



Selectores avanzados y
Animaciones

Selectores avanzados

Selectores avanzados

Utilizan “combinadores”, signos gráficos que establecen la relación entre los elementos y permiten hacer una selección **específica**:

Selector	Caracter	Descripción	Ejemplo
Agrupado	, (coma)	Se utiliza cuando varios elementos comparten una serie de declaraciones iguales, se aplican las reglas CSS a los selectores involucrados. +info	<pre>p, a, div { /*Reglas CSS*/ }</pre>
Descendiente	(espacio)	Apunta a elementos contenidos dentro de otro en la estructura del documento, sin importar la profundidad o los descendientes interpuestos entre ellos +info	<pre>div p { /*Reglas CSS*/ }</pre>
Hijos directos	> (mayor)	Selecciona los elementos que sean hijos directos del contenedor padre, descartando nietos y sucesivos. +info	<pre>span > a { /*Reglas CSS*/ }</pre>

Selectores avanzados

Selector	Caracter	Descripción	Ejemplo
Hermano adyacente	+(más)	Aplica estilos a elementos que siguen inmediatamente a otros. Deben tener el mismo elemento padre ser inmediatamente siguiente a él. +info	<pre>div + p { /*Reglas CSS*/ }</pre>
General de hermanos	~(virgulilla o tilde de la ñ)	Selecciona todos los elementos que son hermanos del especificado, sin necesidad de que sean adyacentes. +info	<pre>div ~ p { /*Reglas CSS*/ }</pre>

Pseudoclases

Una **pseudoclase** es un **selector** que marca los elementos que están en un estado específico o tienen un comportamiento determinado. Todas las pseudoclases se denominan mediante una palabra precedida por dos puntos y se comportan del mismo modo. Afectan a un fragmento del documento que está en un estado determinado y se comportan como si se hubiera añadido una clase a su HTML.

:first-child

Esta pseudoclase modifica el estilo del primer elemento de un grupo de elementos hermanos dentro de un contenedor, es decir “*el primer hijo de su padre*”. [+info](#)

```
<div>
  <p> Párrafo 1 </p>
  <p> Párrafo 2 </p>
  <p> Párrafo 3 </p>
</div>
```

```
p:first-child {
  color: red;
}
```

Párrafo 1

Párrafo 2

Párrafo 3

:last-child

Se utiliza para representar al último elemento entre un grupo de elementos hermanos dentro de un contenedor, es decir “*el último hijo de su padre*”. [+info](#)

```
<div>
  <p> Párrafo 1 </p>
  <p> Párrafo 2 </p>
  <p> Párrafo 3 </p>
</div>
```

```
p:last-child {
  color: blue;
}
```

Párrafo 1

Párrafo 2

Párrafo 3

:nth-child(n)

El selector coincide con cada elemento que es el **n-ésimo** hijo, independientemente del tipo, de su padre. **n** puede ser un número, una palabra clave o una fórmula. [+info](#)

```
<div>
  <p> Párrafo 1 </p>
  <p> Párrafo 2 </p>
  <p> Párrafo 3 </p>
</div>
```

```
p:nth-child(2) {
  background: cyan;
}
```

Párrafo 1

Párrafo 2

Párrafo 3

:nth-child(n)

En el ejemplo siguiente, vemos como de la lista se seleccionan primero los elementos 3, 6 y 9, y luego los que son pares:

```
<ol>
  <li> Item 1</li>
  <li> Item 2</li>
  <li> Item 3</li>
  <li> Item 4</li>
  <li> Item 5</li>
  <li> Item 6</li>
  <li> Item 7</li>
  <li> Item 8</li>
  <li> Item 9</li>
</ol>
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

```
li:nth-child(3n) {
  background:
lightskyblue;}
```

```
li:nth-child(2n) {
  background:
lightgreen;}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

:nth-child(n): Otros ejemplos

```
li:nth-child(3n+4) {  
    background: lightcoral;  
}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

```
li:nth-child(even) {background: lightcoral;}  
/*pares*/  
  
li:nth-child(odd) {background: lightgreen;}  
/*impares*/
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

Pseudoclasas para hipervínculos

Se aplican a las etiquetas `<a>`, que pueden tener cuatro estados: [+info](#)

