El concepto de **cascada**.

Este concepto es un poco más avanzado, por lo que se debe conocer bien el temade **selectores CSS** y dominar algo de CSS para comprenderlo en su totalidad.

Antes de continuar, supongamos que nos encontramos ante el siguiente escenario:

```
<div>Texto del elemento</div>
<style>
div {
color: red; padding: 8px
}
div {
color: blue; background-color: grey
}
</style>
```

En este caso, ¿cuál de las dos reglas prevalece, si tenemos en cuenta que se refieren al mismo elemento y están al mismo nivel? La respuesta es muy fácil: Prevalece siempre la última regla definida, la cual **mezcla** y **sobreescribe** las propiedades anteriores.

En el caso anterior, el resultado final (*computado*) sería el siguiente:

```
div {
color: blue; /* Se sobreescribe la última */padding: 8px;
background-color: grey;
}
```

Sin embargo, puede ocurrir que en determinados casos no esté tan claro cuál es el estilo que debería sobrescribir a los anteriores. Ahí es cuando entra en juego el concepto de **cascada en CSS**, que es el que se encarga de eliminar la ambigüedady determinar el que tiene prioridad. Supongamos el siguiente caso:

```
<div id="nombre" class="clase">Texto del elemento</div>

<style>
```

```
div {
color: red;
}

#nombre { color: blue;
}

.clase {
color: green;
}
</style>
```

Resultado:
Texto del elemento

¿Cómo sabe CSS que estilo aplicar? ¿Cuál tiene prioridad sobre losdemás? Aquí es donde entra en acción el concepto de **cascada en CSS**.

Cascada CSS

Para saber qué bloque de estilos tiene prioridad, CSS analiza (*por orden*) tres conceptos clave del código CSS: su **importancia**, la **especificidad** y su **orden**. Veamos en que se basa cada uno de ellos.

Importancia

La **importancia** de un código CSS se determina dependiendo de las hojas de estilo donde está colocado. Generalmente, no necesitaremos preocuparnos de este factor, pero siemprees una buena idea conocer cómo funciona. Existen varios tipos de hojas de estilo, de menor a mayor importancia:

Tipo de CSS	Descripción	Definido por

Agente de usuario	Son los estilos CSS que aplica el navegador por defecto.	Navegador
CSS de usuario	Son los estilos CSS que añade el usuario, por razones específicas.	Usuario
CSS de autor	Son los estilos CSS que coloca el autor de la página.	Desarrollador

Aunque no es recomendable utilizarlo frecuentemente, se puede añadir al final de cada regla el texto !**important**, consiguiendo que la regla en cuestión tenga prioridad sobre las demás, independientemente del nivel o la altura a la que estén:

```
<div>Texto del elemento</div>

<style>
div {
color: red !important;padding: 8px
}
div {
color: blue; background-color: grey
}
</style>
```

El resultado final de este código CSS sería:div {
color: red; padding: 8px;
background-color: grey
}
Resultado:
Texto del elemento

Especificidad

En segundo caso, y si la importancia no elimina la ambigüedad, se pasa a determinar la especificidad de los selectores CSS. Para ello, se sigue un cálculo matemático basado en 4 componentes: **A**, **B**, **C**, **D**.

Componente	**Descripción**
Componente A	Número de estilos aplicados mediante un atributo **style**.
Componente B	Número de veces que aparece un **id** en el selector.
Componente C	Número de veces que aparece una **clase**, **pseudoclase** o **atributo** en el selector.
Componente D	Número de veces que aparece un **elemento** o un **pseudoelementos** en el selector.

Para saber si un bloque de CSS es más específico que otro (*y, por lo tanto, tiene prioridad*) sólo hay que calcular sus componentes. Se ordenan teniendo en cuentalos valores de cada componente, de izquierda a derecha.

Veamos algunos ejemplos, ordenados de **menor a mayor especificidad**:

```
div { ... }                          /* Especificidad: 0,0,0,1 */
div div { ... }                      /* Especificidad: 0,0,0,2 */
#pagina div { ... }                  /* Especificidad: 0,1,0,1 */#pagina div:hover
{ ... }                             /* Especificidad: 0,1,1,1 */ #pagina div:hover
```

a { ... } /* Especificidad: 0,1,1,2 */ #pagina .sel:hover>a { ... } /* Especificidad: 0,1,2,1 */

Ver: https://specifishity.com/

Orden

En CSS, es posible crear múltiples reglas CSS para definir un mismo concepto. En este caso, la que prevalece ante todas las demás depende de ciertos factores, comoes la *«altura»* a la que está colocada la regla:

- El **CSS embebido** en un elemento HTML es el que tiene mayor precedencia, por lo que siempre será el que tenga prioridad sobre otras reglas CSS.
- En segundo lugar, el **CSS interno** definido a través de bloques **style** en el propio documento HTML será el siguiente a tener en cuenta en orden de prioridad.
- Por último, los documentos **CSS externos** son la tercera opción de prioridada la hora de tomar en cuenta las reglas CSS.

Teniendo esto en cuenta, hay que recordar que las propiedades que prevaleceránserán las que estén en último lugar, siempre respetando la prioridad de la lista anterior.

Fuente: lenguajehtml.com y lenguajecss.com

Los selectores descendientes

El combinador de un espacio en blanco (que se supone que representan un espacio, o mejor dicho uno o más espacios en blanco) combina dos selectores tales que el selector combinado incluye sólo los elementos que coinciden con el segundo selector para los que hay un elemento ancestro que coincide con el primer selector. Los selectores descendientes son similares a selectores hijos , pero que no requieren que la relación entre los elementos coincidentes ser estrictamente entre padres e hijos.

Sintaxis

selector1 selector2 { propiedades de estilos }

Ejemplo

```
span { background-color: white; }

div span { background-color: blue; }

<div>
  <span>Span 1.
    <span>Span 2.</span>
  </span>
</div>

<span>Span 3.</span>
```

La versión CSS 3 incluye todos los selectores de CSS 2.1 y añade otras decenas de selectores, pseudo-clases y pseudo-elementos.

CSS 3 añade tres nuevos selectores de atributos:

• elemento[atributo^="valor"], selecciona todos los elementos que disponen de ese atributo y cuyo valor comienza exactamente por la cadena de texto indicada.

• elemento[atributo$="valor"], selecciona todos los elementos que disponen de ese atributo y cuyo valor termina exactamente por la cadena de texto indicada.

• elemento[atributo*="valor"], selecciona todos los elementos que disponen de ese atributo y cuyo valor contiene la cadena de texto indicada.

De esta forma, se pueden crear reglas CSS tan avanzadas como las siguientes:

```
/* Selecciona todos los enlaces que apuntan a una dirección de correo electrónico */

a[href^="mailto:"] { ... }
```

/* Selecciona todos los enlaces que apuntan a una página HTML */

a[href$=".html"] { ... }

/* Selecciona todos los títulos h1 cuyo atributo title contenga la palabra "capítulo" */

h1[title*="capítulo"] { ... }

Otro de los nuevos selectores de CSS 3 es el "selector general de elementos hermanos", que generaliza el selector adyacente de CSS 2.1. Su sintaxis es elemento1 ~ elemento2 y selecciona el elemento2 que es hermano de elemento1 y se encuentra detrás en el código HTML. En el selector adyacente la condición adicional era que los dos elementos debían estar uno detrás de otro en el código HTML, mientras que ahora la única condición es que uno esté detrás de otro.

Si se considera el siguiente ejemplo:

h1 + h2 { ... } /* selector adyacente */

h1 ~ h2 { ... } /* selector general de hermanos */

<h1>...</h1>

<h2>...</h2>

...

 <h2>...</h2>

<h2>...</h2>

El primer selector (h1 + h2) sólo selecciona el primer elemento <h2> de la página, ya que es el único que cumple que es hermano de <h1> y se encuentra justo detrás en el código HTML. Por su parte, el segundo selector (h1 ~ h2) selecciona todos los elementos <h2> de la página salvo el segundo. Aunque el segundo <h2> se encuentra detrás de <h1> en el código HTML, no son elementos hermanos porque no tienen el mismo elemento padre.

Los pseudo-elementos de CSS 2.1 se mantienen en CSS 3, pero cambia su sintaxis y ahora se utilizan :: en vez de : delante del nombre de cada pseudo-elemento:

::first-line, selecciona la primera línea del texto de un elemento.

::first-letter, selecciona la primera letra del texto de un elemento.

::before, selecciona la parte anterior al contenido de un elemento para insertar nuevo contenido generado.

::after, selecciona la parte posterior al contenido de un elemento para insertar nuevo contenido generado.

CSS 3 añade además un nuevo pseudo-elemento:

::selection, selecciona el texto que ha seleccionado un usuario con su ratón o teclado.

Las mayores novedades de CSS 3 se producen en las pseudo-clases, ya que se añaden 12 nuevas, entre las cuales se encuentran:

elemento:**nth-child**(numero), selecciona el elemento indicado pero con la condición de que sea el hijo enésimo de su padre. Este selector es útil para seleccionar el segundo párrafo de un elemento, el quinto elemento de una lista, etc.

elemento:**nth-last-child**(numero), idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

elemento:**empty**, selecciona el elemento indicado pero con la condición de que no tenga ningún hijo. La condición implica que tampoco puede tener ningún contenido de texto.

elemento:**first-child** y elemento:**last-child**, seleccionan los elementos indicados pero con la condición de que sean respectivamente los primeros o últimos hijos de su elemento padre.

elemento:**nth-of-type**(numero), selecciona el elemento indicado pero con la condición de que sea el enésimo elemento hermano de ese tipo.

elemento:**nth-last-of-type**(numero), idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

Algunas pseudo-clases como :**nth-child**(numero) permiten el uso de expresiones complejas para realizar selecciones avanzadas:

```
li:nth-child(2n+1) { ... }   /* selecciona todos los elementos impares de una lista */

li:nth-child(2n)   { ... }   /* selecciona todos los elementos pares de una lista */


/* Las siguientes reglas alternan cuatro estilos diferentes para los párrafos */

p:nth-child(4n+1) { ... }

p:nth-child(4n+2) { ... }

p:nth-child(4n+3) { ... }

p:nth-child(4n+4) { ... }
```

Empleando la pseudo-clase :nth-of-type(numero) se pueden crear reglas CSS que alternen la posición de las imágenes en función de la posición de la imagen anterior:

```
img:nth-of-type(2n+1) { float: right; }

img:nth-of-type(2n)   { float: left;  }
```

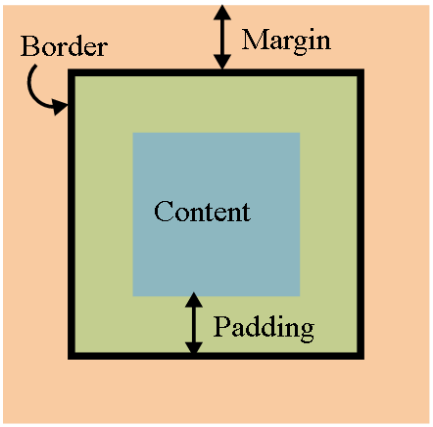
Otro de los nuevos selectores que incluirá CSS 3 es :not(), que se puede utilizar para seleccionar todos los elementos que no cumplen con la condición de un selector:

```
:not(p) { ... }  /* selecciona todos los elementos de la página que no sean párrafos */
:not(#especial) { ... }  /* selecciona cualquier elemento cuyo atributo id no sea "especial" */
```

Modelo de caja

Durante varios años, el denominado **modelo de cajas** fue una pesadilla para losdesarrolladores web, puesto que se mostraba visualmente de forma diferente en**Internet Explorer** respecto a los demás navegadores. Por fortuna, todos los navegadores actuales ya interpretan de la misma forma el modelo de cajas, peroconviene aprender bien la diferencia para no ser como Internet Explorer.

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 delelemento.
- El **márgen** (*margin*). En naranja, es la parte exterior del elemento, por fueradel borde.
- El **relleno** (*padding*). En verde, es la parte interior del elemento, entre elcontenido y el borde.
- El **contenido** (*en azul*). En azul, es la parte interior del elemento, excluyendoel relleno.

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
width	auto |	Tamaño de ancho de un elemento.
height	auto |	Tamaño de alto de un elemento.

En el caso de utilizar el valor **auto** en las propiedades anteriores (*que es lo mismo que no indicarlas, 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. Esto es algo que también puede variar, dependiendo del tipo de elementoque estemos usando, y que veremos más adelante, en el apartado de maquetación.

Hay que ser muy conscientes de que, sin indicar valores de ancho y alto para la caja, el elemento generalmente toma el tamaño que debe respecto a su contenido, mientras que, si indicamos un ancho y alto concretos, **estamos obligando a CSS tener un aspecto concreto** y podemos obtener resultados similares al siguiente (*conocida broma de CSS*) si su contenido es más grande que el tamaño que hemosdefinido:

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, establecemosunos máximos y unos mínimos, donde el ancho o alto podría variar entre esos valores.

```
div {
width: 800px; height: 400px; background: red; max-width: 500px;
}
```

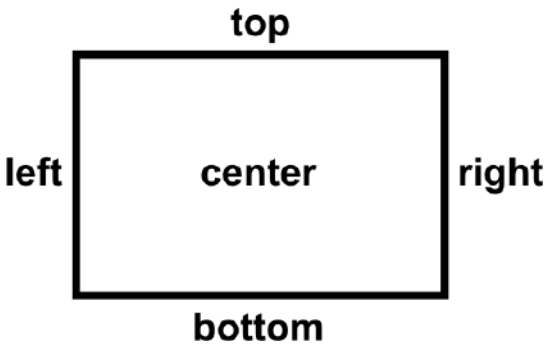
En este caso, por ejemplo, a pesar de estar indicando un tamaño de **800px**, le aplicamos un **max-width** de **500px**, por lo que estamos limitando el elemento a untamaño de ancho de 500 píxeles como máximo y nunca superará ese tamaño.

