Ciclo Formativo de Desarrollo de Aplicaciones Web

Diseño de Interfaces Web

Vicente Javier lópez Belmonte

Maquetar usando estilos

En esta unidad trataremos el uso de estilos para la maquetación de nuestros interfaces web. La maquetación con CSS es fundamental a la hora de obtener unos resultados de calidad en el diseño de tu página web y te simplificará la vida, no sólo al crear la web, sino también a la hora de mantenerla.

Introducción

Antes de la llegada de CSS disponíamos únicamente del HTML, que tenía múltiples carencias a la hora de posicionar elementos en la página, porque cuando fue creado no se esperaba que la web se convirtiera en un multimedia, donde los profesionales aportarían caudales de creatividad y diseños caprichosos. HTML en principio únicamente permitía organizar el texto en párrafos, acompañado de enlaces, listas, imágenes, tablas y poco más.

Como sólo había HTML, los diseñadores utilizaron el único recurso que tenían a mano para posicionar elementos en la página: las tablas, que estaban pensadas para presentar información tabulada (en celdas formadas por filas y columnas), pero no para maquetar una web entera. Anidando tablas (colocando unas tablas dentro de otras) y con el recurso de imágenes de un píxel transparente, se podía obtener una estructura de diseño para luego llenarla con los contenidos que se desease.

Las tablas solucionaron por un buen tiempo las necesidades de los diseñadores de webs, pero tenían diversos problemas, aparte de no facilitar mucho la estructura de sitios con un diseño complejo.

- El contenido se mezcla con las reglas de presentación o formato. Lo que hace que el código de tu página web sea innecesariamente grande y ello deriva en páginas más pesadas. Al final, con tablas tenemos una web más lenta y la transferencia de datos de nuestro servidor también aumenta, con lo que tu servidor podría atender a menos usuarios al mismo tiempo y te saldrá más caro de mantener.
- El rediseño de una web se hace mucho más complicado, porque para cambiar la forma con la que se ve tu página tendrás que actualizar todo el código. Si maquetas utilizando CSS sólo tendrás que cambiar el código CSS para que el aspecto de tu página sea tan distinto como desees.
- Tu página tendrá problemas serios al verse en otros dispositivos, como tablets o teléfonos móviles, que tienen pantallas menores.
- Tendrás que remar contra corriente para intentar que tu página se vea como quieres, porque estás utilizando unas herramientas, las tablas, que no te ofrecen las posibilidades necesarias para maquetar a voluntad. Tendrás que aprender mil truquillos para saltarte las limitaciones de las tablas y a medida que los apliques, tu código se hará más y más pesado, menos entendible y su mantenimiento será cada día más complicado.

¿Por qué maquetar con CSS?

Básicamente porque con css podemos separar el contenido del aspecto o presentación. Si nos paramos a pensar, separar contenido de presentación no tendría por qué aportarnos más ventajas ya que tendríamos lo mismo y encima separado. Pero nada más lejos de la realidad:

- **Separación de forma y contenido.** Generalmente CSS y HTML se encuentran en archivos separados, lo que facilita el trabajo en equipo porque diseñador y programador pueden trabajar independientemente. Por otro lado, permite el acceso a distintos navegadores y dispositivos.
- **Tráfico en el servidor.** Las páginas pueden reducir su tamaño entre un 40% y un 60%, y los navegadores guardan la hoja de estilos en la caché, ésto reduce los costos de envío de información.
- **Tiempos de carga.** Por la gran reducción en el peso de las páginas, mejora la experiencia del usuario, que valora de un sitio el menor tiempo en la descarga.
- **Precisión.** La utilización de CSS permite un control mucho mayor sobre el diseño, especificando exactamente la ubicación y tamaño de los elementos en la página. También se pueden emplear medidas relativas o variables para que la pantalla o la caja contenedora se acomode a su contenido.
- **Mantenimiento.** Reduce notablemente el tiempo de mantenimiento cuando es necesario introducir un cambio porque se modifica un solo archivo, el de la presentación, sin tener que tocar las páginas que contienen la estructura con el contenido.
- **Diseño unificado y flexibilidad.** Es posible cambiar completa o parcialmente el aspecto de un sitio con sólo modificar la hoja de estilos. Por otro lado, el tener el estilo de una web en un solo archivo permite mantener la misma apariencia en todas las páginas.
- **Posicionamiento.** Las páginas diseñadas con CSS tienen un código más limpio porque no llevan diseño, sólo contenido. Esto es semánticamente más correcto y la página aparecerá mejor posicionada en los buscadores.

Un ejemplo que ilustra las posibilidades de la maquetación CSS es la web CssZenGarden. Existe un sitio muy interesante e ilustrador que nos puede ayudar a conocer rápidamente la potencia de las CSS y hacernos una idea de lo que puede significar su uso. Es un sitio donde se muestra un contenido y un diseño bastante logrado. Además, dispone de varios enlaces para poder ver el mismo sitio, con el mismo contenido, pero con distinto aspecto. De ese modo podemos ver cómo se puede llegar a alterar el diseño de una página con tan solo cambiar la hoja de estilos.

La URL del sitio es http://www.csszengarden.com . Es muy interesante que seleccionéis otros diseños para ver el cambio radical que puede tener las páginas web con distintas hojas de estilos.

Métodos por los que se pueden aplicar estilos a las páginas web mediante CSS.

Vamos a ver algunos métodos, quizás ya estés familiarizado con ellos en el módulo de Lenguaje de Marcas pero vamos a centrarnos en la manera en la que los podemos usar para maquetar una web.

Aplicación de estilo a etiquetas

Se puede asignar el estilo a una etiqueta concreta de HTML. Para ello, en la declaración de estilos escribimos la etiqueta y entre llaves, los atributos de estilo que deseemos.

```
body {
      background-color: #f0f0f0;
      color: #333366;
}
```

Podemos aplicar el mismo estilo en un conjunto de etiquetas. Para ello, indicamos las etiquetas seguidas por comas y luego, entre llaves, los atributos que queramos definir.

```
h1, p{
      color: red;
}
```

En este caso se define que los encabezados de nivel 1 y los párrafos, tengan letra roja.