- **:link** se refiere a un enlace que todavía no ha sido visitado.
- **:hover** se refiere a un elemento sobre el que se coloca el puntero del mouse.
- **:visited** se refiere a un enlace que ya ha sido visitado.
- **:active** se refiere a cualquier elemento que ha sido activado por el usuario.

```
<a href="https://google.com" target="_blank">Ir a Google</a>
```

```
a:link {color: red;}  
a:hover {background-color: yellow;}  
a:visited {color: blue;}  
a:active {background-color: green; color: white;}
```

Contacto

Contacto

Contacto

Contacto

Pseudoelementos

Los **pseudoelementos** se añaden a los selectores, pero no describen un estado especial sino que permiten añadir estilos a una parte concreta del documento.

Pseudoelementos

Se utilizan para darle estilos a partes específicas de un elemento. Están precedidos por cuatro puntos (:):

::first-letter se utiliza para darle estilo a la primera letra de un texto: [+info](#)

```
<p>Párrafo con la primera letra de otro color</p>
```

Párrafo con la pri

```
p::first-letter{color:blue;}
```

::first-line se utiliza para darle estilo a la primera línea de un párrafo: [+info](#)

```
p::first-line{background-color: lightgreen;}
```

Lorem ipsum dolor sit amet consectetur adipisicing elit.
Deleniti quaerat asperiores vitae aspernatur ut incidunt
dolores tempora saepe harum at, ullam laudantium

Pseudoelementos

::selection agrega estilos a una parte del documento que ha sido resaltada por el usuario: [+info](#)


```
p::selection{  
    background-color: lightsalmon;  
}
```

Lorem ipsum dolor sit amet consectetur
harum at, ullam laudantium consectetur
molestias eius, magnam explicabo hic a

::before y **::after** agregan contenido antes y después, respectivamente, del contenido: [+info](#) [+info](#)

```
p::before{ content:"✨";}  
p::after{ content:"🙈";}
```

✨ Texto de ejemplo 🙈



Transformaciones, animaciones y transiciones

Transformaciones

Permiten mover, rotar, escalar y sesgar elementos, es decir, efectos visuales en 2D y 3D. Las propiedades principales para realizar transformaciones son las siguientes:

Propiedades	Formato	Significado
<code>transform</code>	<i>función1</i> , <i>función2</i> , ...	Aplica una o varias funciones de transformación sobre un elemento.
<code>transform-origin</code>	<code>POSX</code> <code>POSY</code> <code>POSZ</code>	Cambia el punto de origen del elemento en una transformación.
<code>transform-style</code>	<code>flat</code> <code>preserve-3d</code>	Modifica el tratamiento de los elementos hijos.

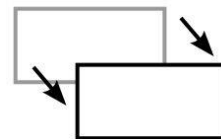
Con la propiedad **transform** podemos indicar una o varias transformaciones para realizar sobre un elemento, ya sean 2D (sobre dos ejes) o 3D (sobre tres ejes).

2D - Traslaciones (translate)

Las **funciones de traslación** 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
<code>translateX(x)</code>	Traslada el elemento una distancia de <small>SIZE</small> <code>x</code> horizontalmente.
<code>translateY(y)</code>	Traslada el elemento una distancia de <small>SIZE</small> <code>y</code> verticalmente.
<code>translate(x, y)</code>	Propiedad de atajo de las dos anteriores.

`translate()`

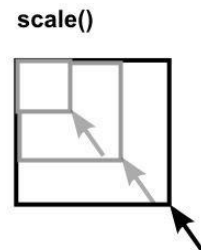


Por ejemplo, **transform: translate(20px, -30px)** traslada el elemento 20 píxeles a la derecha y 30 píxeles hacia arriba, que es equivalente a utilizar **transform: translateX(20px) translateY(-30px)**. [+info](#)

2D - Escalado (scale)

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
<code>scaleX(fx)</code>	Reescala el elemento a un nuevo tamaño con un factor <code>NUMBER</code> fx horizontal.
<code>scaleY(fy)</code>	Reescala el elemento a un nuevo tamaño con un factor <code>NUMBER</code> fy vertical.
<code>scale(fx, fy)</code>	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. [+info](#)

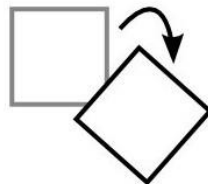
2D - Rotaciones

(rotate)

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

Funciones	Significado
<code>rotateX(xdeg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>xdeg</i> grados sólo para el eje horizontal X.
<code>rotateY(ydeg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>ydeg</i> grados sólo para el eje vertical Y.
<code>rotate(deg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>deg</i> grados sobre sí mismo.

