Lesson 8

Transitions & Animations

In this Lesson

8

CSS

SHARE

One evolution with CSS3 was the ability to write behaviors for transitions and animations. Front end developers have been asking for the ability to design these interactions within HTML and CSS, without the use of JavaScript or Flash, for years. Now their wish has come true.

With CSS3 transitions you have the potential to alter the appearance and behavior of an element whenever a state change occurs, such as when it is hovered over, focused on, active, or targeted.

Animations within CSS3 allow the appearance and behavior of an element to be altered in multiple keyframes. Transitions provide a change from one state to another, while animations can set multiple points of transition upon different keyframes.

Transitions

As mentioned, for a transition to take place, an element must have a change in state, and different styles must be identified for each state. The easiest way for determining styles for different states is by using the `:hover`, `:focus`, `:active`, and `:target` pseudo-classes.

There are four transition related properties in total, including `transition-property`, `transition-duration`, `transition-timing-function`, and `transition-delay`. Not all of these are required to build a transition, with the first three are the most popular.

In the example below the box will change its `background` color over the course of `1` second in a `linear` fashion.

```
.box {
  background: #2db34a;
  transition-property: background;
  transition-duration: 1s;
  transition-timing-function: linear;
```

```
  }
  .box:hover {
    background: #ff7b29;
  }
```

Transition Demo

HTML CSS RESULT EDIT ON CODEPEN

Box

Resources 1× 0.5× 0.25× Rerun

Vendor Prefixes

The code above, as with the rest of the code samples in this lesson, are not vendor prefixed. This is intentionally un-prefixed in the interest of keeping the code snippet small and comprehensible. For the best support across all browsers, use vendor prefixes.

For reference, the prefixed version of the code above would look like the following.

```
.box {
    background: #2db34a;
    -webkit-transition-property: background;
       -moz-transition-property: background;
         -o-transition-property: background;
            transition-property: background;
    -webkit-transition-duration: 1s;
       -moz-transition-duration: 1s;
         -o-transition-duration: 1s;
            transition-duration: 1s;
    -webkit-transition-timing-function: linear;
       -moz-transition-timing-function: linear;
         -o-transition-timing-function: linear;
            transition-timing-function: linear;
}
.box:hover {
  background: #ff7b29;
}
```

Transitional Property

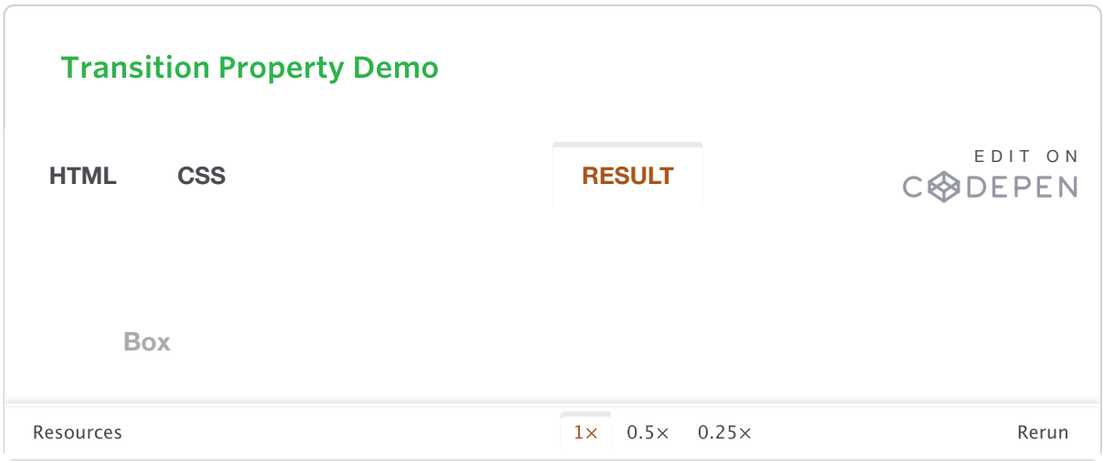
The `transition-property` property determines exactly what properties will be altered in conjunction with the other transitional properties. By default, all of the properties

within an element's different states will be altered upon change. However, only the properties identified within the `transition-property` value will be affected by any transitions.

In the example above, the `background` property is identified in the `transition-property` value. Here the `background` property is the only property that will change over the duration of `1` second in a `linear` fashion. Any other properties included when changing an element's state, but not included within the `transition-property` value, will not receive the transition behaviors as set by the `transition-duration` or `transition-timing-function` properties.

