Sebastian Schaum | crosscan GmbH

Eine Leinwand für den Browser

About Me

Sebastian Schaum

- Software developer at crosscan
- PHP & JS professional since 2008
- „Pottkind“
- @schaumiii

About You?

Motivation

Motivation

- Canvas Usage: From small animations or drawings to a game
- Pluginless drawings and animations inside the Browser
- It is fun to work with

Canvas - What’s that?

<canvas>

<canvas>

- HTML5-Element
- Fixed size
- Used for drawing with JS
  - like Charts, Photo Composition, even Video rendering

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
</head>
<body>
    <canvas
      width="800"
      height="600"></canvas>
</body>
</html>
```

Canvas Contexts

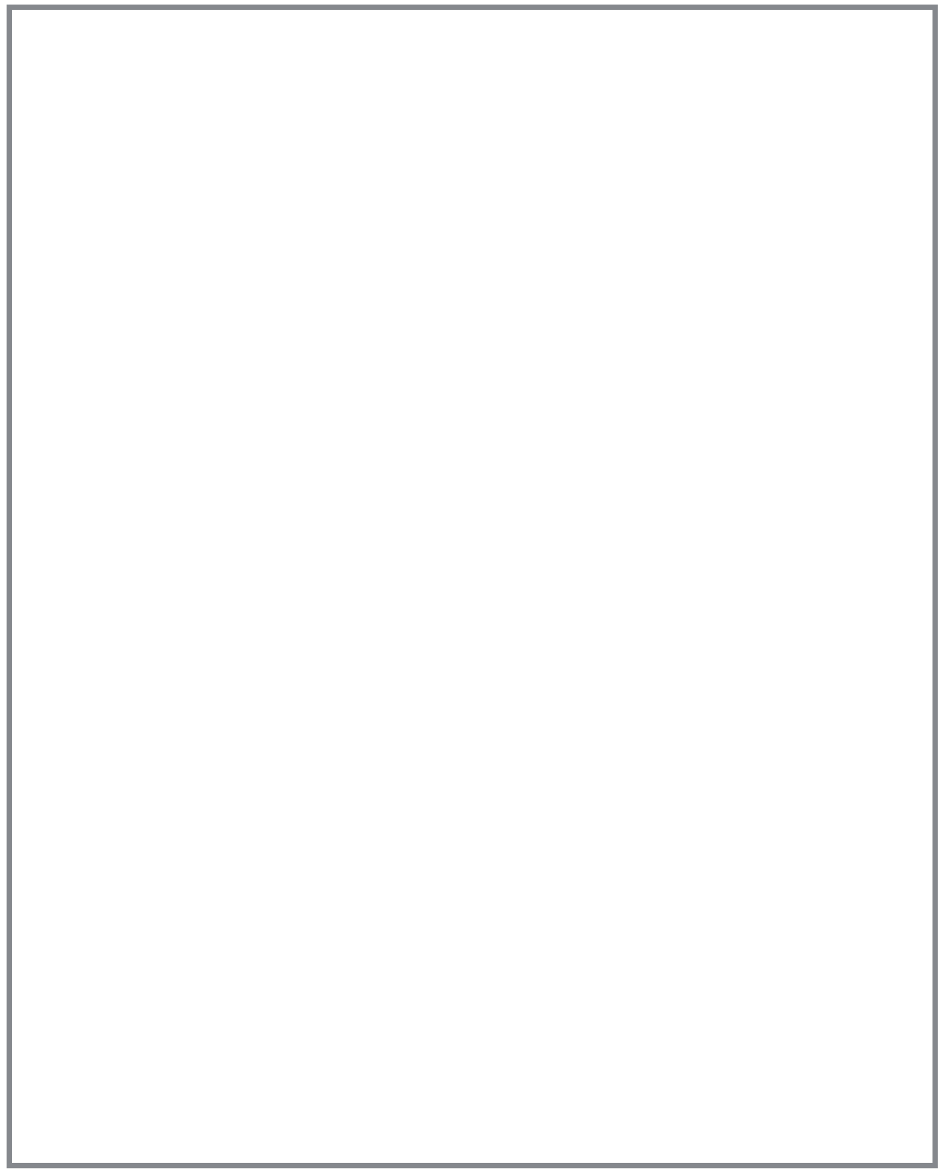
Canvas Contexts

- Rendering context
- Used to create or manipulate content
- Available contexts
  - 2D
  - 3D using WebGL

```
var canvas =
 document.createElement('canvas');
var context =
 canvas.getContext('2d');

var webGlContext =
 canvas.getContext('webgl');
```

Basic drawing

- How about drawing?
  - Simple rectangle



Basic drawing

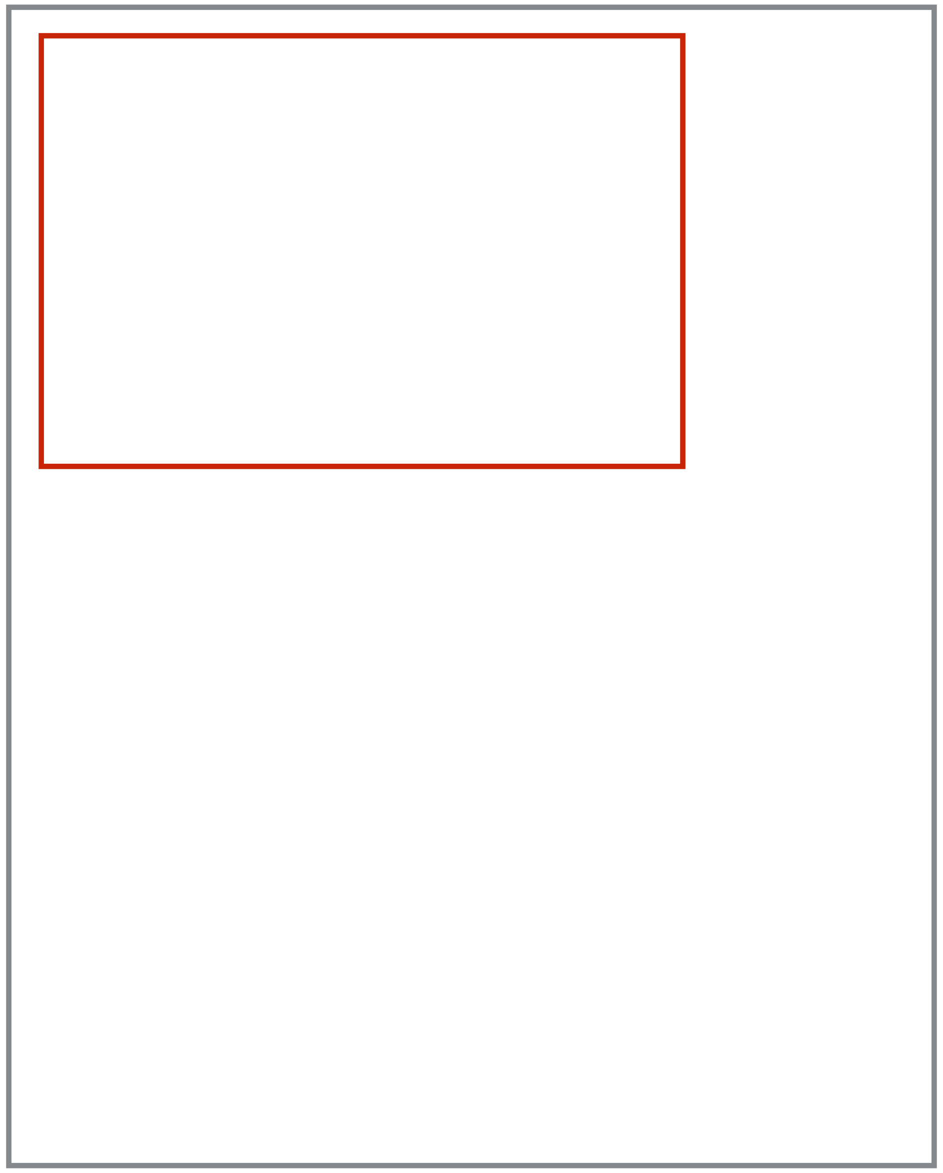
- How about drawing?
  - Simple rectangle

```
var context =
 canvas.getContext('2d');

context.strokeStyle = '#FF0000';
context.strokeRect(
  50, 50, 400, 300
);
```

Basic drawing

- How about drawing?
  - Simple rectangle

```
var context =
 canvas.getContext('2d');

