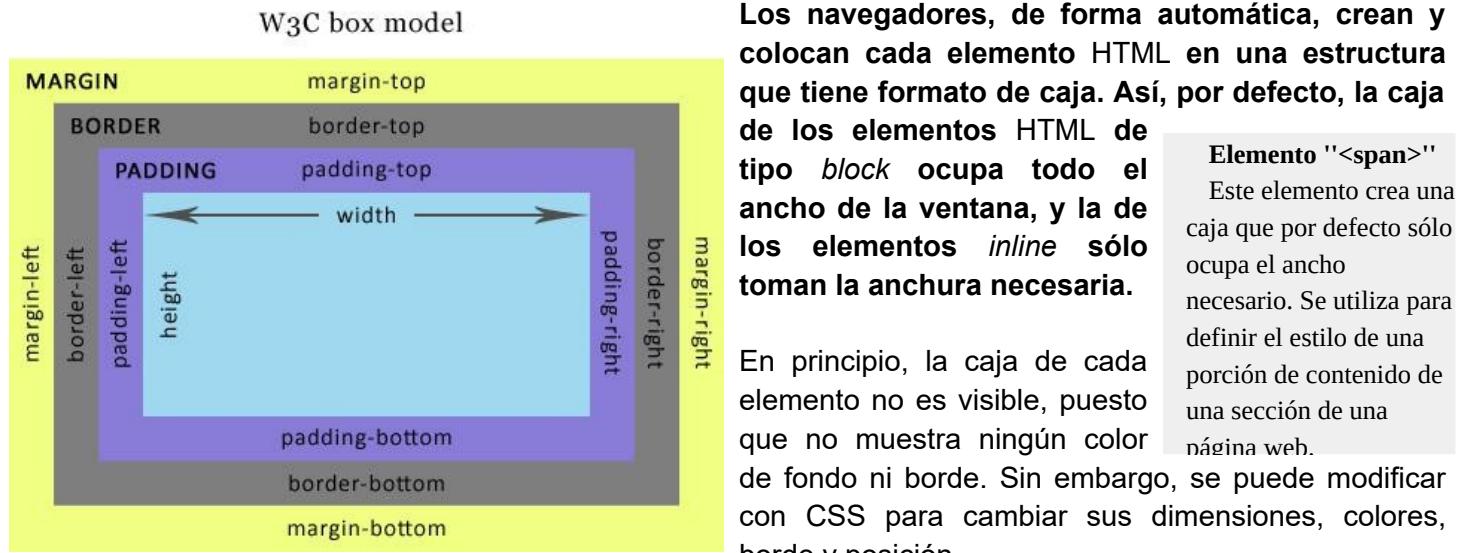


2.5 Modelo de cajas: márgenes, relleno y bordes.



Se pueden modificar las propiedades de la caja de cualquier elemento HTML, pero generalmente se utiliza el elemento `<div>` para organizar y dar estructura al diseño de las páginas web.

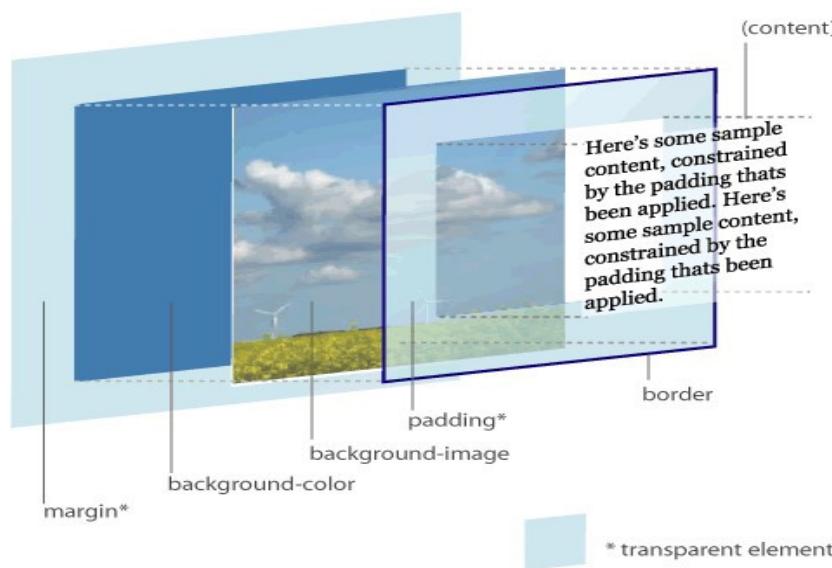
Dimensiones de la caja

Las partes que componen una caja en su orden de visualización desde el punto de vista del usuario son:

- **Contenido , content:** se refiere al contenido del elemento; por ejemplo, el texto, una imagen, etc.
- **Relleno, padding:** espacio libre opcional existente entre el contenido y el borde.
- **Borde , border:** línea que envuelve el contenido y el relleno.

Elemento "<div>"

Este elemento crea una caja que por defecto ocupa todo el ancho del navegador. Se utiliza para definir el estilo de una sección de una página web.



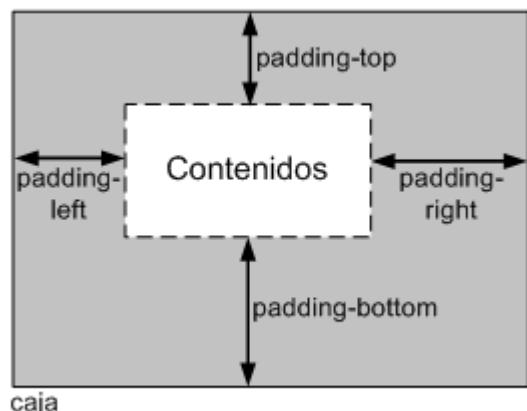
- **Imagen de fondo, background image:** imagen que se muestra por detrás del contenido y el relleno.
- **Color de fondo, background color:** color que se muestra detrás del contenido y el relleno.
- **Márgen, margin:** separación opcional existente entre la caja y el resto de cajas adyacentes.

