

# **PREPROCESADOR CSS**

# índice de Contenidos

**01** INSTALACION DE SASS.  
SINTAXIS DE SASS. VARIABLES  
CON SASS

**02** VARIABLES CON SASS. MIXINS

**03** EXTENDS. MEDIA QUERIES CON  
SASS.

**04** DIRECTIVAS.

# índice de Contenidos

**01** INSTALACION DE SASS.  
SINTAXIS DE SASS. VARIABLES  
CON SASS

**02** VARIABLES CON SASS. MIXINS

**03** EXTENDS. MEDIA QUERIES CON  
SASS.

**04** DIRECTIVAS.

# ¿QUÉ ES UN PREPROCESADOR DE CSS?

Un **preprocesador CSS** es una herramienta software que mejora la funcionalidad del lenguaje CSS. Los preprocesadores CSS añaden características y abstracciones que no están disponibles en CSS puro, lo que facilita la creación y el mantenimiento de hojas de estilo más complejas.

Algunos ejemplos destacados de preprocesadores CSS son **Sass (Syntactically Awesome Style Sheets)**, **Less (Leaner Style Sheets)** y **Stylus**. Estos preprocesadores se utilizan en el desarrollo web para crear hojas de estilo más mantenibles, eficientes y organizadas. Sin embargo, los navegadores web no pueden interpretar directamente código Sass, Less o Stylus, por lo que estos **archivos deben compilarse en CSS estándar** antes de implementarse en un sitio web.

# ¿QUÉ ES UN PREPROCESADOR DE CSS?

La compilación se realiza generalmente mediante **herramientas de línea de comandos, extensiones de editores de código o con herramientas de automatización de tareas**.

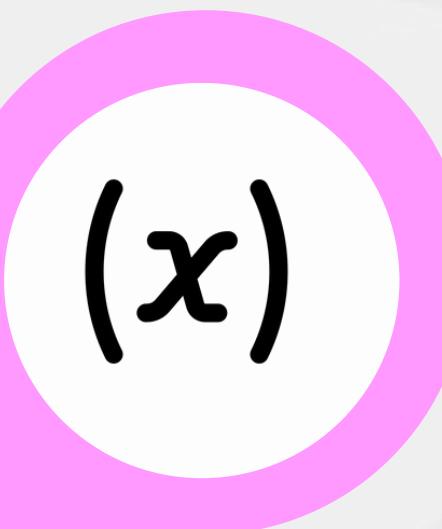
En nuestro caso, utilizaremos una extensión que posee Virtual Studio Code (**Live Sass Compiler**)

# VENTAJAS

**01. Mejora la eficiencia  
de la escritura de  
código CSS**



**02. Reutilización de  
código**



**03. Variables**



**04. Anidación de  
selectores**

# VENTAJAS

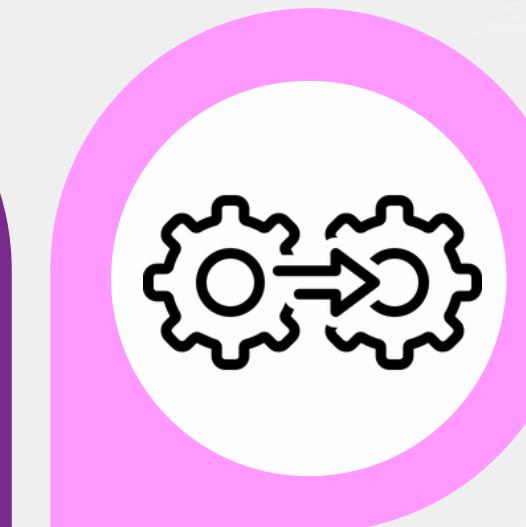
**05. Modularidad**



**06. Facilita el mantenimiento**



**07. Compatibilidad con CSS existente**



**08. Comunidad y herramientas**



# ¿CÓMO TRABAJAMOS CON SASS?

1. Crear un archivo con extensión **.sass** o **.scss**. Este archivo se puede crear instalando la extensión **Sass** en editor de código Virtual Studio Code..
2. Compilar el archivo **.scss**. Existen distintas opciones:
  - Instalar un programa de compilación.
  - Instalar en el editor de código Virtual Studio Code la extensión **Live Sass Compiler**. Ésta nos permite crear un archivo de estilo (.css).
  - Añadir el archivo creado al compilar en el archivo **index.html**

# ELEMENTOS BÁSICOS

Un ejemplo de código .scss puede ser el siguiente:

## código.scss

```
$main: #444;  
.btn{  
    color: $main;  
    display: block;  
}  
.btn-a{  
    color: lighten($main, 30%);  
    &:hover{  
        color: lighten($main, 40%);  
    }  
}
```

## código.css

```
.btn{  
    color: #444444;  
    display: block;  
}  
.btn-a{  
    color: #919191;  
}  
.btn-a:hover{  
    color: #aaaaaa;  
}
```

# ELEMENTOS BÁSICOS

## Comentarios

### Comentarios en scss

Podemos utilizar para comentar: `//` o `/*... */`

### Diferencia:

`// Este comentario`  
`// dejará de aparecer`  
`// al compilarlo`

*/\* Este comentario si aparece al compilar\*/*

### Comentarios en css

Los comentarios en CSS `/*... */`

*/\* Este comentario si aparece al compilar\*/*

# ELEMENTOS BÁSICOS

## ETIQUETA @import

Podemos importar archivos con extensión .scss o .sass y la importación se lleva a cabo durante la compilación en lugar de en el lado del cliente. Al utilizar **@import** en Sass no tenemos que incluir la extensión del archivo.