context.strokeStyle = '#FF0000';
context.strokeRect(
  50, 50, 400, 300
);
```

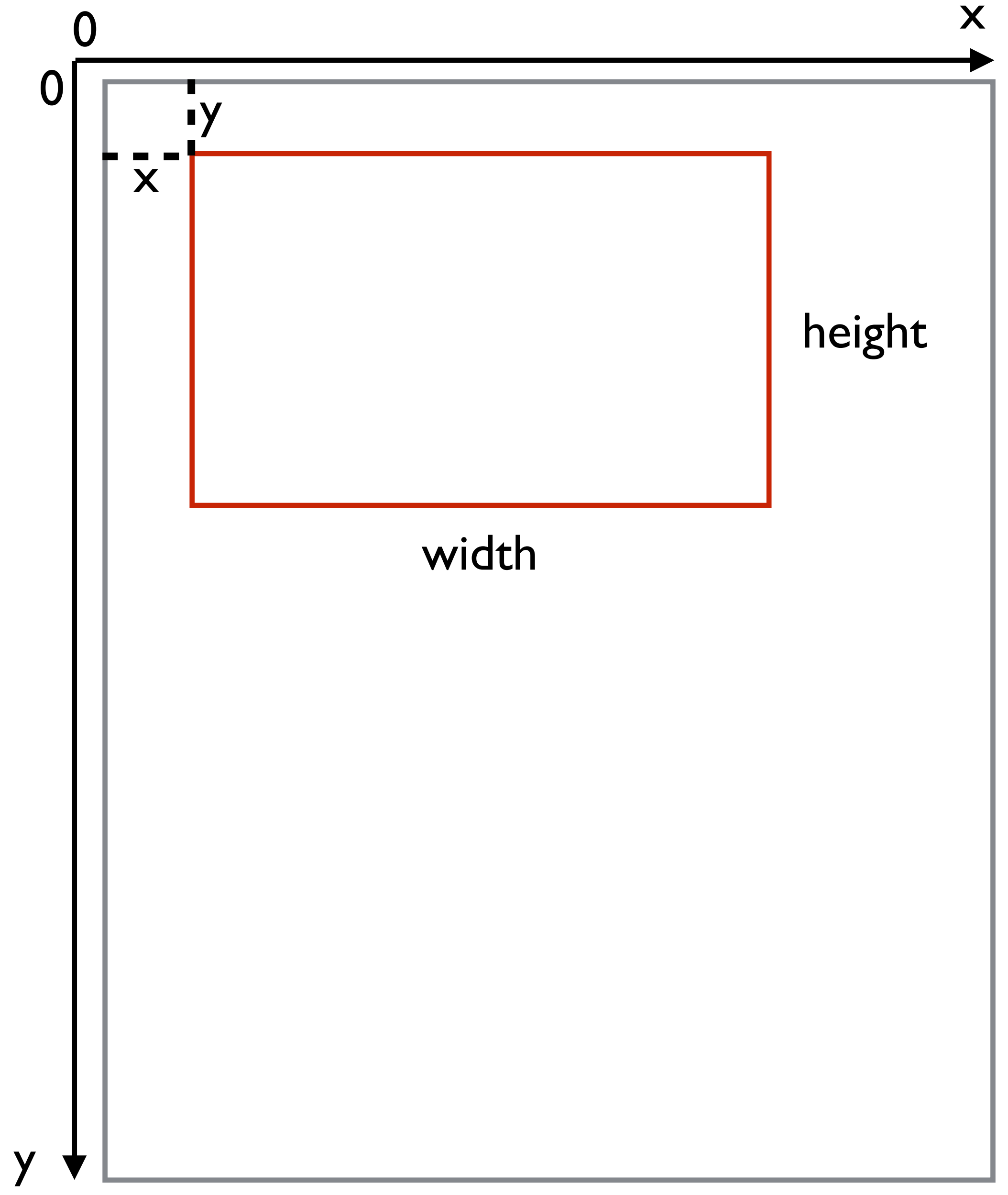


Canvas Geometry

- Understanding canvas coordinates
- Origin at the top left corner by default

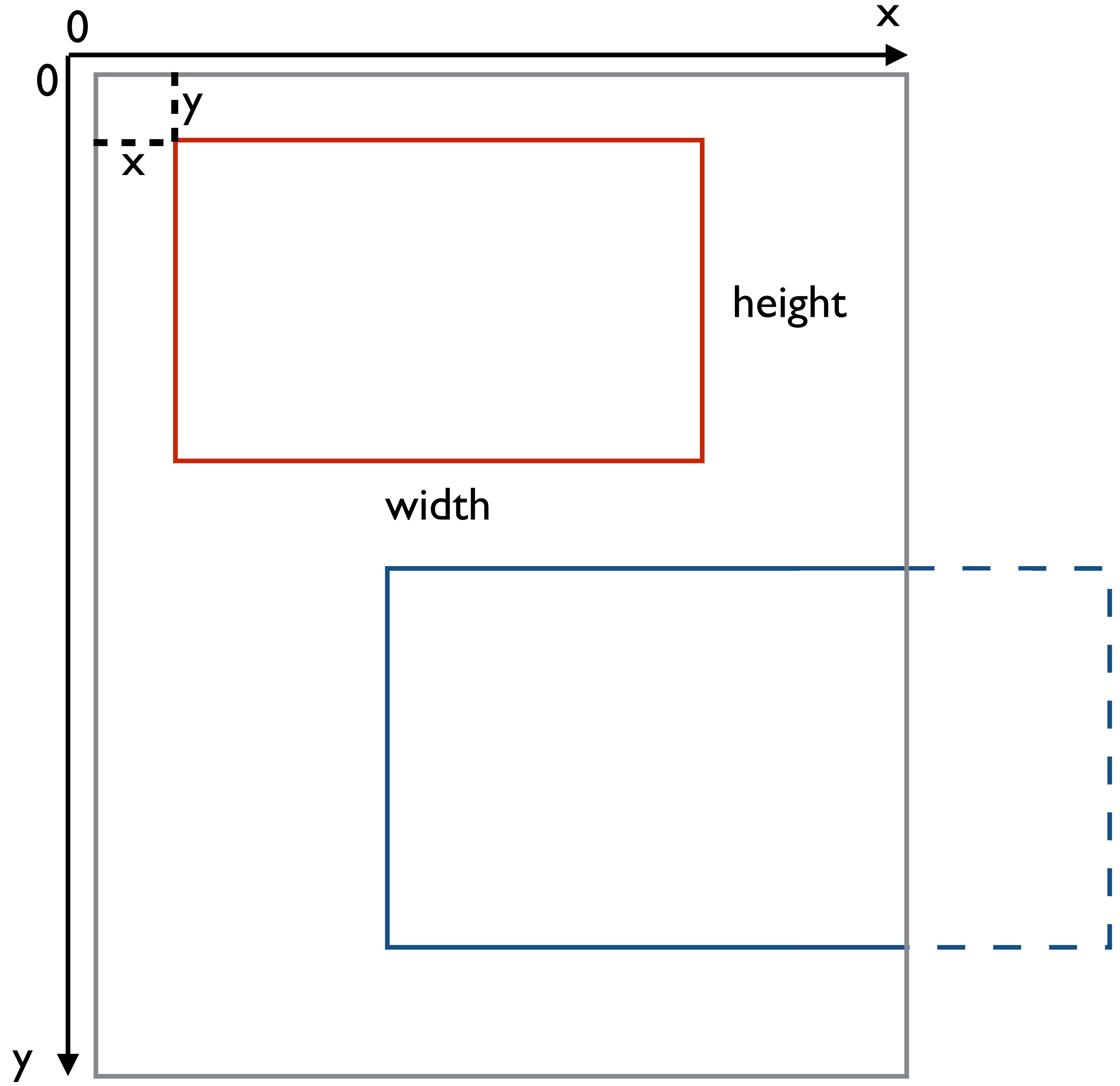
Canvas Geometry

- Understanding canvas coordinates
- Origin at the top left corner by default



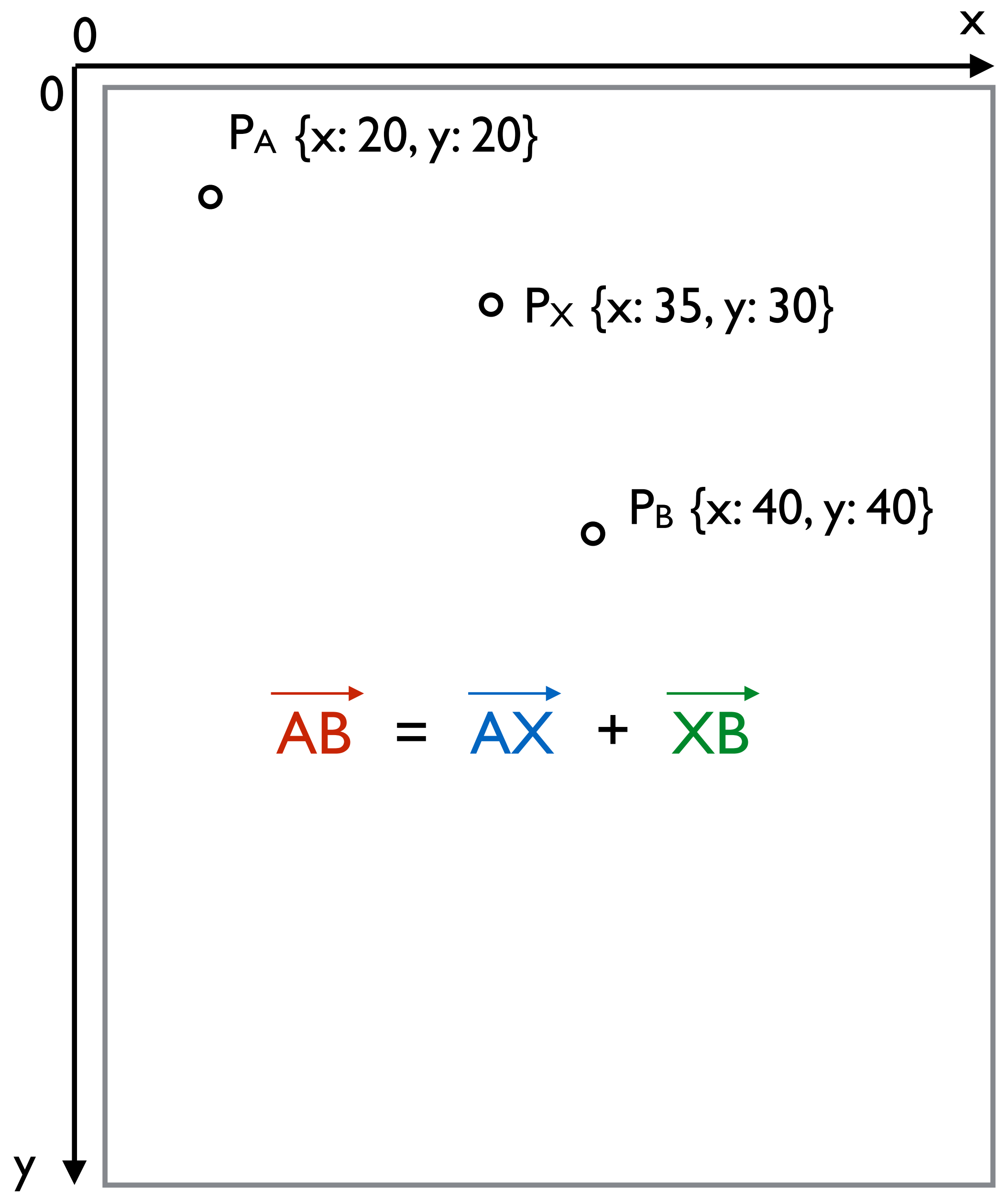
Canvas Geometry

- Understanding canvas coordinates
- Origin at the top left corner by default
- Context sizes may vary from Canvas size



Vector Geometry

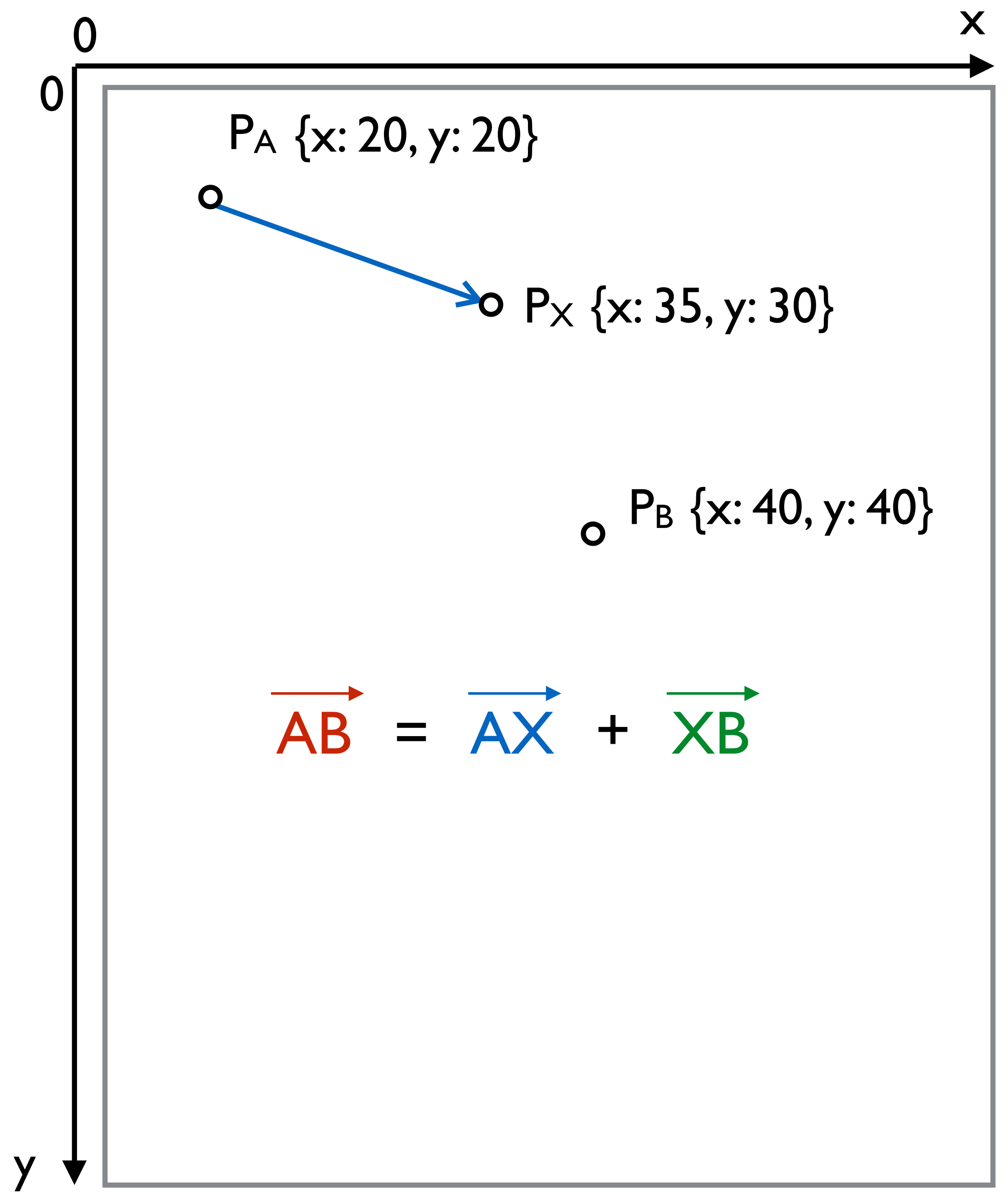
- Description of length and direction
- … from one point to another



0
0
x
y
P_A {x: 20, y: 20}
P_X {x: 35, y: 30}
P_B {x: 40, y: 40}
