CSS3

-

Maquetación

Contenido

1. Introducción

1.1 ¿Qué es la maquetación web?

La maquetación web es la parte que se encarga de definir la disposición de los distintos elementos (imágenes, texto, etc…) que forman nuestra página web para definir la estructura visual de la misma de acuerdo a unos objetivos de comunicación definidos.

Esto se conseguirá mediante el uso de CSS y etiquetas HTML, sobre todo las etiquetas: *div* y las etiquetas semánticas de HTML5: *header, footer, nav, section, article, aside, dialog, main, summary, …* Éstas últimas se comportan semánticamente igual que *div*, pero ayudan al navegador y a los buscadores a comprender mejor que tipo de contenido pueden tener dentro (accesibilidad).

Antes se usaban tablas HTML para maquetar. Esto no se hace hoy en día porque supone una serie de desventajas:

- No hay separación de apariencia y contenido.
- Mucho esfuerzo si hay cambios de diseño.
- Problemas para hacer páginas responsivas.
- Son menos accesibles.
- Se cargan de manera más lenta.
- Son menos legibles.
- Son peores para SEO.

Antes de empezar a maquetar, debemos definir cuál va a ser la estructura de mi página web.

2. Maquetación tradicional CSS

2.1 Recomendaciones para maquetar correctamente

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

- Realiza siempre un esquema previo.
- Diseña siempre “de lo grande a lo pequeño”.
- Utiliza las herramientas disponibles.
- Copia (reutiliza código de cosas que estén ya hechas).
- Prueba en distintos navegadores.
- Paciencia.

Esquemas previos.

En esta, como en otras muchas actividades, es bueno pararse a pensar antes de lanzarse a desarrollar. Si inviertes un poco de tiempo en pensar tu diseño de antemano al final seguro que ahorras tiempo.

Para esto, lo mejor es realizar bocetos de nuestro diseño. Una forma de hacerlo es mediante **wireframes** y/o mediante **mockups**. Un mockup es un modelo o un prototipo que se utiliza para exhibir o probar un diseño. La diferencia con respecto a los wireframes es que los mockups suelen ser desarrollados para conocer la opinión de usuarios o consumidores e incluyen más detalle que los wireframes.

Mockup vs **Wireframe**

Comparison Chart

Mockup	Wireframe
A mockup is a mid-fidelity representation of the design that gives a basic sense of what the website or app will look when finished.	A wireframe is a low-fidelity, skeletal representation of the design focusing on general layout, content hierarchy, and functionality.
A mockup is a static rendering of the actual design that includes fonts, text, colors, images, logos and other visual elements.	A wireframe is a tree diagram or flowchart of a website without colors, texture, imagery or animations.
It is a means of showing how your design will look like when it's put out in the real world.	It is used to define and plan the information hierarchy on a page or screen.
Common tools for creating mockups inlcude Balasmiq, Mockplus, Mockflow, Mockingbird, etc.	The common wireframing tools include Visio, Balasmiq, Axure, UXPin, Whimsical, etc.

DB Difference Between.net

Estos bocetos podremos hacerlos principalmente de dos maneras:

- ***A mano*** que es la forma más fácil y rápida.
- Con ***herramientas*** que, aunque no son tan rápidas me van a permitir operaciones como importar, compartir etc… Algunas herramientas para realizar estos bocetos son:
    - **mockflow.com**
    - **moqups.com**
    - **wireframe.cc** → más orientado a wireframes.
    - **balsamiq.com**
    - **ninjamock** → muy usado.

Usa las herramientas disponibles del navegador.

Actualmente todos los navegadores incluyen por defecto herramientas para desarrolladores que nos van a facilitar mucho la vida. Úsalas y si crees que no las necesitas atrévete a participar en **Codeinthedark**. Prueba tu destreza en el **editor**.

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.

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

Todos los navegadores tienen sus propias hojas de estilos por defecto que pueden hacer que la misma página no se vea igual en todos los navegadores.

Conforme vayas acabando tus diseños deberás comprobar que todo se ve bien en los distintos navegadores. Haz esto sobre todo al principio. Posteriormente tus diseños serán más robustos y no será tan necesario.

Paciencia

La maquetación Web no es difícil, pero requiere la comprensión de ciertas reglas básicas y su perfecto encaje posterior.

En tus inicios muchas veces desesperarás y pensarás que es el navegador el que funciona mal (el navegador no suele equivocarse). No es así, si el navegador no muestra lo que tú quieres es porque está mostrando lo que tú le has dicho. Si eso sucede:

- Revisa los conceptos básicos.
- Revisa la sintaxis de tu web.
- Revisar los estilos por defecto.
- Vuelve al uno hasta que todo esté bien.
- Simplifica el código, coméntalo y poco a poco ve probando hasta localizar el sitio donde se produce el problema.

