

Preprocesadores

Contenidos





- Sass

...pero ¿Que es un preprocesador?

Muy fácil 🙌, es una extensión de CSS que añade características al lenguaje pero que no puede ser procesada directamente por navegadores, tiene que ser traducida (compilada) a CSS antes de poder utilizarla.

Podeis encontrar varios preprocesadores, por citar los principales, [Sass](#) 🔗, [Less](#) 🔗 y [Stylus](#) 🔗. Prácticamente ofrecen todos lo mismo con ligeras diferencias. En este curso nos vamos a centrar en *Sass*, que a priori, es el que está más implantado.

Sass

Como decía , *Sass* tiene una amplia aceptación en la industria (p. ej. Twitter lo usa) y 15 años de desarrollo a sus espaldas . Además, cuenta con varios frameworks como por ejemplo [compass](#)  o [bourbon](#) . El *core* de *Bootstrap* p. ej. está escrito fundamentalmente en *Less* pero soporta ambos preprocesadores.

Instalación

Para empezar a usar *Sass* tenemos dos opciones

- Aplicación de escritorio - Hay muchas de pago 💰, pero [Scout-App](#) es gratis 🥰
- Aplicación de consola de comandos - Lo más fácil es descargarse las fuentes precompiladas de [aquí](#) y añadirlas al `PATH` ([instrucciones](#)).


Si usais [nodejs](#) o gestores de paquetes como [chocolatey](#) o [homebrew](#) teneis mas info [aquí](#)

Sintaxis

Podéis escribir las hojas de estilo *Sass* de dos formas diferentes

- Ficheros **.scss** - Es la que os recomiendo ya que utiliza el mismo estilo que CSS de forma que todo el CSS válido es SCSS válido. Ejemplo 

```
@mixin button-base() {  
  @include typography(button);  
  @include ripple-surface;  
  @include ripple-radius-bounded;  
  
  display: inline-flex;  
  position: relative;  
  height: $button-height;  
  border: none;  
  vertical-align: middle;  
  
  &:hover { cursor: pointer; }  
  
  &:disabled {  
    color: $mdc-button-disabled-ink-color;  
    cursor: default;  
    pointer-events: none;  
  }  
}
```

- Ficheros **.sass** - Utiliza notación indentada, que véis mejor por aqui 

```
@mixin button-base()  
  @include typography(button)  
  @include ripple-surface  
  @include ripple-radius-bounded  
  
  display: inline-flex  
  position: relative  
  height: $button-height  
  border: none  
  vertical-align: middle  
  
  &:hover  
    cursor: pointer  
  
  &:disabled  
    color: $mdc-button-disabled-ink-color  
    cursor: default  
    pointer-events: none
```

Nosotros utilizaremos siempre la sintaxis SCSS.

Comentarios

2 tipos

- *Single-line* - Empiezan con `//`. Estos comentarios no se compilan a CSS.
- *Multi-line* - Comienzan con `/*` y terminan con `*/`. Estos sí compilan a CSS 🙌


```
// Este comentario no se traducirá a CSS

/* Este sí, excepto en modo comprimido */

/* Permiten interpolar:
 * 1 + 1 = #{1 + 1} */

/*! Este comentario se traducirá incluso en modo comprimido */

p /* Los comentarios multilínea se pueden escribir en cualquier lugar
 * y admiten espacios en blanco. */ .sans {
  font: Helvetica, // Los comentarios single-line también admiten espacios en blanco
    sans-serif;
}
```

el código SCSS de arriba se traduce en 

```
/* Este sí, excepto en modo comprimido */
/* Permiten interpolar:
 * 1 + 1 = 2 */
/*! Este comentario se traducirá incluso en modo comprimido */
p .sans {
  font: Helvetica, sans-serif;
}
```

Estilos en SASS

Los estilos en SASS funcionan igual que en CSS

- Eliges a que elementos aplicar un estilo con un selector y declaras propiedades que afectan al aspecto de esos elementos

Hasta aquí todo bien, o eso espero 😊...

