Mastering CSS with Mark Lassoff

Written by Russ Eby and Mark Lassoff

Section 9: Working with CSS Sprite Sheets

The code in this section actually displays two separate animations, one superimposed over the other. Let's take a look at the code and learn how to do sprite animation with CSS. This animation is actually executed only by CSS with no assistance from JavaScript.

We'll start by examining the little bit of HTML used for this animation.

HTML

We have a `div` for the background.

```
<div id="background"></div>
```

We also have a div for the sprite, which is the character who is walking.

```
<div id="sprite"></div>
```

That's the whole display, just these two logical divisions.

Next, we have `audio` autoplay, for the music.

```
<audio id="music" autoplay>
   <source src="walk_music.mp3">
</audio>
```

And then an `iframe` autoplay, for the music.

```
<iframe src="walk_music.mp3" allow="" style="display:none"
id="iframeAudio">
</iframe>
```

Now you may be wondering why I'm using the `audio` tag and the `iframe` tag to play the music. The reason is that in Chrome autoplay no longer functions.

People were getting annoyed with sites that loaded and audio immediately starting. Sometimes, they wouldn't even know where the audio was coming from. The iframe is a hack for the Chrome browser to let you autoplay sound. Don't tell anybody I told you that. :)

The `audio` autoplay still works in other browsers, at least at the time of writing this.

On to the CSS…

CSS

First thing I did was set the margin: 0;, I wanted everything to go edge to edge for the animation, and I didn't want the margin space between the side of the browser and the content, which I find annoying.

```
body {
   margin: 0;
}
```

Background is our first logical division.

```
#background {
```

In the background, I placed the background image, `Ocean.jpg`. This image is designed for precisely this type of animation. If you look at the page that we created, we're only showing a small part of the image at any given time. We are also moving the image to the left very slowly.

```
   background-image: url(Ocean.jpg);
```

Then we set the width and height to the size of the background image.

```
   width: 100%;
   height: 1022px;
```

`z-index` is essentially how layers fall on top of each other. (This is often called *stacking order*.) The higher the number, the more likely it is to be on top. The negative `z-index` ensures that the background image will be on the bottom, and anything we put on top of it will be visible.

```
   z-index: -1;
```

The Animation

We'll be referring to animating the background as the `back` animation. Further down will be defining the `back` animation keyframes. It's a 30-second animation that continues infinitely.

We could change the timing of the animation. For example, we could change it to 60 seconds. And now the background image will slow down. Or if we change it to 5 seconds, it'll speed up.

For the animation to have a real feel, you need to time it correctly. I'm not an animation expert, but timing is, and the best thing to do is judge the timing visually. If something is moving too fast or too slow, it'll spoil the effect.

Instead of `infinite` we could set a specific number of times we want the animation to display by substituting an integer for the word `infinite`.

```
animation: back 30s infinite;
```

This next line allows us to set easing.

Easing is the acceleration and deceleration of items that are animated. We can set easing to `ease-in` or `ease-out` as well as linear , which is where the speed is constant the entire time.

The reason for picking `linear` is our character isn't accelerating or decelerating. I wanted the background to move at the same speed. If you were animating a vehicle that is moving, then you'd want to take advantage of the easing because it would accelerate and decelerate, and speed is not constant.

```
    animation-timing-function: linear;
}
```

A `keyframe` is a point at which we have a stage of the animation displayed. Essentially we're describing for the browser where the edges of each frame of the animation are within the graphic.

So the starting place has the `background-position: 0 0;` with the left edge of the image on the left edge of the screen. And we are telling the browser to move the left edge of the image to position -3067px. The second is the y (height), so we don't want the image to move up or down, just to the left. Above we said for the animation to take 30 seconds so it'll take 30 seconds to make the change.

```
@keyframes back {
    100% {
        background-position: -3067px 0;
    }
}
```

Next is the Sprite `div`.

