SASS

INTRODUCCIÓN

Sass es una extensión de CSS que añade potencia. Se permite el uso de variables, reglas anidadas, mixins, las importaciones en línea, y más, todo con una sintaxis totalmente compatible con CSS. Sass ayuda a mantener grandes hojas de estilo bien organizados, y obtener pequeñas hojas de estilo en marcha rápidamente. 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.

SINTAXIS

Sass permite el uso de dos sintaxis diferentes para crear sus archivos:

- La primera, conocida como SCSS (del inglés, Sassy CSS) es en la que nos vamos a centrar ahora y es una extensión de la sintaxis de CSS3. Esto significa que cualquier hoja de estilos CSS3 válida también es un archivo SCSS válido.
  Los archivos creados con esta sintaxis utilizan la extensión `.scss`.
- La segunda sintaxis, conocida como **"sintaxis indentada"** o simplemente "sintaxis sass" permite escribir los estilos CSS de manera más concisa. En este caso, el anidamiento de selectores se indica con tabulaciones en vez de con llaves y las propiedades se separan con saltos de línea en vez de con puntos y coma. Los archivos creados con esta segunda sintaxis utilizan la extensión .sass.
  Archivo scss

```
$main: #444;
.btn {
  color: $main;
  display: block;
}
```

```
.btn-a {
  color: lighten($main, 30%);
  &:hover {
    color: lighten($main, 40%);
  }
}
```

Archivo css

```
.btn {
  color: #444444;
  display: block;
}
.btn-a {
  color: #919191;
}
.btn-a:hover {
  color: #aaaaaa;
}
```

INSTALAR SASS

Para la instalación de Sass en cualquier sistema podemos visitar el siguiente link. Pero si utilizamos Visual Code, ésta utiliza una extensión que nos permite realizar la transpilación de código sass a css de forma automática. La extensiones que tienes que instalar son las siguientes:

- Live Sass Compiler.
- Sass

COMENTARIOS

Sass también soporta los comentarios de una única línea que utilizan los delimitadores // además de los delimitadores `/* y /*` que pueden ocupar una o más líneas que estaban aceptados ya.

Comentarios sass

```
// Estos comentarios no
// saldrán en los ficheros
// css compilados
/* este saldrá */
```

Comentarios css

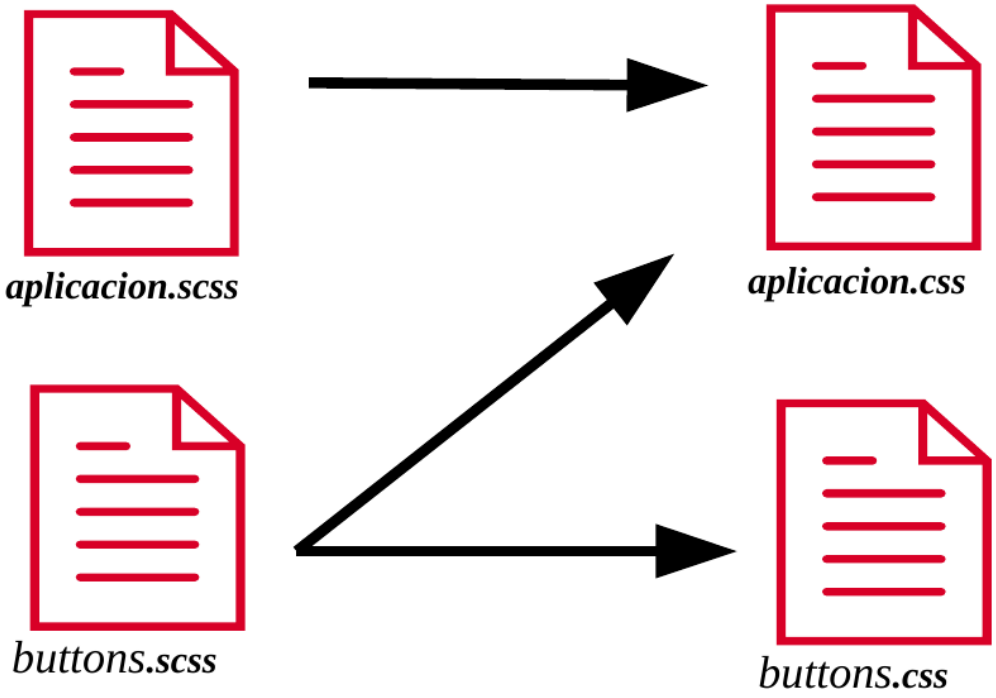
```
/*este saldrá */
```

LA REGLA `@IMPORT`

La regla css @import se ha mejorado previene la descarga paralela. Los archivos importados se buscan automáticamente en el directorio actual.

