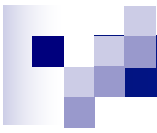




UD4 Bootstrap y SASS.

DISEÑO DE INTERFACES WEB



4.1 Introducción

INSTALACIÓN DE SASS, CREACIÓN DE PROYECTOS, FICHEROS DE CONFIGURACIÓN, SINTAXIS DE SASS, VARIABLES CON SASS

Contenidos

1. Introducción
2. ¿Qué es un precompilador de CSS?
3. ¿Qué es SASS?
4. ¿Qué preprocesadores existen?
5. ¿Por qué usar SASS?
6. ¿Qué puede hacer SASS por mí?
7. Cómo instalar SASS
8. Primeros pasos
9. Declaración y uso
10. Tipos de variables

1. 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 de este curso, vamos a estudiar un precompilador/preprocesador de CSS, en concreto Sass.

2. ¿Qué es un precompilador de CSS?

Un precompilador de CSS no es más que una herramienta que nos permite crear scripts para posteriormente ser convertidos a CSS “real”. Estos scripts estarán formados por variables, condiciones, bucles o funciones, es decir, es un lenguaje de programación que nos permite obtener/generar hojas de estilo CSS.

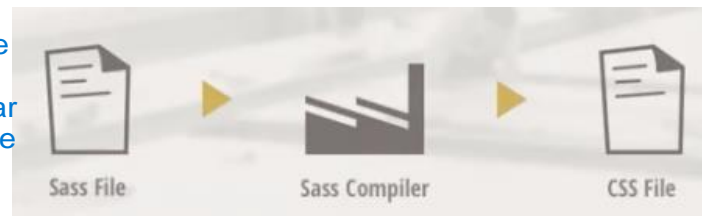
Por tanto, Sass no es más que el acrónimo de Syntactically Awesome Style Sheets. Se suele decir que es **¡CSS con superpoderes!**

3. ¿Qué es Sass?

CSS fue diseñado para ser simple, pero a medida que nuestra aplicación web se incrementa puede ser difícil ir cambiando colores, fuentes y el resto de propiedades. También, nos encontramos con otros problemas como puede ser el **evitar repetir ciertas propiedades** dentro de nuestra hoja de estilos.

Por tanto, para evitar estos problemas Sass es bastante práctico. Hoy en día, **Sass es un preprocesador** como puede ser CoffeeScript o Haml.

CoffeeScript es un lenguaje de programación que se compila a JavaScript. Añade azúcar sintáctico inspirado en Ruby, Python y Haskell para mejorar la brevedad y la legibilidad de JavaScript, y añade características más sofisticadas,



Haml (HTML Abstraction Markup Language) es un lenguaje de marcado ligero que se usa para describir el XHTML de un documento web sin emplear el código embebido tradicional

Sass fue creado por Hampton Catlin, el cuál es el mismo que creó Haml, que es un generador de plantillas para HTML.

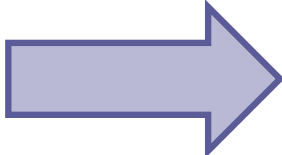
3. ¿Qué es Sass?

Sassy CSS (.scss) es la extensión por defecto de los archivos que posteriormente compilaremos con Sass.

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

lighten function take two parameters, a color and a percentage, it will attempt to lighten the provided color by the percentage given

&: nesting que hace referencia al selector padre



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

Ejemplo de código .scss

De forma que cuando lo compilamos obtenemos:

Los comentarios

Como curiosidad, cuando comentamos nuestra hoja de estilo .scss, **podemos usar tanto // como /* */**, sin embargo, cuando la compilamos, sólo aparecen los comentarios que han sido introducidos con /* */, aquellos que tenían // desaparecen en el css que ha sido generado por el compilador. Por ejemplo:

Archivo: estilo.scss

```
// Este comentario
// dejará de aparecer
// al compilarlo
/* Este comentario sí aparecerá */
```

Archivo: estilo.css

```
/* Este comentario sí aparecerá */
```

@import todavía funciona, pero se recomienda @use y @forward
@use permite importar módulos de forma más eficiente, evitando la duplicación de código

Antes (con @import):

```
@import "variables";
@import "mixins";
@import "buttons";
```

