animation
in html, css, and javascript
Dedicated to...

Bubble Tea ← My dentist thanks you!

Casterly Rock

Kyle Murray (Krilnon)
Trevor McCauley (Senocular)
Jesse Marangoni (TheCanadian)
Andreas Renberg (IQAndreas)

We should totally start a band!

My Parents

Merlin

Leslie, Ben, April, Jerry, Donna, Chris, Ann, Ron, Tom, and Andy

Don Draper

Heisenberg

Flash ← This book was meant for you all along :(

Page | 2
Table of Contents

Chapter 0: Introduction ................................................................................................................................. 9
  Know Your Basic HTML, CSS, and JavaScript .......................................................................................... 12
  About Authoring Tools ............................................................................................................................... 12
  Browser Support ......................................................................................................................................... 13
  Getting Help .................................................................................................................................................. 13

Chapter 1: What is an Animation? .................................................................................................................. 14
  The Start and End States ............................................................................................................................. 14
  Interpolation ................................................................................................................................................ 17
  Animations in HTML .................................................................................................................................. 18
  TL;DR / What You’ve Learned ................................................................................................................... 20

Part I: CSS Animations and Transitions

Chapter 2: All About CSS Animations .......................................................................................................... 22
  Creating a Simple Animation .................................................................................................................... 24
  What Just Happened .................................................................................................................................. 27
  Detailed Look at the CSS Animation Property .......................................................................................... 33
  Reusing Keyframes ................................................................................................................................. 42
  Declaring Multiple Animations ................................................................................................................. 45
  Wrap-up ....................................................................................................................................................... 46
Chapter 3: All About CSS Transitions

Adding a Transition ................................................................. 49
Looking at Transitions in Detail .................................................. 51
The Longhand Properties ............................................................. 57
Longhand Properties vs. Shorthand Properties ......................... 57
Working with Multiple Transitions...and So On ......................... 59
The transitionEnd Event ............................................................ 60
Summary ..................................................................................... 60

Chapter 4: Animations vs. Transitions ........................................... 61
Similarities .................................................................................. 62
Differences .................................................................................. 62
When to Use Which ...................................................................... 70

Chapter 5: Easing Functions in CSS .............................................. 71
Making Sense of Easing Functions .............................................. 73
What You Can and Can’t Do ....................................................... 82
My Name is Curve...Cubic Bezier Curve! .................................. 86
Easing Functions in CSS ............................................................. 89
Meet the Easing Functions / Timing Functions ......................... 91
TL;DR / Wrap-up ......................................................................... 98

Chapter 6: Animating Movement Smoothly ................................ 100
Say Hello to the translate3d Transform .................................... 100
What's Wrong With Setting Margin, Top, Left, etc.? ................. 103

Chapter 7: Say Hello to the Events .............................................. 106
Part II: CSS Examples

Example I: Sliding Background Effect on Link Hover .............................................. 117

Overview of How This Works ...................................................................................... 119
How this Effect Really Works ....................................................................................... 124
Conclusion .................................................................................................................... 136

Example II: Creating a Sweet Content Slider ............................................................. 137

Overview of How It Works ............................................................................................. 140
The Code ........................................................................................................................ 142
Conclusion .................................................................................................................... 154

Example III: The <blink> Tag Shall Live On ................................................................. 155

Re-creating the Blink Effect .......................................................................................... 157
Overview of How It Works ............................................................................................. 159
What Did You Learn ...................................................................................................... 160

Example IV: Simple Text Fade and Scale Animation .................................................. 161

The Example ............................................................................................................... 163
What Did You Learn ...................................................................................................... 170

Example V: Move Element to Click Position ............................................................... 172

The Example ............................................................................................................... 172
Part III: Animations and Code

Chapter 8: Animations Created in Code ................................................................. 193
  Why Animate Using JavaScript ........................................................................ 193
  Breaking Down a JavaScript Animation .......................................................... 198
  Looking at a Real Example .............................................................................. 202
  Going a Little More Crazy ................................................................................ 208

Chapter 9: Meet requestAnimationFrame .......................................................... 212
  Meet requestAnimationFrame .......................................................................... 213
  Using It ............................................................................................................ 214
  Another Example ............................................................................................. 217
  Your Frame Rate .............................................................................................. 220
  Stopping your requestAnimationFrame Loop ................................................. 221
  Conclusion ...................................................................................................... 224

Chapter 10: Vendor Prefixes in JavaScript ........................................................ 225
  Meet the Vendor Prefixes in JavaScript .......................................................... 227
Part IV: Examples Involving Code