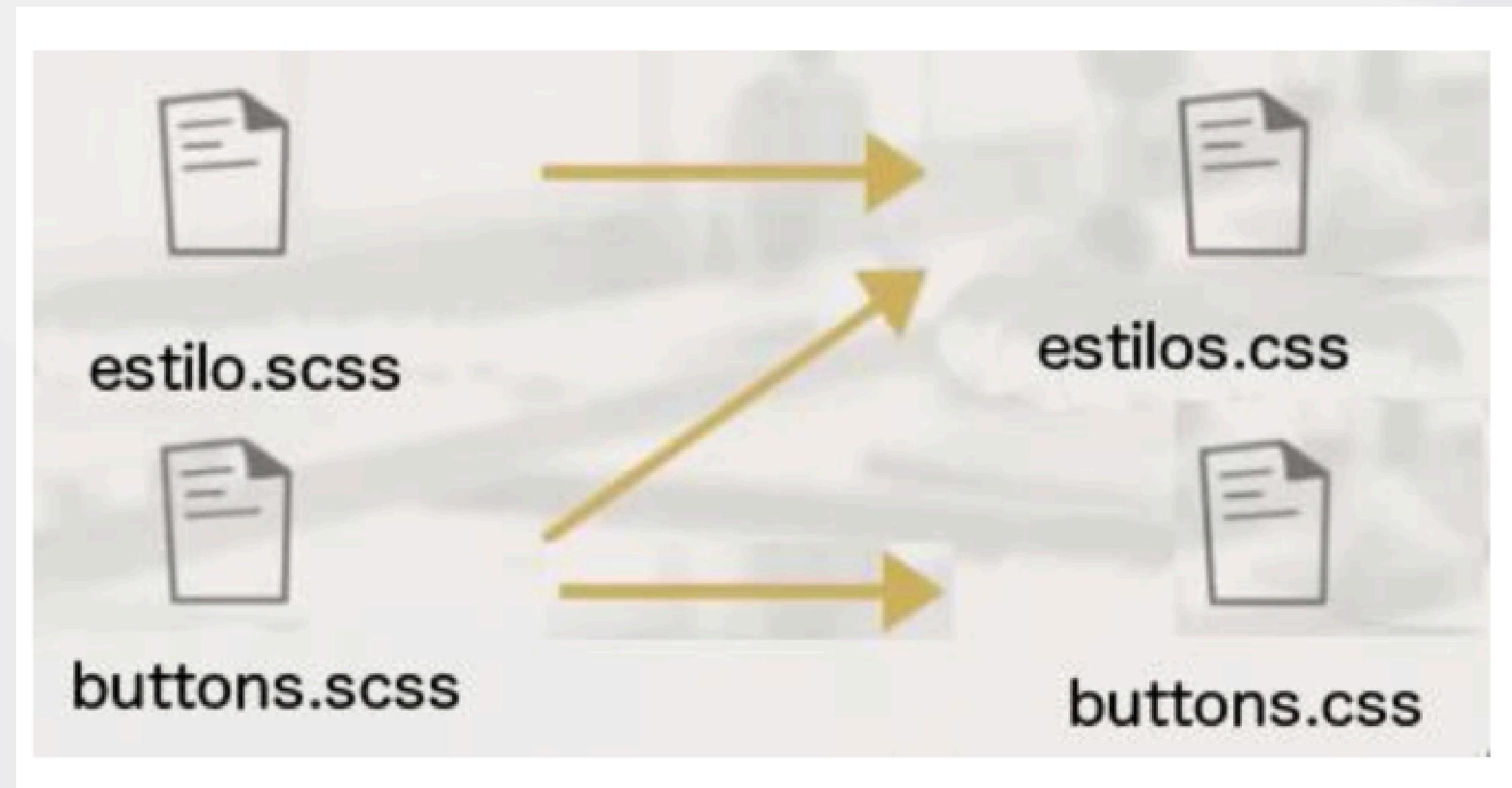
Durante el proceso de compilación donde estilos.css pasa a tener los estilos de los archivos: **estilos.scss** y **buttons.scss**

### Archivo: estilo.scss

```
// Importa los estilos encontrados en buttons.scss  
// Cuando el compilador compila estilo.scss  
@import "buttons";
```

# ELEMENTOS BÁSICOS

## ETIQUETA @import



# ELEMENTOS BÁSICOS

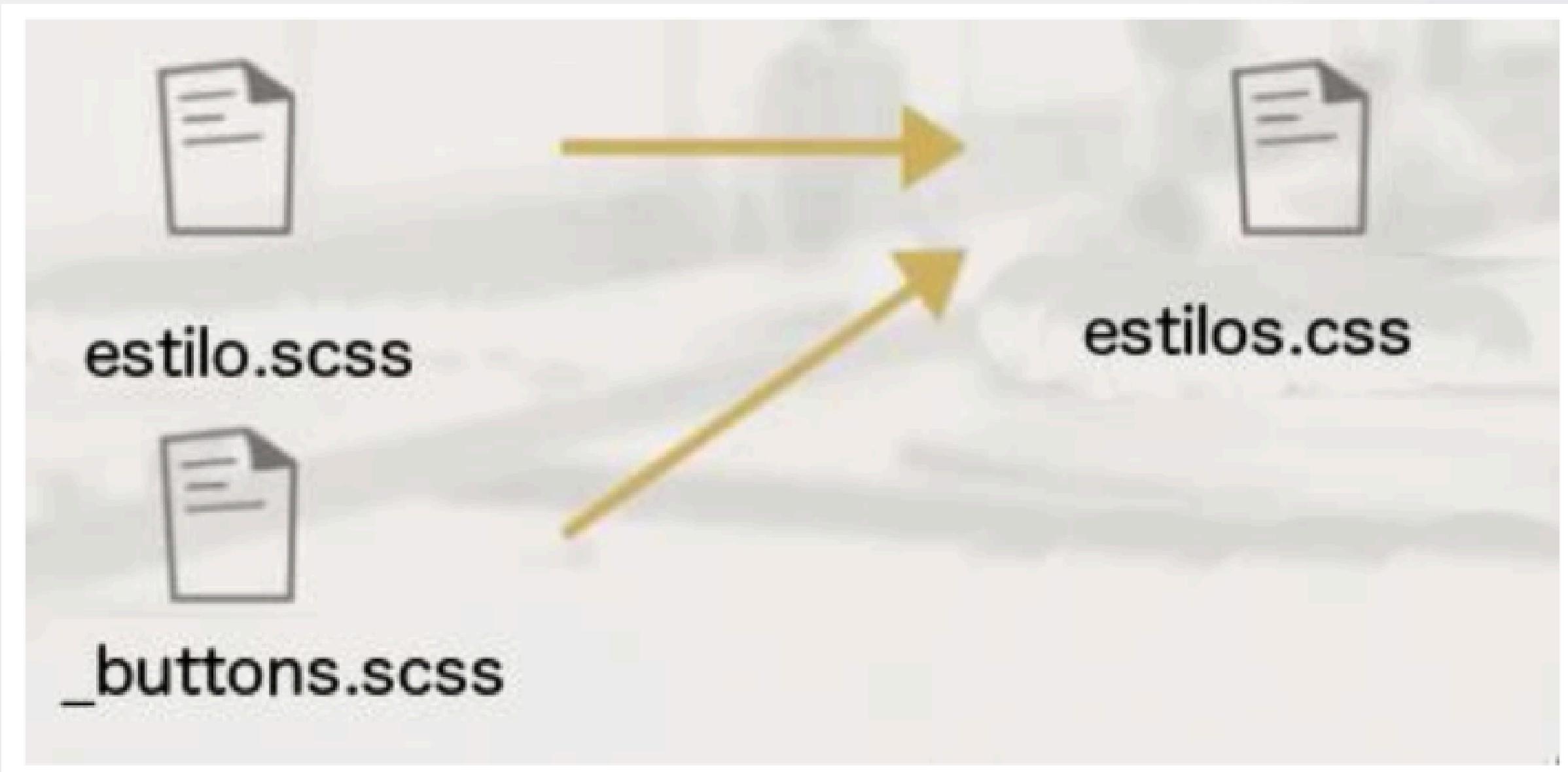
## ETIQUETA @import

Además de incluir la hoja de estilos *buttons*, cuando compilamos y nos genera el archivo *estilos.css*, también nos genera el CSS de *buttons*. Para que no nos cree el archivo de *buttons.css* deberemos nombrar el archivo *buttons.scss* con el subrayado delante, es decir:

- **Si quiero** que me cree el archivo ***buttons.css***, lo llamaré: ***buttons.scss***
- **Si no quiero** que me cree el archivo ***buttons.css***, lo llamaré: ***buttons.scss***

# ELEMENTOS BÁSICOS

## ETIQUETA @import



# ELEMENTOS BÁSICOS

## ETIQUETA @import

Para especificar que se va a importar un archivo “parcial”, es decir, el \_buttons.scss, se realiza de la misma forma:

***@import "buttons"***

# ELEMENTOS BÁSICOS

## Variables

Las variables de Sass permiten **almacenar valores reutilizables** en las hojas de estilo. Se definen con el **símbolo \$ seguido del nombre de la variable** y luego se le asigna un valor. Estas variables pueden contener valores como colores, números, cadenas de texto, etc.

**\$nombre: valor;**

# ELEMENTOS BÁSICOS

## Variables

### archivo.scss

```
$base: #777;  
.sidebar{  
    border: 1px solid $base;  
    p{  
        color: $base;  
    }  
}
```

### archivo.css

```
$base: #777;  
.sidebar{  
    border: 1px solid #777;  
}  
.sidebar p{  
    color: #777;  
}
```

# ELEMENTOS BÁSICOS

## Variables

// Definición de variables

```
$color-primario: #3498db;  
$color-secundario: #e74c3c;  
$tamano-fuente: 16px;  
$espaciado: 20px;
```

// Uso de variables

```
.encabezado {  
    background-color: $color-primario;  
    font-size: $tamano-fuente;  
    padding: $espaciado;  
}  
  
.boton {  
    background-color: $color-secundario;  
    font-size: $tamano-fuente;  
    margin: $espaciado;  
}
```

# ELEMENTOS BÁSICOS

## Variables

Si definimos una variable, ésta puede ser utilizada posteriormente modificando el valor que le habíamos dado antes, de forma que el valor anterior es sobreescrito.

**aplicacion.scss**

```
$titulo: 'Mi blog;  
$titulo: 'Sobre mí';
```

```
h2:before {  
    content:$titulo;  
}
```

**aplicacion.css**

```
$titulo: 'Mi blog;  
$titulo: 'Sobre mí';
```

```
h2:before {  
    content: 'Sobre mí';  
}
```

# ELEMENTOS BÁSICOS

## Variables

Si definimos la segunda variable como: `$titulo: 'Sobre mi'; !default;` lo que hace es comprobar si la variable `$titulo` tiene un valor previo, y en el caso de no tenerlo le otorga el valor 'Sobre mi'

### archivo.scss

```
$titulo: 'Mi blog;  
$titulo: 'Sobre mi'; !default;
```

```
h2:before {  
    content:$titulo;  
}
```

### archivo.css

```
$titulo: 'Mi blog;  
$titulo: 'Sobre mi'; !default;
```

```
h2:before {  
    content: 'Mi blog';  
}
```

# ELEMENTOS BÁSICOS

## Tipos de variables

- **Booleanos:** Establecemos valores booleanos

```
$rounded: false;  
$shadow: true;
```

- **Números** (los podemos establecer con o sin unidades)

```
$rounded: 4px;  
$line-height: 1.5;  
$font-size: 3em;
```

- **Colores**

```
$base: purple;  
$border: rgba(0,255,0,0.5);
```

# ELEMENTOS BÁSICOS

## Tipos de variables

**Strings** (con o sin comillas)

```
$header: 'Helbetia Neue';
$callout: Arial;
$message: "Loading...";
```

- **Listas**

```
$authors: Rosa, Pablo, Jose, Lucia;
$margin: 40px 0 20px 100px;
```

- **Null**

```
$shadow: null;
$border: rgba(0,255,0,0.5);
```

# ELEMENTOS BÁSICOS

## Comentarios

Sass admite dos tipos de comentarios:

- **Comentarios en línea:** Estos comentarios son similares a los comentarios de un línea en CSS y comienza con `//`.  
Este tipo de comentario no se incluye en el fichero a compilar.
- **Comentarios de varias líneas:** Son similares a los comentarios de bloque en CSS y se encierran entre `/*` y `*/`.  
Este tipo de comentario se incluyen en el fichero CSS al compilar.

# ELEMENTOS BÁSICOS

## Anidamiento (nesting)

El anidamiento es una característica de Sass que te permite **anidar selectores dentro de otros**, lo que refleja la estructura de HTML y hace que tu código sea más legible y organizado. Cuando anidas selectores, los estilos aplicados al selector padre también se aplican a los elementos anidados.

# ELEMENTOS BÁSICOS

## Anidamiento

### archivo.html

```
<div class="contenedor">
  <h1>Título</h1>
  <p>Párrafo de ejemplo</p>
</div>
```

### archivo.scss

```
.contenedor{
  background-color: #f0f0f0;
  padding: 20px;
  h1{
    font-size: 24px;
    color:#333;
  }
  p {
    font-size: 16px;
    color:#666;
  }
}
```

### archivo.css

```
.contenedor{
  background-color: #f0f0f0;
  padding: 20px;
}
.contenedor h1{
  font-size: 24px;
  color:#333;
}
.contenedor p {
  font-size: 16px;
  color:#666;
}
```

# ELEMENTOS BÁSICOS

## Anidamiento (nesting)

Otra posibilidad de anidamiento es:

**archivo button.scss**

```
.btn{  
  text: {  
    decoration: underline;  
    transform: lowercase;  
  }  
}
```

**archivo button.css**

```
.btn{  
  text-decoration: underline;  
  text-transform: lowercase;  
}
```

# ELEMENTOS BÁSICOS

## Anidamiento / Uso del símbolo &

El símbolo «&» se usa para **hacer referencia al selector padre dentro de una regla anidada**. Esto es útil cuando quieres aplicar estilos específicos a elementos que son descendientes directos del selector padre.

### archivo.scss

```
.button{  
background-color: blue;  
&:hover{  
background-color:red;  
}  
}
```

El «&» en &:hover hace referencia al selector .button, por lo que cuando el cursor se coloca sobre un elemento con la clase .button, se aplicará el estilo background-color: red; solo a ese elemento en particular.

