DISEÑO DE INTERFACESWEB
TEMA 2: PREPROCESADORES CSS

JESUITAS
Sagrado Corazón
Jesuitas · Logroño

Índice

1. ¿Qué son los pre-procesadores CSS?

- Si tienes algo de experiencia en el desarrollo de sitios web es probable que hayas tenido muchos dolores de cabeza trabajando con hojas de estilos CSS, esto, sobretodo porque son muy difíciles de mantener ordenadas y al no ser propiamente un lenguaje de programación tienen limitantes como el no poder utilizar variables, funciones, realizar operaciones matemáticas, etc.

- La buena noticia es que no eres la única persona con ese problema y algunas de ellas ya se han dedicado ha desarrollar soluciones para este problema y así es como nacen los **pre-procesadores CSS.**

1. ¿Qué son los pre-procesadores CSS?

- Son aplicaciones que procesan cierto tipo de archivos para crear hojas de estilos con algunas de las ventajas de un lenguaje de programación común como el uso de variables, funciones, condiciones, la posibilidad de hacer cálculos matemáticos y que además te permiten utilizar una sintaxis mucho más sencilla e intuitiva.
- Hay muchos pre-procesadores, cada uno con sus ventajas, los 3 más populares: **Sass**, **LESS** y **Stylus.**

2. Instalacción

- Tanto **Sass** como **LESS** y **Stylus** están escritos para funcionar en distintas plataformas.
- **Sass** funciona con Ruby el cual viene pre-instalado en Mac, sin embargo en Windows necesitarás instalar Ruby para poder empezar a usar Sass. Por otro lado para poder utilizar **Sass** tendrás que utilizar la terminal o la línea de comandos lo que vuelve algo complicada la tarea(para algunos). Existen varias herramientas visuales para facilitar este proceso pero en general tienen un costo.

2. Instalacción

- En el caso de **LESS** al estar escrito en JavaScript la instalación es tan sencilla como importar la librería de JavaScript a tu documento HTML, además, también existen algunas herramientas, muchas de ellas gratuitas, que ayudan a compilar tus archivos escritos en **LESS**.
- **Stylus** requiere para su instalación de NodeJs el cual que tiene su propio proceso de instalación y al igual que Sass requiere de trabajo desde línea de comandos para instalar y compilar documentos. Por otro lado las herramientas visuales para ello son más escasas que sus 2 competidores.
- Conclusión: Para cuestiones de instalación **LESS** es la mejor opción.

2. Instalacción

- En el caso de **LESS** al estar escrito en JavaScript la instalación es tan sencilla como importar la librería de JavaScript a tu documento HTML, además, también existen algunas herramientas, muchas de ellas gratuitas, que ayudan a compilar tus archivos escritos en **LESS**.
- **Stylus** requiere para su instalación de NodeJs el cual que tiene su propio proceso de instalación y al igual que Sass requiere de trabajo desde línea de comandos para instalar y compilar documentos. Por otro lado las herramientas visuales para ello son más escasas que sus 2 competidores.
- Conclusión: Para cuestiones de instalación **LESS** es la mejor opción.

3. Sintaxis

- Tanto **Sass** como **LESS** utilizan la sintaxis estándar de CSS por lo que es bastante sencillo convertir una hoja de estilos al formato de **Sass** o **LESS**. La pequeña diferencia es que **Sass** utiliza la extensión **.scss** y **LESS** utiliza la extensión **.less.**

```
SINTAXIS less
h1 {
  color: #ACACAC; }
```

- Como verán no hay ninguna diferencia con un CSS regular. Este ejemplo funciona tanto para Sass como para LESS.
- Para el caso de Sass, este también acepta una sintaxis un poco más antigua que omite las llaves y el punto y coma:

```
SINTAXIS sass
h1
  color: #ACACAC
```

3. Sintaxis

- Para el caso de Stylus la sintaxis es un poco más compleja. Stylus acepta la sintaxis CSS estándar, pero también acepta algunas otras variaciones donde las llaves, dos puntos y puntos y comas son opcionales.
- La extensión de los archivo stylus es **.styl.**

```
SINTAXIS Stylus
h1 {
  color: #0982C1;
}
 /* omitimos llaves */
h1
  color: #0982C1;
 /* omitimos llaves, dos puntos y puntos y comas */
h1
  color #0982C1
```

3. Sintaxis

- Ningún formato ofrece una ventaja real sobre el otro salvo la posibilidad de escribir el código omitiendo ciertos elementos. Bajo este criterio podríamos declararlo un empate, sin embargo, siendo un poco más estricto una sintaxis menos abierta (sin ser demasiado restrictiva por supuesto) ayuda a tener códigos más claros y elegantes y facilitan la futura lectura del mismo. Es decir, considero que los ganadores son: **Less** y **SASS**.