Definición de clases

Podemos utilizar una clase si deseamos crear un estilo específico, para luego aplicarlo a distintos elementos de la página. Las clases en la declaración de estilos se declaran con un punto antes del nombre de la clase.

```
.miclase{
      color: blue;
}
```

Para asignar el estilo definido por una clase en un elemento HTML, simplemente se añade el atributo class a la etiqueta que queremos aplicar dicha clase. El atributo class se asigna al nombre de la clase a aplicar. Por ejemplo:

```
<p class="miclase">este párrafo tiene el estilo definido en la clase "miclase".</p>
```

El párrafo anterior se presentaría con color azul. La definición de clases y su utilización es sencilla, pero veamos un ejemplo más detallado:

Para la siguiente declaración de estilos:

```
body, td, p{
      background-color: #000000;
      color: #ffffff;
}

.inverso{
      background-color: #ffffff;
      color: #000000;
}
```

Se ha definido un fondo negro y color del texto blanco para el cuerpo de la página, así como las celdas y los párrafos. Luego se ha declarado una clase, de nombre "inverso", con los colores al revés, es decir, fondo blanco y texto negro.

```
<body>
<p>Hola esto es un parrafo normal</p>
<p class="inverso">Párrafo con los colores invertidos</p>
<table>
<tr>
      <td class="inverso">INVERSO</td>
      <td>NORMAL</td>
</tr>
</table>
</body>
```

Esta página tiene, generalmente, el fondo negro y el texto blanco. El primer párrafo, que es un párrafo normal, sigue esa definición general de estilos, pero el segundo párrafo, al que se ha aplicado la clase "inverso", tiene el fondo blanco y el texto en negro. Por lo que respecta a la tabla, en su primera celda se ha asignado la clase "inverso", por lo que se verá con fondo blanco y color de texto en negro. Mientras que la segunda celda, que no tiene asignada ninguna clase, se presentará como se definió en la regla general.

Estilos que sólo se utilizan una vez

También podemos tener un estilo específico para un único elemento, que no va a repetirse en ningún otro caso. Para ello tenemos los estilos asignados por identificador. Los identificadores se definen en HTML utilizando el atributo id en la etiqueta que deseamos identificar. El valor del atributo id será el que definamos nosotros.

```
<div id="capa1">
```

En la hoja de estilos, para definir el aspecto de ese elemento con id único, se escribe el carácter almohadilla, seguido del identificador indicado en la etiqueta y entre llaves los atributos css que deseemos.

```
#capa1{
      font-size: 12pt;
      font-family: arial;
}
```

En este caso se ha asignado fuente de tamaño 12 puntos y cuerpo arial.

Como se puede concluir en la lectura de estas líneas, generalmente se prefiere utilizar estilos definidos en clases a los definidos con identificadores, a no ser que estemos seguros que ese estilo no se va a repetir en todo el documento.

Flujo HTML y atributos CSS

El flujo HTML es el modo en el que se van colocando los componentes de la página, a partir de cómo aparecen en el código HTML y los atributos CSS de los elementos.

Merece la pena detenerse a explicar lo que es el flujo HTML, pues es un concepto sencillo y básico para poder entender muchos asuntos acerca del posicionamiento web y en concreto el posicionamiento con CSS.

El flujo de la página es algo así como el flujo de escritura de elementos dentro del lienzo que nos presenta el navegador. Sabemos que las páginas web son codificadas en HTML y los elementos

aparecen en el código en una posición dada. El navegador, en el momento que interpreta el código HTML de la página, va colocando en la página los elementos (definidos por medio de etiquetas HTML) según los va encontrando en el mismo código.

Por ejemplo, pensemos en una página que tiene un titular con H1, luego varios párrafos y alguna imagen. Pues si lo primero que aparece en el código HTML es el encabezamiento H1, pues ese encabezado aparecerá en la página también en primer lugar. Luego se colocarán los párrafos y si la imagen aparecía en el código por último, en la página también aparecerá al final. Es decir, los elementos aparecen colocados tal como estén ordenados en el código. A esto se le llama el flujo HTML, la colocación de los elementos en el lugar que corresponda según su aparición en el código.

Esto en general ocurre con cualquiera de los elementos de la página. Sin embargo, hay algunos atributos HTML que pueden marcar distintas propiedades en el flujo, como que una imagen se alinee a la derecha, con align="right", con el texto del párrafo que pueda haber a continuación rodeando la imagen. Pero con HTML, si por ejemplo, una imagen va antes que un párrafo, nunca vamos a poder intercambiar sus posiciones y colocar la imagen detrás del párrafo que le sigue en el código.

Esto no ocurre de igual manera cuando trabajamos con CSS, puesto que existen diversos atributos que pueden cambiar radicalmente la forma en la que se muestran en la página, por ejemplo el atributo position que puede definir valores como absolute, que rompe el flujo de la página, o mejor dicho, saca del flujo de la página al elemento que se le asigna.

Comportamientos inline y block y cómo afectan al flujo de la página

Cuando tratamos con etiquetas, existen dos modos principales de comportamiento. Etiquetas como una imagen, o una negrita, que funcionan en línea ("inline"), es decir, que se colocan en la línea donde se está escribiendo y donde los elementos siguientes, siempre que también sean "inline" se posicionan todo seguido en la misma línea. Tenemos por otra parte los elementos que funcionan como bloque ("block") que implican saltos de línea antes y después del elemento. Por ejemplo, los párrafos o encabezamientos son elementos con comportamiento predeterminado tipo "block".

Dos etiquetas muy utilizadas en la maquetación CSS son DIV y SPAN. Una de las diferencias principales es que DIV funciona con coportamiento "block" y SPAN funciona como "inline". En realidad este es el comportamiento por defecto, puesto que nosotros con CSS en cualquier momento podemos cambiarlo por medio del atributo display. Por ejemplo:

```
<div style="display: inline;">
Este elemento funcionará en línea
</div>
```

O bien:

```
<span style="display: block;">
```

```
Este span ahora funciona como bloque
</span>
```

Realmente ambas posibilidades funcionan dentro del flujo HTML normal, así que, tanto los elementos display inline como display block, se encuentran dentro del flujo HTML estándar, la única diferencia es que los bloques se escriben en líneas independientes, es decir, con saltos de línea antes y después del elemento, así como una cantidad de margen arriba y abajo que depende del tipo de elemento de que se trate.

Atributo CSS Float y el flujo

Otro atributo que afecta al fluir de los elementos en la página es el atributo float de CSS, que se utiliza bastante en la maquetación web. Este atributo podemos utilizarlo sobre elementos de la página de tipo "block" y lo que hace es convertirlos, en "flotantes" que es un comportamiento parecido a lo que sería el mencionado anteriormente "inline". Con float podemos indicar tanto left como right y conseguiremos que los elementos se posicionen a la izquierda o la derecha, con el contenido que se coloque a continuación rodeando al elemento flotante. La diferencia es que los elementos continúan siendo tipo "block" y aceptan atributos como el margen (atributo CSS margin), para indicar que haya un espacio en blanco a los lados y arriba y abajo del elemento.

Por ejemplo, los elementos de las listas (etiqueta LI) son por defecto de tipo "block", por eso aparecen siempre uno abajo de otro, en líneas consecutivas. Pero nosotros podríamos cambiar ese comportamiento con:

```
li{
   float: right;
}
```

Así, una lista como esta:

```
<ul>
<li>Elemento1</li>
<li>Elemento2</li>
<li>Elemento3</li>
</ul>
```

Veríamos como el primer elemento aparece a la derecha del todo y los otros elementos van colocándose en la misma línea en el siguiente espacio libre que haya. Así, el segundo elemento se colocaría en la misma línea, todo a la derecha que se puede, conforme al espacio que se tenga en el contenedor donde estén colocados.

