HTML5 Canvas

Joachim.pietsch@iadt.ie

Canvas 2d

HTML element

W3C: http://www.w3.org/TR/html5/the-canvas-element.html

Default dimensions: 300 x 150

- Manipulate pixels on the canvas using JS drawing API
- Draw photos to the canvas
- Draw graphic shapes
- Rotate, translate shapes

Fallback Content

Easy to add HTML content that is shown if no Canvas support.

Browsers that support Canvas will ignore the content inside the tag

```
<canvas id="mycanvas">
    Sorry, you need a modern
    browser to view this content!
</canvas>
```

The 2D Context

Drawing APIs are exposed via context objects, created with canvas.getContext().

Example:

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

The name of the 2D drawing context is “2d”.

The canvas specification allows other contexts, for example “webgl” for 3D graphics.

Canvas

- HTML5 element
- but elements are rendered by JS

```
<canvas id="canvas" width="800" height="600"></canvas>
```

- Canvas is now accessed in JS and stored in a variable.
- Then the context of the canvas is set, to be either 2d or 3d

```
var can = document.getElementById("canvas");
var ctx = can.getContext("2d");
```

Drawing a Line

- Start with beginPath();
- then use moveTo() to move to the start of the line
- Then lineTo() to draw to the 2nd point of the line
- lineWidth changes the lineWidth;
- strokestyle(#hexRGB); to draw a coloured line;
- Then stroke() to apply stroke to line draw the line;

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(450, 50);
context.lineWidth = 10;
context.strokeStyle = '#ff0000';
context.stroke();
```

Fills and Styles

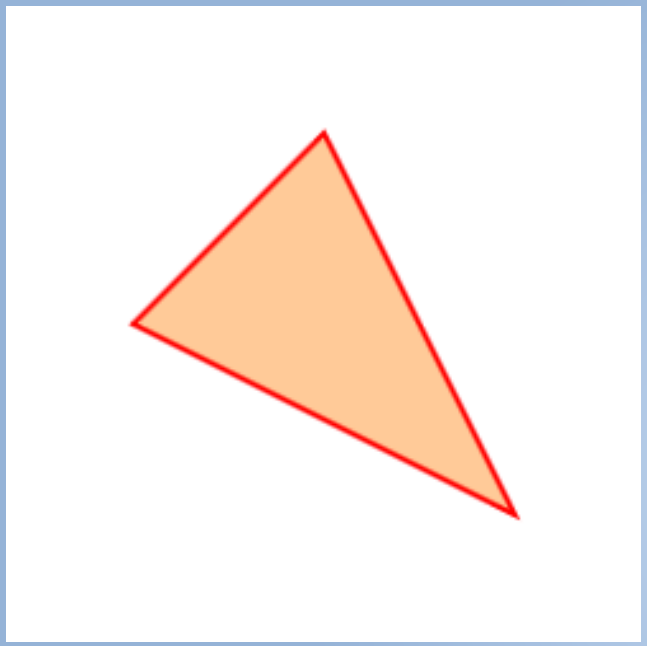
Set ctx.fillStyle and ctx.strokeStyle to change the color of the fill/stroke.

Any CSS color is accepted:

Stroking draws a line along the path. The thickness of line can be controlled with ctx.lineWidth:

```
ctx.fillStyle = "cyan";
ctx.fillStyle = "rgb(255,0,127)";
ctx.strokeStyle = "#a8c022";
ctx.strokeStyle = "hsla(120, 40%, 50%, 0.8)";


ctx.lineWidth = 2; //double width
```

Example: A Triangle



```
ctx.beginPath();
ctx.moveTo(125, 50);
ctx.lineTo(200, 200);
ctx.lineTo(50, 125);
ctx.closePath();

ctx.fillStyle =
"rgba(255,150,50,0.5)";
ctx.strokeStyle = "red";

