HTML5 Canvas

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

Uses:

- Draw photos to the canvas
- Draw graphic shapes
- Rotate, translate shapes

Fallback Content

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 the **context** objects, created with **canvas.getContext().**

```
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.
- We will be looking at 2D contexts specifically.

Canvas

- HTML5 element
- but elements are rendered and manipualted by JS

- let's add an **id** attribute to store it as a JS object

```
<canvas id="canvas">Bla</canvas>
```

- The **canvas** is stored in a JS variable.
- The **context** is a sub-object of the canvas
- It is set, to be either 2d or 3d
- The **context** is the one that we will use for drawing operations.

```
var canvas = document.getElementById("canvas");
var context = 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;

- **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(50, 100);
context.lineTo(250, 50);
context.lineWidth = 3;
context.strokeStyle = '#ff0000';
context.stroke();
```

Example: Canvas_01_01.

CodePen: Straight Line

MMP_Canvas01_01_straightLine
A PEN BY pietschj

Run Pen

7 / 53

CodePen: Straight Line in a Loop

MMP_Canvas01_02_Lines_looped
A PEN BY pietschj

Run Pen

8 / 53

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
```

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


beginPath() & closePath()

```
ctx.beginPath()
```

a new path is being drawn

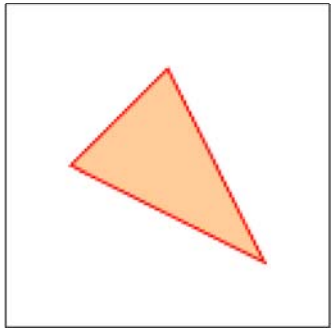
```
ctx.closePath()
```

closes an open path,

i.e. it links the last point of the shape to the startPoint

i.e. it links the last point of the shape to the startPoint

Example: A Triangle



Example: Canvas_01_02.

```
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();
```


Example: Triangle

NB

- last line does not need to be explicitly drawn
- **fill()** and **stroke()** commands are necessary to execute fill and stroke colours.

CodePen: A Triangle

MMP_Canvas01_03_triangle
A PEN BY pietschj

Run Pen

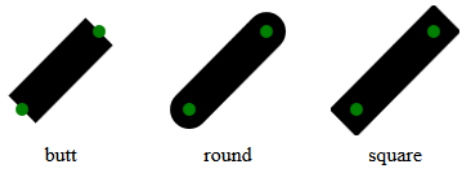
Line Caps

```
ctx.lineCap
```

changes the appearance of line endings when stroking paths:

lineCap property changes the caps of the line 3 possible styles: ‘round’, ‘butt’, ‘square’);

```
ctx.lineCap = 'round';
```

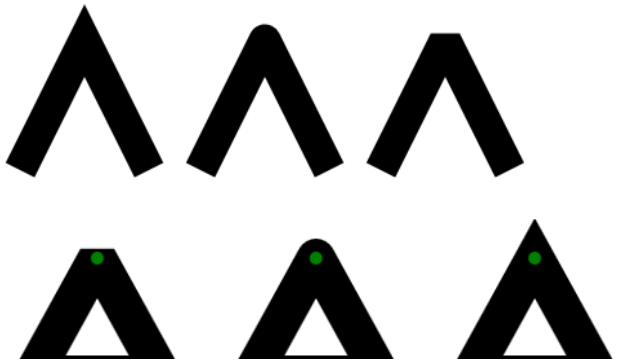


Line Joins

The **lineJoin** property defines how paths are joined together.

- Possible values are ‘miter’,’round’,’bevel’
- Default value: ‘miter’

```
ctx.lineJoin = 'miter’;
```



Rectangles

```
ctx.rect(x,y, w, h)
```

Rect method accepts x,y and width and height parameters

- x,y refers to the rectangle's top left corner.

```
ctx.rect(188, 50, 200, 100);
```

Arcs

There is no circle, ellipse method.

