CSS Transitions and Transforms

CSS3 includes transitions and transforms which allow you to create a richer experience without Javascript. Transitions and transforms allow you to add simple animations to your page via CSS. It is still recommended to use vendor prefixes for the broadest support however.

In the simplest terms, **transforms** involve moving elements in some way (rotations, sizing, moving elements around in a page) while **transitions** involve changing from one state to another (how long will it take to move from one size to another or the transition from one position to another).

Transitions

Transitions are often used in conjunction with transforms and allows you to animate the change from one state to another but transforms are not necessary. Transitions can be used on any animatable property (see pages 11 to 12 for the list of CSS animatable properties).

Transitions allow you to change from one state to another state **gradually**. Without transitions, the element which is being changed will change states suddenly rather than over time. For example, if you want an element to grow in size on hover, without a transition, the element will just jump to the larger size instead of *growing* to the larger size. Because you define a duration for a transition, you can see the change happen gradually (i.e. you see the animation from one state to another happen over the duration of your transition).

Example.

Add the following to your HTML page.

```
<h2>My first transition</h2>
<div id="square-one">First square</div>
```

Now add the following to your CSS file. This will make the square div change from blue to red when it is hovered over.

```
#square-one {
  width: 100px;
  height: 100px;
  background-color: blue;
  color: #fff;
  padding: 10px;
  margin-left: 200px;
}
#square-one:hover {
  background-color: red;
}
```

Notice that the square just changes colour abruptly. Add the transition to your CSS to make the colour change gradually (i.e. animate the background-color change). Add the transition to the `#square-one`

CSS block not on the `:hover` one. This means that the transition will always apply when the state changes.

```
#square-one {
  width: 100px;
  height: 100px;
  background-color: blue;
  color: #fff;
  padding: 10px;
  margin-left: 200px;
  transition: all 1s;
}
```

There are two **required** transition values: `transition-property` and `transition-duration`. The "`all`" is the `transition-property` value and just means that you want the transition to apply to all animatable properties, so if more than one animatable property is being changed on hover, all those changes will be transitioned. The `transition-duration` value is set to 1s, which means that the change in state will occur over 1 second.

Two optional values you can set are: `transition-timing-function` and `transition-delay`. In the above example, only the `transition-property` and `transition-duration` were set.

You can set them separately, but it's better to use the shorthand. The shorthand for the above values is (required properties are bolded):

```
transition: [property] [duration] [timing-function] [delay];
```

You will need to use the vendor prefix `-webkit-` for Safari 6. So for Safari 6, the property will be:

```
-webkit-transition: [your values];
```

As always, your vendor-prefixed statements comes before the non-prefixed statement. Transitions are only supported for IE for versions 10+

transition-property

This tells the browser which property the transition applies to. For example, if there are multiple property changes between states (e.g. background-color, size, etc.) you can set this to "`all`" so that the transition will apply to all property changes. If you only want it to apply to background-color changes, set this to "`background-color`". If you only want to apply the transition to transforms, set this to "`transform`".

Setting this alone (and not in the shorthand) is done by the following in your CSS declaration block (note that `all` is the default):

```
transition-property: all;
```

transition-duration

This is where you set how long the transition will take. In the very first example, the transition duration was set to 1 second, meaning that the transition should take 1 second (i.e. 1 second to change background color). The unit for seconds is "s". Remember that there should be no space between the value and the unit (e.g. "`1s`" is correct but "`1 s`" is not).

transition-timing-function

This is the timing *type*. The different built-in types are as follows:

- ease
- ease-in
- ease-out
- ease-in-out
- linear

ease

This is the default transition-timing-function. If you don't set a value for this, it will be assumed to be "ease". This makes the transition start off slower then speed up quickly before slowing again right at the end.

ease-in

This means that the transition animation should start slow, then speed up at the end.

ease-out

This means that the transition animation should start fast, then end slow.

