Resumen maquetación CSS

Maquetar no es más que definir los cuadraditos, donde cada cuadradito indicaría una parte de mi página. Lo primero será decidir que estructura queremos que tenga nuestra página web.

Un **Layout**, no es más que la maquetación o estructura de mi página web.

En este resumen se tratan las técnicas de maquetación tradicionales que se siguen utilizando en millones de páginas web. Hay otras técnicas de maquetación web, como por ejemplo Flex y Grid (son más avanzadas, se verán más adelante)

Recomendaciones para maquetar correctamente

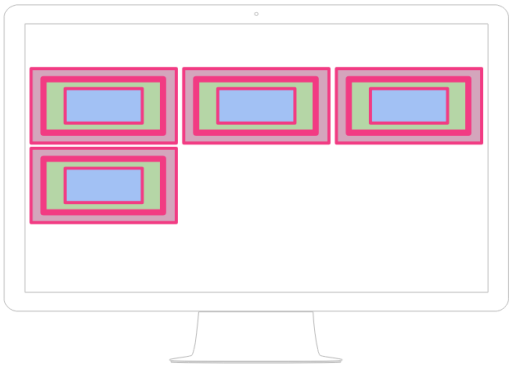
Antes de comenzar propiamente a maquetar veamos unas recomendaciones para que todo el proceso de crear el diseño de vuestra propia sea mucho más fluido.

- Realiza siempre un esquema previo, a mano o bien con alguna herramienta específica (por ejemplo, **balsamiq.com**, **ninjamock**, ...)
- Diseña siempre "de lo grande a lo chico". Por ejemplo, primero se coloca el elemento que contiene la imagen, y después la imagen que va dentro. Siempre primero el cuadrado grande, y después los pequeños.
- Actualmente todos los navegadores incluyen por defecto herramientas para desarrolladores que nos van a facilitar mucho la vida

- Copia.
  - Hay millones de webs. Si ves alguna que te guste usa las herramientas de los navegadores e investiga cómo lo han hecho. Utiliza el conocimiento de otras personas en tú favor. Ver tanto la estructura como los estilos.
  - Pero ojo, no te limites a copiar y pegar. Así no aprenderás. Entiende lo que están haciendo y usa la documentación técnica si hay partes que no ves claras.
- Prueba en distintos navegadores.
- Paciencia.

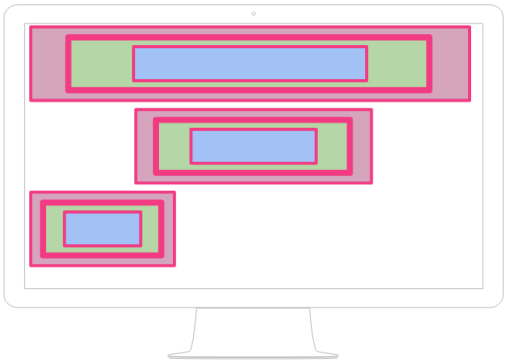
El flujo de los elementos en la ventana del navegador

- Debemos recordar que todas las etiquetas que se van a representar son *cajas*.
- Los navegadores no hacen **NADA** para controlar el diseño de nuestra página Web.
- Los navegadores, lo único que hacen es mostrar los elementos de nuestra página HTML en el mismo **ORDEN** en el que los hemos escrito.
- Siguen solo dos reglas básicas dependiendo de las propiedades de las cajas:



Elementos de línea

Determinados tipos de caja se van poniendo unas detrás de otros mientras quepan en la pantalla. Cuando no caben las cajas pasan a la "siguiente línea" del navegador.



Elementos de bloque

Otros tipos de caja provocan un "salto de línea".

Propiedad Display

Pese a que todos los elementos de mi página HTML son cajas lo cierto es que no todas las cajas se comportan igual cuando las añadimos. El comportamiento viene determinado por la propiedad CSS *display*, es la forma que tenemos de decirle al navegador como quiero que fluya mi caja.

