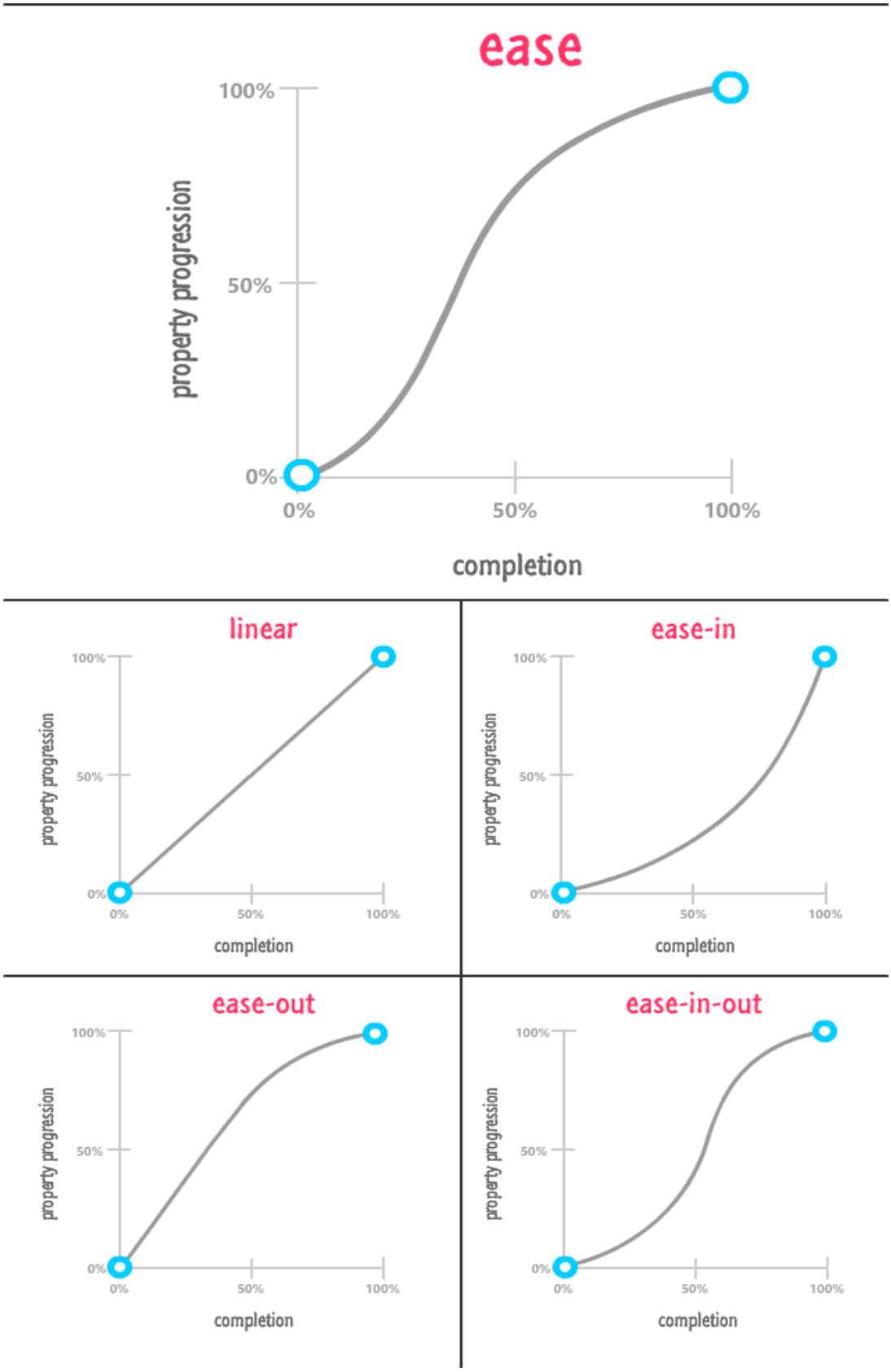
The following image shows you what the timing function curve for the predefined CSS timing functions look like:



You can sorta see from the timing functions how they end up affecting how our animation runs.

Now, in our visualization of timing functions, there are two timing functions that we did not include. They are the steps and cubic-bezier timing functions. These timing functions are ***special*** (which is usually code for complicated)! We will look at each in more detail in the following sections.