ease-in-out

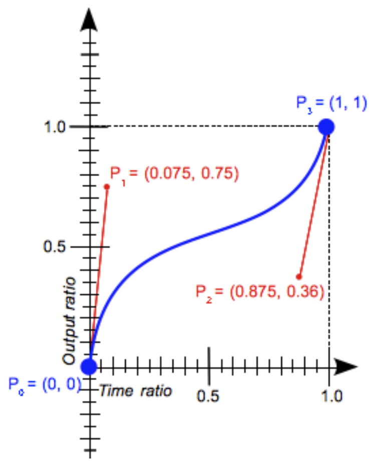
This means that the transition animation should start slow, speed up, then end slow.

linear

This timing-function type specifies that the transition animation should occur at a constant, or linear, speed.

cubic-bezier($P1_x$, $P1_y$, $P2_x$, $P2_y$)

You can also set a custom timing function using a Bézier curve. A cubic Bézier curve looks like the following (taken from MDN):



In the case of `transition-timing-function`, P0 and P3 are taken to be (0, 0) and (1, 1), respectively, so you only need to specify the coordinates of P1 and P2. P1 and P2 are points which, together with P0 and P3, make up tangents for the curve at P0 and P3. So, P0P1 is the tangent line at P0 while P2P3 is the tangent line at P1. (If you've ever played around with shape paths or curved paths in Photoshop or Illustrator, those paths are made up of Bézier curves.)

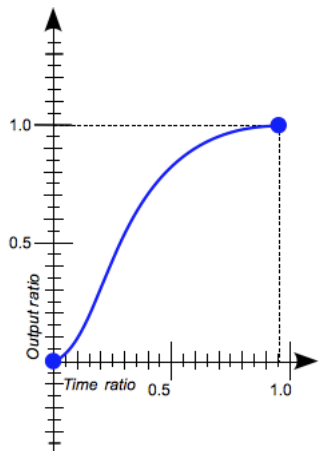
P2 beyond the box defined by (1, 1). This creates a kind of bounce effect. Try the following out with the previous square example:

```
transition: all 1s cubic-bezier(0.075, 0.75, 0.75, 1.5);
```

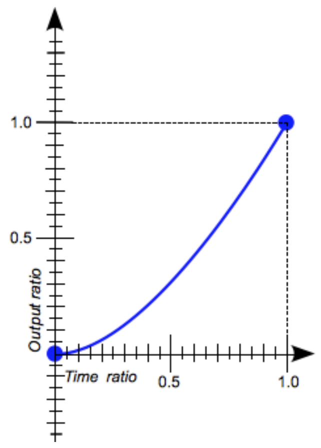
You can actually use the developer tools in your browser to create a Bézier curve using a GUI. You can add the cubic-bezier() function with some random numbers (if you like) in the CSS, then in the element inspector, go to the CSS rule for the item being transitioned (by inspecting the transitioned element) and clicking on the icon next to the cubic-bezier() function. This will open up a graph which you can tweak by dragging and dropping points.

In case you are curious, the curves for the built-in functions are as follows (images are taken from the Mozilla Developer Network):

