



Simplify software development with the One DevOps Platform. Start your free 30 day trial today!

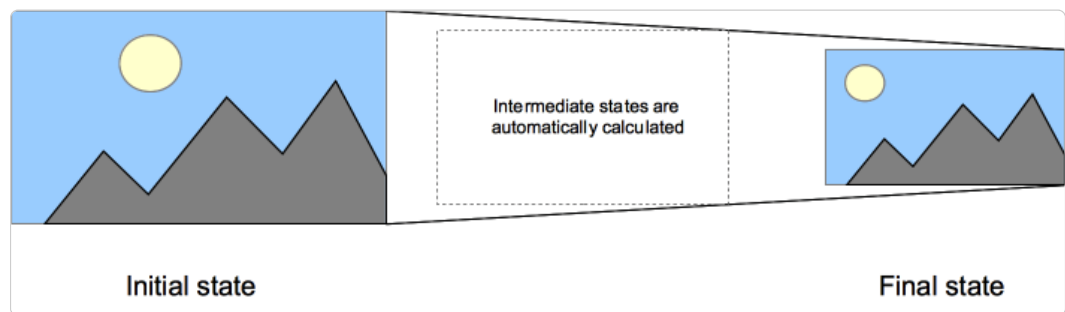
[Mozilla ads](#)

[Don't want to see ads?](#)

## Using CSS transitions

CSS transitions provide a way to control animation speed when changing CSS properties. Instead of having property changes take effect immediately, you can cause the changes in a property to take place over a period of time. For example, if you change the color of an element from white to black, usually the change is instantaneous. With CSS transitions enabled, changes occur at time intervals that follow an acceleration curve, all of which can be customized.

Animations that involve transitioning between two states are often called implicit transitions as the states in between the start and final states are implicitly defined by the browser.





CSS transitions let you decide which properties to animate (by listing them explicitly), when the animation will start (by setting a delay), how long the transition will last (by setting a duration), and how the transition will run (by defining a timing function, e.g., linearly or quick at the beginning, slow at the end).

## Which CSS properties can be transitioned?

The Web author can define which property has to be animated and in

which way. This allows the creation of complex transitions. As it doesn't make sense to animate some properties, the [list of animatable properties](#) is limited to a finite set.

 Note: The set of properties that can be animated is changing as the specification develops.

 Note: The `auto` value is often a very complex case. The specification recommends not animating from and to `auto`. Some user agents, like those based on Gecko, implement this requirement and others, like those based on WebKit, are less strict. Using animations with `auto` may lead to unpredictable results, depending on the browser and its version, and should be avoided.

## Defining transitions

CSS Transitions are controlled using the shorthand [transition](#) property. This is the best way to configure transitions, as it makes it easier to avoid out of sync parameters, which can be very frustrating to have to spend lots of time debugging in CSS.

You can control the individual components of the transition with the following sub-properties:


### [transition-property](#)

Specifies the name or names of the CSS properties to which transitions should be applied. Only properties listed here are animated during transitions; changes to all other properties occur instantaneously as usual.

### [transition-duration](#)

Specifies the duration over which transitions should occur. You can specify a single duration that applies to all properties during the transition, or multiple values to allow each property to transition over a different period of time.

### [transition-timing-function](#)

Specifies a function to define how intermediate values for properties are computed. Timing functions determine how intermediate values of the transition are calculated. Most [timing functions](#) can be specified by providing the graph of the corresponding function, as defined by four points defining a cubic bezier. You can also choose easing from [Easing Functions Cheat Sheet](#) .

### [transition-delay](#)

Defines how long to wait between the time a property is changed and the transition actually begins.

