**CSS Animations**

CSS animations are one of three different ways of bringing animations to life. CSS transitions, and scripted animations (created in JavaScript) are the other two ways. In this tutorial we will focus on CSS animations.

According the the Mozilla Developer Network:

"**CSS animations** make it possible to **animate** transitions from one **CSS**style configuration to another. **Animations** consist of two components, a style describing the **CSS animation** and a set of keyframes that indicate the start and end states of the **animation's** style, as well as possible intermediate waypoints."

We'll be writing our animation manually line by line to learn what is going at each step. By the end of this tutorial, you'll have learned enough to create a broom that flies up and down. Some might even call it a magical broomstick:

Let's Get Started!

The best way to learn about CSS animations is just to start coding an animation. Go ahead and create a new HTML document and add the following HTML and CSS to it (you can do this in a [codepen](https://codepen.io/" \t "_blank) if you would like). I'll give you a large chunk of code at once then we'll break everything down:

1. <!DOCTYPE html>
2. <html>
4. <head>
5. <title>Broomstick Animation</title>
7. <style>
8. body {
9. padding: 50px;
10. }
11. #container {
12. background-color: skyblue;
13. height: 300px;
14. text-align: center;
15. }
16. </style>
17. </head>
19. <body>
20. <div id="container">
21. <img id="broom" src="https://github.com/robgmerrill/online-course-images/blob/master/broom.png?raw=true"/>
22. </div>
23. </body>
25. </html>

Take a look at the results of the code that you just added. There isn't a whole lot to this page at this point. The main thing I want you to note is that we have an image element, and it has an id value of **broom**:

1. <div id="container">
2. <img id="broom" src="https://github.com/robgmerrill/online-course-images/blob/master/broom.png?raw=true"/>
3. </div>

Now, before we move on to the next step, go ahead and preview this page in your browser. If everything worked right, you will see a magical broom...standing boringly still:

Let's fix the boredom by animating our broom image. To do this, we are going to create a CSS animation. To create a CSS animation, you will need to complete two steps:

1. Set the **animation** property. In our case the sub-properties.
2. Define the **keyframes** that specify exactly how and when CSS properties get animated. Each keyframe describes how the animated element should render at a given time during the animation sequence.

We'll tackle both of these steps one at a time. First, we'll deal with our animation property, or animation sub properties. I'll show you how to use the shorthand animation property later. In your style block where your style rules currently live, add the #broom style rule underneath where your #container style rule lives. This targets our img tag with an id of broom:

1. #broom {
2. animation-name: fly;
3. animation-duration: 2s;
4. animation-iteration-count: infinite;
5. }

We aren't going to go over the detail of each sub property now but will return further along in the lesson.

Next, let's go to our next step and specify what our animation does by adding the keyframes. Go ahead and add the following @keyframes style rule just below where where your #broom style rule lives:

1. @keyframes fly {
2. 0% {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. 100% {
9. transform: translateY(20px);
10. }
11. }

Once you've added this style rule, go ahead and preview your page now. You should see your magical broom flying up and down. Great job!

Here is what happened

What you just did was define a CSS animation that caused your broom image to fly up and down. Now, let's stop and examine what exactly is going on at each step. You just copied what I had and saw the results. Let's take a few moments to go back and revisit what we just did and learn more about why we did it.

The first thing we will look at is the animation sub properties:

1. #broom {
2. animation-name: fly;
3. animation-duration: 2s;
4. animation-iteration-count: infinite;
5. }

The animation sub properties are responsible for setting your animation up:

1. animation-name: The name of the animation.
2. animation-duration: The duration of the animation. This should be in seconds, s, or milliseconds ms.
3. animation-iteration-count: The number of times your animation will loop.

We have named our animation fly. We have set the duration of the animation to 2 seconds. Finally, we are looping this animation an infinite amount of times.

If this was the only code we had to look at we wouldn't have a good idea what the animation does. We just know its name, how long it lasts and how many times it will run but we don't know what our animation will do. The actual animation is found in the @keyframes rule.