Cada etiqueta tiene un valor por defecto para esta propiedad pero, para conseguir lo que el diseño que queremos, podremos modificarlas si lo estimamos necesario. Los valores que puede tomar son muchos pero los más usados son:

- *inline*
- *inline-block*
- *block*
- *none*
- Valores relacionados con tablas (veremos una lista detallada más abajo).
- Valores *flex* y *grid* que veremos en próximos cursos.

Elementos inline e inline-block

- Los elementos ***inline*** no rompen el flujo de la línea y se van colocando uno detrás de otro mientras caben. Aceptan *margin* y *padding* pero solo se tienen en cuenta los valores horizontales (no arriba y abajo). Ignoran *width* y *height*. Ejemplos: ,<img>,<a>, <b>, etc…
    - Aclaración, en el caso de <img> si hace caso al width y heigth

- Los elementos ***inline-block*** funcionan exactamente como los anteriores pero podremos asignarles *width* y *height*.

Elementos en bloque

Los elementos en bloque rompen el flujo de la línea y provocan *"un salto de línea"* tanto anterior como posterior. Por defecto, si no lo especificamos, ocuparán toda la anchura de la etiqueta que los contiene, la etiqueta contenedora.
Ejemplos: <h1>,<p>,<section>, ,<li>,<nav>, etc, …

Elementos con valor none en la propiedad display

Son elementos que desaparecen de la página. No dejan un espacio vacío aunque siguen en el código HTML. La propiedad visibility:hidden sí que se deja ese hueco aunque no se muestre.

Elementos con valores relativos a tablas en la propiedad display

Al poner uno de estos valores en la propiedad display a una etiqueta HTML esta etiqueta simulará el comportamiento del elemento de tabla análogo, es decir, la etiqueta a la que se le aplique se comportará como una tabla. De esta manera tenemos los siguientes posibles valores.

- *table*
- *table-row*
- *table-cell*
- *table-caption*
- *table-column*
- *table-colgroup*
- *table-header-group*
- *table-footer-group*
- *table-row-group*

Tipos de Layout

Dependiendo de cómo llevemos a cabo nuestra maquetación podemos tener distintos tipos de layouts:

Fixed Layouts

- **Establecen un tamaño fijo en pixels** para la anchura de los distintos elementos.
- Ventaja: tengo control total, los distintos elementos van a tener siempre el tamaño que yo quiero.
- Problemas derivados de usar este layout:
  - En pantallas pequeñas puede aparecer un scroll horizontal y esto es un gran error en diseño web.
  - En pantallas muy anchas puede que tenga mucho espacio en blanco a los lados del contenedor principal.

Elastic Layouts

- **Establece la anchura de los elementos en *em*** que es el tamaño de letra por defecto (suele ser 16px).
- Ventaja: al escalar el texto haciendo *zoom in* o *zoom out* los elementos cuyas dimensiones se hayan establecido en *em* escalarán correctamente.
- Desventaja : si escalo elementos que ocupen el mismo espacio éstos pueden solaparse y la única forma de solucionar este problema sería comprobar el tamaño de las fuentes en todo tipo de dispositivos y todo tipo de tamaños.
- También tiene el mismo problema que los fixed, ya que si reduzco mucho el tamaño de la pantalla, me saldrá la barra de desplazamiento horizontal.

Fluid Layouts

- **Establecen el ancho de los distintos contenedores en %** (con respecto al contenedor/etiqueta padre).
- Ventaja: los elementos mantienen sus proporciones independientemente del tamaño de la pantalla.
- Sin embargo, debemos afrontar unas importantes desventajas:
  - En pantallas pequeñas las columnas puede ser muy estrechas.
  - En columnas estrechas los textos largos provocan celdas muy alargadas.
  - Si tengo imágenes o vídeos con un tamaño fijo tendré problemas :(

Layout con max/min width (Híbridos)

