CSS3 Transforms and Transitions

You can flip, move, scale items with CSS3 without images or js.

TRANSFORMS
CSS3 gives unprecedent control over elements appearance.
Transform property

You can move elements left, right, up or down…like position:relative. Translation doesn't effect positions of other elements and can only be moved relative to its current posisiton.

Translate(x, y) x—from the left, y-from the top.
--tranlatex for only moving vertically or translate to move horizontally

Older browsers require prefixing

```
.ad-ad2 h1:hover span {
    color: #484848;
    transform: translateX(40px);
}
```

Scaling
--scale(x,y)---scale(1)—stays the same. Scale(.5)—half the size. Scale(2)--double

Rotate()—rotates an element around a point of origin.

```
transform: rotate(10deg) translateX(40px)
scale(1.5);
```

negative degrees mean rotate counterclockwise.

--skew(x, y)---skews the element along the x and y axises.

Changing the Origin of the Transform
--transform-origin. Default is center,

Be careful whether you do the transform or translate first.

In browsers that are not supported, you can use position:relative
Height and width can also be used to alter elements..or changing font-size.

TRANSITIONS—easier in CSS than in JS and more smooth
Steps for creating transitions
1-declare original state as default state.
2-Declare the final state.
3-include the transition function in the default style declaration using transition property, transition duration, transition timeing, transition delay…use webit for older browsers.

---Transition property—defines the css properties fo the element that should be transitioned will all properties being default

Some styles are not transitionable properites…if there is a valid mid-point, it can be transitioned, except…hidden and visible.

“all” can be used to so that every property is anitimated.

Use the -webkit- prefix for all transition-properties to support older browsers.

```
.ad-ad2 h1 span {
    -webkit-transition-property: -webkit-transform;
    transition-property: transform;
}
```

---Transition-duration: sets how long the transition will take…seconds or miliseconds(ms)….(.2s) is a good time or (200ms).

---Transition-timing-function—contorls the pace of the transition for more gradual detain.
--ease, linear, easie-in, ease-out. Or ease-in-out

Set the transition-duration to long when testing this function.
--cubic-bezier will let you define the timing more precise.
--steps function defines the number of steps and the direction with “start” or “end”/

---Transition-delay: introduces a delay before the transition begins---use (s) or (ms) and -webkit-
Negative delay will make it start immediately…and start partway through.

Short-hand

```
.ad-ad2 h1 span {
    transition-property: transform;
    transition-duration: 0.2s;
    transition-timing-function: ease-out;
    transition-delay: 50ms;
}
```

To

```
.ad-ad2 h1 span {
    transition: transform 0.2s ease-out 50ms;
}
```

Multiple transition example:

ANIMATIONS
Transitions are limited in control…but CSS animation lets you control step by step.
--keyframes are snapshots that define a starting or end point of any smooth transition.
Transitions are limited to identifying the frist and last frame.

JS is still best for intricate, stateful UIs with js animation library.

Keyframes
Create a named animation—then attach to an element.
use @-webkit-keyframes for the named animation.
Each frame is a CSS declaration block.---“from” and “to” are keywords to help with values in percentages.

```
@keyframes moveRight {
    from {
    transform: translateX(-50%);
    }
    to {
    transform: translateX(50%);
    }
}

@keyframes appearDisappear {
    0%, 100% {
    opacity: 0;
    }
    20%, 80% {
    opacity: 1;
    }
}

@keyframes bgMove {
    100% {
    background-position: 120% 0;
    }
}
```

AMinmation Properties—use -webkit-

Animation name—used to attach animation to an element.—required to create animation.
```
animation-name: appearDisappear;
```
Within an element declaration.

Animation-druaction---the length of time it takes to have one interation.—not required, but default is 0s

Animation-timing function determines how the animation will progress over its duration.
Ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(), step-start, step-end

Animation-iteration-count lets your define how many times the animation will play through…default is 1…infinite is endless.

Animation-direction, normal (default), reverse, alternate, alternate-reverse.

Animation-delay is used to define how many miliseconds or seconds to wait before the browser begins.
-animation-fill-mode defines what happens before the animation beings when delayed…none, forwards, backwards, both. None is default.

Shorthand Animation Property:

```
.verbose {
    animation-name: appearDisappear;
    animation-duration: 300ms;
    animation-timing-function: ease-in;
    animation-iteration-count: 1;
    animation-direction: alternate;
    animation-delay: 5s;
    animation-fill-mode: backwards;
    animation-play-state: running;
}

/* shorthand */
.concise {
    animation: 300ms ease-in alternate 5s backwards appearDisappear;
}
```

Be Careful not include keywords usch as forwards, running, alternate in animation name.

MOVING ON
Just because you can, doesn't mean you should.

Chapter 12—Canvas, SVG, and Drag and Drop

Canvas—HTML5's Canvas API that allows drawing. You can draw shapes and lines, etc.

