# SVG, transiciones y animaciones (Parte II)

## Conociendo las transiciones de CSS

Como vimos anteriormente el optimizar un SVG nos permitirá trabajar de mejor al interior de un proyecto web, mejorando por supuesto nuestra usabilidad. Pero ¿qué sucede con la del usuario?

Si hacemos memoria al comienzo del módulo hablamos sobre la importancia que ejercía el uso de las animaciones y como esta nos ayuda a guiar e interactuar al usuario dentro de la interfaz que creamos.

### ¿Qué son las transiciones?

Las transiciones son propiedades de CSS que nos permiten realizar cambios sobre valores de otras propiedades de CSS de manera suave y por un tiempo determinado. Con las transiciones, podemos lograr efectos como, por ejemplo, cambiar el ancho de un botón de manera progresiva y suave durante N segundos, determinados por el desarrollador/diseñador.

### ¿Cómo usar las transiciones CSS?

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

1. La propiedad CSS a la que desea agregar un efecto.

2. La duración del efecto.

Si no se especifica la parte de duración, la transición no tendrá efecto, porque el valor predeterminado es 0.

Ejemplo

El elemento rojo de 100px * 100px también ha especificado un efecto de transición para la propiedad de ancho, con una duración de 2 segundos:

```
div {
      width: 100px;
      height: 100px;
      background: red;
      transition: width 2s:
}
```

El efecto de transición comenzará cuando la propiedad CSS especificada (ancho) cambie de valor.

Ahora, especifiquemos un nuevo valor para la propiedad ancho cuando un usuario pasa el mouse sobre el elemento.

```
div {
      width: 300px;
}
```

## Aplicar transiciones a elementos SVG

Ahora, veremos cómo aplicar una transición a un elemento SVG que ya creamos con anterioridad. Para esto, procederemos a crear un documento HTML con el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Transiciones CSS</title>
   <style>
      .estrella {
         fill: lime;
         stroke: blue;C
         stroke-width: 5;
         transition: fill 2s;
      }
      .estrella:hover {
         fill: red;
      }
   </style>
</head>
<body>
   <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 210 500"
width="210" height="500">
   <polygon points="100,10 40,198 190,78 10,78 160,198" class="estrella"
/>
   </svg>
</body>
</html>
```

Luego, guardamos los cambios y abrimos el archivo recién creado en un navegador web. Veremos que hemos vuelto a crear un polígono con forma de estrella, fondo verde y borde azul.

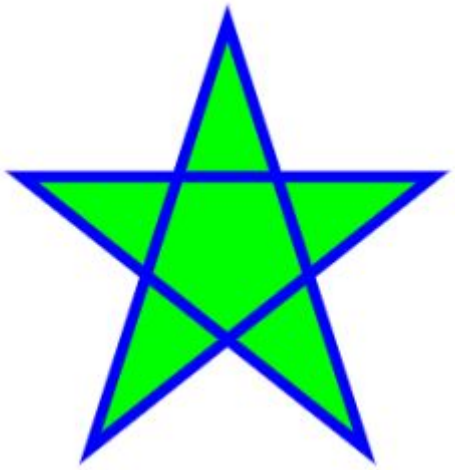

Imagen 1. Polígono con forma de estrella de fondo verde y borde azul.

Ahora, si pasamos el mouse por encima de este polígono, veremos que da lugar a una transición del color verde a un color rojo al cabo de 2 segundos, como el siguiente:

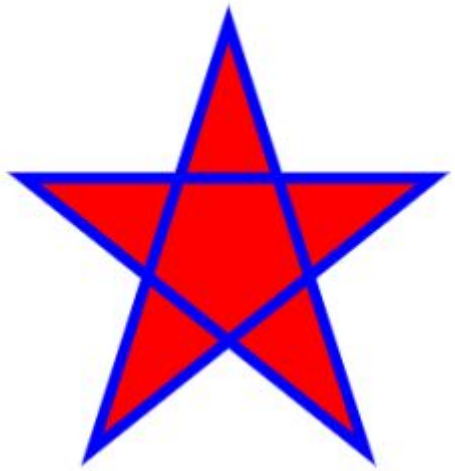

Imagen 2. Cambio de color de fondo del polígono.

Este efecto fue posible gracias a las siguientes modificaciones:

- Dentro de la clase **.estrella** , agregamos la propiedad **transition** con el valor **fill: 2s**. Esto nos quiere decir que el efecto de transición se aplicará a la propiedad **fill** (la que define el color de fondo de la estrella) al cabo de 2 segundos ( **2s** ).

- Seguido de la definición de la clase **.estrella**, incluimos la pseudoclase **:hover** a esta clase, la cual nos servirá para definir el estado final de la transición. Acá es donde establecemos la modificación del color de fondo con **fill: red**, para aplicar un color rojo.

## Aplicar varias transiciones a la vez

CSS nos da la opción de aplicar no sólo 1 transición, sino que más a la vez. Siguiendo con el ejemplo anterior, realizaremos la siguiente modificación en el código CSS:

