

Codo a Codo 4.0

FULL STACK PYTHON

html - css3 - bootstrap - javascript
vue.js - sql - python - django

CSS

Parte 3

CSS



Especificidad

La especificidad hace referencia a la **relevancia** que tiene un estilo sobre un elemento de la página al cual le están afectando varios estilos de CSS al mismo tiempo. Es decir, hace referencia al grado de importancia de un estilo sobre otro.

Los navegadores deciden qué valores de una propiedad CSS son más relevantes para un elemento y, por lo tanto, serán aplicados. La especificidad está basada en las reglas de coincidencia que están compuestas por diferentes tipos de selectores CSS.

La siguiente tabla muestra los niveles de especificidad, desde los más específicos a los más generales:

!important	cualquier-selector { color: #FF0000!important; }	1 , 0, 0, 0, 0
Estilos inline	<p style="color:#FF0000;">Lorem Ipsum</p>	0, 1 , 0, 0, 0
ID	#parrafo { color: #FF0000; }	0, 0, 1 , 0, 0
Clases, atributos y pseudoclases	.parrafo { color: #FF0000; }	0, 0, 0, 1 , 0
Etiquetas y pseudoelementos	p { color: #FF0000; }	0, 0, 0, 0, 1

Más información: <https://developer.mozilla.org/es/docs/Web/CSS/Specificity>
https://www.w3schools.com/css/css_specificity.asp

Orden de aplicación:

1. Estilos marcados como **!important**; 2. Estilos **en línea**; 3. Selectores de **ID**; 4. Selectores de **clase y pseudoclases**; 5. Selectores de **etiquetas y pseudolementos**

```
# especificidad.css X
07- CSS > ejemplos-clase-7 > # especificidad.css > ...
1 p {
2   color: red !important; /* va a sobrescribir a
3     todos */
4 }
5
6 #primer-parrafo {
7   color: blue; /* va a sobrescribir solo a las
8     clases */
9 }
10 /* pseudoclases y etiquetas */
11
12 .parrafo {
13   color: green; /* va a sobrescribir solo a las
14     etiquetas */
15 }
16 /* y pseudoelementos */
17
18 p {
19   color: orange;
20 }
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

especificidad.html X
ejemplos-clase-7 > especificidad.html > html > body > p#primer-parrafo.parrafo
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport"
6     content="width=device-width, initial-scale=1.0">
7   <title>Especificidad</title>
8   <link rel="stylesheet" href="especificidad.css">
9 </head>
10 <body>
11   <p id="primer-parrafo" class="parrafo"
12     style="color: pink;">
13     Lorem ipsum dolor, sit amet consectetur
14     adipiscing elit. Dolor aliquid sed eaque
15     quam tempore aspernatur ea, voluptates sit
16     minus nostrum. Alias, repellendus pariatur
17     voluptatem hic nihil quaerat numquam quidem
18     assumenda.
19   </p>
20 </body>
21 </html>
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

IMPORTANTE

Cuando varias declaraciones tienen igual especificidad, se aplicará al elemento la **última declaración** encontrada en el CSS



Ver ejemplo especificidad (.html y .css). Comentar las reglas de estilo para ver cómo funciona



Ejemplo explicado:

1. Este es el resultado inicial

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Dolor aliquid

... si comentamos esta línea:

```
color:red!important;
```

2. Toma el estilo en línea como siguiente nivel de jerarquía:

```
<p id="primer-parrafo" class="parrafo" style="color:pink">
```

Lorem ipsum dolor,

3. Si sacamos el estilo en línea tendrá jerarquía el ID (#primer-parrafo):

```
<p id="primer-parrafo" class="parrafo">
```

Lorem ipsum dolor, sit amet

4. Si le sacamos el estilo de ID pasará a tener jerarquía el estilo de clase:

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Dolor aliquid

5. Y si finalmente le sacamos el estilo de clase y dejamos sólo el párrafo tendrá jerarquía el estilo de etiqueta:

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Dolo

Selectores

- **selector descendiente:** Se aplican en los elementos que tienen una relación padre-hijo, es decir las etiquetas que están dentro de otras etiquetas.

```
<div>
  <p> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. </p>
  <p> Lorem ipsum dolor
    tempor incididunt
  <p> Lorem ipsum dolor
    tempor incididunt
</div>
```

HTML

```
div p {
  color: green;
  font-size: 20px;
  font-weight: bold;
  background-color: lightgrey;
}
```

CSS

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. </p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. </p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. </p>

En este ejemplo es más fácil agregar un selector descendiente que aplicar una clase a cada elemento <p>



[Ver ejemplo selector-descendiente \(.html y .css\)](#)

- **selector de hijos:** Si no queremos seleccionar todos los elementos descendientes pero si a los hijos directos podemos utilizar el símbolo >.

```
<p>  
    Esto es <strong> muy </strong> importante.  
</p>  
<p>  
    Esto es <i> realmente <strong> muy </strong> </i> importante.  
</p>
```

HTML

