FUNDAMENTOS DE SASS

por Javier Guede

www.guede.me

ÍNDICE

INTRODUCCIÓN A SASS

SASS (Syntactically Awesome StyleSheets) es un lenguaje extendido de CSS que añade un nuevo abanico de posibilidades y características muy potentes y elegantes. Se trata de un preprocesador que permite escribir el código CSS mucho más fácil, mantener el código organizado y aporta herramientas para hacer ese código reutilizable.

SASS permite utilizar características que no existen en CSS todavía como variables, anidación, herencia y otras ventajas que ayudan a mantener nuestro código CSS.

En definitiva, SASS incluye las siguientes características:

- 100% compatible con CSS3.
- Permite el uso de variables, anidamiento de estilos y *mixins*.
- Incluye numerosas funciones para manipular con facilidad colores y otros valores.
- Permite el uso de elementos básicos de programación como las directivas de control y las librerías.
- Genera archivos CSS bien formateados y permite configurar su formato.
- Integración con Firebug gracias a la extensión FireSass.

Una vez que se empieza a utilizar SASS, el archivo SASS preprocesado se compilará en un archivo CSS normal que se puede utilizar en un sitio web.

INSTALACIÓN DE SASS

Se puede instalar a través del siguiente enlace:

http://sass-lang.com/install

Para ello, es necesario tener Ruby instalado (MAC y Linux ya lo trae instalado por defecto y en Windows puede hacerse a través del siguiente enlace: http://rubyinstaller.org/).

Posteriormente, en un terminal con los permisos suficientes de usuario, ejecutamos la siguiente instrucción:

```
gem install sass
```

SINTAXIS SASS

SASS permite el uso de dos sintaxis diferentes para crear sus archivos.

La primera, conocida como **SCSS** (del inglés, *Sassy CSS*) es una extensión de la sintaxis de CSS3. Esto significa que cualquier hoja de estilos CSS3 válida también es un archivo SCSS válido. Los archivos creados con esta sintaxis utilizan la extensión **.scss**.

La segunda sintaxis, conocida como "sintaxis indentada" o simplemente "sintaxis **SASS**" permite escribir los estilos CSS de manera más concisa. En este caso, el anidamiento de selectores se indica con tabulaciones en vez de con llaves y las propiedades se separan con saltos de línea en vez de con puntos y coma. Así mismo, es estrictamente necesario colocar un espacio después de los dos puntos cuando se está asignando un valor a una propiedad CSS concreta. Los archivos creados con esta segunda sintaxis utilizan la extensión **.sass**.

Se considera que esta segunda sintaxis es más sencilla de leer y más rápida de escribir que SCSS. En cualquier caso, las dos sintaxis tienen exactamente las mismas funcionalidades y sólo se diferencia en su sintaxis.

Una de las ventajas de SASS es que los archivos creados con una sintaxis pueden importar cualquier archivo creado con la otra sintaxis. Además, dispones de una utilidad para la línea de comandos que te permite convertir una sintaxis en otra:

```
# Convierte un archivo SASS en SCSS
$ sass-convert estilos.sass estilos.css

#Convierte un archivo SCSS en SASS
$sass-convert estilos.scss estilos.sass
```

En los ejemplos de este manual se va a utilizar principalmente la sintaxis SASS, puesto que es la más actual y la que más optimiza el código.

USANDO SASS

SASS se puede utilizar de tres maneras diferentes:

1. En la consola de comandos
2. Como módulo de Ruby
3. Como plugin de cualquier *framework* compatible con Rack (como por ejemplo Ruby on Rails y Merb)

Utilizando SASS en la línea de comandos se ejecuta cada instrucción encabezada por el comando **sass**.

