Week 8 Notes

CSS3 Transforms and Transitions

Transforms

transform property lets you translate, rotate, scale, and/or skew any element on the page. We can use **transform functions** to manipulate an element's appearance.

Translation

Allow you to move elements left, right, up or down. Similar to **position:relative;**

Move elements without impacting the flow of the document. Can only move relative to its current position, not a parent or other ancestor.

transform: translate(45px, -45px);

Transforms and Older Browsers

Transforms require vendor prefixing for some browsers. To make the code work in older browsers include the following:

-webkit-transform: translate(45px,-45px); /* iOS8, Android

↲4.4.3, BB10 */

-ms-transform: translate(45px,-45px); /* IE9 only */

transform: translate(45px,-45px);

If you only want to move an element vertically or horizontally, you can use the **translatex** or **translatey** functions.

transform: translateX(45px);
transform: translate(45px);

Transforms don't worm on inline elements. To fix: add **display: inline-block;**

Scaling

Use the **scale(x,y)** function to scale an element byt the defined factors horizontally then vertically. If only one value is provided, it will be used for both x and y (maintains aspect ratio).

transform: scale(1.5, 0.25);

Can also use **scaleX(x)** and **scaleY(y)** functions.

Element will grow out from the center, or the center will stay in the same place. To change this use the **transform-origin** property.

Example transform property:

transform: translate(40px) scale(1.5);

Note: only one transform property needed with space separated list of transform functions. More transform properties means the last one would override the previous due to cascading.

No impact on document flow. Scaling could cause overlap problems.

Rotation

rotate() function rotates an element around the point of origin by a specified angle value. Default point of origin is center.

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

Skew

skew(x,y) specifies a skew along the x and y axes. If only one parameter, it will only skew on x axis.

transform: skew(15deg, 4deg);

Also **skewX()** and **skewY()** funcntions.

Changing the Origin of the Transform

Use the **transform-origin** property.

transform-origin: 0 0 ;
transform-origin: 10% 10%;
transform-origin: top center;

Some browser support not available. Use same prefixing when using this property.

Note: Choose your ordering carefully

The order of transform functions matter. rotating before translating, your translate direction will be on the rotated axis.

Support for IE 8 and earlier

Transforms unsupported in IE before 9, but you can mimic effects.

Mimic translation: use **position: relative;** and **top** and **left** values:

position: relative;
top: 200px;
left: 200px;

Mimic scale: Alter **width** and **height** or change the **font-size.** This can affect the layout though!

Mimic rotate: Use filters to rotate, but it is ugly and doesn't perform well.

Transitions

Allow values of CSS properties to change over time, providing ability for simple animations. We can animate any of the transforms from above.'

Steps:

1. Declare the original state of the element in the default style declaration.
2. Declare the final state of your transitioned element; for example a **:hover** state.
3. Include the transition function in your default style declaration using the transition properties, including: **transition-property, transition-duration, transition-timing-funciton,** and **transition-delay.**

-webkit- prefix still needed for older browsers/devices.

Transition property

transition-property property defines the CSS properties of the element that should be transitioned, will **all** for all properties being the default.

You can transition anything with a midpoint. Example: 1px red border to 15px blue border. Midpoint border width is 8px. There is a midpoint numeric value for the color between blue and red that the browser handles. There is no midpoint from changing a border style to dashed from solid, so can not transition.

Must have a pre-state and post-state.

You can provide any number of CSS properties to the **transition-property** declaration, separated by commas. Or you can use the keyword **all** to show that every supported property should be animated as it transitions.

transition-property: transform, color;

This will have no effect until we specify the duration of the transition.

Transition-duration Property

transition-duration property sets how long the transition will take, or how long it takes to go from the default state to the transitioned state. Use either seconds **s** or milliseconds **ms**.

transition-duration: 0.2s;

200ms is generally considered the optimum time for a transition: anything slower will make the website seem slow, drawing generally unwanted attention to what was supposed to be a subtle effect. Anything faster may be too subtle.

Transition-timing-function Property

transition-timing-function property lets you control the pace of the transition in more granular detail. Ex. Start slow and become faster

You can use key terms:

ease, linear, ease-in, ease-out, or **ease-in-out.** The default is **ease**.

Can also describe your timing function more precisely by defining your own **cubic-bezier** function. It takes 4 parameters. Linear is the same as (0, 0, 1, 1).

Can also divide the transition over equidistant steps. Use **steps** function to define the number of steps and the direction of either **start** or **end,** where either the first step happens at the animation start, or the that step happens at animation end.

Example: **steps(5, start)** would jump through the equidistant steps of 0%, 20%, 40%, 60%, and 80%, and **steps(5, end)** would jump through the equidistant steps of 20%, 40%, 60%, 80%, and 100%.

transition-timing-function: ease-out;

Transition-delay Property

transition-delay property delays when the transition begins. Default is 0 or immediately. Use **ms** or **s** to specify milliseconds or seconds to delay the transition. 50ms delay is good time to be sure someone is intentionally mousing over something.