ctx.fill();
ctx.stroke();
```

beginPath & closePath()

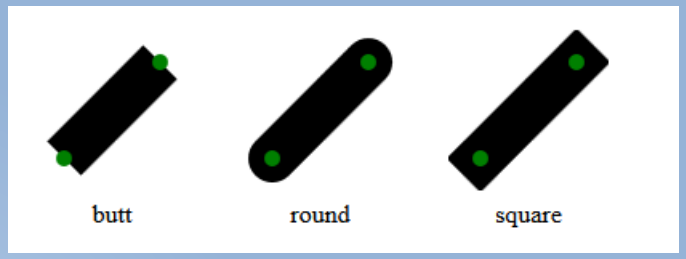
beginPath()

a new shape is being drawn with

closePath()

closes the begun Path, i.e. it links the currentPoint to the startPoint

Line Cap



Use ctx.lineCap to change the appearance of line endings when stroking paths:

lineCap property changes the caps of the line

3 styles: ‘round’, ‘butt’, ‘square’);

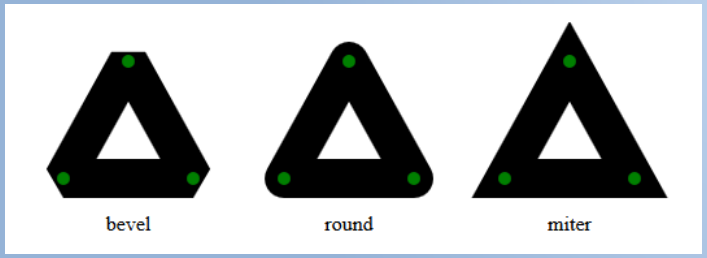
```
context.lineCap = 'round';
```

Line Join

The lineJoin property defines how paths are joined together.

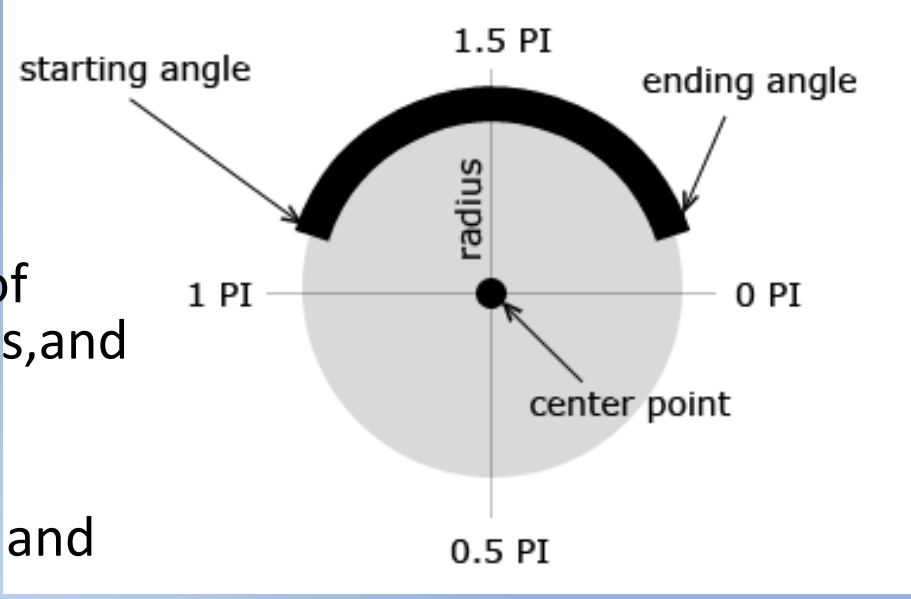
Possible values are ‘miter’,’round’,’bevel’

Default value: ‘miter’



```
context.lineJoin = 'miter’;
```

Arcs

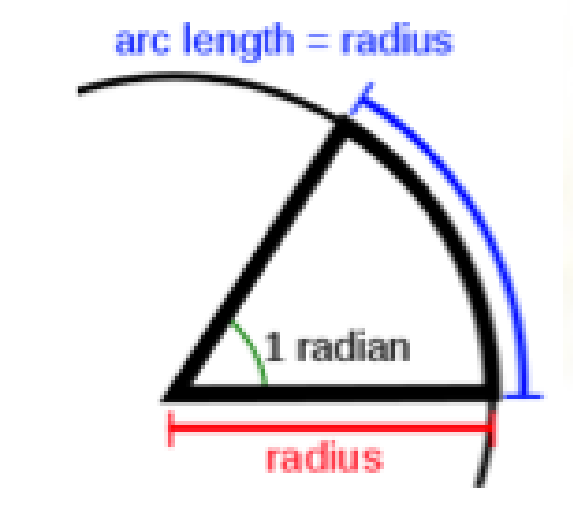


- Arc() method takes 5 arguments: x,y of center, startAngle, endAngle in radians,and drawing direction (optional).
- Angles are expressed in Radians.
- Style Arcs with lineWidth, strokeStyle and lineCap properties.

```
var x = canvas.width / 2;
var y = canvas.height / 2;
var radius = 75;
var startAngle = 1.1 * Math.PI;
var endAngle = 1.9 * Math.PI;
var counterClockwise = false;

context.beginPath();
context.arc(x, y, radius, startAngle, endAngle,
counterClockwise);
```

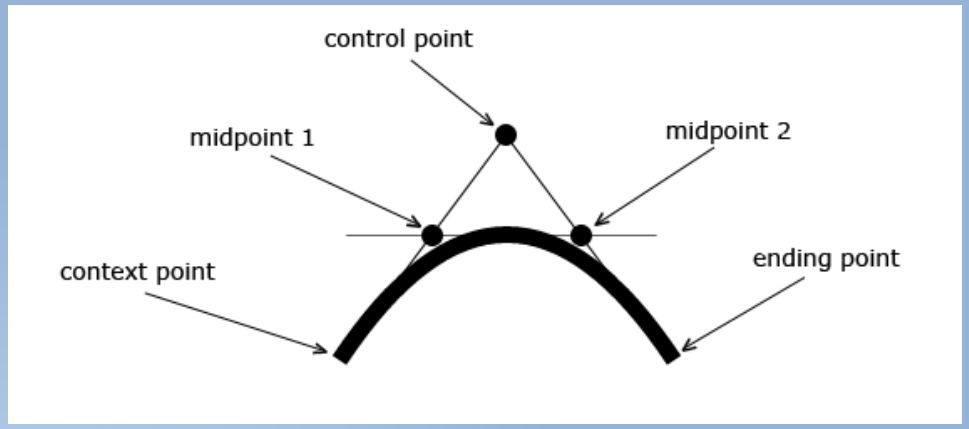
Degrees to Radians helper function

if you wish to use 0-360 degrees instead of radians.




