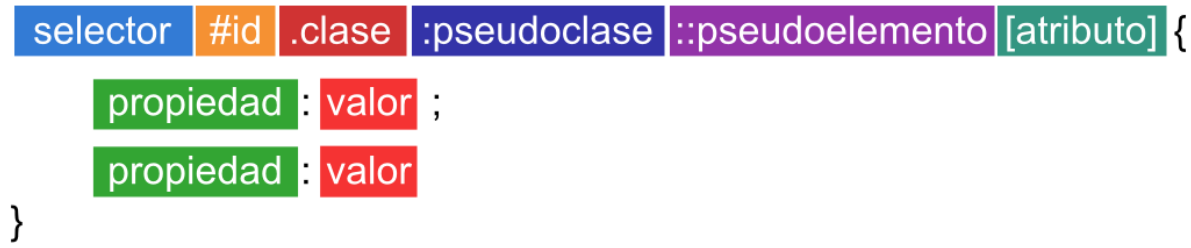
Pseudoclases y Pseudoelementos CSS

Las **pseudoclases** se utilizan para hacer referencia a ciertos comportamientos de los diferentes elementos del documento HTML. Así como los selectores básicos y avanzados se utilizan para dar estilos dependiendo de donde estén colocados estructuralmente, las pseudoclases se utilizan para dar estilos a elementos **respecto al comportamiento** que experimentan en determinado momento.

Recordemos el esquema general de sintaxis de CSS:



Enlaces

Existen algunas pseudo clases 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** 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
}
```

Mouse

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:

```
a:hover {
  background-color: cyan;
  padding: 2px
}
```

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
}
```

Interacción del usuario

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.

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

```
input:focus {
  border: 2px dotted #444
}
```

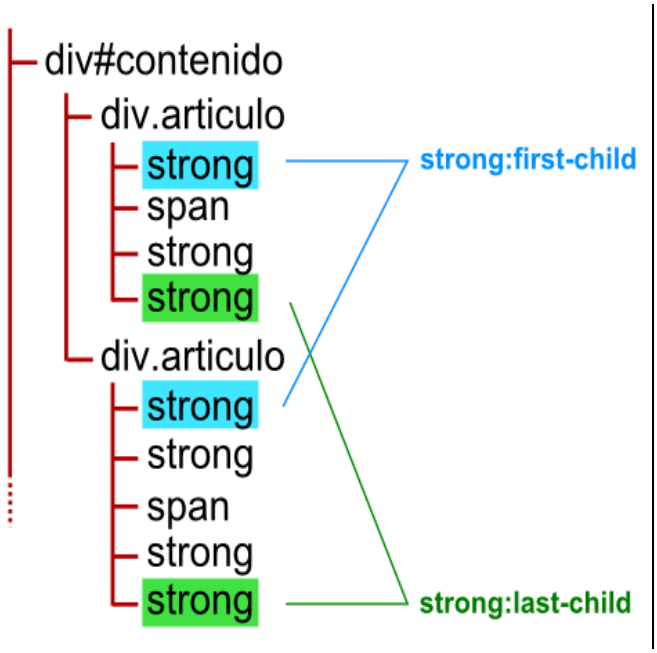
Existen varias pseudoclases que permiten hacer referencias a los elementos del documento HTML según su **posición y estructura de los elementos hijos**. A continuación muestro un pequeño resumen de estas pseudoclases:

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.

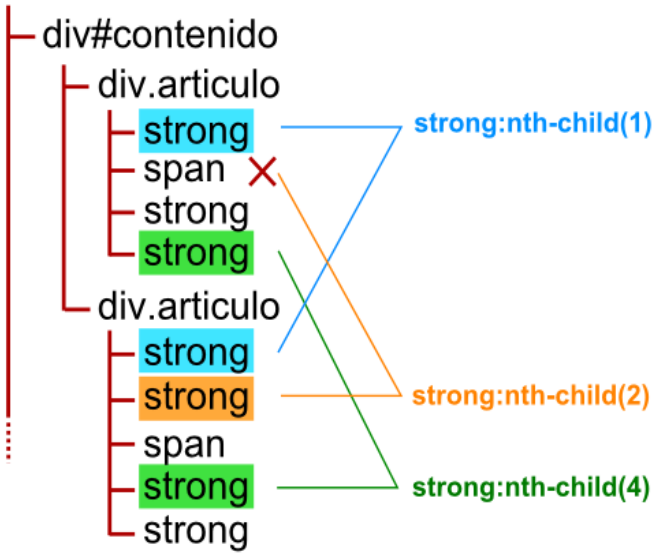
Las dos primeras pseudoclases, **:first-child** y **:last-child** hacen referencia a los primeros y últimos elementos (*al mismo nivel*) respectivamente.

