

*PREPROCESADORES CSS*

Un preprocesador de CSS es una **herramienta de desarrollo** que nos permite **escribir *pseudocódigo CSS.*** Este **luego** será **compilado** para **convertirse en CSS** tal y como lo conocemos. Está formado por **variables, condicionales, bucles o funciones,** elementos habituales de cualquier **lenguaje de programación**. Por este motivo, podríamos decir que tenemos un **lenguaje de programación cuya misión es la de generar el código CSS**.

El circuito de trabajo con los preprocesadores de CSS está formado por 2 pasos. En primer lugar se crean los estilos utilizando la sintaxis del preprocesador. En el segundo paso se compila el archivo para generar el CSS definitivo.

El objetivo de los preprocesadores es el de disponer de un código más limpio y sencillo de mantener y editar.

*SASS*



Sass significa “Syntactically Awesome Stylesheets”, es una herramienta escrita en lenguaje Ruby, que nos permite crear hojas de estilos estructuradas, limpias y fáciles de mantener.

Con SASS vamos a poder escribir hojas de estilo que nos ayudarán a generar ficheros CSS más optimizados, incorporando mayor contenido semántico y permitiendo utilizar funcionalidades que normalmente encontraríamos en lenguajes de programación tradicionales, como el uso de variables, creación de funciones, etc.

**¿Por qué es útil?**

Normalmente crear una hoja de estilos es relativamente sencillo. Lo malo es cuando el proyecto va creciendo en tamaño: su CSS puede acabar siendo muy extenso.

Sass nos permite una sintaxis más simple, más elegante, implementando además bastantes características extras para hacer más manejable nuestra hoja de estilos.

**Sintaxis**

En Sass contamos con dos diferentes tipos de sintaxis: scss y sass.

La **primera** y más popular es conocida como **SCSS** (Sassy CSS), es muy similar a la sintaxis nativa de CSS, tanto así que nos permite importar hojas de estilos CSS (copiar y pegar) directamente en un archivo SCSS y obtener un resultado válido.

Para utilizarla solo debemos crear un archivo con terminación .scss de la siguiente manera: *archivo.scss*

La **segunda** opción es conocida como **Indented Syntax** (sintaxis de indentación). Utiliza la indentación en lugar de corchetes para expresar el anidamiento de selectores y saltos de línea en lugar de (;) punto y coma para separar las diferentes propiedades que se declaren. Usarla también es muy sencillo: creamos un archivo con terminación .sass de la siguiente manera: *archivo.sass*

# NodeJS

Node.js es un entorno en tiempo de ejecución JavaScript, de código abierto y multiplataforma. Funciona para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript. Es asíncrono, con I/O (entrada y salida) de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. Desarrollado por Ryan Dahl en 2009, y su evolución está apadrinada por la empresa Joyent.

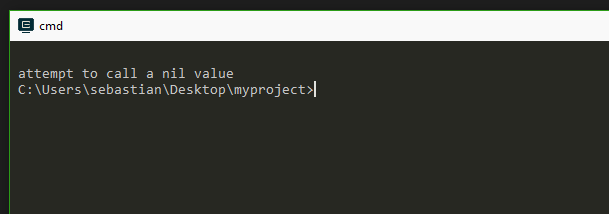
## **Instalar NodeJS**

Para compilar Sass vía la línea de comandos, primero necesitamos instalar NodeJS. Descárgalo del sitio oficial [nodejs.org](https://nodejs.org/), abre el paquete y sigue el asistente de instalación.

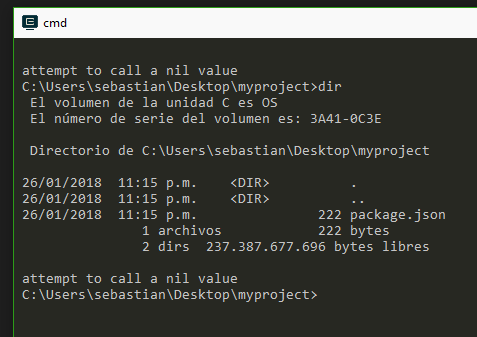
## **Inicializar NPM**

[NPM](https://www.npmjs.com/) es el Gestor de Paquetes de Node para JavaScript, que facilita instalar y desinstalar paquetes de terceros. Para inicializar un proyecto en Sass con NPM, abre tu terminal y CD (cambia de directorio) a tu directorio de proyecto.

