2.6.3 Experiment with CSS Animation

In the old days of the web, animation would have required coding with Adobe Flash or JavaScript. Flash is now **deprecated**, meaning browsers have stopped or will soon stop supporting it. Of course, JavaScript is here to stay, but modern CSS has evolved to the point where it has basic animation built into it. So let's try to make this animation work using CSS.

Two CSS properties help with animation: `animation` and `transition`. We'll experiment with each one before deciding which one best fits our needs.

HIDE PRO TIP

When experimenting with new features, you can use an online code editor like JSFiddle (https://jsfiddle.net/) or CodePen (https://codepen.io/pen/) instead of messing with your current project or creating extra HTML and CSS files on your computer.

Using Keyframes and the Animation Property

Let's look at `animation` first, which involves setting up a series of keyframes. **Keyframes** define the more important moments in a character's or object's movement. For instance, a falling ball would have two pretty important keyframes: the ball at the top of its descent and at the moment it hits the ground. The frames in between can be filled in later by an artist or computer software to make a complete animation.

In CSS, the browser will be the one to fill in those in-between frames. We only need to define the major keyframes, or starting/stopping points. Keyframes can be added to a style sheet using the `@keyframes` at-rule.

REWIND

An **at-rule** instructs the CSS how to behave. Can you remember the other at-rule we used previously? If you guessed `@media`, you're right! We used it to define conditional CSS rules based on viewport size. Only `@media` and a few other CSS at-rules are considered conditional. For more information, see the **MDN web docs on CSS at-rules (https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule)**.

Here's an example of the `@keyframes` at-rule at work:

```
@keyframes fade-in {
  from {
    opacity: 0; /* fully transparent */
  }

  to {
    opacity: 1; /* fully opaque */
  }
}
```

We named the keyframes group `fade-in` because that's the intention of our animation, but the name can be whatever we want. Within the curly brackets, we define the keyframes and what properties should change at each of these steps. We only specify two steps—a starting and stopping point—represented by the keywords `from` and `to`.

To clarify, we want the `opacity` to start at 0 and then animate *from* 0 *to* 1 over a certain period of time. If we set this animation to run for four seconds, the browser will automatically calculate the in-between values. At two seconds, the animation is halfway through, so the `opacity` should be half of the `to` value (0.5).

Another example might be:

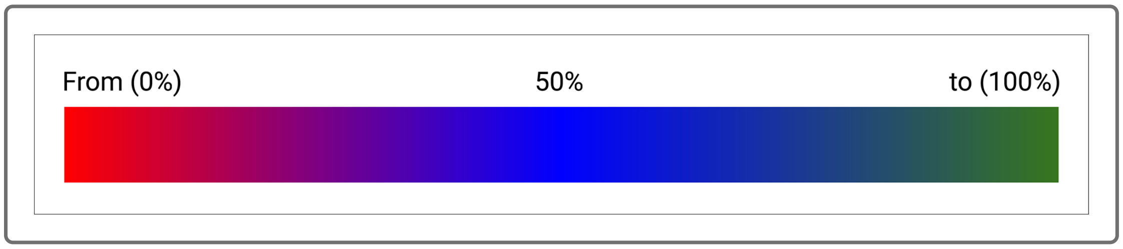
```
@keyframes color-shift {
  from {
    background-color: red;
  }

  50% {
    background-color: blue;
  }

  to {
    background-color: green;
  }
}
```

This time, we have three keyframes/steps. There's no `halfway` keyword in CSS; we can use percentages (50%) instead. We could expand this even more and define keyframes at 25%, 30%, 40%, etc. In this example, we're animating the `background-color` property to change from red to blue first, then from blue to green.

The following image illustrates this progression:



Of course, these keyframes don't do anything on their own. We still have to apply them to an element. For example, if we wanted every `<div>` to fade in, we would reference the name of that keyframes group in conjunction with the `animation-name` property:

```
div {
  animation-name: fade-in;
  animation-duration: 1s; /* animation lasts 1 second */
}
```

Using a second declaration, `animation-duration: 1s`, we can specify how long the animation takes to play. By default, the animation will only play once unless we add another declaration to define how often it should repeat. Why not forever?

```
div {
  animation: fade-in 1s; /* shorthand version */
  animation-iteration-count: infinite;
}
```

Combined with pseudo-classes, we could even withhold the animation until the element enters a certain state:

```
div:hover {
  animation: fade-in 1s infinite; /* more shorthand version */
}
```

DEEP DIVE ▲

DEEP DIVE

Start with an existing [JSFiddle code example (https://jsfiddle.net/5r74m6kq/)](https://jsfiddle.net/5r74m6kq/) and reference the [MDN web docs on CSS animation (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations)](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations) to see what else is possible.

Using the Transition Property

Let's set `animation` aside for a moment and see what the `transition` property has to offer now. This property doesn't use keyframes. Instead, it specifies the speed at which other CSS properties change. Consider this example:

```
div {
  font-size: 20px;
}

div:hover {
  font-size: 200px;
}
```

When your mouse hovers over the `<div>`, the font will abruptly change from `20px` to `200px`. Using `transition`, however, we can slow that down:

```
div {
  font-size: 20px;
  transition-property: font-size;
  transition-duration: 2s;

  /* or shorthand */
  transition: font-size 2s;
}

div:hover {
  font-size: 200px;
}
```

Note that the `transition` property was put on the `div` rule and not `div:hover`. This is because we want the transition from `200px` back to `20px` to also be animated. Try moving the `transition` to the `:hover` state and see what happens.

Did you notice that we declared a `transition` on the `font-size` property only? To animate multiple properties at once, simply omit the property name(s):

```
div {
  color: purple;
  font-size: 20px;
  transition-duration: 2s; /* will apply to both color and font-size *
}

div:hover {
  color: black;
  font-size: 200px;
}
```

DEEP DIVE

DEEP DIVE

Start with an existing **JSFiddle code example (https://jsfiddle.net/L0mw9h7k/)** and experiment with some of the other properties listed on the **MDN web docs for CSS transitions (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions)**.

We've now looked at two different animation techniques, so the next question is...

Which One Should We Use?

These were fairly high-level examples of CSS animation, and Run Buddy's request seems a little daunting in comparison.

Watch the demo animation from the preview again. Notice how the background image has an animated color gradient, and the trainer's name and role slide in at different times. Which technique, `animation` or `transition`, would best accomplish this?

The `transition` property makes the most sense for a few reasons:

- There are only two points of animation for each element, so there's no need for multiple keyframes.
- Elements should animate back to their normal state when the mouse leaves. The `animation` property doesn't give us an easy way to "go backwards."
- The animation happens only once on hover. The `animation` property is great for repeating/looping animations, which we don't need here.

Now that we've determined which tools we should use, let's dive in. We'll be a little more hands-off for the rest of the lesson to give you a chance to shine, but you totally got this!