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

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**:

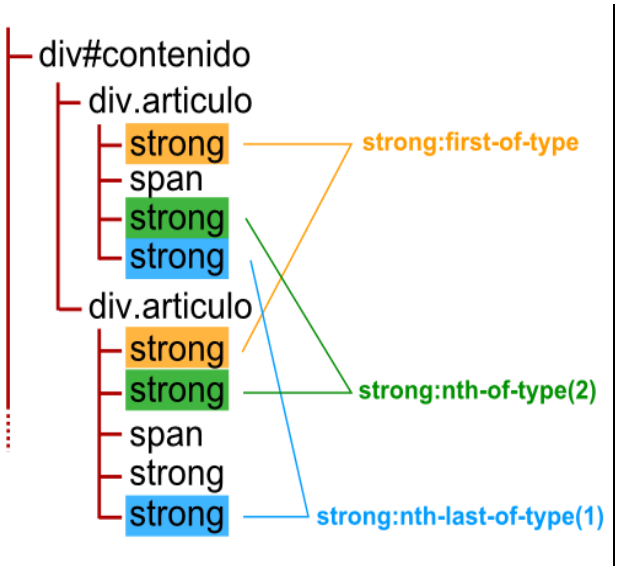
Número	Equivalente a la pseudoclase	Significado
strong:nth-child(1)	strong:first-child {}	Primer elemento hijo, que además es un
strong:nth-child(2)		Segundo elemento hijo, que además es un
strong:nth-child(3)		Tercer elemento hijo, que además es un
strong:nth-child(n)		Todos los elementos hijos que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



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.



Al igual que las pseudoclases, los **pseudo elementos** son otra de las características de CSS que permiten hacer referencias a comportamientos virtuales no tangibles que no tienen relación directa con el contenido del documento en sí.

Generación de contenido

Antes de continuar, hay que saber que en este apartado de pseudo elementos CSS es muy común utilizar algunas propiedades interesantes para generar contenido de forma automática, como es el caso de la propiedad content. Esta útil propiedad se combina con los pseudo elementos ::after o ::before, para añadir información antes o después de un elemento:

Pseudoelemento	Descripción
::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 indicado.	content: attr(href);
url(URL)	Añade la imagen indicada en la URL.	content: url(http://page.com/icon.png);

Por otro lado, los pseudoelementos ::before y ::after permiten hacer referencia a la ubicación justo antes y justo después (respectivamente) del elemento en cuestión.

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

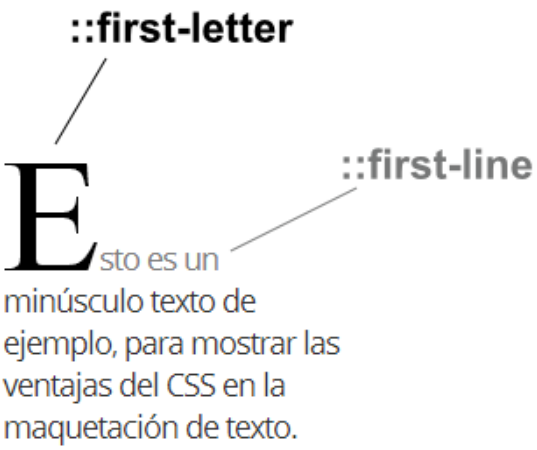
Primera letra y primera línea

También existen pseudo elementos con los que podemos hacer referencia a la primera letra de un texto. Para ello utilizamos el pseudo elemento ::first-letter, así como el pseudo elemento ::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.

