

***RECUERDA PONER A GRABAR LA  
CLASE***





***¿DUDAS DEL ON-BOARDING?***

**MIRALO AQUI**



**Clase 05.** DESARROLLO WEB

# ***BOX MODELING***



## ***OBJETIVOS DE LA CLASE***

- Comprender las cajas y sus propiedades.
- Modelar la página web.
- Conocer las posiciones de un elemento.

# GLOSARIO:

## Clase 4

**Joroba de camello:** permite que se puedan leer de forma más simple palabras compuestas.

**Reset CSS:** contienen en su código fuente definiciones para propiedades problemáticas, que los diseñadores necesitan unificar desde un principio.

### Unidades de medidas

#### Absolutas

- **Px (pixels):** es la unidad que usan las pantallas.

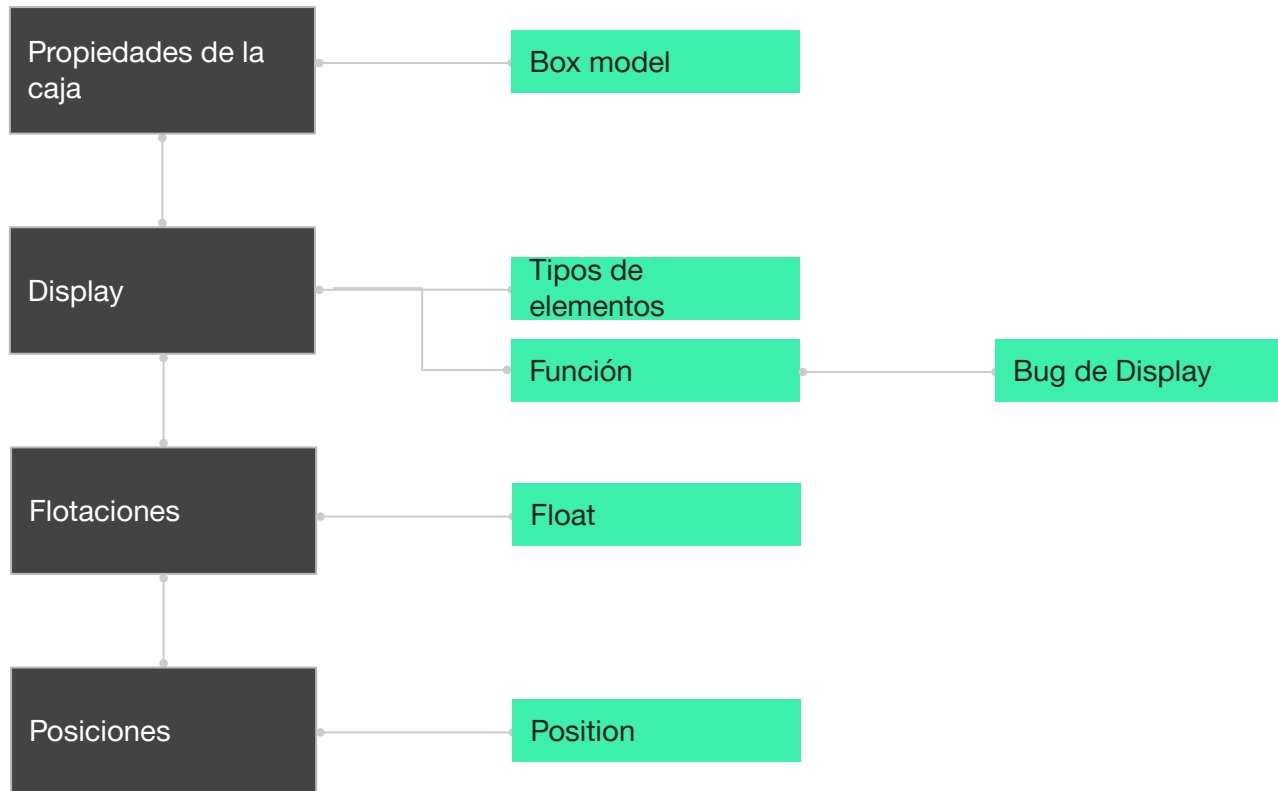
#### Relativas

- **Rem:** relativa a la configuración de tamaño de la raíz (etiqueta html).
- **Porcentaje:** tomando en cuenta que 16px es 100%.
- **Viewport:** se utilizan para layouts responsivos (más adelante).

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 5

¡Para  
recordar!