Creating a canvas element

```
<canvas>
    Sorry! Your browser doesn't support Canvas.
</canvas>
```

If canvas is not supported.

```
<canvas id="myCanvas" class="myCanvas" width="200" height="200">
    Sorry! Your browser doesn't support Canvas.
</canvas>
```

--width and height should be set here.

```
.myCanvas {    border: dotted 2px black;}
```

Get Context in JS

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
```

Filling the Brush

--strokeStyle or fillStyle—take string, CannvasGradient, CanvasPAttern

When making a rectangle—fillRect for the beginning and ending of the rectangle (outline). strokeRect for the outline.

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
context.strokeStyle = "red";
context.fillStyle = "rgba(0, 0, 255, 0.5)";
context.fillRect(10, 10, 100, 100);
context.strokeRect(10, 10, 100, 100);
```

Variations on fillStyle

--CanvasGradient

CanvasPattern—use pattern and set filSytle…see above

--CanvasGradient uses createLinearGradient() or createRadialGradient()

```
function drawGradient() {
    var canvas = document.getElementById("demo3");
    var context = canvas.getContext("2d");
    context.strokeStyle = "red";
```

```
    var gradient = context.createLinearGradient(0, 0, 0, 200);
gradient.addColorStop(0, "blue");
    gradient.addColorStop(1, "white");
    context.fillStyle = gradient;
    context.fillRect(10, 10, 100, 100);
    context.strokeRect(10, 10, 100, 100);

}
```

--addColorStop and offset.

Other Shapes
--three steps: layout the path, then set the strokeStyle and then call fillRect

-use arc for drawing circle.
The signature for the arc method is: arc(x, y, radius, startAngle, endAngle, anticlockwise).
Example: context.arc(50, 50, 30, 0, Math.PI*2, true);

```
function drawCircle(canvas) {
   var context = canvas.getContext("2d");
   context.beginPath();
   context.arc(50, 50, 30, 0, Math.PI*2, true);
   context.closePath();
   context.strokeStyle = "red";
   context.fillStyle = "blue";
   context.lineWidth = 3;
}—includes linewidth
```

Saving Drawings:
Use API's toDataURL method.---
---When button is clicked:

```
function saveDrawing() {
   var canvas5 = document.getElementById("demo5");
   window.open(canvas5.toDataURL("image/png"));
}
```

Call the save drawing function:

```
var button = document.getElementById("saveButton");
button.addEventListener("click", saveDrawing, false);
```

Drawing an Image to the Canvas:
A function to do it:

```
function drawImageToCanvas() {
   var canvas = document.getElementById("demo6");
   var context = canvas.getContext("2d");
   var image = document.getElementById("myImageElem");
}
```

drawImage() method will redraw an image.

Manipulating Images.—after an image is redrawn to canvas, it can be manipulated.
getImageData method to get image data.
Propterites—width, height, data

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
var image = document.getElementById("myImageElem");
// draw the image at x=0 and y=0 on the canvas
context.drawImage(image, 68, 68);
var imageData = context.getImageData(0, 0, 1, 1);
var pixelData = imageData.data;
console.log(pixelData.length);
```

Converting an Image from Color to Black white—use a for loop to iterate through each pixel in the image and change it to grayscale. Determine the grayscale value for each color. grayscale = red * 0.3 + green * 0.59 + blue * 0.11; Final step is to put it back into Canvas"

Manipulating with Video Canvas
--add an event listener that will put an overlay on the video…use the grayscale example above except the we're drawing the video element onto Canvas instead fo a static image.

Displaying Text on Canvas—use a try/catch block to catch errors and keep the video running smoothly

Accessiblity Concerns—Canvas currently has acessiblity problems.

How prolific is Canvas used in webpages and apps?

SVG—scalable vector graphics

XML is extensible markup language…it is a system meant to annotate text.

SVG images are avaible through the DOM…SVG is in a XML file format…it is more accessable than Canvas

Drawing in SVG
Circle:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 400 400">
   <circle cx="50" cy="50" r="25" fill="red"/>
</svg>
```

--viewbox defines the starting location, width and height of the image
--circle defines the element as a circle with "cx and "cy" coordinagtes of the center circle. The "r" is the radius and "fill" defines the style.

Rectangle:

```
<svg xmlns="http://www.w3.org/2000/svg" viewbox="0 0 400 400">
   <desc>Drawing a rectangle</desc>
      <rect x="10" y="10" width="100" height="100"
```

```
        fill="blue" stroke="red" stroke-width="3"  />

</svg>
```

It create complex images, it is best to use an image tool such as Inkscsape…use to make BWimages.

Openclipart.org is free images without copyright.

SVG Filters

Using the Raphael Library—an open source library to make drawings and animation with SVG. You can call Rapael methods into elements that you are using.

Canvas vs SVG---
Canvas
--allows pixel manipulation
--operates in immediate mode.
--once you finish a drawing it no longer exsists in Canvas—can't add to images later.

SVG—
--accessible to DOM
--retained mode, you can modify later
--File format not method—you can't maniputlate

If you won't need to change objects later…use Canvas, if you do, use SVG

DRAG AND DROP

Drag and Drop API—it allows us to specify which items are dragable and what should happen afer they are dragged.

Unsupported in Android and Apple (apple directs you to use the DOM Touch API instead.

Steps to add Drag and Drop
--set the "draggable" attribute to the elements you want to dray
--add an event istener for the "dragstart" event
--add an event listener for the "dragover" and "drop" events.

--in HTML set attribute "draggable" to "true".—not Boolean

The Data Transfer Object—defines two pieces of info: the type of value we're saving of the element being dragged and the value of the date.

Accepting Dropped Elements—
Dragover fires when you drag an item over the element.
Drop fires when you drip an item on it.

Default behavior does not allow elements to be dropped so you should : event.preventDefault();