



# Less.js

Antonio Espín Herranz

# Contenidos

- Introducción.
- Compiladores.
  - Instalar compilador de less y monitor de less.
- Uso del compilador.
- Estructura de los archivos de less.
- Sintaxis:
  - Variables, anidaciones, operaciones
- Mixins:
  - No paramétricos, paramétricos, condicionales
- Ejemplos: Diseño de botones y un grid basado en columnas.
- Enlaces

# Introducción

- Compilador de CSS
  - Escribimos ficheros less que al compilar generan un fichero css.
- Enlace a la librería:
  - <https://lesscss.org/>
- Compilador en línea de less:
  - <https://winless.org/>

# Instalación de less

- **Con node.js**
  - **npm install -g less** (instalación a nivel global)
  - Para compilar los archivos:
  - **lessc styles.less styles.css**
  - Si lo tenemos que desinstalar haremos: **npm uninstall -g package-name**
  - Less compiler + monitoring
    - **npm install -g less-watch-compiler**
  - Para monitorizar una carpeta:
    - **less-watch-compiler less css**
    - Comprueba si hay cambios y compila los archivos de less y genera los css.

# Como trabajar con less

- Crear una carpeta para los ficheros de **less** y otra para los **css**.
- Arrancar el monitor de less y asignarle las carpetas.
  - **less-watch-compiler less css**
- Colocar el workspace de code en la carpeta inmediata superior y abrir los dos archivos en el momento que hacemos cambios en el archivo less y grabamos se plasman en el fichero css.

# Como trabajar con less

```
{} tess.less  ×  [icon] ...
less > {} tess.less > [icon] p
1  // ejemplo de fichero de less.
2
3  @color: [icon] red;
4
5  h1 {
6      background: @color;
7  }
8
9  p {
10     color: @color;
11 }
```

```
# tess.css  ×  [icon] ...
css > # tess.css > [icon] h1
1  h1 {
2      background: [icon] red;
3  }
4  p {
5      color: [icon] red;
6  }
7
```

# Estructura de los archivos de Less

- Podemos tener varios ficheros de less para importarlos en otro fichero.
  - Por ejemplo:
    - Las variables: Se definen con una @ seguidas de identificador
    - CSS Reset: quitando padding, margin, border, a todas las etiquetas HTML
    - Otro con la tipografía
    - Otro para el layout
  - Y después se incluyen todos en otro fichero: **style.less** para generar **styles.css**

# Estructura de los archivos de Less

- Importar todos los archivos a **styles.less**
- `@import "variables.less"`
- `@import "layout.less"`
- `@import "reset.less"`
- `@import "tipografia.less"`
- Permite definir estructuras de CSS genéricas que según nos interese podemos importar o no un archivo.
- Los archivos de less no se suben a producción (sólo el CSS compilado).
- Less permite la optimización del flujo de trabajo



# Estructura de los archivos de Less

- Se pueden importar archivos de less dentro de clases.
- En variables.less definimos una variable con un valor que referenciamos desde el layout.less.
- Si creamos una librería donde modificamos el valor de esa misma variable podemos importar la nueva librería dentro de la clase:
- Por ejemplo: en **layout.less** tenemos:

```
article {  
    float:left;  
    width:50%;  
    @import "small.less"  
    margin:@gutter;  
}
```

```
variables.less  
@gutter:24px
```

```
small.less  
@gutter:12px
```

# Estructura de los archivos de Less

- Se pueden añadir **media queries** y esto habrá que hacerlo dentro de nuestro archivo principal.