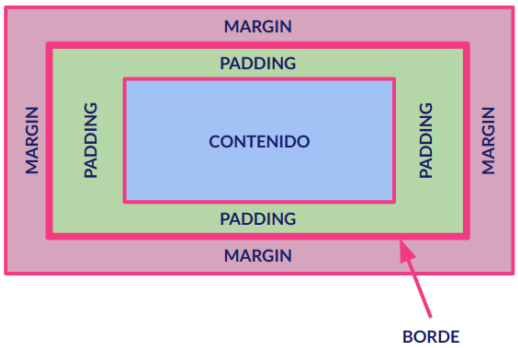
- Una posible técnica para solucionar los problemas que se nos presentaban en los anteriores tipos es la combinación de los principios de uno u otro junto con la asignación a los contenedores de los siguientes atributos:
  - **max-width** que hace que, en el caso de crecer, mi contenedor no supere esa anchura.
  - **min-width** que hace que, en caso de encoger, mi contenedor no sea menor que esa anchura.

Layout Responsivos

- Cambian conforme cambian las características de la pantalla en la que se van a mostrar (sobre todo conforme cambia la anchura). Este cambio es fluido.
- Un concepto similar y que con frecuencia se confunde es el layout adaptivo pero en estos casos no tengo un solo layout si no que tengo varios dependiendo de las características y el cambio de uno a otro se produce de manera brusca.

Con respecto a qué Layout utilizar, depende de lo que queramos hacer, no hay una respuesta única, todo va a depender del tipo de diseño que yo quiera y del tipo de proyecto. Quizá el que mejor escala es Fluid con max-width y min-width, hay diseñadores que prefieren el de em, aunque este con los distintos tipos de pantalla está dejando de utilizarse.

Box-sizing



Ya hemos comentado que todos los elementos de HTML al ser representados en los navegadores son *cajas* y que cada caja tiene contenido, relleno, borde y margen.

También recordamos que la maquetación web consiste en disponer estas cajas para que cada una ocupe el lugar que queremos al ser mostradas en nuestro navegador.

Para conseguir esto nos encontramos que los elementos al ser representados en el navegador ocupan el siguiente espacio:

- **La altura del elemento: altura del contenido + el padding + el borde.**

  (de los dos padding, y los dos border, ojo!)
- **La anchura del elemento: anchura del contenido + el padding + el borde.**

En esas circunstancias, y con un diseño complejo, cuando queramos cuadrar todo perfectamente vamos a tener que echar cuentas de todo, sumar todo para todas las cajas, añadir los márgenes etc…

En layouts sencillos no hay problema pero la cosa se complica si mi diseño es complejo.

La mejor solución para eso es establecer la propiedad CSS ***box-sizing: border-box*** y de esta manera no tendremos que echar cuentas con los bordes y los *paddings*. El tamaño que demos al elemento será la suma de todo.

Si queremos que siempre sea así añadiremos las siguientes líneas a mi CSS:

```
html {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

*,
*:before,
*:after {
  -webkit-box-sizing: inherit;
  -moz-box-sizing: inherit;
  box-sizing: inherit;
}
```

Es algo que **DEBEMOS** hacer siempre ya que nos facilitará mucho la vida a la hora de maquetar.

Otras posibilidades

La propiedad ***box-sizing*** puede tener otros dos valores:

- ***content-box*** que es el funcionamiento por defecto antes comentado.
- ***padding-box*** que no tienen en cuenta el borde pero si el padding y el contenido, la mayoría de los navegadores todavía no lo soportan.

Centrado del Layout

En muchas ocasiones cuando estamos maquetando queremos centrar algo para que todo encaje. Centramos toda la página, un texto de un título, una imagen en unas sección, etc…

Al principio, esta actividad de centrar es algo que se atraganta a los que están aprendiendo HTML y CSS.

