Saturday, July 27, 2019

**Figuring out CSS animation properties with a magic kittencorn**

A little known fact about [SingaporeCSS](https://singaporecss.github.io/) is that we actually have an unofficial mascot. I haven’t named it yet, because naming stuff is the hardest problem in the world. But its birthday is 8 June, 2017 (so just a little over 2 years old now) and it first showed up at Talk.CSS during our [very first ever Codepen edition](https://singaporecss.github.io/19) on 26 July, 2017.

[See HJ Chen's other Tweets](https://twitter.com/hj_chen" \o "View HJ Chen's profile on Twitter)

Soon after that meetup, I thought it’d be nice to add our unofficial mascot to the SingaporeCSS website, so I drew a portrait of it in Sketch. And that’s it. I stopped there. Never did anything with the illustration. I also kept forgetting to bring it along for meetups so I think it showed up like 3 more times after over the next 2 years.

But times have changed. CSS kittencorn (until somebody thinks of a better name) is now standing guard over [Wei’s](https://uuei.io/) desk at work, and has showed up for multiple meetups since the 2 of them got acquainted this year. And, it’s now got a younger sibling. More on that in future.

Long story short, CSS kittencorn will probably no longer be the kind-of-secret mascot that it was for 2 years, and should feature prominently in all future Talk.CSSes moving forward. I hope.



To commemorate this momentous occasion, and also, entertain myself on yet another long-haul plane ride, I had another one of my hare-brained ideas. If you’ve followed my writing for a while, you’ll realise I have many of those. Also, thanks? I appreciate the 3 of you who read my nonsense. Truly.

**Can I has a GIF?**

I had originally wanted to make an animated GIF to send to Wei, just because. I had created animated GIFs in Photoshop before (just trigger my 404 page, go on, I’ll wait), but that was quite a bit of effort I wasn’t prepared to commit to.

So my next best option was create an animation that would run in the browser then make a screencast of it and save it as a GIF. The GIF production was ultimately a failure because it didn’t animate as planned, but WhatsApp supports video so that’s what ended up getting sent. Best laid plans and all.

First things first, is getting the kittencorn illustration into the browser. Was it possible to create kittencorn entirely out of HTML elements and CSS alone? Of course you could. But I’m LAZY. (*\*insert soon-to-be-supported Sloth emoji here*\*)

So SVG it is.

<svg class="kittencorn" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 415 352" fill="none">

<path class="rainbow" stroke="#000" stroke-width="5" d="M123.877 230.922c3.406 2.175 6.516 6.728 8.244 12.418 1.715 5.643 1.948 12.021.031 17.572-1.888 5.468-5.891 10.25-12.963 12.885-7.165 2.669-17.684 3.205-32.61-.401-29.756-7.188-54.915-26.626-69.709-46.127-7.404-9.76-12.078-19.362-13.534-27.273-1.455-7.906.345-13.544 5.04-16.585 2.651-1.719 4.958-2.45 7.019-2.553 2.056-.102 4.063.411 6.135 1.492 4.28 2.234 8.493 6.679 13.401 12.284.685.781 1.38 1.583 2.088 2.399 9.146 10.54 20.398 23.508 37.356 27.282 17.484 3.891 28.625 4.625 36.201 4.894 1.074.038 2.056.066 2.962.093 2.338.068 4.167.121 5.751.285 2.118.219 3.459.614 4.588 1.335z"/>

<path class="rainbow" fill-rule="evenodd" stroke="#000" stroke-linecap="square" stroke-width="5" d="M151.901 58.517S86.599 28.76 71.819 42.603c-14.78 13.845 9.153 90.422 9.153 90.422s11.039-23.8 29.824-42.843c18.785-19.043 41.105-31.665 41.105-31.665z" clip-rule="evenodd"/>

<path fill="#000" fill-rule="evenodd" d="M138 67.311S93.39 46.981 83.292 56.44c-10.097 9.458 6.253 61.771 6.253 61.771s7.54-16.259 20.374-29.268C122.752 75.933 138 67.31 138 67.31z" clip-rule="evenodd"/>

