

# Parte 1 de 4: Sass y Compass

INSTALACIÓN DE SASS Y COMPASS

CREACIÓN DE PROYECTOS

FICHEROS DE CONFIGURACIÓN

SINTAXIS DE SASS

VARIABLES CON SASS

Apuntes de: **Rosa María Medina Gómez**

**Adaptados a Power Point por** José Jesús Torregrosa García

Curso de Formación del Profesorado a distancia  
Cefire Específico de FP de Cheste

Generalitat Valenciana Curso 2018 - 2019



# Introducción

- En informática, los desarrolladores web acaban adquiriendo un rol determinado dependiendo de sus especialidades, frontEnd o backEnd.
- Para ambos existen múltiples herramientas que permiten facilitar y optimizar el trabajo. A lo largo del curso, vamos a estudiar un precompilador/preprocesador de CSS, en concreto **Sass**.

# ¿Qué es un pre-compilador CSS?

- Un precompilador de CSS es una herramienta que nos permite crear scripts para posteriormente ser convertidos a CSS “real”.
- Estos scripts tendrán variables, condiciones, bucles o funciones: es un **lenguaje de programación**
- Por tanto, **Sass** no es más que el acrónimo de **Syntactically Awesome Style Sheets**. ¡Se suele decir que es CSS con superpoderes!

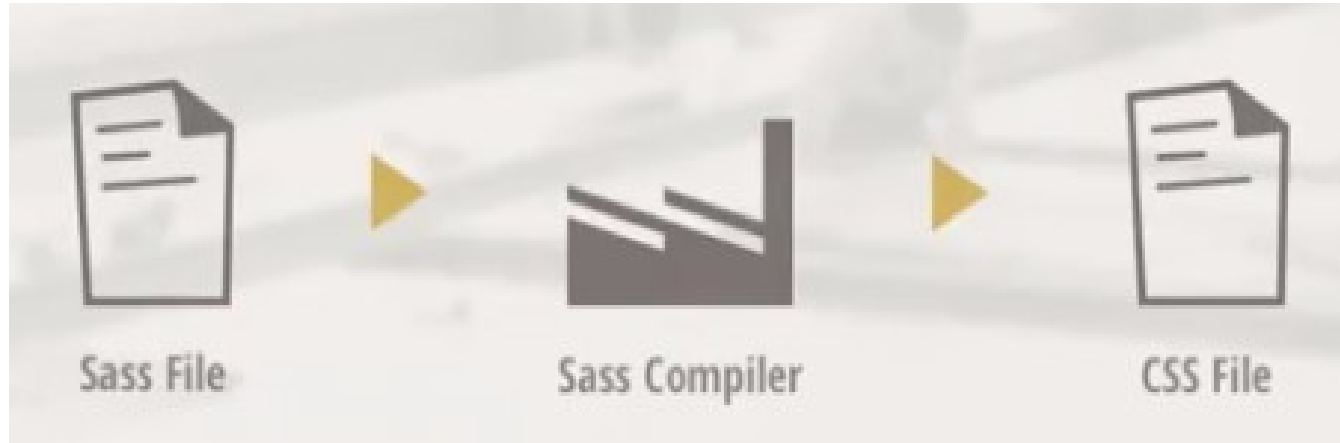
# ¿Qué es compass ?

- **Compass** es un framework CSS de código abierto de los creadores de Sass para generar hojas de estilo.
- Permite **crear** y **mantener** de forma más **fácil** nuestro CSS.
- Permite **compilar** hojas de estilo de forma **fácil** sin tener que realizarlo desde líneas de comando.
- **Compass.app** es una herramienta de Java (JRuby) de código abierto, y se encuentra disponible para las plataformas: **Mac, Linux y Windows**.

# ¿Qué es Sass?

- CSS fue diseñado para ser simple y en gran medida lo es, pero a medida que nuestra aplicación web se incrementa puede ser difícil el ir cambiando colores, fuentes, números, entre otras muchas propiedades.
- También, nos encontramos con otros problemas como puede ser el evitar repetir ciertas propiedades dentro de nuestra hoja de estilos.
- Hoy en día Sass es un preprocesador como puede ser CoffeeScript o Ham

# ¿Qué es Sass?



- Sass fue creado por Hampton Catlin, el cuál es el mismo que creó HamI, que es un generador de plantillas para HTML. Muchas personas escriben "SASS" en mayúsculas, pero es importante mencionar que cuando hablamos de Sass sólo debe ir en mayúsculas la primera letra.
- Sassy CSS (.scss) es la extensión por defecto