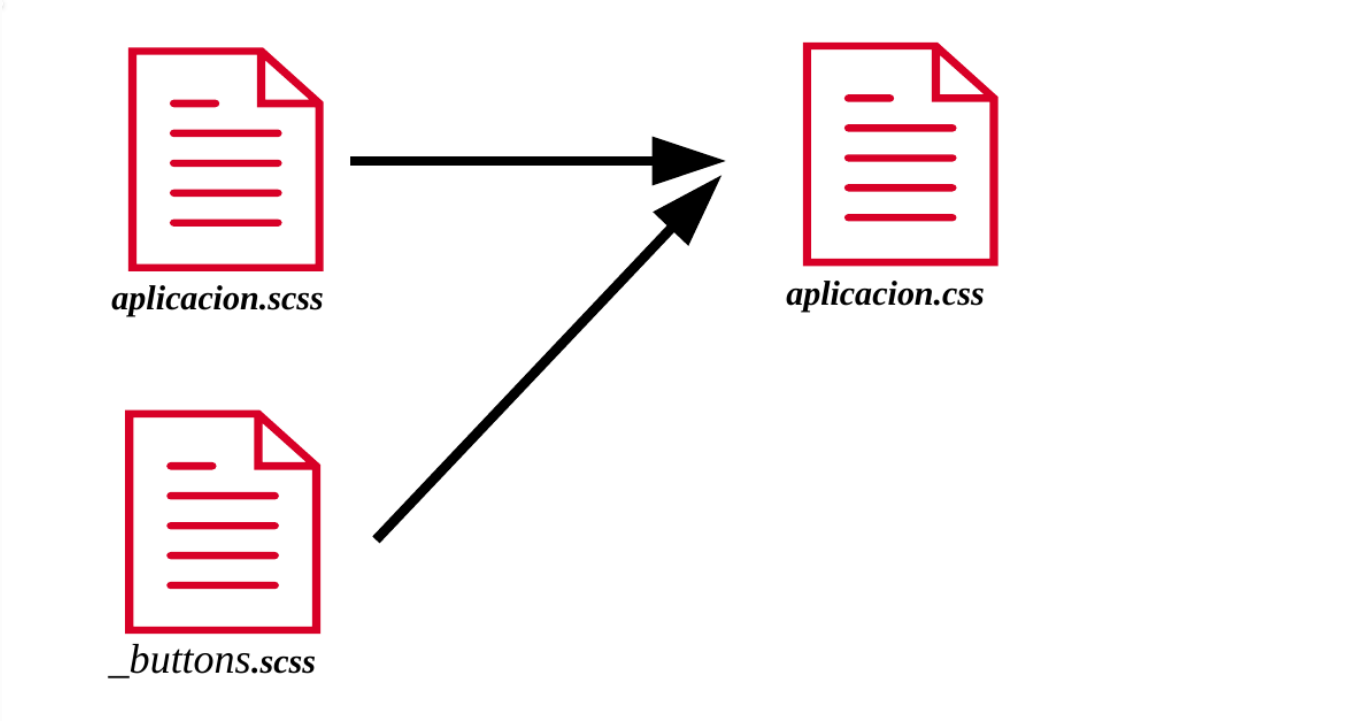
- La importación con .scss o .sass sucede durante la compilación en lugar de en el lado del cliente.
- La extensión del archivo es opcional.

```
// Importa estilos encontrados en 'boto.scss'
// Cuando el compilador procese aplicación.scss
@import "buttons";
```



HOJAS DE ESTILOS 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").



ANIDANDO SELECTORES

Cuando escribimos código HTML podemos observar que tiene una clara disposición jerárquica de manera anidada, por otro lado CSS no lo hace.

Sass permite anidar los selectores CSS de manera que sigue la misma jerárquica visual de HTML. Hay que tener en cuenta que las reglas demasiado anidadas resultan ser difícil de mantener y se considera una mala práctica. Vamos a ver un ejemplo:

Anidamiento sass

```
.content {
  border: 1px solid #ccc;
  padding: 20px;
  H2 {
    font-size: 3em;
    margin: 20px 0;
  }
  p {
    font-size: 1.5em;
    margin: 15px 0;
  }
}
```

Resultado css

```
.content {
  border: 1px solid #ccc;
  padding: 20px;
}
.content h2 {
  font-size: 3em;
  margin: 20px 0;
}
.content p {
  font-size: 1.5em;
  margin: 15px 0;
}
```

¿Cual será la CSS del siguiente código?