`rotate()`



Con **transform: rotate(5deg)** realizamos una rotación de 5 grados del elemento sobre sí mismo. Utilizando **rotateX()** y **rotateY()** podemos hacer lo mismo respecto al eje X o el eje Y respectivamente. [+info](#)

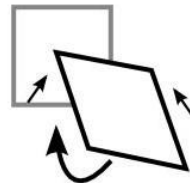
2D - Deformaciones

(skew)

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

Funciones	Significado
<code>skewX(xdeg)</code>	Establece un ángulo de <code>DIRECTION</code> <code>xdeg</code> para una deformación 2D respecto al eje X
<code>skewY(ydeg)</code>	Establece un ángulo de <code>DIRECTION</code> <code>ydeg</code> para una deformación 2D respecto al eje Y

skew()

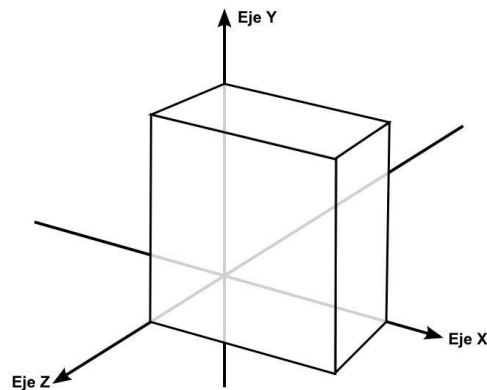


Aunque la función **skew()** existe, no debería ser utilizada, ya que está marcada como obsoleta y serán retiradas de los navegadores en el futuro. En su lugar deberían utilizarse **skewX()** o **skewY()**. [+info](#)

Funciones 3D

A las funciones anteriores, también podemos añadir las funciones equivalentes de CSS para hacer uso del eje Z y con esto acceder a las dimensiones espaciales o "3D". Las propiedades de transformación 3D son las siguientes: [+info](#)

Funciones	Significado
<code>translateZ(z)</code>	Traslada el elemento una distancia de <code>SIZE</code> <code>z</code> en el eje de profundidad.
<code>translate3d(x, y, z)</code>	Establece una translación 3D, donde aplica los parámetros de <code>SIZE</code> a cada eje.
<code>scaleZ(fz)</code>	Reescala el elemento a un nuevo tamaño con factor <code>NUMBER</code> <code>fz</code> de profundidad.
<code>scale3d(fx, fy, fz)</code>	Establece un escalado 3D, donde aplica los factores a cada eje.
<code>rotateZ(zdeg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <code>zdeg</code> grados sólo para el eje de profundidad Z.
<code>rotate3d(x, y, z, deg)</code>	Establece una rotación 3D, aplicando un vector <code>[x,y,z]</code> y el ángulo en <code>DIRECTION</code> <code>deg</code> .
<code>perspective(n)</code>	Establece una perspectiva 3D de <code>SIZE</code> <code>n</code> .
<code>matrix3d(n, n, ...)</code>	Establece una matriz de transformación 3D (<i>16 valores</i>)



Transformaciones múltiples

Si indicamos dos propiedades **transform** en el mismo elemento, con diferentes funciones, la segunda propiedad sobrescribirá a la anterior, como ocurre con cualquier propiedad de CSS:

```
div {  
  transform: rotate(5deg);  
  transform: scale(2,2);    /* Esta línea sobrescribe a la anterior */  
}
```

Para evitar esto se suelen emplear múltiples transformaciones, separándolas mediante un espacio. En el siguiente ejemplo, aplicamos una función de rotación, una función de escalado y una función de traslación de forma simultánea:

```
div { transform: rotate(5deg) scale(2,2) translate(20px, 40px); }
```

Transiciones

Las **transiciones** CSS le permiten cambiar los valores de una propiedad, durante un período determinado. Se basan en un principio muy básico: conseguir un **cambio de estilo** con un efecto suavizado entre un estado inicial y un estado final.