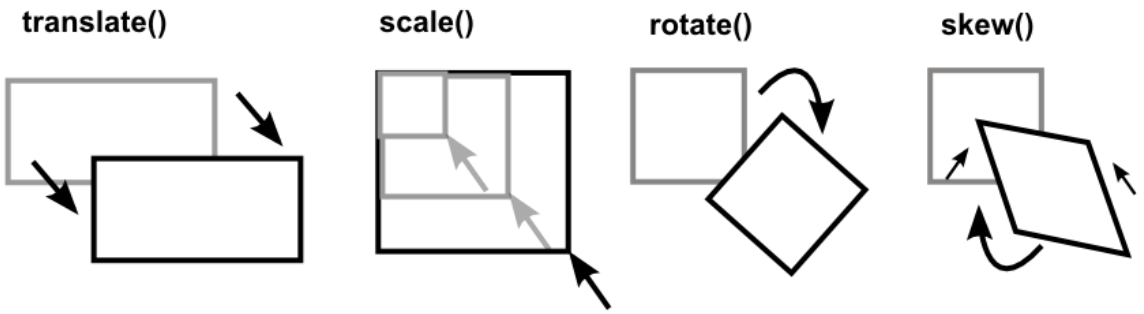
Veamos un ejemplo en acción sobre un párrafo de texto:

```
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;
}
```



Funciones de transformación 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
translate(x, y)	Traslada el elemento una distancia de TAMAÑO x horizontalmente e y verticalmente.
translateX(x)	Traslada el elemento una distancia de TAMAÑO x horizontalmente.
translateY(y)	Traslada el elemento una distancia de TAMAÑO y verticalmente.

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
scale(fx, fy)	Reescala el elemento a un nuevo tamaño con un factor fx horizontal y un factor fy vertical.
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.

Rotaciones

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

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

Con **transform: rotate(5deg)** realizamos una rotación de 5 grados del elemento. 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 o inclinar un elemento en 2D:

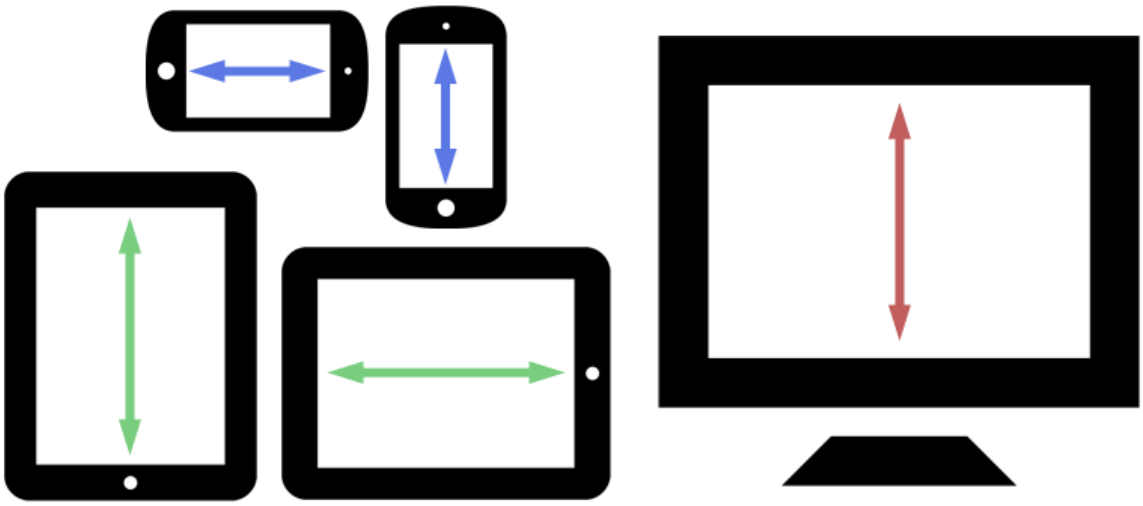
Funciones	Significado
skew(xdeg, ydeg)	Establece un ángulo de DIRECCION xdeg en X y ydeg en Y, para una deformación 2D
skewX(xdeg)	Establece un ángulo de DIRECCION xdeg para una deformación 2D respecto al eje X
skewY(ydeg)	Establece un ángulo de DIRECCION ydeg para una deformación 2D respecto al eje Y

Mediante **transform: skew(10deg, 5deg)** podemos hacer una deformación de varios grados, así como realizarla sólo respecto al eje X o eje Y con **skewX()** y **skewY()**.

Diseño Adaptativo

Una de las características que más se ha popularizado en los últimos años es el **diseño adaptativo** (*Responsive Web Design o RWD*). Se le denomina así a los diseños web que tienen la capacidad de adaptarse al tamaño y formato de la pantalla en la que se visualiza el contenido, respecto a los diseños tradicionales en los que las páginas web estaban diseñadas para un tamaño o formato específico.

De esta forma, se ha comenzado a escapar de los clásicos tiempos en los que se creaban varias versiones de una misma página para diferentes navegadores o resoluciones, donde el principal inconveniente para el desarrollador era que tenía que mantener varias versiones diferentes donde debía hacer los mismos cambios de forma repetitiva o complicaba el desarrollo en general.