* transparent elements

2.5.1 Padding o relleno

El padding es el **margin interno** de un elemento, también se le llama relleno y es la **cantidad de espacio entre el borde y el contenido del elemento**.

Veamos una tabla con las propiedades para dar un margen interno a un elemento.



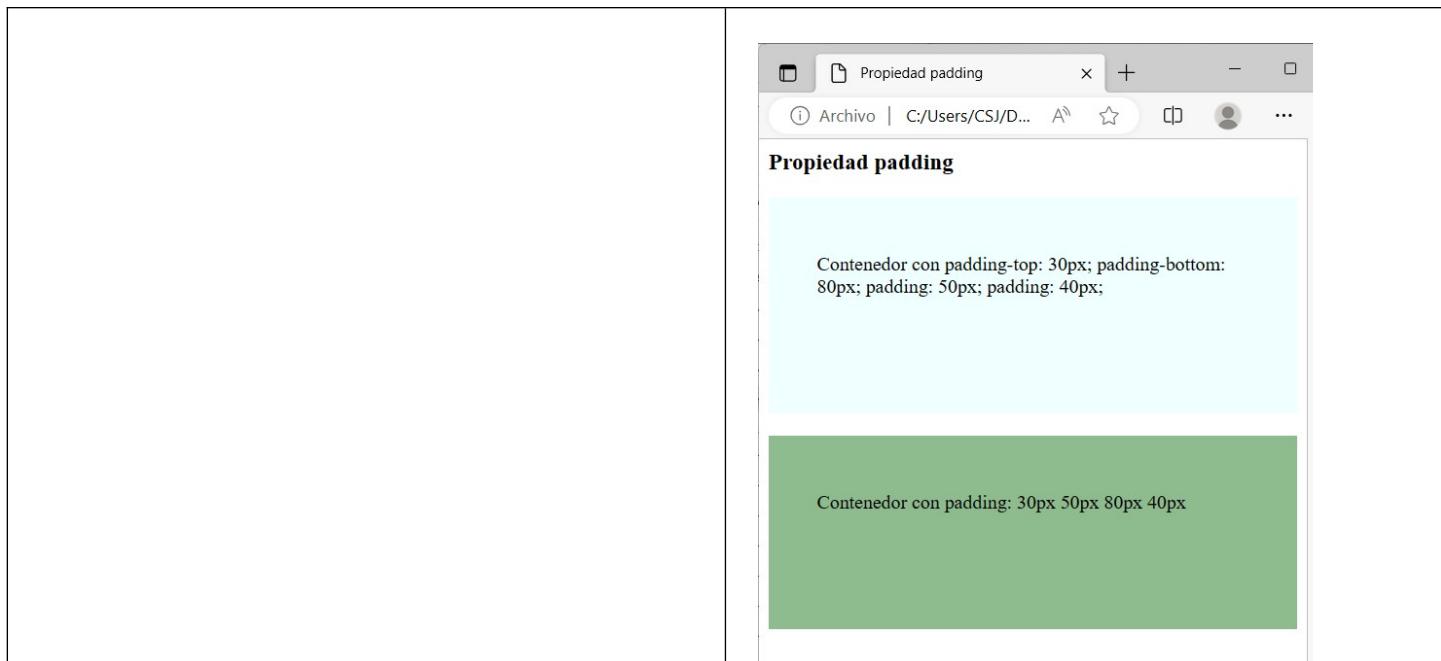
Propiedad	Descripción	Valores
Padding-top padding-right padding-bottom padding-left padding	Tamaño del relleno superior, derecho, inferior e izquierdo Tamaño del relleno	longitud porcentaje longitud porcentaje {1,4}

Valores del padding (también aplicables a la propiedad margin):

- **Un valor:** se aplica el estilo a los **4 lados**.
- **Dos valores:** el primer valor se aplica a **arriba y abajo**, el segundo valor se aplica a **derecha e izquierda**.
- **Tres valores:** el primer valor se aplica a **arriba**, el segundo valor a **derecha e izquierda** y el tercer valor se aplica a **abajo** del elemento
- **Cuatro valores:** el primer valor se aplica a **arriba**, el segundo valor se aplica a la **derecha**, el tercer valor se aplica a **abajo** y el cuarto valor se aplica a la **izquierda**.

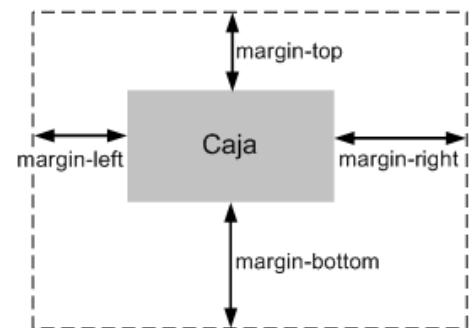
Ejemplo Padding

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad padding</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>Propiedad padding</h3> <div class="a" border="1"> <p>Contenedor con padding-top: 30px; padding-bottom: 80px; padding: 50px; padding: 40px;</p> </div>
 <div class="b" border="1"> <p>Contenedor con padding: 30px 50px 80px 40px</p> </div> </body> </html></pre>	<pre>div.a { padding-top: 30px; padding-bottom: 80px; padding-right: 50px; padding-left: 40px; background-color: azure; } div.b { /*Propiedad corta*/ padding: 30px 50px 80px 40px; background-color: darkseagreen; }</pre>



2.5.2 Margin

El **margin** es el **margen externo** de un elemento, fuera de cualquier borde definido. Veamos las distintas propiedades para dar estilo a los márgenes de un elemento.