*(Fuente:* [*http://npmjs.org/*](http://npmjs.org/) *)*



Una vez en el directorio correcto, ejecuta el comando *npm init*. Se te pedirá responder varias preguntas sobre el proyecto, después de las cuales NPM generará un archivo package.json en tu directorio.  
  
$ npm init



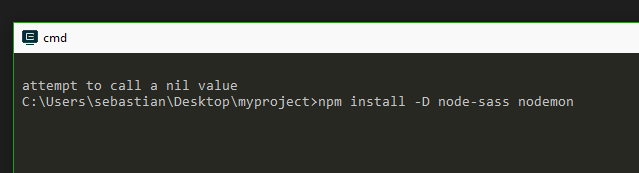
## **Instalar node-sass y nodemon**

Noda-sass se usará para compilar tus archivos *scss* en archivos *css*.

Nodemon es algo que usaremos para ver los cambios en nuestros archivos *scss*. Normalmente, se usa para detectar cambios en el código Node.js del lado del servidor.

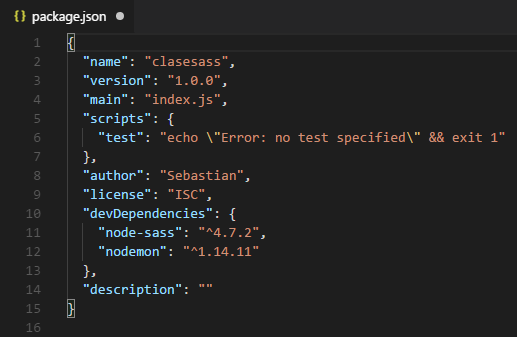
Para instalar nuestras dependencias (node-sass y nodemon) ejecutaremos el siguiente comando en la terminal:

$ npm install -D node-sass nodemon

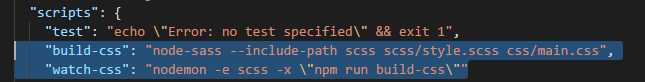


## **Comando para compilar**

Todo está listo para escribir un pequeño script para compilar Sass. Abre el archivo *package.json* en un editor de código. Verás algo como ésto:



En la sección de **scripts** añade un comando **scss** debajo del comando **test**, como se muestra abajo:



“build-css”: “node-sass --include-path scss scss/style.scss css/main.css”,

“watch-css”: “nodemon -e scss -x \”npm run build-css\””

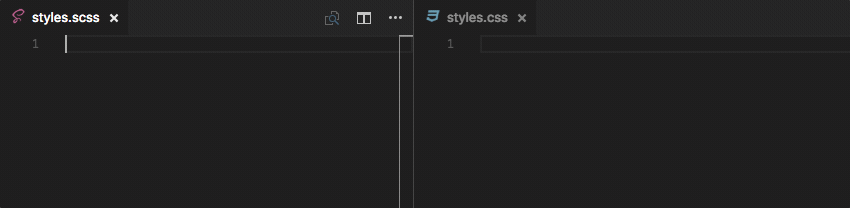
Lo que indicamos en “build-css” es dónde encontrará el documento SCSS de entrada y dónde se depositará el documento CSS de salida. Lo que indica “watch-css” es donde vamos a permanecer observando los cambios sobre la última acción de compilación.

Entonces cuando lo **ejecutamos,** este script **vigilará cada archivo scss** en el directorio indicado, luego guarda el css compilado en el directorio css/ cada vez que cambiamos un archivo scss.

Para ejecutar nuestro script de una línea, necesitamos ejecutar el siguiente comando en la terminal:

$ npm run watch-css

¡Y *voila*! Estamos vigilando y compilando SASS.



