



SASS

Antonio Espín Herranz

Contenidos

- Introducción
- Instalación de SASS
- Comentarios, Variables, Módulos
- Interpolación y Nesting
- Valores en Sass
- Estructuras de control
- Mixins y Funciones
- Módulos en Sass

Introducción

- **SASS** es un preprocesador, que nos permite escribir código CSS de una forma más dinámica, agregando sintaxis de un lenguaje de programación, incluyendo variables, funciones, módulos y distintos tipos de valores.
- Compilador de CSS
- Enlace:
 - <https://sass-lang.com/>

Introducción

- Totalmente compatible con CSS, toda la sintaxis de CSS se puede utilizar con SASS.
- Podemos escribir código más rápidamente
- Muchos FrameWorks se construyen con SASS.
- Hay mucha documentación.

Introducción

- Requisitos para aprender SASS
 - Conocimientos sobre CSS
 - Tener conocimientos de algún lenguaje de programación con condicionales, funciones y ciclos.

Tipos de Sintaxis

- Hay dos posibles sintaxis dentro de SASS: `.sass` y `scss`
- La sintaxis de **`.sass`**:
 - No utiliza llaves
 - Utiliza tabuladores como en Python
- La segunda **`.scss`**:
 - Similar a CSS
 - Lleva llaves.
 - Y agrega funcionalidades extras.

Introducción

- Sass NO reemplaza a CSS
 - Sass es una herramienta para escribir código CSS más rápido, limpio y modular, al final compilaremos nuestros archivos de Sass, para que se conviertan en archivos CSS.

Ejemplo

- **Archivo SASS**

```
.hero {  
  display: flex;  
  justify-content: space-between;  
  
  &__child {  
    flex:1  
  }  
}  
  
.main {  
  @extend .hero;  
}
```

- **Archivo CSS**

```
.hero, .main {  
  display: flex;  
  justify-content: space-between;  
}  
  
.hero__child {  
  flex:1  
}
```


Instalación de SASS

- Primero instalar **Node.js** (es un instalador de paquetes).
 - <https://nodejs.org/en/>
- Se descarga la versión **LTS**.
- Después comprobar desde un terminal de VSCode si está bien instalado.
 - **node -v** (muestra la versión de node).

Instalación de SASS

- Desde una terminal de VSCode, utilizamos el terminal. Y utilizamos el instalador de paquetes npm.
- Crear una nueva carpeta para VSCode.
 - Se puede llamar: **sass**
- Desde el terminal:
 - **npm init** → Para iniciar el instalador de paquetes.
 - Nos hará una serie de preguntas que se puede dar a todas enter.
 - Nos creará un archivo **json** (no lo utilizaremos, solo instalaremos **sass**).

Instalación de SASS

- Para Instalar **sass** tecleamos en el terminal de VSCode:
 - **npm install sass --save-dev**
 - Con esto agregamos sass a nuestro proyecto.
 - Se agregará la carpeta **node_modules** a nuestro proyecto.
- Ahora podemos instalar sass de forma **global**:
 - **npm install -g sass**
- **Con esto ya tenemos todo instalado**, creamos una nueva carpeta que se llame también sass.
 - En esta carpeta se añadirán los archivos de sass.
 - Si lo tenemos que desinstalar haremos: **npm uninstall -g package-name**

Compilar archivos con Sass

- Creamos un archivo **index.scss**

```
body {  
    font-family: 'Arial'  
}
```

- Para compilar (desde la consola) → **sass --watch sass/index.scss** (genera el archivo css).
- O se pueden compilar todos los archivos que estén en la carpeta de **sass** y que los deje en otra carpeta: **css**
- Está comprobando cualquier cambio en la carpeta sass para generar el archivo css y dejarlo en la carpeta css.
 - **sass --watch sass:css**