<path class="rainbow" fill-rule="evenodd" stroke="#000" stroke-linecap="square" stroke-width="5" d="M327 58.517s65.302-29.758 80.082-15.914c14.78 13.845-9.152 90.422-9.152 90.422s-11.039-23.8-29.824-42.843C349.32 71.139 327 58.517 327 58.517z" clip-rule="evenodd"/>

<path fill="#000" fill-rule="evenodd" d="M340.451 67.311s44.611-20.33 54.707-10.871c10.097 9.458-6.252 61.771-6.252 61.771s-7.541-16.259-20.374-29.268c-12.833-13.01-28.081-21.632-28.081-21.632z" clip-rule="evenodd"/>

<path class="rainbow" fill-rule="evenodd" stroke="#000" stroke-width="5" d="M239.5 319c95.269 0 172.5-30.225 172.5-112.067C412 125.091 364.154 43 239.5 43S67 125.091 67 206.933 144.231 319 239.5 319z" clip-rule="evenodd"/>

<path fill="#fff" fill-rule="evenodd" d="M269.421 85.475s-17.077-79.652-29.71-79.652C227.077 5.823 210 85.474 210 85.474s13.603 10.033 29.711 10.033c16.107 0 29.71-10.032 29.71-10.032z" clip-rule="evenodd"/>

<path stroke="#1CAFEF" stroke-linecap="round" stroke-width="5" d="M220.57 43.788s10.301 9.215 18.976 12.264c8.675 3.05 23.971 4.846 23.971 4.846M227.126 24.256s7.002 7.088 13.172 9.746c6.17 2.658 16.23 2.453 16.23 2.453"/>

<path stroke="#000" stroke-linecap="square" stroke-width="5" d="M269.421 85.475l1.484 2.012 1.299-.959-.338-1.578-2.445.525zm0 0l1.483 2.012-.002.002-.004.003-.012.008-.039.029-.136.097c-.116.083-.284.2-.5.347a49.67 49.67 0 0 1-1.856 1.198 64.981 64.981 0 0 1-6.698 3.588c-5.598 2.593-13.398 5.248-21.946 5.248-8.549 0-16.349-2.655-21.947-5.248a64.981 64.981 0 0 1-6.698-3.588 49.67 49.67 0 0 1-1.856-1.198 32.2 32.2 0 0 1-.5-.347l-.136-.097-.039-.029-.012-.008-.004-.003-.002-.002L210 85.475m59.421 0c2.445-.525 2.445-.525 2.444-.526v-.003l-.003-.011-.009-.045-.038-.174-.147-.672a648.068 648.068 0 0 0-2.682-11.6c-1.804-7.49-4.344-17.49-7.284-27.5-2.933-9.987-6.291-20.077-9.742-27.706-1.719-3.8-3.515-7.114-5.364-9.517-1.725-2.242-4.031-4.398-6.885-4.398-2.855 0-5.16 2.156-6.886 4.398-1.849 2.403-3.645 5.717-5.364 9.517-3.451 7.629-6.809 17.719-9.742 27.706-2.94 10.01-5.48 20.01-7.284 27.5a662.824 662.824 0 0 0-2.682 11.6l-.147.672-.037.174-.01.044-.003.012v.003l2.444.526m0 0l-2.444-.525-.339 1.578 1.299.959L210 85.475z"/>

