

PREPROCESADORES CSS LESS Y SASS



Índice Preprocesadores de Lenguajes de Estilo

0. Introducción Preprocesadores.....	4
1. El Preprocesador Less.....	4
1.2. Instalación Less.....	4
1.3 Sintaxis Compilador Less.....	5
1.4 Principales Características.....	5
1.5 Variables.....	6
1.6 Mixins.....	6
1.7 Anidando.....	8
1.8 Ámbito de las variables Less.....	10
1.9 Bubbling.....	11
1.10 Operadores.....	12
1.11 La excepción Calc().....	13
1.12 Bucles en Less.....	13
1.13 Abreviando reglas.....	14
1.14 Funciones Less.....	14
1.15 Namespaces.....	15
1.16 Extends.....	16
1.17 Comentarios.....	17
1.18 Importar archivos.....	17
2. Parte práctica – Creando un estilo para IES Celia Viñas.....	17
2.1 Archivo variables.less.....	18
2.2 Archivo mixins.less.....	19
2.3 Archivo galeria.less.....	20
2.4 Archivo layout.less.....	21
2.5 Archivo HTML.....	22
2.5.1 Cabecera.....	22
2.5.2 Área de navegación.....	23
2.5.3 Sección Proyectos.....	25
2.5.5. Sección Actividades Extraescolares.....	27
2.5.6 Sección Pie de Página.....	28
0. Introducción a Sass.....	30
1. El preprocesador SASS.....	30
1.1 Instalación Sass.....	30
1.2 Variables.....	32
1.3 Operadores aritméticos.....	32
1.4 Anidando.....	33
1.5 Propiedades anidadas.....	35
1.6 Reglas @ y directivas.....	36
1.6.1 @import.....	36
1.6.2 Hojas de estilos parciales.....	36
1.6.3. @media.....	37
1.6.4. @extend.....	38
1.7 Directivas de control y expresiones.....	39
1.7.1 La directiva @if.....	39
1.7.2 La directiva @for.....	40
1.7.3 La directiva @each.....	41
1.7.4 La directiva @while.....	42
1.8 Mixins.....	42

2. Caso práctico – Página principal IES Celia Viñas.....	44
2.1 Principales diferencias.....	45

LESS

0. Introducción Preprocesadores.

Un preprocesador CSS es un lenguaje que permite generar código CSS a partir de su sintaxis única.

El inconveniente de CSS es que es un lenguaje demasiado limitado si queremos realizar estructuras repetitivas o condicionales. Con los preprocesadores se añaden algunos conceptos que provienen de programación, lo que nos permite a la hora de generar código CSS el uso de variables, estructuras de control, mixins, reutilización de código, selectores anidados...

1. El Preprocesador Less

Less es el primero de los preprocesadores que vamos a estudiar. Less, cuyo significado es “Leaner Style Sheets”, tiene como principal característica compilar y convertir el código implementado en código CSS.

Less es de código abierto y está escrito en JavaScript. Su última versión estable es del 27 de febrero de 2014 (según *Wikipedia*). En principio Less nos proporciona los siguientes mecanismos:

- variables
- anidamiento
- operadores
- mixins
- funciones

1.2. Instalación Less.

Existen varias formas de instalar Less en nuestro equipo. La primera de ellas, es a través de la línea de comandos. Para realizar este documento se utiliza un sistema Linux 64 bits.

Instalaremos Less usando npm (el manejador de paquetes por defecto para Node.js). En el caso de que no tengamos instalado el manejador de paquetes, podemos instalarlo usando el siguiente comando:

```
sudo apt install npm
```

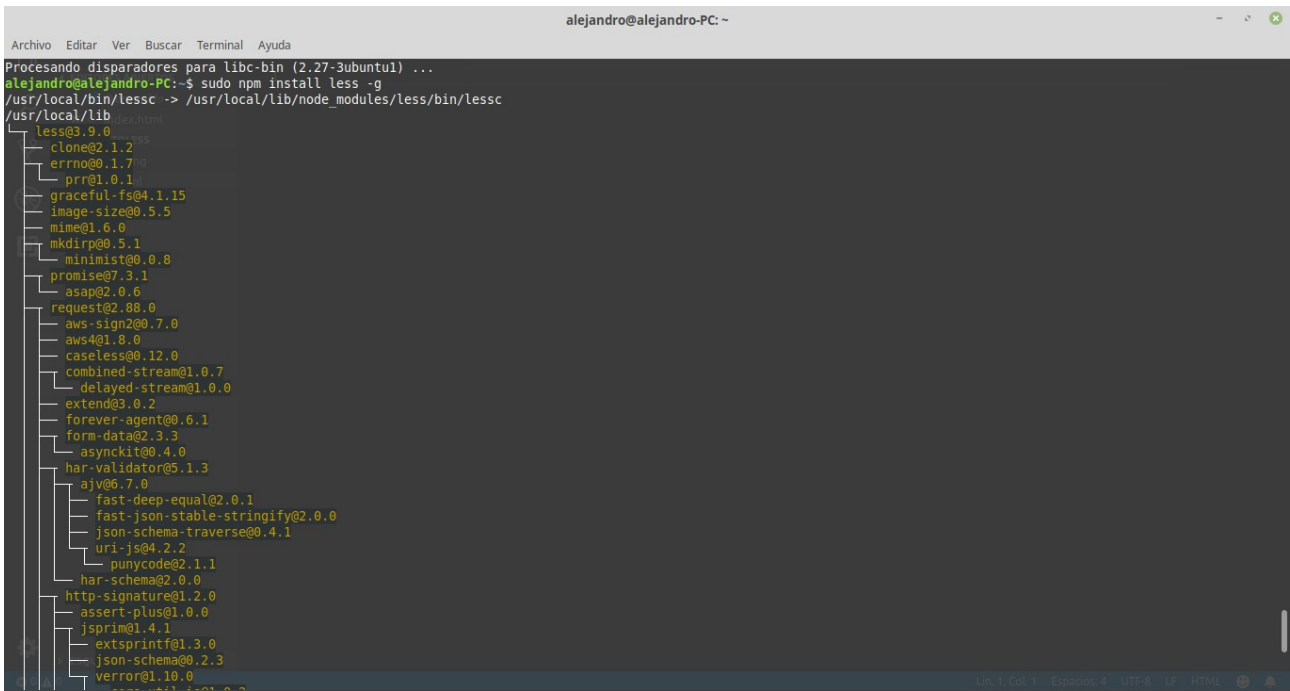
Una vez instalado ya podemos ejecutar:

```
npm install less -g
```

La opción -g instala el comando *lessc* para que esté disponible de forma global. En el caso de que necesites otra versión puedes añadir *@version* después del nombre del paquete, es decir:

```
npm install less@2.6.2 -g
```

Ya podemos usar el comando *lessc* para compilar nuestros proyectos Less y generar los archivos *css* para nuestros proyectos.



1.3 Sintaxis Compilador Less.

El comando lessc tiene la siguiente sintaxis:

```
lessc [option option=parameter] <archivoOrigen.less> [archivoDestino.css]
```

Por ejemplo:

```
lessc estiloProyecto.less main.css
```

Dentro de las opciones que podemos aplicar al comando podemos destacar:

SINTÁXIS	SIGNIFICADO
lessc -h lessc --help	Muestra un mensaje de ayuda con todas las opciones y salidas.
lessc -s lessc --silent	Detiene cualquier mensaje de advertencia que se muestre
lessc -M lessc --depends	Muestra una lista de dependencias por la salida estándar

1.4 Principales Características

En primer lugar estudiaremos como crear ficheros less y cual es la sintaxis en su modo de funcionar.