Example I: Animated Scroll to Top with Easing ................................................................. 346
  Overview of How it All Works .......................................................................................... 348
  The Insides ...................................................................................................................... 353
  Deconstructing the Markup and Code ......................................................................... 357
  Conclusion ...................................................................................................................... 368

Example II: Smooth Parallax Scrolling ............................................................................ 369
  Overview of How this Effect Works ............................................................................. 372
  The Example .................................................................................................................. 377
  Deconstructing the Example ......................................................................................... 384
  A Word About Performance .......................................................................................... 402

Example III: Creating a Strobe Light Generator ............................................................ 404
  The Code ......................................................................................................................... 406
  What You Learned ....................................................................................................... 411

Example IV: Amorphous Circles ..................................................................................... 412
  Overview of How It Works .......................................................................................... 413
  Looking at the Code ...................................................................................................... 417
  Conclusion ...................................................................................................................... 425
Chapter 0

Introduction

Animations are everywhere. Today, anything you do on something with a screen is just one click, tap, or keystroke away from springing to life.
Animations are no longer a luxury or something to keep under wraps. Instead, animations make up a large part of your experience when using applications in your browser, on your phone, or any other “smart” device. It’s no surprise why. Users love animations. When done right, animations can make your applications easier to navigate, they can help your content be more presentable, and they can even help your creations feel more alive and fun. Sometimes, animations look cool and you just gotta throw them in. These are all good things!™

What’s the catch? Well, for all the great things animations have going for them, creating them is hard. In HTML, the way you think about animations is different. The way you create them is different...really different. The challenges that you once painfully overcame in older technologies like Flash are back in new and interesting ways. Fortunately, this is all part of learning something new. All you need is the right guide to help you along the way.

That guide is going to be me:

This is me.
This is me drinking bubble tea.
Bubble tea is my friend.
(This isn't a haiku)
Hi. My name is Kirupa, and I love animation. Ever since my parents bought me a computer in the early 1990’s, I’ve been trying to get things to move on the screen and tell everybody how awesome it is. Nothing much has changed in my world since then.

That’s why you have this book:

This book is all about teaching you how to create great animations for the web. This book starts off at the very beginning and gradually ratchets up the heat as you learn how to create more involved, more lifelike, and more complex animations. By the time you reach the last page, you will have learned a lot about how to create animations in HTML using the one-two punch of
**web standards and best practices.** If I did my job well, hopefully you had a bunch of fun along the way and laughed out loud a few times. More than likely, you will also have uncomfortably cringed a lot...kind of like what you did when I said “one-two punch of web standards and best practices.”

**Know Your Basic HTML, CSS, and JavaScript**

To make the most out of this book, you should be familiar with the basic things in HTML, CSS, and JavaScript. By no means am I expecting you to be an expert in any of these languages, but when you see some HTML or CSS or JavaScript, you should be able to understand what is going on with only a little handholding. This is the equivalent of knowing enough of a local language to get around town by reading the street signs.

If this doesn’t describe you, shuffle (or push / click / flick / scroll) through the pages in this book and see how you feel after glancing through some of the pages. If most of what you see seems like unintelligible gibberish, then I encourage you to take a detour and learn the basics of working in HTML first. There are a lot of great books and online tutorials out there (some even on kirupa.com), so you should be up and running very quickly. After that, you will probably have a much better time with this book.

**About Authoring Tools**

This entire book is written with no expectation that you use a fancy HTML authoring tool. A basic editor that allows you to see HTML, CSS, and JavaScript is all you need. If all you have is a text editor like Notepad, you’ll be fine as well. This book is old school. It’s all about manual labor and having you get your hands dirty when creating animations.

You may be wondering why a book aimed at teaching animation would avoid using animation tools. There are several reasons. The main reason is that tools abstract away important details. They often make choices for you that you may not fully understand. When learning something new and complex as animations in HTML, I strongly believe that you need to know every single detail about what you are doing. That can only be done if you are fully involved in every part of what you are creating.
With all that said, **once you’ve learned** all about animations, their subtleties, and gotchas, take some time and look at the great tools that are out there. Well-designed tools help you to be significantly more productive than you ever will be defining animations by hand, and being productive is yet another good thing!™

**Browser Support**

Everything in this book requires you to be using what is known as a “modern” browser. If you are running a semi-recent version of Chrome, Internet Explorer (version 10 and above), Firefox, or Safari, you should be good to go. Also, before you ask, everything you will see will work beautifully on iPhones and iPads!

