Contents

Características

Sass (acrónimo de Syntactically Awesome StyleSheets) es una extensión de CSS que añade características muy potentes y elegantes a este lenguaje de estilos. Sass permite el uso de variables, reglas CSS anidadas, mixins, importación de hojas de estilos y muchas otras características, al tiempo que mantiene la compatibilidad con CSS.

Sass permite organizar mejor las hojas de estilos grandes y permite ser mucho más productivo con las hojas de estilos pequeñas, sobre todo gracias a la librería compass.

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.

Sintaxis

Sass permite el uso de dos sintaxis diferentes para crear sus archivos. La primera, conocida como SCSS (del inglés, Sassy CSS) es la que vamos a utilizar en este manual y es una extensión de la sintaxis de CSS3. Esto significa que cualquier hoja de estilos CSS3 válida también es un archivo SCSS válido.

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.

Algunos diseñadores consideran 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. Los archivos creados con esta segunda sintaxis utilizan la extensión .sass.

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:

```
sass-convert estilos.sass estilos.scss

sass-convert estilos.scss estilos.sass
```

Usando Sass

Sass se puede usar de tres maneras diferentes:

1. En la consola de comandos
2. Como un módulo de Ruby
3. Como un plugin de cualquier *framework* compatible con Rack

Independientemente de cómo lo utilices, el primer paso siempre consiste en instalar Sass con el siguiente comando:

```
gem install sass
```

Si utilizas Windows, es posible que primero tengas que instalar Ruby.

Para utilizar Sass en la línea de comandos, simplemente ejecuta el comando sass:

```
sass hoja_estilos.scss archivo_generado.css
```

Si lo prefieres, también puedes añadir la opción –watch para decirle a Sass que vuelva a generar el archivo CSS cada vez que se cambie la hoja de estilos original:

```
sass --watch hoja_estilos.scss:archivo_generdo.css
```

Si dispones de un directorio con muchos archivos Sass, también puedes vigilarlos todos por si se producen cambios en alguno de ellos:

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

Extensiones CSS

Reglas anidadas

Sass permite anidar las reglas CSS para que las hojas de estilos sean más concisas y fáciles de escribir. A los selectores anidados se les prefija automáticamente todos los selectores de los niveles superiores. Ejemplo:

```
#main p {
  color: #00ff00;
  width: 97%;

  .redbox {
    background-color: #ff0000;
    color: #000000;
  }
}
```

El código Sass anterior se convierte automáticamente en el siguiente código CSS:

```
#main p {
  color: #00ff00;
  width: 97%;
}
#main p .redbox {
    background-color: #ff0000;
    color: #000000;
}
```

Gracias a las reglas anidadas, se evita tener que repetir una y otra vez los mismos selectores y se simplifica enormemente la creación de hojas de estilos complejas. Ejemplo:

```
#main {
  width: 97%;
```

```
  p, div {
    font-size: 2em;
    a { font-weight: bold; }
  }

  pre { font-size: 3em; }
}
```

El código Sass anterior se transforma 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;
}
```

Refinando a los selectores padre

En ocasiones es necesario modificar el comportamiento por defecto de los selectores anidados. Imagina que quieres aplicar estilos especiales en el estado hover del selector o cuando el elemento <body> de la página tiene una determinada clase.

En estos casos, puedes utilizar el carácter & para hacer referencia al selector padre dentro del cual se encuentra la regla anidada. Ejemplo:

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

El código sass anterior se compilará a:

```
a {
  font-weight: bold;
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
body.firefox a {
  font-weight: normal;
}
```

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; }
  }
}
```

El código Sass anterior se compila de la siguiente manera:

```
#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 se compila de la siguiente manera:

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

Si por cualquier circunstancia no se puede aplicar el sufijo al selector padre, Sass mostrará un mensaje de error indicándote la causa.

Propiedades anidadas

CSS define varias propiedades cuyos nombres paracen 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:

```
.funky {
  font: {
    family: fantasy;
    size: 30em;
    weight: bold;
  }
}
```

El código Sass anterior se compila de la siguiente manera:

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

También es posible aplicar un valor al propio nombre que agrupa las propiedades:

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

El código Sass anterior se compila de la siguiente manera:

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

Selectores variables

Sass también soporta un tipo especial de selector variable que se parece a los selectores de clase o de ID, pero que utiliza % en vez de # o .. No obstante, estos selectores variables solamente deberían usarse con la directiva @extend, tal y como se explica en los siguientes capítulos.

Si utilizas estos selectores sin la directiva @extend, el archivo CSS generado ignorará todas esas reglas Sass.

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; }
```

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. Esto es útil por ejemplo para mantener mensajes como el Copyright de tus hojas de estilos.

SassScript

Además de extender la sintaxis básica de CSS, Sass incluye una serie de extensiones más avanzadas llamadas SassScript. Gracias a estas extensiones, las propiedades pueden utilizar variables, expresiones matemáticas y otras funciones. Sass permite el uso de SassScript para definir cualquier valor de cualquier propiedad.

Shell interactiva

Si quieres experimentar con SassScript antes de empezar a utilizarlo en tus hojas de estilos, puedes hacer uso de "la shell interactiva". Para ello, ejecuta el comando sass añadiendo la opción -i y escribe cualquier expresión válida de SassScript. La shell te mostrará el resultado de evaluar esa expresión o un mensaje de error si no es correcta:

```
$ sass -i
>> "¡Hola Mundo!"
"¡Hola Mundo!"

>> 1px + 1px + 1px
3px

>> #777 + #777
#eeeeee

>> #777 + #888
white
```

Variables

La funcionalidad básica de SassScript es el uso de variables para almacenar valores que utilizas una y otra vez en tus hojas de estilos. Para ello, utiliza cualquier palabra como nombre de la variable, añádele el símbolo $ por delante y establece su valor como si fuera una propiedad CSS normal. Si por ejemplo defines una variable de la siguiente manera:

```
$width: 5em;
```

Ahora ya puedes utilizar la variable llamada $width como valor de cualquier propiedad CSS:

```
#main {
  width: $width;
}
```

Una limitación importante de las variables es que sólo están disponibles dentro del contexto donde se han definido. Esto significa que si defines la variable dentro de una regla anidada, sólo estará disponible para esas reglas anidadas. Si quieres poder utilizar una variable como valor de cualquier propiedad de la hoja de estilos, defínela fuera de cualquier selector.