Posteriormente y para mostrar un ejemplo de uso de este pre-procesador, expondremos un caso práctico en el que se aplicará less en un entorno real.

Para ello, crearemos una web desde cero haciendo uso de esta tecnología, al igual que haremos con Sass para ver las diferencias entre un preprocesador y otro.

Nuestra idea es generar un sistema de Grid, en el que dividiremos nuestra página en filas de 12 columnas. Todos los contenidos se ajustarán a un número de filas entre 1 y 12.

Necesitamos conocer al menos las siguientes tecnologías: HTML5, CSS3 y algo de JavaScript (o jQuery). En archivos Less, mezclaremos lenguaje CSS con lenguaje Less. Una vez generados los ficheros Less, se compilarán y se generará el archivo CSS correspondiente.

1.5 Variables.

Less nos permite definir, como si se tratase de un lenguaje de programación, variables para trabajar con ellas. Para declarar una variable, hay que anteponer el símbolo “@” antes del nombre de la variable seguido de “:”, el valor de la misma y finalizado en “;”:

```
@nombreVariable: valor;
```

Por ejemplo:

```
@altura: 15px;
@anchura: @altura+22px;

#capaPrincipal {
  width: @altura;
  height: @anchura;
}
```

Este código al compilarlo, generaría el siguiente CSS:

```
#capaPrincipal {
  width: 15px;
  height: 37px;
}
```

1.6 Mixins.

Los Mixins son una de las características que hacen a estos preprocesadores tan potentes. Consiste en la reutilización de código. Es decir, un Mixin contiene una serie de reglas que podemos reutilizar dentro de cualquier otra serie de reglas. Por ejemplo

```
.bordes {
  border-top: 1px dotted black;
  border-bottom: 2px solid black;
}
```

Ahora la clase *bordes* podemos usarla dentro de cualquier otro conjunto de reglas, reutilizando así estas reglas del mixin.

```
#menu a {  
    color: #111;  
    .bordes;  
}
```

Ahora las propiedades de *.bordes* aparecerán dentro de *#menu a* además del color establecido. A partir de la versión 3.5 podemos usar mixins y reglas con valores preestablecidos.

```
#colores() {  
    primario: blue;  
    secundario: green;  
}  
  
.boton {  
    color #colores[primario];  
    border: 1px solid #colores[secundario];  
}
```

La salida generada en el fichero CSS sería:

```
.boton {  
    color: blue;  
    border: 1px solid green;  
}
```

También es muy interesante hacer uso de parámetros. En el caso práctico que se propone de ejemplo, se usa funciones con parámetros para calcular la anchura de cada una de las clases *columnas*:

```
.columnas (@n){  
    width: 100/@n;  
}  
  
#primera {  
    .columnas(4);  
    background: #840;  
}
```

El código CSS generado será:

```
#primera {  
    width: 25%;  
    background: #840;  
}
```

1.7 Anidando.

Otra de las características más potentes es la anidación de estilos, lo cual permite ahorrar bastante código. Veámoslo con un sencillo ejemplo:

Imagina que necesitamos crear los siguientes estilos:

```
#cabecera {  
    color: red;  
}  
  
#cabecera .navegacion {  
    font-size: 12px;  
}  
  
#cabecera .logotipo {  
    width: 300px;  
}
```

En Less podemos generar este mismo código de una forma más sencilla:

```
#cabecera {  
    color:red;  
    .navegacion {  
        font-size: 12px;  
    }  
    .logotipo {  
        width: 300px;  
    }  
}
```

Cómo se puede observar, el código resulta más claro, sencillo y corto con respecto a CSS puro. Una característica importante que debemos tener en cuenta es el uso de pseudoclases, tales como :hover, :focus, :first, :last, etc. Para poder hacer uso de anidaciones y pseudoclases, usaremos el pseudoelemento **&**, el cuál hace referencia al elemento padre de la anidación. Veamos un ejemplo:

```
tr {  
    border: 1px dotted black;
```



```

th {
  font-weight: bolder;
  text-align: center;
}
/* La siguiente regla hace referencia al evento de pasar el
ratón por encima de una fila. Es decir el elemento & hace
referencia al elemento tr */
&:hover{
  background-color: orange;
}
&:first {
  background-color: #333;
  color: #eee;
}
}

```

Transformado a CSS:

```

tr {
  border: 1px dotted black;
}

tr th {
  font-weight: bolder;
  text-align: center;
}

tr:hover {
  background-color: orange;
}
tr:first {
  background-color: #333;
  color: #eee;
}

```

Otro uso habitual del pseudoelemento **&** es el de generar clases de forma repetitiva con nomenclaturas parecidas. Por ejemplo:

```

.boton {
  &-confirmar {
    background-color: green;
  }
  &-cancelar {
    background-color: red;
  }
  &-defecto {

```

```
        background-color: blue;
    }
}
```

El código CSS generado será:

```
.boton-confirmar {
    background-color: green;
}
.boton-cancelar {
    background-color: red;
}
.boton-defecto {
    background-color: blue;
}
```

1.8 Ámbito de las variables Less.

La anidación de elementos, permite definir el ámbito de las variables en la hoja de estilos. En Less el ámbito de la variable puede variar cuando se anidan estructuras, y debemos tener especial cuidado al usarlas. Podemos observar en el siguiente ejemplo, como varía el valor de la variable @tamano:

```
@tamano: 1.5em;
```

```
#contenido {

    .cabecera {
        @tamano: 2em;
        nav {
            font-size: @tamano; //El valor total es 2em
        }
    }
    .articulo {
        font-size: @tamano + 0.2em; //El valor total es 1.7em
    }
}
```

El ámbito en Less de variables y mixins es muy similar al ámbito en CSS. Es decir, las variables y mixins en primer lugar se buscan de forma local, y si no se encuentran, se cogería siempre el del elemento padre (inherit).

```
@var: red;
```

```
#pagina {
```

```

@var: white;

#cabecera {
    color: @var; //El color sería blanco.
}
}

```

En Less, el orden en el que se definen las propiedades no es importante, por lo que no es necesario, en el caso de nuestro ejemplo, que la declaración de `@var: white` se encuentre antes de la declaración de `cabecera`. El siguiente ejemplo obtiene el mismo resultado que el anterior:

```

@var: red;

#pagina {
    #cabecera {
        color: @var; //El color sería blanco.
    }
    @var: white;
}

```

1.9 Bubbling.

Así como anidamos selectores, también podemos anidar reglas como `@media` y `@supports`. Este tipo de reglas se colocarán al principio (como padre del selector en cuestión) cuando se transforme en código CSS. Por esto hay que tener muy en cuenta el orden en el que colocamos las reglas. A este tipo de reglas se les llama Bubbling. Por ejemplo:

```

.contenido {
    width: 300px;
    @media (min-width: 768px) {
        width: 600px;
        @media (min-resolution: 192dpi) {
            background-image: url(img/fondoRetina.png);
        }
    }

    @media (min-width: 1280px) {
        width: 800px;
    }
}

```

Este código generará el siguiente archivo CSS:

```

.contenido {
  width: 300px;
}

@media (min-width: 768px) {
  .contenido {
    width: 600px;
  }
}

@media (min-width: 768px) and (min-resolution: 192dpi){
  background-image: url(img/fondoRetina.png);
}

@media (min-width: 1280px) {
  .contenido {
    width: 800px;
  }
}

```

1.10 Operadores.

Como en cualquier lenguaje de programación, haremos uso de los operadores: +, -, * y /, pudiendo operar con cualquier número, color o variable. Las operaciones matemáticas tienen en cuenta las unidades de medida e intentan convertir a la misma unidad antes de sumar, restar o compararlos. En una operación con distintas unidades de medida, intentará coger la unidad del operador de más a la izquierda. Si la conversión es imposible o no tiene sentido, las unidades son ignoradas. Por ejemplo, cuando se intenta realizar una conversión de píxeles a centímetros o de radio a %.

