

# Sass

<b>1. Introducción.....</b>	<b>2</b>
<b>1.1. Tipos de sintaxis Sass .....</b>	<b>2</b>
<b>1.2. Sass con Visual Studio Code.....</b>	<b>3</b>
1.2.1. Configuración de <i>Live Sass Compiler</i> .....	5
<b>2. Lo básico de Sass .....</b>	<b>8</b>
2.1. Variables .....	8
2.2. Comentarios.....	9
2.3. Listas .....	9
2.4. Mapas .....	9
2.5. Interpolación.....	10
2.6. Anidamiento .....	11
2.7. Módulos .....	13
2.8. Herencia .....	14
2.9. Operadores .....	15
2.10. Partial.....	16
<b>3. Webgrafía .....</b>	<b>17</b>

# 1. Introducción

Sass es un preprocesador CSS. Los preprocesadores CSS son herramientas que permiten escribir código CSS de una manera más eficiente y estructurada, facilitando la creación y el mantenimiento de hojas de estilo. Funciona como una capa adicional sobre el lenguaje CSS estándar, proporcionando características adicionales que no están presentes en CSS puro.

Algunos de los preprocesadores CSS más populares son estos:

- **Sass:** Sass (*Syntactically Awesome Style Sheets*) es uno de los preprocesadores más utilizados. Introduce mejoras como variables, anidamiento de selectores, mixins (fragmentos reutilizables de código), funciones, herencia y más. Luego, este código escrito en Sass se compila a CSS estándar antes de implementarse en un sitio web.
- **LESS:** Similar a Sass, LESS ofrece funcionalidades adicionales para escribir estilos CSS de manera más eficiente. Permite el uso de variables, mixins, anidamiento de reglas, funciones matemáticas y más.
- **Stylus:** Es otro preprocesador que sigue una sintaxis más minimalista y flexible. Ofrece características similares a Sass y LESS, incluyendo variables, anidamiento, mixins y funciones.

Estos preprocesadores ayudan a los desarrolladores a escribir código CSS de manera más organizada y mantenible, reduciendo la repetición de código y facilitando la reutilización de estilos. Además, permiten el uso de lógica y funciones que no están presentes en CSS estándar, lo que hace que el proceso de desarrollo sea más rápido y menos propenso a errores.

En resumen, un preprocesador CSS es una herramienta que amplía las capacidades del CSS convencional al agregar características avanzadas, facilitando así la creación y gestión de estilos para sitios web y aplicaciones web.

## 1.1. Tipos de sintaxis Sass

Sass tiene dos sintaxis distintas, que da lugar a dos tipos de archivos distintos:

- **SCSS:** La sintaxis SCSS (.scss) es la más utilizada y la que utilizaremos aquí. Es un superconjunto de CSS, lo que significa que todo CSS válido también es SCSS válido. Utiliza llaves '{}' y puntos y coma ';' para definir bloques de código y separar declaraciones, respectivamente. Un ejemplo de código SCSS es:

```
$color-primary: #3498db;

.header {
  background-color: $color-primary;
  font-size: 18px;
}
```

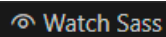
- **Sass:** La sintaxis Sass (.sass) es menos frecuente. Se caracteriza por su sintaxis indentada, lo que significa que utiliza espacios en blanco (indentación) en lugar de llaves '{}' y puntos y coma ';'. La indentación se utiliza para definir la jerarquía y los bloques de código, lo que puede resultar en una estructura visualmente más limpia y con menos caracteres que SCSS. Un ejemplo de código SASS es:

```
$color-primary: #3498db

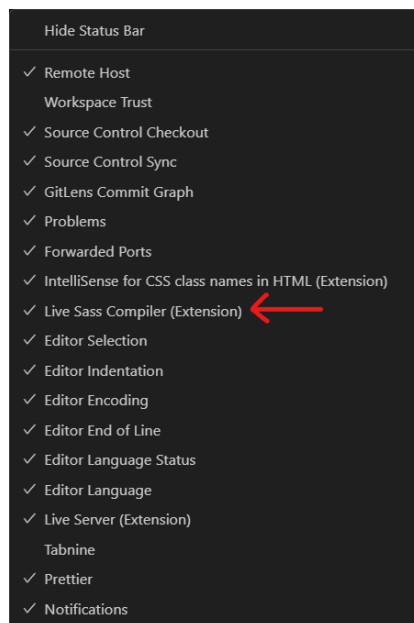
.header
  background-color: $color-primary
  font-size: 18px
```

## 1.2. Sass con Visual Studio Code

Para utilizar Sass con VSC vamos a instalar una extensión que nos facilitará mucho el trabajo, [Live Sass Compiler](#). Una vez instalada nos debería aparecer en la barra inferior de VSC una opción llamada *Watch Sass*:

A dark rectangular button with a circular icon containing a play symbol and the text "Watch Sass" in a light color.