$\overrightarrow{AB} = \overrightarrow{AX} + \overrightarrow{XB}$

Vector Geometry

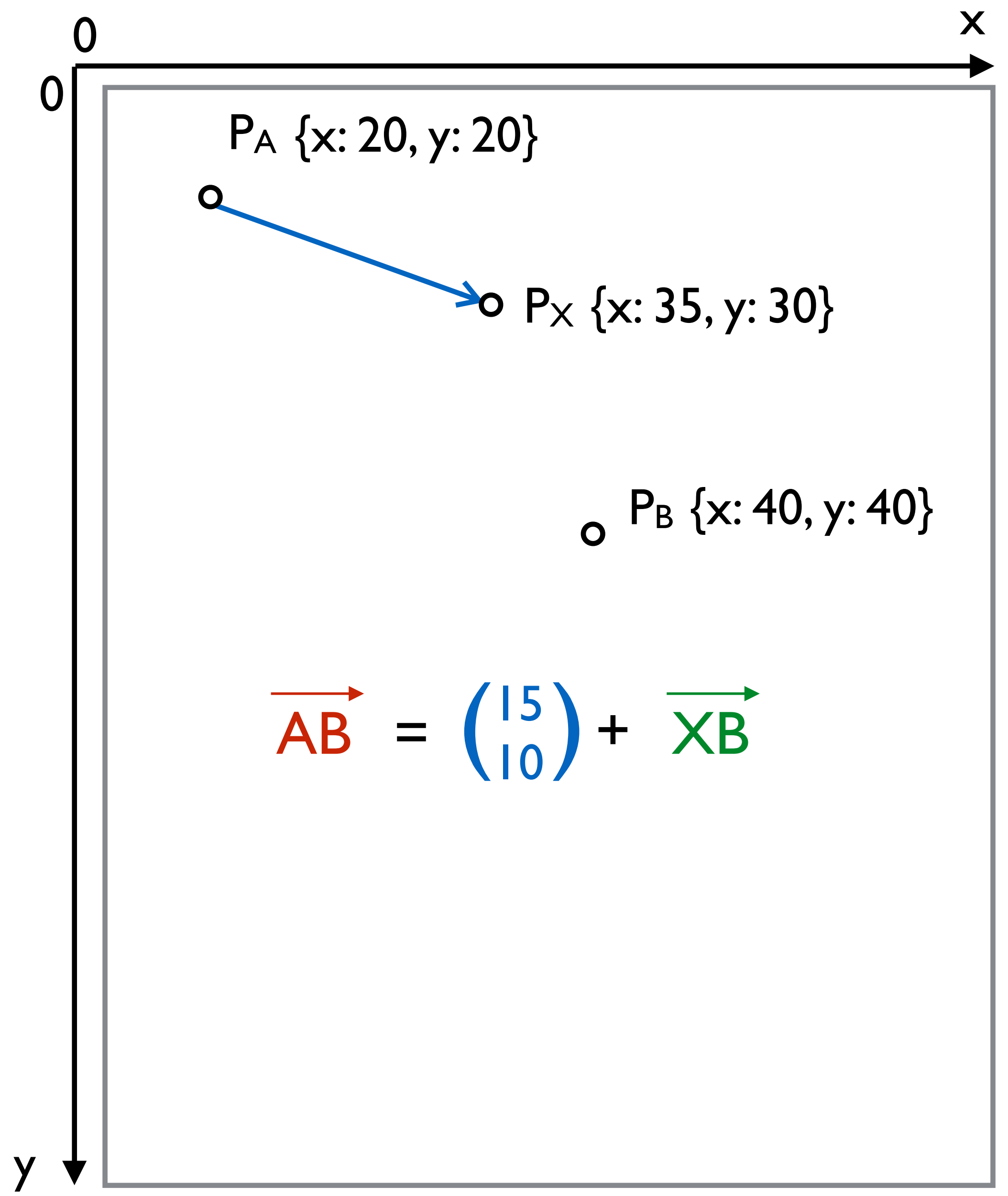
- Description of length and direction
- … from one point to another



$$\overrightarrow{AB} = \overrightarrow{AX} + \overrightarrow{XB}$$

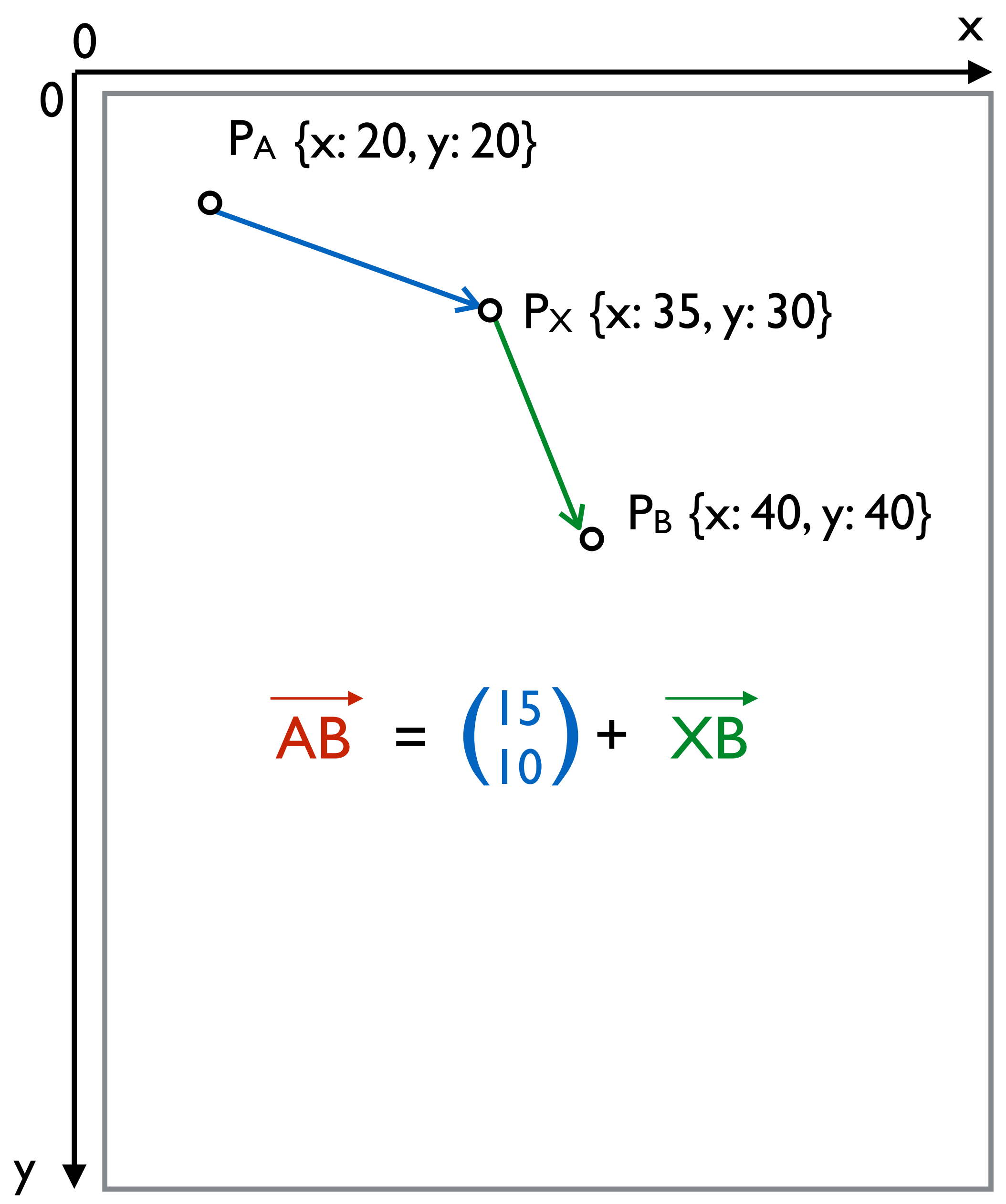
Vector Geometry

- Description of length and direction
- … from one point to another


0
x
0
y
P_A {x: 20, y: 20}
P_X {x: 35, y: 30}
P_B {x: 40, y: 40}
$\overrightarrow{AB} = \binom{15}{10} + \overrightarrow{XB}$

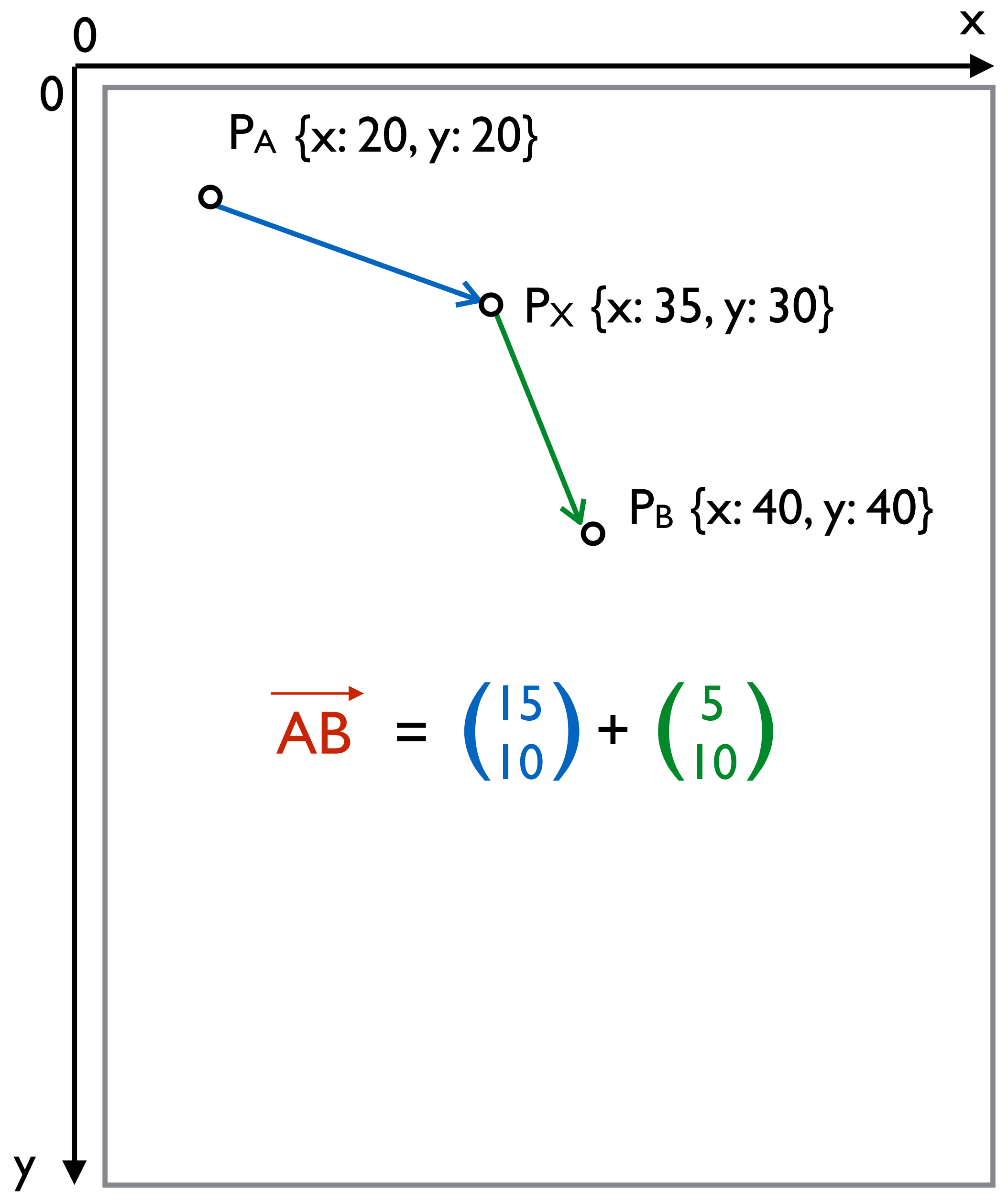
Vector Geometry

- Description of length and direction
- … from one point to another


0
x
0
P_A {x: 20, y: 20}
P_X {x: 35, y: 30}
P_B {x: 40, y: 40}