Para crear un efecto de transición, debemos especificar dos cosas:

- La propiedad CSS a la que desea agregar un efecto (*¿qué propiedad modifico?*)
- La duración del efecto (*¿durante cuánto tiempo?*)

Las propiedades relacionadas que existen son las siguientes:

Propiedades	Valor
<code>transition-property</code>	<code>all</code> <code>none</code> <i>propiedad css</i>
<code>transition-duration</code>	<code>0</code> <code>TIME</code>
<code>transition-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier(A, B, C, D)</code>
<code>transition-delay</code>	<code>0</code> <code>TIME</code>

Transiciones

- **transition-property:** Indica **la propiedad que será afectada por la transición**. Puede ser una propiedad concreta (**width** o color, por ejemplo) o simplemente **all** para que se aplique a todos los elementos. Por otro lado, **none** hace que no se aplique ninguna transición.
- **transition-duration:** Establece la **duración de la transición**. Se recomienda comenzar con valores cortos, para que las transiciones sean rápidas y elegantes. Una duración demasiado grande producirá una transición con detenciones intermitentes, y pueden resultar molestas a muchos usuarios.
- **transition-timing-function:** indica el **ritmo de la transición** que queremos conseguir. Se recomienda comenzar con **linear** (ritmo constante) y luego utilizar otros valores para variar el ritmo sea al inicio y/o al final de la transición.

Transiciones

Los valores que puede tomar la propiedad son los siguientes:

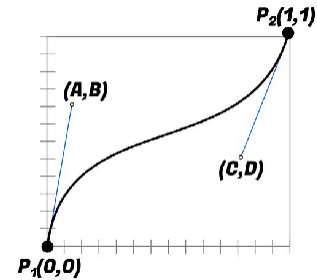
Valor	Inicio	Transcurso	Final	Equivalente en cubic-bezier
ease	Lento	Rápido	Lento	(0.25, 0.1, 0.25, 1)
linear	Normal	Normal	Normal	(0, 0, 1, 1)
ease-in	Lento	Normal	Normal	(0.42, 0, 1, 1)
ease-out	Normal	Normal	Lento	(0, 0, 0.58, 1)
ease-in-out	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
cubic-bezier(<u>A</u> , <u>B</u> , <u>C</u> , <u>D</u>)	-	-	-	Transición personalizada

Un valor **linear** siempre es constante, mientras que **ease** comienza suavemente, aumenta la velocidad y finaliza suavemente. **Ease-in** y **ease-out** son variaciones que van más lento al principio o al final, y **ease-in-out** una mezcla de ambas. La función **Cubic-Bezier()** nos permite configurar con más detalle la transición.

La función de tiempo Cubic-Bezier()

Es una función personalizada. Podemos asignar valores que definen 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



[Simulador de Cubic-Bezier\(\)](#)

transition-delay nos ofrece la posibilidad de retrasar el inicio de la transición los segundos especificados.

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;  
}
```

Fuente: <https://lenguajecss.com/css/animaciones/transiciones/>

Animaciones

Una animación permite que un elemento cambie gradualmente de un estilo a otro. Podemos cambiar tantas propiedades CSS como deseemos, tantas veces como sea necesario.

Las animaciones amplían el concepto de transiciones convirtiéndolo en algo mucho más flexible y potente, partiendo del mismo concepto de realizar cambios en ciertos estados inicial y final pero incorporando 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.**

Para utilizar la animación CSS, primero debemos especificar algunos fotogramas clave (**@keyframes**) para la animación, que contendrán los estilos que tendrá el elemento en determinados momentos. Además tendremos que utilizar las propiedades de las animaciones, que definen el comportamiento de la misma.

Propiedades de animación CSS

Para definir este comportamiento necesitamos conocer las siguientes propiedades CSS:

Propiedades	Valor
<code>animation-name</code>	<code>none</code> <i>nombre</i>
<code>animation-duration</code>	<code>0</code> <code>TIME</code>
<code>animation-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier(A, B, C, D)</code>
<code>animation-delay</code>	<code>0</code> <code>TIME</code>
<code>animation-iteration-count</code>	<code>1</code> <code>infinite</code> <code>NUMBER</code>
<code>animation-direction</code>	<code>normal</code> <code>reverse</code> <code>alternate</code> <code>alternate-reverse</code>
<code>animation-fill-mode</code>	<code>none</code> <code>forwards</code> <code>backwards</code> <code>both</code>
<code>animation-play-state</code>	<code>running</code> <code>paused</code>

animation-name permite especificar el nombre del fotograma a utilizar. **animation-duration**, **animation-timing-function** y **animation-delay** funcionan exactamente igual que en transiciones.

Propiedades de animación CSS

- 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 reproducen 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.

Propiedades de animación CSS

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**).

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;  
}
```

Debemos ser cuidadosos al indicar el tiempo en las propiedades de duración. Al ser una unidad diferente a las que solemos manejar (px, em, etc...) hay que especificar **siempre** la **s**, por segundos, **aunque sea un valor igual a 0**.

