

# Bootcamp Java Developer

Fase 2 - Java Web Developer  
Módulo 14

# Animaciones

# Trabajo con animaciones

Existen propiedades de CSS que permiten aplicar animaciones a los elementos HTML. Veremos algunas de ellas que nos servirán para generar interfaces más funcionales, eficientes y de alto impacto.



## animation-name

Esta es una propiedad obligatoria y es la única forma de vincular el elemento con la animación.

Hace referencia al nombre de la animación, que vamos a generar en la regla. Ejemplo:

```
@keyframes nombreAnimacion {  
  from {propiedad: valor; propiedad: valor; propiedad: valor;} /*cmo empieza la animación*/  
  to { propiedad: valor; propiedad: valor; propiedad:valor;} /*como termina la animación*/  
}  
  
selector { animation-name: nombreAnimacion;}
```

## Animation-duration

Es una propiedad obligatoria, porque **el valor predeterminado es 0**. Es decir, si no se indica ningún valor para duration, la animación nunca sucederá.

Hace referencia a cuánto dura la animación y sus valores posibles pueden estar expresados en ms o s. **Cuanto más tiempo se indique para la animación más lenta será.**

```
#panorama {  
  position: relative;  
  width: 3190px;  
  height: 350px;  
  animation-name: sky;  
  animation-duration: 25s;  
  animation-iteration-count: infinite;  
  animation-timing-function: linear;  
}
```

## Animation-delay

Esta propiedad hace referencia a **cuánto tarda la animación en comenzar** y sus valores posibles pueden ser en ms o s.



```
_animacion.scss x  personalizados.html
css > _animacion.scss > #cubo
15 #cubo {
16     width: 100%;
17     height: 100%;
18     position: absolute;
19     transform-style: preserve-3d;
20     transform: translateZ(-150px);
21     animation-name: girar;
22     animation-duration: 4s;
23     animation-delay: 2s;
24
25
26 }
```

# Animation-timing-function

Es el **ritmo o la forma en que sucede la animación**, los valores posibles son:

- **ease**

Valor predeterminado, inicia despacio, sucede rápido y termina despacio.

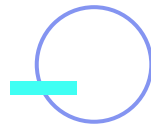
- **linear**

Mantiene la transición con el mismo estilo de velocidad en toda su duración.

- **ease-in**

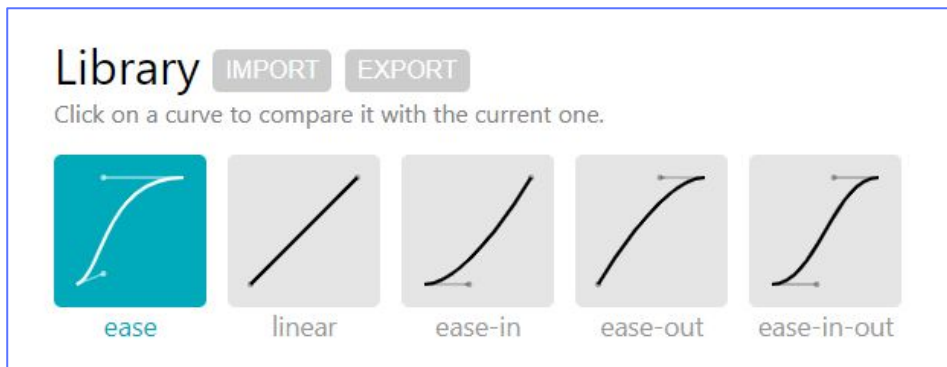
Inicia despacio, luego sucede normal y termina normal.

Para conocer más detalles sobre estos valores les recomendamos visitar la [página de Mozilla](#).



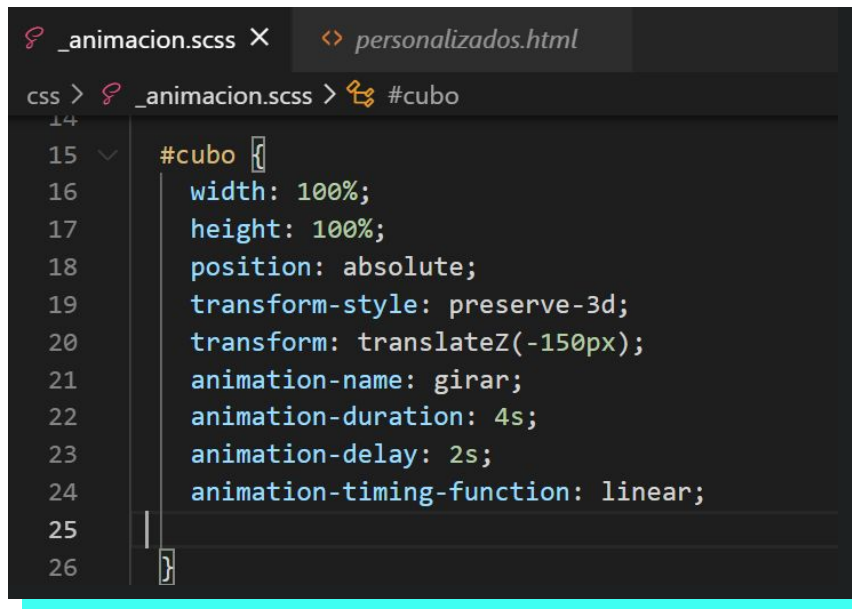
Con el **valor bezier** se indica, de manera personalizada, en una curva, cuál será la velocidad de la animación.

