



Preprocesadores CSS. Nivel I

marzo, 2020



Objetivos del nivel

- Conocer los distintos tipos de pre procesadores y lenguajes de pre procesamiento de CSS
- Compilar archivos CSS usando Sass
- Aprender de combinar archivos CSS
- Conocer las etiquetas que ofrece Sass para facilitar el mantenimiento de archivos CSS

Prerrequisitos del nivel

- HTML 5 y CSS 3 Nivel I
- Lógica de Programación Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADIF1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2020. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Introducción a Los Preprocesadores

- 1.1.- ¿ Qué es un Preprocesador?.
- 1.2.- Lenguajes de Preprocesamiento Css.
- 1.3.- Prepros

Capítulo 2. Preprocesadores CSS: SASS

- 2.1.- Preprocesadores en Css.
- 2.2.- ¿ Cómo Instalar Sass?.
- 2.3.- Ventajas de Sass.

Capítulo 3. Anidamiento

- 3.1.- Alcance de un Selector - Anidamiento.
- 3.2.- Referenciando a Los Selectores Padre.
- 3.3.- Propiedades Anidadas.

Capítulo 4. Trabajando Con Variables

- 4.1.- Uso de Variable.
- 4.2.- Crear Variables.
- 4.3.- Alcance.

Capítulo 5. @import

- 5.1.- Importar Archivos.
- 5.2.- ¿ Cómo Importar Archivos?.

Capítulo 6. @extend

- 6.1.- Extender / Herencia.
- 6.2.- ¿ Cómo Funciona?.

Capítulo 7. Operadores

- 7.1.- Operadores en Sass.
- 7.2.- Operaciones en Sass.
- 7.3.- Interpolación.

Capítulo 8. Directivas de Control. Parte 1

- 8.1.- Operadores.
- 8.2.- Directivas de Control.
- 8.3.- Condicionales.

Capítulo 9. Mapas

- 9.1.- Mapas.
- 9.2.- Funciones de Mapas.

Capítulo 10. Directivas de Control. Parte 2

- 10.1.- Ciclos @for.
- 10.2.- Ciclos @each.
- 10.3.- Ciclos @while.

Capítulo 1. INTRODUCCIÓN A LOS PREPROCESADORES

1.1.- ¿Qué es un Preprocesador?

Un preprocesador es una herramienta que permite tras un proceso de compilación, generar código a partir de la sintaxis del preprocesador. Los preprocesadores, usan lenguajes que compilan a un lenguaje de la web, este lenguaje inicial, el cual será diferente según el preprocesador elegido, suele disponer de utilidades para facilitar la reutilización y modularización del código.

Hay preprocesadores para HTML, JavaScript y CSS.



Un preprocesador de CSS es una herramienta que permite escribir pseudo-código CSS que luego será convertido a CSS real.

El objetivo de estos preprocesadores es tener un código más sencillo de mantener y editar. Los preprocesadores incluyen características tales como variables, funciones, mixins, anidación o modularidad.

Entre las ventajas de usar preprocesadores CSS se encuentran:

- Uso de características que aún no existen en CSS (Ej.: variables, anidamiento, mixins, herencia).
- Asegurar la compatibilidad entre navegadores.

- El Nesting hace que el código sea más fácil de leer, ya que permite anidar selectores dentro de un selector padre.
- Los Imports permiten tener un código modular.
- Las Pre-built Functions permiten ahorrar tiempo.
- Las variables y funciones permiten reutilización de código y ahorro de tiempo.
- Reduce código repetitivo.
- Facilita el mantenimiento del código.
- El uso de un preprocesador es transparente a la página.

1.2.- Lenguajes de Pre procesamiento CSS

Para pre procesamiento de CSS existen varios lenguajes. Entre los más usados están:

- Sass es un preprocesador CSS escrito originalmente en Ruby, aunque actualmente existen otras implementaciones disponibles, como por ejemplo en JavaScript.

Página oficial: <https://sass-lang.com/>



- Less (que significa Leaner Style Sheets) es un lenguaje de hojas de estilos que puede ser compilado en hojas de estilo en cascada (CSS), es de código abierto. Su primera versión fue escrita en Ruby, sin embargo, en las versiones posteriores, se abandonó el uso de

Ruby y se lo sustituyó por JavaScript. La principal diferencia con otros preprocesadores CSS en que Less permite la compilación en tiempo real vía less.js por el navegador.

Página oficial: <http://lesscss.org/>



- Stylus es un preprocesador de CSS que facilita la creación de hojas de estilos CSS de manera más eficiente y rápida. Soporta tanto una sintaxis con sangría y estilo normal de CSS.

Algunas de las principales características de este preprocesador son las siguientes:

ACADEMIA DE SOFTWARE

- No requiere el uso de llaves, dos puntos, comas y, punto y coma.
- Provee una JavaScript API.
- Permite hacer uso de variables, funciones, mixins, condicionales.
- Cuenta con una librería llamada nib que proporciona mixins para usar.

Página oficial: <https://stylus-lang.com/>



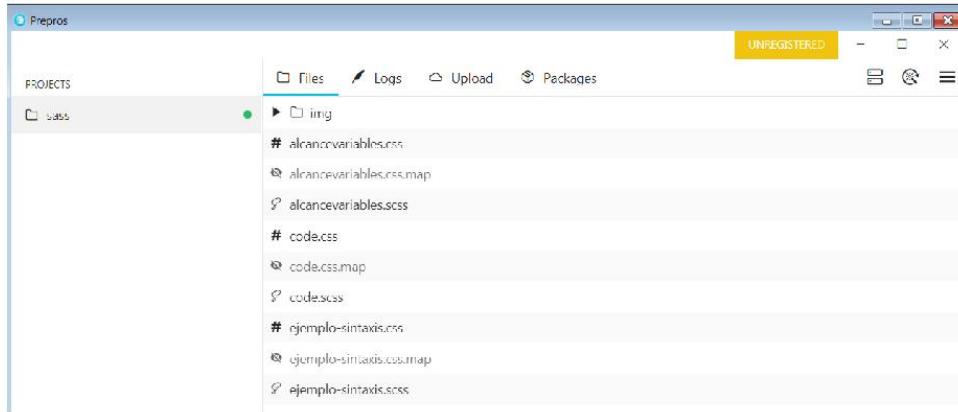
1.3.- Prepros

Prepros es una herramienta para compilar LESS, Sass, Compass, KIT, Stylus, Jade y mucho más, permite testear el resultado en un servidor localhost propio.

