Selectores avanzados en CSS

Fuente: lenguajecss.com

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 elementos dependiendo 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 tardará más 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;
}
```

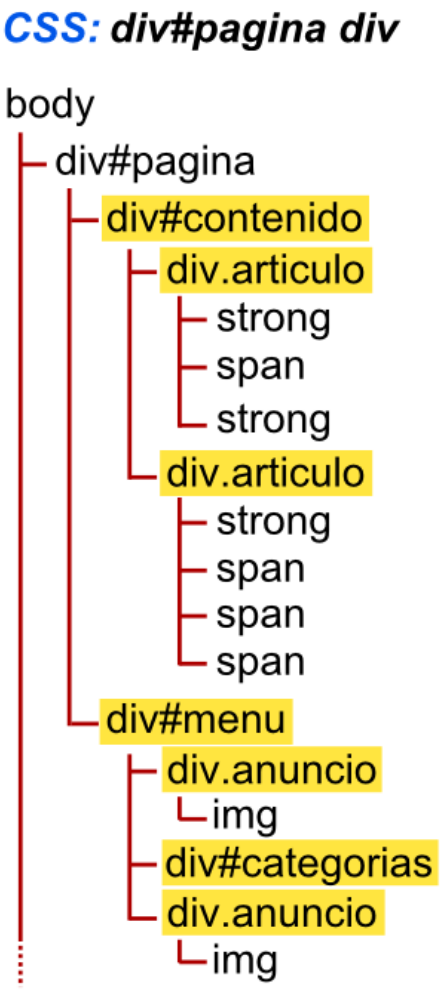
Selector descendiente

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 entiende bien.

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

```
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 un elemento ** ** fuera del ** ** con id **pagina**, no se aplicarán los estilos indicados:



Repasemos varios detalles importantes respecto a este combinador CSS:

- Se están seleccionando todos los elementos ** ** que están dentro de ** ** con ID **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 existan en un selector, más complejo será el procesamiento de dicha regla por los navegadores. Lo recomendable 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;
}
```

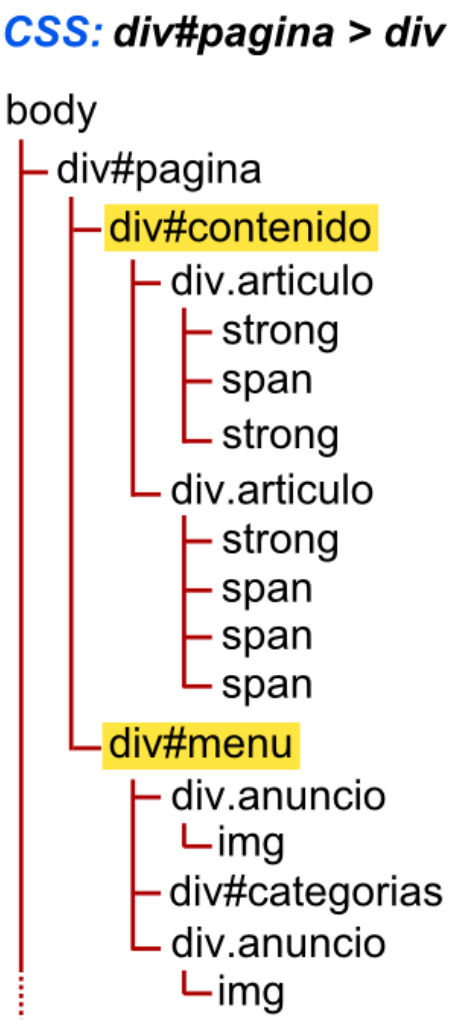
Mientras que la primera es mucho más específica, es una **muy buena práctica** en CSS mantener los selectores lo **menos específicos** posibles para evitar problemas de Especificidad (*a.k.a. CSS Peter Griffin*):

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 así nietos y sucesivos.

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

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



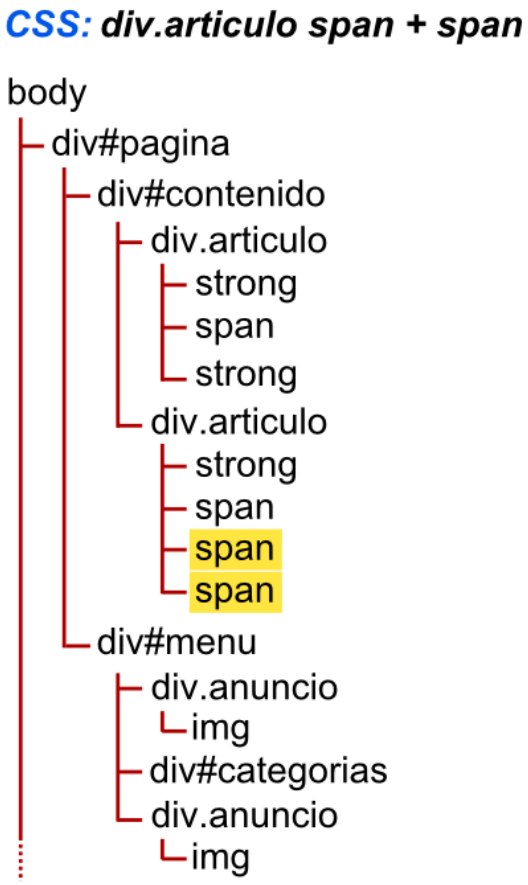
Al contrario que en el caso anterior, no se seleccionan todos los elementos ** ** descendientes, sino solo aquellos que son hijos directos del primer elemento especificado.

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 el símbolo **+** 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;
}
```