Más información acerca de NODEJS y NPM:

<https://nodejs.org/en/about/>

<https://devcode.la/blog/que-es-npm/>

<https://www.w3schools.com/nodejs/nodejs_npm.asp>

<https://www.sitepoint.com/beginners-guide-node-package-manager/>

*Nesting, Import y Vars*



# Nesting o Anidación

HTML sigue una estricta estructura de anidación, mientras que CSS por lo general, es un caos total. Con **la anidación de Sass puedes organizar tu hoja de estilo de una manera que se asemeja a la de HTML**, lo que reduce la posibilidad de conflictos en el CSS.

Echa un vistazo a este ejemplo rápido, en el que definimos una lista que contiene una serie de enlaces:

**SCSS**

| ul {  list-style: none;  li {  padding: 15px;  display: inline-block;  a {  text-decoration: none;  font-size: 16px;  color: #444;  }  }  } |
| --- |
|

**CSS**

| ul {  list-style: none;  }  ul li {  padding: 15px;  display: inline-block;  }  ul li a {  text-decoration: none;  font-size: 16px;  color: #444;  } |
| --- |
|

# Import

Una de las características más útiles de Sass es poder **separar tus hojas de estilo en archivos separados**. A continuación, puedes usar @import para incluir la fuente de tus archivos individuales en una hoja de estilo maestra.

¿Pero cómo debes estructurar tus proyectos Sass? ¿Hay una forma estándar de separar tus archivos CSS?

**Estructura de directorio básica**

| stylesheets/ | |-- modules/ # Modulos Comunes | |-- \_all.scss # Todos los modules | |-- \_utility.scss # Utilitarios | |-- \_colors.scss # Etc... | ... | |-- partials/  | |-- \_base.sass # imports for all mixins + global project variables | |-- \_buttons.scss # buttons | |-- \_figures.scss # figures | |-- \_grids.scss # grids | |-- \_typography.scss # typography | |-- \_reset.scss # reset | ... | |-- vendor/ # CSS o Sass de otros plugins | |-- \_colorpicker.scss | |-- \_jquery.ui.core.scss | ... | `-- main.scss # Archivo Sass principal |
| --- |

Esto nos permite mantener nuestro archivo Sass principal extremadamente limpio, importando código desde otros archivos Scss o CSS:

| // Modules and Variables  @import "partials/base";  // Partials  @import "partials/reset";  @import "partials/typography";  @import "partials/buttons";  @import "partials/figures";  @import "partials/grids";  // ...  // Third-party  @import "vendor/colorpicker";  @import "vendor/jquery.ui.core"; |
| --- |
|

# Vars (variables)

Las variables son una manera de guardar información que necesites reutilizar en tus hojas de estilos: colores, dimensiones, fuentes o cualquier otro valor. Sass utiliza el símbolo dólar ($) al principio de la palabra clave para crear una variable.

Estas variables se comportan como atributos CSS, y su valor puede ser cualquiera que pudiera adquirir un atributo CSS.

A continuación puedes ver un sencillo ejemplo. Te mostramos tanto el código SCSS, como el CSS.

**SCSS**

| $title-font: normal 24px/1.5 'Open Sans', sans-serif;  $cool-red: #F44336;  $box-shadow-bottom-only: 0 2px 1px 0 rgba(0, 0, 0, 0.2);  h1.title {  font: $title-font;  color: $cool-red;  }  div.container {  color: $cool-red;  background: #fff;  width: 100%;  box-shadow: $box-shadow-bottom-only;  } |
| --- |
|

**CSS**

| h1.title {  font: normal 24px/1.5 "Open Sans", sans-serif;  color: #F44336;  }  div.container {  color: #F44336;  background: #fff;  width: 100%;  box-shadow: 0 2px 1px 0 rgba(0, 0, 0, 0.2);  } |
| --- |
|

La idea detrás de todo esto es que luego podamos **volver a utilizar los mismos valores de una manera rápida**, o si es necesario un **cambio**, podemos declarar el **nuevo valor en un solo lugar** (la definición de la variable), en lugar de aplicar el cambio manualmente a todas partes afectadas por la propiedad.