El **Responsive Web Design** es algo muy útil en la era en que vivimos, donde existen dispositivos de todo tipo: smartphones de múltiples gamas y tamaños, tablets, sistemas de escritorio, etc. y sería inviable y poco práctico realizar una versión para cada uno de estos dispositivos.

Aunque en principio el concepto de **web adaptativa** es muy sencillo de comprender, aplicarlo puede ser todo un quebradero de cabeza si no se conocen bien las bases y se adquiere experiencia. En MediaQueri.es puedes encontrar algunos ejemplos de páginas que utilizan Responsive Web Design para tener clara la idea.

Estrategia adaptativa

Antes de comenzar a crear un diseño web adaptativo, es importante tener claro una serie de conceptos:

- Conoce las diferentes resoluciones más utilizadas
- Conoce el público de tu sitio web y tu objetivo
- Elige una estrategia acorde a lo anterior

En primer lugar, es importante **conocer los formatos** de pantalla más comunes con los cuales nos vamos a encontrar. Podemos consultar páginas como Screen Sizes o MyDevices, donde se nos muestra un listado de dispositivos categorizados en

smartphones, tablets o pantallas de escritorio con las características de cada dispositivo: ancho, alto, sistema operativo, etc...

Una vez estés familiarizado con estos detalles, es importante **conocer el público de tu sitio web**. ¿Acceden más usuarios desde móvil o desde escritorio? ¿Predominan las tablets o los móviles? ¿Tu objetivo es tener más usuarios de móvil o de escritorio?

Por último, decide la estrategia de desarrollo web que quieres utilizar. Aunque existen otras estrategias, hay dos vertientes principales:

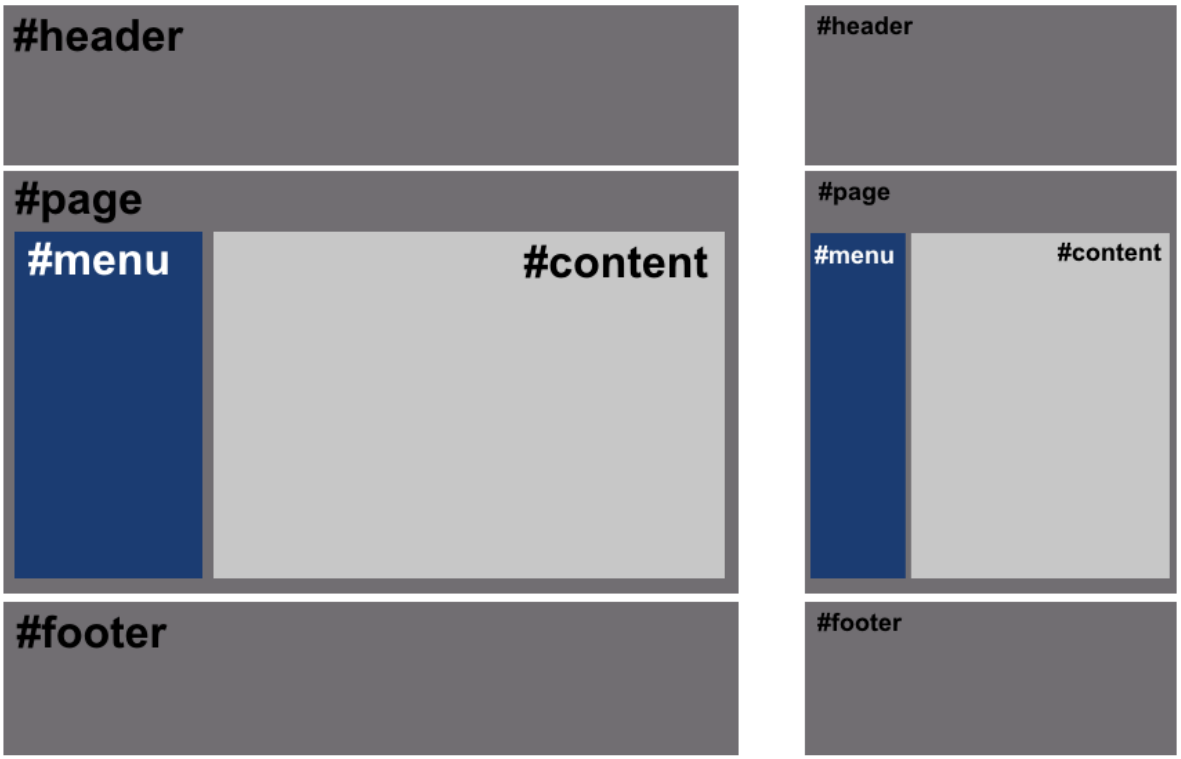
Estrategia	Descripción
Mobile first	Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
Desktop first	Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

Bien, si ya tenemos la teoría clara, toca volver al código.