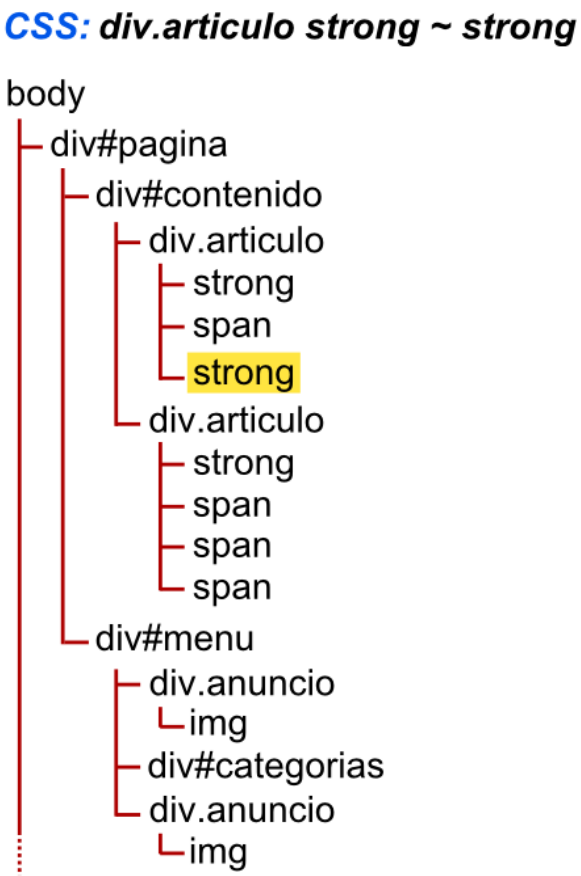
Cómo se podrá ver en este nuevo ejemplo, este combinador CSS hará que se seleccionen los elementos **span** que estén a continuación de un **div.articulo span**:



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

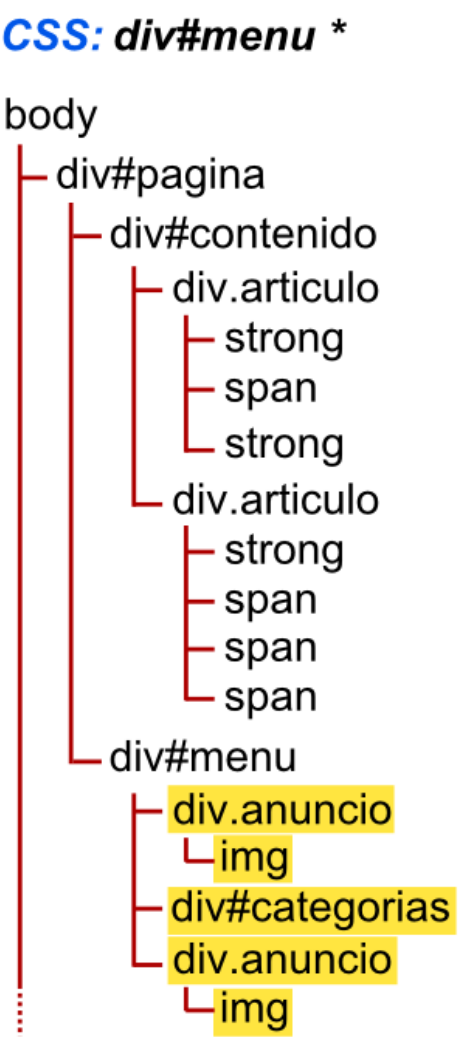
Si pensamos otras opciones en el ejemplo anterior, es posible que necesitemos ser menos específicos y en lugar de querer seleccionar los elementos hermanos **que sean adyacentes**, queramos seleccionar todos los hermanos en general, sin necesidad de que sean adyacentes. Esto se puede conseguir con el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



Como se ve en el ejemplo, no es necesario que el elemento **strong** se encuentre adyacente al 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 aplicar ciertos estilos en **TODOS Y CADA UNO** de los elementos HTML correspondientes.



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

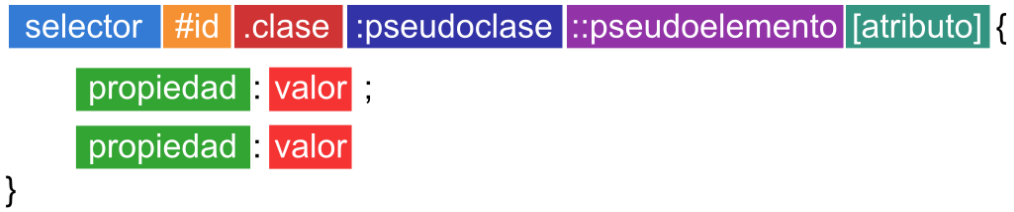
El **selector universal** puede ser muy útil en algunos casos para **resetear** ciertas propiedades de todo un documento, como en el siguiente ejemplo, donde se eliminan los márgenes de todos los elementos del documento HTML, puesto que algunos navegadores ponen márgenes diferentes y esto puede producir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

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

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 dependiendo de 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. En el 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:

: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** los enlaces que **aún no han sido visitados**, cambiando el color de los mismos o su formato, lo que 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 (*Internet Explorer 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 un elemento 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 los estilos 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 páginas webs 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 al anterior. 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 textos que hayan sido seleccionados, se mostrarán en verde.

Pseudoclases de activación

Por norma general, los elementos de un formulario HTML están siempre activados, aunque se 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 o desactivado:

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, podemos seleccionar 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 el atributo **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>** de un 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 por ejemplo 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 matricularse en 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 realizar las 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>**) que no 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 un pequeño 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.

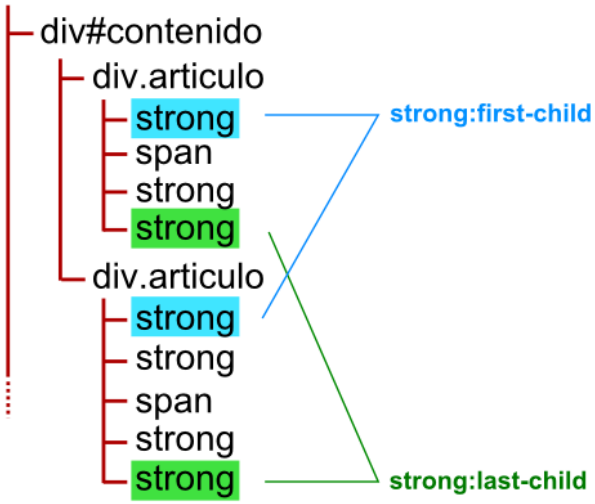
Para ello, volvamos a utilizar una estructura en forma de árbol para ver cómodamente la ubicación de cada uno de los elementos.