Posible error al compilar con sass

```
sass : No se puede cargar el archivo C:\Users\Anton\AppData\Roaming\npm\sass.ps1 porque la ejecución
de scripts está deshabilitada en este sistema. Para obtener más información, consulta el tema
about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkID=135170.
En línea: 1 Carácter: 1
+ sass --watch sass:css
+ ~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

- Cuando lanzamos el comando sass podemos obtener el siguiente error!
- Ejecutar Windows PowerShell como administrador.
 - <https://www.youtube.com/watch?v=gm0gexHWDy0>

En PowerShell

```
PS C:\WINDOWS\system32> Get-ExecutionPolicy -list
```

Scope	ExecutionPolicy
----	-----
MachinePolicy	Undefined
UserPolicy	Undefined
Process	Undefined
CurrentUser	Undefined
LocalMachine	Undefined

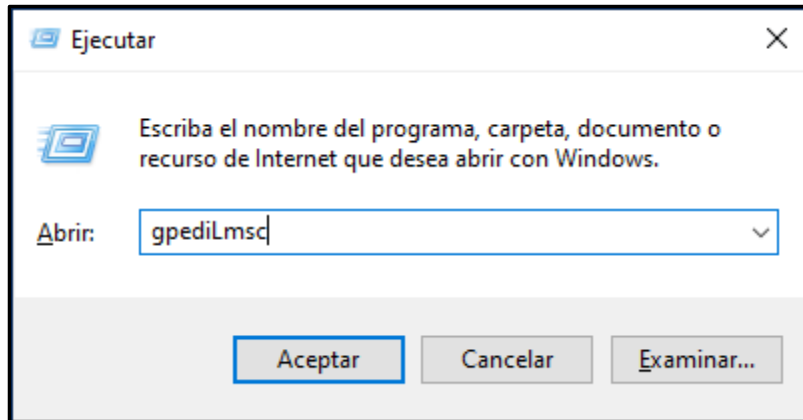
- Listar las políticas de seguridad.
- Están todas deshabilitadas.
- Lanzar el comando:
- Set-ExecutionPolicy RemoteSigned -Force

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned -Force
PS C:\WINDOWS\system32> Get-ExecutionPolicy -list
```

Scope	ExecutionPolicy
----	-----
MachinePolicy	Undefined
UserPolicy	Undefined
Process	Undefined
CurrentUser	Undefined
LocalMachine	RemoteSigned

Hace falta también activar **MachinePolicy**
Pulsar Windows+R
Se hace desde: **regedit**

gpediLmsc



- Este archivo no se encuentra en Windows 10 Home.
 - <https://www.minitool.com/news/group-policy-editor-gpedit-msc-missing.html>
- En estos casos o se intenta instalar en Windows: gpedit.msc o directamente **ejecutar Sass desde una consola de Windows.**

Ejecutar Sass

- *Si tenemos algún problema en ejecutar Sass desde la consola de VSCode, lanzar el mismo comando desde una consola de Windows.*
- Con este comando habrá generado la carpeta **css** con el archivo **index.css** y otro archivo: **index.css.map**
- Si dejamos ejecutándose el comando sass, cualquier otro archivo que se añada a la carpeta sass lo compilará para generar un css.
- **Compila de forma instantánea.**

```
sass --watch sass:css  
[2023-03-04 18:23] Compiled sass\index.scss to css\index.css.  
Sass is watching for changes. Press Ctrl-C to stop.
```


Comentarios

- En los archivos de sass (.scss) se pueden utilizar comentarios `//` (como en java) pero estos no se transfieren al archivo css compilado.
- Pero si utilizamos los comentarios `/* comentario */` al compilar si **se transfieren** al archivo de css.
- Se puede comprobar dividiendo la ventana del VSCode en dos mitades y abriendo el archivo scss y el generado: css.
 - Tener en ejecución el compilador de sass.