# índice de Contenidos

**01** INSTALACION DE SASS.  
SINTAXIS DE SASS. VARIABLES  
CON SASS

**02** VARIABLES CON SASS. MIXINS

**03** EXTENDS. MEDIA QUERIES CON  
SASS.

**04** DIRECTIVAS.

# VARIABLES CON SASS

## Ámbito

- Las variables se deben usar en su mismo ámbito

### application.scss

```
p{  
  $border: #ccc;  
  border-top: 1px solid $border;  
}  
  
h1{  
  border-top: 1px solid $border;  
}
```

Al compilar se produce un error porque las variables que se declaran dentro de las llaves/ámbitos {} no pueden ser utilizadas fuera de ese bloque/scope. Sin embargo, **si definimos una variable fuera de una declaración, esa variable cambia para las futuras instancias ya que se crea como global.**

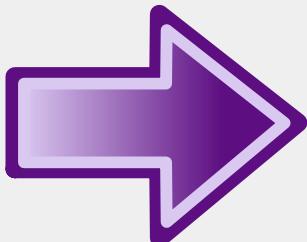
# VARIABLES CON SASS

## Ámbito

Si definimos una variable como si fuera una variable global en nuestro archivo, y posteriormente redefinimos la variable (mismo nombre) para utilizarla en uno de los scopes internos, lo que estamos haciendo es sobrescribiendo el valor que tenía la variable en ese ámbito, y fuera del mismo tendrá el valor que se le había dado antes.

**application.scss**

```
$color-base: #777;  
.sidebar{  
  $color-base: #222;  
  background: $color-base;  
}
```



**application.css**

```
.sidebar{  
  background: #222;  
}
```

La variable toma el valor de la variable definida en el ámbito

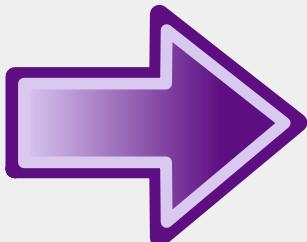
# VARIABLES CON SASS

## 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 sobreescrito por el valor de sidebar

### application.scss

```
$color-base: #777;  
.sidebar{  
  $color-base: #222;  
  background: $color-base;  
}  
  
p{  
  color: $color-base;  
}
```



### application.css

```
.sidebar{  
  background: #222;  
}  
  
p{  
  color: #777;  
}
```

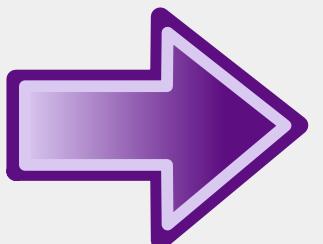
# VARIABLES CON SASS

## Interpolación

Se utiliza la almohadilla seguida de una apertura y cierre de llaves **#{\$variable}** para añadir nuestras variables a selectores, nombres, propiedades o string.

### application.scss

```
$side: top;  
sup{  
  position: relative;  
  #{$side}: -0.5em;  
}
```



### application.css

```
sup{  
  position: relative;  
  top: -0.5em;  
}
```

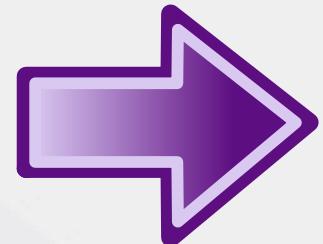
# VARIABLES CON SASS

## Interpolación

Si la interpolación la añadimos como un selector,

### application.scss

```
$side: top;  
sup{  
    position: relative;  
    #{$side}: -0.5em;  
}  
.callout-#${$side}{  
    background: #777;  
}
```



### application.css

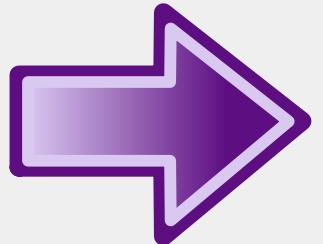
```
sup{  
    position: relative;  
    top: -0.5em;  
}  
.callout-top{  
    background: #777;  
}
```

Es bastante útil considerar el nombre de nuestras variables con un nombre más genérico, de tal forma, que se puedan utilizar en distintos lugares.

# MIXIN

## application.css

```
.btn-a{  
    background: #777;  
    border: 1px solid #ccc;  
    font-size: 1em;  
    text-transform:  
    uppercase;  
}  
  
.btn-b{  
    background: #ff0;  
    border: 1px solid #ccc;  
    font-size: 1em;  
    text-transform:  
    uppercase;  
}
```



## application.scss

```
@mixin button{  
    border: 1px solid #ccc;  
    font-size: 1em;  
    text-transform: uppercase;  
}
```

El **mixin** se utiliza para unir propiedades que se repiten. Se definen como:

**@mixin nombre {}.**

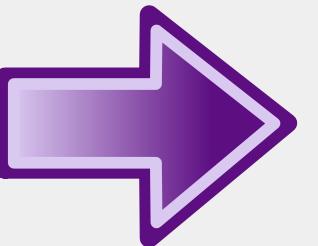
Para usarlo realizaremos la llamada:

**@include nombre**

# MIXIN

## application.scss

```
@mixin button{  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
.btn-a{  
  @include button;  
  background: #777;  
}  
.btn-b{  
  @include button;  
  background: #ff0;  
}
```



## application.css

```
.btn-a{  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
  background: #777;  
}  
.btn-b{  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
  background: #ff0;  
}
```

# 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.

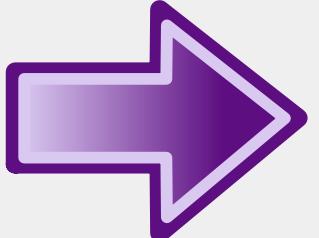
Además, debemos tener en cuenta que, usamos @include cuando añadimos un mixin mientras que si usamos @import lo que hacemos es importar el contenido de un archivo en nuestro archivo de sass.

# MIXIN

Otra forma de utilizar el mixin es en el caso de tener un CSS en el que dependiendo del navegador que estemos utilizando la propiedad del box-sizing varia.

## application.css

```
.content{  
-webkit-box-sizing: border-box;  
-moz-box-sizing: border-box;  
box-sizing: border-box;  
border: 1px solid #ccc;  
padding: 20px;  
}
```



## application.scss

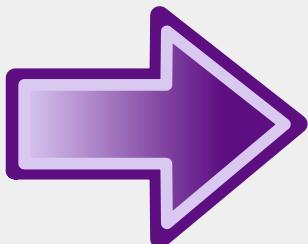
```
@mixin box-sizing {  
-webkit-box-sizing: border-box;  
-moz-box-sizing: border-box;  
box-sizing: border-box;  
}
```

# MIXIN

Otra forma de utilizar el mixin es en el caso de tener un CSS en el que dependiendo del navegador que estemos utilizando la propiedad del box-sizing varia.