**Getting Help**

The last (and most important) thing to know before we get started is what to do when you get stuck or just need some assistance. The fastest and most effective way for getting help is to post in the HTML5 section of the kirupa.com Forums: [www.kirupa.com/forum](http://www.kirupa.com/forum). I or any of the many other friendly posters will help you out.

For any non-technical assistance, you can contact me at kirupa@kirupa.com, tweet to @kirupa, or message me on Facebook ([facebook.com/kirupa](http://facebook.com/kirupa)). I love hearing from readers like you, and I make it a point to personally respond to every message I receive.

And with that, it’s time to get started.
I've mentioned the word animation quite a number of times so far (39 times to be precise), and it is one of those words where everybody you meet has their own version of what it means. To throw one more version into the mix, at its most basic level, an animation is nothing more than a visualization of change.

Let's look at that in more detail.

The Start and End States

If visualizing change is an important part of an animation, we need to create some reference points so that we can compare what has changed. Let's call these reference points the start state and the end state. To help visualize this, I'm going to introduce you to my lovely assistant - the very talented blue circle:
Let's say our start state looks as follows:

Our blue circle is just sitting around and probably up to no good. You look away and come back a few moments later, the blue circle is now in its end state and looking a bit different:
Based just on the information you have on what the blue circle looks like in the start and end states, what can you tell is different?

One change is the position. Our blue circle starts off on the left side of the page. In the end, it is located on the right hand side. Another change is the size. The blue circle became several times larger in the end state compared to its initial size at the beginning. There may be a few other minor things that are different, but we captured the big ones.

Now that we have identified the changes between our end state and start state, we are one step closer to having an animation. If we pretended for a moment that we actually have an animation, what would you see if you just played the start and end states repeatedly? What you would see is something that just bounces from left to right repeatedly. It would not be pretty. In fact, it would be turrible. Just turrible.¹ To create an animation using what we have, we also need a way to

¹ Frank Caliendo’s impression of Charles Barkley: [http://youtu.be/jKtB5Nfyh1o?t=38s](http://youtu.be/jKtB5Nfyh1o?t=38s)
smooth things out between the start and end states. What we need is a healthy dose of something known as **interpolation**.

**Interpolation**

Interpolation is the fancy name given to the intermediate states are generated between your start and end states. This interpolation, which occurs over a **period of time that you specify**, would look something like the following diagram:

You may be wondering who specifies the interpolated states. The answer, which is probably good news, is that it depends. For certain kinds of animations, your browser or HTML rendering engine will take care of the messy details. All you need to specify is the **starting state**, the **ending state**, and the **duration** over which the transition between the two states needs to occur. For certain other kinds of animations that you create in JavaScript, you will specify all of the messy details yourself.
What I just presented is a very simplified view of reality. There are a few other ingredients like
timing functions (aka easing functions), keyframes, and other things that are missing from our
explanation so far, but that’s ok. For now, just revel in this simplified generalization of what makes
up an animation. After you are done reveling, put on your best party clothes, and get ready to
meet the three flavors of animation that you will end up using and learning more about in the
subsequent chapters.

Animations in HTML
In HTML, there isn’t just a single animation approach that you can use. That would be too easy.
You actually have three flavors of animations to choose from, and each one is specialized for
certain kinds of tasks. Let’s take a quick look at all three of them and see how they relate to the
animation generalization you saw in the previous section.

CSS Animations (aka Keyframe Animations)
CSS Animations are your traditional animations that are on some sort of performance enhancing
substance that makes them more awesome. With these kinds of animations, you can define not
only the beginning and the end state but also any intermediate states commonly (and
affectionately!) known as **keyframes**: 

![Keyframe Diagram]
These intermediate states, if you choose to use them, allow you to have greater control over the things you are animating. In the above example, the blue circle isn’t simply sliding to the right and getting larger. The individual keyframes adjust the circle’s size and vertical position in ways that you wouldn’t see if you simply interpolated between the start and end states.

Remember, even though you are specifying the intermediate states, your browser will still interpolate what it can between each state. Think of a keyframe animation as many little animations daisy chained together.

**CSS Transitions**

Transitions make up a class of animations where you only define the start state, end state, and duration. The rest such as interpolating between the two states is taken care of automatically for you:

While transitions seem like a watered down, simplified keyframe animation, don’t let that trick you. They are extremely powerful and probably my favorite animation technique to use in projects.