El navegador web no es capaz de leer archivos **.sass**, por lo tanto, es necesario compilarlos para generar archivos **.css** que puedan ser interpretados por el mismo.

```
sass hoja_estilos.sass:archivo_generado.css
```

Cada vez que se realiza un cambio en un archivo .sass, es necesario volver a generar el archivo .css. Se puede automatizar este proceso mediante la siguiente sentencia:

```
sass --watch hoja_estilos.sass:archivo_generado.css
```

Si se dispone de un directorio con varios archivos SASS, se puede poner bajo vigilancia dicho directorio de manera que se compilen automáticamente al realizar algún cambio en alguno de ellos:

```
sass --watch app/sass:public/stylesheets
```

Se puede obtener una documentación de ayuda de una sentencia determinada de SASS, ejecutando el siguiente comando después de dicha sentencia:

```
sass --help
```

CACHEANDO LOS ARCHIVOS COMPILADOS

Por defecto SASS guarda en la caché la compilación de las hojas de estilos y de los *partials*. Esto hace que la recompilación de los estilos en las aplicaciones complejas sea mucho más rápida. Para obtener los mejores resultados, se aconseja dividir las hojas de estilos grandes en varios archivos pequeños e impórtalos después con la directiva **@import**.

Si no se utiliza un *framework*, SASS guarda los archivos cacheados en el directorio oculto “/sass-cache/”.

HOJAS DE ESTILOS PARCIALES

SASS deja crear hojas de estilos parciales SASS o SCSS (*partials*) que contienen pequeños fragmentos de código y que se pueden incluir en otros archivos SASS. Esta es una buena manera de estructurar el CSS y mantener el código más fácil de mantener.

Estas hojas de estilo parciales SCSS o SASS no se van a compilar como archivos CSS.

Una hoja de estilos parcial simplemente es un archivo SASS que utiliza un guion bajo (_) como primer carácter del nombre del archivo. El guion bajo permite saber a SASS que el fichero es sólo un archivo parcial y que no debe ser generado en un archivo CSS.

Ejemplo:

```
_partials.sass
```

A la hora de importar los archivos parciales se hace con la regla ***@import*** seguidos de su nombre. En este caso, no es necesario que vayan precedidos por el guion bajo ni con la terminación del tipo de archivo, pero ambos casos serían correctos.

Esto nos permite tener divido y organizado nuestro código en varios archivos parciales e importarlos todos en el mismo archivo. De esta manera, se generará un único fichero CSS para todo nuestro sitio web y solamente se hará una petición al servidor para reproducir toda la página.

Ejemplo:

```
@import _partials.sass
@import partials.sass
@import partials
```

No se puede tener en un mismo directorio una hoja de estilos parcial y una hoja de estilos SASS con el mismo nombre.

RECOMENDACIONES

- Se aconseja tener un archivo principal SASS que contenga sólo *@import* con todas las hojas de estilos parciales. De este modo, solamente se compilará un archivo SASS y por lo tanto obtendremos un único archivo CSS.
- El resto del código se debe dividir y organizar de una buena forma en hojas de estilo parciales, bien sea en código relativo a elementos base (estilos básicos), secciones (header, main, footer…), páginas, etc.

COMENTARIOS

SASS soporta el mismo tipo de comentarios que CSS, que utilizan los delimitadores **/*** y ***/** y pueden ocupar una o más líneas.

Además, SASS también soporta los comentarios de una única línea que utilizan los delimitadores **//** y que son muy comunes en todos los lenguajes de programación.

La principal diferencia entre estos dos tipos de comentarios es que los comentarios tradicionales (**/*** ... ***/**) se añaden en el código CSS generado, mientras que los comentarios de una sola línea (**//** ...) se eliminan y no aparecen en el código CSS generado.

Ejemplo:

```
/* Este comentario ocupa varias líneas,
 * y utiliza el formato tradicional de CSS.
 * Su contenido aparecerá en el archivo CSS compilado. */
body { color: black; }
// Estos comentarios ocupan una sola línea cada uno
// Todos estos comentarios se eliminan al generar el
// archivo CSS y por tanto, el usuario no podrá verlos
a { color: green; }
```

El código SASS anterior se compila de la siguiente manera:

```
/* Este comentario ocupa varias líneas,
 * y utiliza el formato tradicional de CSS.
 * Su contenido aparecerá en el archivo CSS compilado. */
body {
    color: black;
}

a {
    color: green;
}
```

Cuando la primera letra de un comentario de una sola línea es **!**, su contenido siempre se incluye en el archivo CSS compilado.