4. Less

- LESS es un preprocesador de texto, como una ampliación de las hojas de estilos, que se comporta como un lenguaje de programación. Nos permite el uso de variables, funciones y operaciones aritméticas entre otras cosas.
- Hemos de tener claro que LESS no reemplaza a CSS, es más, el resultado de la compilación de LESS es un fichero CSS totalmente funcional.
- LESS funciona desde 2009 y, desde el último año se ha vuelto realmente popular. Hasta el punto de que el framework CSS del propio Twitter (Bootstrap) lo implementa.

4.1 Recursos disponibles

- El primer sitio que siempre recomiendo es lesscss.org , donde hay muy buena información y tutoriales bastante ricos en contenido.
- Si te defiendes con el inglés, es posible que te interese echarle un ojo al tutorial de introducción a LESS de Smashing Magazine

4.2 Tecnología lado cliente y servidor

- Para implementar LESS podemos utilizar tecnologías del lado cliente y del lado servidor.
- Para implementarlo del lado cliente tenemos que guardar nuestros ficheros con la extensión .less (fichero.less) e incorporarlo a nuestro HTML tal y como hacemos con un CSS.

```
<link rel="stylesheet/less" type="text/css" href="fichero.less"/>
```

- Una vez que tenemos esto hecho sólo nos falta agregar el fichero LESS.js que compilará nuestro código LESS para convertirlo en un fichero CSS.

```
<script src="less.js" type="text/javascript"></script>
```

4.2 Tecnología lado cliente y servidor

- Por otro lado, podemos instalar el compilador directamente en nuestro servidor y que él se encargue de convertirlo a CSS.

4.3. Compiladores

- Less.app y LiveReload como compiladores en Mac funcionan realmente bien (lo siento usuarios de Windows, podéis probar SimpLESS).
- LESS.app se ha convertido en Codekit (de pago), pero es una inversión que vale la pena, al igual que LiveReload.
- Koala es otra alternativa que nos sirve también para la compilación de sass.

4.4. Variables

- Para declarar una variable en LESS basta con incluir un @ previo al nombre de la variable en sí. Tras ese @ puede ir una combinación de caracteres alfanuméricos puntos y guiones bajos.
- Una vez que le hemos asignado un nombre a nuestra variable, colocamos dos puntos tal y como hacemos con una propiedad CSS y, acto seguido, incluimos el valor que tomará la variable; por ejemplo un código de color en hexadecimal o una medida en píxeles.

```
@blue: #00214D;
@red: #CF142B;
@white: #FFF;
@black: #000;
@baseFontSize: 15px;
@baseLineHeight: 22px;
```

4.4. Variables


```
@blue: #00214D;
@red: #CF142B;
@white: #FFF;
@black: #000;
@baseFontSize: 15px;
@baseLineHeight: 22px;
```

- Ahora puedes utilizar estas variables en cualquier etiqueta de tu archivo LESS simplemente haciendo la llamada a la variable.
- Por ejemplo, vamos a asignar el color que contiene la variable "blue" a todos los encabezados h1.

```
h1{ color: @blue; }
```

- Y, a continuación, vamos a asignarle el color "red" y el "baseFontSize" a todos los párrafos:

```
p { color: @red;
font-size: @baseFontSize; }
```

4.4. Variables

- Imagina que todos tus párrafos, enlaces, encabezados h4, strongs y ems llevan el mismo tamaño de fuente, ¿Cómo lo harías?

4.4. Variables

- Imagina que todos tus párrafos, enlaces, encabezados h4, strongs y ems llevan el mismo tamaño de fuente, ¿Cómo lo harías?
- Está claro que sin LESS tendríamos que colocar la propiedad “font-size: 15px;” en todas y cada una de las etiquetas antes mencionadas, pero con LESS podemos colocar “font-size: @baseFontSize;” de tal manera que cuando queramos cambiar el tamaño general de los textos de nuestra web, sólo cambiaremos el valor asignado a dicha variable y no todas las etiquetas “font-size”.