Una variable se podrá definir fuera o dentro de algún selector. Si se define fuera, dicha variable será **global** y podrá utilizarse en cualquier bloque. Si se define dentro de un selector, la variable será **local** y únicamente se podrá utilizar en el selector que la contiene y en sus selectores anidados.

Una **buena práctica** común consiste en **definir todas las variables globales al principio del fichero**, para que puedan localizarse rápidamente.

**Uso de !default en las variables**

Si hacemos:

| $color: #FF0000;  $color: #000000; |
| --- |
|

El color que se usará es el #000000.

Pero si hacemos:

| $color: #333333;  $color: #000000 !default; |
| --- |
|

El color que se usará será #333333. Esta directiva indicará que la asignación que estamos realizando a la variable solo se haga en caso de que dicha variable no se haya definido anteriormente.

Enlaces Relacionados:  
<https://sass-guidelin.es/es/>

<http://sass-lang.com/>

<https://www.ondho.com/que-es-sass-y-por-que-los-css-pueden-volver-a-divertirnos/>

<https://librosweb.es/libro/sass/>

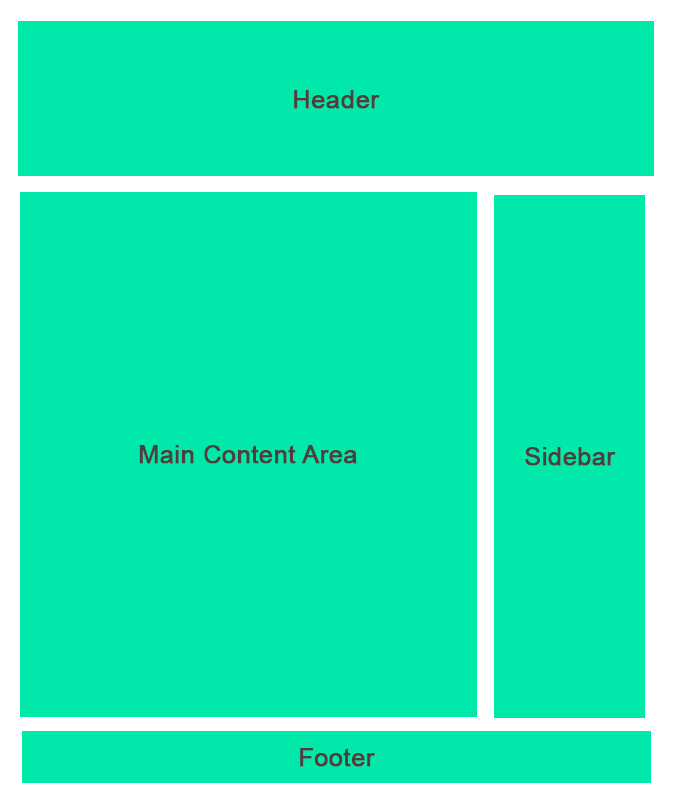
<https://www.codecademy.com/learn/learn-sass>

*BEM*

Todos queremos **hacer que nuestro código sea más fácil de leer**. Esto nos ayuda a trabajar más rápidamente y de manera eficiente, y cuando otros trabajen con nosotros podremos mantener claridad y coherencia. Pareciera que en estos días existen muchos estándares y términos en el mundo del CSS: [OOCSS](http://oocss.org/), [SMACSS](http://webdesign.tutsplus.com/articles/quick-tip-fix-your-messy-css-with-smacss--webdesign-18489), BEM y otros. Todos estos son **acercamientos para mejorar la estructura del CSS**, y hoy vamos a cubrir la metodología BEM.

BEM significa *Modificador de Bloques de Elementos (Block Element Modifier)* por sus siglas en inglés. Sugiere **una manera estructura de nombrar tus clases, basado en las propiedades del elemento en cuestión**. Si alguna vez has visto un nombre de una clase como header\_from-email eso es precisamente BEM en acción. Cuando utilices la metodología BEM, debes tomar en cuenta que solamente usarás nombres de clases (no IDs). Los nombres de clases te permiten repetir el nombre BEM si es necesario, y crear una estructura de código más consistente (en ambos archivos el HTML y CSS/Sass). Ahora vamos a descifrarlo.