Fotogramas (keyframes)

Para definir los fotogramas de una animación utilizaremos la regla **@keyframes**. Primero elegimos un nombre para 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.

```
@keyframes nombre {  
  selectorkeyframe {  
    propiedad : valor ;  
    propiedad : valor  
  }  
}
```

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: blue;  
  animation-name: cambiarColor;  
  animation-duration: 2s;  
  animation-delay: 1s;  
  /* animation: cambiarColor 2s 1s; */ }  
  
@keyframes cambiarColor {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}
```

Fotogramas (keyframes)

En este ejemplo partimos de un primer fotograma con un elemento con color de fondo rojo. Si observamos el último fotograma, indicamos finalice con color de fondo verde. La regla **@keyframes** creará la animación intermedia para conseguir que el elemento cambie de color.

Los selectores **from** y **to** son realmente sinónimos de 0% y 100%. Al modificarlos podremos ir añadiendo nuevos fotogramas intermedios:

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: blue;  
  animation-name: cambiarColor;  
  animation-duration: 2s;  
  animation-timing-function: ease;  
  animation-iteration-count: infinite;  
}
```

```
@keyframes cambiarColor {  
  0% {background: red; width: 200px;}  
    /* Primer fotograma */  
  50% {background: yellow; width: 400px;}  
    /* Segundo fotograma */  
  100% {background: green; width: 600px;}  
    /* Último fotograma */  
}
```