-webkit-transition-delay: 50ms;

transition-delay: 50ms;

Negative delays can be used for different effects. Starts immediately, but transition will start in the middle.

Transition Shorthand Property

transition property is shorthand for the four transion properies listed above.

transition: transform 0.2s ease-out 50ms;

Properties can be in any order, but the duration must precede the delay.

Multiple Transitions

Animations

CSS animations, unlike transitions, allow you to control each step of an animation via keyframes.

A **keyframe** is a snapshot that defines a starting or end point of any smooth transition. CSS animation allow us to add any number of keyframes in between to guide our animation in more complex ways.

Keyframes

To create an animation, use the @keyframes rule for IE10+ and FF16+. Include @-webkit-keyframes for all WebKit implementations followed by a name of your choosing, which will serve as the identifier for the animation. Then, you can specify your keyframes.

For an animation called myAnimation, the @keyframes rule would look like this:

Copy

```
@-webkit-keyframes myAnimation {
  /* put animation keyframes here */
}
@keyframes myAnimation {
  /* put animation keyframes here */
}
```

Examples:

```
@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;
   }
}
```

Animation properties

This property is used to attach an animation that was previously defined using @keyframes to an element.

animation-name: appearDisappear;

animation-duration: 300ms;

animation-timing-function determines how the animation will progress. Options same as transition-timing-function.

animation-timing-function: linear;

animation-iteration-count lets you define how many times the animation will play through. Default is 1 or one time. May also use **infinite** for endlessly repeating the animations.

animation-iteration-count: infinite;

animation-direction is used to change the behavior when an animation is repeated. It goes from 0% to 100% and starts back at 0% keyframe normally and is the default. **reverse** will start the animation at 100% keyframe and work to 0% each time. **alternate** will go from 0% to 100% for one iteration, then

100% to 0% the second and keep alternating for the iterations. Also **alternat-reverse** is same as alternate, just starting with reverse.

animation-direction: alternate;

If animations are played in reverse, so will the the timing functions- ease-in becomes ease-out

animation-delay is used to define how many milliseconds or seconds to wait before animation begins.

animation-delay: 50ms;

animation-fill-mode property defines what happens before the first animation iteration begins and after the last animation iteration concludes. Default is **none**. Also available: **forwards, backwards, both.**

animation-fill-mode: forwards;

animation-play-state property deines whether the animation is running or paused.

Shorthand Animation Property

animation: 300ms ease-in alternate 5s backwards appearDisappear;

Be careful when naming animation not to name it the same as a property.

To declare multiple animations:

animation:

animationOne 300ms ease-in backwards,

animationTwo 600ms ease-out 1s forwards;

Use animations to enhance user experience. Just because you can use it doesn't mean you should.

Canvas, SVG, and Drag and Drop

Canvas

HTML5's Canvas API- we can draw anything we can imaging, all through JavaScript. Improves performance by avoiding downloading images off network. We can manipulate pixels in images and video.

Manipulating Video with Canvas

Start by setting up the canvas and context:

```
function makeVideoOldTimey() {
   var video = document.getElementById("video");
   var canvas = document.getElementById("canvasOverlay");
   var context = canvas.getContext("2d");
}
```

Add new event Listener to react to **play** event firing on the video element:

```
function drawOneFrame(video, context, canvas){
   // draw the video onto the canvas
   context.drawImage(video, 0, 0, canvas.width, canvas.height);

   var imageData = context.getImageData(0, 0, canvas.width,
↲canvas.height);
   var pixelData = imageData.data;
   // Loop through the red, green and blue pixels,
   // turning them grayscale

   var red, green, blue, greyscale;
   for (var i = 0; i < pixelData.length; i += 4) {
   red = pixelData[i];
   green = pixelData[i + 1];
   blue = pixelData[i + 2];
   //we'll ignore the alpha value, which is in position i+3

   grayscale = red * 0.3 + green * 0.59 + blue * 0.11;

   pixelData[i] = grayscale;
```

```
    pixelData[i + 1] = grayscale;
    pixelData[i + 2] = grayscale;
    }

    context.putImageData(imageData, 0, 0);
}
```

Displaying Text on the Canvas

Add error handling try/catch block to catch the error:

```
    function drawOneFrame(video, context, canvas){
    context.drawImage(video, 0, 0, canvas.width, canvas.height);

    try {
    var imageData = context.getImageData(0, 0, canvas.width,
↲canvas.height);
    var pixelData = imageData.data;
    for (var i = 0; i < pixelData.length; i += 4) {
        var red = pixelData[i];
        var green = pixelData[i + 1];
        var blue = pixelData[i + 2];
        var grayscale = red * 0.3 + green * 0.59 + blue * 0.11;
        pixelData[i] = grayscale;
        pixelData[i + 1] = grayscale;
        pixelData[i + 2] = grayscale;
    }

    imageData.data = pixelData;
    context.putImageData(imageData, 0, 0);
    } catch (err) {
```

```
    // error handling code will go here
    }
}
```