```
function degtoRad(deg)
{
	var rad = deg * Math.PI/180;
	return rad;
}
```

Quadratic curve



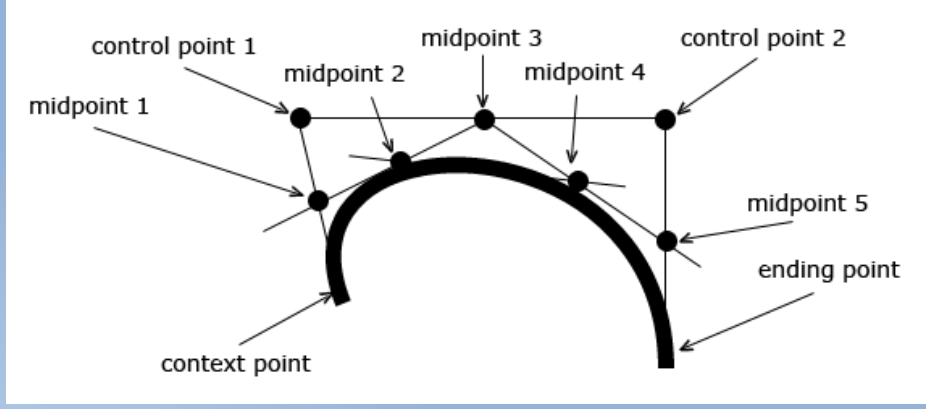
QuadraticCurveTo() draws a curve that is defined between a context point, a control point and and endPoint

Args 1,2 are the controlPoint coordinates

Args 3,4 are the endPoints.

```
context.beginPath();
context.moveTo(188, 150);
context.quadraticCurveTo(288, 0, 388, 150);
context.lineWidth = 10;
```

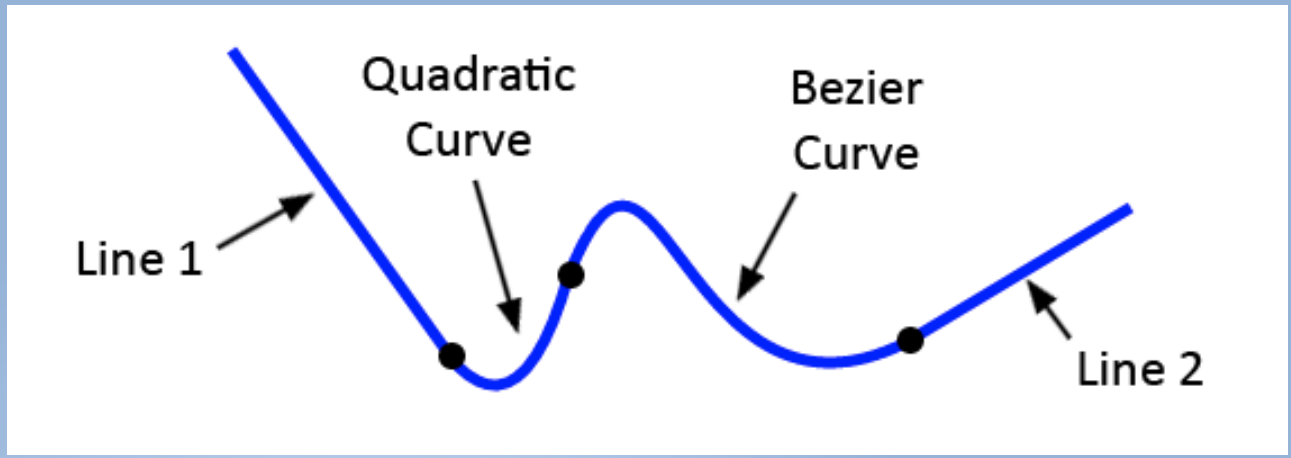
Bezier Curve



- bezierCurveTo() draws a Bezier curve (similar to curves in Illustrator)
- BezierCurves have 2 controlPoints allowing for more complex curved shapes.
- Args 1,2 are coordinates of controlPoint 1
- Args 3,4 are coordinates of controlPoint 2
- Args 5,6 are coordinates of endPoint

```
context.beginPath();
context.moveTo(188, 130);
context.bezierCurveTo(140, 10, 388, 10, 388,
170);
context.lineWidth = 10;
```

Path



- Paths can consist of multiple (joined) subpaths.
- The end point of the previous path becomes the starting point of the next.

```
context.beginPath();
context.moveTo(100, 20);
context.lineTo(200, 160); //line 1
context.quadraticCurveTo(230, 200, 250, 120);
// quadratic curve
context.bezierCurveTo(290, -40, 300, 200, 400, 150);
// Bezier curve
context.lineTo(500, 90); // line 2
```

Custom Shapes

Closed shapes will have the same start and end-point

After completing the path use closePath()

```
context.beginPath();
context.moveTo(170, 80);
context.bezierCurveTo(130, 100, 130, 150, 230, 150);
context.bezierCurveTo(250, 180, 320, 180, 340, 150);
context.bezierCurveTo(420, 150, 420, 120, 390, 100);
context.bezierCurveTo(430, 40, 370, 30, 340, 50);
context.bezierCurveTo(320, 5, 250, 20, 250, 50);
context.bezierCurveTo(200, 5, 150, 20, 170, 80);

// complete custom shape
context.closePath();
```

Rectangles

Rect method accepts x,y and width and height parameters

x,y refer to top left corner.

