



*OPERADORES, CONDICIONALES Y BUCLES*

# Operaciones

Con Sass puedes realizar **operaciones matemáticas básicas** en la misma hoja de estilo: es tan sencillo como poner el símbolo aritmético adecuado.

**SCSS**

| $ancho: 720px;  $blue: #4285F4;  $green: #33D374;  .box\_uno {  background-color: $blue;  width: $ancho/2;  }  .box\_dos {  background-color: $green;  width: ($ancho/2)-50;  }  .box\_tres {  background-color: $blue;  width: ($ancho/2)+50;  }  .box\_cuatro {  background-color: $green;  width: ($ancho/7)\*2;  } |
| --- |
|

**CSS**

| .box\_uno {  background-color: #4285f4;  width: 360px;  }  .box\_dos {  background-color: #33d374;  width: 310px;  }  .box\_tres {  background-color: #4285f4;  width: 410px;  }  .box\_cuatro {  background-color: #33d374;  width: 205.7142857143px;  } |
| --- |
|

Aunque *vanilla CSS* actualmente ofrece esta característica también en forma de *calc()*, la alternativa de Sass es más rápida, tiene operaciones % y puede ser aplicada a una gama más amplia de datos, como por ejemplo colores y strings.

# Condicionales

## **If**

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.**

| $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;  }  } |
| --- |
|

# Bucles

## **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. 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

**SCSS**

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

**CSS**

| .todos-1 {  width: 2em;  }  .todos-2 {  width: 4em;  }  .casitodos-1 {  width: 2em;  }  .casitodos-2 {  width: 4em;  }  .casitodos-3 {  width: 6em;  } |
| --- |
|

## **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 elementos SassScript válida, es decir, una sucesión de elementos separados por comas. Veamos un ejemplo:

**SCSS**

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

**CSS**

| .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");  } |
| --- |
|

Enlaces recomendados donde ahondan sobre distintos tipos de bucles, condicionales y operaciones

<http://thesassway.com/intermediate/if-for-each-while>

<http://librosweb.es/libro/sass/capitulo_8.html>

<https://www.video2brain.com/mx/tutorial/bucles-e-iteraciones-en-sass>

<https://www.emm-gfx.net/2016/07/bucle-for-con-sass-o-scss/>

<http://clubmate.fi/for-while-and-each-loops-in-sass/>

<https://alfonsomarin.com/desarrollo-web/tutoriales/sass-iii-mixins-interpolacion-y-directivas-de-control>

<http://sass-lang.com/documentation/file.SASS_REFERENCE.html>

*MAPS*

Este tipo de dato es relativamente nuevo, fue agregado en la versión 3.3 de Sass. **Se trata de una lista con llaves y valores**. Te sentirás muy cómodo con ellos si has tenido la oportunidad de leer un JSON. La sintaxis es muy similar y fácil de leer, a continuación un pequeño ejemplo:

| $map: (key1: value1, key2: value2, key3: value3); |
| --- |
|

A simple vista podemos apreciar la declaración de una variable de la misma manera que venimos haciéndolo en las unidades anteriores, la diferencia la encontramos en el contenido. **No guardamos un solo valor, sino un conjunto de ellos con sus respectivas llaves o identificadores**.

Hagamos un ejemplo un poco más complejo para que se puede apreciar con mayor claridad el poder de esta característica:

*Supongamos que necesitamos crear una serie de botones para compartir contenido, y nos exigen tres: Twitter, Facebook y envío por correo. Declaramos una variable Maps y agregamos en ella los tres colores que vamos a utilizar en nuestros estilos:*

| $colors: (  twitter: #55acee,  facebook: #3a5795,  send-mail: #C25E30  ); |
| --- |
|

Ahora utilizaremos la directiva *@each* para acceder a las llaves y valores de la variable *$colors* y con ello generamos los estilos de los botones, de la siguiente manera:

| @each $name-color, $bgcolor in $colors {  .btn--#{$name-color} {  background-color: $bgcolor;  }  } |
| --- |
|

El CSS procesado por Sass será el siguiente:

| .btn--twitter {  background-color: #55acee;  }  .btn--facebook {  background-color: #3a5795;  }  .btn--send-mail {  background-color: #C25E30;  } |
| --- |
|

Los **Maps**, como los array o los objetos JSON, también pueden ser **multidimensionales**. Esto es muy poderoso, nos permite elaborar listas que solo cambien su valor dependiendo de las diferentes capas o llaves padres que implementemos

Para más información acerca de los Maps:  
<http://saulsolorzano.com/trabajando-con-mapas-de-sass/>

<https://aula301.com/sass-introduccion/>

<https://www.viget.com/articles/sass-maps-are-awesome/>

<https://www.sitepoint.com/using-sass-maps/>

<https://codepen.io/aldiarian/pen/gMPwag>

<https://styde.net/aprende-sass-tipo-de-dato-maps-y-funciones/>

*MIXINS*

Nos permiten **definir estilos que puedan ser reutilizados en nuestro proyecto**.

Si se fijan, los conceptos de Extends y Mixins son muy parecidos. Por ello en un principio es posible confundirlos. Vamos a explicar dos diferencias entre ellos, que son claves para un entendimiento más claro:

Una de las mayores diferencias con los Extends, es que **los Mixins pueden recibir argumentos, los cuales te permitirán producir una gran variedad de estilos con unas simples líneas.**

La impresión en CSS de los Mixins no es la misma que tienen los Extends. Los Mixins se imprimen separados en cada uno de los lugares donde fueron declarados.

Ya tenemos un poco más claro cuales son las diferencias entre estas importantes características de Sass. Recuerden, los Extends los usaremos para compartir fragmentos de estilos idénticos entre componentes y **los Mixins para reutilizar fragmentos de estilos que puedan tener un resultado diferente en cada lugar donde los declaremos.**

Veamos un ejemplo:

¿Cuántas veces declaras el alto y ancho de un componente a lo largo de un proyecto? para evitar esta repetición, **podemos crear un pequeño Mixin que transforme las habituales dos líneas de código en una sola:  
  
SCSS**

| @mixin sizes($width, $height) {  height: $height;  width: $width;  }  .box {  @include sizes(500px, 50px);  } |
| --- |
|

**CSS**

| .box {  height: 50px;  width: 500px;  } |
| --- |
|

Más Información sobre los Mixins:  
<https://styde.net/aprende-sass-operators-mixins-y-extends/>

<http://sass-lang.com/guide>

<http://librosweb.es/libro/sass/capitulo_9.html>

<https://www.emenia.es/mixins-en-sass/>