

Unidad 4: SASS

2º DAW

Curso 2022/2023

Mari Cruz Gómez García

ÍNDICE

1. INTRODUCCIÓN

2. INSTALACIÓN

2.1. EXTENSIÓN LIVE SASS

3. SINTAXIS BÁSICA

3.1. @import

3.2. variables

3.3. !default

3.4. ámbito de las variables

3.5. interpolación

3.6. comentarios

3.7. selectores anidados

3.8. & selector padre

3.9. anidamiento (nesting)

3.10. operaciones

3.11. mixin

3.12. @extend

3.13. vararg

4. LISTAS (o arreglos)

5. MAPAS

1. INTRODUCCIÓN

SASS es el preprocesador de CSS por excelencia.

Un preprocesador es una herramienta que nos permite escribir pseudocódigo CSS que luego será compilado y convertido en CSS.

Está formado por elementos típicos de cualquier lenguaje de programación: variables, ciclos, condicionales etc.

SASS va a leer ese código, lo va a compilar y va a generar una hoja de estilos CSS.

Las ventajas de SASS es que nos va a permitir organizar el código mucho mejor.

El navegador no lee SASS, lee el CSS.

3

1. INTRODUCCIÓN



4

1.1. VENTAJAS

Sass apareció en el 2006, y ya son muchos los grandes proyectos que han sido basados en Sass, como otros proyectos similares como Less y Stylus, así como una gran cantidad de frameworks y herramientas que son posibles gracias a estos.



5

1.1. VENTAJAS

1. Nos permite organizar nuestros ficheros CSS y hacerlos más modulares:

- Proporciona lógica
- Variables, reglas CSS anidadas
- Importación de hojas de estilos
- Es 100% compatible con CSS3
- Cálculo matemático
- Mixins: Reutilización de código
- Incluye un gran número de funciones para manipular colores, etc.
- Permite el uso de elementos básicos de programación como las directivas de control y las librerías.
- Herencia
- Nesting (anidamiento)

6

1.1. VENTAJAS

2. Puedes crear varios archivos scss y luego compilarlos todos como si fuera un único CSS
3. Es muy fácil de entender y aplicar para cualquiera que sepa CSS.

7

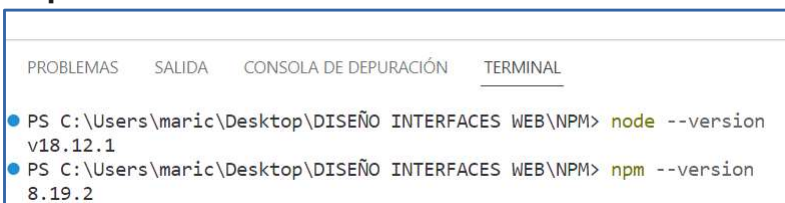
2. INSTALACIÓN

Hay 2 formas.

FORMA 1) Instalamos primero Node.js (entorno de ejecución de javascript que permite ejecutar aplicaciones en el servidor e incluye npm, que es un sistema de gestión de paquetes)