Diseño con porcentajes

El primer paso para crear un diseño adaptativo correctamente, es comenzar a familiarizarse con un tipo de unidades: **los porcentajes**. Recordemos que los porcentajes son relativos al contenedor padre, por lo que si especificamos un porcentaje a un elemento, el navegador va a tomar dicho porcentaje del contenedor.

Podemos comenzar usando porcentajes en las propiedades **width**. De esta forma, se adaptarán dependiendo del ancho de la pantalla o navegador. Por ejemplo, si establecemos un ancho de **100%** a los elementos grises (*#header, #page y #footer*), un **30%** al azul (*#menu*) y un **70%** al gris claro (*#content*), a los cuales se les indicará un **display:inline-block** para estar contiguos, podríamos obtener este diseño:



Sin embargo, utilizar porcentajes no nos garantiza un diseño adaptativo de calidad. El primer problema que encontraremos será que si sumamos el tamaño de los elementos (*70% + 30%*), no es inferior al 100% del contenedor padre, por lo que **no cabe en su interior**, así que el segundo elemento se desplaza a la zona inferior, descuadrando todo el diseño. Esto es algo muy habitual en CSS. Y frustrante.

Comprobaremos, sin embargo, que si reducimos alguno de los dos porcentajes interiores (*70% o 30%*), si conseguiremos el diseño propuesto. Luego, si intentamos aumentar su relleno (*padding*) o el tamaño del borde (*border-width*) veremos que se vuelve a **descuadrar**. Esto ocurre porque el tamaño en porcentaje no incluye rellenos o el tamaño del borde, salvo que se utilice la propiedad **box-sizing:border-box**.

Estar pendiente de realizar todos estos ajustes (*sobre todo en diseños más complejos*) no suele ser algo práctico, por lo que vamos a ver otra serie de propiedades que suelen facilitarnos un poco todo este desarrollo.

Tamaños máximos y mínimos

Si buscamos un grado de control aún mayor, podríamos recurrir a las propiedades **max-width** y **min-width**, con las que podemos indicar el ancho de un elemento como máximo y el ancho de un elemento como mínimo respectivamente, consiguiendo así garantizar cierto control del diseño.

```
div {
  max-width: 800px;
  min-width: 300px;
}
```

En este caso, el elemento tiene un tamaño máximo de 800 píxeles, y un tamaño mínimo de 300 píxeles, por lo que si ajustamos el ancho de la ventana del navegador, dicho elemento ** ** iría variando en un rango de 300 a 800 píxeles, nunca haciéndose más pequeño de 300 o más grande de 800.

> **Nota**: Es importante darse cuenta de que este ejemplo funciona porque no hay definido un **width** (*por omisión, es igual a* ***width:auto***). Desde que exista un **width**, las otras propiedades pierden efecto.

Con las imágenes, videos y contenidos multimedia, se puede hacer lo mismo, consiguiendo así que las imágenes se escalen y adapten al formato especificado o incluso al tamaño de pantalla de los diferentes dispositivos utilizados:

```
img, video, object, embed {
  max-width:100%;      /* Aquí el tamaño máximo de tus imágenes */
  height:auto;         /* No es necesario, sólo para clarificar */
}
```

Transiciones CSS
En CSS3 se introduce uno de los aspectos más interesantes en la web interactiva: **las transiciones**. En versiones anteriores de CSS sólo se podían utilizar ciertas funcionalidades interactivas con pseudo clases como **:hover** o **:focus**. Sin embargo, dichas transiciones ocurrían de golpe, pasando de un estado a otro. Ahora, con 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
transition-duration	0 | TIEMPO
transition-timing-function	ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(A, B, C, D)
transition-delay	0 | TIEMPO

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.

> **Nota**: Debes saber que no todos los elementos permiten animación debido a su complejidad. Por ejemplo, los **background-image** de gradientes no son animables actualmente.