- Se puede hacer así:

```
@media screen and (min-width: 320px) {  
  @import "320.less";  
}
```

- Mejor utilizando variables:

```
@media screen and (min-width: @bp1) {  
  @import "bp1.less";  
}
```

**@bp1** Se añade la variable al fichero: variables.less

Y tendremos un archivo nuevo: **bp1.less**

- **De esta forma lo podemos ir variando de un proyecto a otro.**

# Sintaxis: Variables

- Podemos sacar los colores a variables a un solo archivo de less.
- Y luego esas variables se pueden referenciar en varios archivos de less.
  - Por ejemplo: en tipografia.less y layout.less
- **variables.less**
  - **@basecolor:** #280501;
  - **@secondarycolor:** #E4E5D3;

## **Layout.less**

```
body {  
    padding: 24px;  
    margin: 0;  
    background-color: @basecolor;  
}
```

# Sintaxis: Variables

- Las variables se pueden copiar:
- `@elementobg: @basecolor`
- Las variables pueden tener texto (comillas simples)
  - `@separador: '- ';`
- Se puede aplicar a:
- `a:link::before{`
  - `content: @separador;`
- `}`

# Sintaxis: Variables

- Para las urls también vienen muy bien.
  - Podemos tener diferentes rutas a las imágenes en desarrollo que en producción.
  - Es otro sitio donde pueden venir bien las variables.
  - @imgpath: 'img' → la supuesta carpeta donde están las imágenes.
- Se pueden utilizar luego para establecer una url:

```
article {  
    /* Partimos de : */  
    background-image: url("img/bg_image.png");  
  
    /* Para utilizar la variable ponemos: */  
    background-image: url("@{imgpath}/bg_image.png");  
}
```
- Al subirlo a producción si tenemos otra ruta para la imagen:
  - @imgpath: 'http://cdn.dominio.com/imagenes'; → Nos evita tocar las url en CSS.

# Sintaxis: Variables

- Una variable permite añadir operaciones con las variables.
- Por ejemplo, tenemos un `margin:24px` en algunas clases y otras tenemos `12px`.
- Podemos definir una variable `@margen:24px` y luego hacer operaciones para pasar a la mitad.
- Si hacemos operaciones ponerlas entre paréntesis cuando se acompañan de otras variables u operaciones.

- **variables.less**

- `@margen: 24px;`

```
article {  
    margin: @margen / 2;  
}
```

Si hacemos: **`margin: (@margen / 2) @margen;`** /\* Es la mitad en vertical y 24 en los laterales.\*/

# Sintaxis: Variables

- Las variables pueden interactuar con JavaScript:
- `@longitud: "cadena de texto".length`;`
  - Ojo con las comillas invertidas para que lo interprete como código JS.
- También podemos hacer:
- `@texto: "Cadena de texto";`
- `@longitud: `"@{texto}".length`;`
- Para compilar con JS, desde una consola:
- **`lessc ./less/estilos.less ./css/estilos.css`**

# Sintaxis: Anidaciones

- Ojo no hacer todo el código de CSS anidado en less:

```
article {  
  background-color: olive;  
  header {  
    font-size: 1.5em;  
  }  
}
```

- **Genera:**

```
article {  
  background-color: olive;  
}  
article header {  
  font-size: 1.5em;  
}
```



# Sintaxis: Anidaciones

- El navegador lee los selectores de izq a der cuantos más haya más va a tardar el navegador en pintar / renderizar la página.

- `/* MALAS PRACTICAS */`

```
body{  
  nav{  
    ul{  
      li{  
        a{  
          color:orange;  
        }  
      }  
    }  
  }  
}
```

```
body nav ul li a {  
  color: orange;  
}
```

- Funciona, pero está poco optimizado. Estamos poniendo selectores de más:

```
nav{  
  li{  
    a{  
      color: olive; // ul no es necesario y body tampoco  
    }  
  }  
}
```

# Sintaxis: Anidaciones descendientes

- **Less**

```
p {  
  font-family: Helvetica, Arial, Verdana;  
  font-size: 1em;  
  color: #222222;  
  .highlight {  
    color: orange;  
  }  
  &.small {  
    font-size: .8em;  
  }  
}
```

- `<p class="small"><span class="highlight">Lorem</span> ipsum, dolor sit amet consectetur adipisicing elit. Repellat unde beatae reiciendis nemo dolores deserunt? corrupti, dignissimos consectetur a.</p>`