Ejemplos:

```

//números convertidos a la misma unidad
@conversion-1: 5cm + 10mm; //resultado es 6 cm
@conversion-2: 2 - 3cm - 5mm; //resultado es -1.5cm, ya que el
número 2 lo transforma a cm

```

```

//conversión imposible
@unidades-incompatibles: 2 + 5px - 3cm; //resultado es 4 px ya que
siempre coge la unidad de más
//a la izquierda

```

```

//uso de variables
@base: 5%;
@doble: @base * 2; //el resultado es 10
@otro: @base + @doble; //resultado es 15%

```

Less opera con números y les asigna una determinada unidad en el resultado. Además nos permite hacer operaciones aritméticas sobre colores:

```
@color: #224488; / 2 //el resultado es: #112244
background-color: @color + #111; //El resultado es #223355
```

****Establecer espacios entre operandos y operadores.**

1.11 La excepción Calc().

Esta función (que pertenece a las características de CSS3) no evalúa expresiones matemáticas, pero evaluará variables y operaciones matemáticas en funciones anidadas.

En realidad, calc() nos va a ayudar a realizar cálculos con unidades de medida relativas (% , em , px , ...), pudiendo incluso mezclarlas.

Por ejemplo:

```
@variable: 50vh / 2; //recordamos que vh es una unidad de medida
relativa: 1vh = 1% de la altura del viewport
```

```
width: calc (50% + (@variable - 20px)) //el resultado sería la
propiedad CSS3 calc(50% + (25vh -20px))
```

*****Viewport*** → es el área visible que tiene el usuario sobre la página web. Este área varía con el dispositivo, y puede ser desde un dispositivo móvil pequeño hasta la pantalla de un PC.

1.12 Bucles en Less.

Podemos hacer uso de estructuras repetitivas, normalmente usadas a la hora de crear estilos en un sistema de Grid en una interfaz web (tal y como lo hace Bootstrap por ejemplo). En nuestro caso práctico podemos ver un ejemplo:

```
.columna(@n) {
    width: @n*(100% / @columnas);
    .columna-flotante;
}
```

```
//definimos el bucle que genera el estilo de las columnas del
grid: col-1, col-2,...
```

```
.grid(@i) when (@i<=@columnas){
    .col-@{i}{
        .columna(@i);
        border: 1px solid black;
```

```

    }
    .grid(@i+1);
}

```

De esta forma se crearán tantos estilos como columnas definamos en la variable `@columns`. Además debemos pasar como parámetro el valor inicial, que en este caso puede ser 1. Por ejemplo, si definimos `@columns=6` y hacemos la llamada `.grid(1)` se generarán los estilos: `col-1`, `col-2`, ..., `col-6`. Este nombre viene especificado en la regla:

```

    ...
    .col-@{i}{
    ...

```

1.13 Abreviando reglas.

Esta característica permite asignar un nombre a un valor de una propiedad CSS o a la misma propiedad, asignándola a una variable. De esta forma, podemos usar esa variable para hacer referencia a una determinada propiedad en cualquier parte del código:

```
@minimo: ~"(min-width:768px)";
```

```

.elementoClase {
  @media @minimo {
    font-size: 1.2em;
  }
}

```

El código CSS sería:

```

@media (min-width:768px){
  .elementoClase {
    font-size: 1.2em;
  }
}

```

Hay que tener en cuenta, que la regla *@media* siempre se pondrá dentro del elemento al que se aplicará este media query.

1.14 Funciones Less

Less tiene una gran variedad de funciones para transformar colores, manipular strings o realizar diversas operaciones matemáticas. Puedes encontrar la especificación de todas ellas en:

<http://lesscss.org/functions/>

Hacer uso de estas funciones siempre nos puede resultar beneficioso a la hora de crear efectos en nuestra web.

Por ejemplo, este código convierte un valor en porcentaje, satura el color de letra en un 5% y establece un color de fondo aclarado en un 25% y girado en 8 grados.

```
@base: #f04615 //Color base para el fondo
@width: 0.5;
.clase {
    width: percentage(0.5) //Convierte el valor 0.5 a 50%
    color: saturate (@base, 5%);
    background-color: spin(lighten(@base, 25%), 8);
}
```

1.15 Namespaces.

Algunas veces puedes necesitar agrupar tus mixins, ya sea por cuestiones de organización o simplemente por ofrecer encapsulamiento en tu código. Less ofrece vías para realizar esto de una forma clara y sencilla. Lo vemos con un ejemplo que quedará más claro:

```
#encapsula() {
    .boton {
        display: block;
        border: 1px solid black;
        background-color: grey;
        &:hover {
            background-color: white;
        }
    }
}

.tabulador {
    ...
}

.citas {
    ...
}
```

Si ahora quieres usar el `mixin` del `.boton` (es decir, sus estilos) en “`#cabecera a`”, se podría hacer lo siguiente:

```
#cabecera a {  
    color:orange;  
    #encapsula.boton();  
}
```

1.16 Extends

Less permite la extensión o ampliación de tipos. Por ejemplo, imagina que tenemos la clase `animal`:

```
.animal {  
    background-color: black;  
    font-size: 1.5em;  
    color:red;  
}
```

Ahora queremos especificar el subtipo de animal clase `perro`. Esta clase, tendrá como color de fondo `brown` y además un efecto de subrayado. Haciendo uso de `extends` podríamos especificar el siguiente código:

```
.perro {  
    &:extend(.animal);  
    background-color: brown;  
    text-decoration: underline;  
}
```

El código CSS resultante:

```
.animal {  
    background-color: black;  
    font-size: 1.5em;  
    color:red;  
}  
.perro {  
    background-color: brown;  
    text-decoration: underline;  
    font-size: 1.5em;  
    color:red;  
}
```


1.17 Comentarios.

En Less se pueden crear dos tipos de comentarios:

- comentarios de una línea.
`//Comentario de una línea`
- Comentarios de varias líneas.
`/* Comentario de
varias
líneas */`

1.18 Importar archivos.

En un archivo Less podemos importar otros archivos .less, lo cuál hará que se puedan usar todas las variables y mixins que el archivo a importar contenga. Además, también podemos importar reglas de otros archivos CSS. Debemos tener en cuenta que, al importar archivos .less, no es necesario especificar su extensión:

```
@import "variables"; //variables.less
```

```
@import "miEstilo.css";
```

2. Parte práctica – Creando un estilo para IES Celia Viñas.

Como se comentó al inicio, se va a crear una página a modo ejemplo. Para ello vamos a usar la siguiente estructura de ficheros:

```
Práctica/  
|  
|--less/                                #Aquí generaremos los archivos.less  
|  |  
|  |-- variables.less  
|  |-- principal.less  
|  |-- ...  
|--css                                  #Aquí se generarán los archivos css  
|  |  
|  |-- principal.css  
|  
|  
|--img/  
|  |  
|  |-- imagen1.png  
|  |-- imagen2.jpg  
|  |-- ...  
|-- index.html
```

Crearemos un sistema de *grid* sencillo y parecido al usado por varios frameworks (como por ejemplo el que usa Bootstrap). Para ello dividiremos la interfaz web en filas, y cada fila estará dividida en 12 columnas, de tal forma que podemos usar y disponer de estas 12 columnas como deseemos a la hora de crear contenido.

Usaremos las siguientes clases, las cuales aplicaremos a elementos *div*, *article* o *section*:

- `.fila` → Se usa para declarar una fila con 12 columnas.
- `.col-n` → Se usa para especificar el número de columnas (n) que usaremos en cada fila.