If multiple properties need to be transitioned they may be comma separated within the `transition-property` value. Additionally, the keyword value `all` may be used to transition all properties of an element.

```
.box {
    background: #2db34a;
    border-radius: 6px
    transition-property: background, border-radius;
    transition-duration: 1s;
    transition-timing-function: linear;
  }
  .box:hover {
    background: #ff7b29;
    border-radius: 50%;
  }
```



Transitional Properties

It is important to note, **not all properties may be transitioned**, only properties that have an identifiable halfway point. Colors, font sizes, and the alike may be transitioned from one value to another as they have recognizable values in-between one another. The `display` property, for example, may not be transitioned as it does not have any midpoint. A handful of the more popular transitional properties include the following.

- `background-color`
- `background-position`
- `border-color`
- `border-width`
- `border-spacing`
- `bottom`
- `clip`

- color
- crop
- font-size
- font-weight
- height
- left
- letter-spacing
- line-height
- margin
- max-height
- max-width
- min-height
- min-width
- opacity
- outline-color
- outline-offset
- outline-width
- padding
- right
- text-indent
- text-shadow
- top
- vertical-align
- visibility
- width
- word-spacing
- z-index

Transition Duration

The duration in which a transition takes place is set using the `transition-duration` property. The value of this property can be set using general timing values, including seconds (`s`) and milliseconds (`ms`). These timing values may also come in fractional measurements, `.2s` for example.

When transitioning multiple properties you can set multiple durations, one for each property. As with the `transition-property` property value, multiple durations can be declared using comma separated values. The order of these values when identifying individual properties and durations does matter. For example, the first property identified within the `transition-property` property will match up with the first time identified within the `transition-duration` property, and so forth.

If multiple properties are being transitioned with only one duration value declared, that one value will be the duration of all the transitioned properties.

```
.box {
  background: #2db34a;
  border-radius: 6px;
  transition-property: background, border-radius;
  transition-duration: .2s, 1s;
  transition-timing-function: linear;
}
.box:hover {
  background: #ff7b29;
  border-radius: 50%;
}
```

```
11
```

Transition Duration Demo

Transition Timing

The `transition-timing-function` property is used to set the speed in which a transition will move. Knowing the duration from the `transition-duration` property a transition can have multiple speeds within a single duration. A few of the more popular keyword values for the `transition-timing-function` property include `linear`, `ease-in`, `ease-out`, and `ease-in-out`.

The `linear` keyword value identifies a transition moving in a constant speed from one state to another. The `ease-in` value identifies a transition that starts slowly and speeds up throughout the transition, while the `ease-out` value identifies a transition that starts quickly and slows down throughout the transition. The `ease-in-out` value identifies a transition that starts slowly, speeds up in the middle, then slows down again before ending.

Each timing function has a cubic-bezier curve behind it, which can be specifically set using the `cubic-bezier(x1, y1, x2, y2)` value. Additional values include `step-start`, `step-stop`, and a uniquely identified `steps(number_of_steps, direction)` value.

When transitioning multiple properties, you can identify multiple timing functions. These timing function values, as with other transition property values, may be declared as comma separated values.

```
.box {
  background: #2db34a;
  border-radius: 6px;
  transition-property: background, border-radius;
  transition-duration: .2s, 1s;
  transition-timing-function: linear, ease-in;
}
.box:hover {
  background: #ff7b29;
  border-radius: 50%;
}
```

Transition Timing Demo

Transition Delay

On top of declaring the transition property, duration, and timing function, you can also set a delay with the `transition-delay` property. The delay sets a time value, seconds or milliseconds, that determines how long a transition should be stalled before executing. As with all other transition properties, to delay numerous transitions, each delay can be declared as comma separated values.

```
.box {
  background: #2db34a;
  border-radius: 6px
  transition-property: background, border-radius;
  transition-duration: .2s, 1s;
  transition-timing-function: linear, ease-in;
  transition-delay: 0s, 1s;
}
.box:hover {
  background: #ff7b29;
  border-radius: 50%;
}
```

Transition Delay Demo

Shorthand Transitions

Declaring every transition property individually can become quite intensive, especially with vendor prefixes. Fortunately there is a shorthand property, `transition`, capable of supporting all of these different properties and values. Using the `transition` value alone, you can set every transition value in the order of `transition-property`, `transition-duration`, `transition-timing-function`, and lastly `transition-delay`. Do not use commas with these values unless you are identifying numerous transitions.