```
.estrella {
   fill: line;
   stroke: blue;
   stroke-width: 5;
   transition: fill 2s, stroke 5s, transform 2s;
}
.estrella:hover {
   fill: red;
   stroke: orange;
   transform: scale(0.7)
}
```

Como se puede apreciar, dentro de la propiedad **transition** de la clase **.estrella**, tenemos definidas 3 transiciones: **fill** , **stroke** y **transform** , las cuales afectarán al color de fondo al cabo de 2 segundos, el borde al cabo de 5 segundos y las definiciones de transformaciones que se puedan aplicar (escalar, rotar, trasladar, etc.) al cabo de 2 segundos.

Por otro lado, dentro de la pseudoclase **:hover** para **.estrella** , tenemos los valores finales de transición: color rojo para el fondo de la estrella, el borde cambia al color naranjo y se aplica un escalado de su 100% a un 70%.

Si guardamos los cambios y posicionamos el puntero por sobre la estrella, veremos como se aplican en conjunto las 3 transiciones, dándonos un resultado como el siguiente:

Imagen 3. Polígono en conjunto de 3 transiciones.

## Agregar retraso a las transiciones

Si se desea, es posible generar un retraso en la ejecución de los efectos de transición. Para ésto, podemos valernos de la propiedad transition-delay , el cual recibe como valor el tiempo en segundos para el cual se esperará para la ejecución del efecto.

Por ejemplo, con la siguiente modificación en el código CSS:

```
.estrella {
   fill: line;
   stroke: blue;
   stroke-width: 5;
   transition: fill 2s, stroke 5s, transform 2s;
   transition-delay: 3s;
}
.estrella:hover {
   fill: red;
   stroke: orange;
   transform: scale(0.7)
}
```

Podemos lograr que, al mantener el puntero por encima de la figura de estrella durante 3 segundos, recién se inicie la ejecución de los efectos de transición.

También, disponemos de la propiedad **transition-timing-function** , la cual nos permite establecer la velocidad con la cual se realiza la transición del efecto. Existen aplicaciones de este efecto muy interesantes, las cuales pueden encontrar en el siguiente enlace.

## Organizando formas SVG

Comencemos definiendo la estructura de las formas usando los selectores de clase. En primer lugar movamos el primer **<path>** al interior del grupo. Esta forma parte de la ampolla de una lámpara.

```
<svg class="undefined">
   <g class="ampolla">
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
      <path d="comandos de forma" class="a"/>
   </g>
   <path d="comandos de forma" class="b"/>
   <polygon points="comandos de forma" class="c"/>
</svg>
```

Es importante acotar que las clases que usaremos tendrán como nombre los elementos que componen una lámpara común y corriente.

Bien, ahora que tenemos estos elementos cambiaremos la clase que define a todo el grupo y a sus hijos también. Cambiemos el nombre de la clase que compone al grupo por **.ampolla** y luego eliminamos todas las clases que incluyen los elementos **<path>**.

```
<g class="ampolla" >
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
   <path d="comandos de forma" />
</g>
```

En el **<path>** que se encuentra afuera con clase .b , lo mantendremos ahí y cambiaremos su clase por **.reflejo**.

```
<path d="comandos de forma" class="reflejo" />
```

Luego, cambiaremos la clase del elemento **<polygon>** por **.casquillo**.

```
 <polygon points="comandos de forma" />
```

Y para finalizar agruparemos todos las formas en un gran contenedor al cual llamaremos **.lampara** y eliminaremos la clase **.undefined** del contenedor **<svg>**.

Si ahora vemos nuestro código podremos distinguir de mejor manera el elemento al cual afectaremos.

```
    <svg xmlns="http://www.w3.org/2000/svg" x="0" y="0" width="400px"
height="400px" viewBox="0 0 53 53">
      <g class="lampara">
      <g class="ampolla">
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
        <path d="comandos de forma" />
      </g>
      <path d="comandos de forma" class="reflejo" />
      <polygon points="comandos de forma" class="casquillo"/>
    </svg>
```

Antes de continuar, cambiemos el nombre de las clases que agregamos en el elemento **<style>**. Primero cambiemos la clase **.a** por **.ampolla** , **.b** por **reflejo** y finalmente **.c** por **casquillo**.