```
context.beginPath();
context.rect(188, 50, 200, 100);
```

Circles

Use the arc() method to draw a full circle

startAngle: 0 radians (3 o'clock)

endAngle: 2* Math.PI

```
 var centerX = canvas.width / 2;
 var centerY = canvas.height / 2;
 var radius = 70;
 context.arc(centerX, centerY, radius, 0, 2 *
Math.PI, false);
//alternatively
 context.arc(centerX, centerY, radius, degtoRad(0),
degtoRad(360), false);
```

Semi Circle

Semi circle has an endAngle that is the sum of startAngle + PI.

```
context.beginPath();
context.arc(288, 75, 70, 0, Math.PI, false);
context.closePath();
```

Color Fill

To fill with a solid color use fillStyle(#hex);

And the fill() method;

```
 context.fillStyle =
'#8ED6FF';
 context.fill();
 context.strokeStyle =
'blue';
 context.stroke();
```

Colours

Different ways of describing colours in JS.

```
// these all set the fillStyle to 'orange'
ctx.fillStyle = "orange";
ctx.fillStyle = "#FFA500";
ctx.fillStyle = "rgb(255,165,0)";
ctx.fillStyle = "rgba(255,165,0,1)";
```

Gradients

Styles can also be gradients.

Two types:
- linear gradients
- radial gradients

Linear gradient example:

```
var gradient = ctx.createLinearGradient(
    50, 50, 250, 400
);
gradient.addColorStop(0, "red");
gradient.addColorStop(0.5, "green");
gradient.addColorStop(1.0, "blue");

ctx.fillStyle = gradient;
```

Fill: Radial Gradient

createRadialGradient(x1,y1,d1,x2,y2,d2);

Radial Gradients are created

```
var grd = context.createRadialGradient(238, 50, 10, 238, 50, 300);
grd.addColorStop(0, '#8ED6FF');
grd.addColorStop(1, '#004CB3');
context.fillStyle = grd;
context.fill();
```

Radial Gradients

```
// create radial gradient with
// inner circle at (200, 200) radius 25
// and outer circle at (200, 200) radius 150
      var gradient = ctx.createRadialGradient(
    200, 200, 25, 200, 200, 150
);

// add some color stops
      gradient.addColorStop(0, "white");
      gradient.addColorStop(0.7, "yellow");
      gradient.addColorStop(1, "orange");

      ctx.fillStyle = gradient;
```

Fill: Pattern

createPattern() method of the canvas context returns a pattern object

Requires an image object

Need to define the repeat opiton (repeat, repeat-x, repeat-y, no-repeat.)

fillStyle gets set to the pattern

```
var imageObj = new Image();
imageObj.src = ‘pattern.gif’;
var pattern = context.createPattern(imageObj, 'repeat');
context.fillStyle = pattern;
context.fill();
```

Clipping Paths

Use clipping paths to mask an area so only that area is affected when drawing:

```
// create a star-shaped clipping path
ctx.beginPath();
ctx.moveTo(270,200);
for (var i=0;i<=20;i++) {
    ctx.lineTo(
        200 + Math.cos(i/10 * Math.PI) * 70 * (i%2+1),
        200 + Math.sin(i/10 * Math.PI) * 70 * (i%2+1)
);
}
ctx.clip();

// (try to) fill the entire canvas with a light salmon
ctx.fillStyle = "lightsalmon";
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

Clipping Paths

Compositing Effects

There are two settings that affect how new content is composited with the existing:

- `ctx.globalAlpha`
- `ctx.globalCompositeOperation`

`ctx.globalAlpha`

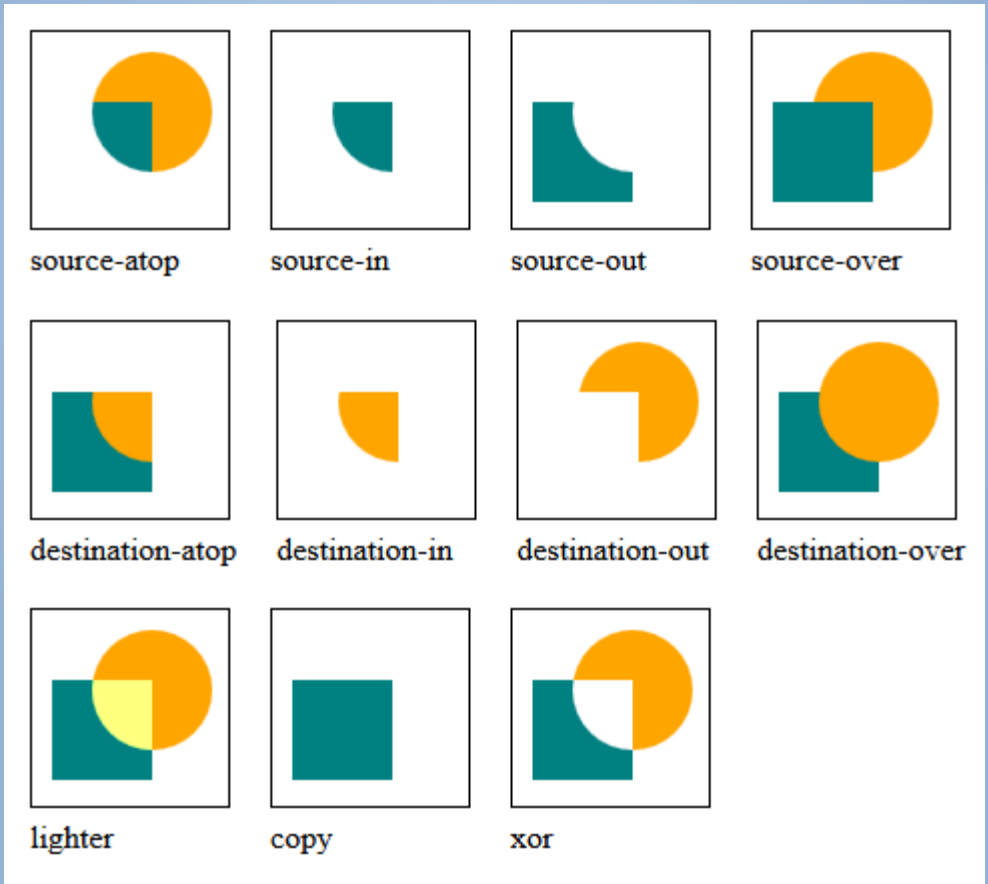
Affects the transparency of the added shapes.

`ctx.globalCompositeOperation`

Applies various operations to change the result.

Compositing Effects

Drawing a teal square on top of an orange circle with different compositing ops (default op is `source-over`):



The State Stack

Values such as the fill and stroke style are part of the current context state.

The current state can be saved pushed on to a stack by calling `ctx.save()`.

Calling `ctx.restore()` replaces the current state with the top state on the stack.

This allows you to make temporary changes to the state:

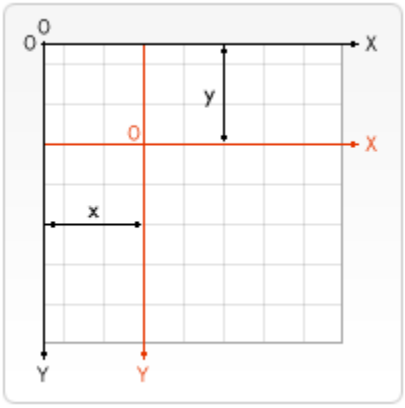
```
ctx.fillStyle = "red"; // fillStyle is now red

ctx.save(); // save current state

ctx.fillStyle = "blue"; // fillStyle is now blue

ctx.restore(); // fillStyle
```

Transforms > Save /Restore

- Transforms allow to move, rotate and scale graphics on the canvas
- Transform operations affect the whole canvas coordinate system.
- Each time you call transform(), it builds on the previous transformation matrix

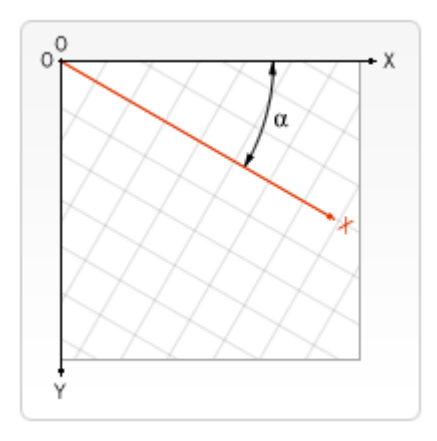
Transform > Translate

translate(*x*, *y*) moves the canvas and its origin on the grid




```
for (var i=0;i<3;i++) {
      for (var j=0;j<3;j++) {
      //save state
      ctx.save();
      ctx.translate(15+j*100, 15
+i*100);
      //spool crop at 0,0
      ctx.drawImage(myImage,0 ,0,
150,170,0,0, 50, 65);
      //restore previous canvas state
      ctx.restore();
      }
}
```

Transform > Rotate



rotate(*angle*)Rotates the canvas clockwise around the current origin by the angle number of radians.

```
for (var i=0;i<10;i++) {
	ctx.save();

	ctx.translate(canvas.width/2, canvas.height/2);
	var degree = degToRad(i*36);
	ctx.rotate(degree);
	//rectangle
	ctx.fillRect(0,75,50,50);
	//spool
	//ctx.drawImage(myImage,0 ,0, 150,170,0,75, 50,
65);
	ctx.restore();
}
```

Transform > Scale

increases or decreases the units in our canvas grid.

used to draw scaled down or enlarged shapes

Ctx.scale(x,y);

```
for (var i=0;i<3;i++) {
      for (var j=0;j<3;j++) {
            ctx.save();
            //spool crop
            ctx.translate(j*100, i*100);
            ctx.scale(i*0.25+.1,j*0.25+.1);
            ctx.drawImage(myImage,0 ,0, 150,170,0,0,
150, 170);
            ctx.restore();
      }
}
```

The Transformation Matrix

All content that is drawn on the canvas is transformed by a transformation matrix.