$\overrightarrow{AB} = \begin{pmatrix} 15 \\ 10 \end{pmatrix} + \overrightarrow{XB}$
y

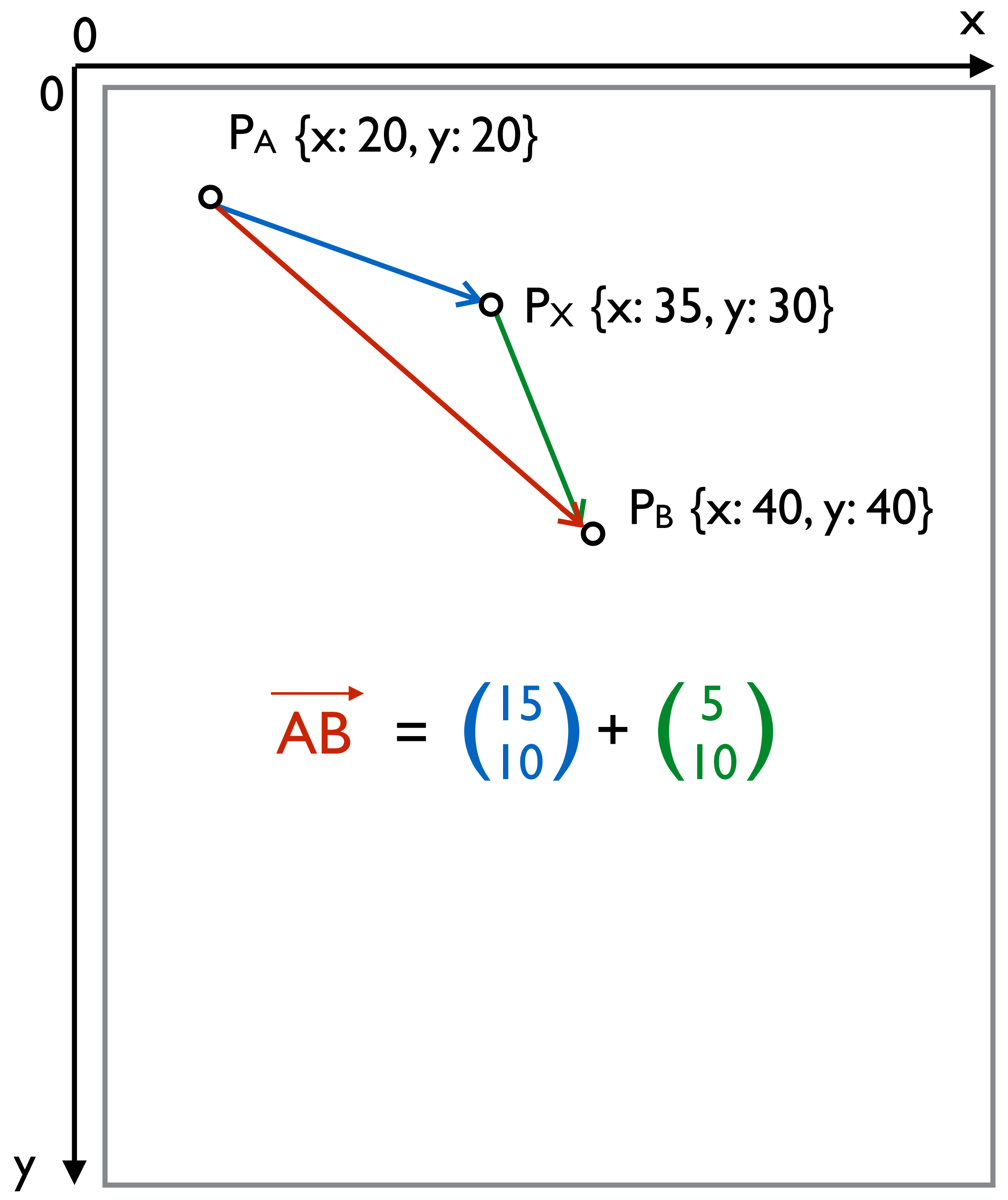
Vector Geometry

- Description of length and direction
- … from one point to another



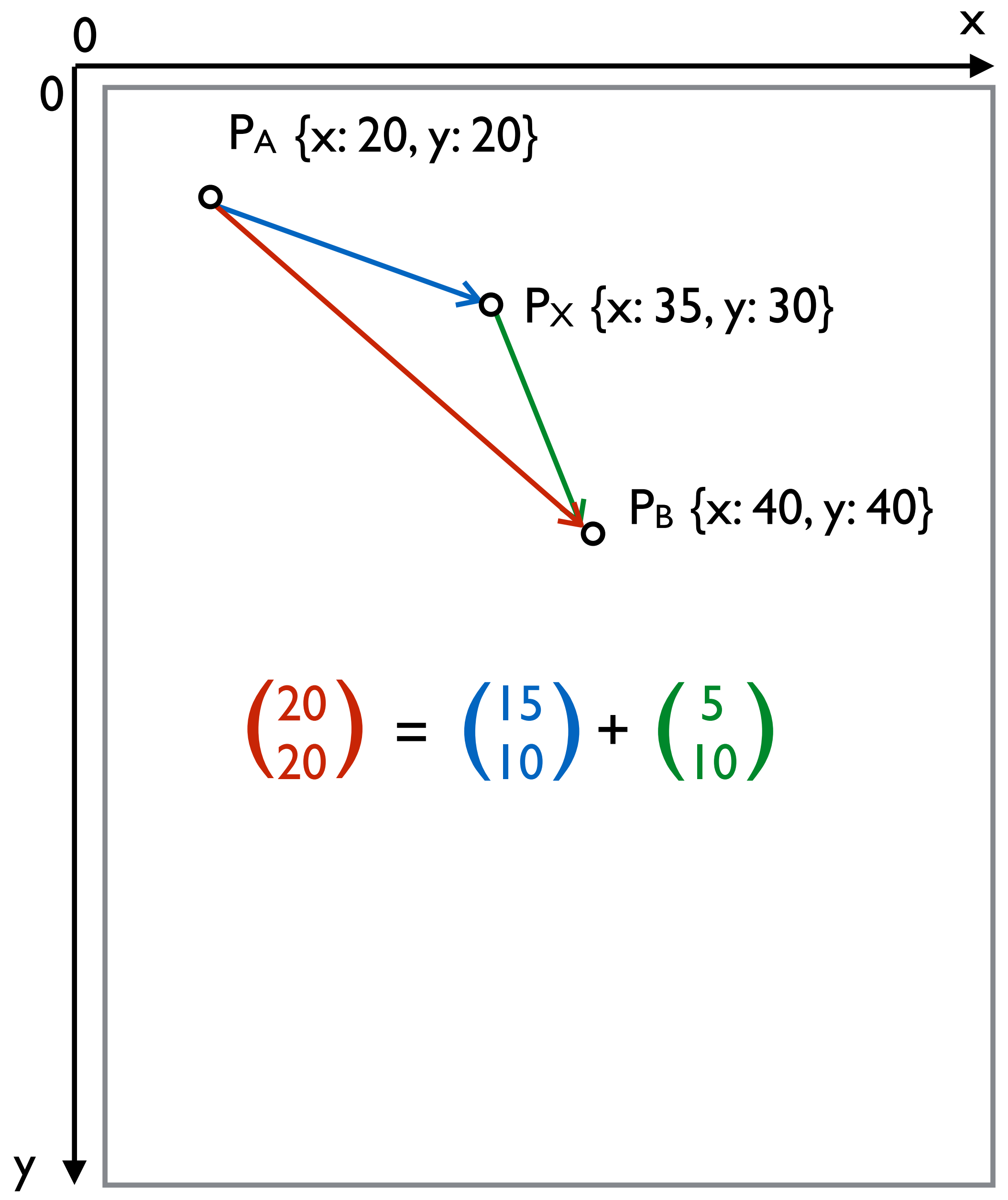
Vector Geometry

- Description of length and direction
- … from one point to another


0
x
0
y
PA {x: 20, y: 20}
PX {x: 35, y: 30}
PB {x: 40, y: 40}
$\overrightarrow{AB} = \begin{pmatrix}15\\10\end{pmatrix} + \begin{pmatrix}5\\10\end{pmatrix}$

Vector Geometry

- Description of length and direction
- … from one point to another



$$\begin{pmatrix}20\\20\end{pmatrix} = \begin{pmatrix}15\\10\end{pmatrix} + \begin{pmatrix}5\\10\end{pmatrix}$$

Drawing Paths

- Starting a Path