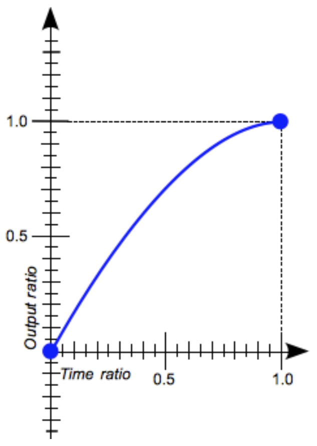
ease



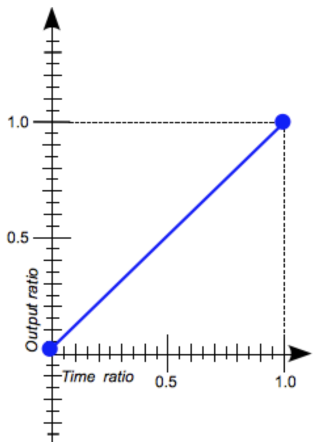
ease-in



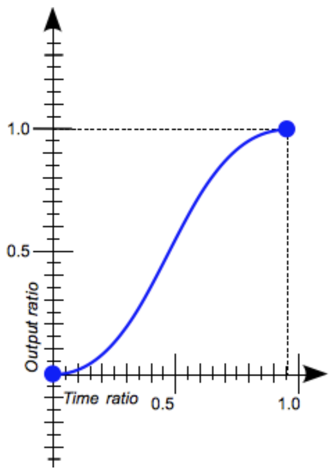
ease-out



linear



ease-in-out



transition-delay

This allows you to specify how long to delay **before** the transition occurs.
Example. To demonstrate this property, let's make a countdown timer. Make 6 divs with a green circle background. The first five divs will contain 5, 4, 3, 2, and 1 respectively. The sixth div will contain "Go!". Enclose the 6 divs in a container so that we can trigger the transitions when hovering over the container.

```
<h2>Countdown</h2>
<div id="countdown">
  <div class="circle circle1">5</div>
  <div class="circle circle2">4</div>
  <div class="circle circle3">3</div>
  <div class="circle circle4">2</div>
  <div class="circle circle5">1</div>
  <div class="circle circle6">Go!</div>
</div>
```

In your CSS, you want each circle div to be green. You can also center your text and make sure the divs are side-by-side.

```
.circle {
  border-radius: 50%;
  width: 40px;
  height: 40px;
  background-color: green;
  text-align: center;
  padding: 10px;
  display: inline-block;
  color: #fff;
}
```

Now, when the #countdown is hovered on, you want the circle background-color to change to red.

```
#countdown:hover .circle {
  background-color: red;
}
```

Now you'll see why you needed to also give each circle a different class name. You need this to set a different transition duration for each circle. After 1 second, one circle each will turn red starting from 5 down until "Go!". (Or, you can forego the different class names and use nth-child.)

```
.circle1 {
  transition: all 0.3s 1s;
}
.circle2 {
  transition: all 0.3s 2s;
}
.circle3 {
  transition: all 0.3s 3s;
}
.circle4 {
  transition: all 0.3s 4s;
}
.circle5 {
  transition: all 0.3s 5s;
}
.circle6 {
  transition: all 0.3s 6s;
}
```

Multiple transitions on an element

You can add define multiple transitions on an element in the event that you want to specify transitions for certain animatable properties but not for *all*.

To do this, separate the different transitions using commas in **one transition property statement**.

For example:

```
.circle6 {
  transition: background-color 0.3s 6s,
              color 0.3s 6s;
}
```

Transforms

Transforms involve moving or changing the shape or size of your element. There is **no** animation for these unless you use transitions or keyframe animations. (Transforms are animatable properties but changes are not animated by default.) There are following types of transforms:

- scale
- rotate
- skew
- translate

There is also a special property that can be used with transforms: `transform-origin`. This property is especially useful for rotations.

For transforms, use the `-webkit-` vendor prefix for Safari and Chrome and the `-ms-` vendor prefix for IE9. Non-prefixed transforms are supported for IE10+.

scale()

Scaling allows you to enlarge or shrink an element. The value for scale is a multiplier (it has no unit).

You can also use `scaleX()` or `scaleY()` to expand or shrink along the X-axis and Y-axis, respectively. So, to scale 2 times the size along the X-axis, use:

```
transform: scaleX(2);
```

The `scale()` function is the shorthand function. You can specify the scaling in shorthand as follows:

```
transform: scale([scaleX value], [scaleY value]);
```

Note: If you only have one value in `scale()`, this value will apply for both axes, hence `scale(1.5)` meant to grow 1.5 times along both axes. In other words, scale proportionately 1.5 times.

Example. Create a square div and make it grow twice in size on hover.

```
<div id="square-grow">Square</div>

#square-grow {
  background-color: red;
  width: 100px;
  height: 100px;
  padding: 10px;
  margin-left: 200px;
  transition: all 2s;
}
#square-grow:hover {
  transform: scale(2);
}
```

rotate()

This allows you to rotate an element. The format is:

```
transform: rotate([number]deg);
```

Don't forget the deg unit suffix. Positive values make the rotation move clockwise while negative values make the rotation move counter-clockwise. Note that you can specify values higher than 360deg for more than one rotation.