## application.css

```
.content{  
-webkit-box-sizing: border-box;  
-moz-box-sizing: border-box;  
box-sizing: border-box;  
border: 1px solid #ccc;  
padding: 20px;  
}
```



## application.scss

```
@mixin box-sizing {  
-webkit-box-sizing: border-box;  
-moz-box-sizing: border-box;  
box-sizing: border-box;  
}
```

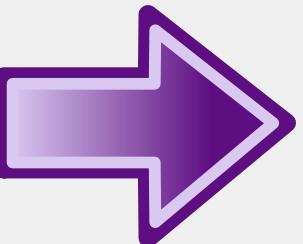
Se ha definido un mixin box-sizing en el que se han incluido los prefijos.

# MIXIN

Otra forma de utilizar el mixin es en el caso de tener un CSS en el que dependiendo del navegador que estemos utilizando la propiedad del box-sizing varia.

## application.css

```
.content{  
  -webkit-box-sizing: border-box;  
  -moz-box-sizing: border-box;  
  box-sizing: border-box;  
  border: 1px solid #ccc;  
  padding: 20px;  
}
```



## application.scss

```
@mixin box-sizing {  
  -webkit-box-sizing: border-box;  
  -moz-box-sizing: border-box;  
  box-sizing: border-box;  
}
```

Se ha definido un mixin box-sizing en el que se han incluido los prefijos.

## application.scss

```
.content {  
  @include box-sizing;  
  border: 1px solid #ccc;  
  padding: 20px;  
}
```

En el content de la nuestro bloque llo que debemos hacer es llamar al @include y esta propiedad será compilada

# MIXIN

El mixin se utiliza para pasar argumentos al mixin. Una vez le hemos definido el mixin, podemos poner entre paréntesis y pasar los argumentos que queramos, por ejemplo, en este caso le llamaremos x y lo usaremos en los valores del box-sizing

## application.scss

```
@mixin box-sizing ($x){  
  -webkit-box-sizing: $x;  
  -moz-box-sizing: $x;  
  box-sizing: $x;  
}
```

## application.scss

```
.content {  
  @include box-sizing(border-box);  
  border: 1px solid #ccc;  
  padding: 20px;  
}
```

El valor del box-sizing que estamos enviando, es un string y se le asignará a la variable x que tiene definida como parámetro de entrada nuestra definición del mixin (box-sizing).

# MIXIN

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

## application.scss

```
@mixin box-sizing ($x: border-box){  
  -webkit-box-sizing: $x;  
  -moz-box-sizing: $x;  
  box-sizing: $x;  
}  
.
```

```
.content {  
  @include box-sizing;  
  border: 1px solid #ccc;  
  padding: 20px;  
}  
.
```

```
.callout{  
  @include box-sizing (content-box);  
}
```

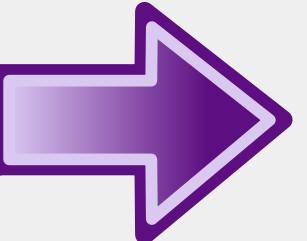
Como no se le está pasando ningún parámetro al include, se usa por defecto border-box.

Se emplea la variable que se le pasa y no la de por defecto como se usa en .content

# MIXIN

application.scss

```
@mixin box-sizing ($x: border-box){  
  -webkit-box-sizing: $x;  
  -moz-box-sizing: $x;  
  box-sizing: $x;  
}  
.content {  
  @include box-sizing;  
  border: 1px solid #ccc;  
  padding: 20px;  
}  
.callout{  
  @include box-sizing (content-box);  
}
```



application.css

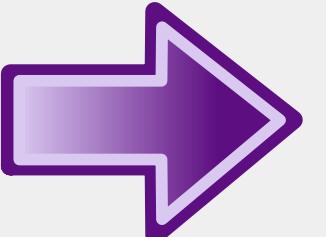
```
.content{  
  -webkit-box-sizing: border-box;  
  -moz-box-sizing: border-box;  
  box-sizing: border-box;  
  border: 1px solid #ccc;  
  padding: 20px;  
}  
.callout{  
  -webkit-box-sizing: content-box;  
  -moz-box-sizing: content -box;  
  box-sizing: content -box;  
}
```

# MIXIN

Se puede utilizar más de un argumento siempre que estén separados por comas.

## application.scss

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



## application.css

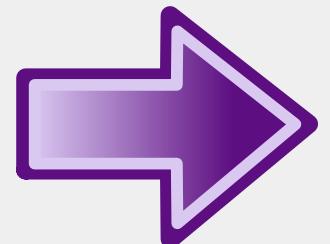
```
.btn-a {  
  border-radius: 4px;  
  color: #000;  
}
```

# MIXIN

-  Siempre hemos de enviar el número de argumentos que se ha definido en el mixin.
-  Establecer que un parámetro opcional y que en caso de no enviarlo se le asigne uno por defecto.

## application.scss

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



## application.css

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

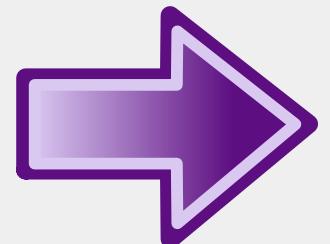
Cuando se incluyen valores por defecto y se crean argumentos opcionales en los mixin, los argumentosopcionales necesitan estar al final de la cadena de argumentos.

# MIXIN

-  Siempre hemos de enviar el número de argumentos que se ha definido en el mixin.
-  Establecer que un parámetro opcional y que en caso de no enviarlo se le asigne uno por defecto.

## application.scss

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



## application.css

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

Cuando se incluyen valores por defecto y se crean argumentos opcionales en los mixin, los argumentosopcionales necesitan estar al final de la cadena de argumentos.

# índice de Contenidos

**01** INSTALACION DE SASS.  
SINTAXIS DE SASS. VARIABLES  
CON SASS

**02** VARIABLES CON SASS. MIXINS

**03** EXTENDS. MEDIA QUERIES CON  
SASS.

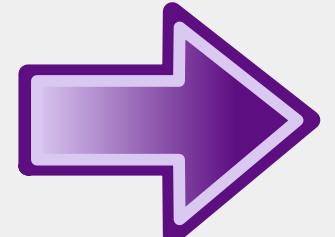
**04** DIRECTIVAS.

# EXTEND

La directiva **extend** nos permite combinar selectores de la siguiente manera:

## \_buttons.scss

```
.btn-a {  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
  
.btn-b{  
  @extend .btn-a;  
  background: #ff0;  
}
```



## application.css

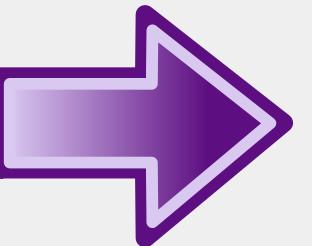
```
.btn-a, .btn-b{  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
  
.btn-b{  
  background: #ff0;  
}
```

# EXTEND

¿Qué ocurre si usamos selectores **nested** y usamos **extend**?