```
p > strong {  
    color: orange;  
}
```

CSS

Esto es **muy** importante.

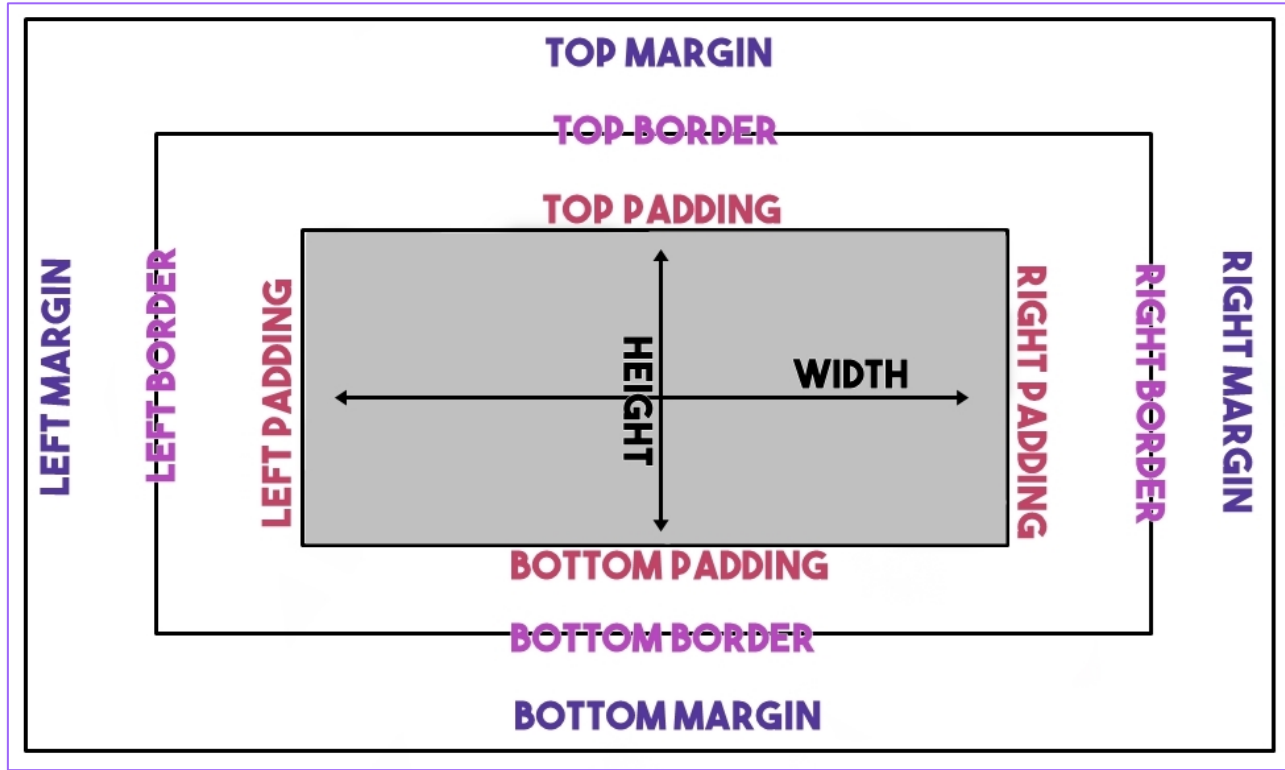
Esto es *realmente* **muy** importante.

¿Por qué no aplica la regla al segundo strong?

- **otros selectores:**

https://www.w3schools.com/cssref/css_selectors.asp

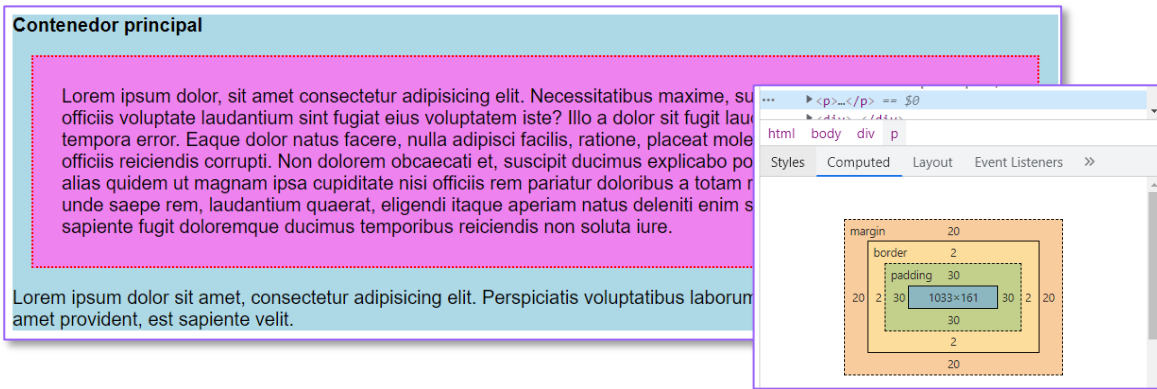
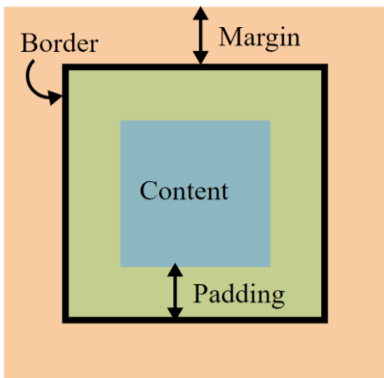
Modelo de caja



Modelo de caja

La representación básica del **modelo de cajas** es la siguiente, donde podemos observar varios conceptos importantes a diferenciar:

- El **borde** (*border*). En negro, es el límite que separa el interior del exterior del elemento.
- El **margen** (*margin*). En naranja, es la parte exterior del elemento, por fuera del borde.
- El **relleno** (*padding*). En verde, es la parte interior del elemento, entre el contenido y el borde.
- El **contenido** (*content*). En azul, es la parte interior del elemento, excluyendo el relleno.



Para ver el modelo de caja desde el navegador Chrome debemos inspeccionar y seleccionar la pestaña Computed del panel inferior.



[Ver ejemplo modelo-caja.html](#)

Dimensiones (ancho y alto)

Para dar tamaños específicos a los diferentes elementos de un documento HTML, necesitaremos asignarles valores a las propiedades **width** (ancho) y **height** (alto).

Propiedad	Valor	Significado
<code>width</code>	<code>auto</code> <small>SIZE</small>	Tamaño de ancho de un elemento.
<code>height</code>	<code>auto</code> <small>SIZE</small>	Tamaño de alto de un elemento.

En el caso de utilizar el valor **auto** en las propiedades anteriores (*que es lo mismo que no indicarlo, ya que es el valor que tienen por defecto*), el navegador se encarga de calcular el ancho o alto necesario, dependiendo del contenido del elemento.

Es importante recalcar que el tamaño automático dado a un elemento depende del **tipo de elemento** (bloque, en línea...).

Dimensiones en modelo de caja

Al no indicar valores de ancho y alto para una caja, el elemento generalmente *toma el tamaño que debe respecto a su contenido*, pero si indicamos un ancho y alto concretos, **estamos obligando mediante CSS a tener un aspecto concreto** y podemos obtener resultados inesperados si su contenido es más grande que el tamaño que hemos obligado a tener:

CSS
IS
AWESOME

Otra forma de lidiar con esto es utilizar las propiedades hermanas de **width** *min-width* y *max-width* y las propiedades hermanas de **height** *min-height* y *max-height*. Con estas propiedades, en lugar de establecer un tamaño fijo, establecemos unos máximos y unos mínimos, donde el ancho o alto podría variar entre estos valores.