Propiedad	Descripción	Valores
Margin-top margin-right margin-bottom margin-left margin	Tamaño del margen superior, derecho, inferior e izquierdo Ancho de los márgenes	longitud porcentaje auto longitud porcentaje auto {1,4}

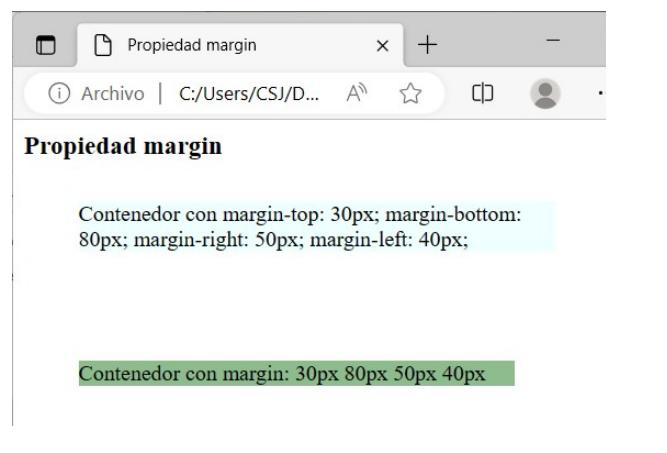
Ejemplo Margin

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad margin</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>Propiedad margin</h3> <div class="a" border="1"> <p>Contenedor con margin-top: 30px; margin-</pre>	<pre>div.a { margin-top: 30px; margin-bottom: 80px; margin-right: 50px; margin-left: 40px; background-color: azure; } div.b /*Propiedad corta*/ margin: 30px 80px 50px 40px; background-color: darkseagreen;</pre>

```

bottom: 80px; margin-right: 50px; margin-left:
40px;</p>
</div>
<div class="b" border="1">
<p>Contenedor con margin: 30px 80px 50px
40px</p>
</div>
</body>
</html>

```



2.5.3 Border

La propiedad **border** en CSS permite **especificar el estilo, el ancho y el color de los bordes de un elemento**. Puedes usar diferentes valores para crear distintos tipos de bordes, como líneas lisas, de puntos, redondeados, etc. Veamos las propiedades para dar estilo a los bordes de un elemento.

Propiedad	Descripción	Valores
border-top-width border-right-width border-bottom-width border-left-width	Anchura del borde superior, derecho, inferior o izquierdo	thin medium thick longitud
border-width	Anchura del borde (reducida)	thin medium thick longitud {1,4}
border-top-color border-right-color border-bottom-color border-left-color	Color del borde superior, derecho, inferior e izquierdo	color transparent
border-color	Color del borde (reducida)	color transparent {1,4}
border-top-style border-right-style border-bottom-style border-left-style	Estilo del borde superior, derecho, inferior e izquierdo	none hidden dotted dashed solid double groove ridge inset outset
border-style	Estilo del borde (reducida)	none hidden dotted dashed solid double groove ridge inset outset {1,4}
Border-top border-right border-bottom border-left	Ancho, estilo y color para el borde superior, derecho, inferior e izquierdo	border-top-width border-top-style border-top-color
border	Ancho, estilo y color para todos los bordes (reducida)	border-width border-style border-color
border-radius	Curvatura del borde	longitud porcentaje {1,4}

Las propiedades del borde no tendrán efecto hasta que se defina la propiedad **border-style.

Ejemplo border

HTML	CSS
<pre> <html> <head> <meta charset="utf-8"> <title>Propiedad margin</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>Propiedad border</h3> </pre>	<pre> .a { border: 4px solid blue; } .b { border: 6px dotted green; } .c { border: 3px dashed purple; } .d { border-top: 3px double orange; border-right: 3px double brown; border-bottom: 3px double brown; border-left: 3px double brown; } </pre>

```

<p class="a">Párrafo con un borde sólido de color azul</p>
<p class="b">Párrafo con borde verde punteado</p>
<p class="c">Párrafo con borde violeta a rayas</p>
<p class="d">Párrafo con borde mixto</p>
</body>
</html>

```

Propiedad border

Párrafo con un borde sólido de color azul

Párrafo con borde verde punteado

Párrafo con borde violeta a rayas

Párrafo con borde mixto

}

Ejemplo border-radius

HTML	CSS
<pre> <html> <head> <meta charset="utf-8"> <title>Propiedad margin</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="a"></div>
 <div class="b"></div>
 <div class="c"></div>
 </body> </html> </pre>	<pre> .a { background-color: blue; border-radius: 20px 20px 0 0; height:100px; width: 100px; } .b { background-color: green; border-radius: 0 0 40px 40px; height:100px; width: 100px; } .c { background-color: purple; border-radius: 50px; height:100px; width: 100px; } </pre>

2.5.4 Propiedades width, height, max-width y min-height

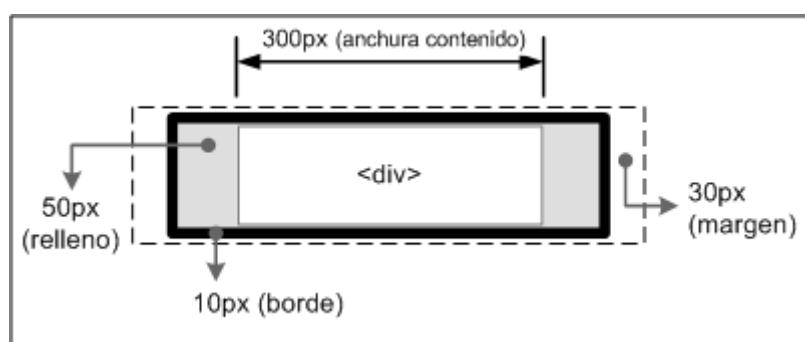
Veamos una tabla resumen de las propiedades que vamos a tratar en esta sección:

Nombre propiedad	Descripción	Valores
width	Establece el ancho del área de contenido de un elemento	Unidad de longitud (px, em, %, etc.), auto, initial, inherit
height	Establece el alto del área de contenido de un elemento	Unidad de longitud (px, em, %, etc.), auto, initial, inherit
max-width	Establece el ancho máximo que puede tener un elemento	Unidad de longitud (px, em, %, etc.), none, initial, inherit
min-width	Establece el ancho mínimo que debe tener un elemento	Unidad de longitud (px, em, %, etc.), 0, initial, inherit

La anchura y altura de un elemento no solamente se calculan teniendo en cuenta sus propiedades width y height. El **margin, el relleno y los bordes establecidos a un elemento determinan la anchura y altura final** del elemento. En el siguiente ejemplo se muestran los estilos CSS de un elemento:

```
div {  
    width: 300px;  
    padding-left: 50px;  
    padding-right: 50px;  
    margin-left: 30px;  
    margin-right: 30px;  
    border: 10px solid black;  
}
```

La **anchura total** con la que se muestra el elemento **no son los 300 píxel indicados en la propiedad width, sino que también se añaden todos sus márgenes, rellenos y bordes**:



De esta forma, la anchura del elemento en pantalla sería igual a la **suma de la anchura original, los márgenes, los bordes y los rellenos**:

$$30\text{px} + 10\text{px} + 50\text{px} + 300\text{px} + 50\text{px} + 10\text{px} + 30\text{px} = 480 \text{ píxel}$$

Así, la **anchura/altura establecida con CSS siempre hace referencia a la anchura/altura del contenido**. La **anchura/altura total del elemento debe tener en cuenta además los valores del resto de partes que componen la caja del box model**.

IMPORTANTE: La propiedad **box-sizing** puede ser usada para ajustar este comportamiento:

- **content-box** es el comportamiento CSS **por defecto** para el tamaño de la caja (box-sizing). Si se define el ancho de un elemento en 100 pixeles, la caja del contenido del elemento tendrá 100 pixeles de ancho, y el ancho de cualquier borde o relleno se añadirá al ancho final desplegado.
- **border-box** le dice al navegador que **tome en cuenta cualquier valor que se especifique de borde o de relleno para el ancho o alto de un elemento**. Es decir, si se define un elemento con un ancho de 100 pixeles **incluirán** cualquier borde o relleno que se añadan, y la caja de contenido se encogerá para absorber ese ancho extra. Esto típicamente **hace mucho más fácil dimensionar elementos**.

Se estudiará la propiedad en el siguiente apartado.

Ejemplos de uso de width y height con diferentes unidades:

- Para darle un ancho y un alto fijo a un elemento, podemos usar píxeles como unidad. Por ejemplo, si queremos que un elemento tenga un ancho de 200px y un alto de 100px, podemos usar este código:

```
div {  
    width: 200px;  
    height: 100px;  
}
```

- Para darle un ancho y un alto relativo al elemento contenedor, podemos usar porcentajes como unidad. Por ejemplo, si queremos que un elemento ocupe el 50% del ancho y el 25% del alto del elemento padre, podemos usar este código:

```
div {  
    width: 50%;  
    height: 25%;  
}
```

Las propiedades CSS **max-width** y **max-height** se usan para establecer el ancho y el alto máximo de un elemento. Estas propiedades nos permiten limitar el tamaño de un elemento para que no supere cierto valor, incluso si le damos un ancho o un alto mayor con las propiedades **width** o **height**.

Esto puede ser muy útil para crear diseños responsivos que se adapten a diferentes tamaños de pantalla. Por ejemplo, podemos usar **max-width** para evitar que una imagen se salga del contenedor o que se distorsione al cambiar la resolución. También podemos usar **max-height** para controlar la altura de un elemento que tenga contenido variable, como un menú desplegable o una caja de texto.

Ejemplos de uso de max-width y max-height con diferentes unidades:

- Para darle un ancho y un alto máximo a un elemento en píxeles, podemos usar esta unidad como valor. Por ejemplo, si queremos que una imagen no supere los 300px de ancho ni los 200px de alto, podemos usar este código:

```
img {  
    max-width: 300px;  
    max-height: 200px;  
}
```

- Para darle un ancho y un alto máximo a un elemento en porcentajes, podemos usar esta unidad como valor. Por ejemplo, si queremos que una imagen no ocupe más del 50% del ancho ni del 25% del alto del elemento contenedor, podemos usar este código:

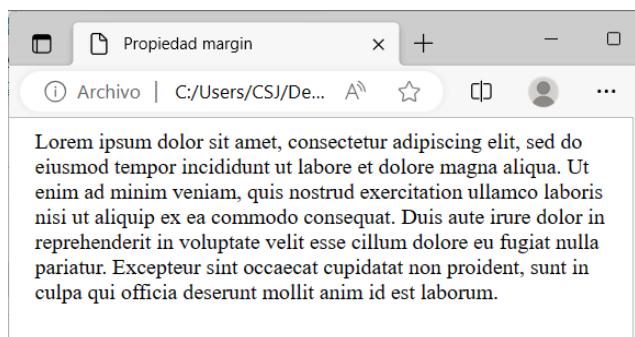
```
img {  
    max-width: 50%;  
    max-height: 25%;  
}
```

Veamos un ejemplo práctico: Para crear un contenido a pantalla completa que no se haga más grande al

llegar a un ancho máximo, puedes usar la propiedad **max-width** junto con la propiedad **width**. Así, le das al elemento un ancho relativo al tamaño de la pantalla, pero también le pones un límite absoluto.

