CSC 470 – Section 3

Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 7

Scalable Vector Graphics (SVG)

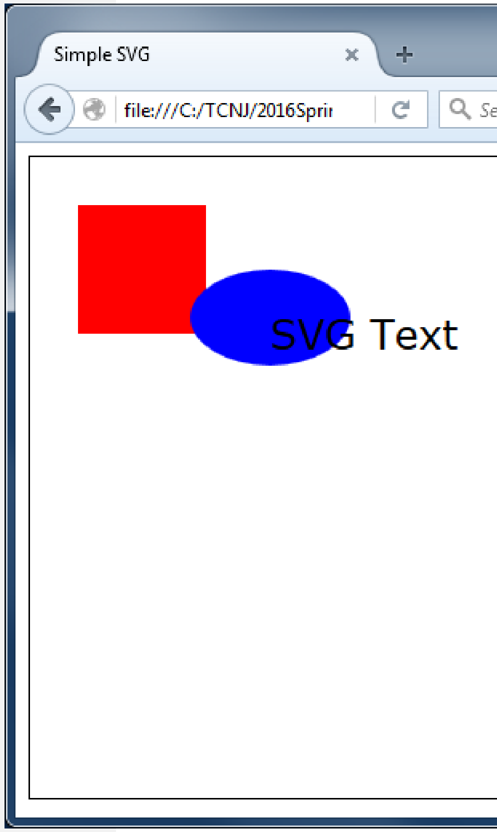
Scalable Vector Graphics

- A standard markup language for describing two-dimensional vector graphics
- SVG provides special elements for circles, rectangles, and simple and complex curves.
- SVG graphics are **scalable and resolution-independent**.
- Support features like masking, clipping, patterns, full gradients, groups, and more.
- SVG elements may be styled with CSS and manipulated with JavaScript
- A simple SVG document consists of an `<svg>` root element containing and several graphic elements that build a visualization.
- Applications exist for creating graphics that may be saved in an SVG format, and opened in a browser
  - See InkScape (https://inkscape.org/en/)
- May be animated. Animations execute concurrently.

https://developer.mozilla.org/en-US/docs/Web/SVG

Sample SVG in an HTML Document

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Simple SVG</title>
  </head>
  <body>
    <svg width="500" height="400" style="border: 1px solid;">

      <!-- a red rectangle -->
      <rect x="30" y="30" width="80" height="80" fill="red" />

      <!-- a blue ellipse -->
      <ellipse cx="150" cy="100" rx="50" ry="30" fill="blue" />

      <!-- text with given font and size -->
      <text x="150" y="120" font-family="Verdana" font-size="25px">
        SVG Text
      </text>

    </svg>
  </body>
</html>
```



07/example1.html

`<svg>` Element

- Contains all SVG graphics and other SVG elements
- Used to embed an SVG diagram into an HTML document
- Establishes a canvas that is effectively infinite in all dimensions
- viewport is the window through which the graphic is viewed – limits view

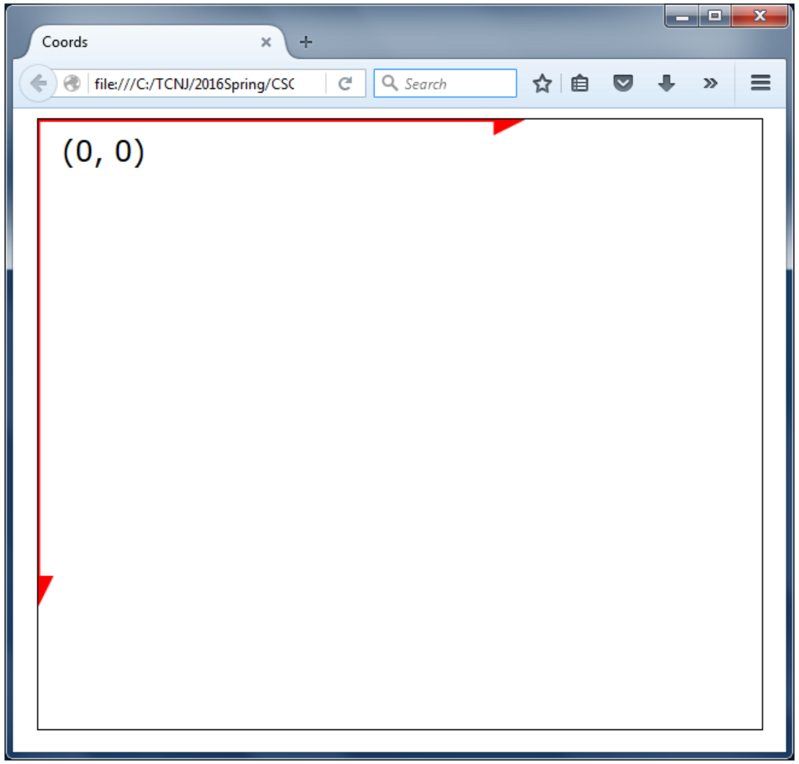
Important Attributes:

- `width = "…"`
  - width of the rectangular viewport
- `height = "…"`
  - Height of the rectangular viewport
- `viewBox = "min-x, min-y, width, height"`
  - Different than the viewport
  - Rectangle in user space mapped to the viewport established by the <svg> element

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/svg

SVG Coordinate System

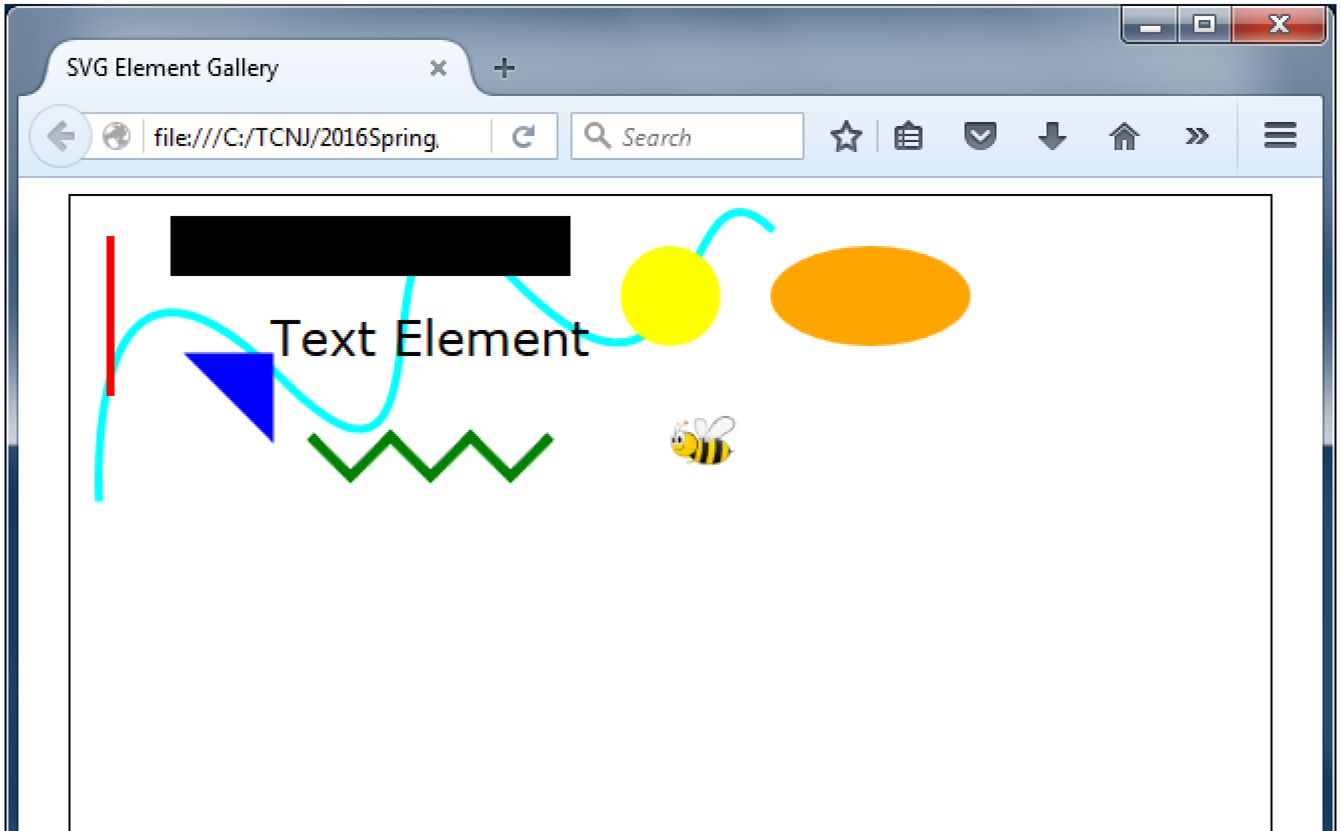
- Two arrows rendered with starting point at (0, 0)
- Both arrows point in the positive direction (numerically)
- Origin of coordinate system is at upper-left corner of viewport
  - X increases left to right
  - Y increases top to bottom



Two arrows:
(0, 0) to (0, 400)
(0, 0) to (400, 0)

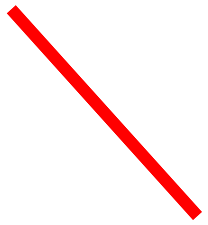
SVG Element Examples

- SVG provides multiple graphic elements
- Examples include:
  - `<rect>, <circle>, <ellipse>, <text>, <image>, <line>, <polyline>, <polygon>, <path>, …`



07/gallery.html

`<line … />`



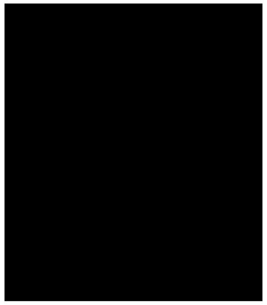
- Straight line shape
- Attributes

`x1`	: x-coordinate of start point
`y1`	: y-coordinate of start point
`x2`	: x-coordinate of end point
`y2`	: y-coordinate of end point
`stroke`	: color of line
`stroke-width`	: width of line
…	

```
<line x1="20" y1="20" x2="20" y2="100" stroke="red" stroke-width="4px" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/line

`<rect … />`



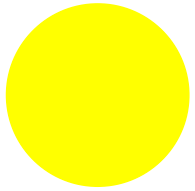
- Rectangle shape
- Attributes

`x`	: x-coordinate of upper-left corner
`y`	: y-coordinate of upper left corner
`width`	: width of shape
`height`	: height of shape
`stroke`	: color of outline
`stroke-width`	: width of outline
`fill`	: fill color
`rx`	: x-radius on corners, if any
`ry`	: y-radius on corners, if any
…	

```
<rect x="50" y="10" width="200" height="30" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/rect

`<circle … />`



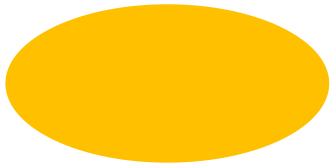
- Circle shape
- Attributes

`cx`	: x-coordinate of shape center
`cy`	: y-coordinate of shape center
`r`	: radius of shape
`stroke`	: color of outline
`stroke-width`	: width of outline
`fill`	: fill color
…	

```
<circle cx="300" cy="50" r="25" fill="yellow" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/circle

`<ellipse … />`



- Ellipse shape
- Attributes

`cx`	: x-coordinate of shape center
`cy`	: y-coordinate of shape center
`rx`	: radius of shape in x-direction
`ry`	: radius of shape in y-direction
`stroke`	: color of outline
`stroke-width`	: width of outline
`fill`	: fill color
…	

```
<ellipse cx="400" cy="50" rx="50" ry="25" fill="orange" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/ellipse

`<text … >…</text>`

ABC

- Text element
- Attributes

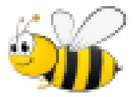
`x`	: x-coordinate of lower-left corner
`y`	: y-coordinate of lower-left corner
`font-family`	: Font to use to render text
`font-size`	: Size of font
`stroke`	: color of outline
`stroke-width`	: width of outline
`fill`	: fill color
…	

- Note: Text to be rendered is placed between start and end tags, not an attribute

```
<text x="100" y="80" font-family="Verdana" font-size="24px">ABC</text>
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/text

<image … />



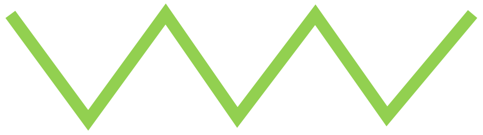
- Image Shape
- Attributes

`x`	: x-coordinate of upper-left corner
`y`	: y-coordinate of upper left corner
`width`	: width of shape
`height`	: height of shape
`xlink:href`	: URL/path to image
…	

```
<image xlink:href="bee.png" x="300" y="100" height="45" width="33" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/image

`<polyline … />`



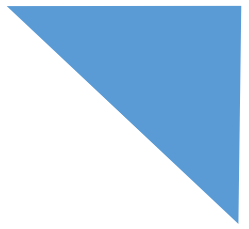
- Polyline Shape
- Attributes

`points`	: list of space-separated x,y coordinate pairs
`stroke`	: color of outline
`stroke-width`	: width of outline
`fill`	: fill color
…	

```
<polyline points="120,120 140,140 160,120 180,140 200,120 220,140 240,120"
          stroke="green" stroke-width="5px" fill="none"/>
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/polyline

`<polygon … />`



- Polygon Shape (Closed)
- Attributes

• `points`	: list of space-separated x,y coordinate pairs
• `stroke`	: color of outline
• `stroke-width`	: width of outline
• `fill`	: fill color
• …	

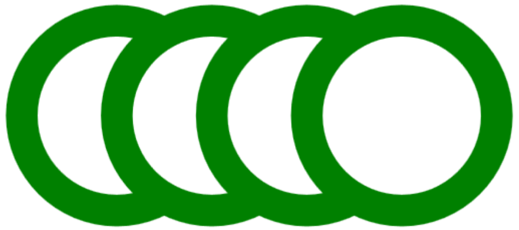
```
<polygon points="100,80 60,80 100,120" stroke="blue" stroke-width="3px"
         fill="blue" />
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/polygon

<g> … </g>

- A container used to group other SVG elements allowing them to be manipulated together
- Attribute values of a <g> element are applied to all contained elements

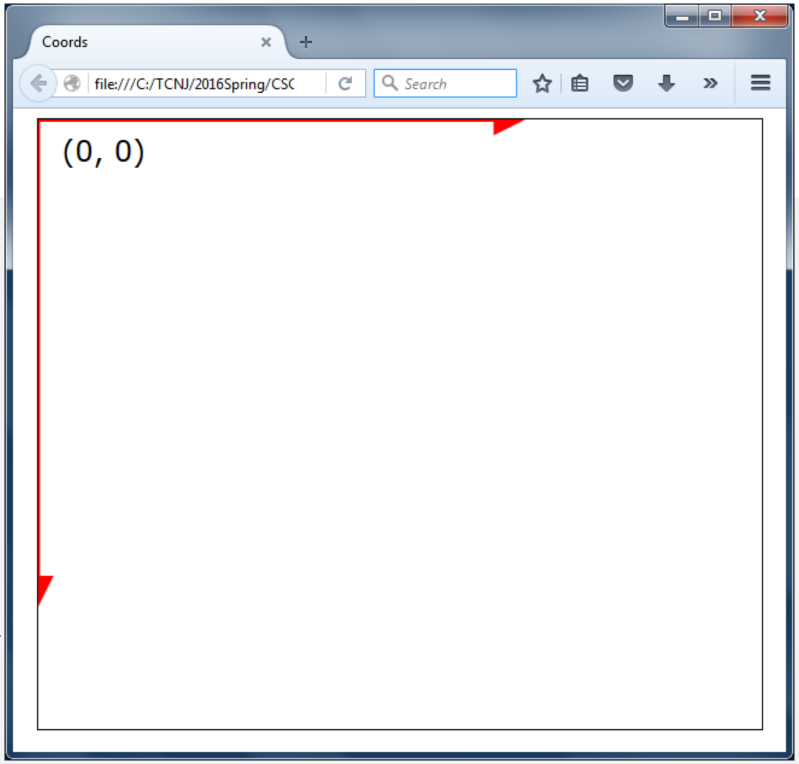
```
<svg viewBox="0 0 95 50">
   <g stroke="green" fill="white" stroke-width="5">
     <circle cx="25" cy="25" r="15"/>
     <circle cx="40" cy="25" r="15"/>
     <circle cx="55" cy="25" r="15"/>
     <circle cx="70" cy="25" r="15"/>
   </g>
</svg>
```



https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g

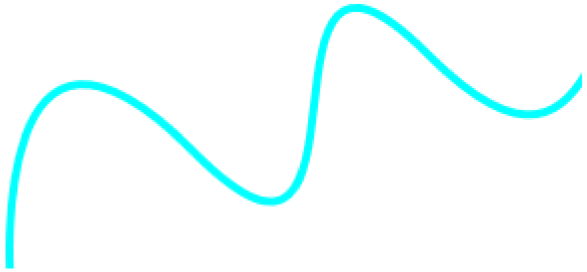
Example

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Coords</title>
    <style>
      #div1 {
        border: 1px solid black;
        width: 600px;
        margin: 0 auto;   /* center content horizont
      }
    </style>
  </head>
  <body>
    <div id="div1">
      <svg id="cvs" width="600" height="500">
        <line x1="0" y1="0" x2="0" y2="400" stroke="red" stroke-width="4" />
        <polygon points="400,0 380,-10 380,10" stroke="red" stroke-width="4" fill="red" />
        <line x1="0" y1="0" x2="400" y2="0" stroke="red" stroke-width="4" />
        <polygon points="0,400 -10,380 10,380" stroke="red" stroke-width="4" fill="red" />
        <text x="20" y="35" font-family="Verdana" font-size="24">(0, 0)</text>
      </svg>
    </div>
  </body>
</html>
```