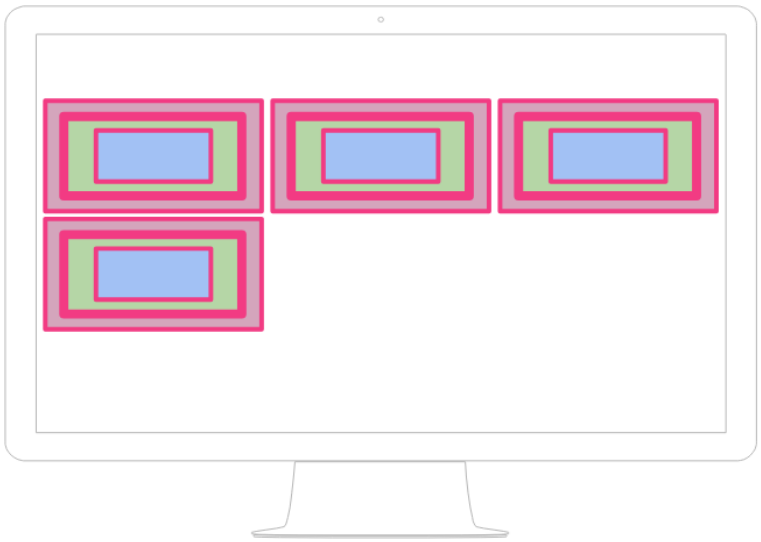
Es decir…***TEN PACIENCIA***

2.2 El flujo de los elementos en la ventana de nuestro navegador

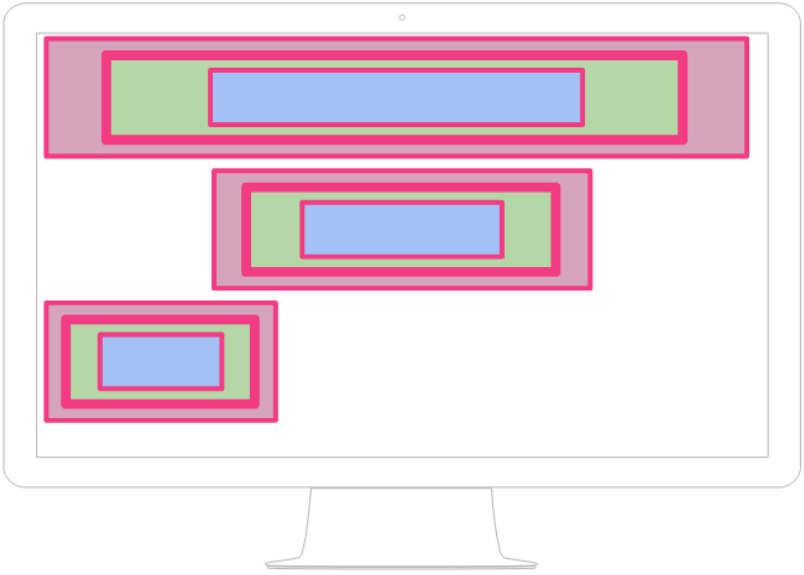
Después de los consejos prácticos del apartado anterior, en este apartado vamos a aclarar una serie de consideraciones en relación a la maquetación Web.

Consideraciones Importantes

- 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:



1. Determinados tipos de caja se van poniendo unas a continuación de otras mientras quepan en la pantalla.
2. Cuando no caben, las cajas pasan a la “siguiente línea” del navegador.



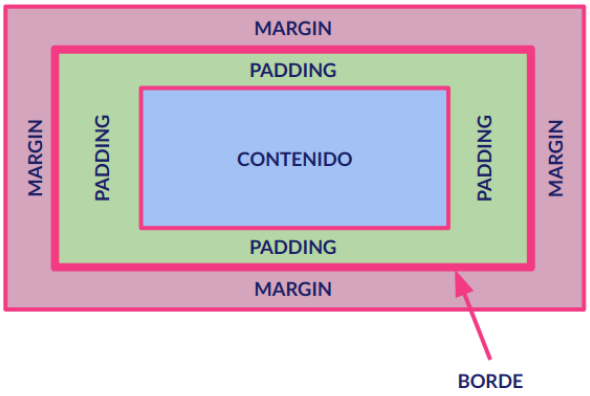
Otros tipos de caja provocan un “salto de línea”.