Por ejemplo, si quieras que el contenido ocupe el 95% del ancho de la pantalla, pero que no supere los 1200px, puedes usar este código:

```
.content {
  width: 95%; /* El contenido ocupa el 95% del ancho de la pantalla */
  max-width: 1200px; /* El contenido no ocupará más de 1200px */
  margin: 0 auto; /* Centrar el contenido, primer valor para arriba y abajo, segundo valor para izd y dcha */
}
```

HTML	CSS
<pre><code><html> <head> <meta charset="utf-8"> <title>Propiedad margin</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="content">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. 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.</div> </body> </html></code></pre>	<pre><code>.content { width: 95%; /* El contenido ocupa el 95% del ancho de la pantalla */ max-width: 1200px; /* El contenido no ocupará más de 1200px */ margin: 0 auto; /* Centrar el contenido, primer valor para arriba y abajo, segundo valor para izd y dcha */</code></pre> 

2.5.5 Posición y comportamiento de contenedores CSS

En el proceso de creación de una web es imprescindible **organizar elementos como imágenes, textos o tablas**. Para ello, necesitaremos conocer los elementos de ordenación y las propiedades que nos ayudan a organizar todos los componentes. Las propiedades más importantes se definen en la siguiente tabla:

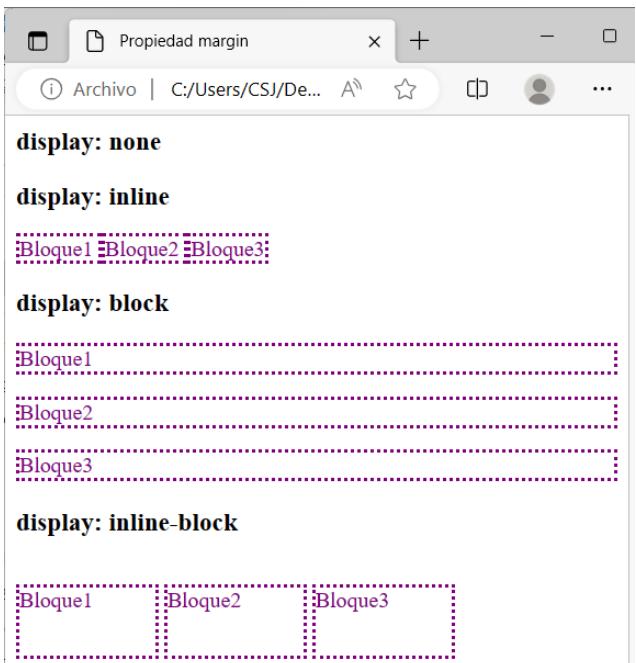
Propiedad	Descripción	Valores
display	Comportamiento del contenedor	inline block inline-block none
position	Esquema de posicionamiento	static relative absolute fixed sticky
Top right bottom left	Desplazamiento de la caja respecto al borde superior, derecho, inferior o izquierdo	longitud porcentaje auto
float	Posicionamiento flotante	left right none
clear	Control de cajas adyacentes a las float	none left right both
z-index	Nivel de la capa	auto número entero
box-sizing	Control de bordes y relleno en el comportamiento del contenedor	content-box border-box
visibility	Muestra u oculta un elemento ocupando el espacio	visible hidden

1. Display

Valores: **none | inline | block | inline-block**

- **none**: los elementos se ocultan y no se muestra el espacio reservado.
- **Inline**: los elementos se muestran en la misma línea (respetando el flujo) y no se aceptan las propiedades width, height ni márgenes verticales.
- **Block**: los elementos se muestran en líneas independientes y se aceptan las propiedades width, height y márgenes verticales.
- **Inline-block**: su comportamiento es una mezcla entre los dos anteriores, los elementos se muestran en la misma línea (respetando el flujo) y se aceptan las propiedades width, height y márgenes verticales.

Ejemplo display

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad margin</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>display: none</h3> <p class="a">Bloque1 </p> <p class="a">Bloque2 </p> <p class="a">Bloque3 </p> <h3>display: inline</h3> <p class="b">Bloque1 </p> <p class="b">Bloque2 </p> <p class="b">Bloque3</p> <h3>display: block</h3> <p class="c">Bloque1 </p> <p class="c">Bloque2 </p> <p class="c">Bloque3 </p> <h3>display: inline-block</h3> <p class="d">Bloque1 </p> <p class="d">Bloque2 </p> <p class="d">Bloque3 </p> </body> </html></pre>	<pre>.a { display: none; } .b { display: inline; width: 100px; height: 50px; } .c { display: block; } .d { display: inline-block; width: 100px; height: 50px; } p { color: purple; border: dotted; }</pre> 

2. Position

Valores: **static | relative | absolute | fixed | sticky**

- **static**: los elementos se posicionan de acuerdo al flujo normal de la página. Es la **posición natural de los elementos**. No son afectados por las propiedades top, bottom, left y right.
- **Relative**: los elementos se posicionan de forma relativa a su posición normal.
- **Fixed**: los elementos se posicionan de forma relativa a la ventana del navegador. Su posición

permanece fija aunque se desplace la ventana.

- **Absolute:** los elementos se posicionan de forma relativa al primer elemento padre que tenga una posición distinta a static.
- **sticky:** los elementos son posicionados de forma relativa hasta que su bloque contenedor alcanza un límite establecido.

Ejemplo position: static

Las propiedades top, bottom, left y right no están permitidas con «position: static».

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad position static</title> <link rel="stylesheet" href="style.css"> </head> <body> <h3>Propiedad position: static</h3> <p class="a">Bloque1 </p> <p class="b">Bloque2 </p> <p class="c">Bloque3 </p> </body> </html></pre>	<p>.a { position: static; top: 800px; left: 400px;} /*Como puedes ver, en position static no funcionan las propiedades top, bottom, left y right*/ .b { position: static; } .c { position: static; } p { width: 100px; color: purple; border: dotted;}</p> <div style="border: 1px solid black; padding: 10px; text-align: center;"><p>Propiedad position: static</p><p>Bloque1</p><p>Bloque2</p><p>Bloque3</p></div>

Ejemplo position: relative

Aplica el siguiente estilo sobre un elemento y comprueba que con las propiedades left: 20px y top: 10px el elemento se desplaza 20px hacia la derecha y 10 px hacia abajo desde su posición por defecto (sin eliminar el hueco de su posición por defecto).