Por un lado, tenemos las propiedades de mínimos **min-width** y **min-height**, que pordefecto tienen valor **0**, mientras que, por otro lado, tenemos las propiedades de máximos **max-width** y **max-height**, que por defecto tienen valor **none**:

Propiedad	Propiedad	Significado
max-width	**none** |	Ancho máximo que puede ocupar unelemento.
min-width	**0** |	Ancho mínimo que puede ocupar un elemento.
max-height	**none** |	Alto máximo que puede ocupar un elemento.
min-height	**0** |	Alto mínimo que puede ocupar un elemento.

Zonas de un elemento

Antes de continuar, es importante saber que en CSS existen ciertas palabras clavepara hacer referencia a una zona u orientación concreta sobre un elemento. Son conceptos muy sencillos y prácticamente lógicos, por lo que no tendrás ningún problema en comprenderlos. Son los siguientes:



- **Top**: Se refiere a la parte superior del elemento.
- **Left**: Se refiere a la parte izquierda del elemento.
- **Right**: Se refiere a la parte derecha del elemento.
- **Bottom**: Se refiere a la parte inferior del elemento.
- **Center**: En algunos casos se puede especificar el valor **center** para referirsea la posición central entre los extremos horizontales o verticales.

Desbordamiento

Volvamos a pensar en la situación de la imagen anterior: Damos un tamaño de ancho y alto a un elemento HTML, pero su contenido de texto es tan grande que nocabe dentro de ese elemento. ¿Qué ocurriría? Probablemente lo que vimos en la imagen: el contenido se desbordaría.

Podemos modificar ese comportamiento con la propiedad de CSS **overflow**, o conalguna de sus propiedades específicas **overflow-x** o **overflow-y**:

Propiedad	**Valor**	**Significado**
overflow	visible | hidden | scroll | auto	Establece el comportamiento dedesbordamiento.
overflow-x	visible | hidden | scroll | auto	Establece el desbordamiento sólo para el eje X (horizontal).
overflow-y	visible | hidden | scroll | auto	Establece el desbordamiento sólo para el eje Y (vertical).

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

Propiedad	**Valor**	**Significado**
visible	Se muestra el contenido que sobresale (comportamiento por defecto)	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

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

Márgenes y rellenos

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

Márgenes

Dichos márgenes se pueden considerar en conjunto (*de forma general*) o de forma concreta en cada una de las zonas del elemento. Veamos primero las propiedadesespecíficas para cada zona:

Propiedad	Valor	Significado
margin-top	**auto** |	Establece un tamaño de margen superior.
margin-left	**auto** |	Establece un tamaño de margen a laizquierda.
margin-right	**auto** |	Establece un tamaño de margen a laderecha.
margin-bottom	**auto** |	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

automáticaindicando el valor **auto**.

Truco: 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 repartirloequitativamente entre el margen izquierdo y el margen derecho, quedando centradoel elemento.

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

Obsérvese también el siguiente ejemplo para ilustrar el **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** entotal, y no **40px** (*la suma de cada uno*) como podríamos pensar en un principio,

Rellenos

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

Propiedad	Valor	Significado
padding-top	**0** |	Aplica un relleno interior en el espacio superior de un elemento.

padding-left	**0** |	Aplica un relleno interior en el espacio izquierdo de un elemento.
padding-right	**0** |	Aplica un relleno interior en el espacio derecho de un elemento.
padding-bottom	**0** |	Aplica un relleno interior en el espacio inferior de un elemento.

Atajo: Modelo de cajas

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

Propiedad	Valor	Significado
margin		1 parámetro. Aplica el mismomargen a **todos** los lados.
		2 parámetros. Aplica margen **top/bottom** y **left/right**.

		3 parámetros. Aplica margen **top**,**left/right** y **bottom**.
		4 parámetros. Aplica margen **top**,**right**, **bottom** e **left**.

Con las propiedades padding y border-width pasa exactamente lo mismo, actuando en relación a los rellenos, en lugar de los márgenes en el primer caso, y en relación al grosordel borde de un elemento en el segundo.

Ojo: Aunque al principio es muy tentador utilizar márgenes negativos para ajustar posiciones y colocar los elementos como queremos, se aconseja no utilizar dicha estrategiasalvo para casos muy particulares, ya que a la larga es una mala práctica que hará que nuestro código sea de peor calidad

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

Propiedad	**Valor**	**Significado**
border-color		Especifica el color que se utilizará en el borde.
border-width	thin | **medium** | thick |	Especifica un tamaño predefinido para el grosor del borde.
border-style	**none** |	Define el estilo para el borde a utilizar (ver más adelante).

En primer lugar, **border-color** establece el color del borde, de la misma forma que lo hicimos en apartados anteriores de colores. En segundo lugar, con **border-width** podemos establecer la anchura o grosor del borde utilizando tanto **palabras clave** predefinidas como un tamaño concreto con cualquier tipo de las **unidades** ya vistas.

Estilos de borde

Por último, con **border-style** podemos aplicar un estilo determinado al borde de un elemento. En **estilo de borde** podemos elegir cualquiera de las siguientes opciones:

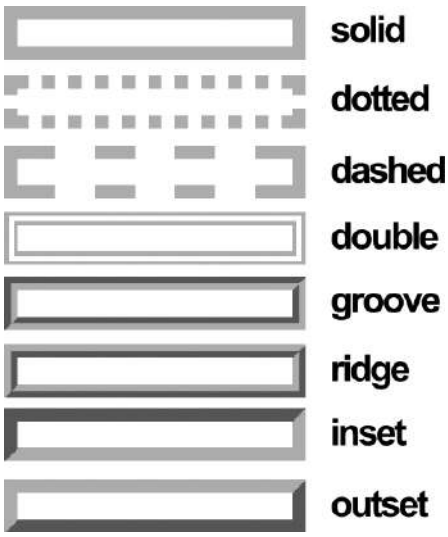
Propiedad	**Valor**	**Significado**
hidden	Oculto. Idéntico al anterior salvo para conflictos con tablas.	
dotted	Establece un borde basado en puntos.	
dashed	Establece un borde basado en rayas (línea discontínua).	
solid	Establece un borde sólido (línea contínua).	
double	Establece un borde doble (dos líneas contínuas).	
groove	Establece un borde biselado con luz desde arriba.	

ridge	Establece un borde biselado con luz desde abajo. Opuesto a **groove**.	
inset	Establece un borde con profundidad «haciadentro».	
outset	Establece un borde con profundidad «hacia fuera». Opuesto a **inset**.	

Veamos un ejemplo sencillo:

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

Sin embargo, el borde más frecuente suele ser **solid**, que no es más que un bordeliso. Pueden utilizarse cualquiera de los estilos indicados en la tabla anterior.
Veamos cómo se verían los diferentes estilos de borde utilizando **10 píxeles** de grosor y color **gris**:



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, bordederecho, borde inferior y borde izquierdo*). Sin embargo, podemos especificar uno, dos, tres o cuatro parámetros, dependiendo de lo que queramos hacer:

Propiedad	**Valor**	**Significado**
border-color		1 parámetro. Aplica el mismo color a todos losbordes.
		2 parámetros. Aplica al borde **top/bottom**, y al **left/right**.
		3 parámetros. Aplica al **top**, al **left/right** y al **bottom**.
		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 delborde*). 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 dotted solid;
}
```

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

Atajo: Bordes

Pero ya habremos visto que con tantas propiedades, para hacer algo relativamentesencillo, nos pueden quedar varias lineas de código complejas y difíciles de leer. Aligual que con otras propiedades CSS, podemos utilizar la propiedad de atajo **border**, con la que podemos hacer un resumen y no necesitar utilizar las propiedades individuales por separado, realizando el proceso de forma más corta:

Propiedad	Valor	Significado
border		Propiedad de atajo para simplificar valores.

Por ejemplo:

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

Así pues, estamos aplicando un borde de **1 píxel** de grosor, estilo **sólido** y color **negro** a todos los bordes del elemento, ahorrando mucho espacio y escribiéndolotodo en una sola propiedad.

Consejo: Intenta organizarte y aplicar siempre los atajos si es posible. Ahorrarás mucho espacio en el documento y simplificarás la creación de diseños. El orden, aunque no es obligatorio, si es recomendable para mantener una cierta coherenciacon el estilo de código.

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-**:

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

Esto dibujaría **sólo un borde inferior** negro de 2 píxeles de grosor y con estilo punteado. Ahora imaginemos que queremos un elemento con todos los bordes en rojo a 5 píxeles de grosor, salvo el borde superior, que lo queremos con un borde de15 píxeles en color naranja. Podríamos hacer lo siguiente:

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

El ejemplo anterior conseguiría nuestro objetivo. La primera propiedad establece todos los bordes del elemento, sin embargo, las siguientes propiedades modificansólo el borde superior, cambiándolo a las características indicadas.

Recuerda que también existen atajos para estas propiedades de bordes en zonasconcretas, lo que nos permite simplificar aún más el ejemplo anterior, haciéndolo más fácil de comprender:

```
div {
border: 5px solid red;
border-top: 15px solid orange;
}
```

Ojo: Es muy importante entender como se está aplicando la herencia en los ejemplos anteriores, puesto que es una de las características más complejas de dominar de CSS junto a la cascada. Por ejemplo, si colocaramos el **border-top** antes del **border**, este último sobreescribiría los valores de **border-top** y no funcionaría de la misma forma.

Fuente: lenguajehtml.com y lenguajecss.com

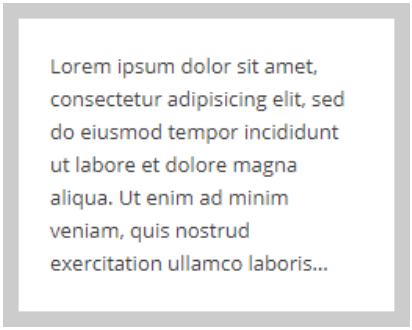
Box-Sizing

box-sizing es una propiedad CSS para cambiar el modelo de caja por defecto delos navegadores.

El ancho de un elemento se altera si se le aplica un borde o un padding. Eso es porque **la anchura del elemento que tu especificas con CSS, por defecto noincluye borde ni padding**.

Un ejemplo: Éste es el efecto que tiene un padding y un borde sobre un elemento de200px de ancho:

```
<div style="width:200px;padding: 20px;

border: 10px solid #ccc;margin: 0 auto;">

Lorem ipsum...

</div>
```



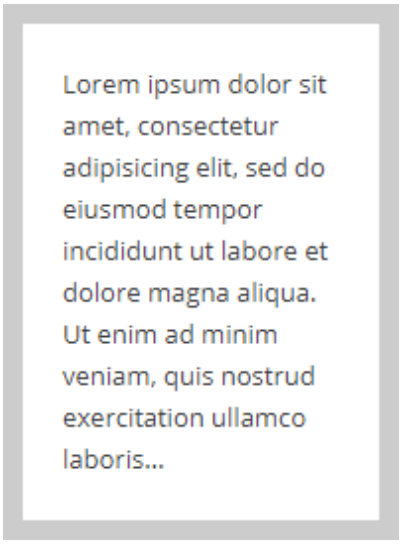
Como se puede comprobar, no mide 200px de ancho, sino **260px**. Es decir: 200px de ancho inicial, más 20px de padding izquierdo, más 20px de padding derecho, más 10px de borde izquierdo, más 10px de borde derecho.

Éste es el modo en el que los navegadores tratan los anchos por defecto. Sería elequivalente a **box-sizing: content-box;**

box-sizing: border-box

Con border-box, hacemos que el ancho especificado sea el equivalente al **anchototal**. Es decir:

```
<div style="width: 200px;padding: 20px;

border: 10px solid #ccc;box-sizing: border-box; margin: 0 auto;">

Lorem ipsum...