Flujo y el atributo position

El atributo position de CSS sí que es capaz de cambiar radicalmente el flujo de los elementos de la página. Este atributo, por defecto tiene el valor "static", que indica que el elemento forma parte del flujo HTML normal de la página.

Sin embargo, con el atributo CSS position, podemos indicar otros valores que hacen que los elementos salgan del flujo HTML y se posicionen en lugares fijos, que no tienen que ver con la posición en la que aparezcan en el código HTML. Por ejemplo:

```
<div style="position: absolute; top: 10px; left: 100px;">
Este elemento tiene posicionamiento absoluto
</div>
```

Hace que ese elemento quede fuera del flujo de elementos en la página y entonces aparecería en el lugar que se indica con los atributos top y left (top indica la distancia desde la parte de arriba y left la distancia desde el borde izquierdo). Los otros elementos que formen parte del flujo de la página no quedan afectados por los elementos con posicionamiento absoluto.

Otro valor para el atributo position que hace que los elementos queden posicionados fuera del fluir normal de elementos en la página es "fixed".

Posicionamiento CSS

Las hojas de estilo en cascada incorporan múltiples formas para posicionar elementos en la página, lo que comúnmente se conoce como posicionamiento CSS.

Atributo position:

Este atributo es, digamos, el principal para definir el tipo de posicionamiento de un elemento. Sus valores posibles son static, absolute, fixed, relative, sticky, initial e inherit.

Atributos top, left, right, bottom:

Sirven para indicar la posición de un elemento, cuando éste tiene los valores de position "absolute", "relative" o "fixed" (en otros valores del atributo position estos atributos son ignorados). El atributo top indica la distancia desde el borde superior de la página y left desde el borde de la izquierda.

También se puede indicar opcionalmente la posición con bottom, que es la distancia desde abajo y right, que es la distancia desde la derecha.

Atributos float y clear:

Float sirve para establecer que un elemento tiene que "flotar", colocándose los valores "right" o "left" para que floten a izquierda o derecha. Por si sirve de aclaración, que los elementos floten es algo así como lo que pasa cuando definimos el atributo HTML align="right" o align="left" en las imágenes o tablas. Con el atributo clear hacemos que el elemento se coloque en el primer área libre que tenga al lugar donde se indique. Por ejemplo el valor de clear "right" hace que el elemento se coloque en el primer lugar donde no tenga ningún elemento flotando a la derecha. El valor de clear "both" hace que el elemento se coloque donde no tenga elementos flotando, tanto a la derecha como a la izquierda.

Atributo display:

Especifica el tipo de caja que debe que tener un elemento, que puede ser de diversas formas. Este atributo también tiene bastante utilización y entre los valores más corrientes podríamos destacar: "none", que hace que esa caja o elemento no aparezca en la página ni se reserve espacio para ella. "block", que sirve para que la caja sea un bloque y se muestre en una línea o líneas independientes de otros elementos de la página. "inline", que indica que esa caja tiene que mostrarse en la misma línea que otros elementos escritos antes o después.

Atributo visibility:

Atributo para definir la visibilidad de un elemento. Con este atributo podemos decir que ciertos elementos de la página sean visibles o invisibles, pero atención, aunque un elemento sea invisible, continúa ocupando espacio en la página. Si queremos que no sea invisible y no se le reserve espacio en la página, hay que utilizar el atributo display con el valor "none". Los valores más corrientes de visibility son: "visible", que hace que el elemento se vea (valor por defecto) y "hidden", que hace que el elemento sea invisible, aunque continúe ocupando espacio.

Atributo z-index:

Este atributo tiene como valor cualquier número entero. Sirve para indicar qué capa se tiene que ver por encima o por debajo de otra u otras, en caso que varias capas estén superpuestas. A mayores valores de z-index, la capa se coloca más al frente, tapando otras capas que tengan valores menores de z-index.

Soluciones a problemas típicos y cómo proceder

Usa un contenedor global para todas las cajas (cuando las cosas se disparan)

De esta forma estas prefijando globalmente el orden de todas las demás cajas. En referencia a este contenedor ordena el resto de las cosas interiores. Es como si haces una cerca o valla para que nada es escape. Obviamente estamos hablando de sitios no elásticos.

Ejemplo para un contenedor de 900px centrado:

```
#contenedor {
margin-top: 0px;
margin-right: auto;
margin-bottom: 0px;
margin-left: auto;
width: 900px;
}
```

Que flote a la izquierda (cuando las cajas se superponen)

Esta es una muy buena forma de evitar incompatibilidades entre navegadores. El uso de hacks de CSS se debía en gran parte porque se trabajaba "centrando" las cajas. Si por ejemplo precisas poner tres cajas de 300px en un contenedor de 900px puedes hacer lo siguiente.

Ejemplo:

```
#caja {
float: left;
width: 300px;
}
```

Calcular bien los paddings o rellenos (cuando las cajas se van abajo)

Casi todos los dolores de cabeza y maldiciones sobre el CSS se deben al mal uso o a la mala interpretación que se hace del padding. Pero es más simple de lo que parece.

¿Para que sirven los paddings o rellenos? Bueno, lo que hace es generar un "relleno" de determinada medida para dar por ejemplo como un margen a los elementos, pero lo hace sobre el ancho en píxeles que esté prefijado. Por ejemplo: si tenemos una caja de 300px y le aplicamos un padding de 10px en la izquierda, ahora tendremos una caja de 310px. Esto hará desbordar al resto de las cajas y las desplazarán para abajo. Ahí es cuando el diseñador principiante se vuelve loco. El tema es que si hay una diferencia de hasta un 1px se producirán estos desbordes, sino fíjate cuando le incluyes bordes a tu caja, se producirán difrencias.

Lo que se debe hacer es simple, calcular bien y recordar cada ajuste que se haga de los rellenos. Ahora tendremos que hacer una caja de 290px con paddings de 10px a la izquierda.

Ejemplo:

```
#caja {
float: left;
width: 290px;
padding-left: 10px;
background-color: #FFE6DD;
}
```

El pié de página con ancho fijo (cuando el pié de página enloquece)

Para entender mejor cómo funciona el uso de cajas en CSS se puede pensar en un grupo de objetos de diferentes formas que luchan por adaptarse y ocupar el espacio que se ha prefijado. Sucede muchas veces que los pié de página son los más problemas traen cuando se maqueta un sitio. O se va para arriba, se alinea a la izquierda, o se desborda, etc. Muchos resolvíamos este tema prefijando valores fijos a las alturas de cajas, pero no tiene sentido. Lo que se debe hacer es de nuevo establecer un valor de ancho fijo. De esta forma el pié se va a hacer su lugar del resto e irá a parar donde tiene que ir.

Ejemplo:

```
#pie {
width: 900px;
background-color: #666666;
}
```

No todo es 1+1=2 en CSS (cuando los anchos no cierran)

Un problema común en css es pensar que todos los anchos entre cajas cierran perfectamente. A veces es necesario jugar con los valores de los contenedores, a veces contrario a la lógica hay que añadir algunos px a los contenedores.

