

Parte 2 de 4: Sass y Compass

VARIABLES CON SASS MIXINS

Apuntes de: **Rosa María Medina Gómez**

Adaptados a Power Point por José Jesús Torregrosa García

Curso de Formación del Profesorado a distancia
Cefire Específico de FP de Cheste

Generalitat Valenciana Curso 2018 - 2019



Introducción

A lo largo de esta unidad vamos a estudiar los ámbitos de una variable y su interpolación. Además, estudiaremos el término de mixin y cómo modificar estilos.

Variables con Sass: Ámbito


Abajo hemos definido la variable **\$border** dentro del estilo del párrafo, sin embargo, cuando en h1 se pretende usar esa misma variable que habíamos definido arriba nos da un error de compilación debido a que no podemos acceder a ella desde fuera de ese ámbito.

```
error sass/teoriaSesion2/application.scss (Line 5: Undefined variable: "$border")
```

```
application.scss x
1  p{
2      $border: #ccc;
3      border-top: 1px solid $border;
4  } h1{
5      border-top: 1px solid $border;
6  }
```



Variables con Sass: Ámbito

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 (**variables locales**). Sin embargo, si definimos una variable fuera de una declaración, esa variable cambia para las futuras instancias ya que se crea como **variable global**. Su archivo scss:

```
7   $border:  #ccc;
8   p{
9     |   border-top: 1px solid $border;
0   } h1{
1     |   border-top: 1px solid $border;
2   }
```

Variables con Sass: Ámbito

- Su CSS:


```
6  /* line 8, ../../sass/teoriaSesion2/application.scss */
7  p {
8  |   border-top: 1px solid  #ccc;
9  | }
10
11 /* line 10, ../../sass/teoriaSesion2/application.scss */
12 h1 {
13 |   border-top: 1px solid  #ccc;
14 | }
```

Variables con Sass: Ámbito

También podemos redefinir variables según su ámbito, su **scss**:

```
application.scss x
1  $color-base: #777;
2  .sidebar{
3      $color-base: #222;
4      background: $color-base;
5  }
```

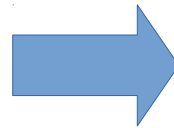
Variables con Sass: Ámbito

```
1  /* line 2, ../../sass/teor  
2  .sidebar {  
3  |   background:  #222;  
4  }
```

Variables con Sass: Ámbito

Si ahora intentamos usar esa misma variable `$color-base` para establecer otro estilo como puede ser un párrafo, vemos que el valor de la variable que habíamos definido como global, ha sido sobrescrito por el valor de sidebar

```
application.scss x
1  $color-base: #777;
2  .sidebar{
3      $color-base: #222;
4      background: $color-base;
5  }
6  p{
7      color: $color-base;
8  }
```

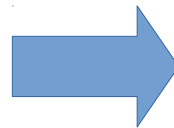


```
# application.css x
1  /* line 2, ../../sass/
2  .sidebar {
3      background: #222;
4  }
5
6  /* line 6, ../../sass/
7  p {
8      color: #777;
9  }
```


Interpolación

El último punto que veremos dentro del apartado de variables es la 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 x
1  $side: top;
2  sup{
3      position: relative;
4      #{ $side}: -0.5em;
5  }
6  $color-base: #777;
7  .sidebar{
8      $color-base: #222;
9      background: $color-base;
10 }
```

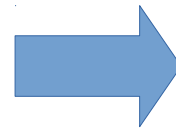


```
# application.css x
1  /* line 2, ../../sass/te
2  sup {
3      position: relative;
4      top: -0.5em;
5  }
```

Interpolación

Si añadimos esa interpolación como un selector, tenemos:

```
application.scss x
1  $side: top;
2  sup{
3    position: relative;
4    #{ $side}: -0.5em;
5  }
6  .callout-#{ $side}{
7    background: #777;
8  }
```



```
# application.css x
1  /* line 2, ../../sass/
2  sup {
3    position: relative;
4    top: -0.5em;
5  }
6
7  /* line 6, ../../sass/
8  .callout-top {
9    background: #777;
10 }
```

Interpolación

Es bastante útil el considerar el nombre de nuestras variables como hemos hecho con `.callout-top`, si usamos cosas que son más semánticas como `$color-base` conseguimos que tenga un mayor uso antes que si lo nombramos como `$color-blue`, principalmente si trabajamos con diferentes proyectos.

Mixin

Antes de comenzar con mixin, vamos a ver un scss “normal”.