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 **tipo de transición** que queremos conseguir. Cuando comenzamos, recomiendo utilizar **linear**, que realiza una transición lineal, siempre a velocidad constante. Sin embargo, podemos variar dicha velocidad para que sea más rápida al principio y más lenta al final, o viceversa, entre otras posibilidades.
Los valores que puede tomar la propiedad son los siguientes:

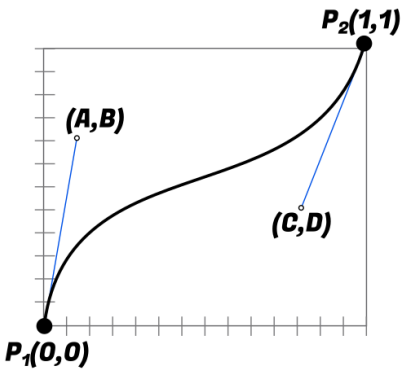
Valor	Inicio	Transcurso	Final	Equivalente en cubic-bezier
ease	Lento	Rápido	Lento	(0.25, 0.1, 0.25, 1)
linear	Normal	Normal	Normal	(0, 0, 1, 1)
ease-in	Lento	Normal	Normal	(0.42, 0, 1, 1)
ease-out	Normal	Normal	Lento	(0, 0, 0.58, 1)
ease-in-out	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
cubic-bezier(A, B, 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 dependiendo 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:

Parámetro	Valor	Descripción	Pertenece a
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.com, 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;
}
```

> **Truco**: Si nos fijamos bien, este estilo se aplica sólo al mover el ratón sobre el enlace (*transición de entrada*). Sin embargo, si movemos el ratón fuera del enlace, no se produce transición sino que realiza el cambio de forma brusca. Podemos incluir las propiedades de transición también en el primer bloque y así contemplar también la **transición de salida**.

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 CSS3, es muy fácil adaptarnos al concepto de animaciones CSS3, el cual amplía el concepto de transiciones convirtiéndolo en algo mucho más flexible y potente.

Como habíamos mencionado, las transiciones son la manera de suavizar un cambio de un estado inicial a un estado final. La idea de las animaciones CSS es permitir la adición de más estados, pudiendo realizar cambios desde un estado inicial, a un estado posterior, y luego a otro estado posterior, y así sucesivamente.

Para crear animaciones debemos tener dos cosas claras. Por un lado, la regla **@keyframes**, que incluye los fotogramas de la animación, mientras que por otro lado tenemos las propiedades CSS de las **animaciones**, que definen el comportamiento de la misma.

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 | TIEMPO
animation-timing-function	ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(A, B, C, D)
animation-delay	0 | TIEMPO

animation-iteration-count	1 | infinite | número
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

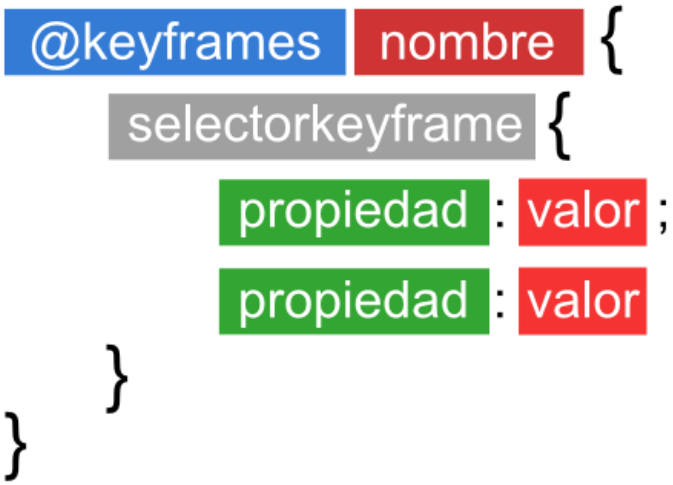
Nuevamente, CSS ofrece la posibilidad de resumir todas estas propiedades en una sola, para hacer nuestras hojas de estilos más específicas. El orden de la propiedad de atajo sería el siguiente:

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

> **Consejo**: Mucho cuidado al indicar los segundos en las propiedades de duración. Al ser una unidad diferente a las que solemos manejar (*px, em, etc...*) hay que especificar **siempre** la **s**, aunque sea un valor igual a **0**.

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 cual 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 */
}
```

Animate CSS : https://daneden.github.io/animate.css/

Truco: Si tienes fotogramas que van a utilizar los mismos estilos que uno anterior, siempre puedes separarlos con comas, por ejemplo: **0%, 75% { /* Estilos CSS */ }**, que utilizarían dichos estilos al inicio de la animación y al 75% de la misma.

.