The `transition` shorthand CSS syntax is written as follows:

```
div {  
  transition: <property> <duration> <timing-function> <delay>;  
}
```



## Examples

### Simple example

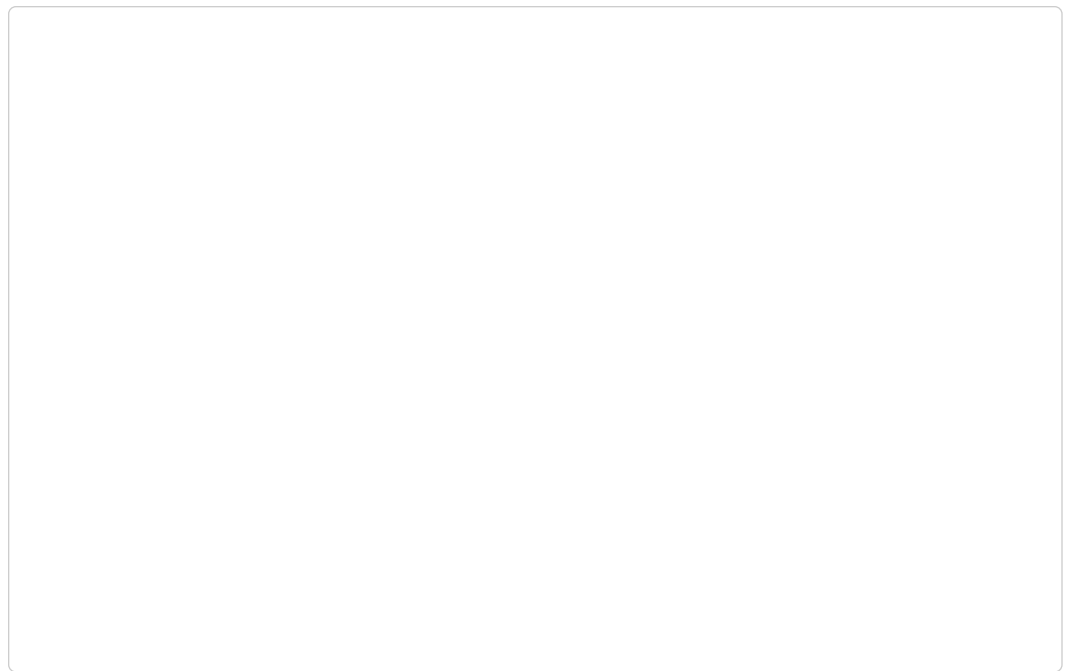
This example performs a four-second font size transition with a two-second delay between the time the user mouses over the element and the beginning of the animation effect:

```
#delay {  
  font-size: 14px;  
  transition-property: font-size;  
  transition-duration: 4s;  
  transition-delay: 2s;  
}  
  
#delay:hover {  
  font-size: 36px;  
}
```

## Multiple animated properties example

### CSS Content

```
.box {  
  border-style: solid;  
  border-width: 1px;  
  display: block;  
  width: 100px;  
  height: 100px;  
  background-color: #0000ff;  
  transition: width 2s, height 2s, background-color 2s, rotate 2s;  
}  
  
.box:hover {  
  background-color: #ffcccc;  
  width: 200px;  
  height: 200px;  
  rotate: 180deg;  
}
```



## When property value lists are of different lengths

If any property's list of values is shorter than the others, its values are repeated to make them match. For example:

```
div {  
  transition-property: opacity, left, top, height;  
  transition-duration: 3s, 5s;  
}
```



This is treated as if it were:

```
div {  
  transition-property: opacity, left, top, height;  
  transition-duration: 3s, 5s, 3s, 5s;  
}
```



Similarly, if any property's value list is longer than that for `transition-property`, it's truncated, so if you have the following CSS:

```
div {
```



```
transition-property: opacity, left;  
transition-duration: 3s, 5s, 2s, 1s;  
}
```

This gets interpreted as:

```
div {  
  transition-property: opacity, left;  
  transition-duration: 3s, 5s;  
}
```

## Using transitions when highlighting menus

A common use of CSS is to highlight items in a menu as the user hovers the mouse cursor over them. It's easy to use transitions to make the effect even more attractive.