```
<style>
   .ampolla {
      fill:#EFCE4A;
   }
   .reflejo {
      fill:#F7E6A1;
   }
   .casquillo {
      fill:#556080;
   }
</style>
```

Si vamos al navegador y recargamos la página veremos que los estilos funcionan correctamente.

## Nuestra primera transición

Bien, ahora mejoramos el flujo de trabajo de nuestro documento y comenzaremos a ver las propiedades que componen a las transiciones.

### Transition-property

```
.button:hover {
  box-shadow: 10px 10px black;
  transition-property: all;
}
```

La primera propiedad que veremos es transition-property. Esta nos permitirá definir qué propiedades podrán ser afectadas por una transición.

Si deseamos que la transición afecte sólo a un elemento podemos hacerlo revisando las propiedades disponibles para hacerlo. Estas las podremos encontrar en un artículo de MDN relacionado a esto. Por el caso contrario si deseamos que esta transición afecte a todos los elementos debemos dejar como valor el keyword **all**.

### Transition-duration

```
.button:active {
  background-color: gold;
  transition-property: all;
  transition-duration: 1s;

}
```

La siguiente propiedad que veremos es **transition-duration**. Esta propiedad nos permite definir la duración que tendrá la transición de un estado a otro. Asimismo, podremos definir valores en milisegundos (**ms**) y en segundos también (**s**).

Transition-timing-function

```
.button:active {
  background-color: gold;
  transition-property: all;
  transition-duration: 1s;
  transition-timing-function: ease-in;
}
```

Luego, nos encontraremos con una de las propiedades transición más difíciles entender pero que si logramos asimilarlas se transformarán en una gran aliada.

La propiedad **transition-timing-function** nos ayudará a definir la aceleración y movimiento que tendrá una transición mediante una función llamada **cubic-bezier()** o usando keywords con curvas predefinidas.

Conceptualmente la curva de bézier permite definir con precisión una curva siguiendo un plano cartesiano. En el caso de esta propiedad nos permitirá modelar la velocidad que tendrá la transición en función del tiempo que nosotros hayamos definido con la propiedad transition-duration.

Veamos gráficamente este efecto revisando una herramienta que podremos encontrar dentro de MDN. Ingresemos al siguiente link para crear nuestra primera curva de bézier.

Primera curva de bézier

Cuando estemos dentro, veremos un gráfico con dos ejes: El primero es el eje y el cual define la longitud de la curva y el segundo es el eje x que define el tiempo de duración de la curva.

Debajo veremos cuatro parámetros que representan a puntos que nos permitirán crear la curva. En el caso de la función **cubic-bezier()** esta podrá tener un valor mínimo de **0** y un máximo de **1**.

Creemos una curva de prueba que será lineal. Para eso agregaremos un primer parámetro de valor 0. Esto significa que el primer punto de la curva quedará en la coordenada 0 de x. El segundo valor será 0 a modo de interceptar las puntos. El tercer punto lo definiremos arriba en la coordenada 1 y el último parámetro también en 1.

Si revisamos el gráfico veremos que hemos formado una línea.

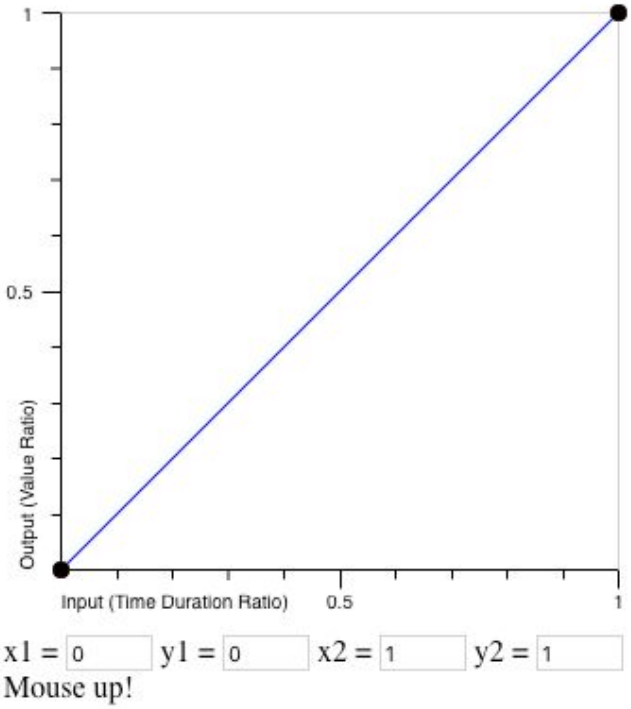


Imagen 4. Curva de Bézier.

El ejemplo visto anteriormente contiene los mismos parámetros que debemos agregar al interior de la función **cubic-bezier()**, o sea, x1, y1, x2 e y2.

De hecho, podremos obtener un resultado más visual usando estos mismos parámetros para encender la lámpara.

Ingresemos a nuestro nuestro editor de texto para crear esta transición. Agreguemos debajo de la clase **.ampolla**, esta misma clase pero incluyendo un pseudo selector **:hover**. Al interior de este agregaremos una propiedad fill con un color gold.

Debajo de este pondremos un **transition-property: all**; para afectar a todas la propiedades con esta transición, luego definiremos la duración de la transición en 1 segundo y finalmente agregaremos la propiedad transition-timing-function con un un valor **cubic-bezier()** de **`0, 0, 1, 1`**.