# ***CRONOGRAMA DEL CURSO***

## Clase 4



### **CSS - Medidas, colores, fuentes**



PRÁCTICAS DE LO VISTO EN CLASE



ESTILOS



TIPOGRAFÍA



ASIGNANDO ESTILOS



PRIMERA ENTREGA DEL PROYECTO FINAL

## Clase 5



### **Box Modeling**



PRÁCTICAS DE LO VISTO EN CLASE



BOX MODEL

## Clase 6



### **Flexbox**



PRÁCTICAS DE LO VISTO EN CLASE



APLICAR FLEXBOX





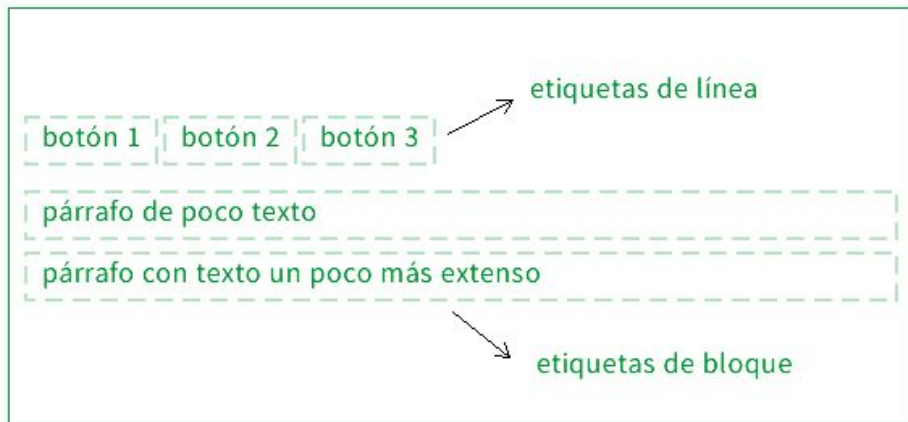
# ***GUIÓN DE LA CLASE***

Accede al material complementario [aquí](#).

# ***PROPIEDADES DE LA CAJA***

# PROPIEDADES DE LA CAJA

Todos los elementos del HTML son cajas. Un `<strong>`, un `<h2>` y demás, son rectangulares:



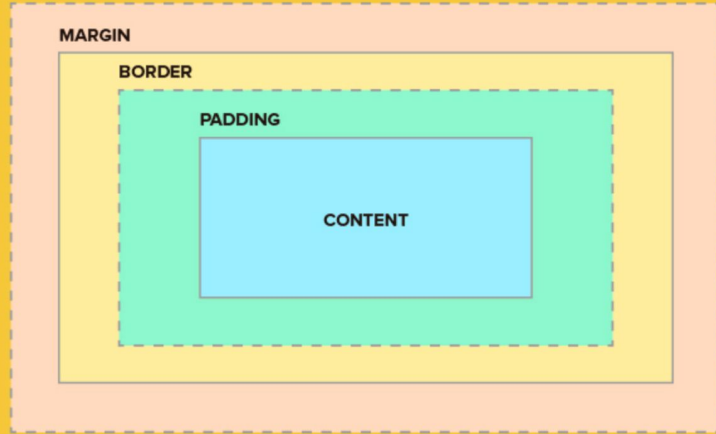
- En los elementos de **línea**, se verá uno al lado del otro.
- En cambio en los de **bloque**, uno debajo del otro.

# ***BOX MODEL***

Ese concepto de que “todo es una caja”, da lugar a algo llamado en web como **box model**. Sin importar si son de línea o de bloque (pero tienen su incidencia en lo que sean), todas las etiquetas tienen propiedades en común.

# PROPIEDADES EN COMÚN

## Box Model Css



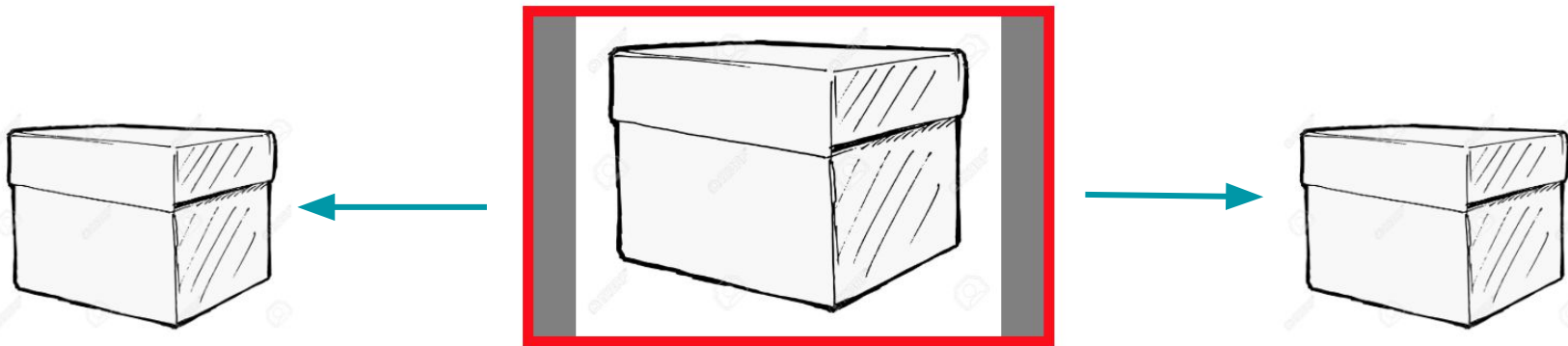
**CONTENT:** el espacio para el texto o imagen.