- **CSS generado:**

```
p {  
  font-family: Helvetica, Arial, Verdana;  
  font-size: 1em;  
  color: #222222;  
}  
p .highlight {  
  color: orange;  
}  
p.small {  
  font-size: 0.8em;  
}
```

- ***& sirve para eliminar los espacios en less***

# Sintaxis: Anidaciones descendientes

```
p {  
  font-family: Helvetica, Arial, Verdana;  
  font-size: 1em;  
  color: #222222;  
  .highlight {  
    color: orange;  
  }  
  &.small {  
    font-size: .8em;  
    &::after {  
      content: " leer más";  
      color: orange;  
    }  
  }  
}
```

```
p {  
  font-family: Helvetica, Arial, Verdana;  
  font-size: 1em;  
  color: #222222;  
}  
p .highlight {  
  color: orange;  
}  
p.small {  
  font-size: 0.8em;  
}  
p.small::after {  
  content: " leer mas";  
  color: orange;  
}
```

# Sintaxis: Operaciones

- Transferencia de variables. Se encierran entre llaves:
  - @var1: “valor1”;
  - @var2: “valor2”;
  - @todo: “@{var1} y @{var2}”;
  - color: @todo;
- Genera en CSS:
  - color: “valor1 y valor2”;
- No es obligatorio poner las cadenas con comillas, pero es aconsejable.

# Sintaxis: Operaciones

- Con variables numéricas:

@numero1: 10;

@numero2: 20;

@suma: @numero1 + @numero2;

@resta: @numero1 - @numero2;

@division: @numero1 / @numero2;

@mul: @numero1 \* @numero2;

```
.contenido {  
    content: (@suma *2) + @resta;  
}
```

- En el momento que cambiamos el valor de una variable y compilamos, se calcula el valor y se sustituye.
- Se puede crear varias operaciones con los paréntesis.

# Sintaxis: Operaciones

- Se pueden utilizar funciones para cálculos matemáticos:
  - @cos: cos(1);
  - @funcion: ceil(@cos);
- También trabaja con funciones destinadas a los colores.
- @negro: #000;
- @blanco: #fff;
- @verde: #00ff00;
- @bordercolor: **darken**(@verde, 20%) → oscurecer al 20% el color.

# Ejemplo

```
/* Funciones aplicadas a colores */
```

```
@negro: #000;
```

```
@blanco: #fff;
```

```
@verde: #00ff00;
```

```
@bordercolor : darken(@verde, 20%);
```

```
.elemento {  
    border-top: 1px solid @bordercolor;  
}
```

•

```
.elemento {  
    border-top: 1px solid #009900;  
}
```

•

# Ejemplo: escala de grises

- @blanco: #fff;
- @grisoscuro:       darken(@blanco, 80%);
- @gris:                darken(@blanco, 60%);
- @grisclaro:         darken(@blanco, 40%);
- @grismasclaro:      darken(@blanco, 20%);



# Más funciones sobre colores

- Se pueden obtener los colores complementarios:
- `@complementario_verde: spin(@verde,180);`
- Gira el valor de la variable 180°.
- Obtendremos el valor opuesto al verde: #00FF00

# Más funciones sobre colores

- Las funciones se pueden anidar:
- @color: **darken**(**spin**(@verde, 180), 70%);
- Primero calcula el color complementario y luego lo oscurece al 70%.
- **fade**(color, %) se puede aplicar a un box-shadow

# Ejemplo

- **lighten** color y porcentaje. Realiza un degradado:

@baseheadersize: 2.5em;

@baseheadercolor: #280501;

h1 { font-size:@baseheadersize; color: baseheadercolor; }

h2 { font-size:@baseheadersize \* .8; color: lighten(baseheadercolor,10); }

h3 { font-size:@baseheadersize \* .6; color: lighten(baseheadercolor,20); }

h4 { font-size:@baseheadersize \* .4; color: lighten(baseheadercolor,30); }

h5 { font-size:@baseheadersize \* .2; color: lighten(baseheadercolor,40); }

# Resultado

```
h1 {  
  font-size: 2.5em;  
  color: #280501;  
}  
h2 {  
  font-size: 2em;  
  color: #5a0b02;  
}  
h3 {  
  font-size: 1.5em;  
  color: #8c1103;  
}...
```