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

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

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

}



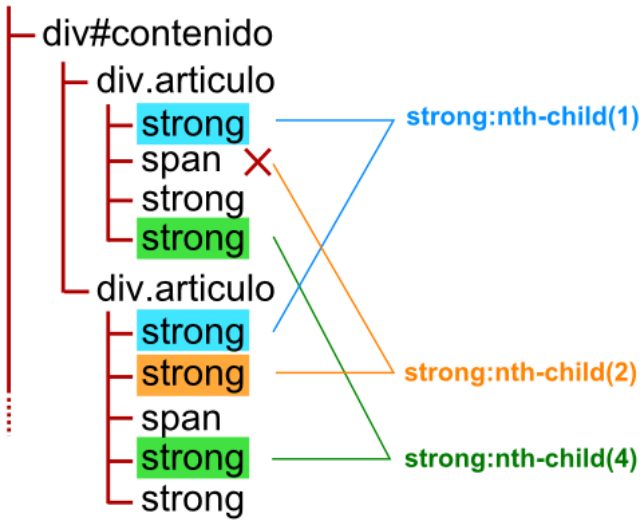
Sin embargo, si no queremos quedarnos en los primeros o últimos elementos y necesitamos más potencia para elegir, podemos hacer uso de la pseudoclase **:nth-child(A)**, que permite especificar el elemento deseado, simplemente estableciendo su número en el parámetro **A**:

Número	Equivalente a la pseudoclase	Significado
strong:nth-child(1)	**strong:first-child {}**	Primer elemento hijo, que además es un ****
strong:nth-child(2)		Segundo elemento hijo, que además es un ****

strong:nth-child(3)		Tercer elemento hijo, que además es un ****
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** del segundo caso, donde si existe.

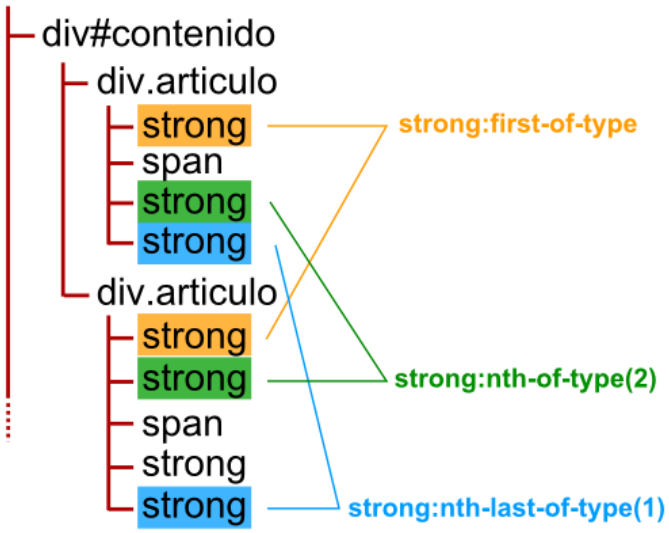
Elementos del mismo tipo

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, análogos a 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 otro lado, **: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 anterior pero 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 CSS que 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 ver fragmentos de código con pseudoelementos con la sintaxis de pseudoclase con **un solo par de puntos :**.

```
selector #id .clase :pseudoclase ::pseudoelemento [atributo] {
    propiedad : valor ;
    propiedad : valor
}
```

Generación de contenido

Dentro de la categoría de los **pseudoelementos CSS**, como punto central, se encuentra la propiedad **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 información mediante 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 que por 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 elemento HTML **<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 del enlace 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. De esta 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.