07/coords.html

`<path … />`



- Generic shape defined by a path expression made up of lines and curves
- Attributes
  - `d` : path expression string defining a path
  - `stroke` : color of outline
  - `stroke-width` : width of outline
  - `fill` : fill color
  - …

```
<!doctype html>
<html>
  <body>
    <svg>
      <path d="m  14.29,152.36
               s -10.0,-162.41 91.68,-60.74 19.38,-146.08 116.89,-48.57
               c  97.51, 97.51 78.20,-76.08 128.57,-25.71"
               fill="none" stroke="cyan" stroke-width="4px" />
    </svg>
  </body>
</html>
```

https://developer.mozilla.org/en-US/docs/Web/SVG/Element/path

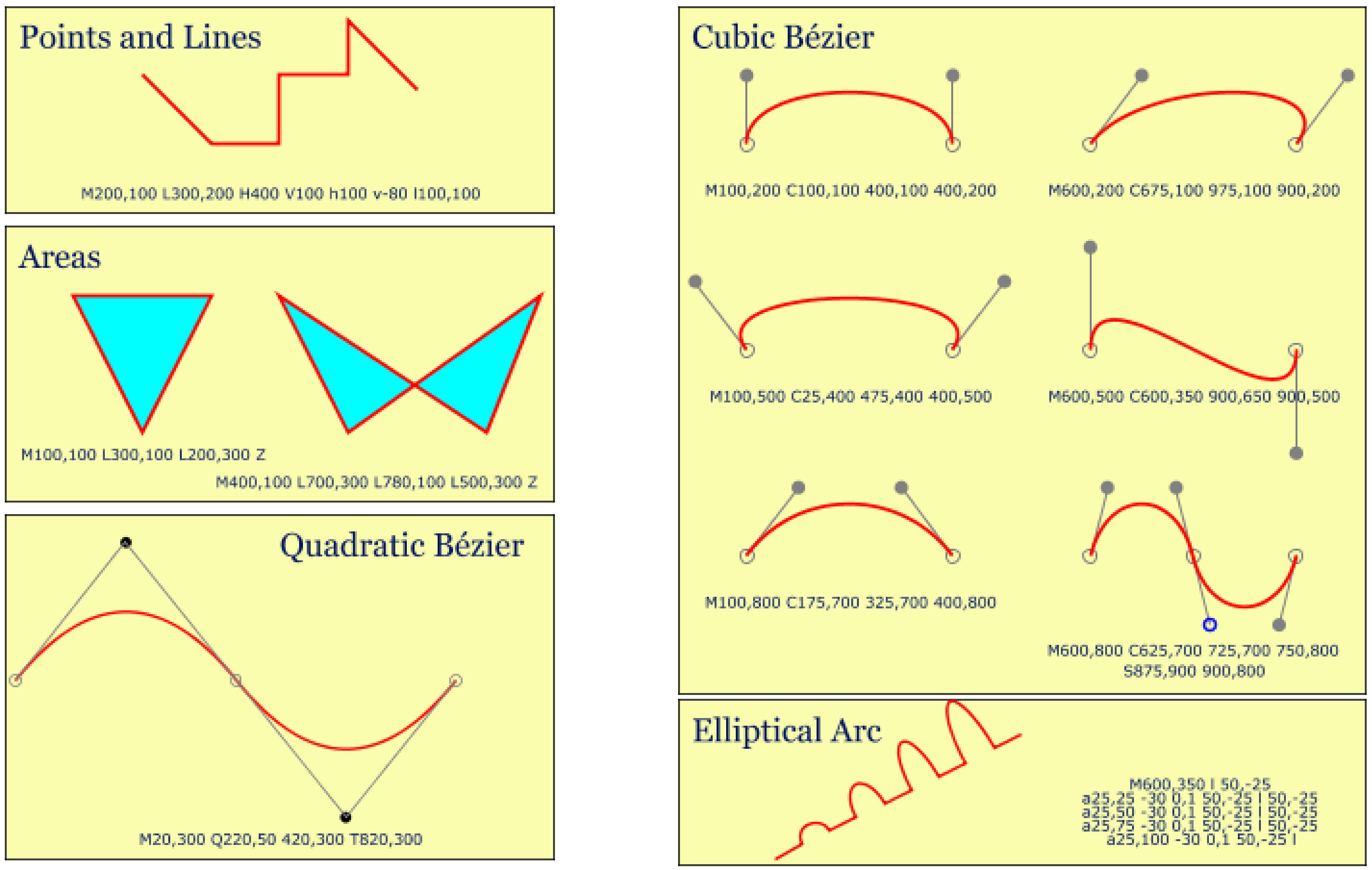
07/path1.html

Path Expressions (attribute=`d`)

- A string that defines a continuous path made up of segments.
- Each segment def'n starts with a single character identifying its type followed by one or more coordinates `(x,y)` that parameterize the segment
- Coordinates parameterizing a segment may be absolute or relative to the previous segment
  - Upper case letters precede absolute coordinates, lower case letters precede relative coordinates

`M/m`	move to an absolute/relative point
`L/l`	add a line to an absolute/relative point
`H/h`	add horizontal line to an absolute/relative point
`V/v`	add a vertical line to an absolute/relative point
`C/c`	add a curve to an absolute/relative point
`S/s`	add a smooth curve to an absolute/relative point
`Q/q`	add a quadratic Bézier curve
`T/t`	add a smooth quadratic Bézier curve to an absolute/relative point
`A/a`	add an elliptical Arc to an absolute/relative point
`Z/z`	close the path

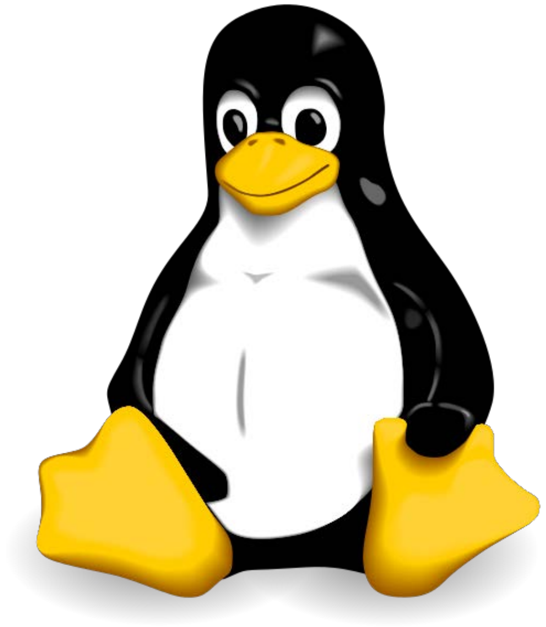
Path Examples



```
M200,100 L300,200 H400 V100 h100 v-80 l100,100

M100,100 L300,100 L200,300 Z
M400,100 L700,300 L780,100 L500,300 Z

M20,300 Q220,50 420,300 T820,300

M100,200 C100,100 400,100 400,200
M600,200 C675,100 975,100 900,200
M100,500 C25,400 475,400 400,500
M600,500 C600,350 900,650 900,500
M100,800 C175,700 325,700 400,800
M600,800 C625,700 725,700 750,800
S875,900 900,800

M600,350 l 50,-25
a25,25 -30 0,1 50,-25 l 50,-25
a25,50 -30 0,1 50,-25 l 50,-25
a25,75 -30 0,1 50,-25 l 50,-25
a25,100 -30 0,1 50,-25 l
```

https://www.w3.org/2002/Talks/SVG-HongKong-IH/7.html

Path Examples



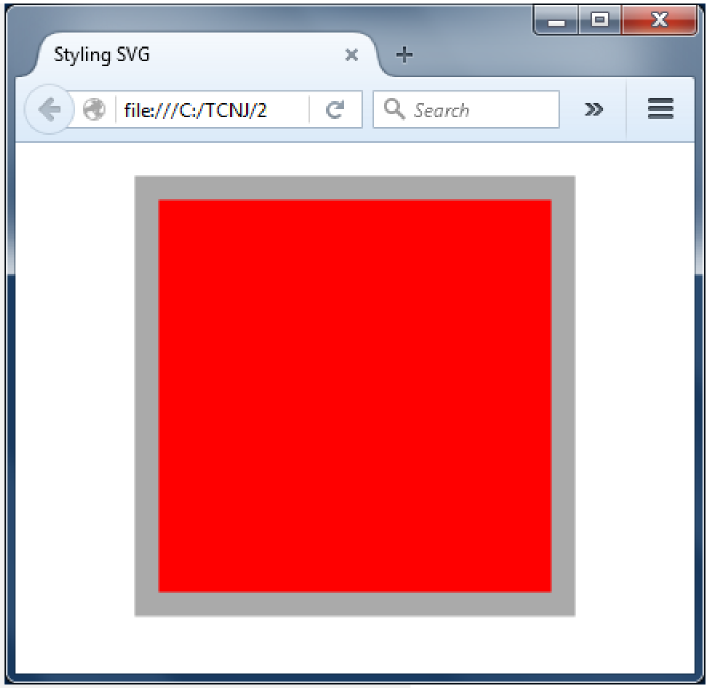
```
<path fill="#666"    d="m148.47 94.049c4.319-1.728 3.592-1.958 6.472
<path fill="#6d6d6d" d="m148.47 94.023c4.293-1.717 3.563-1.954 6.425
<path fill="#757575" d="m148.471 93.996c4.264-1.706 3.533-1.95 6.377
<path fill="#7c7c7c" d="m148.471 93.969c4.235-1.694 3.506-1.946 6.32
<path fill="#848484" d="m148.471 93.943c4.209-1.684 3.477-1.942 6.28
<path fill="#8c8c8c" d="m148.471 93.916c4.181-1.672 3.448-1.938 6.23
<path fill="#939393" d="m148.472 93.889c4.152-1.661 3.419-1.934 6.18
<path fill="#9b9b9b" d="m148.472 93.863c4.125-1.65 3.391-1.93 6.141-
<path fill="#a3a3a3" d="m148.472 93.836c4.097-1.639 3.361-1.926 6.09
<path fill="#aaa"    d="m148.472 93.809c4.069-1.628 3.334-1.922 6.04
<path fill="#b2b2b2" d="m148.473 93.782c4.041-1.617 3.304-1.918 5.99
<path fill="#bababa" d="m148.473 93.756c4.014-1.606 3.275-1.914 5.95
<path fill="#c1c1c1" d="m148.473 93.729c3.985-1.595 3.247-1.91 5.904
<path fill="#c9c9c9" d="m148.473 93.702c3.958-1.583 3.219-1.906 5.85
<path fill="#d1d1d1" d="m148.474 93.676c3.93-1.573 3.188-1.902 5.809
<path fill="#d8d8d8" d="m148.474 93.649c3.901-1.562 3.16-1.898 5.762
<path fill="#e0e0e0" d="m148.474 93.622c3.875-1.55 3.132-1.894 5.715
<path fill="#e8e8e8" d="m148.474 93.596c3.847-1.54 3.104-1.89 5.668-
<path fill="#efefef" d="m148.475 93.569c3.817-1.528 3.073-1.886 5.62
<path fill="#f7f7f7" d="m148.475 93.542c3.791-1.517 3.046-1.882 5.57
<path fill="#fff"    d="m148.475 93.516c3.763-1.506 3.017-1.878 5.52
```

https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg

Styling Elements with CSS

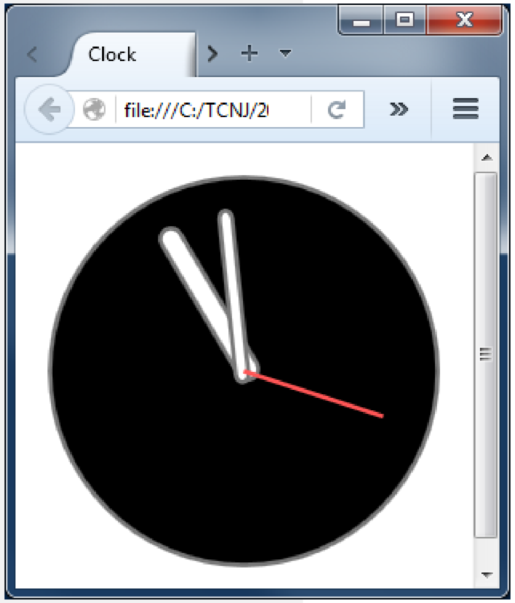
- CSS can be used to style SVG elements much like it is used with HTML
- Use SVG attributes as style keys and SVG values as style values

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Styling SVG</title>
    <style type="text/css">
      div {
        width : 300px;
        margin: 0 auto;
      }
      #r1 {
        fill         : red;
        stroke       : #A0A0A0;
        stroke-width: 15px;
      }
    </style>
  </head>
  <body>
    <div>
      <svg width='300' height='300'>
        <rect id='r1' x='20' y='20' width='260' height='260' />
      </svg>
    </div>
  </body>
</html>
```



SVG Clock with Timer

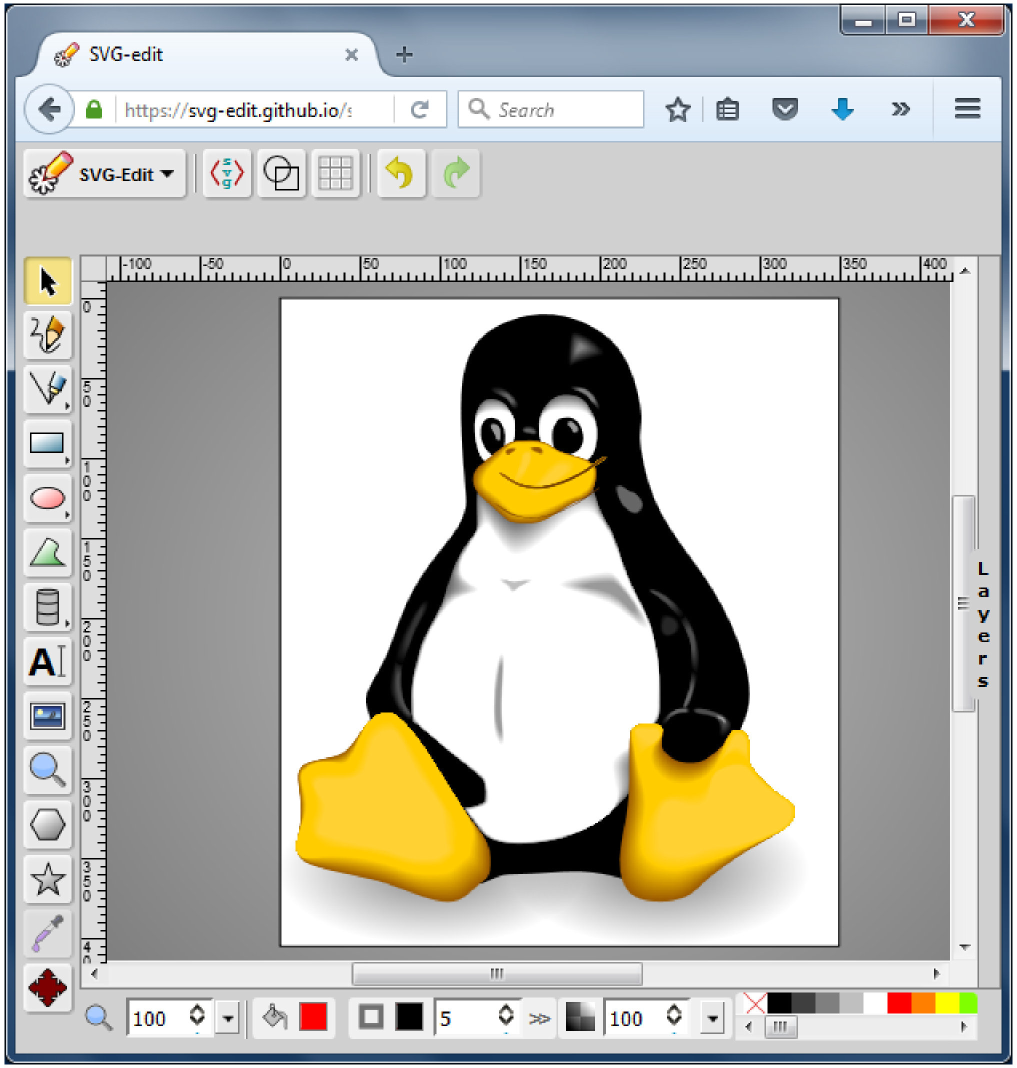
```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Clock</title>

    <style>
      #face { stroke-width: 1px; stroke: #777; }
      #hour, #min, #sec { stroke-width: 1px; fill: #fff; stroke: #777;}
      #sec { stroke: #f55; }
    </style>

    <script type="text/javascript">
      setInterval(function() {
        function r(el, deg) {
          el.setAttribute('transform', 'rotate('+ deg +' 50 50)')
        }
        var d = new Date()
        r(sec,  6*d.getSeconds())
        r(min,  6*d.getMinutes())
        r(hour, 30*(d.getHours()%12) + d.getMinutes()/2)
      }, 1000);
    </script>

  </head>
  <body>
    <svg id="clock" viewBox="0 0 100 100">
      <circle id="face" cx="50" cy="50" r="45"/>
      <g id="hands">
        <rect id="hour" x="48.5" y="12.5" width="5" height="40" rx="2.5" ry="2.55" />
        <rect id="min"  x="48"   y="12.5" width="3" height="40" rx="2"   ry="2"/>
        <line id="sec"  x1="50"  y1="50"  x2="50"  y2="16" />
      </g>
    </svg>
  </body>
</html>
```



http://thenewcode.com/943/An-SVG-Analog-Clock-In-6-Lines-of-JavaScript

07/clock.html

svgedit – An Online SVG Editor



https://github.com/SVG-Edit/svgedit

Snap.svg

- A JavaScript library for working with SVG in a browser
- API for animating and manipulating both existing SVG content, and SVG content generated with Snap
- Created by Dmitry Baranovskiy
- Wraps the SVG browser API into an intuitive JavaScript library

- To use, download the library at http://snapsvg.io
- Load the main Snap.svg JavaScript library in a `<script>` tag.

