SASS-SCSS

Preprocesadores CSS

¿Qué es un preprocesador?

Un preprocesador es una **tecnología que toma un lenguaje que está orientado a compilars**e o a transformarse y **dar como resultado un archivo en otro lenguaje más simple**.

Esto en la práctica se traduce en que tenemos un lenguaje que produce otro que es más pesado de escribir.

En **caso de CSS tenemos SASS, LESS y Stylus** y alguno más.

En el caso de **este módulo trabajaremos con SCSS que es una sintaxis de SASS**, pero dejaremos algunos links para aprender los otros tipos de preprocesadores que son casi idénticos.

¿Para qué se usa?

En el proceso de desarrollo de CSS hay un **montón de código que tenemos que estar escribiendo una y otra vez**, y que llega un momento en el que consumimos bastante tiempo para poder entrar a hacer cosas más interesantes.

Es decir el **uso de la técnica nos impide avanzar en el proceso de diseño** y por lo tanto necesitamos métodos de desarrollo más agiles.

Entonces **un preprocesador CSS es simplemente un código más sencillo que el CSS, que nos produce CSS de una manera ágil**.

Workflow de desarrollo con SASS

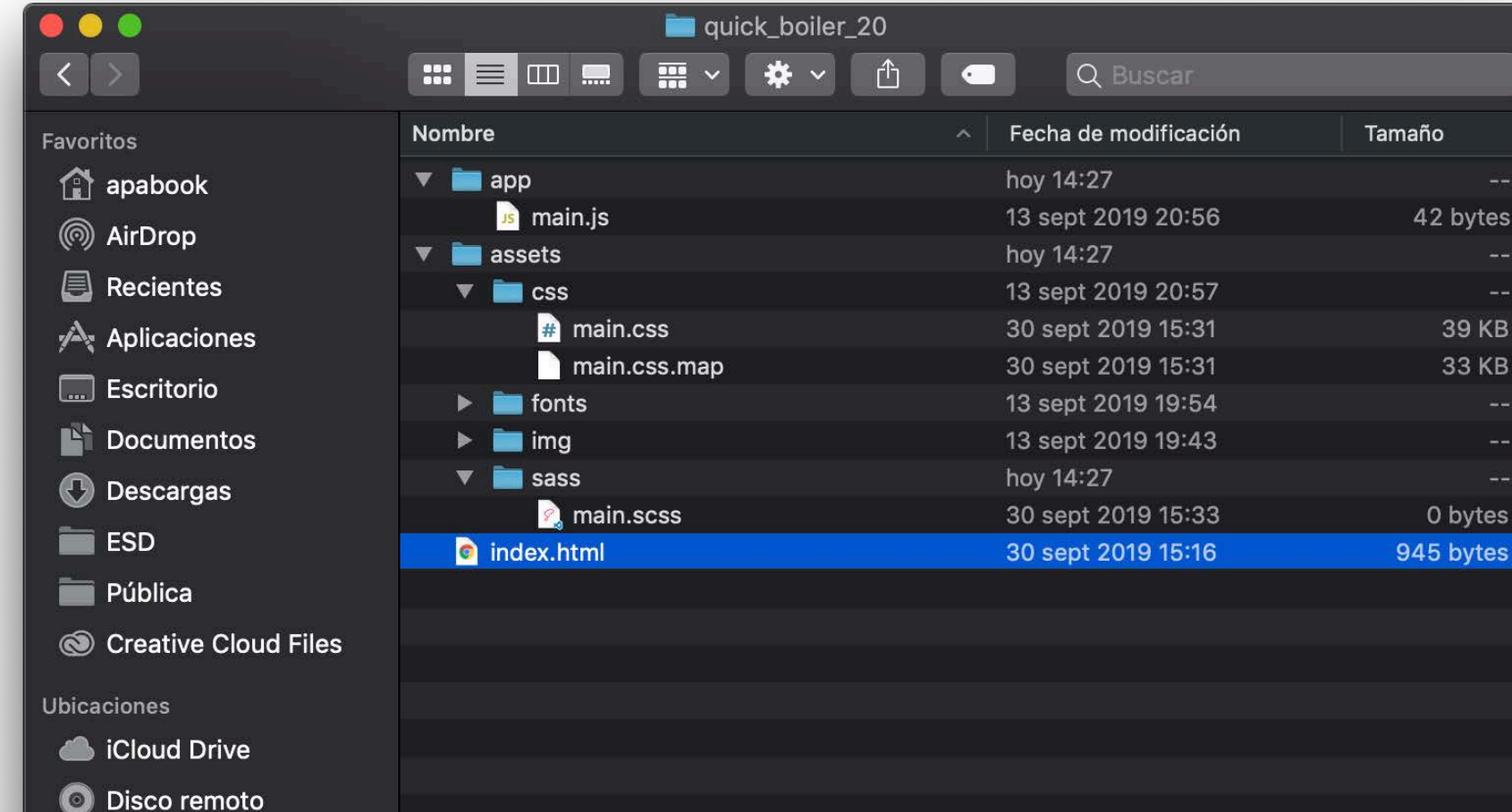
El SASS no se interpreta en el navegador. Es una tecnología que nos genera CSS, que es lo que se interpretará en el navegador.