ANIDACIÓN DE SELECTORES

SASS permite anidar los selectores CSS de manera que siguen la misma jerarquía que el código HTML correspondiente y de este modo las hojas de estilos resultan más concisas y fáciles de escribir.

De esta manera, se evita tener que repetir una y otra vez los mismos selectores y se simplifica enormemente la creación de hojas de estilos complejas. Sin embargo, las reglas excesivamente anidadas resultan se traduce en CSS que podría ser difícil de mantener y, en general se considera una mala práctica.

Ejemplo:

```
main
    width: 97%
    p, div
        font-size: 2em
        a
            font-weight: bold
    pre
        font-size: 8em
```

Se convierte en el siguiente código CSS:

```
#main {
    width: 97%;
}
#main p, #main div {
    font-size: 2em;
}
#  main p a, #main div a {
  font-weight: bold;
}
#main pre {
    font-size: 3em;
}
```

REFERENCIANDO A LOS SELECTORES PADRE

En ocasiones es necesario modificar el comportamiento por defecto de los selectores anidados.

Por ejemplo, si se quiere aplicar estilos especiales en el estado *hover* del selector o cuando el elemento <body> de la página tiene una determinada clase.

En estos casos, se puede utilizar el carácter **&** para hacer referencia al selector padre dentro del cual se encuentra la regla anidada. El carácter especial **&** siempre se reemplaza por el selector padre tal y como aparece en el archivo CSS. Esto significa que si tiene una regla anidada, primero se calcula el selector padre completo y después se reemplaza por **&**.

Ejemplo:

```
main
    color: black
    a
        font-weight: bold
        &:hover
            color:red
```

Se convierte en el siguiente código CSS:

```
#main {
    color: black;
}
#main a {
    font-weight: bold;
}
#main a:hover {
    color: red;
```

El carácter **&** siempre debe aparecer al principio de los selectores compuestos, pero sí que puede ir seguido de un sufijo que se aplicará al selector padre.

Ejemplo:

```
main
    color: black
    &-sidebar
        border: 1px solid
```

El código SASS anterior compila de la siguiente manera:

```
#main {
    color: black;
}
#main-sidebar {
    border: 1px solid;
}
```

PROPIEDADES ANIDADAS

CSS define varias propiedades cuyos nombres parecen estar agrupados de forma lógica.

Así, por ejemplo, las propiedades *font-family, font-size* y *font-weight* están todas relacionadas con el grupo *font*. En CSS es obligatorio escribir el nombre completo de todas estas propiedades.

SASS permite utilizar el siguiente atajo para definir las propiedades relacionadas y también se puede aplicar un valor al propio nombre que agrupa las propiedades:

Ejemplo:

```
funky
    font: 2px/3px
       family: fantasy
       size: 30em
       weight: bold
```

El código SASS anterior compila de la siguiente manera:

```
.funky {
    font: 2px/3px;
    font-family: fantasy;
    font-size: 30em;
    font-weight: bold;
}
```

VARIABLES

SASS permite crear variables y se utilizan para almacenar valores que se pueden reutilizar a lo largo de toda la hoja de estilos. Se pueden almacenar cosas como colores, fuentes o cualquier valor CSS que se desee volver a usar.

Se declaran del siguiente modo: se inician con la letra **$** más un identificador deseado y se les asigna un valor determinado después de dos puntos.

Sintaxis:

```
$var: valor
```

De este modo, en el lugar en el que se haya asignado una variable a una propiedad, esta será sustituida exactamente por su valor.

