

CSS Versus JavaScript Animations



By Paul Lewis

Paul is a Design and Perf Advocate



By Sam Thorogood

Evangelises Chrome and the mobile web in the Developer Relations team at Google.

There are two primary ways to create animations on the web: with CSS and with JavaScript. Which one you choose really depends on the other dependencies of your project, and what kinds of effects you're trying to achieve.

TL;DR

- Use CSS animations for simpler "one-shot" transitions, like toggling UI element states.
- Use JavaScript animations when you want to have advanced effects like bouncing, stop, pause, rewind, or slow down.
- If you choose to animate with JavaScript, use the Web Animations API or a modern framework that you're comfortable with.

Most basic animations can be created with either CSS or JavaScript, but the amount of effort and time differs (see also [CSS vs JavaScript Performance](#)). Each has its pros and cons, but these are good guidelines:

- **Use CSS when you have smaller, self-contained states for UI elements.** CSS transitions and animations are ideal for bringing a navigation menu in from the side, or showing a tooltip. You may end up using JavaScript to control the states, but the animations themselves will be in your CSS.
- **Use JavaScript when you need significant control over your animations.** The Web Animations API is the standards-based approach, available today in Chrome and Opera. This provides real objects, ideal for complex object-oriented applications. JavaScript is also useful when you need to stop, pause, slow down, or reverse.
- **Use `requestAnimationFrame` directly when you want to orchestrate an entire scene by hand.** This is an advanced JavaScript approach, but can be useful if you're building a game or drawing to an HTML canvas.

Alternatively, if you're already using a JavaScript framework that includes animation functionality, such as via jQuery's [.animate\(\).](#) [🔗](#) method or [GreenSock's TweenMax](#), then you may find it more convenient overall to stick with that for your animations.

Animate with CSS

Animating with CSS is the simplest way to get something moving on screen. This approach is described as *declarative*, because you specify what you'd like to happen.

Below is some CSS that moves an element 100px in both the X and Y axes. It's done by using a CSS transition that's set to take 500ms. When the `move` class is added, the `transform` value is changed and the transition begins.

```
.box {  
  -webkit-transform: translate(0, 0);  
  -webkit-transition: -webkit-transform 500ms;  
  
  transform: translate(0, 0);  
  transition: transform 500ms;  
}  
  
.box.move {  
  -webkit-transform: translate(100px, 100px);  
  transform: translate(100px, 100px);  
}
```

[Try it](#) [🔗](#)

Besides the transition's duration, there are options for the *easing*, which is essentially how the animation feels. For more information about easing, see [The Basics of Easing](#) guide.

If, as in the above snippet, you create separate CSS classes to manage your animations, you can then use JavaScript to toggle each animation on and off:

```
box.classList.add('move');
```



Doing this provides a nice balance to your apps. You can focus on managing state with JavaScript, and simply set the appropriate classes on the target elements, leaving the browser to handle the animations. If you go down this route, you can listen to `transitionend` events on the element, but only if you're able to forego support for older versions of Internet Explorer; version 10 was the first version to support these events. All other browsers have supported the event for some time.

The JavaScript required to listen for the end of a transition looks like this:

```
var box = document.querySelector('.box');
box.addEventListener('transitionend', onTransitionEnd, false);

function onTransitionEnd() {
  // Handle the transition finishing.
}
```



In addition to using CSS transitions, you can also use CSS animations, which allow you to have much more control over individual animation keyframes, durations, and iterations.

Note: If you're new to animations, keyframes are an old term from hand-drawn animations. Animators would create specific frames for a piece of action, called key frames, which would capture things like the most extreme part of some motion, and then they would set about drawing all the individual frames in between the keyframes. We have a similar process today with CSS animations, where we instruct the browser what values CSS properties need to have at given points, and it fills in the gaps.

You can, for example, animate the box in the same way with transitions, but have it animate without any user interactions like clicking, and with infinite repetitions. You can also change multiple properties at the same time:

```
/**
 * This is a simplified version without
 * vendor prefixes. With them included
 * (which you will need), things get far
 * more verbose!
 */
.box {
  /* Choose the animation */
  animation-name: movingBox;

  /* The animation's duration */
  animation-duration: 1300ms;
```



```

/* The number of times we want
   the animation to run */
animation-iteration-count: infinite;

/* Causes the animation to reverse
   on every odd iteration */
animation-direction: alternate;
}

@keyframes movingBox {
  0% {
    transform: translate(0, 0);
    opacity: 0.3;
  }

  25% {
    opacity: 0.9;
  }

  50% {
    transform: translate(100px, 100px);
    opacity: 0.2;
  }

  100% {
    transform: translate(30px, 30px);
    opacity: 0.8;
  }
}

```

[Try it](#) [↗](#)

With CSS animations you define the animation itself independently of the target element, and use the `animation-name` property to choose the required animation.

CSS animations are still somewhat vendor prefixed, with `-webkit-` being used in Safari, Safari Mobile, and Android. Chrome, Opera, Internet Explorer, and Firefox all ship without prefixes. Many tools can help you create the prefixed versions of the CSS you need, allowing you to write the unprefixed version in your source files.

Animate with JavaScript and the Web Animations API

Creating animations with JavaScript is, by comparison, more complex than writing CSS transitions or animations, but it typically provides developers significantly more power. You

can use the [Web Animations API](#) to either animate specific CSS properties or build composable effect objects.


JavaScript animations are *imperative*, as you write them inline as part of your code. You can also encapsulate them inside other objects. Below is the JavaScript that you would need to write to re-create the CSS transition described earlier:

```
var target = document.querySelector('.box');
var player = target.animate([
  {transform: 'translate(0)'},
  {transform: 'translate(100px, 100px)'}
], 500);
player.addEventListener('finish', function() {
  target.style.transform = 'translate(100px, 100px)';
});
```



By default, Web Animations only modify the presentation of an element. If you'd like to have your object remain at the location it has moved to, then you should modify its underlying styles when the animation has finished, as per our sample.

[Try it](#) 

The Web Animations API is a new standard from the W3C. It is supported natively in Chrome and Opera, and is in [active development for Firefox](#) . For other modern browsers, [a polyfill is available](#).

With JavaScript animations, you're in total control of an element's styles at every step. This means you can slow down animations, pause them, stop them, reverse them, and manipulate elements as you see fit. This is especially useful if you're building complex, object-oriented applications, because you can properly encapsulate your behavior.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.