Variables en Sass

- Las variables en Sass empiezan por el \$
- Si ponemos guiones en el nombre de la variable. Sass no hace distinción entre el **guión medio y bajo**.
 - Las variables: \$font-size es lo mismo que \$font_size
 - Las variables luego se asignan a las propiedades de css (pero dentro del archivo scss).
 - Se inicializan con los dos puntos.
 - **\$nombreVariable: valor**

Ejemplo

prueba.scss

```
$primary: red;  
h1 {  
    color: $primary;  
}
```

prueba.css

```
h1 {  
    color: red;  
}
```

- Se pueden declarar variables locales.
- Se pueden asignar a varias reglas.
- El uso de variables permite cambios rápidos en las css, se cambian en el archivo scss y se vuelve a compilar.

Módulos y Partials en Sass

- Para crear un partial en Sass, creamos el archivo con un `_` (guión bajo como primer carácter del nombre).
- Este fichero `_fichero.scss` no se compila en un `css`. Este fichero se va a llamar desde otro elemento con la regla:
- `@use 'nombre_sin_extension'`
 - Se coloca al principio del fichero: **scss**
- Creamos **`_variables.scss`**
- **Desde otro fichero SCSS**, lo llamamos:
- `@use 'variables';` → sin el guión.

Partials

- Los **partials** en sass **son creados como módulos** para ser importados desde otros archivos.
- Dentro del partial se pueden **definir variables**.
- Cuando importamos el partial desde otro fichero se pueden **referenciar las variables, pero tienen que ir precedidas del nombre del módulo** (aunque se esté importando).

Ejemplo

_variables.scss

```
$color: red;
```

variables_uso.scss

```
@use 'variables';
```

```
body {  
    color: variables.$color;  
}
```

Los partials se pueden utilizar para definir
Múltiples variables que sean colores, fuentes, etc
Y después se importan a otro fichero para crear las reglas.

Variables

- A las variables se las puede dar un **valor por defecto** y a la hora de **importar se puede cambiar ese valor**:

- En la definición: **\$color:#fff !default;**

- A la hora de importar:

```
@use 'variables' with (  
  $color:green  
);
```

// Se pueden cambiar varias variables en el mismo **with** (separar por comas).

```
body {  
  background-color: variables.$color;  
}
```

Scope de las variables

- Tenemos un scope global y local a una regla.
- Si se definen las variables fuera de las reglas tienen un alcance global y se pueden referenciar en cualquier regla.
- Si se define una variable dentro de una regla, esa variable tendrá un scope local y sólo se podrá referenciar dentro de la propia regla.

```
$color: red; // scope global
```

```
$body {  
    $border: 1px solid black; // Scope local.  
    background: $color; // ok  
    border: $border;  
}
```

```
h1 {  
    color: $color; // ok  
    border: $border; // ERROR!!  
}
```


Variables - anidamientos

- Dentro de Sass se puede utilizar el anidamiento en las reglas y las variables locales siguen siendo accesibles:

```
body {  
  $font: 2rem;  
  
  h1 {  
    font-size: $font;  
  }  
}
```

Modificar Variables

- Las variables globales se pueden modificar en un ámbito local:
- Pero hay que indicar que son variables globales.

\$global: Arial;

```
body {  
    $global:cursive !global;  
    font-family: $global;  
}
```

```
h2 {  
    font-family: $global;  
}
```

- ***En la CSS generada, el tipo será cursive. Si colocamos la regla h2 por encima de body, la regla h2 tendría font-family: Arial.***

Operaciones en las variables

`$size:16*4px; // Se pueden añadir las medidas`

```
.title {  
    font-size:$size;  
}
```

- `// Error: $size: 2px * 3px;`
- `// ok: $size: 2px * 3;`
- `// ok: $size: 16/38;`

Operaciones en las variables

- `$size:38/16; // 16 el valor de 1em`
- `.title {`
 - `font-size: $size * 1em;`
- `}`

- Genera
 - `.title {`
 - `font-size: 2.375em;`
 - `}`

- Con JavaScript NO se puede acceder a las variables de Sass.