Initially, the transformation matrix is set to the identity matrix, so coordinates are unchanged:

```
1 0 0
0 1 0
0 0 1
```

Changing the transformation matrix:

```
ctx.setTransform(a, b, c, d, e, f);
// sets the matrix to:
a c e
b d f
0 0 1
```

Transformations

Transforming:

```
ctx.transform(a, b, c, d, e, f);
// multiplies with the matrix
   a c e
   b d f
   0 0 1
```

3.The matrix has 9 numbers of which last row always remains **[0 0 1]**.Other parameters are as follows:

a :To **scale** the object across X axis.

b :To **skew** the object **horizontally**(i.e horizontal shear).

c :To **skew** the object **vertically**(i.e vertical shear).

d :To **scale** the object across Y axis.

e :To **translate** the object across X axis.

f :To **translate** the object across Y axis.

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Compositing Effects

There are two settings that affect how new content is composited with the existing:

- `ctx.globalAlpha`
- `ctx.globalCompositeOperation`

`ctx.globalAlpha`

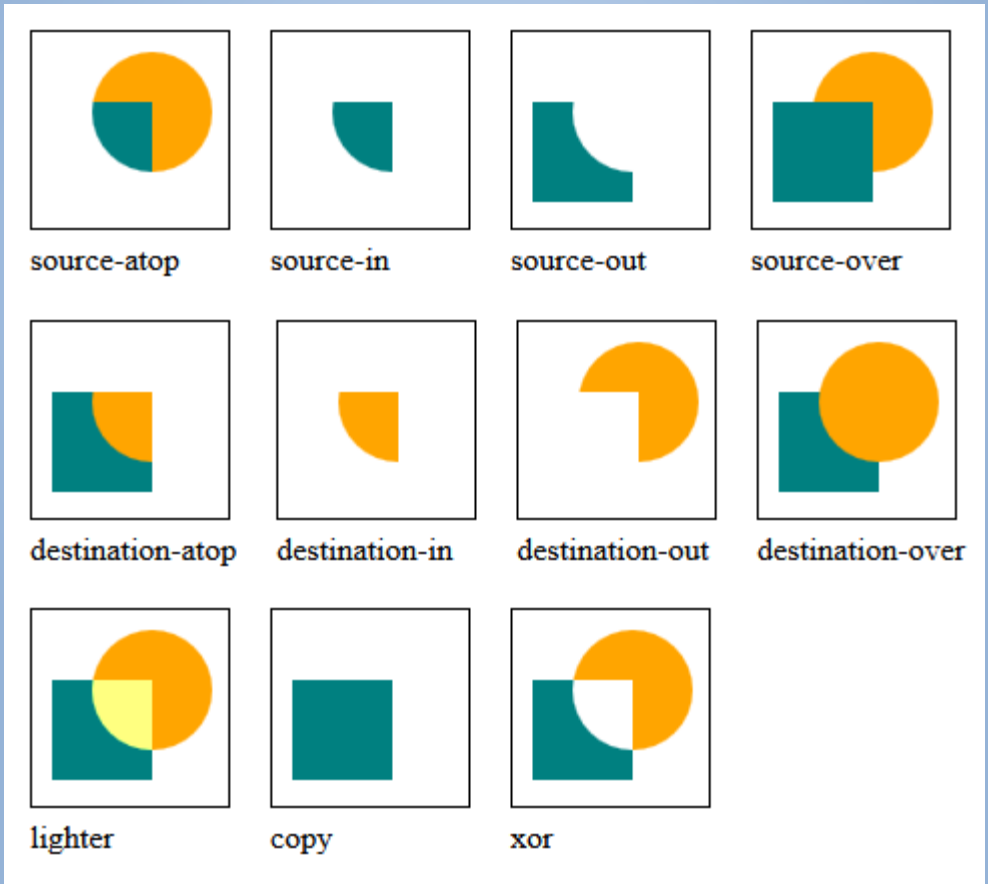
Affects the transparency of the added shapes.

`ctx.globalCompositeOperation`

Applies various operations to change the result.

Compositing Effects

Drawing a teal square on top of an orange circle with different compositing ops (default op is `source-over`):



Text Alignment

Changing text alignment:

```
ctx.textAlign = "center";
```

Accepted values:
- center
- left
- right
- start
- end

The meaning of the values “start” and “end” depend on the locale, e.g. “start” means “left” for left-to-right locales.