```
.ampolla {
  fill:#EFCE4A;
}

.ampolla:hover {
  fill: gold;
  transition-property: all;
  transition-duration: 1s;
  transition-timing-function: cubic-bezier(0, 0, 1, 1);
}
```

Si pasamos por arriba ampolla veremos que la lámpara comienza a encenderse de manera lineal.

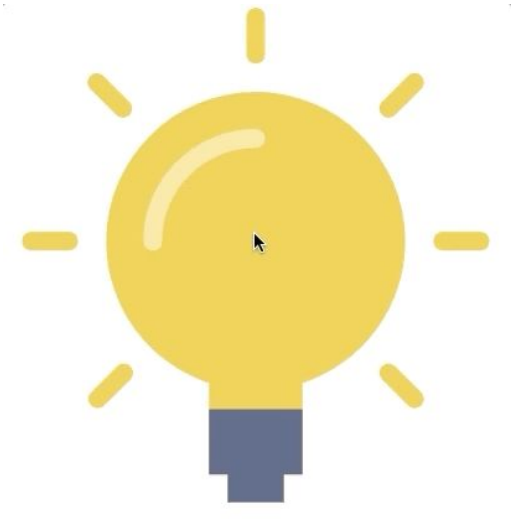

Imagen 5. Resultado cubic lineal.

Ahora si cambiamos el valor de primer punto de y a .5. veremos que la forma en que enciende la lámpara es más rápida.

```
    transition-timing-function: cubic-bezier(0,.5, 1, 1);
```

Asimismo, existen keywords que podremos usar para crear esta misma curva, pero de manera más fácil. Los keyword que podremos usar son ease, linear, ease-in, ease-out y ease-in-out.

Además, podemos definir estos parámetros de manera sencilla usando herramientas para crear curvas de bézier. Podemos usar una herramienta para definirla con el sitio cubic-bezier.com o usando pátrones definidos con la fantástica herramienta de easings.net.

Transition-delay

```
.button:hover {
  background-color: gold;
  transition-property: all;
  transition-duration: 1s;
  transition-timing-function: ease-in;
  transition-delay: 1s;
}
```

La última propiedad que veremos se llama **transition-delay** y nos permite definir cuánto tiempo de retraso tendrá una transición luego de que un elemento cambie de estado.

Probemos esta propiedad agregando debajo de **transition-timing-function** un **`transition-delay: 1s;`**.

```
.ampolla:hover {
  fill: gold;
  transition-property: all;
  transition-duration: 1s;
  transition-timing-function: cubic-bezier(0, .5, 1, 1);
  transition-delay: 1s;
}
```

Si recargamos veremos que ahora nuestra lámpara demora más en encender.

## Transition

Antes de terminar es bueno saber que existe una forma de agregar todas las propiedades vistas en una sola usando **transition**. Esta nos permite definir diferentes características como la definir a qué propiedad afectaremos con una transición, su duración, la velocidad del tiempo y el tiempo que demorará en comenzar la transición.

Los valores a agregar deberán seguir el mismo orden usado en la propiedades que agregamos, o sea, all, 1s, cubic-bezier(0, .5, 1, 1) y 1s.

Probemos su funcionamiento agregando la propiedad transition al interior de la clase .ampolla:hover, en donde agregaremos todos los valores de las otras reglas. Para finalizar borraremos las propiedades de transición anteriores.