First, we set up the menu using HTML:

```
<nav>  
  <a href="#">Home</a>  
  <a href="#">About</a>  
  <a href="#">Contact Us</a>  
  <a href="#">Links</a>  
</nav>
```

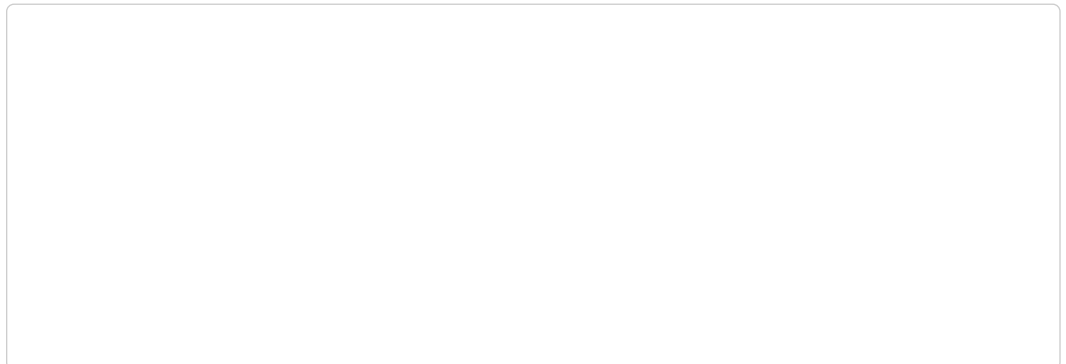
Then we build the CSS to implement the look and feel of our menu:

```
nav {  
  display: flex;  
  gap: 0.5rem;  
}  
  
a {  
  flex: 1;  
  background-color: #333;
```

```
color: #fff;
border: 1px solid;
padding: 0.5rem;
text-align: center;
text-decoration: none;
transition: all 0.5s ease-out;
}

a:hover,
a:focus {
  background-color: #fff;
  color: #333;
}
```

This CSS establishes the look of the menu, with the background and text colors both changing when the element is in its `:hover` and `:focus` states:



## JavaScript examples



Note: Care should be taken when using a transition immediately after:

- adding the element to the DOM using `.appendChild()`
- removing an element's `display: none;` property.

This is treated as if the initial state had never occurred and the element was always in its final state. The easy way to overcome this limitation is to apply a `setTimeout()` of a handful of milliseconds before changing the CSS property you intend to transition to.

## Using transitions to make JavaScript functionality smooth

Transitions are a great tool to make things look much smoother without having to do anything to your JavaScript functionality. Take the following example.

```
<p>Click anywhere to move the ball</p>  
<div id="foo" class="ball"></div>
```



Using JavaScript you can make the effect of moving the ball to a certain position happen:

```
const f = document.getElementById("foo");  
document.addEventListener(  
  "click",  
  (ev) => {  
    f.style.transform = `translateY(${ev.clientY - 25}px)`;  
    f.style.transform += `translateX(${ev.clientX - 25}px)`;  
  },  
  false  
);
```