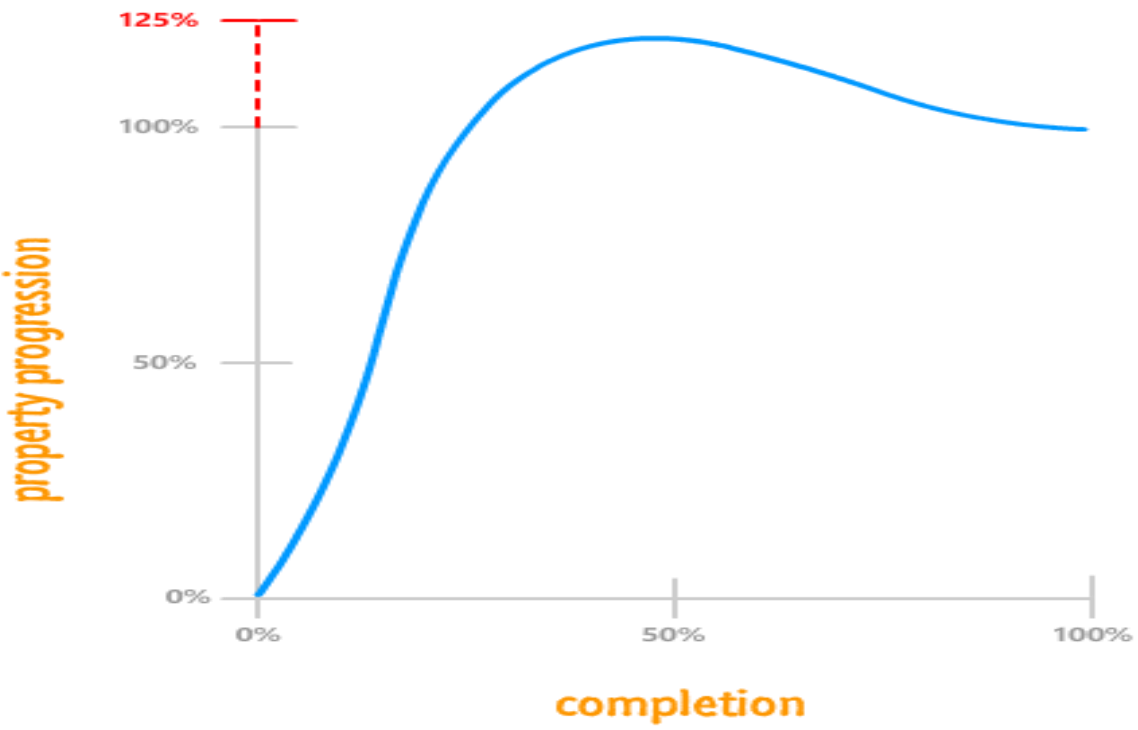
The cubic-bezier() Timing Function

The cubic-bezier timing function is the most awesome of the timing functions we have. The way it works is a little complicated. It takes four numbers as its argument:

```
.foo {
  transition: transform .5s cubic-bezier(.70, .35, .41, .78);
}
```

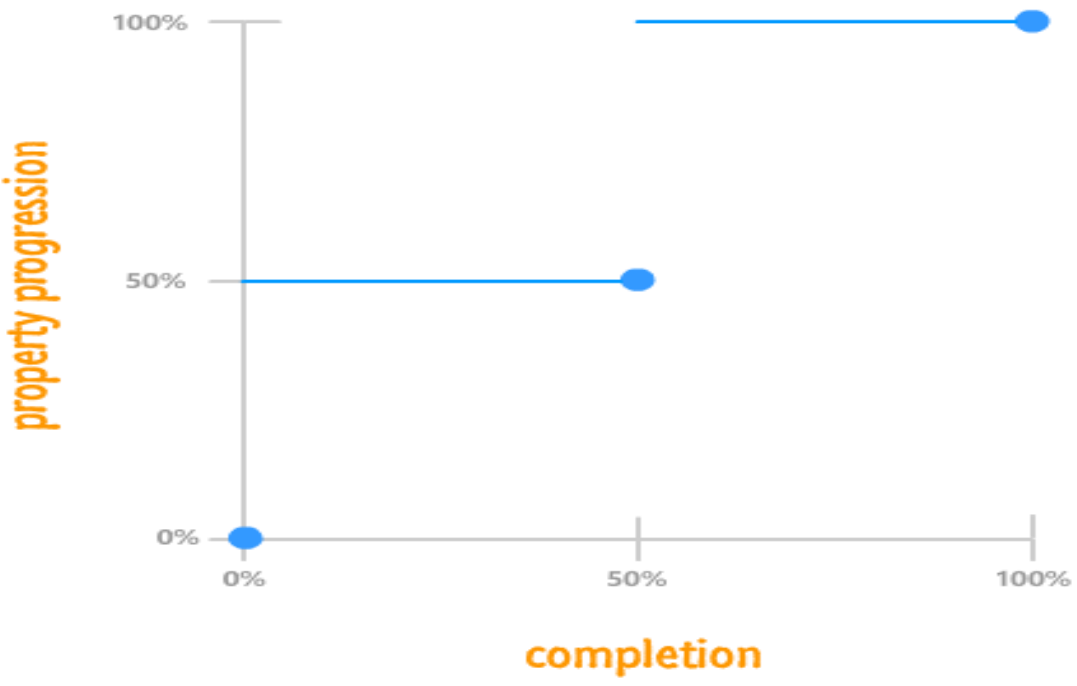
These four numbers define precisely how our timing function will affect the property that is getting animated. With the right numerical values, you can re-create all of our predefined timing functions like ease, ease-in-out, etc. Now, that's not particularly interesting. What is really REALLY interesting is the large variety of custom timing functions that you can create instead.