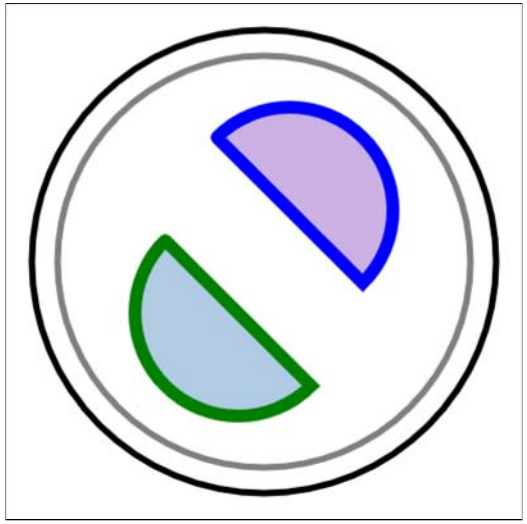
How to draw circles

```
ctx.arc(centerX, centerY, startAngle, endAngle in radians, [drawing direction])
```

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

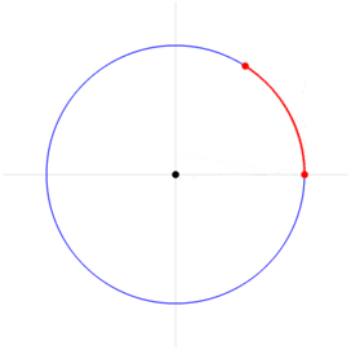
Arcs



Radians

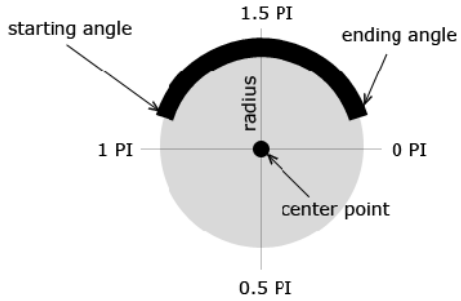
What is a Radian?

The length of an arc of a unit circle is numerically equal to the measurement in radians of the angle that it subtends; one radian is just under 57.3 degrees



Radians Degs

Radians 2



Radians	**Degs**
PI/2	90
PI	180
PI	270
+ PI/2	
2 x PI	360

Degrees to Radians helper function

If you wish to use familiar 0-360 degrees instead of radians. use this handy conversion function:

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

Circles

```
ctx.arc()
```

The following angles draw a full circle

- **startAngle:** 0 radians (starts at 3 o'clock)
- **endAngle:** 2* Math.PI

```
var centerX = canvas.width / 2;
var centerY = canvas.height / 2;
var radius = 70;
var startAngle = 0
var endAngle = 2 * Math.PI;
ctx.arc(centerX, centerY, radius, startAngle, endAngle, false);
//alternatively with Radian conversion option
//ctx.arc(centerX, centerY, radius, degtoRad(0), degtoRad(360), false);
```

Semi Circle

Semi circles have an endAngle that is the sum of startAngle + PI.

```
ctx.beginPath();
var startAngle = 0
var endAngle = startAngle + Math.PI;
ctx.arc(288, 75, 70, startAngle, endAngle, false);
ctx.closePath();
```

CodePen: Arcs

MMP_Canvas01_04_arcs
A PEN BY pietschj

Run Pen

24 / 53

CodePen: Bullseye

MMP_Canvas01_05_bullsEye
A PEN BY pietschj

Run Pen

25 / 53

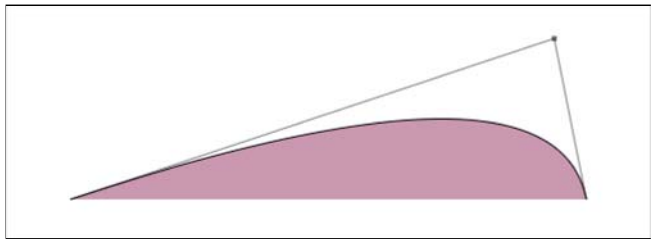
Quadratic curve

```
ctx.quadraticCurveTo()
```

draws a curve that is defined between a previous ctx point, a control point and and endPoint

- Args 1,2 are the controlPoint coordinates
- Args 3,4 are the endPoints.

```
ctx.beginPath();
ctx.moveTo(50,150); //last ctx point
ctx.quadraticCurveTo(425,25,450,150);
ctx.lineWidth = 10;
```