<path class="rainbow" stroke="#000" stroke-width="5" d="M112.172 294.814c.162-.44.936-1.327 3.831-1.852 2.712-.491 6.499-.505 11.099-.061 9.159.885 20.929 3.525 32.386 7.041 11.461 3.517 22.442 7.861 30.052 12.086 3.835 2.128 6.632 4.129 8.23 5.859 1.691 1.831 1.355 2.513 1.273 2.66-3.557 6.449-6.703 10.517-10.829 13.38-4.148 2.877-9.531 4.708-17.896 6.183-16.152 2.848-28.509-1.923-48.52-10.911-6.312-2.835-9.911-7.947-11.371-14.096-1.477-6.22-.747-13.496 1.745-20.289zM366.828 294.814c-.162-.44-.936-1.327-3.831-1.852-2.712-.491-6.499-.505-11.099-.061-9.159.885-20.929 3.525-32.386 7.041-11.461 3.517-22.442 7.861-30.052 12.086-3.835 2.128-6.632 4.129-8.23 5.859-1.691 1.831-1.355 2.513-1.273 2.66 3.557 6.449 6.703 10.517 10.829 13.38 4.148 2.877 9.531 4.708 17.896 6.183 16.152 2.848 28.509-1.923 48.52-10.911 6.312-2.835 9.911-7.947 11.371-14.096 1.477-6.22.747-13.496-1.745-20.289z"/>

<path fill="#000" fill-rule="evenodd" d="M142.729 141.367l52.56 22.575s-6.123 54.384-47.317 41.688c-23.433-7.223-5.243-64.263-5.243-64.263z" clip-rule="evenodd"/>

<path stroke="#000" stroke-linecap="round" stroke-width="5" d="M139.5 140.45l56 24"/>

<path fill="#fff" fill-rule="evenodd" d="M180 181.95c2.761 0 5-4.03 5-9 0-4.971-2.239-9-5-9s-5 4.029-5 9c0 4.97 2.239 9 5 9z" clip-rule="evenodd"/>

<path fill="#000" fill-rule="evenodd" d="M336.271 141.367l-52.56 22.575s6.123 54.384 47.317 41.688c23.433-7.223 5.243-64.263 5.243-64.263z" clip-rule="evenodd"/>

<path stroke="#000" stroke-linecap="round" stroke-width="5" d="M339.5 140.45l-56 24"/>

<path fill="#fff" fill-rule="evenodd" d="M332 181.95c-2.761 0-5-4.03-5-9 0-4.971 2.239-9 5-9s5 4.029 5 9c0 4.97-2.239 9-5 9z" clip-rule="evenodd"/>

<path fill="#000" fill-rule="evenodd" d="M231.498 213.403c2.378-2.079 13.645-1.791 16.729 0 3.084 1.791-3.754 9.199-8.09 9.199-4.335 0-11.017-7.119-8.639-9.199z" clip-rule="evenodd"/>

<path stroke="#000" stroke-linecap="round" stroke-width="5" d="M247.756 262.45s-5.481-13.5-7.878-13.5c-2.398 0-7.878 13.5-7.878 13.5"/>

</svg>

It needs to be an inline SVG because I wanted to apply CSS animations to it, and target specific parts of the SVG with CSS classes. So you can see the rainbow class on some of the <path>s and the whole SVG has the kittencorn class.

In my mind, I wanted the kittencorn to spin while rotating through the colours of the rainbow, then the word “Magic” needed to appear, followed a random twinkle to end things off. Turns out all of those things can be achieved with CSS animations. Fun!

2 more additions to the markup then:

<p class="magic">Magic</p>

<svg class="sparkle" viewBox="0.0 0.0 50.0 50.0" fill="none" stroke="none" stroke-linecap="square" stroke-miterlimit="10">

<clipPath id="p.0">

<path d="m0 0l50.0 0l0 50.0l-50.0 0l0 -50.0z" clip-rule="nonzero"></path>

</clipPath>

<g clip-path="url(#p.0)">

<path fill-opacity="0.0" d="m0 0l50.0 0l0 50.0l-50.0 0z" fill-rule="nonzero"></path>

<path fill="#fff" d="m0.62204725 25.0l20.068499 -4.323374l4.309454 -20.13332l4.309454 20.13332l20.068499 4.323374l-20.068499 4.323374l-4.309454 20.133318l-4.309454 -20.133318z" fill-rule="nonzero"></path>

