CSS Animation for Beginners

Rachel Cope – December 04, 2014

DESIGN, CSS, ANIMATION

The human brain is hardwired to pay attention to moving objects. Because of this natural reflex to notice movement, adding animation to your website or app is a powerful way to draw users attention to important areas of your product and add interest to your interface.

When done well, animations can add valuable interaction and feedback, as well as enhance the emotional experience, bring delight, and add personality to your interface. In fact, *to animate* means *to bring to life.*

> *Emotional design's primary goal is to facilitate human-to-human communication. If we're doing our job well, the computer recedes into the background, and personalities rise to the surface.*
>
> *— Aarron Walter, Designing For Emotion*

In this post we're going to walk through the basics of CSS animation. You can follow along and view the CSS code for the example animations in this post.

The Building Blocks of Animations

CSS animations are made up of two basic building blocks.

1. **Keyframes** - define the stages and styles of the animation.
2. **Animation Properties** - assign the @keyframes to a specific CSS element and define *how* it is animated.

Let's look at each individually.

Building Block #1: Keyframes

Keyframes are the foundation of CSS animations. They define what the animation looks like at each stage of the animation timeline. Each `@keyframes` is composed of:

- **Name of the animation:** A name that describes the animation, for example, `bounceIn`.
- **Stages of the animation:** Each stage of the animation is represented as a percentage. `0%` represents the beginning state of the animation. `100%` represents the ending state of the animation. Multiple intermediate states can be added in between.
- **CSS Properties:** The CSS properties defined for each stage of the animation timeline.

Let's take a look at a simple `@keyframes` I've named "bounceIn". This `@keyframes` has three stages. At the first stage `(0%)`, the element is at opacity 0 and scaled down to 10 percent of its default size, using CSS transform scale. At the second stage `(60%)` the element fades in to full opacity and grows to 120 percent of its default size. At the final stage `(100%)`, it scales down slightly and returns to its default size.

The `@keyframes` are added to your main CSS file.

```
@keyframes bounceIn {
  0% {
    transform: scale(0.1);
    opacity: 0;
```

```
    }
    60% {
      transform: scale(1.2);
      opacity: 1;
    }
    100% {
      transform: scale(1);
    }
  }
```

(If you're unfamiliar with CSS Transforms, you'll want to brush up on your knowledge. Combining CSS transforms in the animations is really where the magic happens.)

Building Block #2: Animation Properties

Once the `@keyframes` are defined, the animation properties must be added in order for your animation to function.

Animation properties do two things:

1. They assign the `@keyframes` to the elements that you want to animate.
2. They define *how* it is animated.

The animation properties are added to the CSS selectors (or elements) that you want to animate. You must add the following two animation properties for the animation to take effect:

- `animation-name`: The name of the animation, defined in the `@keyframes`.
- `animation-duration`: The duration of the animation, in seconds (e.g., 5s) or milliseconds (e.g., 200ms).

Continuing with the above `bounceIn` example, we'll add `animation-name` and `animation-duration` to the div that we want to animate.

```
div    {
  animation-duration: 2s;
  animation-name: bounceIn;
}
```

Shorthand syntax:

```
div    {
  animation: bounceIn 2s;
}
```

By adding both the @keyframes and the animation properties, we have a simple animation!

Animation Property Shorthand

Each animation property can be defined individually, but for cleaner and faster code, it's recommended that you use the animation shorthand. All the animation properties are added to the same `animation:` property in the

following order:

```
animation    : [animation-name] [animation-duration] [animation-
timing-function]
[animation-delay] [animation-iteration-count] [animation-direction]
[animation-fill-mode] [animation-play-state];
```

Just remember for the animation to function correctly, you need to follow the proper shorthand order AND specify at least the first two values.

Note About Prefixes

As of late 2014, many Webkit based browsers still use the -webkit-prefixed version of both animations, keyframes, and transitions. Until they adopt the standard version, you'll want to include both unprefixed and Webkit versions in your code. (For simplicity, I'll only be using the unprefixed versions in my examples.)