# Ejemplo de código SCSS

Un ejemplo de código .scss puede ser el siguiente: buttons.scss

```
1  $main:  #444;
2  .btn{
3    |   color: $main;  display: block;
4    |
5  .btn-a{
6    |   color: lighten($main, 30%);
7    |   &:hover{
8    |       |   color: lighten($main, 40%);
9    |       |
10   |   }
11 }
```

# Ejemplo de código CSS compilado

# buttons.css x

```
1  /* line 2, ../sass/buttons.scss */
2  .btn {
3    color: #444;
4    display: block;
5  }
6
7  /* line 5, ../sass/buttons.scss */
8  .btn-a {
9    color: #919191;
10 }
11 /* line 7, ../sass/buttons.scss */
12 .btn-a:hover {
13   color: #aaaaaa;
14 }
15
```

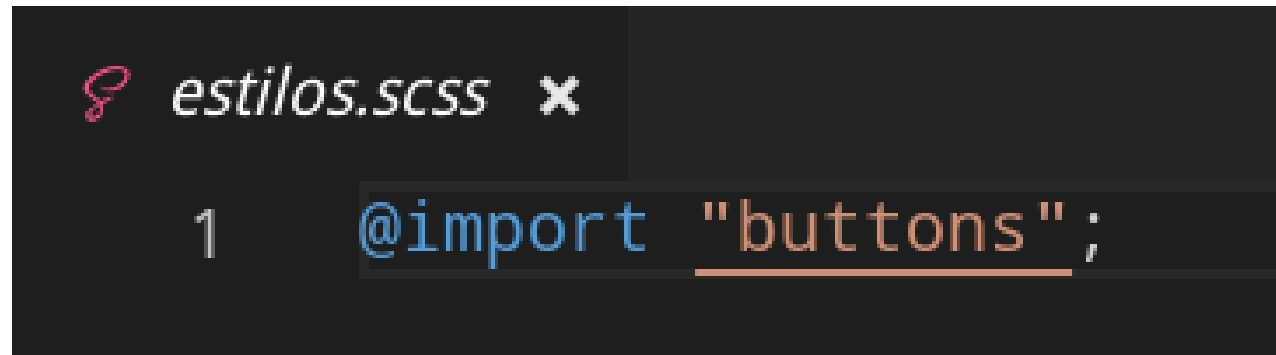


# La etiqueta **@import**

- **@import en CSS:** se desaconseja ya que se va descargando poco a poco los archivos CSS importados en el cliente
- **@import en Sass:** 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. Además, no tenemos que incluir la extensión del archivo, es opcional.

# La etiqueta @import

Por ejemplo, tenemos el archivo estilos.scss que importa buttons.scss

A screenshot of a code editor with a dark background. At the top, a tab is labeled 'estilos.scss' with a red squiggle icon on the left and a close 'x' icon on the right. Below the tab, the first line of code is '@import "buttons";'. The '@import' is in blue, and '"buttons"' is in orange and underlined. A line number '1' is visible to the left of the code.

```
1 @import "buttons";
```

Por tanto, es durante el proceso de compilación donde estilos.css pasa a tener los estilos de los archivos: estilos.scss y pruebas.scss

# La etiqueta @import



Además de incluir la hoja de estilos buttons, cuando compilamos y nos genera el archivo `estilos.css`, nos genera el CSS de buttons. **Esto no es deseable, por eso ...**

# La etiqueta **@import**

- ....renombramos el archivo `buttons.scss` a `_buttons.scss` y la cosa cambia:



# Selectores anidados

- Cuando tenemos un archivo de este tipo en CSS:

# estilos.css x

```
1  .content{
2    border: 1px solid ■#ccc; padding: 20px;
3  }
4
5  .content h2{
6    font-size: 3em; margin: 20px 0;
7  }
8
9  .content p{
10   font-size: 1.5em; margin: 15px 0;
11 }
```

# Selectores anidados

- Si creamos el archivo sass que lo va a generar tendremos (aquí vemos anidamiento o nesting):

```
otroEstilos.scss x
1  .content{
2
3      border: 1px solid #ccc; padding: 20px;
4
5      h2{
6          font-size: 3em; margin: 20px 0;
7      }
8
9      p{
10         font-size: 1.5em; margin: 15px 0;
11     }
12 }
```

# Selectores anidados

- Otro ejemplo de anidamiento:

 \_buttons.scss x

```
1  .btn{
2      |  text: {
3          |      decoration: underline;
4          |      transform: lowercase;
5          |  }
6  }
```

# Selectores anidados

- Anidamos la propiedad text y este es el resultado;

# estilos.css x

```
1  /* line 1, ../sass/_buttons.scss */
2  .btn {
3    | text-decoration: underline;
4    | text-transform: lowercase;
5    }
```