Por lo tanto necesitamos algo que nos convierta el SASS en CSS.

Imagen tomada de https://lienbtk88.blogspot.com/

Estructura de carpetas

Pues la misma que siempre, solo que **con una carpeta nueva que se llama SASS**, donde vamos a poner archivos .sass que se compilarán y que se convertirán en archivos .css que son los que luego **enlazamos en nuestra hoja de .html como ya conocemos**.



¿Cómo hago esto de manera práctica?

Hay multitud de herramientas, podemos usar **node-sass** que es una librería de NodeJS para compilar SASS, o hay herramientas con Ruby, con Python, etc… Hay algunos GUIS como Koala-app o Scout-app.

Nosotros en este módulo vamos a usar un plugin de VSCode que se llama Live Sass Compiler.

Vamos a instalar el Live Sass Compiler, a configurarlo bien para que funcione dentro de nuestra estructura de carpetas.

Imagen tomada de la home de Koala app

Tenemos que hacer un último setting

Antes de finalizar conviene hacer un pequeño ajuste en VSCode.

Si nos fijamos, cuando usamos el plugin de VSCode llamado Live-Sass-Compiler. Lo que hace es compilar y guardar el output de CSS en la misma carpeta donde tenemos el SASS. Esto, puede funcionar sin problema, pero en algunas ocasiones es mejor que vayan a carpetas diferentes.

Para ello tenemos que modificar un setting de VSCode. Para ello abrimos los settings
https://code.visualstudio.com/docs/getstarted/settings

Y vamos al apartado de JSON, y ponemos este snippet actualizando correctamente la carpeta donde queremos poner el CSS.

```
"liveSassCompile.settings.formats": [
    {
        "extensionName": ".css",
        "format": "compressed",
        "savePath": "assets/css/"
    }
]
```

Sintaxis del SASS

La misma que la del CSS pero con superpoderes.
Es decir, **en un archivo SASS puedo escribir CSS normal** y me lo va a compilar en CSS normal. Pero **puedo hacer cosas más interesantes** como:

- Usar **variables**
- **Anidar** código
- **Modularizar** el código
- Hacer **mixins** de funciones
- **Usar funciones** que ya incorpora el lenguaje SASS de manera **nativa**
- Usar **bucles**, **condicionales**, y demás técnicas y **estructuras propias de un lenguaje de programación**.

Nosotros **vamos a ver las tres primeras caracter**ísticas, y dejaremos material para las siguientes.

Diferencias entre SASS y SCSS

Sass y scss es lo mismo, exactamente, **solo que cambia ligeramente la sintaxis**, en el caso .scss ponemos llaves para abrir y cerrar las reglas y punto y coma después de las propiedades, como en CSS y en el caso de .sass, no ponemos ni llaves ni punto y comas.