Si no aparece, deberías poder añadirla haciendo clic derecho sobre la barra y seleccionando *Live Sass Compiler*:

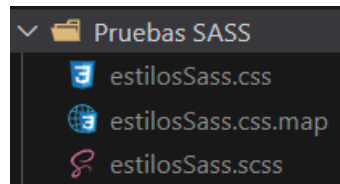


Crea un fichero Sass cualquiera y añade algo de código Sass. Por ejemplo:

```
$color-primary: #3498db;

.header {
  background-color: $color-primary;
  font-size: 18px;
}
```

Si ahora pulsas sobre *Watch Sass* verás que se generan automáticamente dos archivos nuevos dentro del mismo directorio donde ubicaste el archivo Sass:



El archivo con extensión CSS es el resultado de compilar el archivo Sass en CSS.

Además, aparece otro archivo con extensión *map*. Este es un archivo de mapa de origen, conocido como **Source Map** en inglés. Este archivo es generado por el preprocesador Sass durante la compilación y está diseñado para mapear el código CSS resultante de vuelta al código fuente original en SCSS. Su objetivo principal es ayudar en el proceso de depuración y desarrollo.

El *Source Map* contiene información que relaciona el código CSS producido con el código SCSS original. Cuando inspeccionas el código CSS generado en el navegador, el *Source Map* proporciona referencias y enlaces hacia las líneas de código en el archivo SCSS correspondiente. Esto es útil para identificar y depurar problemas, ya que te permite trabajar directamente con el código fuente SCSS en lugar de tener que referirte solo al código CSS compilado.

Los *Source Maps* son particularmente útiles en entornos de desarrollo donde se utiliza código preprocesado (como SCSS) que luego se compila a CSS para su implementación en un sitio web o una aplicación web. Siempre que esté habilitada la opción de generar *Source Maps* durante la compilación de SCSS, se creará un archivo *.map* junto al archivo CSS resultante.

Probemos este archivo *.map*. Vamos a crear un archivo HTML simple que utilice el CSS generado a partir de Sass:

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Prueba Sass</title>
  <link rel="stylesheet" href="estilosSass.css">
</head>

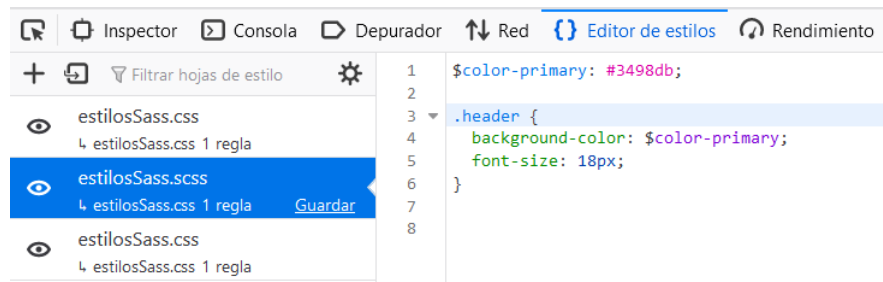
<body>
  <div class="header">
    <h1>Hola Sass</h1>
  </div>
</body>