Looking at the sprite image for the character. This is a six-frame repeating animation of this character moving their feet and walking. And unlike the background where we are slowly scrolling through the entire image, here we are showing a small portion of the image. And we are showing one frame after another. We are not sliding them through. We are changing the frames through the period of the animation giving the illusion they are walking. This is a slightly different animation here with the character.

```
#sprite {
```

The first thing we need to do is position the `div`. We used absolute to place the character 550 pixels from the top and 330 pixels from the left. Which puts the figure on the ground. I needed to experiment a little bit to get the character in precisely the right spot. If we changed the `top` to 500px, then they'd be floating in the air, and that wouldn't make any sense. The character will stay in that spot no matter how wide the window is.

```
position: absolute;
top: 550px;
left: 300px;
```

256 pixels is the width and height of our `div` but I didn't come by that by accident. Let's retake a look at the image for the walk. Each one of the cells is 256 pixels by 256 pixels. That's why the `div` the character lives in is 256 pixels by 256 pixels.

```
width: 256px;
height: 256px;
```

The `background-image` is using a URL at `walk.png`.

```
background-image: url(walk.png);
```

And then the animation. The keyframe for the animation is a sprite. We've had the character go through the cycle in .6 seconds. Which means each cell is on the screen for a tenth of a second. I experimented with this also to try to sell it, so it looked like walking. A typical walking animation would have many more frames. But this only has 6 frames, so not much to work with. By changing the time, we can slow it down or speed it up.

`steps(6)` instead of scrolling through the background, we're doing this in steps; 256-pixel

steps. Moving through the image 256 pixels at a time. 256 pixels is a sixth of the image.

`infinite`, so the animation will keep going, over and over.

```
  animation: sprite .6s steps(6) infinite;
```

The `z-index` is one since it's higher than -1 (the background), the character will be on top.

```
  z-index: 1;
}
```

Just like the background keyframe, we're changing the background-position so it'll move from the starting point (assumed 0%) of `background-position:0 0;` and end up at 100% of -1356px. Since the image is (256 times 6) 1536 pixel long, it'll move the entire image. If we took out the steps, we'd see the image slide past, which wouldn't look very good. But the steps will jump the image 256px every tenth of a second.

```
@keyframes sprite {
  100% {
    background-position: -1536px 0;
  }
}
```

We can have sprite sheets with more than one graphic. I don't think it's necessary to make these large sprite sheets with all sorts of graphics. It just makes it more difficult for the developer.

And then we just have the sound which I know you like. :)

I hope you enjoyed this segment, and you learned about sprite animation with CSS.

Submit this

I mentioned that the `@keyframes` is a set of points in the animation that you create, and the browser will morph the styles from one location to another. So let's try some out.

Let's assume we have a `div` on the screen.

```
div {
  width: 100px;
  height: 100px;
  background: red;
  animation: moveIt 1s infinite;
}
```

Here I'm creating four frames. 0% and 100% you'll notice are the same. Since I want the start and end to be the same as the starting point, I can actually remove those. So if I just left 33% and 66% in place, the animation would look precisely the same.

```
@keyframes moveIt {
  0% {
    background: red;
    width: 100px;
    height: 100px;
    border-radius: 0;
  }

  33% {
    background: darkred;
    width: 110px;
    height: 110px;
    border-radius: 10px;
  }

  66% {
    background: pink;
```

```
    width: 90px;
    height: 90px;
    border-radius: 100px;
  }

  100% {
    background: red;
    width: 100px;
    height: 100px;
    border-radius: 0px;
  }
}
```

Let's try this out, using just some `div`s make a sunrise.

- Take a `div` and color it blue for the sky. Let's say 500px by 500px.
- Turn a `div` into a circle, using border-radius and color it yellow for the sun.
- Make sure the sky has `position: relative` and the sun has `position: absolute` so that you can move the sun around the sky.
- Make keyframes that move the sun across the sky.
- Remember to use `z-index`, so the sun stays in front of the sky.
- For an added challenge, add a `div` for the ground that the sun starts behind, and then rises into the sky and settles back down behind the ground.
- Next challenge, change the sun's color, so it starts red, and when it's up in the sky, it then turns yellow and when it settles back down turns red again.

Don't over think this. Start with creating the `div` and step through the frames. As you type the rules, check the browser often.

Most likely your HTML will be something like this.

```
<div id="sky">
   <div id="sun"></div>
   <div id="ground"></div>
</div>
```

Turn in your assignment using the same Dropbox link as the other Mastering CSS sections.

Looking forward to seeing some excellent Sun rises.