UNIDAD2.
CSS. BUENAS PRÁCTICAS

Diseño de Interfaces WEB
CFGS DAW

Alfredo Oltra
alfredo.oltra@ceedcv.es
2021/2022
Versión:211027.0105

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

ÍNDICE DE CONTENIDO

UD02. BUENAS PRÁCTICAS.

1.¿QUÉ SON LAS BUENAS PRÁCTICAS?

La exigencia de las aplicaciones actuales hace que la generación de código sea cada vez una tarea más complicada: los proyectos son más grandes, más complejos técnicamente, implicar diferentes agentes (clientes, diseñadores, gestores de contenidos, programadores...) y suelen requerir de unos tiempos de desarrollo muy cortos con escalabilidades muy altas, es decir, con previsión de que la vida del producto sea larga. Eso convellará con bastante probabilidad que sean varios los desarrolladores que trabajen en el proyecto.

Por todo ello, nació el concepto de buenas prácticas. Una serie de recomendaciones que tiene por objetivo que el código que escribimos sea lo más escalable, reutilizable, limpio y sencillo de entender y de mantener.

> 📣 No hay que confundir las buenas prácticas con la notación utilizada. La notación es sólo una de los componentes de las buenas prácticas.

La naturaleza del lenguaje en el que codifiquemos va a marcar en cierta manera estas recomendaciones (no es lo mismo programar en programación estructurada, que en funcional, o orientado a objetos o redactar un documento HTML o una hoja de estilos), pero al menos tres conceptos pueden aplicarse a todas ellas:

- Ser consistente. Posiblemente la más importante. En interés de la compatibilidad, seguir un estándar suele ser una buena idea. Pero si se quiere seguir, siempre se puede eligir una manera personal de hacer las cosas, pero eso sí, mantenerla siempre.
- Ser simple (*KISS* = *Keep In Simple Stupid*). Hay que buscar la sencillez en todo momento, objetivo que en muchas ocasiones no resulta sencillo: no hay que comentar lo obvio, utilizar nombres descriptivos,...
- No repetir (*DRY* = *Don't Repeat Yourself*). Repetir siempre es confuso y al mismo tiempo un foco de errores.

2. ESTRUCTURA DE UN DOCUMENTO CSS

2.1 Número de ficheros

En cuanto nos enfrentamos a un fichero CSS muy largo, la primera duda que suele surgir es si vale la pena mantenerlo o dividirlo en varios ficheros más pequeños.

Hoy en día, debido sobre todo al uso de preprocesadores, la tendencia es trocearlo en ficheros más pequeños con funcionalidades concretas que suele enlazarse mediante *@import*. El problema es que esta regla puede generar problemas de eficiencia al no permitir en ocasiones la carga simultánea de los ficheros *.css.* Una posible solución suele ser trabajar en fase de desarrollo (donde la velocidad de trabajo no suele ser tan problemática) con *@import* y en entornos de producción combinarlo en un único fichero o realizar la llamada mediante *<link>*.

2.2 Longitud de línea

Como en casi todos los lenguajes, conviene no sobrepasar el límite de los 80 caracteres por línea, situación que suele ser habitual sobretodo en los comentarios. Con longitudes mayores puede ser más complicado leer el código y como ventajas añadidas nos permitirá tener varios documentos abiertos simultáneamente en pantalla además de facilitar la lectura en sistemas de gestión de código (como *BitBucket*) o incluso en el terminal.