Otros trucos rápidos

Trucos sencillos, de los que no hace falta explicar mucho pero que son muy prácticos y te harán más fácil el trabajo y evitarán posibles errores.

- Usa colores diferentes para distinguir las cajas.
- Pon una palabra descriptiva en cada caja.
- Comenta el código fuente y señala los finales de los contenedores grandes.
- No ahorres espacios entre los divs.
- No seas un fundamentalista y no quieras escribir tu CSS con dos o tres líneas. Si no quieres errores escribe lo necesario.
- Cuidado con el tamaño de las imágenes que insertas, estas cambian el ancho de los contenedores.
- Elige bien los nombres de cada div y trata de ser ordenado en el código.
- Si vas a trabajar con varias cajas, trata de agruparlas, esto es muy importante. Por ejemplo un contenedor que agrupe tres o cuatro cajas.

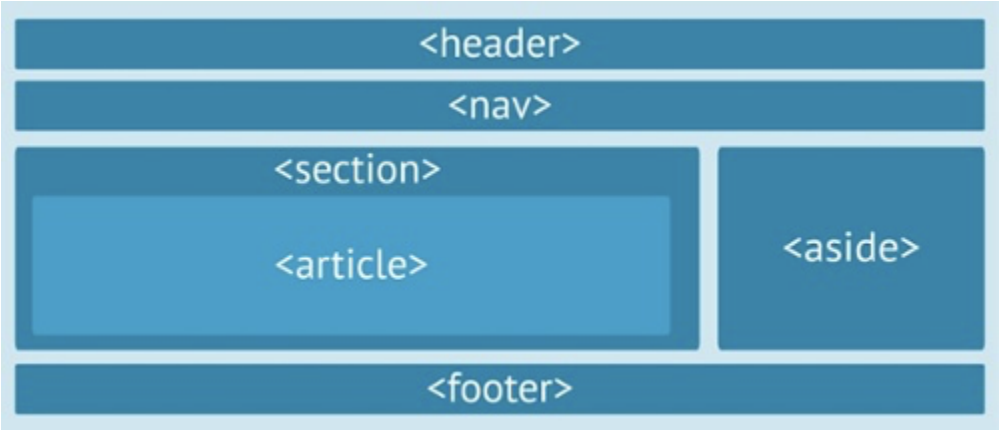
¿Que pasa cuando no puedes resolver un problema con CSS o similar? A mi me ha dado resultado levantarme un rato, hacer cualquier otra cosa y luego volver e intentar de nuevo.

Elementos semánticos

Las etiquetas semánticas ayudan a definir la estructura del documento y permiten que las páginas web sean mejor indexadas por los buscadores.

Una etiqueta se califica como semántica si tiene que ver el significado, es decir, si nos informa sobre lo que trata su contenido. Por ejemplo, la etiqueta SECTION nos dice que contiene una sección o capítulo dentro de la página.

El esquema de una página con etiquetas semánticas podría ser el siguiente.



Antes de la existencia de las etiquetas semánticas, el contenido se estructuraba con etiquetas DIV que no aportaban ninguna información sobre el tema que trataban, salvo que se añadiese aprovechando el valor dado a la propiedad ID, o a la propiedad CLASS para hacer referencia a su contenido.

Las etiquetas a las que vamos a prestar más atención en este artículo son las que definen la estructura general del sitio (HEADER, ARTICLE, SECTION, ASIDE, FOOTER, NAV) aunque también existen otra nuevas etiquetas semánticas que señalan elementos concretos de la página como la etiqueta FIGURE, para imágenes y gráficos, la etiqueta FIGCAPTION, que es un pie de imagen, la etiqueta TIME para definir fechas y horas, también MAIN, MARK, SUMARY Y DETAILS.

Vamos a ver con un poco más de detalle las características de cada etiqueta.

header

Se utiliza como una introducción del elemento que la contiene. Si colocamos un HEADER en el BODY, a primer nivel, será una introducción o presentación de toda la página. Si colocamos un HEADER dentro de un SECTION, contendrá una introducción a esa sección. También se pueden colocar dentro de un ARTICLE.

El HEADER suele llevar el título, la descripción corta y el logo de la página. El HEADER por si mismo no separa contenidos, no se debe utilizar aislado, sino dentro de otra etiqueta semántica, o a primer nivel en el BODY.

El HEADER no es equivalente a la parte superior de la página, aunque frecuentemente está en esa posición. La etiqueta HGROUP que se solía utilizar junto con el HEADER ha quedado obsoleta.

Es bastante frecuente utilizar las etiquetas H1, H2, ... para escribir los títulos dentro del HEADER.

section

Una SECTION agrupa contenido con un tema común, por ejemplo, las diferentes partes de un periódico: política, sociedad, deportes, ...

Un sitio de cocina podría estar organizado en secciones. Una SECTION podría ser recetas de carne, otra recetas de pescado y otra de verduras. Dentro de cada SECTION tendríamos un ARTICLE por cada receta.

Cada SECTION puede estar encabezada por una etiqueta H1, H2,... dentro de una SECTION suelen haber etiquetas ARTICLE, y también etiquetas DIV y P.

Una SECTION no debe ser usada con el único fin de dar formato, para eso se utiliza la etiqueta DIV.

article

Representa una unidad de contenido, es decir, contenido que responde a un tema concreto. El ejemplo más claro es un artículo dentro de una revista, por ejemplo, en una página de cocina, un ARTICLE sería cómo elaborar un determinado plato. En un foro, un ARTICLE sería un post, o entrada en el foro.

Se pueden anidar un ARTICLE dentro de otro ARTICLE. Dentro de un ARTICLE el contenido se suele estructurar con las etiquetas DIV y P. También está permitido poner dentro de un ARTICLE una SECTION, siempre que el artículo sea largo y tenga diferentes secciones, pero lo más natural es que el ARTICLE vaya dentro de una SECTION. Más adelante trataremos cómo crear estructuras de páginas complejas, con tres niveles.

nav

Esta etiqueta está pensada para contener la barra de navegación o los enlaces a las páginas del sitio web.

Normalmente va en la parte superior de la página o en un lateral. Suele estar formada por una lista de enlaces con las etiquetas UL y LI. Aunque hay multitud de diseños diferentes de barras de navegación y menús.

aside

Esta etiqueta está diseñada para contenido tangencial, es decir, menos importante que el cuerpo de la página, y que suele ser de un tema relacionado indirectamente (o no relacionado) con el tema principal. Se suele colocar en los laterales de la página. Pero no todo lo que va en los laterales es ASIDE. Por ejemplo, si la página tiene publicidad en el lateral derecho, puede colocarse en una etiqueta ASIDE, también una pequeña reseña sobre el autor podría ir en esta etiqueta.

footer

Normalmente va al final de la página y contiene información del tipo: autor, copyright, contacto, mapa del sitio, etc. Se pueden poner HEADER y FOOTER dentro de SECTION y ARTICLE, aunque no es lo más normal. No hay nada prohibido, pero debe tener cierta lógica o sentido semántico.