Y no hay más. Todas estas propiedades de las cajas y toda la maquetación la debemos hacer nosotros usando CSS.

Y eso es lo que vamos a empezar a ver a partir del siguiente apartado.

2.3 En línea y en bloque, la 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.



Algunos elementos fluyen uno a continuación de otro y otros se insertan directamente debajo de los anteriores, como por ejemplo los párrafos (<p>, <hn>, …). El comportamiento viene determinado por la propiedad CSS ***display***. Además, *display* se puede usar también para especificar el diseño usado por sus hijos.

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

- ***inline*** → El elemento se colocará a continuación del anterior, en la misma fila, si cabe. Muestra en la misma línea (respetando el flujo) todos los elementos y no acepta las propiedades width, height ni margenes verticales.
- ***block*** → Los elementos en bloque rompen el flujo de la línea y provocan *“un salto de línea”* tanto anterior como posterior. Acepta las propiedades width, height y márgenes verticales.
- ***inline-block*** → Su comportamiento es una mezcla entre los dos anteriores, se muestran en la misma línea (respetando el flujo) todos los elementos y, además, acepta las propiedades width, height y márgenes verticales.
- ***none*** → Una vez fijado, el elemento desaparece.
- ***contents*** → Al aplicarla, el padre es reemplazado por sus hijos y pseudoelementos. Es decir, aunque en el dom sigue apareciendo, visualmente no lo vemos, ya que no genera caja. Y sus hijos pasarían a ser hijos del padre del elemento reemplazado. Se usa sobre todo con *grid y flex.*
- **Valores relacionados con tablas**. En general, el elemento crecerá en anchura y altura como lo haría una tabla, una celda de una tabla, una columna de una tabla o una fila de una tabla. Un ejemplo de esto es que una celda cambia su anchura y/o altura y se ajusta al de la celda (de la fila

o de la columna) con más contenido. Es decir, podemos simular el comportamiento de una tabla aplicando estos valores a otros elementos. Posibles valores: *table, inline-table, table-row, table-column, table-cell*

- Valores ***flex*** y ***grid*** que veremos más adelante.

Si estamos interesados podemos descubrir la lista completa de valores **aquí**.

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. Ignoran *width* y *height*. Ejemplos: ,<img>,<a>, <b>, etc… Esta opción es la más apropiada para texto.
- 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. De esta manera tenemos los siguientes posibles valores.

- ***table*** → *maqueta como una tabla. Dentro del componente maquetado con esta opción podremos maquetar elementos con table-row o table-column.*
- ***table-row*** → *similar a filas de una tabla, es decir, el* ***alto*** *se ajustará al contenido de la celda (componente con display:table-cell) mayor.*
- ***table-cell*** → *celda de una tabla. Podremos maquetar con esta opción elementos que estén dentro de otro con table-row o table-column.*
- ***table-caption*** → similar al caption de una tabla.
- ***table-column*** → *similar a columnas, es decir, el* ***ancho*** *se ajustará al contenido de la celda (componente con display:table-cell) mayor.*
- ***table-column-group*** → *similar a colgroup. Podremos meter componentes usando table-column.*
- ***table-header-group***
- ***table-footer-group***
- ***table-row-group*** → *similar a <tbody>. Podremos meter componentes usando table-row.*

2.4 Tipos de layouts

Los layouts son la **ordenación y colocación de todos los elementos que componen una página web**. Dependiendo de cómo llevemos a cabo nuestra maquetación podemos tener distintos tipos de layouts:

1. Fixed
2. Elastic
3. Fluid
4. Max/min Layouts
5. Responsive

Fixed Layouts

Este tipo de layouts establecen un **tamaño fijo en pixels** para la anchura de los distintos elementos.

La principal ventaja al usar este tipo de layouts es que tengo control total, los distintos elementos van a tener siempre el tamaño que yo quiero.

Sin embargo, esta ventaja no compensa los principales problemas derivados que son:

- En pantallas pequeñas puede aparecer un scroll horizontal y esto es un **gran error** en diseño web, ya que las páginas con scroll horizontal son poco usables.
- En pantallas muy anchas puede que tenga mucho espacio en blanco a los lados del contenedor principal.

Elastic Layouts

Este tipo de layouts establece la **anchura de los elementos en *em*** que es el tamaño de letra por defecto (suele ser 16px).

- La principal ventaja es que al escalar el texto haciendo ***zoom in*** o ***zoom out*** los elementos cuyas dimensiones se hayan establecido en *em* escalarán correctamente. Se utilizan para páginas con gran contenido de texto.

Sin embargo, hay algunas desventajas, puede ser difícil de mantener y modificar y, además, pueden también provocar scroll horizontal como el *fixed layout*.