```
<script type="text/javascript" src="snap.svg.js"></script>
```

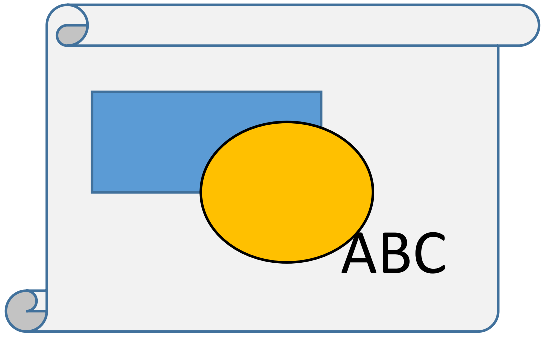
Snap.svg Objects

- `Snap`
  - The top-level object
  - Used to create a new drawing surface or wrap an existing `<svg>`
  - In addition, holds many shared functions as properties
- `Paper`
  - Drawing surface object encapsulation `(<svg>` element)
  - Holds all functions to create shapes and other related elements
  - May also be used to created nested `<svg>` elements
- `Element`
  - Object that encapsulates SVG shape elements
  - Exposes numerous functions to manage a shape element and all properties
- `Matrix`
  - Matrix object implementing standard transformation operations
- `Set`
  - Data structure used to hold and manage multiple shape elements
  - May be used to iterate over contents and animate all as a group
- `mina`
  - Defines and manages all animations, especially easing models

Paper – The Snap Drawing Surface

- Use the `Snap` object to create a new drawing surface or to wrap an existing `<svg>` element
- Produces an object of type `Paper`
- The `Paper` object can then be used to create and add new Shape elements and manage them

```
// Create a new Paper object that is 300 pixels wide and 200 pixels high
var cvs1 = Snap(300, 200);

// Create a new Paper object by wrapping <svg id='cvs'></svg>
var cvs2 = Snap('#cvs');
```



Creating Shape Elements

- `aPaper.rect(…)`
- `aPaper.circle(…)`
- `aPaper.text(…)`
- `aPaper.image(…)`
- `aPaper.ellipse(…)`
- `aPaper.path(…)`
- `aPaper.line(…)`
- `aPaper.polyline(…)`
- `aPaper.polygon(…)`

- Use methods of a Paper object to add new shape elements
- Arguments closely follow SVG element attributes
- See http://snapsvg.io/docs/

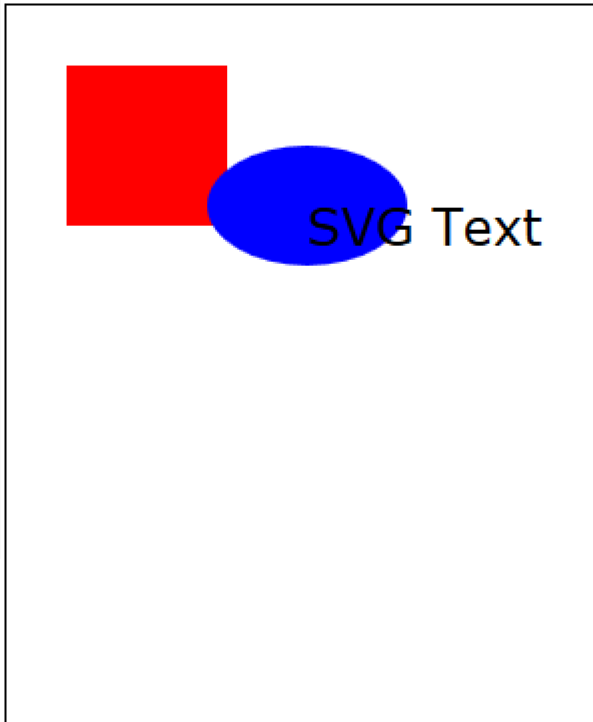
- `aPaper.g(…)`
- `aPaper.clear()`

- Group element for grouping shapes
- Remove all child elements from the Paper object

Shape Element Attributes

- Use the **`attr(…)`** method of the Shape Element to get or set properties
- Provide an object argument with key/value pairs defining all attribute values to set
- Provide a string argument alone with the name of the attribute value to get a value

```
// snap1.js
// Create a new drawing surface based on an existing <svg>
element
var cvs = Snap("#cvs");

// Create shape elements and set attributes
var rect1 = cvs.rect(30, 30, 80, 80);
rect1.attr({fill: 'red'});

var ellipse1 = cvs.ellipse(150, 100, 50, 30);
ellipse1.attr({fill: 'blue'});

var text1 = cvs.text(150, 120, 'SVG Text');
text1.attr({'font-family': 'Verdana', 'font-size': '25px'});
```



Manipulating SVG Colors

Snap.svg provides functions for manipulating color strings

- `Snap.rgb(r,g,b)`
  - Converts RGB values to a hex representation of the color
- `Snap.color(clr)`
  - Parses the color string and returns an object featuring the color's component values
- `Snap.getRGB(color)`
  - Parses color string as RGB object

- And many more functions dealing with other color models, such as HSB, HSL and conversion between models…

```
Snap.rgb(255,127,64)    // "#ff7f40"
```

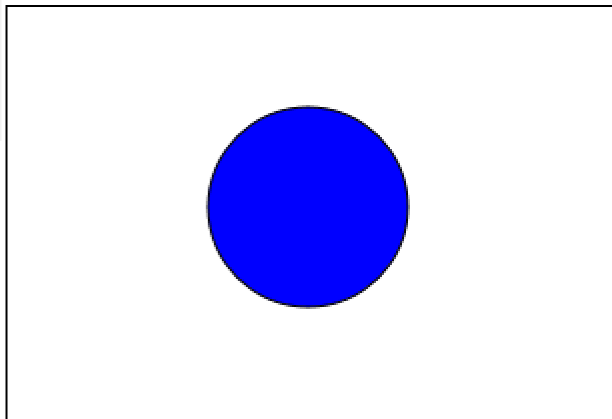
SVG Events

- Snap.svg simplifies the process of attaching and detaching event handler functions to SVG elements
- To attach an event handler to an SVG Element Event, invoke the appropriate method
- To detach an event handler, use the 'un' version of the method
- Includes touch events for mobile devices
- '`this`' in event function context refers to Snap Element

```
anElem.click()        anElem.unclick()         // mouse click
anElem.dblclick()     anElem.undblclick()      // mouse double-click
anElem.mousedown()    anElem.unmousedown()     // mouse button pressed down
anElem.mouseup()      anElem.unmouseup()       // mouse button released
anElem.mousemove()    anElem.unmousemove()     // mouse moved
anElem.mouseover()    anElem.unmouseover()     // mouse moved on element
anElem.mouseout()     anElem.unmouseout()      // mouse moved off element
anElem.hover()        anElem.unhover()         // mouseover and mouseout
anElem.drag()         anElem.undrag()          // element is dragged
anElem.touchstart()   anElem.untouchstart()    // touch started
anElem.touchmove()    anElem.untouchmove()     // touch moved
anElem.touchend()     anElem.untouchend()      // touch ended
anElem.touchcancel()  anElem.untouchcancel()   // touch cancelled
```

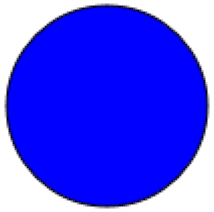
Events Example

```
// events1.js

// Drawing surface
var cvs = Snap("#cvs");

// Create ellipse
var elp = cvs.ellipse(150, 100, 50, 50);
elp.attr({fill: 'blue', stroke: 'black'});

// Attach event to ellipse shape element
elp.click( function(e, x, y) {
  // Compute random color and set to fill color
  var r = Math.floor(Math.random() * 256);
  var g = Math.floor(Math.random() * 256);
  var b = Math.floor(Math.random() * 256);
  var clr = Snap.rgb(r,g,b);
  elp.attr({fill:clr});
});
```



07/events1.html

Events Example



Multiple events handled

```
// events2.js
// Drawing surface
var cvs = Snap("#cvs");

// Create ellipse
var elp = cvs.ellipse(150, 100, 50, 50);
elp.attr({fill: 'blue', stroke: 'black'});

// Attach event to ellipse shape element
elp.click(     function(e) { console.log('click');      } );
elp.dblclick(  function(e) { console.log('dblclick');   } );
elp.mousedown( function(e) { console.log('mousedown');  } );
elp.mouseup(   function(e) { console.log('mouseup');    } );
elp.hover(     function(e) { console.log('hover in');   },
               function(e) { console.log('hover out');  } );
elp.mouseover( function(e) { console.log('mouseover');  } );
elp.mouseout(  function(e) { console.log('mouseout');   } );
elp.mousemove( function(e) { console.log('mousemove');  } );
elp.drag(      function(e) { console.log('drag move');  },
               function(e) { console.log('drag start');},
               function(e) { console.log('drag end');   } );
```

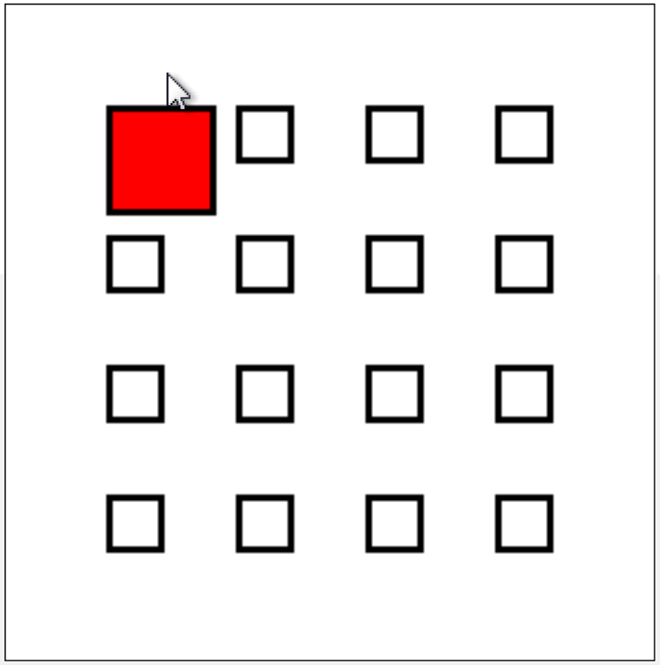
07/events2.html

Animation

- `anElement.animate(attrs, duration, [easing], [callback])`
  - Animates the given attributes of the element
  - `attrs` : object with target values to achieve as a result of the animation
  - `duration` : the number of milliseconds to animate
  - `easing` : mina function defining the move profile
  - `callback` : function to call when animation is complete

- `anElement.stop()`
  - Stops all in-progress animations for the current element

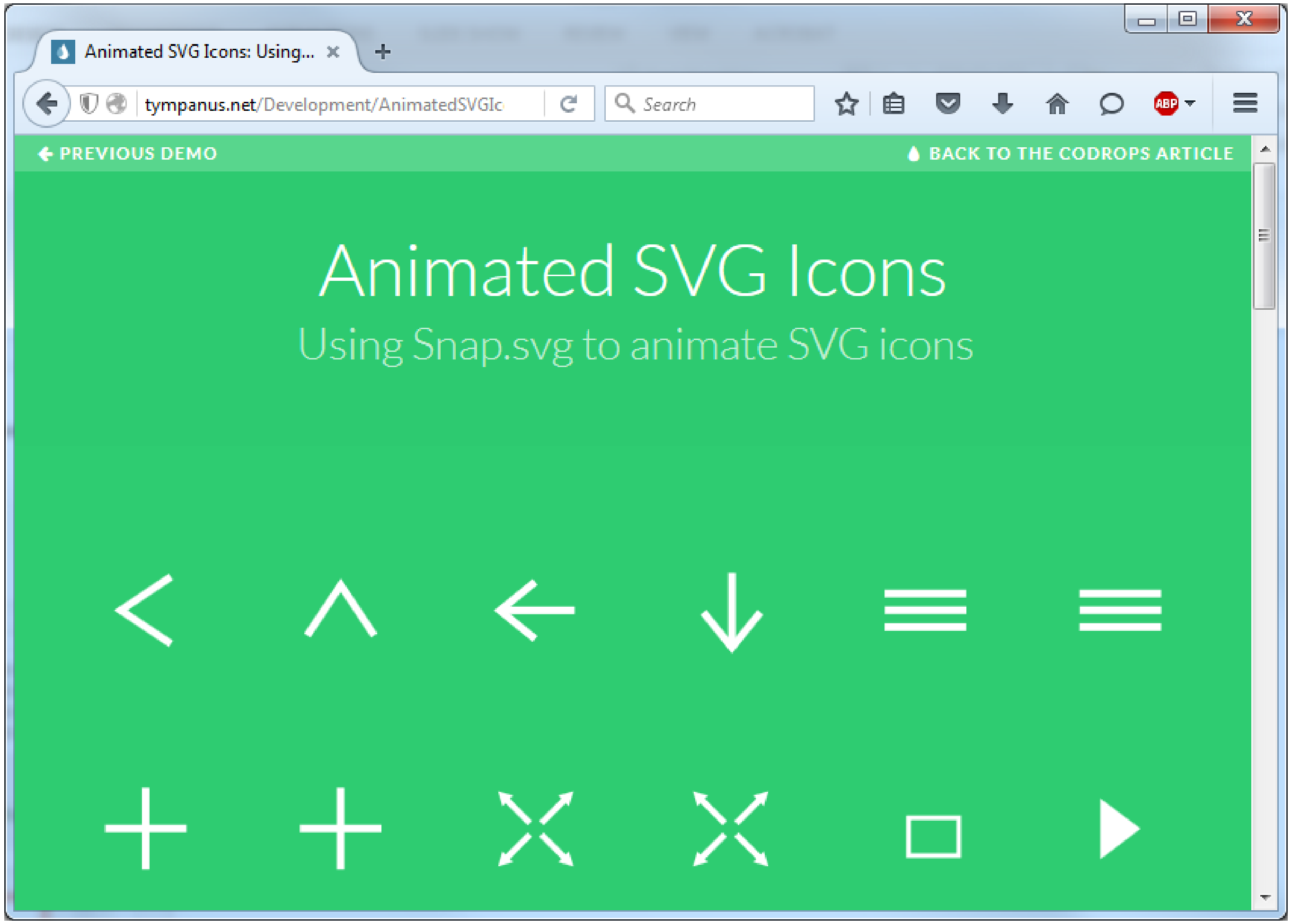
Animation Example



```
// animate1.js
// Drawing surface
var cvs = Snap("#cvs");

// Create ellipses
for (var x = 100; x <= 400; x += 100) {
  for (var y = 100; y <= 400; y += 100) {

    // Create and style ellipse
    var el = cvs.rect(x-40, y-40, 80, 80);
    el.attr({fill: 'white', stroke: 'black', strokeWidth: '5px'});

    // Add events
    el.hover(
      function(e) { this.animate({width:'80', height:'80', fill:'red' }, 100); },
      function(e) { this.animate({width:'40', height:'40', fill:'white' }, 100); }
    );
  }
}
```

07/animate1.html

Animation Examples


Animated SVG Icons: Using...
tympanus.net/Development/AnimatedSVGIc
Search
PREVIOUS DEMO
BACK TO THE CODROPS ARTICLE
Animated SVG Icons
Using Snap.svg to animate SVG icons

http://tympanus.net/Development/AnimatedSVGIcons/
http://tympanus.net/Development/IconHoverEffects/
http://tympanus.net/Tutorials/ShapeHoverEffectSVG/

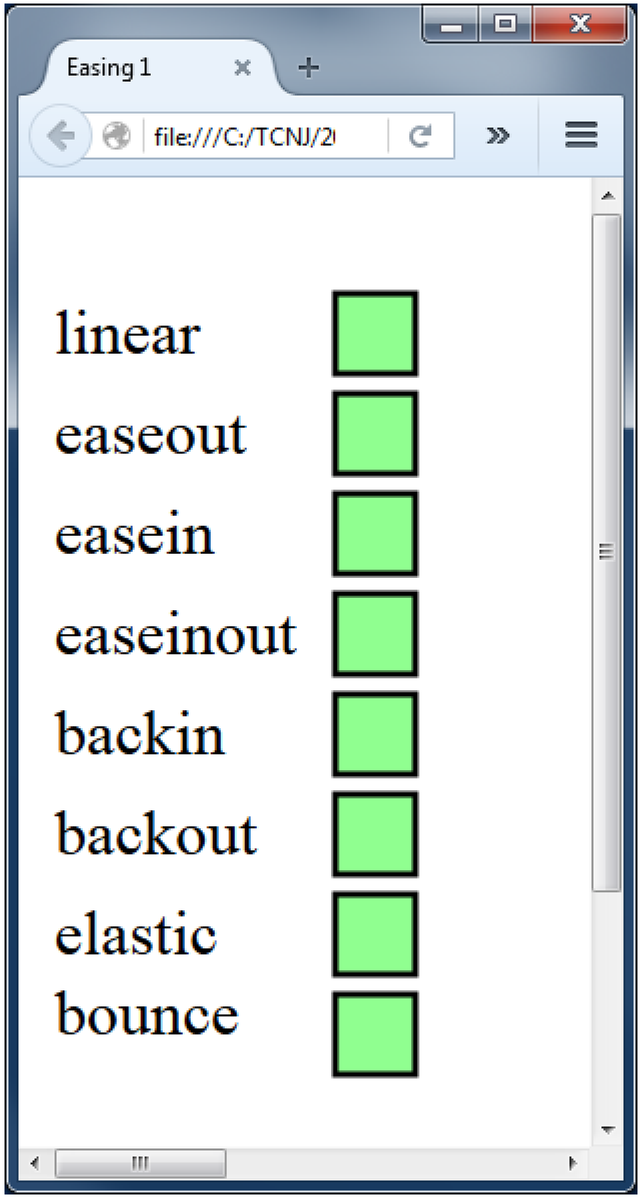
Easing

Determines profile of how attribute values are modified over time as an animation transitions from a initial value to a final value

- `mina.linear`
  - Uniform value change throughout animation
- `mina.easeout`
  - Slow down as approaching final value
- `mina.easein`
  - Slowly accelerate to final value
- `mina.easeinout`
  - Combination of easein and easeout
- `mina.backin`
- `mina.backout`
- `mina.elastic`
- `mina.bounce`

Easing Profiles

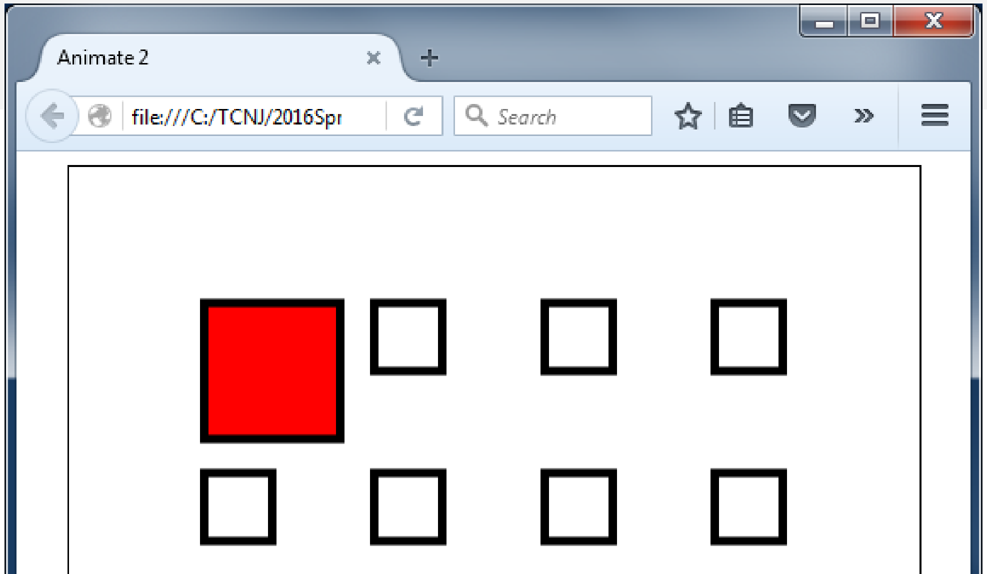
```
// easing1.js