```
div{  
  width: 800px;  
  height: 800px;  
  background: red;  
  max-width: 500px;  
}
```

CSS

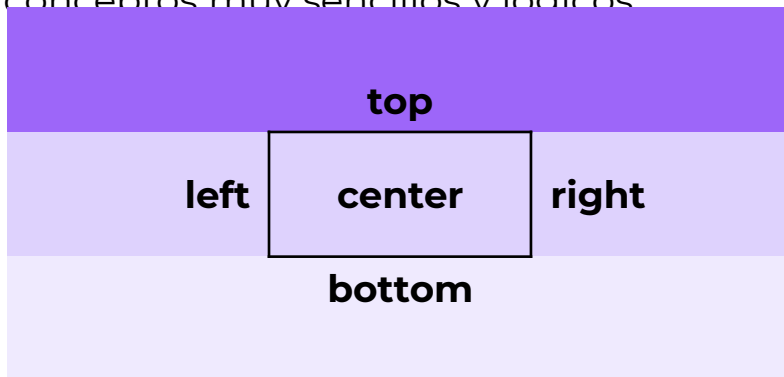
En este caso, por ejemplo, a pesar de estar indicando un tamaño de **800px**, le aplicamos un **max-width** de **500px**, por lo que estamos limitando el elemento a un tamaño de ancho de 500 píxeles como máximo y nunca superará ese tamaño.

Las propiedades de mínimos **min-width** y **min-height** por defecto tienen valor **0**, mientras que las propiedades de máximos **max-width** y **max-height**, tienen por defecto valor **none**.

Más información: [min-height](#), [max-height](#), [min-width](#) y [max-width](#)

Zonas de un elemento

En CSS existen ciertas palabras clave para hacer referencia a una zona u orientación concreta sobre un elemento. Son conceptos muy sencillos y lógicos:



- **Top:** Parte superior
- **Left:** Parte izquierda
- **Right:** Parte derecha
- **Bottom:** Parte inferior
- **Center:** Se refiere a la posición central entre los extremos horizontales y verticales

Desbordamiento

Puede suceder que le demos un tamaño de alto y ancho a un elemento HTML pero su contenido de texto es tan grande que no cabe dentro de ese elemento.

En ese caso lo que ocurriría es que el contenido se desborde, pero podemos modificar este comportamiento con la propiedad de CSS **overflow** o con alguna de sus propiedades específicas **overflow-x** u **overflow-y**

Propiedad	Valor	Significado
overflow	visible hidden scroll auto	Establece el comportamiento de desbordamiento.
overflow-x	visible hidden scroll auto	Establece el desbordamiento sólo para el eje X (<i>horizontal</i>).
overflow-y	visible hidden scroll auto	Establece el desbordamiento sólo para el eje Y (<i>vertical</i>).

https://www.w3schools.com/css/css_overflow.asp

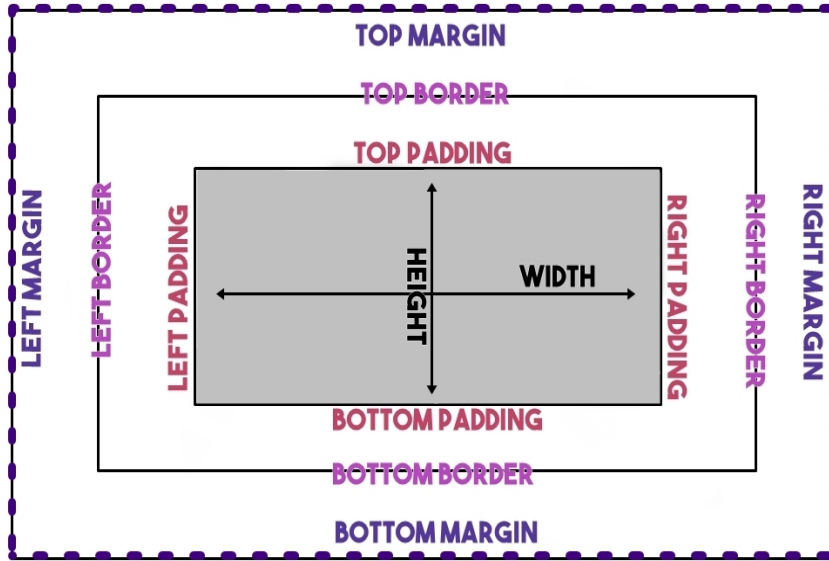
Dichas propiedades pueden tomar varios valores, donde **visible** es el valor que tiene por defecto, le permite que haya desbordamiento. Otras opciones son las siguientes, donde **no se permite desbordamiento**:

Valor	¿Qué ocurre si se desborda el contenedor?	¿Desbordamiento?
visible	Se muestra el contenido que sobresale (<u>comportamiento por defecto</u>)	Sí
hidden	Se oculta el contenido que sobresale.	No
scroll	Se colocan barras de desplazamiento (horizontales y verticales).	No
auto	Se colocan barras de desplazamiento (sólo las necesarias).	No

*CSS3 añade las propiedades **overflow-x** y **overflow-y** para cada eje individual, que antiguamente sólo era posible hacerlo con **overflow** para ambos ejes. Estas propiedades son útiles cuando no queremos mostrar alguna barra de desplazamiento (habitualmente la barra de desplazamiento horizontal).*

Margin

Se utilizan para crear espacio alrededor de los elementos, fuera de los bordes definidos.



- margin-top
 - margin-right
 - margin-bottom
 - margin-left
- auto
 - px, em, rem, etc.
 - porcentaje

En el modelo de cajas, los **márgenes** (*margin*) son los espacios exteriores de un elemento. El espacio que hay entre el borde de un elemento y el borde de otros elementos adyacentes es lo que se considera **margen**.

Se pueden considerar en conjunto (de forma general) o de forma concreta en cada una de las zonas del elemento. Estas son las propiedades específicas de cada zona:

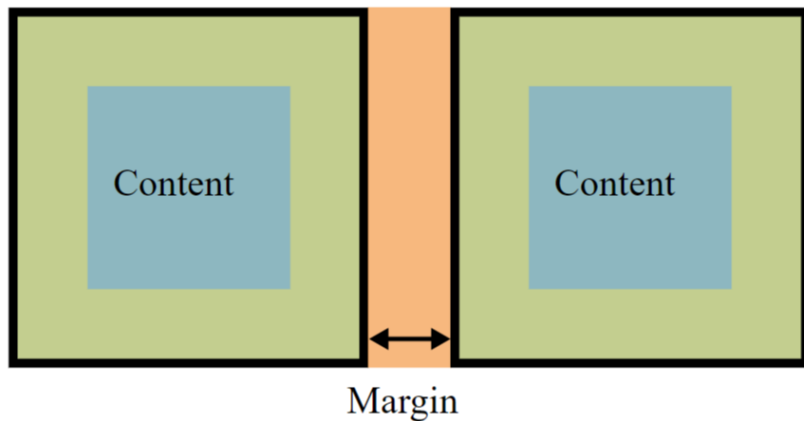
Propiedad	Valor	Significado
<code>margin-top</code>	<code>auto</code> <code>SIZE</code>	Establece un tamaño de margen superior.
<code>margin-left</code>	<code>auto</code> <code>SIZE</code>	Establece un tamaño de margen a la izquierda.
<code>margin-right</code>	<code>auto</code> <code>SIZE</code>	Establece un tamaño de margen a la derecha.
<code>margin-bottom</code>	<code>auto</code> <code>SIZE</code>	Establece un tamaño de margen inferior.

Podemos aplicar diferentes márgenes a cada zona de un elemento utilizando cada una de estas propiedades, o dejando al navegador que lo haga de forma automática indicando el valor **auto**.

*Existe un **truco** muy sencillo y práctico para centrar un elemento en pantalla. Basta con aplicar un ancho fijo al contenedor, **width: 500px** (por ejemplo) y luego aplicar un **margin: auto**. De esta forma, el navegador, al conocer el tamaño del elemento (y, por omisión, el resto del tamaño de la ventana) se encarga de repartirlo equitativamente entre el margen izquierdo y el margen derecho, quedando centrado el elemento*