```
application.scss x
1  .btn-a{
2      background: #777;
3      border: 1px solid #ccc;
4      font-size: 1em;
5      text-transform: uppercase;
6  }
7  .btn-b{
8      background: #ff0;
9      border: 1px solid #ccc;
10     font-size: 1em;
11     text-transform: uppercase;
12 }
```

Mixin

No es un buen **scss** ya que tenemos tres propiedades repetidas, para evitarlo utilizaremos **mixin**:

```
application.scss x
1  @mixin button{
2      border: 1px solid #ccc;
3      font-size: 1em;
4      text-transform: uppercase;
5  }
6  .btn-a{
7      @include button;
8      background: #777;
9  }
10 .btn-b{
11     @include button;
12     background: #ff0;
13 }
```



```
# application.css x
1  /* line 6, ../../sass/teoriaSe
2  .btn-a {
3      border: 1px solid #ccc;
4      font-size: 1em;
5      text-transform: uppercase;
6      background: #777;
7  }
8
9  /* line 10, ../../sass/teoriaS
10 .btn-b {
11     border: 1px solid #ccc;
12     font-size: 1em;
13     text-transform: uppercase;
14     background: #ff0;
15 }
```

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. Ya que, si tratamos de usar un mixin sin haber sido definido antes del include, Sass va a mostraros un error.

Además el CSS generado sigue siendo ineficiente (**repetitivo**)

Mixin

Por ahora es fácil escribir un CSS normal usando una coma separando lo selectores entre nuestros dos botones de forma que se comparta la propiedad.

```
application.scss x
1  .btn-a, .btn-b {
2      border: 1px solid #ccc;
3      font-size: 1em;
4      text-transform: uppercase;
5      background: #777;
6  }
7  .btn-b{
8      background: #ff0;
9  }
```



```
# application.css x
1  /* line 1, ../../sass/teoriaSesio
2  .btn-a, .btn-b {
3      border: 1px solid #ccc;
4      font-size: 1em;
5      text-transform: uppercase;
6      background: #777;
7  }
8
9  /* line 7, ../../sass/teoriaSesio
10 .btn-b {
11     background: #ff0;
12 }
```

Mixin

Otro de los puntos donde es bastante útil mixin es por ejemplo cuando tenemos por ejemplo este CSS, donde como vemos dependiendo del navegador que estemos utilizando la propiedad del box-sizing varía.

```
application.scss x
1  @mixin box-sizing {
2      -webkit-box-sizing: border-box;
3      -moz-box-sizing: border-box;
4      box-sizing: border-box;
5  }
6  .content {
7      @include box-sizing;
8      border: 1px solid #ccc;
9      padding: 20px;
10 }
```


Mixin

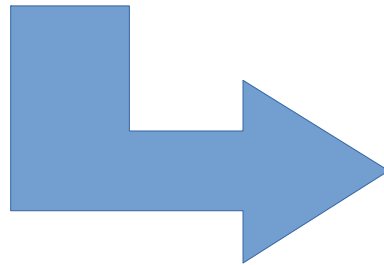
Estamos usando el **mixin** para copiar las propiedades y, por tanto, no es un uso muy eficiente que digamos del CSS.

El verdadero poder de mixin se encuentra en la **posibilidad de pasar argumentos** al mixin, una vez que le hemos definido el mixin, podemos poner entre paréntesis y pasar los argumentos que queramos,

Mixin

application.scss x

```
1 @mixin box-sizing ($x){
2     -webkit-box-sizing: $x;
3     -moz-box-sizing: $x;
4     box-sizing: $x;
5 }
6 .content {
7     @include box-sizing(border-box)
8     border: 1px solid #ccc;
9     padding: 20px;
10 }
11 .callout{
12     @include box-sizing(border-box)
13 }
```



application.css x

```
1 /* line 6, ../../sass/teoriaSesion
2 .content {
3     -webkit-box-sizing: border-box;
4     -moz-box-sizing: border-box;
5     box-sizing: border-box;
6     border: 1px solid #ccc;
7     padding: 20px;
8 }
9
10 /* line 11, ../../sass/teoriaSesion
11 .callout {
12     -webkit-box-sizing: border-box;
13     -moz-box-sizing: border-box;
14     box-sizing: border-box;
15 }
```

Mixin

Dentro del apartado de mixin, los parámetros de entrada pueden ser opcionales, de forma que si no se les da valor se utiliza uno por defecto.