// Drawing surface
var cvs = Snap("#cvs");

// Parameters
var fontSize = '24pt';
var startX   = 150;
var endX     = 450;
var duration = 750;

// Reset all rects
cvs.click( function(e, x, y) {
  cvs.selectAll('rect').attr({x:startX});
});

// Create and style ellipses
// Add easing animation on click event

// linear
var b1 = cvs.rect(startX,  50, 40, 40);
b1.attr({fill: '#90FF90', stroke: 'black', strokeWidth: '3px'});
b1.click( function(e, x, y) {
  b1.animate({x:endX}, duration, mina.linear);
  e.stopPropagation();
});
cvs.text(10, 80, 'linear').attr({fontSize: fontSize});

// easeout
var b2 = cvs.rect(startX, 100, 40, 40);
b2.attr({fill: '#90FF90', stroke: 'black', strokeWidth: '3px'});
b2.click( function(e, x, y) {
  b2.animate({x:endX}, duration, mina.easeout);
  e.stopPropagation();
});
cvs.text(10, 130, 'easeout').attr({fontSize: fontSize});

// easein
var b3 = cvs.rect(startX, 150, 40, 40);
b3.attr({fill: '#90FF90', stroke: 'black', strokeWidth: '3px'});
b3.click( function(e, x, y) {
  b3.animate({x:endX}, duration, mina.easein);
  e.stopPropagation();
```



07/easing1.html

Easing Example

```
// animate2.js
// Drawing surface
var cvs = Snap("#cvs");

// Create ellipses
for (var x = 100; x <= 400; x += 100) {
  for (var y = 100; y <= 400; y += 100) {

    // Create and style ellipse
    var el = cvs.rect(x-20, y-20, 40, 40);
    el.attr({fill: 'white', stroke: 'black', strokeWidth: '5px'});

    // Add events
    el.hover(
      function(e) { this.animate({fill:'red', width:'80', height:'80'}, 500, mina.bounce); },
      function(e) { this.animate({fill:'white', width:'40', height:'40'}, 500, mina.bounce); }
    );
  }
}
```



07/animate2.html

Adding Data to an Element

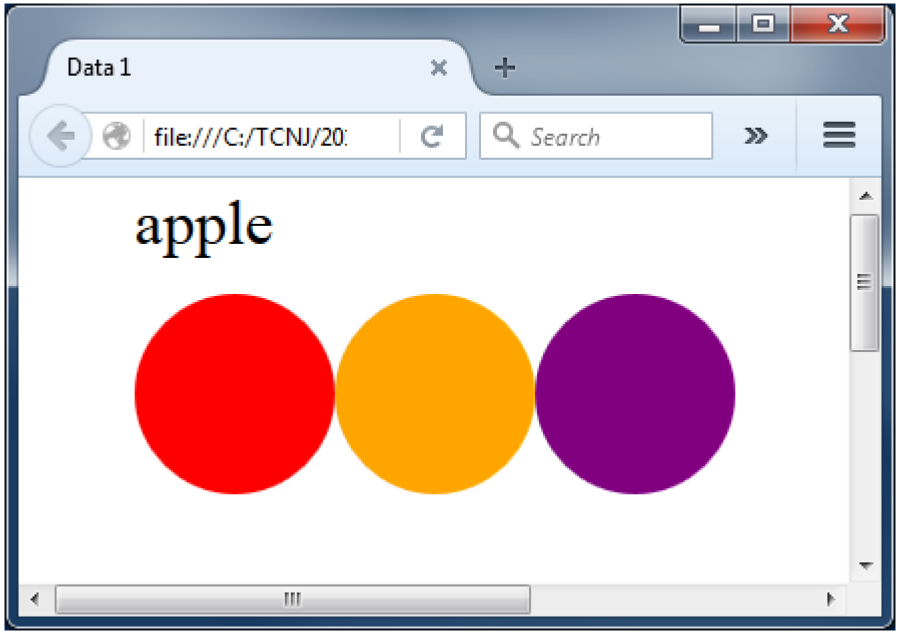
- Arbitrary key-value pairs may be attached to a Snap SVG Element
- Values may be read from a Snap SVG Element
- All data may be removed from a Snap SVG Element
- Handy for storing data associated with individual elements or groups

- `anElement.data(key, [value])`
  - Add data to `anElement`, `key=value`
  - Read data attached to `anElement` by providing `key` only

- `anElement.removeData()`
  - Remove all data attached to `anElement`

Example - Elements and Data

```
// data1.js

// Drawing surface
var cvs = Snap("#cvs");

// Text element for display
var message = cvs.text(50, 25, "");
message.attr({fontSize: '24pt'});

// Create three circles
// Attach data and event handlers
var apple  = cvs.circle(100, 100, 50);
apple.attr({fill: 'red'});
apple.data('name', 'apple');
apple.mouseover( function(e) {
  var name = this.data('name');
  message.attr({text: name});
});

var orange = cvs.circle(200, 100, 50);
orange.attr({fill: 'orange'});
orange.data('name', 'orange');
orange.mouseover( function(e) {
  var name = this.data('name');
  message.attr({text: name});
});

var grape  = cvs.circle(300, 100, 50);
grape.attr({fill: 'purple'});
grape.data('name', 'grape');
grape.mouseover( function(e) {
  var name = this.data('name');
  message.attr({text: name});
});
```



07/data1.html

Selecting SVG Elements

- Shape Elements subordinate to a `Paper` object or another `Element` object may be selected using CSS selector notation
- The first matching or all matching elements may be selected

`aPaper.select(` *`selector`* `)`

- Find the first Element that matches the CSS selector

`aPaper.selectAll(` *`selector`* `)`

- Find all Elements that match the CSS selector

`anElement.select(` *`selector`* `)`

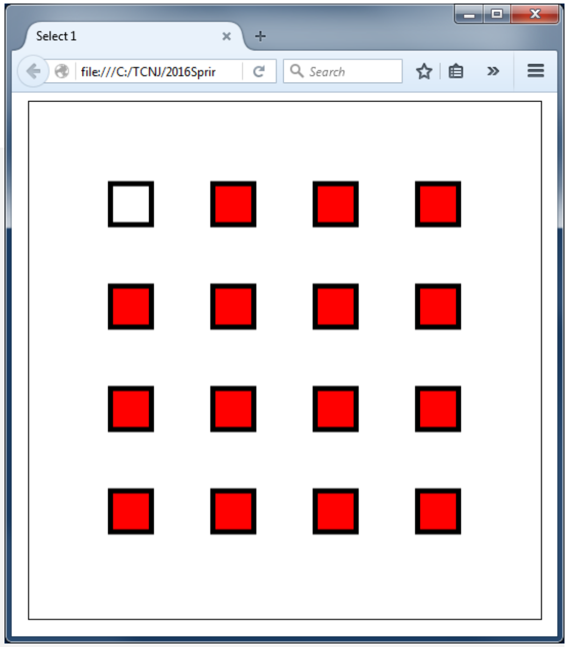
- Find the first nested Element that matches the CSS selector

`anElement.selectAll(` *`selector`* `)`

- Find all nested Elements that match the CSS selector

Select and Style SVG Elements

```
// select1.js
// Drawing surface
var cvs = Snap("#cvs");

// Create ellipses
for (var x = 100; x <= 400; x += 100) {
  for (var y = 100; y <= 400; y += 100) {

    // Create ellipse
    var el = cvs.rect(x-20, y-20, 40, 40);

  }
}
```



```
// Select and apply styles to ALL matching SVG elements
cvs.selectAll("rect").attr({fill: 'red', stroke: 'black', strokeWidth: '5px'});

// Select and apply styles to FIRST matching SVG elements
cvs.select("rect").attr({fill: 'white', stroke: 'black', strokeWidth: '5px'});
```

07/select1.html

Working with Element Structure

`anElement.append( element )`
- Appends the given element to current one

`anElement.appendTo( element )`
- Appends the current element to the given one

`anElement.insertBefore( element )`
- Inserts the element after the given one

`anElement.insertAfter( element )`
- Inserts the element after the given one

`anElement.prepend()`
- Prepends the given element to the current one

`anElement.prependTo()`
- Prepends the current element to the given one

`anElement.before( element )`
- Inserts given element before the current one

`anElement.after( element )`
- Inserts given element after the current one

`anElement.remove()`
- Removes element from the DOM

`anElement.parent()`
- Returns the element's parent

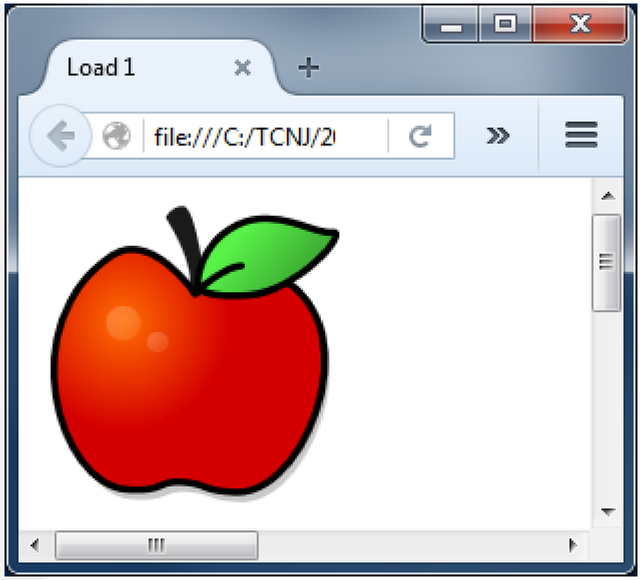
`anElement.clone()`
- Creates a clone of the element and inserts it after the element

Working with Existing SVG

- `Snap.load(url, callback)`
  - Loads external SVG file
- `Snap.ajax( … )`
  - More sophisticated version of Snap.load()
- `Snap.parse( svg )`
  - Parses SVG and converts it into a Fragment

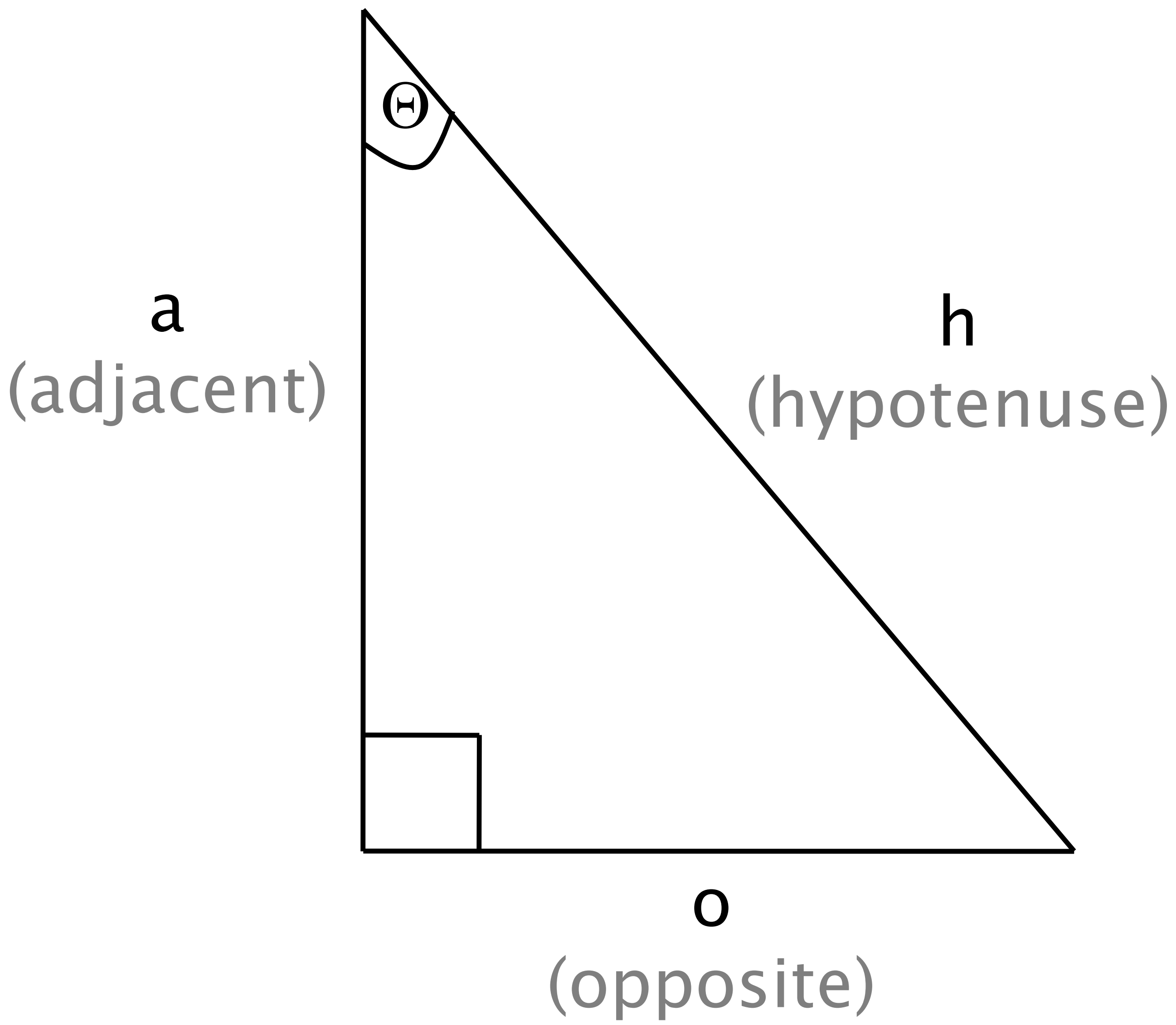
Loading SVG from a File

```
// load1.js

// Drawing surface
var cvs = Snap("#cvs");

// Insert a <g> element into canvas
var grp = cvs.group();

var notify = function( fragment ) {
  console.log("SVG loaded");
  grp.append( fragment );
};