<path stroke-width="1.0" stroke-linejoin="round" stroke-linecap="butt" d="m0.62204725 25.0l20.068499 -4.323374l4.309454 -20.13332l4.309454 20.13332l20.068499 4.323374l-20.068499 4.323374l-4.309454 20.133318l-4.309454 -20.133318z" fill-rule="nonzero"></path>

</g>

</svg>

**Basics of CSS animation**

The specification that defines how CSS animations work is the [CSS Animations Level 1](https://www.w3.org/TR/css-animations-1/), currently in Working Draft status. This specification allows developers to:

*specify the changes in CSS properties over time as a set of keyframes*

They are fairly similar to [CSS transitions](https://www.w3.org/TR/css-transitions-1/) with the key difference being:

*while transitions trigger implicitly when property values change, animations are explicitly executed when the animation properties are applied*

Let’s also look at some fun facts about CSS animations which are outlined in the specification.

1. Animations will override all normal rules, but are overridden by !important rules
2. If there are multiple animations specifying behaviour for the same property, the animation that occurs last wins
3. An animation does not affect the computed value before the application of the animation or after it is removed
4. The animation computes the value of the properties during the running of the animation but other values may take precedence over the animated value
5. An animation starts when the style applying the animation and corresponding @keyframes rule are both resolved, but dynamically updating keyframe style rules does not start or restart an animation
6. Changes to the values of animation properties while the animation is running apply as if the animation had those values from when it began
7. The same @keyframes rule name may be repeated within an animation-name
8. Setting the display property to none will terminate any running animation applied to the element and its descendants

Keyframes are used to define the values for the animating properties at specific points during the animation. They are written as follows:

@keyframes animation-name { ... }

There’s some stuff you need to know about naming your @keyframesblock though. It can be either a custom identifier (no quotes) or a string (uses quotes). The name is fully case-sensitive, which means if every codepoint of the name matches, then they are considered the same.

/\* both have the same name, so the first block is ignored \*/

@keyframes magical { ... }

@keyframes "magical" { ... }

/\* because of case-sensitivity, this is considered different \*/

@keyframes MAGICAL { ... }

And reserved keywords like none or initial won’t work as custom identifiers BUT you can use them as strings. So do the quotes thing and your keyframe magically becomes valid. See what I did there? No? Never mind…

/\* these will not work \*/

@keyframes None { ... }

@keyframes initial { ... }

/\* but these will \*/

@keyframes "None" { ... }

@keyframes "initial" { ... }

**The syntax of CSS animations**

animation is a shorthand property, which covers the following (values are initial default values):

* animation-name: none
* animation-duration: 0s
* animation-timing-function: ease
* animation-delay: 0s
* animation-iteration-count: 1
* animation-direction: normal
* animation-fill-mode: none
* animation-play-state: running

If you want to have multiple animations on an element, separate each one with a comma. I’m only very briefly covering what these individual properties do, but for full explanations, please read the extremely digestible [CSS Animations Level 1](https://www.w3.org/TR/css-animations-1/).

**animation-name**

We talked about this a bit earlier, and it is used to select the @keyframerule which provides the property values for the animation. If this name does not match any keyframes, no animation for you. none is a keyword value, so if you use it as a custom identifier, no animation for you either.

**animation-duration**

This property defines the duration of a single animation cycle. Essentially how long it takes for the animation to run from start to finish.

Because CSS obeys the laws of physics, negative time values are invalid. Although if you define this as 0s, even though the keyframes have no effect, the animation still occurs but instantaneously.

**animation-timing-function**

This property describes how the animation function will progress between each pair of keyframes. There’s even a separate CSS specification for timing functions called [CSS Easing Functions Level 1](https://www.w3.org/TR/css-easing-1/).

Because animation is way more complicated than most of us care to think about. Also, during animation-delay, the animation-timing-function is not applied.

**animation-delay**

This property defines when the property will start, so you can make the animation start a bit later, or make it appear to have started before it was applied. This bit needs some explanation. So unlike the animation-duration property, a negative time value in this case is actually valid.