Text Baseline

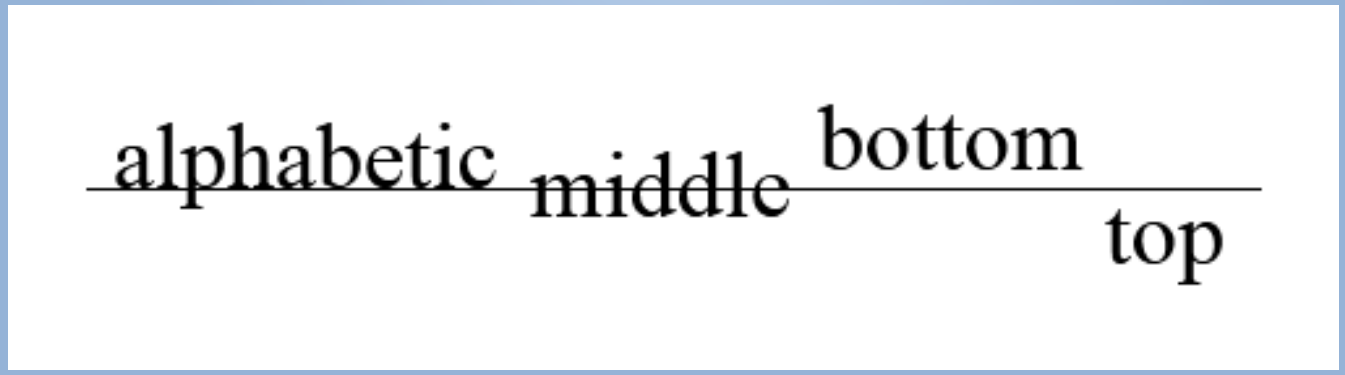
Changing the text baseline:

```
ctx.textBaseline = "middle";
```

Accepted values:

- top
- middle
- bottom
- hanging
- alphabetic
- ideographic

“hanging” and “ideographic” are not well supported.



Text > Font, Size, Style

Set the font property of the canvas

Use the filltext() and stroketext() Method to draw the font

```
 context.font = 'italic 40pt Calibri';
 context.fillText('IADT', 150, 100);
context.strokeText("What's up? ", 150, 150);
```

Images

Obtain a reference to existing images on the same page as the canvas

- document.images collection
- document.getElementsByTagName() method
- If you know the ID use document.getElementById() to retrieve that specific image

Create an Image from scratch in JS:

```
var img = new Image(); // Create new img element
img.src = 'myImage.png'; // Set source path
```

Images > load event

- Image needs to be loaded before it can be used.\
- Otherwise JS will throw an exception
- Monitor the **load** event

```
var img = new Image();

img.onload =  function() {
// execute drawImage statements here
}, false);

 img.src = 'myImage.png';
```

Images > Data URLs

Images can be loaded from files

- formats can be: gif, jpeg, png

But also via the data:url schema.

Base64 encoding of string characters to represent image data.

```
myImage.src =
"
4ge4SUNDX1BST0ZJTEUAAQEAAAeoYXBwb…… etc.
```

Reduces amount of HTTP requests on server

Less efficient than img compression.

Use for small images (<4K only);

Images

Can be added to the canvas

1. Create an Image Object
2. Wait until Image has loaded
3. If loaded it can be used inside the 2d context.
4. Images must be completely loaded before drawing them to a canvas

```
var imageObj = new Image();
imageObje.src = 'path';
imageObj.onload = function() {
      Context.drawImage(
      imageObj, 50, 50);
}
```

Images: > Resize

Images can be resized on the canvas by providing

`Width, height`

Arguments.

```
var imageObj = new Image();
imageObje.src = ‘path’;
imageObj.onload = function()
{
      Context.drawImage(
      imageObj, 50, 50,
      25,25);
}
```

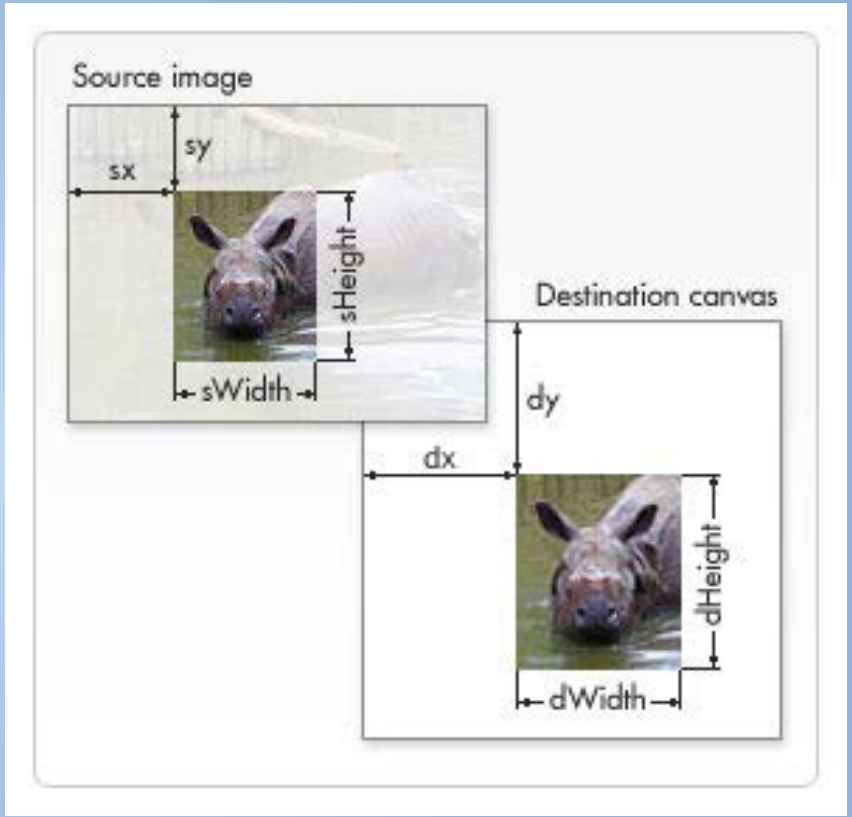
Images > Crop

Only a portion of a loaded image can be displayed by proiding the following arguments