Variables Sass vs Custom Property

- Se puede hacer interpolación entre las variables de Sass y las custom property.

```
$primary: red;
:root {
  --primary: #{$primary};
}
```

- Se genera:

```
:root {
  --primary: red;
}
```

También podemos hacer:
@use 'variables';

```
.title {
  // Referenciamos la custom property
  color: var(--primary)
}
```

Interpolación y Nesting

- La interpolación en Sass nos permite incrustar expresiones de Sass dentro de nuestra hoja de estilos.
 - Estas expresiones se pueden incluir con: **`#{ expresión Sass }`**
 - `#{ $variable }`
- Normalmente las variables se utilizan para almacenar variables que luego se asignan a propiedades de las CSS.
- Se puede guardar un selector dentro de una variable.

Ejemplo con un selector

- **\$selector:** '.hero'
 - **#{\$selector}** {
 - color: red;
 - }
- **Se genera:**
 - .hero {
 - color: red;
 - }

Otro ejemplo

```
$selector: '.title';  
$color: red;  
$propiedad: 'background-color';
```

```
{ $selector } {  
    #{ $propiedad }: $color;  
}
```

- Se pueden utilizar las variables para los selectores, las propiedades y los valores.
- En el caso de las propiedades y los selectores hay que utilizar la interpolación: #{ \$variable }

- Se genera:

```
.title {  
    background-color: red;  
}
```

- La interpolación siempre convertirá todo a una cadena sin comillas.

Interpolación y Nesting

- También se puede utilizar la interpolación con las **custom-property** de CSS.

- Esto NO funciona:

```
$color: red;  
:root {  
    --color: $color;  
}
```

- Hay que hacerlo así:

```
$color: red;  
:root {  
    --color: #{ $color };  
}
```

Interpolación y Nesting

- También soporta las operaciones:

```
$rem: 16px;
```

```
$propiedad2: 'font-size';
```

```
$selector: '.title';
```

```
{ $selector }
```

```
  { $propiedad2}: $rem * 4; // En la variable rem NO es necesario interpolar.
```

```
}
```

- Se genera:

```
.title {
```

```
  font-size: 64px;
```

```
}
```

Nesting en Sass

- Existe la idea de incluir en un futuro el anidamiento en el CSS nativo.
- En CSS utilizamos: (vamos repitiendo el mismo selector)

```
.hero {  
    background-color: red;  
}  
.hero nav {  
    display: flex;  
}  
.hero nav li {  
    list-style: none;  
}  
.hero nav li a {  
    text-decoration: none;  
}
```

Ejemplo

- Se escribe de forma anidada (y nos genera el código anterior):
- Tener en cuenta que si se bajan muchos niveles en SASS luego puede tardar más en renderizarse.

```
.hero {  
  background-color: red;  
  
  nav {  
    display: flex;  
  
    li {  
      list-style: none;  
  
      a {  
        text-decoration: none;  
      }  
    }  
  }  
}
```

Nesting en SASS

- Se pueden anidar varios elementos simultáneamente, es decir, utilizando selectores de grupo:

```
.hero, .main {  
    font-size: 2rem;  
  
    .cta {  
        color: #000000;  
    }  
  
    .title {  
        font-size: 4rem;  
  
        span {  
            font-weight: bold;  
        }  
    }  
}
```

```
.hero, .main {  
    font-size: 2rem;  
}  
.hero .cta, .main .cta {  
    color: #000000;  
}  
.hero .title, .main .title {  
    font-size: 4rem;  
}  
.hero .title span, .main .title span {  
    font-weight: bold;  
}
```

Selectores en Nesting

- Dentro de SASS también se pueden utilizar los siguientes selectores de CSS:
 - Selectores de hijo directo: >
 - Selector de hermano adyacente: +
 - Selector de hermanos generales: ~ separa dos selectores y selecciona el segundo elemento sólo si está precedido por el primero y ambos comparten un padre común.