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>Propiedad position: relative</h3> <p>Bloque1 </p> <p class="bloque2">Bloque2 </p> <p>Bloque3 </p> </body> </html></pre>	<p>.bloque2 { position: relative; left: 20px; top: 10px; } p { width: 100px; color: purple; border: dotted;}</p> <div style="border: 1px solid black; padding: 10px; text-align: center;"><p>Propiedad position: relative</p><p>Bloque1</p><p>Bloque2</p><p>Bloque3</p></div>

Ejemplo position: fixed

Los elementos con position:fixed toman como referencia la ventana del navegador y permanecen fijos.

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <h3>Propiedad position: fixed</h3> <p>Bloque1 </p> <p class="bloque2">Bloque2 </p> <p>Bloque3 </p> </body> </html></pre>	<pre>.bloque2 { position: fixed; top: 130px; left: 100px; } p { width: 100px; color: purple; border: dotted; }</pre> <div style="border: 1px solid black; padding: 10px; text-align: center;"><p>Propiedad position: fixed</p><p>Bloque1</p><p>Bloque3</p><p>Bloque2</p></div>

Ejemplo position: absolute

Establece el siguiente estilo sobre dos contenedores y comprueba cómo el elemento «.b» deja de seguir la posición del flujo normal de la página, sin crearse espacio alguno para el elemento, y se posiciona de forma relativa al primer elemento padre que tiene una posición distinta a static, en este caso el elemento «.a». **Si no hubiese ningún elemento padre con propiedad distinta a static, se ubicaría de forma relativa al contenedor inicial.** Haz la prueba cambiando el valor de position a «static» en el elemento a. Como puedes ver, su posición final está definida por los valores de top, right, bottom, y left.

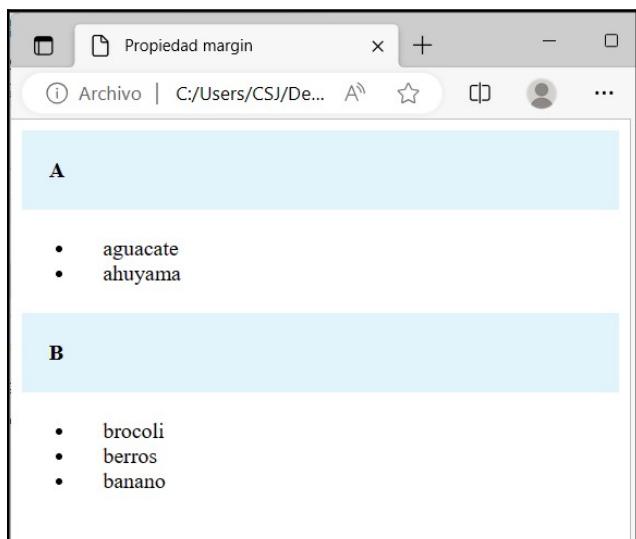
	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="a"> <p>Contenedor con position: relative</p> <div class="b"><p>Contenedor con position: absolute</p></div> </div> </body> </html></pre>	<pre>.a { position: relative; /*cambia este valor a static para ver la diferencia*/ width: 300px; height: 300px; border: 2px dotted purple; } .b { position: absolute; background-color: #e1f4fc; top: 60px; right: 0px; width: 200px; height: 80px; border: 3px solid blue; }</pre>

Contenedor con position: relative
 Contenedor con position: relative
 Contenedor con position: absolute
 Contenedor con position: relative

Ejemplo position: sticky

La posición **sticky** se usa cuando queremos que un elemento tenga una posición relativa hasta un punto y que luego cambie a una posición fija, usando solo CSS sin necesidad de código JavaScript. Por ejemplo si tenemos un banner de publicidad flotante en el sidebar y nos interesa que una vez aparezca al hacer scroll se mantenga fijo y visible.

Otro ejemplo puede ser el que se muestra a continuación y que nos sirve para posicionar los encabezados en una lista alfabética.

	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <p> A </p> aguacate ahuyama <p> B </p> brocoli berros banano </pre>	<pre>p { position: -webkit-sticky; position: sticky; top: 0px; background-color: #e1f4fc; padding: 20px; font-weight: bold; } ul{ margin: 20px 0; } li{ padding-left: 20px; }</pre> 

```
</li>
</ul>
</body>
</html>
```

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.**

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

```
css {
  .contenedor {
    position: relative;
  }

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

- Cuando desconocemos la altura del elemento.**

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

Centrado horizontal de elementos

Cuando un elemento tiene los valores de margin-left y margin-right configurados en auto, su comportamiento varía según el tipo de visualización y el contexto en el que se encuentre.

1. Elementos de bloque (display:block) dentro de un contenedor con ancho definido

Cuando un elemento de bloque (como un div) tiene un ancho (width) especificado y sus márgenes izquierdo (margin-left) y derecho (margin-right) se establecen en auto, el navegador distribuye el espacio sobrante equitativamente entre los márgenes izquierdo y derecho. Esto hace que el elemento **se centre horizontalmente** dentro de su contenedor.

Ejemplo:

```
html
```

Copiar código

```
<div class="centrado">Este es un elemento centrado</div>
```

css

Copiar código

```
.centrado {  
    width: 300px;  
    margin-left: auto;  
    margin-right: auto;  
    background-color: lightblue;  
}
```

Explicación:

- Si el ancho del contenedor padre es mayor que el ancho del elemento (300px en este caso), el espacio restante se divide igualmente entre los márgenes izquierdo y derecho, lo que centra el elemento dentro de su contenedor.

2. Elementos de bloque sin un ancho específico (width:auto o no definido)

Si no se especifica un ancho para el elemento (o su width es auto), el elemento ocupará todo el ancho disponible dentro de su contenedor. En este caso, los márgenes auto no tienen efecto visible porque no queda espacio sobrante para distribuir.

Ejemplo:

html

Copiar código

```
<div class="full-width">Este elemento ocupa todo el ancho</div>
```

css

Copiar código