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 2s linear 7.5s,   /* Comienza a los 7.5s (5 + 2.5) */  
    disappear 2s linear 9.5s; /* Comienza a los 9.5s (2 + 7.5) */  
}
```

Hemos aplicado varias animaciones a la vez, estableciendo un retardo equivalente a la suma de la duración de las animaciones anteriores, encadenando una animación con otra. [+info](#)

Librería de animaciones Animate.css

Podemos utilizar **Animate.css** para dar dinamismo a nuestro contenido. [Ingresar](#)

- En pocos pasos se pueden agregar animaciones CSS a cualquier elemento con esta sencilla librería.
- En la creación de cualquier contenido web resulta interesante incorporar animaciones que nos ayuden a mejorar la experiencia del usuario durante la interacción con el contenido.
- Permite disponer de una gran variedad de animaciones CSS sin necesidad de crearlas nosotros mismos.
- Esta librería permite conseguir que el contenido sea más atractivo y dinámico.



Introducción Responsive Web Design

Diseño Web Responsivo

El **diseño web responsivo** se trata de usar **HTML** y **CSS** para *cambiar el tamaño, ocultar, reducir o ampliar automáticamente un sitio web*, para que se vea bien en todos los dispositivos (computadoras de escritorio, tabletas y teléfonos).

Una web responsive es aquella que es capaz de adaptarse a cualquier dispositivo, ya sean computadoras, portátiles, tablets o smartphones. Y en cada uno de estos dispositivos el sitio web debe visualizarse correctamente.

El diseño responsivo se encarga precisamente de esto, de responder al tamaño de los dispositivos desde los que se visualizan los contenidos web, adaptando sus dimensiones y mostrando los componentes de manera optimizada y ordenada sin importar el tipo de soporte que sea.

Diseño responsivo vs Diseño adaptativo

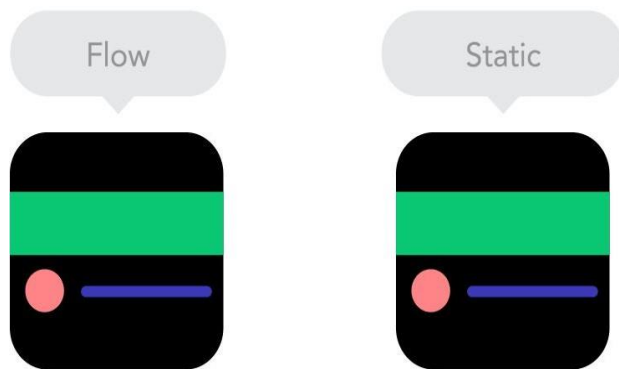
Un diseño **responsivo** responde a las dimensiones del dispositivo, mientras que un diseño **adaptativo** se adapta, pero no necesariamente responde en todo momento, tiene cierto delay, estamos hablando casi de lo mismo.

El diseño web responsivo **adapta la estructura y los diferentes elementos** de cada página web a las dimensiones y características de cada pantalla. Por otro lado, el diseño web adaptativo es **menos flexible**, y se basa en el uso de tamaños y características pre-establecidas. Las diferencias entre ambos métodos se encuentran en el proceso creativo y de diseño, en el resultado final y la experiencia del usuario.



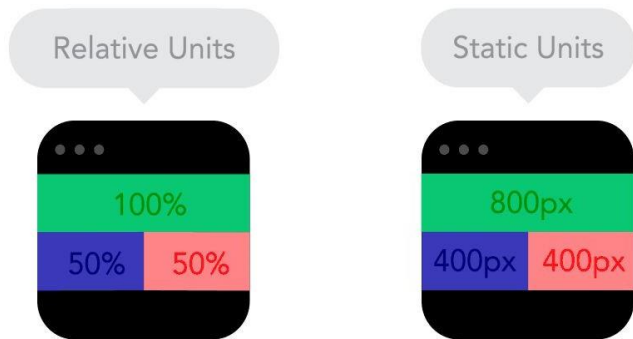
Flujo (The Flow) vs Estático (Static)

- Cuando una pantalla se vuelve más pequeña, el contenido comienza a crecer verticalmente ocupando más espacio, y el contenido que se encuentra debajo va a ser desplazado hacia abajo, eso se llama **flujo**.
- Si es estático ese flujo de elementos no se desplaza, no se adapta al ancho del viewport y se pierde contenido o cierto contenido tapa a otro.



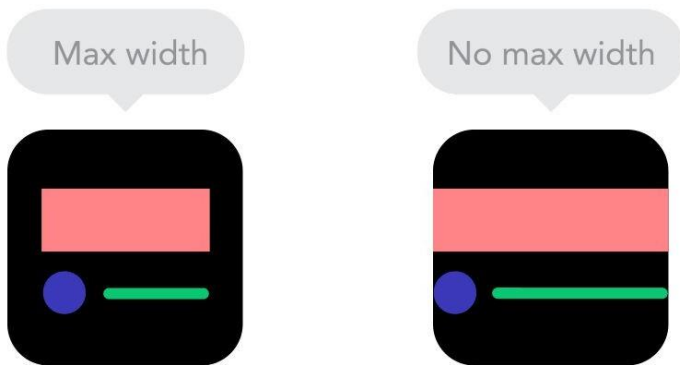
Unidades Relativas vs Unidades Absolutas

- La densidad de píxeles de cada dispositivo puede variar, por eso necesitamos unidades que sean flexibles y funcionen sin importar el dispositivo. Ahí es donde las unidades relativas como los porcentajes son útiles. Entonces, hacer algo con un 50% de ancho significa que siempre ocupará la mitad de la pantalla (viewport, el tamaño de la ventana del navegador abierta), independientemente del dispositivo.



Valores Mínimos y Máximos

- En un celular nos puede interesar que determinado contenido ocupe todo el ancho de la pantalla, pero al pasar a un TV 4K podríamos cambiar de idea. Por ejemplo podríamos tener un **width: 100%**, pero con un **max width: 1000px**.
- En un celular, las imágenes pueden tener un ancho diferente a las que vemos en otras pantallas. El alto no importa tanto en mobile, porque podemos *scrollear*.



Puntos de corte (Breakpoints)

Estos puntos de corte permiten al diseño cambiar en determinados puntos, por ejemplo, en un monitor podemos tener 3 columnas, pero sólo 1 en un celular (que es más angosto). Estos puntos de quiebre se crean con los *media queries*, que nos permiten determinar que si el mínimo del ancho de la pantalla tiene un valor en lugar de otro, en vez de distribuir el contenido en tres columnas colocarlo en una sola, con varias filas.

With Breakpoints

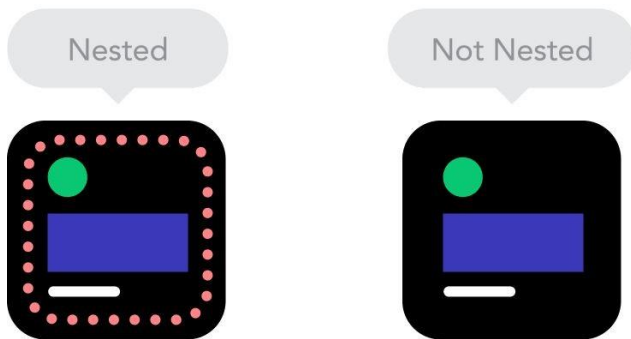


Without Breakpoints



Objetos anidados (Nested Objects)

- Tener muchos objetos que dependan de otros puede ser difícil de controlar, sin embargo, agruparlos en contenedores nos puede simplificar las cosas.
- ¿Por qué usamos contenedores? Porque a la hora de pensar contenido responsive nos va a facilitar posicionar un grupo de elementos en otro lugar.



Mobile first vs Desktop first

- **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.
- Estadísticamente, los dispositivos que acceden a los sitios Web.

Desktop first



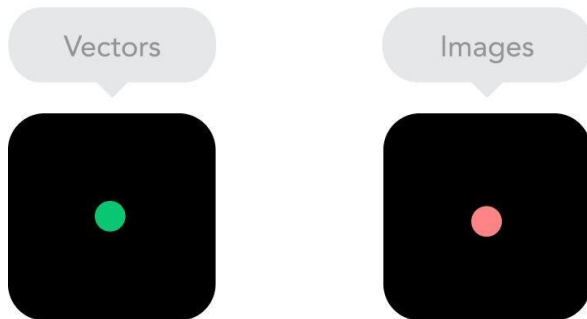
Mobile first



Bitmaps vs Vectors

Bitmaps: JPG, PNG, GIF. Recomendadas para muchos detalles y efectos.

Vectors: SVG (gráficos basados en vectores escalables), si voy a mostrar un ícono uso Icon Fonts, que son más livianos, pero algunos exploradores viejos no los soportan.



Texto responsivo

Recordemos que el tamaño del texto se puede configurar con una unidad "vw", que es el "ancho de la ventana gráfica". De esa forma, el tamaño del texto seguirá el tamaño de la ventana del navegador.