Esta herramienta funciona sobre Windows, Mac OS y Linux, tiene una versión gratuita y permite transformar una directiva sass/scss en código CSS convencional, transformando un archivo .scss en un archivo .css

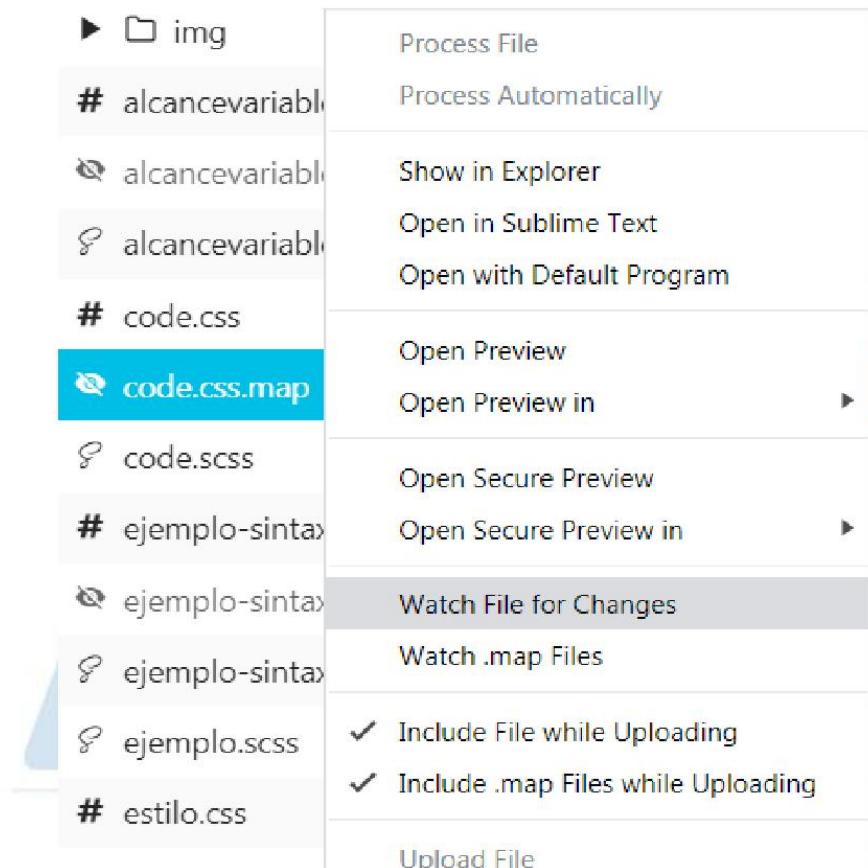
Una de las características potenciales de prepros, es que se pueden sincronizar archivos locales con los archivos del servidor, subiendo de forma automática los cambios al servidor al momento de guardarlos

El primer paso para usar prepros luego de instalarlo en el computador y abrir la herramienta, será arrastrar y soltar la carpeta del proyecto en la ventana de Prepros para agregarlo. En la siguiente imagen se muestra cómo se visualiza la lista de archivos que contiene la carpeta del proyecto creado.



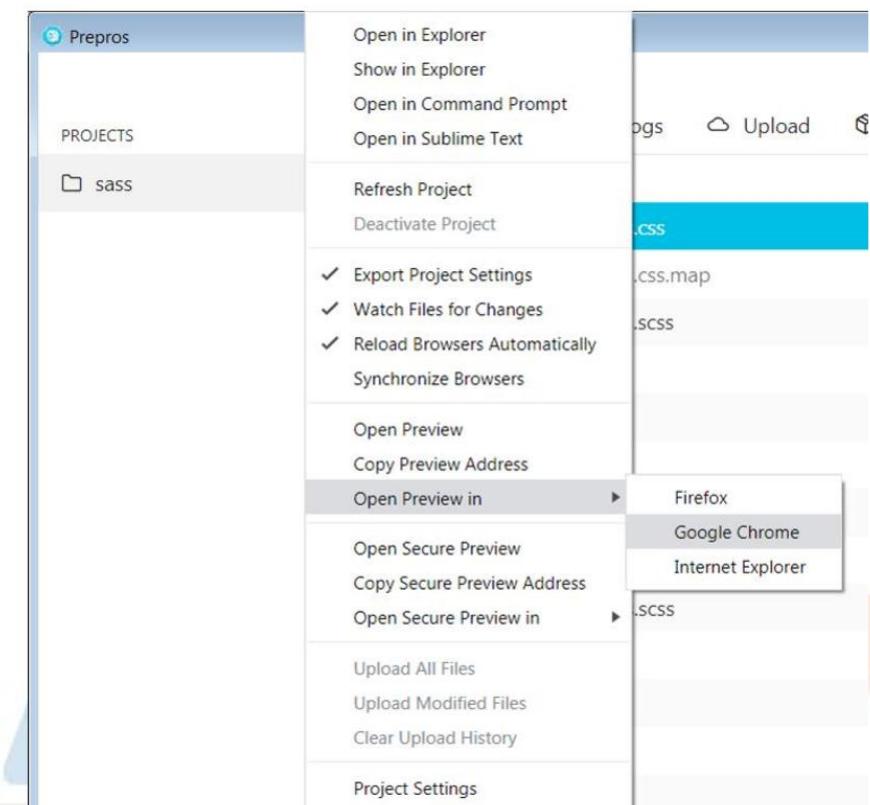
Si en la carpeta del proyecto hay archivos que no necesitan ser compilados y recargados automáticamente, éstos se deben "ignorar" para evitar problemas de rendimiento. Esto es necesario debido a que prepros "vigila" todos los archivos en la carpeta del proyecto de manera predeterminada. Para ignorar los archivos se debe hacer clic derecho sobre ellos en el árbol de archivos tal como se muestra en la siguiente imagen:





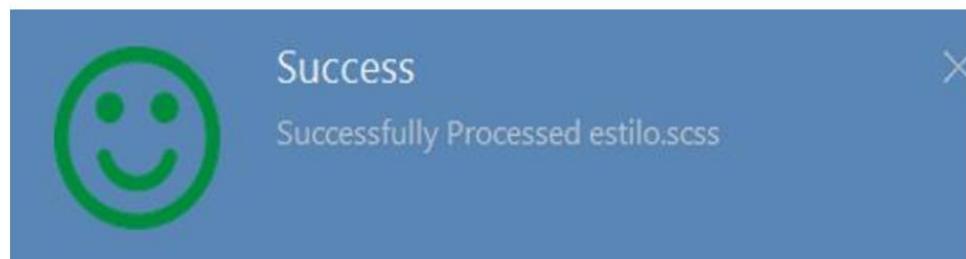
Si el sitio web consta solo de archivos HTML estáticos, se puede abrir la vista previa de inmediato presionando CTRL+Len Windows y Linux y CMD+Len macOS, lo que ocurrirá es que se abrirá el documento en el navegador. Si el sitio web tiene contenido dinámico y requiere un servidor como XAMPP, Mamp, Wamp, WordPress, etc., se puede configurar Prepros para solicitudes proxy a un servidor externo.

En la siguiente imagen se observa como abrir la vista previa de un archivo en el navegador.