Tipos de datos

SassScript soporta seis tipos de datos:

- Numeros

- Cadenas de texto con o sin comillas simples o dobles
- Colores
- Valores lógicos o booleanos
- Valores nulos (null)
- Lista de valores, separados por espacios en blanco o comas
- Pares formados por una clave y un valor separado por :

SassScript también soporta todos los otros tipos de datos soportados por CSS, como por ejemplo los caracteres Unicode o la palabra reservada !important. No obstante, Sass no trata estos valores de manera especial y se limita a considerarlos como si fuera una cadena de texto normal y corriente.

Cadenas de texto

CSS define dos tipos de cadenas de texto: las que tienen comillas (dobles o simples) como por ejemplo "Lucida Grande" o 'http://sass-lang.com'; y las que no tienen comillas, como por ejemplo sans-serif o bold.

SassScript soporta y reconoce estos dos tipos de cadenas. En general, el archivo CSS compilado mantendrá el mismo tipo de cadena que el que se utilizó en el archivo Sass original.

La única excepción es cuando se utiliza la interpolación #{} que se explica en los próximos capítulos. En este caso, las cadenas siempre se generan sin comillas. Ejemplo:

```
@mixin firefox-message($selector) {
  body.firefox #{$selector}:before {
    content: "Hi, Firefox users!";
  }
}

@include firefox-message(".header");
```

El código Sass anterior se compila a:

```
body.firefox .header:before {
  content: "Hi, Firefox users!";
}
```

Listas

Las listas son el tipo de dato que utiliza Sass para representar los valores que normalmente se utilizan en las propiedades CSS como margin: 10px 15px 0 0 o font-face: Helvetica, Arial, sans-serif. Las listas son simplemente una colección de valores separados por comas o espacios en blanco. Técnicamente, cada elemento de la lista también se considera una lista simple de un solo elemento.

Por si solas las listas no sirven para mucho, pero gracias a las funciones para listas definidas por SassScript que se explican en los siguientes capítulos, puedes conseguir resultados muy avanzados. La función nth() por ejemplo permite acceder al enésimo elemento de una lista, la función join() puede concatenar todos los valores y la función append() puede fusionar varias listas en una sola. Por último, la directiva @each permite aplicar estilos a cada elemento de una lista.

Además de contener valores simples, las listas pueden contener en su interior otras listas. Así por ejemplo, la lista 1px 2px, 5px 6px es una lista de dos elementos, que a su vez son las listas 1px 2px y 5px 6px. Si las listas interiores utilizan el mismo carácter para separar sus elementos que la lista principal, puedes añadir paréntesis para indicar claramente cuáles son los elementos de las listas anidadas. Así por ejemplo, la lista (1px 2px) (5px 6px) también es una lista de dos elementos cuyos valores son a su vez dos listas con los valores 1px 2px y 5px 6px.

Cuando se genera el archivo CSS, Sass no mantiene los paréntesis de las listas porque CSS no es capaz de entenderlos. Así que los valores (1px 2px) (5px 6px) y 1px 2px 5px 6px de Sass generan el mismo código cuando se compilan a CSS. No obstante, en Sass estos dos valores son diferentes: el primero es una lista que tiene dos listas en su interior y el segundo es una lista de cuatro números.

Las listas también pueden estar vacías y no contener ningún elemento. Estas listas vacías se representan mediante () y no se pueden incluir directamente en el archivo CSS compilado. Así que si defines una regla como font-family: (), Sass mostrará un mensaje de error. Si una lista contiene valores vacíos o nulos, como por ejemplo 1px 2px () 3px o 1px 2px null 3px, estos valores se eliminan antes de convertir la lista a código CSS.

Las listas separadas por comas pueden incluir una coma después del último elemento. Esto es muy útil por ejemplo para crear listas de un solo elemento. Así por ejemplo (1,) es una lista que contiene el elemento 1, mientras que (1 2 3,) es una lista separada por comas cuyo primer elemento es a su vez una lista separada por espacios en blanco y que contiene los

elementos 1, 2 y 3.

Mapas

Los mapas son asociaciones de claves y valores. La clave se utiliza para acceder fácilmente al valor de cualquier elemento del mapa. Se utilizan principalmente para agrupar valores y acceder a ellos dinámicamente. En CSS no existe ningún elemento equivalente a los mapas, pero su sintaxis es similar a las expresiones media query:

```
$map: (clave1: valor1, clave2: valor2, clave3: valor3);
```

A diferencia de las listas, los mapas siempre se encierran con paréntesis y los pares clave: valor deben separarse con comas. Tanto las claves como los valores de los mapas pueden utilizar cualquier función o expresión de SassScript. Las claves de un mapa deben ser únicas, por lo que si quieres asociar varios valores a una misma clave, debes utilizar una lista.

Al igual que sucede con las listas, los mapas se pueden manipular mediante funciones de SassScript. La función map-get() por ejemplo busca un valor dentro del mapa a partir de la clave indicada y la función $\text{map}_{\text{merge}}()$ añade nuevos pares clave: valor a un mapa existente. Además, la directiva @each se puede emplear para aplicar estilos a cada par clave: valor de un mapa.

Los mapas también se pueden utilizar en cualquier función preparada para manipular listas. Si pasas un mapa a una función que espera una lista, el mapa se transforma primero en un lista de pares de valores. Así por ejemplo, si pasas el mapa (clave1: valor1, clave2: valor2) a una función para listas, este se transforma automáticamente en clave1 valor1, clave2 valor2. Lo contrario no es cierto, ya que no puedes utilizar listas en las funciones preparadas para mapas. La única excepción es la lista vacía (), que representa tanto a un mapa vacío como a una lista vacía.

Los mapas no se pueden convertir directamente a código CSS. Por tanto, si utilizar un mapa como valor de una variable o como argumento de una función CSS, Sass mostrará un mensaje de error.

Operadores

Todos los tipos de datos soportan el operador de igualdad (`= y !`) para comprobar si dos valores son iguales o distintos. Además, cada tipo de dato define otros operadores propios.