**JavaScript Animations**

If you want full control over what your animation does right down to how it interpolates between two states, you can use JavaScript:
There are a lot of cool things you can do when you opt-out of the interpolation the browser does for you, and you’ll get a good dose of that in the later chapters that look at JavaScript Animations in greater detail.

**TL;DR / What You’ve Learned**

An animation is nothing more than a visualization of something changing over a period of time. In HTML, you have not one, not two, but THREE different ways of bringing animations to life: CSS Animations, CSS Transitions, and Animations Created in JavaScript.

This book will cover all three of these ways in great detail starting with the next chapter.
Part I

CSS Animations and Transitions
As mentioned in the previous chapter, one of the three ways you have for animating content in HTML is by using something known as **CSS animations**. What CSS animations do is pretty simple. They allow you to animate CSS properties on the elements they affect. This allows you to create all sorts of cool things like making things move, having things fade in and out, seeing things change color, etc.

First, let's look at an example. Given that this is a book, this requires a certain amount of imagination on your part. The animation you need to imagine is one where two clouds are bouncing up and down. In the following page, I have placed four still images of what such an animation would look like:
If you still have difficulty visualizing an animation printed on paper but have access to a device with an internet connection, you can see it live at the following location:


In this chapter, you will learn all about CSS animations to not only create something similar to the moving clouds but also (hopefully) something much cooler / useful / etc. You will also learn how to define a CSS animation using the animation property, how to work with keyframes, and how to tweak the various animation-related properties to have your animation do exactly what you want. Not a bad way to start off, ehh?

Onwards!
Creating a Simple Animation

The easiest (and most fun!) way to learn about CSS animations is to simply get your hands messy with using them and then learning about why they work the way they do afterwards. Go ahead and create a new HTML document and add the following HTML and CSS into it:

```html
<!DOCTYPE html>
<html lang="en-us">
<head>
  <meta charset="utf-8">
  <title>Bouncing Clouds</title>
  <script src="http://www.kirupa.com/js/prefixfree.min.js"></script>
  <style>
    #mainContent {
      background-color: #A2BFCE;
      border-radius: 4px;
      padding: 10px;
      width: 600px;
      height: 300px;
      overflow: hidden;
    }
    .cloud {
      position: absolute;
    }
    #bigcloud {
      margin-left: 100px;
      margin-top: 15px;
    }
  </style>
</head>
<body>
</body>
```
If you preview all of this, you will see a slightly less exciting version of the example I presented earlier. You will see a single, off-center cloud standing perfectly still:

The reason is that no animation has been defined yet, so let's go ahead and add an animation to fix the boredom. Adding a CSS animation is made up of two steps. The first step is to set the animation property, and the second step is to define the keyframes that specify exactly what gets animated.
From the markup you added, find the `#bigCloud` style rule and add the following highlighted line:

```
#bigcloud {
  animation: bobble 2s infinite;
  margin-left: 100px;
  margin-top: 15px;
}
```

The details of what this line says aren't important for now, for we'll have time to get acquainted with it later. In the meantime, let's add the keyframes. Go ahead and add the following `@keyframes` style rule below where you have your `#bigCloud` style block:

```
@keyframes bobble {
  0% {
    transform: translate3d(50px, 40px, 0px);
    animation-timing-function: ease-in;
  }

  50% {
    transform: translate3d(50px, 50px, 0px);
    animation-timing-function: ease-out;
  }

  100% {
    transform: translate3d(50px, 40px, 0px);
  }
}
```

Once you've added this style rule, go ahead and preview your page now. You should see your cloud bobbling around happily as if it has no care in the world. Awww!
What Just Happened

What you just did was add a CSS animation that causes your cloud to bobble around. Now, what a CSS animation does is pretty straightforward. It allows you to specify the start state, any intermediate states (aka keyframes), and the end state of the properties on the element (or elements) you are wishing to animate. Our cloud's movement is pretty simple, so learning how it animates is a great starting point!

The first thing we will look at is the animation property itself:

```css
animation: bobble 2s infinite;
```

The animation property is responsible for setting your animation up. In the shorthand variant that is shown here (and a variant you will commonly use), you will specify three things at a minimum:

1. The name of your animation
2. The duration
3. The number of times your animation will loop

Our animation declaration is no different. The name of our animation is called bobble, the duration of the animation is 2 seconds, and it is set to loop an infinite number of times.

What About Vendor Prefixes?