Selectores de hijo >

- Sintaxis: selector1 > selector2 { style properties }

- Ejemplo:

```
span { background-color: white; }  
div > span {  
    background-color: DodgerBlue;  
}
```

Span #1, dentro del div. Span #2, dentro del span que está en el div.
Span #3, no está dentro del div.

Selector de hermanos adyacentes +

- Se hace referencia a este selector como selector adyacente o selector del próximo hermano. Sólo seleccionará un elemento especificado que esté inmediatamente después de otro elemento especificado.
- Sintaxis: elemento_anterior + elemento_afectado { estilos }
- Ejemplo: `img + span.caption { font-style: italic; }`

```
<span class="caption">The first photo</span>  
<span class="caption">The second photo</span>
```


Selector de hermanos generales ~

- Sintaxis: elemento ~ elemento { estilos }
- Ejemplo: p ~ span { color: red; }

```
<span>Este span no es rojo.</span>  
<p>Aquí hay un párrafo.</p>  
<code>Aquí hay algo de código.</code>  
<span>Aquí hay un span. Es rojo porque va precedido de un párrafo y ambos comparten el mismo padre.</span>
```

Este span no es rojo.

Aquí hay un párrafo.

Aquí hay algo de código. Aquí hay un span. Es rojo porque va precedido de un párrafo y ambos comparten el mismo padre.

Selectores en Nesting

// Selector de hermanos generales Dentro de menú
tenemos 3 elementos y se lo queremos aplicar solo a 2

```
.menu {  
  ~{ // Aquí también se puede aplicar: + o >  
    .close {  
      display: none;  
    }  
    .open {  
      display: block;  
    }  
  }  
  a {  
    text-decoration: none;  
  }  
}
```

- **Se genera:**

```
.menu ~ .close {  
  display: none;  
}  
.menu ~ .open {  
  display: block;  
}  
.menu a {  
  text-decoration: none;  
}
```

Media queries

- Dentro del elemento anidado también se puede especificar la etiqueta **@media** y siempre se aplicará a su elemento padre.

- Por ejemplo:

```
.menu {  
    @media (min-width:500px){  
        background-color: blue;  
    }  
}
```

Ejemplo

```
.menu2 {  
  
  a {  
    text-decoration: none;  
  }  
  
  @media (min-width:500px){  
    background-color: #fff;  
    display: inline;  
    text-decoration: none;  
  }  
}
```

- Se genera (siempre se añade al elemento que está declarado como padre).

```
.menu a {  
  text-decoration:none;  
}  
  
@media (min-width: 500px) {  
  .menu2 {  
    background-color: #fff;  
    display: inline;  
    text-decoration: none;  
  }  
}
```

Ejemplo 2

// Si lo definimos dentro del elemento a:

```
.menu2 {  
  a {  
    @media (min-width:500px){  
      background-color: #fff;  
      display: inline;  
      text-decoration: none;  
    }  
  }  
}
```

- Se genera para .menu2 a

```
@media (min-width: 500px) {  
  .menu2 a {  
    background-color: #fff;  
    display: inline;  
    text-decoration: none;  
  }  
}
```

@keyframes

- Dentro de un elemento anidado se pueden crear keyframes:

```
@keyframes show {  
  100% {  
    transform.: unset;  
  }  
}
```

- Se genera:

```
@keyframes show {  
  100% {  
    transform.: unset;  
  }  
}
```

Selector de padres

- En CSS disponemos del selector de padres **:has()**
- Dentro del argumento se puede indicar: >, +, ~
- **a:has(> img){}**
 - El selector se aplica a los enlaces que tengan una imagen como hijo directo del enlace.
- **h2:has(+ h3){}**
 - Todos los h2 que tengan un h3 inmediatamente después de ellos.
- **ul:has(:not(li + li)){}**
 - Selecciona todas las listas de un solo item de lista. Las dos pseudoclases es intercambiable: `ul:not(:has(li+li))`

Valores en Sass

- Crear un nuevo partial (con el guión bajo).
- Se puede utilizar este fichero con use en otro fichero principal.
- Valores:
 - Números enteros: `$variable:2;`
 - Números con unidades: `$pixeles: 3px;`
- `@debug` expresión. Puedo mandar expresiones a la consola para su depuración:
 - `@debug 100 * 100;` → Se envía a la consola.