4.4. Variables


```
@blue: #00214D;
@red: #CF142B;
@white: #FFF;
@black: #000;
@baseFontSize: 15px;
@baseLineHeight: 22px;
```

- Para que te hagas una idea, el siguiente ejemplo de LESS (creado con las variables de arriba) compila como se muestra a la derecha:

```
h1 { color: @red; }
h2 { color: @blue; }
h3 { color: @black; }
p { color: @black;
font-size: @baseFontSize;
line-height: @baseLineHeight;}
```

```
h1 { color: #CF142B; }
h2 { color: #00214D; }
h3 { color: #000; }
p { color: #000;
font-size: 15px;
line-height: 22px; }
```

4.4. Variables

- TEXTO EN LAS VARIABLES
- Las variables no tiene por qué ser solamente de colores o tamaños de fuente. También podemos definir variables con textos (cadenas de caracteres), tal y como se hace en javascript o php.
- Esto se utiliza mucho cuando utilizas Icon Fonts en tu diseño web.
- Podemos asignar textos a las variables así como códigos hexadecimales o tamaños de fuente:

```
@description: "Me gusta LESS.";
```

4.4. Variables

Práctica sobre less. Para eso descarga el compilador SimpLESS y prueba los ejemplos anteriores.

4.5. Mixins

- Un mixin en LESS es un conjunto de propiedades CSS agrupadas. Esto nos permite utilizar el grupo en sí y no repetir mil veces las mismas propiedades.
- Si estuviésemos hablando de programación (como tal), esto sería lo más parecido a un array. Otra forma de entenderlo podría ser una variable con múltiples valores.
- Los mixins son, junto a las variables y funciones, uno de los tres principales pilares (y quizá de las razones de ser) de LESS.
- Para entender cómo funciona un mixin vamos a ver un ejemplo real, que es como se aprende.

4.5. Mixins

- Cuando hablamos de mixins paramétricos estamos hablando de aquellos mixins que reciben parámetros (al igual que las funciones).
- Por ejemplo, a la hora de realizar un border-radius resulta bastante pesado tener que escribir los prefijos para los diferentes navegadores cada vez, ¿verdad?
- Veamos cómo solucionarlo con un mixin

```
.border-radius(@radius) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius(4px); }
```

4.5. Mixins


```
.border-radius(@radius) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }
.sidebar { .border-radius(4px); }
```

- Si te fijas, en nuestro mixin incluimos todos los prefijos de CSS3 que permiten que el borde redondeado sea visible en webkit, mozilla… Además, como queremos reutilizar ese mixin, le asignamos el parámetro *@radius*, lo que nos permitirá asignarle diferentes valores cada vez que realicemos una llamada a dicho mixin. El resultado del código anterior, una vez compilado, es:

```
.sidebar {
-webkit-border-radius: 4px;
-moz-border-radius: 4px;
border-radius: 4px; }
```

4.5. Mixins

- Además, para no tener que volvernos locos con los valores por defecto, podemos hacer que el mixin paramétrico tenga un valor por defecto.

```
.border-radius(@radius: 6px) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius; }
.sidebar2 { .border-radius(12px); }
```

- La idea es la misma que el ejemplo anterior, sólo que ahora nuestro mixin *.border-radius* tiene un valor por defecto (que se aplica a *.sidebar*) y que puede ser fácilmente modificado (aplicado en *.sidebar2*)

4.5. Mixins


```
.border-radius(@radius: 6px) {
-webkit-border-radius: @radius;
-moz-border-radius: @radius;
border-radius: @radius; }

.sidebar { .border-radius; }
.sidebar2 { .border-radius(12px); }
```

- El código compilado seria el siguiente:

```
.sidebar { -webkit-border-radius:
6px; -moz-border-radius: 6px;
border-radius: 6px; }

.sidebar2 { -webkit-border-radius:
12px; -moz-border-radius: 12px;
border-radius: 12px; }
```

4.6. Ejemplos

- En el caso de CSS supongamos el siguiente escenario: estamos maquetando una web y tenemos un párrafo con sus reglas CSS. Además, para la etiqueta estándar podemos tener varias clases, para un párrafo de introducción y para un párrafo resaltado. Digamos que el párrafo estándar lo definimos con una fuente sans-serif con un tamaño normal, line-height, etc. Así es como se vería nuestro CSS (es sólo un ejemplo, nada de LESS):

```
p {
color: #232323;
font-family: Helvetica,
Arial, sans-serif;
font-size: 14px;
line-height: 21px; }
```