Centrado en horizontal

- Si queremos centrar elementos en línea añadiremos la propiedad ***text-align:center*** al contenedor padre.
- Si queremos centrar un elemento en bloque dentro de su etiqueta contenedora añadiremos la propiedad CSS ***margin: X auto*** al elemento que queremos centrar (X es una distancia expresada en cualquier unidad). El elemento debe tener anchura.
- Si tenemos varios elementos en bloque dentro de un contenedor deberemos
  - Añadir la propiedad ***text-align:center*** al contenedor padre.
  - Añadir la propiedad ***display:inline-block*** a los elementos a centrar.
  - Deben tener una anchura concreta
- Usaremos contenedores **flex** aunque este tipo de contenedores no es objeto de este curso.

Centrado en vertical

- **Centrado vertical para elementos en línea.** Se puede conseguir:
  - Con el mismo ***padding*** arriba y abajo.
  - Añadiendo ***vertical-align:middle*** si estamos dentro de una celda de una tabla o lo estamos simulando con la propiedad display.
  - Con contenedores **flex** (en cursos posteriores).

- **Centrado vertical para elementos en bloque.** Se puede conseguir:
  - Utilizando la propiedad ***position*** en el contenedor y en el elemento (lo veremos en el próximo apartado).
  - Con contenedores **flex** (en cursos posteriores).

Propiedad Position. Posicionando elementos

La propiedad ***position*** cuyos valores van íntimamente asociados a las propiedades CSS ***top***, ***bottom***, ***left***, ***right*** y ***z-index***. Las 4 primeras de estas propiedades indican desplazamientos conforme a un punto de referencia en concreto y la propiedad ***z-index*** nos va a permitir trabajar con capas:

Los distintos valores que puede tomar esta propiedad son:

- ***static:*** es el valor por defecto. El elemento sigue el flujo que lo que corresponde. Aunque use top, bottom, left,right o z-index al elemento no se aplica ningún desplazamiento.
- ***relative:*** Es como *static* pero si atiende a los desplazamientos expresados en top,bottom,left,right o z-index.
- ***fixed:*** Se le aplica top,bottom,right o z-index en relación con documento. No atiende al scroll una vez generada por lo que su posición siempre permanece fija.
- ***absolute:*** Se comporta como fixed pero en relación con la primera etiqueta padre que tenga *position:relative*.
- ***sticky:*** Se comporta como *relative* hasta llegar a una posición de scroll y a partir de entonces *fixed*.

Centrado vertical de elementos de bloque

Una vez ya sabemos usar la propiedad position podemos centrar verticalmente elementos de bloque. Nos encontraremos con dos casos:

- Cuando conocemos la altura del elemento.
- Cuando desconocemos la altura del elemento.

Si conocemos la altura del elemento y esta es por ejemplo 150px;

```
css {
.contenedor {
  position: relative;  /* al padre */
}

.elemento_a_centrar {
  height: 150px;
  margin-top: -75px; /** La mitad de la altura **/
  position: absolute;  /* a los hijos */
  top: 50%;
```

```
}
```

Si desconocemos la altura del elemento:

```
.contenedor {
  position: relative;
}
.elemento_a_centrar {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
}
```

Columnas

Usando CSS podemos dividir el contenido que queremos mostrar en varias columnas tal y como podemos ver en un periódico.

Para ello utilizaremos las siguientes propiedades CSS:

- ***column-count:*** para especificar el número de columnas que tendrá el elemento contenedor.
- ***column-width:*** si queremos fijar en ancho de las columnas del contenedor. El navegador calculará cuántas columnas al menos caben con ese ancho.
- ***column-gap:*** permite establecer la distancia que separa las distintas columnas.
- ***column-rule:*** funciona de manera muy similar a borde y me permite especificar el estilo, color y anchura de la línea que separa las columnas.
- ***column-span:*** con valores *all* o *none* para indicar si el elemento en cuestión, que estará dentro del contenedor donde hemos especificado que habrá columnas, sigue el flujo en columnas o no.
- ***column-fill:*** para establecer cómo se rellenan las columnas. El contenedor debe tener altura. Los valore son *auto* o *balance* (todas las columnas la misma altura)
- ***break-inside:*** con valor *void* si queremos que el elemento no quede roto de una columna a otra.