Operadores para números

SassScript soporta los cinco operadores aritméticos básicos: suma +, resta -, multiplicación *, división / y módulo %. El operador módulo calcula el resto de la división sin decimales (ejemplo: 5 módulo 2 = 1, % % 2 = 1). Además, si realizas operaciones sobre números con diferentes unidades, Sass convertirá automáticamente las unidades siempre que sea posible:

```
p {
  width: 1in + 8pt;
}
```

El código css resultante será:

```
p {
  width: 1.111in;
}
```

Con los números también se pueden utilizar los operadores relacionales (<, >, <=, >=) y los de igualdad (`=, !`).

- El problema del carácter / con la división de 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 SassScript, 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 código css que genera es:

```
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 código css que genera es:

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

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;
}
```

En la mayoría de los casos, es mejor utilizar las funciones especiales de SassScript para colores que se explicarán más adelante, en vez de realizar operaciones matemáticas sobre ellos.

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:

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

El código css resultantes será:

```
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 css resultante:

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

Por defecto, si dos valores son contiguos, se concatenan con un espacio en blanco:

```
p {
  margin: 3px + 4px auto;
}
```

Esto dará el siguiente css:

```
p {
  margin: 7px auto;
}
```

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**":

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

El código css resultante:

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

Cuando interpolas una cadena de texto, los valores nulos se consideran cadenas vacías:

```
$value: null;

p:before {
  content: "¡Me he comido #{$valor} pasteles!";
}
```

Esto dará:

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

Operadores para valores lógicos o booleanos

SassScript soporta el uso de los tradicionales operadores and, or y not sobre los valores lógicos o booleanos.

Operadores para listas

Sass no define ningún operador específico para las listas de elementos, ya que estas se manipulan mediante las funciones especiales que se explican en los siguientes capítulos.

Paréntesis

Puedes añadir paréntesis a cualquier expresión Sass para afectar al orden en el que se realizan las operaciones:

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

El código Sass se compila:

```
p {
  width: 7em;
}
```

Funciones

SassScript define algunas funciones muy útiles para crear las hojas de estilos y que utilizan la misma sintaxis que CSS:

```
p {
  color: hsl(0, 100%, 50%);
}
```

Esto se compila:

```
p {
  color: #ff0000;
}
```

Argumentos con nombre

Para que su uso sea más flexible, a las funciones de Sass les puedes pasar argumentos con nombre. De esta manera no es obligatorio respetar el orden en el que se definieron los argumentos, sólo su nombre:

```
p {
  color: hsl($hue: 0, $saturation: 100%, $lightness: 50%);
}
```

Aunque obviamente esta forma de usar las funciones no es tan concisa, hace que las hojas de estilo resultantes sean mucho más fáciles de leer. Además permite que las funciones tengan interfaces más flexibles y fáciles de usar, aún cuando incluyan muchos argumentos.

Los argumentos con nombre se pueden pasar en cualquier orden y puedes omitir los que tienen un valor por defecto. Además, como los argumentos con nombre en realidad son nombres de variables, puedes utilizar indistintamente guiones medios y bajos.

En los próximos capítulos se detalla la lista completa de funciones Sass y los nombres de todos sus argumentos.

Interpolación

Las variables definidas con SassScript se pueden utilizar incluso en los nombres de los selectores y de las propiedades:

```
$name: foo;
$attr: border;

p.#{$name} {
  #{$attr}-color: blue;
}
```

Que dará lugar a:

```
p.foo {
  border-color: blue;
}
```

También es posible usar #{ } en los valores de las propiedades. Normalmente es mejor utilizar una variable, pero la ventaja de usar #{ } es que todas las operaciones que estén cerca suyo se interpretan como código CSS normal y corriente. Ejemplo:

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

Dando lugar a:

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

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;
}
```

Que dará el siguiente código css:

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

Al utilizar !default, las variables con valores nulos se considera que no han sido asignadas:

```
$contenido: null;
$contenido: "Contenido no nulo" !default;

#main {
  contenido: $contenido;
}
```

El código Sass anterior se compila de la siguiente manera:

```
#main {
  contenido: "Contenido no nulo";
}
```

Reglas @ y directivas

Sass soporta todas las reglas @ (también llamadas "reglas at") definidas por CSS3. Además, Sass incluye varias reglas específicas llamadas directivas.

La regla @import

Sass mejora la regla @import de CSS para poder importar también archivos SCSS y Sass. Todos los archivos importados, independientemente de su tipo, acaban fusionándose antes de generar el archivo CSS final. Además, cualquier variable o mixin definidos en los archivos importados se pueden utilizar en la hoja de estilos principal.

Los archivos importados se buscan automáticamente en el directorio actual. Utiliza la opción de configuración :load$_{\text{paths}}$ para configurar todos los directorios adicionales en los que quieras buscar archivos. También puedes utilizar la opción –load-path del comando sass.

La regla @import espera como argumento el nombre del archivo a importar. Por defecto busca un archivo Sass y lo importar directamente, pero a veces esta regla se deja tal cual al compilar el archivo CSS:

- Si la extensión del archivo importado es .css
- Si el nombre del archivo empieza por `http://`
- Si el nombre del archivo se indica mediante url()
- Si la regla @import tiene alguna *media query*

Si no se da ninguna de las anteriores circunstancias, y la extensión del archivo importado es .scss o .sass, entonces se importan directamente los contenidos de ese archivo. Si no se indica la extensión, Sass tratará de buscar un archivo con ese nombre y con las extensiones .scss o .sass. Ejemplos:

Regla @	Resultado
@import "foo.scss";	Se importa el archivo foo.scss
@import "foo";	Se importa el archivo foo.scss
@import "foo.css";	@import "foo.css";
@import "foo" screen;	@import "foo" screen
@import "`http://foo.com/bar`";	@import "`http://foo.com/bar`";
@import url(foo)	@import url(foo)

También es posible importar varios archivos con una sola regla **@import**. Ejemplo:

```
@import "rounded-corners", "text-shadow";
```

El nombre del archivo importado también se puede establecer con la interpolación #{ }, pero con ciertas restricciones. No se puede importar dinámicamente un archivo Sass en base al nombre de una variable, pero sí que se puede importar de esta manera un archivo CSS. De forma que la interpolación solamente funciona en la práctica cuando se utiliza url(). Ejemplo:

```
$family: unquote("Droid+Sans");
@import url("http://fonts.googleapis.com/css?family=#{$family}");
```

Esto resultarán en:

```
@import url("http://fonts.googleapis.com/css?family=Droid+Sans");
```

Hojas de estilo parciales

Si quieres importar un archivo SCSS o Sass pero no quieres que se compile como archivo CSS, utiliza un guión bajo como primer carácter del nombre del archivo. De esta manera, Sass no generará un archivo CSS para esa hoja de estilos, pero podrás utilizarla importándola dentro de otra hoja de estilos. Este tipo de archivos que no se compilan se denominan "hojas de estilos parciales" o simplemente "parciales" (en inglés, "partials").