Drawing only part of an image:

```
ctx.drawImage(img,
    sx, sy, sw, sh,
    dx, dy, dw, dh
);
```

Takes the rectangle (sx, sy) to (sx+sw, sy+sh) from img and draws it on the canvas, scaled to fit the rectangle (dx, dy) to (dx+dw, dy+dh).



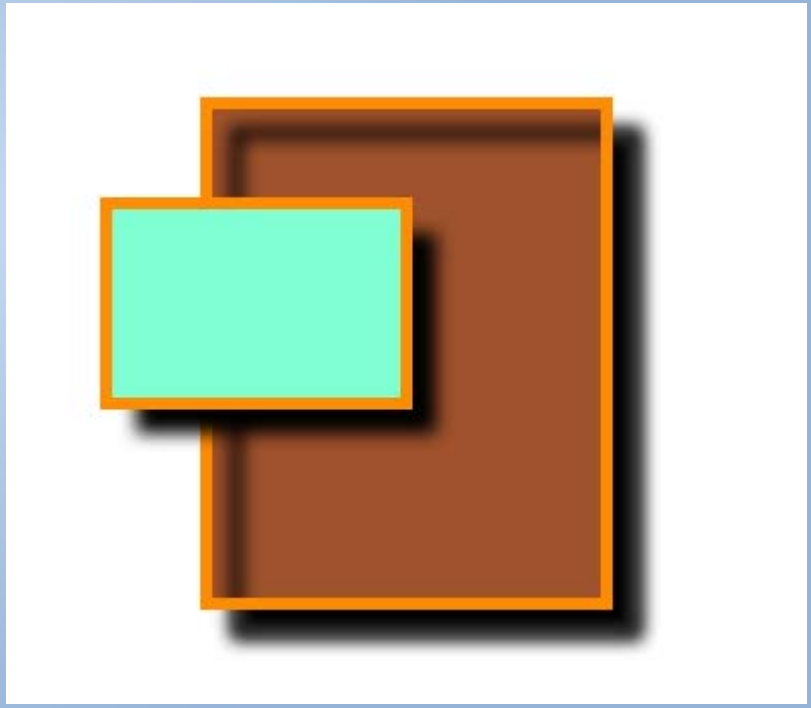
Images > Crop

```
var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');
      var imageObj = new Image();

      imageObj.onload = function() {
        // draw cropped image
        var sourceX = 150;
        var sourceY = 0;
        var sourceWidth = 150;
        var sourceHeight = 150;
        var destWidth = sourceWidth;
        var destHeight = sourceHeight;
        var destX = canvas.width / 2 - destWidth / 2;
        var destY = canvas.height / 2 - destHeight / 2;

        context.drawImage(imageObj, sourceX, sourceY,
sourceWidth, sourceHeight, destX, destY, destWidth, destHeight);
      };
      imageObj.src = ‘path';
```

Effects: Shadows

A shadow effect can be applied to all drawn content:

```
ctx.shadowColor = "black";
        ctx.shadowOffsetX = 15;
        ctx.shadowOffsetY = 15;
        ctx.shadowBlur = 8;
```



Watch out for shadows
on stroked paths.

Accessing Pixel Data

The canvas allows you to access the RGBA pixel data:

```
// retrieve the image data for the
// rectangle (0, 0) to (100, 100)
var imgData = ctx.getImageData(0, 0, 100, 100);
```

The object returned by `ctx.getImageData()` has the properties `width`, `height` and `data`.

The `data` property is an array of `width` * `height` * 4 values in the range [0, 255], representing the pixel data.

Each pixel uses 4 values, one for each of the red, green, blue and alpha channels.

Use `ctx.putImageData(imgData, x, y)` to write data.

Image Data Example

```
// get image data for the entire canvas
var imgData = ctx.getImageData(
    0, 0, canvas.width, canvas.height
);

// loop through all pixels
for (var x=0;x<canvas.width;x++) {
    for (var y=0;y<canvas.height;y++) {
        var p = (y * canvas.width + x) * 4;
        // invert the red, green and blue values
        imgData.data[p] = 255 - imgData.data[p];
        imgData.data[p+1] = 255 - imgData.data[p+1];
        imgData.data[p+2] = 255 - imgData.data[p+2];
    }
}

// put the image data back at (0, 0)
ctx.putImageData(imgData, 0, 0);
```

Same-Origin Policy

Canvas has security restrictions with regards to using external content.

If `ctx.drawImage()` is used with an image that originates from another domain, the canvas becomes *tainted*.

This means that access to image data with `ctx.getImageData()` is no longer allowed.

The same applies when using patterns or fonts from other domains.