2.3 Índice o tabla de contenidos (TOC)

Cuando nos enfrentamos a un CSS muy largo es muy interesante tener alguna forma rápida de poder localizar las reglas que nos interesan. La primera de ellas es la creación de un índice general y, si es el caso, de un índice en cada uno de los ficheros en que se haya dividido el CSS.

```
/*****************************************************************
 * CSS prueba
 * Creado por: elautor (autor@dominio.com)
 * Modificado por: el modificador (modificador@dominio.com)
 *
 * CONTENIDO
 *
 * CONFIGURACIONES
 * Reset
 *
 * ELEMENTOS BASE
 * Cabeceras
 * Párrafos
 *
 * CONTENEDORES
```

```
 * Columnas
 * Botones
 * Navegación
 *
 ************************************************************************/
```

Además es necesario que cada sección aparezca indicada con su correspondiente cabecera, precedida por algún tipo de símbolo que permita su rápida localización evitando confusiones: un %, &...

```
/*--------------------*
  $CONTENEDORES
 *--------------------/

/**
 Comentario general de la sección
**/

.regla {}
```

2.4 Las reglas

Si muchas de estás prácticas responden a experiencias personales y, por lo tanto, tienen un gran parte de subjetivas, la manera de definir reglas es posible la más afectada por esta situación. Unas buenas recomendaciones suele ser:

- poner un sólo espacio antes de la llave de apertura
- un punto y coma detrás de todas las declaraciones, incluyendo la última
- la llave de apertura al final de la línea del último selector
- la llave de cierre en una nueva línea
- identar con espacios (no tabuladores)
- cada declaración en una nueva línea
- cada regla separada de la siguiente por un línea
- si una regla afecta a varios selectores (separados por comas) separarlos por líneas en función de la relación entre ellos.
- si las propiedades están relacionas tipo, es conveniente alinear los valores

```
sel1,sel2,sel3
sel4 {
   propiedad-tipo1:       valor1;
   propiedad-tipo1-larga: valor2;
```

```
    propiedad1-tipo2:         valor3;
    propiedad-distinta-tipo2: valor4;
}
```

> 📣 Gran parte de estos consejos tienen que ver con el uso de otros programas. Por ejemplo, utilizar espacios en vez de tabuladores permite mantener un formato homogéneo con otros editores que pueden tener otra configuración en lo que respecta a los tabuladores. O repartir las propiedades en diferentes líneas nos reduce las posibilidades de conflictos en el sistema de control de versiones

Por lo que respecta al HTML en relación con el CSS, la recomendación más importante es encerrar siempre entre comillas las clases, aunque sólo se asigne una.

2.5 Nomenclatura

En general se usa una nomenclatura que separe las palabras que definen la clase con guiones (*kebab case*): *reloj-grande*.

Sin embargo, es recomendable que ese mismo nombre sea lo más autoexplicativo posible, lo más lleno de semántica, evitando el uso de comentarios y ayudando a entender mejor tanto el fichero CSS como el documento HTML. En estos casos los objetivos a la hora de nombrar las clases van más allá de un simple nombre; deben indicar qué hace la clase, dónde puede ser usada y con que se puede relacionar.

Cuando queremos trabajar así, la nomenclatura viene definida por la arquitectura que utilicemos a la hora de definir nuestra aplicación. o documento.

2.6 Comentarios

Bien es sabido que los comentarios son parte importante en cualquier código, pero hay que saber utilizarlos en su justa medida. Una falta de comentarios puede llevarnos a perder mucho tiempo en entender el funcionamiento de alguna parte (o peor aún, a malinterpretarlo). Sin embargo, abusar de ellos tampoco es bueno ya que confunden al programador haciéndole perderse entre mucho texto que al final no le aporta la información que necesita.

En CSS los comentarios básicamente se deben usar para explicar aquellas técnicas (o trucos) en los que hace falta profundizar un poco más allá que la simple asignación de propiedades.

3. SELECTORES

Los selectores son una parte fundamental en nuestras reglas. De hecho, una de las complicaciones mayores a la hora de cear un CSS que funcione de la manera más eficiente posible, es la declaración correcta de los mismo. Desafortunadamente (o afortunadamente, según como se vea), existen muchas opciones a la hora de definir un selector para una misma regla.

En general la primera regla a tener en cuenta va a ser la especificidad. Cuanto menos ambigua, más específica sea la regla, más seguridad tendremos en que seleccionamos lo que queremos seleccionar.