ACADEMIA DE SOFTWARE

Finalmente, Prepros mirará los archivos en modo watch, los compilará y generará el archivo correspondiente. En la siguiente imagen se muestra el mensaje exitoso cuando prepros compila un archivo scss generando el CSS correspondiente. En caso de archivos HTML, volverá a cargar el navegador de manera automática.



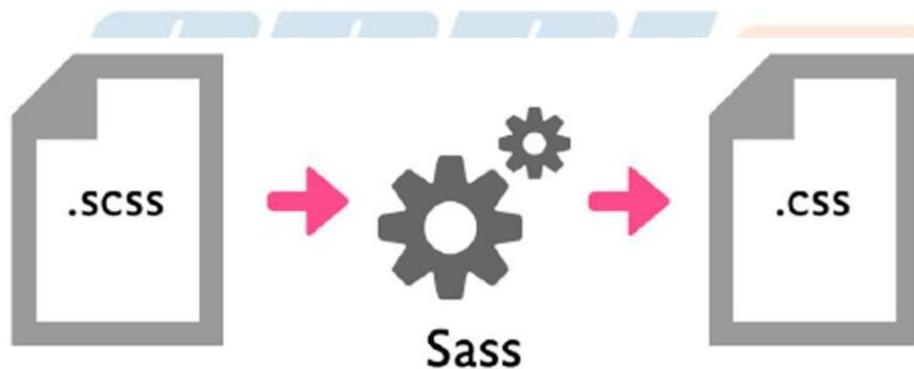
Capítulo 2. PREPROCESADORES CSS: SASS

2.1.- Preprocesadores en CSS

SASS es un preprocesador de código CSS, originalmente creado en Ruby, aunque actualmente existen otras implementaciones disponibles, como por ejemplo en JavaScript e incluyendo una en PHP para Drupal.

Sass es un metalenguaje de hojas de estilos en cascada (CSS) que se compila en CSS y permite usar variables, reglas anidadas, mixins, funciones y más, con una sintaxis totalmente compatible con CSS.

Las siglas SASS significan syntactically awesome stylesheets.



SASS utiliza sus propios archivos. Tiene dos opciones:

- Ficheros .sass: En estos ficheros no es necesario escribir llaves ni punto y coma al final.
 - Ficheros .scss: En estos ficheros si es necesario escribir llaves, puntos y coma al final.
- > El código SASS, aunque muy parecido al de CSS nativo, no puede ser interpretado por los navegadores.
- > El código sirve para el desarrollo, pero luego debe ser compilado por el preprocesadores, transformándolo a CSS estándar.



2.2.- ¿ Cómo Instalar Sass?

Para usar SASS es necesario un compilador que transforme los ficheros .scss o .sass en archivos .css estándares. La forma más fácil es instalar SASS mediante su paquete global de NPM (Node Package Manager).

Npm es el sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript.

El comando para instalar SASS a través de npm es el siguiente:

npm install -g sass

```
C:\Users\asesor>npm install -g sass
C:\Users\asesor\AppData\Roaming\npm\sass -> C:\Users\asesor\AppData\Roaming\npm\node_modules\sass\sass.js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.1.2 (node_modules\sass\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
)
+ sass@1.26.1
added 15 packages from 18 contributors in 11.827s

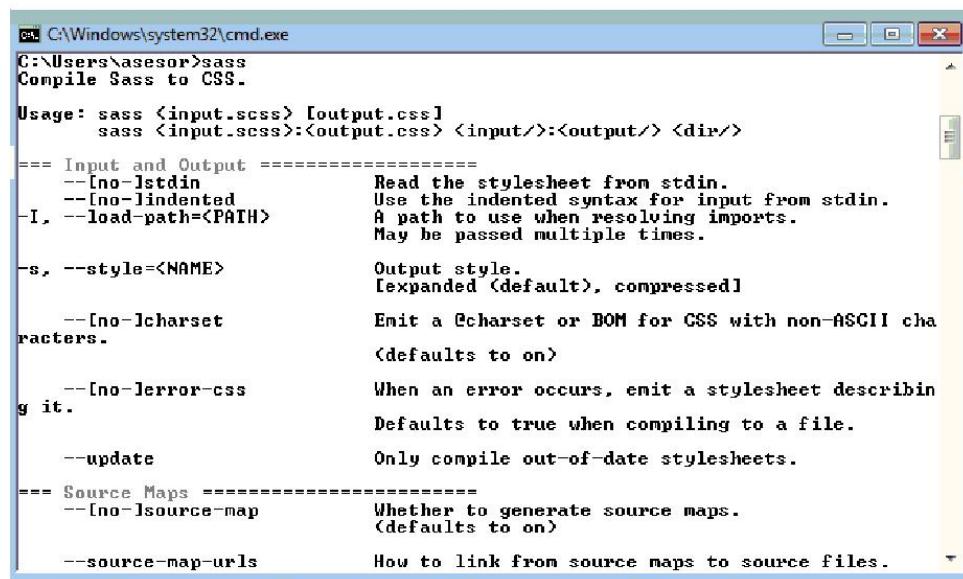
?
?           New major version of npm available! 5.6.0 -> 6.14.1
Changelog: https://github.com/npm/cli/releases/tag/v6.14.1
Run npm install -g npm to update!
?

C:\Users\asesor>
```

Luego de instalado, usando el comando "sass" se puede ver en la consola, la información de compilación tal como aparece en la siguiente imagen:

Simplemente tomará lo que haya en el fichero input y lo compilará en el fichero css output. Hay un parámetro para sass que hace que automáticamente se compilen los archivos sass en archivos css cuando detecta un cambio:

```
sass --watch input.scss output.css
```



```
C:\Windows\system32\cmd.exe
C:\Users\nasesor>sass
Compile Sass to CSS.

Usage: sass <input.scss> [output.css]
       sass <input.scss>:<output.css> <input/>:<output/> <dir/>

==== Input and Output =====
  --[no-]stdin           Read the stylesheet from stdin.
  --[no-]indented         Use the indented syntax for input from stdin.
  -I, --load-path=<PATH>   A path to use when resolving imports.
                           May be passed multiple times.

  -s, --style=<NAME>        Output style.
                           [expanded (default), compressed]

  --[no-]charset          Emit a charset or BOM for CSS with non-ASCII characters.
                           [defaults to on]

  --[no-]error-css         When an error occurs, emit a stylesheet describing it.
                           Defaults to true when compiling to a file.

  --update                Only compile out-of-date stylesheets.

==== Source Maps =====
  --[no-]source-map        Whether to generate source maps.
                           [defaults to on]

  --source-map-urls        How to link from source maps to source files.
```

Otra forma de instalar SASS es a través del IDE Visual Studio Code. Primero se debe tener instalado Visual Studio Code, luego es necesario instalar dos extensiones: Live Sass Compiler y Live Server.