```
context.beginPath();
```

Drawing Paths

- Starting a Path

```
context.beginPath();
```

- Moving to a position

Drawing Paths

- Starting a Path

```
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

Drawing Paths

- Starting a Path

```
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

- Drawing a line

Drawing Paths

- Starting a Path

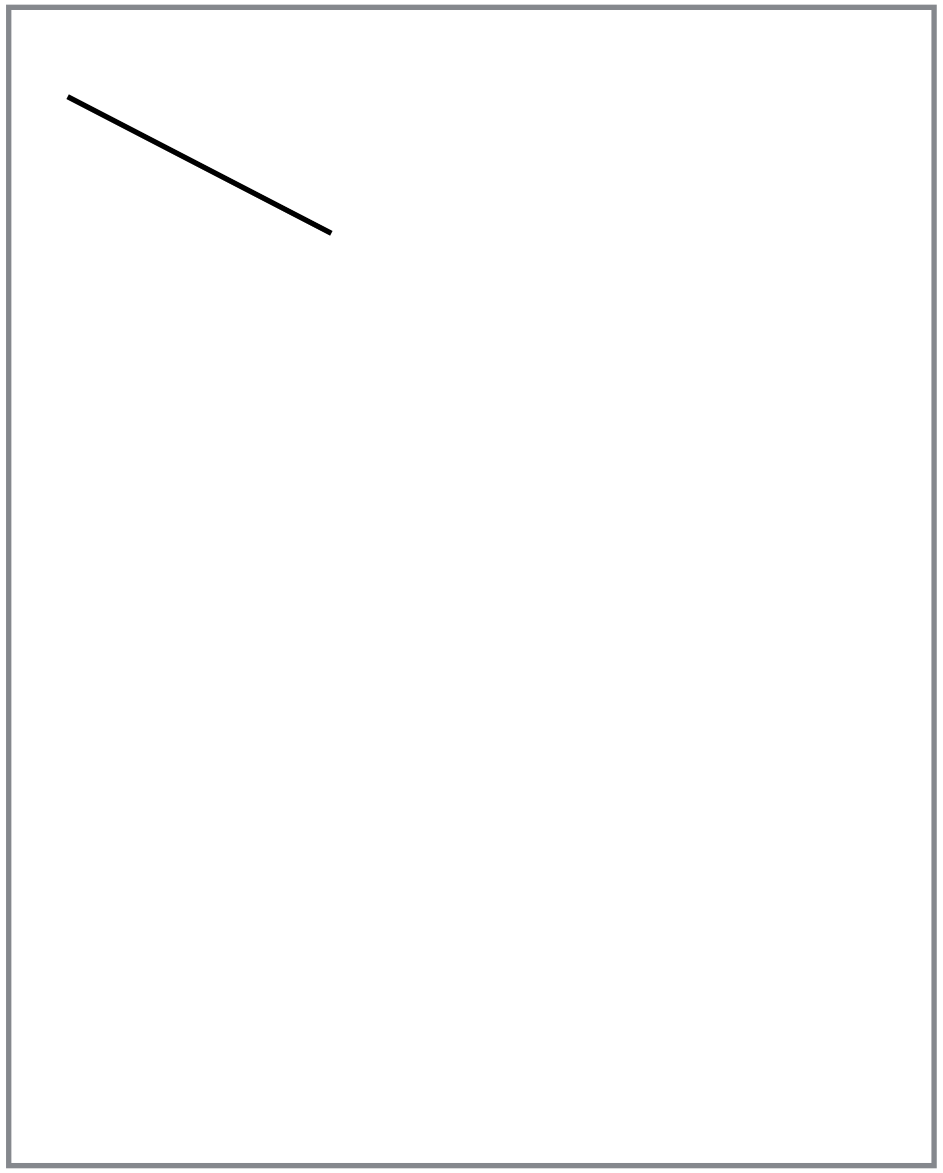
```
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

- Drawing a line

```
context.lineTo(150, 75);
// … more lineTo then stroke
context.stroke();
```



Dev-Env Setup

- Download the source repository

  https://github.com/schaumiii/javascript-days-2016/archive/master.zip

- Extract files and run

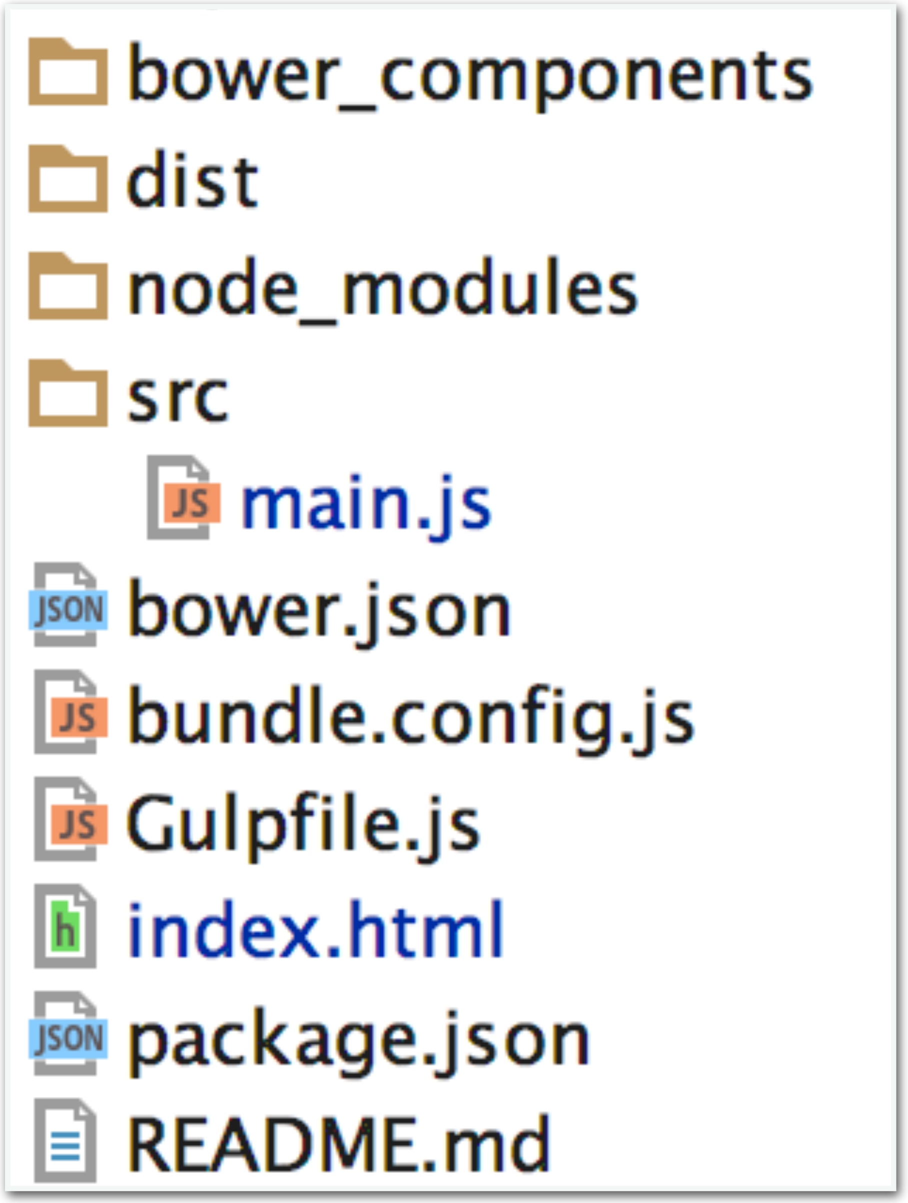
```
$ gulp serve
```

- Open browser http://localhost:3000/

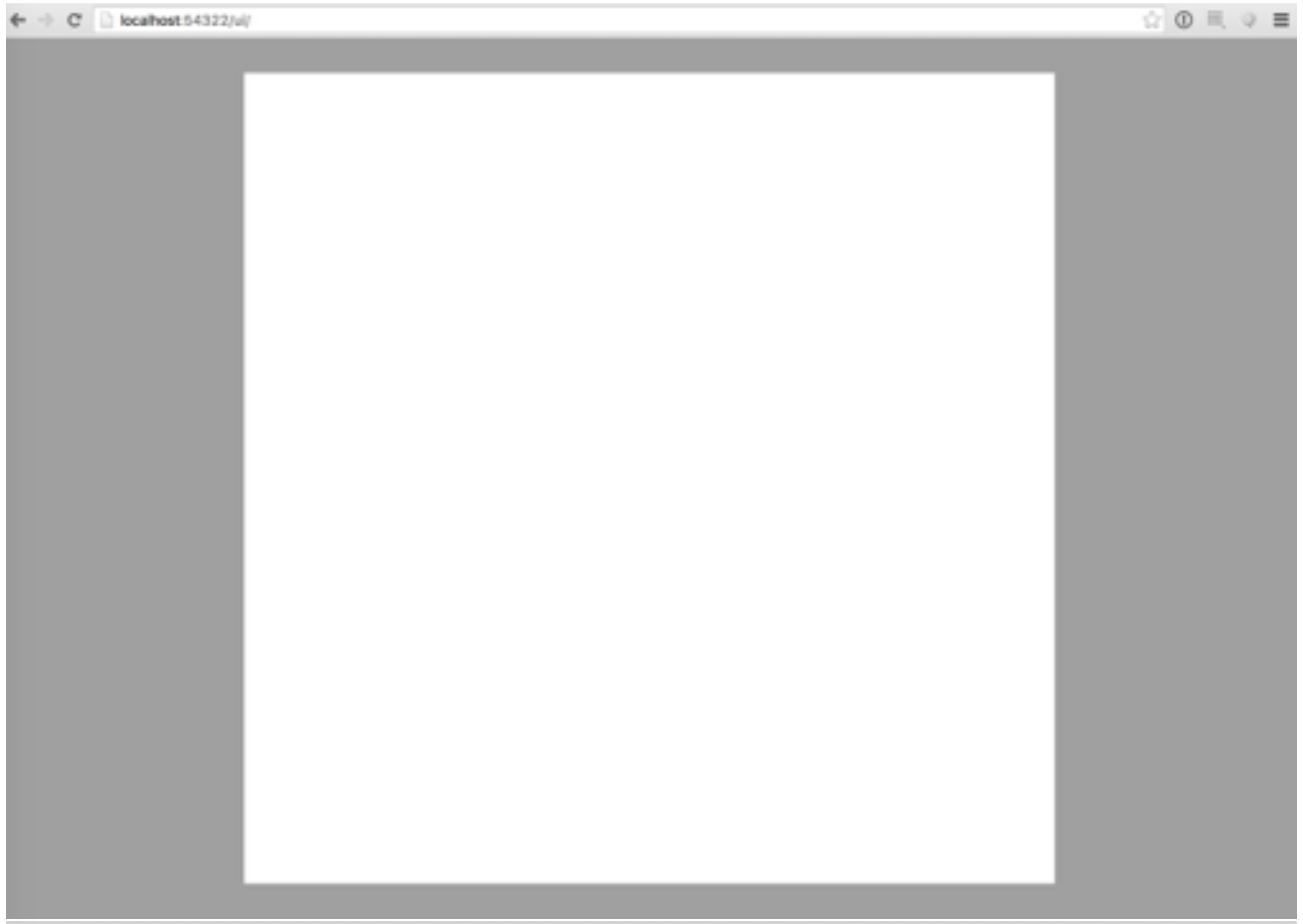
Writing code

Your directory should look like this.