Pedagógicamente recomiendo usar SCSS antes que SASS, por facilitarnos la búsqueda de errores si los tenemos.

```scss
.app {
    header {
        .header_logo {
            img {
                padding: 1rem;
            }
        }
    }
    main {
    }
    aside {
    }
}
```

```sass
.app
    header
        .header_logo
            img
                padding: 1rem

    main
    aside
```

Uso de variables

La variable se declara así:

```
$generalPadding: 3rem;
$primaryColor: #e22850;
$secondaryColor: white;
$bodyFont: "Source Sans Pro", Helvetica, sans-serif;
$shadow03: 0 3px 3px rgba(0, 0, 0, .2);
```

Uso de variables

Y **las variables pueden ser de cualquier tipo de contenido**, podemos poner lo que queramos que será reemplazado por el contenido que hayamos metido.

```scss
$generalPadding: 3rem;
$primaryColor: #e22850;
$secondaryColor: white;
$shadow03: 0 3px 3px rgba(0, 0, 0, .2);

.btn {
    background-color: $primary-color;
    color: $secondaryColor;
    padding: $generalPadding;
    box-shadow: $shadow03;
}
```

```css
.btn {
    background-color: #e22850;
    color: white;
    padding: 3rem;
    box-shadow: 0 3px 3px rgba(0, 0, 0, .2);
}
```

Uso de variables

Las variables **valen para recuperar información las veces que sea necesario**, de hecho este tipo de variables interpola literalmente el contenido de la variable donde se aplique, esto quiere decir que el contenido de la variable es exactamente el mismo que el que se quiera aplicar.

Las variables pueden ser **strings**, **números**, **colores**, **medidas**, **booleanos**, **listas**, etc... Para ver más información **sobre tipos de variables** y su uso, recomiendo leer la **documentación oficial**.

https://sass-lang.com/

Interpolación de variables

En algunos casos no vale con poner la variable, si no que **hay que interpolar su valor**. La manera de interpolar es parecida a JS: #{expression}. Ponemos un caso:

```
$colWidth: 70;
$myProperty: width;
$myClassName: column;

.#{$myClassName} {
    #{$myProperty}: calc(1080px * #{$colWidth}/100);
}

.column {
    width: calc(1080px * 70/100);
}
```

Interpolación de variables

Este tipo de casos se da porque **para nombres de clases**, **nombres de propiedades** o **uso de variables dentro de funciones** de CSS como calc, repeat, minmax, etc… **tenemos que interpolar la variable**. Si no lo hiciéramos el intérprete **generaría un CSS literal**:

```
calc(1080px * $colWidth / 100);
```

Y esto **el navegador no sabe cómo interpretarlo** y es un motivo de dolor de cabeza para desarrolladores inexperimentados.

```
body#page-course-view-topics {
⚠ width: calc(1080px * $colWidth/100);
  : ;
}
```

Anidamiento

Junto a las variables, **es una de las características más interesantes de SASS**, ya que **permite ordenar el código** y jerarquizarlo de una manera rápida y cómoda.

Básicamente **nos permite jerarquizar el código metiendo reglas CSS dentro de reglas CSS** para controlar la profundidad de los selectores.

No conviene pasarse anidando. Tengamos en cuenta que si estamos haciendo 6 anidamientos, a lo mejor hay que refactorizar.

```
ul.list_inline {
    margin: 0;
    padding: 0;
    list-style: none;

    li {
        display: inline-block;

        > a {
            display: block;
        }
    }
}
```

Anidamiento

```
ul.list_inline {
    margin: 0;
    padding: 0;
    list-style: none;

    li {
        display: inline-block;

        > a {
            display: block;
        }
    }
}
```

```
ul.list_inline {
    margin: 0;
    padding: 0;
    list-style: none;
}

ul.list_inline li {
    display: inline-block;
}

ul.list_inline li > a {
    display: block;
}
```

Anidamiento con referencia al padre

```sass
.btn {
    background-color: #334411;

    &:hover {
        background-color: #e22850;
    }
}
```

```sass
.btn {
    background-color: #334411;
}
.btn:hover {
    background-color: #e22850;
}
```

Comentarios

Los comentarios en Sass se pueden hacer como en JS, tanto con la sintaxis de /**/ como la sintaxis de líneas //

```
// Aquí podemos poner un comentario en SCSS o SASS

/*
Aquí podemos poner un comentario en bloque
con varias líneas en SCSS o SASS
*/

// Acordáos que los comentarios son muy importantes
```

Archivos parciales

Se puede (y en cierto modo se debe) hacer un código **SASS modul**ar, es decir que esté **repartido en pequeños archivos SASS**.

Se **juntan** todos en uno **mediante la etiqueta @import**, y que al final se compilan todos juntos en una hoja de CSS.

