Mixins

Un *mixin* es un fragmento de código Sass que podremos reutilizar para aplicar a los selectores que deseemos. Podríamos verlos como recetas donde agruparemos distintas reglas que luego se aplicarán a aquellos selectores a los que se asocie el *mixin.*

Para definirlos utilizaremos la palabra clave @mixin seguida del nombre que le queramos dar, seguido del bloque de reglas que queramos incoporar dentro de él:

[one_half]

```
// SCSS
$azulado: #165BFF;

@mixin link_chulo{
  color: $azulado;
  text-decoration: none;
  &amp;:hover{
        color: lighten($azulado, 20);
  }
}

.content {
  a {
    @include link_chulo;
  }
}
```

[/one_half]
[one_half_last]

```
/* CSS generado */
.content a {
  color: #165bff;
  text-decoration: none; }
  .content a:hover {
    color: #7ca3ff; }
```

[/one_half_last]

En este ejemplo hemos definido un *mixin* 'link_chulo' que hemos asignado a un determinado selector a través de *@include*. Como ves, el mixin es un bloque de reglas reutilizable que podremos aplicar a tantos selectores queramos. También he querido mostrar que dentro de un *mixin* podremos hacer uso de variables y que

también podemos anidar las reglas, como hemos hecho con *$azulado* y *&:hover* respectivamente.

Los *mixins* son muy apropiados a la hora de aplicar propiedades que no son estándar o debamos incluir varias versiones con prefijos que puedan funcionar en distintos navegadores. Por ejemplo, si queremos definir un borde redondeado de 5 píxeles de radio, podríamos definir el siguiente *mixin*:

```
@mixin bode_redondeado_5px{
  border-radius: 5px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  -o-border-radius: 5px;
}
.redondeado{
  @include borde_redondeado_5px;
}
```

Como podrás intuir, de esta forma generaremos un código más limpio y menos propenso a errores, ya que únicamente tendremos que incluir una línea cada vez que queramos el mismo borde de 5 píxeles de radio.

El problema en el ejemplo anterior es que solo podremos generar bordes de 5 píxeles de radio, teniendo que reescribir el mixin si queremos bordes de otro tamaño diferente. Para evitar eso podemos hacer uso de argumentos o parámetros como si de una función se tratase, de forma que podamos especificar qué valores deseamos que se apliquen dentro del mixin.

Podremos definir tantos argumentos como queramos, le indicaremos un nombre y opcionalmente un valor por defecto. A continuación se muestra un ejemplo donde hemos reescrito el mixin anterior para que soporte una variable, indicando además que su valor por defecto será 5px;

```
@mixin bode_redondeado($size: 5px){
  border-radius: $size;
  -moz-border-radius: $size;
  -webkit-border-radius: $size;
  -o-border-radius: $size;
}
```

```
.redondeado{
  @include borde_redondeado(10px);
}
```

@content: pasando bloques de contenido a un mixin

A la hora de utilizar un mixin, es posible definir un bloque de estilos específico que el mixin podría incorporar mediante la directiva @content. Veamos un ejemplo:

[one_half]

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

[/one_half]
[one_half_last]

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

[/one_half_last]

Como podemos ver, a la hora de invocar al mixin hemos definido el bloque de estilos asociado que sustituirá a la directiva @content dentro del mixin.

Interpolación

Las variables en Sass no solo las podremos utilizar para asignar valores a las propiedades, como ya hemos visto. También podremos utilizarlas para formar el nombre de las propiedades o incluso los propios selectores. Solo debemos utilizar #{$variable} allí donde queramos que sass sustituya por el valor de la variable:

[one_half]

```scss
/* SCSS */
$posicion_borde: left;
@mixin coche($marca, $color){
  .coche.#{$marca}{
          border-#{$posicion_borde}: 2px;
          background-color: $color;
          background-image: url('images/#{$marca}-#{$color}.jpg')
  }
}

@include coche('audi', 'green');
```

[/one_half]
[one_half_last]

```css
/* CSS generado */
.coche.audi {
  border-left: 2px;
  background-color: "green";
  background-image: url("images/audi-green.jpg"); }
```

[/one_half_last]

Como se puede comprobar en el ejemplo anterior, se pueden hacer uso de la interpolación en cualquier parte de nuestras reglas, donde también podremos utilizar los parámetros de los mixins pues no dejan de ser variables locales.

Directivas de control @if, @each, @for y @while

Sass nos ofrece distintas estructuras de control que nos permitirán incluir estilos en base a ciertas condiciones o construir conjuntos de estilos similares con pequeñas variaciones. Estas directivas van principalmente enfocadas en la generación de mixins o funciones reutilizables.

Por una parte tenemos @if, que nos permitirá establecer condiciones bajo las que se aplicarán las reglas o no. Estas condiciones podrán incluir comparadores típicos (==, !=, <, >) entre variables, constantes o cualquier expresión intermedia. Solo en caso de cumplirse la condición se ejecutará la generación de código del bloque asociado.