Estructura básica de un documento HTML5

Todos tenemos una estructura basica al realizar un sitio web y en HTML5 no hay excepción, lo que es muy facil de hacer ya que HTML5 tiende a ser una estructura muy semantica y es lo mejor de todo, sus etiquetas son muy básicas por eso vamos a ver como es una Estructura basica de un documento de HTML5. Este lenguaje tiende a hacerse mas simple y humano al escribir el código.

Empecemos con el DOCTYPE, es muy sencillo de escribir y ha sido recortado, después la etiqueta HTML la cual tiene el atributo LANG para el idioma del sitio.

```
<!DOCTYPE html>
<html lang="es">
<head>
```

Muy sencillo para empezar ¿no?, ahora toca el turno del titulo, este sigue siendo el mismo, con diferencia de la etiqueta META, donde le decimos al navegador que tipo de codificado es el documento html, de preferencia usemos el UTF-8 que acepta nuestros acentos y ñ con mas compatibilidad.

En realidad es algo demasiado corto y que no quita mucho tiempo, todo esto sigue estando en la etiqueta HEAD que no ha cambiado para nada, es algo muy parecido a XHTML, la cuestión es resumir el código.

```
<title>Titulo de la web</title>
<meta charset="utf-8" />
```

Pasemos a las etiquetas link, son las que enlazan nuestros estilos CSS, los Favicones y los Feeds. Estas son muy cortas con excepción de el feed que sigue casi igual, solo que ahora usaremos algo que ya existia, el atributo REL que usamos en los links para cuestiones de SEO.

REL es para decir, que estamos enlazando, ya que el atributo TYPE no se utiliza a menos que usemos para enlazar el FEED.

```
<link rel="stylesheet" href="estilos.css" />
<link rel="shortcut icon" href="/favicon.ico" />
<link rel="alternate" title="Pozolería RSS"
type="application/rss+xml" href="/feed.rss" />
```

Ya terminamos con la parte no visible de la pagina web, es el turno del cuerpo o BODY, esta etiqueta sigue siendo útil para lo mismo, pero a continuación empieza algo interesante que son las etiquetas de la estructura de el sitio visual.

La etiqueta HEADER es la cabecera donde va el nombre del sitio, el logotipo y descripción de la pagina web, ejemplo.

Recomiendo que solo se use un H1 en una cabecera por cuestiones de SEO, pero es posible poner mas de un H1 y HEADER.

```
<body>
<header>
     <h1>Mi sitio web</h1>
     <p>Mi sitio web creado en html5</p>
</header>
```

Como pueden ver, la etiqueta HEADER es muy equivalente a crear un DIV con ID=”header”, pero ahora tenemos una etiqueta especial para ello.

Pasamos con el contenido o SECTION usando H2 de titulo y el contenido en etiquetas P dentro de una etiqueta llamada ARTICLE para articulos o post, esto por cuestiones de SEO que es muy recomendable tambien, ejemplo.

```
<section>
    <article>
        <h2>Titilo de contenido<h2>
         <p> Contenido (ademas de imagenes, citas, videos etc.)
</p>
    </article>
</section>
```

SECTION es el equivalente a un DIV con ID=”contenido” y ARTICLE a un DIV con ID=”post” o “articulo”

Otra etiqueta interesante es la etiqueta ASIDE, lo que viene siendo nuestro SIDEBAR o barra lateral y es muy facil de implementar con H3 de titulo y P de contenido dentro de ella.

```
<aside>
```

```
            <h3>Titulo de contenido</h3>
            <p>contenido</p>
    </aside>
```

ASIDE es nuestro equivalente a un DIV con ID=”sidebar” o “barra”

Y finalizando con el pie de la pagina con la etiqueta muy obvia FOOTER , un ejemplo.

```
    <footer> </span>
    <p  style="padding-left:  30px;"><span  style="color:  #333399;">
    Creado por mi el 2011</span></p>
    <span style="color: #333399;"></footer>
```

FOOTER es nuestro equivalente a un DIV con ID=”footer” o “pie”

Ahora veamos nuestro código completo:

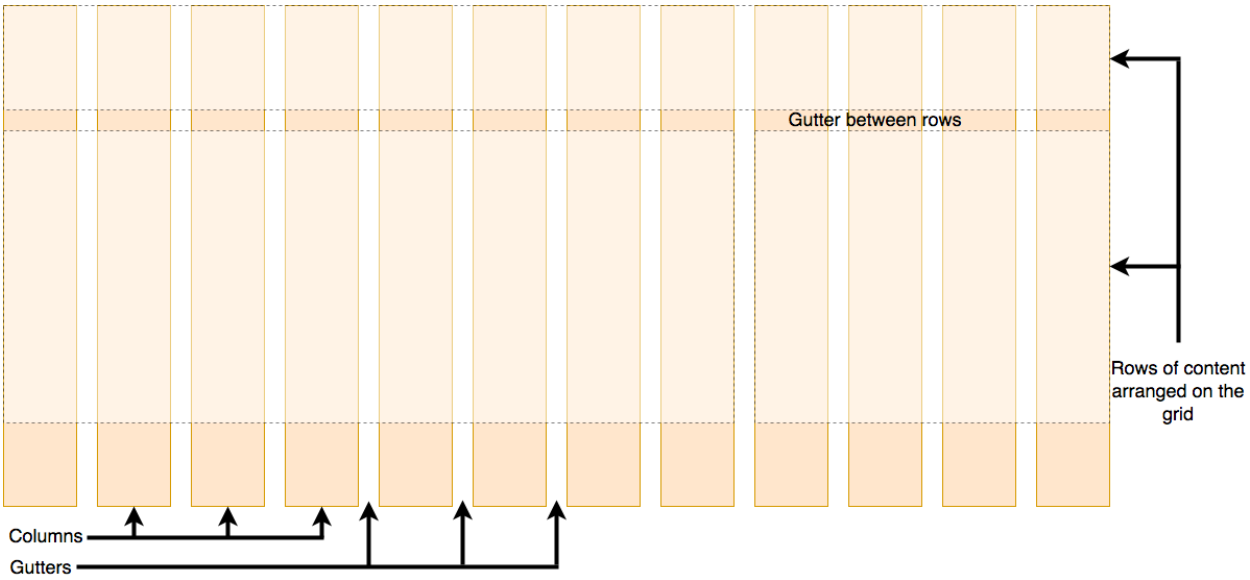
```
    <!DOCTYPE html>
    <html lang="es">
    <head>
    <title>Titulo de la web</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="estilos.css" />
    <link rel="shortcut icon" href="/favicon.ico" />
    <link        rel="alternate"        title="Pozolería        RSS"
    type="application/rss+xml" href="/feed.rss" />
    </head>
    <body>
        <header>
           <h1>Mi sitio web</h1>
           <p>Mi sitio web creado en html5</p>
        </header>
        <section>
           <article>
               <h2>Titilo de contenido<h2>
                   <p> Contenido (ademas de imagenes, citas, videos
    etc.) </p>
           </article>
        </section>
        <aside>
           <h3>Titulo de contenido</h3>
               <p>contenido</p>
        </aside>
        <footer>
            Creado por mi el 2011
        </footer>
    </body>
    </html>
```

Diseño basado en rejillas

Las rejillas (o grids) son una herramienta establecida de diseño, y los layouts de muchos sitios web modernos están basados en ellas. Vamos a echar un vistazo al diseño basado en grids y cómo podemos usar CSS para crearlos, así como las herramientas y las nuevas tecnologías que ya están a nuestra disposición en los navegadores.