Entonces ¿qué tiene SASS que ofrecernos en este contexto? Pues dos cosas, anidamiento 🥚 e interpolación. Lo vemos ⬇️

Anidamiento

Se ve mejor que se cuenta 👁️👁️ ⬇️

En **SCSS**

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

y en **CSS**

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

pero cuidado ⚠, podemos hacer más cosas, como utilizar listas de selectores ⬇

SCSS

```
.alert, .warning {  
  ul, p {  
    margin-right: 0;  
    margin-left: 0;  
    padding-bottom: 0;  
  }  
}
```

CSS

```
.alert ul, .alert p, .warning ul, .warning p {  
  margin-right: 0;  
  margin-left: 0;  
  padding-bottom: 0;  
}
```

y aun hay mas 🥳🥳

SCSS

```
ul > {  
  li {  
    list-style-type: none;  
  }  
}  
  
h2 {  
  + p {  
    border-top: 1px solid gray;  
  }  
}  
  
p {  
  ~ {  
    span {  
      opacity: 0.8;  
    }  
  }  
}
```

CSS

```
ul > li {  
  list-style-type: none;  
}  
  
h2 + p {  
  border-top: 1px solid gray;  
}  
  
p ~ span {  
  opacity: 0.8;  
}
```


... y sí, aun hay más que decir sobre anidamiento 🧑, mirad ⬇️

SCSS

```
.enlarge {  
  font-size: 14px;  
  transition: {  
    property: font-size;  
    duration: 4s;  
    delay: 2s;  
  }  
  
  &:hover { font-size: 36px; }  
}
```

CSS

```
.enlarge {  
  font-size: 14px;  
  transition-property: font-size;  
  transition-duration: 4s;  
  transition-delay: 2s;  
}  
.enlarge:hover {  
  font-size: 36px;  
}
```

Las diferentes propiedades `transition` se pueden anidar en lugar de utilizar el clásico *shortcut css*

En resumen *syntactic sugar* 🔍🔍



Interpolacion

Nos permite inyectar valores desde expresiones como variables o llamadas a funciones. Mirad 👁️ el siguiente SCSS

```
@mixin define-emoji($name, $glyph) {  
  span.emoji-#{ $name } {  
    font-family: IconFont;  
    font-variant: normal;  
    font-weight: normal;  
    content: $glyph;  
  }  
}  
  
@include define-emoji("women-holding-hands", "👭");
```

y aquí  su traducción

```
@charset "UTF-8";  
span.emoji-women-holding-hands {  
  font-family: IconFont;  
  font-variant: normal;  
  font-weight: normal;  
  content: "👩👧👦";  
}
```

hay varias anotaciones que todavía no hemos visto, pero básicamente inyecta
"women-holding-hands" en el selector y "👩👧👦" en la propiedad content  

Esconder declaraciones

En ocasiones solo queremos mostrar ciertas propiedades si se cumplen unas condiciones determinadas. *Sass* no compilará las propiedades con valores `null` o *strings* vacíos `""`


```
$rounded-corners: false;

.button {
  border: 1px solid black;
  border-radius: if($rounded-corners, 5px, null);
}
```

en el código de arriba, como la variable `rounded-corners` es `false`, `border-radius` se establece a `null` y no compila a `css`

```
.button {
  border: 1px solid black;
}
```

Selector padre

Seguro que os habeis dado cuenta de que en alguno de los códigos que hay  utilizo un `&` y quereis saber con desesperación qué es 😄...

Os presento al... SELECTOR PADRE ! 🧐

Es un selector que hace referencia al selector ascendiente o padre... sin más 🙄🙄. Es útil cuando usamos selectores anidados

```
.alert {  
  // Se puede usar en pseudo-clases  
  &:hover {  
    font-weight: bold;  
  }  
  
  // y como argumento para selectores de pseudo-clases  
  :not(&) {  
    opacity: 0.8;  
  }  
}
```

que se traduce a

```
.alert:hover {  
  font-weight: bold;  
}  
:not(.alert) {  
  opacity: 0.8;  
}
```

Variables

Y ahora vamos a ver el 🍞 and 🧈 (*bread and butter*) de todo lenguaje, las variables 😍.