Después (con @use y @forward):

```
// _variables.scss
$primary-color: blue;

// _mixins.scss
@mixin button-style {
  background-color: $primary-color;
  border-radius: 5px;
}
```

```
// _buttons.scss
@use "variables";
@use "mixins";
```

```
.btn { @include mixins.button-style; }
```

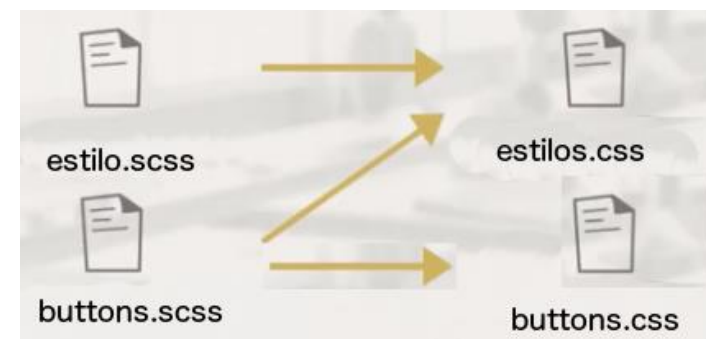
```
// styles.scss
@use "buttons";
```


La etiqueta @import

La etiqueta @import en CSS está desaconseja ya que se va descargando poco a poco los archivos CSS importados, sin embargo, cuando usamos @import en Sass, es bastante diferente, ya que **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, cuando usamos @import en Sass no tenemos que incluir la extensión del archivo.

Archivo: estilo.scss

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



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

La 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** o lo que se llama **partials**, es decir:

- Si quiero que cree el archivo buttons.css, lo llamaré buttons.scss
- Si no quiero que cree el archivo buttons.css, lo llamaré `_buttons.scss`



Y a la hora de especificar que se va a importar un archivo “parcial”, es decir, el `_buttons.scss`, se realiza del mismo modo: `@import "buttons";`

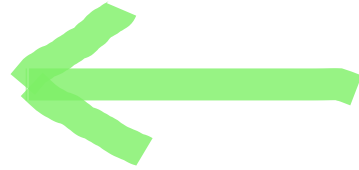
Es decir, **no ponemos el subrayado** en este caso dentro del valor del nombre del archivo a importar.

¿Qué son los **selectores anidados** en Sass?

Cuando generamos un archivo CSS como el siguiente:

Archivo estilo.css

```
.content {  
  border: 1px solid #ccc;  
  padding: 20px;  
}  
.content h2 {  
  font-size: 3em;  
  margin: 20px 0;  
}  
.content p {  
  font-size: 1.5em;  
  margin: 15px 0;  
}
```



Archivo estilo.scss

```
.content{  
  border: 1px solid #ccc;  
  padding: 20px;  
  h2 {  
    font-size: 3em;  
    margin: 20px 0;  
  }  
  p {  
    font-size: 1.5em;  
    margin: 15px 0;  
  }  
}
```

De modo, que dentro de la **indentación de la clase .content**, se han establecido los estilos para los párrafos y h2 que estén dentro de un elemento con la clase .content.

Otra de las posibilidades que tenemos de **anidamiento** (nesting) es: Nestear propiedades

Archivo button.scss

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

Archivo estilo.css

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

En este caso, tenemos nuestro text-decoration y text-decoration: underline para la clase .btn

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

El archivo estilo.scss

```
.content{
  border: 1px solid #ccc;
  padding: 20px;
  .callout{
    border-color: red;
  }
  &.callout{
    /* referencia a .content,
    es un selector compuesto:
    .content.callout */
    border-color: green;
  }
}
```

El archivo estilo.css

```
.content{
  border: 1px solid #ccc;
  padding: 20px;
}
.content .callout{
  border-color: red;
}
.content.callout{
  border-color: green;
}
```

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

Archivo estilo.scss

```
a {
  color: #999;
  &:hover {
    color: #777;
  }
  &:active {
    color: #888;
  }
}
```

Archivo estilo.css

```
a {
  color: #999;
}
a:hover {
  color: #777;
}
a:active {
  color: #888;
}
```

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

Archivo estilo.scss

```
.content {  
  background: #ccc;  
  .cell {  
    h2 {  
      a {  
        &:hover{  
          color: red;  
        }  
      }  
    }  
  }  
}
```

Archivo estilo.css

```
.content{  
  background: #ccc;  
}  
.content .cell h2 a:hover{  
  /* Es peligroso debido al nivel  
  de especificación que llega, ya  
  que luego es muy difícil de ser  
  sobrescrito. */  
  color: red;  
}
```

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

4. ¿Qué preprocesadores existen?