It’s just that the browser will progress the animation to a point where it would have been had the animation started some time in the past. So it seems like it started partway through its active duration.

**animation-iteration-count**

This property specifies the number of times the animation gets played. By default the animation will run once from start to finish but you can always make it run multiple times or keep looping with a value of infinite. Because who doesn’t love infinitely running animations? (loads of people, actually)

Usually people use this with an animation-direction of alternate so the animation can play in reverse on alternate cycles.

**animation-direction**

This property, as mentioned earlier, defines if an animation should play in reverse on some or all of the cycles. When you play an animation in reverse, the timing functions also end up getting reversed. So an ease-in ends up becoming an ease-out.

Your options for this are normal, reverse, alternate and alternate-reverse. Because choices.

**animation-fill-mode**

This property defines what values are applied by the animation outside the time it is actually running. By default, the animation will not affect property values after it has done running, but animation-fill-mode can override this behaviour.

For example, if you animated something from the left of the screen to the right, after the animation ends, your thing will morph back to its original position if you don’t do anything. So if you had wanted your thing to stay put at the end, you’d apply a value of forwards for this property.

**animation-play-state**

This property defines whether the animation is running or paused. Pretty straightforward (as compared to some of the other properties, imho). running means the animation proceeds as normal, while pausedmeans the animation is paused. ¯\\_(ツ)\_/¯

If the animation is set to paused during the animation delay phase, then the delay clock is also paused.

**Some layout things**

Wei created a gorgeous set of hand-doodled slides for an elevator pitch about a new meetup she’s starting up (more on that at the end), and they looked great. I am a big advocate of [sketching things out on pencil and paper](https://www.chenhuijing.com/blog/how-i-design-with-css-grid/)), but never really got into the digital side of things.

I guess I was too used to the feel of a pencil (or paintbrush) but sketching on a tablet isn’t the worse experience in the world. The only way to get better at something is to do it, right? Anyway, here was the plan in my head sketched out.



Before animating anything, we must lay out all the things. Remember this was supposed to be a screen capture to begin with? So there’s nothing else on the page other than what you saw in the above sketch. Which makes it so much easier to position it right smack on the centre of the page.

html {

box-sizing: border-box;

height: 100%;

}

\*,

\*::before,

\*::after {

box-sizing: inherit;

margin: 0;

padding: 0;

}

body {

display: flex;

height: 100%;

font-family: 'Lemon', cursive;

}

These are pretty much my standard reset styles nowadays. May not work for you, but works out great for me. If you don’t want to do the height: 100% thing on multiple elements, it’s perfectly fine to chuck a height: 100vh on the <body> element instead.

div {

display: grid;

grid-template-columns: max-content max-content;

gap: 1em;

align-items: center;

margin: auto;

position: relative;

cursor: pointer;

}

All 3 bits to be animated are wrapped in a single <div>, which will have an id of magic for the tiny bit of Javascript to be sprinkled on at the end. What do you do when you have a single item that needs to be centred in its parent? Auto-margins, that’s what.

And since we were already set up with a parent-child relationship for the 3 animatable bits, rather than fiddle around with inline-block and vertical alignment, or flex and the browser doing its own sizing thing, I went with grid to place them instead. Content-based sizing is cool, yo.

.kittencorn {

height: 50vmin;

}

.rainbow {

fill: #a3e048;

}

If you’re using SVGs that use viewBox over explicitly set width and height values, make sure you set a height on the SVG with CSS. Also, I went with setting the default fill colour of my kittencorn in CSS instead of on the SVG itself, but you can always put the fill in the SVG itself.

p {

font-size: calc(1.5em + 7vmin);

writing-mode: vertical-rl;

text-orientation: upright;

text-transform: uppercase;

overflow: hidden;

height: 0;

}

The overflow and height: 0 thing on the text is a set up for the letters to appear in the later animation. My original plan was to make each letter materialise 1 by 1, with the opacity property, but that would have required animating each letter individually.