Hay que recordar diferenciar bien los **márgenes** de los **rellenos**, ya que no son la misma cosa. Los **rellenos** (*padding*) son los espacios que hay entre los bordes del elemento en cuestión y el contenido (*por la parte interior*). Mientras que los **márgenes** (*margin*) son los espacios que hay entre los bordes del elemento en cuestión y los bordes de otros elementos (*parte exterior*).

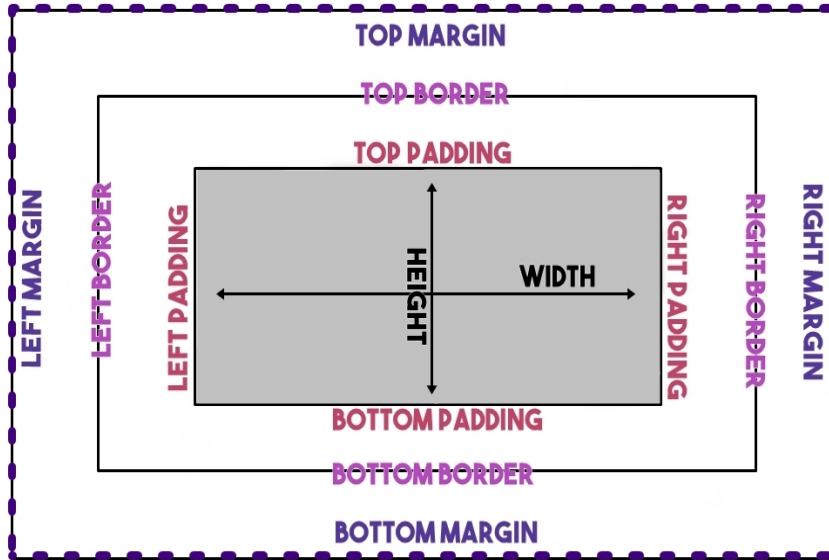
En el siguiente ejemplo nos encontramos con un **solapamiento de márgenes**. Por defecto, si tenemos dos elementos adyacentes con, por ejemplo, **margin: 20px** cada uno, ese espacio de margen se solapará y tendremos **20px** en total, y no **40px** (*la suma de cada uno*) como podríamos pensar en un principio.



https://www.w3schools.com/css/css_margin.asp

Padding

Se utiliza para generar espacio alrededor del contenido de un elemento dentro de los bordes definidos.



- padding-top
 - padding-right
 - padding-bottom
 - padding-left
- px, em, rem, etc.
 - % en relación al ancho del contenedor

En el modelo de cajas, los **rellenos** (*padding*) son los espacios interiores de un elemento. El espacio que hay entre el borde de un elemento y su contenido es lo que se considera **relleno**.

Al igual que con los márgenes, los rellenos tienen varias propiedades para indicar cada zona:

Propiedad	Valor	Significado
<code>padding-top</code>	0 <small>SIZE</small>	Aplica un relleno interior en el espacio superior de un elemento.
<code>padding-left</code>	0 <small>SIZE</small>	Aplica un relleno interior en el espacio izquierdo de un elemento.
<code>padding-right</code>	0 <small>SIZE</small>	Aplica un relleno interior en el espacio derecho de un elemento.
<code>padding-bottom</code>	0 <small>SIZE</small>	Aplica un relleno interior en el espacio inferior de un elemento.

Como se puede ver en la tabla, por defecto no hay relleno (*el relleno está en cero*), aunque puede modificarse tanto con las propiedades anteriores como la propiedad de atajo: modelo de cajas (en la siguiente página)

https://www.w3schools.com/css/css_padding.asp

Atajo: Modelo de cajas

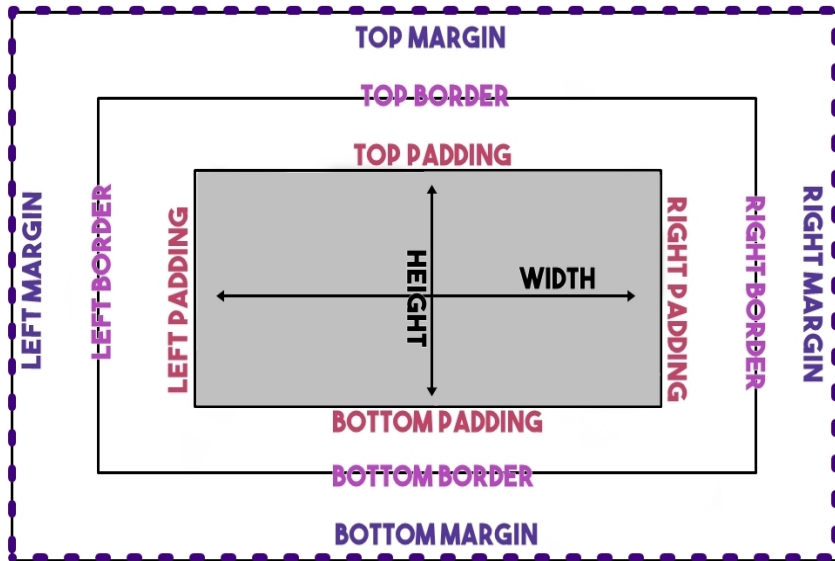
Al igual que en otras propiedades de CSS, también existen atajos para los márgenes y los rellenos:

Propiedad	Valores	Significado
<code>margin</code> o <code>padding</code> 1234	SIZE	1 parámetro. Aplica el mismo margen a todos los lados.
	SIZE SIZE	2 parámetros. Aplica margen top/bottom y left/right .
	SIZE SIZE SIZE	3 parámetros. Aplica margen top , left/right y bottom .
	SIZE SIZE SIZE SIZE	4 parámetros. Aplica margen top , right , bottom e left .

margin o padding:	10px; top/right/bottom/left			
	top/bottom 10px	right/left 20px;		
	top 10px	right/left 20px	bottom 10px;	
	top 10px	right 20px	bottom 10px	left 20px;

Border

Permiten especificar el estilo, el ancho y el color del borde de un elemento.



- border-top
- border-right
- border-bottom
- border-left
- border-color
- border-style
- border-width

En CSS es posible especificar el aspecto que tendrán los **bordes** de cualquier elemento HTML, pudiendo incluso, dar diferentes características a zonas particulares del borde, como por ejemplo, el borde superior, el borde izquierdo, el borde derecho o el borde inferior.

Las propiedades básicas y específicas de los bordes en CSS son las siguientes:

Propiedad	Valor	Significado
<code>border-color</code> 1234	<code>black</code> COLOR	Especifica el color que se utilizará en el borde.
<code>border-width</code> 1234	<code>thin</code> <code>medium</code> <code>thick</code> SIZE	Especifica un tamaño predefinido para el grosor del borde.
<code>border-style</code> 1234	<code>none</code> STYLE	Define el estilo para el borde a utilizar

Estilos de borde

Valor	Descripción
<code>hidden</code>	Oculto. Idéntico a none , salvo para conflictos con tablas.
<code>dotted</code>	Borde basado en puntos.
<code>dashed</code>	Borde basado en rayas (línea discontinua).
<code>solid</code>	Borde sólido (línea continua).
<code>double</code>	Borde doble (dos líneas continuas).

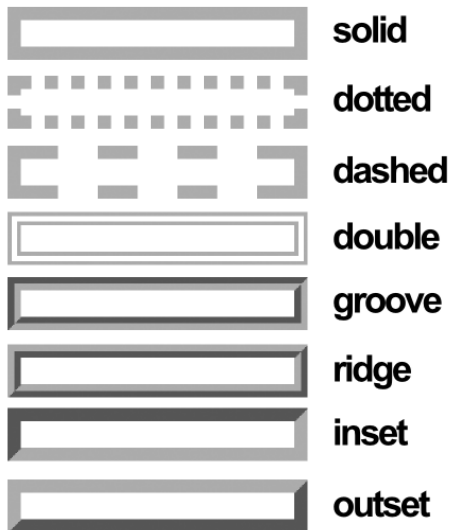
Valor	Descripción
<code>groove</code>	Borde biselado con luz desde arriba.
<code>ridge</code>	Borde biselado con luz desde abajo. Opuesto a groove .
<code>inset</code>	Borde con profundidad «hacia dentro».
<code>outset</code>	Borde con profundidad «hacia fuera». Opuesto a inset .