Un grid es, simplemente, una colección de líneas horizontales y verticales que crean un patrón en el que podemos alinear nuestros elementos. Nos ayudan a crear diseños en donde los elementos no cambian de tamaño cuando nos movemos de una página a otra, proporcionando una mayor consistencia a nuestros sitios web.

Un grid, generalmente, consta de filas, columnas y huecos entre filas y columnas, también llamadas "gutters" (canaletas??)



Para garantizar una experiencia consistente en un sitio web o aplicación basada en un sistema de grid no necesitas pensar en el ancho que ocupa un elemento respecto a los demás. Sólo te limitarás a decidir “cuántas columnas del grid ocupa dicho elemento”.

Lo único que tendrás que hacer, en el proceso de diseño, qué grid utilizar.

Actualmente la mayor parte de los grids se crean usando bloques flotantes.

El framework de grid más fácil de crear es el de ancho fijo - sólo necesitamos decidir el ancho total que queremos dar a nuestro diseño, elegir cuántas columnas queremos y el tamaño de los espacios entre columnas (carriles). Si por el contrario decidimos que nuestro diseño es un grid con columnas que crecen y decrecen según el tamaño del navegador, necesitamos definir el ancho de las columnas y carriles por medio de porcentajes.

Para el diseño de ancho fijo empieza añadiendo en el body de un fichero html el siguiente código.

```
<div class="wrapper">
  <div class="row">
    <div class="col">1</div>
    <div class="col">2</div>
    <div class="col">3</div>
    <div class="col">4</div>
    <div class="col">5</div>
    <div class="col">6</div>
    <div class="col">7</div>
    <div class="col">8</div>
    <div class="col">9</div>
    <div class="col">10</div>
    <div class="col">11</div>
    <div class="col">12</div>
  </div>
  <div class="row">
    <div class="col span1">13</div>
    <div class="col span6">14</div>
    <div class="col span3">15</div>
    <div class="col span2">16</div>
  </div>
</div>
```

El objtetivo es visualizar una demostración de nuestro grid mediante dos filas de doce columnas — la fila superior muestra el tamaño de cada columna, y la segunda áreas de diferente tamaño en el grid.

En el elemento <style> o en una hoja de estilo a parte, debes añadir el siguiente código, que primero hace que el tamaño de todos los elementos de la página incluyen el tamaño de su borde y le da un tamaño de 980 píxeles al contenedor "wrapper", con un padding a la derecha de 20 píxeles. Esto nos deja un ancho de 960 píxeles por columna.

```
* { box-sizing: border-box; }

.wrapper {
  width: 980px;
  margin: 0 auto;
  padding-right: 20px;
}
```

Ahora usaremos un contenedor "row" para envolver cada fila del grid separándola del resto. Para ello añade la siguiente regla a la anterior:

```
.row {
  clear: both;
}
```

De esta forma conseguimos no tener que rellenar completamente una fila con las doce columnas. Cada fila queda separada del resto y no interfiere a las demás.

Los carriles entre columnas son de 20 píxeles de ancho. Los crearemos como un margen a la izquierda de cada columna — incluyendo la primera, para corregir los 20 píxeles de padding a la derecha que le dimos al contenedor "wrapper". Así conseguimos 12 carriles — 12 x 20 = 240.

Tenemos que restar los 240 pixeles al ancho total de 960 píxeles, con lo que nos quedan 720 para usar con nuestras columnas. Como tenemos 12, cada columna medirá 60 píxeles de ancho.

El siguiente paso es crear la clase .col, flotando a la izquierda, con un margin-left de 20 píxeles para crear el carril y un width de 60px. Añade la siguiente regla al final de tu CSS:

```
.col {
  float: left;
  margin-left: 20px;
  width: 60px;
  background: rgb(255,150,150);
}
```

La fila superior con las columnas individuales se mostrará claramente como un grid.

Nota: Hemos dado también un color de fondo rojo a cada columna para que se pueda ver perfectamente cuánto espacio ocupa cada una.

Los contenedores que deben ocupan más de una columna necesitan reglas especiales para ajustar su width al número de columnas requeridas (más los correspondientes carriles). Necesitamos crear una clase adicional para permitir tamaños de 2 hasta 12 columnas. El tamaño de cada una de elllas es el resultado de sumar el ancho del número de columnas que ocupan más los carriles entre ellas (los carriles serán siempre un número menos que el número de columnas).

Añade el siguiente código al final de tu CSS:

```
/* Dos columnas de ancho (120px) y un carril (20px) */
.col.span2 { width: 140px; }
/* Tres columnas de ancho (180px) y dos carriles (40px) */
.col.span3 { width: 220px; }
/* Y asi sucesivamente ... */
.col.span4 { width: 300px; }
.col.span5 { width: 380px; }
.col.span6 { width: 460px; }
.col.span7 { width: 540px; }
.col.span8 { width: 620px; }
.col.span9 { width: 700px; }
.col.span10 { width: 780px; }
.col.span11 { width: 860px; }
.col.span12 { width: 940px; }
```

Con estas clases ahora podemos conseguir columnas con distinto ancho en nuestro grid. Guarda el código y carga la página en tu navegador para verlo en acción.

Prueba a modificar las clases de los elementos del ejemplo, o añade o elimina algunos contenedores para ver cómo va variando su disposición. Por ejemplo, puedes cambiar el aspecto de una fila poniendo una columna de la clase span8 y otra de la clase span4.

Creando un Grid Fluido

Nuestro grid funciona perfectamente, pero tiene un ancho fijo. Ahora queremos un grid flexible (fluido) cuyo tamaño crezca y disminuya según el espacio disponible en el navegador. Para conseguirlo podemos usar como referencia los píxeles de ancho actuales y convertirlos en porcentajes.

La ecuación que convierte el ancho fijo en uno flexible basado en porcentajes es la siguiente.

objetivo / contexto = resultado

Para nuestro grid fijo, nuetro "objetivo" es 60 píxeles (el ancho de una columna) y nuestro "contexto" es el ancho del contenedor principal, es decir 960 píxeles. Sustituyendo los valores en la ecuación podemos calcular el porcentaje.