```
application.scss x
1  @mixin box-sizing ($x:border-box){
2      -webkit-box-sizing: $x;
3      -moz-box-sizing: $x;
4      box-sizing: $x;
5  }
6  .content {
7      @include box-sizing(border-box);
8      border: 1px solid #ccc;
9      padding: 20px;
10 }
11 .callout{
12     @include box-sizing;
13 }
```

Mixin

O también:

application.scss x

```
1  @mixin box-sizing ($x: border-box){
2      -webkit-box-sizing: $x;
3      -moz-box-sizing: $x; box-sizing: $x;
4  }
5  .content {
6      @include box-sizing;
7      border: 1px solid #ccc;
8      padding: 20px;
9  }
10 .callout{
11     @include box-sizing(content-box);
12 }
```


Mixin

Podemos utilizar más de un argumento, estos deben estar separados por comas, por ejemplo:

```
application.scss x
1  @mixin button ($radius, $color){
2    |    border-radius: $radius;
3    |    color: $color;
4    |  }
5  .btn-login{
6    |    @include button(4xp,  #000);
7    |  }
```

Mixin

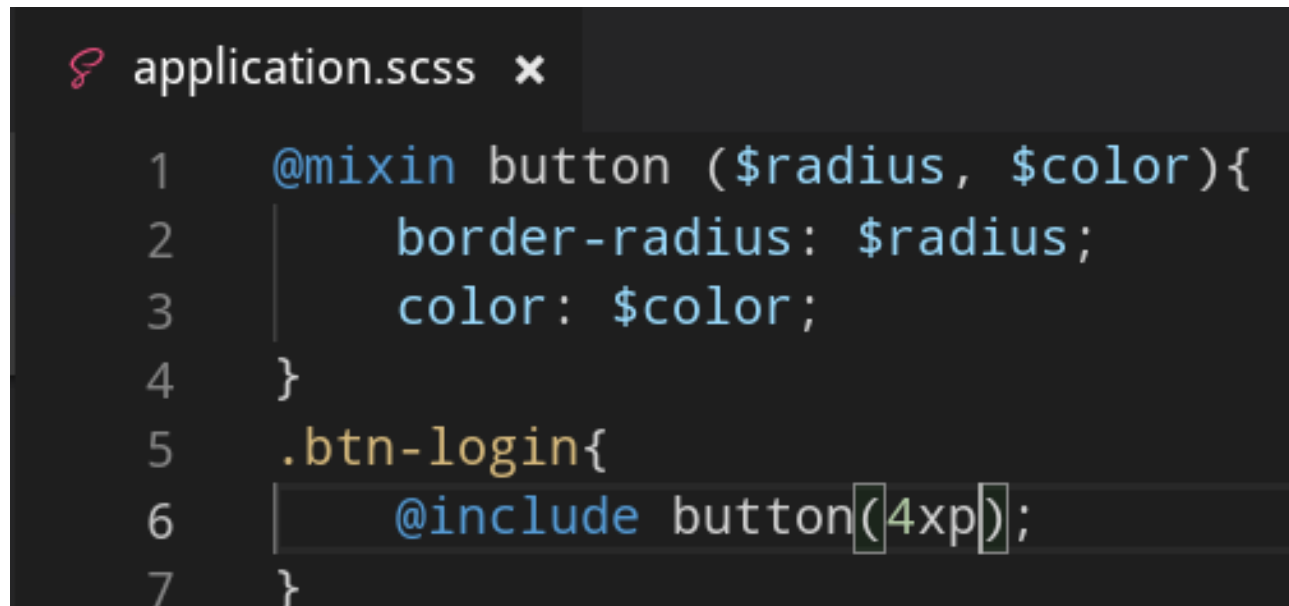
Y el resultado será:

```
# application.css x
1  /* line 5, ../../sass/teor
2  .btn-login {
3      border-radius: 4xp;
4      color:  #000;
5  }
```

Mixin

Si por ejemplo en lugar de enviarle dos argumentos, le enviamos sólo uno, en la compilación nos aparece un error:

```
error sass/teoriaSesion2/application.scss (Line 6: Mixin button is missing argument $color.)  
Compilation failed in 1 files.
```

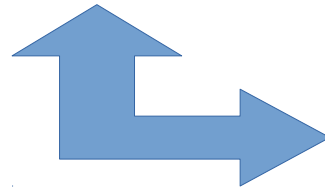


```
application.scss x  
1  @mixin button ($radius, $color){  
2      border-radius: $radius;  
3      color: $color;  
4  }  
5  .btn-login{  
6      @include button(4xp);  
7  }
```

Mixin

Lo que podemos hacer, es establecer que el segundo parámetro sea opcional y que en el caso de no enviarlo se le asigne uno por defecto.