Es bastante complicado hacerlo, por eso existen aplicaciones como [Cubic-bezier.com](https://cubic-bezier.com).





En este ejemplo se observa el uso **de la propiedad:**



The image shows a code editor with two tabs: `_animacion.scss` and `personalizados.html`. The `_animacion.scss` tab is active, and the code is being viewed through a preview window. The code defines a CSS class `#cubo` with the following properties:

```
14  
15 #cubo {  
16     width: 100%;  
17     height: 100%;  
18     position: absolute;  
19     transform-style: preserve-3d;  
20     transform: translateZ(-150px);  
21     animation-name: girar;  
22     animation-duration: 4s;  
23     animation-delay: 2s;  
24     animation-timing-function: linear;  
25  
26 }
```



## Llamar a la animación

Si en el caso anterior quisiera crear la animación que va a ser llamada como vimos en la explicación anterior una opción podría ser esta:

```
@keyframes girar {  
  0% {  
    transform: translateZ(-150px) rotateY(0deg);  
  }  
  100% {  
    transform: translateZ(-150px) rotateY(360deg);  
  }  
}
```

## animation-play-state

Hace referencia al estado de la animación que puede ser:

- **running** (ejecutándose)
- **paused** (pausada).

Si se pausa la animación, se puede activar con JS.



```
<!DOCTYPE html>
<head>
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation-name: animacion;
  animation-duration: 3s ;
  animation-timing-function: linear;
  animation-delay: 1s;  }

@keyframes animacion {

  from { background: red; width: 300px; transform: rotate(34deg); }
  to { background: blue; height: 30px;  }

}

</style>

</head>
```

Podemos generar cortes en la animación:

```
<!DOCTYPE html>
<head>
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation-name: animacion;
  animation-duration: 6s ;
  animation-timing-function: linear;
  animation-delay: 1s;  }

@keyframes animacion {

  0% { background: red; width: 300px; transform: rotate(34deg); }
  20% { background: blue; height: 30px; }
  50% {background: green;}
  100% { background: lightblue; }
}

</style>
```

## shorthand

También, es posible utilizar un shorthand, es decir, **trabajar todas las propiedades en una sola línea:**

```
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation: animacion 3s linear ;

}

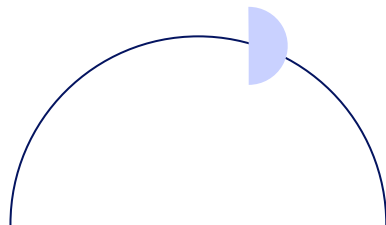
@keyframes animacion {
  0% { background: red; width: 300px; transform: rotate(34deg); }
  20% { background: blue; height: 30px; }
  50% {background: green;}
  100% { background: lightblue; }
}

</style>
```

## animation-direction

Define la dirección de la animación:

- **normal**  
Va del **from** al **to**, es decir, la dirección normal.  
Este es el valor predeterminado.
- **reverse**  
Va del **to** al **from**, es decir, al revés.
- **alternate**  
Va del **to** al **from**, y vuelve del **from** al **to**.
- **alternate-reverse**  
Va del **from** al **to**, y luego vuelve del **to** al **from**.



## animation-fill-mode

Permite saber **cómo termina la animación** y en **qué modo**:

- **Backwards**

**Comienza desde el from**, luego termina en el elemento **sin animación**.

- **Forwards**

Comienza normalmente, pero no vuelve al final al elemento sin animación sino que termina en el to (depende igual la propiedad **animation-direction** si termina **en el from o en el to**).

- **Normal**

Comienza el elemento común sin animación, **luego va del from al to, y vuelve al elemento sin animación**.