To set numerous transitions at once, set each individual group of transition values, then use a comma to separate each additional group of transition values.

```
.box {
  background: #2db34a;
  border-radius: 6px;
  transition: background .2s linear, border-radius 1s ease-in 1s;
}
.box:hover {
  color: #ff7b29;
  border-radius: 50%;
}
```

Shorthand Transitions Demo

Transitional Button

HTML

```
<button>Awesome Button</button>
```

CSS

```
button {
  border: 0;
  background: #0087cc;
  border-radius: 4px;
  box-shadow: 0 5px 0 #006599;
  color: #fff;
  cursor: pointer;
  font: inherit;
  margin: 0;
  outline: 0;
  padding: 12px 20px;
  transition: all .1s linear;
}
button:active {
  box-shadow: 0 2px 0 #006599;

```

```
    transform: translateY(3px);
  }
```

Demo

Card Flip

HTML

```
<div class="card-container">
  <div class="card">
    <div class="side">...</div>
    <div class="side back">...</div>
  </div>
</div>
```

CSS

```
.card-container {
  height: 150px;
  perspective: 600;
  position: relative;
  width: 150px;
}
.card {
  height: 100%;
  position: absolute;
  transform-style: preserve-3d;
  transition: all 1s ease-in-out;
  width: 100%;
}
.card:hover {
  transform: rotateY(180deg);
}
.card .side {
  backface-visibility: hidden;
  height: 100%;
  position: absolute;
  width: 100%;
}
.card .back {
  transform: rotateY(180deg);
}
```

```
25
```

Demo

Animations

Transitions do a great job of building out visual interactions from one state to another, and are perfect for these kinds of single state changes. However, when more control is required, transitions need to have multiple states. In return, this is where animations pick up where transitions leave off.

Animations Keyframes

To set multiple points at which an element should undergo a transition, use the `@keyframes` rule. The `@keyframes` rule includes the animation name, any animation breakpoints, and the properties intended to be animated.

```
@keyframes slide {
  0% {
    left: 0;
    top: 0;
  }
  50% {
    left: 244px;
    top: 100px;
  }
  100% {
    left: 488px;
    top: 0;
  }
}
```

Vendor Prefixing the Keyframe Rule

The `@keyframes` rule must be vendor prefixed, just like all of the other `transition` and `animation` properties. The vendor prefixes for the `@keyframes` rule look like the following:

- `@-moz-keyframes`
- `@-o-keyframes`
- `@-webkit-keyframes`

The animation above is named `slide`, stated directly after the opening `@keyframes` rule. The different keyframe breakpoints are set using percentages, starting at `0%` and working to `100%` with an intermediate breakpoint at `50%`. The keywords `from` and `to` could be used in place of `0%` and `100%` if wished. Additional breakpoints, besides `50%`, may also be stated. The element properties to be animated are listed inside each of the breakpoints, `left` and `top` in the example above.

It is important to note, as with transitions only individual properties may be animated. Consider how you might move an element from top to bottom for example. Trying to animate from `top: 0;` to `bottom: 0;` will not work, because animations can only apply a transition within a single property, not from one property to another. In this case, the element will need to be animated from `top: 0;` to `top: 100%;`.

Animations Keyframes Demo

Hover over the ball below to see the animation in action.

Animation Name

Once the keyframes for an animation have been declared they need to be assigned to an element. To do so, the `animation-name` property is used with the animation name, identified from the `@keyframes` rule, as the property value. The `animation-name` declaration is applied to the element in which the animation is to be applied to.

```
.stage:hover .ball {
  animation-name: slide;
}
```

Using the `animation-name` property alone isn't enough though. You also need to declare an `animation-duration` property and value so that the browser knows how long an animation should take to complete.

Animation Duration, Timing Function, & Delay

Once you have declared the `animation-name` property on an element, animations behave similarly to transitions. They include a duration, timing function, and delay if

desired. To start, animations need a duration declared using the `animation-duration` property. As with transitions, the duration may be set in seconds or milliseconds.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
}
```

A timing function and delay can be declared using the `animation-timing-function` and `animation-delay` properties respectively. The values for these properties mimic and behave just as they do with transitions.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
}
```

The animation below should cause the `ball` to bounce once while moving to the left, however only when hovering over the `stage`.

HTML

```
<div class="stage">
  <figure class="ball"></figure>
</div>
```

CSS

```
 @keyframes slide {
   0% {
     left: 0;
     top: 0;
   }
   50% {
     left: 244px;
     top: 100px;
   }
   100% {
     left: 488px;
     top: 0;
   }
 }
 .stage {
   height: 150px;
   position: relative;
 }
```

```
.ball {
    height: 50px;
    position: absolute;
    width: 50px;
}
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
}
```

Animation Demo

Hover over the ball below to see the animation in action.