Keyframes and animations with WebKit prefixes:

```
div    {
  -webkit-animation-duration: 2s;
  animation-duration: 2s;
  -webkit-animation-name: bounceIn;
  animation-name: bounceIn;
}
```

```
@-webkit-keyframes bounceIn { /* styles */ }
@keyframes bounceIn { /* styles */ }
```

To make your life easier, consider using Bourbon, a SASS mixin library which

contains up-to-date vendor prefixes for all modern browsers. Here's how simple it is to generate vendor-prefixed animations and keyframes using Bourbon:

```
div    {
  @include animation(bounceIn 2s);
}


@include keyframes(bouncein) { /* styles */}
```

Additional Animation Properties

In addition to the required **animation-name** and **animation-duration** properties, you can further customize and create complex animations using the following properties:

- `animation-timing-function`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`
- `animation-fill-mode`
- `animation-play-state`

Let's look at each of them individually.

Animation-timing-function

The `animation-timing-function:` defines the speed curve or pace of the animation. You can specify the timing with the following predefined timing options: `ease` , `linear` , `ease-in` , `ease-out` , `ease-in-out` , `initial` , `inherit` . (Or for more advanced timing options, you can

creating custom timing functions using cubic-bezier curve.)

The default value, if no other value is assigned, is `ease`, which starts out slow, speeds up, then slows down. You can read a description of each timing function here.

CSS syntax:

```
animation-timing-function : ease-in-out ;
```

Animation shorthand syntax (recommended):

```
animation : [animation-name] [animation-duration] [animation-timing-function];
animation : bounceIn 2s ease-in-out ;
```

Animation-Delay

The `animation-delay:` allows you to specify when the animation (or

pieces of the animation) will start. A positive value (such as 2s) will start the animation 2 seconds after it is triggered. The element will remain unanimated until that time. A negative value (such as -2s) will start the animation at once, but starts 2 seconds into the animation.

The value is defined in seconds (s) or milliseconds (mil).

CSS syntax:

```
animation-delay : 5s ;
```

Animation shorthand syntax (recommended):

```
animation : [animation-name] [animation-duration] [animation-timing-function]
[animation-delay];
```

```
animation : bounceIn 2s ease-in-out 3
s ;
```

Animation-iteration-count

The `animation-iteration-count:` specifies the number of times that the animation will play. The possible values are:

`#` - a specific number of iterations (default is `1`)

`infinite` - the animation repeats forever

`initial` - sets the iteration count to the default value

`inherit` - inherits the value from the parent

CSS syntax:

```
    animation-iteration-count    : 2;
```

Animation shorthand syntax (recommended):

```
animation: [animation-name] [animation-duration] [animation-timing-
function]
[animation-delay] [animation-iteration-count];
animation:  bounceIn 2s ease-in-out 3s 2;
```

Animation-direction

The `animation-direction:` property specifies whether the animation should play forward, reverse, or in alternate cycles.

The possible values are:

`normal` (default) - The animation plays forward. On each cycle the animation resets to the beginning state (0%) and plays forward again (to 100%).

`reverse` - The animation plays backwards. On each cycle the animation resets to the end state (100%) and plays backwards (to 0%).

`alternate` - The animation reverses direction every cycle. On each odd cycle, the animation plays forward (0% to 100%). On each even cycle, the animation plays backwards (100% to 0%).

`alternate-reverse` - The animation reverses direction every cycle. On each odd cycle, the animation plays in reverse (100% to 0%). On each even cycle, the animation plays forward (0% or 100%).

CSS syntax:

```
animation-direction : alternate ;
```

Animation shorthand syntax (recommended):

```
animation : [animation-name] [animation-duration] [animation-
timing-function]
[animation-delay] [animation-iteration-count] [animation-direction];
animation : bounceIn 2s ease-in-out 3
s 3 alternate ;
```

Animation-fill-mode

The `animation-fill-mode:` specifies if the animation styles are visible before or after the animation plays. This property is a little confusing, but once understood it is very useful.