**BORDER:** el límite entre el elemento y el espacio externo.

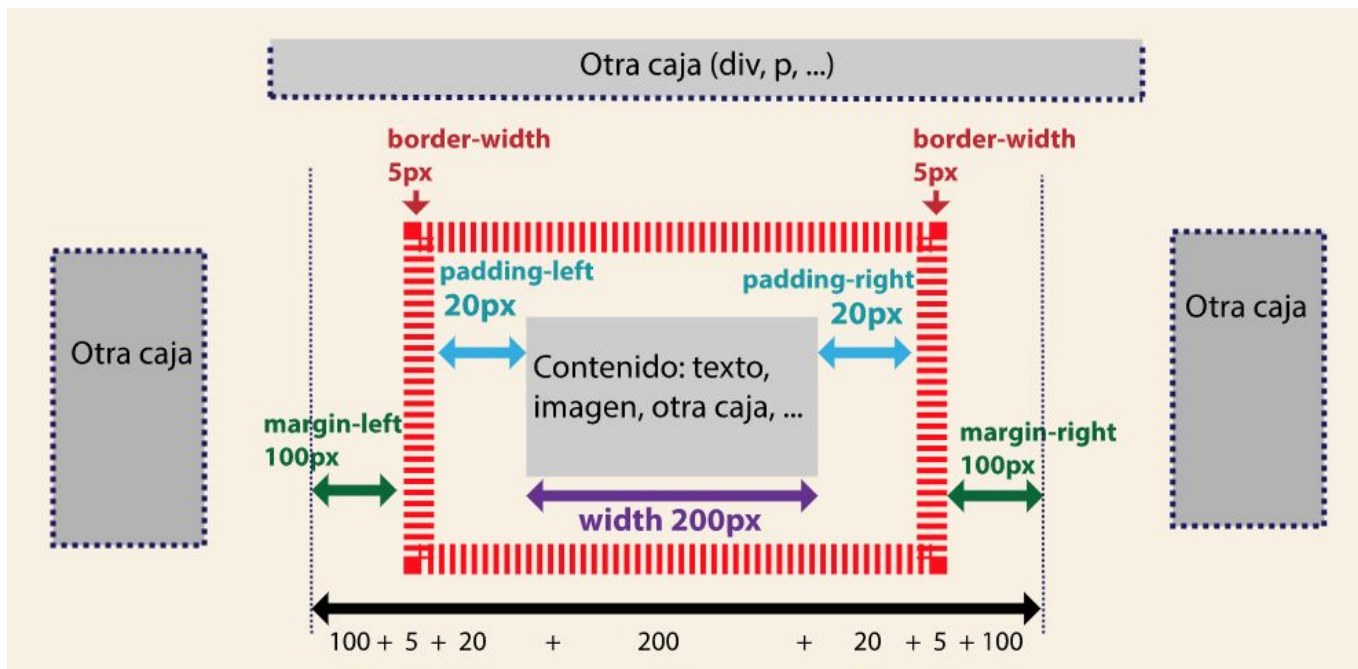
**PADDING:** separación entre el borde y el contenido de la caja. Es un espacio interior.

**MARGIN:** separación entre el borde y el afuera de la caja. Es un espacio exterior.

# ***EJEMPLO***



# EJEMPLO



# ***ALTO Y ANCHO*** ***(de los elementos)***

## ***Ancho***

Se denomina **width** a la propiedad CSS que controla la anchura de la caja de los elementos.

Dicha propiedad no admite valores negativos, y aquellos en porcentaje se calculan a partir de la anchura de su elemento padre.

## ***Alto***

La propiedad CSS que controla la altura de la caja de los elementos se denomina **height**.

No admite valores negativos, y aquellos en porcentaje se calculan a partir de la altura de su elemento padre.



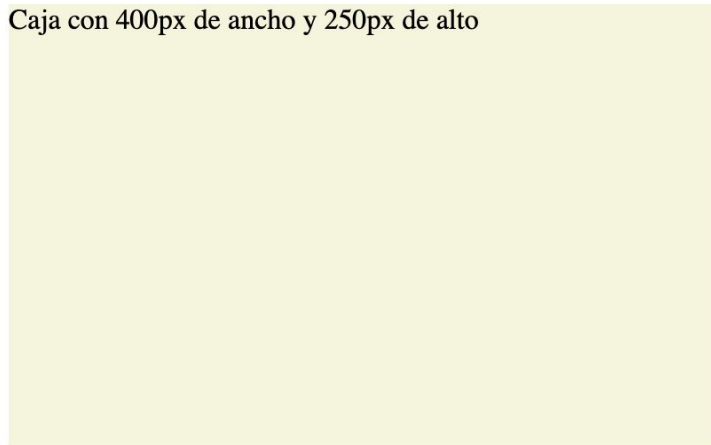
# ALTO Y ANCHO

## CSS

```
div {  
    background-color: beige;  
  
    width: 400px; /* ancho */  
    height: 250px; /* alto */  
}
```

## Se ve así

Caja con 400px de ancho y 250px de alto



Valores comunes: unidad (px, porcentaje, rem, viewport) | [Ejemplos y más información.](#)

# PROBLEMA COMÚN

## CSS

```
div {  
    background-color: beige;  
  
    /* no va a funcionar poner la  
    altura así, si su padre es body */  
    height: 100%;  
}
```

Se ve así

Lorem



???

Como el padre es BODY, y no tiene una altura definida no podrá aplicarse un 100% ([ver más](#)).

# SOLUCIÓN

## CSS

```
div {  
    background-color: beige;  
  
    /* al usar medida 'vh'  
    (viewport height) */  
    height: 100vh;  
}
```

## Se ve así

Lorem



Usando medidas “viewport” (tamaño de la ventana del navegador) es posible solucionarlo ([ver más](#)).