CodePen: Quadratic/Cubic Curve

MMP_Canvas01_06_cubicCurve
A PEN BY pietschj

Run Pen

27 / 53

Bezier Curve

also called a Cubic Curve

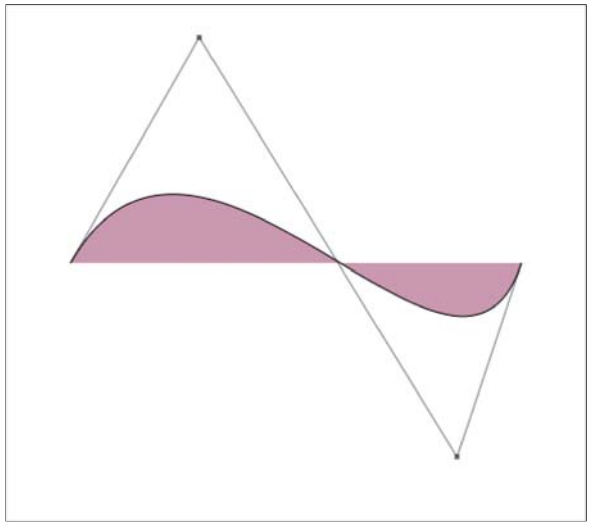
```
ctx.bezierCurveTo()
```

draws a Bezier curve that is defined between a previous ctx point, 2 control points and an endPoint (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

```
ctx.beginPath();
ctx.moveTo(50,200); //last ctx point
ctx.bezierCurveTo(150,25,350,350,400,200);
ctx.lineWidth = 10;
```


Bezier curve



CodePen: Bezier Curve

MMP_Canvas01_07_BezierCurve
A PEN BY pietschj

Run Pen

30 / 53

Custom Shapes

- Closed shapes will have the same start and end-point
- After completing the shape use **closePath()**

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


CodePen: Complex Path

MMP_Canvas01_XX

A PEN BY pietschj

Run Pen

32 / 53

Color Fill

- To fill with a solid color use **fillStyle(#hex)** ;
- And the fill() method as the directive to fill the shape;

```
//some shape
ctx.fillStyle = '#8ED6FF';
```

```
ctx.fill();
//some other shape
ctx.strokeStyle = 'blue';
ctx.stroke();
```


Colours

Different ways of expressing colour 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)";
```

*rgba also allows you to set an alpha value for a fill or stroke.

Gradients

Styles can also be gradients. Two types:

- linear gradients
- radial gradients

Linear gradient

```
ctx.createLinearGradient(x1,y1,x2,y2)
```

defines the slope of the Gradient

```
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; // apply this as a fill.
```

Fill: Radial Gradient

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

Radial Gradients are created between inner and outer circle.

- 3 arguments per circle: x,y of centerpoint and diameter.

```
var grd = ctx.createRadialGradient(238, 50, 10, 238, 50, 300);
grd.addColorStop(0, '#8ED6FF');
grd.addColorStop(1, '#004CB3');
ctx.fillStyle = grd;
ctx.fill(); // apply this as a 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;
```

38 / 53

CodePen: Gradients

MMP_Canvas01_09_Gradients
A PEN BY pietschj

Run Pen

Fill: Pattern

```
ctx.createPattern()
```

returns a pattern object

- Requires an image object
- Set the repeat option (repeat, repeat-x, repeat-y, no-repeat.)
- fillStyle gets set to the pattern

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

40 / 53

CodePen: Patterns

MMP_Canvas01_10_Patterns
A PEN BY pietschj

Run Pen

41 / 53

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);
```

CodePen: Clipping Paths

MMP_Canvas01_11_ClippingPath
A PEN BY pietschj

Run Pen

Text > Font, Size, Style

Set the font property of the canvas Use the filltext() and stroketext() Method to draw the font

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

MMP_Canvas01_12_fillText
A PEN BY pietschj

Run Pen

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

```
var img = new Image(); // Create new img element
img.src = 'myImage.png'; // Set source path
img.addEventListener("load", function() {
  //do something with the image here
}
```

Create an Image Obj from scratch in JS:

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.
- Reduces amount of HTTP requests on server
- Less efficient than img compression.
- Use for small images (<4K only);

```
myImage.src = "
JRgABAQEASABIAAD/4g
e4SUNDX1BST0ZJTEUAA
QEAAAeoYXBwb..... etc."
```

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 ctx.
4. Images must be completely loaded before drawing them to a canvas

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

MMP_Canvas01_13_Images

A PEN BY pietschj

Run Pen

49 / 53

Images: > Resize

- Images can be resized on the canvas by providing Width, height, Arguments.

```
var imageObj = new Image();
imageObje.src = ‘path’;
imageObj.onload = function() {
ctx.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;

        ctx.drawImage(imageObj, sourceX, sourceY, sourceWidth, sourceHeight, destX
      };
      imageObj.src = ‘path';
```

53 / 53