3.1 Reusabilidad

Cuando hablamos de reusabilidad hablamos de la posibilidad de poder volver a usar parte de los componentes creados en un proyecto en otro.

Para eso conviene centrarse en el uso de clases y olvidar el uso de los ID. Estos, además de ser demasiado específicos, no pueden utilizarse más de una vez en una página determinada, mientras que las clases pueden reutilizarse una cantidad infinita de veces.

Hay que tener en cuenta que todo lo que se define debería ser factible poder ser reutilizado, desde el tipo de selector hasta su nombre.

3.2 Independencia de la localización

Es importante, no diseñar las cosas en función de dónde se encuentren, sino de lo que son. Es decir, el estilo de nuestros componentes no debe depender de dónde los colocamos, sino que deben permanecer totalmente independientes de la ubicación.

Por ejemplo en vez de usar una regla como esta

```
.nav .ul {  list-style: none; }
```

usar

```
.list-without-bullets: {  list-style: none; }
```

3.3 Rendimiento del selector

El rendimiento del selector se mide en el tiempo en el que el navegador es capaz de encontrar el elemento que se le indica.

En general este tiempo es proporcional al número de componentes que tiene el selector: cuantos más componentes más lento.

```
h1 div ul  { } /* lento */

.cabecera-con-div-y-ul { } /* rápido */
```

Esto es debido a que los navegadores buscan desde el elemento clave (el más a la derecha) hacia la izquierda. En este caso busca todos los ul, después repasa todo el DOM para encontrar si cualquiera de ellos es descendiente de un div y así de manera consecutiva. Obviamente en el segundo caso la búsqueda es única.

 El hecho de utilizar el selector descendiente empeora la situación. Si es posible resulta más eficiente utilizar el selector de hijos, que limita la búsqueda únicamente al elemento superior.

 En general, hay que intentar huir del selector *. Este actúa como comodín, seleccionando todos y cada uno de los elementos del DOM y eso, como es obvio, no es muy eficiente. Aún así, su impacto en el proceso del CSS no es demasiado grande.

4. CSS RESET

Un *reset* en CSS es un conjunto de reglas cuyo objetivo es tener un punto de referencia a la hora de empezar a trabajar con un documento CSS.

Como ya se ha comentado en temas anteriores, cada navegador tiene sus propios estilos predefinidos, de manera que los estilos no declarados específicamente por el diseñador tengan una valor que asegure la legibilidad de la página por defecto. Esto provoca bastantes problemas a los diseñadores a la hora de hacer que sus páginas se vean de la misma manera en diferentes navegadores.

La solución al problema reside en la creación de una serie de reglas que ponen a valores por defecto (*resetean*) a todas las propiedades de todos los elementos que se consideren importantes. Reglas que serán aplicadas al principio de la lectura del fichero CSS.

Las ventajas que aporta está técnica son principalmente dos: partimos de un punto de referencia claro y bien definido y, una vez declaradas, son reutilizables, es decir, podemos copiarlas y pegarlas en cualquiera de nuestros ficheros.

> Aún así, no todo está de acuerdo con la utilización de estas reglas de reinicio. Muchos desarrolladores opinan que resetear y posteriormente definir es hacer un trabajo doble (que va en contra del principio DRY de buenas prácticas). Si no asignas una propiedad en un elemento será porque no te es necesario hacerlo.
>
> Además, aparecen otros problemas como que el peso de la página aumenta,

4.1 Posibles CSS Reset

En internet podemos encontrar muchas y muy diferentes soluciones para la creación del código de inicialización, unas más genéricas y otras más específicas.