[one_half]

```scss
/* SCSS */
$animal: gato;
p {
  @if 1 + 1 == 2 {border: 2px solid black}
  @if $animal == gato {
    color: blue;
  } @else if $animal == perro {
    color: red;
  } @else if $animal == caballo {
    color: green;
  } @else {
    color: black;
  }
}
```

[/one_half]
[one_half_last]

```scss
/* CSS Generado */
p {
  border: 2px solid black;
  color: blue; }
```

[/one_half_last]

Las otras tres directivas, @each, @for y while, nos permitirán iterar sobre conjuntos de valores con los que podremos generar conjuntos de reglas que sean similares con pequeñas variaciones. Su estructura es muy similar a la que nos pudiéramos encontrar en los lenguajes de programación tradicionales. Veamos un ejemplo de cada una de ellas donde veremos cómo se definen y cual es su funcionamiento:

@for

Podremos definir una estructura @for de la siguiente manera:

```
@for $var from  [to|through]  {
  //Bloque de reglas donde podremos utilizar $var mediante interpolación
}
```

$var será el nombre de la variable que queramos utilizar en nuestro bloque, tanto <*start*>como <*end*> tendrán que ser expresiones SassScript válidas que devuelvan números enteros, y por último si indicamos ‘through’ se tendrán en cuenta los valores

<start> y <end> dentro del bucle, y si utilizamos ‘to’ no se tendrá en cuenta el valor <end> dentro del bucle. Veamos un ejemplo

[one_half]

```
/* SCSS */
@for $i from 1 to 3 {
  .todos-#{$i} { width: 2em * $i; }
}

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

[/one_half]
[one_half_last]

```
/* CSS Generado */
.todos-1 {
  width: 2em; }

.todos-2 {
  width: 4em; }

.casitodos-1 {
  width: 2em; }

.casitodos-2 {
  width: 4em; }

.casitodos-3 {
  width: 6em; }
```

[/one_half_last]

@each

Podemos definir una estructura @each de la siguiente manera:

```
@each $var in {
  //Bloque de reglas donde podremos utilizar $var
mediante interpolación
}
```

En este caso, <list> será cualquier expresión que devuelva una lista de elementós SassScript válida, es decir, una sucesión de elementos separados por comas. Veamos un ejemplo:

[one_half]

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

[/one_half][one_half_last]

```css
/* CSS Generado */
.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"); }
```

[/one_half_last]

@while

Para definir la directiva @while debemos asociarle una expresión SassScript que devuelva un valor booleano, y mientras dicha expresión sea cierta continuará generando los estilos del bloque interno que hayamos definido. Veamos un ejemplo.

[one_half]

```scss
/* SCSS */
$i: 6;
@while $i &gt; 0 {
  .item-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

[/one_half][one_half_last]

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

.item-4 {
  width: 8em; }

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

[/one_half_last]

Directiva @extend

En muchas ocasiones nos encontramos con la situación en la que una clase tiene todos los estilos de otra clase, además de los suyos propios.

En esos casos podremos hacer uso de @extend, el cual nos permite indicar una especie de herencia entre clases. Su funcionamiento quedará muy claro en el siguiente ejemplo:
[one_half]

```
/* SCSS */
.alerta {
  background: orange;
  display: block;
  font-weight: bold;
}
.alertaCritica{
  @extend .alerta;
  background: red;
}
```

[/one_half]
[one_half_last]

```
/* CSS generado */
.alerta, .alertaCritica {
  background: orange;
  display: block;
  font-weight: bold; }

.alertaCritica {
  background: red; }
```

[/one_half_last]

Como vemos, las dos clases del ejemplo comparen el mismo conjunto de estilos, y posteriormente se incluyen las personalizaciones de la segunda clase.

El ejemplo que acabamos de ver es de los más sencillos ya que únicamente intervienen dos clases, pero el concepto de extender se puede extender a cualquier tipo de selector siempre que este incluya un único componente, como por ejemplo a:hover, .link.disabled o a.user[href^=”https://”]. Es decir, no podremos extender selectores del tipo “a .disabled” o “.clase1 + .clase2”.

Otra funcionalidad que nos ofrece Sass es la posibilidad de crear “plantillas” destinadas exclusivamente a ser extendidas, de forma que si no las utilizamos, no se generará ningún CSS asociado. Estas plantillas se definen incluyendo un selector ficticio que empieza por % y que utilizaremos como identificador de la plantilla, que será sustituirdo por la clase que está extendiendo a la plantilla. Veamos un ejemplo:

[one_half]

```
/* SCSS
 Esta regla no generará CSS
 por sí misma
*/
#cuerpo a%plantilla{
  font-weight: bold;
  font-size: 2em;
}

/* Usamos la plantilla */
.alerta {
  @extend %plantilla;
}
```

[/one_half]
[one_half_last]

```
/* CSS generado */
#cuerpo a.alerta {
  font-weight: bold;
  font-size: 2em; }
```

[/one_half_last]

Directiva @media

@media es una directiva CSS bastante conocida ya que se introdujo en CSS2 para definir distintos tipos de estilos según el medio (display, print, etc.), y en CSS3 se ha enriquecido con las denominadas ***media queries***, las cuales permiten aplicar estilos en base determinados valores del medio, como por ejemplo el tamaño de pantalla del dispositivo. Esta última característica ha dado paso al denominado ***responsive design***, el cual permite que nuestro contenido se adapte a distintos tamaños de pantalla, dado que podremos aplicar estilos específicos a cada tamaño.

A partir de Sass 3.1 se ha introducido el concepto de **@media bubbling**, el cual nos permite definir *media queries* como si de selectores se tratase, pudiendo anidarlas dentro de nuestras estructuras de selectores. Sass extraerá el contenido del bloque asociado y generará una condición @media donde se asociará dicho contenido al selector formado según la ruta de anidamiento que correspondiese. Veamos un ejemplo para aclarar este funcionamiento:

[one_half]

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

[/one_half]
[one_half_last]

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

[/one_half_last]

Vemos que Sass forma un bloque @media aplicando los estilos asociados (width: 500px) al selector padre (.sidebar).

Además, también podremos utilizar variables e interpolación en cualquier parte de la definición de la *media query*:

[one_half]

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

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

```
}
```

[/one_half]
[one_half_last]

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

[/one_half_last]