Hay algún ejemplo de uso por 📦 pero en resumen ⚡

```
$base-color: #c6538c;  
$border-dark: rgba($base-color, 0.88);  
  
.alert {  
  border: 1px solid $border-dark;  
}
```

que compila a

```
.alert {  
  border: 1px solid rgba(198, 83, 140, 0.88);  
}
```


Valores por defecto

Si os apetece que vuestros módulos *Sass* sean configurables por el usuario cuando los carguen en sus hojas de estilo podéis marcar las variables con la flag `!default`.

Mirad el siguiente `SCSS` 

```
// _library.scss

$black: #000 !default;
$border-radius: 0.25rem !default;
$box-shadow: 0 0.5rem 1rem rgba($black, 0.15) !default;

code {
  border-radius: $border-radius;
  box-shadow: $box-shadow;
}
```

```
// style.scss

@use 'library' with (
  $black: #222,
  $border-radius: 0.1rem
);
```

Cuando el usuario compila el fichero `style.scss` con el *at-rule* `@use` está inyectando

```
$black: #222,  
$border-radius: 0.1rem
```

en el estilo declarado en `_library.scss`, que compilará a

```
code {  
  border-radius: 0.1rem;  
  box-shadow: 0 0.5rem 1rem rgba(34, 34, 34, 0.15);  
}
```

Una nota sobre interpolación

Ya hemos hablado sobre el uso de la interpolación pero quería hacer hincapie en su omnipotencia con la siguiente lista de lugares en los que puede usarse 😊...

- Selectores en estilos
- Nombres de propiedades
- Valores de propiedades
- *at-rules* CSS
- @extends
- @imports planos CSS
- *Strings*
- Funciones especiales
- Nombres planos de funciones CSS
- *Comentarios* tipo `//`

😓😓😓... casi me fundo una *diapo* con esto

At-Rules

Hasta ahora hemos visto algunas cosas útiles... y mucho azúcar sintáctico 🔍🔍, así que vamos al lío

@use

Otro viejo-nuevo conocido. Básicamente, nos permite importar *mixins*, funciones y variables desde otras hojas de estilo *Sass*.

Las hojas de estilo cargadas con `@use` se llaman **módulos**. *Sass* proporciona varios módulos *built-in* con funciones útiles.


Veamos un ejemplo básico

```
// foundation/_code.scss
code {
  padding: .25em;
  line-height: 0;
}
```

```
// foundation/_lists.scss
ul, ol {
  text-align: left;

  & & {
    padding: {
      bottom: 0;
      left: 0;
    }
  }
}
```

```
// style.scss
@use 'foundation/code';
@use 'foundation/lists';
```

Cuando compilamos `style.scss` obtenemos el siguiente fichero css 

```
code {  
  padding: .25em;  
  line-height: 0;  
}  
  
ul, ol {  
  text-align: left;  
}  
  
ul ul, ol ol {  
  padding-bottom: 0;  
  padding-left: 0;  
}
```

Esto nos permite modularizar nuestros estilos, lo que nos da un *boost* en mantenibilidad y hace nuestro código fácil de reutilizar.

Miembros, alias y miembros privados

Esto os sonara de *Java*, pero cuando incluís un modulo con `@use` podeis acceder a sus miembros con notación de punto 👁️👁️ ⬇️

```
// src/_corners.scss
$radius: 3px;

@mixin rounded {
  border-radius: $radius;
}
```

```
// style.scss
@use "src/corners";

.button {
  @include corners.rounded;
  padding: 5px + corners.$radius;
}
```

Podéis controlar la accesibilidad a los miembros de un módulo añadiendo un
delante

```
// src/_corners.scss
$-radius: 3px;




@mixin rounded {
  border-radius: $-radius;
}
```

```
// style.scss
@use "src/corners";

.button {
    @include corners.rounded;

    // ERRORRRRRRRRRRRRRR
    padding: 5px + corners.$-radius;
}
```


Configuración de módulos

Ya vimos [aquí](#)  como pasar parámetros para la carga de un módulo con ```@use...with(...)` , pero existe otra manera usando `@mixins``, mirad , o mejor, vamos a hacerlo 