****Debemos usar 12 columnas exactamente, ni más ni menos, para que la interfaz sea correcta.**

Por ejemplo:

col-6			col-6			
.col-2		.col-5			.col-5	
.col-1	.col-1	.col-4		.col-4		.col-2

A la hora de crear nuestros documentos Less, la idea es crear distintos archivos con distinto tipo de contenido (variables, mixins,...) para luego importarlos en el archivo Less principal, que para nosotros será *layout.less*.

2.1 Archivo variables.less

Nuestro primer documento Less, está destinado a almacenar aquellas variables globales que configuran los elementos principales de nuestro portal, se trata del archivo *variables.less*. En un principio las variables a destacar son:

- Número de columnas en las que dividiremos cada fila: 12.
- Color primario: #042351.
- Color secundario: #c70000.
- Color hiperenlace: #f2f2f2. Utilizado en el área de navegación.
- Color hover: #ddd. Utilizado en el área de navegación cuando se le pasa el ratón por encima a un hiperenlace.
- Tipo de letra: Tahoma, Verdana, sans-serif.
- Tamaño de la letra: 0.8em.
- Valores para el tipo de letra:
 - Negrita: 700.
 - Normal: 300.
 - Claro: 100.
- Ruta del logotipo. Para establecer esta ruta, debemos tener en cuenta donde estará ubicada la imagen desde el archivo generado CSS. Debido a que hemos estructurado de una forma óptima nuestros archivos de la web, podemos indicar la siguiente ruta:
 - `../img/escudo.png`.

Este es el código:

```

//Colores
@colorPrincipal: #042351;
@colorSecundario: #c70000;
@columnas: 12;

//Tipografía
@tipoLetra: 'Tahoma', 'Verdana', 'sans-serif';
@tamanoLetra: 0.8em;

@regular: 300;
@negrita: 700;
@claro: 100;

@colorHover: #ddd;;
@colorHiperenlace: #f2f2f2;

//Otros
@logotipo: "../img/escudo.png";

```

2.2 Archivo mixins.less

Este archivo contiene los mixins que generan automáticamente las columnas de cada una de las filas.

```

//Especifica un posicionamiento flotante a la izquierda para cada
elemento.
.columna-flotante {
    float: left;
    box-sizing: border-box;
}

//Calcula la anchura para cada columna, dependiendo del número de
columnas que se usen.
.columna(@n) {
    width: @n*(100% / @columnas);
    .columna-flotante;

}

//definimos el bucle que genera el estilo de las columnas del
grid (col-1, col-2,...):

```

```

.grid(@i) when (@i<=@columnas){
  .col-@{i}{
    .columna(@i);
    border: 1px solid black;
  }
  .grid(@i+1);
}

```

Como se puede observar, en esta parte de código es donde hacemos uso de estructuras de control repetitivas, como en cualquier lenguaje de programación. Para ello, cuando se realice la llamada a la clase `.grid`, habrá que indicarle un parámetro, `@i`, indicando el inicio de la iteración. En nuestro caso, será 1, ya que vamos a usar desde filas con 1 columna hasta filas con 12 columnas.

También se ha puesto una clase para centrar en la página cualquier tipo contenido:

```

.centrado{
  display: block;
  margin-left: auto;
  margin-right: auto;
}

```

2.3 Archivo galeria.less

Este archivo alberga los estilos para la galería fotográfica que se creará en la sección de actividades. De esta forma vamos creando un diseño modular del archivo CSS.

Fuente del código la galería fotográfica: https://www.w3schools.com/css/css_image_gallery.asp

Hemos adaptado a la sintaxis Less.

```

div.gallery {
  margin: 5px;
  border: 1px solid #ccc;
  float: left;
  width: 180px;
  &:hover {
    border: 1px solid #777;
  }
  img {
    width: 100%;
    height: auto;
    display: block;
    margin: 0 auto;
  }
}

```

```

}
div.desc {
    padding: 15px;
    text-align: center;
}
}

```

2.4 Archivo layout.less

Este será el archivo principal y el que compilaremos. Para ello debemos importar el resto de ficheros .less, de forma que al crear el archivo CSS final, contenga todos los estilos de todos los ficheros. Por otro lado será necesario usar las variables definidas en *variables.less* en el resto de ficheros.

```

@import 'variables';
@import 'mixins';
@import 'galeria';

```

Notar que **no** es necesario especificar la extensión, siempre y cuando sean archivos .less. Una vez importados los ficheros, generaremos los estilos principales de la página, los que especifican el layout o la interfaz de la misma:

```

#contenedor {
    max-width: 960px;
    margin: 0 auto;
    width: 90%;
    font-family: @tipoLetra;

    .fila {
        &:before,
        &:after {
            content: " "; /* 1 */
            display: table; /* 2 --> Estos estilos se definen ya que, al tener los
                elementos de las filas en posición flotante, el resto de elementos puede
                que se superpongan (por la definición propia de los elementos flotantes).
                Por ello estableciendo estos estilos se evita tal efecto.*/
        }
        &:after {
            clear: both;
        }
    }
}

/* Generando filas y columnas */

.fila {
    width: 100%;
    max-width: 1080px;

```

```

margin: 10px auto;
.grid(1); // Realizamos la llamada al bucle que generará todas las clases de
columna, es decir, col-1, col-2,...
}
}
...

```

A continuación, comentaremos con detalle el archivo HTML generado e iremos comentado el código que se ha introducido a partir del mismo en el archivo *layout.less*

2.5 Archivo HTML.

Nuestra estructura HTML será la siguiente, en la que cada sección corresponderá con una fila:

- Cabecera. Contendrá el logotipo y el nombre del centro como cabecera de nivel 1. Usaremos 4 columnas para el logotipo y 6 para el resto.
- Área de navegación. Compuesta por un menú con hiperenlaces a las secciones de la página. Ocupará las 12 columnas disponibles.
- Sección Proyectos. 3 columnas a la izquierda para últimos proyectos, otras 5 columnas para los proyectos más visitados y 4 columnas con hiperenlaces a entidades colaboradoras.
- Sección Actividades Extraescolares. Aparecerá una galería fotográfica con las imágenes más destacadas, la cual ocupará 8 columnas y las 4 restantes se dejará para información sobre cada una de las actividades.
- Sobre nosotros. Esta dividido en dos secciones de 6 columnas cada una. Se insertará información de contacto y ubicación en la parte izquierda y un mapa de Google en la parte derecha.
- Pie de página. Ocupará las 12 columnas y aparecerá el nombre de los participantes en el grupo de trabajo.

2.5.1 Cabecera

Como se ha comentado, la cabecera de nuestra página estará dividida en en dos secciones: una para la imagen del logotipo de 4 columnas y el resto para el título de la página.

El código HTML es el siguiente:

```

<div class='fila' id="cabecera">
  <div class='col-4' id="inicio">
    
  </div>
  <div class="col-8">
    <h1>I.E.S Celia Viñas - Practicando con Less</h1>
  </div>
</div>

```

Para una mejor adaptación de la plantilla se han asignado los siguientes estilos en el archivo layout.less:

```
#cabecera {
  img{
    width: 20%;
    .centrado;
  }
  h1{
    text-align: center;
    color: @colorPrincipal;
    font-weight: @negrita;
  }
}
```

2.5.2 Área de navegación.

