multilínea. Por defecto, el comentario aparecerá en CSS, salvo en el caso de que éste sea comprimido. Por eso se llaman loud comments.
  + Si el comentario empieza con /\*! aparecerá incluso si está comprimido.
  + Si el comentario está en un lugar donde podría ir una sentencia (expresión a evaluar) o contiene interpolación (resultado de una expresión SassScript), siempre aparecerá:

Texto

Descripción generada automáticamente

Imagen que contiene interior, pájaro, ave

Descripción generada automáticamente

## 3.4.- Selectores placeholder.

Existe un tipo de selector específico en Sass y SCSS llamado placeholder, que parece y actúa como un selector de clase estándar pero empieza con % y no se incluirá luego en el CSS de salida:

Texto

Descripción generada automáticamente

**Este selector nos permitirá extender (heredar) de él dentro de otros estilos**.

De este modo, **estos placeholder selectors nos permitirían crear una plantilla de estilos que podríamos extender en un momento dado** (como una librería en un archivo aparte) y sólo aparecerán en el CSS de salida si los hemos utilizado.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 3.5.- Variables.

El uso de variables en Sass/SCSS es bastante simple: asignamos un valor a un nombre que empiece por $ (como en PHP) y luego podemos referirnos a él en cualquier lugar en vez de referirnos al valor. **Las variables nos van a permitir realizar operaciones matemáticas, configurar librerías y mucho más**.

Al contrario que las propiedades, **las variables se pueden definir en cualquier lugar, sea dentro o fuera de las reglas**.

Es importante saber que los nombres de variables son tratados como el resto de identificadores Sass, de modo que **los guiones bajos y medios son tratados como iguales**, y por tanto $font\_size y $font-size harán referencia a la misma variable.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Importante**: No hay que confundir las variables de Sass/SCSS con las variables de CSS. Éstas últimas pertenecen a CSS puro pero, como es sabido, tienen muchos problemas de compatibilidad entre navegadores, mientras que las variables de Sass/SCSS se trasladan a CSS puro mediante compilación, pero no aparecen en el archivo de salida, con lo que podemos utilizarlas sin problemas de compatibilidad.

## 3.6.- Operadores.

* == y != se usan para comprobar si dos valores son iguales o no
* +, -, \*, / y % efectúan la correspondiente operación matemática entre los valores
* <=, > y >= son los operadores de comparación
* and, or y not tienen el comportamiento usual de los operadores booleanos
* +, - y / también se pueden usar para encadenar strings, si bien - y / funcionan como operadores unarios y están en desuso.

Es importante saber cuál es el orden de mayor a menor prioridad en el que se efectúan las operaciones (y aplicar paréntesis de forma matemática en nuestras expresiones si fuera necesario):

1. Los operadores unarios **not, +, - y /**.
2. Los operadores matemáticos **\*, / y %**
3. Los operadores matemáticos **+ y –**
4. Los operadores de comparación **<=, > y >=**
5. Los operadores de igualdad **== y !=**
6. El operador booleano **and**
7. El operador booleano **or**
8. El operador **=** (en desuso, antes se usaba en funciones para mantener la compatibilidad con Internet Explorer)

## 3.7.- Tipos de datos.

Tipos de datos provenientes directamente de CSS:

* **Números**, que pueden tener unidades o no, como 12 o 200px.
* **Strings**, que pueden tener comillas o no, como "Helvetica Neue" o bold.
* **Colores**, que pueden indicarse en hexadecimal (#F00), por el nombre (red), o como funciones (rgb(255,0,0) o hsl(0, 100%, 50%))
* **Listas de valores**, que pueden estar separadas por espacios (1.5em 1em 0 2em), comas (Helvetica, Arial) o corchetes ([col1-start]).

Propios de Saas/SCSS:

* **Mapas** que permiten asociar valores con claves, como ("background": red, "foreground": pink).
* **Booleanos**, cuyos posibles valores serán true o false.
* **Null** o ausencia de valor, muy útil para el retorno de funciones.

### 3.7.1.- Mapas.

**Los mapas contienen pares de índices y valores**, y hacen más sencillo asignar un valor a partir de su correspondiente índice.

**Es posible añadir elementos al mapa mediante la función map-merge()**, pero hay que tener en cuenta que los mapas son inmutables, por lo que en ningún caso estaremos modificando la lista original, a no ser que asignemos el nuevo mapa a otra variable.