Ejemplo 1

Vamos a definir en un módulo *Sass* 3 variables configurables, una de ellas será opcional. Luego cargaremos el módulo completo, lo configuraremos utilizando un `@mixin` e incluiremos el estilo resultante.

```
// _library.scss
$-black: #000;
$-border-radius: 0.25rem;
$-box-shadow: null;

/// Si el usuario ha configurado `$-box-shadow`, devuelve su valor configurado.
/// sino devuelve el valor derivado de `$-black`.
@function -box-shadow() {
  @return $-box-shadow or (0 0.5rem 1rem rgba($-black, 0.15));
}

@mixin configure($black: null, $border-radius: null, $box-shadow: null) {
  @if $black {
    $-black: $black !global;
  }
  @if $border-radius {
    $-border-radius: $border-radius !global;
  }
  @if $box-shadow {
    $-box-shadow: $box-shadow !global;
  }
}

@mixin styles {
  code {
    border-radius: $-border-radius;
    box-shadow: -box-shadow();
  }
}
```

```
// style.scss
@use 'library';

@include library.configure(
  $black: #222,
  $border-radius: 0.1rem
);


@include library.styles;
```

y el resultado tras compilarlo

```
code {
  border-radius: 0.1rem;
  box-shadow: 0 0.5rem 1rem rgba(34, 34, 34, 0.15);
}
```

Esta forma de configurar módulos es mucho más flexible que `@use...with(...)` y es la recomendada para usos avanzados.

Ficheros Index

Esto es muy sencillo, si escribes un fichero `_index.scss` en un directorio se carga de forma automática a partir de la *URL* del propio directorio 

```
// foundation/_code.scss
code {
  padding: .25em;
  line-height: 0;
}
```

```
// foundation/_lists.scss
ul, ol {
  text-align: left;

  & & {
    padding: {
      bottom: 0;
      left: 0;
    }
  }
}
```

```
// foundation/_index.scss  
@use 'code';  
@use 'lists';
```

```
// style.scss  
@use 'foundation';
```

compilará a

```
code {  
  padding: .25em;  
  line-height: 0;  
}  
  
ul, ol {  
  text-align: left;  
}  
ul ul, ol ol {  
  padding-bottom: 0;  
  padding-left: 0;  
}
```

@forward

Con `@forward` cargamos un modulo como con `@use` pero con la diferencia de que nos permite acceder a todos los miembros públicos del módulo cargado como si hubieran sido definidos en el fichero donde se usa `@forward` ... 😄😄

Vale, vale, un ejemplo

```
// src/_list.scss
@mixin list-reset {
  margin: 0;
  padding: 0;
  list-style: none;
}
```

```
// bootstrap.scss
@forward "src/list";
```

```
// styles.scss
@use "bootstrap";


li {
  @include bootstrap.list-reset;
}
```

que compila a

```
li {
  margin: 0;
  padding: 0;
  list-style: none;
}
```

y la clave  esta en `@include bootstrap.list-reset;` que nos permite acceder al miembro `list-reset` del módulo `_list` como si estuviera definido en `bootstrap`.

@mixin e @include

Como hemos visto en los ejemplos de arriba, un `@mixin` permite encapsular estilos y parametrizarlos mediante argumentos. Con `@include` inyectamos los `@mixins` en el contexto actual. Lo recordamos con un ejemplo 

```
@mixin reset-list {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
  
@mixin horizontal-list {  
  @include reset-list;  
  
  li {  
    display: inline-block;  
    margin: {  
      left: -2px;  
      right: 2em;  
    }  
  }  
}  
  
nav ul {  
  @include horizontal-list;  
}
```

compilará a

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
nav ul li {  
  display: inline-block;  
  margin-left: -2px;  
  margin-right: 2em;  
}
```

y sobre esto hay poco que contar... 💡 aunque podríamos tratar los tipos de argumentos que acepta un `@mixin` y de *content blocks*

Opcionales

Como dice su nombre, son argumentos que el usuario puede pasar o no

```
@mixin replace-text($image, $x: 50%, $y: 50%) {  
  text-indent: -99999em;  
  overflow: hidden;  
  text-align: left;  
  
  background: {  
    image: $image;  
    repeat: no-repeat;  
    position: $x $y;  
  }  
}  
  
.mail-icon {  
  @include replace-text(url("/images/mail.svg"), 0);  
}
```

y al compilarlo