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

Además, podemos anidar ciertas propiedades con coincidencias en el nombre de la propiedad, de la siguiente manera. Veamos un ejemplo:

```
.btn
{
text: {
decoration: underline;
transform: lowercase;
}
```

Resultado css

```
.btn {
  text-decoration: underline;
  text-transform: lowercase;
}
```

EL SELECTOR PADRE &

Cuando estemos anidando los elementos podemos utilizar el siguiente & símbolo haciendo referencia al padre. Vemos el siguiente ejemplo:

```
.content {
  border: 1px solid #ccc;
  padding: 20px;
  .callout {
    border-color: red;
  }
  &.callout {
    // Hacer referencia a .content
    border-color: green;
  }
}
```

Resultado css

```
```

```
.content {
  border: 1px solid #ccc;
  padding: 20px;
}
.content .callout {
  border-color: red;
}
.content.callout {
  border-color: green;
}
```

Otro ejemplo muy utilizado es el siguiente:

```
a {
  color: #999;
  &:hover {
    color: #777;
  }
  &:active {
    color: #888;
  }
}
```

Resultado css

```
a {
  color: #999;
}
a:hover {
  color: #777;
}
a:active {
  color: #888;
}
```

Además, los selectores pueden ser añadidos antes del símbolo &:

```
```

```
.sidebar {
  float: right;
  width: 300px;
}
.users & {
  .users .sidebar {
    width: 400px;
  }
}
```

Resultado css

```
.sidebar {
  float: right;

  width: 300px;
}

.users .sidebar {
  width: 400px;
}
```

El padre hace referencia según el contexto dónde esta incluido. Vamos a ver un ejemplo.

```
.sidebar {
  float: right;
  width: 300px;
  h2 {
    color: #777;
    .users & {
      color: #444;
    }
  }
}
```

Resultado css

```
```

```
.sidebar {
  float: right;
  width: 300px;
}
.users .sidebar h2 {
  width: 400px;
}
```

El anidamiento como acabamos de observar es muy fácil. Pero como dijimos anteriormente es muy peligroso. Un anidamiento innecesario incrementa demasiado la especificidad de los selectores. Aquí tenemos un ejemplo.

```
.content {
  background: #ccc;
  .cell {
    h2 {
      a {
        &:hover {
          color: red;
        }
      }
    }
  }
}
```

Resultado css

```
.content {
  background: #ccc;
}
.content .cell h2 a:hover {
  color: red;
}
```

Como podemos observar en el css, tenemos un cuarto nivel de especificidad. Esto puede ser muy difícil de anular posteriormente. Deberemos tratar de limitar su anidamiento a tres o

cuatro niveles de profundidad. Si pasamos este limite, deberemos pensar en crear nuevas clases o refactorizar el nivel de profundidad.

VARIABLES

En esta sección veremos los tipos de variables, el ámbito de las variables y el concepto de interpolación.
El soporte nativo de variables en CSS esta todavía en desarrollo pero Sass nos permite una manera de asignar valores **reutilizables**. Los nombres de las variables empiezan con `$`, como `$base`.

Veamos un ejemplo:

```
$base: #777777;
.sidebar {
border: 1px solid
border: 1px solid $base;
}
```

Resultado css

```
.sidebar {
  border: 1px solid #777777;
}
.sidebar p {
  color: #777777;
}
```

Las variables en su definición pueden opcionalmente tener el flag `!default`. De tal manera que cuando dos variables con el mismo nombre y distintos valores, aquella que tenga el flag será el valor por defecto. Vamos a ver un ejemplo:

```
```

```
$title: 'Mi blog';
$title: 'Sobre mi' ;

 h2:before {
content: $title;
}
```

Resultado css

```
h2:before {
  content: "Sobre mi";
}
```

Acabamos de observar en el ejemplo de arriba que el valor de la variable se sobre escribe. Sin embargo, si ponemos el flag o indicador `!default` a la segunda variable, este será el valor por defecto pero no sobrescribirá el valor de la anterior.

```
$title: 'Mi blog';
$title: 'Sobre mi' !default ;

 h2:before {
content: $title;
}
```

Resultado css

```
h2:before {
  content: "Mi blog";
}
```

Que sucede cuando utilizamos este indicador en otros archivos scss importados. Una práctica bastante buena es utilizar lo valores por defecto en los archivos importados. Si

después queremos definir un valor diferente al que tenemos en el archivo importado colocaremos el valor antes de la importación del archivo. Veamos un ejemplo.

```
//aplicacion.scss
$rounded: 5px;
@import "buttons";
```

Mientras buttons.scss será:

```
$rounded: 3px !default;
.btn-a {
border-radius: $rounded;
color: #777;
}
.btn-b {
border-radius: $rounded;
color: #222;
}
```

Resultado css

```
.btn-a {
  border-radius: 5px;
  color: #777;
}
.btn-b {
  border-radius: 5px;
  color: #222;
}
```

Tipos de variables

Hay diferentes tipos de variables como:

Booleans

```
$rounded: false;
$shadow: true;
```

Números.