4.1.1 Universal

Tal vez la primera aproximación a la creación de un CSS Reset es el llamado reset universal. Es muy sencillo y consiste únicamente en poner a cero pixeles el *margen* y el *padding*

```
* { margin: 0px; padding: 0px; }
```

aunque en alguna ocasión se amplia con el uso de `border: 0px`

Este selector es básico y para proyectos sencillos puede ser suficiente, pero tiene problemas debido al uso del selector universal. Por ejemplo:

- El selector se aplica a todos los elementos de la página, lo cual penaliza el renderizado (aunque no es ni mucho menos el proceso que más le puede afectar)
- Este selector pone a cero todos ambas propiedades de todos los elementos, dando por hecho que después vamos a inicializarlas. Pero si se nos olvida o utilizamos algún elemento nuevo muy posiblemente tendremos una renderización que no es la esperada y posiblemente peor que la que proporciona la de por defecto del navegador.

4.1.2 Eric Meyer's CSS

Muy posiblemente es el más utilizado hoy en día. Está creado por uno de los gurús de CSS. En este caso lo que se crean son reglas para cada uno de los elementos de manera independiente (o agrupándolos si es el caso) pero no únicamente declarando las propiedades a 0, sino asignando los valores que pueden ser más representativos.

El propio autor recomienda que no sea utilizado directamente, sino que se adapte a las circunstancias de la página en la que va a ser usado.

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
      margin: 0;
      padding: 0;
      border: 0;
      font-size: 100%;
      font: inherit;
      vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
      display: block;
}

body {
      line-height: 1;
}

ol, ul {
      list-style: none;
}

blockquote, q {
      quotes: none;
}

blockquote:before, blockquote:after,
q:before, q:after {
      content: '';
```

```
      content: none;
}

table {
      border-collapse: collapse;
      border-spacing: 0;
}
```

4.1.3 HTML5 Reset Stylesheet

Surge como un *fork* del código anterior y está más orientado hacia los nuevos elementos HTML5. Elimina soporte a elementos que han sido eliminados en esta nueva versión, además de incorporar estilos acordes a etiquetas como <*abr*> o <*dfn*> o eliminar los viñetas de las listas ordenadas de las barra de navegación <*nav*>.

```
/*
html5doctor.com Reset Stylesheet
v1.6.1
Last Updated: 2010-09-17
Author: Richard Clark - http://richclarkdesign.com
Twitter: @rich_clark
*/

html, body, div, span, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
abbr, address, cite, code,
del, dfn, em, img, ins, kbd, q, samp,
small, strong, sub, sup, var,
b, i,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, figcaption, figure,
footer, header, hgroup, menu, nav, section, summary,
time, mark, audio, video {
 margin:0;
 padding:0;
 border:0;
 outline:0;
 font-size:100%;
 vertical-align:baseline;
 background:transparent;
}

body {
 line-height:1;
}

article,aside,details,figcaption,figure,
footer,header,hgroup,menu,nav,section {
 display:block;
}

nav ul {
 list-style:none;
}
```

```
blockquote, q {
 quotes:none;
}

blockquote:before, blockquote:after,
q:before, q:after {
 content:'';
 content:none;
}

a {
 margin:0;
 padding:0;
 font-size:100%;
 vertical-align:baseline;
 background:transparent;
}

/* change colours to suit your needs */
ins {
 background-color:#ff9;
 color:#000;
 text-decoration:none;
}

/* change colours to suit your needs */
mark {
 background-color:#ff9;
 color:#000;
 font-style:italic;
 font-weight:bold;
}

del {
 text-decoration: line-through;
}

abbr[title], dfn[title] {
 border-bottom:1px dotted;
 cursor:help;
}

table {
 border-collapse:collapse;
 border-spacing:0;
}

/* change border colour to suit your needs */
hr {
 display:block;
 height:1px;
 border:0;
 border-top:1px solid #cccccc;
 margin:1em 0;
 padding:0;
}

input, select {
 vertical-align:middle;
}
```

4.1.4 Mini Reset

Es un reset muy minimalista que únicamente resetea los componentes más habituales que suelen usarse en las páginas web.