Valores en Sass

- También se puede mandar texto con @debug
 - @debug 'El valor es igual a' 100 * 100;
- Se puede interpolar:
 - @debug 'El valor es #{100 * 100}';
- Se utiliza para imprimir por la consola en Sass.
- Puedo también operar con los números con unidades.
 - @debug 100 * 100px; → 10000px
 - @debug 100px * 100px; → **ERROR!**

Valores en Sass

- Tampoco se pueden multiplicar pixels por em.

```
body {  
    font-size: 2px * 3em;  
}
```

- Multiplicar no, pero sumar sg y msg si podemos:

```
body {  
    animation-duration: 10s + 100ms;  
}
```

- Sumar pixeles también funciona: width: 30px + 40px; **OK!**
- Restas también!
- Se puede operar con cm y px: width: 1cm + 3px; **ok!**

Valores en Sass

- Las divisiones utilizando variables también es posible:
- `$variable: 10px / 2; → ok!`
- `$variable2: 10px / 2px; → ok!`

```
body {  
    width: $variable; → ok  
}
```

Valores en Sass

```
@use 'sass:math';  
body {  
  font-size: math.div(25,5); → font-size:5  
}
```

- Texto en Sass (2 tipos):
 - Quoted string " o "" simples en dobles
 - Unquoted string sin comillas

Valores en Sass

```
$selector: "h1";
```

```
#{ $selector } {  
    color: #000;  
}
```

// Se quitan las comillas:

```
h1 {  
    color: #000;  
}
```

Ejemplos

```
$strings-comillas: "header header header";
```

```
$string-sincomillas: center;
```

```
.title {  
    text-align: $string-sincomillas;  
}
```

```
.grid {  
    grid-template-areas: $strings-comillas;  
}
```

- **Se genera;**

```
.title {  
    text-align: center;  
}
```

```
.grid {  
    Grid-template-areas: "header header header";  
}
```

strings

- Disponemos del módulo de sass : string
- @use 'sass:string';
- Para poner comillas:
 - propiedadCSS: string.quote(\$variable);
- Para quitar comillas:
 - propiedadCSS: string.unquote(\$variable);

sass:meta

- Para saber el tipo que tiene una variable disponemos del módulo meta.
- En el mismo debug se puede imprimir:
 - `@debug meta.type-of($string-comillas);`
 - `@debug meta.type-of($numero);`
- Detecta tipos como number, color, list, string, bool

Listas en Sass

- Cuando tenemos varios valores separados por espacios o por comas:
- `$boxshadow: 1px 1px 20px;`
- `@debug meta.type-of($boxshadow);` → **list**
- O también entre corchetes.
 - `$boxshadow: [1px 1px 20px];`
- Empiezan en uno:
 - `$margen: 10px 20px 40px 30px;`

Listas en Sass

```
@use 'sass:list';  
$margen: 10px 20px 40px 30px;  
.grid {  
    margin-top: list.nth($margen, 1);  
}
```

Se genera:

```
.grid {  
    margin-top: 10px;  
}
```

Mapas en Sass

- Se definen pares de valores: key / value
- Las claves entre comillas.
- No se pueden repetir las claves, pero si los valores.

```
$colores: (  
  "primary": blue,  
  "secondary": green;  
  "buttons": teal;  
);
```

Mapas en Sass

- `@use 'sass:map';`

```
$colores: (  
  "primary": blue,  
  "secondary": green;  
  "buttons": teal;  
);
```

```
h1 {  
  color: map.get($colores, "primary");  
}
```

- Se puede definir mapas de colores, de fuentes y se colocan en un `_partial` general.