</div>
```



Como se puede comprobar, ahora ese elemento **exactamente 200px**, ni uno más,ni uno menos.

Esto es **muy útil para elementos fluidos**, cuando necesitas que el elemento ocupe(por ejemplo) un 33% del ancho, y si ocupa un píxel más toda la estructura
se estropearía.

Fuente: emiliocobos.net

Unidades de Medidas

Unidades absolutas

Se llaman absolutas porque su valor es el declarado. No hay cálculo del mismo. Esel que es y de ahí no se mueve

Pixel

Px: Los píxeles son unidades de tamaño fijo. Un pixel es igual a un punto en la pantalla del ordenador (la división más pequeña de la resolución de su pantalla) ypor lo tanto, es indivisible.

Su problema es que no es escalable, atentado contra la accesibilidad. Era una cuestión que muchos obviaban porque les permitía un control "al milímetro" de susrealizaciones.

Pero hoy a esa falla en la accesibilidad por algunos colectivos se suma el comportamiento en dispositivos de pequeñas dimensiones y grandes resoluciones.

Al usar pixeles para definir las dimensiones de los objetos y los estilos de fuente nosolo estamos definiendo tamaños rígidos, sino que también estamos ignorando las configuraciones que cada usuario pueda tener en su navegador.

Ni permite escalar a más los textos a los usuarios con problemas visuales ni amenos en dispositivos como los smartphones.

Puntos

Pt: El punto es una herencia de los medios impresos. Un punto es igual a 1/72 depulgada (según diversas fuentes), aprox. 0´04mm.

Al ser también una medida absoluta tiene todos los problemas de los pxaumentados.

Otras unidades absolutas:

mm, cm, in: Milímetros, centímetros, o pulgadas.
pc: picas (12 puntos.)

Unidades relativas

El valor final resultante (computado) está en función de un valor previo.

Em

Es básicamente el tamaño de una letra «M» (bien podría ser cualquier otra letra) del elemento al cual se esté aplicando esta medida. Es decir, si en elemento tiene aplicado un tamaño de fuente de 16 pixeles, entonces 1 em será igual a 16px(los navegadores de manera predeterminada definen un font-size de 16px al elemento HTML, por lo tanto, por defecto 1em es igual a 16px).

La unidad em es escalable y siempre depende de su elemento padre. Por ejemplo, si el elemento body tiene un tamaño de fuente de 16px y un elemento hijo tiene una fuente con tamaño 1.3em, este texto se mostrará de un tamaño un 30% más grandeque el del body (20.8px), mientras que si dentro de ese elemento tenemos otro hijo con un font-size de 1.3 em, el tamaño de fuente de este objeto sería un 30% más grande que el tamaño de su padre (27.04px).

Body = 1em (16px)
Hijo = 1.3em (16px x 1.3 = 20.8px) Nieto = 1.3em (20.8px x 1.3 = 27.04px)

Porcentaje %

La unidad de medida porcentual es la que se usa por defecto en los
elementos HTML en donde de manera predeterminada cada elemento de bloque usa un ancho del 100%, es por eso que cuando achicamos la ventana del navegador con una página que no tenga estilos, la página se adapta, ya que siempre usará el ancho total visible. Pero nosotros podemos utilizar los porcentajesde una manera más avanzada tratando de generar layouts más complejos.
Supongamos, por ejemplo, que tienes un div que contiene todos los elementos de lapágina y, según el diseño, este elemento debiera medir 1200 pixeles. En lugar de caer en la tentación de simplemente usar esa medida en pixeles, te recomendaría usar una medida en porcentajes, en donde el máximo ancho del elemento sea
esos 1200px:

CSS
.container { margin: 0 auto;
width: 90%;
max-width: 1200px;
}

Con estas 3 propiedades de CSS conseguimos que a) El elemento se centre en la página, b) tenga un ancho del 90% de la ventana y c) su ancho nunca sea superiora 1200 píxeles.

Es decir, hemos conseguido que el elemento con la clase container sea responsive sin la necesidad siquiera de escribir un media query.

El uso de los porcentajes también lo podemos llevar a elementos interiores del layout, en donde, por ejemplo, podemos asignar a la columna principal de contenidoy a la barra lateral unas medidas de ancho del 70% y el 30% respectivamente, haciendo que sean completamente adaptables al tamaño de su elemento contenedor.

Rem:

La unidad de medida rem es muy similar a em, con la única diferencia de que no esescalable, esto quiere decir que no depende del elemento padre, sino del elementoraíz del documento, el elemento HTML. Rem significa «Root Em«, o sea, es un em basado en la raíz.

Esto significa que si el elemento HTML tiene un tamaño de fuente de 16px (como espor defecto), entonces 1rem, sería igual a 16px, y si queremos aplicar un tamaño basado en rem a

cualquier elemento de la página, no importará cual sea el tamaño de fuente que tenga asociado ese elemento, ya que 1 rem siempre será igual a 16 pixeles a no ser que se modifique el elemento raíz.

Usar rem nos permite cierta estructura para poder definir ciertas partes del layout,pero al mismo tiempo nos entrega cierta escalabilidad para respetar las configuraciones de cada usuario.

Esta unidad de medida es recomendable para aplicar a elementos del layout que requieran medidas fijas y eventualmente también para textos que deseemos que tengan un tamaño de fuente que no dependa de su elemento padre.

Los rems, son una unidad muy interesante también para definir los media queries deCSS.

Si quisiéramos refinar el ejemplo anterior para no usar pixeles deberíamos usar algocomo lo siguiente:
CSS

```
.container { margin: 0 auto;
width: 90%;
max-width: 75rem;
}
```

Para poder convertir una medida de pixeles a rem solo tienes que multiplicar el tamaño que quieres obtener por el número 0.0625, eso te dará el tamaño que debesusar en rem. Así es como se define que 75rem es igual a 1200px:
75rem = 1200px x 0.0625

Para evitarnos este cálculo se puede definir el font-size del elemento html con un tamaño de 62.5%, de esta forma, conseguimos que 1 rem sea equivalente a 10px, haciendo más fácil el cálculo.

ex, ch: Son respectivamente la altura de la x minúscula, y el ancho del número 0. Aunqueno son tan soportadas por los navegadores como los rems.
vw, vh: Estas son respectivamente 1/100 del ancho de la ventana, y 1/100 de la altura de la ventana. Tampoco son tan soportadas como los rems.

Fuente: lenguajehtml.com y lenguajecss.com

Posicionamiento

A grandes rasgos, si tenemos varios elementos en línea (uno detrás de otro) aparecerán colocados de izquierda hacia derecha, mientras que si son elementos en bloque se verán colocados desde arriba hacia abajo. Estos elementos se puedenir combinando y anidando (incluyendo unos dentro de otros), construyendo 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 ordennatural 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 laposición en donde aparecen los diferentes elementos y su contenido.
A la propiedad position se le pueden indicar los siguientes valores:

Valor	Significado
static	Posicionamiento estático. Utiliza el orden natural de los elementos HTML.
relative	Posicionamiento relativo. Los elementos se mueven ligeramente en base a suposición estática.
absolute	Posicionamiento absoluto. Los elementos se colocan en base al contenedor padre.
fixed	Posicionamiento fijo. Idem al absoluto, pero, aunque hagamos scroll no se mueve.

Propiedad	**Valor**	**Significado**
top:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte superiorhacia el inferior.
bottom:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte inferiorhacia la superior.
left:	auto | TAMAÑO	Empuja el elemento una distancia desde laparte izquierda hacia la derecha.

right:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte derechahacia la izquierda.
z-index:	auto | nivel	Ordena en el eje de profundidad, superponiendo uocultando.

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

Antes de pasar a explicar los tipos de posicionamiento, debemos tener claras las propiedades top, bottom, left y right, que 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 20píxeles hacia la derecha.

Pero pasemos a ver cada tipo de posicionamiento por separado y sucomportamiento:

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 colocanexactamente 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:40px, el elemento se colocará 40 píxeles a la derechadesde la izquierda donde estaba colocado en principio, mientras que si especificamos right:40px, el elemento se colocará 40 píxeles a la izquierda desde laderecha donde estaba colocado en principio.

Posicionamiento absoluto
Si utilizamos la palabra clave absolute estamos indicando que el elemento pasará autilizar posicionamiento absoluto, que no es más que utilizar el documento completocomo 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 derechadel 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.
Como mencionaba anteriormente, aunque este es el funcionamiento del posicionamiento absoluto, hay algunos detalles más complejos en su modo de trabajar. 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 mirarel 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.

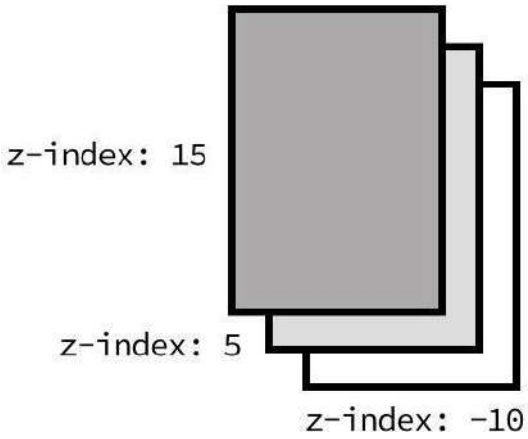
Posicionamiento fijo
Por último, el posicionamiento fijo es hermano del posicionamiento absoluto. Funciona exactamente igual, salvo que hace que el elemento se muestre en una posición fija dependiendo de la región visual del navegador. Es decir, aunque el usuario haga scroll y se desplace hacia abajo en la página web, el elemento seguiráen el mismo sitio posicionado.
Ejemplo: Si establecemos top:0 y right:0, el elemento se colocará justo en la esquinasuperior

derecha y se mantendrá ahí aunque hagamos scroll hacia abajo en la página.

Profundidad (niveles)

Es interesante conocer también la existencia de la propiedad z-index, que 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.

Su funcionamiento es muy sencillo, sólo 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.



Nota: 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.

Fuente: lenguajehtml.com y lenguajecss.com

Selectores avanzados en CSS

Al margen de la selección «básica» de elementos a través de CSS, que suele realizarse mediante clases e IDs, existe un amplio abanico de métodos para **seleccionar elementosdependiendo de la estructura del documento HTML** denominados **combinadores CSS**:

Nombre	Símbolo	Ejemplo	Significado
Agrupación de selectores	,	p, a, div { }	Se aplican estilos a varios elementos.
Selector descendiente		#page div { }	Se aplican estilos a elementos dentro de otros.
Selector hijo	>	#page > div { }	Se aplican estilos a elementos hijos directos.
Selector hermano adyacente	+	div + div { }	Se aplican estilos a elementos que siguen a otros.
Selector hermano general	~	div ~ div { }	Se aplican estilos a elementos al mismo nivel.
Selector universal	*	#page * { }	Se aplican estilos a todos los elementos.

Agrupación de selectores

En muchas ocasiones nos ocurrirá que tenemos varios bloques CSS con selectores diferentes pero con los mismos estilos exactamente, algo que generalmente no es apropiado. Si esto ocurre a menudo, el tamaño del documento CSS ocupará más y tardarámás en descargarse:

```
.container-logo { border-color: red; background: white;
 }

.container-alert { border-color: red; background: white;
 }

.container-warning {border-color: red; background: white;
 }
```

Una buena práctica es **ahorrar texto y simplificar** nuestro documento CSS lo máximo posible, por lo que podemos hacer uso de la **agrupación CSS** utilizando la **,** (*coma*).

De esta forma, podemos pasar de tener el ejemplo anterior, a tener el siguiente ejemplo(*que es totalmente equivalente*), donde hemos utilizado la agrupación para decirle al navegador que aplique dichos estilos las diferentes clases:

```
.container-logo, .container-alert, .container-warning {border-color: white;
 background: red;
 }
```

Selector descendiente

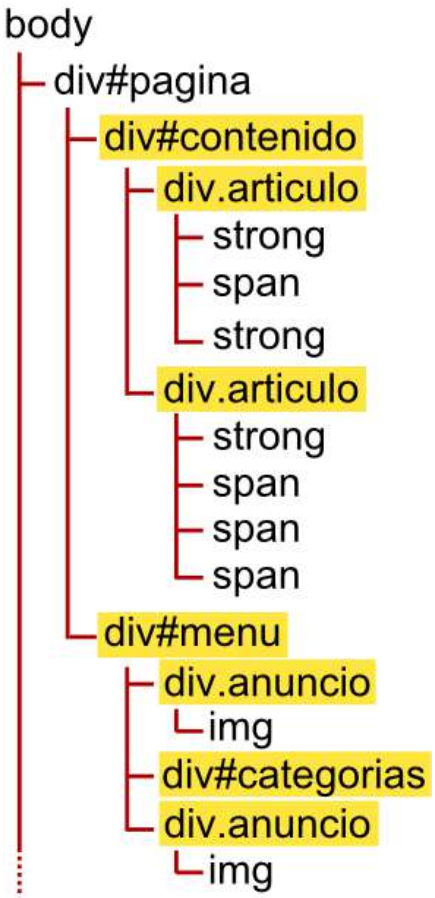
En CSS podemos utilizar lo que se llama el selector descendiente, que no es más que una forma de seleccionar ciertos elementos que están dentro de otros elementos. Esto puede parecer sencillo, pero cuidado, ya que puede ser una fuente de problemas si no se entiendebien.

Su sintaxis se basa en colocar los elementos uno a continuación de otro, separado por unespacio:

```
div#pagina div { background-color: blue;
 }
```

En el ejemplo anterior, aplicamos los estilos CSS (*color azul de fondo*) a todos los elementos ** ** que estén dentro de un ** ** con ID **pagina**. De esta forma, si existe unelemento ** ** fuera del ** ** con id **pagina**, no se aplicarán los estilos indicados:

CSS: div#pagina div



Repasemos varios detalles importantes respecto a este combinador CSS:

- Se están seleccionando todos los elementos ** ** que están dentro de ** ** conID **pagina**.
- Observa que se seleccionan independientemente del nivel al que estén (*hijos,abuelos, ...*)
- En este caso, el **div** de **div#pagina** es innecesario, ya que habíamos dicho que los**IDs** no se pueden repetir. Si ya existe un elemento con ID **pagina**, no hace falta diferenciarlo también por etiqueta. Si se tratase de una clase, si podría usarse.

Se pueden construir selectores muy complejos con tantos elementos como se quiera, pero una buena práctica es mantenerlos simples. Cuántos más elementos descendientes existanen un selector, más complejo será el procesamiento de dicha regla por los navegadores. Lorecomendable es ser despierto y utilizar sólo los necesarios.

```
<div class="menu">
<div class="options">
<ul>
```

```
<li><a href="/one">Option 1</a></li>
<li><a href="/two">Option 2</a></li>
<li><a href="/three">Option 3</a></li>
</ul>
</div>
</div>
```

Observando el fragmento de código HTML anterior, veamos las siguientes 2 formas de aplicar estilos CSS a los enlaces **<a>**:

```
/* Forma 1 */
.menu .options ul li a {color: orange;
}

/* Forma 2 */
.menu a { color: orange;
}
```

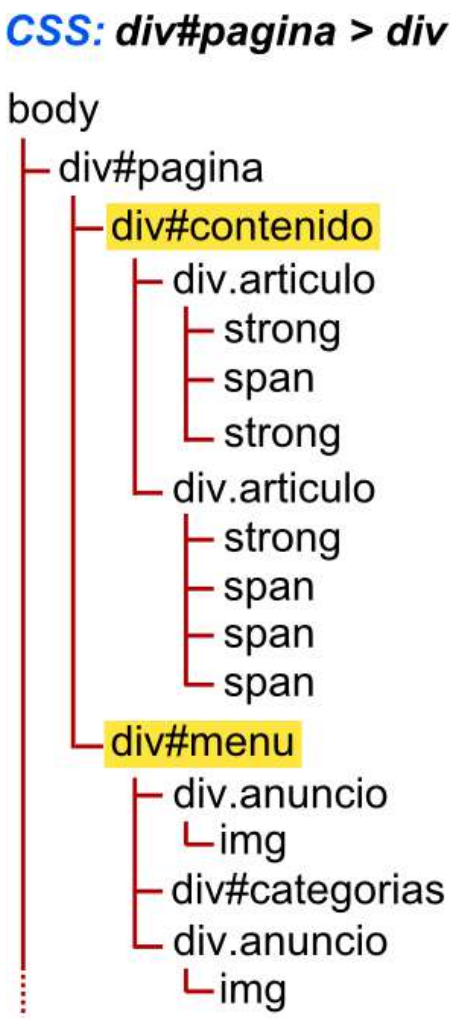
Mientras que la primera es mucho más específica, es una **muy buena práctica** en CSSmantener los selectores lo **menos específicos** posibles para evitar problemas de Especificidad (*a.k.a. CSS Peter Griffin*):

Selector hijo

Aunque el selector descendiente es bastante interesante, nos puede interesar hacer la misma operación, pero en lugar de seleccionar todos los elementos descendientes, seleccionar sólo los descendientes directos del elemento con el símbolo >, descartando asínietos y sucesivos.