Utilizaremos un sencillo menú cuyo código HTML vemos a continuación:

```
<div class='fila' id="navegacion">
  <div class="col-12 navegacion" id="miMenu">
    <a href="#inicio" class="activo">Inicio</a>
    <a href="#proyectos">Proyectos</a>
    <a href="#Noticias">Actividades</a>
    <a href="#sobreNosotros">Sobre Nosotros</a>
  </div>
</div>
<!--Acaba área de navegación-->
```

Nos fijamos en los siguientes aspectos:

- Existe un elemento del menú, el cual indica que sección está activa. Esta clase activa, cambiará dependiendo de la sección en la que estemos, para ello se ha añadido un sencillo código jQuery.
- Se ha añadido otro fragmento de código para que el menú esté siempre visible en la parte superior del documento de manera fija, aunque se haga *scroll*.

El código Less para esta sección es el siguiente:

```
#areanav {
  background-color: spin(lighten(@colorPrincipal, 25%), 8);
  max-width: 960px;
}

.navegacion {
```

```

background-color:spin(lighten(@colorPrincipal, 25%), 8);
overflow: hidden;

}

/* Estilo de los hiperenlaces del área de navegación */
.navegacion a {
float: left;
display: block;
color: @colorHiperenlace;
text-align: center;
padding: 14px 16px;
text-decoration: none;

}
a, h1, h2, h3, h4 {
color: spin(lighten(@colorPrincipal, 25%), 8);

}

/* Cambia el color de los hiperenlaces del área de navegación cuando se le
pasa el ratón por encima */
.navegacion a:hover {
background-color: @colorHover;
color: black;
}

/* Clase para un elemento que esté activo */
.activo {
background-color:@colorSecundario;
color: white;
}

/* Oculta el enlace que debería abrirse y cierra el área de navegación en
pantallas pequeñas */
.navegacion .icono {
display: none;
}
.fijar {
position:fixed;
top:0;
max-width: 960px;

}

```

Con este código conseguiremos el estilo deseado para el área de navegación, la cuál, será siempre visible.

2.5.3 Sección Proyectos.

A modo ejemplo, publicaremos algunos proyectos en los que el centro está implicado. Para ello y como hemos comentado anteriormente, la sección de proyectos se divide en 3 zonas. Una zona para los últimos proyectos (3 columnas), otra zona para los proyectos más visitados (5 columnas) y por último los enlaces a las “supuestas” entidades colaboradoras.

```
<div class="fila contenido" id="proyectos">
  <div class="col-3" id="nuevos">
    <h1>Últimos proyectos</h3>
    <h2>BiblioWeb</h4>
    <p>En este proyecto ha sido implementado por los alumn@s de Ciclo
    Formativo de Desarrollo de Aplicaciones Web.</p>
    <p>Se han digitalizado las publicaciones más antiguas e importantes para
    que todo el mundo tenga acceso desde su propia casa.</p>
    
    <p>El catálogo se puede ver en: <a
    href="https://iescelia.org/web">Biblioteca Virtual</a></p>
    <h2>Museo Virtual</h2>
    <p>Comenzamos un nuevo curso y pretendemos que la Biblioteca no sea solo
    una sala de lectura sino también una sala de exposiciones. En esta primera etapa
    hemos recuperado cuatro mapas franceses de 1911 de la colección de Louis André,
    un mapa de los reinos de España de 1961 y otro de la España Agrícola y Ganadera
    de 1958.</p>
    <p>Para poder verlos solo tienes que pinchar en el siguiente enlace: <a
    href="https://iescelia.org/museovirtual/" alt="Imagen museo virtual">Museo
    Virtual</a></p>
    
  </div>
  <div class="col-5" id="masVisitados">
    <h1>Proyectos más visitados</h1>
    <h2><a href="https://iescelia.org/celia360/">Celia Tour 360</a></h2>
    <p>La aplicación Celia Tour presenta a todo el mundo nuestro centro
    histórico en Andalucía, a través de imágenes en 360°. Este proyecto ha supuesto
    una implicación de alumnado del centro de todas las etapas de las que dispone,
    1º, 2º, 3º, 4º de ESO, 1º y 2º Bachillerato, y 1º y 2º de ciclos formativos de
    la rama de Informática.</p>
    
    <h2><a href="https://iescelia.org/erasmus/">Erasmus + KA2</a></h2>
```

<p>Gracias a este programa y al esfuerzo de profesores y alumnos se ha conseguido realizar intercambios con otros centros de Europa: Eslovaquia, Letonia, Inglaterra... </p>

</div>

<div class="col-4" id="colaboradores">

<h1>Entidades Colaboradoras con el centro</h1>

<h3>Junta de Andalucía</h3>

<h3>Amigos de la Alcazaba</h3>

<h3>Bibliotecas Virtuales</h3>

</div>

</div><!--Termina sección proyectos-->

El código Less aplicado para este fragmento de la página es el siguiente:

```
#proyectos {
  font-size: @tamanoLetra - 0.1em;

  a {
    font-weight: @negrita;
  }
  #nuevos{
    img {
      width: 80%;
      .centrado;
    }
    padding: 1.2em;
  }
  #masVisitados {
    img {
      width: 50%;
      .centrado;
    }
    padding: 1.2em;
  }
}
```

```

#colaboradores{
  img {
    width: 50%;
    margin: 0 auto;
    display: block;
  }
  h3 {
    text-align: center;
    text-decoration: underline;
  }
}
}

```

2.5.5. Sección Actividades Extraescolares.

Esta sección, está dividida en 2 secciones, una que ocupa 8 columnas, destinada a introducir una breve descripción y la galería de fotos, y otra de 4 columnas con hiperenlaces a las supuestas empresas colaboradoras.

El código HTML es el siguiente:

```

<div class="fila" id="actividades">
  <div class="col-8">
    <h1>Actividades Extraescolares</h1>
    <p>Desde el centro se anima a participar y crear diversidad de actividades
    extraescolares: intercambios con centros de otros países, viajes de estudios
    multiaventura, día de Andalucía, visita telescopios Calar Alto... </p>
    <p>A continuación os mostramos distintas imagenes:</p>
    <!--Galería fuente w3school -->
    <div class="gallery">
      <a target="_blank" href="img/teatrogreco.jpeg">
        
      </a>
      <div class="desc">Teatro GrecoLatino</div>
    </div>

    <div class="gallery">
      <a target="_blank" href="img/arquitectura.jpg">
        
      </a>
      <div class="desc">Colegio de Aparejadores Almería</div>
    </div>
    ...<!-- todas las imágenes que queramos poner -->
  </div>
  <div class="col-4">
    <h2>Nuestros Viajes</h2>
    <ul>

```

```

        <li><a href="https://iescelia.org/web/nos-vamos-a-paris/" >Viaje a
París</a></li>
        <li><a href="https://iescelia.org/web/teatro-grecolatino/">Teatro Greco-
Latino</a></li>
        <li><a href="https://iescelia.org/web/jornadas-informativas-
arquitectura-tecnica/">Visita al Colegio de Aparejadores</a></li>
    </ul>
</div>
</div>

```

Para esta sección de la página se propone el siguiente código Less:

```

#actividades {
    img {
        width: 60%;
    }
    ul {
        list-style: none;
    }
    ul li a:hover {
        font-size: 1.1em;
    }
}

```

Es decir, se establecerá para las imágenes que pertenecen a la capa *#actividades* un tamaño del 60% del espacio disponible, así como eliminar las viñetas de la lista y un sencillo efecto que amplía el tamaño de letra en 0.1 em al pasar el puntero del ratón por encima.

2.5.6 Sección Pie de Página

Esta última sección ocupará todo el ancho disponible, es decir, 12 columnas. Además tiene un tamaño de letra algo más pequeño que el resto de la página.