Fluid Layouts

Este tipo de layouts establecen el ancho de los distintos contenedores en **%** (**con respecto al contenedor/etiqueta padre**).

Si decidimos usar este tipo de layout la principal ventaja es que los elementos mantienen sus proporciones independientemente del tamaño de la pantalla.

Sin embargo, debemos afrontar unas importantes desventajas:

- En pantallas pequeñas las columnas pueden 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 (padres) 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.

Podría aplicarse también a la altura, aunque ya hemos visto que la altura es algo "menos" importante que la anchura a la hora de maquetar.

Layout Responsivos

Son aquellos layouts que 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.

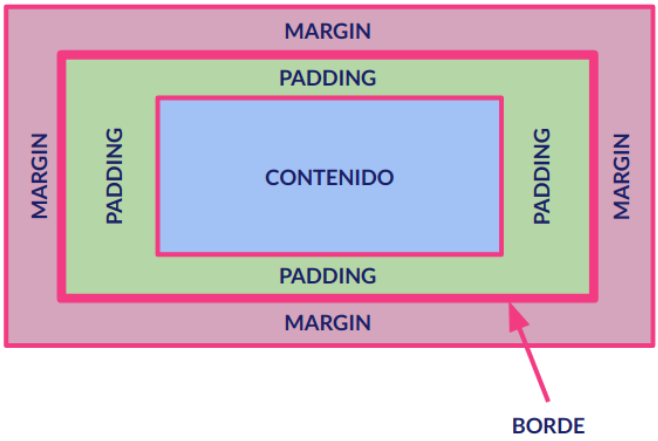
Un concepto similar y que con frecuencia se confunde es el ***layout adaptativo***, pero en estos casos no tengo un solo layout si no que tengo varios dependiendo de las características, y el cambio de uno a otro se produce de manera brusca. **En el diseño adaptativo (adaptive) se crean puntos fijos (breakpoints), estos se utilizan para fijar el límite en píxeles entre los dispositivos, ya sea móvil – tablet – escritorio.** Estos breakpoints ayudan a planificar y adaptar el diseño web a cada dispositivo y sus dimensiones; normalmente se hacen sobre los formatos estándar más utilizados (320px, 480px, 760px, 960px, 1200px, y 1600px).

No obstante, estos dos últimos se verán más adelante.

Entonces, ¿cuál es el mejor de todos? Depende de tu diseño y lo que queramos hacer. El diseño fluido con un ancho máximo y mínimo puede ser una de las soluciones más completas en muchos casos. En otros casos el uso de *em* puede ser también buena solución.

2.5 Box-sizing

En apartados anteriores, hemos comentado que todos los elementos de HTML al ser representados en los navegadores son *cajas*. Esas cajas tienen los siguiente elementos y apariencia:



- **Contenido** es la zona donde va el elemento propiamente dicho.
- **Padding** es la distancia entre el contenido y el borde.

- El **Borde** es el límite entre el elemento y el resto de los elementos.
- EL **Margin** es la separación entre el elemento y los demás elementos.

También recordamos que la maquetación web consiste en disponer estas cajas para que cada una ocupa 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. El margen no cuenta.
- La anchura del elemento: anchura del contenido + el padding + el borde. El margen no cuenta.