60 / 960 = 0.0625

Para conseguir el porcentaje, multiplicamos por 100 (para mover el decimal dos posiciones) y obtenemos el valor 6.25%. En nuestro CSS, reemplazamos los 60 píxeles del ancho de la columna por 6.25%.

Hacemos lo mismo con el ancho de los carriles:

20 / 960 = 0.02083333333

Tenemos que reemplazar los 20 píxeles del margin-left de la clase .col, y los 20 píxeles de padding-right del .wrapper, por 2.08333333%.

Usando como punto de partida el código anterior. Tenemos que actualizar:

```
.wrapper {
  width: 90%;
  max-width: 980px;
  margin: 0 auto;
  padding-right: 2.08333333%;
}
```

```
.col {
  float: left;
  margin-left: 2.08333333%;
  width: 6.25%;
  background: rgb(255,150,150);
}
```

```
/* Dos columnas de ancho (12.5%) y un carril (2.08333333%) */
.col.span2 { width: 14.58333333%; }
/* Tres columnas de ancho (18.75%) y dos carriles (4.1666666) */
.col.span3 { width: 22.91666666%; }
/* Y así sucesivamente... */
.col.span4 { width: 31.24999999%; }
.col.span5 { width: 39.58333332%; }
.col.span6 { width: 47.91666665%; }
.col.span7 { width: 56.24999998%; }
.col.span8 { width: 64.58333331%; }
.col.span9 { width: 72.91666664%; }
.col.span10 { width: 81.24999997%; }
.col.span11 { width: 89.5833333%; }
.col.span12 { width: 97.91666663%; }
```

Cuando usamos un sistema como éste, tienes que tener cuidado de que los anchos de las columnas y carriles sean los correctos, y que no incluyes elementos en una fila que ocupen más columnas de las que una fila puede contener. Debido a la manera en la que trabajan los elementos flotantes, si el ancho total llega a ser superior, el elemento que exceda el tamaño se verá en la siguiente línea.

Si el contenido de los elementos los obliga a hacerse más anchos, esto también hará que el ancho total sea más del 100% y ocurrirá lo mismo.

El contenedor de clase row ayuda a prevenir que nuestro diseño caiga en un caos y se mezclen los diferentes bloques con la reproducción de la web en diferentes resoluciones de pantalla.

Solución al problema de los decimales

Para solucionar los problemas anteriormente mencionados, se puede recurrir al uso de cálculos y variables en CSS.

Para ello debemos de incluir al principio del documento css la declaración de las variables que necesitemos utilizar:

```
:root {
  --base:1600;
  --tbase:calc(var(--base) * 1px);
  --tcolumna: calc(var(--tbase) / 12);
}
```

En el código superior tenemos por un lado definido el tamaño de base a 1600, luego la variable --tbase nos permite utilizar este tamaño de base usando el pixel como unidad. y el tamaño de columna será la doceava parte del tamaño de base, por lo que ahora para pasar de un grid de un tamaño a otro solamente tendremos que modificar el valor de la variable --base.

¿Pero sigue siendo fijo, no nos soluciona el problema? Solución

```
:root {
  --base:1600;
  --tbase:calc(var(--base) * 1px);
  --tcolumna: calc(var(--base) / 12);
  --pcolumna: calc(var(--tcolumna)/var(--base) * 100%);
}
```

Como ya podemos calcular usando las variables anteriores creamos la variable --pcolumna y aplicamos la misma fórmula pero ahora solo lo haremos una vez para tener, en porcentaje, el tamaño de una columna. A partir de aquí, vamos a modificar el código para usar estas variables.

```
.wrapper{
  width: 100%;
  max-width: var(--tbase);
  margin: 0 auto;
}
```

En el caso de la definición de las clases de columna, lo que podemos introducir es una col-1 para las columnas de tamaño 1, y hacer el resto sin que hereden de la clase col que teníamos, con lo que quedaría la siguiente distribución

```
.col-1 {
    float:left;
    width: var(--pcolumna);
}
.col-2{
  float:left;
  width: calc( var(--pcolumna) * 2);}
.col-3{
  float:left;
  width: calc( var(--pcolumna) * 3);
}
.col-4{
  float:left;
  width: calc( var(--pcolumna) * 4);
}
.col-5{
  float:left;
  width: calc( var(--pcolumna) * 5);
}
.col-6{
  float:left;
  width: calc( var(--pcolumna) * 6);
}
.col-7{
  float:left;
  width: calc( var(--pcolumna) * 7);
}
.col-8{
  float:left;
  width: calc( var(--pcolumna) * 8);
}
.col-9{
  float:left;
  width: calc( var(--pcolumna) * 9);
}

.col-10{
  float:left;
  width: calc( var(--pcolumna) * 10);
}
.col-11{
  float:left;
  width: calc( var(--pcolumna) * 11);
}
.col-12{
  float:left;
  width: calc( var(--pcolumna) * 12);
}
```

Como se puede observar, la propiedad float:left, que heredaba de la clase col principal hemos tenido que incluirla en todas las clases, pero esto nos facilitará en nuestra transición hacia un entorno de maquetación CSS en el que podamos adaptarnos a los distintos tipos de pantalla o grid Responsive.

Menus

Un Menú o barra de navegación es importante para cualquier sitio web. Con css se puede transformar menús HTML en barras con aspecto atractivo.

Una barra de navegación necesita un HTML estándar como base. Una barra de navegación es básicamente una lista de enlaces de la siguiente manera:

```
<ul>
  <li><a href="default.asp">Home</a></li>
  <li><a href="news.asp">News</a></li>
  <li><a href="contact.asp">Contact</a></li>
  <li><a href="about.asp">About</a></li>
</ul>
```

Tendremos que quitarle el tipo de la lista, los márgenes y el relleno:

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

Con este css quitamos los puntos de la lista y quitamos los márgenes y rellenos por defecto del navegador.

Esta lista la vamos a usar de base para crear nuestros menús tanto horizontales como verticales.

Menú vertical

Para el menú vertical aplicamos el siguiente css:

```
li a {
  display: block;
  width: 60px;
}
```

Con display:block hacemos que toda la zona que ocupe el elemento del menú se convierta en zona de enlace, y no sólo el texto. Los elementos ocupan por defecto todo el ancho disponible por lo que tendremos que especificar el ancho. Con esto ya podemos aplicar márgenes y rellenos a la zona para hacerla más grande.