# Bloque

El *bloque* es un contenedor o contexto donde el elemento se encuentra presente. Piensa como si fueran partes estructurales de código más grande. Puede que tengas un encabezado, pie de página, una barra lateral y un área de contenido principal; cada uno de estos sería considerado como un bloque. Mira la siguiente imagen:

El bloque de elementos forma la raíz de la clase y siempre irá primero. Solo debes saber que una vez que has definido tu bloque, estarás listo para comenzar a nombrar tus elementos.

| .block\_\_element {  background-color: #FFFFFF;  } |
| --- |
|

El doble guión bajo te permite visualizar, navegar rápidamente y manipular tu código. Aquí hay algunos ejemplos de cómo funciona la metodología de elementos:

| .header\_\_logo {}  .header\_\_tagline {}  .header\_\_searchbar {}  .header\_\_navigation {} |
| --- |
|

Como puedes ver, hay espacio para la creatividad y apropiarte de esta metodología. En el ejemplo, "navigation" puede ser cambiado a "nav"; "tagline" puede cambiarse a "tag" o "tagLine". El punto es **mantener los nombres simples, claros, y precisos**. No lo pienses demasiado, y solo porque tus hojas de estilos y html sean estáticos, no quiere decir que tengas que volver a repetir el mismo código.

Actualizar el nombre de las clases no debería ser un problema cuando encuentras una mejor semántica que funcione (solo debes tratar de ser consistente y apegarte a ella). Los elementos se convertirán en el centro de los nombres de tus clases, y te ayudarán a entender cómo estructurar tus hojas de estilos y cómo manejar tu diseño.

# Modificadores

Cuando nombras una clase, la intención es ayudar a que ese elemento pueda ser repetido para que no tengas que escribir nuevas clases en otras áreas del sitio si los elementos de estilo son los mismos. Cuando necesitas **modificar el estilo de un elemento específico**, puedes usar un modificador. Para lograr esto, añades un doble guión *--* luego del elemento (o bloque). Aquí tenemos un corto ejemplo:

| .block--modifier {}  .block\_\_element--modifier {} |
| --- |
|

Ten cuidado al usar los modificadores, recuerda que se quiere mantener todo más simple y no tener que repetir lo mismo o crear clases extra a menos que sea absolutamente necesario. Lo explicaremos con un código usando el encabezado del sitio como nuestro bloque:

| .header\_\_navigation {}  .header\_\_navigation--secondary {} |
| --- |
|

Si estás usando una segunda navegación, lo más probable es que el diseño y espaciado no cambien, pero puede que la navegación secundaria tenga un color distinto. Puedes duplicar los estilos originales, o mejor aún, usar un pre-procesador. Con Sass, podrías @extender el elemento principal (así el elemento secundario heredará todas las propiedades) y cambiar los estilos apropiados.

| .header\_\_navigation {  background: #008cba;  padding: 1rem 0;  margin: 2rem 0;  text-transform: uppercase;  }  .header\_\_navigation--secondary {  @extend .header\_\_navigation;  background: #dfe0e0;  } |
| --- |
|

Puede que pienses, "¡pero el nombre de la clase es muy largo!". Piénsalo así: Los nombres de las clases BEM son específicos, claros, fáciles de leer dentro del html y comunican claramente para qué existen.

Cuando usamos BEM solo utilizamos un nombre de clase por cada etiqueta html. Fíjate cómo funciona para la etiqueta "label". Selectores estándares:

| .label .label-default {}  .label .label-alert {} |
| --- |
|

vs. selectores BEM:

| .label {}  .label--alert {} |
| --- |
|

Mas información acerca de BEM:

<https://en.bem.info/>

<https://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax/>

<https://coding.smashingmagazine.com/2012/04/a-new-front-end-methodology-bem/>

<https://www.intelygenz.es/como-organizar-css-metodologias-oocss-bem-smacss/>

<https://www.espai.es/blog/2016/07/metodologias-css-oocss-bem-smacss/>