It’s definitely do-able, but I had to wrap each letter of the word “Magic“ in its own span. In the end, I built both versions so you can see how either implementation works. Animating each letter individually also meant more timings to take care of.

The lazy version involved animating the height of the p element from 0 to 100% so it looked like the text was flowing in from somewhere.

.sparkle {

position: absolute;

opacity: 0;

top: 25%;

right: 35%;

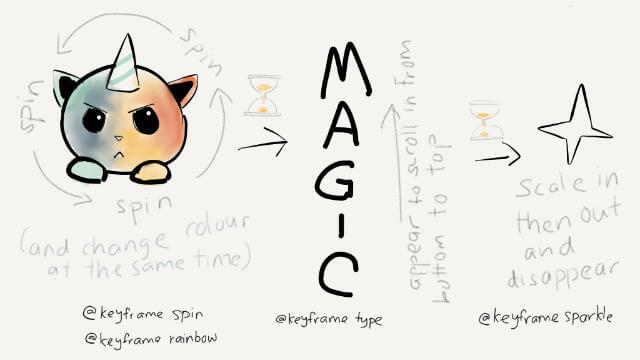
height: 2em;

}

Lastly, the sparkle. I wanted it on the kittencorn itself, so position: absolute was the way to go in this case. Just remember to set position: relative on the direct parent otherwise its going to be positioned relative to the whole page instead.

**Let’s make it magical**

It was going to be 3 animations happening sequentially. But the thing about chaining animations is that momentary tiny pause or a momentary overlap between animations that makes it feel quite different. I’m not an expert in animation so I don’t know the scientific explanation for this. Don’t quote me.



As with all my unprofessional endeavours, I just tweaked the timing until it felt right. Especially for the alternate implementation which involved animating each letter of the word “Magic”. Anyway, keyframe time!

The colour change involves animating the SVG’s fill property through a series of rainbow colours.

@keyframes rainbow {

0% { fill: #a3e048 }

14.3% { fill: #f7d038 }

28.6% { fill: #eb7532 }

42.9% { fill: #e6261f }

57.2% { fill: #49da9a }

71.5% { fill: #34bbe6 }

85.8% { fill: #4355db }

100% { fill: #d23be7 }

}

The spin is a rotation transform. I tried doing a single spin with 2 iteration counts, but it wasn’t smooth. A double spin of 720 degrees run once was much better.

@keyframes spin {

100% {

transform: rotate(720deg);

}

}

This bit is for the text. The keyframes are straight-forward, it was the timings which needed a bit more effort.

@keyframes type {

0% { height: 0 }

100% { height: 100% }

}

/\* This is the alternate version for individual letters \*/

@keyframes type {

0% { opacity: 0 }

100% { opacity: 1 }

}

Again, a double rotation for the sparkle, but also a bit of a scaling effect, for good measure. When using multiple transforms on a single element (or keyframe), remember to put them all in the same transform property.

@keyframes sparkle {

0% {

opacity: 0;

transform: rotate(0deg) scale(0);

}

50% {

opacity: 1;

transform: rotate(360deg) scale(1.3);

}

100% {

opacity: 0;

transform: rotate(720deg) scale(0);

}

}

Not sure if you really went through each of the animation properties, but some of them are required for my idea to work as I imagined.

.animate {

.kittencorn { animation: spin 2s }

.rainbow {

animation: rainbow 2s;

animation-fill-mode: forwards;

}

p {

animation: type 1s linear;

animation-delay: 1.9s;

animation-fill-mode: forwards;

}

.sparkle {

animation: sparkle 2s;

animation-delay: 3.3s;

animation-fill-mode: forwards;

}

}

Because I wanted the ending keyframe to persist, animation-fill-modewas set to forwards for all the animations. Not necessary for spinbecause the start and end position is exactly the same. The rest of it was manually tweaking the animation-delay until I got the feeling I was looking for.