Live Sass Compiler

Ritwick Dey | ⚡ 492.011 | ★★★★★ | Repositorio | Licencia
Compile Sass or Scss to CSS at realtime with live browser reload.

[Instalando](#)

[Detalles](#) [Contribuciones](#) [Registro de cambios](#) [Dependencias](#)

Live Sass Compiler

[If you like the extension, please leave a review, it puts a smile on my face.]

[If you found any bug or if you have any suggestion, feel free to report or suggest me.]

VSCode Marketplace v3.0.0 downloads 620k rating 4.6/5 (112) license MIT

Live Server

Ritwick Dey | ⚡ 4.314.464 | ★★★★★ | Repositorio | Licencia
Launch a development local Server with live reload feature for static & dynamic pages

[Deshabilitar](#) [Desinstalación](#)

[Detalles](#) [Contribuciones](#) [Registro de cambios](#) [Dependencias](#)

I'm sorry but I'm super busy now. If you want to be a maintainer of the project, please feel free to contact me! You've to be

Live Server

Live Server loves # your multi-root workspace

Live Server for server side pages like PHP. [Check Here](#)

[For 'command not found error' #78]

vscode marketplace v5.6.1 downloads 7.5M rating 4.5/5 (226)

license MIT

Luego, se podrá compilar código SCSS en modo watch, la consola mostrará los archivos generados inmediatamente tras la compilación.

The screenshot shows a code editor with a dark theme and a terminal window below it.

Code Editor Content:

```
alcancevariables.scss ✘
1 $global-variable-color: #f2f2f2;
2
3 .content {
4   $global-variable-color: #414141;
5   background: $global-variable-color;
6 }
7
8 .sidebar {
9   background: $global-variable-color;
10 }
```

Terminal Content:

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
Compiling Sass/Scss Files:
-----
Watching...
-----
Not Watching...
-----
Compiling Sass/Scss Files:
-----
Watching...
-----
Change Detected...
alcancevariables.scss
-----
Generated :
j:\sass\alcancevariables.css
j:\sass\alcancevariables.css.map
-----
```

2.3.- Ventajas de Sass

- Sass ayuda a mantener bien organizadas las hojas de estilo grandes y facilita compartir diseños dentro y entre proyectos.
- Se puede extender el código CSS para hacerlo más reutilizable y dinámico, agilizando el desarrollo y escribiendo menos código CSS.



Capítulo 3. ANIDAMIENTO

3.1.- Alcance de un Selector - Anidamiento

El anidamiento consiste en escribir reglas de estilo de uno o más elementos CSS sin repetir los mismos selectores una y otra vez.

Los selectores CSS se anidan de manera que siga la misma jerarquía visual que HTML, permitiendo escribir reglas de estilos dentro de otra, combinando automáticamente el selector de la regla externa con la de la regla interna.

El siguiente es un ejemplo de anidamiento con selectores de elementos HTML: nótese que los ul, li y selectores están anidados dentro del nav selector.



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

Las reglas anidadas son inteligentes sobre el manejo de listas de selectores (es decir, selectores separados por comas). Cada selector complejo (los que están entre las comas) se anidan por separado, y luego se combinan nuevamente en una lista de selector.

El siguiente es un ejemplo de anidamiento con listas de selectores de clases:

```

.alert, .warning {
    ul, p {
        margin-right: 0;
        margin-left: 0;
        padding-bottom: 0;
    }
}

.alert ul, .alert p, .warning ul, .warning p {
    margin-right: 0;
    margin-left: 0;
    padding-bottom: 0;
}

```

Consideraciones sobre el anidamiento:

- Las reglas demasiado anidadas darán como resultado un CSS SOBRECALIFICADO que podría resultar difícil de mantener y generalmente se considera una mala práctica.
- Pueden dificultar la visualización de la cantidad de CSS que realmente está generando.
- Cuanto más profundo anide, más ancho de banda se necesita para servir su CSS y más trabajo requiere el navegador para representarlo.
- No se deben definir estilos CSS a selectores que aniden más de tres elementos.

3.2.- Referenciando a Los Selectores Padre

El selector principal `&`, es un selector especial que se usa en selectores anidados para referirse al selector externo. Permite reutilizar el selector externo de formas más complejas, como agregar una pseudoclase o agregar un selector antes del padre.

Cuando se usa un selector principal en un selector interno, se reemplaza con el selector externo correspondiente. Esto sucede en lugar del comportamiento normal de anidamiento. Al definir código anidado, podamos continuar en el uso del selector sobre el que se está trabajando, a fin de no repetirlo de nuevo.

```

.enlarge {
    font-size: 14px;
    transition: {
        property: font-size;
        duration: 4s;
        delay: 2s;
    }
    &:hover {
        font-size: 36px;
    }
}

.enlarge {
    font-size: 14px;
    transition-property: font-size;
    transition-duration: 4s;
    transition-delay: 2s;
}
.enlarge:hover {
    font-size: 36px;
}

```

Otro uso de este operador para referirse al selector padre es para agregar sufijos adicionales al selector externo. Siempre que el selector externo termine con un nombre alfanumérico (como los selectores de clase, ID y elemento), puede usar el selector principal para agregar texto adicional.

Ejemplo: para poder definir todos los estilos de la clase form de una vez y usando anidación, se puede escribir algo como esto.

```
.form {  
  margin: 10px;  
  padding: 15px;  
  &--black {  
    color: black;  
    background-color: black;  
  }  
  &__submit {  
    color: red;  
    &--desactivado {  
      color: #999;  
    }  
  }  
}
```



```
.form {  
  margin: 10px;  
  padding: 15px;  
}  
  
.form--black {  
  color: black;  
  background-color: black;  
}  
  
.form__submit {  
  color: red;  
}  
  
.form__submit--desactivado {  
  color: #999;  
}
```

```
.accordion {
  max-width: 600px;
  margin: 4rem auto;
  width: 90%;
  font-family: "Raleway", sans-serif;
  background: #f4f4f4;

  &__copy {
    display: none;
    padding: 1rem 1.5rem 2rem 1.5rem;
    color: gray;
    line-height: 1.6;
    font-size: 14px;
    font-weight: 500;

    &--open {
      display: block;
    }
  }
}
```



```
.accordion {
  max-width: 600px;
  margin: 4rem auto;
  width: 90%;
  font-family: "Raleway", sans-serif;
  background: #f4f4f4;
}

.accordion__copy {
  display: none;
  padding: 1rem 1.5rem 2rem 1.5rem;
  color: gray;
  line-height: 1.6;
  font-size: 14px;
  font-weight: 500;
}

.accordion__copy--open {
  display: block;
}
```

3.3.- 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 aplicar anidamiento sobre declaraciones de propiedades CSS, de forma que se pueden escribir como propiedades anidadas: los nombres de las propiedades externas se agregan a los internos, separados por un guion.