Aunque el nombre del archivo tenga un guión bajo, no es necesario indicarlo en la regla @import. Así por ejemplo, si creas un archivo llamado _colors.scss, entonces no se generará un archivo _colors.css. Sin embargo, podrás utilizarlo en tus hojas de estilos con la regla @import "colors";, que importará el archivo _colors.scss.

Obviamente no puedes tener en un mismo directorio una hoja de estilos normal y una parcial con el mismo nombre. Siguiendo el ejemplo anterior, en el mismo directorio no puedes tener un archivo llamado _colors.scss y otro llamado colors.scss.

Anidando reglas @import

Normalmente las reglas @import se colocan en el primer nivel jerárquico de la hoja de estilos. No obstante, también es posible colocarlas dentro de reglas CSS y reglas @media.

El funcionamiento de las reglas anidadas es el mismo, pero todos los contenidos importados se incluyen en el mismo nivel en el que se hayan importado. Si por ejemplo el archivo example.scss contiene lo siguiente:

```
.example {
  color: red;
}
```

Si importas este archivo dentro de una regla CSS:

```
#main {
  @import "example";
}
```

El archivo css compilado será:

```
#main .example {
  color: red;
}
```

Los archivos importados con reglas @import anidadas no pueden contener elementos y directivas que sólo pueden colocarse en el primer nivel jerárquico de las hojas de estilos, como @mixin o @charset.

Tampoco es posible anidar reglas @import dentro de los mixin y las directivas de control.

La regla @media

Las reglas @media en Sass funcionan prácticamente igual que en CSS, con una salvedad: se pueden anidar dentro de las reglas CSS. Si incluyes una regla @media dentro de una regla CSS, se aplicará a todos los selectores que se encuentren desde esa regla hasta el primer nivel de la hoja de estilos. Esto hace que sea muy fácil definir estilos dependientes de los dispositivos sin tener que repetir los selectores y sin tener que romper el flujo normal de la hoja de estilos Sass. Ejemplo:

```
.sidebar {
  width: 300px;
  @media screen and (orientation: landscape) {
    width: 500px;
  }
}
```

El código dará como resultado:

```
.sidebar {
  width: 300px;
}

@media screen and (orientation: landscape) {
  .sidebar {
    width: 500px;
  }
}
```

Las reglas @media también se pueden anidar entre sí. El resultado la combinación de todas ellas utilizando el operador and. Ejemplo:

```
@media screen {
  .sidebar {
    @media (orientation: landscape) {
      width: 500px;
    }
  }
}
```

Dará lugar a:

```
@media screen and (orientation: landscape) {
  .sidebar {
    width: 500px;
  }
}
```

Por último, las reglas @media también pueden contener expresiones SassScript (incluyendo variables, funciones y operadores) tanto en los nombres como en los valores. Ejemplo:

```
$media: screen;
$feature: -webkit-min-device-pixel-ratio;
$value: 1.5;

@media #{$media} and ($feature: $value) {
  .sidebar {
    width: 500px;
  }
}
```

El código Sass anterior se compila de la siguiente manera:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5) {
  .sidebar {
    width: 500px;
  }
}
```

La regla @extend

En ocasiones, es necesario que una clase CSS contenga todos los estilos aplicados a otra regla CSS, además de sus propios estilos. La solución habitual

en estos casos consiste en crear una clase genérica que puedan utilizar los dos elementos. Imagina que quieres aplicar estilos a dos tipos de mensajes de error diferentes, uno normal y otro más grave. El código HTML podría ser algo como:

```
<div class="error seriousError">
  ¡Acabas de ser hackeado!
</div>
```

Los estilos CSS podrían ser los siguientes:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  border-width: 3px;
}
```

El problema de esta solución es que tienes que acordarte que siempre que apliques la clase .seriousError también tienes que aplicar la clase .error. Esto hace que el mantenimiento de las hojas de estilos se complique y el código HTML de las páginas se complique sin una justificación clara.

Gracias a la regla @extend puedes evitar todos estos problemas. Esta regla le indica a Sass que un determinado selector debería heredar todos los estilos de otro selector. Ejemplo:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

El código anterior se compila de la siguiente manera:

```
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
```

```
}

.seriousError {
  border-width: 3px;
}
```

Ahora, todos los estilos que definas para el selector .error también se aplican automáticamente al selector .seriousError, al margen de los estilos propios que pueda definir .seriousError. En la práctica esto significa que cuando apliques la clase .seriousError es como si estuvieras aplicando a la vez la clase .error.

Cualquier otra regla que se aplique al selector .error también se aplicará al selector .seriousError. Imagina que defines el siguiente estilo que se aplica simultáneamente a dos clases CSS:

```
.error.intrusion {
  background-image: url("/image/hacked.png");
}
```

Si ahora añades en tus páginas un elemento como , también se le aplicará el estilo definido por el selector .error.intrusion.

Funcionamiento interno

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.error.intrusion {
  background-image: url("/image/hacked.png");
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

Este código Sass dará el siguiente código css:

```
.error, .seriousError {
  border: 1px #f00;
```

```
  background-color: #fdd;
}

.error.intrusion, .seriousError.intrusion {
  background-image: url("/image/hacked.png");
}

.seriousError {
  border-width: 3px;
}
```

Al combinar los selectores, la regla @extend es lo bastante inteligente como para evitar las duplicidades innecesarias (un selector como .seriousError.seriousError se transforma automáticamente en .seriousError). También tiene en cuenta los selectores que nunca podrían seleccionar ningún elemento, como por ejemplo #main#footer.