Para **indicar que un archivo es parcial** se le tiene que poner un **guion bajo antes**: **_myModule**.scss.

Para **importar el archivo se quita el _ en la regla de importación**.

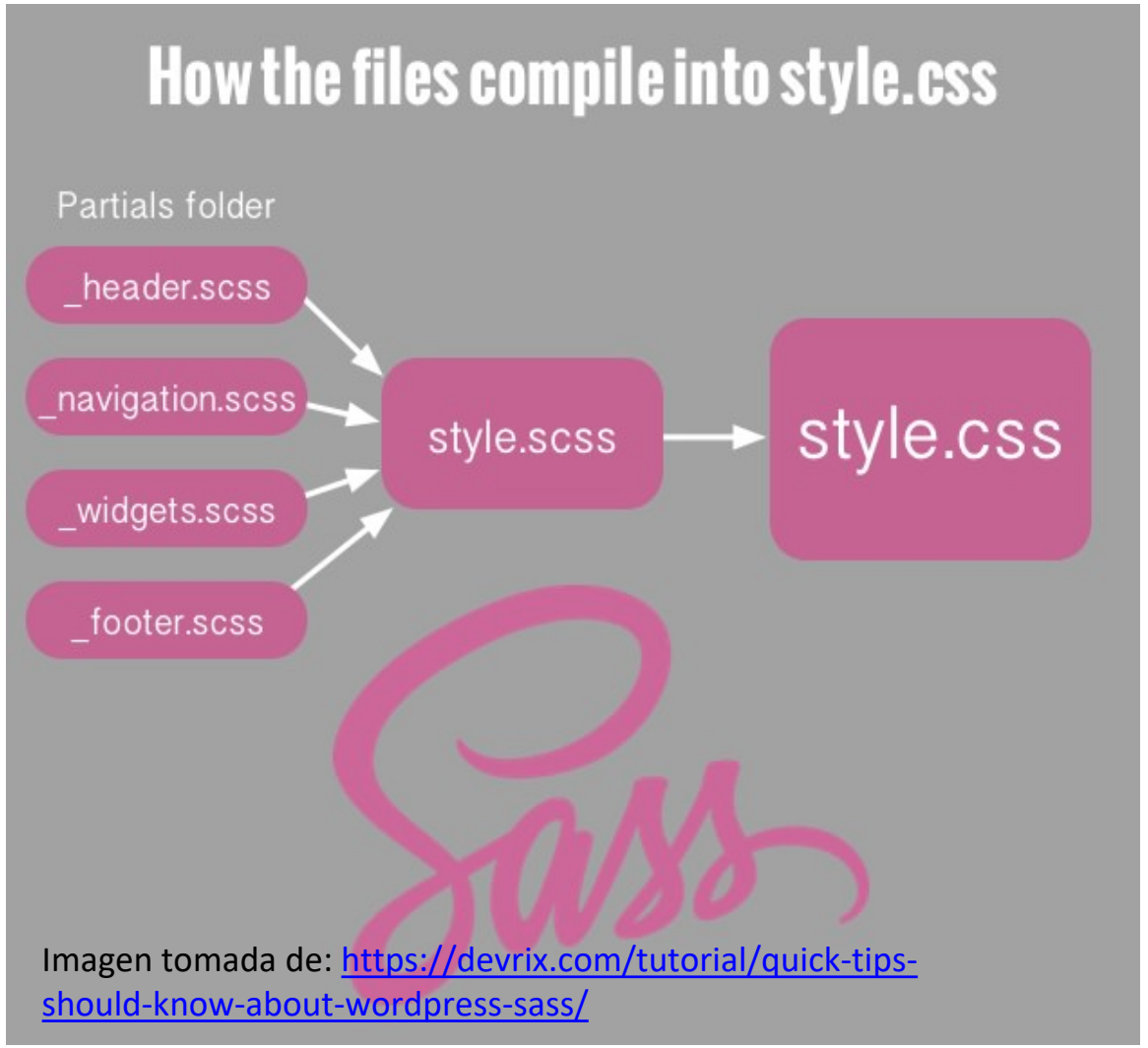


Imagen tomada de: https://devrix.com/tutorial/quick-tips-should-know-about-wordpress-sass/