Let's look at our @keyframes rule to learn more:

1. @keyframes fly {
2. 0% {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. 100% {
9. transform: translateY(20px);
10. }
11. }

The first thing to notice when you look at our @keyframes rule is how it looks. On the outside, it contains the @keyframes declaration followed by a name, fly:

1. @keyframes fly {
2. 0% {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. 100% {
9. transform: translateY(20px);
10. }
11. }

On the inside, it contains style rules (aka the actual keyframes) whose selectors are percentage values 0% 50% 100% in this case (or the keywords **from** and **to**...we will talk about them later):

1. @keyframes fly {
2. 0% {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. 100% {
9. transform: translateY(20px);
10. }
11. }

These style rules, often referred to as **keyframe style rules**, are pretty much what you would expect. They just contain CSS properties such as **transform** whose value will get applied when the rule becomes active.

This tutorial isn't teaching the CSS property transform. The transform property applies a 2D or 3D transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements. In this case we will use the translateY() CSS function to move our item up and down along the Y axis. You can read more documentation about it [here](https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/translateY).

We are simply saying that at 0% we want our broom 20px down the Y axis.  At **50%** we want our broom 60px down the Y axis. Finally, at 100% we want our broom back at 20px down the Y axis.

At 0%, we can replace the percentage with the keyword from. At 100% we can replace our percentage with the keyword to.

1. @keyframes fly {
2. from {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. to {
9. transform: translateY(20px);
10. }
11. }

Despite the animation property being declared in another style rule and your keyframes being declared in their own @keyframes rule, they are dependent upon each other and will not function without the existence of both of them.

Let's start by first looking at how the animation-name property and the @keyframes rule are tied together.

The Name

The name you give your @keyframes rule acts as an identifier the animation property uses to know where the keyframes are. In this case the animation name is fly:

1. #broom {
2. animation-name: fly;
3. animation-duration: 2s;
4. animation-iteration-count: infinite;
5. }
7. @keyframes fly {
8. 0% {
9. transform: translateY(20px);
10. }
11. 50% {
12. transform: translateY(60px);
13. }
14. 100% {
15. transform: translateY(20px);
16. }
17. }

It is required that our animation-name property value and the name of our @keyframes rule to be exactly the same or our animation will not work.

Time Duration and the Keyframes

So now that we understand that our animation-name property value is how we connect to the correct keyframes we will now look at when a particular keyframe style rule becomes active. Hint: It has to do with the percentages.

When we defined the keyframe style rules inside our @keyframes rule, our selector wasn't an actual time value. It was a percentage value:

1. @keyframes fly {
2. 0% {
3. transform: translateY(20px);
4. }
5. 50% {
6. transform: translateY(60px);
7. }
8. 100% {
9. transform: translateY(20px);
10. }
11. }

**What these values represent is the percentage of the animation that has completed**. Using the values from our example, the 0% keyframe represents the start of our animation. The 50% keyframe represents our animation's mid-way point. The 100% keyframe represents the end of our animation.

We don't measure the animation by time but by percentage. It is the animation-duration property that gives us the value for time that our percentages use.

1. #broom {
2. ...
3. animation-duration: 2s;
4. ...
5. }

The animation-duration value not only specifies the total length of our animation. It also helps specify the time a particular keyframe will become active.

Taking our 2-second long animation as an example:

The 0% keyframe becomes active at the beginning after 0 seconds have elapsed. The 50% keyframe becomes active after 1 second has elapsed. The 100% keyframe becomes active at the end once 2 seconds have elapsed.

Looping

You can probably guess that the  animation-iteration-count property determines the number of times your animation will play. We have set it to infinite so that it continues to play over and over. Try a number such as 3 and you can see that your animation will only loop 3 times for the duration of 6 seconds.

The Shorthand Version

The animation property can be written in a much more concise way without naming all the animation sub properties. The shorthand variant for the longhand version we've seen so far looks as follows:

1. #broom {
2. animation: fly 2s infinite;
3. }

Conclusion

We learned all about how to declare an animation using the animation properties and how the @keyframes rule with its keyframe style rules work. What we've seen is just a fraction of everything you can do with CSS animations. There are more sub-animations you can try out to make your animations more flexible to what you are trying to do.

animation-timing-function: Configures the timing of the animation; that is, how the animation transitions through keyframes, by establishing acceleration curves.

animation-delay: Configures the delay between the time the element is loaded and the beginning of the animation sequence.

animation-direction: Configures whether or not the animation should alternate direction on each run through the sequence or reset to the start point and repeat itself.

animation-fill-mode: Configures what values are applied by the animation before and after it is executing.

animation-play-state: Lets you pause and resume the animation sequence.