**Card Flip[[1]](#footnote-1)**

We now have all the tools to start making 3D objects. Let’s get started with the basics, flipping a card.

Here’s the basic markup we’ll need:

<div class="scene">

<div class="card">

<div class="card\_\_face card\_\_face--front">front</div>

<div class="card\_\_face card\_\_face--back">back</div>

</div>

</div>

|  |
| --- |
|  |

The .scene will house the 3D space. The .card acts as the 3D object. Two separate .card\_\_face elements are used for the faces of the card. I recommend using this same pattern for any 3D transform: scene, object, and faces. Keeping the 3D space element and the object separate element establishes a paradigm that is simple to understand and easier to style.

We’re ready for some 3D stylin’. First, apply necessary perspective to the containing 3D space, along with any size or positioning styles.

.scene {

width: 200px;

height: 260px;

perspective: 600px;

}

Now the .card element can be transformed in its parent’s 3D space. We’ll add width: 100%;and height: 100%; so the card’s transform-origin will occur in the center of container. More on transform-origin later. position: relative is used to position card faces absolutely.

Let’s add a CSS3 transition so users can see the transform take effect.

.card {

width: 100%;

height: 100%;

position: relative;

transition: transform 1s;

transform-style: preserve-3d;

}

An element’s perspective only applies to direct descendant children, in this case .card. In order for subsequent children to inherit a parent’s perspective, and live in the same 3D space, the parent can pass along its perspective with transform-style: preserve-3d. Without 3D transform-style, the faces of the card would be flattened with its parents, and the back face’s rotation would be nullified.

To position the faces in 3D space, we’ll need to reset their positions in 2D with position: absolute. In order to hide the back-side of the faces when they are faced away from the viewer, we use backface-visibility: hidden.

.card\_\_face {

position: absolute;

height: 100%;

width: 100%;

backface-visibility: hidden;

}

To flip .card\_\_face--back, we add a basic 3D transform of rotateY(180deg).

.card\_\_face--front {

background: red;

}

.card\_\_face--back {

background: blue;

transform: rotateY( 180deg );

}

With the faces in place, the .card requires a corresponding style for when it is flipped.

.card.is-flipped {

transform: rotateY(180deg);

}

Now we have a working 3D object. To flip the card, we can toggle the is-flipped class. When .is-flipped, the .card will rotate 180 degrees, thus exposing .card\_\_face--back.

|  |
| --- |
|  |

[Edit this demo on CodePen](https://codepen.io/desandro/pen/LmWoWe)

|  |
| --- |
|  |

## Slide-flip

Take another look at the Weather App 3D transition. You’ll notice that it’s not quite the same effect as our previous demo. If you follow the right edge of the card, you’ll find that it stays flush with the container. Instead of pivoting from the horizontal center, it pivots on that right edge. But the transition is not just a rotation – the edge moves horizontally from right to left. We can reproduce this transition just by modifying a couple lines of CSS from our original card flip demo.

The pivot point for the rotation occurs at the right side of the card. By default, the transform-origin of an element is at its horizontal and vertical center (50% 50% or center center). An element’s transforms are applied from its transform-origin. Let’s change it to the right side:

.card { transform-origin: center right; }

That flip now needs some horizontal movement with translateX. We’ll set the rotation to -180deg so it flips right side out.

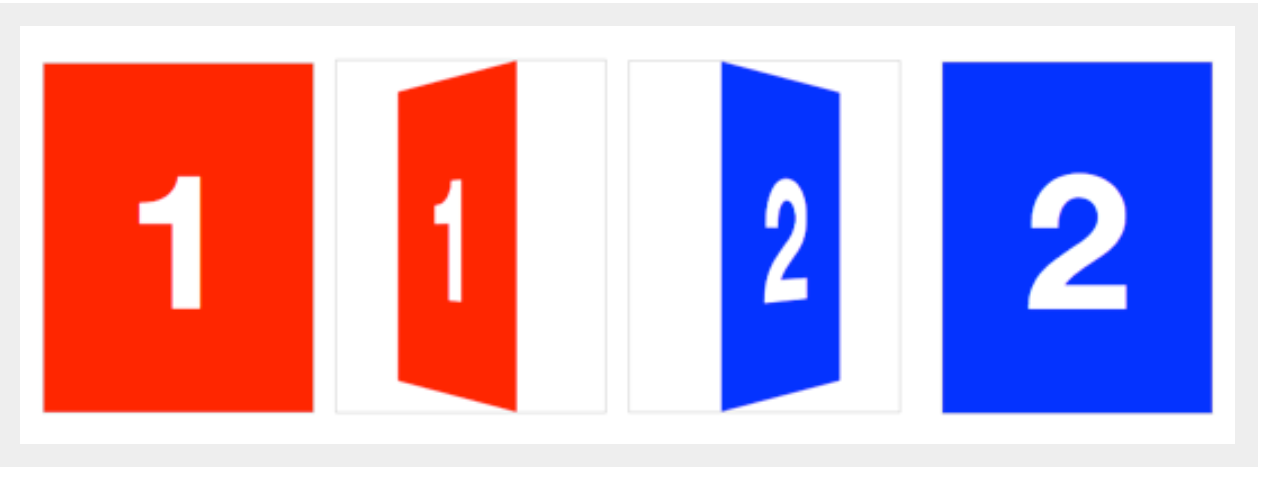
.card.is-flipped {

transform: translateX(-100%) rotateY(-180deg);

}

|  |
| --- |
|  |

[Edit this demo on CodePen](https://codepen.io/desandro/pen/LmWozd)



1. <https://3dtransforms.desandro.com/card-flip> [↑](#footnote-ref-1)