The animation property is still pretty new, so a lot of browsers require it to be vendor prefixed in order to have it work. Do not clutter up your markup with them. Instead, use something like the -prefix-free library that this example uses to keep your markup simple while still allowing older browsers to be able to view your animation.
You can learn more about vendor prefixing and the –prefix-free library from the following link: http://www.kirupa.com/html5/avoid_using_vendor_prefixes.htm

As you can see, the animation declaration doesn't really contain much in terms of details on what gets animated. It sets the high-level definition of what your animation will do, but the actual substance of a CSS animation actually resides in its @keyframes rule.

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

```
@keyframes bobble {
    0% {
        transform: translate3d(50px, 40px, 0px);
        animation-timing-function: ease-in;
    }

    50% {
        transform: translate3d(50px, 50px, 0px);
        animation-timing-function: ease-out;
    }

    100% {
        transform: translate3d(50px, 40px, 0px);
    }
}
```

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:
On the inside, it contains style rules (aka the actual keyframes) whose selectors are either **percentage values** or the keywords **from** and **to**:

```css
@keyframes bobble {
  0% {
    transform: translate3d(50px, 40px, 0px);
    animation-timing-function: ease-in;
  }
  50% {
    transform: translate3d(50px, 50px, 0px);
    animation-timing-function: ease-out;
  }
  100% {
    transform: translate3d(50px, 40px, 0px);
  }
}
```

These keyframe style rules are pretty much what you would expect. They just contain CSS properties such as `transform` and `animation-timing-function` whose value will get applied when the rule becomes active. We'll come back to this shortly, for there is an important detail about the keyframe style rules you need to be aware of.
Now, what I have just explained is the part that easily makes sense. Here is where things could get a little bit confusing. Despite the animation property being declared in another style rule and your keyframes being declared in their own @keyframes rule, they are very much tied to the hip and don’t really function without each other being present. It would be quite romantic if it wasn’t so complicated and messy, but it is our job to unravel the mess, figure out what is going on, and...ultimately be able to create a messier mess in more complicated ways.

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

**The Name**

The name you give your @keyframes rule acts an identifier the animation property uses to know where the keyframes are:

```
#bigcloud {
  animation: bobble 2s infinite;
  margin-left: 100px;
  margin-top: 15px;
}

@keyframes bobble {
  0% {
    transform: translate3d(50px, 40px, 0px);
    animation-timing-function: ease-in;
  }

  50% {
    transform: translate3d(50px, 50px, 0px);
    animation-timing-function: ease-out;
  }

  100% {
    transform: translate3d(50px, 40px, 0px);
  }
```
It isn’t a coincidence that our `animation` property refers to `bobble`, and the name of our `@keyframes` rule is also `bobble`. If there is ever an inconsistency in the names, your animation will not work since the keyframes that make up the substance of your animation won’t get applied.

**Duration and the Keyframes**

From the previous section you learned how our `animation` property is made aware of its keyframes. That solves one mystery. The bigger mystery that we are going to look at now is the one between the animation’s duration and when a particular keyframe style rule actually becomes active.

As you recall, when you define the keyframe style rules inside your `@keyframes` rule, your selector isn’t an actual time value. It is either a percentage value or the `from/to` keyword:

```css
#bigcloud {
  animation: bobble 2s infinite;
  margin-left: 100px;
  margin-top: 15px;
}

@keyframes bobble {

  0% {
    transform: translate3d(50px, 40px, 0px);
    animation-timing-function: ease-in;
  }

  50% {
    transform: translate3d(50px, 50px, 0px);
    animation-timing-function: ease-out;
  }

  100% {
    transform: translate3d(50px, 40px, 0px);
  }
```
In our example, the percentage values for our keyframe selectors are 0%, 50%, and 100%. What they represent is the percentage of the animation that has completed. When your animation is just starting, you have completed 0% of your animation. The 0% keyframe will become active. When your animation is half-way done, the 50% keyframe becomes active. At the end of your animation, the 100% keyframe becomes active.

The duration value you specify all the way over there on the animation property, besides setting the total time your animation runs, helps you to rationalize the percentage values with actual units of time. You know, the way things are normally done.

Below is how our percentage values map to units of time for our 2 second animation:

![Diagram showing keyframes and percentage values mapping to units of time](image)

This, to me is the confusing part. Once you understand how duration is mapped to the individual keyframes, you jumped a major hurdle in being able to visualize these animations in your mind.