Sass apareció en el 2006, y 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.



COMPARATIVA:

<https://www.linkedin.com/pulse/comparaci%C3%B3n-de-sass-less-y-stylus-carolina-romero/?originalSubdomain=es>

5. ¿Por qué usar Sass?

1. Permite **organizar** ficheros CSS y hacerlos **más modulares**
 - Proporciona lógica. Ejemplo: Si el tipo de letra es Arial, muéstrala en negro, si no en verde
 - Variables, reglas CSS anidadas
 - Importación de hojas de estilos o Es 100% compatible con CSS3
 - Cálculo matemático
 - Mixins: Reutilización de código
 - Incluye un gran numero 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 todos como si fuera un **único css**.
3. Muy **fácil** de entender y aplicar para cualquiera que sepa CSS.

6. ¿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

7. ¿Cómo instalar Sass?

En la web de Sass (<http://sass-lang.com/install>), podemos encontrar las **dos formas** que hay para poder utilizar Sass en nuestro equipo, usando una aplicación grafica o mediante línea de comandos. Debemos tener en cuenta que antes de instalar Sass ~~debemos instalar Ruby~~, pero esto lo explicaremos dependiendo de si utilizaremos Sass desde consola o por medio de una aplicación y el sistema operativo que tengáis:

1. **Si queremos usarla en modo gráfico:** Como podemos ver, hay una gran variedad de aplicaciones gráficas. Antes se veía Compass.app, principalmente porque es la aplicación que crearon los propietarios de Sass, pero a día de hoy la tienen abandonada y la versión de Sass que tiene Compass es muy antigua.

7. ¿Cómo instalar Sass?

- a. CodeKit (Paid)
- b. ~~Compass.app (Paid, Open Source)~~
- c. Ghostlab (Paid)
- d. Hammer (Paid)
- e. Koala (Open Source)
- f. LiveReload (Paid, OpenSource)
- g. Prepros (Paid)
- h. Scout (Open Source)

Independientemente del sistema operativo, si tenemos Node instalado, podemos instalar Sass desde el terminal tecleando:

```
npm install -g sass
```

Instalación de Sass en MacOS

Con [Homebrew](#) previamente instado, ejecutamos en terminal:

```
brew install sass/sass/sass
```

o directamente añadiendo el siguiente paquete al PATH:
<https://github.com/sass/dart-sass/releases/tag/1.27.0>

Para añadirlo al PATH, haremos:

1. Con el editor de texto que queramos, abrimos el archivo oculto `.bash_profile` (ruta: `/Users/nuestro-usuario/.bash_profile`)
2. Añadimos `export PATH="your-dir:$PATH"` en la última línea del archivo, donde `your-dir` es el directorio que quieres añadir, es decir, donde tenemos descargado y descomprimido el paquete
3. Guardamos el archivo, y reiniciamos el terminal.

Instalación de Sass en Windows

Tenemos dos opciones, o bien instalarlo usando el administrador de paquetes [chocolatey](#) y ejecutando el comando:

```
choco install sass
```

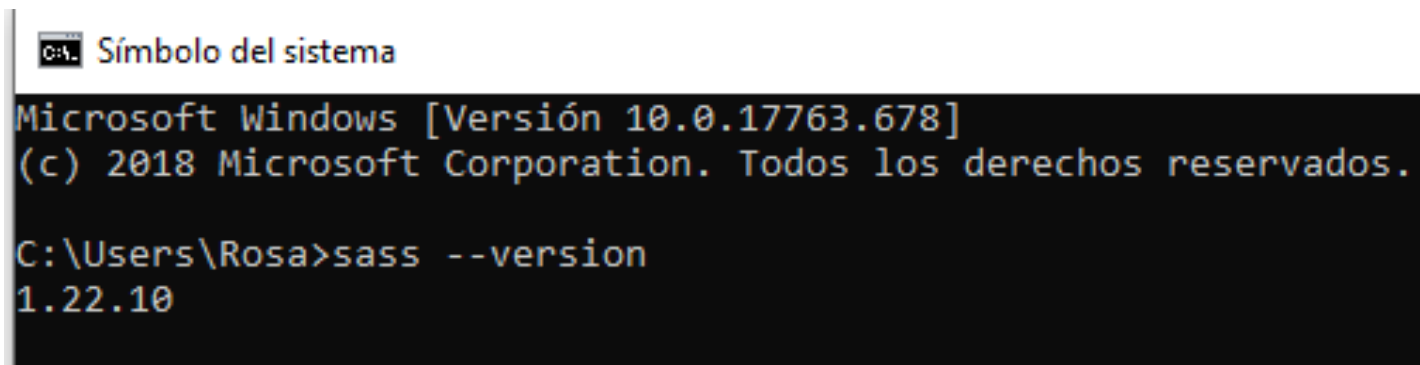
o bien añadiendo el siguiente paquete a nuestro PATH: <https://github.com/sass/dart-sass/releases/tag/1.27.0>, para ello:

1. Dependiendo de la versión:
 - ☐ Windows 7: botón derecho sobre mi pc → propiedades
 - ☐ Windows 8 o 10: Panel de control → sistema y seguridad → Sistema
2. Seleccionamos Configuración avanzada del Sistema
3. Opciones avanzadas → variables de entorno...
4. En variables del Sistema seleccionamos PATH y pulsamos sobre editar, en caso de no tener la variable PATH, hacemos clic en nueva.
5. Añadimos el directorio al principio en donde se encuentra sass concatenando con el valor existente con un punto y coma (ver ejemplo de la siguiente diapositiva).

Instalación de Sass en Windows

por ejemplo, si mi path contenía: C:\WINDOWS\system32 lo he cambiado por: C:\Users\Rosa\Desktop\dart-sass; C:\WINDOWS\system32

6. Comprobamos que se ha instalado correctamente. Abrimos el terminal (cmd) y tecleamos `sass --version`

A screenshot of a Windows Command Prompt window. The title bar reads 'C:\> Símbolo del sistema'. The window content shows the following text: 'Microsoft Windows [Versión 10.0.17763.678]', '(c) 2018 Microsoft Corporation. Todos los derechos reservados.', 'C:\Users\Rosa>sass --version', and '1.22.10'.

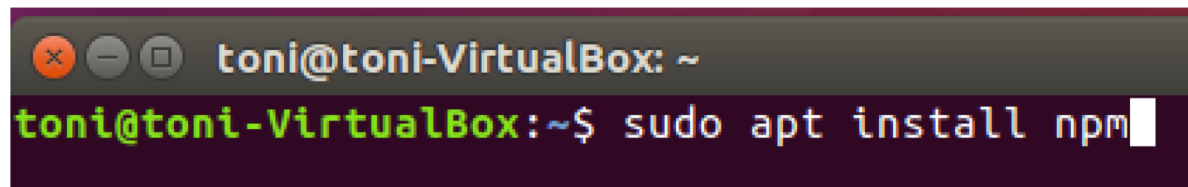
```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.17763.678]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

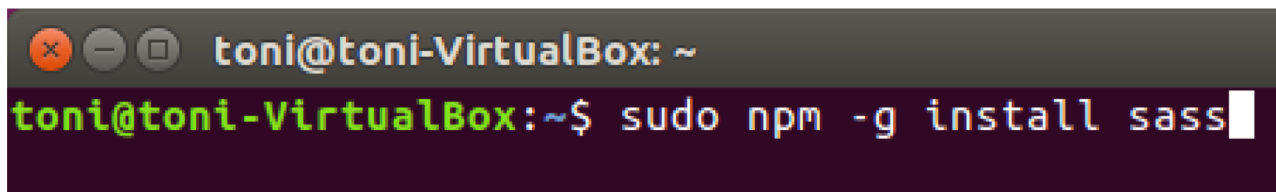
C:\Users\Rosa>sass --version
1.22.10
```

Instalación de Sass en Linux

- Instalar npm consola: `sudo apt install npm`
- Instalar Sass: `sudo npm -g install sass`



```
toni@toni-VirtualBox: ~  
toni@toni-VirtualBox:~$ sudo apt install npm
```



```
toni@toni-VirtualBox: ~  
toni@toni-VirtualBox:~$ sudo npm -g install sass
```

npm (Node Package Manager) es el administrador de paquetes de Node.js
- instalar, compartir y gestionar paquetes (bibliotecas y herramientas) en JavaScript

Cómo crear nuestro primer proyecto

Una vez que hemos instalado Sass en nuestro ordenador, pasaremos a crear nuestro **primer proyecto de prueba**. En primer lugar para mantener una estructura ordenada y limpia de nuestro proyecto nos situaremos en el directorio que queramos crear el proyecto y tecleamos en la consola lo siguiente o bien creamos la estructura manualmente:

En Windows: `mkdir proprueba\sass proprueba\stylesheets`

En Linux o MacOS: `mkdir -p proprueba/{sass,stylesheets}`

Y en el directorio de sass nos **creamos un archivo** llamado `style.scss` que será el archivo de ejemplo con el que veremos cómo funciona y se compila Sass.