## application.scss

```
.content{  
  border: 1px solid #ccc;  
  padding: 20px;  
  
  h2{  
    font-size: 3em;  
    margin: 20px 0;  
  }  
}  
  
.callout{  
  @extend .content;  
  background: #ddd;  
}
```



## application.css

```
.content, .callout{  
  border: 1px solid #ccc;  
  padding: 20px;  
}  
  
.content h2, .callout h2{  
  font-size: 3em;  
  margin: 20px 0;  
}  
  
.callout{  
  background: #ddd;  
}
```

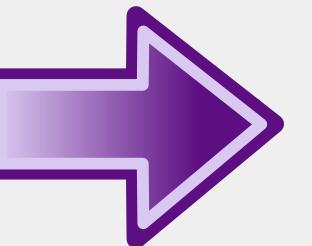
Estamos extendiendo `.content` en nuestra declaración, pero `.content` tiene un bloque para los `h2` dentro.

Al compilar el código, se añade lo que contiene `.content` a `.callout`, y hereda el `h2`.

# EXTEND

## \_buttons.scss

```
.btn-a{  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
.btn-b{  
  @extend .btn-a;  
  background: #ff0;  
}  
.sidebar .btn-a{  
  text-transform: lowercase;  
}
```



## application.css

```
.btn-a, .btn-b{  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
.btn-b{  
  background: #ff0;  
}  
.sidebar .btn-a, .sidebar .btn-b{  
  text-transform: lowercase;  
}
```

Al heredar .btn-b las propiedades de .btn-a, las hereda tanto del selector .btn-a como también del selector .sidebar .btn-b (cuando .btn-a está dentro de .sidebar), por lo que también se crea una regla idéntica para cuando .btn-b está dentro de .sidebar.

# EXTEND

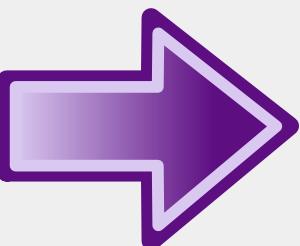
Por lo tanto, desde el momento en el que el `.btn-b` extiende de `.btn-a`, cualquier cambio realizado en `.btn-a` usando otros selectores afecta a `.btn-b`.

Para evitar este problema se utilizan selectores **placeholder**. Se nombran con el símbolo del porcentaje (%), además, pueden ser extendidos, pero nunca serán un selector en sí mismos.

# EXTEND

## \_buttons.scss

```
%btn{  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
.btn-b{  
  @extend %btn;  
  background: #ff0;  
}  
.sidebar .btn-a{  
  text-transform: lowercase;  
}
```



## application.css

```
.btn-a, .btn-b{  
  background: #777;  
  border: 1px solid #ccc;  
  font-size: 1em;  
  text-transform: uppercase;  
}  
.btn-b{  
  background: #ff0;  
}  
.sidebar .btn-a{  
  text-transform: lowercase;  
}
```

# EXTEND

Los selectores placeholder son bastante útiles cuando queremos reusar estilos CSS en hojas de estilos.

Hemos de tener cuidado al utilizar las directivas *extend* o *import*, ya que el número de selectores CSS por archivo está limitado en los navegadores y si sobrepasamos el límite todo lo que sobrepase es ignorado.

# MATH - COLOR

## Operaciones aritméticas básicas

Con SASS tenemos la posibilidad de realizar operaciones con números :

- Suma (+)
- Resta (-)
- Multiplicación (\*)
- División (/)
- Porcentaje (%)

Por defecto, al realizar una operación matemática SASS devuelve, si el resultado es decimal, 5 dígitos tras la coma redondeados hacia arriba.

**Ejemplo:** *font: normal 2em/1.5 Helvetica, sans-serif;*

# MATH - COLOR

## Operaciones aritméticas básicas

### CONCATENACIÓN DE STRINGS

La forma de concatenar string en SASS es mediante el operador `+`:

```
font: "Helvetica" + "Neue"; // "Helvetica Neue"
```

```
font: 'sans-' + serif; // 'sans-serif'  
font: sans- + 'serif'; // sans-serif
```

Si concatenamos **string**, **si** el operando de la izquierda tiene comillas simples, el resultado es un **string** con comillas simples. Sin embargo, si no va entre comillas, el resultado será sin ellas (aunque el otro operando las tenga).

# MATH - COLOR

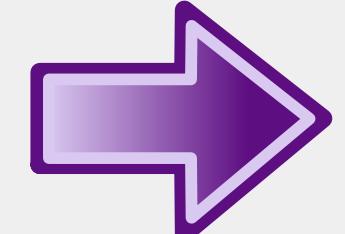
## Operaciones aritméticas básicas

### OPERACIONES

Cuando realizamos operaciones en SASS donde las unidades entre las operaciones varían:

**application.scss**

```
h2{  
  font-size:10px + 4pt;  
}
```



**application.css**

```
h2{  
  font-size: 15.33333px;  
}
```

SASS hace la conversión

# MATH - COLOR

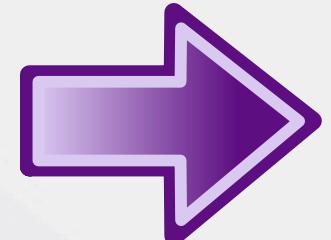
## Operaciones aritméticas básicas

### OPERACIONES

Si utilizamos unidades relativas como **em**, no es capaz de hacer la conversión y devuelve un error diciendo que no son compatibles las unidades.

#### application.scss

```
h2{  
  font-size: 10px + 4em;  
}
```



#### application.css

*Incompatible units: 'em' and 'px'*

# MATH - COLOR

## Funciones matemáticas

Estas son algunas de las funciones matemáticas predefinidas en SASS:

- **round(\$number)**: Redondea al número entero más cercano.
- **ceil(\$number)**: Redondea hacia arriba.
- **floor(\$number)**: Redondea hacia abajo.
- **abs(\$number)**: Obtiene el valor absoluto.
- **min(\$number)**: Obtiene el número más pequeño de una lista.
- **max(\$number)**: Obtiene el número más grande de una lista.
- **percentage(\$number)**: Convierte el número en un porcentaje.

# MATH - COLOR

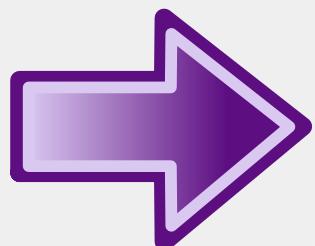
## Funciones matemáticas

### ¿CÓMO PODEMOS UTILIZAR ESTAS FUNCIONES?

Estas funciones se invocan de la misma manera que si las hubieramos creado nosotros.