Que puede estar definidos con sus unidades o no.

```
$rounded: 4px;
$line-height: 1.5;
$font-size: 3rem;
```

Colores

```
$base: purple;
$border: rgba(0, 255, 0, 0.5);
$shadow: #333;
```

Cadenas

Que pueden tener o no comillas

```
$header: 'Helvetica Neue';
$callout: Arial;
$message: "Loading...";
```

Listas.

Los valores pueden estar separados con comas o con espacios.

```
$authors: nick, dan, aimee, drew;
$margin: 40px 0 20px 100px;
```

Null

```
$shadow: null;
```

Ámbito (scope) de las variables

El ámbito de las variables va ha depender del lugar donde las declaremos. Si declaramos una variable dentro del selector ese será su ámbito de utilización.
Ejemplo:

```
p {
$border: #ccc;
border-top: 1px solid $border;
}
h1 {
border-top: 1px solid $border;
}
```

Si compilamos el ejemplo anterior, nos proporcionará error de sintaxis. Dado que, la variable `$border` sólo está definida para `p`.
Además, si definimos nuevos valores fuera de la declaración cambia el futuro valor.

```
$color-base: #777777;
.sidebar {
    $color-base: #222222;
    background: $color-base;
    }
p {
```

```
    color: $color-base;
}
```

Resultado css

```
.sidebar {
  background: #222222;
}
p {
  color: #222222;
}
```

Hay que tener cuidado con el nombre de las variables. Por ejemplo, `$color-base` es más reutilizable que `$color-azul`.

Interpolación

Vamos a utilizar el esquema de variables que utiliza Ruby para `#{$variable}` con el fin de introducir las variables a los selectores, nombre de propiedades y cadenas. Veamos un ejemplo.

```
$side: top;
.sup {
position: relative;
top: -0.5em;
}
.callout-#{$side} {
background: #777;
}
```

Resultado css

```
sup {
  position: relative;
  top: -0.5em;
```

```
}
.callout-top {
  background: #777;
}
```

MIXIN

Algunas cosas en CSS son un poco tediosas para escribir, especialmente con CSS3 y los prefijos de muchos proveedores que existen. Un mixin te permite crear grupos de declaraciones CSS que deseas reutilizar en todo tu sitio. Incluso puede pasar valores para que su mixin sea más flexible. Un buen uso de un mixin es para prefijos de proveedores.

```
.btn-a {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
.btn-b {
  background: #ff0;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
```

Como podemos observar tenemos mucho código repetido con algunas propiedades diferentes.Podemos reutilizar este código de la siguiente manera:

```
@mixin button {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
```

El código quedaría de la siguiente manera.

```
@mixin button {
}
border: 1px solid #ccc;
font-size: 1em;
text-transform: uppercase;
}
.btn-a {
@include button;
background: #777;
}
.btn-b {
@include button;
background: #ff0;
}
```

Resultado css

```
.btn-a {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #777;
}
.btn-b {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #ff0;
}
```

El anterior bloque es ineficiente. Con css podemos generar un código más eficiente como el siguiente:

```
.btn-a,
.btn-b {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
```

```
.btn-b {
  background: #ff0;
}
```

Hay que asegurarse que el bloque `@mixin` está antes del `@include`, especialmente cuando importamos ficheros que contienen @mixin. Recordar que @include es para incluir mixin y @import es para incluir archivos.

Entonces, si esto es lo que pasa con mixin. Para qué son buenos?. Vamos a ver como añadirles argumentos.

Añadiendo argumentos

Vamos a ver, a través del ejemplo siguiente, como podemos utilizar mixin, pasándola argumentos que potencialmente cambien la salida css.

```
@mixin box-sizing {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

.content {
  @include box-sizing;
  border: 1px solid #ccc;
  padding: 20px;
}
```

Resultado css

```
.content {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  border: 1px solid #ccc;
  padding: 20px;
}
```

Ahora podemos al mixin anterior pasarle un argumento. Con lo quedaría de la siguiente manera.

```
@mixin box-sizing($x) {
  -webkit-box-sizing: $x;
  -moz-box-sizing: $x;
  box-sizing: $x;
}
.content {
  @include box-sizing(border-box);
  border: 1px solid #ccc;
  padding: 20px;
}

.callout {
  @include box-sizing(content-box);
}
```