Customizing Animations

Animations also provide the ability to further customize an element's behavior, including the ability to declare the number of times an animation runs, as well as the direction in which an animation completes.

Animation Iteration

By default, animations run their cycle once from beginning to end and then stop. To have an animation repeat itself numerous times the `animation-iteration-count` property may be used. Values for the `animation-iteration-count` property include either an integer or the `infinite` keyword. Using an integer will repeat the animation as many times as specified, while the `infinite` keyword will repeat the animation indefinitely in a never ending fashion.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
  animation-iteration-count: infinite;
}
```

Animation Iteration Demo

Hover over the ball below to see the animation in action.

Animation Direction

On top of being able to set the number of times an animation repeats, you may also declare the direction an animation completes using the `animation-direction` property. Values for the `animation-direction` property include `normal`, `reverse`, `alternate`, and `alternate-reverse`.

The `normal` value plays an animation as intended from beginning to end. The `reverse` value will play the animation exactly opposite as identified within the `@keyframes` rule, thus starting at `100%` and working backwards to `0%`.

The `alternate` value will play an animation forwards then backwards. Within the keyframes that includes running forward from `0%` to `100%` and then backwards from `100%` to `0%`. Using the `animation-iteration-count` property may limit the number of times an animation runs both forwards and backwards. The count starts at `1` running an animation forwards from `0%` to `100%`, then adds `1` running an animation backwards from `100%` to `0%`. Combining for a total of `2` iterations. The `alternate` value also inverses any timing functions when playing in reverse. If an animation uses the `ease-in` value going from `0%` to `100%`, it then uses the `ease-out` value going from `100%` to `0%`.

Lastly, the `alternate-reverse` value combines both the `alternate` and `reverse` values, running an animation backwards then forwards. The `alternate-reverse` value starts at `100%` running to `0%` and then back to `100%` again.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

Animation Direction Demo

Hover over the ball below to see the animation in action.

Animation Play State

The `animation-play-state` property allows an animation to be played or paused using the `running` and `paused` keyword values respectively. When you play a paused animation, it will resume running from its current state rather than starting from the very beginning again.

In the example below the `animation-play-state` property is set to `paused` when making the `stage` active by clicking on it. Notice how the animation will temporarily pause until you let up on the mouse.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
.stage:active .ball {
  animation-play-state: paused;
}
```

Animation Play State Demo

Hover over the ball below to see the animation in action. Click to pause the animation.

Animation Fill Mode

The `animation-fill-mode` property identifies how an element should be styled either before, after, or before and after an animation is run. The `animation-fill-mode` property accepts four keyword values, including `none`, `forwards`, `backwards`, and `both`.

The `none` value will not apply any styles to an element before or after an animation has been run.

The `forwards` value will keep the styles declared within the last specified keyframe. These styles may, however, be affected by the `animation-direction` and `animation-iteration-count` property values, changing exactly where an animation ends.

The `backwards` value will apply the styles within the first specified keyframe as soon as being identified, before the animation has been run. This does include applying those styles during any time that may be set within an animation delay. The `backwards` value may also be affected by the `animation-direction` property value.

Lastly, the `both` value will apply the behaviors from both the `forwards` and `backwards` values.

```
.stage:hover .ball {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: .5s;
  animation-fill-mode: forwards;
}
.stage:active .ball {
  animation-play-state: paused;
}
```

Animation Fill Mode Demo

Hover over the ball below to see the animation in action. Click to pause the animation.

Shorthand Animations

Fortunately animations, just like transitions, can be written out in a shorthand format. This is accomplished with one `animation` property, rather than multiple declarations. The order of values within the `animation` property should be `animation-name`, `animation-duration`, `animation-timing-function`, `animation-delay`, `animation-`

`iteration-count`, `animation-direction`, `animation-fill-mode`, and lastly `animation-play-state`.

```
.stage:hover .ball {
  animation: slide 2s ease-in-out .5s infinite alternate;
}
.stage:active .ball {
  animation-play-state: paused;
}
```

Shorthand Animations Demo

Hover over the ball below to see the animation in action. Click to pause the animation.

Resources & Links

- Understanding CSS3 Transitions via A List Apart
- CSS Cubic-Bezier Builder via Rob LaPlaca
- The Guide To CSS Animation: Principles and Examples via Smashing Magazine
- Using CSS Animations via Mozilla Developer Network

Lesson 7
Transforms

Lesson 9
Feature Support & Polyfills