# ***ALGO MÁS PARA ACLARAR***

Cuando un elemento tiene un alto o ancho fijos, cualquier contenido que exceda la caja será visible. El inconveniente que esto genera es que, si luego se suma otro contenido, los mismos se van a superponer.



# EJEMPLO

## HTML

```
<div>
  CSS IS <strong>AWESOME</strong>
</div>
```



## CSS

```
div {
  /* propiedades decorativas */
  border: solid 1px black;
  padding: 5px;
  display: inline-block;
  font-size: 32px;
  font-family: Arial;

  /* propiedades que hacen el "problema" */
  width: 100px;
  height: 110px;
}
```

# ***OVERFLOW***

Propiedad: **overflow**

Tiene 4 valores posibles:

- **visible:** valor por defecto. El excedente es visible.
- **hidden:** el excedente no se muestra (lo corta) → **recomendado.**
- **scroll:** genera una barra de scroll en los dos ejes (x/y) del elemento, aunque no se necesite.
- **auto:** genera el scroll solo en el eje necesario.

Veamos cómo se ve aplicando el **overflow: hidden.**

# ***SOLUCIÓN***

## HTML

```
<div>  
  CSS IS <strong>AWESOME</strong>  
</div>
```



## CSS

```
div {  
  /* propiedades decorativas */  
  border: solid 1px black;  
  padding: 5px;  
  display: inline-block;  
  font-size: 32px;  
  font-family: Arial;  
  
  /* propiedades que hacen el "problema" */  
  width: 100px;  
  height: 110px;  
  
  /* solucion */  
  overflow: hidden;  
}
```

Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

***CODER HOUSE***

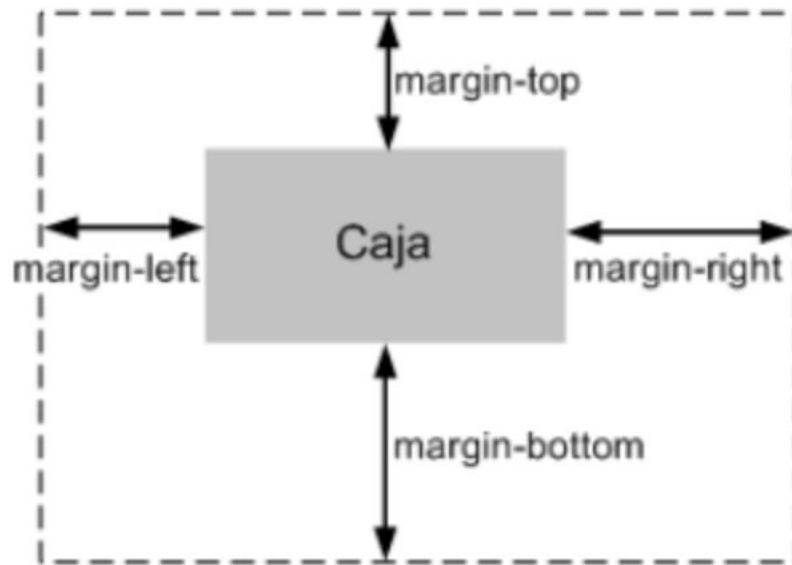


# ESPACIO EXTERIOR

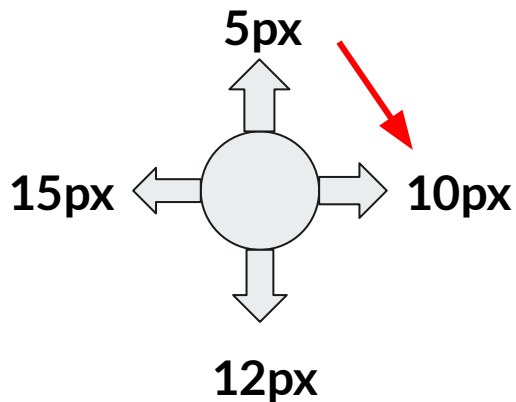
## Margin (márgenes)

Las propiedades `margin-top`, `margin-right`, `margin-bottom` y `margin-left`, se utilizan para definir los márgenes de cada uno de los lados del elemento por separado.

Puedes definir los 4 lados (forma abreviada “`margin`”) o sólo aquellos que necesites.



# CÓDIGO EJEMPLO



```
div {  
    margin-top: 5px;  
    margin-right: 10px;  
    margin-bottom: 12px;  
    margin-left: 15px;  
}
```