<https://nodejs.org/es/>

Comprueba desde la terminal de Visual Studio Code que lo tienes instalado:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
● PS C:\Users\maric\Desktop\DISEÑO INTERFACES WEB\NPM> node --version
v18.12.1
● PS C:\Users\maric\Desktop\DISEÑO INTERFACES WEB\NPM> npm --version
8.19.2
```

8

2. INSTALACIÓN

Crea el entorno:



Ahora tienes que instalar Node.js en tu proyecto con el comando: **npm init -y**

Te generará el package.json

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
● PS C:\Users\maric\Desktop\DISEÑO INTERFACES WEB\SASS> npm init -y
Wrote to C:\Users\maric\Desktop\DISEÑO INTERFACES WEB\SASS\package.json:

{
  "name": "sass",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

9

2. INSTALACIÓN

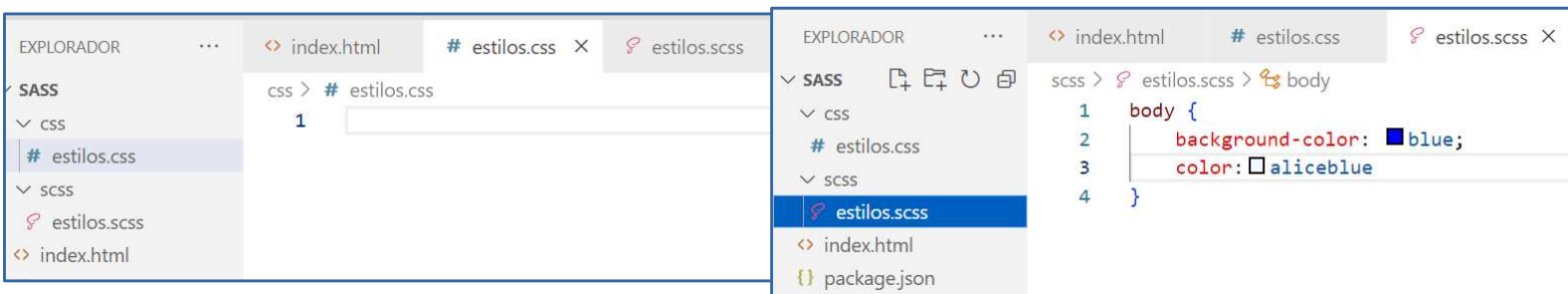
Ahora vamos a <https://sass-lang.com/> y a Install y vemos en la parte de comandos:

npm install -g sass

Ya lo tendríamos instalado. Ahora solo falta empezar a usarlo.

Recuerda que estos pasos habría que hacerlos en CADA PROYECTO.

Creamos los ficheros pero solo estilos.scss contiene código.



10

2. INSTALACIÓN

Se van a generar los estilos css cuando compilemos el código SASS con el siguiente comando en la terminal:

Sass - -watch carpetaOrigen:carpetaDestino

(`sass - -watch scss:css`)

Verás que se ha creado automáticamente todos los estilos en estilos.css.

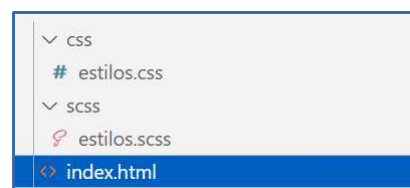
11

2.1. INSTALACIÓN CON UNA EXTENSIÓN

Instalamos la extensión Live Sass Compiler de Visual Studio:



Creamos también la estructura del proyecto:



12

2.1. INSTALACIÓN CON UNA EXTENSIÓN

Puedes ver que abajo del todo en Visual Studio aparece:



Introducimos los estilos en estilos.scss y automáticamente, está pendiente de cambios para ir generándolos solo en estilos.css.

13

3. SINTAXIS BÁSICA: @IMPORT

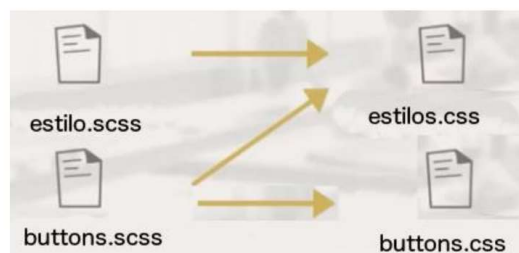
Cuando usamos `@import` en Sass podemos importar archivos con extensión `.scss` o `.sass` y la importación se lleva a cabo durante la compilación en lugar de en el lado del cliente. Además, no tenemos que incluir la extensión del archivo, es opcional.

Archivo: **estilo.scss**

```
// Importa los estilos encontrados en buttons.scss  
// Cuando el compilador compila estilo.scss
```

```
@import "buttons";
```

Por tanto, es durante el proceso de compilación donde `estilos.css` pasa a tener los estilos de los archivos: `estilos.scss` y `buttons.scss`



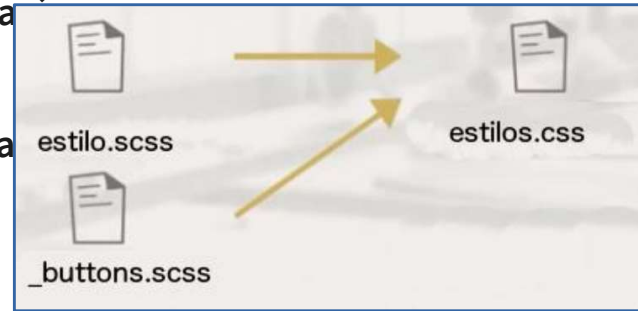
14

3. SINTAXIS BÁSICA: @IMPORT

Además de incluir la hoja de estilos buttons, cuando compilamos y nos genera el archivo estilos.css, también nos genera el CSS de buttons.

Para que no nos cree el archivo de buttons.css deberemos nombrar el archivo buttons.scss con el subrayado delante o lo que se llama partials, es decir:

- Si quiero que me cree el archivo buttons.css, lo llamo buttons.scss
- Si no quiero que me cree el archivo buttons.css, lo llamo _buttons.scss



15

3. SINTAXIS BÁSICA: variables

- Crear variables: `$nombre: valor;`