// Load SVG image and invoke callback
var apple = Snap.load("apple.svg", notify);
```



07/load1.html

Basics of Trigonometry



Definition

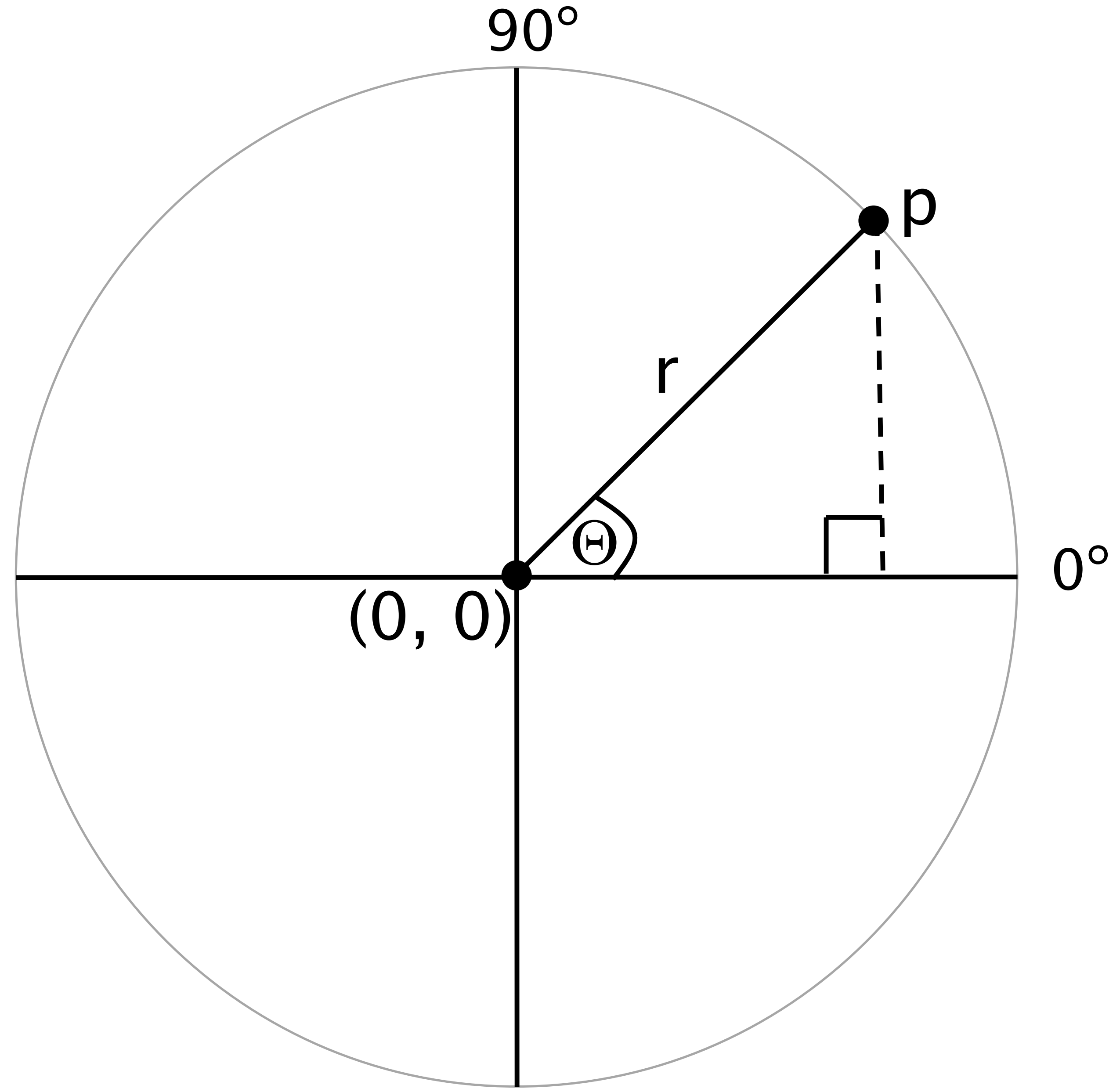
- $\sin(\Theta) = o/h$
- $o = h*\sin(\Theta)$

- $\cos(\Theta) = a/h$
- $a = h*\cos(\Theta)$

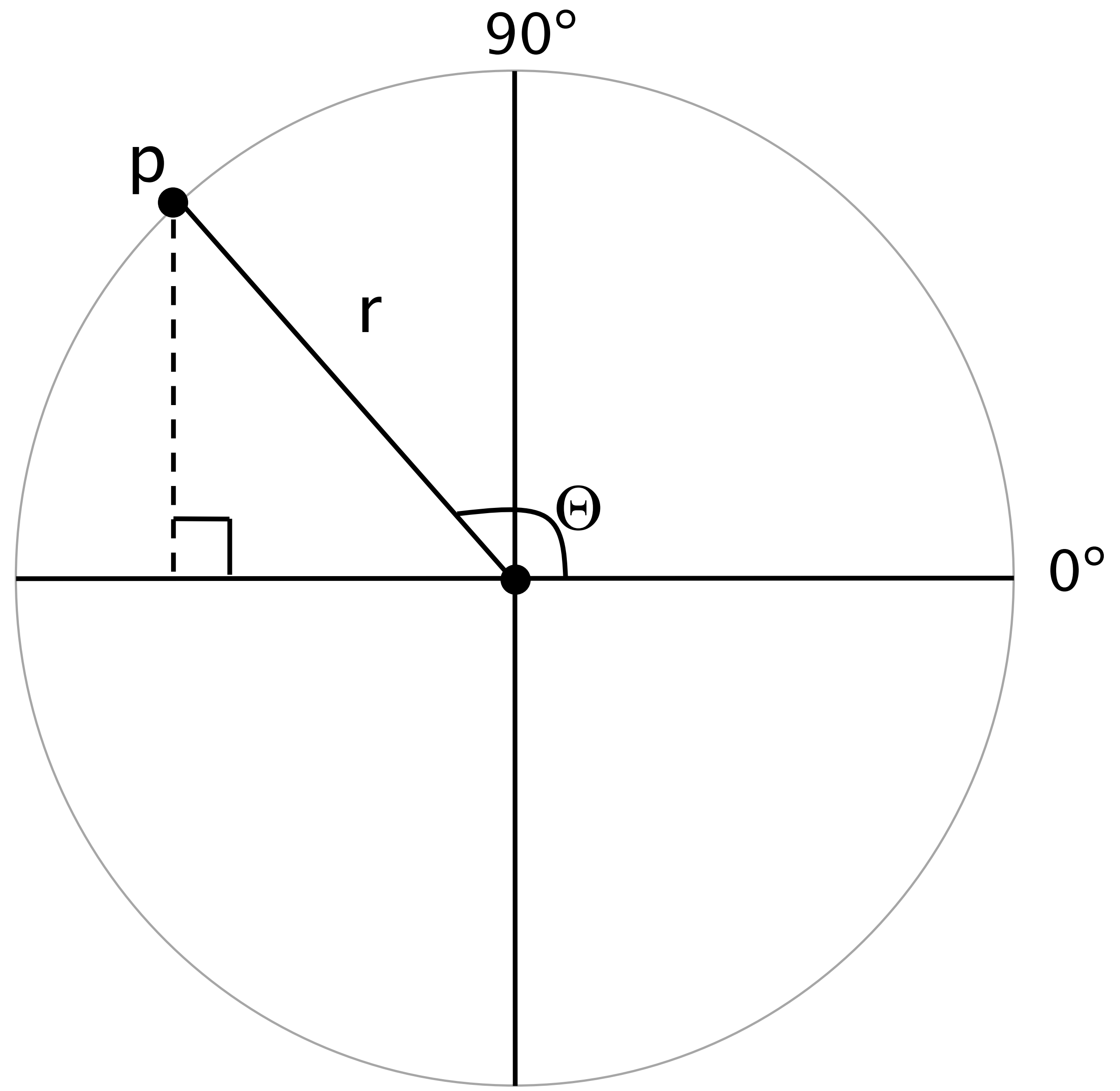
- $\tan(\Theta) = o/a = \sin(\Theta)/\cos(\Theta)$

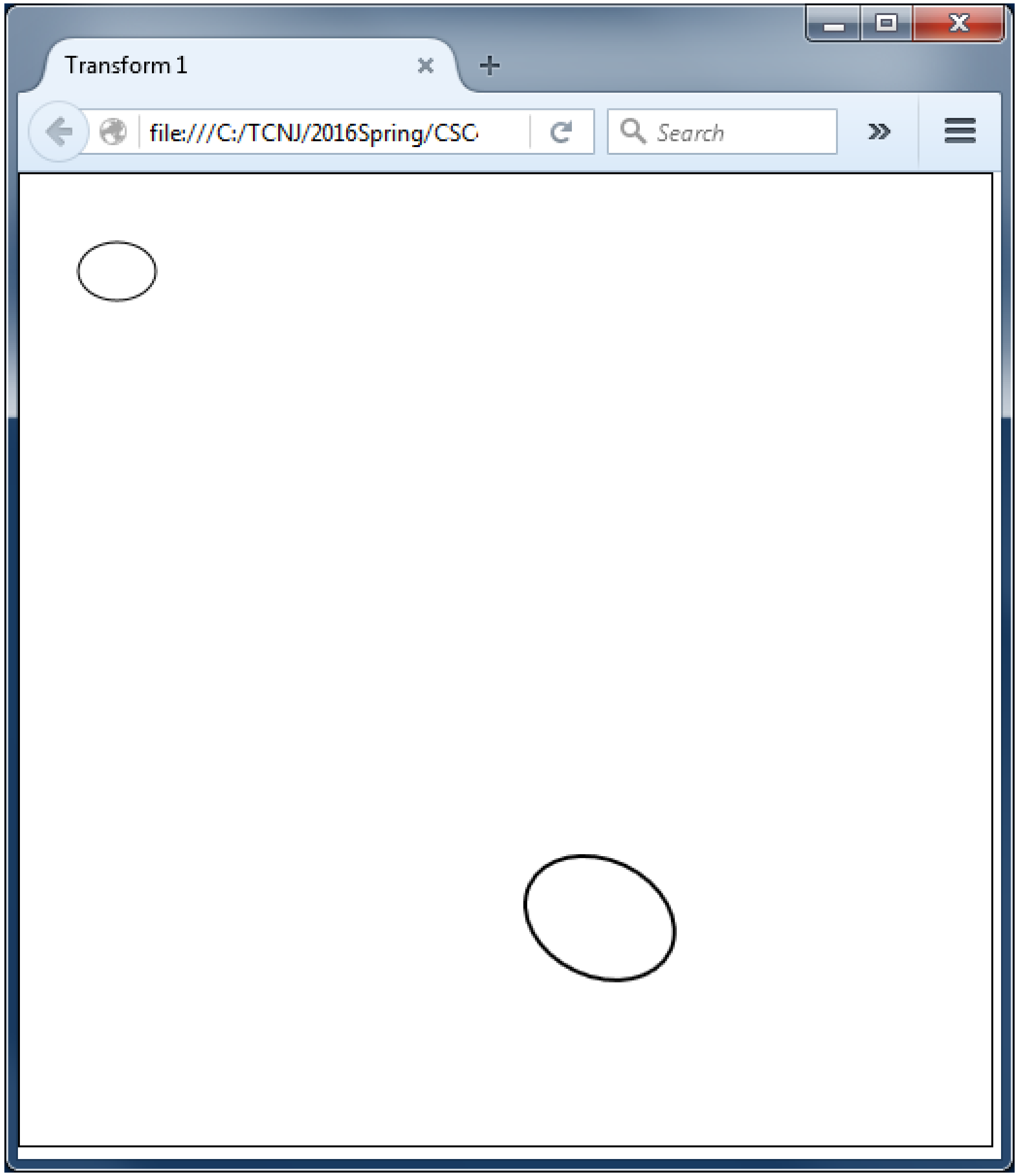
sohcahtoa

Trigonometry on a unit circle



Trigonometry on a unit circle

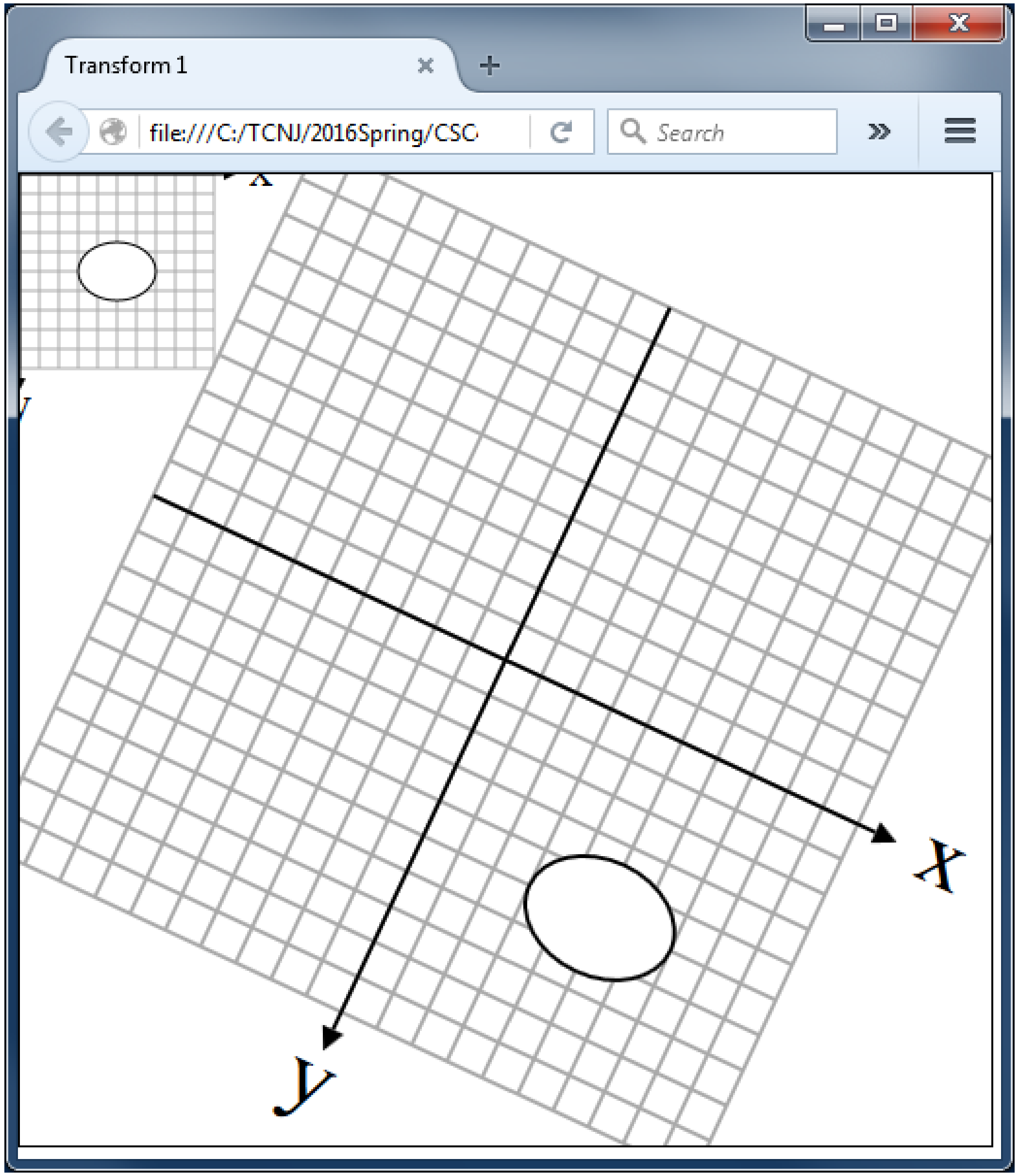




The command that draws these two ellipses are identical.

What changed is the coordinate system on which they are drawn.

07/xform1.html



The commands that draw these two ellipses are identical.

What changed is the coordinate system in which they are drawn.

07/xform1.html

Five ways to <u>transform</u> the coordinate system:

1. **Translate**
   - Move axes left, right, up, down …
2. **Scale**
   - Magnify, zoom in, zoom out …
3. **Rotate**
   - Tilt clockwise, tilt counter-clockwise …
4. **SkewX**
   - Skew along x-axis…
5. **SkewY**
   - Skew along y-axis…

Scale

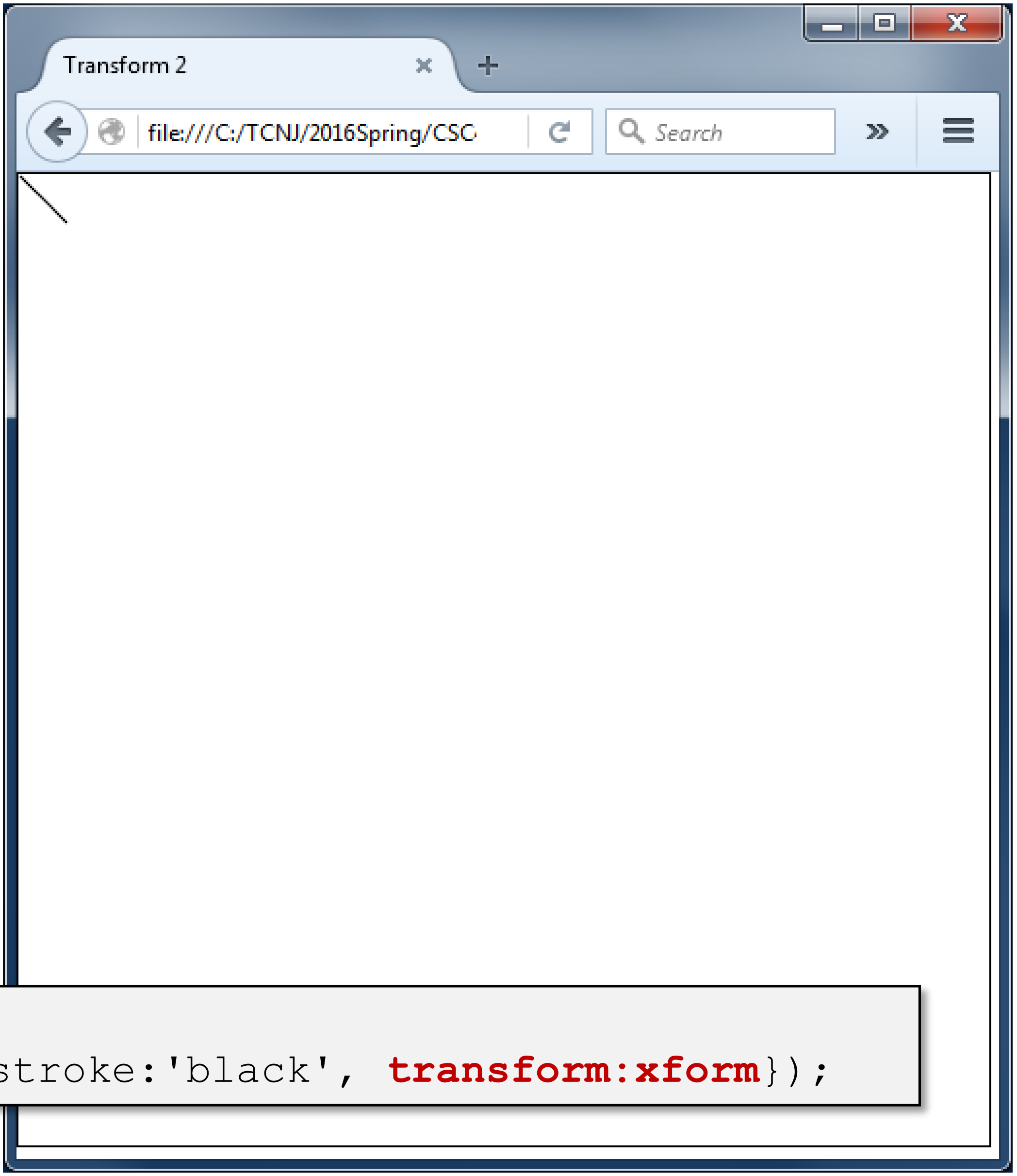
- Scales the coordinate system for this shape only
- All coordinates are multiplied by an x-scale-factor and a y-scale-factor.
- The size of everything is "magnified" about the origin (0,0)
- Stroke width is also scaled.

Transformation string