/\* forma abreviada pone en top,  
right, bottom, left \*/

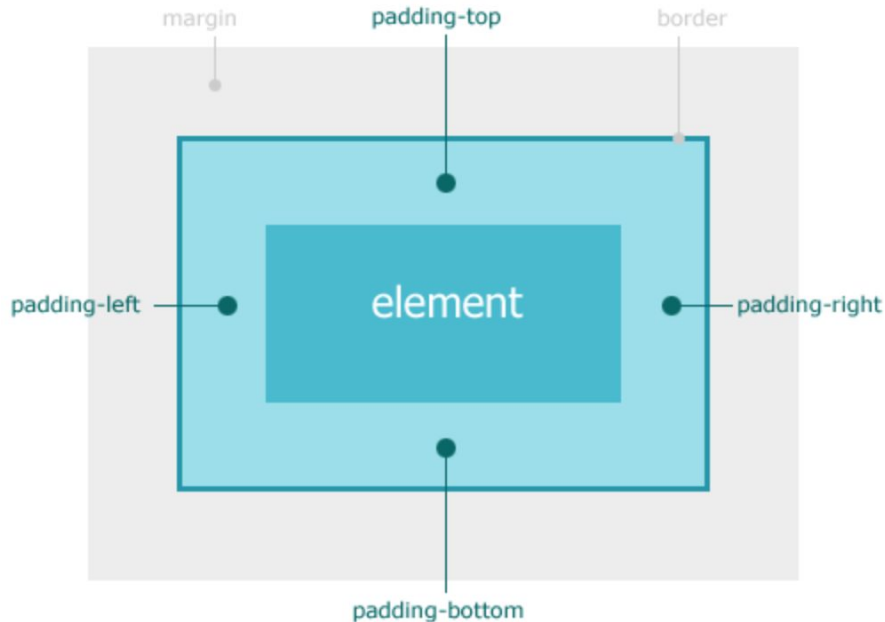
```
div {  
    margin: 5px 10px 12px 15px;  
}
```

# ESPACIO INTERIOR

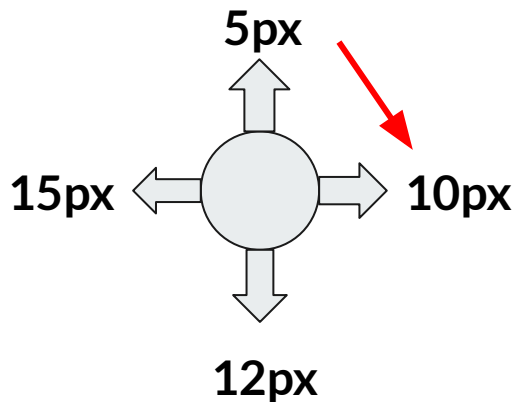
## ***Padding (relleno)***

Las propiedades `padding-top`, `padding-right`, `padding-bottom` y `padding-left`, se utilizan para definir los espacios internos de cada uno de los lados del elemento, por separado.

Puedes definir los 4 lados (forma abreviada “`padding`”) o sólo aquellos que necesites.



# CÓDIGO EJEMPLO



```
div {  
    padding-top: 5px;  
    padding-right: 10px;  
    padding-bottom: 12px;  
    padding-left: 15px;  
}
```

*/\* forma abreviada \*/*

```
div {  
    padding: 5px 10px 12px 15px;  
}
```

Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

***CODER HOUSE***

# BORDES

## ***Border***

Las propiedades `border-top`, `border-right`, `border-bottom`, y `border-left`, se utilizan para definir los bordes de cada lado del elemento por separado.

Puedes definir los 4 lados (forma abreviada “`border`”) o sólo aquellos que necesites.

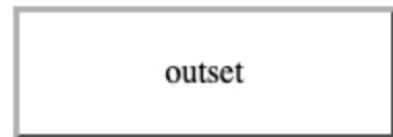
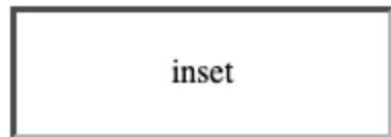
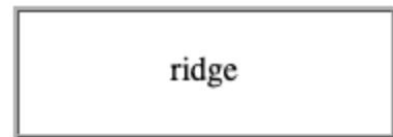
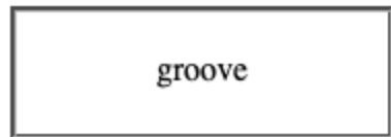
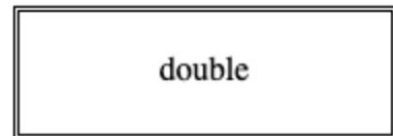
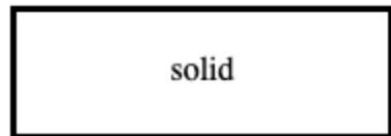


# BORDES

## Nota

A diferencia de los márgenes y padding, los bordes se forman con 3 valores:

- Tipo de borde ([border-style](#)).
- Grosor (-width).
- Color (-color).



none

hidden

# BORDES

## CSS

```
div {  
    border-top:solid 5px red;  
    border-right:solid 10px cyan;  
    border-bottom:solid 7px green;  
    border-left:solid 12px yellow;  
}
```

## Se ve así



Valores comunes: estilo grosor color| [Ejemplos y más información](#)



Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

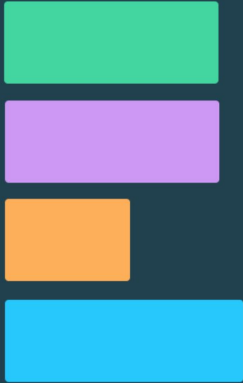
***DISPLAY***

# ***TIPOS DE ELEMENTOS***

- El estándar HTML clasifica a todos sus elementos en dos grandes grupos: elementos **en línea (*inline*)** y **de bloque (*block*)**.
- Los elementos de bloque siempre empiezan en una nueva línea, y ocupan todo el espacio disponible hasta el final de la misma (100%).
- Por otra parte, los elementos en línea no empiezan necesariamente en nueva línea y sólo ocupan el espacio necesario para mostrar sus contenidos.

# ***TIPOS DE ELEMENTOS***

## BLOCK VS INLINE



**display: block;**

Block elements stack, regardless of their width.