Reset the width or height of the canvas to clear the canvas:

```
function drawOneFrame(video, context, canvas){
    context.drawImage(video, 0, 0, canvas.width, canvas.height);

    try {
    …
    } catch (err) {
    canvas.width = canvas.width;
    }
}
```

Change the background color from black to transparent because canvas element is on top of the video:

```
        function drawOneFrame(video, context, canvas){
    context.drawImage(video, 0, 0, canvas.width, canvas.height);

    try {
    …
    } catch (err) {
    // clear the canvas
    context.clearRect(0,0,canvas.width,canvas.height);
    canvas.style.backgroundColor = "transparent";
    context.fillStyle = "white";
    }
}
```

Set up the style of the text:

```
function drawOneFrame(video, context, canvas){
   context.drawImage(video, 0, 0, canvas.width, canvas.height);

   try {(review code style)
   …
   } catch (err) {
   // clear the canvas
   context.clearRect(0,0,canvas.width,canvas.height);
   canvas.style.backgroundColor = "transparent";
   context.fillStyle = "white";
   context.textAlign = "left";
   }
}
```

Set the font:

```
function drawOneFrame(video, context, canvas){
   context.drawImage(video, 0, 0, canvas.width, canvas.height);

   try {
   …
   } catch (err) {
   // clear the canvas
   context.clearRect(0,0,canvas.width,canvas.height);
   canvas.style.backgroundColor = "transparent";
   context.fillStyle = "white";
   context.textAlign = "left";

   context.font = "18px LeagueGothic, Tahoma, Geneva, sans-serif";
```

```
    }
}
```

Finally, draw the test using **fillText(“text goes here”, x, y);**

```
function drawOneFrame(video, context, canvas){
    context.drawImage(video, 0, 0, canvas.width, canvas.height);

    try {
    …
    } catch (err) {
    // clear the canvas
    context.clearRect(0,0,canvas.width,canvas.height);
    canvas.style.backgroundColor = "transparent";
    context.fillStyle = "white";
    context.textAlign = "left";
    context.font = "18px LeagueGothic, Tahoma, Geneva, sans-serif";
    context.fillText("There was an error rendering ", 10, 20);
    context.fillText("the video to the canvas.", 10, 40);
    context.fillText("Perhaps you are viewing this page from", 10,
↲70);
    context.fillText("a file on your computer?", 10, 90);
    context.fillText("Try viewing this page online instead.", 10,
↲130);

    return false;
    }
}
```

As a last step, we return false. This lets us check in the draw function whether an exception was thrown. If it was, we want to stop calling drawOneFrame for each video frame, so we exit the draw function:

```
function draw(video, context, canvas) {
  if (video.paused || video.ended) return false;

  drawOneFrame(video, context, canvas);

  // Start over!
  setTimeout(function(){ draw(video, context, canvas); }, 0);
}
```

Downside to Canvas is lack of accessibility. No DOM node, not text-based, so essentially invisible to screen readers.

SVG

Scalable Vector Graphics- allows you to describe vector graphics using XML.

What's XML?

eXtensible Markup Language. Meant to annotate text. XML tags used to describe data.

Drawing in SVG

Draw a circle in SVG:

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

Draw a rectangle in SVG:

```
<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>
```

Good to use image editor like inkscape to make complex images!

Raphael Library

Open source JavaScript Library that makes drawing and animating with SVG easier.

First make a div for a spinner in HTML:

```
<article class="ad-ad4">
  <div id="mapDiv">
  <h1 id="geoHeading">Where in the world are you?</h1>
  <form id="geoForm">
    <input type="button" id="geobutton" value="Tell us!">
  </form>
  <div class="spin" id="spinner"></div>
  </div>
</article>
```

Style dive to be placed in the center of the parent:

```
.spin {
  position: absolute;
  top: 8px;
  left: 55px;
}
```

Put the spinner in place while fetching the map. Turn div into a Raphael container:

```
function determineLocation() {
  if (navigator.onLine) {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(displayOnMap);
```

```
    var container = Raphael(document.getElementById("spinner"),
↲125, 125);
```

Draw spinner SVG into container with Raphael method image:

```
var container = Raphael(document.getElementById("spinner"), 125,
↲125);
var spinner = container.image("images/spinnerBW.svg", 0, 0, 125,
↲125);
```