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;
}
```

O también:

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

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.

2.6 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 una sección etc…

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

En este apartado y en alguno de los siguientes vamos a dar unas pocas pautas generales que, una vez aprendidas, harán que todo esto de centrar sea mucho más fácil.

Para centrar vamos a diferenciar entre:

- El centrado en horizontal, es el más importante.
- El centrado en vertical.

Centrado en horizontal

Para centrar los elementos en horizontal:

- Si queremos centrar elementos en línea añadiremos la propiedad ***text-align:center*** al contenedor **padre**. Solo afectará a los elementos en línea (img, a, input, …) y no a los elementos en bloque (p, hn, …).

- Si queremos centrar **un elemento en bloque**, dentro de su etiqueta contenedora añadiremos la propiedad CSS ***margin: X auto*** **al elemento que queremos centrar** (***Recuerda:*** X es una distancia expresada en cualquier unidad, que expresa los márgenes por arriba y por abajo, mientras que el segundo parámetro, expresa los márgenes a izquierda y derecha). **El elemento debe tener anchura especificada.**

- Si tenemos varios elementos **en bloque** que queremos centrar y están 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.

- Usaremos contenedores **flex** que se verán más adelante.

Centrado en vertical

Centrado vertical para elementos en línea:

Se puede conseguir:

- Con el mismo ***margin*** arriba y abajo y convirtiéndolos en elementos en bloque con ***display:block***.
- Añadiendo ***vertical-align:middle*** si estamos dentro de una celda, de una tabla o lo estamos simulando con la propiedad display (ej: ***display:table*** o ***display:table-cell***).
- Con contenedores **flex** (se verá posteriormente).

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** (se verá posteriormente).

2.7 Propiedad Position: Posicionando elementos

En apartados anteriores hemos visto cómo se comportaban las etiquetas HTML, a nivel de diseño y estructura, cuando eran mostradas en el navegador.

Conforme vayamos aprendiendo, querremos hacer diseños más complejos y mover los elementos con respecto a dónde les correspondería según el flujo normal, atendiendo a su propiedad *display*.

Para este posicionamiento más exacto y concreto CSS nos proporciona 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 le corresponde. Aunque use top, bottom, left, right o z-index al elemento **no se aplica ningún desplazamiento**.
- ***relative:*** Es como *static* pero sí **atiende a los desplazamientos** expresados en top, bottom, left, right o z-index. Así, podré mover el elemento con ***position*** en función de la posición normal que le correspondería.
- ***fixed*** (fija)***:*** Se le aplica top, bottom, left, right o z-index **en relación con documento**. Así, este elemento permanecerá siempre en una misma posición tanto horizontal como vertical. **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***. Se usa para colocar o mover un elemento dentro de su elemento padre.
- ***sticky:*** Se comporta como *relative* hasta que el scroll llega a la posición del elemento y, a partir de entonces, se comporta como *fixed*.
- ***Inherit:*** La propiedad *position* **no se propaga en cascada**, como sí lo hacen otras propiedades CSS. Si queremos que sea así, añadiremos el valor ***inherit*** a los hijos que queremos que hereden.

La propiedad z-index

Al posicionar los elementos con las propiedades ***position*** puede suceder que nos encontremos que haya elementos que lleguen a solaparse por tener que ocupar la misma área.

Con z-index podemos establecer capas decidiendo el orden de solapamiento de estos elementos. Se mostrará **delante** aquel elemento de los que se solapen con el **mayor** valor de z-index.

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;

```
.contenedor {
    position: relative;
}

.elemento_a_centrar {
    height: 150px;
```

```
    margin-top: -75px; /** La mitad de la altura **/
    position: absolute;
    top: 50%;
}
```

Si desconocemos la altura del elemento:

```
.contenedor {
    position: relative;
}

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

2.8 Elementos flotantes

Aunque ya se ha visto, está muy relacionada con la maquetación. Pasamos a recordar qué hace esta propiedad.

Los elementos flotantes son aquellos que tienen la propiedad ***float***. Esta propiedad puede tener los valores: *left, right*. Está pensada para posicionar una imagen con respecto a un texto que la acompaña.

Para forzar que un elemento deje de "flotar" y pase a ocupar su posición normal, debemos usar la propiedad ***clear***. Esta propiedad acepta valores: *right, left o both*. Tras aplicar *clear,* ese elemento ya se añadirá tras el fin en vertical del elemento flotante.

clearfix hack → Si tenemos un contenedor y empezamos a flotar elementos, no podemos fijar el alto de ese contenedor. Una solución es usar la propiedad ***overflow-y*** o fijar una altura suficiente como para que los elementos no se salgan del contenedor:

```
.contenedor {
    overflow-y: auto;
    height: altura suficiente; /*Usar una de las dos*/
}
```

2.9 Overflow

La propiedad ***overflow*** permite regular la visibilidad de los contenidos que sobresalen de una caja html. Permite regular si los contenidos que sobresalen se seguirán viendo, si se ocultarán o si aparecerá una barra de scroll en el documento.

Los valores que puede tomar son: **hidden, scroll, visible** y **auto**.

- *hidden:* hace que el texto que sobresalga por encima del div no sea visible.

- *scroll:* crea una barra deslizante lateral.
- *visible:* hace que el texto sobresalga por encima del div.
- *auto:* similar a scroll, pero añade barra deslizante solo cuando son necesarios.

2.10 Visibility

Esta propiedad establece la visibilidad de un elemento. A diferencia de la propiedad **display**, cuando ocultamos elementos mediante la propiedad **visibility**, el espacio que ocupaban se mantendrá en el documento.

Para ocultar un elemento HTML usando la propiedad **visibility**, debemos establecer su valor como **hidden** o como **collapse**:

```
<div id="elemento" style="visibility:hidden;"></div>
```

Si queremos volver a poner visible el elemento, establecemos el valor de **visibility** a **visible**.