Tips and tricks for using Animation in CSS that you need to know

Animations and transitions are an important part of web design. In this article, let's learn tips on using CSS Animation to design beautiful web!

Animations and transitions are an important part of web design. In this article, let's learn tips on using Animation in CSS to design beautiful web!

Toggle an element on mouse pointer

A common design practice is to have an element expand when interacted with. For example, you might want to shift the buttons up a bit when someone points the mouse over it. You can achieve this by using the transform property of CSS.

Assuming you have a button click:

??Click me

You style the document body and button like this:

```
/* C?n nút b?m ? gi?a trang */ body { ????display: flex; ????
height: 100vh; ????align-items: center; ????justify-content: center; ????
background-color: black; } /* T?o ki?u nút b?m */ button { ????
padding: 1em 2em; ????background: blue; ????border: 0; ????color: white;
????border-radius: 0.25rem; ????cursor: pointer; ????font-size: 2rem; ????
transition: transform 500ms; } /* Tr?ng thái tr? chu?
```

```
t */ button:hover, button:focus { ????
transform: translateY(0.75rem) 500ms; }
```

With the last block, you set the hover state and focus on the button click. In both states, you shift the knob along the Y axis by about 0.75rem. This button will look like this:

When the mouse pointer over the button, it moves in an upward direction. The transition takes about half a second to complete. This is a pattern that you can implement not only on buttons but also on other elements (e.g. images).

Declare multiple keyframes with one declaration

Another common pattern in CSS animations is repeating the same value over and over again. It can be color, size or direction. You can achieve this by using a CSS keyframe effect. Basically, you just need to declare multiple keyframes with one declaration.

Consider the button you created in the previous section. Maybe you want to repeat some background color when clicking the button, even the same color at different animation stages. Let's see how to do that in code.

First, you want to animate the button when it is clicked, so you created the **script.js** file . You access the button and enable a class on the button when clicked in this file. Specifically:

```
const button = document.querySelector("button") button.addEventListener("click",
????button.classList.toggle('party-time') })
```

The example used querySelector to access a button from a web page. **The party-time** class activates the **party** name animation :

```
.party-time { ????animation: party 2000ms infinite; }
```

For this animation, you start with red and change to yellow at 25%. Then you go back to red at 50% before going back to yellow at 75%. Finally, at 100%, you'll choose a dark blue color:

```
@keyframes party { ????0%, 50% { ????????background-color: red; ????} ????
25%, 75% { ????????background-color: yellow; ????} ????100% { ????????
```

```
background-color: hsl(200, 72%, 35%); ????} }
```

This approach is quite useful for alternating between background colors. Since you can repeat multiple keyframes in a variable, it's super easy to use the same attribute at different stages of animation.

Use @property to animate custom properties

As you probably already know, not all CSS properties can be animated. If you want to animate a property that cannot be 'animated', the best solution is to use the **@property** directive .

Start by changing the button's background color to a linear gradient:

```
button { ?????// other CSS ????
background: linear-gradient(90deg, blue, green); ????// other CSS }
```

Result:

Many people often want to animate the color gradient on the button. While there are ways to move the gradient around, it's practically impossible to 'animate' it. That's because background (and background-image) is not an animable property. This is where to use **@property** .

The @property directive allows you to register custom properties. When using @property, you must provide 3 values ??for it, including: syntax, inherits and initial-value:

```
@property --color-1 { ??syntax: ""; ??inherits: true; ??
initial-value: red; } @property --color-2 { ??syntax: ""; ??
inherits: true; ??initial-value: blue; }
```

The first is the start attribute, the second is the target attribute. Now instead of transitioning a background image (which you can't transition), you'll transition from **--color-1** to **--color-2** (custom property) in a second:

```
button { ??transition: --color-1 1000ms, --color-2 1000ms; }
```

This technique is useful because you can also add other customizations. For example, you add a delay to give it a smoother experience. Those possibilities are endless.

Use negative animation delay

Animation delay is important for creating smooth animations. Let's analyze the example below if you want to understand better. In this section, add a div element with 15 dots (dots) at the top of the button:

```
 ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ???? ????
```

Here are some basic styles to transform each child div into a dot:

```
.dots { ????display: flex; ????gap: .5rem; ????
margin-bottom: 20px; } .dot { ????width: 10px; ????aspect-ratio: 1; ????
background-color: red; ????border-radius: 50%; }
```

Here use Flexbox to place the dots in a horizontal line. Inside **script.js** , add the code that activates the effect for the dot. You are enabling dance class on dot:

```
button.addEventListener("click", (e) => { ????
button.classList.toggle('party-time') ????// Code m?i ????
dots.forEach((dot) => { ????????dot.classList.toggle('dance') ????}) })
```

The dance class activates the **rise** effect :

```
.dot.dance { ????animation: rise 2000ms infinite alternate; }
```

For this animation, simply translate the dots -100px along the Y axis:

```
@keyframes rise { ????100% { ????????transform: translateY(-100px) ????} }
```

Now is the time to do something interesting. Instead of increasing the dots simultaneously, you want to create a dot effect that moves like a wave? To do this, you add animation-delay to the dots, incrementing each dot by 100ms:

```
.dot:nth-child(1) { ????animation-delay: 100ms; } .dot:nth-child(2) { ????
animation-delay: 200ms; } .dot:nth-child(3) { ????
animation-delay: 300ms; } .dot:nth-child(4) { ????
animation-delay: 400ms; } /* Ti?p t?c cho t?i khi có 15 dot */
```

This creates a smooth animation where the dots move up and down in a waveform. The following image captures the dot image in the middle of the effect:

Use prefers-reduced-motion to enable the option

Remember, a lot of people don't like motion-based effects. In fact, most users have the option to disable motion in the browser. Movement can distract the senses. In severe cases, it can make viewers dizzy.

Fortunately, you can easily work around this by wrapping the animation inside a media **no-preference** query like this:

```
@media(prefers-reduced-motion: no-preference) { ??.dot.dance { ????
animation: rise 2000ms infinite alternate; ??} }
```

Now if prefers-reduced-motion is enabled in the browser, the animation will not run.

Above are **tips and tricks using CSS to create animation effects** . Hope the article is useful to you.

You finished reading the article "**Tips and tricks for using Animation in CSS that you need to know**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.