```
<h1 style="font-size:10vw">Hello World</h1>
```

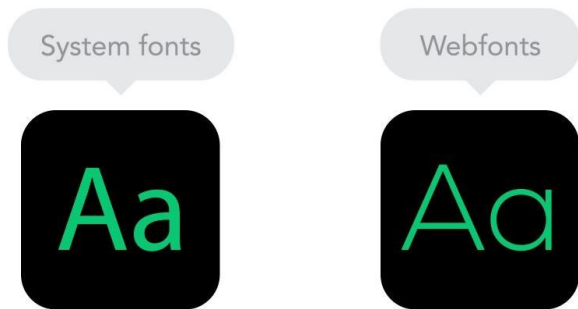
Viewport es el tamaño de la ventana del navegador.

1vw = 1% del ancho de la ventana gráfica.

System Font vs WebFonts

- **Fuentes de la Web:** son descargadas por lo que, cuantas más haya, más lento cargará el sitio.
- **Fuentes del Sistema:** más rápidas, pero si NO están en el cliente navegador del usuario se usa una por defecto.

Cuando estamos trabajando con dispositivos móviles tenemos que tener en cuenta que todo se carga.



Imágenes responsivas

Las imágenes responsivas son imágenes que se escalan bien para adaptarse a cualquier tamaño de navegador.

Si la propiedad CSS **width** se establece en 100%, la imagen responderá y se ampliará y reducirá.

Una imagen grande puede ser perfecta en una pantalla de computadora grande, pero inútil en un dispositivo pequeño. ¿Por qué cargar una imagen grande cuando tiene que reducirla de todos modos? Para reducir la carga, o por cualquier otro motivo, puede utilizar **media queries** para mostrar diferentes imágenes en diferentes dispositivos. [+info](#)