Extendiendo selectores complejos

Además de los selectores de clase, Sass permite extender cualquier otro elemento que haga referencia a un único elemento, como por ejemplo .special.cool, a:hover o a.user[hrefˆ="http://"]. Ejemplo:

```
.hoverlink {
  @extend a:hover;
}
```

Al igual que en el caso de los selectores de clase, este estilo implica que todos los estilos definidos para el selector a:hover también se aplicarán al selector .hoverlink. Ejemplo:

```
.hoverlink {
  @extend a:hover;
}
a:hover {
  text-decoration: underline;
}
```

El código anterior compilará en:

```
a:hover, .hoverlink {
  text-decoration: underline;
}
```

Al igual que sucedía antes con el selector .error.intrusion, cualquier regla que utilice el selector a:hover también funcionará para el selector .hoverlink, incluso cuando se combinan con otros selectores. Ejemplo:

```
.hoverlink {
  @extend a:hover;
}
.comment a.user:hover {
  font-weight: bold;
}
```

El código Sass anterior compilará en:

```
.comment a.user:hover, .comment .user.hoverlink {
  font-weight: bold;
}
```

Extendiendo de varios selectores

Los selectores pueden extender de más de un selector para heredar todos sus estilos. Ejemplo

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.attention {
  font-size: 3em;
  background-color: #ff0;
}
.seriousError {
  @extend .error;
  @extend .attention;
  border-width: 3px;
}
El código Sass anterior se compila de la siguiente manera:

.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
}
```

```
.attention, .seriousError {
  font-size: 3em;
  background-color: #ff0;
}

.seriousError {
  border-width: 3px;
}
```

En este ejemplo, cualquier elemento con la clase .seriousError es como si también tuviera aplicadas las clases .error y .attention. Como importa el orden en el que se extienden los selectores, el selector .seriousError tiene un color de fondo igual a #ff0 en vez de #fdd, ya que .attention se define después que .error.

La extensión de más de un selector también se puede indicar mediante una lista de selectores separados por comas. Así por ejemplo, el código @extend .error, .attention es equivalente a @extend .error; @extend.attention.

Extendiendo a varios niveles

Sass también permite extender de un selector que a su vez extiende de otro selector diferente. Ejemplo:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
.criticalError {
  @extend .seriousError;
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%;
}
```

Ahora aplicar la clase .seriousError equivale también a aplicar la clase .error y la clase .criticalError equivale a aplicar también las clases .seriousError y .error. El código Sass anterior se compila de la siguiente manera:

```
.error, .seriousError, .criticalError {
  border: 1px #f00;
  background-color: #fdd;
}

.seriousError, .criticalError {
  border-width: 3px;
}

.criticalError {
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%;
}
```

Secuencia de selectores

Las secuencias de selectores, como por ejemplo .foo .bar o .foo + .bar, todavía no se pueden extender. No obstante, sí que es posible utilizar la regla @extend en los selectores anidados. Ejemplo:

```
#fake-links .link {
  @extend a;
}

a {
  color: blue;
  &:hover {
    text-decoration: underline;
  }
}
```

El código Sass anterior se compila de la siguiente manera:

```
a, #fake-links .link {
  color: blue;
}
a:hover, #fake-links .link:hover {
  text-decoration: underline;
}
```

- Combinando secuencias de selectores

  En ocasiones una secuencia de selectores extiende otro selector que está incluido en otra secuencia de selectores. En este caso, se combinan las dos secuencias de selectores. Ejemplo:

  ```
  #admin .tabbar a {
    font-weight: bold;
  }
  #demo .overview .fakelink {
    @extend a;
  }
  ```

  Aunque técnicamente sería posible generar todos los selectores resultantes de combinar todos los selectores entre sí, esto haría que la hoja de estilos resultante fuera demasiado larga. Un código tan sencillo como el mostrado anteriormente generaría por ejemplo diez selectores. Así que en vez de generar todas las combinaciones posibles, Sass solamente genera aquellos selectores que probablemente van a ser de utilidad.

  Cuando las dos secuencias que se van a combinar no tienen selectores en común, entonces se generan dos nuevos selectores: uno con la primera secuencia por delante de la segunda y otro con la segunda secuencia por delante de la primera. Ejemplo:

  ```
  #admin .tabbar a {
    font-weight: bold;
  }
  #demo .overview .fakelink {
    @extend a;
  }
  ```

  El código resultante:

```
#admin .tabbar a,
#admin .tabbar #demo .overview .fakelink,
#demo .overview #admin .tabbar .fakelink {
  font-weight: bold;
}
```

Si las dos secuencias tienen algunos selectores en común, se combinan esos selectores y las diferencias, si exsten, se alternan. En el siguiente ejemplo, las dos secuencias tienen el selector #admin, así que los selectores resultantes serán el resultado de combinar esos dos selectores de id:

```
#admin .tabbar a {
  font-weight: bold;
}
#admin .overview .fakelink {
  @extend a;
}
El código Sass anterior se compila de la siguiente manera:

#admin .tabbar a,
#admin .tabbar .overview .fakelink,
#admin .overview .tabbar .fakelink {
  font-weight: bold;
}
```

Selectores exclusivos para reglas @extend

Las hojas de estilos también pueden contener clases que no se utilizan directamente en el código HTML y que sólo se definen para agrupar estilos que luego se utilizan mediante reglas @extend. Esto es común cuando se escriben librerías para Sass, ya que puede ser interesante ofrecer a los diseñadores la posibilidad de extender o ignorar algunas clases en sus estilos.

Si utilizaras clases normales, acabarías generando un código CSS demasiado grande y poco optimizado. Incluso correrías el peligro de generar colisiones con otras clases que sí que se utilizan en el código HTML. Por este motivo Sass soporta los selectores variables con la sintaxis %foo.