8. Primeros pasos `sudo snap install --classic code`

Una vez creado el archivo (style.scss), podéis usar el editor de texto que queráis, Sublime, Visual Studio Code, Atom, etc. Yo usaré **Visual Studio Code** por lo bien que se adapta a Sass y la ayuda que da sobre todo a la hora de mostrar los errores.

Antes de editar el archivo style.scss, vamos a **activar el visor** para que cada cambio que generemos (cada vez que guardemos) nos compile y nos genere su correspondiente CSS:

Windows: `sass --watch sass\style.scss:stylesheets\style.css`

Linux/MacOS: `sass --watch sass/style.scss:stylesheets/style.css`

El primer parámetro es el **archivo origen** (en su respectivo directorio) y el segundo es el **fichero destino** (en su respectivo directorio de destino) donde creará el CSS, el terminal se quedará congelado hasta que pulsemos **Ctrl+C**.

```
MBP-de-Antonio:proprueba Toni$ sass --watch sass/style.scss:stylesheets/style.css
Compiled sass/style.scss to stylesheets/style.css.
```

UD 4 [Sass is watching for changes. Press Ctrl-C to stop.

8. Primeros pasos

Sass en Visual Studio Code

<https://www.youtube.com/watch?v=x-lZql468A4>

Ahora con el editor **SIN CERRAR NI CANCELAR** la ejecución del terminal, vamos al archivo que hemos creado (style.scss), y pegamos:

```
$variable: 10px;
body{
    padding: $variable;
}
```

Al haberle dado a guardar al archivo, podemos observar que el terminal ha cambiado, nos indica que los cambios que ha detectado en style.scss han sido aplicados a style.css y style.css.map:

```
MBP-de-Antonio:proprueba Toni$ sass --watch sass/style.scss:stylesheets/style.css
Compiled sass/style.scss to stylesheets/style.css.
[Sass is watching for changes. Press Ctrl-C to stop.]
[
  Compiled sass/style.scss to stylesheets/style.css.
]
```

9. Declaración y uso

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

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

De este modo, 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, será sustituido por el valor que le habíamos dado previamente cuando lo compilemos y generemos el fichero css.

aplicación.scss

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

Obtenemos aplicación.css

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

9. Declaración y uso

Normalmente si definimos una variable, **puede ser utilizada posteriormente** modificando el valor que le habíamos dado antes, de forma que el valor anterior es sobrescrito. Por ejemplo:

aplicación.scss

```
$titulo: 'Mi blog';  
$titulo: 'Sobre mi';  
h2:before{  
    content: $titulo;  
}
```

Al compilar el anterior 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'.

Obtenemos aplicación.scss

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

9. Declaración y uso

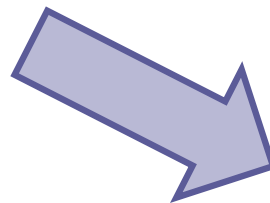
Sin embargo, si la segunda variable la hubiéramos definido así:

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

lo que estaría haciendo sería comprobar si la variable \$titulo tiene un **valor previo, y en el caso de no tenerlo** le daría el valor 'Sobre mi'. Por tanto, el ejemplo anterior:

aplicación.scss

```
$titulo: 'Mi blog';  
$titulo: 'Sobre mi' !default;  
h2:before{  
    content: $titulo;  
}
```



aplicación.css

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

10. 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;
```

Números (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;
```

Strings (con o sin comillas)

```
$header: 'Helvetica Neue';
```

```
$callout: Arial;
```

```
$message: "Loading...";
```

Listas

```
$authors: Rosa, Pablo, Lucia;
```

```
$margin: 40px 0 20px 100px;
```

Null

```
$shadow: null;
```