Changes will be made in main.js



```
bower_components
dist
node_modules
src
    main.js
bower.json
bundle.config.js
Gulpfile.js
index.html
package.json
README.md
```



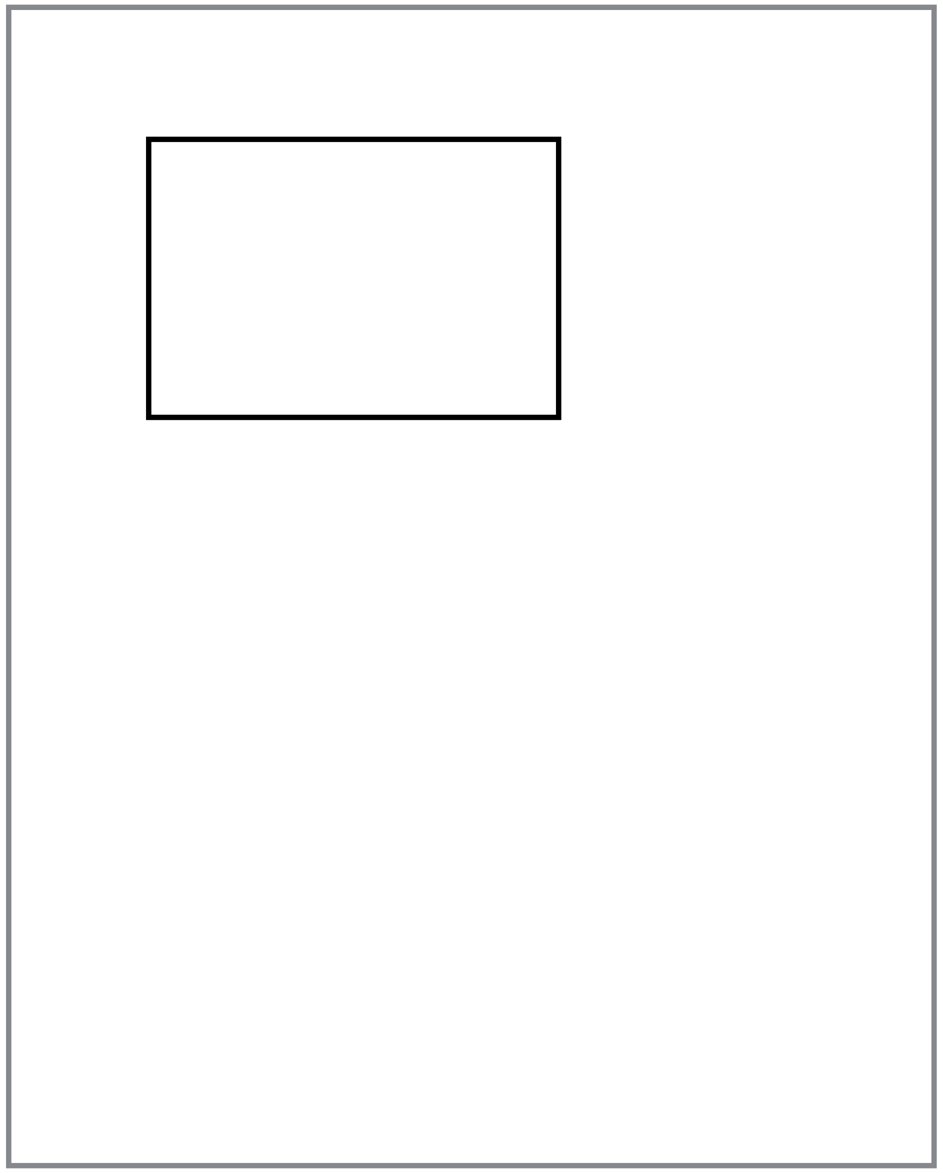
That’s it

First steps

Draw a rectangle

- Starting a Path

```
context.strokeStyle = 'black';
context.beginPath();
```



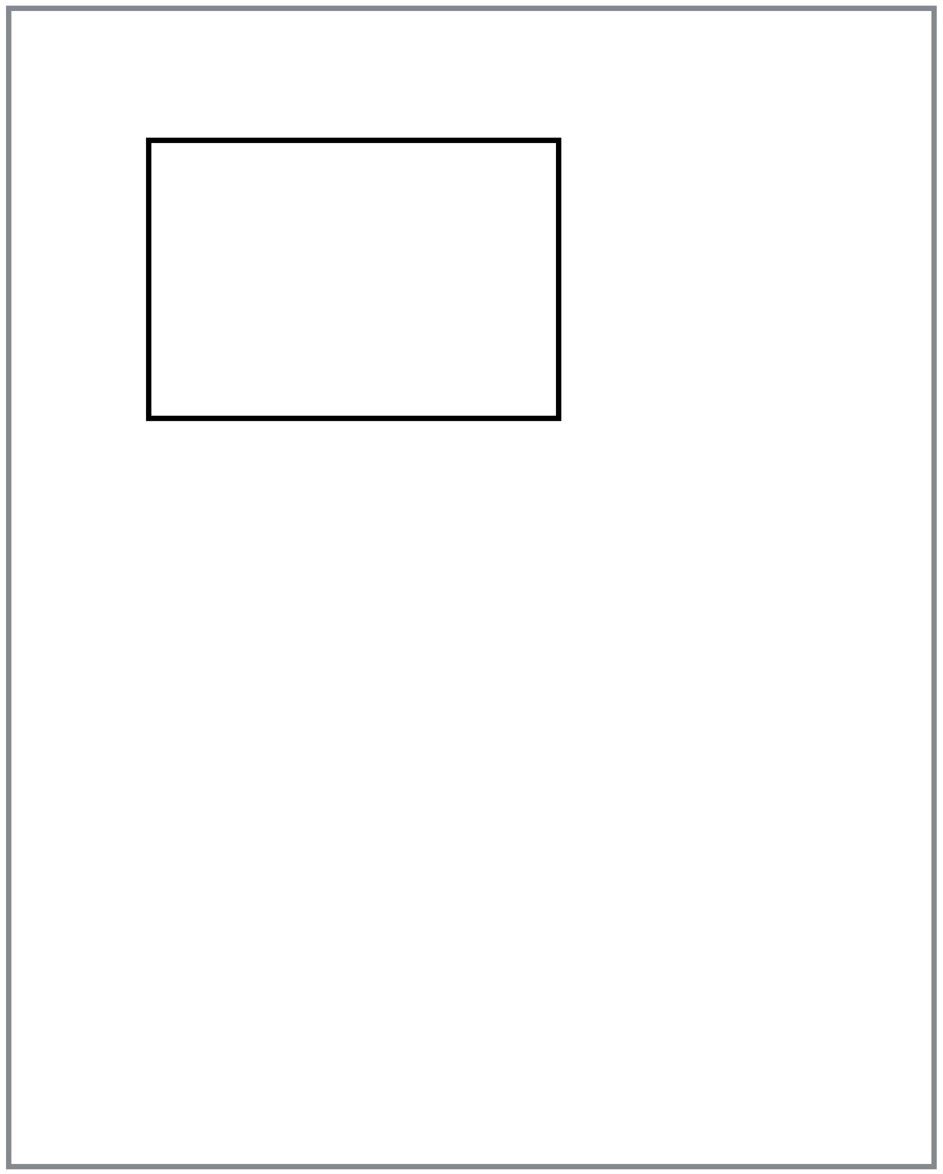
First steps

Draw a rectangle

- Starting a Path

```
context.strokeStyle = 'black';
context.beginPath();
```

- Moving to a position



First steps

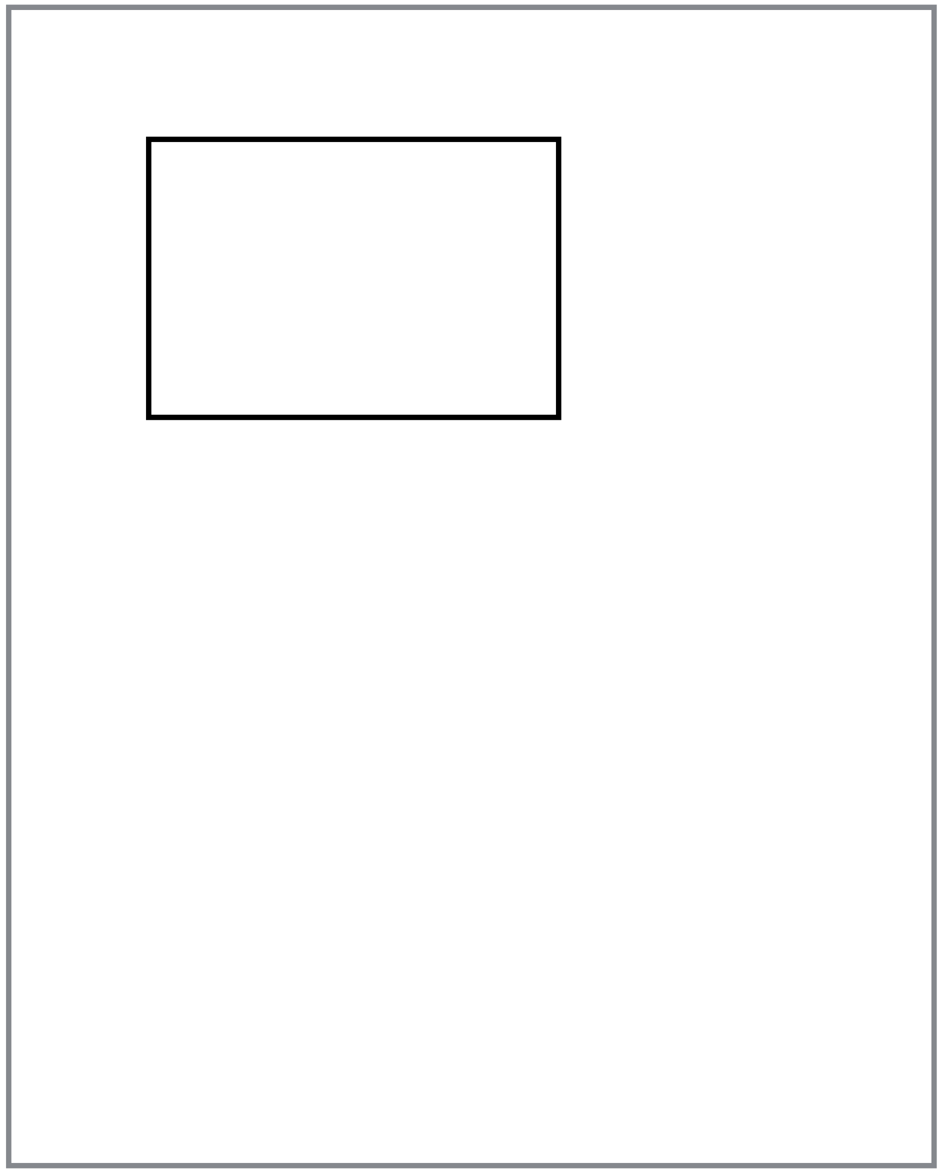
Draw a rectangle

- Starting a Path

```
context.strokeStyle = 'black';
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```



First steps

Draw a rectangle

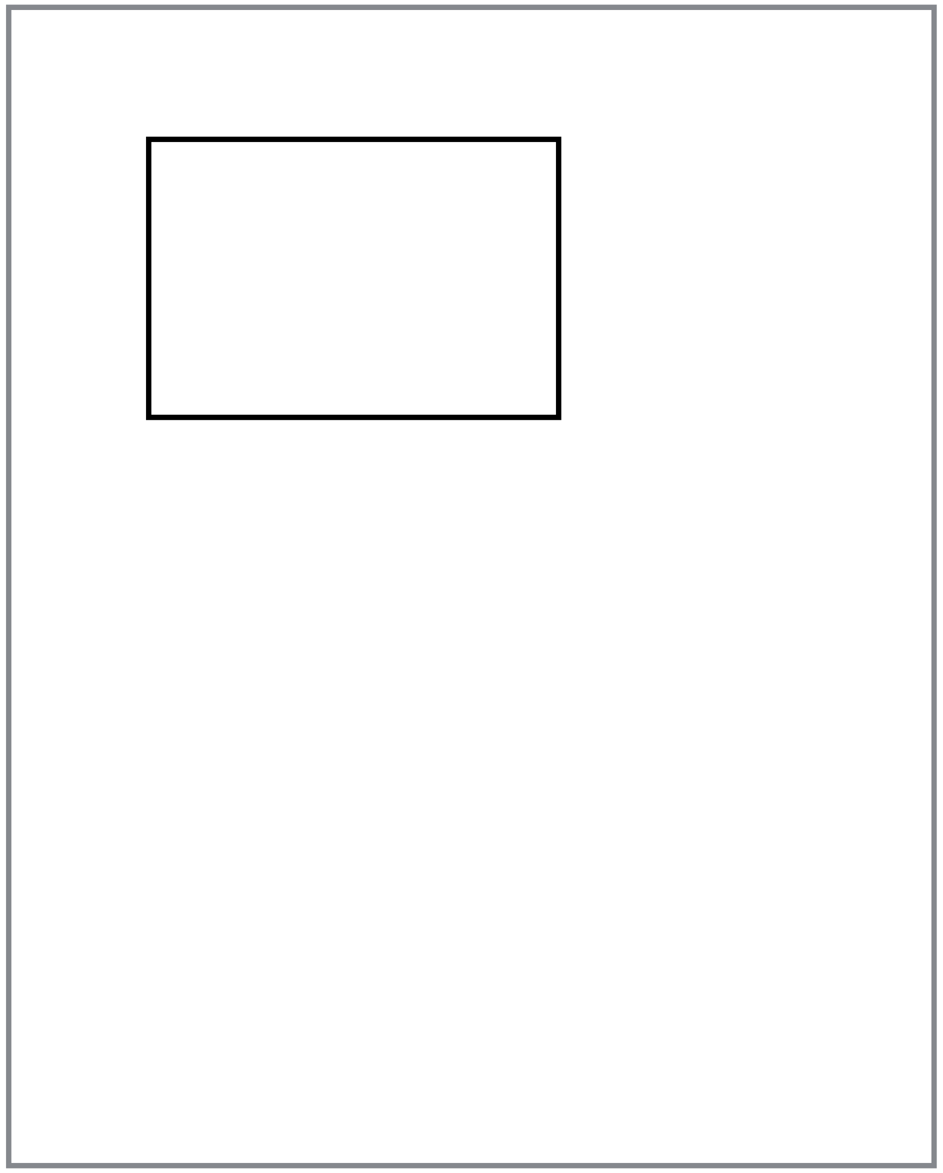
- Starting a Path

```
context.strokeStyle = 'black';
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

- Drawing a line



First steps

Draw a rectangle

- Starting a Path

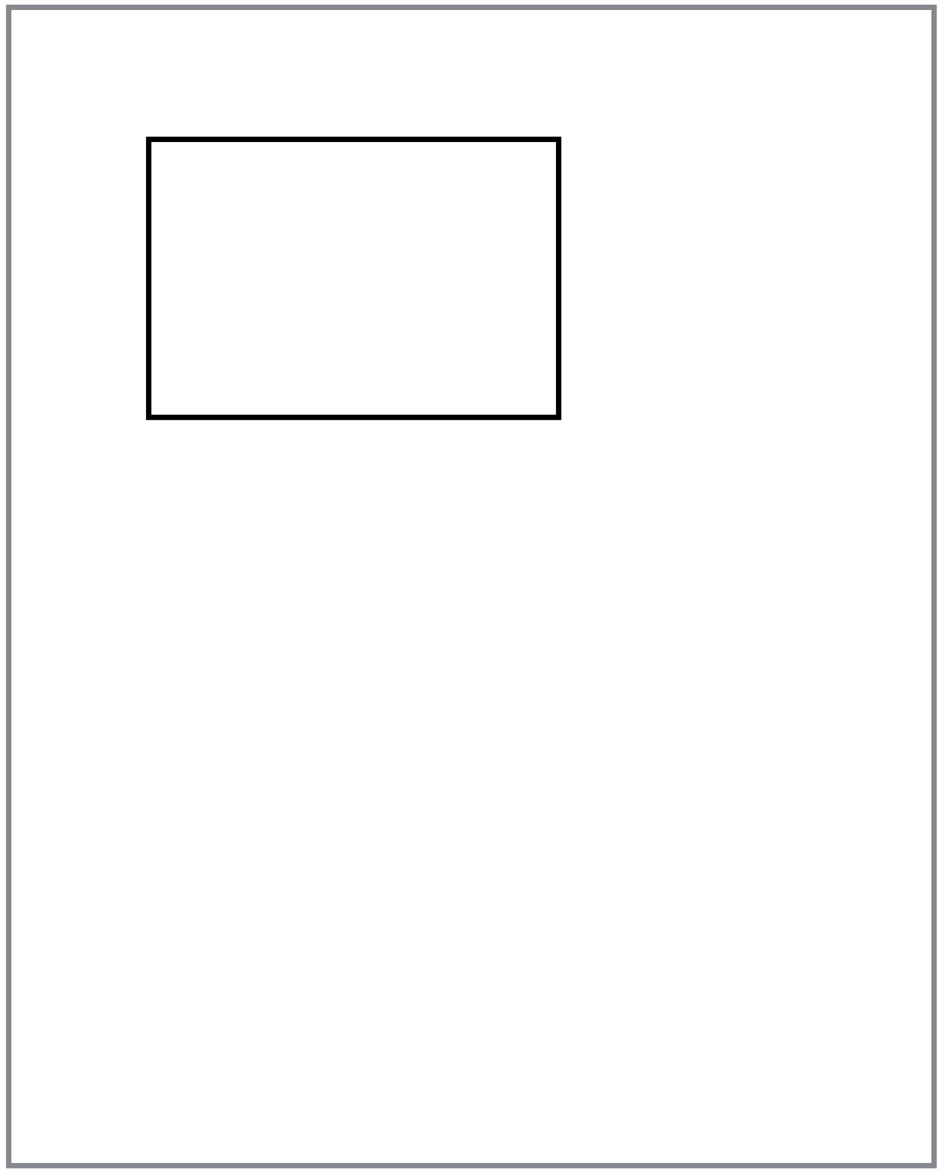
```
context.strokeStyle = 'black';
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

- Drawing a line

```
context.lineTo(150, 75);
// … more lineTo then stroke
context.stroke();
```



First steps

Draw a rectangle

- Starting a Path

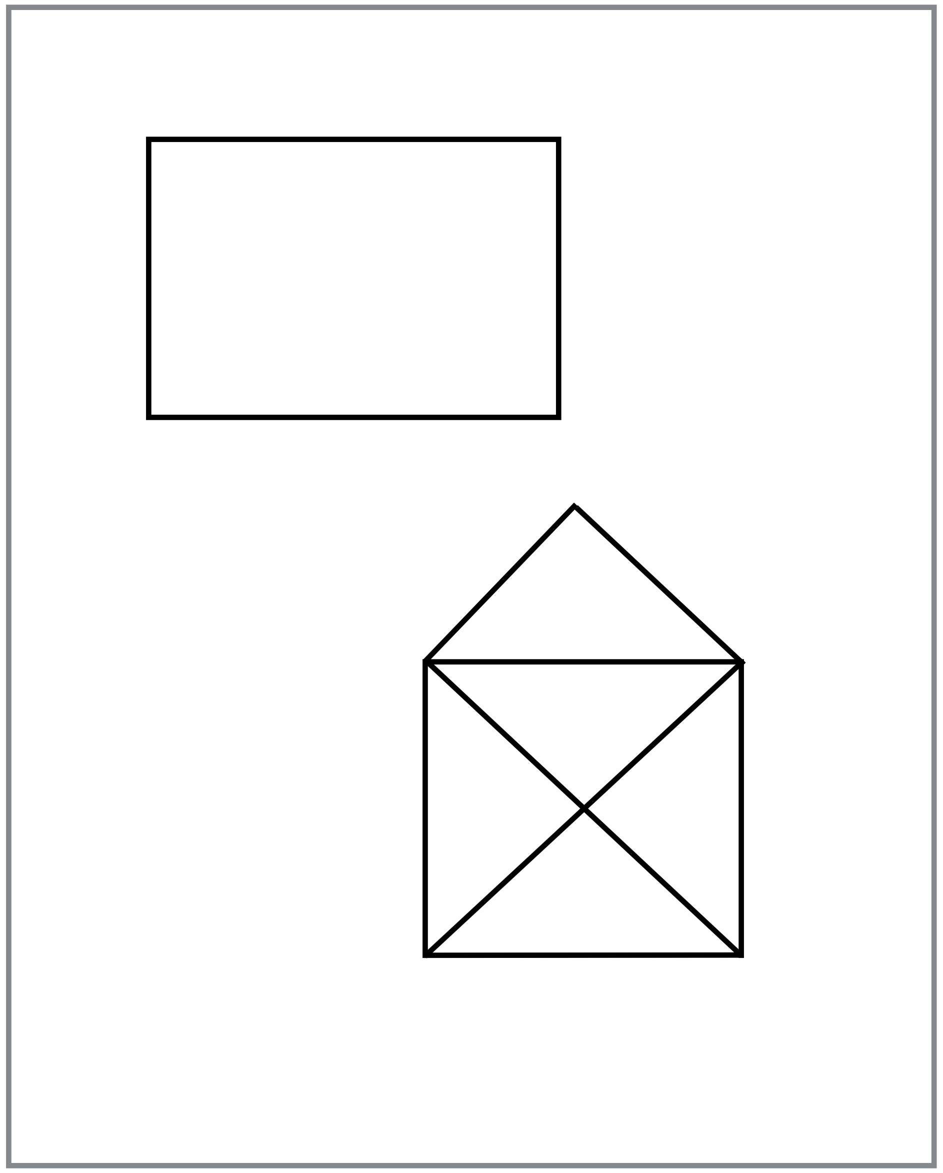
```
context.strokeStyle = 'black';
context.beginPath();
```

- Moving to a position