# Selectores anidados

Otra de las características de nesting es el `&`, donde hace referencia al selector padre, veamos un ejemplo:

```
otroEstilos.scss x
1  .content{
2      border: 1px solid  #ccc; padding: 20px;
3      .callout{
4          border-color:  red;
5      }
6      &.callout{
7          /* referencia a .content,
8             es un selector compuesto: .content.callout */
9          border-color:  green;
10     }
11 }
```

# Selectores anidados

# otroEstilos.css x

```
1  /* line 1, ../sass/otroEstilos.scss */
2  .content {
3      border: 1px solid  #ccc;
4      padding: 20px;
5  }
6  /* line 3, ../sass/otroEstilos.scss */
7  .content .callout {
8      border-color:  red;
9  }
10 /* line 6, ../sass/otroEstilos.scss */
11 .content.callout {
12     /* referencia a .content,
13     es un selector compuesto: .content.callout */
14     border-color:  green;
15 }
```

# Selectores anidados

Por tanto, usar "parent selector" es muy útil en casos donde tenemos una pseudoclase y pseudoelementos como son los enlaces

otroEstilos.scss x

```
1  a{
2      color: #999;  &:hover{
3      color: #777;
4      }
5      &:active{
6      color: #888;
7      }
8  }
```

# Selectores anidados

# otroEstilos.css x

```
1  /* line 1, ../sass/otroEstilos.scss */
2  a {
3    color:  #999;
4  }
5  /* line 2, ../sass/otroEstilos.scss */
6  a:hover {
7    color:  #777;
8  }
9  /* line 5, ../sass/otroEstilos.scss */
10 a:active {
11   color:  #888;
12 }
```

# Selectores anidados

Los selectores pueden ser también añadidos antes de la referencia del &. Por ejemplo:

otroEstilos.scss x

```
1  .sidebar{
2      float: right; width: 300px;
3      .users & {
4          //El & referencia a sidebar
5          width: 400px;
6      }
7  }
```

# Selectores anidados

otroEstilos.scss x

```
1  .sidebar{
2      float: right; width: 300px;
3      .users & {
4          //El & referencia a sidebar
5          width: 400px;
6      }
7  }
```

# Selectores anidados

- Su CSS

# otroEstilos.css x

```
1  /* line 1, ../sass/otroEstilos.scss */
2  .sidebar {
3      float: right;
4      width: 300px;
5  }
6  /* line 3, ../sass/otroEstilos.scss */
7  .users .sidebar {
8      width: 400px;
9  }
```

# Selectores anidados

Como podemos ver nesting es bastante sencillo, pero es peligroso.

```
otroEstilos.scss x
1  .content{
2      background: #ccc;
3      .cell{
4          h2{
5              a{
6                  &:hover{
7                      color: red;
8                  }
9              }
10         }
11     }
12 }
```



# Selectores anidados

# otroEstilos.css x

```
1  /* line 1, ../sass/otroEstilos.scss */
2  .content {
3    background: ■ #ccc;
4  }
5  /* line 6, ../sass/otroEstilos.scss */
6  .content .cell h2 a:hover {
7    // Es peligroso debido al nivel de especificación que llega
8    // ya que luego es muy difícil de ser sobrescrito.
9    color: red;
10 }
```

Por tanto, como consejo, intenta limitar tu nesting en 3 o 4 niveles y considera refactorizar tu código.

# ¿Qué pre- procesadores existen?



**Sass** apareció en el **2006**, y ya son muchos los grandes proyectos que han sido basados en Sass, como otros proyectos similares como **Less** y **Stylus**, así como una gran cantidad de frameworks y herramientas que son posibles gracias a estos.

# ¿Por qué usar Sass?

- 1. Nos permite organizar nuestros ficheros CSS y hacerlos más modulares*
  - Proporciona lógica: Si el tipo de letra es Arial, muéstrala en negro, si no en **verde**
  - Variables, reglas CSS anidadas
  - Importación de hojas de estilos
  - Es 100% compatible con CSS3
  - Cálculo matemático
  -

# ¿Por qué usar Sass?

- Mixins: Reutilización de código
  - Incluye un gran número de funciones para manipular colores, etc. Permite el uso de elementos básicos de programación como las directivas de control y las librerías.
  - Herencia
  - Nesting
2. Puedes crear varios archivos scss y luego compilarlos y linkarlos todos como si fuera un único css
3. Es muy fácil de entender y aplicar para cualquiera que sepa CSS.