**Por ejemplo**:

$font-weights: ("regular": 400, "medium": 500, "bold": 700);

@debug map-get($font-weights, "medium"); // 500

@debug map-get($font-weights, "extra-bold"); // null

$light-weights: ("lightest": 100, "light": 300);  
$heavy-weights: ("medium": 500, "bold": 700);  
@debug map-merge($light-weights, $heavy-weights);  
// (  
// "lightest": 100,  
// "light": 300,  
// "medium": 500,  
// "bold": 700  
// )  
$font-weights: ("regular": 400, "medium": 500, "bold": 700);  
@debug map-merge($font-weights, ("extra-bold": 900));  
// ("regular": 400, "medium": 500, "bold": 700, "extra-bold": 900)  
$font-weights: ("regular": 400, "medium": 500, "bold": 700);  
@debug map-merge($font-weights, ("medium": 600));  
// ("regular": 400, "medium": 600, "bold": 700)  
$prefixes-by-browser: ("firefox": moz, "safari": webkit, "ie": ms);  
@mixin add-browser-prefix($browser, $prefix) {  
$prefixes: map-merge($prefixes-by-browser, ($browser: $prefix));  
}  
@include add-browser-prefix("opera", o);  
@debug $prefixes-by-browser;

Además, podemos **trabajar con el mapa de modo que hagamos algo para cada par valor e índice**.

**SCSS:**

Texto

Descripción generada automáticamente

**CSS**:

Texto

Descripción generada automáticamente



**Importante**: Al igual que las listas, los mapas de Sass/SCSS son inmutables, esto es, todas las funciones devolverán un nuevo mapa sin afectar al original. Para modificar el mapa original, debemos asignar a esa misma variable el valor modificado

## 3.8.- Funciones.

Además de la posibilidad de definir nuestras propias funciones, que veremos más adelante, Sass/SCSS viene con gran cantidad de funciones predefinidas a las que podremos llamar utilizando la sintaxis  
estándar de CSS y, en algunos casos, una sintaxis especial de Sass.

**Estas funciones se dividen en categorías**:

* **Funciones de CSS**, aquéllas que no son reconocidas por Sass como propias y que se  
  compilarán como funciones de CSS estándar (var(), calc(), url(), etc.).
* **Funciones de Números**, normalmente de carácter matemático.
* **Funciones de Strings**, para crearlos, combinarlos, dividirlos y hacer búsquedas.
* **Funciones de Colores**, para generarlos, mezclarlos o modificarlos.
* **Funciones de Listas**, para acceder a ellas o modificarlas.
* **Funciones de Mapas**, para trabajar con ellos.
* **Funciones de Selectores**, para acceder al motor de selectores de Sass
* **Funciones de Introspección**, para mostrar detalles de la forma interna de trabajar de Sass.

Para conocer todas las funciones incluidas puedes consultar la documentación oficial.

## 3.9.- Interpolado.

La **interpolación** es algo que podemos utilizar casi en cualquier lugar de una hoja de Sass/SCSS **para incrustar el resultado de una expresión SassScript en CSS**, si bien su uso más común es **inyectar valores en Strings**.

**Cómo se hace**: debemos encerrar dicha expresión mediante #{} en cualquiera de los siguientes lugares:

* **Selectores en reglas de estilo**.
* **Nombres de propiedades en declaraciones**.
* **Valores de propiedades personalizadas**.
* Directivas (@).
* Herencias (@extends).
* @imports de CSS.
* Strings con o sin comillas.
* Funciones especiales.
* Funciones de CSS.
* Comentarios de tipo loud.

**Ejemplo de uso en selectores**:

Texto

Descripción generada automáticamente

Imagen que contiene Logotipo

Descripción generada automáticamente

# 4.- Reglas y directivas básicas.

En este apartado introduciremos reglas y directivas básicas: importación, mixinse includes, funciones, y herencia y extensión.

## 4.1.- Importación (@import) y Parciales (Partials).

**Sass/SCSS** extiende la regla @import de CSS **añadiendo la capacidad de importar tanto hojas de estilo CSS como hojas de Sass/CSS**, permitiendo así el acceso a mixins, funciones y variables de dichas hojas y la combinación de todas ellas en una sola.

Además, Sass/SCSS realiza toda la importación durante la compilación, en vez de obligar al navegador a hacer una petición HTTP por cada @import como hace CSS.

**Sintaxis**: similar a la de CSS, salvo que en este caso podremos realizar varios @import a la vez,  
separando las rutas de los archivos por comas:

Texto

Descripción generada automáticamente

**Nota**: Las rutas de los archivos se escriben en formato url, usando / incluso en Windows. Por otra parte, no es necesario el uso de ./ para @import relativos, pues estos siempre están disponibles.