4.6. Ejemplos

- Hacemos que el tamaño de fuente sea un poquito más grande y que todo el texto sean mayúsculas. Por último, para el párrafo resaltado pondremos negrita y azul.

```
p {
color: #232323;
font-family: Helvetica, Arial, sans-serif;
font-size: 14px;
line-height: 21px; }

p .intro {
font-variant: small-caps;
font-size: 16px;
line-height: 24px; }

p .highlight {
color: #00214D;
font-weight: bold; }
```

4.6. Ejemplos

- Ahora con less

```
// Variables
@textColor: #232323;
@textHighlight: #00214D;
@fontFamily: Helvetica, Arial, sans-serif;
@fontSize: 14px;
@lineHeight: 21px;
@introSize: 16px;
@introLineHeight: 24px;
@introFontVariant: small-caps;
// LESS for Paragraph

 p { color: @textColor;
font-family: @fontFamily;
font-size: @fontSize;
line-height: @lineHeight; }
```

4.6. Ejemplos

- Ahora con less

```
// Variables //
…
// LESS for Paragraph // ------------------
p { color: @textColor;
font-family: @fontFamily;
font-size: @fontSize;
line-height: @lineHeight;

.intro { font-variant: @introFontVariant;
font-size: @introSize;
line-height: @introLineHeight; }
// End of .intro class
.highlight { color: @textHighlight;
font-weight: bold; }
// End of .highlight class   }
// End of paragraph rule
```

4.6. Ejemplos

A pesar de que el código con less es más largo. Enumera las ventajas de usarlo.

4.7. Funciones

- OPERACIONES
- Una explicación rápida de qué son podría ser “matemáticas”. Pero si nos metemos en una explicación un poco más larga, podemos decir que nos sirven para cambiar pixeles rápida y dinámicamente, porcentajes, ems, colores y mucho más sólo con una línea de código; de forma relativa a una línea inicial.

4.7. Funciones

- LESS puede identificar qué unidad ha sido asignada y después sumarle un número a dicha unidad. En este ejemplo, LESS debería añadir 4px (pixeles) al tamaño fuente de la etiqueta blockquote.

```
// Variables para el ejemplo //
@baseFontSize: 16px;
@baseFontFamily: Helvetica, sans-serif;
@quoteFontFamily: Georgia, serif;

// Damos estilos al párrafo y blockquote con
LESS //
p { font-family: @baseFontFamily;
font-size: @baseFontSize; }

blockquote { font-family: @quoteFontFamily;
font-size: @baseFontSize + 4; }
```

4.7. Funciones


```
// Variables para el ejemplo //
@baseFontSize: 16px;
@baseFontFamily: Helvetica,
sans-serif; @quoteFontFamily:
Georgia, serif;

// Damos estilos al párrafo y
blockquote con LESS //
p { font-family:
@baseFontFamily;
font-size: @baseFontSize; }

blockquote { font-family:
@quoteFontFamily; font-size:
@baseFontSize + 4; }
```

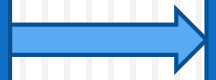
```
p { font-family:
Helvetica, sans-
serif; font-size:
16px; }

blockquote { font-
family: Georgia,
serif; font-size:
20px; }
```

4.7. Funciones

- OPERANDO CON EL COLOR

```
@color: #888 - #222;
h2 { color: @color; }
```



```
h2 { color: #666666; }
```

```
lighten(@color, 10%); // Devuelve un color 10% más claro que @color
darken(@color, 10%); // Devuelve un color 10% más oscuro que @color
saturate(@color, 10%); // Devuelve un color 10% más saturado que @color
desaturate(@color, 10%); // Devuelve un color 10% menos saturado que @color
fadein(@color, 10%); // Devuelve un color 10% menos transparente que @color
fadeout(@color, 10%); // Devuelve un color 10% más transparente que @color
fade(@color, 50%); // Devuelve un color con 50% de transparencia
spin(@color, 10); // Devuelve un color con 10 grados más en el tono de
@color
spin(@color, -10); // Devuelve un color con 10 grados menos en el tono de
@color
mix(@color1, @color2); // Devuelve una mezcla entre los colores @color1 y
@color2
```

4.7. Funciones

- FUNCIONES MATEMÁTICAS
- Por último, pero no lo menos importante…
- LESS permite utilizar funciones matemáticas para realizar cálculos (básicos).
- Entre ellas destacan: round(), ceil(), floor() y percentage().
- Si introducimos un número entre sus paréntesis nos creará un número redondeado, un número redondeado hacia arriba, un número redondeado hacia abajo y un número convertido a porcentaje, respectivamente.