```
scale(<x> [<y>])
```

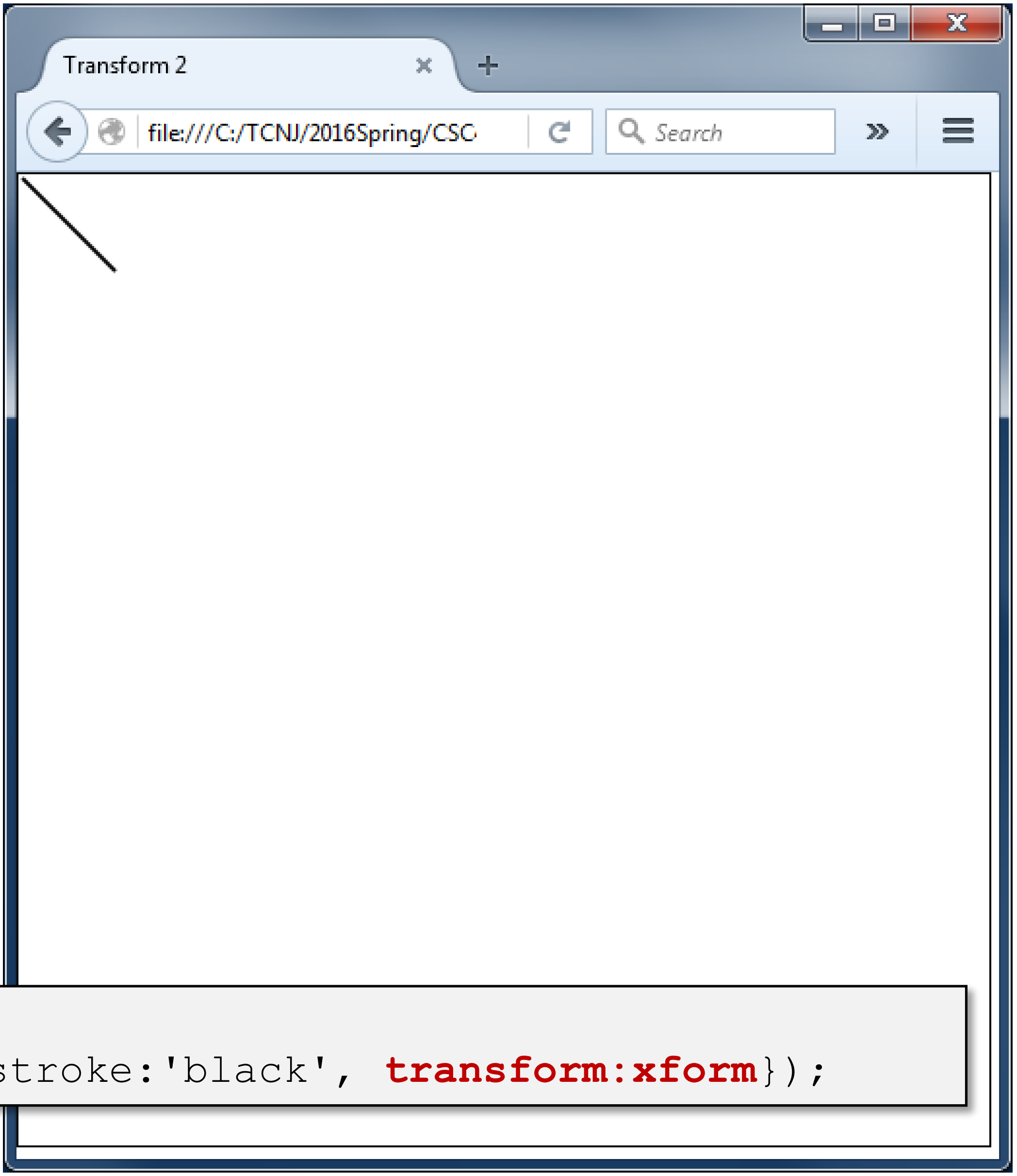
To transform a Snap SVG element…

- set the `transform` attribute to the transform string
- or invoke the `transform()` method of a Snap SVG Element



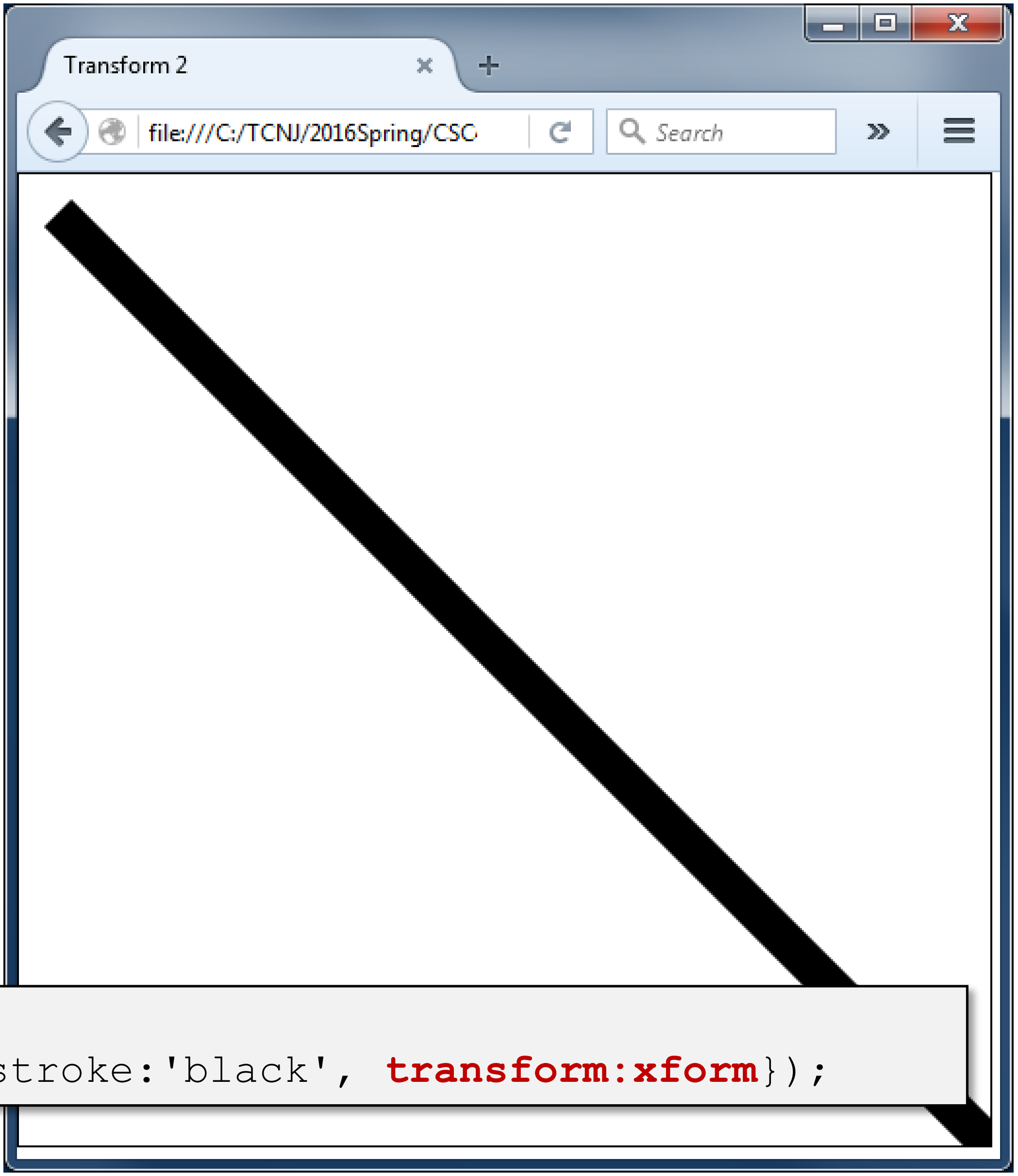
```
var xform = '';
cvs.line(1, 1, 25, 25).attr({stroke:'black', transform:xform});
```

07/xform1.html



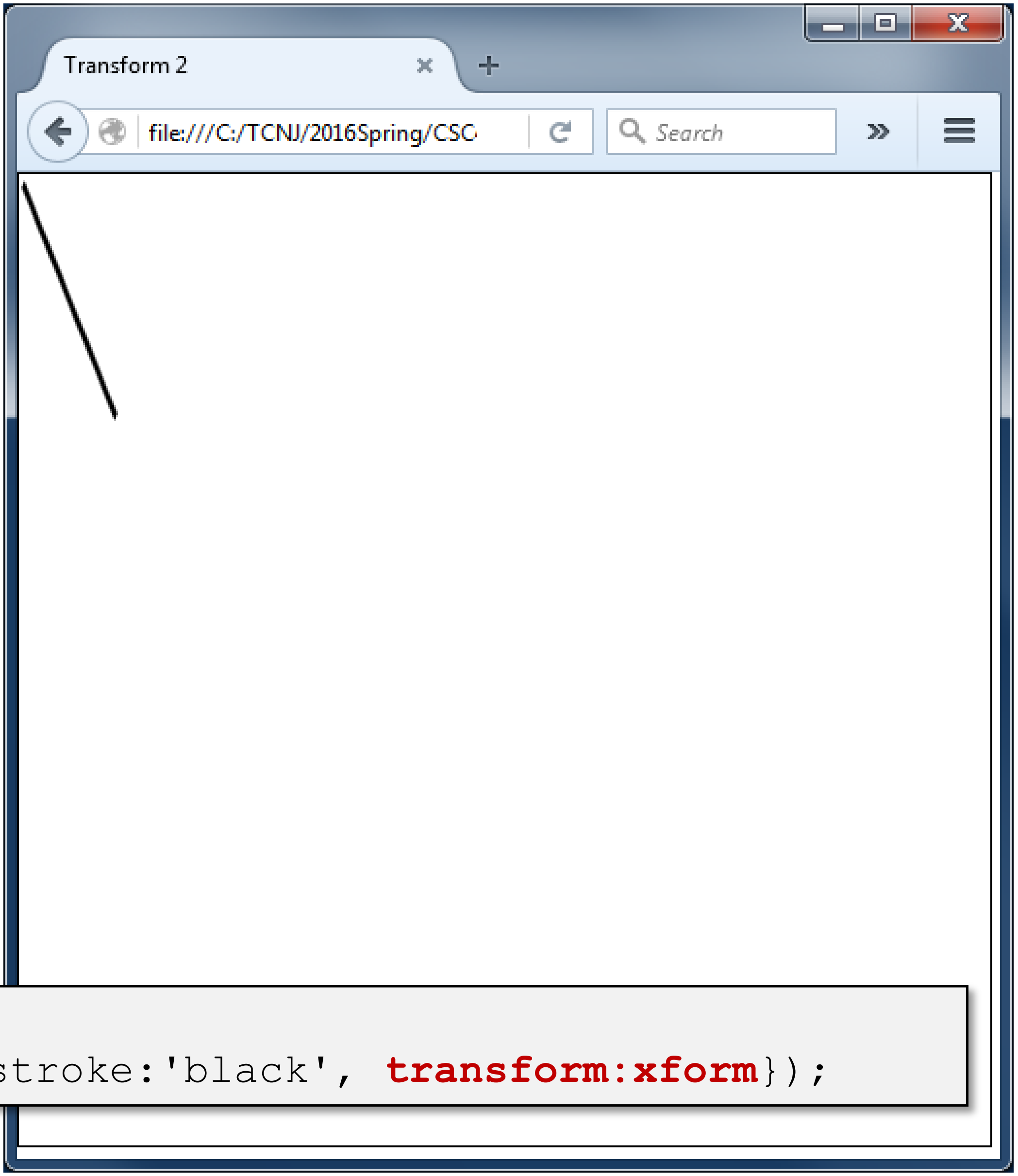
```
var xform = 'scale(2)';
cvs.line(1, 1, 25, 25).attr({stroke:'black', transform:xform});
```

07/xform2.html



```
var xform = 'scale(20)';
cvs.line(1, 1, 25, 25).attr({stroke:'black', transform:xform});
```

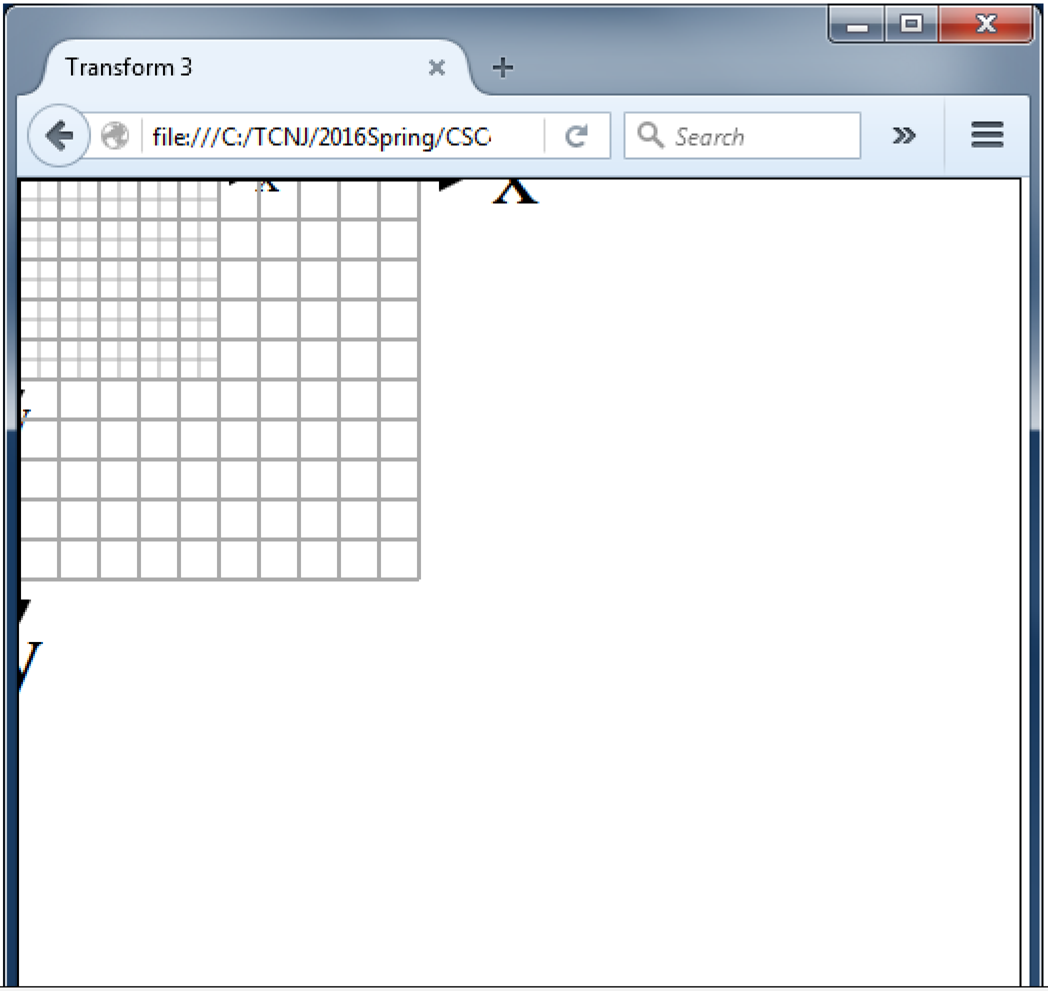
07/xform2.html



```
var xform = 'scale(2 5)';
cvs.line(1, 1, 25, 25).attr({stroke:'black', transform:xform});
```

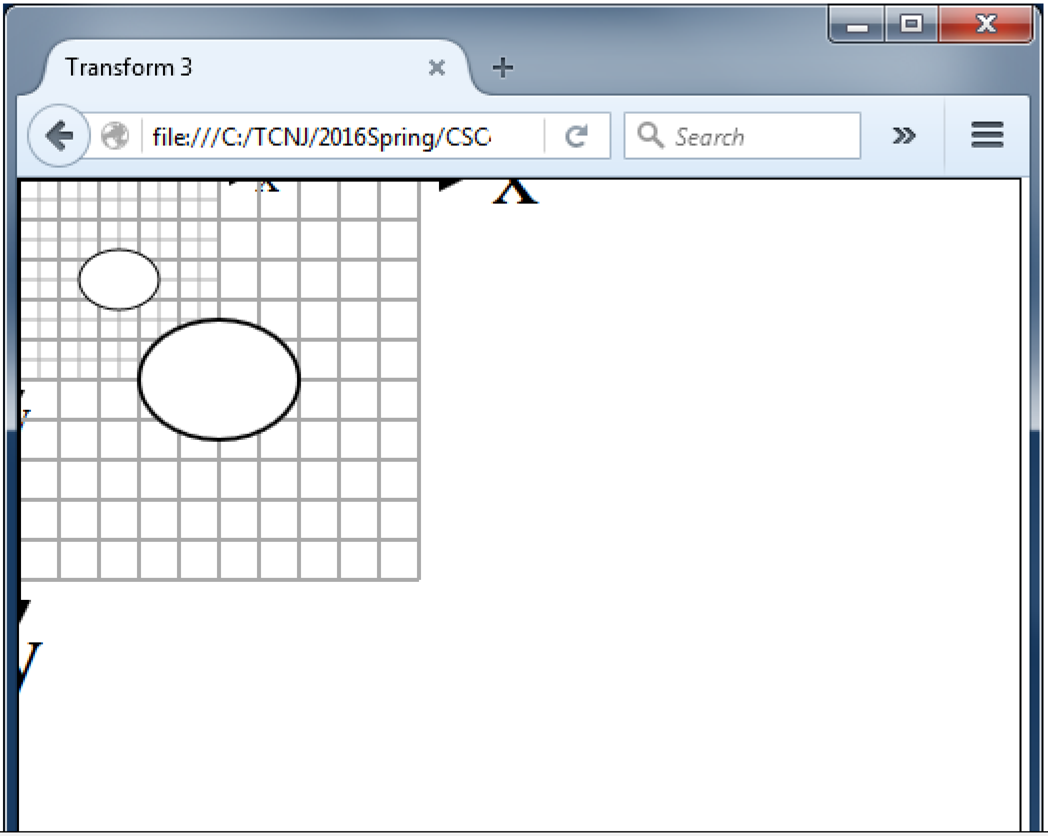
07/xform2.html

The best way to see what is happening, is to look at a grid drawn in the coordinate system.