El código HTML es el siguiente:

```

<div class="fila" id="pie">
    <div class="col-12">
        <p>Proyecto realizado para el Grupo de Trabajo del IES Celia Viñas: </p>
        <ul>
            <li>Pablo Escobar</li>
            <li>Manuel Piñero</li>
            <li>Alejandro Ramallo</li>
            <li>Jose Juan Sánchez</li>
        </ul>
    </div>
</div>

```

```
</div>
```

```
</div>
```

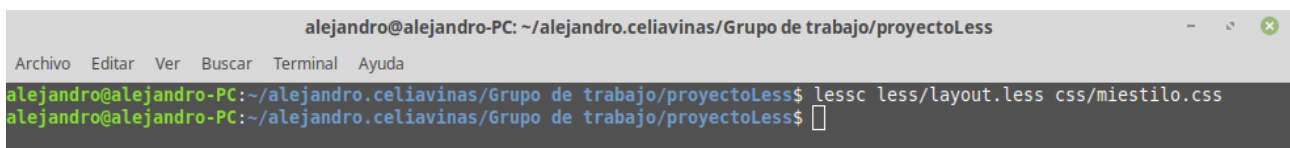
El código Less:

```
#pie {  
  p, ul {  
    font-size: .7em;  
    text-align: center;  
    color: @colorPrincipal;  
  }  
  ul {  
    list-style-type: none;  
    margin-left: 28.5%;  
  }  
  ul li {  
    float: left;  
    margin-left: .6em;  
  }  
  
  margin-bottom: .7em;  
}
```

Una vez generado el documento *layout.less*, desde la carpeta raíz de la página web ejecutaremos en un terminal el siguiente comando:

```
lessc less/layout.less css/miestilo.css
```

Si no hemos cometido ningún error se generará el archivo *miestilo.css* en la carpeta correspondiente.

A screenshot of a terminal window. The title bar reads "alejandro@alejandro-PC: ~/alejandro.celiavinas/Grupo de trabajo/proyectoLess". The terminal shows the command "lessc less/layout.less css/miestilo.css" being entered and executed, with a cursor on the line below.

```
alejandro@alejandro-PC:~/alejandro.celiavinas/Grupo de trabajo/proyectoLess$ lessc less/layout.less css/miestilo.css  
alejandro@alejandro-PC:~/alejandro.celiavinas/Grupo de trabajo/proyectoLess$
```

SASS

0. Introducción a Sass

Sass (Syntactically Awesome StyleSheets), es otro preprocesador CSS, el cuál permite, al igual que Less, el uso de variables, anidamientos, mixins, importación de hojas de estilo y otras muchas características. El famoso framework Bootstrap, desde su versión 4, usa este lenguaje para implementar sus archivos CSS. Anteriormente se hacía uso de Less. Conociendo Less y Sass podemos adaptar estos frameworks a nuestro gusto y hacer diseños mucho más personalizados y adaptados.

1. El preprocesador SASS.

Sass usa dos sintaxis:

- una sintaxis en la que los anidamientos se especifican con llaves y cada regla termina en punto y coma.
- una “sintaxis indentada”, de forma que el anidamiento de selectores CSS se indica con tabulaciones en vez de con llaves, y las propiedades se separan con saltos de línea en vez de con puntos y coma.

Cualquiera de las dos es una notación válida y podemos exportar documentos de una notación a otra.

Para los ficheros, usaremos la extensión “.scss” o la extensión “.sass”, el primero usará sintaxis con llaves y puntos y comas y la segunda una sintaxis indentada.

Antes de nada, veremos como realizar la instalación de SASS en nuestro equipo.

1.1 Instalación Sass.

Sass podemos usarlo de tres formas distintas:

- en la consola o terminal de comandos.
- como módulo de Ruby
- como plugin de cualquier framework compatible con Rack¹.

Nosotros usaremos Sass a través de la línea de comandos. Aún así existen muchas aplicaciones que interpretan y traducen código Sass:

- [Scout-App](#)

¹ Rack proporciona una interfaz modular y adaptable para el desarrollo de aplicaciones web en Ruby. Al encapsular las solicitudes y respuestas HTTP, unifica la API para servidores web, marcos web y software en medio

- [Compass.app](#)
- [Koala](#)
- [Live Reload](#)

En primer lugar instalamos Sass para todo el sistema usando siguiente comando:

```
npm install -g sass
```

```

alejandro@alejandro-PC:~$ sudo npm install -g sass
[sudo] contraseña para alejandro:
/usr/local/bin/sass -> /usr/local/lib/node_modules/sass/sass.js
/usr/local/lib
├── sass@1.19.0
│   ├── chokidar@2.1.5
│   │   ├── async-each@1.0.3
│   │   ├── braces@2.3.2
│   │   │   ├── snapdragon@0.8.2
│   │   │   └── base@0.11.2
│   │   │       └── component-emitter@1.3.0
│   ├── is-binary-path@1.0.1
│   │   └── binary-extensions@1.13.1
│   └── is-glob@4.0.1

```

Una vez instalado el paquete podemos usar el comando `sass` cuya sintaxis es:

```
sass archivoSass.scss archivoCss.css
```

Existe una opción muy interesante del comando `sass` que permite indicarle a Sass que vuelva a generar el archivo CSS cada vez que hagamos modificaciones en el archivo `sass`:

```
sass --watch archivoSass.scss archivoCss.css
```

Dentro de las opciones que podemos aplicar al comando podemos destacar:

SINTÁXIS	SIGNIFICADO
<code>sass --help</code>	Muestra un mensaje de ayuda con todas las opciones y salidas.
<code>sass --scss</code>	Usa la sintaxis SCSS.
<code>sass --stdin</code>	Lee de la entrada estándar los datos en lugar de un fichero.
<code>sass --trace</code>	Muestra un resumen completo de los errores.

Una vez instalado el intérprete, la sintaxis que debe interpretar depende de la extensión de cada archivo. Sin embargo, en ocasiones el código Sass no está definido en ningún archivo y no es posible determinar automáticamente cuál es su sintaxis. Por defecto, usará la sintaxis tabulada, pero se puede usar la opción `--scss` para forzar la sintaxis SCSS. También se puede usar el comando `scss`, el cuál considera la sintaxis por defecto la de SCSS.

1.2 Variables.

Para definir variables, usaremos el carácter \$ seguido del nombre de la variable:

```
$variable: valor;
```

Por ejemplo:

```
$altura: 15px;  
$anchura: $altura+22px;
```

```
#capaPrincipal {  
    width: $altura;  
    height: $anchura;  
}
```

Este código al compilarlo, generaría el siguiente CSS:

```
#capaPrincipal {  
    width: 15px;  
    height: 37px;  
}
```

1.3 Operadores aritméticos.

Podemos utilizar los operadores habituales de los lenguajes de programación. En este caso, todos los tipos de datos manejados en Sass. En concreto se soportan los operadores aritméticos básicos: suma (+), resta (-), multiplicación (*), división (/) y módulo (%), además de los típicos operadores relacionales de igualdad (==), distinto (!=), mayor que (>), mayor o igual (>=), menor que (<), menor que o igual (<=). Lo interesante es que si se realizan operaciones con números de distintas unidades, Sass las convertirá automáticamente, si es posible. Por ejemplo:

```
p {  
    width: 1in+8pt;  
}
```

Producirá el siguiente código:

```
p {  
    width: 1.111in;  
}
```

El único problema que puede surgir a la hora de usar el operador de división, ya que en CSS este carácter es un shorthand que se usa para especificar como primer valor el tamaño de letra y el segundo el alto de línea. Por ejemplo:


```
p {
  font: 12px/10px sans-serif;
  /*Especifica 12 píxeles de tamaño de letra y 10 píxeles como
alto de línea.*/
}
```

Este código sass generaría el mismo código en CSS. Sin embargo, existen ciertas situaciones en las que este carácter se va a interpretar siempre como una operación aritmética:

- Si uno de los operandos de la división es una variable o el resultado devuelto por una función.
- Si el valor está encerrado en paréntesis.
- Si el valor se utiliza como parte de una operación matemática.

```
p {
  font: 12px/10px sans-serif;
  /*Especifica 12 píxeles de tamaño de letra y 10 píxeles como
alto de línea.*/
  $anchura: 1000px;
  width: $anchura/2;
  width: round(2.4)/2;
  height: (445px/2);
  padding: 5px + 8px/2px;
}
```

Si lo que necesitamos es utilizar este carácter de forma normal en todas las situaciones usaremos #{}:

```
p {
  $tamano-fuente: 12px;
  $tamano-interlineado: 18px;
  font: #{$tamano-fuente}/#{$tamano-interlineado};
}
```

1.4 Anidando.

Al igual que Less, Sass permite anidar las reglas CSS para que las hojas sean más claras y fáciles de escribir. A los selectores interiores se les prefija el selector padre una vez que se compile:

```
#principal p {
  color: blue;
```

```
padding: 1.5em;
.cajaroja {
  background-color:red;
  color: #000000;
}
}
```

Tendríamos el siguiente código CSS:

```
#principal p {
  color: blue;
  padding: 1.5em;
}
```

```
#principal p .cajaroja {
  background-color:red;
  color: #000000;
}
```

En ciertas ocasiones es necesario modificar el comportamiento de ciertos selectores, utilizando pseudoclasas como *hover*, *visited*, *link*... Para ello usaremos el selector `&`. Este carácter especial `&` se sustituirá por el selector padre cuando se transforme el código:

```
#principal a {
  text-decoration:none;
  color: red;
  &:hover {
    text-decoration: underline;
    font-size: 1.3em;
  }
}
```

Este código se transformará en el siguiente CSS:

```
#principal a {
  text-decoration:none;
  color: red;
}

#principal a:hover {
  text-decoration: underline;
  font-size: 1.3em;
}
```

```
}
```

Se le pueden aplicar más propiedades a este selector. Si el selector se acompaña de un sufijo, éste se le aplicará al selector padre:

```
#principal {  
  color: #00ff00;  
  &-lateral {  
    border: 1px solid #11ee00;  
  }  
}
```

El código CSS:

```
#principal {  
  color: #00ff00;  
}  
  
#principal-lateral {  
  border: 1px solid #11ee00;  
}
```

1.5 Propiedades anidadas.

En CSS existen cierto tipo de propiedades que están agrupadas. Normalmente este tipo de reglas usan un método abreviado que resulta algo tedioso de comprender. En Sass, además de poder anidar selectores, podemos anidar este tipo de reglas. Una de las reglas que usa este método es la propiedad font con todas sus variantes:

```
#principal {  
  font: {  
    family: sans-serif;  
    size: 3em;  
    weight: bold;  
  }  
}
```

El resultado en CSS sería:

```
#principal {  
  font-family: sans-serif;
```

```

font-size: 3em;
font-weight: bold;
}

```

1.6 Reglas @ y directivas.

Cuando hablamos de reglas @ (o “reglas at”) nos estamos refiriendo a reglas definidas por CSS3. Además, Sass incluye reglas específicas llamadas directivas.

1.6.1 @import

Usamos “@import” para importar archivos CSS, SCSS y Sass. Todos los archivos importados, acaban fusionándose antes de generar el archivo CSS final. De esta forma, cualquier mixin o variable definido en un archivo importado se podrán usar en el archivo principal en el que se importan.

Esta regla tiene como parámetro el archivo a importar. Por defecto busca un archivo Sass y lo importa directamente, . Si no se indica extensión, Sass tratará de buscar un archivo con ese nombre y con las extensiones .scss o .sass.

Regla @	Resultado al compilar
@import “variables.scss”;	Importa el archivo “variables.scss”
@import “variables”;	Importa el archivo “_variables.scss”
@import “estilos.css”;	@import “estilos.css”
@import url(variables)	@import url(variables);
@import “http://variables.com/bar”	@import “http://variables.com/bar”
@import “bordes”, “sombras”;	Importa los dos archivos sass.

1.6.2 Hojas de estilos parciales.

Hay veces que necesitamos importar un archivo SCSS o Sass, pero no es necesario que se compile como archivo CSS. Es decir, imagina que tienes un archivo solo con las variables, otro solo con mixins... en principio no es necesario generar un archivo CSS para ellos. Para ello se utilizará un guión bajo como primer carácter del nombre del archivo. De esta forma, Sass no generará un archivo CSS para esa hoja de estilos, pero se puede utilizar importándola dentro de otra hoja de estilos. A este tipo de archivos que no se compilan se les llama “hojas de estilos parciales”.

A la hora de importar estos archivos, no es necesario indicarlo en la regla @import.

Por ejemplo, si tenemos un archivo dedicado a almacenar todas las variables que utilizaremos, llamado *_variables.scss* y otro archivo principal, el cuál compilaremos, llamado *main.scss*, colocaríamos al principio del documento principal:

```
@import “variables”;
```

1.6.3. @media

Este tipo de reglas funcionan igual que en CSS, salvo que se pueden anidar dentro de las reglas CSS. Si incluimos una regla *@media* dentro de una regla CSS, ésta se aplicará a todos los selectores que se encuentren desde esa regla hasta el primer nivel de la hoja de estilos. Esto lo hace realmente útil para definir estilos dependientes de los dispositivos sin tener que repetir los selectores. Por ejemplo:

```
.lateral {  
  width: 300px;  
  @media screen and (orientation: landscape) {  
    width: 500px;  
  }  
}
```

Se compilará de la siguiente forma:

```
.lateral {  
  width: 300px;  
}  
@media screen and (orientation: landscape){  
  .lateral {  
    width: 500px;  
  }  
}
```

Además, las reglas *@media* también pueden contener variables, funciones y operadores, tanto en los nombres como en los valores:

```
$dispositivo: screen;  
$caracteristica: webkitmin-device-pixel-ratio;  
$valor: 1.5;
```

```
@media #{$dispositivo} and ($caracteristica: $valor){  
  .lateral {  
    width: 500px;  
  }  
}
```

Esto se compilará como:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5){
```

```
.lateral{
  width: 500px;
}
}
```

1.6.4. @extend

Aplicaremos esta regla cuando queramos extender a otras, un concepto parecido al de herencia en algunos lenguajes de programación. Es decir, a veces es necesario usar los estilos aplicados a algún selector y ampliarlos con algunos otros. Por ejemplo, imagina que tenemos los siguientes estilos:

```
.animal {
  color: brown;
  font-size: 13px;
}
.perro {
  text-decoration: underline;
}
```

Siempre que queramos utilizar la clase perro, deberíamos usar también la clase animal:

```
<div class="animal perro">...</div>
```

Si hacemos lo siguiente:

```
.animal {
  color: brown;
  font-size: 13px;
}
.perro {
  @extend .animal;
  text-decoration: underline;
}
```

Ahora todos los estilos definidos para el selector *.animal* también se aplican automáticamente al selector *.perro* por lo que no sería necesario aplicar esta clase en la capa correspondiente. Cualquier otra regla que se aplique a *.animal* se aplicará igualmente a *.perro*.

Cuando usamos la directiva *@media* existen algunas restricciones a la hora del uso de *@extend*. Por ejemplo no está permitido extender dentro de una directiva *@media* si la clase que se quiere extender está fuera del ámbito de *@media*. Es decir:

- Incorrecto:

```
.animal {
  color: brown;
```

```

    font-size: 13px;
}
@media print {
    .perro {
        @extend .animal;
        text-decoration:underline;
    }
}

```

- Correcto:

```

@media print {
    .animal {
        color: brown;
        font-size: 13px;
    }
    .perro {
        @extend .animal;
        text-decoration:underline;
    }
}

```

1.7 Directivas de control y expresiones.

Este tipo de directivas de control y expresiones nos servirán para incluir estilos solamente si se cumplen determinadas condiciones o normalmente se suelen utilizar para incluir el mismo estilo varias veces con ligeras variaciones, por ejemplo, en sistemas de Grid (tal y como hace el famoso framework Bootstrap).

1.7.1 La directiva @if.

Esta directiva evalúa una condición y aplica los estilos que se encuentren dentro de ella si dicha comprobación devuelve un valor distinto de false o null. Por ejemplo:

```

p {
    if 1+1==2 {border: 1px solid black;}
    if 5<3 {border: 2px dotted red;}
}

```

Este código se compilará en:

```
p {  
    border: 1px solid black;  
}
```

A su vez, y como en los lenguajes de programación, la directiva `@if` puede ir seguida de una `@else if` o una `@else`. Por ejemplo:

\$tipo: gato;

```
p {  
    @if $tipo == perro {  
        color: brown;  
    } @else if $tipo == gato {  
        color: black;  
    } @else if $tipo == pato {  
        color: blue;  
    } @else {  
        color: red;  
    }  
}
```

Este código se compilará como:

```
p {  
    color: black;  
}
```

1.7.2 La directiva `@for`

Esta directiva muestra repetidamente un conjunto de estilos. En cada repetición se usa el valor de una variable que hace la función de contador. Esta directiva usa dos sintaxis:

- `@for $var from <inicio> through <final>`
- `@for $var from <inicio> to <final>`

La variable `$var` puede ser cualquier variable, mientras que `<inicio>` y `<final>` son expresiones que deben devolver números enteros (o un número entero directamente). La diferencia entre las dos sintaxis es la siguiente: la sintaxis que usa *through* los estilos se repiten desde `<inicio>` hasta `<final>` ambos inclusive, en la sintaxis que usa *to* los estilos se repiten desde `<inicio>` hasta `<final>` sin incluir `<final>`.

Por ejemplo, para crear un grid de 12 columnas por fila podríamos hacer:

```
@for $i from 1 through 12 {  
  .columna-#{ $i } {width: 100% / $i;  
}
```

Este código se compilará como:

```
.columna-1 {width: 100%;}  
.columna-2 {width: 50%;}  
...  
columna-12 {width: 8.33%;}
```

1.7.3 La directiva @each.

La sintaxis de la directiva @each es:

```
@each $variable in <lista o mapa> {  
  estilos  
}
```

Con esta directiva se recorre toda la lista o mapa y en cada iteración, se asigna un valor diferente a la variable *\$variable* antes de compilar los estilos. Por ejemplo:

```
@each $animal in leon, puma, perro, gato {  
  .#{animal}-icon {  
    background-image: url('/imagenes/#{ $animal }.png');  
  }  
}
```

El código Sass se compila de la siguiente forma:

```
.leon-icon {  
  background-image: url('/imagenes/leon.png');  
}  
.puma-icon {  
  background-image: url('/imagenes/puma.png');  
}  
.perro-icon {
```

```

    background-image: url('/imagenes/perro.png');
}
.gato-icon {
    background-image: url('/imagenes/gato.png');
}

```

1.7.4 La directiva @while

Funciona igual que una estructura *while* de cualquier lenguaje de programación. Sintaxis:

```

@while condición {
    estilos;
}

```

Por ejemplo:

```

$contador: 5;
@while $i>0 {
    .capa-#{ $i } {width: 2em * $i;}
    $i: $i - 2;
}

```

Esto se traducirá en:

```

.capa-6 {width: 12em;}
.capa-4 {width: 8em;}
.capa-2 {width: 4em;}

```

1.8 Mixins

Los mixins, al igual que ocurría con Less, permite definir estilos reutilizables en toda la hoja de estilos. Los mixins también contener reglas CSS y cualquier otro tipo de elemento definido por Sass, incluso adminten el uso de argumentos, como si usáramos funciones en un lenguaje de programación.

Los mixins se definen con la directiva *@mixin* seguida del nombre del mixin y los argumentos (si es que los tiene). A continuación se establece el contenido que definen los estilos del mixin. Por ejemplo:

```

@mixin texto-grande {

```

```

font: {
    family: Arial;
    size: 20px;
    weight: bold;
}
color: #ff0000;
}

```

Para poder usar los mixins es necesario usar la directiva *@include* seguida del nombre del mixin. Es decir:

```

#principal .titulo {
    @include texto-grande;
    padding: 4px;
    margin: 10px 0 0 0;
}

```

Este código se traduce a:

```

#principal .titulo{
    font-family: Arial;
    font-size: 20px;
    font-weight: bold;
    color: #ff0000;
}

```

Uso de argumentos.

Los argumentos pueden estar formados por cualquier expresión. Estos argumentos estarán disponibles dentro del mixin en forma de variable. Al definir un argumento, estos aparecerán como variables separadas por comas y entre paréntesis. Cuando se use el mixin, se deben poner estos argumentos en el mismo orden.

Por ejemplo:

```

@mixin borde-estilo ($color, $ancho){
    border: {
        color: $color;

```

```

        width: $anchura;
        style: dashed;
    }
}

```

```
p { @include borde estilo(blue, 1in);}
```

El código anterior se traduce en:

```

p {
    border-color: blue;
    border-width: 1in;
    border-style: dashed;
}

```

Para establecer un valor por defecto a un argumento, establecemos ese valor directamente al lado del argumento y separado por “:”, es decir:

```

@mixin borde-estilo ($color, $ancho: 5px){
    border: {
        color: $color;
        width: $anchura;
        style: dashed;
    }
}

```

Ahora si se realiza la llamada y no se establece valor para ese argumento, tomará por defecto el valor establecido: 5px. Podríamos hacer la llamada al mixin de la siguiente forma:

```

        @include borde-estilo(red);
o
        @include borde-estilo(red, 10px);

```

2. Caso práctico – Página principal IES Celia Viñas.

Para este apartado se realizará el caso práctico realizado con Less pero usando la sintaxis Sass. Es por esto que solo se comentará el código Sass generado.

Igualmente tendremos los siguientes archivos:

- variables.scss. Archivo destinado a almacenar las principales variables que configurarán el archivo de estilos.
- mixins.scss. Código reutilizable que se usará en el código sass del archivo principal, *layout.scss*.
- galeria.scss. Estilos para la galería de fotos.
- layout.scss. Archivo principal scss de la página web.

2.1 Principales diferencias.

El código generado entre Sass y Less no tiene muchas diferencias, simplemente adaptando la sintaxis entre uno y otro y algunos cambios menores obtenemos el mismo resultado.

Por ejemplo la función que usamos para establecer un color de fondo del área de navegación es propia de Sass:

```
adjust-hue(lighten($colorPrincipal, 25%), 8);
```

También sufre algún cambio el código utilizado para formar las filas y columnas de la interfaz, ubicado en el archivo mixins.scss:

```
//Especifica un posicionamiento flotante a la izquierda para cada elemento.
```

```
@mixin columna-flotante {  
    float: left;  
    box-sizing: border-box;  
}
```

```
//Calcula la anchura para cada columna, dependiendo del número de columnas que se usen.
```

```
@mixin columna($n) {  
    width: $n*(100% / $columnas);  
    @include columna-flotante;  
}
```

```
//Llamada para la generación de los estilos para las columnas
```

```
@mixin grid($columnas){  
    @for $i from 1 through $columnas {  
        .col-#{ $i } {  
            @include columna($i);  
        }  
    }  
}
```