**display: inline;**

Inline elements flow from one line to the next.

# ***TIPOS DE ELEMENTOS***

- Los elementos **en línea** definidos por HTML son aquellos que se usan para marcar texto, imágenes y formularios.

[Ver listado de etiquetas de “en línea”.](#)

- Los elementos **de bloque** definidos por HTML se utilizan para marcar estructura (división de información/código)

[Ver listado de etiquetas de en bloque](#)

# ***DISPLAY***

Se encarga de definir **cómo se ve un elemento HTML**. Los dos comportamientos más importantes son:

- Pasar un elemento de bloque a uno de línea.
- Pasar un elemento de línea a uno de bloque.

Eso se hace con los valores **block** e **inline** respectivamente:

- **block:** convierte el elemento en uno de bloque.
- **Inline:** transforma el elemento en uno de línea.

# DISPLAY

HTML

```
<p>Lorem ipsum dolor sit  
amet, consectetur  
adipiscing elit.  
<span>Laudantium </span>  
perspiciatis itaque  
veritatis ea fugit qui.  
</p>
```

CSS

```
p { /*es un elemento en bloque que  
convierto en línea*/  
  display: inline;  
  background-color: yellow;}  
span { /*es un elemento en línea que  
convierto en bloque*/  
  display: block;  
  background-color: grey;}
```

Con este ejemplo podemos verificar cómo modifico el display de las etiquetas, puedes probar más [acá](#).

# DISPLAY

## *Inline-block*

Hay una propiedad que permite tomar lo mejor de ambos grupos, llamada “*inline-block*”. Brinda la posibilidad tener “*padding*” y “*margin*” hacia arriba y abajo.

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

Haz clic [aquí](#) para ver más ejemplos.



# ***BUG DEL DISPLAY: INLINE-BLOCK***

El display *inline-block* con ancho fijo pone las cosas una al lado de la otra (sí, lo dijimos recién). Pero si tienes los anchos milimétricamente calculados, puede ser que el último aparezca abajo (y no al lado).

Si entre los elementos de línea (inline o inline-block) hay “aire” (sea uno o 500 enter, espacios o tab), el mismo se muestra como un espacio de barra espaciadora. Esa es la razón por la cual no entran, y se muestra uno debajo de los demás.

# ***MENÚ CON DISPLAY***

## HTML

```
<ul>
  <li><a href="#home"
    class="activo">
    Home</a></li>
  <li><a href="#nosotros">
    Nosotros</a></li>
  <li><a href="#contacto">
    Contacto</a></li>
</ul>
```

## CSS

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

li {
  float: left; }

li a {
  display: inline-block;

.activo {
  background-color: blue;}
```

# ***TABLA COMPARATIVA***

Dependiendo de si la etiqueta de HTML es “ **de bloque**” o “**en línea**”, algunas propiedades serán omitidas ([más información](#)).

	<b>Width</b>	<b>Height</b>	<b>Padding</b>	<b>Margin</b>
<b>Bloque</b>	SI	SI	SI	SI
<b>En línea</b>	NO	NO	Solo costados	Solo costados
<b>En línea y bloque</b>	SI	SI	SI	SI

# QUITAR UN ELEMENTO

El display tiene también un valor para quitar un elemento del layout *display: none*; Lo oculta, y además lo quita (no ocupa su lugar).

```
div {  
    display: none;  
}
```

Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

# ***FLOTACIONES***

# ***FLOAT***

La flotación consiste en mover un elemento hacia la derecha o izquierda de su línea, y todo lo que viene después se acomodará en el “hueco” que queda vacío. Es una manera ‘old fashion’ de hacer una columna. Se usa la propiedad *float* que acepta dos valores:

- *left*: corre la caja a la izquierda.
- *right*: corre la caja a la derecha.

**Importante:** cuando un elemento flota, deja de pertenecer al flujo normal del HTML.

# FLOAT

## CSS

```
divgreen {  
  width: 400px;  
  background-color: rgb(238, 255, 65);  
  float: left;  
  padding: 10px;  
}
```

Este es el div que contiene dos columnas. Tiene un fondo rojo y un padding de 10 px;

Esta es la columna 1, Tiene fondo color verde, Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Esta es la columna 2, Tiene fondo color naranja, Duis aute irure dolor in reprehenderit in voluptate velit esse

Esto viene despues del div de dos columnas



# ***ADVERTENCIA DE FLOAT***

Todo elemento flotado, deja de “empujar” en alto a su contenedor. Este último colapsa su altura si todos los elementos flotan.

## **¿Cómo solucionarlo?**

Si googleas sobre el tema, encontrarás muchas técnicas para solucionar el problema del float. Al elemento que se colapsa, dale un *overflow* (excedente) con cualquier valor **-menos scroll-**.

# ***CLEAR***

La propiedad *clear* permite modificar el comportamiento por defecto del posicionamiento flotante, para forzar a un elemento a mostrarse debajo de cualquier caja flotante. La regla CSS que se aplica al segundo párrafo del ejemplo anterior (el cuadro azul) es la siguiente →

## **CSS**