Los selectores variables (en inglés, "placeholder parameters") se parecen a los selectores de clase o de id, pero utilizan el carácter % en vez de . o #. Estos nuevos selectores se pueden utilizar en cualquier lugar en el que

utilices los selectores de clase o de id y están preparados para no generar código CSS al compilar las hojas de estilos. Ejemplo:

```
// Este estilo no se incluirá en el archivo CSS compilado
#context a%extreme {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

La ventaja de los selectores variables es que se pueden extender, de la misma manera que el resto de selectores. Ejemplo:

```
.notice {
  @extend %extreme;
}
```

El código Sass anterior se compila de la siguiente manera:

```
#context a.notice {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

La opción !optional

Cuando extiendes un selector que no existe, Sass genera un error. Si utilizas por ejemplo el código a.important {@extend .notice} pero no existe el selector .notice, entonces se produce un error. También se produciría un error si el único selector que contiene la clase .notice fuera h1.notice, ya que h1 entraría en conflicto con a y no se generaría ningún selector.

No obstante, en ocasiones puede ser útil permitir que @extend no genere ningún selector. Para ello, añade la opción !optional justo después del selector. Ejemplo:

```
a.important {
  @extend .notice !optional;
}
```

Usando @extend en las directivas

Existen algunas restricciones que impiden usar @extend en el interior de directivas como @media. Sass por ejemplo no es capaz de hacer que las reglas CSS que se encuentran fuera de la directiva @media se apliquen a los selectores de su interior sin generar un código CSS gigantesco con selectores y estilos duplicados por todas partes. Por lo tanto, si utilizas @extend con la directiva @media o con otras directivas CSS, sólo debes extender los selectores que están encerrados por esas directivas.

El siguiente ejemplo funciona correctamente:

```
@media print {
  .error {
    border: 1px #f00;
    background-color: #fdd;
  }
  .seriousError {
    @extend .error;
    border-width: 3px;
  }
}
```

Pero el siguiente produciría un error:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}

@media print {
  .seriousError {
    // ESTILO INVÁLIDO: .error se utiliza fuera de la directiva "@media print"
    @extend .error;
    border-width: 3px;
  }
}
```

La regla @at-root

Las directivas @at-root hacen que una o más reglas se generen en la raíz de la hoja de estilos en vez de anidarse en sus selectores. Se puede utilizar tanto con selectores individuales como con bloques de selectores. Ejemplo:

```
// selector individual
.parent {
  @at-root .child { ... }
}

// bloques de selectores
.parent {
  @at-root {
    .child1 { ... }
    .child2 { ... }
  }
}
```

El código anterior Sass se compila de la siguiente manera:

```
.child { ... }

.child1 { ... }
.child2 { ... }
```

Modificando la regla @at-root con with y without

Por defecto la regla @at-root simplemente excluye todos los selectores. No obstante, también es posible modificar su comportamiento para que salga o no de cualquier directiva @media en la que se encuentre esa regla. Ejemplo:

```
@media print {
  .page {
    width: 8in;
    @at-root (without: media) {
      color: red;
    }
  }
}
```

Esto compila en:

```
@media print {
  .page {
    width: 8in;
  }
```

```
}
.page {
  color: red;
}
```

La regla @at-root (without: ...) hace que el estilo se aplique en la raíz de la hoja de estilos y fuera de cualquier media query. También es posible excluir varias directivas separándolas con espacios en blanco: @at-root (without: media supports) saca los estilos fuera de las queries @media y @supports.

La regla @at-root admite otros dos valores especiales. El valor rule se refiere a las reglas CSS normales, por lo que @at-root (without: rule) es equivalente a @at-root sin ninguna query. Por su parte, la regla @at-root (without: all) significa que los estilos deben sacarse de cualquier directiva o regla CSS.

Si en vez de indicar las directivas o reglas CSS que se excluyen quieres indicar explícitamente las que se inlcuyen, utiliza with en vez de without. Así por ejemplo, los estilos @at-root (with: rule) se moverán fuera de cualquier directiva pero mantendrán todas las reglas CSS.

La regla @debug

La regla @debug muestra por la consola el valor de la expresión SassScript indicada. Se trata de una regla útil para depurar hojas de estilos muy complejas y que utilizan expresiones SassScript muy avanzadas. Ejemplo:

```
@debug 10em + 12em;
```

El código anterior mostraría en la consola el siguiente mensaje:

```
Line 1 DEBUG: 22em
```

La regla @warn

La regla @warn muestra el valor de una expresió SaasScript en forma de mensaje de error. Se trata de una regla muy útil para que los creadores de las librerías avisen a los diseñadores sobre el uso de características que se han declarado obsoletas. También sirve para mostrar errores en el uso de mixins que Sass ha podido corregir automáticamente. Existen dos diferencias principales entre @warn y @debug:

1. Puedes desactivar los mensajes de error con la opción –**quiet** de la línea de comandos o con la opción de configuración :quiet de Sass

2. Los mensajes de error de @warn también se incluyen en la hoja de estilos generada para que el usuario pueda ver tanto los errores como el lugar exacto en el que se producen.

Ejemplo:

```
@mixin adjust-location($x, $y) {
  @if unitless($x) {
    @warn "Assuming #{$x} to be in pixels";
    $x: 1px * $x;
  }
  @if unitless($y) {
    @warn "Assuming #{$y} to be in pixels";
    $y: 1px * $y;
  }
  position: relative; left: $x; top: $y;
}
```

Directivas de control y expresiones

SassScript 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.

NOTA: Las directivas de control son una característica muy avanzada que rara vez se utiliza directamente en las hojas de estilos. Sin embargo, son muy útiles para definir mixins y otras características avanzadas de librerías como Compass.

La función if()

La functión if() permite tomar decisiones para que una hoja de estilos incluya unos u otros estilos en función de unas determinadas condiciones. La función if() solamente evalúa el argumento que corresponde al valor que va a devolver, por lo que en el otro valor puedes hacer referencia a variables que no existen o realizar cálculos que en circunstancias normales causarían algún error (como por ejemplo dividir por cero).

La directiva @if

La directiva @if evalúa una expresión SassScript 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 anterior compila en:

```
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;
  }
}
```

Esto va a compilar en:

```
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: @for $var from <inicio> through <final> and @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 SassScript 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; }
}
```

Esto compilará en:

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

La directiva @each

La sintaxis habitual de la directiva @each es la siguiente @each $var in <lista o mapa>. El valor $var puede ser cualquier variable y <lista o mapa> es una expresión **SassScript*** que devuelve una lista o un mapa.

El funcionamiento de @each es el siguiente: se recorre toda la lista o mapa y en cada iteración, se asigna un valor diferente a la variable $var antes de compilar los estilos. Ejemplo:

```
@each $animal in puma, sea-slug, egret, salamander {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}
```

Esto compila en:

```
.puma-icon {
  background-image: url('/images/puma.png');
}
.sea-slug-icon {
  background-image: url('/images/sea-slug.png');
}
.egret-icon {
  background-image: url('/images/egret.png');
}
.salamander-icon {
  background-image: url('/images/salamander.png');
}
```

Asignación múltiple

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;
  }
}
```

Esto compila en:

```
.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;
}
```

Como los mapas se consideran listas formadas por pares clave: valor, también en este caso se puede utilizar la asignación múltiple. Ejemplo:

```
@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) {
  #{$header} {
    font-size: $size;
  }
}
```

El código Sass compilará en:

```
h1 {
  font-size: 2em;
}
h2 {
  font-size: 1.5em;
}
h3 {
  font-size: 1.2em;
}
```

La directiva @while

La directiva @while toma una expresión SassScript 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;
}
```

Esto compilará en:

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

.item-4 {
  width: 8em;
}

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

Directivas mixin

Los *mixins* permiten definir estilos reutilizables en toda la hoja de estilos sin tener que recurrir a clases CSS no semánticas tipo .float-left. Los *mixins* también pueden contener reglas CSS y cualquier otro elemento definido por Sass. Los mixins incluso admiten el uso de argumentos, como si fueran funciones, para poder modificar su comportamiento y ofrecer así una mayor flexibilidad.

Definiendo mixins con la directiva @mixin

Los mixins se definen con la directiva @mixin seguida del nombre del mixin (y opcionalmente una lista de argumentos) y seguida por el bloque de contenidos que definen los estilos del mixin. El siguiente ejemplo define un mixin sin argumentos llamado large-text:

```
@mixin large-text {
```

```
  font: {
    family: Arial;
    size: 20px;
    weight: bold;
  }
  color: #ff0000;
}
```

Además de estilos, los mixins también pueden contener selectores, incluso con referencias al selector padre. Ejemplo:

```
@mixin clearfix {
  display: inline-block;
  &:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
  }
  * html & { height: 1px }
}
```

Incluyendo mixins con @include

Los mixins se incluyen en las hojas de estilos mediante la directiva @include seguida del nombre del mixin y opcionalmente por una lista de argumentos. El resultado es que todos los estilos definidos por el mixin se incluyen en el mismo punto en el que se llama al mixin. Ejemplo:

```
.page-title {
  @include large-text;
  padding: 4px;
  margin-top: 10px;
}
```

Esto compila en:

```
.page-title {
  font-family: Arial;
  font-size: 20px;
```

```
  font-weight: bold;
  color: #ff0000;
  padding: 4px;
  margin-top: 10px;
}
```

Los mixins también se pueden incluir en el nivel jerárquico superior de la hoja de estilos, es decir, fuera de cualquier selector o regla. Obviamente, estos mixins no pueden incluir ninguna referencia al selector padre, ya que se produciría un error. Ejemplo:

```
@mixin silly-links {
  a {
    color: blue;
    background-color: red;
  }
}

@include silly-links;
```

Esto se compila de la siguiente manera:

```
a {
  color: blue;
  background-color: red;
}
```

Los mixins también pueden incluir en su interior otros mixins. Ejemplo:

```
@mixin compound {
  @include highlighted-background;
  @include header-text;
}

@mixin highlighted-background { background-color: #fc0; }
@mixin header-text { font-size: 20px; }
```

Aunque no es muy habitual, los mixins también pueden incluirse a sí mismos de manera recursiva. En las versiones de Sass anteriores a la 3.3 esta recursividad no estaba permitida.

Argumentos

Los argumentos de los mixins pueden estar formados por cualquier expresión SassScript. Estos argumentos están disponibles en el interior del mixin en forma de variables.

Cuando se define un mixin, los argumentos se definen como una serie de variables separadas por comas, y todo ello encerrado entre paréntesis. Después, cuando se utiliza un mixin deben pasarse los valores de los argumentos en ese mismo orden. Ejemplo:

```
@mixin sexy-border($color, $width) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}

p { @include sexy-border(blue, 1in); }
```

Este código se compila como:

```
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed;
}
```

Los mixins también pueden especificar valores por defecto para sus argumentos. De esta manera, si al llamar a un mixin no se pasa el valor de ese argumento, se utiliza en su lugar el valor por defecto. Ejemplo:

```
@mixin sexy-border($color, $width: 1in) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
p { @include sexy-border(blue); }
h1 { @include sexy-border(blue, 2in); }
```

Esto compila en:

```
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed;
}

h1 {
  border-color: blue;
  border-width: 2in;
  border-style: dashed;
}
```

Argumentos con nombre

Cuando se utiliza un mixin también es posible indicar el nombre de sus argumentos:

```
p { @include sexy-border($color: blue); }
h1 { @include sexy-border($color: blue, $width: 2in); }
```

Aunque esta sintaxis es menos concisa que la anterior, hace que las hojas de estilos sean más fáciles de leer. Además permite que los mixins tengan interfaces más flexibles y fáciles de usar, aún cuando incluyan muchos argumentos.

Los argumentos con nombre se pueden pasar en cualquier orden y puedes omitir los que tienen un valor por defecto. Además, como los argumentos con nombre en realidad son nombres de variables, puedes utilizar indistintamente guiones medios y bajos.

Argumentos variables

En ocasiones es necesario que un mixin acepte un número indeterminado de argumentos. Si por ejemplo tienes un mixin que añade sombras a los elementos HTML, es preciso que ese mixin acepte cualquier número de sombras como argumentos. Por eso Sass soporta la creación de mixins con un número variable de argumentos.

Para indicar que un mixin tiene un número variable de argumentos, después del último argumento se añaden tres puntos (...). Esto hará que todos los argumentos sobrantes se guarden como una lista en ese último argumento. Ejemplo:

```
@mixin box-shadow($shadows...) {
  -moz-box-shadow: $shadows;
  -webkit-box-shadow: $shadows;
  box-shadow: $shadows;
}