Ejemplo completo del uso de esta propiedad:

```
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation: animacion 3s linear forwards infinite alternate ; }

@keyframes animacion {

  0% { background: red; width: 300px; transform: rotate(34deg); }
  20% { background: blue; height: 30px; }
  50% {background: green;}
  100% { background: lightblue; }
}

</style>



</head>
<body>

<div> transiciones </div>
```



## Cortes en la animación

En las animaciones, se pueden generar cortes, es decir, inicialmente podemos tener:





```
@keyframes animacion {  
  
  from { background-color:  red; transform: scale(2,1.5)}  
  to { background-color :  green; transform: scale(0.5,1.5)}  
  
}
```

Sin embargo, para darle más potencia, CSS permite agregar **intervalos de animación**. Por ejemplo, las reglas anteriores son equivalentes a:

```
✓ @keyframes animacion {  
  
  0% { background-color: ■ red; transform: scale(2,1.5)}  
  100% { background-color : ■ green; transform: scale(0.5,1.5)}  
  
}
```

Podemos acrecentar la animación así:



```
@keyframes animacion {  
  
  0% { background-color:  red; transform: scale(2,1.5);}   
  20% { background-color:  yellow; transform: scale(2,1.5) rotate(20deg);}   
  50% { background-color:  rgb(255, 0, 191); transform: scale(1,1) rotate(-120deg);}   
  100% { background-color :  green; transform: scale(0.5,1.5);}   
  
}
```

## animation-play-state

Podemos generar una alternativa en los estados de la animación:

```
<!DOCTYPE html>
<head>
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation-name: animacion;
  animation-duration: 6s ;
  animation-timing-function: linear;
  animation-delay: 1s;  }

@keyframes animacion {
  0% { background: red; width: 300px; transform: rotate(34deg); }
  20% { background: blue; height: 30px; }
  50% {background: green;}
  100% { background: lightblue; }
}

</style>
```

# Actualización

## Propiedades adicionales

Ahora agregaremos propiedades adicionales a tener en cuenta para trabajar con CSS.

### object-fit

Esta propiedad es similar al uso que damos a **background-size** en sus valores posibles.

Sin embargo en este caso, lo hacemos sobre la propia imagen inserta.



## object-fit

Si tenemos inserta una imagen, en nuestro HTML:

```

```

Pero el tamaño indicado no es el que verdaderamente tiene la imagen, ésta se deformará. La propiedad **object-fit** permite adaptarla a formas diversas.

A continuación, probemos en nuestros estilos otros valores posibles.



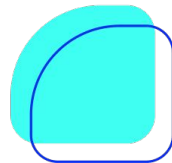
Ejemplo desde CSS

```
img {object-fit: fill;}  
img {object-fit: contain;}  
img {object-fit: cover;}  
img {object-fit: scale-down;}
```



No utilizar todos los valores en forma conjunta **ya que siempre prevalecería la palabra clave scale-down porque está en último lugar.**

Probar los valores de a uno **para apreciar su diferencia.**





# Animaciones y JavaScript

## Animation + JS

Es posible controlar una animación con **JavaScript**. Esta es la parte más interesante de todo lo aprendido.

Es importante que en todo caso, **o en la mayoría de los casos**, se trabaje con la propiedad **animation-play-state en paused** como se observa en la imagen de la derecha:



```
#cubo {  
  width: 100%;  
  height: 100%;  
  position: absolute;  
  transform-style: preserve-3d;  
  transform: translateZ(-150px);  
  animation-name: girar;  
  animation-duration: 4s;  
  animation-delay: 2s;  
  animation-play-state: paused; }
```

Luego, desde nuestro **archivo** y a través de **botones** o **cualquier otro elemento** con el que pueda interactuar el usuario, se activa la animación cambiando el valor de **animation-play-state**:



```
#cubo {  
  width: 100%;  
  height: 100%;  
  position: absolute;  
  transform-style: preserve-3d;  
  transform: translateZ(-150px);  
  animation-name: girar;  
  animation-duration: 4s;  
  animation-delay: 2s;  
  animation-play-state: paused; }
```

Desde el archivo de JavaScript, podemos modificar todos los valores de propiedades CSS que creamos conveniente. Siempre utilizaremos el método **style** para poder acceder a la propiedad en cuestión.

Si la propiedad tuviese un valor compuesto por ejemplo **background-color**, en lugar del guión utilizaremos **camelCase** (destacar con una mayúscula a la segunda palabra):

```
personalizados.html  JS animacion.js X
> JS animacion.js > ...
1  document.getElementById('cambiar').onclick = cambiar;
2
3  function cambiar() {
4  document.getElementById('caja').style.backgroundColor = 'red';}
5
```

Por ejemplo, en el caso del trabajo con animaciones una opción puede ser:

```
<> personalizados.html    JS animacion.js X
js > JS animacion.js > ...
1  document.getElementById('btn_girar').onclick = girar;
2  document.getElementById('btn_parar').onclick = parar;
3
4
5  ✓ function girar() {
6    document.getElementById('cubo').style.animationPlayState = 'running';
7    document.getElementById('cubo').style.animationDelay = '3s';}
```

## animation-iteration-count

Indica cuántas veces se reproduce la animación.