Resultado css

```
.content {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  border: 1px solid #ccc;
  padding: 20px;
}

.callout {
  -webkit-box-sizing: content-box;
  -moz-box-sizing: content-box;
  box-sizing: content-box;
}
```

Opcionalmente, podemos incluir valores por defecto. De la siguiente manera.

```
@mixin box-sizing($x: border-box) {
  -webkit-box-sizing: $x;
  -moz-box-sizing: $x;
  box-sizing: $x;
```

```
}
.content {
  @include box-sizing;
  border: 1px solid #ccc;
  padding: 20px;
}
.callout {
  @include box-sizing(content-box);
}
```

Resultado css

```
.content {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  border: 1px solid #ccc;
  padding: 20px;
}

.callout {
  -webkit-box-sizing: content-box;
  -moz-box-sizing: content-box;
  box-sizing: content-box;
}
```

También podemos pasar múltiples argumentos. Pasados en orden y separados por coma.

```
@mixin button($radius, $color) {
  border-radius: $radius;
  color: $color;
}
.btn-a {
  @include button(4px, #000);
}
```

Resultado css

```
```

```
.btn-a {
  border-radius: 4px;
  color: #000;
}
```

Si nos olvidamos, por ejemplo, de uno de los argumentos, nos va a dar error de compilación mostrándonos el argumento que falta.
Opcionalmente, si ponemos un valor por defecto podríamos poner solamente el valor que falta. Veamos un ejemplo:

```
@mixin button($radius, $color: #000) {
  border-radius: $radius;
  color: $color;
}
.btn-a {
  @include button(4px);
}
```

Resultado css

```
.btn-a {
  border-radius: 4px;
  color: #000;
}
```

Si ponemos valores por defecto a los argumentos, como hemos visto, estos tienen que estar los primeros por la derecha. El siguiente ejemplo nos daría error de compilación.

```
@mixin button($color: #000, $radius) {
  border-radius: $radius;
  color: $color;
}
.btn-a {
  @include button(4px);
}
```

Además, podemos pasar el valor de cada uno de los argumentos, sin importar el orden, de la siguiente manera.

```
@mixin button($radius, $color: #000) {
  border-radius: $radius;
  color: $color;
}
.btn-a {
  @include button($color: #777, $rapius: 5 px);
}
```

Resultado css

```
.btn-a {
  border-radius: 5px;
  color: #777;
}
```

Qué pasa cuando las propiedades tengan un valor separado por una coma. Si intentamos pasarlo nos dará error de compilación. El compilador entiende que estamos pasando dos argumentos. El siguiente ejemplo nos proporciona error de compilación.

```
@mixin transition($val) {
  -webkit-transition: $val;
  -moz-transition: $val;
  transition: $val;
}
.btn-a {
  @include transition(color 0.3s ease-in, background 0.5s ease-out);
}
```

Para arreglarlo deberemos introducir al argumento del mixin `...`

```
@mixin transition($val...) {
  -webkit-transition: $val;
  -moz-transition: $val;
  transition: $val;
}
.btn-a {
  @include transition(color 0.3s ease-in, background 0.5s ease-out);
}
```

Resultado css

```
.btn-a {
  -webkit-transition: color 0.3s ease-in, background 0.5s ease-out;
  -moz-transition: color 0.3s ease-in, background 0.5s ease-out;
  transition: color 0.3s ease-in, background 0.5s ease-out;
}
```

También podríamos realizarlo al revés. Podríamos crear una lista y pasársela al mixin. Esta lista es dividida en argumentos por el mixin.

```
@mixin button($radius, $color) {
  border-radius: $radius;
  color: $color;
}
$propiedades: 4px, #000;
.btn-a {
  @include button($propiedades...);
}
```

Resultado css

```
.btn-a {
  border-radius: 4px;
  color: #000;
}
```

Interpolación y mixin

El objetivo es controlar el lado del borde y poder cambiar el color . Podemos crear el siguiente código como aproximación al objetivo.

```
@mixin highlight-t($color) {
  border-top-color: $color;
}
@mixin highlight-r($color) {
  border-right-color: $color;
}
@mixin highlight-b($color) {
  border-bottom-color: $color;
}
@mixin highlight-l($color) {
  border-left-color: $color;
}
.btn-a {
  @include highlight-r(#ff0);
}
```

Resultado css

```
.btn-a {
  border-right-color: #ff0;
}
```

Pero como vemos, es una utilización horrible e ineficiente del mixin. La solución será la utilización de la interpolación.

```
@mixin highlight($color, $side) {
  border-#{$side}-color: $color;
}
.btn-a {
  @include highlight(#ff0, right);
}
```

Resultado css

```
.btn-a {
  border-right-color: #ff0;
}
```

AMPLIACIONES (EXTEND).