```
#pagina > div { background-color: blue;
}
```

Veamos los elementos seleccionados en el documento de ejemplo para afianzar conceptos:



Al contrario que en el caso anterior, no se seleccionan todos los elementos ** ** descendientes, sino solo aquellos que son hijos directos del primer elemento especificado.

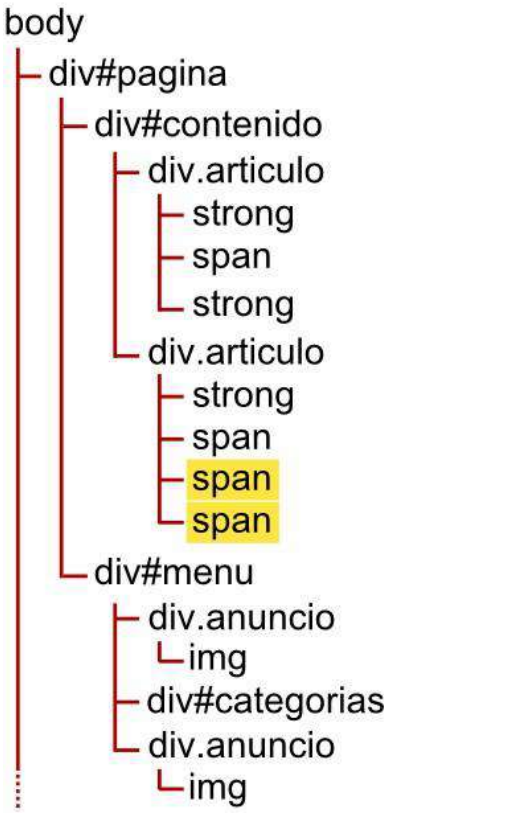
Selector hermano adyacente

Es posible también hacer referencia a los elementos hermanos, es decir, aquellos elementos que están directamente a continuación del elemento especificado. Mediante elsímbolo **+** del **selector hermano adyacente**, se pueden seleccionar aquellos elementos hermanos que están seguidos el uno de otro (*en el mismo nivel*):

```
div.articulo span + span {color: blue;
}
```

Cómo se podrá ver en este nuevo ejemplo, este combinador CSS hará que se seleccionenlos elementos **span** que estén a continuación de un **div.articulo span**:

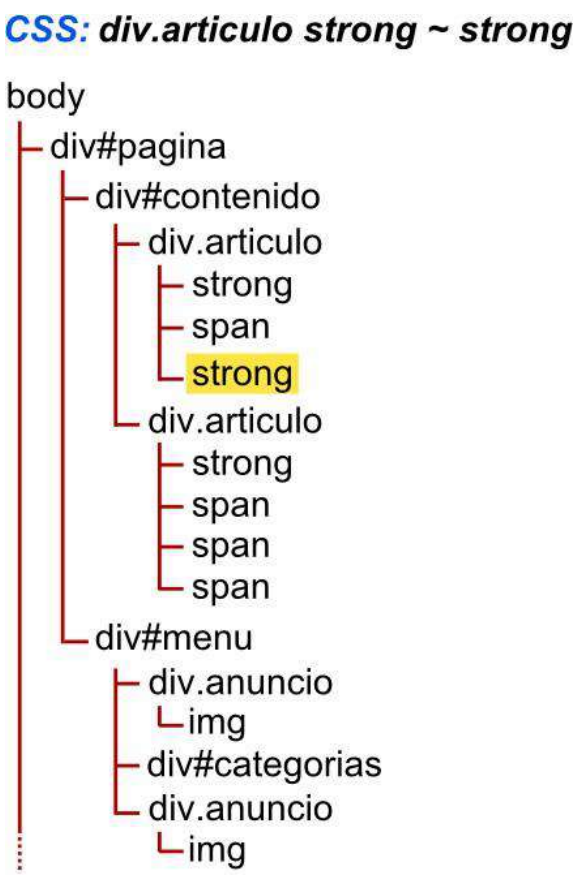
CSS: *div.articulo span + span*



Obsérvese que el primer elemento **** no es seleccionado, puesto que es el que estamos tomando de base. Una buena forma para entenderlo es leerlo de la siguiente forma: «*todo elemento **** que esté inmediatamente precedido de un *****».

Selector hermano general

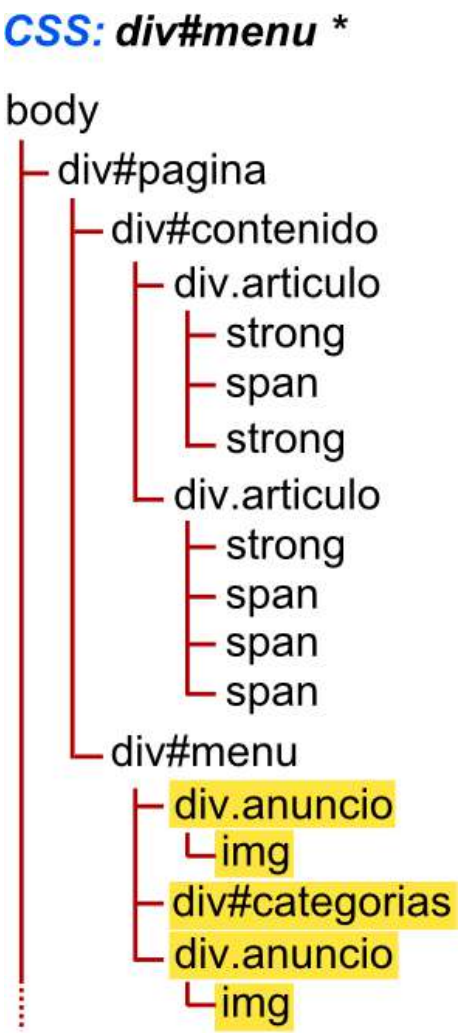
Si pensamos otras opciones en el ejemplo anterior, es posible que necesitemos ser menosespecíficos y en lugar de querer seleccionar los elementos hermanos **que sean adyacentes**, queramos seleccionar todos los hermanos en general, sin necesidad de que sean adyacentes. Esto se puede conseguir con el **selector hermano general**, simbolizadocon el carácter **~**:



Como se ve en el ejemplo, no es necesario que el elemento **strong** se encuentre adyacenteal primero, sino que basta con que sean hermanos en el mismo nivel.

Selector universal

Por último, el **selector universal** se simboliza con un asterisco ***** y es la forma de aplicarciertos estilos en **TODOS Y CADA UNO** de los elementos HTML correspondientes.



Este ejemplo selecciona todos los elementos dentro de **div#menu**. Es importante recalcar ladiferencia de seleccionar **#menu**, a seleccionar todos los elementos dentro de **#menu**, que es lo que estamos haciendo en este caso.

El **selector universal** puede ser muy útil en algunos casos para **resetear** ciertas propiedades de todo un documento, como en el siguiente ejemplo, donde se eliminan los márgenes de todos los elementos del documento HTML, puesto que algunos navegadoresponen márgenes diferentes y esto puede producir ciertas inconsistencias en los diseños:

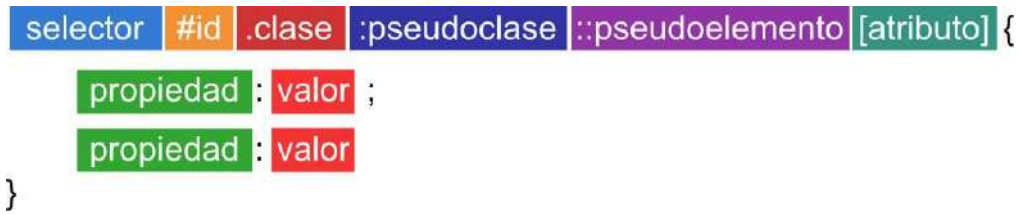
```
/* Elimina márgenes y rellenos de todos los elementos de un documento HTML */
* {
margin: 0;
padding: 0;
}
```

Fuente: [lenguajecss.com]

Pseudoclases

Las **pseudoclases** se utilizan para hacer referencia a ciertos comportamientos de los elementos HTML. Así como los combinadores CSS se utilizan para dar estilos dependiendode donde estén colocados en la estructura del HTML, las pseudoclases se utilizan para dar estilos a elementos **respecto al comportamiento** que experimentan en determinado momento.

Volvamos a recordar el esquema general de sintaxis de CSS:



Las pseudoclases se definen añadiendo **dos puntos** antes de la pseudoclase concreta. Enel caso de existir selectores de etiqueta, id o clases, estas se escribirían a su izquierda.

Pseudoclases de enlaces

Existen algunas pseudoclases orientadas a los enlaces o hipervínculos. En este caso, permiten cambiar los estilos dependiendo del comportamiento del enlace:

Pseudoclase	Descripción
:link	Aplica estilos cuando el enlace no ha sido visitado todavía.
:visited	Aplica estilos cuando el enlace ha sido visitado anteriormente.

A continuación veremos un ejemplo donde seleccionamos mediante un simple selector **a** losenlaces que **aún no han sido visitados**, cambiando el color de los mismos o su formato, loque mostrará dichos enlaces de color verde y en negrita:

```
a:link {
color: green; font-weight: bold
}
```

Por otro lado, la pseudoclase **:visited** puede utilizarse para dar estilo a los enlaces que **hayan sido visitados previamente** en el navegador del usuario:

```
a:visited { color: purple;
font-weight: bold
}
```

Pseudoclases de mouse

Originalmente, las siguientes pseudoclases se utilizaban solamente en enlaces (*InternetExplorer no los soportaba en otros elementos*). Sin embargo, actualmente pueden ser utilizadas con seguridad en cualquier otro elemento, sin necesidad de ser **<a>**.

Pseudoclase	Descripción
:hover	Aplica estilos cuando pasamos el ratón sobre un elemento.
:active	Aplica estilos cuando estamos pulsando sobre el elemento.

La primera de ellas, **:hover**, es muy útil e interesante, ya que permite aplicar estilos a unelemento justo cuando el usuario está pasando el ratón sobre él. Es una de las pseudoclases más utilizadas:

```
/* Usuario mueve el ratón sobre un enlace */a:hover {
background-color: cyan;
padding: 2px
}
```

```
/* Usuario mueve el ratón sobre un div y resalta todos los enlaces que contiene */div:hover a {
background-color: steelblue;color: white;
}
```

Observese que podemos realizar acciones un poco más específicas, como el segundo ejemplo anterior, donde al movernos sobre un elemento div (*div:hover*), aplicaremos losestilos a los enlaces (*a*) que están dentro del mencionado **div**.

Por otro lado, la segunda pseudoclase, **:active**, permite resaltar los elementos que se encuentran activos, donde el usuario está pulsando de forma activa con el ratón:

```
a:active {
border: 2px solid #FF0000;padding: 2px
}.
```

Pseudoclases de interacción

Existen pseudoclases orientadas principalmente a los campos de formulario de páginaswebs y la interacción del usuario con ellos, veamos otro par interesante:

Pseudoclase	Descripción
:focus	Aplica estilos cuando el elemento tiene el foco.
:checked	Aplica estilos cuando la casilla está seleccionada.

Cuando estamos escribiendo en un campo de texto de un formulario de una página web, generalmente pulsamos TAB para cambiar al siguiente campo y SHIFT+TAB para volver alanterior. Cuando estamos posicionados en un campo se dice que ese campo **tiene el foco**,mientras que al pulsar TAB y saltar al siguiente, decimos que **pierde el foco**.

El comportamiento de «ganar el foco» puede gestionarse mediante la pseudoclase **:focus**:

```
/* El campo ha ganado el foco */input:focus {
```

```
border: 2px dotted #444
}
```

Por otro lado, la pseudoclase **:checked** permite aplicar el estilo especificado a los elementos **<input>** (*casillas de verificación o botones de radio*) u **<option>** (*la opción seleccionada de un* ***<select>***).

Por ejemplo, se podría utilizar el siguiente fragmento de código:

```
input:checked + span {color: green;
}
```

Este ejemplo añade el selector hermano **+** para darle formato al **** que contiene el texto y se encuentra colocado a continuación de la casilla **<input>**. De esta forma, los textosque hayan sido seleccionados, se mostrarán en verde.

Pseudoclases de activación

Por norma general, los elementos de un formulario HTML están siempre activados, aunquese pueden desactivar añadiendo el atributo **disabled** (*es un atributo booleano, no lleva valor*) al elemento HTML en cuestión. Esto es una práctica muy utilizada para impedir al usuario escribir en cierta parte de un formulario porque, por ejemplo, no es aplicable.

Existen varias pseudoclases para detectar si un campo de un formulario está activado odesactivado:

Pseudoclase	Descripción
:enabled	Aplica estilos cuando el campo del formulario está activado.
:disabled	Aplica estilos cuando el campo del formulario está desactivado.

:read-only	Aplica estilos cuando el campo es de sólo lectura.
:read-write	Aplica estilos cuando el campo es editable por el usuario.

Utilizando las dos primeras pseudoclases, bastante autoexplicativas por si solas, podemosseleccionar elementos que se encuentren activados (*comportamiento por defecto*) o desactivados:

```
/* Muestra en fondo blanco las casillas que permiten escribir */input:enabled {
background-color: white;
}

/* Muestra en fondo gris las casillas que no permiten escribir */input:disabled {
background-color: grey;
}
```

Por otro lado, las pseudoclases **read-only** y **read-write** nos permiten seleccionar y diferenciar elementos que se encuentran en modo de solo lectura (*tienen especificado elatributo **readonly** en el HTML*) o no:

```
input:read-only { background-color: darkred;color: white
}
```

En el ejemplo anterior, la pseudoclase **:read-only** le da estilo a aquellos campos **<input>** deun formulario que están marcados con el atributo de sólo lectura **readonly**. La diferencia entre un campo con atributo **disabled** y un campo con atributo **readonly** es que la información del campo con **readonly** se enviará a través del formulario, mientras que la del campo con **disabled** no se enviará. Aún así, ambas no permiten modificar el valor.

Por otro lado, la pseudoclase **:read-write** es muy útil para dar estilos a todos aquellos elementos que son editables por el usuario, sean campos de texto **<input>** o **<textarea>**.