Anyway, I think we've looked at how a simple CSS animation works in sufficient detail. You learned all about how to declare an animation using the animation property and how the @keyframes
rule with its keyframe style rules look. We also took some time to understand how everything comes together to work the way it does. We are by no means done yet, though. In the next section, we’ll go off the beaten path a bit and look at all of the extra beautiful scenery the animation property has to offer.

About the from/to Selector

Instead of specifying 0% as your starting selector, you can use the equivalent from keyword. For 100%, you have the to keyword you can use as the selector. I don’t quite know why anybody would want to use this, but it exists and it’s good for you to know about its existence in case you run into it in the wild.

In this tutorial, I will not explicitly pay homage to the from and to keywords. With this sentence, they are officially dead to me and their existence will no longer be acknowledged further. (Actually, due to intense lobbying from the from/to PAC supporters, you will see them appear in an example in Act II.)

Detailed Look at the CSS Animation Property

The animation property is a whole lot more feature-filled than what we’ve just seen. Now that you got your feet wet with creating an animation, let’s do something less exciting and learn about all that the animation property brings to the table. To help with learning more about it, let’s first expand our shorthand property and look at the properties in their expanded form. Our shorthand version, as you’ve seen many times already, looks as follows:

```
animation: bobble 2s infinite;
```

Its expanded, longhand version will look like this:

```
animation-name: bobble;
animation-duration: 2s;
```
The three properties our shorthand version expands to are `animation-name`, `animation-duration`, and `animation-iteration-count`. What these properties do should be ingrained in your mind by now, so let’s just move on to tasks that use the properties we haven’t seen yet - properties such as: `animation-play-state`, `animation-delay`, `animation-direction`, `animation-fill-mode`, and `animation-timing-function`.

**Pausing and Resuming an Animation**

By default, your animation will start once the style rule containing your `animation` property becomes active. For our simple case, this basically means the moment the page loads and the CSS is parsed.

First, let’s loosely visualize a 2 second long animation that is set to loop infinitely as follows:

Each yellow rectangle represents one iteration of your animation. If you put each iteration of the animation side by side, you get something that looks like what I’ve shown above.

Once an animation starts, it won’t stop until it reaches the last keyframe. If your animation is set to loop, it will just keep running by starting over from the first keyframe once the last keyframe is hit. This animation will never end. It will just restart from the beginning over and over and over again.
You have control over changing this default play behavior. If you want your animation to not play immediately once the style rule containing your animation definition becomes active or if you want to pause your animation in the middle, you can fiddle with the animation-play-state property. This property allows you to toggle whether your animation plays or not by reacting to the **running** value or the **paused** value.

By default, the animation-play-state property is set to **running**. You can set the value to **paused** to stop your animation dead in its tracks:

```
animation-play-state: paused;
```

When the animation is paused, it retains whatever the last computed value the animation had:

![Paused animation](image)

It is almost as if time just stood still. You can resume it by setting the animation-play-state property back to **running**. There is no sudden restart where the animation goes back to the 0% mark before resuming:

![Running animation](image)
Your animation smoothly picks up from where it left off just like what you would expect if you used the Play and Pause functionality on a media player.

**Delaying and Offsetting the Animation**

If you want your animation to play but not really do anything for a period of time, you will want to meet the `animation-delay` property. This property allows you to specify the number of seconds of time will elapse before your animation starts to run.

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

A delay isn’t a case where the 0% keyframe gets hit and then you wait for 5 seconds. The delay occurs before the 0% keyframe from your first iteration is even hit:

![5 second delay diagram]

Once your animation starts running, the delay value never comes into play again. Every subsequent iteration (if there are any left) of your animation just flows seamlessly from the end of one to the beginning of the other.

Now, here is something else you can do with this property. Instead of specifying a positive time value for `animation-delay`, you can actually specify a negative time value as well:

```css
animation-delay: -.25s;
```

When you specify a negative value, your animation starts immediately but offset by the duration that you specified. Here is what an `animation-delay` of -.25s would look like:
The negative sign acts as a signal to tell your browser to treat this value as an offset instead of a delay. Yes, that is a little strange, especially given that this property is still called animation-delay, but I am merely the messenger here. Something less strange - if you specify an offset that is greater than the duration of a single iteration of your animation, that isn't a problem at all. Your animation will just start at whatever point in whichever iteration the starting point falls at. Just make sure you have enough iterations to account for the starting point in a future iteration. If you don't have enough iterations and you specify a large offset, your animation simply won't run.