```
/* CSS Mini Reset */

html, body, div, form, fieldset, legend, label {
  margin: 0;
  padding: 0;
}

table {
  border-collapse: collapse;
  border-spacing: 0;
}

th, td {
  text-align: left;
  vertical-align: top;
}

h1, h2, h3, h4, h5, h6, th, td, caption { font-weight:normal; }

img { border: 0; }
```

Existen muchas otras opciones: *Reset* o *Marx* de enfoque más general u otras como *Typeset* o C*leanslate* más pensadas para blogs y para redistribución de contenidos embebidos.

4.2 Normalización

El los últimos años la tendencia empieza a ser no a resetear, sino a normalizar. La idea que reside bajo la normalización es aumentar al máximo la compatibilidad de un mismo CSS en diferentes navegadores buscando que la renderización sea lo más similar posible en todos ellos.

Para ello:

- Preserva los valores por defecto de los navegadores que se pueden considerar útiles (al contrario que los reset que descarta todos). En caso de que un elemento tenga diferentes estilos en diferentes navegadores los normaliza haciendo que sean lo más estándar posible.
- Corrige errores de funcionamiento de ciertas propiedades en algunos navegadores.
- Están diseñados de manera modular, de manera que es posible eliminar aquellos módulos que no nos interesen para ahorrar espacio.

Las dos referencias más claras son *normalize* y *sanitize*, basado en el primero.

5. ARQUITECTURA

5.1 OOCSS

Esta metodología se basa en seguir las pautas de la programación orientada a objetos con el objetivo de hacer nuestro código más pequeño y más escalable, es decir, de poder incorporar más código de una manera muy sencilla.

> La finalidad de la herencia es eliminar código, de manera que características que estén en varios elementos únicamente estas estén definidas en el padre.

El primer paso consiste en dejar de utilizar identificadores en los selectores. Siguiendo la filosofía de la programación orientada a objetos todo deben ser clases. Estas tienen la gran ventaja de que pueden ser reutilizadas y siempre puede sustituir a un identificador usándola únicamente una vez.

El siguiente paso es aplicar la herencia, pero no la herencia en el sentido de padres e hijos que aporta la estructura en árbol del documento HTML. Es una herencia en la que los padres e hijos son reglas.

Veamos un ejemplo. Supongamos que nuestra página contiene varias columnas, cuyas dimensiones son las mismas pero el color de fondo y los bordes cambian.

```
<div class="columna1"></div>
<div class="columna2"></div>

.columna1 {
  background: blue;
  border-radius: 15px 50px 30px 5px:
  display: inline-block;
  width: 200px;
  height: 500px;
  margin: 20px;
}

.columna2 {
  background: green;
  border-radius: 15px 50px:
  display: inline-block;
  width: 200px;
  height: 500px;
  margin: 20px;
}
```

Se puede apreciar que hay bastante código repetido, lo cual no es bueno por varias razones:

- Como decidamos cambiar la altura de la columna hay que cambiarlas en todas las columnas, lo cual es propenso a errores tanto de escritura como de que olvidemos actualizar alguna.
- El tamaño del fichero es más grande
- Por supuesto... no seguimos DRY.

Una mejor opción sería considerar una clase columna que defina las propiedades comunes, y otra clase "hija" que defina las particulares de cada una de ellas.

```
<div class="columna columna--azul"></div>
<div class="columna columna--verde"></div>

.columna {
  display: inline-block;
  width: 200px;
  height: 500px;
  margin: 20px;
}