Boolean en Sass

- \$condicional: true;
- Cualquier valor que no sea ni **false** ni **null** se considera verdadero.

```
@if ($condicional){  
  .grid{  
    display: grid;  
  }  
}  
  
@else {  
  .flexbox {  
    display: flex;  
  }  
}
```

Genera: .grid {display: grid; }

Boolean

- Disponemos de todos los operadores relaciones: ==, !=, <, <=, >, >=
- La misma cadena: @debug 'hola' == hola
- El mismo color: @debug #000 == rgb(0,0,0);

Estructuras de Control: if

```
$display: grid;  
.hero {  
  @if $display == grid {  
    display: grid;  
    grid-template-columns: 1fr 1fr;  
  }  
  @else {  
    display: $display;  
  }  
}
```

- Asignar valores a propiedades en función de condiciones.
- El @else no es obligatorio.

Estructuras de Control: **for**

```
@for $i from 1 to 10 {  
    .selector-#{ $i } {  
        color: blue;  
    }  
}
```

-

- El 10 no lo incluye.

- Para que **incluya el 10** → @for \$i from 1 **through** 10 {...}
- Para que sea **descendente**: @for \$i from 10 **through** 1 {...}

- Genera reglas de forma dinámica de:

- .selector-1 { color:blue; } ... aselector-10 { color:blue; }

Estructuras de Control: **each**

- Sirve para recorrer las **listas** y los **mapas** de Sass.

```
$lista: 1 2 3 4 5;
```

```
@each $i in $lista {  
  .elemento-#{ $i } {  
    color: #000;  
  }  
}
```

Estructuras de Control: **each**

```
$margenes: 1 2 3 4 5 6;
```

```
@each $margen in $margenes {  
  .m-1#{ $margen } {  
    margin: $margen*1em;  
  }  
}
```

- Crea selectores de .m-1 a .m-6




Estructuras de Control: each

- **Each** para mapas: 2 variables.

```
$mapa: (  
    "primary": crimson;  
    "secondary": blue;  
    "shadow": 1px 1px solid #000;  
);  
  
@each $key, $value in $mapa {  
    @debug $key $value;  
}
```

Estructuras de Control: each

```
@each $clave, $valor in $mapa {  
  .#{$clave}{  
    @if ($clave != 'shadow'){  
      color: $valor;  
    } @else {  
      box-shadow: $valor;  
    }  
  }  
}
```

```
.primary {  
  color:  crimson;  
}  
  
.secondary {  
  color:  steelblue;  
}  
  
.shadow {  
  box-shadow: 1px 1px solid  #000;  
}
```

Estructuras de Control: **each**

- Se pueden recorrer listas anidadas:

\$lista:

```
“card” color 3s,  
“nav” clip-path 2s,  
“footer” margin 1s;
```

- Expandimos los 3 elementos de cada lista en 3 variables en el bucle **each**:

```
@each $selector, $propiedad, $duración in $lista {  
  .#{$selector}{  
    transition-property: $propiedad;  
    transition-duration: $duracion;  
  }  
}
```

Mixins

- Los mixins definen estilos que pueden ser utilizados a lo largo del código scss.
- Primero se definen con **@mixin** y luego se incluyen con **@include**:

```
@mixin center {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```

```
header {  
    @include center;  
}
```

Mixins

- No hay diferencia en los mixins (en general en Sass) de utilizar un **guión medio** en la declaración y al utilizarlo poner un **guión bajo**:

- // Funciona igualmente:

```
@mixin center-flexbox { ... }
```

```
header {  
    @include center-flexbox;  
}
```

- Los mixins pueden tener **argumentos** (van entre **paréntesis**)
- A los mixins NO se les puede llamar antes de que estén declarados.

Mixins con parámetros

```
@mixin crear-grid($rows, $columns, $gap){  
  display: grid;  
  grid-template-rows: repeat($rows, 1fr);  
  grid-template-columns: repeat($columns, 1fr);  
  gap: $gap;  
}  
$filas: 5;  
$columnas: 5;  
$gutters: 30px;  
  