```
var xform = 'scale(2)';

grid();
grid(xform);
```

07/xform3.html



```
// Transform string
var xform = 'scale(2)';

// Create grids
grid();
grid(xform);

// Create two ellipses
cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black'});
cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black', transform:xform});
```

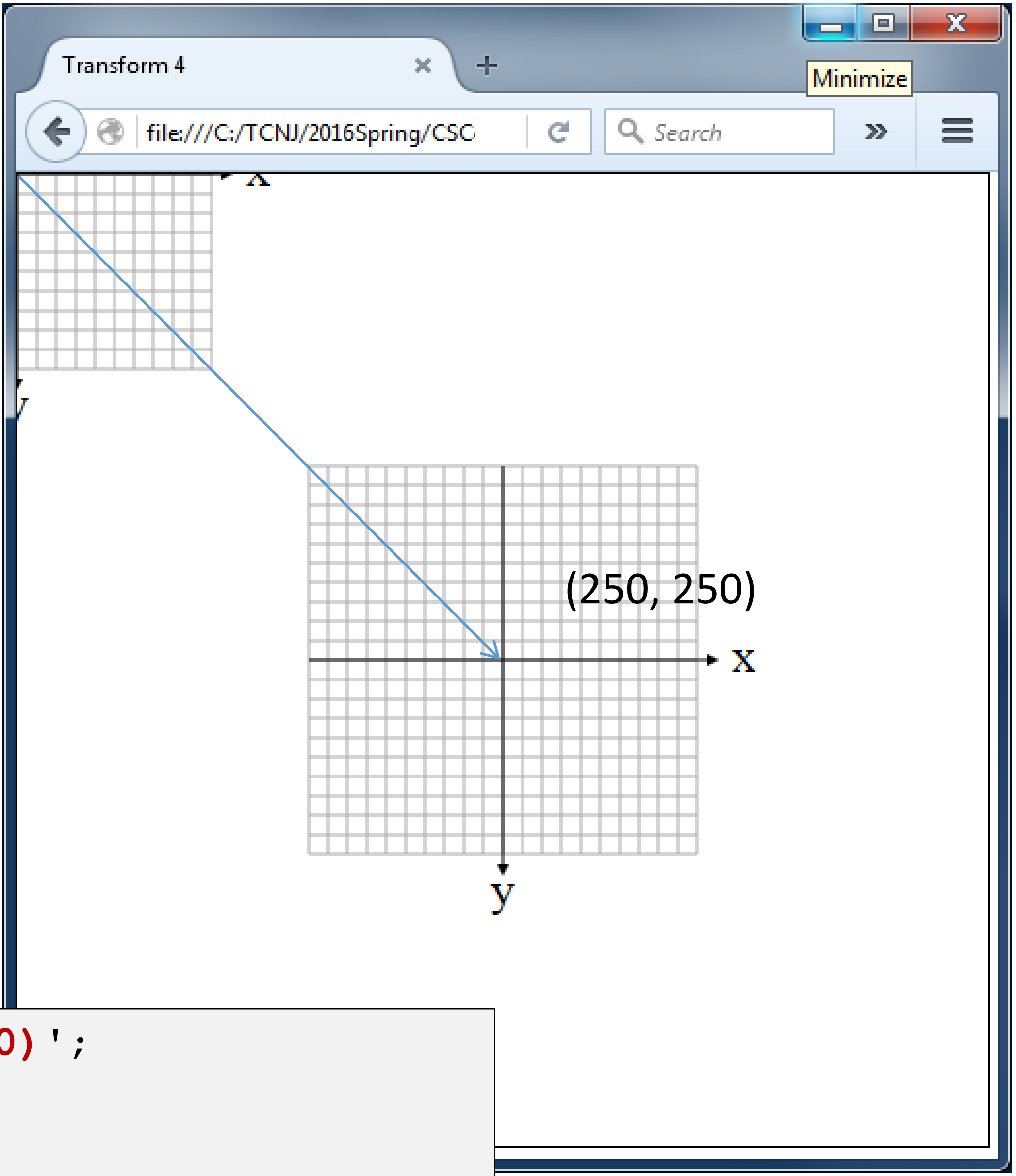
07/xform3.html

Translate

- Moves the coordinate system for this shape only
- The origin of the coordinate system (0,0) is shifted by the given amount in the x and y directions.

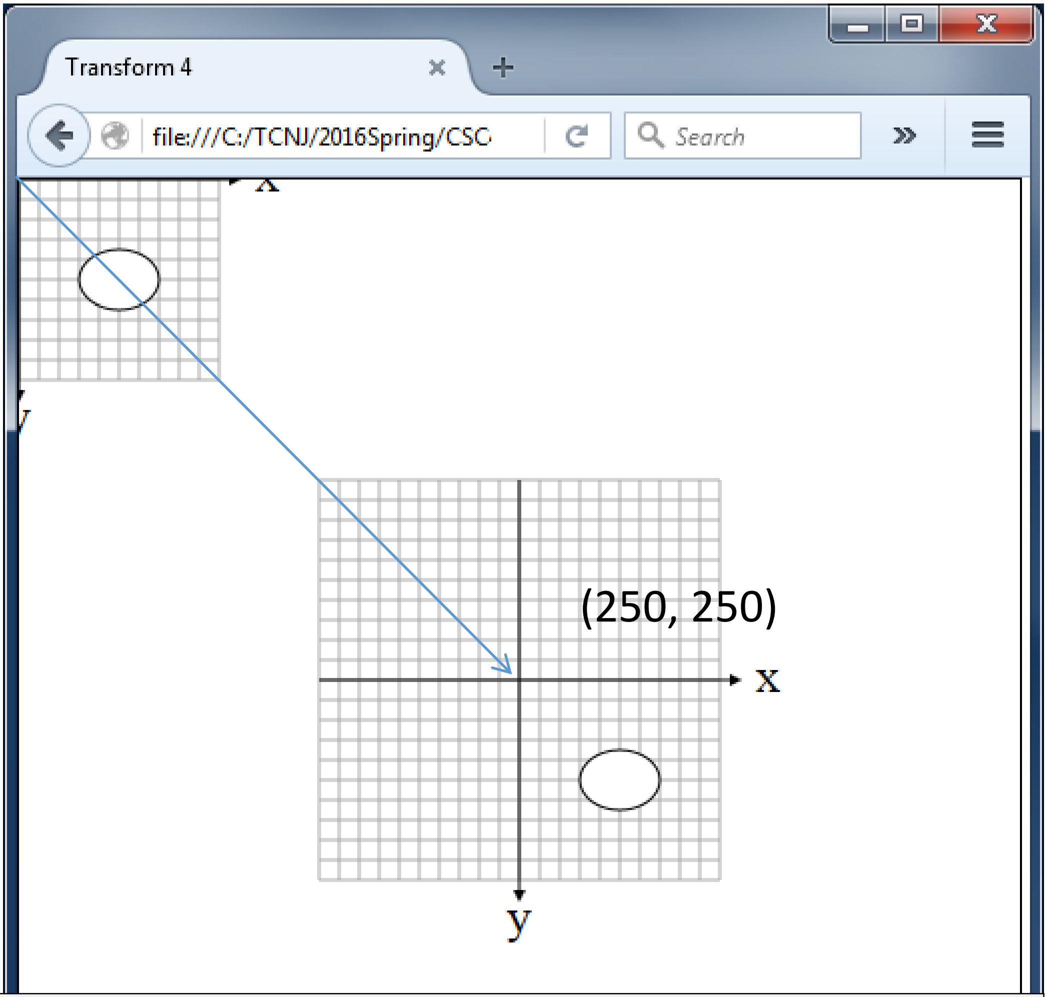
Transformation string

```
translate(<x> [<y>])
```



```
var xform = 'translate(250 250)';

grid();
grid(xform);
```

07/xform4.html



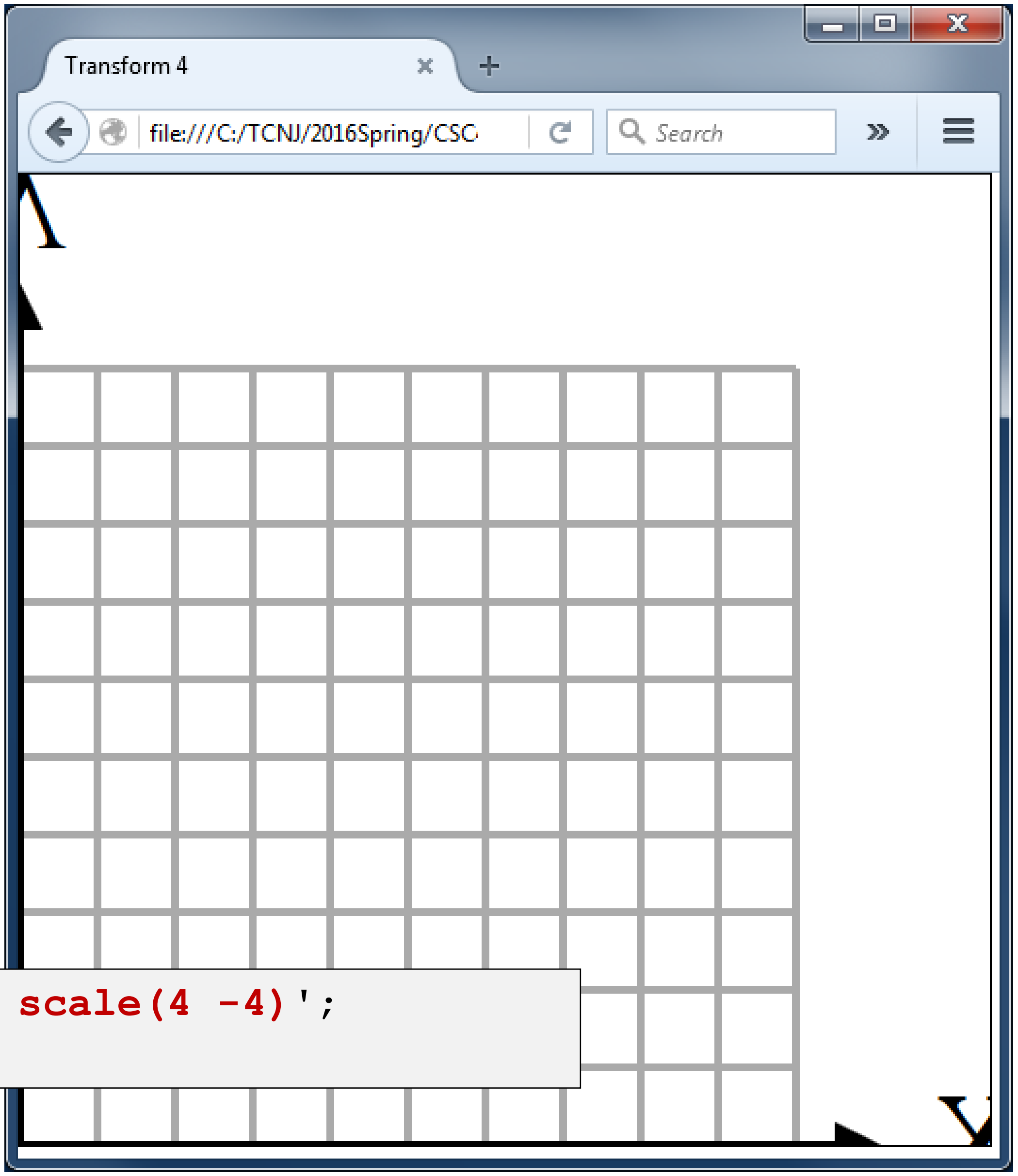
```
var xform = 'translate(250 250)';

grid();
grid(xform);

cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black'});
cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black', transform:xform});
```

07/xform4.html

Transformations can be combined

- Combine Scale and Translate to create a coordinate system with the y-axis that increases in the upward direction
- Axes can be flipped using negative scale factors

Order in which transforms are applied matters!



```
var xform = 'translate(0 500) scale(4 -4)';
grid(xform);
```

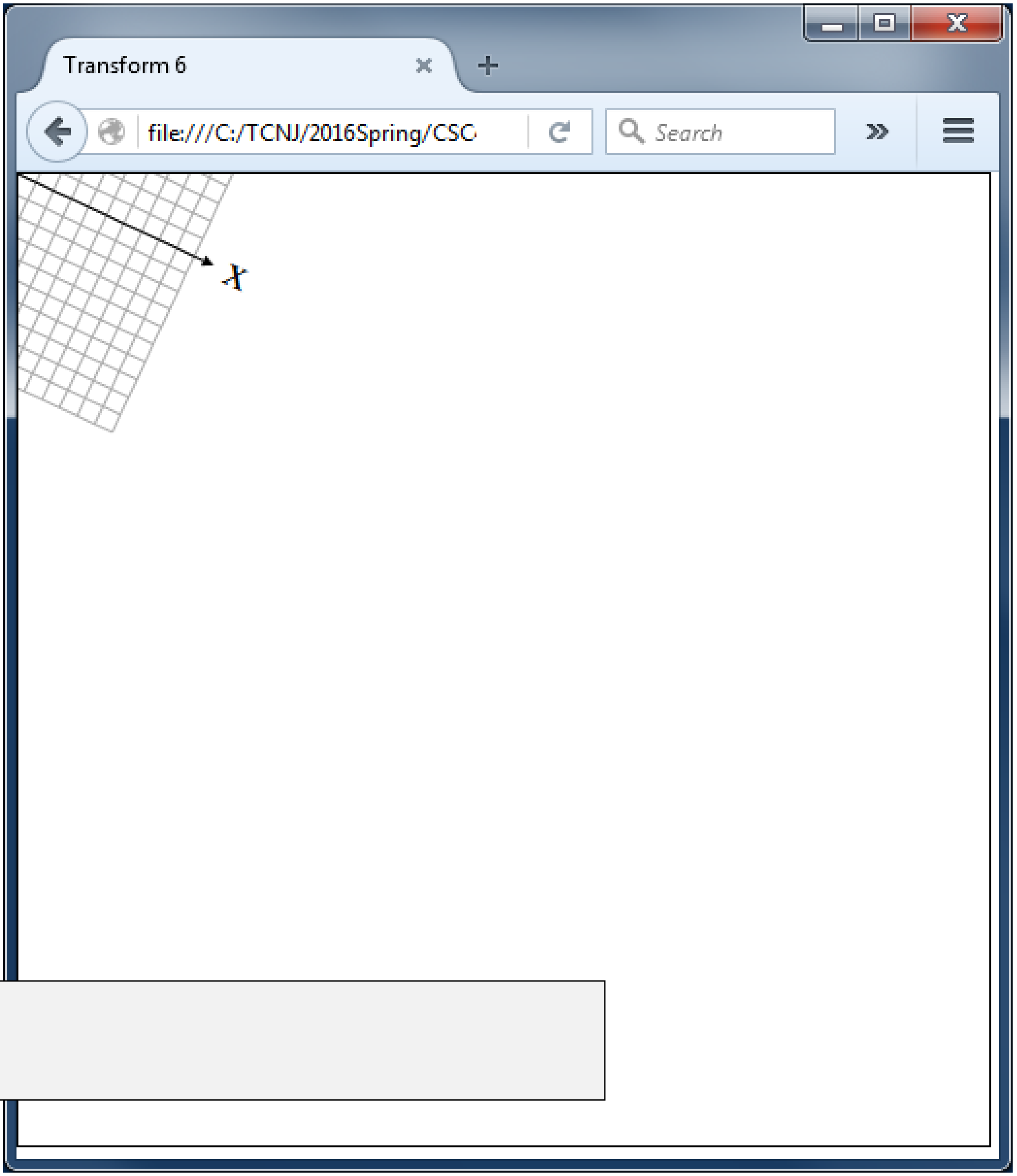
07/xform5.html

Rotate

- Rotate the coordinate system for this shape only
- The coordinate system is rotated around `(x, y)` by the given angle `a` (in degrees).

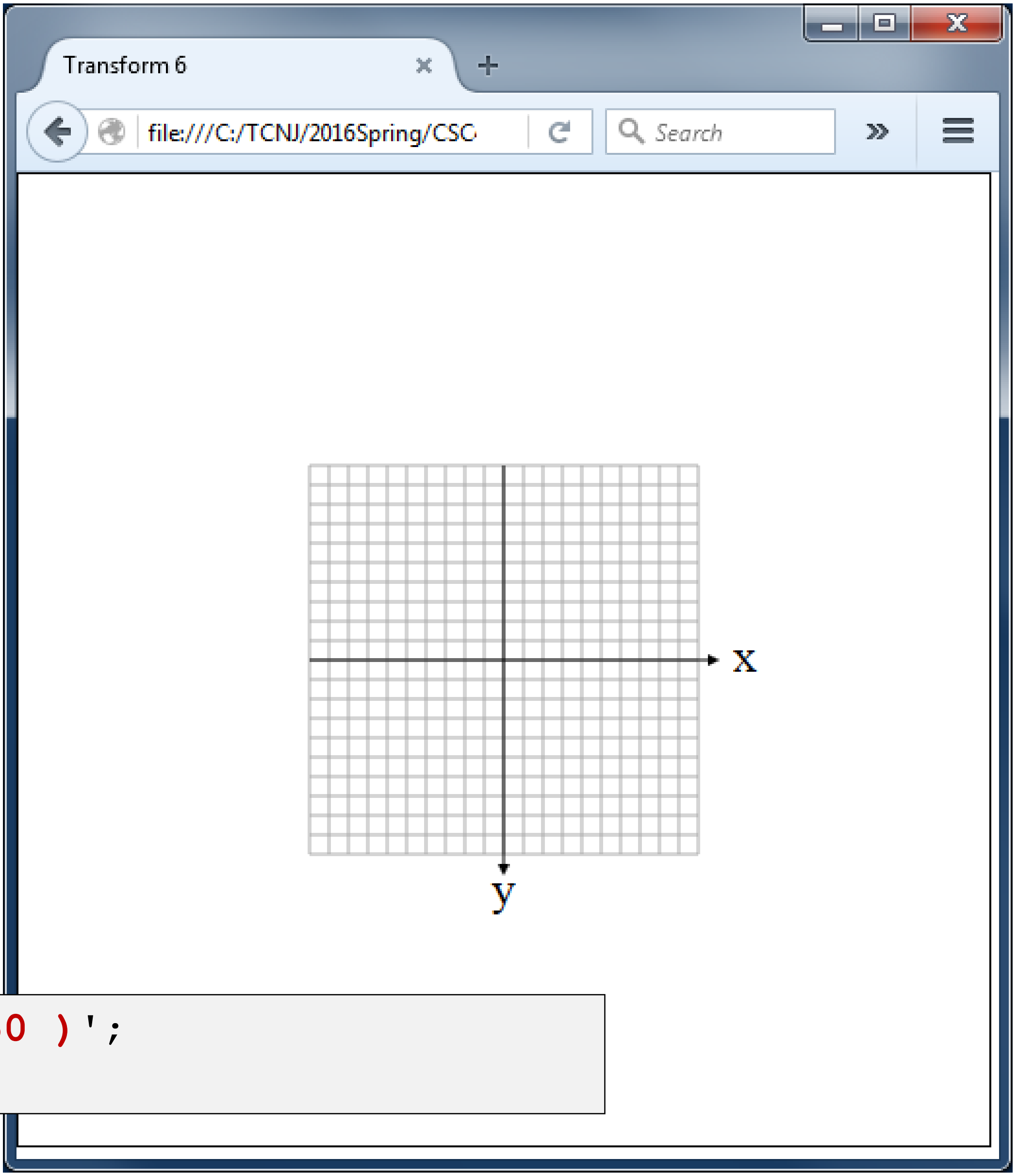
Transformation string

