Best Practices

Animations can greatly improve an interface's UX, but it's important to follow some guidelines to not overdo it and deteriorate the user experience on your web-things. Following the following rules should provide a good start.

Meaningful animations

You should avoid animating an element just for the sake of it. Keep in mind that animations should make an intention clear. Animations like attention seekers (bounce, flash, pulse, etc) should be used to bring the user's attention to something special in your interface and not only as a way to bring "flashiness" to it.

Entrances and exit animations should be used to orientate what is happening in the interface, clearly signaling that it's transitioning into a new state.

It doesn't mean that you should avoid adding playfulness to the interface, just be sure that the animations are not getting in the way of your user and that the page's performance is not affected by an exaggerated use of animations.

Don't animate large elements

Avoid it as it won't bring much value to the user and will probably only cause confusion. Besides that, there is a good chance that the animations will be junky, culminating in bad UX.

Don't animate root elements

Animating the `<html/>` or `<body/>` tags is possible, but you should avoid it. There were some reports pointing out that this could trigger some weird browser bugs. Besides, making the whole page bounce would hardly provide good value to your UX. If you indeed need this sort of effect, wrap your page in an element and animate it, like this:

```
<body>
  <main class="animate__animated animate__fadeInLeft">
    <!-- Your code -->
  </main>
</body>
```

Infinite animations should be avoided

Even though Animate.css provides utility classes for repeating animations, including an infinite one, you should avoid endless animations. It will just distract your users and might annoy a good slice of them. So, use it wisely!

Mind the initial and final state of your elements

All the Animate.css animations include a CSS property called `animation-fill-mode`, which controls the states of an element before and after animation. You can read more about it here. Animate.css defaults to `animation-fill-mode: both`, but you can change it to suit your needs.

Don't disable the `prefers-reduced-motion` media query

Since version 3.7.0 Animate.css supports the `prefers-reduced-motion` media query which disables animations based on the OS system's preference on supporting browsers (most current browsers support it). This is a **critical accessibility feature** and should never be disabled! This is built into browsers to help people with vestibular and seizure disorders. You can read more about it here. If your web-thing needs the animations to function, warn users, but don't disable the feature. You can do it easily with CSS only. Here's a simple example:

See the Pen Prefers-reduce-motion media query by Elton Mesquita (@elton-mesquita) on CodePen.

Gotchas

You can't animate inline elements

Even though some browsers can animate inline elements, this goes against the CSS animation specs and will break on some browsers or eventually cease to work. Always animate block or inline-block level elements (grid and flex containers and children are block-level elements too). You can set an element to `display: inline-block` when animating an inline-level element.

Overflow

Most of the Animate.css animations will move elements across the screen and might create scrollbars on your web-thing. This is manageable using the `overflow: hidden` property. There's no recipe to when and where to use it, but the basic idea is to use it in the parent holding the animated element. It's up to you to figure out when and how to use it, this guide can help you understand it.

Intervals between repeats

Unfortunately, this isn't possible with pure CSS right now. You have to use Javascript to achieve this result.