```
div{  
  border-color: gray;  
  border-width: 1px;  
  border-style: dotted;  
}
```

CSS

El borde más frecuente suele ser **solid** (borde liso y continuo). Pueden utilizarse cualquiera de los estilos indicados en la tabla anterior e incluso combinar con otras propiedades. Así se verían los diferentes estilos de borde utilizando **10 píxeles** de grosor y color **gris**:



https://www.w3schools.com/css/css_border.asp

Bordes redondeados:

https://www.w3schools.com/css/css3_borders.asp

<https://lenguajecss.com/css/modelo-de-cajas/border-radius/>

Bordes múltiples (diferentes)

Hasta ahora, sólo hemos utilizado un parámetro en cada propiedad, lo que significa que se aplica el mismo valor para cada borde de un elemento (*borde superior*, *borde derecho*, *borde inferior* y *borde izquierdo*). Sin embargo, podemos especificar uno, dos, tres o cuatro parámetros, dependiendo de lo que queramos hacer:

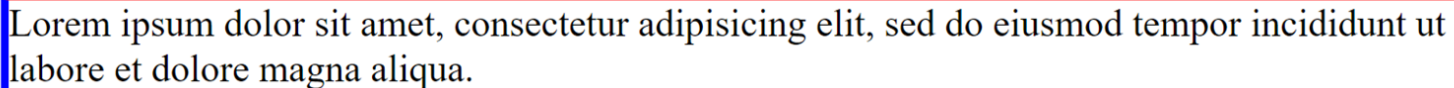
Propiedad	Valor	Significado
<code>border-color</code> <small>1234</small>	<code>COLOR</code>	1 parámetro. Aplica el mismo color a todos los bordes.
	<code>COLOR</code> <code>COLOR</code>	2 parámetros. Aplica al borde top/bottom , y al left/right .
	<code>COLOR</code> <code>COLOR</code> <code>COLOR</code>	3 parámetros. Aplica al top , al left/right y al bottom .
	<code>COLOR</code> <code>COLOR</code> <code>COLOR</code> <code>COLOR</code>	4 parámetros. Aplica al top , right , bottom y left .

De la misma forma, podemos hacer exactamente lo mismo con las propiedades **border-width** (respecto al ancho del borde) y **border-style** (respecto al estilo del borde). Teniendo en cuenta esto, disponemos de mucha flexibilidad a la hora de especificar esquemas de bordes más complejos.