**Hold My Keyframe Properties, Please!**

If you don't tell your animation to loop, you will notice that once your animation ends, any properties the keyframes set are removed and your elements return to a pre-animation state. This is because the properties applied by your keyframes are transient. They exist while the keyframes are active, but at any time outside that window, those property values are not maintained. If you didn't intend for that behavior, your animation may seem to suddenly jump into position at the start or suddenly reset after it has run to completion. Let's examine these two cases with a little more specificity and then look at how to change the default behavior.

**Waiting to Start**

The first case occurs when you are dealing with an animation-delay set. For example, to reuse what you saw a few moments earlier, let's say you have a 5s delay specified:
For the five seconds your animation is waiting, your keyframes are not active yet. Any properties the first keyframe contains will not get applied while the delay is active.

**Animation is Done**

The second case is when your animation has run to completion. Let's say you specified your animation to loop three times:

At the end, any properties specified by the last keyframe on the 3rd iteration will disappear. Your animated elements will return to a state where there was no evidence of an animation ever having existed.

**Meet animation-fill-mode**

If you want your starting keyframe's properties to apply during a delay or your last keyframe's properties to be retained after your animation has ended, you can set the animation-fill-mode property. You can set its value to be:

1. **none**
   There is no faking the property values here. If you want the keyframe's property values to apply, your keyframe must be active.

2. **forwards**
   After your animation has run to completion, any property values the animation had at the end will be maintained.
3. **backwards**
   The animation will apply the property values from the starting keyframe even if that keyframe is not active yet.

4. **both**
   This is the ultimate solution. Your animation will apply the property values of the first keyframe at the beginning and maintain the property values of the last keyframe at the end.

The animations I tend to create loop forever and don’t have delays at the beginning. Many animations I have created also do not have a significant difference in property values between the starting keyframe, the ending keyframe, and the animated elements in their un-animated state. Because of that, I’ve never really had to stay up at night worrying, so don’t feel pressured by your peers to declare the `animation-fill-mode` property if you don’t want to.

### Reversing an Animation (or Alternating Directions)

Now, let’s look at a slightly trippy property. Your animations play sequentially from 0% to 100% by default. You have the ability to change this behavior by setting the `animation-direction` property to either `normal`, `reverse`, `alternate`, or `alternate-reverse`. Both `normal` and `reverse` should be straightforward to figure out what they do, so let’s look at the more interesting values: `alternate` and `alternate-reverse`.

When you set the `animation-direction` to `alternate-reverse`, your animation starts off normal. Starting with the second iteration, it plays in reverse and alternates between normal and reverse from there on out:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>2s</td>
<td>2s</td>
<td>2s</td>
<td>2s</td>
<td></td>
<td>2s</td>
</tr>
</tbody>
</table>

Setting your `animation-direction` to just `alternate` has a similar but slightly different behavior:
Your animation starts off in reverse and alternates between normal and reverse from that point on.

At Easing, Soldier!

The last animation-related property we have to cover is `animation-timing-function`. This property allows you to specify an easing function that changes how your animation interpolates the property values between the beginning and the end. I cover easing functions in much greater detail in Chapter 6, but if you need to know now, feel free to jump ahead. I'll just patiently be waiting here.

The Animation Shorthand

What we have primarily looked at in the past few sections are the longhand properties for declaring your animation:

```css
#somethingSomethingDarkSide {
  animation-name: deathstar;
  animation-duration: 25s;
  animation-iteration-count: 1;
  animation-play-state: paused;
  animation-delay: 0s;
  animation-direction: normal;
  animation-fill-mode: both;
  animation-timing-function: ease-out;
}
```

Some of you may prefer using shorthand properties where all of the properties and their values are specified inside the `animation` property itself. In fact, as you know, our very own bobble animation is represented in its shorthand variant:
animation: bobble 2s infinite;

All of the longhand properties you see above can be represented in their shorthand form - just good luck in remembering it all. If you don't doubt me, here is what the mapping looks like:

animation: <animation-name> <animation-duration> <animation-timing-function> <animation-delay> <animation-iteration-count> <animation-direction> <animation-fill-mode>;

Simply substitute the appropriate value for the property that is displayed inside the angled brackets. Note that the animation-play-state property is not something that can be represented in shorthand. You will have to explicitly spell out that property and its value.

Anyway, to put our longhand example into shorthand, here is how everything would look:

```css
#somethingSomethingDarkSide { animation: deathstar 25s ease-out 0s 1 normal both; animation-play-state: paused; }
```

Is the shorthand version more compact than the equivalent longhand version? Absolutely! Is it more understandable? That's a tough one to answer and entirely based on your (or your team's) preference.