Example. Create another square div with a yellow background and make it rotate twice on hover.

```
<h2>Rotation</h2>
<div id="square-two">Rotate me</div>
```

Add the CSS to make the square and make it yellow. Also move to the right a bit so you can see the rotation well.

```
#square-two {
  background-color: yellow;
  width: 100px;
  height: 100px;
  padding: 10px;
  margin-left: 200px;
}
```

Now make it rotate on hover and make it last 3 seconds. To rotate twice, you will need to set it to rotate for 720 degrees, which is 360 degrees times two.

```
#square-two {
  background-color: yellow;
  width: 100px;
  height: 100px;
  padding: 10px;
  margin-left: 200px;
  transition: all 3s;
}
#square-two:hover {
  transform: rotate(720deg);
}
```

Notice that you **DO NOT need** to use transforms with transitions if you don't want any animation. For example, if you only want to position some element at an angle at all times, you only need to use the rotate transform without a transition.

transform-origin

This allows you to specify an origin point of your transform. This property is separate from the transform property so you cannot specify the origin within the transform declaration, but it works together *with* transforms. By default, the origin is the center of the element, which is the same as:

```
transform-origin: 50% 50%;
```

or

```
transform-origin: center center; /* using the "center" keyword */
```

The syntax is:

```
transform-origin: [x-location] [y-location];
```

The X location can be specified with keywords (e.g. `left`, `right`, or `center`), with a length (e.g. px, em, etc.) or with percentages, where 0% is the left-most location and 100% is the right-most location. Similarly, the Y location can be specified with keywords (e.g. `top` or `bottom`), with a length (e.g. px, em, etc.) or with percentages, where 0% is the top-most location and 100% is the bottom-most location.

This property is especially useful if you want to change the point around which an element will rotate. Try changing the rotate example to rotate around the right bottom corner.

skew()

Skewing an element will skew or squish its *shape* by tilting the shape in one or other direction. Skew can be set by the following:

```
transform: skewX([number]deg);
transform: skewY([number]deg);
transform: skew([skewX value], [skewY value]);
```

You can use `skewX()` to tilt horizontally or `skewY()` to tilt vertically or you can use the shorthand `skew()` to tilt in both axes. Positive angles tilt elements left in the case of `skewX` and tilt elements downward in the case of `skewY`. Negative angles tilt elements right in the case of `skewX` and tilt elements upward in the case of `skewY`. Notice that there are no spaces between the numbers and the `deg` unit suffix.

Skewing affects all child elements as well so if you don't want the children to skew, you will need to skew the child element again in the equal and opposite value to negate the skewing.

Example. Create an orange parallelogram with the white text "Parallelogram". Make sure the text is not skewed.

```
<h2>Skew</h2>
<div id="parallelogram"><div>Parallelogram</div></div>
```

(Notice that I enclosed the text in a second div.) Now add the CSS to make the parallelogram. First make a rectangle 150px wide and 100px tall then skew left by 25 degrees (this can be any number if the instructions do not specify).

```
#parallelogram {
  width: 150px;
  height: 100px;
  padding: 20px;
  color: #fff;
  margin-left: 200px;
  background-color: orange;
  transform: skewX(25deg);
}
```

If you check in the browser, you'll notice that the text is skewed as well. To correct that, you will need to skew the text back in the opposite direction. This is why the text was enclosed in a second div.

```
#parallelogram > div {
  transform: skewX(-25deg);
}
```

translate()

To translate an element means to *move* an element. The syntax is as follows:

```
transform: translate([X value], [Y value]);
```

A positive X value will shift the element to the right by the specified length while a negative X value will shift the element to the left by the specified length. Similarly, a positive Y value will shift the element down by the specified length while a negative Y value will shift the element up by the specified length.

For example, `translate(100px, 50px)` means to shift the element to the right by 100px and downward by 50px.

When using translate() with percentage values, the percentage length is based on the element's width and height, so **`translateY(100%)`** means the element will shift down by the same distance as the element's height.