```
div {  
  border-color: red blue green;  
  border-width: 2px 10px 5px;  
  border-style: solid double solid;  
}
```

CSS

En el ejemplo anterior hemos utilizado **3 parámetros**, indicando un elemento con borde superior rojo sólido de 2 píxeles de grosor, con borde izquierdo y derecho doble azul de 10 píxeles de grosor y con un borde inferior verde sólido de 5 píxeles de grosor.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

En relación al modelo de cajas, con la propiedad border-width pasa exactamente lo mismo que con margin y padding, actuando en este caso en relación al grosor del borde de un elemento (puedo determinar entre 1 y 4 parámetros).

Atajo: Bordes

Con tantas propiedades, para hacer algo relativamente sencillo nos pueden quedar varias líneas de código complejas y difíciles de leer. Al igual que con otras propiedades CSS, podemos utilizar la propiedad de atajo **border**, con la que podemos hacer un resumen y no necesitar indicar múltiples propiedades individuales por separado, realizando el proceso de forma más corta:

Propiedad	Valor	Significado
border	SIZE STYLE COLOR	Propiedad de atajo para simplificar valores.

```
div {  
  border: 1px solid #000000;  
}
```

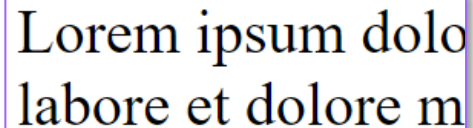
CSS

Lorem ipsum dolor sit amet, con
labore et dolore magna aliqua.

Bordes específicos

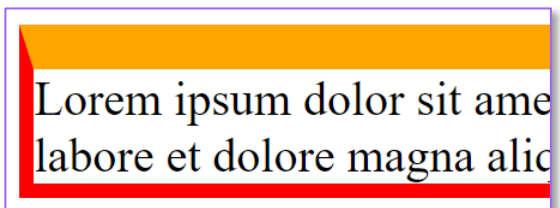
Otra forma, quizás más intuitiva, es la de utilizar las propiedades de bordes específicos (*por zonas*) y aplicar estilos combinándolos junto a la *herencia* de CSS. Para utilizarlas bastaría con indicarle la zona justo después de **border-**:

Esto dibujaría sólo un borde inferior negro de 2 píxeles de grosor y con estilo punteado:



Lorem ipsum dolor
labore et dolore m

Ahora imaginemos que queremos un elemento con todos los bordes en rojo a 5 píxeles de grosor, salvo el borde superior, con un borde de 15 píxeles en color naranja:



Lorem ipsum dolor sit ame
labore et dolore magna alic

```
div {  
  border-bottom-width: 2px;  
  border-bottom-style: dotted;  
  border-bottom-color: black;  
}
```

CSS

```
div {  
  border: 5px solid red;  
  border-top-width: 15px;  
  border-top-color: orange;  
  border-top-style: solid;  
  /* Esta última propiedad no es  
     necesaria (se hereda) */  
}
```

CSS

Info sobre herencia: <https://lenguajecss.com/css/introduccion/herencia-css/>

Box-sizing

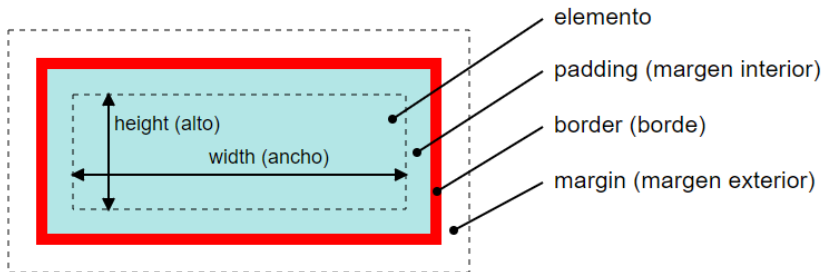
Indica cómo se debe calcular el ancho y el alto total de un elemento. Esta propiedad ayuda a crear diseños de cajas más fácil y mucho más intuitivos. Acepta los valores:

- **box-sizing: content-box:** Es el valor que cualquier caja tiene asignada **por defecto**. Las propiedades width y height no incluyen el borde, padding o margin.
- **box-sizing: border-box:** Las propiedades width y height incluyen el contenido, padding y borde pero no el margin.

Cambio del modelo de caja: box-sizing

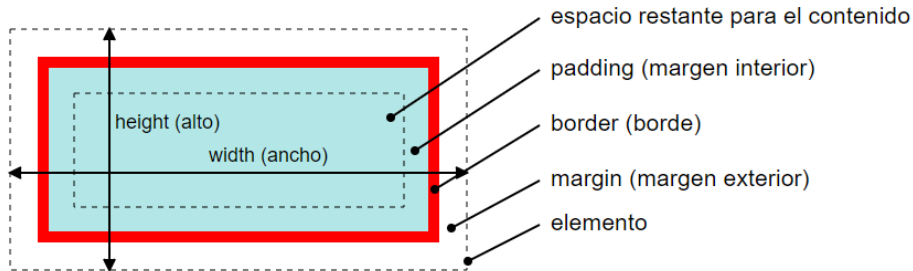
En el modelo de caja CSS "clásico", el borde y los márgenes interior y exterior se añaden al tamaño del elemento definido con las propiedades **width** y **height**.

Modelo de caja CSS "clásico" (box-sizing: border-box)



La propiedad **box-sizing**, introducida en la recomendación CSS Basic User Interface Module Level 3 (CSS3 UI) (aprobada en junio de 2018), permite modificar este comportamiento y hacer que el borde y los márgenes interior y exterior se puedan incluir en el interior del tamaño definido con las propiedades **width** y **height**. En este caso, lógicamente, se reducirá el espacio disponible para el contenido.

Modelo de caja CSS "alternativo" (**box-sizing: content-box**)



https://www.w3schools.com/css/css3_box-sizing.asp

Más info: <https://www.mclibre.org/consultar/amaya/css/css-modelo-caja.html>



[Ver ejemplo box-sizing.html](#)

Medidas

Las medidas en CSS se emplean para definir la altura, el ancho, los márgenes de los elementos y para establecer el tamaño de letra del texto. Todas las medidas se indican como un valor numérico entero o decimal seguido de una unidad de medida (sin ningún espacio en blanco entre el número y la unidad de medida).

CSS divide las unidades de medida en: **absolutas**, **relativas** y **flexibles**.

- **Absolutas:** Las unidades absolutas son medidas fijas, su valor real es directamente el valor indicado que se ve igual en todos los dispositivos. Indican cantidades exactas en alguna unidad de medida. No son relativas a nada, no dependen de otro valor de referencia, por eso se llaman absolutas. La desventaja que tienen es que son muy poco flexibles.
- **Relativas:** Las medidas relativas definen su valor en relación con otra medida, por lo que para obtener su valor real, se debe realizar alguna operación con el valor indicado.
- **Flexibles:** Dentro de las medidas relativas están las flexibles que son relativas al tamaño del viewport.

Medidas absolutas

La principal ventaja de las unidades absolutas es que su valor es directamente el valor que se debe utilizar, sin necesidad de realizar cálculos intermedios. Pero la desventaja es que **son muy poco flexibles y no se adaptan** fácilmente a los diferentes medios y por esto no suelen ser utilizadas. La más utilizada es el pixel (px).

Una medida indicada mediante unidades absolutas está completamente definida, ya que su valor no depende de otro valor de referencia.

- **cm:** centímetros (10 mm).
- **mm:** milímetros.
- **px:** pixeles. Un pixel equivale a unos 0.26 milímetros.
 - celular Moto G4: 360 * 640px - Celular Iphone x: 375 * 812 px
 - Tablet: 1.280 x 800 pixeles
 - Monitores:
 - 1366 x 768 píxeles (16:9) Monitores de 17 y 19"
 - 1920 x 1080 píxeles (16:9) Monitores de 24, 25, 27, 32". Conocido como Full HD.
- **pt:** puntos. Un punto equivale a unos 0.35 milímetros.
- **in:** pulgadas: Una pulgada equivale a 2.54 centímetros (25,4 mm).
- **pc:** picas. Una pica equivale a unos 4.23 milímetros.

El punto (pt) es una medida que puede utilizarse para documentos CSS en los que se fija el tamaño de las fuentes en medios impresos.

Medidas relativas

Las unidades relativas, a diferencia de las absolutas, es que no están completamente definidas, ya que su valor siempre está referenciado respecto a otro valor (*resolución, densidad de pantalla, etc.*). Son las más utilizadas por la flexibilidad con la que se adaptan a los diferentes medios y su potencia.

Unidad	Significado	Medida aproximada
em	<<M>>	1em = tamaño de fuente establecida en navegador.
ex	<<X>> (0,5 em apróx)	1ex = mitad del tamaño de la fuente del navegador aproximadamente.
ch	<<zero width>>	1ch = tamaño de ancho del cero (0).
rem	<<root M>>	1rem = tamaño fuente raíz.
%	Porcentaje	Relativa a herencia (contenedor padre)

La unidad **em** se utiliza para hacer referencia al **tamaño actual de la fuente** que ha sido establecida en el navegador, que habitualmente es un valor aproximado a **16px** (*salvo que se modifique por el usuario*). De esta forma, podemos trabajar simplificando las unidades a medidas en base a ese tamaño.

Por ejemplo, imaginemos que el tamaño de la fuente establecida en el navegador del usuario es exactamente **16px**. Una cantidad **1em** equivaldría a **16px**, mientras que una cantidad de **2em** sería justo el doble: **32px**. Por otro lado, una cantidad de **0.5em** sería justo la mitad: **8px**.



Entonces, em es relativa respecto del tamaño de letra del elemento. Por defecto el tamaño de letra debería ser de 16px que equivaldrían a 1em pero, por ejemplo, si le diéramos un font-size de 10px al body, 1em equivaldría a 10px. **Siempre va a variar dependiendo cual es el tamaño del elemento padre.** 1.2em sería 20% más que el tamaño de su elemento padre.

Existen ciertas unidades menos utilizadas dentro de las unidades relativas, como por ejemplo las unidades **ex** o **ch**. Mientras que la unidad **em** es el tamaño de la fuente establecida por el navegador del usuario, la unidad **ex** es **la mitad del tamaño** de la fuente establecida por el navegador del usuario, por lo que se cumple que **1ex** es igual a **0.5em**.



Realmente, la medida **ex** está basada en la **altura de la x minúscula**, que es aproximadamente un poco más de la mitad de la fuente actual (depende de la tipografía utilizada). La unidad **ch** por su parte, equivale al tamaño de ancho del **o** de la fuente actual, aunque como hemos dicho, en la práctica es un tipo de unidad que no se suele utilizar frecuentemente.

La unidad rem (root em)

Una unidad muy interesante y práctica para tipografías es la unidad **rem** (root em). Esta unidad toma la idea de la unidad em, pero permitiendo establecer un **tamaño base** personalizado (*generalmente para el documento en general, utilizando **html** o la pseudoclase **:root***). De esta forma, podemos trabajar con múltiplos del tamaño base:

```
:root {  
  font-size: 22px; /* Tamaño base */  
}  
h1 {  
  font-size: 2rem; /* El doble del tamaño base: 44px */  
}  
h2 {  
  font-size: 1rem; /* El mismo tamaño base: 22px */  
}
```

Podremos ir utilizando la unidad **rem** en ciertas partes del documento. Con esto, estamos indicando el factor de escala (respecto al tamaño base). En el ejemplo anterior, los elementos **<h1>** tendrán **44 píxels** de tamaño, ya que hemos establecido **2rem**, que significa «el doble que el tamaño base». Por otro lado, los elementos **<h2>** tendrían el mismo tamaño: **22 píxels**.

Esto nos da una ventaja principal considerable: Si queremos cambiar el tamaño del texto en general, sólo tenemos que cambiar el **font-size** de la pseudoclase **:root**, puesto que el resto de unidades son factores de escalado y se modificarán todas en consecuencia al cambio del **:root**.

Medidas flexibles

Las unidades flexibles son todas relativas a las dimensiones tanto del ancho o alto del viewport (región visible de la página Web en el navegador, no el body) en el que se visualice nuestra página, ya sea un dispositivo móvil o de escritorio.

- **vw**: viewport width, esta medida es relativa al 100% del viewport. Lo que quiere decir que si decimos que un div debe medir 50vw, es equivalente al 50% del ancho total del viewport.
- **vh**: viewport height, va a ser un porcentaje relativo a la altura total del viewport. Entonces, si definimos que un div mide 50vh y el alto del viewport es 800px, nuestro div medirá 400px.

Para seguir investigando:

https://www.w3schools.com/css/css_units.asp

<https://lenguajecss.com/css/modelo-de-cajas/unidades-css/>

Posicionamiento



Si tenemos varios **elementos en línea** (*uno detrás de otro*) aparecerán colocados de **izquierda a derecha**, mientras que si son **elementos en bloque** se verán colocados desde **arriba hacia abajo**. Estos elementos se pueden ir combinando y anidando (*incluyendo unos dentro de otros*), construyendo así esquemas más complejos.

Hasta ahora, hemos estado trabajando sin saberlo en lo que se denomina posicionamiento **estático** (*static*), donde todos los elementos aparecen con un orden natural según donde estén colocados en el HTML. Este es el **modo por defecto** en que un navegador renderiza una página.

Sin embargo, existen otros modos alternativos de posicionamiento, que podemos cambiar mediante la propiedad **position**, que nos pueden interesar para modificar la posición en donde aparecen los diferentes elementos y su contenido.

A la propiedad **position** se le pueden indicar los siguientes valores:

- **static**: es el **valor por defecto**, un elemento con este valor no está posicionado.
- **relative**: se comporta igual que static a menos que le agreguemos las propiedades: top | bottom | right y/o left y así causamos un reajuste en su posición. Va a depender de su contenedor. Se va a posicionar en forma relativa a su contenedor.

Posicionamiento

- **absolute:** la posición del elemento se define de forma relativa al primer contenedor que posea una posición no estática. Siendo la posición del html en caso de no poseer padre con posicionamiento distinto de static.
- **fixed:** hace que la caja este posicionada con respecto a la ventana del navegador, lo que significa que se mantendrá en el mismo lugar incluso al hacer scroll en la página. La referencia es el viewport, la parte visual del navegador.
- **sticky:** se posiciona según el estado de desplazamiento del usuario. Se "pega" en su lugar, después de alcanzar una posición de desplazamiento determinada.

https://www.w3schools.com/cssref/pr_class_position.asp

Ver ejemplos:



- [ej-posicionamiento1.html](#) y [ej-posicionamiento1.css](#)

- [posicionamiento-sticky.html](#)

- [ej-posicionamiento2.html](#) y [ej-posicionamiento2.css](#)

Para seguir investigando: <https://developer.mozilla.org/es/docs/Web/CSS/position>

Posicionamiento

Si utilizamos un modo de posicionamiento diferente al estático (*absolute*, *fixed*, *sticky* o *relative*), podemos emplear una serie de propiedades para modificar la posición de un elemento. Estas propiedades son las siguientes:

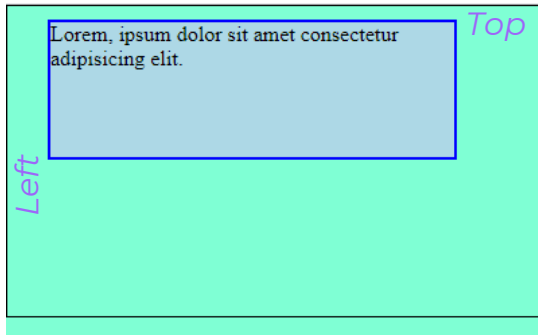
Propiedad	Valor	Significado
top:	auto <small>SIZE</small>	Empuja el elemento una distancia desde la parte superior hacia el inferior.
bottom:	auto <small>SIZE</small>	Empuja el elemento una distancia desde la parte inferior hacia la superior.
left:	auto <small>SIZE</small>	Empuja el elemento una distancia desde la parte izquierda hacia la derecha.
right:	auto <small>SIZE</small>	Empuja el elemento una distancia desde la parte derecha hacia la izquierda.
z-index:	auto <small>NUMBER</small>	Coloca un elemento en el eje de profundidad, más cerca o más lejos del usuario.

Las propiedades **top**, **bottom**, **left** y **right** sirven para mover un elemento desde la orientación que su propio nombre indica hasta su extremo contrario. Esto es, si utilizamos **left** e indicamos **20px**, estaremos indicando mover **desde la izquierda** 20 píxeles hacia la **derecha**.

Posicionamiento relativo

Si utilizamos la palabra clave **relative** activaremos el modo de *posicionamiento relativo*, que es el más sencillo de todos. En este modo, los elementos se colocan exactamente igual que en el posicionamiento estático (permanecen en la misma posición), pero dependiendo del valor de las propiedades **top**, **bottom**, **left** o **right** variaremos ligeramente la posición del elemento.

*Ejemplo: Si establecemos **left:30px**, el elemento se colocará 30 píxeles a la derecha **desde la izquierda** donde estaba colocado en principio, mientras que si especificamos **right:30px**, el elemento se colocará 30 píxeles a la izquierda **desde la derecha** donde estaba colocado en principio.*