```
1 // VARIABLES
2 $colorFondo: #ccc;
3 $colorPrincipal: crimson;
4
5 *{
6     margin: 0;
7     padding: 0;
8     box-sizing: border-box;
9     font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu, Cantarell, 'Open Sans',
10     'Helvetica Neue', sans-serif;
11 }
12
13 body{
14     background: $colorFondo;
```

- ¿Que regla se crea en el css?

16

3. SINTAXIS BÁSICA: variables

Hay una gran cantidad de diferentes tipos de valores, podemos almacenar variables incluyendo booleanos como cierto/falso, números con o sin unidades, colores, strings, listas y nulos!

Booleanos

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

Números (podemos establecerlos con o sin unidades)

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

Colores

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

Strings (con o sin comillas)

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

Listas

```
$authors: Rosa, Pablo, Jose, Lucia;  
$margin: 40px 0 20px 100px;
```

Null

```
$shadow: null;
```

17

3. SINTAXIS BÁSICA: variables - !default

Con **!default** si no está definido en otro lado será usado por defecto.

aplicación.scss

```
$titulo: 'Mi blog';  
$titulo: 'Sobre mi' !default;  
  
h2:before{  
  content: $titulo;  
}
```

¿Qué regla se aplicaría en aplicación.css?

18

3. SINTAXIS BÁSICA: ámbito de las variables

¿Hay algún error al compilar?

application.scss

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

19

3. SINTAXIS BÁSICA: ámbito de las variables

Las variables que declaramos dentro de las llaves/ámbito/{} no pueden ser utilizadas fuera de ese bloque/scope, al igual que ocurre en la mayoría de lenguajes de programación.

Sin embargo, si definimos una variable fuera de una declaración, esa variable cambia para las futuras instancias ya que se crea como global.

Si definimos una variable como si fuera una variable global en nuestro archivo, y posteriormente redefinimos la variable (mismo nombre) para utilizarla en uno de los scopes internos, lo que estamos haciendo es sobrescribiendo el valor que tenía la variable en ese ámbito, y fuera del mismo tendrá el valor que se le había dado antes.

20

3. SINTAXIS BÁSICA: ámbito de las variables

¿Qué valor tendría al compilar?

```
application.scss
$color-base: #777;
.sidebar{
  $color-base: #222;
  background: $color-base;
}
```

¿Y aquí?

```
application.scss
$color-base: #777;
.sidebar{
  $color-base: #222;
  background: $color-base;
}
p{
  color: $color-base;
}
```

21

3. SINTAXIS BÁSICA: interpolación

Podemos usar el método “Ruby-esque” usando una almohadilla seguida de una apertura y cierre de llaves para añadir nuestras variables a selectores, nombres, propiedades o string.

```
application.scss
$side: top;
sup{
  position: relative;
  #{ $side }: -0.5em;
}
```



```
application.css
sup{
  position: relative;
  top: -0.5em;
}
```

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



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

22

3. SINTAXIS BÁSICA: comentarios

Como curiosidad, cuando comentamos nuestra hoja de estilo .scss, podemos usar tanto `//` como `/* */`, sin embargo, cuando la compilamos, sólo aparecen los comentarios que han sido introducidos con `/* */`, aquellos que tenían `//` desaparecen en el css que ha sido generado por el compilador.

Archivo: estilo.scss

```
// Este comentario
// dejará de aparecer
// al compilarlo

/* Este comentario sí aparecerá */
```

Archivo: estilo.css

```
/* Este comentario sí aparecerá */
```

3. SINTAXIS BÁSICA: selectores anidados

Archivo estilo.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;
}
```



Archivo estilo.scss

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

3. SINTAXIS BÁSICA: & es el selector padre

Archivo estilo.scss

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



Archivo estilo.css

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

25

3. SINTAXIS BÁSICA: 2 tipos de anidamiento (NESTING)

Ejemplo: sintaxis.html

```
<div class="articulo">
  <h1 class="titulo">Bienvenido a SASS</h1>
  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
</div>
```