4.8. Herencia

- Cuando escribes un hoja de estilos y quieres aplicar el mismo estilo a varios elementos los escribes de la siguiente forma:

```
p,
ul,
ol {
/* aqui van los estilos */
}
```

- Esto es una excelente solución hasta que tenemos que modificar una propiedad específica de un elemento lo cual empieza a ser un problema cuando nos damos cuenta de lo desordenada que tenemos nuestra hoja de estilos. Para ello, cada uno de los pre-procesadores tiene su forma de lidiar con esta situación.

4.8. Herencia

- Para el caso de **LESS** también puedes utilizar las herencias pero en el momento de compilar, en vez de crear aplicar los mismos estilos a múltiples clases repite cada uno de los elementos.

```
.bloque {
margin: 10px 5px;
padding: 2px;
}

p {
. bloque; /* Hereda estilos de
'.bloque' */
border: 1px solid #EEE;
}
ul, ol {
.bloque; /* Hereda estilos de
'.bloque' */
color: #333;
text-transform: uppercase;
}
```

```
.bloque {
margin: 10px 5px;
padding: 2px;
}
p {
margin: 10px 5px;
padding: 2px;
border: 1px solid #EEE;
}
ul,
ol {
margin: 10px 5px;
padding: 2px;
color: #333;
text-transform: uppercase;
}
```

Saas

Indica las principales diferencias entre less y saas con respecto a

Sintaxis
Variables
Código anidado
Mixins
Importación
Funciones de colores
Operaciones

5. Post CSS

- PostCSS es una herramienta que “transforma código CSS mediante plugins de JavaScript”, pero no es una herramienta más… **puede hacer “más cosas” que pre-procesadores tipo SASS, LESS, Stylus**… pero de forma **modular**, es decir es más rápido, y solo consume lo que uses.
- **¿Es mejor que SASS, LESS, Stylus?** Por si solo no hace nada…, pero usándolo con determinados plugins llega ha ser una herramienta más completa.

5.1. Ventajas de PostCSS

- Puedes usarlo junto con herramientas de **Automatización de tareas** (gulp, grunt…) por lo que ademas de crear CSS, podemos hacer al mismo tiempo otras cosas desde la misma consola, como unir archivos js, minificar código, comprimir imagenes, etc…
- Es **modular**, solo usa lo que instalemos, y por el orden que lo hagamos haciéndolo más potente y rápido.
- Hay una gran colección de **plugins** con los que puedes hacer cosas que no puedes con los pre-procesadores actuales.
- Puedes crear tus propios plugins fácimente en **javascript**, si así lo necesitas.

5.1. Ventajas de PostCSS

- Puedes usarlo con archivos css “normales”, pero también usando el plugin que toque con archivos de otros pre-procesadores (sass, less, stylus…).
- Es usado por Google, Shopify, Twitter, Bootstrap, Codepen… por lo tanto, algo bueno tiene que tener.
- Usado con el plugin CSSNEXT puedes adelantarte unos años y empezar a escribir el código css del futuro (con variables, y demás características), pero te lo compilará al CSS actual con los respectivos prefijos.

5.2. Plugins de PostCSS

- **Autoprefixer**
- Con este plugin puedes escribir código css3 y el solo te lo transforma luego con los diferentes prefijos (-webkit, -moz, etc…). Es decir no hace falta escribir código del estilo rollo @include …. Como en SASS para usar mixins de ese tipo solo para lo que es crear prefijos de propiedades de CSS. Ya solo por esto puede merecer la pena.

```
.cabecera {
  position:relative;
  margin:0;
  transition: all 1s;
}
```

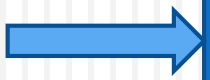
```
.cabecera {
  position:relative;
  margin:0;
  -webkit-transition: all 1s;
  -moz-transition: all 1s;
  -ms-transition: all 1s;
  transition: all 1s;
}
```

5.2. Plugins de PostCSS

- **CSS MQPacker**
- Con este plugin puedes escribir código de media queries por diferentes partes del archivo y al procesarlo te las imprime todas agrupadas al finalizar el archivo CSS. Ejemplo:

```
.cabecera {
  width: 100%;
  @media (min-width: 460px) {
    width: 50%;
  }
}

.pie {
  width: 100%;
  @media (min-width: 960px) {
    width: 50%;
  }
}
```



```
.cabecera {
  width: 100%;
}
 .pie{
  width: 100%;
}
 @media (min-width: 960px) {
  .cabecera {
    width: 50%;
  }
   .pie {
    width: 50%;
  }
}
```