I generally like to use the shorthand version for specifying the animation-name, animation-duration, and animation-timing-function because that is easy for me to remember. Once I go beyond three property values, I have to start searching through the documentation on what the additional values refer to.

Your mileage may vary with regards to your views on longhand vs. shorthand properties, so use whatever you feel most comfortable with. And...with that brilliant piece of insight, it's time to bid
adieu to this detailed look at the animation property and focus on other sights on our never-ending scenic trip through animation country.

**Reusing Keyframes**

The last thing I want to talk about is using the same keyframes for another animation declaration. I lamented earlier how the disconnected nature of the animation property declaration from the actual @keyframes rule makes working with animations a bit clunky. Even in clunky things, there are some nice things you could do if you try hard enough.

One such thing is being able to reuse the same keyframes for another declaration of the animation property. It may be hard to see what I mean by this, so let's just extend our current example to highlight what I am talking about.

In your current HTML document that contains only a single cloud that is bouncing, go ahead and add **ONLY** the following highlighted lines:

```html
<!DOCTYPE html>
<html lang="en-us">
<head>
  <meta charset="utf-8">
  <title>Bouncing Clouds</title>
  <script src="http://www.kirupa.com/js/prefixfree.min.js"></script>

  <style>
    #mainContent {
      background-color: #A2BFCE;
      border-radius: 4px;
      padding: 10px;
      width: 600px;
      height: 300px;
      overflow: hidden;
    }
  
```
.cloud {
    position: absolute;
}

#bigcloud {
    animation: bobble 2s infinite;
    margin-left: 100px;
    margin-top: 15px;
}

#smallcloud {
    animation: bobble 4s infinite;
    margin-top: 65px;
    margin-left: 200px;
}

@keyframes bobble {
    0% {
        transform: translate3d(50px, 40px, 0px);
        animation-timing-function: ease-in;
    }

    50% {
        transform: translate3d(50px, 50px, 0px);
        animation-timing-function: ease-out;
    }

    100% {
        transform: translate3d(50px, 40px, 0px);
    }
}
</style>
</head>
Once you have added both the highlighted `#smallCloud` style rule and the second `img` element, go ahead and preview your page. If everything was done correctly, you will now see two clouds happily bouncing away...just like what you saw from the working example at the beginning of this chapter.

Now that your example works, let's look at how we were able to do this. The trick lies in the `animation` declaration in your `#smallCloud` style rule:

```html
#smallcloud {
    animation: bobble 4s infinite;
    margin-top: 65px;
    margin-left: 200px;
}
```

Notice that we are referencing the exact same `@keyframes` rule whose name is `bobble`. The only difference between this `animation` declaration and the `animation` declaration in the `#bigCloud` style rule that we have been looking at is the duration. The duration of the animation that applies to your small cloud is 4 seconds - twice as long as the duration of the animation that applies to your large cloud:
What this means is that the properties you defined in your bobble keyframes apply just the same for both our clouds. The only difference is that in one animation these keyframes run through in 2 seconds, and in the other animation, these keyframes run through at 4 seconds:

This independence between the keyframes and the animation declaration allows you to get away with doing something like this. Any alterations you make in the declaration of your animation property will affect your keyframes on a superficial level - just like you saw here with the duration. Every animation property I explained a few sections ago can be set to alter how your keyframes behave without having to directly touch your keyframes.

You have to admit, that is pretty cool.

## Declaring Multiple Animations

The last thing (ok, for real this time) we will quickly look at is how to declare multiple animations in the same animation property. In your shorthand declaration, simply comma separate each of your animations as shown below:
Notice that each animation is pointing to a different @keyframes rule. If for whatever reason you decide to point to the same @keyframes rule from within the same animation property declaration, based on CSS order precedence, the last one you declared will win.

When declaring your animations in longhand, you would do something that looks as follows:

```css
#oppaGangnamStyle {
    animation-name: hey, sexy, lady;
    animation-duration: 2s, 1s, 5s;
    animation-iteration-count: infinite;
}
```

Again, this should be pretty straightforward as well. Everything is comma separated in CSS, so when in doubt about how to declare multiple values for a property that supports multiple values, just add a comma.

**Wrap-up**

The animation property in CSS is a pretty important property to befriend and know more about - especially if you want to make your content livelier. Now that you've learned the basics of how to work with animations, we’re going to jump to the next station and meet the even more awesome CSS transitions.