```
divblue {  
    clear: left;  
    background-color: blue;  
    width: 400px;  
}
```

# ***POSICIONES***

# ***POSITION***

Es una propiedad CSS pensada para ubicar un elemento, con una libertad muy flexible. Algunos ejemplos de uso:

- Superponer elementos.
- Crear publicidades que te sigan con el scroll o un menú.
- Hacer un menú con submenú adentro.

Valores posibles: relative, absolute, fixed, o sticky (cualquiera excepto static).

# ¿CÓMO UBICAR UN ELEMENTO?

1

Define qué tipo de posición quieres usar.

2

Indica desde dónde calcular la distancia (si será desde arriba, derecha, abajo o izquierda).

3

Determina un valor numérico para las propiedades **top**, **bottom**, **left**, **right**.

¿Recuerdas cuando ubicábamos los fondos? Similar, tiene algunas particularidades.

**CODER HOUSE**

# ***POSITION***

Al aplicar esta propiedad, puedes usar 4 propiedades para posicionar los elementos, y debes darles un valor numérico. Ellas son:

- **top:** calcula desde el borde superior (ej: `top: 100px`).
- **right:** calcula desde el borde derecho (ej: `right: 50px`).
- **bottom:** calcula desde el borde inferior (ej: `bottom: 100px`).
- **left:** calcula desde el borde izquierdo (ej: `left: 50%`).

Haz clic [aquí](#) para acceder a más información.

# ***POSITION: RELATIVE***

El elemento es posicionado de acuerdo al flujo normal del documento, y luego es **desplazado *en relación a sí mismo***.

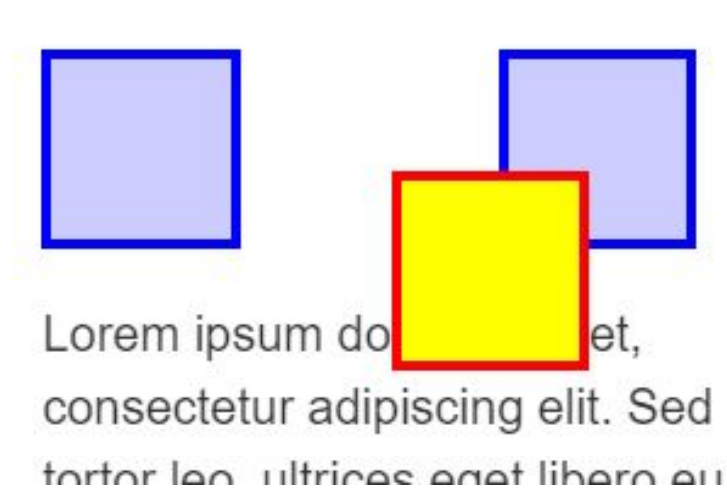
El desplazamiento no afecta la posición de ningún otro elemento, provocando que se pueda superponer sobre otro.

# ***POSITION: RELATIVE***

## CSS

```
div {  
  width: 100px;  
  height: 100px;  
  
  position: relative;  
  top: 40px;  
  left: 40px;  
}
```

Se ve así





# ***POSITION: ABSOLUTE***

El elemento es removido del flujo normal del documento, sin crearse espacio alguno para el mismo en el esquema de la página.

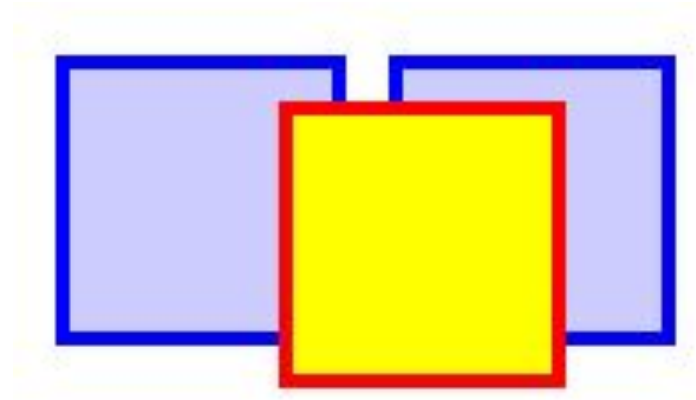
Es posicionado relativo a su padre, siempre y cuando su padre tenga “position:relative”. De lo contrario, se ubica relativo al body. Se recomienda establecer un ancho y alto (width, height).

# ***POSITION: ABSOLUTE***

## **CSS**

```
div {  
  width: 100px;  
  height: 100px;  
  
  position: absolute;  
  top: 40px;  
  left: 40px;  
}
```

Se ve así



# ***POSITION: FIXED Y STICKY***

Ambos métodos permiten que el elemento se mantenga visible, aunque se haga scroll.

# ***FIXED***

Esta posición es similar a la absoluta, con la excepción de que el elemento contenedor es el “viewport”, es decir, la ventana del navegador.

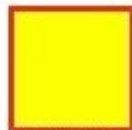
Puede ser usada para crear elementos que floten, y que queden en la misma posición aunque se haga scroll.

# FIXED

## CSS

```
div {  
  width: 300px;  
  background-color: yellow;  
  
  position: fixed;  
  top: 0;  
  left: 0;  
}
```

## Se ve así



Now you can control the position property for the yellow box.



To see the effect of sticky positioning, select the `position: sticky` option and scroll this container.

# ***STICKY***

El elemento es posicionado en el “flow” natural del documento, podría decirse que es un valor que funciona de forma híbrida, es decir, como “relative” y también “fixed”.

Esto es, cuando llega el “viewport” (la ventana del navegador) hasta donde se encuentra, se “pegará” sobre el borde superior.

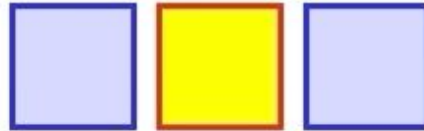
# STICKY

## CSS