Archivos parciales - Ejemplo

_buttons.scss

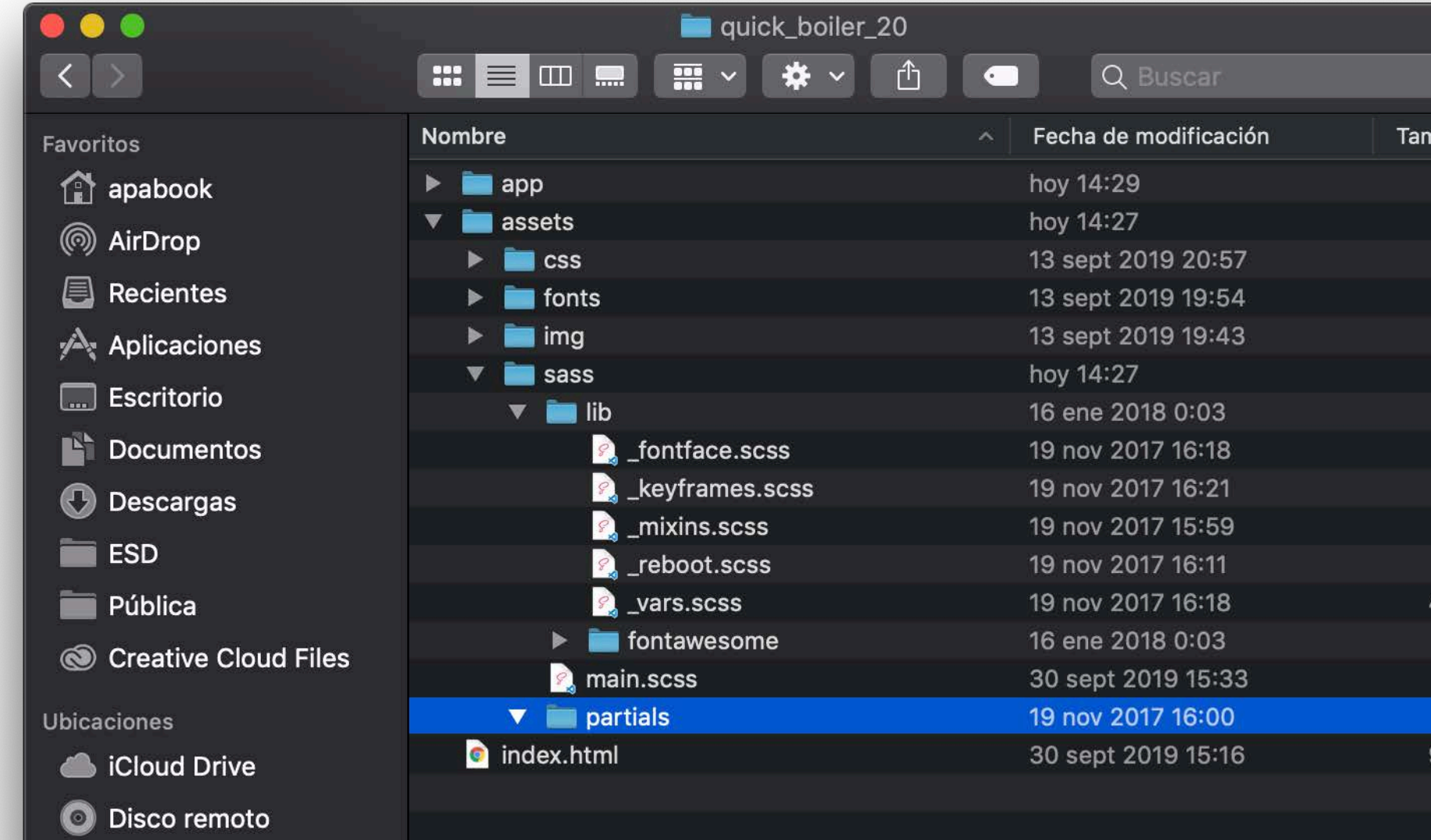
```scss
.button {
    display: block;
    &.button_success {
        background-color: green;
    }
}
```

main.scss

```scss
@import 'buttons';
```

Estructura final de archivos con módulos

La estructura de carpetas ahora quedaría más compleja.



Muchas gracias!