Nos acordamos del anterior ejemplo cuando los mixin no producían un código efectivo y teníamos que utilizar código CSS puro.

```
@mixin button {
}
border: 1px solid #ccc;
font-size: 1em;
text-transform: uppercase;
}
.btn-a {
@include button;
background: #777;
}
.btn-b {
@include button;
background: #ff0;
}
```

Resultado css

```
.btn-a {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #777;
```

```
}
.btn-b {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #ff0;
}
```

Sass incorpora las función @extend para ampliar las propiedades de cada uno de los elementos. El anterior código lo podemos realizar de la siguiente manera.

```
.btn-a {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #777;
}
.btn-b {
  @extend .btn-a;
  background: #ff0;
}
```

Resultado css

```
.btn-a,
.btn-b {
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
  background: #777;
}
.btn-b {
  background: #ff0;
}
```

¿Qué es lo que hace?.

```
```

```
.btn-b {
  @extend .btn-a;
  background: #ff0;
}
```

Resultado css

```
.btn-a,
.btn-b {
}
.btn-b {
  /* Añade una segunda declaración para los valores únicos */
}
```

¿Qué pasa cuando utilizamos anidamientos y la ampliación?. Vamos a ver un ejemplo.

```
.content {
  border: 1px solid #ccc;
  padding: 20px;
  h2 {
    font-size: 3em;
    margin: 20px 0;
  }
}
.callout {
  @extend .content;
  background: #ddd;
}
```

Resultado css

```
.content,.callout {
border: 1px solid #ccc;
padding: 20px;
}
.content h2,.callout h2 {
font-size: 3em;
```

```
  margin: 20px 0;
  }
  .callout {
  background: #ddd;
  }
  }
```

Pero, hay que tener cuidado porque podemos generar estilos no deseados. En el siguiente ejemplo se han generado la clase de `.sidebar .btn-b` con las propiedades de `.sidebar .btn-a`. La del botón b no se quería incluir.

```
.btn-a {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
.btn-b {
  @extend .btn-a;
  background: #ff0;
}
.sidebar .btn-a {
  text-transform: lowercase;
}
```

Resultado css

```
.btn-a,
.btn-b {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
.btn-b {
  background: #ff0;
}
.sidebar .btn-a,
.sidebar .btn-b {
  text-transform: lowercase;
}
```

Este efecto lo podemos contrarrestar con la utilización del selector de referencia (placeholder selector). Antes de utilizar este tipo de selector, una buena práctica es siempre comprobar el css que se ha generado.

Los placeholder selector son identificados mediante con un `%`. Este, puede ser extendido pero nunca va a ser un selector propio.

```
$btn {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
.btn-b {
  @extend %btn;
  background: #ff0;
}
.sidebar .btn-a {
  text-transform: lowercase;
}
```

Resultado css

```
.btn-a,
.btn-b {
  background: #777;
  border: 1px solid #ccc;
  font-size: 1em;
  text-transform: uppercase;
}
.btn-b {
  background: #ff0;
}
.sidebar .btn-a {
  text-transform: lowercase;
}
```

Como vemos ya la clase .btn-b ya no esta en el ámbito. Además, es una buena práctica extender bloques comunes para evitar crear clases extras. Por ejemplo, la versiones anteriores a la 9 de IE tiene un limite de 4095 selectores por fichero CSS.

DIRECTIVAS

En esta sección veremos las siguientes directivas:

- Functions
- If
- Each
- For + While
- Mixin' In

Funciones

Para ver esta las funciones, antes vamos a ver una formula que utilizaremos para un diseño adaptable. Por ejemplo, si queremos una caja de 350px con un padre de 1000px hay que poner target / context . Es decir, 350px / 1000px = 0.35 0.35 * 100 = 35%. Crearemos el siguiente código. Los argumentos siguen las mismas reglas que los mixin.

```
$theme: dark;
Header {
  @if $theme == dark {
    background: #000;
  }
}
```

Resultado css

```
header {
  background: #000;
}
```

Estructuras de condición.

Ahora, vamos a utilizar la directiva `@if` con la que podemos condicional la salida de nuestro código.

```
@function fluidize($target, $context) {
  @return ($target / $context) * 100%;
}
.sidebar {
  width: fluidize(350px, 1000px);
}
```

Resultado css

```
.sidebar {
  width: 35%;
}
```

Las comparaciones posibles son:

- == igual a
- != no igual a
- > más grande que
- > = más grande que o igual a
- < menor que
- <= menor que o igual a.