```
.ampolla:hover {
  fill: gold;
  transition: all 1s cubic-bezier(0, .5, 1, 1) 1s;
}
```

Como vimos en toda la unidad, las transiciones son una gran forma animar los estados de un elemento, ya que nos permitirá definir la duración y velocidad que tendrá una interacción con el usuario.

## Creando animaciones con CSS: @keyframes

Luego de ver lo que podemos hacer con las transiciones es probable que sintamos que una pequeña interacción no es suficiente para satisfacer el objetivo de un usuario, y de hecho lo es, ya que las transiciones sólo nos permitirán definir la duración que tendrá el cambio de un estado a otro, pero no podremos definir en ellos cosas como repeticiones o precisar en donde queremos que ocurra un cambio de color o generar un movimiento.

Todo lo antes mencionado lo podemos hacer usando animaciones de CSS.

### ¿Qué son las animaciones de CSS?

Las animaciones en CSS nos permiten cambiar, de manera gradual y especifica, estilos de CSS asociados a un elemento. Estos cambios de estilo pueden ser tantos como sean necesarios y aplicados cuantas veces se requiera. Para realizar esto, las animaciones en CSS utilizan lo que se conoce como @keyframes , que consisten en reglas que nos permiten establecer en qué tiempos, de manera gradual, aplicaremos un estilo u otro. Estos @keyframes deben tener asociado un nombre, el cual debe ser referenciado dentro de la regla de estilo a la cual necesitamos aplicar la animación.

Veamos esto de mejor manera realizando un ejemplo muy simple.

En la misma hoja que creamos la transición creemos un botón con clase .button que este contenido por un  .

```
<div>
  <button type="button" class="button">Haz Click</button>
</div>
```

En **<style>** agregaremos un @keyframes  con nombre **mover** que contenga los siguientes elementos al interior:

- Agreguemos un 0% y en su interior pongamos en margin-left: 0;. Debajo agregaremos un 100% y después agregaremos la misma propiedad, pero ahora el margen será de 10em.

```
@keyframes mover {
  0% { margin-left: 0 }
  100% { margin-left: 10em; }
}
```

- Finalmente, agreguemos la clase **.button** y en su interior insertemos la propiedad animation con valor mover 3s ease-in-out 0s infinite.

```
.button {
  animation:mover 3s ease-in-out 0s infinite;
}
```

Guardemos y luego recarguemos el navegador. Si nos fijamos y aunque parezca mágico el botón se mueve hacia la derecha.

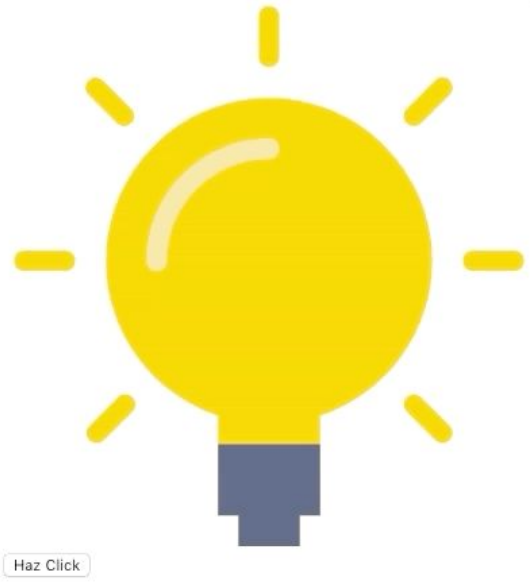


Imagen 6. Resultado animación de botón en movimiento.

Esto en esencia no ocurre por arte de magia sino que es la suma de diversos factores que provocan que un elemento se mueva o realice una acción.

Para entender de mejor manera el cómo crear una animación comenzaremos a identificar los elementos que lo componen comenzando por los @keyframes.

¿Qué son los @keyframes?

```
@keyframes opacidad {
  0% { opacity: 0; }
  60% { opacity: .5; }
  100% { opacity: 1; }
}
```

Los @keyframes, como vimos anteriormente, son el componente más importante al animar con CSS, ya que con él podremos controlar las etapas que tendrá nuestra animación y definir las reglas que accionaran en cada una de estas etapas.

Para hacerlo tendremos que agregar, en primer término, la regla @keyframes seguido de llaves.

```
@keyframes opacidad {
  /* Etapas de la animación */
}
```

Al interior, debemos definir las etapas que tendrá una animación. Estas etapas se refieren a las acciones o cambios que ocurrirán en un elemento afectado por esta animación.