```
.contenedor {
  width: 50%;
  border-radius: 10px;
  padding: 50px;
  text-align: center;
  margin: 50px auto;
  {
    bottom: 10px;
    top: 2px;
  }
}
```



```
.contenedor {
  width: 50%;
  border-radius: 10px;
  padding: 50px;
  text-align: center;
  margin: 50px auto;
  margin-bottom: 10px;
  margin-top: 2px;
}
```

En los siguientes ejemplos se observa como usar el anidamiento para las propiedades transition y font.

```
.enlarge {  
    font-size: 14px;  
    transition: {  
        property: font-size;  
        duration: 4s;  
        delay: 2s;  
    }  
}
```

```
.enlarge {  
    font-size: 14px;  
    -webkit-transition-property: font-size;  
    transition-property: font-size;  
    -webkit-transition-duration: 4s;  
    transition-duration: 4s;  
    -webkit-transition-delay: 2s;  
    transition-delay: 2s;  
}
```

```
h2 {  
    color: #666;  
    font: {  
        family: verdana;  
        weight: bold;  
        size: 0.9em;  
    }  
}
```

```
h2 {  
    color: #666;  
    font-family: verdana;  
    font-weight: bold;  
    font-size: 0.9em;  
}  
/*# sourceMappingURL=estilo.css.map */
```

ACADEMIA DE SOFTWARE

Capítulo 4. TRABAJANDO CON VARIABLES

4.1.- Uso de Variable

Sass permite la definición de variables, que al igual que en todos los lenguajes de programación, permiten almacenar información que se desea reutilizar en la hoja de estilo, por ejemplo: colores, pilas de fuentes o cualquier valor CSS que se deseé reutilizar.

En el caso de CSS, permite por un lado evitar la repetición de valores (por ejemplo, un color, una familia tipográfica o una medida), y por otro centralizar dicho dato de forma tal que para su modificación haya que cambiar el valor una sola vez.

Para identificar las variables, Sass usa el símbolo \$ y la asignación con dos puntos (:).

```
1 $fuente-principal: Helvetica, sans-serif
2 $color-primario: #333
3
4 body {
5   font: 100% $fuente-principal
6   color: $color-primario
```

```
1 body {
2   font: 100% Helvetica, sans-serif;
3   color: #333;
4 }
5
```

Los tipos de datos para variables son:

- Números (ejemplo: 1.2, 13, 10px)
- Strings (ejemplo "foo", "bar" baz, o encerrados entre comillas simples)
- Colores (ejemplo blue, #04a3f9, rgba(255, 0, 0, 0.5))
- Booleanos (true, false)
- Listas de valores, separados por espacios en blanco o comas (ejemplo 1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
- Pares formados por una clave y un valor separados por : (ejemplo (key1: value1, key2: value2))

Las variables pueden ser resultados o argumentos de varias funciones disponibles. Durante el proceso de traducción, los valores de las variables son insertados en el documento CSS de salida.

Las variables Sass, como todos los identificadores Sass, tratan los guiones y guiones bajos como idénticos. Esto significa que \$font-size y \$font_size ambos se refieren a la misma variable. Este es un remanente histórico de los primeros días de Sass, cuando solo permitía guiones bajos en los nombres de los identificadores. Una vez que Sass agregó soporte para guiones para que coincida con la sintaxis de CSS, los dos se hicieron equivalentes para facilitar la migración.

4.2.- Crear Variables

La declaración de una variable se parece mucho a una declaración de propiedad.

Se escribe <variable>: <expression>.

A diferencia de una propiedad, que solo se puede declarar en una regla de estilo o en regla, las variables se pueden declarar en cualquier lugar que desee. Para usar una variable, simplemente se incluye en el valor de una propiedad.

The diagram illustrates the compilation process of Sass code into CSS. On the left, a dark grey box contains the original Sass code:

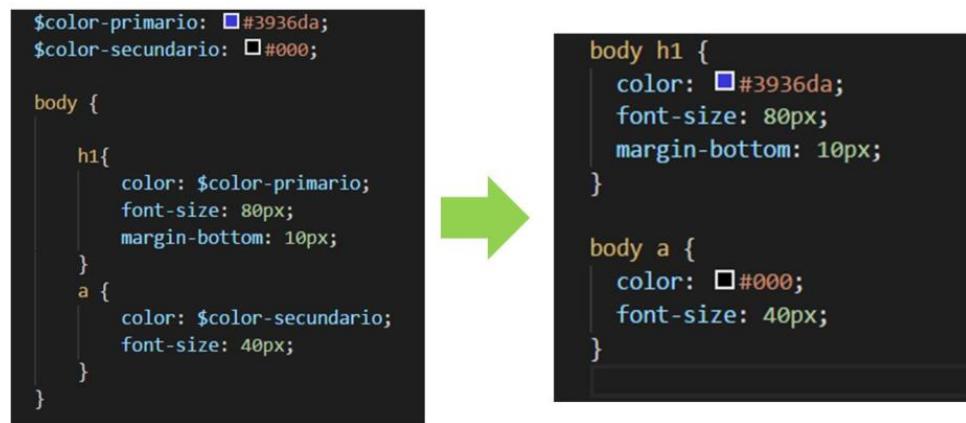
```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

An orange arrow points from this box to a light grey box on the right, which contains the resulting CSS code:

```
1 @charset "UTF-8";
2
3 body {
4   font: 100% Helvetica, sans-serif;
5   color: #333;
6 }
7
```

En la siguiente imagen, se observa otro ejemplo del uso de variables:



4.3.- Alcance

Las variables declaradas en el nivel superior de una hoja de estilo son globales. Esto significa que se puede acceder a ellos desde cualquier lugar de su módulo después de haber sido declarados. Pero eso no es cierto para todas las variables. Los declarados en bloques (llaves en SCSS o código sangrado en Sass) suelen ser locales, y solo se puede acceder a estas dentro del bloque en el que se declararon.

```
$global-variable-color: #f2f2f2;
.content {
    $local-variable-color: #414141;
    background: $global-variable-color;
    background-image: $local-variable-color;
}

.sidebar {
    background-image: $local-variable-color;
}
```

Las variables locales pueden incluso declararse con el mismo nombre que una variable global. Si esto sucede, en realidad hay dos variables diferentes con el mismo nombre: una local y otra global. Esto ayuda a garantizar que un autor que escribe una variable local no cambie accidentalmente el valor de una variable global que ni siquiera conocen.