**application.scss**

```
$context: 1000px;  
h2{  
  line-height: ceil(1.2);  
}  
.sidebar{  
  width: percentage(350px/$context);  
}
```



**application.css**

```
h2{  
  line-height: ceil(2);  
}  
.sidebar{  
  width: 35%;  
}
```

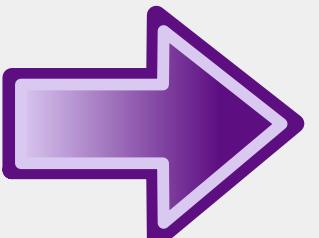
# MATH - COLOR

## Funciones matemáticas + colores

Estas funciones simplifican bastante los cambios de colores en los elementos de una web. Por ejemplo:

**application.scss**

```
$color-base: #333;  
.addition{  
    background: $color-base + #123;  
}
```



**application.css**

```
.addition{  
    background: #456;  
}
```

Al compilar el archivo, se genera el color resultante de haber sumado en hexadecimal

#333333 + #112233 = #445566

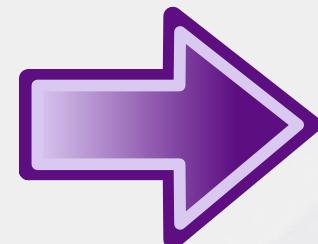
# MATH - COLOR

## Funciones matemáticas + colores

Otras funciones típicas son: **subtraction**, **multiplication** y **division**.

**application.scss**

```
$color-base: #333;  
.subtraction{  
    background: $color-base - #123;  
}  
.multiplication{  
    background: $color-base *2;  
}  
.division{  
    background: $color-base /2;  
}
```



**application.css**

```
.subtraction{  
    background: #210;  
}  
.multiplication{  
    background: #666;  
}  
.division{  
    background: #191919;  
}
```

# MATH - COLOR

## Funciones matemáticas + colores

Otras funciones que se pueden definidas son:

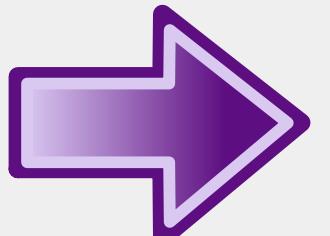
- **lighten(\$color, \$amount)**: Genera un color más claro.
- **darken(\$color, \$amount)**: Genera un color más oscuro.
- **saturate(\$color, \$amount)**: Modifica la intensidad del color (añadiéndole).
- **mix(\$color1, \$color2, [\$weight])**: Mezcla de dos colores, el tercer parámetro es opcional y lo que hace es indicr el % del primer color que usamos.
- **grayscale(\$color)**: Convierte un color en escala de grises.
- **invert(\$color)**: Devuelve el inverso de un color.
- **complement(\$color)**: Devuelve el complementario.

# MATH - COLOR

## Funciones matemáticas + colores

application.scss

```
$color-base: #333;  
$color-escala: #87bf64;  
.lighten{  
    color: lighten($color-base, 20%);  
}  
.darken{  
    color: darken($color-base, 20%);  
}  
.saturate{  
    color: saturate($color-base, 20%);  
}
```



application.css

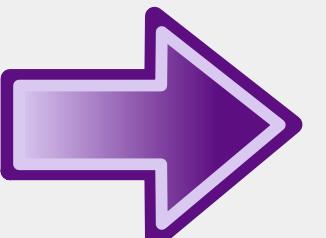
```
.lighten{  
    color: #666;  
}  
.darken{  
    color: black;  
}  
.saturate{  
    color: #82d54e;  
}
```

# MATH - COLOR

## Funciones matemáticas + colores

application.scss

```
$color-base: #333;  
$color-escala: #87bf64;  
.desaturate{  
    color: desaturate($color-base, 20%);  
}  
.mix-a{  
    color: mix(#4ff0, $107fc9);  
}  
.mix-b{  
    color: mix(#4ff0, $107fc9, 30%);  
}
```



application.css

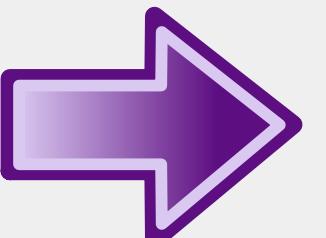
```
.desaturate{  
    color: #323130;  
}  
.mix-a{  
    color: #87bf64;  
}  
.mix-b{  
    color: #57a58c;  
}
```

# MATH - COLOR

## Funciones matemáticas + colores

application.scss

```
$color-base: #333;  
$color-escala: #87bf64;  
.grayscale{  
  color: grayscale($color-escala);  
}  
.invert{  
  color: invert($color-escala);  
}  
.complement{  
  color: complement($color-escala);  
}
```



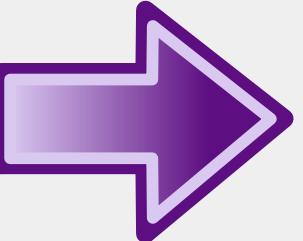
application.css

```
.grayscale{  
  color: #929292;  
}  
.invert{  
  color: #78409b;  
}  
.complement{  
  color: #9c64bf;  
}
```

# RESPONSIVE

application.css

```
.sidebar{  
    border: 1px solid #ccc;  
}  
  
@media(min-width: 700px){  
    // Para los dispositivos con al menos  
    700px de ancho  
    .sidebar{  
        float: right;  
        width: 30%;  
    }  
}
```



application.scss

```
.sidebar{  
    border: 1px solid #ccc;  
    @media(min-width: 700px){  
        float: right;  
        width: 30%;  
    }  
}
```

# RESPONSIVE

Podemos combinar fácilmente las *media queries* con *mixin* utilizando la directiva **@content**. Esta directiva lo que hace es incluir un bloque de propiedades dentro de un mixin.

## application.scss

```
@mixin respond-to{  
  @media(min-width: 700px){  
    @content  
  }  
}  
.sidebar{  
  border: 1px solid #ccc;  
  @include respond-to{  
    float: right;  
    width: 30%;  
  }  
}
```

# RESPONSIVE

Una de las prácticas más frecuentes en estos casos es enviar al mixin un argumento sobre el media, de forma que se realice esta comprobación, por ejemplo, aplicar un estilo sólo si estamos usando una tablet.

En esta declaración podríamos añadir un **else** o **else-if** si queremos seguir realizando comprobaciones para los diferentes dispositivos.

```
application.scss
@mixin respond-to ($media){
  @if $media == tablet {
    @media(min-width: 700px){
      @content
    }
  }
}

.sidebar{
  border: 1px solid #ccc;
  @include respond-to{
    float: right;
    width: 30%;
  }
}
```

# RESPONSIVE

Esta puede considerarse una solución más correcta.

Con esto, lo que conseguimos es que se incluya si se cumple con el ancho mínimo que nosotros definimos, de forma que podamos usar este mixin en cualquier lugar donde tengamos un estilo para un valor *min-width*.