```
.contenedor {  
  position: relative;  
  left: 30px;  
  top: 10px;  
  border: 2px solid blue;  
  background-color: lightblue;  
  width: 300px;  
  height: 100px;  
  margin: 0;  
}
```

CSS

Posicionamiento absoluto

Si utilizamos la palabra clave **absolute** estamos indicando que el elemento pasará a utilizar *posicionamiento absoluto*, que no es más que utilizar el documento completo como referencia. Esto no es exactamente el funcionamiento de este modo de posicionamiento, pero nos servirá como primer punto de partida para entenderlo.

*Ejemplo: Si establecemos **left:40px**, el elemento se colocará 40 píxeles a la derecha del extremo izquierdo de la página. Sin embargo, si indicamos **right:40px**, el elemento se colocará 40 píxeles a la izquierda del extremo derecho de la página.*

Realmente, este tipo de posicionamiento coloca los elementos **utilizando como punto de origen el primer contenedor con posicionamiento diferente a estático**.

Por ejemplo, si el contenedor padre tiene posicionamiento estático, pasamos a mirar el posicionamiento del padre del contenedor padre, y así sucesivamente hasta encontrar un contenedor con posicionamiento no estático o llegar a la etiqueta **<body>**, en el caso que se comportaría como el ejemplo anterior.

Posicionamiento fijo

El **posicionamiento fijo** es hermano del **posicionamiento absoluto**. Funciona exactamente igual, salvo que hace que el elemento se muestre en una posición fija **dependiendo de la región visual del navegador**. Es decir, aunque el usuario haga *scroll* y se desplace hacia abajo en la página web, el elemento seguirá en el mismo sitio posicionado.

*Ejemplo: Si establecemos **top:0** y **right:0**, el elemento se colocará justo en la esquina superior derecha y se mantendrá ahí aunque hagamos scroll hacia abajo en la página.*

Posicionamiento sticky

El **posicionamiento sticky** se suele utilizar cuando queremos que un elemento se posicione en un lugar específico de forma fija («*sticky*», *pegajoso*), como por ejemplo, cuando al hacer *scroll* llegamos a un elemento y queremos que ese elemento se quede fijo en la parte superior mientras continuamos haciendo scroll. No es como el fijo ya que no queda en una posición fija, sino que flota respecto del fondo y se queda adherido a la parte superior.

Material extra:

[Propiedad position CSS](#)

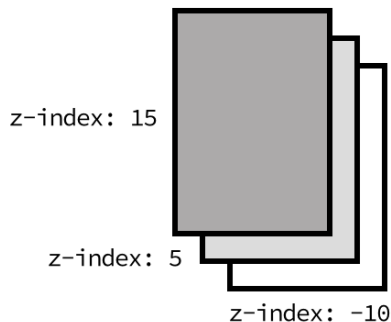
[HTML Layout](#)

[Video extra con un ejemplo de posicionamiento](#)

Profundidad (z-index)

La propiedad **z-index** establece el nivel de profundidad en el que está un elemento sobre los demás. De esta forma, podemos hacer que un elemento se coloque encima o debajo de otro, superponiéndose o quedando “apilados”. Los elementos con mayor valor z-index van a cubrir a aquellos con menor valor, para lo cual hay que indicar un número que representará el nivel de profundidad del elemento. Los elementos un número más alto estarán por encima de otros con un número más bajo, que permanecerán ocultos detrás de los primeros.

z-index: auto | number | initial | inherit;



Los niveles z-index, así como las propiedades top, left, bottom y right no funcionan con elementos que estén utilizando posicionamiento estático. Deben tener un tipo de posicionamiento diferente a estático.



Ver ejemplos [z-index.html](#) y [z-index2.html](#)

https://www.w3schools.com/cssref/pr_pos_z-index.asp

Juegos para aprender CSS

- <http://cssgridgarden.com>
- <http://www.flexboxdefense.com>
- <https://flexboxfroggy.com>
- <https://cssbattle.dev>
- <https://flukeout.github.io/>