# Operaciones

- Se pueden multiplicar los colores por un valor y supera en este caso el color #FFFFFF (blanco) ya se va a quedar con ese valor.
- Por ejemplo, operaciones de multiplicación por un valor:

```
p {  
  color: @baseheadercolor * 2;  
}
```

- Mas funciones:

<https://lesscss.org/functions/#color-operations>

# Mixins no paramétricos

- Un mixin no paramétrico **es una clase de CSS** que se puede añadir a otras clases de CSS para **evitar la duplicidad de estilos entre clases distintas**.
- Los podemos utilizar cuando tengamos estilos que se repiten en distintos selectores.
- Por ejemplo, tenemos dos clases distintas, pero tienen los colores en común.
  - Estos colores los podemos sacar a otra clase e incluso esa clase se puede guardar en otro fichero **less** distinto.
  - El nuevo **mixins** tendrá esos estilos en común y la clase que los representa se añade directamente (anidando) en las clases anteriores.

# Ejemplo

- **mixins.less**

// Mixins NO paramétricos

```
.square {  
    background-color: @basecolor;  
    border: 2px solid @bordercolor;  
}
```

- **En layout.less**

```
.square1 {  
    width: 100px;  
    height: 100px;  
    float: left;  
    .square;  
}
```

```
.square2 {  
    width: 200px;  
    height: 100px;  
    float: right;  
    .square;  
}
```

- **En styles.less**

```
@import "variables.less";  
@import "mixins.less";  
@import "layout.less";
```

# Mixins paramétricos

- Mixins con parámetros.
  - Cuando se declaran se añaden parámetros a las clases que estamos definiendo como si fuera una función.

- **Ejemplo**

```
.max-width {  
    max-width: @maxwidth;  
}
```

```
.font-size(@size){  
    declaración  
    font-size: @size;  
}
```

```
article {  
    .font-size(16px); // Utilización  
}
```



```
.max-width {  
    max-width: 960px;  
}  
article {  
    font-size: 16px;  
}
```



# Ejemplo

- Mixin paramétrico:

```
.border-box(@box){  
  -webkit-box-sizing: @box;  
  -moz-box-sizing: @box;  
  -ms-box-sizing: @box;  
  box-sizing: @box;  
}
```
- Utilización:

```
article {  
  .border-box(content-box); // Nos genera las 4 líneas.  
}
```
- Resultado CSS:

```
article {  
  -webkit-box-sizing: content-box;  
  -moz-box-sizing: content-box;  
  -ms-box-sizing: content-box;  
  box-sizing: content-box;  
}
```
- **content-box** es el comportamiento CSS por defecto para el tamaño de la caja (box-sizing).

# Mixin paramétrico con valor por defecto

```
.border-box(@box:border-box){  
  -webkit-box-sizing: @box;  
  -moz-box-sizing: @box;  
  -ms-box-sizing: @box;  
  box-sizing: @box;  
}
```

- *Cuando no se rellena el parámetro, las propiedades toman el valor border-box por defecto.*
- <https://midu.dev/que-es-y-para-que-sirve-box-sizing-border-box/>

- Posibles usos:

```
article {  
  .border-box(content-box);  
}
```

```
section {  
  Border-box();  
}
```

```
header {  
  .border-box(); // Toma el valor por defecto  
}
```

*box-sizing: **border-box***

*Incluyen el padding y el borde para calcular el tamaño de la caja.*

# Mixins condicionales

- Se pueden crear mixins condicionales y que ejecuten una regla en función del valor de una variable:

- En el layout:

```
// Variables:
```

```
@colorbase: olive;
```

```
// Posibles opciones: vivo / opaco
```

```
@color_esquema: opaco;
```

```
h1 {
```

```
  .esquema(@color_esquema);
```

```
}
```

```
h2 {
```

```
  .esquema(vivo);
```

```
}
```

# Mixins condicionales

// Elige uno u otro en función del valor que aplicamos:

```
.esquema(opaco;) {  
    color: darken(@colorbase, 20%);  
}  
  
.esquema(vivo;) {  
    color: lighten(@colorbase, 20%);  
}
```

## RESULTADO CSS:

```
h1 {  
    color: #1a1a00;  
}  
h2 {  
    color: #e6e600;  
}
```

Si añadimos un valor que no sea ni vivo  
Ni opaco obtendremos un error.

# Ejemplo I : Diseño de Botones

- Se trata de crear unos mixins parametrizados para generar botones donde el color será un degradado vertical que se calcula a partir del color base que indicamos y vamos a poder ajustar otros parámetros como el radio de las esquinas.
- Generar botones a partir de otros colores como: error, success, info o alert.
- Se partirá en 4 ficheros de less.

# Botones

- En el fichero principal: **styles.less**
  - @import "variables.less";
  - @import "mixins.less";
  - @import "botones.less";
  - @import "layout.less";

# Botones

- En este ejemplo vamos a tener 4 ficheros.
- Uno con las variables: **variables.less**
- El fichero **mixins.less** contendrá **2 mixins paramétricos**:
  - Uno será **vertical** que recibe el **color origen y destino** (en el degradado se suele indicar un color origen y otro destino).
  - Otro radio del botón que se encarga de redondear esquinas. Por si los navegadores no soportaran la propiedad **border-radius** podemos indicar dentro del mixin dicha propiedad según el navegador:
  - Por ejemplo:
    - `-webkit-border-radius: @radius;`
    - `-moz-border-radius: @radius;`
    - `-ms-border-radius: @radius;`
    - `-o-border-radius: @radius;`
    - `border-radius: @radius;`

# Botones

- En el fichero **botones.less** creamos el mixins principal para generar un **botón** que recibe el color y este llamará a los otros dos mixins: **vertical** y **border-radius** a parte rellena varias propiedades:
  - El borde.
  - Color de texto, la sombra del texto
  - Height, padding, etc.



# Botones

- En el layout hacemos:

```
ul {  
    margin:0;  
    padding: 0;  
    li {  
        .boton(@alertcolor); // Llamar al mixin principal con un color.  
    }  
}
```

# Botones

- **En el index.html**

```
<link rel="stylesheet" href="css/style.css">
</head>
<body>
  <ul>
    <li>Botón 1</li>
    <li>Botón 2</li>
    <li>Botón 3</li>
  </ul>
</body>
</html>
```

# Ejemplo II: Grid

- Diseño de columnas:



# Grid

- El área total de las columnas: el ancho de las columnas y la separación del margen entre las columnas.



- El **total** será:  
 $(\text{columnas} * \text{ancho\_columnas}) + (\text{columnas} * \text{ancho\_margen})$
- Cada columna tiene 10px por cada lado.
- La primera y la última tendrán 10px y las centrales son 10px por cada lado.

# Grid

- El fichero principal: **styles.less**

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

```
@import "grid.less";
```

```
@import "layout.less";
```

- El fichero de variables: **variables.less**

```
@columnas :12;
```

```
@ancho_margen:20;
```

```
@ancho_columnas : 60;
```

# Enlaces

- Bootstrap Customize:
  - <https://getbootstrap.com/docs/5.0/customize/overview/>
- Bootstrap **Less**
  - <https://getbootstrap.com/2.0.4/less.html>
- Página principal de **less**: <https://lesscss.org/>
- [https://es.wikipedia.org/wiki/LESS\\_\(lenguaje\\_de\\_hojas\\_de\\_estilo\)](https://es.wikipedia.org/wiki/LESS_(lenguaje_de_hojas_de_estilo))