The diagram illustrates the compilation process from SCSS to CSS. On the left, SCSS code is shown with a green arrow pointing to the right, indicating the transformation. The SCSS code includes a global variable \$global-variable-color and a local variable \$global-variable-color used within the .content and .sidebar selectors. The resulting CSS output on the right shows the global variable being resolved to its value (#f2f2f2) and applied to the .content selector, while the local variable is resolved to its value (#414141) and applied to the .sidebar selector.

```
$global-variable-color: #f2f2f2;  
  
.content {  
  $global-variable-color: #414141;  
  background: $global-variable-color;  
}  
  
.sidebar {  
  background: $global-variable-color;  
}  
  
→  
  
.content {  
  background: #414141;  
}  
  
.sidebar {  
  background: #f2f2f2;  
}
```

Es importante tener en cuenta que, aunque CSS ya dispone de variables, e incluso puede realizar cálculos sencillos, al hacer uso de ellos se necesita que el navegador empleado tenga una versión en la que ya se soporte dichas funcionalidades. Este problema no ocurre en SCSS ya que la compilación se realiza antes de desplegar nuestra web, por lo que el CSS generado ya no incluirá las variables o funciones, y por ello será transparente al navegador.

Capítulo 5. @IMPORT

5.1.- Importar Archivos

CSS tiene una manera de importar ficheros CSS unos dentro de otros, por lo que Sass también hace lo mismo. Sin embargo, con el import de Sass no se generan nuevas peticiones HTTP. Cada vez que se importe un fichero, Sass va a tomar el contenido del fichero importado y lo va a combinar con el otro fichero, las importaciones Sass se manejan completamente durante la compilación.

Sass amplía esta regla de CSS con la capacidad de importar hojas de estilo Sass y CSS, proporcionando acceso a mixins, funciones y variables, y combinando CSS de varias hojas de estilo.

Las importaciones Sass tienen la misma sintaxis que las importaciones CSS, excepto que permiten que las importaciones múltiples estén separadas por comas en lugar de requerir que cada una tenga su propia regla @import.

Eso permite además de tener una mejor organización modularizando los estilos, reusar un mismo módulo en diferentes ficheros, como el caso del popular framework frontend Bootstrap. Este producto está escrito en SASS y dispone de un fichero SCSS de variables, con el que configura diferentes aspectos claves de la visualización de cada uno de los elementos.

Gracias a ello, se puede cargar el fichero principal de Bootstrap en el SCSS (luego de haber descargado los fuentes SCSS de bootstrap) y hacer una sobrecarga de alguna de las propiedades que se deseen antes de cargar el fichero SCSS principal. Así se dispondrá de un Bootstrap completamente personalizado

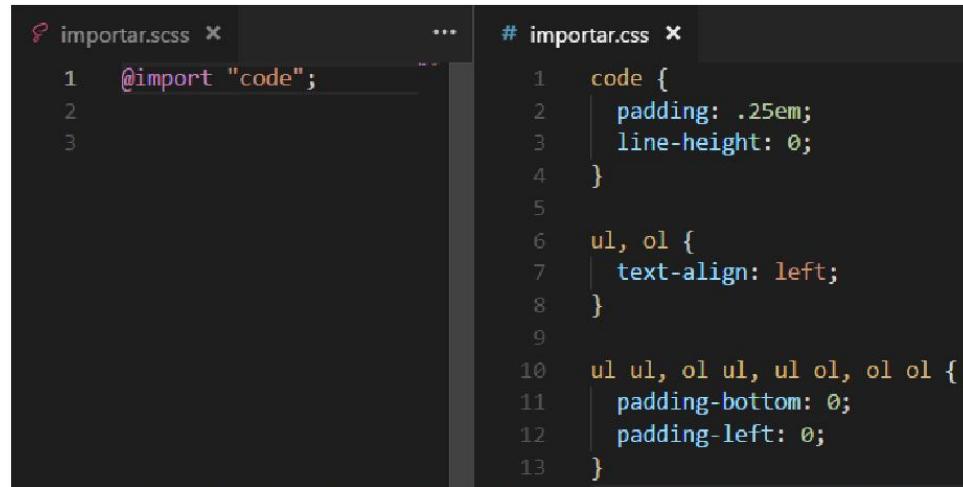
Descargas > bootstrap-4.4.1 > bootstrap-4.4.1 > scss >			
biblioteca	Compartir con	Grabar	Nueva carpeta
Nombre	Fecha de modifica...	Tipo	Tamaño
mixins	28/11/2019 04:59 a...	Carpeta de archivos	
utilities	28/11/2019 04:59 a...	Carpeta de archivos	
vendor	28/11/2019 04:59 a...	Carpeta de archivos	
_alert.scss	28/11/2019 04:59 a...	Archivo SCSS	2 KB
_badge.scss	28/11/2019 04:59 a...	Archivo SCSS	2 KB
_breadcrumb.scss	28/11/2019 04:59 a...	Archivo SCSS	2 KB
_button-group.scss	28/11/2019 04:59 a...	Archivo SCSS	4 KB
_buttons.scss	28/11/2019 04:59 a...	Archivo SCSS	3 KB
_card.scss	28/11/2019 04:59 a...	Archivo SCSS	6 KB
_carousel.scss	28/11/2019 04:59 a...	Archivo SCSS	5 KB
_close.scss	28/11/2019 04:59 a...	Archivo SCSS	1 KB
_code.scss	28/11/2019 04:59 a...	Archivo SCSS	1 KB
_custom-forms.scss	28/11/2019 04:59 a...	Archivo SCSS	16 KB
_dropdown.scss	28/11/2019 04:59 a...	Archivo SCSS	5 KB
_forms.scss	28/11/2019 04:59 a...	Archivo SCSS	9 KB
_functions.scss	28/11/2019 04:59 a...	Archivo SCSS	4 KB
_grid.scss	28/11/2019 04:59 a...	Archivo SCSS	2 KB
_images.scss	28/11/2019 04:59 a...	Archivo SCSS	2 KB
_input-group.scss	28/11/2019 04:59 a...	Archivo SCSS	6 KB
_jumbotron.scss	28/11/2019 04:59 a...	Archivo SCSS	1 KB

5.2.- ¿ Cómo Importar Archivos?

Cuando Sass importa un archivo, ese archivo se evalúa como si su contenido apareciera directamente en lugar del @import. Todos los mixins, funciones y variables del archivo importado están disponibles, y todo su CSS se incluye en el punto exacto donde @import se escribió.

Importante: Si la misma hoja de estilo se importa más de una vez, se evaluará nuevamente cada vez. Si solo define funciones y mixins, esto generalmente no es un gran problema, pero si contiene reglas de estilo, se compilarán en CSS más de una vez.

Además de importar archivos .sass y .scss, Sass puede importar archivos .css antiguos simples. La única regla es que la importación no debe incluir explícitamente la extensión .css, ya que se usa para indicar un CSS simple @import.



```
importar.scss
1  @import "code";
2
3

# importar.css
1  code {
2    padding: .25em;
3    line-height: 0;
4  }
5
6  ul, ol {
7    text-align: left;
8  }
9
10 ul ul, ol ul, ul ol, ol ol {
11   padding-bottom: 0;
12   padding-left: 0;
13 }
```

ACADEMIA DE SOFTWARE

Capítulo 6. @EXTEND

6.1.- Extender / Herencia

Esta es una de las características más útiles de Sass. Usando @extend se permite compartir un conjunto de propiedades CSS de un selector a otro.