El valor **predeterminado es 1**. Se utilizan **números enteros** para representar la **cantidad de veces que la animación se va a reproducir**.

Es importante saber que contamos con el valor **infinite** para hacer que la **animación genere un loop**.



## iteration-count + direction

Cuando se trabaja con iteration count en valores **superiores a 1** es importante saber cómo será la animación. Si se deja el valor de **direction** en **reverse** o **normal**, se notará un **corte** que en general no es agradable.

```
#cubo {  
  width: 100%;  
  height: 100%;  
  position: absolute;  
  transform-style: preserve-3d;  
  transform: translateZ(-150px);  
  animation-name: girar;  
  animation-duration: 4s;  
  animation-delay: 2s;  
  animation-play-state: paused;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}
```

## animation-fill-mode

Ejemplo completo de animación que incluye esta propiedad:

```
<style>
div {
  width: 200px;
  height: 200px;
  border-radius: 20px;
  padding: 20px;
  background: blue;
  animation: animacion 3s linear forwards infinite alternate ;
}

@keyframes animacion {
  0% { background: red; width: 300px; transform: rotate(34deg); }
  20% { background: blue; height: 30px; }
  50% {background: green;}
  100% { background: lightblue; }
}
</style>
```



## El uso de steps para animaciones complejas

Un valor de **animation-timing-function** es el uso de **steps**. Éste nos **permite hacer cortes o temporizar la animación**, lo cual permite claramente generar animaciones más complejas.

En realidad, podíamos lograr antes un objetivo similar por ejemplo de la siguiente manera:

```
@keyframes animacion {  
  0% {  
    |   transform: translate(0px);  
  }  
  
  50% {  
    |   transform: translate(100px);  
  }  
  
  70% {  
    |   transform: translate(100px);  
  }  
  
  80% {  
    |   transform: translate(300px);  
  }  
  
  100% {  
    |   transform: translate(300px);  
  }  
}
```

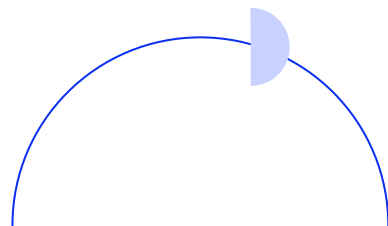
Sin embargo, el trabajo de la pantalla anterior no es completamente acertado y genera muchas dificultades al momento de lograr un resultado deseado.

Para comprender el trabajo con `steps`, tenemos que comprender en primer lugar su sintaxis:

```
steps(n, dirección)
```

El valor del número **(n)** indica cuántos cortes tendrá la animación, mientras que la **dirección** puede ser:

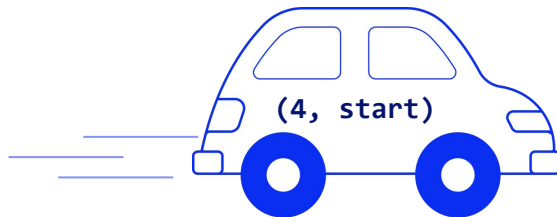
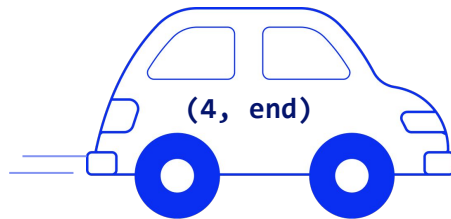
- **end**: este valor completa el tiempo de un paso antes de que la animación comience.
- **start**: este valor empieza la animación inmediatamente.



## Usos de valores diferentes de dirección

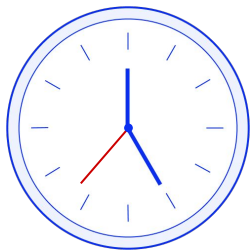
El uso de `start` y `end` implementada de manera conjunta sobre dos objetos con la misma animación, genera un efecto de animación compleja donde siempre se verá una diferencia entre ambos elementos.

Un ejemplo claro es una carrera de autos donde siempre un objeto se encontrará por delante de otro, como se muestra a la derecha.



## Conclusiones

Mostramos un ejemplo práctico de un reloj trabajado con **steps**. El mismo toma un tiempo de 60s en girar la aguja de los segundos, y éste, a su vez, se temporiza en 60 cortes para lograr el efecto deseado.



**Nota:** steps puede implementarse también en transiciones para hacer cortes en las mismas de la misma forma que hemos visto su implementación en animaciones.

```
.segundero {  
  animation: reloj 60s steps(60, end) infinite;  
}  
  
@keyframes reloj {  
  to {  
    transform: rotate(360deg);  
  }  
}
```

# Revisión

- Repasar los conceptos vistos de **animation**.
- Practicar todos los **elementos en forma individual**.
- Ver todos los videos y materiales necesarios antes de continuar



**¡Sigamos  
trabajando!**