CSS

```
.articulo {
  width: 500px;
  height: 400px;
  background: #FFF;
  padding: 20px;
  display: flex;
  align-items: center;
  flex-direction: column;
  justify-content: center;
  border: 2px solid crimson;
  transition: all .5s;
}

.articulo .titulo {
  color: crimson;
  font-size: 35px;
  font-weight: 500;
  font-style: italic;
  text-align: center;
  text-transform: uppercase;
  text-shadow: 0 0 5px rgba(0, 0, 0, 0.5);
}
```

SCSS



```
.articulo{
  width: 500px;
  height: 400px;
  background: #FFF;
  padding: 20px;
  display: flex;
  align-items: center;
  flex-direction: column;
  justify-content: center;
  border: 2px solid $colorPrincipal;
  transition: all .5s;
}

.titulo{
  color: $colorPrincipal;
  font: {
    size: 35px;
    weight: 500;
    style: italic;
  }
  text: {
    align: center;
    transform: uppercase;
    shadow: 0 0 5px rgba(#000000, .5);
  }
}
```

26

3. SINTAXIS BÁSICA: operaciones

Ver ejemplo operaciones.html

Podemos aplicar operaciones de +, -, * y / directamente, sin la función calc.

Si mezclamos unidades, SASS las convierte si son compatibles, sino, da fallo.

```
.container{
  width: 600px;
  margin: 0 auto;

  section{
    float: left;
    //Tenemos 600px en total, repartimos uno en 200 y otro 400 y sacamos la proporcion en %s
    width: 200px / 600px * 100%;
    height: 400px;
    background: #222 * 2;
  }

  aside{
    float: left;
    width: 400px / 600px * 100%;
    height: 400px;
    background: orangered * 2;
  }
}
```

27

3. SINTAXIS BÁSICA: operaciones

En cadenas, + es concatenar:

```
font: "Helvetica " + "Neue"; // "Helvetica Neue"
```

Cuando concatenamos string, si el operando de la izquierda tiene comillas simples, el resultado es un string con comillas simples. Sin embargo, si no va entre comillas, el resultado será sin ellas (aunque el otro operando las tenga).

```
font: 'sans-' + serif; // 'sans-serif'
font: sans- + 'serif'; // sans-serif
```

28

3. SINTAXIS BÁSICA: funciones numéricas predefinidas

Hay un gran número de funciones matemáticas predefinidas en Sass:

- round(\$number): Redondea al número entero más cercano
- ceil(\$number): Redondea hacia arriba
- floor(\$number): Redondea hacia abajo
- abs(\$number): Obtiene el valor absoluto
- min(\$list): Obtiene el número más pequeño de una lista
- max(\$list): Obtiene el número más grande de una lista
- percentage(\$number): Convierte el número en un porcentaje

3. SINTAXIS BÁSICA: funciones de colores

Dentro de este rango de funciones podemos encontrar:

- lighten(\$color, \$amount): Genera un color más claro
- darken(\$color, \$amount): Genera un color más oscuro
- saturate(\$color, \$amount): Modifica la intensidad del color (añadiéndole)
- desaturate(\$color, \$amount): Modifica la intensidad del color (quitándole)
- mix(\$color1, \$color2, [\$weight]): Mezcla dos colores, el tercer parámetro es opcional y lo que hace es indicar el % del primer color que usamos.
- grayscale(\$color): Convierte un color en escala de grises
- invert(\$color): Devuelve el inverso de un color
- complement(\$color): Devuelve el complementario.