```
.full-width {  
    margin-left: auto;  
    margin-right: auto;  
    background-color: lightgreen;  
}
```

3. Elementos en línea (display: inline o inline-block)

Los márgenes en elementos en línea (inline) o en línea bloque (inline-block) no se comportan de la misma manera. En este caso, los valores margin-left: auto y margin-right: auto **no centran el elemento**, porque estos elementos solo ocupan el espacio necesario para su contenido, y los márgenes auto no redistribuyen el espacio horizontalmente de la misma manera que en los elementos de bloque.

Para centrar un elemento inline-block, **es mejor usar un contenedor con text-align: center**.

Resumen:

- **Elementos de bloque con width definido:** Los márgenes auto en los lados izquierdo y derecho centran el elemento horizontalmente.
- **Elementos de bloque sin width definido:** Ocuparán todo el ancho del contenedor, y los márgenes auto no tendrán efecto visible.
- **Elementos en línea o inline-block:** Los márgenes auto no funcionan de la misma manera; En este caso, debes usar otra técnica (como text-align: center en el contenedor) para centrar el elemento.

Tipos de Layout

1. Fixed

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.

2. Elastic

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.

Sin embargo, hay una **desventaja importante**, 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.

3. Fluid

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 puede ser muy estrechas.
- En columnas estrechas los textos largos provocan celdas muy alargadas.
- Si tengo imágenes o vídeos con un tamaño fijo tendrá problemas

4. Con Min/Máx Sizing (Híbrido)

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.

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.

5. Responsive

Son aquellos layout 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.

3. Float

Valores: **left | right | none**

Cuando a un elemento HTML se le aplica un estilo con la propiedad float, el elemento sale del flujo normal y aparece posicionado a la izquierda o a la derecha de su contenedor, donde el resto de elementos de la página se posicionarán alrededor.

Ejemplo float

Aplica los siguientes estilos sobre diferentes contenedores y comprueba el resultado.

	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="a"> <div></div> </div> <div class="a"> <div></div> </div> <div class="b"> <div></div> </div> <div class="b"> <div></div> </div> <div class="c"> <div></div> </div> <div class="d"></pre>	<pre>.a { float: left; padding: 10px; } .a div{ height: 100px; width: 100px; background-color: #2980B9; } .b { float: right; padding: 10px; } .b div{ height: 100px; width: 100px; background-color: #17A589; } .c { float: right; padding: 10px; } .c div{ height: 100px; width: 100px; background-color: red; } .d {</pre>