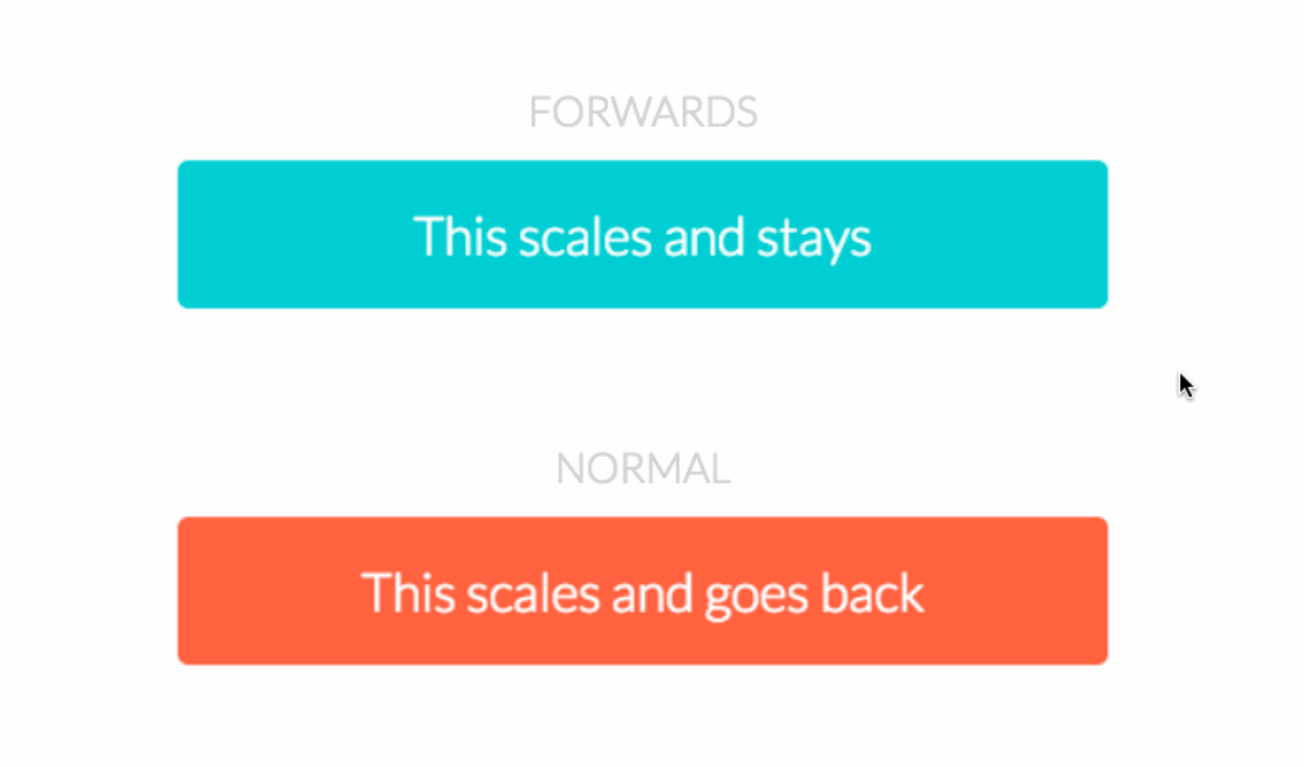
By default, the animation will not effect the styles of the element before the animation begins (if there is an animation-delay) or after the animation is finished. The `animation-fill-mode` property can override this behavior with the following possible values:

`backwards` - Before the animation (during the animation delay), the styles of the initial keyframe (0%) are applied to the element.

`forwards` - After the animation is finished, the styles defined in the final keyframe (100%) are retained by the element.

`both` - The animation will follow the rules for both forwards and backwards, extending the animation properties before and after the animation.

`normal` (default) - The animation does not apply any styles to the element, before or after the animation.