Rotate the spinner with Raphael animate method. First tell which attribute to animate. Crate object that specifies how many degrees of rotation:

```
var container = Raphael(document.getElementById("spinner"),125,125);
var spinner = container.image("images/spinnerBW.png",0,0,125,125);
var attrsToAnimate = { transform: "r720" };
```

Use animate method to tell how long animation should last:

```
var container = Raphael(document.getElementById("spinner"),125,125);
var spinner = container.image("images/spinnerBW.png",0,0,125,125);
var attrsToAnimate = { transform: "r720" };
spinner.animate(attrsToAnimate, 60000);
```

Use displayOnMap function so that spinner is not on map after it loads:

```
function displayOnMap(position) {
  document.getElementById("spinner").style.display = "none";
}
```

Drag and Drop

Drag and Drop API – Specify if elements are draggable, then specify what should happen when these elements are dragged over or dropped onto other elements

Supported by Android, not iOS.

Add drag and drop to your page:

Set the draggable attribute on any HTML elements you'd like to be draggable.

Add an event listener for the dragstart event on any draggable HTML elements.

Add an event listener for the dragover and drop events on any elements that you want to have accept dropped items.

First, add images to HTML file:

```
<article id="ac3">
  <h1>Wai-Aria? HAHA!</h1>
  <h2>Form Accessibility</h2>

  <img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/cat.png" alt="WAI-ARIA Cat">

  <div class="content">
  <p id="mouseContainer" class="mc">
    <img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg"
↵alt="mouse treat" id="mouse1" draggable="true">
    <img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg"
↵alt="mouse treat" id="mouse2" draggable="true">
    <img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg"
↵alt="mouse treat" id="mouse3" draggable="true">
  </p>
…
```

Next, make elements draggable:

<img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg" width="30"

↲alt="mouse treat" id="mouse1" draggable="true">

<img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg" width="30"

↲alt="mouse treat" id="mouse2" draggable="true">

<img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/computer-mouse-pic.svg" width="30"

↲alt="mouse treat" id="mouse3" draggable="true">

*draggable is no Boolean! We have to explicitly set it to true.

Set an event listener for the dragstart event on each image:

```
var mice = document.querySelectorAll("#mouseContainer img");
var mouse = null;
for (var i=0; i < mice.length; i++) {
   mouse = mice[i];
   mouse.addEventListener('dragstart', function (event) {
   // handle the dragstart event
   });
}
```

Data Transfer Object

Allow us to set and get data about the elements that are being dragged. Two pieces defined:

the type of data we're saving of the draggable element and the value of the data itself.

Save some plain text by passing in the string text/plain. Then give ID of mouse image:

```
mouse.addEventListener("dragstart", function (event) {
   event.dataTransfer.setData("text/plain", this.id);
});
```

by default, elements on the page aren't set up to receive dragged items. In order to override the default behavior on a specific element, we must stop it from happening. We can do that by creating two more event listeners.

The two events we need to monitor for are dragover and drop. As you'd expect, dragover fires when you drag an item over an element, and drop fires when you drop an item on it.

```
<article id="ac3">
  <h1>Wai-Aria? HAHA!</h1>
  <h2 id="catHeading">Form Accessibility</h2>

  <img data-src="https://learnable-static.s3.amazonaws.com/premium/reeedr/books/html5-css3-for-the-real-world-2nd-edition/images/cat.png" id="cat" alt="WAI-ARIA Cat">
```

Now let's handle the dragover event:

```
var cat = document.getElementById("cat");
cat.addEventListener("dragover", function(event) {
  event.preventDefault();
});
```

store these options inside an object called mouseHash, where the first step is to declare our object:

```
cat.addEventListener("drop", function(event) {
  var mouseHash = {};
```

Next, we're going to take advantage of JavaScript's objects allowing us to store key/value pairs inside them, as well as storing each response in the mouseHash object, associating each response with the ID of one of the mouse images:

```
cat.addEventListener("drop", function(event) {
  var mouseHash = {
  mouse1: 'NOMNOMNOM',
  mouse2: 'Meow',
  mouse3: 'Purrrrrr ...'
  };
}
```

Our next step is to grab the h2 element that we'll change to reflect the cat's response:

```
var catHeading = document.getElementById('catHeading');
```

Remember when we saved the ID of the dragged element to the DataTransfer object using setData? Well, now we want to retrieve that ID. If you guessed that we'll need a method called getData for this, you guessed right:

```
var mouseID = event.originalEvent.dataTransfer.getData("text/plain");
```

we just need to change the text to the appropriate response:

```
catHeading.innerHTML = mouseHash[mouseID];
```

Remove mouse from page:

```
var mousey = document.getElementById(item);

mousey.parentNode.removeChild(mousey);
```

Prevent default behavior of not allowing elements to be dropped on our cat image:

```
event.preventDefault();
```