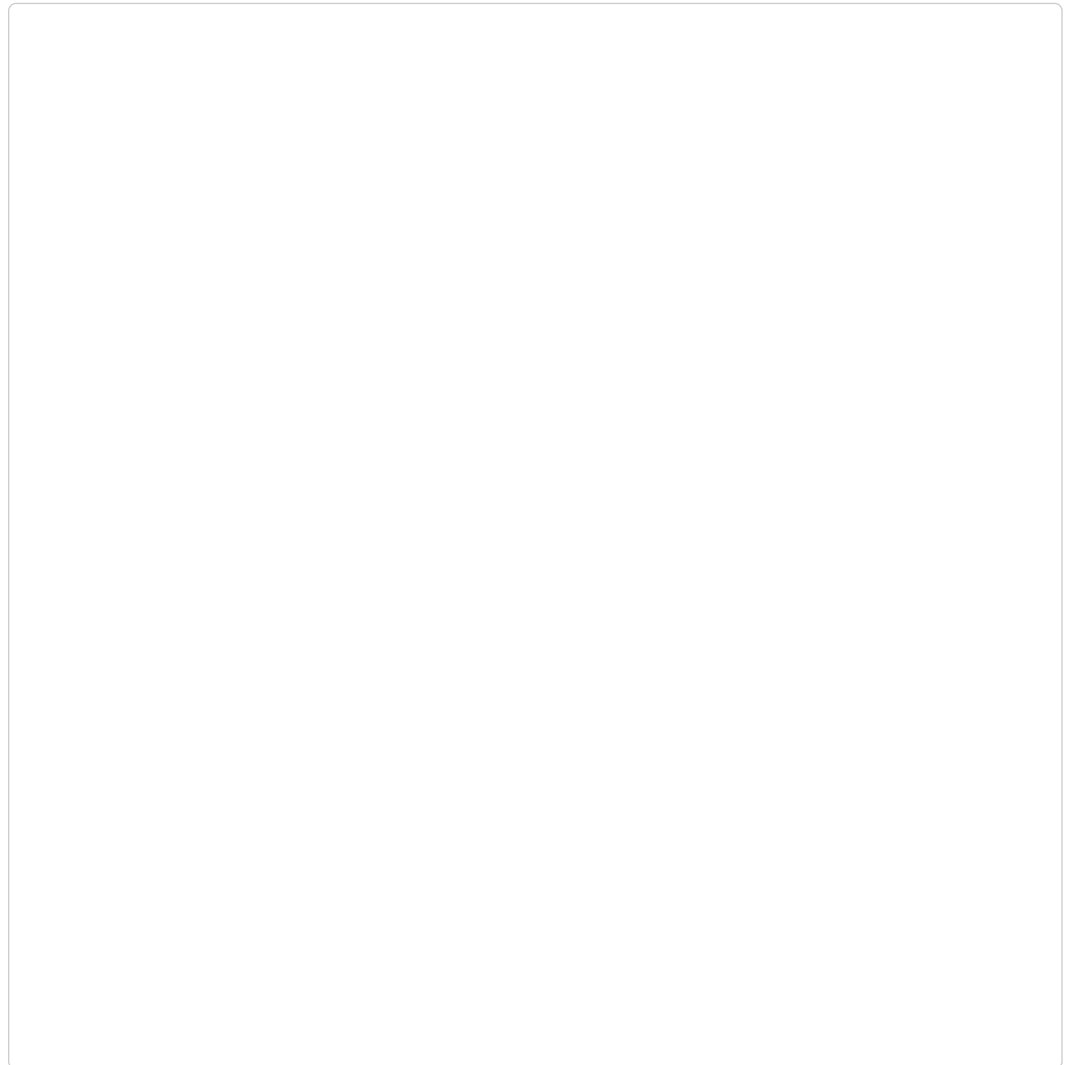
With CSS you can make it smooth without any extra effort. Add a transition to the element and any change will happen smoothly:

```
.ball {  
  border-radius: 25px;  
  width: 50px;
```





```
height: 50px;  
background: #c00;  
position: absolute;  
top: 0;  
left: 0;  
transition: transform 1s;  
}
```



## Detecting the start and completion of a transition

You can use the [transitionend](#) event to detect that an animation has finished running. This is a [TransitionEvent](#) object, which has two added properties beyond a typical [Event](#) object:

`propertyName`

A string indicating the name of the CSS property whose transition completed.

`elapsedTime`

A float indicating the number of seconds the transition had been running at the time the event fired. This value isn't affected by the value of [transition-delay](#).

As usual, you can use the `addEventListener()` method to monitor for this event:

```
el.addEventListener("transitionend", updateTransition, true);
```



You detect the beginning of a transition using [transitionrun](#) (fires before any delay) and [transitionstart](#) (fires after any delay), in the same kind of fashion:

```
el.addEventListener("transitionrun", signalStart, true);  
el.addEventListener("transitionstart", signalStart, true);
```



Note: The `transitionend` event doesn't fire if the transition is aborted before the transition is completed because either the element is made `display : none` or the animating property's value is changed.

## Specifications

Specification

[CSS Transitions](#)

## See also

- The [TransitionEvent](#) interface and the [transitionend](#) event
- [Using CSS animations](#)

This page was last modified on Mar 31, 2023 by [MDN contributors](#).