```
application.scss x
1  @mixin button ($radius, $color: #000){
2      border-radius: $radius;
3      color: $color;
4  }
5  .btn-login{
6      @include button(4xp);
7  }
```



```
# application.css x
1  /* line 5, ../../sass.
2  .btn-login {
3      border-radius: 4xp;
4      color: #000;
5  }
```


Mixin

Cuando incluimos valores por defecto, y creamos argumentos opcionales en nuestros mixin, **los argumentos opcionales necesitan estar al final de nuestra cadena de argumentos**. Si el ejemplo anterior, hubiéramos puesto el color delante del radius e intentamos pasarle sólo un valor, el compilador nos mostrará un error:

error sass/teoriaSesion2/application.scss (Line 1: Required argument \$radius must come before any optional arguments.)

🔗 application.scss x

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

Mixin

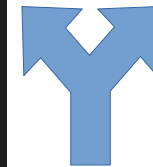
Podemos usar los argumentos de nuestro mixin para crear las parejas de valores que emplearemos para invocarlo independientemente del orden

application.scss x

```
1 @mixin button ($radius, $color: #000){
2   border-radius: $radius;
3   color: $color;
4 }
5 .btn-login{
6   @include button($color: red,$radius: 4xp);
7 }
```

application.css x

```
1 /* line 5, ../../sass
2 .btn-login {
3   border-radius: 4xp;
4   color: red;
5 }
```



Mixin

Algunos CSS pueden ser problemáticos cuando definimos múltiples argumentos, sin embargo, aquí tenemos la propiedad `transition`, la cual nos da la posibilidad de mediante comas separar diferentes valores de entrada para la `transition`

application.scss x

```
1  .btn-transition {  
2      -webkit-transition: color 0.3s ease-in, background 0.5s ease-out;  
3      -moz-transition: color 0.3s ease-in, background 0.5s ease-out;  
4      transition: color 0.3s ease-in, background 0.5s ease-out;  
5  }
```

Mixin

Si tratamos de enviar estas comas separadas en un CSS válido en mixin, vamos a tener algún que otro problemilla.

Por ejemplo, si escribimos una transición de mixin similares a: **_buttons.scss DARÁ ERRORES CUANDO SE USE**

 _buttons.scss x

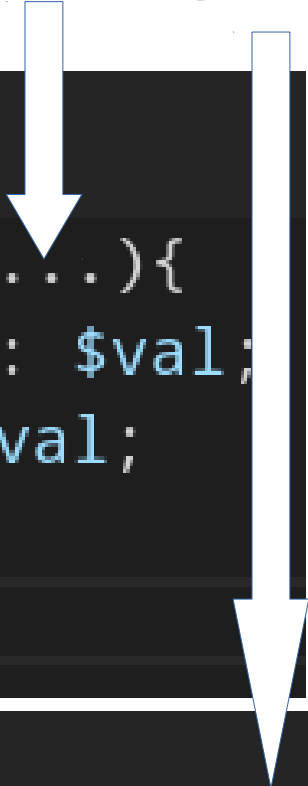
```
1  @mixin transition($val){
2      -webkit-transition: $val;
3      -moz-transition: $val;
4      transition: $val;
5  }
```


Mixin

Lo correcto sería poner y ya se puede usar:

 _buttons.scss x

```
1  @mixin transition($val...){
2      -webkit-transition: $val;
3      -moz-transition: $val;
4      transition: $val;
5  }
```



 application.scss x

```
1  @import "_buttons";
2  .btn-a{
3      @include transition(color 0.3s ease-in, background 0.5s ease-out);
4  }
```

Mixin



Ahora vamos a hacer uso de la variable arguments a la inversa. Si volvemos a la definición que habíamos hecho antes de:

application.scss x

```
1  @mixin button ($radius, $color: #000){
2      border-radius: $radius;
3      color: $color;
4  }
5  .btn-login{
6      @include button($color: red, $radius: 4px);
7  }
```

Mixin

Abajo **\$properties** tiene dos valores separados por comas, por tanto, podemos pasarlas en nuestro mixin usando la variable argument properties pero con los ...