.header {  
  @include crear-grid($filas, $columnas, $gutters);  
}
```


Mixins con valores opcionales

- Selector es obligatorio y los otros dos opcionales.
- Los argumentos obligatorios mejor al principio.
- Se pueden utilizar los nombres de los parámetros en la llamada (argumentos nominales)

```
@mixin pruebas($selector, $param:1, $param2:1){  
    ...  
}
```

```
.header {  
    /* Podemos saltar $param y rellenar $param2 */  
    @include pruebas(img, $param2:20);  
}
```

Mixins con short-hands

- Dentro del mixin se pueden utilizar las subpropiedades:

```
@mixin addImage($img, $repeat, $size) {  
    Position: fixed;  
    top:0;  
    left:0;  
    background: {  
        image: url($img);  
        repeat: $repeat;  
        size: $size;  
    }  
}
```

```
header {  
    @include addImage('uno.png', no-repeat, cover);  
}
```

Mixins: lista de argumentos

```
@mixin addTransition($propiedad, $valor, $selecctores ...){  
    @debug $selecctores;  
}
```

```
header {  
    @include addTransition(opacity, 0, h1, h2, h3, a);  
}
```

Funciones

- Sass permite la creación de funciones para luego reutilizarlas:
- `@function nombreFun(){ ... }`

```
@use 'sass:math';
```

```
@function pixelesToEm(@pixeles){  
  // Si el usuario envía 16px, quitamos px dividiendo por la unidad.  
  // 1em → 16px  
  
  $unidad: math.div($pixeles, 1px);  
  $resul: math.div($unidad, 16);  
  @return $resul * 1em; // Para añadir la unidad al valor  
}
```

- La llamada:
 `@debug pixelesToEm(16px);`

 `.title {
 font-size: pixelesToEm(16px);
 }`

Funciones

- Desde otro módulo podemos utilizar la función creada:

`@use 'funciones';`

```
.hero {  
    font-size: funciones.pixelesToEm(100px);  
}
```

- Los módulos integrados de Sass se referencian con:
 - `@use 'sass:nombre_modulo'`

Funciones

- A los argumentos de las funciones se pueden utilizar los nombres de los argumentos dentro de la llamada a la función.
- Si tenemos parámetros obligatorios y opcionales, mejor poner primero los obligatorios y luego los opcionales.

Funciones

- Se pueden pasar listas a los argumentos:

```
@function sumar($args...){  
    // Con tres puntos suspensivos.  
    $suma: 0;  
  
    @each $i in $args {  
        @suma: @suma + $i;  
    }  
    @return $suma;  
}
```

```
.suma {  
    font-size: sumarElementos(40, 50, 4, 90);  
}
```

Módulos en Sass

- Módulos integrados:
 - <https://sass-lang.com/documentation/modules>
- @use se utiliza para importar desde otro archivo a los módulos integrados de Sass o a los módulos que hemos desarrollado.
- Sass permite trabajar de forma modular.
- Los módulos que desarrollamos se llaman **partials** y tienen un guión de subrayado como primer carácter.
- Los archivos / módulos que empiezan por _ no se compila hasta que no se utilizan desde otro archivo.

Módulos en Sass

- `_modulo.scss`
 - `// Contenido de mi módulo`
- Desde `styles.scss`
- `@use 'modulo';` → No se indica el `_` y tampoco la extensión.
- Si se encuentra en otra carpeta indicarlo al importar:
- `@use 'carpeta/modulo';` → namespace

Diferencia con Less

- **_colores.scss**
 - \$primary: red;
- **_modulo.scss**
 - @use 'colores';
 - .hero {
 - color: colores.\$primary; // Hay que poner el nombre del módulo.
 - }
- **index.scss**
 - @use 'modulo';

Enlaces

- <https://openwebinars.net/blog/que-es-sass-ventajas-desventajas-y-ejemplos-de-desarrollo/>