

Transitions



- [Transition properties](#)
 - [transition-property](#)
 - [transition-duration](#)
 - [transition-timing-function](#)
 - [transition-delay](#)
 - [shorthand: transition](#)
 - [What can and can't transition?](#)
 - [Transform](#)
 - [Color](#)
 - [Shadows](#)
 - [Filters](#)
 - [Transition triggers](#)
 - [Different transitions for enter or exit](#)
 - [Accessibility considerations](#)
 - [Performance considerations](#)
-

When interacting with a website, you might notice that many elements have *state*. For example, dropdowns can be in opened or closed states. Buttons might change color when focused or hovered. Modals appear and disappear.

By default, CSS switches the style of these states instantly.

Using CSS transitions, we can interpolate between the initial state and the target state of the element. The transition between the two enhances the user experience by providing visual direction, support, and hints about the cause and effect of the interaction.

{% Aside 'key-term' %} *Interpolation* is the process of creating "in-between" steps that smoothly transition from one state to another. {% endAside %}

```
{% Codepen { user: 'web-dot-dev', id: 'zYzNrJV', height: 320 } %}
```

Transition properties

{% BrowserCompat 'css.properties.transition-property' %}

To use transitions in CSS, you can use the various transition properties or the `transition` shorthand property.

transition-property

The `transition-property` property specifies which style(s) to transition.

```
.my-element {  
  transition-property: background-color;  
}
```

The `transition-property` accepts one or more CSS property names in a comma-separated list.

Optionally, you may use `transition-property: all` to indicate that every property should transition.

{% Codepen { user: 'web-dot-dev', id: 'VwWPeEj', height: 400 } %}

transition-duration

The `transition-duration` property is used to define the length of time that a transition will take to complete.

{% Codepen { user: 'web-dot-dev', id: 'wvegMYp', height: 400 } %}

`transition-duration` accepts time units, either in seconds (`s`) or milliseconds (`ms`) and defaults to `0s`.

transition-timing-function

Use the `transition-timing-function` property to vary the speed of a CSS transition over the course of the `transition-duration`.

By default, CSS will transition your elements at a constant speed (`transition-timing-function: linear`). Linear transitions can end up looking somewhat artificial, though: in real life, objects have weight and can't stop and start instantly. Easing into or out of a transition can make your transitions more lively and natural.

```
{% Codepen { user: 'web-dot-dev', id: 'QWgdyZx' } %}
```

Our [module on CSS Animation](#) has a good overview of timing functions.

You can use [DevTools](#) to experiment with different timing functions in real-time.

```
{% Img src="image/eiKy1JcjHqPp3gaedjAQWjPJ8YK2/dRwKg0RI5y5wWVzkUFUA.png", alt="Chrome DevTools visual transition timing editor.", width="800", height="418" %}
```

transition-delay

Use the `transition-delay` property to specify the time at which a transition will start. If `transition-duration` is not specified, transitions will start instantly because the default value is `0s`. This property accepts a time unit, for example seconds (`s`) or milliseconds (`ms`).

```
{% Codepen { user: 'web-dot-dev', id: 'jOwyWep' } %}
```

This property is useful for staggering transitions, achieved by setting a longer `transition-delay` for each subsequent element in a group.

```
{% Codepen { user: 'web-dot-dev', id: 'yLXgeRQ', height: 410 } %}
```

`transition-delay` is also useful for debugging. Setting the delay to a negative value can start a transition further into the timeline.

shorthand: transition

Like most CSS properties, there is a shorthand version. `transition` combines `transition-property`, `transition-duration`, `transition-timing-function`, and `transition-delay`.

```
.longhand {
  transition-property: transform;
  transition-duration: 300ms;
  transition-timing-function: ease-in-out;
  transition-delay: 0s;
}

.shorthand {
  transition: transform 300ms ease-in-out 0s;
}
```

What can and can't transition?

When writing CSS, you can specify which properties should have animated transitions. See [this MDN list of animatable CSS properties](#).

In general, it's only possible to transition elements that can have a "middle state" between their start and final states. For example, it's impossible to add transitions for `font-family`, because it's unclear what the "middle state" between `serif` and `monospace` should look like. On the other hand, it is possible to add transitions for `font-size` because its unit is a length that can be interpolated between.

```
{% Img src="image/eiKy1JcjHqPp3gaedjAQWjPJ8YK2/1Lr1iC56nyUyYWWv6qF2.jpg", alt="Diagram of shapes transitioning smoothly from one state to another, and two lines of text in different fonts that cannot be transitioned smoothly.", width="800", height="600" %}
```

Here are some common properties you can transition.