```
5  @mixin button ($radius, $color:  #000){
6      border-radius: $radius;
7      color: $color;
8  }
9  $properties: 4px,  #000;
10 .btn-a{
11     @include button($properties...);
12 }
```

Mixin

_buttons.scss


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

The diagram illustrates the flow of data in the SCSS code. A yellow arrow points from the `$properties` variable to the `@include button($properties...);` call in the `.btn-a` class. Another yellow arrow points from the `@include` call to the `@mixin button` definition. A third yellow arrow points from the `$properties` variable to the `$radius` parameter in the `@mixin button` definition. A fourth yellow arrow points from the `$properties` variable to the `$color` parameter in the `@mixin button` definition.

Mixin

- Abajo aparece el fichero CSS

```
# application.css x
```

```
9      .btn-a {
10          border-radius: 4px;
11          color:  #000;
12      }
```

Mixin


Por último, dentro de los mixin vamos a ver algunos códigos no muy correctos, y vamos a ver cómo resolverlos. Por ejemplo:

 _buttons.scss x

```
1  @mixin highlight-t($color){  
2  |      border-top-color: $color;  
3  |  }
```

Mixin

En este mixin estamos permitiendo el enviar un color que será el que nos cambie el color del borde superior, las tres siguientes definiciones nos ponen el color al resto de los bordes

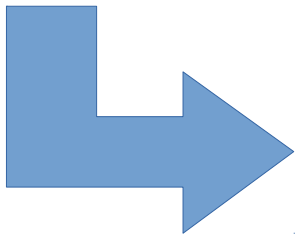
 _buttons.scss x

```
1  @mixin highlight-t($color){
2    |    border-top-color: $color;
3  }
4  @mixin highlight-r($color){ border-right-color: $color;
5  }
6  @mixin highlight-l($color){ border-left-color: $color;
7  }
8  @mixin highlight-b($color){ border-bottom-color: $color;
9  }
```

Mixin

Por tanto, si vamos a cambiar el lado derecho de nuestro botón incluiríamos en el fichero anterior, vemos claramente la ineficiencia:


```
application.scss x
1  @import "_buttons";
2  .btn-a{
3    @include highlight-r(■ #f00);
4  }
```



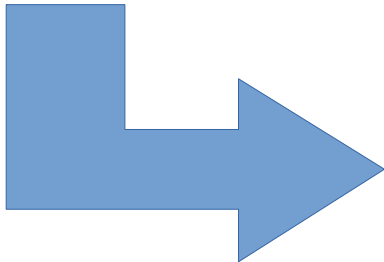
```
# application.css x
1  /* line 2, ../../sass/teoriaSesion2/application.scss */
2  .btn-a {
3    border-right-color: ■ #f00;
4  }
```


Mixin

Si usamos la interpolación que estudiamos previamente, podremos limpiar un poco nuestro código y que no sea tan repetitivo. Vamos a ver qué podemos hacer:

 _buttons.scss x

```
1  @mixin highlight($color, $side){  
2  |    border-#{ $side }-color: $color;  
3  | }
```



 application.scss x

```
1  @import "_buttons";  
2  .btn-a{  
3  |    @include highlight(■ #f00, right);  
4  | }
```

Bibliografía y/o páginas de interés

- Sass: <http://sass-lang.com/>
- Ruby: <https://www.ruby-lang.org/en/>
- Sass Wikipedia: [https://en.wikipedia.org/wiki/Sass_\(stylesheet_language\)](https://en.wikipedia.org/wiki/Sass_(stylesheet_language))
- Less wikipedia: [https://en.wikipedia.org/wiki/Less_\(stylesheet_language\)](https://en.wikipedia.org/wiki/Less_(stylesheet_language))
- Stylus wikipedia:
[https://en.wikipedia.org/wiki/Stylus_\(stylesheet_language\)](https://en.wikipedia.org/wiki/Stylus_(stylesheet_language))
- compass: <http://compass.kkbox.com/>
- scout: <http://mhs.github.io/scout-app/>
- Koala: <http://koala-app.com/>