Here’s the alternate version of the text animation:

span:first-child {

animation: type 1s linear;

animation-delay: 1.9s;

animation-fill-mode: forwards;

}

span:nth-child(2) {

animation: type 1s linear;

animation-delay: 2.5s;

animation-fill-mode: forwards;

}

span:nth-child(3) {

animation: type 1s linear;

animation-delay: 3.1s;

animation-fill-mode: forwards;

}

span:nth-child(4) {

animation: type 1s linear;

animation-delay: 3.7s;

animation-fill-mode: forwards;

}

span:nth-child(5) {

animation: type 1s linear;

animation-delay: 4.3s;

animation-fill-mode: forwards;

}

If you want to see the code for yourself or modify it to do other things, here’s the Codepens for both. The first one is the animate-height-of-text version and the second one is the fade-in-each-letter version.

There’s also a full page [no Codepen demo version](https://www.chenhuijing.com/demos/magical-kittencorn/), and source code on [GitHub](https://github.com/huijing/demos/tree/master/magical-kittencorn).

**So you want to restart your animation**

While I was building out the whole thing, because my setup was browser-sync-ified, every time I hit “save”, the animation triggered itself. But I soon realised it was probably a good idea to allow people to restart the animation somehow.

[Chris Coyier](https://www.chenhuijing.com/blog/figuring-out-css-animation-with-magic-kittencorn/) had me covered with his 8-year-old article, [Restart CSS animation](https://css-tricks.com/restart-css-animation). The code is not complicated, it’s about adding and removing a CSS class, but there is a magic line in there.

const magic = document.getElementById('magic')

// Technique from CSS Tricks article, Restart CSS Animation (https://css-tricks.com/restart-css-animation)

magic.addEventListener('click', function(e) {

e.preventDefault

magic.classList.remove('animate')

// This is the magic line

void magic.offsetWidth

magic.classList.add('animate')

}, false)

Anyway, people can now click on the kittencorn to restart the animation. Because you can’t just let it run once, right?

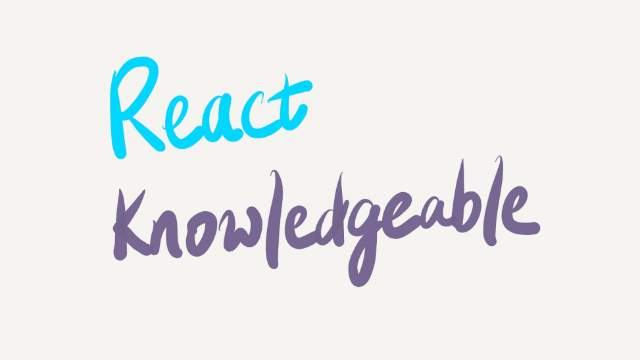
In case the CodePens don’t load or something, here’s the end result for both.

Spin clockwise…Spin anti-clockwise…

**Wrapping up**

Thank you for sitting through another edition of Hui Jing does useless things. This useless thing was quite fun to do though. And maybe you learned a little bit about how CSS animations work as well.

I want to take this opportunity to highlight a new meetup in town run by Wei called [React Knowledgeable](https://reactknowledgeable.org/) AKA <RK⚡️ />.



It’s going to be a lightning talk series of 4–5 lightning talks each meetup with a networking break in between. Read [the origin story](https://reactknowledgeable.org/story/stories/baby-react-knowledgeable-is-born/) to find out more. And [submit a talk](https://github.com/react-knowledgeable/talks/issues/new?assignees=&labels=talk&template=talk.md&title=%E2%9A%A1%EF%B8%8F+how+not+to+get+caught+and+be+eaten) if you feel like it.

CSS kittencorn’s kid sibling is kind of the unofficial mascot for <RK⚡️ />, I think? Go ask Wei about it. ¯\\_(ツ)\_/¯

So if you happen to be in Singapore during the first week of any month, drop by [Talk.CSS](https://www.meetup.com/SingaporeCSS/) or [React Knowledgeable](https://www.meetup.com/React-Knowledgeable/) and come see the kids. The meetups will be at least not bad. This is an official Hui Jing guarantee.