Combining transforms

You can apply more than one transform all in one transform statement. Just use the transform shorthand and string each transform you want to use in the same statement. For example:

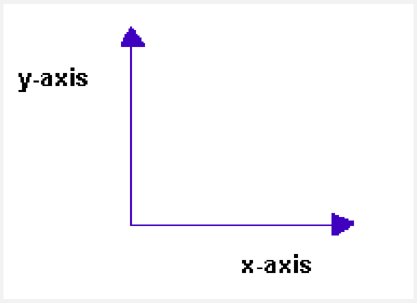
```
transform: scale(2) rotate(360deg);
```

3D transforms

There are 3D transform functions as well:

- `rotateX()`
- `rotateY()`
- `rotateZ()`
- `translateZ()`
- `scaleZ()`

These work similarly to their 2D counterparts. The Z axis is pointing outwards from the screen. Think of the X and Y axes like so:



The Z-axis is perpendicular and pointing out from the screen. (Think of `z-index` and what it does.)

There are also 3D shorthand versions available:

```
translate3d([translateX], [translateY], [translateZ]);
scale3d([scaleX], [scaleY], [scaleZ]);
rotate3d([x], [y], [z], [angle]);
```

The notation for `rotate3d()` can be a little confusing. For those really interested, you can find a full explanation including the math for how you can figure out the numbers you need for `x`, `y`, `z`, and `angle` if you want to combine angles here: http://stackoverflow.com/questions/15207351/rotate3d-shorthand#answer-15208858.

To take a look at some good examples (if you're interested), go to:

https://3dtransforms.desandro.com/

CSS Animatable Properties

Recall that there are certain properties you can set for transition-property.

For example:

```
transition: transform 1s;
```

```
transition: color 1s;
transition: background-color 5s;
transition: all 3s;
```

This will animate the transitions for those properties. The `all` keyword animates all property changes for properties which are *animatable*. If you are only planning on animating only **one** property, list the property rather than using `all` because this is more efficient (note that `all` is the default).

The animatable properties are:

- background-color
- background-position
- background-size
- border-radius
- border-width
- box-shadow (Remember: this is the shadow of an *element*. Elements are boxes.)
- color
- font-size
- height
- letter-spacing
- line-height
- opacity
- outline-width
- outline-offset
- outline-color
- padding
- text-indent
- text-shadow (Remember: this is the shadow for *text*.)
- top
- transform
- transform-origin
- width

You can check the W3C spec as well for a full and more up-to-date list of animatable properties.

Some examples showing animations of these properties can be found here:

https://projects.verou.me/animatable/

CSS Transitions and Javascript

In one of the previous exercises, you've seen it is possible to trigger CSS transition animations by adding a CSS class (containing the transition) to an element via Javascript (see Exercise 2).

It is also possible to detect the *end* of a CSS transition via Javascript.

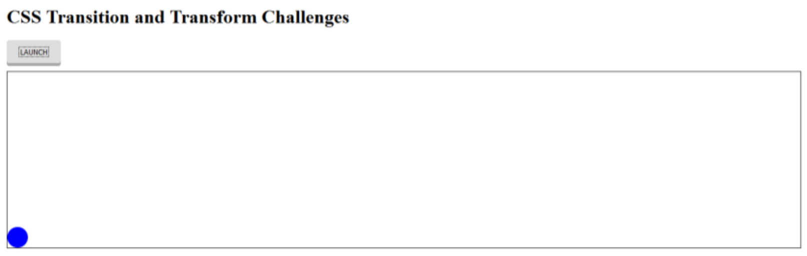
```
//myElement is the element the transition is occurring on
```

```
var myElement = document.getElementById("my-element");

myElement.addEventListener("transitionend", function(){
  //callback function to run upon end of transition
});
```

This allows you to daisy chain a series of events. In other words, you can trigger something to run only **after the animated transition has ended**.

Challenge: Create a basic ball launcher using only transitions and transforms. (A working example will be shown in class.) On clicking a push button labelled "Launch", a little block will launch a ball. After the ball falls out of view of the stage, the ball appears back at the start (i.e. it restarts). This will require the `transitionend` event to reset everything after the transitions complete.


CSS Transition and Transform Challenges
LAUNCH