```
@keyframes opacidad {
  0% { fill: #E6F4FE;}
  60% { fill:#EFCE4A;}
  100% { fill: gold;}
}
```

Ahora, analicemos lo que sucede aquí:

- La primera línea de nuestra animación, establece que al iniciar el valor de fill será de #E6F4FE .

- Luego, al momento de haber transcurrido el 60% del tiempo definido (es decir, si definimos 10 segundos como total, el 60% será 6 segundos) el valor de fill alcanzará el valor #EFCE4A de manera gradual. Finalmente, al alcanzar el 100% del tiempo establecido, el valor de fill será gold . Ésto, al aplicarlo a un elemento HTML, nos puede dar la sensación de estar iluminando de manera básica un objeto.

- De forma similar, podemos emular el desplazamiento de un elemento de izquierda a derecha con la propiedad margin-left :

```
@keyframes mover {
  0% { margin-left: 0;}
  100% { margin-left: 10em;}
}
```

De esta manera podremos hacer diferentes combinaciones como agregar en la mitad la animación un color diferente al dispuesto al principio o desaparecer el elemento agregando opacidad en 3/4 de la animación.

```
@keyframes mover {
  0% { margin-left: 0;}
  50% { color: red; }
  100% { margin-left: 10em;}
}
```

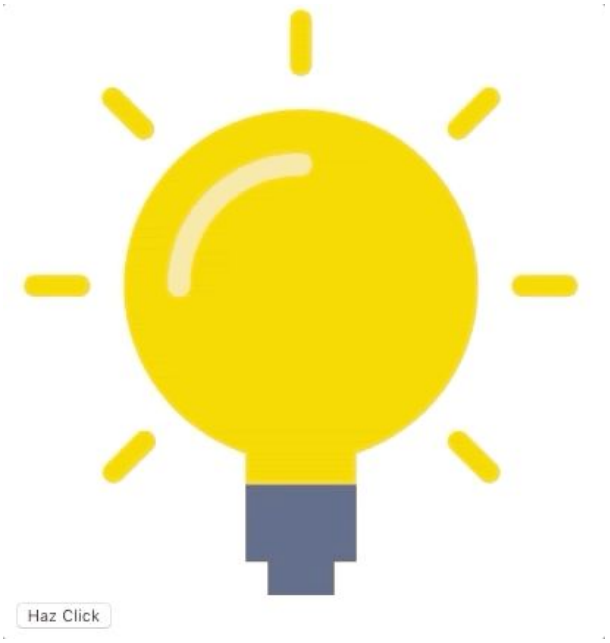


Imagen 7. Resultado botón en movimiento y cambio de opacidad.

En definitiva usar un @keyframes nos permitirá definir todas las acciones que ocurrirán a lo largo de una animación, por lo tanto es de suma importancia identificar las acciones que realizará nuestra animación en elemento específico.

## Creando animaciones con CSS: animation

Ahora que conocemos cómo definir las acciones que tendrá una animación el siguiente paso será conocer todas las propiedades que tendremos a disposición para definir diferentes aspectos de una animación.

CSS3 permite la animación de elementos HTML sin usar JavaScript o Flash.

Las propiedades que podemos encontrar dentro de **animation** son:

- animation-name
- animation-duration
- animation-timing-function
- animation-delay
- animation-iteration-count
- animation-direction
- animation-fill-mode
- animation
- keyframes

## Animation-name

Comencemos por la propiedad que nos será más familiar llamada **animation-name**, está especifica el nombre de la animación @keyframes.

Con esta propiedad podremos llamar a la animación creada en un @keyframes. Esto quiere decir que si deseamos llamar al @keyframes que creamos en la unidad anterior deberemos usar su nombre para insertarlo en el elemento que deseemos animar.

En este caso deberíamos agregar al interior de la clase **.button**, la propiedad **animation-name** y luego agregar el nombre del **@keyframes** el cual es mover.

Aprovechemos de borrar todo el contenido de la propiedad animation