For example, with the cubic-bezier timing function you can create a timing function that looks like this:



The steps Timing Function

The last timing function we will look at is something that affects the rate at which our properties change but isn't technically a timing function. This non-timing function thing is known as a **step timing function**:



What a step function does is pretty unique. It allows you to play back your animation in fixed intervals. For example, in the step function graph you see above, your animated property progression starts at 0%. At the 50% mark, it jumps to 50%. At the end of the animation, your property progression reaches 100%. There is no smooth transition between the various frames or "steps". The end result is something a bit jagged.

In CSS, the step function can be defined by using the appropriately named **steps** function:

```
.pictureContainer img {
  position: relative;
  top: 0px;
  transition: top 1s steps(2, start);
}
```

The **steps** function takes two arguments:

i. Number of steps

ii. A value of **start** or **end** to specify whether the first step should occur at the beginning of the animation or whether the last step occurs when the animation ends

For example, if we want our animation to have five steps and have a step when the animation ends, our **steps** function declaration would look as follows:

```
.pictureContainer img {
  position: relative;
  top: 0px;
  transition: top 1s steps(5, end);
}
```

One thing to note is that, the more steps you specify, the smoother your animation will be. After all, think of an individual step as a frame of your animation. The more frames you have over the same duration, the smoother your final result will be. That same statement applies for steps as well.

Conclusion

The icing on your animation or transition flavored cake is the timing function. The type of timing function you specify determines how life-like your animation will be. By default, you have a handful of built-in timing functions you can specify as part of your CSS animations and transitions. Whatever you do, just don't forget to specify an timing function! The default **ease** timing function you get automatically isn't a great substitute for some of the better ones you can use, and your animation or transition will never forgive you for it.

Also, before I forget, here is the full markup for the three sliding circles that you saw at the beginning:

```
<style>
 .circle {
  width: 100px;
  height: 100px;
  border-radius: 50%;
  margin: 30px;
  animation: slide 5s infinite;
 }
 #circle1 {
  animation-timing-function: ease-in-out;
  background-color: #E84855;
 }
 #circle2 {
  animation-timing-function: linear;
  background-color: #0099FF;
 }
 #circle3 {
  animation-timing-function: cubic-bezier(0, 1, .76, 1.14);
  background-color: #FFCC00;
 }
 #container {
  width: 550px;
  background-color: #FFF;
  border: 3px #CCC dashed;
```

```
  border-radius: 10px;
  padding-top: 5px;
  padding-bottom: 5px;
  margin: 0 auto;
 }
 @keyframes slide {
  0% {
   transform: translate3d(0, 0, 0);
  }
  25% {
   transform: translate3d(380px, 0, 0);
  }
  50% {
   transform: translate3d(0, 0, 0);
  }
  100% {
   transform: translate3d(0, 0, 0);
  }
 }
</style>

<div id="container">
 <div class="circle" id="circle1"></div>
 <div class="circle" id="circle2"></div>
 <div class="circle" id="circle3"></div>
</div>
```

There is nothing crazy going on in this example, so I'll leave you all to it and sneak away!

Just a final word before we wrap up. If you have a question and/or want to be part of a **friendly**, **collaborative** community of over 220k other developers like yourself, post on the forums for a quick response!