```
.mail-icon {  
  text-indent: -99999em;  
  overflow: hidden;  
  text-align: left;  
  background-image: url("/images/mail.svg");  
  background-repeat: no-repeat;  
  background-position: 0 50%;  
}
```

Clave-Valor

Son argumentos que se pasan por nombre el orden no importa

```
@mixin square($size, $radius: 0) {  
  width: $size;  
  height: $size;  
  
  @if $radius != 0 {  
    border-radius: $radius;  
  }  
}  
  
.avatar {  
  @include square(100px, $radius: 4px);  
}
```

y el css

```
.avatar {  
  width: 100px;  
  height: 100px;  
  border-radius: 4px;  
}
```

mencionar que los mixins también soportan un número arbitrario de argumentos
(info [aquí](#) )

@function

Una función de las de toda la vida 🧐 que acepta 0 o más parámetros de un usuario, hace cosas dentro y devuelve el resultado ⬇️

```
@function pow($base, $exponent) {  
  $result: 1;  
  @for $_ from 1 through $exponent {  
    $result: $result * $base;  
  }  
  @return $result;  
}  
  
.sidebar {  
  float: left;  
  margin-left: pow(4, 3) * 1px;  
}
```


compila a

```
.sidebar {  
  float: left;  
  margin-left: 64px;  
}
```

Por otro lado, el uso de argumentos es idéntico al de los mixins

@error, @warn y @debug

Sirven para imprimir por consola mensajes de error, advertencia y *debug* en tiempo de compilación.

```
@mixin reflexive-position($property, $value) {  
    @if $property != left and $property != right {  
        @error "Property #{$property} must be either left or right."  
    }  
  
    $left-value: if($property == right, initial, $value);  
    $right-value: if($property == right, $value, initial);  
  
    left: $left-value;  
    right: $right-value;  
    [dir=rtl] & {  
        left: $right-value;  
        right: $left-value;  
    }  
}  
  
.sidebar {  
    @include reflexive-position(top, 12px);  
    //  
    // Error: Property top must be either left or right.  
}
```

Si compilamos el fichero de 

Error: "Property top must be either left or right."

[illegible]

```
example.scss 3:5  reflexive-position()
example.scss 19:3 root stylesheet
```

Flow control

Todo esto os resultará muy familiar, así que me voy a limitar a poner os ejemplos para que conozcáis la sintaxis

@if, @else y @else if

```
@use "sass:math";

@mixin triangle($size, $color, $direction) {
  height: 0;
  width: 0;

  border-color: transparent;
  border-style: solid;
  border-width: math.div($size, 2);

  @if $direction == up {
    border-bottom-color: $color;
  } @else if $direction == right {
    border-left-color: $color;
  } @else if $direction == down {
    border-top-color: $color;
  } @else if $direction == left {
    border-right-color: $color;
  } @else {
    @error "Unknown direction #{ $direction }.";
  }
}

.next {
  @include triangle(5px, black, right);
}
```

que compila a

```
.square-av {  
  width: 100px;  
  height: 100px;  
}  
  
.circle-av {  
  width: 100px;  
  height: 100px;  
  border-radius: 50px;  
}
```

@each

```
$sizes: 40px, 50px, 80px;

@each $size in $sizes {
  .icon-#{ $size } {
    font-size: $size;
    height: $size;
    width: $size;
  }
}
```

y el css

```
.icon-40px {  
  font-size: 40px;  
  height: 40px;  
  width: 40px;  
}  
  
.icon-50px {  
  font-size: 50px;  
  height: 50px;  
  width: 50px;  
}  
  
.icon-80px {  
  font-size: 80px;  
  height: 80px;  
  width: 80px;  
}
```


@for

```
$base-color: #036;  
  
@for $i from 1 through 3 {  
  ul:nth-child(3n + #{ $i }) {  
    background-color: lighten($base-color, $i * 5%);  
  }  
}
```

compila a

```
ul:nth-child(3n + 1) {  
  background-color: #004080;  
}  
  
ul:nth-child(3n + 2) {  
  background-color: #004d99;  
}  
  
ul:nth-child(3n + 3) {  
  background-color: #0059b3;  
}
```