Transform

```
{% BrowserCompat 'css.properties.transform' %}
```

The `transform` CSS property is commonly transitioned because it is a GPU-accelerated property that results in smoother animation that also consumes less battery. This property lets you arbitrarily scale, rotate, translate, or skew an element.

```
{% Codepen { user: 'web-dot-dev', id: 'GRErowE' } %}
```

Check out [the section on transforms](#) in [our Functions module](#).

Color

Before, during, and after interaction, color can be a great indicator of state. For example, a button might change color if it's being hovered. This color change can provide feedback to the user that the button is clickable.

The `color`, `background-color`, and `border-color` properties are just a few places where color can be transitioned upon interaction.

```
{% Aside %} Color transitions do not usually need to be behind a reduced motion preference. Use your best judgment. {% endAside %}
```

Check out [our module on color](#).

Shadows

Shadows are often transitioned to indicate elevation change, like from user focus.

```
{% Codepen { user: 'web-dot-dev', id: 'gORgPQx', height: 300 } %}
```

Check out [our module on shadows](#).

Filters

`filter` is a powerful CSS property that lets you add graphic effects on the fly. Transitioning between different `filter` states can create some pretty impressive results.

```
{% Codepen { user: 'web-dot-dev', id: 'PojWZxJ', height: 350 } %}
```

Check out [our module on filters](#).

Transition triggers

Your CSS must include a change of state *and* an event that triggers that state change for CSS transitions to activate. A typical example of such a trigger is the `:hover` [pseudo-class](#). This pseudo-class matches when the user hovers over an element with their cursor.

Below is a list of some pseudo-classes and events that can trigger state changes in your elements.

- `:hover`: matches if the cursor is over the element.
- `:focus`: matches if the element is focused.
- `:focus-within`: matches if the element or any of its descendants are focused.
- `:target`: matches when the current URL's [fragment](#) matches the element's id.
- `:active`: matches when the element is being activated (typically when the mouse is pressed over it).
- `class` change from JavaScript: when an element's CSS `class` changes via JavaScript, CSS will transition eligible properties that have changed.

Different transitions for enter or exit

By setting different `transition` properties on hover/focus, it's possible to create some interesting effects.

```
.my-element {  
  background: red;  
  
  /* This transition is applied on the "exit" transition */  
  transition: background 2000ms ease-in;  
}  
  
.my-element:hover {  
  background: blue;  
  
  /* This transition is applied on the "enter" transition */  
  transition: background 150ms ease;  
}
```

{% Codepen { user: 'web-dot-dev', id: 'OJgWMaO' } %}

Accessibility considerations

CSS transitions are not for everyone. For some people, transitions and animations can cause motion sickness or discomfort. Thankfully, CSS has a media feature called `prefers-reduced-motion` that detects if a user has indicated a preference for less motion from their device.

```
/*  
  If the user has expressed their preference for  
  reduced motion, then don't use transitions.  
*/  
@media (prefers-reduced-motion: reduce) {  
  .my-element {  
    transition: none;  
  }  
}
```

```
/*
  If the browser understands the media query and the user
  explicitly hasn't set a preference, then use transitions.
*/
@media (prefers-reduced-motion: no-preference) {
  .my-element {
    transition: transform 250ms ease;
  }
}
```

Check out our blog post [prefers-reduced-motion: Sometimes less movement is more](#) for more information on this media feature.

{% Aside %} In the Learn Accessibility module on [animation and motion](#) discover how to add delight to your site without causing problems for some users. {% endAside %}

Performance considerations

When working with CSS transitions, you may encounter performance issues if you add transitions for certain CSS properties. For example, when properties such as `width` or `height` change, they push content around on the rest of the page. This forces CSS to calculate new positions for every affected element for each frame of the transition. When possible, we recommend using properties like `transform` and `opacity` instead.

Check out our [guide on high-performance CSS animations](#) for a deep-dive on this topic.

{% Assessment 'transitions' %}

👤 #2
👤 2026-10-31 22:46:37 UTC
👤 2026-10-31 22:46:39 UTC