```
input:read-write { background-color: green;color: white
}
```

Pseudoclases de validación

En HTML5 es posible dotar de capacidades de validación a los campos de un formulario, pudiendo interactuar desde Javascript o incluso desde CSS. Con estas

validaciones podemos asegurarnos de que el usuario escribe en un campo de un formulario el valor esperado que debería. Existen algunas pseudoclases útiles para las validaciones, como porejemplo las siguientes:

Pseudoclase	¿Cuándo aplica estilos?
:required	Cuando el campo es obligatorio, o sea, tiene el atributo **required**.
:optional	Cuando el campo es opcional (por defecto, todos los campos).
:invalid	Cuando los campos no cumplen la validación HTML5.
:valid	Cuando los campos cumplen la validación HTML5.
:out-of-range	Cuando los campos numéricos están fuera del rango.
:in-range	Cuando los campos numéricos están dentro del rango.

En un formulario HTML es posible establecer un campo obligatorio que será necesario rellenar para enviar el formulario. Por ejemplo, el DNI de una persona que va a matricularseen un curso, o el nombre de usuario de alta en una plataforma web para identificarse.
Campos que son absolutamente necesarios.

Para hacer obligatorios dichos campos, tenemos que indicar en el HTML el atributo **required**, al cual será posible darle estilo mediante la pseudoclase **:required**:

```
input:required {
border: 2px solid blue;
}
```

Por otra parte, los campos opcionales (*no obligatorios, sin el atributo **required***) pueden seleccionarse con la pseudoclase **:optional**:

```
input:optional {
border: 2px solid grey;
}
```

Las **validaciones en formularios HTML** siempre han sido un proceso tedioso, hasta la llegada de HTML5. HTML5 brinda un excelente soporte de validaciones desde el lado del cliente, pudiendo comprobar si los datos especificados son correctos o no antes de realizarlas validaciones en el lado del servidor, y **evitando la latencia** de enviar la información al servidor y recibirla de vuelta.

Pseudoclases de negación

Existe una pseudoclase muy útil, denominada **pseudoclase de negación**. Permite seleccionar todos los elementos que no cumplan los selectores indicados entre paréntesis.

Veamos un ejemplo:

```
p:not(.general) {
border: 1px solid #DDD;
padding: 8px; background: #FFF;
}
```

Este pequeño fragmento de código nos indica que todos los párrafos (*elementos **<p>***) queno pertenezcan a la clase **general**, se les aplique el estilo especificado.
Existen varias pseudoclases que permiten hacer referencias a los elementos del documento HTML según su **posición y estructura de los elementos hijos**. A continuación muestro unpequeño resumen de estas pseudoclases:

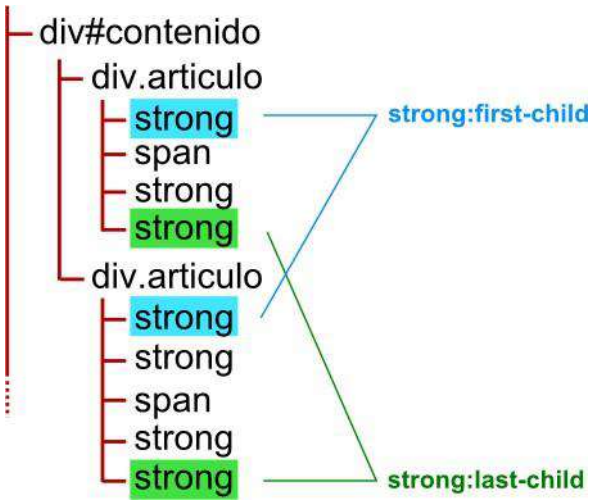
Pseudoclase	**Descripción**

:first-child	Primer elemento hijo (de cualquier tipo).
:last-child	Último elemento hijo (de cualquier tipo).
:nth-child(n)	N-elemento hijo (de cualquier tipo).
:nth-last-child(n)	N-elemento hijo (de cualquier tipo) partiendo desde el final.

Para ello, volvamos a utilizar una estructura en forma de árbol para ver cómodamente laubicación de cada uno de los elementos.

Las dos primeras pseudoclases, **:first-child** y **:last-child** hacen referencia a los primeros yúltimos elementos (*al mismo nivel*) respectivamente.

```
.articulo strong:first-child {background-color:cyan;
}

.articulo strong:last-child {background-color:green;
}
```

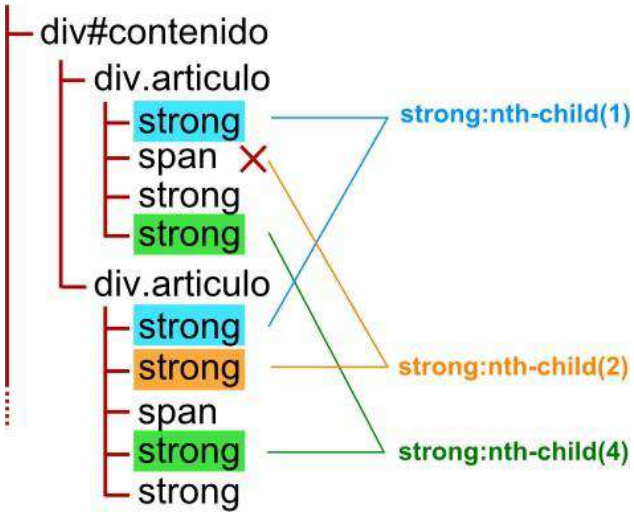


Sin embargo, si no queremos quedarnos en los primeros o últimos elementos y necesitamos más potencia para elegir, podemos hacer uso de la pseudoclase **:nth-child(A)**, que permite especificar el elemento deseado, simplemente estableciendo su número en el parámetro **A**:

Número	Equivalente a la pseudoclase	Significado
strong:nth-child(1)	**strong:first-child {}**	Primer elemento hijo, que además esun ****
strong:nth-child(2)		Segundo elemento hijo, que ademáses un ****
strong:nth-child(3)		Tercer elemento hijo, que además esun ****
strong:nth-child(n)		Todos los elementos hijos que son ****
strong:nth-child(2n)		Todos los elementos hijos pares ****
strong:nth-child(2n-1)		Todos los elementos hijos impares ****

:nth-child()

Veamos además un ejemplo gráfico:



Como se aprecia en el ejemplo, en el caso **:nth-child(2)** se puede ver como el segundo elemento lo ocupa un elemento **span**, por lo que sólo se selecciona el elemento **strong** delsegundo caso, donde si existe.

Elementos del mismo tipo

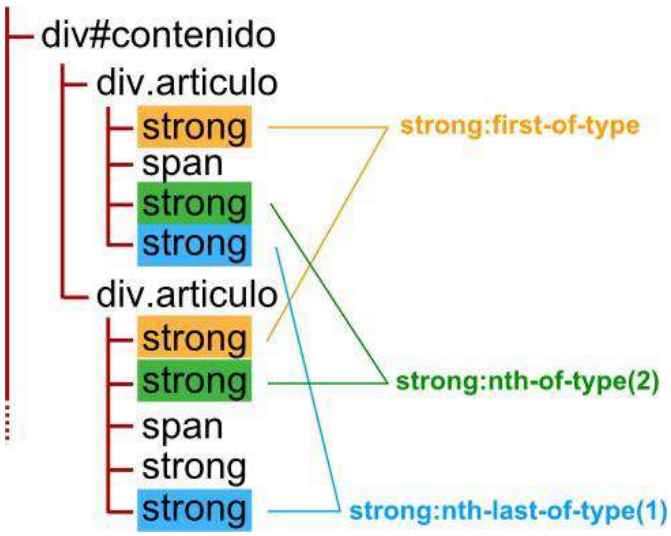
En los casos anteriores, seleccionamos elementos independientemente de que elemento sea. Simplemente, hacemos caso a la posición donde está ubicado. Si queremos hacer referencia sólo a elementos del mismo tipo, utilizaremos los selectores siguientes, análogosa los anteriores, pero haciendo referencia sólo a elementos del mismo tipo:

Pseudoclase	Descripción
:first-of-type	Primer elemento hijo (de su mismo tipo).
:last-of-type	Último elemento hijo (de su mismo tipo).

:nth-of-type(n)	N-elemento hijo (de su mismo tipo).
:nth-last-of-type(n)	N-elemento hijo (de su mismo tipo) partiendo desde el final.

Las pseudoclases **:first-of-type** y **:last-of-type** son las equivalentes a **:first-child** y **:last- child** pero sólo teniendo en cuenta elementos del mismo tipo. Por otro lado, la pseudoclase

:nth-of-type(A) es la equivalente a **:nth-child(A)** y **:nth-last-of-type(A)** es la equivalente a
:nth-last-child(A). Veamos un ejemplo sobre el ejercicio anterior:



En este ejemplo, se puede ver como **:nth-of-type(2)** selecciona el segundo elemento **strong** en ambos casos, a pesar de que en el primero ocupa la tercera posición. En este caso se selecciona porque es el segundo elemento de su mismo tipo (****). Por otrolado, **:nth-last-of-type(A)** hace una selección de forma inversa, empezando por el último elemento.

Hijos únicos

Existen también varias pseudoclases para la gestión de hijos únicos. Son las siguientes:

Pseudoclase	Descripción
:only-child	Elemento que es hijo único (de cualquier tipo).
:only-of-type	Elemento que es hijo único (de su mismo tipo).
:empty	Elemento vacío (sin hijos, ni texto).

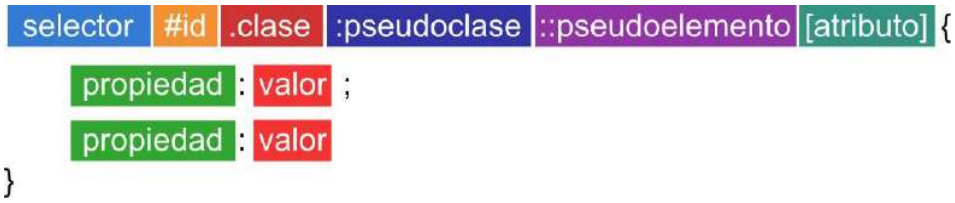
La propiedad **:only-child** nos proporciona un método para aplicar estilo a aquellos elementos que sean el único hijo de su elemento padre. Además, como ha ocurrido anteriormente, también existe la pseudoclase **:only-of-type** que es equivalente al anteriorpero sólo para elementos del mismo tipo, es decir, que puede ser que no sea el único elemento hijo, pero sí el único de su tipo.

Muy relacionada está también la pseudoclase **:empty**, que permite seleccionar los elementos que estén vacíos. Ojo con esto, ya que un elemento que contenga comentarios HTML **<!>** la pseudoclase **:empty** lo detectará como vacío, pero si contiene espacios en
blanco, no.

Pseudoelementos

Al igual que las pseudoclases, los pseudoelementos son otra de las características de CSSque permiten hacer referencias a «comportamientos virtuales no tangibles», o lo que es lo mismo, se le puede dar estilo a elementos que no existen realmente en el HTML, y que se pueden generar desde CSS.

Recordemos la sintaxis de los pseudoelementos, que está precedida de **dos puntos dobles** (**::**) para diferenciarlos de las pseudoclases, las cuales sólo tienen **dos puntos** (**:**). No obstante, este cambio surgió posteriormente, por lo que aún hoy en día es frecuente verfragmentos de código con pseudoelementos con la sintaxis de pseudoclase con **un solo par de puntos :**.



Generación de contenido

Dentro de la categoría de los **pseudoelementos CSS**, como punto central, se encuentra lapropiedad **content**. Esta propiedad se utiliza en selectores que incluyen los pseudoelementos **::before** o **::after**, para indicar que vamos a crear contenido antes o después del elemento en cuestión:

Propiedad/Pseudoelemento	Descripción
content	Propiedad para generar contenido. Sólo usable en **::before** o **::after**.
::before	Aplica los estilos **antes** del elemento indicado.
::after	Aplica los estilos **después** del elemento indicado.

La propiedad **content** admite parámetros de diverso tipo, incluso concatenando informaciónmediante espacios. Podemos utilizar tres tipos de contenido:

Valor	Descripción	Ejemplo

"*string*"	Añade el contenido de texto indicado.	**content: "Contenido:";**
attr(*atributo*)	Añade el valor del atributo HTML indicado.	**content: attr(href);**
url(*URL*)	Añade la imagen indicada en la **URL**.	**content: url(icon.png);**

Por otro lado, los pseudoelementos **::before** y **::after** permiten hacer referencia a «justo antes del elemento» y «justo después del elemento», respectivamente. Así, se podría generar información (*usualmente con fines decorativos*) que no existe en el HTML, pero quepor circunstancias de diseño sería apropiado colocar:

```
q::before { content: "«";color: #888;
}

q::before { content: "»";color: #888;
}
```

Los ejemplos anteriores insertan el carácter **«** antes de las citas indicadas con el elementoHTML **<q>** y el carácter **»** al finalizar la misma, ambas de color gris.

Atributos HTML

Es interesante recalcar la utilidad de la expresión **attr()**, que en lugar de generar el contenido textual que le indiquemos, permite recuperar esa información del valor del **atributo HTML** especificado. Veamos un ejemplo para clarificarlo, concatenándolo con texto:

```
a::after {
content: " ( " attr(href) " )";
}
```

Este pequeño ejemplo muestra a continuación de todos los enlaces la URL literalmente, dentro de dos paréntesis. Esto puede ser realmente útil en una página de estilos que se aplica a una página en el momento de imprimir, en los cuales se pierde

la información delenlace al no ser un medio interactivo.

Primera letra y primera línea

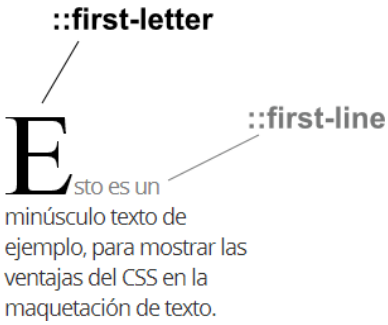
También existen pseudoelementos con los que podemos hacer referencia a la **primera letra**de un texto. Para ello utilizamos el pseudoelemento **::first-letter**, así como el pseudoelemento **::first-line** si queremos hacer referencia a la **primera línea** de un texto. Deesta forma, podemos dar estilo a esas secciones concretas del texto:

Pseudoelemento	Descripción
::first-letter	Aplica los estilos en la primera letra del texto.
::first-line	Aplica los estilos en la primera línea del texto.

Veamos un ejemplo en acción sobre un párrafo de texto:

```
p {
color: #333;
font-family: Verdana, sans-serif;font-size: 16px;
}

p::first-letter {color: black;
font-family: 'Times New Roman', serif;font-size: 42px;
}

p::first-line { color: #999;
}
```

Esto puede darnos la posibilidad de dar formato a un texto con ciertas propiedades, comocuentos clásicos:



Otros pseudoelementos

Existen otros pseudoelementos quizás menos conocidos:

Pseudoelemento	Descripción	
::backdrop	Aplica estilos al fondo exterior de la ventana de diálogo mostrada.	
::input- placeholder	Aplica estilos a los textos de sugerencia de los camposde entrada.	Soporte
::selection	Aplica estilos al fragmento de texto seleccionado por el usuario.	Soporte

Por último, una característica muy interesante de CSS es la posibilidad de aplicar estilos dependiendo de la existencia o el contenido de ciertos **atributos de los elementos HTML**.En CSS, estos atributos se rodean de corchetes **[ ]** y hay varias formas de utilizarlos, inspirados en un concepto llamado expresiones regulares:

Atributo	¿Cuándo se aplica el estilo?
[href]	Si el elemento tiene atributo **href**.
[href="#"]	Si el elemento tiene atributo **href** y su valor es **#**.
[href*="emezeta"]	Si el elemento tiene atributo **href** y su valor contiene **emezeta**.
[href^="https://"]	Si el elemento tiene atributo **href** y su valor comienza por **https://**.
[href$=".pdf"]	Si el elemento tiene atributo **href** y su valor termina por **.pdf** (unenlace a un PDF).
[class~="emezeta"]	Si el elemento tiene atributo **class** con una lista de valores y uno deellos es **emezeta**.
[lang|="es"]	Si el elemento tiene atributo **lang** con una lista de valores, dondealguno empieza por **es-**.

Atributo existente

Para empezar, podemos utilizar el atributo **[style]** para seleccionar todas las etiquetas HTML que contengan un atributo **style** para darles estilos en línea a un elemento. Estoselementos, aparecerían con fondo rojo:

```
[style] { background: red;
}
```

Este ejemplo es didáctico y no tiene finalidad práctica de diseño, ya que la idea sería mostrar visualmente que elementos tienen esa característica, algo que podría interesarle aun desarrollador. Si el elemento no tiene un atributo **style** definido, no se le aplican los estilos.

Atributo con valor exacto

Pero la potencia de los atributos en CSS es que podemos indicar el valor exacto que debentener para que sean seleccionados. Para ello, simplemente utilizamos el **=** y escribimos el texto entre comillas dobles:

```
a[rel="nofollow"] { background: red;
}
```

Este ejemplo selecciona los enlaces **<a>** que tienen un atributo **rel** establecido a **nofollow**. Esta es una característica que le indica a Google (*u otros robots o crawlers*) que ese enlace**no se debería tener en cuenta** para seguirlo, algo que puede ser realmente útil para desincentivar SPAM en comentarios, por ejemplo.

Atributo contiene texto

En lugar de un valor específico, también podemos querer indicar un fragmento de texto quedebe estar incluído, pero que no es el texto íntegro, casando con varias posibles coincidencias:

```
a[href*="emezeta"] { background-color: orange;
}
```

En la siguiente tabla se pueden ver varios ejemplos de enlaces, y cuáles se seleccionaríanen este caso:

De la misma forma, existe una variante que utiliza el comparador **~=**. Esta variante nos permitiría seleccionar los elementos HTML que tengan un atributo con una lista de palabrasseparadas por espacios, donde una de ellas es el texto que hemos escrito a continuación. Se trata de una versión más restrictiva del comparador ***=**.

Prefijos

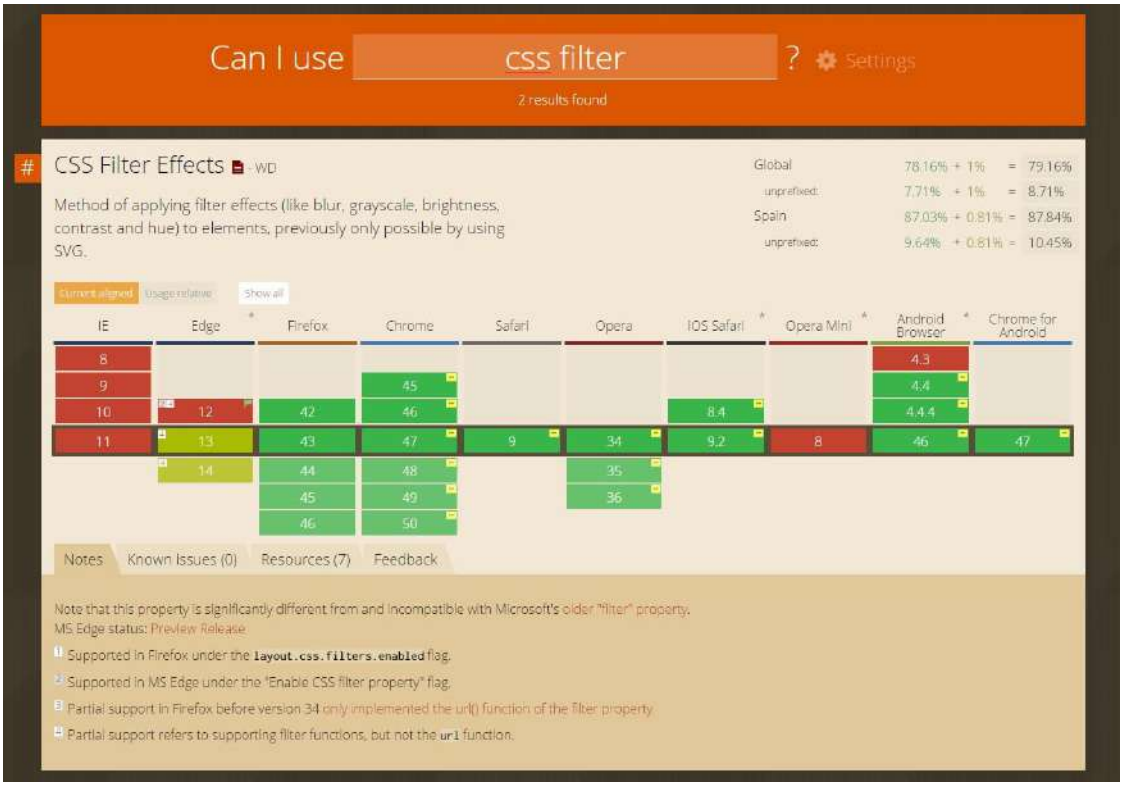
Algunas de las propiedades que veremos no están definidas por completo, sólo son borradores o pueden variar en la especificación definitiva, por lo que los navegadores las implementan utilizando una serie de**vendor prefixes** (*prefijos por navegador*), que facilitan la segmentación de funcionalidades.

De esta forma, podemos utilizar varios prefijos para asegurarnos que aunque dichas funcionalidades tengan un comportamiento o sintaxis diferente en cada navegador, podemos hacer referencia a cada unade ellas por separado:

```
div {

transform: ... /* Navegadores que implementan especificación oficial */

-webkit-transform: ... /* Versiones antiguas de Chrome (Motor WebKit) */

-moz-transform: ... /* Versiones antiguas de Firefox (Motor Gecko) */

-ms-transform: ... /* Versiones antiguas de IE (Motor Trident) */

-o-transform: ... /* Versiones antiguas de Opera (Motor Presto) */

}
```

En el ejemplo anterior, la propiedad **transform** se refiere a los navegadores que tengan implementada laespecificación definitiva por completo e ignorará el resto de propiedades. Por otro lado, otro navegador (*o el mismo en una versión más antigua*) puede tener implementada una versión anterior a la definitiva, por lo que hará caso a las propiedades con un prefijo concreto.

Debido al ritmo y la rápida velocidad de Internet en estas cuestiones, es muy complicado obtener una lista de funcionalidades implementadas en cada navegador, algo que puede variar incluso en cuestión desemanas. Aconsejo utilizar la página Can I Use, una web colaborativa para saber el estado actual, previoe incluso futuro de las propiedades CSS o elementos HTML en cada navegador.



En esta página se puede buscar, a través de un buscador y de forma rápida y cómoda, el estado de ciertascaracterísticas por parte de las diferentes versiones de los navegadores.

Actualmente, los **vendor prefixes** están en proceso de desaparecer. Las principales compañías de navegadores han optado por favorecer el uso de flags en el navegador del usuario para activar o desactivar opciones experimentales o crear especificaciones más pequeñas y breves que puedan ser estables mucho más rápido. Por esta razón, se aconseja utilizar vendor prefixes solo cuando necesitas soporte específico en navegadores muy antiguos.

En el caso de querer utilizar *vendor prefixes**, recomiendo encarecidamente utilizar sistemas como autoprefixer (*el más popular, que forma parte de PostCSS*) o prefix-free que añaden de forma automática y transparente los prefijos, basándose en información de herramientas como Can I Use. Busca extensiones en el **editor** que utilices o la opción para activarlas, ya que te ahorrará mucho tiempoy te permitirá tener un código más legible y modular al no tener que repetir código.

Transiciones CSS

En CSS aparecen uno de los aspectos más interesantes de la web interactiva: **las transiciones**. En versiones anteriores de CSS sólo se podían utilizar ciertas funcionalidades interactivas con pseudoclases como **:hover** o **:focus**. Sin embargo, dichastransiciones ocurrían de golpe, pasando de un estado inicial a otro final. Mediante las transiciones, tenemos a nuestra disposición una gran flexibilidad que nos permitirá dotar deatractivos y elegantes efectos de transición que multiplicarán por mil las posibilidades de nuestros diseños.

Las **transiciones** se basan en un principio muy básico, conseguir un efecto suavizado entreun estado inicial y un estado final. Las propiedades relacionadas que existen son las siguientes:

Propiedades	**Valor**
transition-property	**all** | none | *propiedad css*
transition-duration	**0** |
transition-timing-function	**ease** | linear | ease-in | ease-out | ease-in-out | cubic-bezier(*A*, *B*, *C*, *D*)
transition-delay	**0** |

En primer lugar, la propiedad **transition-property** se utiliza para especificar la **propiedad ala que que afectará la transición**. Podemos especificar la propiedad concreta (***width*** *o* ***color****, por ejemplo*) o simplemente especificar **all** para que se aplique a todos los elementoscon los que se encuentre. Por otro lado, **none** hace que no se aplique ninguna transición.

Con la propiedad **transition-duration** especificaremos la **duración de la transición**, desdeel inicio de la transición, hasta su finalización. Se recomienda siempre comenzar con valores cortos, para que las transiciones sean rápidas y elegantes.

Si establecemos una duración demasiado grande, el navegador realizará la transición con detención intermitentes, lo que hará que la transición vaya a golpes. Además, transicionesmuy largas pueden resultar molestas a muchos usuarios.

Función de tiempo

La propiedad **transition-timing-function** permite indicar el **ritmo de la transición** que queremos conseguir. Cuando estamos aprendiendo CSS, recomiendo utilizar **linear**, querealiza una transición a un ritmo constante. Sin embargo, podemos utilizar otros valores para conseguir que el ritmo sea diferente al inicio y/o al final de la transición.

Los valores que puede tomar la propiedad son los siguientes:

Valor	Inicio	Transcurso	Final	Equivalente en cubic-bezier
ease	Lento	Rápido	Lento	(0.25, 0.1, 0.25, 1)
linear	Normal	Normal	Normal	(0, 0, 1, 1)
ease-in	Lento	Normal	Normal	(0.42, 0, 1, 1)

ease-out	Normal	Normal	Lento	(0, 0, 0.58, 1)
ease-in-out	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
cubic-bezier(*A*, *B*, *C*, *D*)	-	-	-	Transición personalizada

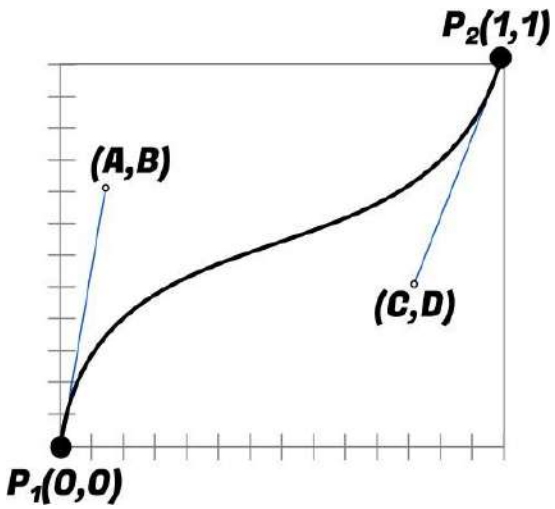
Una función de tiempo **linear** siempre es constante, mientras que **ease** comienza suavemente, continua de forma más rápida y termina suavemente de nuevo. **Ease-in** y **ease-out** son variaciones que van más lento al principio o al final, y **ease-in-out** una mezclade las dos.

Cubic-Bezier()

La función de tiempo **cubic-bezier()** es una función personalizada, donde podemos darle unos valores concretos depediendo de la velocidad que queramos que tenga la transición.En la última columna de la tabla anterior podemos ver los valores equivalentes a cada unade las palabras clave mencionadas. En principio, el formato de la función es **cubic- bezier(A, B, C, D)**, donde:

A	X_1	Eje X del primer punto que orienta la curvabezier.	P_1
B	Y_1	Eje Y del primer punto que orienta la curvabezier.	P_1

C	X_2	Eje X del segundo punto que orienta la curvabezier.	P_2
D	Y_2	Eje Y del segundo punto que orienta la curvabezier.	P_2



También puedes utilizar la página Cubic Bezier, donde puedes ver de forma interactiva lavelocidad de las transiciones dependiendo de los parámetros utilizados.

Por último, la propiedad **transition-delay** nos ofrece la posibilidad de **retrasar el inicio dela transición** los segundos especificados.

Veamos un pequeño ejemplo de todo ello:

```
a {
background: #DDD;color: #222; padding: 2px;
border: 1px solid #AAA;
}

a:hover { background: #FFF;color: #666; padding: 8px 14px;
border: 1px solid #888;

transition-property: all; transition-duration: 0.2s;
transition-timing-function: ease-in;
}
```

Atajo: Transiciones

Como siempre, podemos resumir todas estas operaciones en una propiedad de atajo denominada **transition**. Los valores del ejemplo superior, se podrían escribir como se puede ver a continuación (*si no necesitas algún valor, se puede omitir*):

```
div {
/* transition: <property> <duration> <timing-function> <delay> */transition: all 0.2s ease-in;
}
```

Animaciones CSS

Una vez conocemos las transiciones CSS, es muy fácil adaptarnos al concepto de animaciones de CSS, el cual amplía el concepto de transiciones convirtiéndolo en algo mucho más flexible y potente.

Las transiciones son una manera de suavizar un cambio de un estado inicial a un estado final. La idea de las animaciones CSS parte del mismo concepto, permitiendo añadir más estados, pudiendo realizar cambios desde un estado inicial, a un estado posterior, a otro estado posterior, y así sucesivamente. Además, esto será posible de forma automática, sinque el usuario tenga que realizar una acción concreta.

El primer paso para crear animaciones es tener dos cosas claras. Por un lado, utilizaremos la regla **@keyframes**, que incluye los fotogramas de la animación. Por otro lado, tendremosque utilizar las propiedades de las **animaciones**, que definen el comportamiento de la misma.

Propiedades de animación CSS

Para definir dicho comportamiento necesitamos conocer las siguientes propiedades, queson una ampliación de las transiciones CSS:

Propiedades	Valor

animation-name	**none** | *nombre*
animation-duration	**0** |
animation-timing-function	**ease** | linear | ease-in | ease-out | ease-in-out | cubic-bezier(*A*, *B*, *C*, *D*)
animation-delay	**0** |
animation-iteration-count	**1** | infinite |
animation-direction	**normal** | reverse | alternate | alternate-reverse
animation-fill-mode	**none** | forwards | backwards | both
animation-play-state	**running** | paused

La propiedad **animation-name** permite especificar el nombre del fotograma a utilizar, mientras que las propiedades **animation-duration**, **animation-timing-function** y **animation-delay** funcionan exactamente igual que en el tema anterior de **transiciones**.

La propiedad **animation-iteration-count** permite indicar el número de veces que se repite la animación, pudiendo establecer un número concreto de repeticiones o indicando **infinite**para que se repita continuamente. Por otra parte, especificando un valor en **animation- direction** conseguiremos indicar el orden en el que se reproducirán los fotogramas, pudiendo escoger un valor de los siguientes:

Valor	Significado
normal	Los fotogramas se reproducen desde el principio al final.
reverse	Los fotogramas se reproducen desde el final al principio.
alternate	En iteraciones par, de forma normal. Impares, a la inversa.
alternate-reverse	En iteraciones impares, de forma normal. Pares, normal.

Por defecto, cuando se termina una animación que se ha indicado que se reproduzca sólo una vez, la animación vuelve a su estado inicial (*primer fotograma*). Mediante la propiedad**animation-fill-mode** podemos indicar que debe mostrar la animación cuando ha finalizadoy ya no se está reproduciendo; si mostrar el estado inicial (*backwards*), el estado final (*forwards*) o una combinación de ambas (*both*).

Por último, la propiedad **animation-play-state** nos permite establecer la animación a estadode reproducción (*running*) o pausarla (*paused*).

Atajo: Animaciones

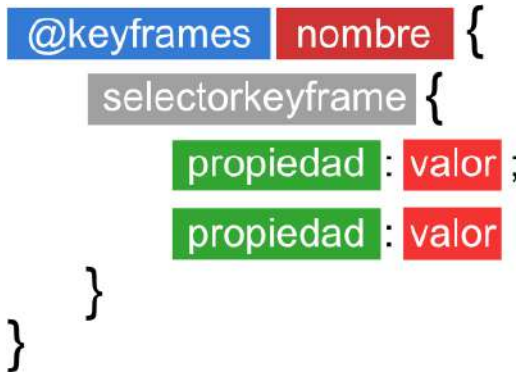
Nuevamente, CSS ofrece la posibilidad de resumir todas estas propiedades en una

sola,para hacer nuestras hojas de estilos más específicas. El orden de la propiedad de atajo sería el siguiente:

```
div {
/* animation: <name> <duration> <timing-function> <delay>
<iteration-count> <direction> <fill-mode> <play-state> */ animation: changeColor 5s linear 0.5s
4 normal forwards running;
}
```

Fotogramas (keyframes)

Ya sabemos como indicar a ciertas etiquetas HTML que reproduzcan una animación, con ciertas propiedades. Sin embargo, nos falta la parte más importante: definir los fotogramas de dicha animación. Para ello utilizaremos la regla **@keyframes**, la cuál es muy sencilla deutilizar y se basa en el siguiente esquema:



En primer lugar elegiremos un **nombre** para la animación (*el cuál utilizamos en el apartadoanterior, para hacer referencia a la animación, ya que podemos tener varias en una mismapágina*), mientras que podremos utilizar varios selectores para definir el transcurso de los fotogramas en la animación.

Veamos algunos ejemplos:

```
@keyframes changeColor {
from { background: red; } /* Primer fotograma */to { background: green; }  /* Último fotograma */
}

.anim { background: grey;color: #FFF; width: 150px; height: 150px;
animation: changeColor 2s ease 0 infinite;  /* Relaciona con @keyframes */
}
```

En este ejemplo nombrado **changeColor**, partimos de un primer fotograma en el que el elemento en cuestión será de color de fondo rojo. Si observamos el último

fotograma, le ordenamos que termine con el color de fondo verde. Así pues, la regla **@keyframes** se inventará la animación intermedia para conseguir que el elemento cambie de color.

Los selectores **from** y **to** son realmente sinónimos de **0%** y **100%**, así que los modificaremos y de esta forma podremos ir añadiendo nuevos fotogramas intermedios.

Vamos a modificar el ejemplo anterior añadiendo un fotograma intermedio e indentando,ahora sí, correctamente el código:

```
@keyframes changeColor {0% {
background: red;              /* Primer fotograma */
} 50% {
background: yellow;           /* Segundo fotograma */width: 400px;
} 100% {
background: green;            /* Último fotograma */
}
}

.anim { background: grey;color: #FFF; width: 150px; height: 150px;
animation: changeColor 2s ease 0 infinite;  /* Relaciona con @keyframes */
}
```

Encadenar animaciones

Es posible encadenar múltiples animaciones, separando con comas las animaciones individuales y estableciendo un tiempo de retardo a cada animación posterior:

```
.animated {animation:
moveRight 5s linear 0,          /* Comienza a los 0s */lookUp 2.5s linear 5s,          /* Comienza a los 5s */
moveLeft 5s linear 7.5s, /* Comienza a los 7.5s (5 + 2.5) */ disappear 2s linear 9.5s; /* Comienza a los 9.5s (5 + 2.5 + 2) */
}
```

En este caso, lo que hemos hecho es aplicar varias animaciones a la vez, pero estableciendo un retardo (*cuarto parámetro*) que es la suma de la duración de las animaciones anteriores. De esta forma, encadenamos una animación con otra.

Transformaciones

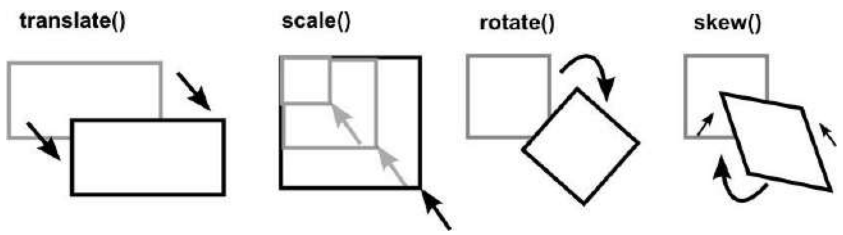
Las transformaciones es uno de los elementos más interesantes que se introducen en CSS3 para convertir el lenguaje de hojas de estilo en un sistema capaz de realizar todo tipode efectos visuales, incluido 2D y 3D. Las propiedades principales para realizar transformaciones son las siguientes:

Propiedades	Formato	Significado
transform	*función1*, *función2*, ...	Aplica una o varias funciones de transformación sobre un elemento.
transform-origin		Cambia el punto de origen del elemento en una transformación.
transform-style	**flat** | preserve-3d	Modifica el tratamiento de los elementos hijos.

Comencemos por la propiedad **transform**, mediante la cual podemos indicar una o varias transformaciones para realizar sobre un elemento, ya sean 2D (*sobre dos ejes*) o 3D (*sobretres ejes*).

Funciones 2D

Existen múltiples propiedades CSS que ofrecen diferentes funcionalidades de transformación en dos dimensiones, que veremos a continuación:



Translaciones

Las **funciones de translación** son aquellas que realizan una transformación en la que **mueven** un elemento de un lugar a otro. Si especificamos un valor positivo en el eje X

(*horizontal*), lo moveremos hacia la derecha, y si especificamos un valor negativo, lo moveremos hacia la izquierda. Lo mismo con el eje Y (*vertical*):

Funciones	Significado
translateX(x)	Traslada el elemento una distancia de *x* horizontalmente.
translateY(y)	Traslada el elemento una distancia de *y* verticalmente.
translate(x, y)	Propiedad de **atajo** de las dos anteriores.

Por ejemplo, **transform: translate(20px, -30px)** traslada el elemento 20 píxeles a la derecha y 30 píxeles hacia arriba, que es equivalente a utilizar **transform: translateX(20px)translateY(-30px)**.

Escalado

Las **funciones de escalado** realizan una transformación en la que aumentan o reducen el tamaño de un elemento, basándose en el parámetro indicado, que no es más que un factorde escala:

Funciones	Significado
scaleX(fx)	Reescala el elemento a un nuevo tamaño con un factor *fx* horizontal.

scaleY(fy)	Reescala el elemento a un nuevo tamaño con un factor *fy* vertical.
scale(fx, fy)	Propiedad de **atajo** de las dos anteriores.

En este ejemplo, **transform: scale(2, 2)** realiza una transformación de escalado del elemento, ampliándolo al doble de su tamaño original. Si utilizamos **scale()** con dos parámetros iguales, estamos manteniendo la proporción del elemento, pero si utilizamosdiferentes valores, acabaría deformándose.

Rotaciones

Las **funciones de rotación** simplemente giran el elemento el número de grados indicado:

Funciones	Significado
rotateX(xdeg)	Establece una rotación 2D en *xdeg* grados sólo para el eje horizontal X.
rotateY(ydeg)	Establece una rotación 2D en *ydeg* grados sólo para el eje vertical Y.
rotate(deg)	Establece una rotación 2D en *deg* grados sobre si mismo.

Con **transform: rotate(5deg)** realizamos una rotación de 5 grados del elemento sobre

si mismo. Utilizando **rotateX()** y **rotateY()** podemos hacer lo mismo respecto al eje X o el ejeY respectivamente.

Deformaciones

Por último, las **funciones de deformación** establecen un ángulo para torcer, tumbar o inclinar un elemento en 2D:

Funciones	Significado
skewX(xdeg)	Establece un ángulo de *xdeg* para una deformación 2D respecto al eje X
skewY(ydeg)	Establece un ángulo de *ydeg* para una deformación 2D respecto al eje Y

Aunque la función **skew()** existe, no debería ser utilizada, ya que está marcada como obsoleta y serán retiradas de los navegadores en el futuro. En su lugar deberían utilizarse**skewX()** o **skewY()**.

Animaciones con CSS

En algunas ocasiones necesitamos realizar animaciones en nuestro sitio o proyecto web de una manera rápida y sencilla, para esto les presento a Animate.css, una **librería CSS3 para crear animaciones** fácilmente.

Aprenderemos a trabajar con ella, primero descargamos la librería CSS3 del siguiente link: https://animate.style/ , dando click en el enlace “Download Animate.css”.

Una vez descargado, crearemos nuestra estructura básica en HTML y agregaremos un enlace hacia el archivo “animate.min.css” que descargamos anteriormente.

Una vez hecho esto ya podremos mostrar diferentes tipos de animaciones en nuestra página web, para esto, dentro del elemento afectado debemos agregar algunos valores en la propiedad “class” como los siguientes:

- animated: Para identificar el elemento a animar.
- infinite: Para que la animación siga un bucle infinito.
- animación (Ej. shake): La animación como tal, la animación “shake” puede reemplazarse por cualquiera de las más de 70 animaciones que nos ofrece la librería.

De esta manera hemos visto que con el uso de esta librería podemos crear animaciones sin necesidad de JavaScript o conocimientos de CSS3, lo recomendable es aprender CSS3 y conocer el funcionamiento de estas animaciones por un tema de formación, optimización y para crear nuestras propias animaciones.

Más información en: https://animate.style/

Introducción Responsive Web Design

Hasta hace algunos años era imprescindible utilizar el ordenador para navegar por internet; ahora en cambio, es muy probable que la mayoría de accesos se realicen desde plataformas mobile o móviles. Hoy en día todos llevamos un smartphone encima y nos comunicamos y buscamos información constantemente, por lo que se ha convertido en algo esencial optimizar los sitios web para un buen uso en estos tipos de dispositivos.

Si no sabes si tu sitio Web está optimizado para móviles, existe una herramienta online para que puedas comprobarlo.

Solo tienes que en ella (https://search.google.com/test/mobile-friendly?utm_source=mft&utm_medium=redirect&utm_campaign=mft-redirect&hl=es) e introducir la URL principal de tu sitio Web; en pocos segundos, te indicará si la página tiene un diseño optimizado o no.

De manera general podemos distinguir dos formas de optimización: “Responsive Web Design” y “Mobile First Web”.

¿En qué se diferencian Responsive Web Design y Mobile First Web?

Aunque van de la mano, hay que diferenciar entre Responsive Web Design o, lo que viene a ser lo mismo, Diseño Web Adaptativo, y Mobile First Web. Como su propio nombre indica, el Diseño Web Adaptativo es aquel capaz de adaptarse a diferentes tamaños y dispositivos, es decir, dependiendo de qué dispositivo sea en el que se cargue, tu sitio web se verá más accesible y fácil de usar. Sin embargo, lo que propone el término Mobile First es empezar a diseñar un sitio web desde la resolución más pequeña para ir creciendo y adaptando el contenido y el diseño a la resolución más grande.

Hasta ahora todos los sitios web han sido diseñados solo para equipos de sobremesa y el proceso de navegar por las webs en los teléfonos móviles era bastante incómodo, debido a una mala experiencia de usuario. Sin embargo, las tecnologías están cambiando y el principio de Mobile First se está convirtiendo en un concepto cada vez más extendido.

Responsive vs. Mobile first

¿Qué conseguimos con el principio Mobile First?
Si empezamos maquetando un sitio web para la versión de escritorio y un usuario se conecta desde un dispositivo, primero cargará todo el contenido utilizado en la primera versión, hasta cargar los recursos necesarios para móvil. Por lo tanto, lo más recomendable es empezar a maquetar para la versión más pequeña, siempre optimizando el contenido que se utilice (hojas de estilos, ficheros, imágenes…), así conseguiremos que el usuario no cargue más recursos de los necesarios, reduciendo el tiempo de carga del sitio web.

La clave es conocer el tamaño, resolución o posibles orientaciones de las pantallas en las que necesitamos mostrar nuestro contenido basándonos en los usuarios que tenemos como objetivo.

Los pilares principales del Responsive son las Media Queries y la etiqueta Viewport.

Media Queries

Las Media Queries son las herramientas fundamentales que se encargan de aplicar diferentes estilos para diferentes dispositivos, y proporcionan la mejor experiencia para cada tipo de usuario que se encuentra navegando en tu sitio web. Nacen de la necesidad de crear breakpoints o puntos de ruptura en la hoja de estilos CSS que tengas predefinida. Permite que tu sitio Web sea manejable desde diferentes dispositivos.

Breakpoints

Si no te ha quedado muy claro, las Media Queries son un módulo de CSS que sirve para detectar el tipo de dispositivo por el que se está navegando; de esa manera el contenido consigue adaptarse al dispositivo concreto a través de las distintas condiciones que tú mismo asignas, como pueden ser ancho y alto de la ventana del navegador, ancho y alto del dispositivo, la resolución del dispositivo o la orientación de la pantalla. Son declaraciones lógicas que actúan dependiendo de las condiciones específicas que tú mismo declaras en la hoja de estilos. Si la premisa se cumple, se aplicaran los estilos definidos; si no, los omitirá por completo.

Hay dos formas de implementarlas:

La primera opción para poner en funcionamiento las Media Queries es a través del atributo media de la etiqueta <link>. Como sabemos, esta etiqueta es la que se usa para enlazar una hoja de estilo con un documento HTML. En ese enlace podemos especificar condiciones que deben cumplirse para que los estilos enlazados se apliquen. Debería ir dentro del <head> de nuestro HTML.

Recuerda que la etiqueta <link> tiene esta forma:

<link rel="stylesheet" href="estilos.css">

Pues ahora simplemente agregamos el atributo media indicando la condición que se debe cumplir para que estos estilos se apliquen:

<link rel="stylesheet" media="only screen and (max-width: 768px)" href="estilos.css">

Lo que concretamente le estamos indicando es que cargue la hoja de estilos “estilos.css” si se cumple que el dispositivo de salida es una pantalla, no una impresora u otro dispositivo (only screen), y si la anchura de la ventana del navegador tiene de máximo 768 píxeles (max-width: 768px).

Si se cumplen las condiciones, los estilos se mostraran correctamente, en caso contrario, los estilos se omiten por completo, y el contenido se muestra sin estilos definidos:

Carga de hojas de estilo

Cargar de esta manera las Media Queries supone un problema, y es que cada vez que queramos cargar diferentes estilos dependiendo de ciertas condiciones que queramos aplicar para distintos dispositivos, tendríamos que cargar una hoja de estilos nueva. Esto conlleva una carga más lenta de tu sitio web, ya que se hacen solicitudes HTTP adicionales, que se podrían evitar.

Hay otro sistema más recomendable para aplicar las Media Queries: basta con incluir todas las condiciones necesarias dentro de un único archivo CSS. Así, incorporaríamos la construcción @media seguido de las condiciones que queremos definir para cada tipo de dispositivo y donde se apliquen entre llaves { } los estilos concretos para cada uno de ellos. Es la manera más aconsejable, ya que la carga es de un único archivo CSS.

La forma de incluir Media Queries dentro de la hoja de estilos CSS es la siguiente:

```
@media (max-width:320px){
  <!—- Aquí van todos los estilos CSS -->
}
```

Esta Media Query se ejecutará sólo cuando la anchura de la ventana del navegador sea menor de 320 píxeles.

```
@media (min-width:768px){
  <!—- Aquí van todos los estilos CSS -->
}
```

Esta Media Query se ejecutará sólo cuando la anchura de la ventana del navegador sea mayor de 768 píxeles.

Además de las características para determinar las resoluciones y anchos de pantalla, podemos determinar otros parámetros, como por ejemplo la orientación del dispositivo, importante en dispositivos móviles:

```
@media (orientation: landscape) {
  <!-- Aquí van todos los estilos CSS -->
}
```

Portrait: orientación vertical
Landscape: orientación horizontal
Portrait vs. Landscape

Operadores lógicos para las Media Queries

También se pueden combinar más de una condición en la misma Media Query para concretar todavía más un rango de resolución, mediante los operadores lógicos:

Operador and: las dos condiciones deben cumplirse para que se apliquen los estilos.
Operador not: es una negación de una condición. Cuando esta condición no se cumpla, se aplicarán las media queries definidas.
Operador only: se aplican las reglas solo en el caso de que se cumpla.
Operador or: se pueden poner varias condiciones separadas por comas y en el momento que se cumpla cualquiera de ellas, se aplicarán los estilos.

```
@media only screen and (min-width:320px) and (max-width:480px){
<!—- Aquí van todos los estilos CSS -->
}
```

Para esta Media Querie se mostrarán los estilos CSS cuando la anchura de la ventana del navegador sea entre 320 pixeles y 480 pixeles, ambos incluidos.

Estos son algunos de los parámetros generales que se pueden emplear a la hora de construir las condiciones en las Media Queries:

width: anchura de la ventana del navegador.
height: altura de la ventana del navegador.
device-width: anchura de la resolución de pantalla.
device-height: altura de la resolución de pantalla.
orientation (portrait/landscape): dispositivo en horizontal o en vertical.
resolution: densidad de píxeles.
Excepto la orientación, el resto de parámetros admiten los valores “max” y “min”.

max-width: La anchura será menor que la indicada.
min-width: La anchura será mayor que la indicada.

Algo a tener en cuenta a la hora de utilizar las Media Queries, es diferenciar entre el ancho de ventana del navegador y la resolución de la pantalla del dispositivo, es decir:

```
@media only screen and (min-device-width: 960px){
  /* Aquí van todos los estilos CSS */
}
```

En esta Media Querie que hemos definido, el atributo min-device-width se refiere a la resolución de la pantalla del dispositivo a la hora de cargar la hoja de estilos definida.

atributo min-device-width

Esto quiere decir que si reducimos el ancho del navegador, seguirá mostrándose de la misma manera, ya que la resolución de la pantalla seguirá siendo la misma y no se adaptara al nuevo ancho de la ventana del navegador (es decir, si la pantalla de nuestro móvil tiene 450 px y el navegador detecta que lo óptimo sería visualizarla con 600 px así lo hará si no usamos la meta-etiqueta Viewport).

En caso de usar los atributos para la resolución de la pantalla, la etiqueta Viewport es necesaria.

¿Qué hace la meta-etiqueta Viewport?

El Viewport es el área visual donde se plasma el contenido del documento HTML de tu sitio web. Se podría traducir como vista o ventana y nos sirve para definir qué área de pantalla está disponible al renderizar un documento, la escala/zoom que debe mostrar inicialmente. Todo ello, con parámetros que le damos a la propia etiqueta meta, separados por comas en caso de utilizar más de uno.

Prácticamente todos los navegadores de smartphones al entrar a un sitio analizan el tamaño total y lo escalan para que se muestre completo en la pantalla, este procedimiento genera muchas veces resultados inapropiados.

Por ejemplo, esta imagen mide 320 píxeles al igual que la pantalla del dispositivo, ahora bien, la imagen aparece con un tamaño inferior a causa del efecto de la escala automática.

Sin etiqueta Viewport

La escala automática se puede evitar y controlar muy fácil con el uso de este atributo Viewport: es un atributo del tag <meta> que debe incluirse entre las etiquetas <head> del documento HTML de tu sitio web:

<meta name="viewport" content="width=device-width"/>

Con solo agregar estas líneas de código, la imagen se adaptará al dispositivo:

Con etiqueta Viewport

Es posible definir un tamaño específico para el área visible del documento; muchos sitios web ajustan directamente el Viewport a 320 px para ajustar la apariencia al display vertical de un smartphone, usando un código similar a este:

<meta name="viewport" content="width=320"/>

Pero, con los diferentes equipos, dispositivos y tamaños de pantalla, definir un tamaño específico puede ser una mala práctica que puede mostrar resultados erróneos en algunos equipos o cuando el dispositivo cambia de posición. Afortunadamente podemos configurar el viewport para ajustarse

dinámicamente al tamaño de cada dispositivo usando el atributo “device-width”, que es equivalente al 100% del ancho de la pantalla del dispositivo, independiente de su tamaño, posición o resolución:

<meta name="viewport" content="width=device-width"/>

El alto de la pantalla también es configurable con las mismas propiedades a través del atributo “height” , aunque –salvo condiciones muy específicas– no es necesario definirlo. Esta propiedad se asignará automáticamente a través del scroll.

También podemos controlar la escala de la vista con el atributo “initial-scale“. El sitio se mostrará al doble de su tamaño original:

<meta name=”viewport” content=”width=device-width; initial-scale=2“/>

Atributo “initial-scale“

Es posible además, limitar el tamaño al que se puede escalar el sitio con el atributo “maximum-scale” . El siguiente ejemplo muestra el documento en escala correcta y permite ampliar (zoom) hasta al doble de su tamaño.

<meta name="viewport" content="width=device-width, maximum-scale=2"/>

Por último, está el atributo “user-scalable”, que controla los permisos de reducir/ampliar el documento. Con el siguiente código, el sitio se muestra en su escala original y no es posible cambiar el tamaño desde el dispositivo móvil (importante mencionar que no se recomienda deshabilitar la opción de escalar el contenido).

<meta name="viewport" content="width=device-width, user-scalable=no"/>

En general, el atributo viewport permite muchas configuraciones, pero para asegurar compatibilidad con la mayor cantidad de pantallas y navegadores móviles, se recomienda utilizar este formato como base:

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

Web optimizada para los diferentes anchos de pantalla

Para conseguir que nuestro sitio web se adapte a los diferentes anchos de pantalla, estos parámetros serán muy útiles:

Lo primero, y lo más importante es dejar de usar píxeles y usar porcentajes a la hora de tomar medidas (por ejemplo: width: 60%).

Que el ancho de la página sea 100% no significa que queramos que la pantalla este en una alta resolución, sino que, si queremos limitar el ancho/alto junto al máximo/mínimo del contenido, debemos usar los diferentes parámetros

apropiados para ello (max-width o si quisiésemos establecer un alto máximo max-height; para establecer el mínimo sería min-width y min-height).

Las posiciones absolutas o fijas no son recomendables usarlas para posicionar contenido (menos cuando hagan falta). Lo mejor es utilizar el atributo float (float:left/right), es una técnica muy usada.
Hay que hacer que las imágenes y vídeos no sobresalgan de la estructura; si no, aparecerá un scroll lateral en los dispositivos móviles que descolocará totalmente el diseño.

En resumen, ¿cuál es la mejor opción para tu sitio web? La experiencia del usuario siempre será lo primero.

A los usuarios no les importa que versión utilices ni como estés optimizando tu sitio web; su objetivo es poder encontrar lo que buscan de manera más eficiente y rápida. Por tanto, si tu web está más centrada en el contenido, es mejor una Responsive Web Design. Pero si necesitas que el usuario interaccione mucho con la web, es mejor una Mobile First, ya que cada vez se consume más información desde los dispositivos móviles.

La conclusión es sencilla, los clientes están yendo más rápido que las propias empresas y estas deben adaptarse a ellos y a sus nuevas costumbres de consumo online a través de dispositivos. Esto es una solución para reducir la tasa de rebote, haciendo que el usuario pase más tiempo en la página por su facilidad, comodidad, y óptima visualización y lectura de los contenidos.

Es de destacar que actualmente Google valora todas aquellas páginas web que se adaptan perfectamente a cualquier dispositivo, ya sea PC, smartphone, tablet… por ello, es necesario optimizar un sitio web de modo que cualquier usuario pueda visualizar la página sin importar el medio por el cual acceda.