3. SINTAXIS BÁSICA: mixin

application.css

```
.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 vemos tenemos un par de botones y su CSS, pero no es un buen CSS ya que tenemos tres propiedades repetidas en cada declaración. Para ayudar a combatir estas repeticiones, podemos usar mixin.

3. SINTAXIS BÁSICA: mixin

Un mixin comienza con @mixin y entonces le damos un nombre y dentro del bloque ponemos la parte del código que es común entre los dos botones.

application.scss

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


3. SINTAXIS BÁSICA: mixin

Cuando usamos mixin, debemos estar seguros de que nuestro estilo mixin está definido antes que el include que usamos, especialmente cuando usamos archivos importados que tienen mixin. Si tratamos de usar un mixin sin haber sido definido antes del include, Sass va a mostrarnos un error.

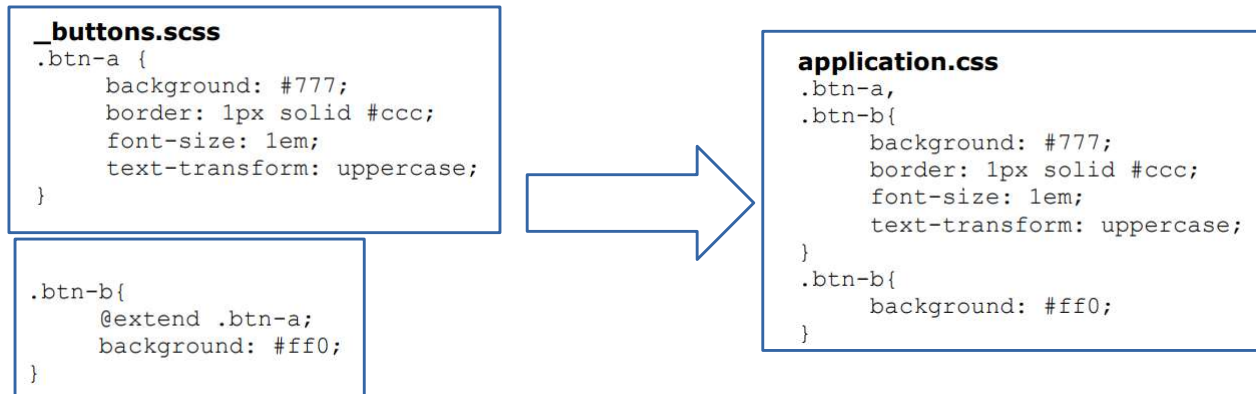
Además, debemos tener en cuenta que, usamos @include cuando añadimos un mixin mientras que si usamos @import lo que hacemos es importar el contenido de un archivo en nuestro archivo de sass.

Por tanto, si volvemos al archivo CSS que se había generado antes, cuando usamos el @mixin button, vemos que se ha generado el mismo CSS que teníamos antes de haber utilizado Sass, es decir, que el resultado que hemos obtenido tras la compilación tiene las mismas tres propiedades en los dos botones a los que les dimos estilo, y por tanto seguimos sin tener un CSS eficiente.

¿Cómo podemos arreglar ese CSS?

33

3. SINTAXIS BÁSICA: @extend



34

3. SINTAXIS BÁSICA: mixin

El verdadero poder de mixin se encuentra en la posibilidad de pasar argumentos.

Una vez hemos definido el mixin, podemos poner entre paréntesis y pasar los argumentos que queramos.

```
// MIXIN O FUNCIONES
@mixin estilosTexto($color, $fontSize, $fontWeight) {
  font-size: $fontSize;
  color: $color;
  font-weight: $fontWeight;
  text-transform: uppercase;
  text-align: center;
}

.articulo{
  width: 500px;
  margin: 50px auto;
  padding: 40px;
  border: 1px solid #ccc;

  h1{
    @include estilosTexto(lightseagreen, 40px, 500);
  }

  h2{
    @include estilosTexto(lightskyblue, 30px, 600);
  }
}
```

35

3. SINTAXIS BÁSICA: mixin

Los parámetros de entrada pueden ser opcionales y si no se les da valor, se usan los que están por defecto:

```
// MIXIN O FUNCIONES
@mixin estilosTexto($color: lightseagreen, $fontSize: 40px, $fontWeight: 600) {
  font-size: $fontSize;
  color: $color;
  font-weight: $fontWeight;
  text-transform: uppercase;
  text-align: center;
}

.articulo{
  width: 500px;
  margin: 50px auto;
  padding: 40px;
  border: 1px solid #ccc;

  h1{
    @include estilosTexto;
  }

  h2{
    @include estilosTexto(lightskyblue, 30px, 600);
  }
}
```

36

3. SINTAXIS BÁSICA: mixin

¿Daría error la instrucción en rojo?

application.scss

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

37

3. SINTAXIS BÁSICA: mixin

¿Cómo se arregla el ejemplo anterior?

OJO: los argumentos opcionales necesitan estar al final de nuestra cadena de argumentos. Por eso, el siguiente código daría error.

application.scss

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

38

3. SINTAXIS BÁSICA: mixin

Si quieres olvidarte del orden:

application.scss

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

39

3. SINTAXIS BÁSICA: vararg

Podemos crear una variable (vararg) en los parámetros de entrada de forma semejante al vargs de C/C++.