En el siguiente ejemplo vamos a ver una estructura más compleja. En el que se utiliza, con `@if , @else if y @else`.

```
$theme: pink;
header {
  @if $theme == dark {
    background: #000;
  } @else if $theme == pink {
    background: pink;
  } @else {
```

```
    background: #fff;
  }
}
```

Resultado css

```
header {
  background: pink;
}
```

Interaccionar con una lista

La directiva `@each` nos permite realizar un bucle a través de cada uno de los elementos de esa lista.

```
$authors: nick aimee dan drew;
@each $author in $authors {
  .autor-#{$author} {
    background: url(author-#{$author}.jpg);
  }
}
```

Resultado css

```
.author-nick {
  background: url(author-nick.jpg);
}
.author-aimee {
  background: url(author-aimee.jpg);
}
.author-dan {
  background: url(author-dan.jpg);
}
.author-drew {
  background: url(author-drew.jpg);
}
```

Otras directivas son `@for y @while`.

```
.item {
  position: absolute;
  right: 0;
  @for $i from 1 through 3 {
    &.item-#{$i} {
      top: $i * 30px;
    }
  }
}
```

Resultado css

```
.item {
  position: absolute;
  right: 0;
}
.item.item-1 {
  top: 30px;
}
.item.item-2 {
  top: 60px;
}
.item.item-3 {
  top: 90px;
}
```

```
$i: 1;
.item {
position: absolute;
right: 0;
}
@while $i < 4 {
}
&.item-#{$i} {
top: $i * 30px;
}
```

```
$i:$i+1;
}
}
```

Resultado css

```
.item {
  position: absolute;
  right: 0;
}
.item.item-1 {
  top: 30px;
}
.item.item-2 {
  top: 60px;
}
.item.item-3 {
  top: 90px;
}
```

Como vemos @for y @while necesita un contador en los dos ejemplos (`$i`). Estas dos directivas nos dan más control que `@each` . Y aun más `@while` , con el cual podemos controlar los saltos. Por ejemplo, en el anterior código podemos hacer saltos de dos en dos.

```
$i: 2;pagebreak
.item {
position: absolute;
right: 0;
}
@while $i < 4 {
}
&.item-#{$i} {
top: $i * 30px;
}
$i:$i+2;
}
}
```

Resultado css

```
.item {
  position: absolute;
  right: 0;
}
.item.item-2 {
  top: 30px;
}
.item.item-4 {
  top: 60px;
}
.item.item-6 {
  top: 90px;
}
```

Vamos a hacer resumen de algunos conceptos:

- **Mixins:** Se utiliza para asignar propiedades similares utilizadas muchas veces con muy pocas variaciones.
- **Extend:** Asigna propiedades que son exactamente iguales.
- **Funtions:** Comúnmente se utiliza para retornar valores calculados que la vamos a utilizar a lo largo del código.

Podemos utilizar los Mixin con las directiva de condición. De la siguiente forma:

```
@mixin button($color, $rounded: true) {
  color: $color;
  @if $rounded == true {
    border-radius: 4px;
  }
}

.btn-a {
  @include button(#000, false);
}
.btn-b {
  @include button(#333);
}
```

Resultado css

```
.btn-b {
  color: #333;
  border-radius: 4px;
}

.btn-a {
  color: black;
}
```

También podemos recodificar el ejemplo anterior de la siguiente forma. Si `$rounded` es verdadero o null.

```
@mixin button($color, $rounded: false) {
  color: $color;
  @if $rounded {
    border-radius: 4px;
  }
}

.btn-a {
  @include button(#000);
}
.btn-b {
  @include button(#333, 4px);
}
```

Resultado css

```
.btn-b {
  color: #333;
  border-radius: 4px;
}

.btn-a {
  color: #333;
}
```

OPERACIONES MATEMÁTICAS Y COLOR.

Las operaciones básicas que se pueden realizar son:

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

Hay tener en cuenta que Sass por defecto redondea por encima de 5.
Con respecto a la división hay varias formulas que activan la division.

- Cuando hay una variable involucrada `$size/10`
- Cuando lo ponemos entre paréntesis `(100px / 20)`
- Otra tipo de operaciones como `20px\*5/7`.