```
.button {
  animation-name: mover;
}
```

Si guardamos y revisamos la animación veremos que esta no hace nada.

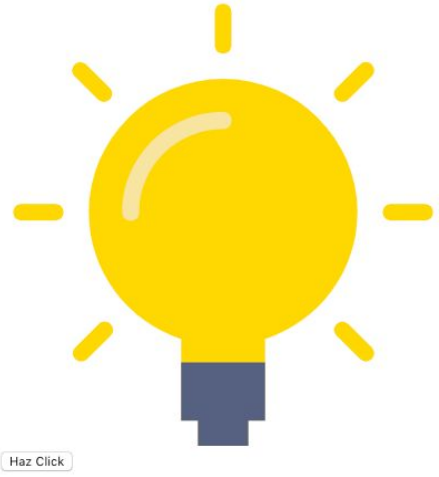


Imagen 8. Botón sin movimiento.

Esto es debido a que aún no hemos definido la duración que tendrá nuestra animación. Para hacerlo debemos usar la propiedad **animation-duration**.

### Animation-duration

Esta propiedad como mencionamos anteriormente nos permitirá definir la duración que tendrá una animación. Asimismo podemos usar segundos (s) o milisegundos (ms) para definir el tiempo.

Probemos esta propiedad agregando esta al interior de la clase .button. Definamos un valor de 1s para esta regla.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
}
```

Si revisamos en el navegador veremos que el botón se mueve hacia la derecha, y luego de llegar al final no hace nada más.

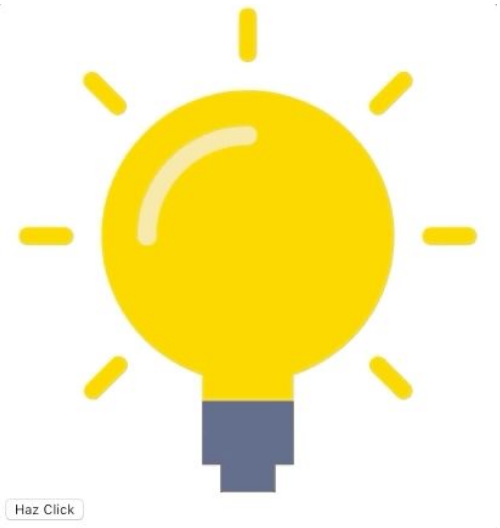


Imagen 9. Duración de animación.

Este es el comportamiento normal, el cual ya definimos en el @keyframes, pero si queremos controlar la aceleración de la animación, tal como lo hicimos con las transiciones, debemos usar la propiedad **animation-timing-function**.

### Animation-timing-function

Esta propiedad comparte muchas similitudes con transition-timing-function, ya que con ella podemos definir la velocidad que tendrá la animación en función al tiempo que definimos para esta.

Para definir esta velocidad podremos usar la curva de bézier con la función **cubic-bezier()** o usar los keywords ease, linear, ease-in, ease-out y ease-in-out.
Ingresa a `https://css-tricks.com/ease-out-in-ease-in-out/` para conocer más acerca de estas keywords

Para nuestro botón la mejor opción que tendremos es definir la velocidad con el keyword ease-in, ya que este comienza de manera leve y termina de manera rápida, tal como si el protagonista de nuestra ficticia obra corriera hacia la lámpara gigante.

Agreguemos en la clase .button la propiedad transition-timing-function con un valor de ease-in.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
  animation-timing-function: ease-in;
}
```

Revisemos el resultado de esta propiedad recargando el navegador.


Imagen 10. Ejemplo de botón con movimiento.

## Animation-delay

Especifica un retraso para el inicio de una animación.

En ocasiones, podemos requerir que nuestras animaciones sufran retrasos de tiempo al ejecutar las animaciones. Para lograr este efecto usaremos una propiedad llamada **animation-delay**.

Agreguemos esta propiedad a la clase **.button** y luego demos un tiempo de demora de 3s.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
  animation-timing-function: ease-in;
  animation-delay: 3s;
}
```

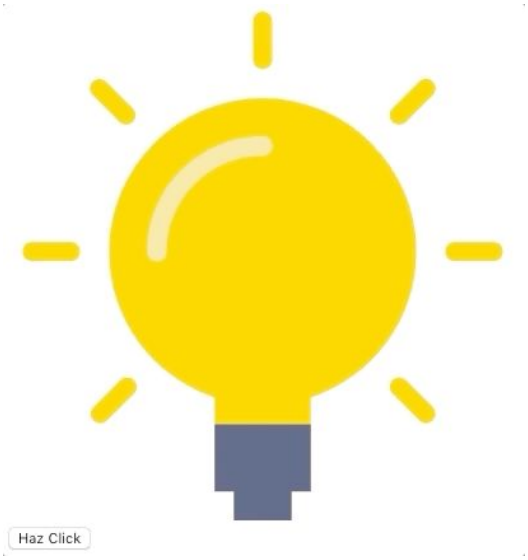


Imagen 11. Ejemplo de botón con movimiento.