Ejemplo:

```
//Creación de variables
$font-stack: Helvetica, sans-serif
$primary-color: #333

//Estilos
body
    Font: 100% $font-stack
    color: $primary-color
```

El código SASS anterior es compilado a CSS de la siguiente manera:

```
body {
    font: 100% Helvetica, sans-serif;
    color: #333;
}
```

MIXIN

Algunas propiedades en CSS son un poco tediosas de escribir, especialmente con CSS3 y algunos prefijos de navegador o propietarios (usados para que los navegadores puedan implementar algunas propiedades css) existentes.

Un *mixin* permite hacer grupos de declaraciones CSS que se quieren reutilizar en todo el sitio web. Incluso se pueden pasar valores para hacer tu *mixin* más flexible. Un buen uso de un *mixin* es para los prefijos de los navegadores.

Los *mixins* se definen de maneras diferentes dependiendo de la sintaxis:

- Sintaxis SCSS: con la directiva ***@mixin*** seguida de un identificador con el que deseamos nombrar al *mixin* y opcionalmente una lista de argumentos.
- Sintaxis SASS: con el signo **=** seguido de un identificador con el que deseamos nombrar al *mixin* y opcionalmente una lista de argumentos.

Así mismo, los mixins también se incluyen de manera diferente en función de su sintaxis:

- Sintaxis SCSS: con la directiva ***@include*** seguida del nombre del *mixin* y opcionalmente una lista de argumentos.
- Sintaxis SASS: con el signo **+** seguido del nombre del *mixin* (y opcionalmente una lista de argumentos).

El resultado es que todos los estilos definidos en el *mixin* se incluyen en el mismo punto en el que se llama al *mixin*.

Ejemplo Sintaxis SCSS:

```
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
     -ms-border-radius: $radius;
     border-radius: $radius;
}

.box { @include border-radius(10px); }
```

Ejemplo Sintaxis SASS:

```
=border-radius($raidus)
    -webkit-border-radius: $radius
    -moz-border-radius: $radius
    -ms-border-radius: $radius
    border-radius: $radius

.box
    +border-radius(10px)
```

El CSS generado del código anterior se parece a esto:

```
.box {
    -webkit-border-radius: 10px;
    -moz-border-radius: $10px;
    -ms-border-radius: 10px;
    border-radius: 10px;
}
```

EXTENDED Y HERENCIA

Esta es una de las características más útiles de SASS. En ocasiones, es necesario que una clase CSS contenga todos los estilos aplicados a otra regla CSS.

Usando ***@extend*** permite compartir un grupo de propiedades desde un selector a otro. Esto ayuda a evitar tener que escribir múltiples clases CSS sobre los elementos HTML.

La solución habitual en estos casos consiste en crear una clase genérica que puedan utilizar los otros elementos.

Ejemplo:

```
.message
    border: 1px solid #ccc;
    padding: 10px;
    color: #333;

.success
    @extend .message;
    border-color: green;

.error
    @extend .message;
    border-color: red;

.warning
    @extend .message;
    border-color: yellow;
```

El ejemplo anterior se compila en el siguiente código CSS:

```
.message, .success, .error, .warning {
    border: 1px solid #cccccc;
    padding: 10px;
    color: #333;
}

.success {
    border-color: green;
}

.error {
    border-color: red;
}

.warning {
    border-color: yellow;
}
```

OPERADORES

TIPOS DE DATOS

SASS tolera seis tipos de datos:

- Números (1.2, 35, 18px)
- Cadenas de texto (“awesome”, ‘fonts’, line)
- Colores (*blue*, *#04a3f9* y *rgba(255,0,0,0.5)*)
- Valores lógicos o booleanos (*true* or *false*)
- Valores nulos (*null*)
- Listas de valores, separados por espacios en blanco o comas (1.5em 1em 0 2em) (Helvetica, Arial, sans-serif)
- Los mapas que son pares formados por una clave y un valor separados por**:** (key: value1, key2: value2)

SASS también soporta todos los otros tipos de datos soportados por CSS, como por ejemplo los caracteres Unicode o la palabra reservada *!important*.