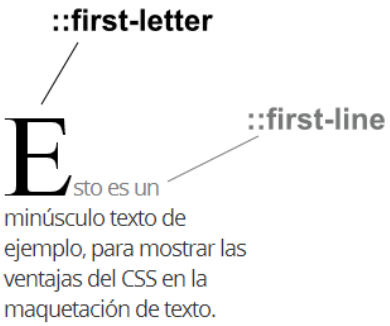
Veamos un ejemplo en acción sobre un párrafo de texto:

```
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, como cuentos 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 campos de 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** (un enlace a un PDF).
[class~="emezeta"]	Si el elemento tiene atributo **class** con una lista de valores y uno de ellos es **emezeta**.
[lang|="es"]	Si el elemento tiene atributo **lang** con una lista de valores, donde alguno 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. Estos elementos, 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 a un 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 deben tener 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 que debe 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 seleccionarían en 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 palabras separadas por espacios, donde una de ellas es el texto que hemos escrito a continuación. Se trata de una versión más restrictiva del comparador ***=**.

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, dichas transiciones 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 de atractivos 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 entre un 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 a la 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 elementos con 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**, desde el 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, transiciones muy 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**, que realiza 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)

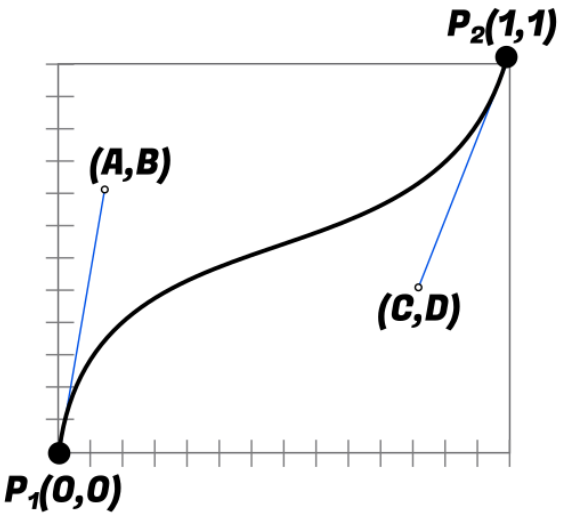
linear	Normal	Normal	Normal	(0, 0, 1, 1)
ease-in	Lento	Normal	Normal	(0.42, 0, 1, 1)
ease-out	Normal	Normal	Lento	(0, 0, 0.58, 1)
ease-in-out	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
cubic-bezier(*A*, *B*, *C*, *D*)	-	-	-	Transición personalizada

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 mezcla de 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 una de 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 curva bezier.	P_1
B	Y_1	Eje Y del primer punto que orienta la curva bezier.	P_1
C	X_2	Eje X del segundo punto que orienta la curva bezier.	P_2
D	Y_2	Eje Y del segundo punto que orienta la curva bezier.	P_2



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

Por último, la propiedad **transition-delay** nos ofrece la posibilidad de **retrasar el inicio de la 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, sin que 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, tendremos que 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, que son 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 finalizado y 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 estado de reproducción (*running*) o pausarla (*paused*).

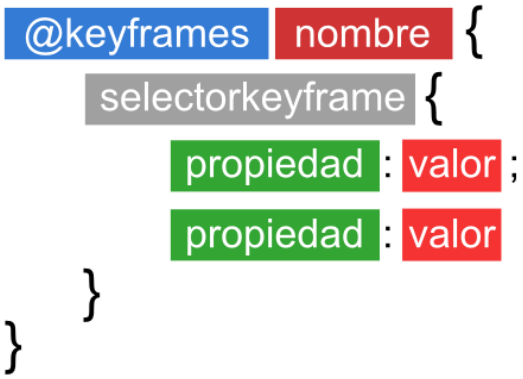
Atajo: Animaciones

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 de utilizar y se basa en el siguiente esquema:



En primer lugar elegiremos un **nombre** para la animación (*el cuál utilizamos en el apartado anterior, para hacer referencia a la animación, ya que podemos tener varias en una misma página*), 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.

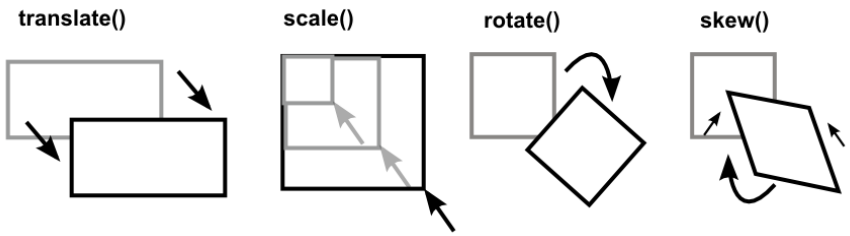
Las transformaciones es uno de los elementos más interesantes que se introducen en CSS3 para convertir el lenguaje de hojas de estilo en un sistema capaz de realizar todo tipo de efectos visuales, incluido 2D y 3D. Las propiedades principales para realizar transformaciones son las siguientes:

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 (*sobre tres 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 factor de 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 utilizamos diferentes 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 eje Y 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()**.