```

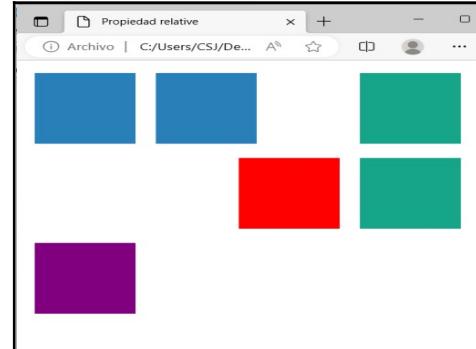
<div></div>
</div>
</body>
</html>

```

```

float: none;
clear: both;
padding: 10px;
}
.d div{
height: 100px;
width: 100px;
background-color: purple;
}

```



4. Clear

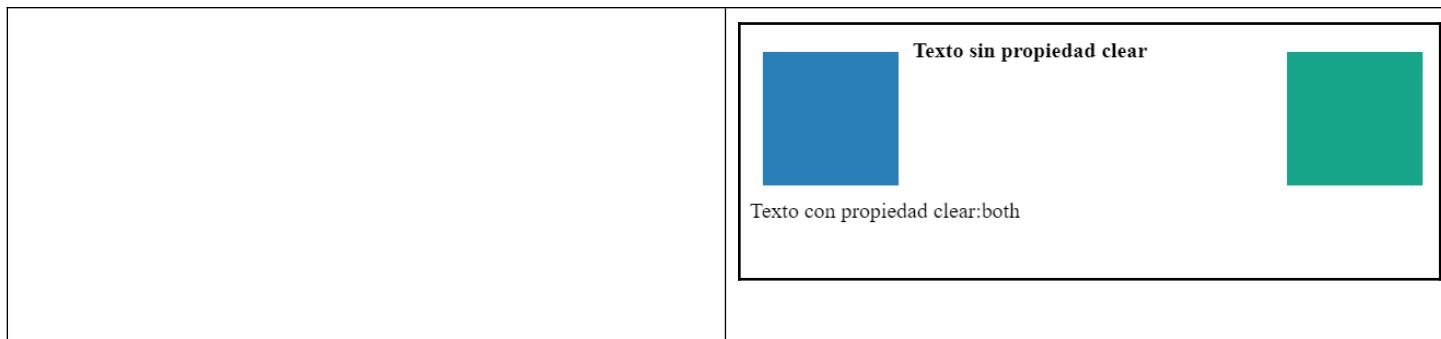
Valores: **none | left | right | both**

La propiedad clear establece si un elemento debe estar al lado de los elementos flotantes que lo preceden o si debe situarse bajo de ellos. Se suele utilizar para restaurar el flujo normal del documento y así los elementos dejan de flotar hacia la izquierda, la derecha o ambos lados.

Ejemplo clear

Crea un contenedor y aplícale la propiedad float con el valor left. Sitúa un texto bajo del contenedor creado utilizando la propiedad clear.

HTML	CSS
<pre> <html> <head> <meta charset="utf-8"> <title>Propiedad relative</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="a"> <div></div> </div> <div class="b"> <div></div> </div> <h4>Texto sin propiedad clear</h4> <p>Texto con propiedad clear:both</p> </body> </html> </pre>	<pre> .a { float: left; padding: 10px; } .a div{ height: 100px; width: 100px; background-color: #2980B9; /*Azul*/ } .b { float: right; padding: 10px; } .b div{ height: 100px; width: 100px; background-color: #17A589; } p{ clear: both; } </pre>



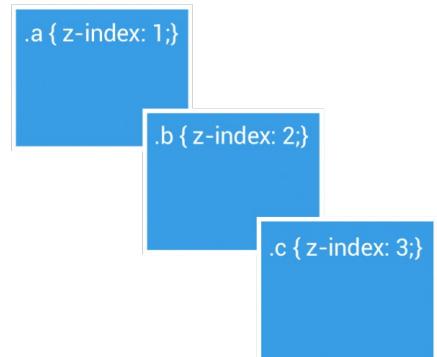
5. Z-index

Valores: **auto | número entero**

Mediante el atributo z-index podemos organizar cada uno de los elementos del contenido de una página web.

Ejemplo z-index

Aplica los siguientes estilos sobre diferentes contenedores que se encuentren superpuestos y comprueba el resultado.



HTML	CSS
<pre> <html> <head> <meta charset="utf-8"> <title>Propiedad z-index</title> <link rel="stylesheet" href="style.css"> </head> <body> <h3>Propiedad z-index</h3> <div class="a"></div> <div class="b"></div> <div class="c"></div> </body> </html></pre>	<pre> .a { width: 150px; height: 200px; background-color: purple; position: relative; z-index: 3; } .b { width: 150px; height: 200px; background-color: black; position: relative; left: 50px; top: -110px; z-index: 1; } .c { width: 150px; height: 200px; background-color: grey; position: relative; left: 100px; top: -210px; z-index: 2; }</pre>

6. Box-sizing

Por defecto en el modelo de cajas de CSS, el ancho y alto asignado a un elemento es aplicado solo al contenido de la caja del elemento. Si el elemento tiene algún borde (border) o relleno (padding), este es entonces añadido al ancho y alto del tamaño de la caja o contenedor. Esto significa que **cuando se define el ancho y alto, se tiene que ajustar el valor para permitir cualquier borde o relleno que se pueda añadir.**

Valores: **content-box | border-box**

- **content-box** es el comportamiento CSS por defecto para el tamaño de la caja (box-sizing). Si se define el ancho de un elemento en 100 píxeles, la caja del contenido del elemento tendrá 100 píxeles de ancho, y el ancho de cualquier borde o relleno será añadido al ancho final desplegado.
- **Border-box** toma en cuenta cualquier valor que se especifique de borde o de relleno para el ancho o alto de un elemento. Es decir, si se define un elemento con un ancho de 100 píxeles. Esos 100 píxeles incluirán cualquier borde o relleno que se añada, y la caja de contenido se encogerá para absorber ese ancho extra. Esta propiedad es especialmente útil para redimensionar cualquier elemento.

Ejemplo box-sizing

Crea un contenedor que ocupe el 100% del ancho de la pantalla. Posiciona tres imágenes en línea y define que cada imagen ocupe el 33,333%. Observa que el conjunto ocupa el 100% de la pantalla.

Si a continuación dotamos a las imágenes de un padding o relleno, el conjunto ocupará más del 100%. En este punto podríamos establecer un “box-sizing: border-box” para incluir en el conjunto el relleno definido. Acuérdate de añadir los prefijos para los navegadores:

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad z-index</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div class="img-container"> </div> <div class="img-container"> </div> <div class="img-container"> </div> </body> </html></pre>	<pre>.img-container { box-sizing: border-box; -webkit-box-sizing: border-box; float: left; width: 33.33%; padding: 5px; } .img-container img{ width: 100%; }</pre> 

7. Visibility

La propiedad visibility indica si un elemento es visible o permanece oculto (ocupando el mismo espacio).

Valores: **visible | hidden**

Ejemplo visibility

Aplica las siguientes propiedades sobre un elemento y observa las diferencias. Como puedes ver, la diferencia principal es que display: none **no reserva el espacio del elemento mientras que** visibility: hidden **sí**.

HTML	CSS
<pre><html> <head> <meta charset="utf-8"> <title>Propiedad visibility</title> <link rel="stylesheet" href="styles.css"> </head> <body> <div> <p>Caja2. display:none</p> <p>Caja7. visibility:hidden</p> <div class="cajaFlotanteDcha"> Caja1 </div> <div class="cajaFlotanteDcha" style="display:none"> Caja2 </div> <div class="cajaFlotanteDcha"> Caja3 </div> <div class="cajaFlotanteDcha"> Caja4 </div> <div class="cajaFlotanteDcha"> Caja5 </div> <div class="cajaFlotanteDcha"> Caja6 </div> <div class="cajaFlotanteDcha" style="visibility:hidden"> Caja7 </div> <div class="cajaFlotanteDcha"> Caja8 </div> </div> </body> </html></pre> <div style="background-color: #f0f0f0; padding: 10px;"><p>Caja2. display:none Caja7. visibility:hidden</p><p>Caja1 Caja3 Caja4 Caja5</p><p>Caja6 Caja8</p></div>	<pre>div { position: relative; width: 250px; height: 180px; border: 1px solid black; margin: 2px; box-sizing: border-box; padding: 2px; background-color: rgba(255, 0, 0, 0.2); } .caJaInLine { height: 20px; display: inline; } .caJaBlock { width: 142px; height: 20px; margin-top: 8px; } .caJaRel { position: relative; top: 10px; left: 5px; width: 50px; height: 20px; display: inline; } .caJaAbs { position: absolute; top: 80px; left: 100px; width: 50px; height: 20px; display: inline; } .caJaFija { position: fixed; top: 250px; left: 100px; width: 50px; height: 20px; display: inline; } .caJaFlotanteDcha{ float: right; width: 50px; height: 20px; } .caixaFlotanteDcha{</pre>

```
width: 50px;  
height: 20px; }  
  
body {  
font-family: arial;  
font-size: 0.8em; }  
p {  
margin: 2px; }
```