## application.scss

```
@mixin respond-to ($query){  
  @media(min-width: $query){  
    @content  
  }  
}  
.sidebar{  
  border: 1px solid #ccc;  
  @include respond-to{  
    float: right;  
    width: 30%;  
  }  
}
```

# RESPONSIVE

Si se desea que este mixin no sólo se use para un mínimo, sino que pueda ser para un máximo también, la posibilidad que tenemos para modificarlo es:

```
application.scss
@mixin respond-to ($val, $query){
  @media($val: $query){
    @content
  }
}
.sidebar{
  border: 1px solid #ccc;
  @include respond-to (max-width, 600px) {
    float: right;
    width: 30%;
  }
}
```

# índice de Contenidos

**01** INSTALACION DE SASS.  
SINTAXIS DE SASS. VARIABLES  
CON SASS

**02** VARIABLES CON SASS. MIXINS

**03** EXTENDS. MEDIA QUERIES CON  
SASS.

**04** DIRECTIVAS.

# DIRECTIVAS

## Funciones

Las funciones nos van a permitir definir operaciones complejas que podremos abstraer de una manera sencilla y utilizar en cualquier otra parte de la hoja.

En SASS la estructura para definir una función es la siguiente:

**`@function nombreFuncion ([argumento]) {...}`**

# DIRECTIVAS

## Funciones

Para invocarla, los paréntesis **son obligatorios** aunque no se reciba ningún parámetro como entrada, por ejemplo:

### application.scss

```
@function fluidize(){    // @function nombreFuncion([arg])
    @return 35%;
}
```

# DIRECTIVAS

## Funciones

### application.scss

```
@function fluidize($target, $content) {  
  @return ($target/$content) * 100%;  
}  
.sidebar {  
  width: fluidize (350px, 1000px);  
}
```

La función se pasa con dos variables

Dependiendo de los valores de *target* y  
*content* nos devolverá un valor.

# DIRECTIVAS

## if

La sentencia **if** corresponde a una estructura condicional. Por ejemplo, si queremos que nuestra cabecera tenga un fondo negro si el tema es dark:

### application.scss

```
$theme: dark;  
header{  
  @if $theme == dark{  
    background: #000;  
  }  
}
```

# DIRECTIVAS

**if**

Hay varios operadores para realizar una comparativa en SASS:

- Igual: ==
- No es igual: !=
- Mayor que: >
- Mayor o igual que: >=
- Menor que: <
- Menor o igual que: <=



**Los 4 últimos operadores sólo sirven para comparar números**

# DIRECTIVAS

## if

En este caso el *theme* será *light*, pero dependiendo del tema si es *dark* o no, lo que haremos será aplicar un color de fondo u otro, y para eso se utiliza la sentencia **if-else**

## application.scss

```
$theme: light;  
header{  
  @if $theme == dark{  
    background: #000;  
  } @else{  
    background: #fff;  
  }  
}
```

# DIRECTIVAS

if

## COMPARACIÓN EN CASCADA

Para realizar una comparación en cascada se añade otra comparativa, en nuestro ejemplo, si el tema es rosa el fondo será rosa también.

application.scss

```
$theme: pink;  
header{  
  @if $theme == dark{  
    background: #000;  
  } @else if $theme==pink {  
    background. pink;  
  } @else{  
    background: #fff;  
  }  
}
```

# DIRECTIVAS

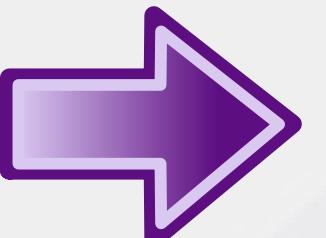
## each

La directiva **@each** nos permite iterar una lista por cada uno de los items que tiene.

La manera de invocarla es:**@each \$item in \$lista;**

### application.scss

```
$autores: rosa lucia pablo jose;  
@each $autor in $autores {  
  .author-#${$autor}{  
    background: url(author-#${$autor}.jpg);  
  }  
}
```



### application.css

```
.author-rosa{  
  background: url(author-rosa.jpg);  
}  
.author-lucia{  
  background: url(author-rosa.jpg);  
}  
.author-pablo{  
  background: url(author-rosa.jpg);  
}  
.author-jose{  
  background: url(author-rosa.jpg);  
}
```

# DIRECTIVAS

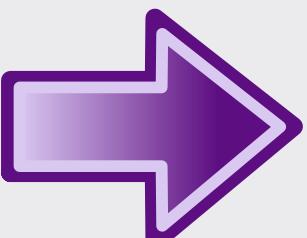
## for y while

Estas sentencias funcionan como el @each pero a diferencia del anterior se encuentran asociado por lo general a números.

Por ejemplo, si queremos realizar una iteración del 1 al 3 de forma que cada uno de los valores del bucle (1..3) lo multiplicaremos por 30 píxeles de forma que cuando compilemos tengamos una lista de ítems que use los *i* valores.

### application.scss

```
.item{  
  position: absolute;  
  right: 0;  
  @for $i from 1 through 3{  
    &.item-#{$i}{  
      top: $i * 30px;  
    }  
  }  
}
```



### application.css

```
.item{  
  position: absolute;  
  right: 0;  
}  
.item.item-1{  
  top: 30px;  
}  
.item.item-2{  
  top: 60px;  
}  
.item.item-3{  
  top: 90px;
```

# DIRECTIVAS

## for y while

Con esto conseguimos que para cada uno de los ítems tenga un valor de top diferente  
**application.scss**

Algo parecido sería si en lugar de usar el **@for** usamos el **@while**, la diferencia radica en que con el **@while** debemos ir actualizando el índice en cada iteración.

```
$i: 1;  
.item{  
  position: absolute;  
  right: 0;  
  @while $i < 4{  
    &.item-#{$i}{  
      top: $i * 30px;  
    }  
    $i: $i+1;  
  }  
}
```

# CUÁNDO USAR FUNCIONES, MIXIN . CÚANDO EXTEND

- **Mixin** debe ser usado cuando definimos propiedades similares múltiples veces con pequeñas variaciones.
- **Extends** cuando tenemos propiedades qe son exactamente iguales, es decir, declaraciones copiadas un y otra vez.
- **Funciones** son operaciones que necesitamos usar muchas veces y que nos **devuelven** un valor.

# CUÁNDO USAR FUNCIONES, MIXIN . CÚANDO EXTEND

## application.scss

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

Se asigna un *border-radius* si el mixin recibe el valor de cierto o lo toma por defecto.

# CUÁNDO USAR FUNCIONES, MIXIN . CÚANDO EXTEND

## application.scss

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

En este ejemplo, lo que se hace es si recibimos un valor de rounded, entonces le asignamos ese valor como border-radius.

En este caso, a .btn-a no le estmos dando un border, pero en cambio a .btn-b sí, ya que le estamos pasando como segundo argumento un valor distinto de falso o nula y por tanto entra en el if y le asigna el border-radius que estmos enviándole por parámetro.