```
1  @mixin button ($radius, $color: #000){
2    border-radius: $radius;
3    color: $color;
4  }
5
6  $properties: 4px, #000;
7
8  .btn-a{
9    @include button($properties...);
10 }
11
```

Cuando le pasamos una variable separada entre comas, este lo usa como un solo argumento.

40

3. SINTAXIS BÁSICA: ejemplo

¿Es eficiente?

_buttons.scss

```
@mixin highlight-t($color){
  border-top-color: $color;
}
@mixin highlight-r($color){
  border-right-color: $color;
}
@mixin highlight-l($color){
  border-left-color: $color;
}
@mixin highlight-b($color){
  border-bottom-color: $color;
}
```

41

3. SINTAXIS BÁSICA: solución ejemplo

_buttons.scss

```
@mixin highlight($color, $side){
  border-#{ $side }-color: $color;
}
.btn-a{
  @include highlight(#f00, right);
}
```

42

4. LISTAS (o arreglos)

Una lista o vector es un tipo de variable que puede contener múltiples valores.

```
1 // Listas es un arreglo
2 $colores: blue, red, green, yellow, orange, pink;
3 $colores2: indigo, black, white;
```

Podemos acceder a los valores de la lista con la función nth. Admite 2 parámetros: nombre de la lista y posición (recuerda que parte de la posición 1 y no 0 como en otros lenguajes).

```
5 // Acceder a un elemento de una lista
6 h1{
7   color: nth($colores, 4);
8 }
```

43

4. LISTAS (o arreglos)

Combinar listas:

```
11 // Combinar listas
12 $coloresTotales: join($colores, $colores2, comma);
13 // blue, red, green, yellow, orange, pink, indigo, black, white
```

Agregar elementos:

```
21 // Agregar elementos a mis arreglos o listas
22 $agregarElemento: append($colores, black, comma);
23 // blue, red, green, yellow, orange, pink, black
24
25 h1{
26   color: nth($agregarElemento, 7);
27 }
```

44

5. MAPAS

A partir de su versión 3.3 Sass incorpora un nuevo tipo de dato: map, que permite crear y gestionar una lista o array clave/valor dentro de las hojas de estilo.

```
1 // Mapas son objetos
2 $titulos: (
3   h1: 40px,
4   h2: 30px,
5   h3: 20px,
6   h4: 15px,
7   h5: 10px,
8   h6: 8px
9 );
10
11 $colores: (
12   azul: blue,
13   rojo: red,
14   verde: green
15 );
```

45

5. MAPAS

Acceder a elementos con la función **map-get**:

```
18 // ACCEDER A ELEMENTOS DEL MAPA
19 h1{
20   font-size: map-get($titulos, h1);
21   color: map-get($colores, rojo);
22 }
```

También podemos unir dos mapas:



```
25 // MEZCLAR ELEMENTOS DEL MAPA
26 $titulosColores: map-merge($titulos, $colores);
27
28 // h1: 40px,
29 // h2: 30px,
30 // h3: 20px,
31 // h4: 15px,
32 // h5: 10px,
33 // h6: 8px,
34 // azul: blue,
35 // rojo: red,
36 // verde: green
37
38 h1{
39   font-size: map-get($titulosColores, h1);
40   color: map-get($titulosColores, verde);
41 }
42
```

46

5. MAPAS

Eliminar elementos:

```
45 // ELIMINAR ELEMENTOS DEL MAPA
46 $nuevoTitulo: map-remove($titulos, h1, h2, h3);
47 // h4: 15px,
48 // h5: 10px,
49 // h6: 8px
50
51 h1{
52   font-size: map-get($nuevoTitulo, h4);
53 }
```

map-get(map, key): Devuelve el valor de una clave dada.

map-has-key(map, key): Comprueba si existe un clave determinada dentro del map.

map-keys(map): Devuelve una lista con las claves del map.

map-values(map): Devuelve una lista con los valores del map.

map-merge(map1, map2): Permite fusionar maps.

map-remove(map, keys...): Elimina elementos dentro del map..