Bootcamp Web Class Day 07:
CSS Transforms, Transitions and Animations

Citation

- Much of the information here comes from the invaluable blog CSS Tricks by Chris Coyer. Thank you Chris!!!

To Cover:

- Transforms
- Transitions
- Animations

General

- These are experimental properties so we must use the vendor prefix for browser compatibility. For example when using the `transform: scale()` property:

  ```
  -webkit-transform: scale(1.5);
  -moz-transform: scale(1.5);
   -ms-transform: scale(1.5);
   -o-transform: scale(1.5);
      transform: scale(1.5);
  ```

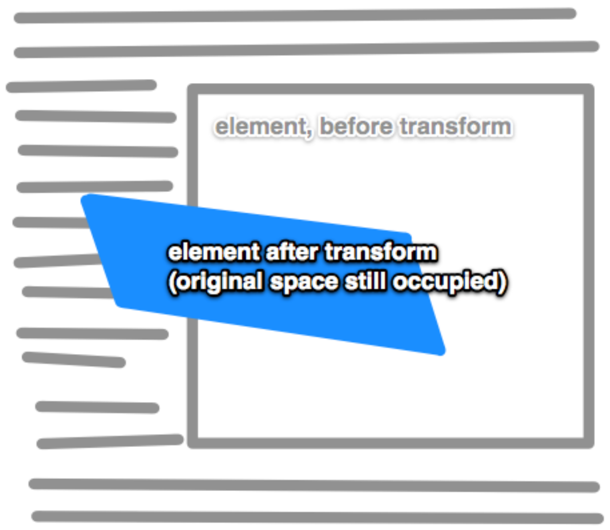
- **tip:** install this handy Vendor Prefix Sublime Text 2 snippet by Jack Brewer

  - then in Sublime type `-` and press the `tab` key.

- **note:** for the remainder of this document it is assumed you are using the vendor prefixes mentioned above.

Transforms

- works like: `transform: transform-function || none; /* can list multiple, space-separated */`
- the actual space on the page that the element occupies remains the same as before transform:



(credit: CSS tricks)

Transform Properties

Scale

- Changes the size of an element.
- Scale values are in proportions.
- can use one value for an even scale or two values for a vertical and horizontal scale.
- like: `transform: scale(value, [value]);`
- or:

```
transform: scaleX(value);
transform: scaleY(value);
```

Rotate

- Rotates clockwise from current position: `transform: rotate(angle);`
- must use the `deg` suffix for units in degrees.

Skew

- sort of like turning a rectangle into a parallelogram
- must use separate x and y functions:

```
  transform: skewX(value);  /* e.g. skewX(-35deg) */
  transform: skewY(value);
```

Translate

- moves the position of an element up or down. Similar to the CSS offset properties.
- Can use shorthand `transform: translate(value [, value]);` or longhand:

```
transform: translateX(value);
transform: translateY(value);
```

Matrix

- a way to combine all the above into one line of code. ***Not human readable / friendly***
- use a tool to do this such as [this goofy one]
- example:
  - `rotate(-45deg) translate(10px,25px)` would be this: `matrix(0.7071067811865476, -0.7071067811865475, 0.7071067811865475, 0.7071067811865476, 24.74873734`

3D Transformations

- Many of the above have 3D options such as
  - `translate3d(x, y, z)` or `translateZ(z)`
  - `scale3d(sx, sy, sz)` or `scaleZ(sz)`
  - rotate:

    ```
    rotateX(value)
    rotateY(value)
    rotate3d(x, y, z)
    ```

  - `matrix3d()`
- `perspective(value)` used for child elements to give them a consistant depth.
- `perspective-origin` configures the perspective of the viewer.

Transform Origin

- used in combination with the other transformation properties in order to change the point of origin.
- for example:

```
.box {
    transform: rotate(360deg);
    transform-origin: top left;
  }
```

- default point of origin is 50% 50%
- Values can be lengths, percentages or the keywords top, left, right, bottom, and center.
- values are horizontal position, vertical position and z position (last is only relevant for 3d transformations)

Transitions

- allows for a smooth change in state with an element
- shorthand syntax:

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

- can use on properties such as background color:

```
  div {
      transition: background-color 0.5s ease;
      background-color: red;
  }
  div:hover {
       background-color: green;
  }
```

- you may specify specific properties to transition or use the `all` for the `transition-property` portion of the shorthand.
- commas can be used to separate different properties:

```
div {
transition: background 0.2s ease,
            padding 0.8s linear;
}
```

- order of properties doesn't matter unless a delay is being used. The first time value will be interpretted as a duration so 2 time values must be used for a delay.
- **note:** not all properties are animatable. There is a list from the W3C here:

Property Name	**Type**
background-color	as color
background-position	as repeatable list of simple list of length, percentage, or calc
border-bottom-color	as color
border-bottom-width	as length
border-left-color	as color
border-left-width	as length
border-right-color	as color
border-right-width	as length
border-spacing	as simple list of length
border-top-color	as color
border-top-width	as length
bottom	as length, percentage, or calc
clip	as rectangle
color	as color
font-size	as length
font-weight	as font weight
height	as length, percentage, or calc
left	as length, percentage, or calc
letter-spacing	as length
line-height	as either number or length
margin-bottom	as length
margin-left	as length
margin-right	as length
margin-top	as length
max-height	as length, percentage, or calc
max-width	as length, percentage, or calc
min-height	as length, percentage, or calc
min-width	as length, percentage, or calc

opacity	as number
outline-color	as color
outline-width	as length
padding-bottom	as length
padding-left	as length
padding-right	as length
padding-top	as length
right	as length, percentage, or calc
text-indent	as length, percentage, or calc
text-shadow	as shadow list
top	as length, percentage, or calc
vertical-align	as length
visibility	as visibility
width	as length, percentage, or calc
word-spacing	as length
z-index	as integer

- it is **important** to specify the transition on the element itself, not inside a pseudo class. In other words this example will transition on hover over but not on hover out:

  ```
  .box {
      width: 150px;
      height: 150px;
      background: red;
      margin-top: 20px;
      margin-left: auto;
      margin-right: auto;
    }

    .box:hover {
      background-color: green;
      cursor: pointer;
      -webkit-transition: background-color 2s ease-out;
      -moz-transition: background-color 2s ease-out;
      -o-transition: background-color 2s ease-out;
      transition: background-color 2s ease-out;
    }
  ```

  (^ what not to do!)

Animations

- The animation property is used to call and control an `@keyframe` animation.
- eg: `animation: animation-name 5s infinite;`
- refers to a `@keyframe` property:

```
@keyframes animation-name {
    0%   { opacity: 0; }
    100% { opacity: 1; }
  }
```

- it's recommended to use the vendor prefixes:

```
 @-webkit-keyframes animation-name {
   0%   { opacity: 0; }
   100% { opacity: 1; }
 }
 @-moz-keyframes animation-name {
   0%   { opacity: 0; }
   100% { opacity: 1; }
 }
 @-o-keyframes animation-name {
   0%   { opacity: 0; }
   100% { opacity: 1; }
 }
 @keyframes animation-name {
   0%   { opacity: 0; }
   100% { opacity: 1; }
 }
```

  and when calling the animation:

```
#box {
  -webkit-animation: animation-name 5s infinite;
  -moz-animation: animation-name 5s infinite;
  -o-animation: animation-name 5s infinite;
  animation: animation-name 5s infinite;
}
```

Calling with separate properties:

```
.box {
 animation-name: bounce;
 animation-duration: 4s;
 animation-iteration-count: 10;
 animation-direction: alternate;
 animation-timing-function: ease-out;
 animation-fill-mode: forwards;
 animation-delay: 2s;
}
```

property name	property value
timing-function	ease, ease-out, ease-in, ease-in-out, linear, cubic-bezier(x1, y1, x2, y2) (e.g. cubic-bezier(0.5, 0.2, 0.3, 1.0))
duration & delay	Xs or Xms
duration-count	X
fill-mode	forwards, backwards, both, none
animation-direction	normal, alternate

Shorthand

- Space different values. Order only matters when specifiying both duration and delay.
- example: `animation: test 1s 2s 3 alternate backwards`

Combine transform and animation

```
@keyframes infinite-spinning {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Mulitple Animations

- comma separate multiple animations on the same selector

```
.animate-this {
   animation:
      first-animation 2s infinite,
      another-animation 1s;
}
```

Steps

- The `steps()` function controls exactly how many keyframes will render in the animation in the timeframe
- for example:

  ```
  @keyframes move {
    from { top: 0; left: 0; }
    to   { top: 100px; left: 100px; }
  }
  ```

- then setting the steps() to 10 (`steps(10)`) will make sure only 10 keyframes will happen in the allotted time, in this case 10 seconds so that there is 1 keyframe 10px movement to the left and down per second:

  ```
  .move {
    animation: move 10s steps(10) infinite alternate;
  }
  ```

- good example of using steps() to make a sprite.

resources / references

- Some good code examples at Shay Howe's tutorial: advanced html css
- CSS Tricks transform blog post
- CSS Tricks intro to CSS animation
- MDN transform doc