```
div {  
    position: sticky;  
    top: 20px;  
}
```

## Se ve así

In this demo you can control the `position` property for the yellow box.



To see the effect of `sticky` positioning, select the `position: sticky` option and scroll this container.

# ***MENÚ CON SUBMENÚ***

- El position (tanto *relative* como *absolute*) se usa, entre otras, para hacer un menú que tenga un submenú emergente. Los ítems del primero son relativos, sirven como borde de cualquier hijo.
- La lista dentro de un list-item es absoluta. Por defecto, la sublista tiene *display: none*. Recién cuando un list-item detecte el *:hover*, si adentro tiene una lista, dale *display: block* (no te preocupes, esto quedará quedará más claro con el ejemplo que veremos a continuación).



# ***MENÚ CON SUBMENÚ***

Este list-item es relative

Home	A propos	Contact	Archives	Catégories	
				Apple	
				iPod/iPhone	
				Mac	
				GNU/Linux et Open-Source	
				Scripts Shell	
				Internet Explorer	
				Windows Vista	
				Windows XP	

Por defecto la lista es display none  
Cuando se pasa el mouse por arriba  
del li se muestra la sub-lista con  
display block

```
ul li:hover ul{ display: block }
```

La lista es absolute.

Como está en el <li> relative, lo usará como límites del top, bottom, left y right

# HTML

```
<ul>
  <li><a href="">Item</a></li>
  <li><a href="">Item</a></li>
  <li><a href="">Item</a>
    <ul>
      <li><a href="">Subitem</a></li>
      <li><a href="">Subitem</a></li>
    </ul>
  </li>
  <li><a href="">Item</a></li>
</ul>
```

# CSS

```
ul {
  list-style: none;
  font-size: 0 /* truco por el uso de inline-block */
}
li {
  display: inline-block;
  width: 25%;
  position: relative;
  font-size: 14px
}
ul ul {
  position: absolute;
  display: none;
}
ul ul li {
  display: block;
}
ul li:hover ul {
  display: block;
}
```

# ***PROPIEDAD Z-INDEX***

***(para el orden de superposición)***

El z-index entra en juego cuando dos elementos que tienen *position* se superponen. Esta propiedad acepta como valor un número (sin ninguna unidad, ni px, ni cm, ni nada); a valor más alto, se mostrará por encima de los demás elementos.

Por defecto, todos los objetos tienen z-index:1. Si dos objetos tienen el mismo valor de z-index y se superponen, el que fue creado después en el HTML se verá encima del otro.

Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

***CODER HOUSE***



## ***BOX MODEL***

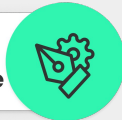
Incluye box model en tu proyecto final.

# BOX MODEL

**Formato:** archivo HTML y CSS. Debe tener el nombre “Idea+Apellido”.

**Sugerencia:** carpeta en formato ZIP o RAR, con el o los archivos HTML y CSS.

Desafío  
entregable



**>> Consigna:** agrega al CSS del Proyecto Final (¿Cuál necesitan: espacio exterior, interior, bordes?).

- Márgenes.
- Rellenos.
- Bordes.
- Menú.

**>>Aspectos a incluir en el entregable:** modifica el menú con las propiedades de Display. Sumar márgenes, rellenos y bordes a las secciones dentro de las páginas. No necesariamente tienen que estar las tres propiedades a la misma sección, pueden estar aplicadas a diferentes secciones diferentes propiedades.

**>>Ejemplo:** [Carpeta comprimida con los archivos de box modeling](#)

**CODER HOUSE**



## ***BOX MODEL***

Incluir box modeling en tu proyecto final.

# ***BOX MODEL***

**Formato:** Archivo html y css

**Sugerencia:** carpeta en formato zip o rar con el/los archivos html y CSS.

Desafío  
Complementario

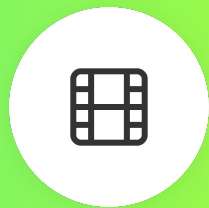


**>> Consigna:** Luego de resolver el desafío entregable Box Modeling, agrega al proyecto el menú con submenú.



***¿PREGUNTAS?***





***¿QUIERES SABER MÁS? TE DEJAMOS  
MATERIAL AMPLIADO DE LA CLASE***

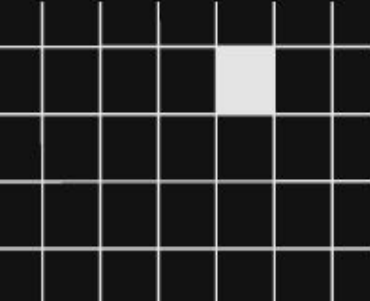


- [Imágenes de relleno](#) | ***placekitten***
- [Imágenes de relleno](#) | ***placedog.net***
- [The CSS Box Model](#) | ***CSS-TRICKS***



# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Cajas y sus propiedades.
  - Modelación de la página web.
  - Posiciones de un elemento.
- 



***OPINA Y VALORA ESTA CLASE***