### Animation-iteration-count

Especifica el número de veces que se debe reproducir una animación, esto a través de la propiedad **animation-iteration-count**.

Esta propiedad como bien dice su nombre es la encargada de definir cuantas veces queremos que se repita la animación.

Definamos que nuestro botón repetirá la misma animación tres veces.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
  animation-timing-function: ease-in;
  animation-delay: 3s;
  animation-iteration-count: 3;
}
```

En el caso que deseemos repetir infinitas veces la animación podremos usar el keyword infinite para hacerlo.

Animation-direction

Especifica si una animación se debe reproducir hacia adelante, hacia atrás o en ciclos alternativos. Nos permite definir la dirección que tendrá una animación.

En general las posibilidades que tendremos para definir la dirección son bastantes. Podemos escoger una dirección de izquierda a derecha (**normal**), de derecha a izquierda (**reverse**), de izquierda a derecha y luego de derecha a izquierda (**alternate**) o de derecha a izquierda y después izquierda a derecha **alternate-reverse**.

Usemos esta propiedad en nuestro botón agregando dentro de .botton una propiedad **animation-direction** con un valor de alternate para que el botón se mueva como una bola de ping pong.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
  animation-timing-function: ease-in;
  animation-delay: 3s;
  animation-iteration-count: 3;
  animation-direction: alternate;
}
```


Imagen 12. Animación con indicación de dirección.

Animation-fill-mode

Especifica un estilo para el elemento cuando la animación no se está reproduciendo (antes de que comience, después de que termine, o ambos).

En palabras simples la propiedad animation-fill-mode permite mantener los estilos definidos en la animación, pudiendo elegir los estilos del comienzo (backwards), o sea, lo que se encuentra al principio del @keyframes, los de al final (forwards) o ambos (both).

Para ver de mejor manera esta propiedad la agregaremos a nuestro botón y la propiedad animation-fill-mode y como valor utilizamos el keyword forwards.

```
.button {
  animation-name: mover;
  animation-duration: 1s;
  animation-timing-function: ease-in;
  animation-iteration-count: 3;
  animation-direction: alternate;
  animation-fill-mode: forwards;
}
```

Si recargamos veremos que después de la tercera repetición de la animación el botón desapareció. Esto se debe a que el última acción que definimos en la animación fue usar un **opacity:0**, por lo tanto el botón desapareció.


Imagen 13. Animación fill-mode.

Otro punto interesante es el hecho que todas las propiedades usadas hasta el momento pueden ser agregadas en una sola propiedad llamada animation, lo que reducirá enormemente la cantidad de líneas usadas.

Agreguemos esta propiedad debajo de animation-fill-mode: forwards; y en él pongamos los valores de todas las propiedades usadas usando el mismo orden que tienen. Finalmente borremos todas las otras propiedades y dejemos sólo animation.

```
.button {
  animation: mover 1s ease-in 3 alternate forwards;
}
```

## Prefijos de animaciones

Antes de terminar la unidad es importante siempre revisar la compatibilidad que tienen nuestros estilos con los navegadores. Por lo que es una buena idea revisar en Can I Use si es necesitamos usar prefijos de CSS para nuestro propiedades o no.

Por lo demás, igual los agregaremos debido a que algunas propiedades de las vistas en animaciones se encuentran en periodo de experimentación.

Así que vamos a la página principal de Autoprefixer, copiemos los estilos relacionados a la animación y peguemos estos en la herramienta. Cuando aparezca el código, copiaremos el resultado y lo pegaremos en el código.

```
.button {
  -webkit-animation: mover 1s ease-in 3 alternate forwards;
  animation: mover 1s ease-in 3 alternate forwards;
}

@-webkit-keyframes mover {
  0% { margin-left: 0;}
  50% { color: red; }
  75% { opacity: 0.5; }
  100% { margin-left: 10em; opacity: 0;}
}
@keyframes mover {
  0% { margin-left: 0;}
  50% { color: red; }
  75% { opacity: 0.5; }
  100% { margin-left: 10em; opacity: 0;}
}
```

Como pudimos ver en todo el módulo, ayudar al usuario usando tecnologías como SVG, animaciones o transiciones nos dará un plus gigantesco como profesionales, puesto que podremos realizar tareas más complejas al interior de una interfaz.

De hecho esto nos permitirá conocer más sobre campos relacionados a la investigación de usuarios como UX o UI, o adentrarse en campos más técnicos utilizando SVG para visualizar datos o implementar diseños más avanzados dentro de una maqueta.