El ancho también lo podemos definir en la lista, por tanto el siguiente código tendría el mismo efecto:

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 60px;
}

li a {
  display: block;
}
```

A partir de aquí podemos ir jugando con las propiedades de color, eliminar los subrayados a los enlaces e ir cambiando el aspecto del menú. Además podemos utiizar li a:hover para poder cambiar el aspecto de los elementos cuando pasamos el ratón por encima.

Por ejemplo de la siguiente manera:

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 200px;
  background-color: #f1f1f1;
}

li a {
  display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #555;
  color: white;
}
```

Por último podemos añadir la propiedad de un enlace activo. Como el menú se va a repetir en todas las páginas de nuestro sitio podemos crear una clase .active que nos permita identificar en nuestro menú la página que estamos visitando. Bastaría con aplicar esa clase en el elemento correspondiente de cada código HTML.

```
.active {
  background-color: #4CAF50;
```

```
  color: white;
}
```

Menú horizontal

Para convertir un menu vertical en horizontal bastaría con aplicar a nuestro código base la siguiente propiedad:

```
li {
  display: inline;
}
```

Ten en cuenta que la aplicamos al elemento de la lista y no al enlace (que seguirá teniendo el display block).

Otra forma de crear una barra de navegación horizontal es hacer flotar los elementos:

```
li {
  float: left;
}

a {
  display: block;
  padding: 8px;
    background-color: #dddddd; /* para añadir color gris al
elemento */
}
```

Añadimos el color de fondo a la lista general en lugar de cada elemento a si queremos que el color se aplique horizontalmente a toda la barra de navegación.

```
ul {
  background-color: #dddddd;
}
```

Un ejemplo completo de un menu horizontal sería el siguiente.

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
```

```
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #111;
}
.active {
  background-color: #4CAF50;
}
```

También es común poner los elementos del menú horizontal a la izquierda de la barra de navegación y un elemento a la derecha, de la siguiente manera:

Esto lo podemos conseguir añadiendo un identificador para el último elemento de la lista y añadirle la propiedad float:right.

Otro ejemplo que nos puede servir de ayuda sería utilizar un menu pegajoso o Sticky Navbar, podemos usar position:sticky; en las propiedades de la lista de tal manera que teniendo la barra de navegación debajo de la cabecera, cuando hagamos scroll y la cabecera desaparezca, el menú quede en la parte superior de la página y siempre esté visible.

```
ul {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
}
```

Grid Responsive

En los proximos contenidos, vamos a maquetar usando entornos de maquetación que se basan el diseño de rejilla, adaptándose a los diferentes medios en los que se reproducen. O lo que se conoce como diseño responsive, o adaptativo. El entorno que utilizaremos será BOOTSTRAP 4, pero para comprenderlo mejor, vamos a realizar un paso previo a éste, se trata de crearnos nuestro propio entorno de maquetación que se comporte de manera similar a éstos.

Partiendo del grid fluido visto en el desarrollo del tema podremos conseguir las clases suficientes para esto, lo que tendremos que decir al navegador es qué clase aplicar o utilizar en cada momento, para ello utilizamos las media queryes de CSS.

```
@media screen and (min-width:767px) {
```

Con esto le vamos a indicar al navegador qué partes del código queremos que se apliquen a partir de un determinado tamaño, en el ejemplo podemos ver que lo contenido a partir del corchete de esa media query se va a aplicar cuando el tamaño de la pantalla sea mayor de 767 px, es decir, aplicaremos las col-1, col-2, col-3… que teníamos anteriormente mientras la pantalla sea menor que 767 px y a partir de ahi podremos cambiar el comportamiento de los elementos aplicando otras clases, como, por ejemplo las siguientes:

```
@media screen and (min-width:767px) {
.col-sm-1 {
    float:left;
    width: var(--pcolumna);
}
.col-sm-2{
  float:left;
  width: calc( var(--pcolumna) * 2);}
.col-sm-3{
  float:left;
  width: calc( var(--pcolumna) * 3);
}
.col-sm-4{
  float:left;
  width: calc( var(--pcolumna) * 4);
}
```

Como podemos observar, se trata de la misma clase pero con un interfijo (sm) correspondiente a aquellos elementos que sean mayores de 767 px.

Si hacemos lo mismo para 971px y el interfijo md, y 1280px y el interfijo l, nos quedarían 4 tamaños a los que nuestra web se podría adaptar: Sin interfijo para móvil, con sm, para tablets, con md para escritorios pequeños o tablets en horizontal y con l para escritorio.

¿Pero esto cómo lo aplico al html?

Pues simplemente tenemos que añadir las clases a las columnas según queramos que se comporten, esto es, si una columna siempre va a tener el tamaño máximo de ancho, para todos los dispositivos, pues usaremos col-12 porque al no aplicar otras clases que se apliquen en los distintos tamaños siempre va a mantener el tamaño.

```
<header class="row">
    <div class="col-12">
        <h1>Mi sitio web</h1>
        <p>Mi sitio web creado en html5</p>
    </div>
</header>
```

Sin embargo, si queremos que una columna cambie su tamaño adaptándose a diferentes medios deberemos de aplicar las clases correspondientes a cada tamaño (indicado con el interfijo)

```
<article class="col-12 col-sm-6 col-md-4 col-l-3 ">
    <img class="casa" src="imagen/casa1.jpg">
    <h2>Titulo de contenido<h2>
    <p> Contenido (ademas de imagenes, citas, videos etc.) </p>
</article>
```

En el ejemplo superior, el article se comporta teniendo ocupando toda la pantalla para móvil, luego ocupa la mitad de la pantalla para tablets, para el siguiente tamaño ocupa un tercio de la pantalla y por último para tamaños de escritorio ocupará un cuarto de la pantalla.

Además, con la posibilidad de elegir comportamientos podríamos añadir márgenes o quitarlos, bastaría con añadir los tamaños de columna que queramos dejar de margen y añadir esas clases al media query correspondiente.

```
.ml-1{
 margin-left: calc( var(--pcolumna) * 1);
}
```

```
.ml-2{
 margin-left: calc( var(--pcolumna) * 2);
}
.ml-3{
 margin-left: calc( var(--pcolumna) * 3);
}
```

Y otra utilidad que podríamos añadir sería la que hubiese elementos que solamente se mostrasen para algunos tamaños. Por ejemplo creándonos las clases ver y nover:

```
.ver{
 display: block;
}
.nover{
 display: none;
}
```

Y añadiéndolas también a cada Media query.

```
 .ver-l{
     display: block;
   }
   .nover-l{
     display: none;
   }
```

Por ejemplo, con el siguiente código html podremos mostrar el article correspondiente a cada tamaño. Prueba a hacerlo y verás como cambia para cada uno de los tamaños que nos hemos definido.

```
<section class="row">
           <article class="col-12 nover-sm">Movil</article>
           <article class="col-12 nover ver-sm nover-md">Tablet</article>
           <article class="col-12 nover ver-md nover-l">Escritorio pequeño</article>
           <article class="col-12 nover ver-l">Escritorio</article>
</section>
```

Fíjate bien en el orden a la hora de declarar cada uno, como el móvil tiene el tamaño general, solamente tenemos que ocultarlo para el siguiente tamaño. El resto de elementos por defecto, en el tamaño general sin interfijo, estarán ocultos y se mostrarán u ocultarán en cada cambio de tamaño. Exactamente igual que al cambiar los márgenes o tamaños de columna entre los diferentes tipos de pantalla.

Bibliografía

https://www.w3schools.com/