Cuando una clase extiende a otra, Sass aplica estilos a todos los elementos que coinciden con el extensor como si también coincidieran con la clase que se está extendiendo.

Cuando un selector de clase extiende a otro, funciona exactamente como si hubiera agregado la clase extendida a cada elemento en su HTML que ya tenía la clase extendida. Simplemente puede escribir class="error--serious", y Sass se asegurará de que tenga el mismo estilo que tenía class="error".

Por supuesto, los selectores no solo se usan solos en las reglas de estilo. Sass sabe extenderse a todas partes donde se usa el selector. Esto garantiza que sus elementos tengan el mismo estilo que si coincidieran con el selector extendido.

The diagram illustrates the transformation of Sass code using the `@extend` directive. On the left, the original Sass code is shown:

```
.error {  
  border: 1px solid #f00;  
  background-color: #fff;  
  
  &--serious {  
    @extend .error;  
    border-width: 3px;  
  }  
}
```

A large green arrow points from the original code to the resulting CSS output on the right:

```
.error, .error--serious {  
  border: 1px solid #f00;  
  background-color: #fff;  
}  
  
.error--serious {  
  border-width: 3px;  
}
```

6.2.- ¿ Cómo Funciona?

La regla `@extend` actualiza las reglas de estilo que contienen el selector extendido para que también contengan el selector extendido.

Al extender los selectores, Sass realiza la unificación inteligente:

- Nunca genera selectores como `#main #footer` ese no pueden coincidir con ningún elemento.
- Asegura que los selectores complejos estén intercalados para que funcionen sin importar el orden en que los elementos HTML estén anidados.
- Recorta los selectores redundantes tanto como sea posible, al tiempo que garantiza que la especificidad sea mayor o igual que la del extensor.
- Sabe cuándo un selector coincide con todo lo que hace otro y puede combinarlos.
- Maneja inteligentemente combinadores, selectores universales y pseudo-clases que contienen selectores.

```
.content nav.sidebar {  
  @extend .info;  
}  
// Esto no se extenderá, porque `p` es incompatible con `nav`.  
.info {  
  background-color: ■#dee9fc;  
}  
// No hay forma de saber si `<div class =" guide ">` estará adentro o  
// fuera de `<div class =" content ">`, por lo que Sass genera ambos para estar a salvo.  
.guide .info {  
  border: 1px solid rgba(□#000, 0.8);  
  border-radius: 2px;  
}  
// Sass sabe que cada elemento que coincide con "main.content" también coincide con ".content"  
// y evita generar selectores intercalados innecesarios.  
main.content .info {  
  font-size: 0.8em;  
}
```



```
p.info {  
  background-color: ■#dee9fc;  
}  
  
.guide .info, .guide .content nav.sidebar, .content .guide nav.sidebar {  
  border: 1px solid □rgba(0, 0, 0, 0.8);  
  border-radius: 2px;  
}  
  
main.content .info, main.content nav.sidebar {  
  font-size: 0.8em;  
}
```

Capítulo 7. OPERADORES

7.1.- Operadores en Sass

Un operador es un símbolo que denota un conjunto de operaciones que han de realizarse. Los operadores disponibles en Sass son:

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos

Los operadores aritméticos disponibles Sass son los siguientes:

- + Suma
- Resta
- * Multiplicación
- / División
- % Módulo

7.2.- Operaciones en Sass

Para realizar la suma, se debe tener en cuenta que los valores deben tener las mismas unidades, o al menos que uno tenga indicada la unidad (en este caso, el valor con la unidad debe estar de primero, a la izquierda).

```
p {  
    font-size: 16px + 4;  
}  
h3 {  
    width: 2px * 5 + 12px;  
}  
h2 {  
    width: 8px + (12px / 2) * 3;  
}
```



```
p {  
    font-size: 20px;  
}  
h3 {  
    width: 22px;  
}  
h2 {  
    width: 26px;  
}
```

Si se utilizan unidades diferentes, al compilar arrojará un error, tal como se muestra en la imagen:

The screenshot shows a code editor interface with a dark theme. At the top, there is a code snippet:

```
25
26     div {
27         height: 45% + 6px;
28         font-size: 16px+1em;
29     }
30
31
32
```

Below the code, there are four tabs: PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, and TERMINAL. The CONSOLA DE DEPURACIÓN tab is selected, displaying the following error message:

```
Compilation Error
Error: Incompatible units: 'px' and '%'.
      |       on line 27 of sass/f:\sass\operadores.scss
>>      height: 45% + 6px;
      |       ^
```

El operador +, se usa también para concatenar. No obstante, se debe considerar lo siguiente:

- Si una cadena con comillas viene antes de una sin comillas, se citará la cadena resultante.
- Por otro lado, si una cadena sin comillas viene antes de una con comillas, el resultado es una cadena sin comillas.

Por ejemplo:

```
div:before{  
    content: "Soy un string con " + comillas;  
    font-family: Arial + ", sans serif"  
}  
  
div:before {  
    content: "Soy un string con comillas";  
    font-family: Arial, sans serif;  
}
```



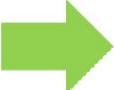
Al igual que la suma, cuando se realiza una operación de resta se debe tener en cuenta que los valores deben tener las mismas unidades, o al menos que uno tenga indicada la unidad (en este caso, el valor con la unidad debe estar el primero, a la izquierda). Además, se debe dejar un espacio entre los operadores y los operandos.

```
div {  
    padding: 2em -1 ;  
    height: 20% - 6;  
}  
  
div {  
    padding: 1em;  
    height: 14%;  
}
```



Para la multiplicación se usa el operador *, este operador se utiliza de la misma forma que la suma y la resta, solo que uno de los valores no debe tener unidad. Siguiendo esto, el siguiente código sería válido:

```
div {  
    font-size: 10px * 3;  
}  
  
div {  
    font-size: 30px;  
}
```



Para la división se usa el operador /.

El uso de este carácter es bastante particular, debido a que CSS permite el uso del carácter / para separar números y 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 tal razón, para utilizar este operador para dividir, se deben encerrar los valores entre paréntesis.