.columna--azul {
  background: blue;
  border-radius: 15px 50px 30px 5px:
}
.columna--verde {
  background: green;
  border-radius: 15px 50px:
}
```

> 📣 Una forma de detectar rápidamente si estamos aplicando correctamente la OOCSS es comprobar si tenemos una misma propiedad con el mismo valor en elementos relacionados. Si eso es así es que se puede aplicar mejor la OOCSS

El problema que nos encontramos es que la relación entre elementos no se visualiza directamente. Las relaciones son relaciones ficticias que no se declaran en ninguna parte (no es como en un lenguaje de programación orientado a objetos al uso, donde se especifica con una palabra clave la herencia: `class` *`MiClase`*`: extends` *`MiClasePadre`*). Es por eso que necesitamos algún mecanismo basado en la nomenclatura que aporte esa semántica que necesitamos.

5.2 BEM

Cuando queremos definir una clase para elementos que tienen cierta complejidad semántica, es necesario implantar alguna codificación que, de manera rápida, nos indique los objetivos que hemos comentado cuando hablamos de nomenclatura..

Para ello una de las metodologías habituales es BEM. Esta metodología divide el nombre del elemento en tres partes:

- el bloque (*Block*), el raíz del problema. Generalmente hace referencia al contenedor en el se encuentra el elemento.
- el elemento (*Element*), una parte del bloque
- el modificador (*Modifier*), una particularización del bloque.

Los elementos se separan con 2 subrayados (__), mientras que los modificadores se indican con 2 guiones (–)[1].

La mejor manera de entenderlo es con un ejemplo:

```
/* regla para la barra de navegación */
.nav {}

/* regla para los menús de la barra de navegación */
.nav__menu {}

/* regla para la barra de navegación superior */
.nav--superior {}

/* regla para de contenidos */
.nav-contenidos {}

/* regla para los menús izquierdos de la barra de navegación */
.nav__menu--izquierdo {}
```

Evidentemente este tipo de nomenclatura requiere práctica y debe analizarse con cuidado. Por ejemplo:

```
/* regla para un bloque de información dentro del contenido. MAL */
.contenido—informacion {}

/* regla para un bloque de información dentro del contenido. OK */
.contenido .informacion {}
```

La diferencia es sutil, pero se puede explicar viendo que el bloque información es un bloque en sí mismo, que no tiene porque estar dentro del contenido. Este ejemplo nos hace ver que parte del trabajo consiste en identificar muy bien los bloques y sus elementos.

El objetivo final es no sólo hacer el código CSS más entendible, sino también el propio fichero HTML. Un código de este estilo

1 Aunque en algunas ocasiones se utiliza al contrario

```
<div class="box perfil usuario-pro">
   <img class="avatar imagen" />
   <p class="bio">...</p>
</div>
```

donde las clases no aparecen relacionadas entre sí, podría convertirse en algo como

```
<div class="box perfil perfil__usuario-pro">
   <img class="avatar perfil__imagen" />
   <p class="perfil__bio">...</p>
</div>
```

6. MINIFICAR

Uno de lo objetivos de usabilidad de nuestra web es que su tiempo de carga sea pequeño. Para conseguirlo hay dos caminos: aumentar la velocidad del canal o reducir el tamaño de los ficheros. El primero no está en nuestras manos, así que debemos centrarnos en el segundo.

Minificar permite eliminar los elementos existentes en el fichero CSS, que están incluidos para mejorar la legibilidad del código, pero que no son necesarios en cuanto al formato del mismo. Es decir que el documento sigue permaneciendo bien validado y el preprocesador CSS de nuestro navegador lo leerá sin mayor problema.

Aplicaciones para minificar nuestro CSS hay varias aplicaciones, como *YUI Compressor* o *cssmin.js*

> Está técnica se puede aplicar a todos los componentes de una aplicación web además del CSS. Para JavaScript disponemos por ejemplo de *Closure Compiler* o *YUI Compresor* mientras que en HTML podemos encontrar *PageSpeed Insights*.

7. OTROS MATERIALES

Mini reset

http://www.vcarrer.com/2010/05/css-mini-reset.html

Minificador HTML

https://developers.google.com/speed/pagespeed/insights/?hl=es-419

8. BIBLIOGRAFÍA

[1] HTML5 Style Guide and Coding Conventions

http://www.w3schools.com/html/html5_syntax.asp

[2] Grouping related classes in your markup

http://csswizardry.com/2014/05/grouping-related-classes-in-your-markup/

[3] High-level advice and guidelines for writing sane, manageable, scalable CSS

http://cssguidelin.es/

[4] CSS Reset

http://cssreset.com/