</html>
```

Si ahora usamos las herramientas de depuración del navegador y nos situamos sobre la clase *header*, veremos que nos enlaza al archivo SCSS:



Y si hacemos clic en *estilosSass.scss* nos lleva al editor de estilos del archivo SCSS:



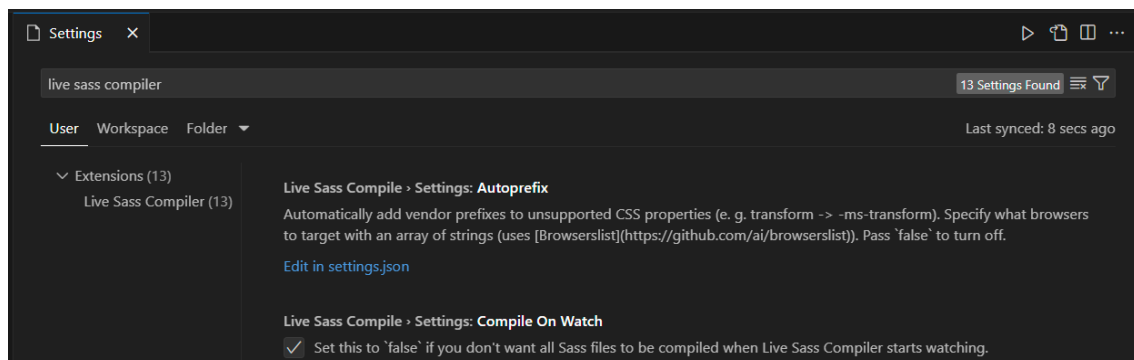
Ahora prueba a eliminar el archivo *.map*, verás como el navegador nos enlaza al archivo CSS, no al SCSS:



### 1.2.1. Configuración de *Live Sass Compiler*

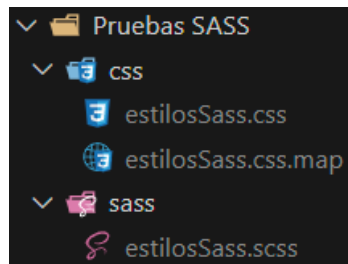
Vamos a ver algunas opciones de configuración interesantes de esta extensión. Puedes revisarlas todas en su propia [documentación](#). Incluso el autor tiene un vídeo en [YouTube](#) donde explica estas opciones de configuración.

Abrimos las opciones de VSC pulsando **Ctrl + ,** y filtramos por “*live sass compiler*”:



La primera opción interesante es ***compileOnWatch***. Por defecto, cuando pulsamos *Watch Sass* la extensión busca y compila todos los archivos Sass que encuentra en el proyecto. Poniendo esta opción a *false* forzamos a que solo compile archivos Sass que son modificados mientras la extensión está *mirando* (*watching*).

La otra opción interesante es ***formats***, con la que podemos modificar los formatos y las rutas donde se almacenan los archivos compilados .css y .map. Para mejorar la legibilidad y organización de nuestros proyectos, vamos a almacenar todos nuestros ficheros sass dentro de un carpeta llamada “sass” y queremos que los archivos .css y .map generados estén en otra carpeta llamada “css”. Algo así:



Sin embargo, la extensión coloca por defecto los archivos generados en el mismo directorio en el que encuentra los archivos Sass. Para modificar este comportamiento usaremos la opción ***formats***:

Live Sass Compile > Settings: **Formats**  
Set your exported CSS Styles, Formats & save location.  
[Edit in settings.json](#)

Vamos a editar esta opción en el fichero de configuración *json*:

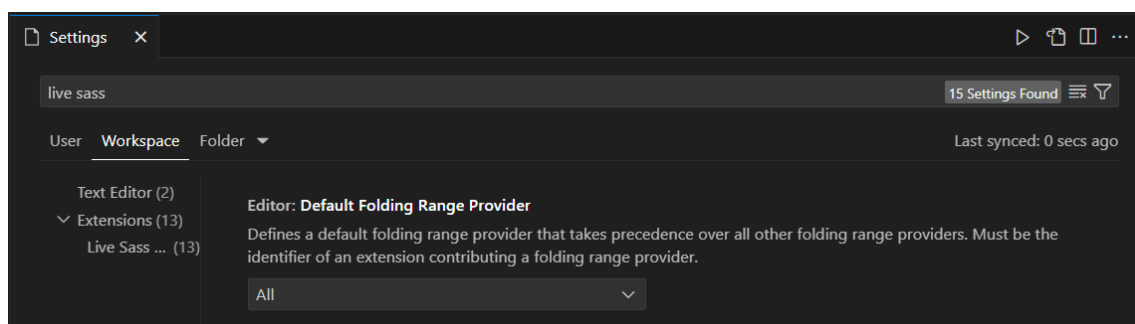
```
"liveSassCompile.settings.formats": [  
  {  
    "format": "expanded",  
    "extensionName": ".css",  
    "savePath": "~/../css",  
    "savePathReplacementPairs": null  
  },  
  {  
    "format": "compressed",  
    "extensionName": "-min.css",  
    "savePath": "~/../css",  
    "savePathReplacementPairs": null,  
    "generateMap": false  
  }  
]
```

***liveSassCompile.settings.formats*** es un array que contiene diferentes objetos, cada uno definiendo un formato de salida específico para los archivos CSS resultantes. En este caso tenemos dos objetos. En el primer objeto:

- **format:** Define el formato de salida del CSS, en este caso, “expanded” (expandido). Esto significa que el CSS generado mantendrá una estructura legible y con espacios para facilitar la lectura humana.
- **extensionName:** Es la extensión que se utilizará para los archivos CSS compilados en este formato, en este caso, “.css”.
- **savePath y savePathReplacementPairs:** Especifican la ruta de guardado para los archivos CSS compilados. Si se usan ambos, savePath tiene preferencia. Si se especifica *null* se utilizará la ubicación predeterminada. En este caso usamos *savePath* e indicamos que se guarden en una carpeta llamada “css” que es hermana de la que contiene el archivo. Así, si tenemos la estructura que hemos definido antes, se guardarán en el directorio “css” hermano de “sass”.
- **generateMap:** Especifica si se generará un archivo *.map* o no.

En el segundo objeto se modifica el formato de salida a “compressed” (comprimido), lo que indica que el CSS generado en este formato será *minificado* para ocupar menos espacio, eliminando espacios en blanco y comentarios. Es la versión que se debería utilizar en producción, no en pruebas. Además, se modifica el nombre del fichero css añadiéndole “-min”, que es un estándar, y se elimina la opción de crear un archivo *.map*, dado que producción ya no son necesarios estos archivos.

Si quieres establecer alguna característica personalizada para un workspace concreto o para una carpeta concreta, puedes hacerlo seleccionando su pestaña en la interfaz *Settings* (Ctrl + ,).



## 2. Lo básico de Sass

Puedes encontrar más información en <https://sass-lang.com/guide/>.

### 2.1. Variables

Antes de la aparición de las *custom properties* en CSS, la posibilidad de usar variables era un lujo que sólo proporcionaban los preprocesadores CSS.

En Sass definimos las variables con el símbolo '\$'. Por ejemplo:

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Da como resultado:

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Podemos distinguir entre los siguientes tipos de datos: números, cadenas, colores, booleanos, null, mapas y listas. Tienes información adicional en el siguiente enlace de la documentación: <https://sass-lang.com/documentation/values/>

En Sass podemos distinguir entre:

- Variables globales: Estarán disponibles a lo largo de todo el código Sass y se definen normalmente al principio de los ficheros Sass.
- Variables locales: Se declaran dentro de bloques ('{ }') y sólo están disponibles y tienen valor dentro de ese bloque.

Por ejemplo:

```
// Variable global fuera de todo bloque
$logo-width: 50%;

.header {
  // Variable local
  $header-width: 50%;
}
```

Si una variable local y una variable global tienen el mismo nombre prevalece el valor de la variable local.



## 2.2. Comentarios

Sass permite utilizar comentarios de una línea o multilínea.

```
// Este es un comentario de una línea

/*
  Este es un comentario multilínea
*/
```

Los comentarios de una línea no aparecen en CSS, (*silent comments*)

Los comentarios de varias líneas sí aparecen en el CSS, (*loud comments*).

Algunos compiladores de Sass permiten generar CSS comprimido. De forma predeterminada, los comentarios de varias líneas se eliminarán del CSS compilado en modo comprimido salvo que comiencen con “*/\*!*”.

## 2.3. Listas

Las listas son colecciones de valores de datos. Se declaran de la siguiente manera:

```
// Forma general de la definición
$variable_lista: (v1, v2, v3);

// Por ejemplo
$sizes: (40px, 80px, 160px);

// Otra posibilidad (conforme a las recomendaciones)
$sizes: (
  // Explicación
  40px,
  // Explicación
  80px,
  // Explicación
  160px
);
```

## 2.4. Mapas

Los mapas son colecciones de valores de datos a los que accedemos por clave:

```
// Forma general de la definición
$nombre_mapa: (
  "clave1": valor1,
  "clave2": valor2,
  ...
  "claveN": valorN,
);
```

Por ejemplo:

```
$breakpoint: (  
  "pequeño": 576px,  
  "medio": 768px,  
  "grande": 992px,  
);
```

## 2.5. Interpolación

Es una de las características más útiles y más usadas de Sass. La Interpolación nos permite, casi en cualquier sitio de un documento Sass, incrustar una expresión, cuyo resultado, al ser evaluado, formará un trozo de código CSS.

Para que esto ocurra debemos incluir la expresión de la siguiente manera:

```
#{expresión_a_evaluar}
```

Algunos de los lugares donde podemos usar la interpolación son los siguientes:

- Selectores.
- Nombres de propiedades.
- Comentarios.
- Reglas de Sass como `@import`, `@extend` y `@mixins`.
- Cadenas con o sin comillas.
- Funciones.

Veamos algunos ejemplos:

```
// Interpolación en selectores  
$button-type: "error";  
$btn-color: #f00;  
$name: "send";  
  
.btn-#{ $name } {  
  background-color: $btn-color;  
}  
  
// Interpolación en el uso de funciones  
$fondo: "images/fondos/default.png";  
  
.container {  
  background-image: url("#{ $fondo }");  
}  
  
// Interpolación en comentarios  
$autor: "Isaac";
```

```
/*  
  Web desarrollada por #{$autor}  
*/
```

Este código Sass compila en:

```
.btn-send {  
  background-color: #f00;  
}  
  
.container {  
  background-image: url("images/fondos/default.png");  
}  
  
/*  
  Web desarrollada por Isaac  
*/
```

## 2.6. Anidamiento

Conforme las hojas de estilos se van haciendo más grandes y complejas, los selectores y sus niveles de anidamiento se van alargando. Sass nos permite anidar selectores para que, además de escribir menos, agrupemos selectores relacionados dentro de una misma organización sintáctica.

Por ejemplo, esta es la estructura CSS para un menú simple:

```
nav ul {  
  margin: 0;  
}  
  
nav li {  
  display: inline-block;  
}  
  
nav a {  
  padding: 6px 12px;  
}
```

Usando las posibilidades de anidamiento de Sass, sería:

```
nav {  
  ul {  
    margin: 0;  
  }  
}
```

```
li {  
  display: inline-block;  
}  
  
a {  
  padding: 6px 12px;  
}  
}
```

Además, podemos hacer referencia al selector padre usando el carácter '&':

```
a {  
  color: blue;  
  
  &:hover {  
    color: red;  
  }  
}
```

Compilaría a:

```
a {  
  color: blue;  
}  
  
a:hover {  
  color: red;  
}
```

Podemos incluso usar ese carácter '&' para crear nuevos selectores extendiendo el nombre del selector. Un ejemplo:

```
.btn {  
  padding: 6px 12px;  
}  
  
.btn-small {  
  font-size: 12px;  
}  
  
.btn-big {  
  font-size: 16px;  
}
```

En Sass sería:

```
.btn {  
  padding: 6px 12px;  
  
  &-small {  
    font-size: 12px;  
  }  
  
  &-big {  
    font-size: 16px;  
  }  
}
```

```
}
```

## 2.7. Módulos

No es necesario escribir todo el código Sass en un sólo archivo. Puedes dividirlo como quieras e ir uniéndolos todos con la regla `@use`. Esta regla carga otro archivo Sass como módulo, lo que significa que puede hacer referencia a sus variables, mixins y funciones en un archivo Sass distinto del original con un espacio de nombres basado en el nombre del archivo.

Por ejemplo, digamos que tenemos un archivo Sass “`_base.scss`” donde establecemos algunas configuraciones básicas:

```
// Fichero _base.scss
$primary-color: #333;

body {
  color: $primary-color;
}
```

Y queremos incluir ese fichero en nuestro fichero de estilos general:

```
// Fichero estilos.scss
@use "base";

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

Aquí hay 3 cosas importantes:

- Los ficheros destinados a ser módulos comienzan con un guion bajo ‘`_`’.
- Cuando usamos la regla `@use`, el guion bajo no es necesario.
- Para acceder a una variable declarada en un módulo usamos el nombre del módulo y un punto ‘`.`’: `base.$primary-color`.

A pesar de que tenemos dos archivos, el compilador entiende que `_base.scss` es un módulo y no debe generar un fichero CSS. Por tanto, solo generará `estilos.css` y, si así lo hemos configurado, `estilos.css.map`.

```
body {
  color: #333;
}
```

```
.inverse {  
  background-color: #333;  
  color: white;  
}/*# sourceMappingURL=estilos.css.map */
```

## 2.8. Herencia

La regla **@extend** permite compartir un conjunto de propiedades CSS de un selector a otro, evitando así la repetición de código y promoviendo la reutilización.