```
rotate(<a> [<x> <y>])
```



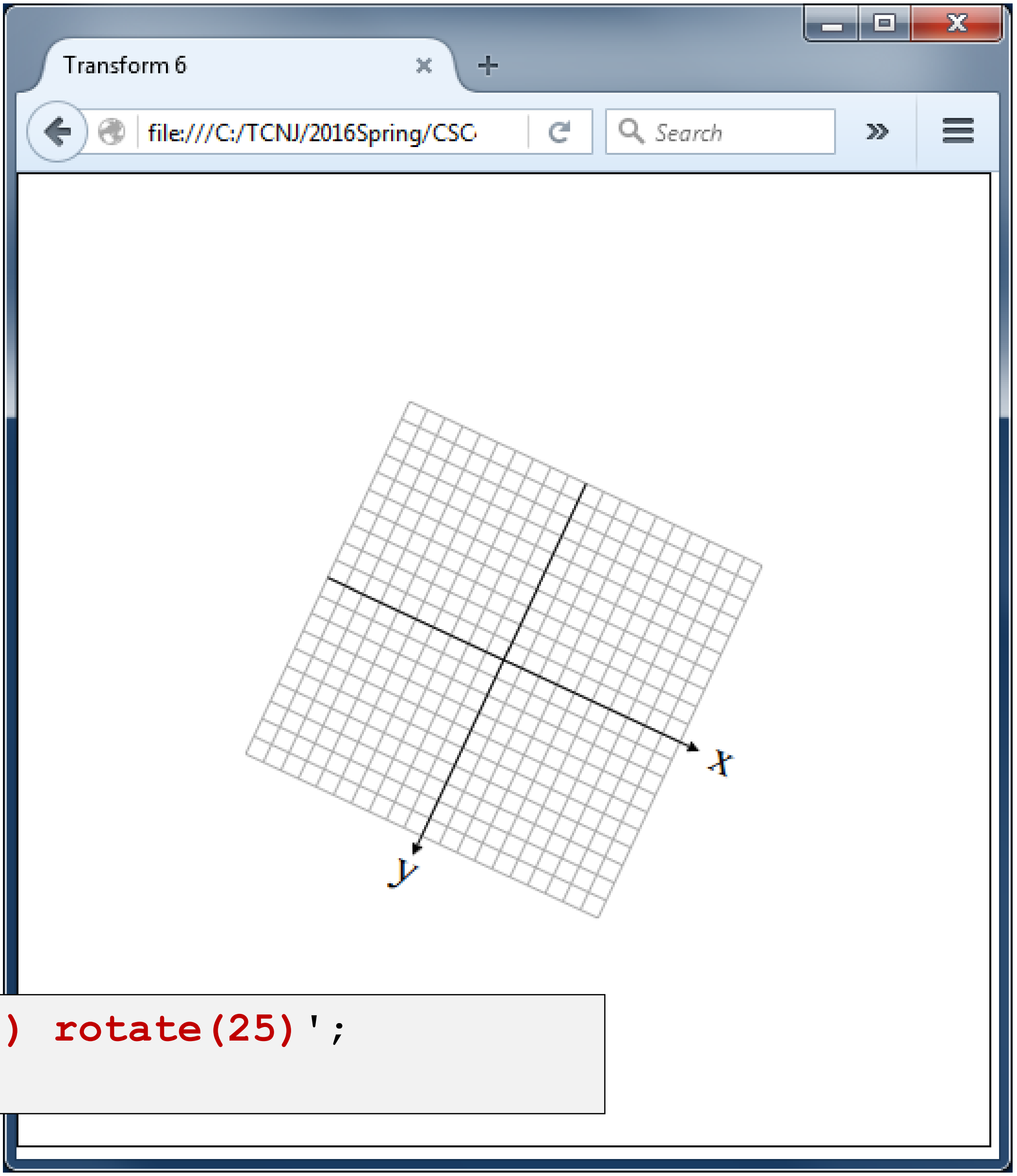
```
var xform = 'rotate(25)';
grid(xform);
```

07/xform6.html



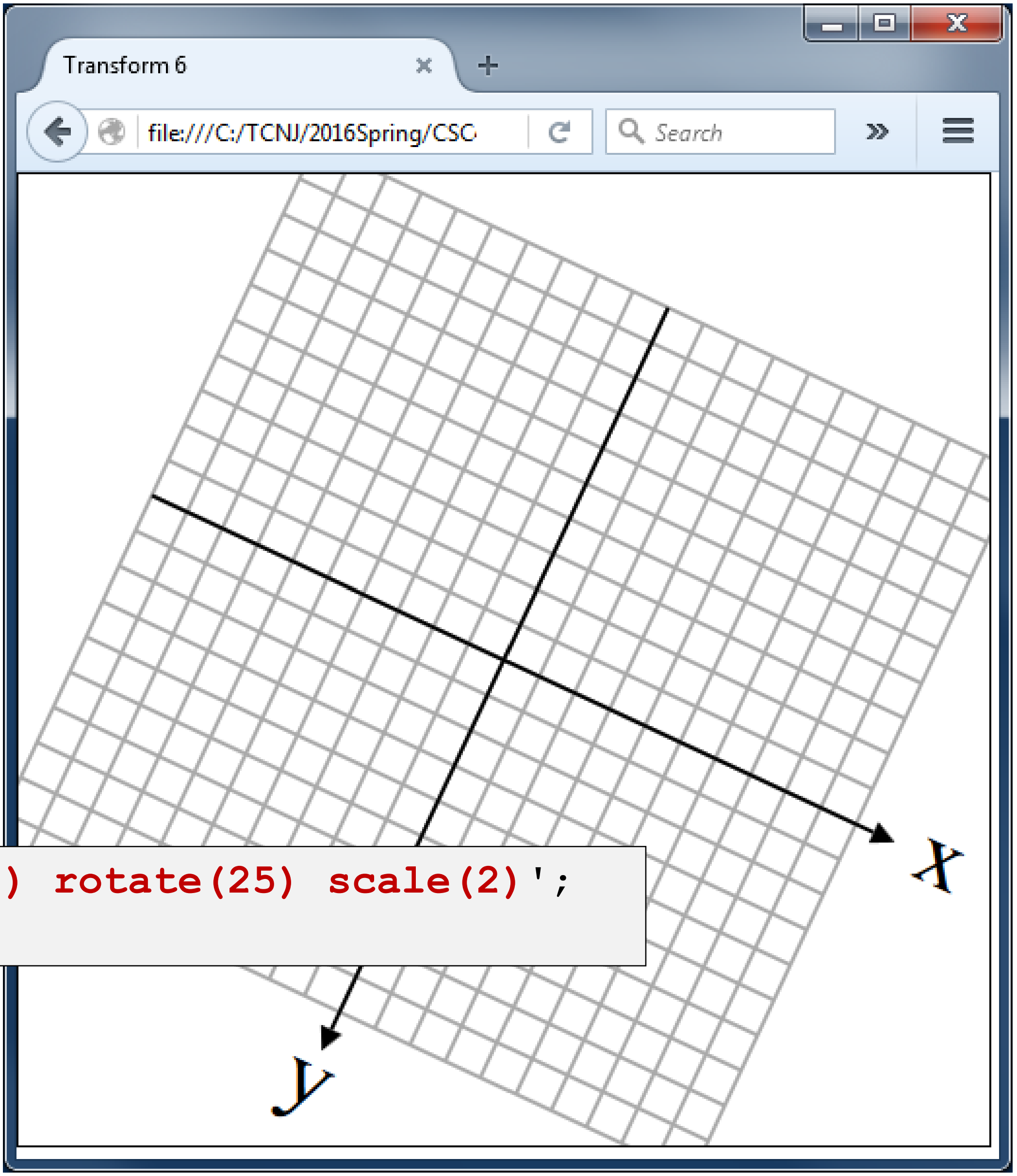
```
var xform = 'translate( 250 250 )';
grid(xform);
```

07/xform6.html



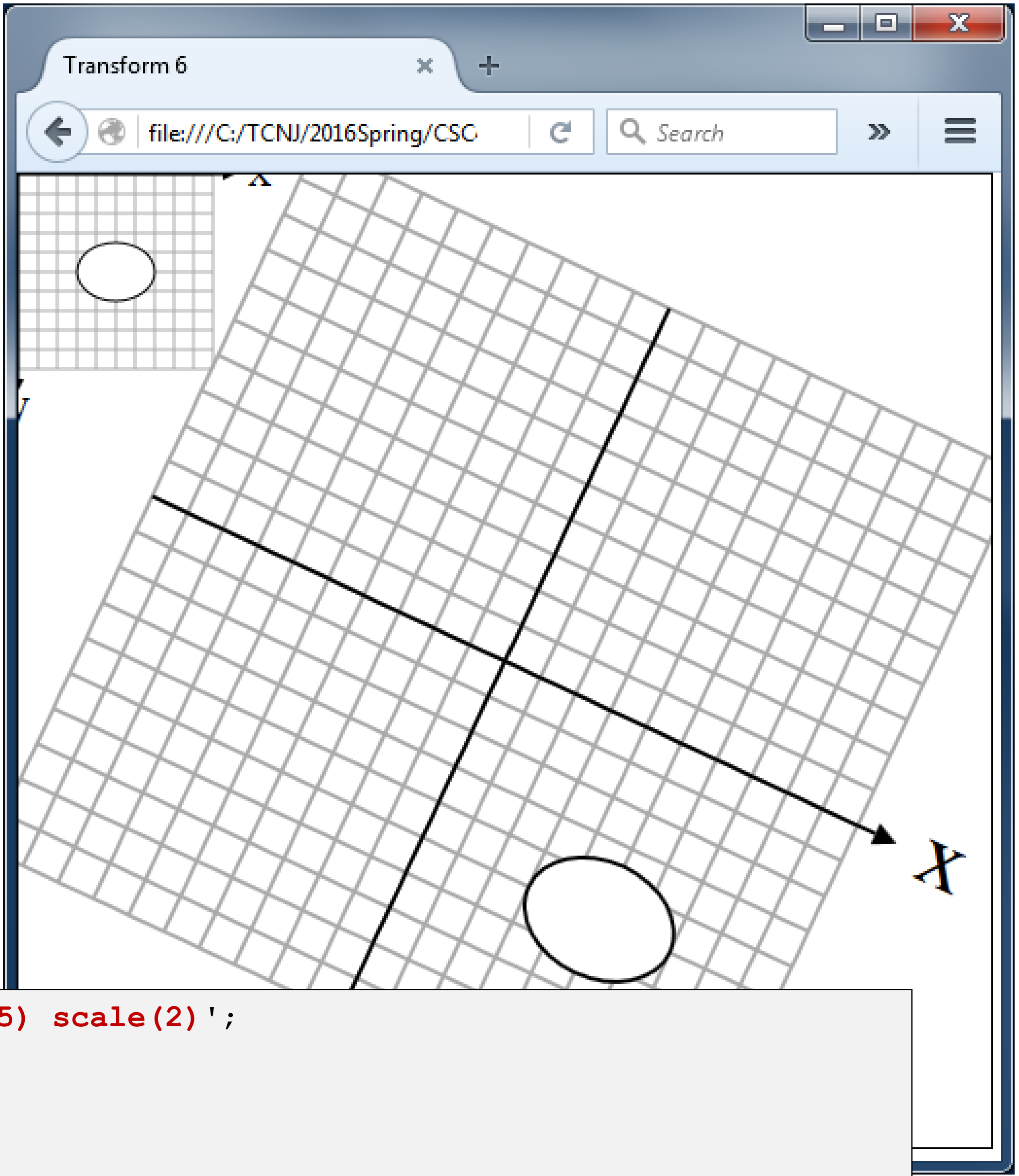
```
var xform = 'translate(250 250) rotate(25)';
grid(xform);
```

07/xform6.html



```
var xform = 'translate(250 250) rotate(25) scale(2)';
grid(xform);
```

07/xform6.html



```
var xform = 'translate(250 250) rotate(25) scale(2)';

grid();
grid(xform);

cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black'});
cvs.ellipse(50, 50, 20, 15).attr({fill:'white', stroke:'black', transform:xform});
```

07/xform6.html

Snap SVG Transformation Strings

- Snap SVG provides a **shorthand string notation** for transformations
- To transform an Snap SVG element, set the `transform` attribute to the shorthand string

Shorthand notation for transformations

`t dx dy`

  - Translate coordinate system by `(dx, dy)`

`s sx [sy] x y`

  - Scale coordinate system by `sx` around the point `(x, y)`
  - **Note the expanded notation**

`r a [x y]`

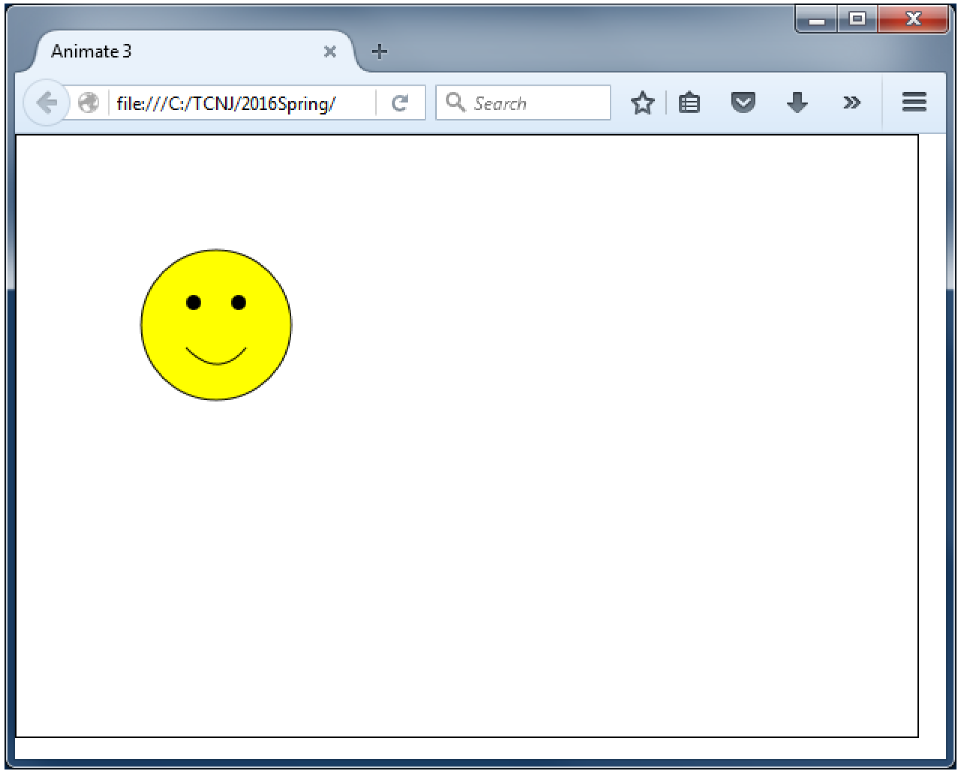
  - Rotate coordinate system by `a` degrees around the point `(x, y)`

- Whitespace between items in string necessary only when needed to distinguish between items
- Also may use ',' instead of space
- Combine transformations by concatenating strings

Animating with Transforms

```
// animate3.js

// Drawing surface
var cvs = Snap("#cvs");

// Face
var face = cvs.circle(50, 50, 50);
face.attr({fill: 'yellow', stroke: 'black'});

// Eyes
var eye1 = cvs.circle(35, 35, 5);
eye1.attr({fill: 'black'});

var eye2 = cvs.circle(65, 35, 5);
eye2.attr({fill: 'black'});

// Smile
var smile = cvs.path('M30,65 S50,90,70,65');
smile.attr({stroke: 'black', fill: 'none'})

// Add all items to a group
var g = cvs.group(face, eye1, eye2, smile);

cvs.mousemove( function(e, x, y) {
  // Calculate distance to new location
  var bbox = g.getBBox();
  var dx = x - (bbox.x+50);
  var dy = y - (bbox.y+50);
  var dur = 2*Math.sqrt(dx*dx + dy*dy);

  // Stop any animations in progress
  g.stop();

  // Start a new animation
  var tstring = 'T' + (x-50) + ' ' + (y-50);
  g.animate({transform: tstring}, dur);
});
```

- Happy Face that follows the mouse
- Group to process all elements together
- Transform string animated
- **Set target transformation in animate() method**



07/animate3.html

More Snap SVG Examples

- http://snapsvg.io/demos/
- http://svg.dabbles.info/