Las siguientes son tres situaciones en las que el carácter / se interpreta como una división matemática:

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



```

p {
    font: 10px/8px;
    $width: 1000px;

    width: $width/2;
    width: round(1.5)/2;
    height: (500px/2);
    margin-left: 5px + 8px/2px;
}

```

```

p {
    font: 10px/8px;
    width: 500px;
    width: 1;
    height: 250px;
    margin-left: 9px;
}

```

7.3.- Interpolación

ACADEMIA DE SOFTWARE

Dentro de una cadena de texto se puede 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";
}

```

```

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

```

Capítulo 8. DIRECTIVAS DE CONTROL. PARTE 1

8.1.- Operadores

Además de los operadores para realizar operaciones matemáticas, Sass permite el uso de operadores de comparación.

Operador	Condiciones	Descripción
<code>==</code>	<code>x == y</code>	Devuelve true si x e y son iguales
<code>!=</code>	<code>x != y</code>	Devuelve true si x e y no son iguales
<code>></code>	<code>x > y</code>	Devuelve true si x es mayor que y
<code><</code>	<code>x < y</code>	Devuelve true si x es menor que y
<code>>=</code>	<code>x >= y</code>	Devuelve true si x es mayor o igual a y
<code><=</code>	<code>x <= y</code>	Devuelve true si x es menor o igual a y

Los operadores lógicos disponibles en SassScript son:

Operador	Condiciones	Descripción
and	x and y	Devuelve true si x e y son true
or	x or y	Devuelve true si x o y son true
not	not x	Devuelve true si x no es true

8.2.- Directivas de Control

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.

Entre las directivas de control que se pueden usar en SassScript, se encuentran:

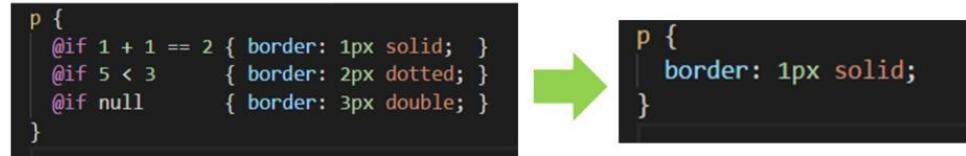
- @if y @else
- @for
- @each
- @while

8.3.- Condicionales

En SassScript se puede usar la función if() o la directiva @if para condicionar los estilos.

La función if() permite tomar decisiones para que una hoja de estilos incluya unos u otros estilos bajo determinadas condiciones. La función if() solamente evalúa el argumento que corresponde al valor que va a devolver, por lo que en el otro valor se puede 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 evalúa una expresión SassScript y solamente incluye los estilos definidos en su interior si la expresión devuelve un valor distinto a false o null.

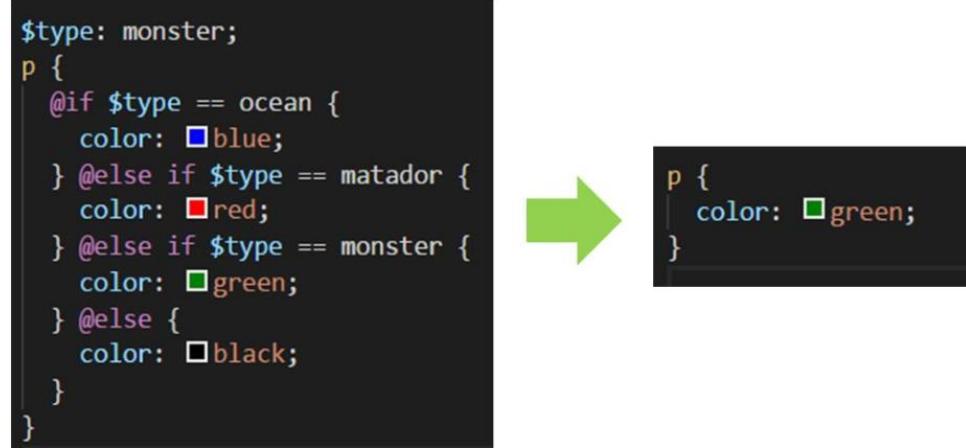


```
p {  
  @if 1 + 1 == 2 { border: 1px solid; }  
  @if 5 < 3 { border: 2px dotted; }  
  @if null { border: 3px double; }  
}
```

```
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.

Por ejemplo:



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

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

Capítulo 9. MAPAS

9.1.- Mapas

Los mapas de Sass, permiten agrupar llaves y valor dentro de un objeto. Una vez el mapa se haya creado se puede hacer un loop por él para obtener los valores o acceder directamente al valor que se desea obtener.

Para declarar un mapa, se comienza con el nombre de una variable y luego se ingresan algunas claves con valores separadas por coma, todo va dentro de paréntesis.

```
$mapa: (  
    llave1 valor1;  
    llave2: valor2;  
    llave3: valor3;  
)
```

Ejemplo:



```
$mapa: (  
    llave1: valor1,  
    llave2: valor2,  
    llave3: valor3  
)
```

9.2.- Funciones de Mapas

Existen varias funciones que nos permiten manipular los mapas fácilmente.

- map-get(\$map, \$key): devuelve el valor de una llave.
- map-merge(\$map1, \$map2): une dos variables.
- map-merge(\$map, \$key...): elimina llaves de una variable.

- map-keys(\$map): devuelve una lista de todas las llaves en una variable.
- map-values(\$map): devuelve una lista de todos los valores en una variable.
- map-has-key(\$map, \$key): devuelve si una variable Map cuenta con una llave del mismo nombre del segundo argumento (\$key).
- Keywords(\$args): devuelve las palabras claves pasadas a una función que toma o recibe argumentos de variables.

Para obtener los valores de un mapa y utilizarlos, se debe usar la función map-get(\$map, \$key), donde el primer parámetro es el nombre del mapa y el segundo es la llave que está dentro de ese mapa.

Ejemplo:

```
1 $colores: (
2   azul: #rgb(3, 95, 216),
3   verde: #rgb(17, 228, 10),
4   naranja: #rgb(248, 99, 0),
5   rojo: #rgb(189, 30, 30)
6 );
7
8 .text-azul{
9   color: map-get($colores, azul);
10 }
11
12 .text-verde{
13   color: map-get($colores , verde);
14 }
```



```
.text-azul {
  color: #035fd8;
}

.text-verde {
  color: #11e40a;
}
```

Si se quiere acceder a un mapa que no existe, al momento de compilar arrojará un error, igualmente si se quiere acceder a una llave que no existe, CSS no lo va a tomar como válido y no lo reconocerá.

```
16 .text-naranja{  
17   color: map-get($color, naranja);  
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Error: Undefined variable.

```
17 |   color: map-get($color, naranja);  
     ^^^^^^
```

style.scss 17:20 root stylesheet



Capítulo 10. DIRECTIVAS DE CONTROL. PARTE 2

10.1.- Ciclos @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>
```

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

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.

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 decremente en vez de incrementarse. Por Ejemplo:

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



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

10.2.- Ciclos @each

La sintaxis habitual de la directiva @each es la siguiente:

@each \$var in <lista o mapa>.

El valor \$var puede ser cualquier variable
<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.

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



```
.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");  
}
```

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.

Por 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;  
    }  
  }
```

```
.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;  
}
```

10.3.- Ciclos @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.

Por ejemplo:

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



```
.item-6 {  
  width: 12em;  
}  
  
.item-4 {  
  width: 8em;  
}  
  
.item-2 {  
  width: 4em;  
}
```

ACADEMIA DE SOFTWARE