Cuando usas **@extend** estás indicando que un selector CSS debe heredar todos los estilos definidos en otro selector. Por ejemplo:

En nuestro ejemplo, vamos a crear una serie simple de mensajes para *errors*, *warnings* y *successes* usando otra característica que va de la mano con **@extend**, *placeholder classes*. Una clase de marcador de posición es un tipo especial de clase que solo se imprime cuando está extendida y puede ayudar a mantener limpio y ordenado el CSS compilado. Las *placeholder classes* se crean utilizando el símbolo de porcentaje, **%**. Estos selectores no se compilan directamente en el CSS resultante, a menos que sean referenciados por un selector concreto.

La idea detrás de las *placeholder classes* es que actúan como plantillas de estilos que se pueden heredar en otros selectores. Esto permite una escritura más modular y reutilizable del código CSS.

SCSS:

```
// Placeholder class  
%propiedades-mensajes {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
  
// Placeholder class  
%img-responsive {  
  max-width: 100%;  
  display: block;  
}  
  
.error,  
.warning,  
.success {  
  @extend %propiedades-mensajes;  
}  
  
.success {  
  border-color: green;  
}  
  
.warning {  
  border-color: yellow;
```

```
}  
  
.error {  
  border-color: red;  
}
```

CSS:

```
.error,  
.warning,  
.success {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
  
.success {  
  border-color: green;  
}  
  
.warning {  
  border-color: yellow;  
}  
  
.error {  
  border-color: red;  
}
```

Lo que hace el código anterior es definir dos *placeholder classes*: **%propiedades-mensajes** y **%img-responsive**. La primera se usa con **@extend** en las clases **success**, **error** y **warning**, por lo que en el CSS generado las tres clases tienen las 3 propiedades de **%propiedades-mensajes**. Como la *placeholder class* **%img-responsive** no se utiliza en ningún sitio, no aparece en el CSS generado.

Esto ayuda a evitar tener que escribir varios nombres de clases en elementos HTML.

## 2.9. Operadores

Las versiones actuales de CSS incorporan la posibilidad de hacer cálculos matemáticos. Sin embargo, los desarrolladores ya podían hacer cálculos desde hace tiempo con la ayuda de Sass.

Sass tiene un puñado de operadores matemáticos estándar como **+**, **-**, **\***, **math.div()** y **%**. En nuestro ejemplo, vamos a hacer algunos cálculos simples para calcular el ancho de un artículo y aparte.

SCSS:

```
@use "sass:math";  
  
$numero1: 10;
```

```
$numero2: 3;

$resultado_suma: $numero1 + $numero2;
$resultado_resta: $numero1 - $numero2;
$resultado_multiplicacion: $numero1 * $numero2;
$resultado_division: math.div($numero1, $numero2);
$resultado_modulo: $numero1 % $numero2;

// Mostrar los resultados
body {
  content: "Suma: #{ $resultado_suma}, Resta: #{ $resultado_resta},
Multiplicación: #{ $resultado_multiplicacion}, División:
#{ $resultado_division}, Módulo: #{ $resultado_modulo}";
}
```

CSS:

```
body {
  content: "Suma: 13, Resta: 7, Multiplicación: 30, División: 3.3333333333,
Módulo: 1";
}
```

## 2.10. Partials

Un *partial* en Sass es un archivo que contiene fragmentos de código CSS o Sass que se pueden incluir en otros archivos Sass. Los nombres de los *partials* comienzan con un guion bajo ('\_') y tienen la extensión de archivo .scss o .sass. El propósito principal de los *partials* es modularizar y organizar el código, facilitando su reutilización y mantenimiento.

Cuando se importa un *partial* en otro archivo Sass, el guion bajo y la extensión del archivo se omiten en la importación. Esto se hace para indicar que no se debe compilar el *partial* como un archivo CSS independiente.

Por ejemplo, supongamos que tienes un archivo `_botones.scss`:

Un *partial* es un archivo Sass nombrado con un guion bajo inicial ('\_'). Por ejemplo, el archivo `_partial.scss`. El guion bajo permite a Sass saber que el archivo es un archivo parcial y que no debe generarse en un archivo CSS. Los parciales Sass se utilizan con la regla `@use`.

Puedes crear archivos Sass pequeños que contengan pequeños fragmentos de CSS y luego incluirlos en otros archivos Sass. Esta es una excelente manera de **modularizar** CSS y ayudar a que el mantenimiento sea más fácil.



### 3. Webgrafía

- <https://sass-lang.com/>
- <https://sass-guidelin.es/es/>