CSS syntax:

```
animation-fill-mode : forwards ;
```

Animation shorthand syntax (recommended):

```
animation : [animation-name] [animation-duration] [animation-timing-function]
[animation-delay] [animation-iteration-count] [animation-direction]
[animation-fill-mode];
animation : bounceIn 2s ease-in-out 3s 3 forwards ;
```

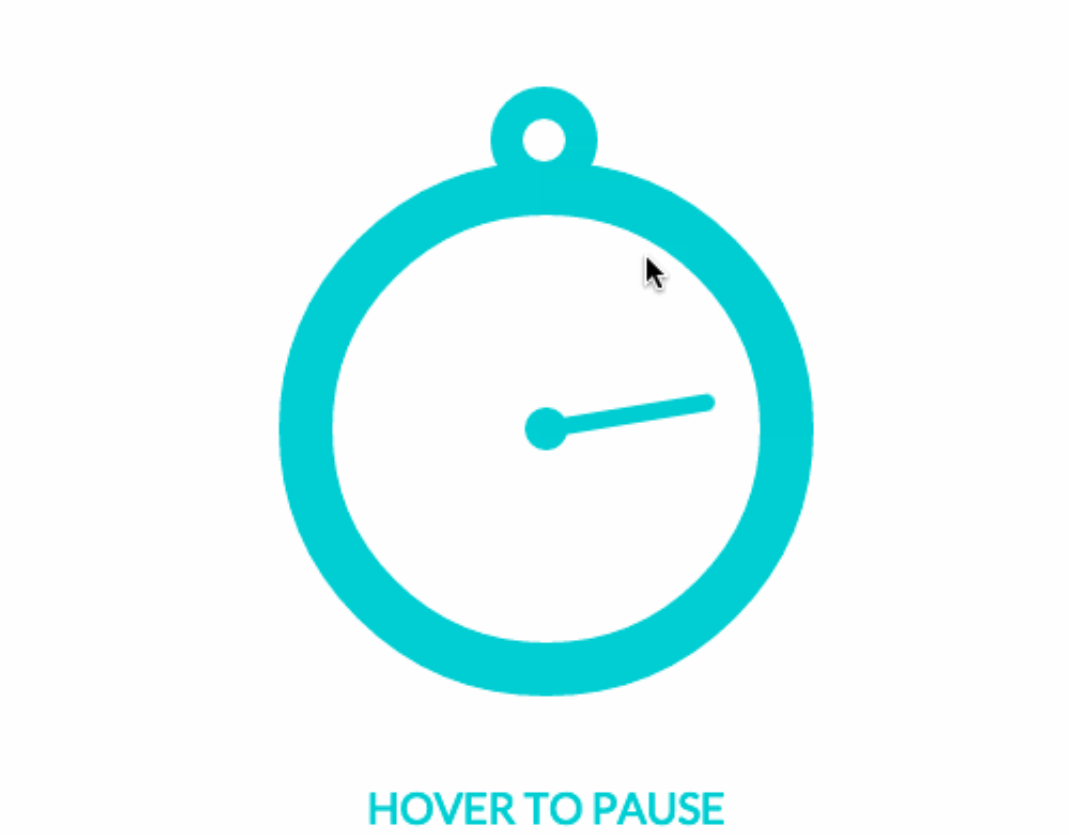
Animation-play-state

The `animation-play-state:` specifies whether the animation is playing or paused. Resuming a paused animation starts the animation where it was left off.

The possible values are:

`playing` - The animation is currently running

`paused` - The animation is currently paused



Example :

```
.div:hover {
  animation-play-state: paused;
}
```

Multiple Animations

To add multiple animations to a selector, you simply separate the values with a comma. Here's an example:

```
.div {
  animation: slideIn 2s, rotate 1.75s;
}
```

Go Forth and Animate

That's it! With those basic properties, the possible animations you can create are endless. The best way learn is to jump in and start animating.

Here are a couple of resources to get you started:

Upcase for Designers - an online learning community with courses on CSS animation, CSS transforms, and other front-end design and development techniques.

CodePen - a CSS playground where you can edit your code and immediately see your results.

Animate.css - a library with dozens of fun animations to get you started and use on your projects.

Rachel Cope