**Partials**: Como convención, **los ficheros Sass/SCSS están pensados para ser importados y no compilados directamente** (a modo de ficheros parciales). Suelen empezar su nombre por \_, como el anterior \_code.scss. Ese “\_” le dice a las herramientas de Sass que no lo compilen por sí mismo. No obstante, no es necesario poner el \_ para importar el partial.

**Si creamos un archivo \_index.scss o \_index.sass en un directorio, cuando importamos (con @import) dicho directorio, el fichero index será cargado en su lugar**, lo cual puede resultar util para mejorar la estructura de nuestras hojas y partials.

Si bien **los @import** están pensados para ser utilizados al principio de la hoja, **es posible anidarlos dentro de reglas de estilo u otras directivas**.

## 4.2.- Mixins (@mixin) e Includes (@include).

Los **mixins nos permiten definir estilos para ser re-utilizables** en todas las hojas de estilo o ser distribuidos en forma de librerías.

**Sintaxis**:

* **Declaración**: @mixin <nombre> {…}
  + O bien: @mixin <nombre>(<argumentos>).
* **Llamadas**: @include <nombre>
  + O bien: @include <nombre>(<argumentos>).

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 4.3.- Funciones (@function).

Las funciones **nos van a permitir definir operaciones complejas que podremos abstraer de una manera sencilla** y utilizar en cualquier otra parte de la hoja.

**Definición**: @function <nombre>(<argumentos...>) { ... }.

**Solo pueden contener sentencias universales** (variables, directivas de control o directivas @error, @warn y @debug).

Los argumentos funcionan de manera similar a los mixins, tal como se puede observar. **Por ejemplo**:

Texto

Descripción generada automáticamente con confianza media

Imagen que contiene Texto

Descripción generada automáticamente

Texto, Escala de tiempo

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Importante**: la directiva **@return** que hemos visto en algunos de los ejemplos anteriores indica que la ejecución finaliza y se devuelve el resultado. Esta directiva es **obligatoria** y solo se puede utilizar dentro de @function.

## 4.4.- Herencia y extensión (@extend).

Hay muchos casos en los que nos encontramos en la **necesidad de que una clase tenga todos los estilos de otra, además de algo específico para ella**. Para estos casos, utilizamos la **directiva @extend** mediante la sintaxis @extend <selector>. Esto **nos permitirá modificar reglas de una forma más sencilla y óptima que con un mixin**:

Texto

Descripción generada automáticamente Imagen que contiene Gráfico

Descripción generada automáticamente

# 5.- Directivas de control y heredadas.

En este apartado introduciremos las directivas heredadas de CSS puro y las estructuras de control, básicas para gestionar el comportamiento de las reglas en distintos ámbitos y también incluidas dentro de las at-rules :

* **Condicionales** (@if y @else).
* **Bucles** (@each, @for y @while).
* **Directivas heredadas** (@namespace, @font-face, @media, @support).

## 5.1.- Condicionales (@if y @else).

La directiva @if permite comprobar si un bloque es evaluado o no, y como en cualquier otro lenguaje, opcionalmente puede incorporar la directiva @else o la directiva @elseif.

**Sintaxis**: @if <expression> {…}

**Por ejemplo**:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 5.2.- Bucles (@each, @for, @while).

|  |  |
| --- | --- |
| Tipo de bucle | Sintaxis y ejemplo |
| @each | Permite **recorrer una lista y emitir estilos o evaluar código** para cada uno de sus elementos.  **Sintaxis**: @each <variable> in <expresión> { ... } |
| **Ejemplo**: |
| @for | Permite contar de un nº superior a otro inferior (o viceversa) y realizar una acción en cada pasada.  **Sintaxis**:   * **Si queremos excluir el valor final**: @for <variable> from <expresión> to <expresión> {... } * **Si queremos incluir el valor final**: @for <variable> from <expresión> through <expresión> { ... } |
| **Ejemplo**: |
| @while | Esta directiva evalúa una expresión y ejecuta acciones mientras sea cierta.  **Sintaxis**: @while <expresión> { ... }.  **Nota**: Como en cualquier otro lenguaje, para evitar bucles infinitos y porque además son más rápidas, es mejor utilizar @for o @each frente a @while. |
| **Ejemplo**: |

## 5.3.- Directivas heredadas de CSS.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Texto

Descripción generada automáticamente