So far, we've seen how a transition helps animate an element from one state to another. Let's look at the second way we can bring animation to our designs, using keyframes and the `animation` property. **Keyframes** are a part of CSS that we could use to create more complex animations.

While a transition might help us move one state to another, keyframes go further and give us a tool to create animations that go through as many different states as we need them to. Keyframe animations can also play automatically, this means they don't rely on a change of state.

In the transition examples, we relied on a hover action, or focus, or applying a class with JavaScript, to kickstart the transition movement. With keyframes, we can have animations that play automatically, when a page loads, or when an element is added to a page. Keyframes also give us a lot more control over the way the animations work.

We can set the direction, such as whether the animation plays forward or back, and we can control how many times it repeats. The control given to us by keyframe animations allows for much more complex and nuanced animations. In this lesson, we'll see some examples of this.

Let's take a look at the keyframe syntax:

```
@keyframes mymove {
    0%   {top: 0px;}
    50%  {top: 100px;}
    100% {top: 0px;}
}
```

It starts with the `@keyframes` keyword, that tells us, and the browser, that we're creating a series of keyframes, followed by the name of this set of keyframes. Then, inside the outer most curly braces, we define our keyframes.

Keyframes are a series of states that the browser steps through one at a time. We use them to set out each of the important steps along an animation and the browser does the tweening, filling in the gaps between the keyframes with animation. In this example, we have three keyframes set at 0%, 50% and 100%.

When we apply this set of keyframes to an element, the browser starts with the CSS defined at 0%. Then looks at the next keyframe, which is 50% in this case, and transitions the state to what we define in the 50% keyframe.

It then looks at the CSS inside the 100% keyframe, and transitions to that. The sequence we define here can be a looping animation, which means it'll go back to the start and play again, or it might stop at 100% or even alternate back and forth. If we think of the keyframes as a definition of what the animation is, then we need to have some way of applying these keyframes to an element on our page.

Let's take a look at how we can use the animation property to apply keyframes. Keyframes aren't much use to us by themselves. They're just the shape of the animation. We need a way to tell the browser to apply the keyframes to an element, and we do this using the `animation` property.

```
div {
    animation: mymove 2s 1s ease-out forwards;
}
```

First we have the name (`mymove`) - this is the name we used for the keyframes earlier.

Next we say how long an animation will run for; in this case 2 seconds, this is the duration.

After we set the duration, we then specify delay. This means that the animation won't start straight away, but instead will wait one second before starting to run through the sequence of keyframes. We then specify a transition timing-function, in this case "ease-out". Again, we'll go into what these are all about later.

Lastly, we specify something called "fill-mode". This is how we tell the browser, whether to revert to the original styles of the element after the animation has completed, or whether the end state of the keyframes, is the style we want to apply. This is set to "forwards", which means that we want the final keyframe, which is usually defined as 100%, to be the style that sticks around after the animation finishes.

So that's the properties we have to work with. Let's take a look at some examples.