.shadows {
  @include box-shadow(0px 4px 5px #666, 2px 6px 10px #999);
}
```

Este código da como resultado de la compilación:

```
.shadows {
  -moz-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  -webkit-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
}
```

Los argumentos variables también contienen todos los argumentos con nombre pasados al mixin o función. Puedes acceder a ellos mediante la función keywords($args), que devuelve un mapa de cadenas de texto en las que el nombre de la variable no contiene el carácter $.

Los argumentos variables también se pueden utilizar cuando se llama a un mixin. Utilizando la misma sintaxis de los tres puntos (...) puedes expandir una lista de valores para pasar cada elemento de la lista como si fuera un argumento. Cuando esta sintaxis se utiliza con mapas, cada par clave: valor se transforma en un argumento con el nombre clave. Ejemplo:

```
@mixin colors($text, $background, $border) {
  color: $text;
  background-color: $background;
  border-color: $border;
}

$values: #ff0000, #00ff00, #0000ff;
.primary {
  @include colors($values...);
}

$value-map: (text: #00ff00, background: #0000ff, border: #ff0000);
```

```
.secondary {
  @include colors($value-map...);
}
```

Esto compila a:

```
.primary {
  color: #ff0000;
  background-color: #00ff00;
  border-color: #0000ff;
}

.secondary {
  color: #0000ff;
  background-color: #ff0000;
  border-color: #00ff00;
}
```

También es posible pasar una lista de argumentos y un mapa siempre que la lista se pase primero, como por ejemplo: @include colors($values..., $map...).

Los argumentos variables pueden servir por ejemplo para crear un mixin que modifique otro mixin existente añadiendo nuevos estilos. Ejemplo:

```
@mixin wrapped-stylish-mixin($args...) {
  font-weight: bold;
  @include stylish-mixin($args...);
}

.stylish {
  // El argumento $width se pasa con nombre al mixin "stylish-mixin"
  @include wrapped-stylish-mixin(#00ff00, $width: 100px);
}
```

Pasando bloques de contenidos a los mixins

A los mixins también se les puede pasar un bloque entero de reglas CSS. Este contenido se incluirá en el lugar donde el mixin haya definido la directiva @content. Gracias a esta característica es posible abstraer ciertas partes de la definición de los selectores y directivas. Ejemplo:

```
@mixin apply-to-ie6-only {
  * html {
    @content;
  }
}
@include apply-to-ie6-only {
  #logo {
    background-image: url(/logo.gif);
  }
}
```

Esto compila en:

```
 html #logo {
  background-image: url(/logo.gif);
}
```

Estos mixins también se pueden definir mediante los siguientes atajos:

```
    @content

+apply-to-ie6-only
  #logo
    background-image: url(/logo.gif)
```

NOTA: Cuando se incluye la directiva @content más de una vez o se incluye dentro de un bucle, los contenidos se repiten para cada aparición de @content.

Contexto variables y bloques contenidos

Los bloques de contenidos pasados a los mixins se evalúan en el contexto en el que están definidos, no en el contexto del mixin. Esto significa que los bloques de contenidos no pueden utilizar las variables locales definidas en el mixin. Ejemplo:

```
$color: white;
@mixin colors($color: blue) {
  background-color: $color;
```

```
  @content;
  border-color: $color;
}
.colors {
  @include colors { color: $color; }
}
```

Que compila a:

```
.colors {
  background-color: blue;
  color: white;
  border-color: blue;
}
```

De esta forma, las variables que se utilizan en los bloques que se pasan a los mixins siempre hacen referencia a las variables definidas alrededor de ese bloque o directamente en el nivel jerárquico superior de la hoja de estilos. Ejemplo:

```
#sidebar {
  $sidebar-width: 300px;
  width: $sidebar-width;
  @include smartphone {
    width: $sidebar-width / 3;
  }
}
```

Directivas de función

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 código Sass anterior se compila de la siguiente manera:

```
#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.

Las funciones propias también admiten el uso de argumentos con nombre. De hecho, la función del ejemplo anterior también se puede utilizar de la siguiente manera:

```
#sidebar { width grid-width($n: 5);}
```

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. Una buena idea consiste en utilizar como prefijo tu nombre o el de tu empresa. Si trabajas por ejemplo para la empresa ACME S.A., la función anterior podría haberse llamado -acme-grid-width.

Por último, las funciones propias también soportan el uso de un número variable de argumentos, tal y como se explicó en el capítulo de los mixins.

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.

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 estilo nested es muy útil cuando se generan hojas de estilos CSS muy complejas, ya que de un vistazo puedes entender toda su estructura.

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-wei
```

Extendiendo Sass

Sass proporciona una serie de características adicionales para usuarios que tengan requerimientos muy especiales y dispongan de conocimientos avanzados de Ruby.

Definiendo funciones propias para Sass

Utilizando la API de Ruby es posible definir tus propias funciones. Consulta la documentación oficial para saber cómo hacerlo.

Sistemas de caché

Sass cachea la compilación de los archivos Sass o SCSS originales para poder reutilizarlos cuando no se han producido cambios. Por defecto estos archivos se cachean en el directorio indicado por la opción $:\text{cache}_{\text{location}}$.

Si no puedes cachear estos archivos en un directorio o quieres compartirlos entre varios procesos Ruby de diferentes máquinas, puedes crear tu propio sistema de caché y utilizarlo mediante la opción de configuración $:\text{cache}_{\text{store}}$.

Consulta la documentación de la clase CacheStores::Base para conocer todos los detalles sobre cómo crear tu propio sistema de caché.

Importadores propios

Los importadores de Sass se encargan de encontrar los archivos Sass adecuados a partir de los valores proporcionados en las directivas @import. Por defecto el código siempre se importa desde algún directorio del sistema de archivos, pero también se puede cargar desde una base de datos o incluso mediante servicios web.

Cada importador se encarga de gestionar un tipo diferente de importación. Todos ellos se pueden configurar en la opción de configuración :load$_{\text{paths}}$ y se pueden utilizar junto a los importadores normales del sistema de archivos.

Cuando se resuelve el valor de una directiva @import, Sass recorre todos los importadores registrados hasta encontrar con alguno que pueda importar el valor indicado. Los importadores propios siempre deben heredar de la clase {Sass::Importers::Base}.