# ¿Qué puede hacer Sass por mí?

1. Facilita la reusabilidad
2. Genera CSS optimizado
3. Compila los ficheros sass/scss y crea CSS legible por el navegador
4. Otorga flexibilidad (variables)
5. Facilita el mantenimiento en proyectos grandes

# Instalación de Sass

Ver anexo a unidad 1

- Instalación de Sass de Rosa Maria Medina, profesora del CEFIRE

# Declaración y uso de variables

En Sass el nombre de una variable comienza con \$ seguido del nombre, por ejemplo:


```
$base: #777;
```

Por tanto, aquí tenemos una variable definida y con un valor asignado, en este caso es un color. Por tanto, con nuestro código de Sass, podemos llamar a \$base simplemente con su nombre en cualquier punto de nuestra hoja de estilos, de forma que el nombre de nuestra variable en el código será sustituido por el valor

# Declaración y uso de variables

- Variable \$base

 otroEstilos.scss x

```
1  $base:  #777;  
2  .sidebar{  
3      border: 1px solid $base;  
4      p{  
5          color: $base;  
6      }  
7  }
```



# Declaración y uso de variables

- Fichero compilado

# otroEstilos.css x

```
1  /* line 2, ../sass/otroEstilos.scss */
2  .sidebar {
3      border: 1px solid #777;
4  }
5  /* line 4, ../sass/otroEstilos.scss */
6  .sidebar p {
7      color: #777;
8  }
```

# Declaración y uso de variables

Al compilar el siguiente archivo, el valor que tiene content es 'Sobre mi', ya que la variable \$titulo fue sobrescrita pasando de tener el valor de 'Mi blog' por 'Sobre mi'

```
1  $titulo: 'Mi blog';  
2  $titulo: 'Sobre mi';  
3  
4  h2:before{  
5    content: $titulo;  
6  }
```

# Declaración y uso de variables

- Ahí aparece sobrescrita

# otroEstilos.css x

```
1  /* line 4, ../sass/otroEstilos.scss */
2  h2:before {
3      content: "Sobre mi";
4  }
```

# Declaración y uso de variables

Sin embargo, si la segunda variable la hubiéramos definido así:



```
otroEstilos.scss x
1 $titulo: 'Mi blog';
2 $titulo: 'Sobre mi' !default;
3
4 h2:before{
5   content: $titulo;
6 }
```

lo que estaría haciendo sería comprobar si la variable tiene valor previo, y en el caso de no tenerlo le daría el valor 'Sobre mi'.

# Declaración y uso de variables

- Por lo tanto se queda con el valor anterior

# otroEstilos.css x

```
1  /* line 4, ../sass/otroEstilos.scss */
2  h2:before {
3      content: "Mi blog";
4  }
```

# Declaración y uso de variables

Si por ejemplo estamos utilizando una definición modular en Sass como puede ser un archivo para dar estilos a los botones (un partial), si un valor no es definido previamente y tiene la característica de !default, éste será usado por defecto. El ejemplo de lo dicho aparece en la diapositiva siguiente.

# Declaración y uso de variables

\_buttons.scss x

```
1 $rounded: 3px !default;
2 // Si no está definido en otro lado será usado por defecto
3 .btn-a{
4     border-radius: $rounded;  color: #777;
5 };
6 .btn-b{
7     border-radius: $rounded;  color: #222;
8 }
```

# Tipos de variables

Hay una gran cantidad de diferentes tipos de valores, podemos almacenar variables incluyendo booleanos como cierto/falso, números con o sin unidades, colores, strings, listas y nulos!

## Booleanos

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



# Tipos de variables

**Números** (podemos establecerlos con o sin unidades)

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

## Colores

```
$base: purple;  
$border: rgba(0, 255, 0, 0.5);  
$shadow: #333;
```

# Tipos de variables

## Strings (con o sin comillas)

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

## Listas

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

## Null

```
$shadow: null;
```

# Bibliografía y/o páginas de interés

- Sass: <http://sass-lang.com/>
- Ruby: <https://www.ruby-lang.org/en/>
- Sass Wikipedia:  
[https://en.wikipedia.org/wiki/Sass \(stylesheet language\)](https://en.wikipedia.org/wiki/Sass_(stylesheet_language))
- Less wikipedia:  
[https://en.wikipedia.org/wiki/Less \(stylesheet language\)](https://en.wikipedia.org/wiki/Less_(stylesheet_language))
- Stylus wikipedia:  
[https://en.wikipedia.org/wiki/Stylus \(stylesheet language\)](https://en.wikipedia.org/wiki/Stylus_(stylesheet_language))
- compass: <http://compass.kkbox.com/>
- scout: <http://mhs.github.io/scout-app/>
- Koala: <http://koala-app.com/>