OPERADORES ARITMÉTICOS

SASS soporta cinco operadores aritméticos básicos tales como:

+	Suma
-	Resta
/	División
*	Multiplicación
%	Módulo (sin decimales)

Si se realizan operaciones sobre números con diferentes unidades, SASS convertirá las unidades automáticamente siempre que sea posible.

Ejemplo:

```
.container
    width: 100%

article[role="main"]
    float: left
    width: 600px / 960px * 100%

aside[role="complimentary"]
    float: right
    width: 300px / 960px * 100%
```

El ejemplo anterior se compila en el siguiente código CSS:

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

article[role="main"] {
    float: left;
    width: 62.5%;
}

aside[role="complimentary"] {
    float: right;
    width: 31.25%;
}
```

PROBLEMA CON EL CARÁCTER / CON LA DIVISIÓN CON NÚMEROS

CSS permite el uso del carácter **/** para separar números. Como SASS es totalmente compatible con la sintaxis de CSS, debe soportar el uso de esta característica. El problema es que el carácter **/** también se utiliza para la operación matemática de dividir números. Por todo esto, si utilizas el carácter **/** para separar dos números en SASS, en el archivo CSS compilado aparecerán tal cual los has escrito.

No obstante, existen tres situaciones en las que el carácter / siempre se interpreta como una división matemática:

1. Si uno de los operandos de la división es una variable o el resultado devuelto por una función.
2. Si el valor está encerrado entre paréntesis.
3. Si el valor se utiliza como parte de una expresión matemática.

Ejemplo:

```
p
// El carácter '/' se interpreta como código CSS normal
    font: 10px/8px
    $width: 1000px

// El carácter '/' se interpreta como una división
    width: $width/2 //Uno de los operandos es una variable
    width: round(1.5)/2 //Uno de los operados es el resultado de
una función
    height: (500px/2) //Los paréntesis encierran la expresión
    margin-left: 5px + 8px/2px
    //El '+' indica que es una expresión matemática
```

El ejemplo anterior se compila en el siguiente código CSS:

```
p {
    font: 10px/8px;
    width: 500px;
    height: 250px;
    margin-left: 9px;
}
```

Si quieres utilizar el carácter / normal de CSS incluso cuando empleas variables, encierra las variables con **#{}**.

Ejemplo:

```
p
    $font-size: 12px;
    $line-height: 30px;
    font: #{$font-size}/#{$line-height};
```

El ejemplo anterior se compila en el siguiente código CSS:

```
p {
    font: 12px/30px;
}
```

OPERADORES DE COMPARACIÓN

Con los números también se pueden utilizar operadores de comparación:

>	Mayor que
<	Menor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a*
!=	No igual a

OPERADORES LÓGICOS O BOOLEANOS

SASS también admite los operadores lógicos o booleanos:

and	Y Lógica
or	O Lógica
not	Negación

PARÉNTISIS

Se puede añadir paréntesis a cualquier expresión SASS para afectar al orden en que se realizan las operaciones.

Ejemplo:

```
p
    width: 1em + (2em * 3)
```

El código CSS compilado resultante es:

```
P {
    Width: 7em;
}
```

OPERADORES PARA COLORES

Los operadores aritméticos también se pueden aplicar a los valores que representan colores. En este caso, los cálculos siempre se realizan sobre cada componente del color. Esto significa que antes de cada operación, el color se descompone en sus tres componentes R, G y B, para después aplicar la operación a cada componente.

Ejemplo:

```
P
    color: #010203 + #040506
```

Las tres operaciones realizadas son 01 + 04 = 05, 02 + 05 = 07 y 03 + 06 = 09, por lo que el código CSS compilado resultante es:

```
p {
    color: #050709;
}
```

Las operaciones matemáticas también se pueden realizar combinando colores y números.

Ejemplo:

```
P
    color: #010203 *2
```

Las tres operaciones realizadas son 01 * 2 = 02, 02 * 2 = 04 y 03 * 2 = 06, por lo que el código CSS compilado resultante es:

```
p {
    color: #020406;
}
```

Si realizas operaciones sobre colores que incluyen un canal *alpha* (por ejemplo los que han sido creados con las funciones *rgba()* o *hsla()*) los dos colores deben tener el mismo valor *alpha* para poder realizar la operación con éxito. El motivo es que los cálculos no afectan al valor *alpha.*

Ejemplo:

```
p
    color: rgba(255, 0, 0, 0.75) + rgba(0, 255, 0, 0.75)
```

El código CSS compilado resultante es:

```
P {
    color: rgba(255, 255, 0, 0.75);
}
```

El canal *alpha* de un color se puede ajustar con la función ***opacify()*** o ***transparentize()***.

Ejemplo:

```
$translucent-red: rgba(255, 0, 0, 0.5)

p
    color: opacify($translucent-red, 0.3)
    background-color: transparentize($translucent-red, 0.25)
```

El código SASS anterior se compila de la siguiente manera:

```
p {
    color: rgba(255, 0, 0, 0.8);
    background-color: rgba(255, 0, 0, 0.25);
}
```

OPERADORES PARA CADENAS DE TEXTO

El operador **+** se puede utilizar para concatenar dos o más cadenas de texto.

Ejemplo:

```
P
    cursor: e + -resize
```

El código SASS anterior se compila de la siguiente manera:

```
p {
    cursor: e-resize;
}
```

Si la cadena que está a la izquierda del operador **+** está encerrada por comillas, el resultado de la operación será una cadena con comillas. Igualmente, si la cadena de la izquierda no tiene comillas, el resultado será una cadena sin comillas.

Ejemplo:

```
p:before
    content: "Foo " + Bar
    font-family: sans- + "serif"
```

El código SASS anterior se compila de la siguiente manera:

```
p:before {
    content: "Foo Bar";
    font-family: sans-serif;
}
```

Dentro de una cadena de texto puedes utilizar la sintaxis **#{ }** para realizar operaciones matemáticas o para evaluar expresiones antes de incluirlas en la cadena. Esta característica se llama "*interpolación de cadenas de texto*".

Ejemplo:

```
p:before
    content: "¡Me he comido #{5 + 10} pasteles!"
```

El código SASS anterior se compila de la siguiente manera:

```
p:before {
    content: "¡Me he comido 15 pasteles!";
}
```

VARIABLES CON VALORES POR DEFECTO

La palabra reservada ***!default*** permite controlar la asignación de valores a las variables de manera mucho más precisa. Si una variable ya tenía un valor asignado, *!default* hace que se mantenga sin cambios. Si la variable no existía o no tenía ningún valor, se utiliza el nuevo valor asignado.

Ejemplo:

```
$contenido: "Primer contenido"
$contenido: "¿Segundo contenido?" !default
$nuevo_contenido: "Tercer contenido" !default

#main
    contenido: $contenido
    nuevo-contenido: $nuevo_contenido
```

El código SASS anterior se compila de la siguiente manera:

```
#main {
  contenido: "Primer contenido";
  nuevo-contenido: "Tercer contenido";
}
```

ESTRUCTURAS DE CONTROL

SASS define algunas directivas de control básicas y expresiones para incluir estilos solamente si se cumplen determinadas condiciones o para incluir el mismo estilo varias veces con ligeras variaciones.

LA DIRECTIVA @IF

La directiva ***@if*** evalúa una expresión SASS y solamente incluye los estilos definidos en su interior si la expresión devuelve un valor distinto a *false* o *null*.

Ejemplo:

```
p
    @if 1 + 1 == 2 { border: 1px solid
    @if 5 < 3      { border: 2px dotted
    @if null       { border: 3px double
```

El código SASS anterior se compila de la siguiente manera:

```
p {
    border: 1px solid;
}
```

La directiva ***@if*** puede ir seguida de una o más directivas ***@else if*** y una directiva ***@else***. Si la expresión evaluada por *@if* es *false* o *null*, SASS evalúa por orden el resto de directivas *@else if* hasta que alguna no devuelva *false* o *null*. Si ninguna directiva *@else if* llega a ejecutarse, se ejecuta la directiva *@else* si existe.

Ejemplo:

```
$type: monster
p
    @if $type == ocean
        color: blue;
    @else if $type == matador
        color: red
    @else if $type == monster
        color: green
    @else
        color: black
```

El código SASS anterior se compila de la siguiente manera:

```
p {
    color: green;
}
```

LA DIRECTIVA @FOR

La directiva ***@for*** muestra repetidamente un conjunto de estilos. En cada repetición se utiliza el valor de una variable de tipo contador para ajustar el resultado mostrado. La directiva puede utilizar dos sintaxis:

Sintaxis 1:

```
@for $var from <inicio> through <final>
```

Sintaxis 2:

```
@for $var from <inicio> to <final>.
```

La diferencia entre las dos sintaxis es el uso de las palabras clave *through* o *to*. El valor *$var* puede ser cualquier variable, mientras que <inicio> y <final> son expresiones SASS que deben devolver números enteros. Cuando el valor de <inicio> es mayor que el de <final> el valor del contador se decrementa en vez de incrementarse.

En cada repetición del bucle, la directiva *@for* asigna a la variable *$var* el valor del contador y repite los estilos utilizando el nuevo valor de *$var*. En la sintaxis *from ... through*, los estilos se repiten desde <inicio> hasta <final>, ambos inclusive. Por su parte, en la sintaxis *from ... to* los estilos se repiten desde <inicio> hasta <final>, sin incluir este último.

Ejemplo:

```
@for $i from 1 through 3
    .item-#{$i}
        width: 2em * $i
```

El código SASS anterior se compila de la siguiente manera:

```
.item-1 {
    width: 2em;
}
.item-2 {
    width: 4em;
}
.item-3 {
    width: 6em;
}
```

LA DIRECTIVA @EACH

La directiva ***@each*** también puede utilizar varias variables de forma simultánea, como por ejemplo: *@each $var1, $var2*, ... in <lista>. Si <lista> es una lista formada por listas, a cada variable se le asigna un elemento de cada sublista.

Ejemplo:

```
@each $animal, $color, $cursor in (puma, black, default),
                                  (sea-slug, blue, pointer),
                                  (egret, white, move)
    .#{$animal}-icon {
        background-image: url('/images/#{$animal}.png')
        border: 2px solid $color
        cursor: $cursor
```

El código Sass anterior se compila de la siguiente manera:

```
.puma-icon {
    background-image: url('/images/puma.png');
    border: 2px solid black;
    cursor: default;
}
.sea-slug-icon {
    background-image: url('/images/sea-slug.png');
    border: 2px solid blue;
    cursor: pointer;
}
.egret-icon {
    background-image: url('/images/egret.png');
    border: 2px solid white;
    cursor: move;
}
```

LA DIRECTIVA @WHILE

La directiva **@while** toma una expresión SASS y repite indefinidamente los estilos hasta que la expresión da como resultado *false*. Aunque esta directiva se usa muy poco, se puede utilizar para crear bucles más avanzados que los que se crean con la directiva *@for*.

Ejemplo:

```
$i: 6
@while $i > 0
    .item-#{$i}
        width: 2em * $i
    $i: $i - 2
```

El código SASS anterior se compila de la siguiente manera:

```
.item-6 {
  width: 12em;
}

.item-4 {
  width: 8em;
}

.item-2 {
  width: 4em;
}
```

FUNCIONES

Al margen de las funciones propias definidas por SASS, también es posible definir funciones propias para que puedas utilizarlas en tus hojas de estilos.

Ejemplo:

```
$grid-width: 40px
$gutter-width: 10px

@function grid-width($n)
    @return $n * $grid-width + ($n - 1) * $gutter-width

#sidebar
    width: grid-width(5)
```

El ejemplo anterior se compila en el siguiente código CSS:

```
#sidebar {
    width: 240px;
}
```

Al igual que sucede con los *mixins*, las funciones pueden acceder a cualquier variable global y también pueden aceptar argumentos. El contenido de una función puede estar formado por varias líneas, pero siempre debe acabar con una directiva de tipo ***@return*** para devolver el resultado de su ejecución.

Para evitar posibles conflictos en el nombre de las funciones, es aconsejable añadirles un prefijo. Así además los usuarios sabrán claramente que esas funciones no forman parte ni de SASS ni de CSS.

Ejemplo:

```
@function -buzinger-grid-width($n)
    @return $n * $grid-width + ($n - 1) * $gutter-width
```

FORMATO DE SALIDA

El formato utilizado por SASS para compilar los archivos CSS no sólo es adecuado, sino que refleja bien la estructura del documento. No obstante, como los gustos (y las necesidades) de los diseñadores/as son muy particulares, SASS permite configurar cómo se generan los archivos.

En concreto, SASS permite elegir entre cuatro formatos diferentes mediante la opción de configuración **:style** o mediante la opción **--style** de la consola de comandos.

EL FORMATO :NESTED

Este es el estilo por defecto de SASS, que indenta y anida todos los selectores y estilos para reflejar fielmente la estructura del archivo SASS original. Cada propiedad se muestra en su propia línea y cada regla se indenta tanto como sea necesario en función de su anidamiento.

Ejemplo:

```
#main {
    color: #fff;
    background-color: #000; }

    #main p {
        width: 10em; }

.huge {
    font-size: 10em;
    font-weight: bold;
    text-decoration: underline; }
```

EL FORMATO :EXPANDED

Este estilo es más parecido al que utilizaría un diseñador/a al crear manualmente la hoja de estilos CSS. Cada propiedad y cada regla se muestran en una nueva línea, pero las reglas no se indentan de ninguna manera especial.

Ejemplo:

```
#main {
  color: #fff;
  background-color: #000;
}
#main p {
  width: 10em;
}

.huge {
  font-size: 10em;
  font-weight: bold;
  text-decoration: underline;
}
```

EL FORMATO :COMPACT

Este estilo ocupa menos líneas que los estilos *nested* o *expanded* y prioriza los selectores por encima de las propiedades. De hecho, cada regla CSS solamente ocupa una línea, donde se definen todas las propiedades. Las reglas anidadas se muestran seguidas unas de otras (sin ningún salto de línea) y solamente se añade una línea en blanco para separar los grupos de reglas CSS.

Ejemplo:

```
#main { color: #fff; background-color: #000; }
#main p { width: 10em; }

.huge { font-size: 10em; font-weight: bold; text-decoration:
underline; }
```

EL FORMATO :COMPRESSED

Este estilo es el más conciso de todos porque no añade ningún espacio en blanco, salvo el que sea estrictamente necesario para separar los selectores. El único salto de línea que se añade es el del final del archivo. Este formato también realiza otras optimizaciones y compresiones en valores como los colores. Aunque no está pensado como formato para que lo lean los humanos, puede ser muy útil para comprimir al máximo las hojas de estilos CSS antes de servirlas a los usuarios.

Ejemplo:

```
#main{color:#fff;background-color:#000}#main
p{width:10em}.huge{font-size:10em;font-weight:bold;text-
decoration:underline}
```

REFERENCIAS

- **SASS**: Hampton Catlin, Nathan Weizenbaum, Chris Eppstein. “SASS Basics”. Disponible en la web http://sass-lang.com/guide
- **SASS Guidelines**: Hugo Giraudel, Nathan. “SASS Guidelines”. Disponible en la web https://sass-guidelin.es/es/
- **FalconMasters**: Carlos Arturo. “Escribe CSS como un Pro, Curso Básico SASS”. Disponible en la web http://www.falconmasters.com/cursos/curso-basico-de-sass/
- **LibrosWeb**: Hampton Catlin, Nathan Weizenbaum, Chris Eppstein. “SASS, el manual oficial”.Disponible en la web https://librosweb.es/libro/sass/
- **CodeSchool**: Walsh Nick. “Assembling SASS”. Disponible en la web https://www.codeschool.com/courses/assembling-sass

www.guede.me