Pero cuando está involucrado en la propiedad font no realiza la división. La compilación del siguiente código se queda de la misma forma. Ya que, 2em se refiere al tamaño y 1.5 a la separación de linea.

```
font: normal 2em/1.5 Helvetica, sans-serif;
```

Otro aspecto a considerar es cuando utilizamos + con dos string. No suma sino concatena dos strings. Por ejemplo.
Si utilizamos diferentes unidades Sass trata de combinarlas.

```
$family: "Helvetica " + "Neue"; // "Helvetica Neue"
$family: 'sans-' + serif // 'sans-serif'
$family: sans- + 'serif' // sans-serif
```

Si utilizamos diferentes unidades Sass trata de combinarlas.

```
```

```
h2 {
  font-size: 10px + 4pt;
}
```

Resultado css

```
h2 {
  font-size: 15.33333px;
}
```

Pero, algunas unidades son incompatibles.

```
h2 {
  font-size: 10px + 4em;
}
```

Resultado css

```
/* Incompatible units: 'em'
and 'px'. */
```

Tenemos algunas utilidades como:

- `round($number)` – Redondear al número entero más cercano
- `ceil($number)` - Redondea al número superior.
- `floor($number)` – Redondea al número inferior.
- `abs($number)` – Valor absoluto del número.
- `min($list)` – Devuelve el número mínimo de la lista.
- `max($list)` - Devuelve el número mínimo de la lista.
- `percentage($number)` – Convierte a formato porcentaje.

Ejemplos de utilización de las anteriores funciones:

Ejemplo 1

```
h2 {
  line-height: ceil(1.2);
}
```

Resultado css

```
h2 {
  line-height: 1;
}
```

Ejemplo 2

```
.caja {
  height: percentage(250px/500px);
}
```

Resultado css

```
.caja {
  height: 50%;
}
```

Ejemplo 3

```
$contexto: 500px .caja {
  height: percentage(250px / $contexto);
}
```

Resultado css

```
.caja {
  height: 50%;
}
```

UTILIZACIÓN DE LOS COLORES.

Con Sass podemos utilizar los colores de una manera más efectiva. Vamos a poder modificarlos mediante funciones y los podemos almacenar en variables que nos facilitan su recuerdo. Además de utilizar atajos.

```
$color-base: #333;
.suma {
  background: $color-base + #112233;
}
.resta {
  background: $color-base - #112233;
}
.mul {
  background: $color-base * 2;
}
.division {
  background: $color-base / 2;
}
```

Resultado css

```
.suma {
  background: #445566;
}

.resta {
```

```
  background: #221100;
}
.mul {
  background: #666666;
}
.division {
  background: #191919;
}
```

En el siguiente ejemplo vemos la utilización de una función.

```
$color-base: #333;
.aplha {
  background: rgba($color, 0.8);
}
.beta {
  background: rgba(#000, 0.8);
}
```

Resultado css

```
.beta {
  background: rgba(0, 0, 0, 0.8);
}
.alpha {
  background: rgba(51, 51, 51, 0.8);
}
```

Como vemos también podemos utilizar valores hexadecimales.

Siempre que sea posible, utilice las funciones de utilidad de color en lugar de la aritmética del color: más fácil de predecir y mantener.

Vamos a ver algunas funciones que nos van a interesar.

Para hacer color más brillante o menos . Toma un color y un número entre 0% y 100%, y devuelve un color con el brillo aumentado o disminuido. en esa cantidad.

```
$color-base: #333;
.brillo {
  color: lighten($color-base, 20%);
}
.oscuro {
  color: darken($color-base, 20%);
}
```

Resultado css

```
.brillo {
  background: #666666;
}
.oscuro {
  background: black;
}
```

Para saturar o desaturar un color.

```
$color-base: #333;

.saturate {
  color: saturate($color, 20%);
}
.desaturate {
  color: desaturate($color, 20%);
}
```

Resultado css

```
.saturate {
  background: #82d54e;
}
.desaturate {
```

```
  background: #323130;
}
```

Para mezclar utilizaremos la función mix. Podemos pasar dos argumentos y creará la mezcla con la media y opcionalmente ponderado (.desaturate).

El peso especifica la cantidad del primer color que debe incluirse en el color devuelto. El valor predeterminado, 50%, significa que se debe utilizar la mitad del primer color y la mitad del segundo color. 25% significa que se utilizará una cuarta parte del primer color y tres cuartos del segundo color.

```
.mix-a {
  color: mix(#ffff00, #107fc9);
}
.desaturate {
  color: mix(#ffff00, #107fc9, 30%);
}
```

Resultado css

```
.mix-a {
  background: #87bf64;
}
.mix-b {
  background: #57a58c;
}
```

Otras funciones sobre colores:

```
$color: #87bf64;
.grayscale {
  color: grayscale($color);
}
.invert {
  color: invert($color);
```

```
}
.complement {
  color: complement($color);
}
```

Resultado css

```
.grayscale {
  color: #929292;
}

.invert {
  color: #78409b;
}
.complement {
  color: #9c64bf;
}
```

Pero hay muchas más funciones. Como podemos ver en el siguiente link.