```
context.moveTo(50, 50);
```

- Drawing a line

```
context.lineTo(150, 75);
// … more lineTo then stroke
context.stroke();
```



Drawing Arcs

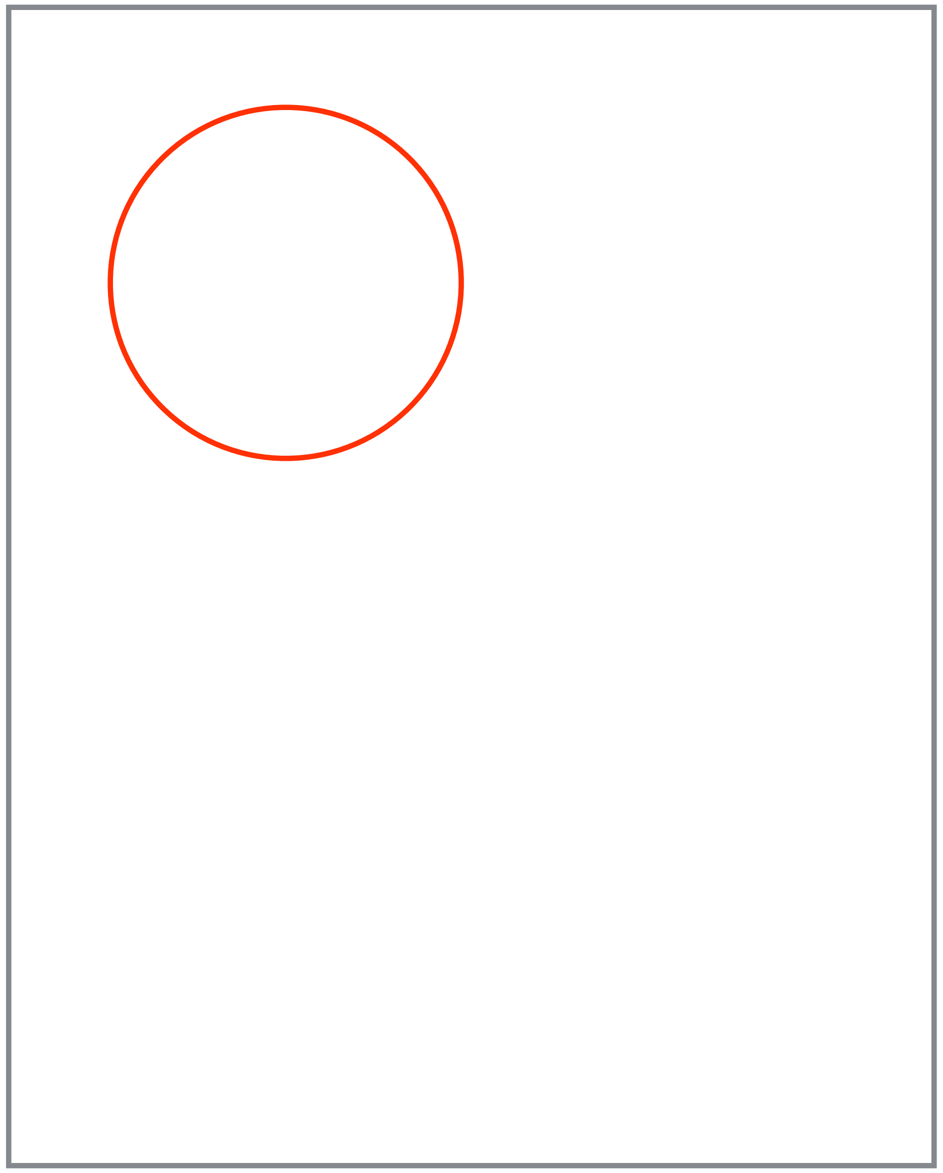
- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

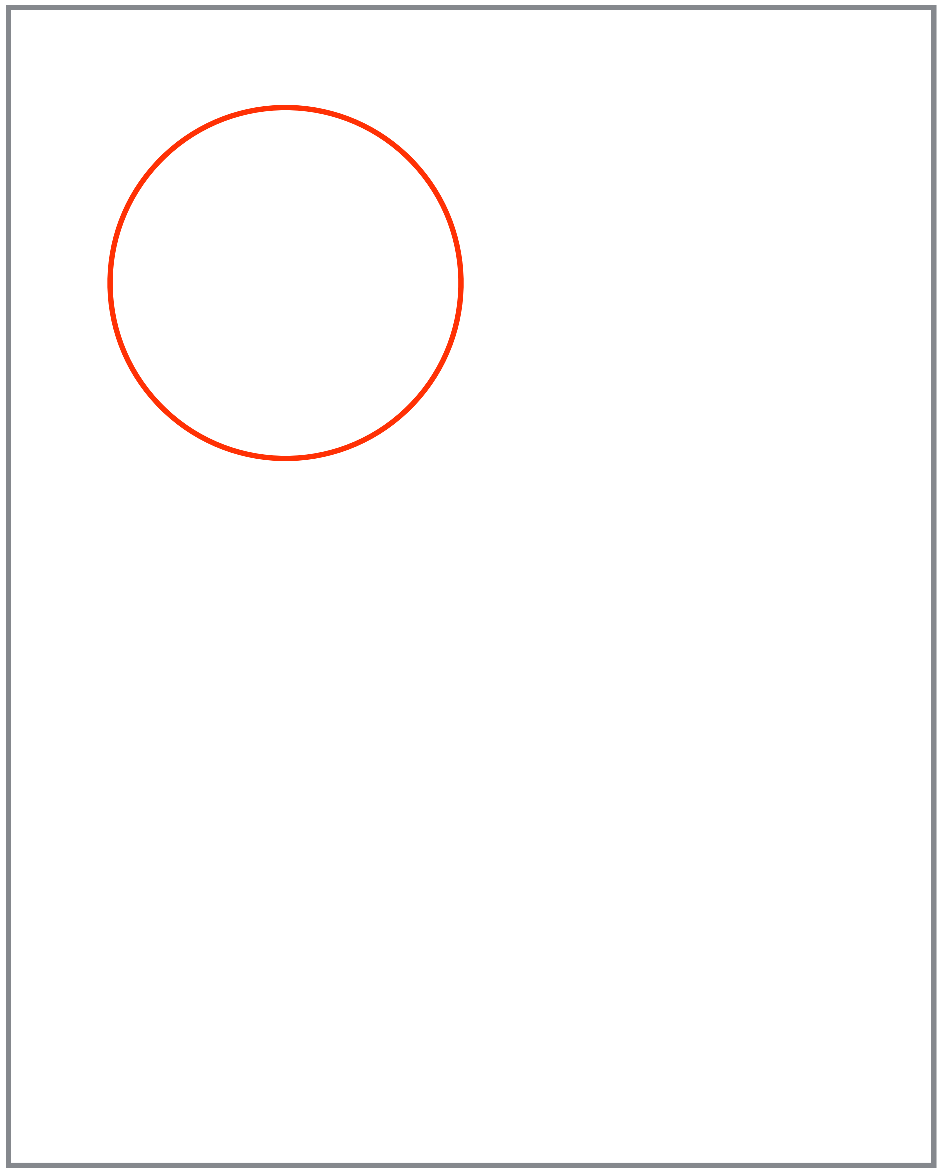


Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

- Arc function in detail:



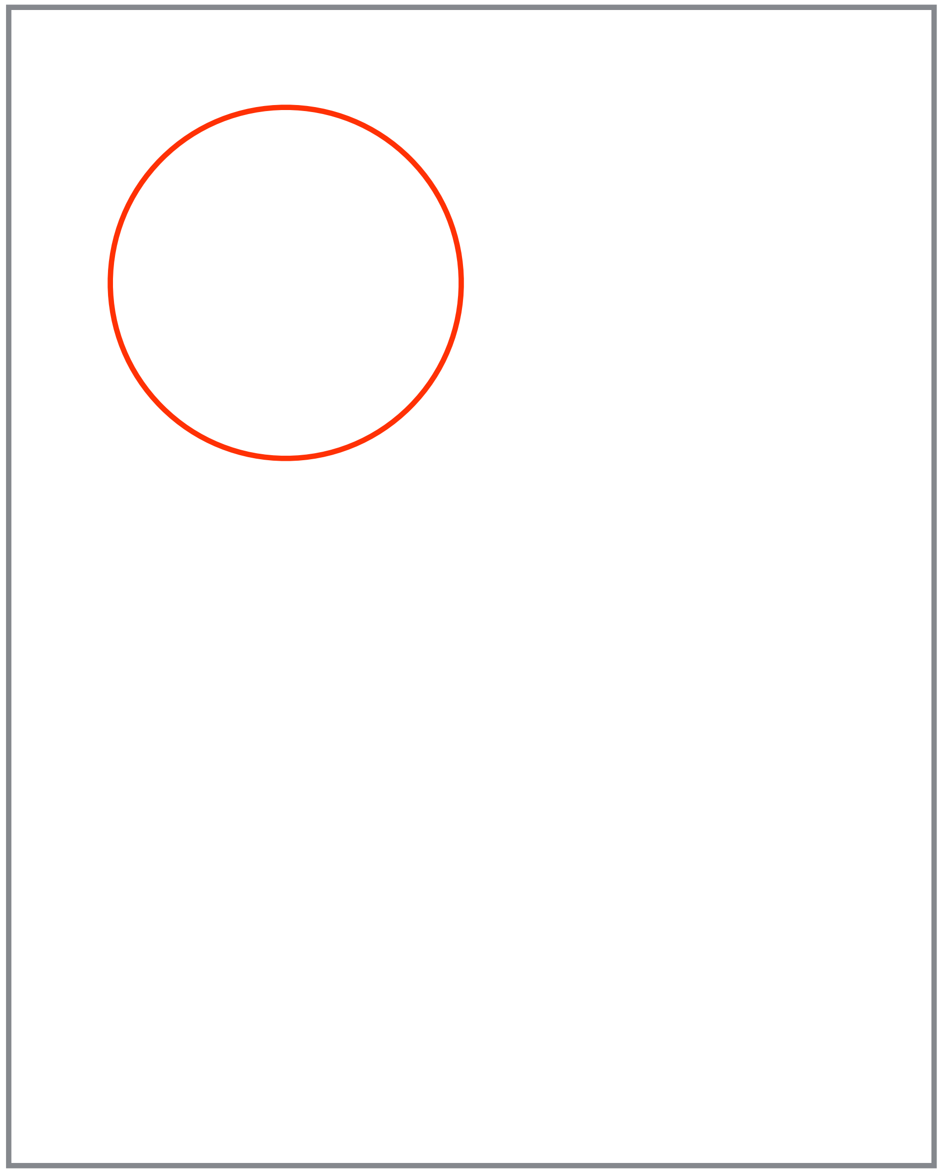
Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

- Arc function in detail:

```
context.arc(
    x, y, r,
    radiantStart,
    radiantEnd,
    cc
);
```



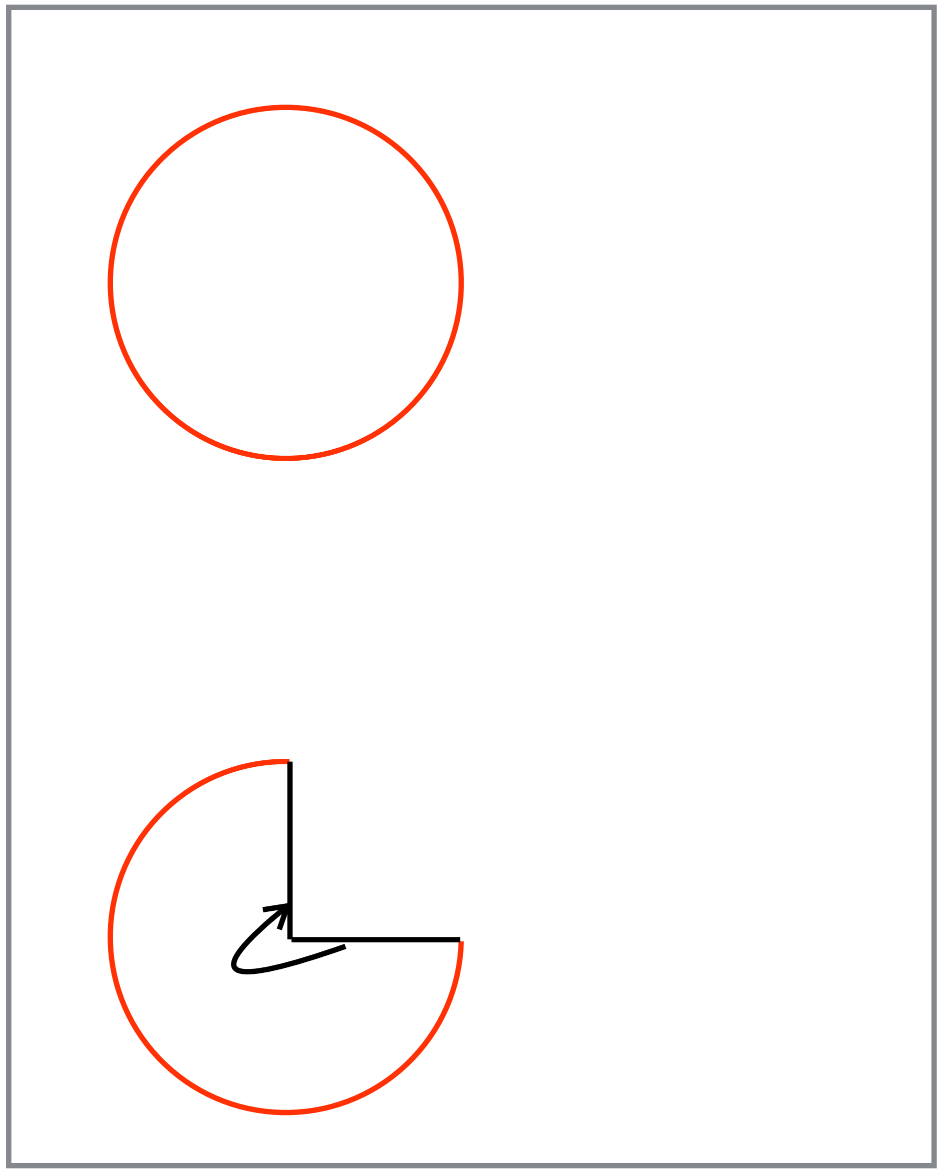
Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

- Arc function in detail:

```
context.arc(
    x, y, r,
    radiantStart,
    radiantEnd,
    cc
);
```



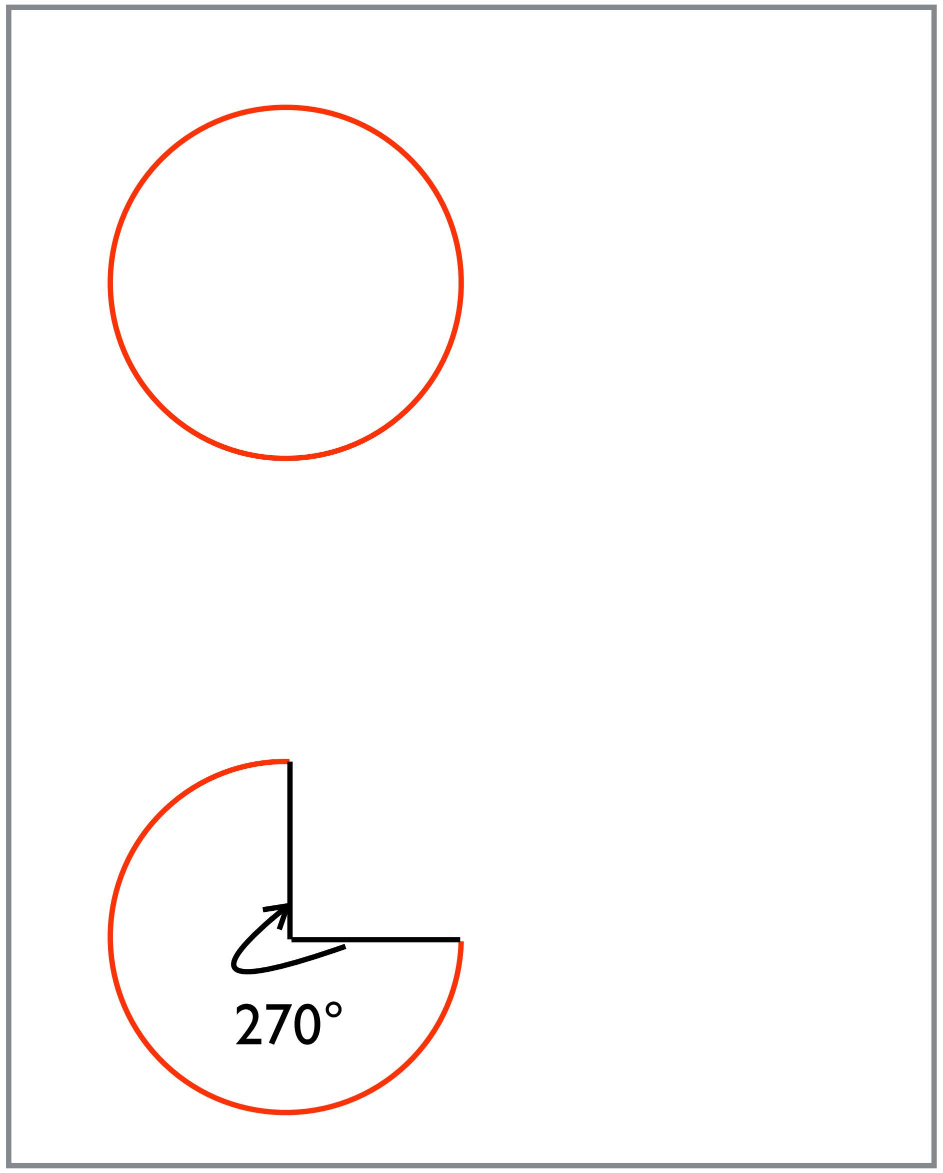
Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

- Arc function in detail:

```
context.arc(
    x, y, r,
    radiantStart,
    radiantEnd,
    cc
);
```



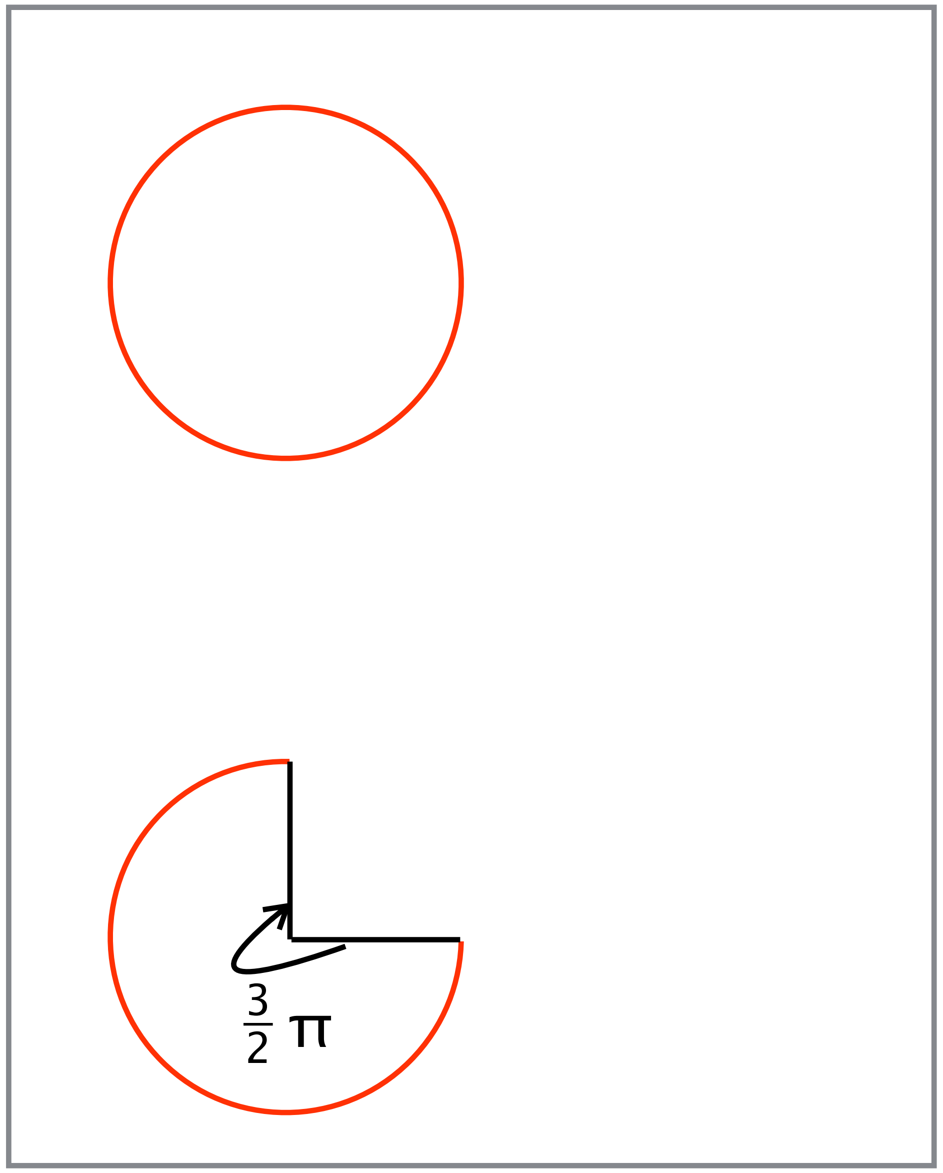
Drawing Arcs

- Drawing simple arcs is easy:

```
context.strokeStyle = 'red';
context.beginPath();
context.arc(
  200, 200, 100, 0, Math.PI*2
);
context.stroke();
```

- Arc function in detail:

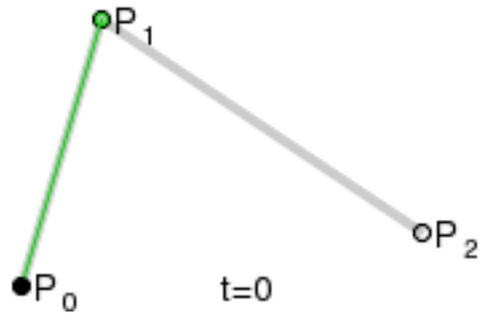
```
context.arc(
    x, y, r,
    radiantStart,
    radiantEnd,
    cc
);
```

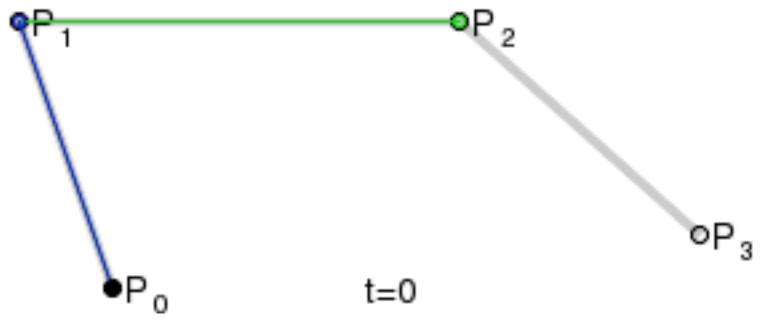


Bezier Curves

- Simple ones are the quadratic bezier curves

```
context.quadraticCurveTo(
    cpX, cpY,
    x, y
);
```



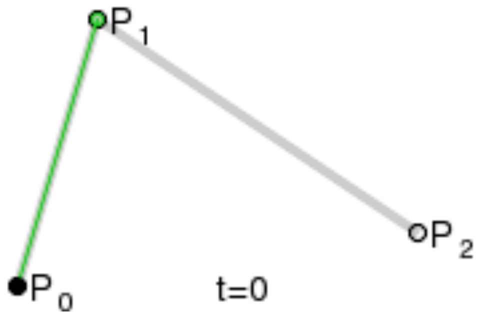


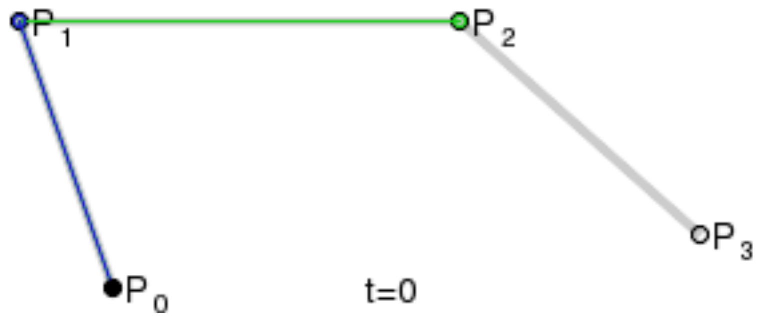
Beispiele aus der Wikipedia: https://de.wikipedia.org/wiki/Bézierkurve

Bezier Curves

- Simple ones are the quadratic bezier curves

```
context.quadraticCurveTo(
    cpX, cpY,
    x, y
);
```





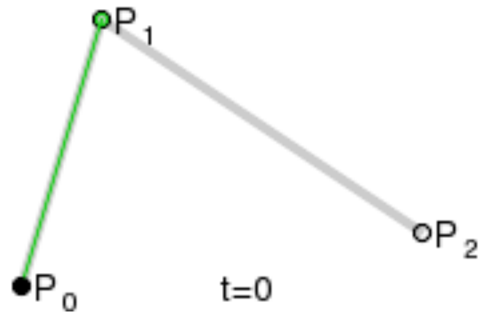
Beispiele aus der Wikipedia: https://de.wikipedia.org/wiki/Bézierkurve

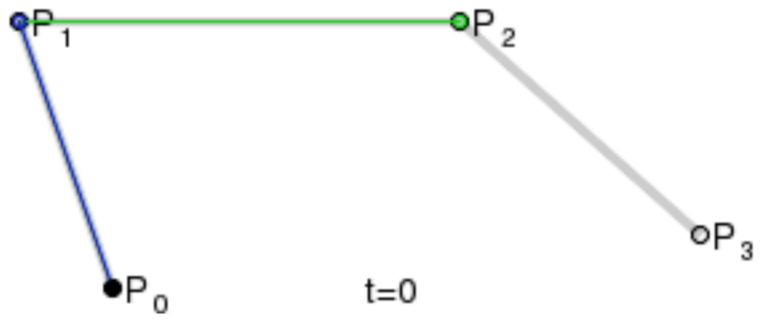
Bezier Curves

- Simple ones are the quadratic bezier curves

```
context.quadraticCurveTo(
    cpX, cpY,
    x, y
);
```

- More complex ones have more control points





Beispiele aus der Wikipedia: https://de.wikipedia.org/wiki/Bézierkurve

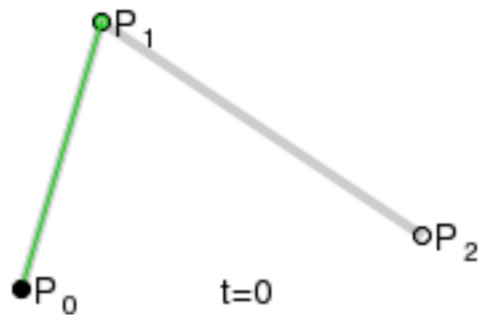
Bezier Curves

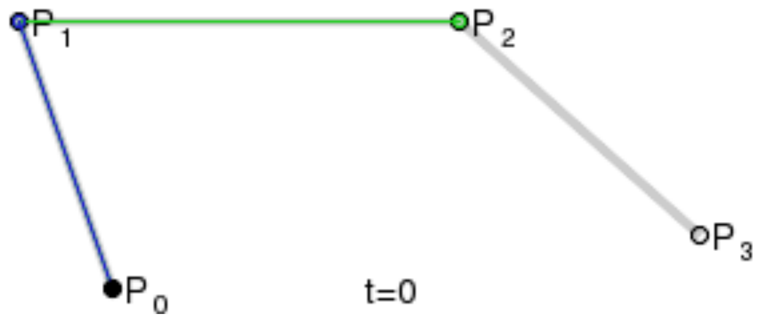
- Simple ones are the quadratic bezier curves

```
context.quadraticCurveTo(
    cpX, cpY,
    x, y
);
```

- More complex ones have more control points

```
context.bezierCurveTo(
    cpX1, cpY1,
    cpX2, cpY2,
    x, y
);
```





Beispiele aus der Wikipedia: https://de.wikipedia.org/wiki/Bézierkurve

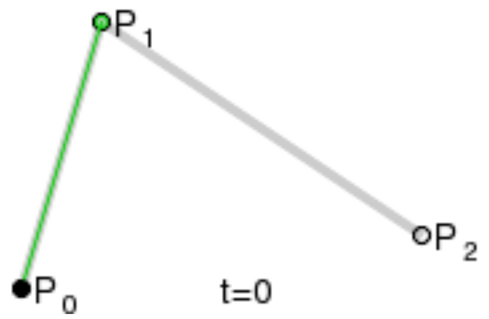
Bezier Curves

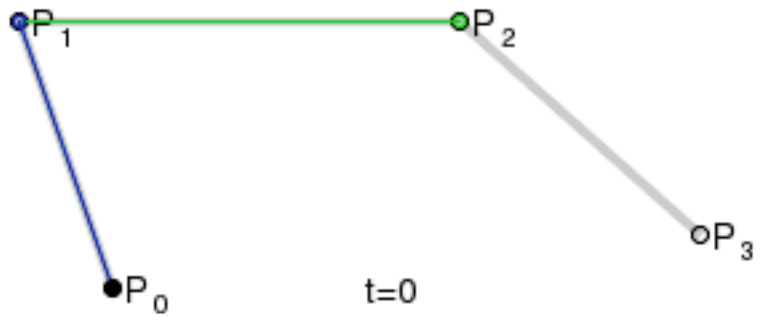
- Simple ones are the quadratic bezier curves

```
context.quadraticCurveTo(
    cpX, cpY,
    x, y
);
```

- More complex ones have more control points

```
context.bezierCurveTo(
    cpX1, cpY1,
    cpX2, cpY2,
    x, y
);
```





Beispiele aus der Wikipedia: https://de.wikipedia.org/wiki/Bézierkurve

Coloring the world

- Using colors is simple
  - strokeStyle

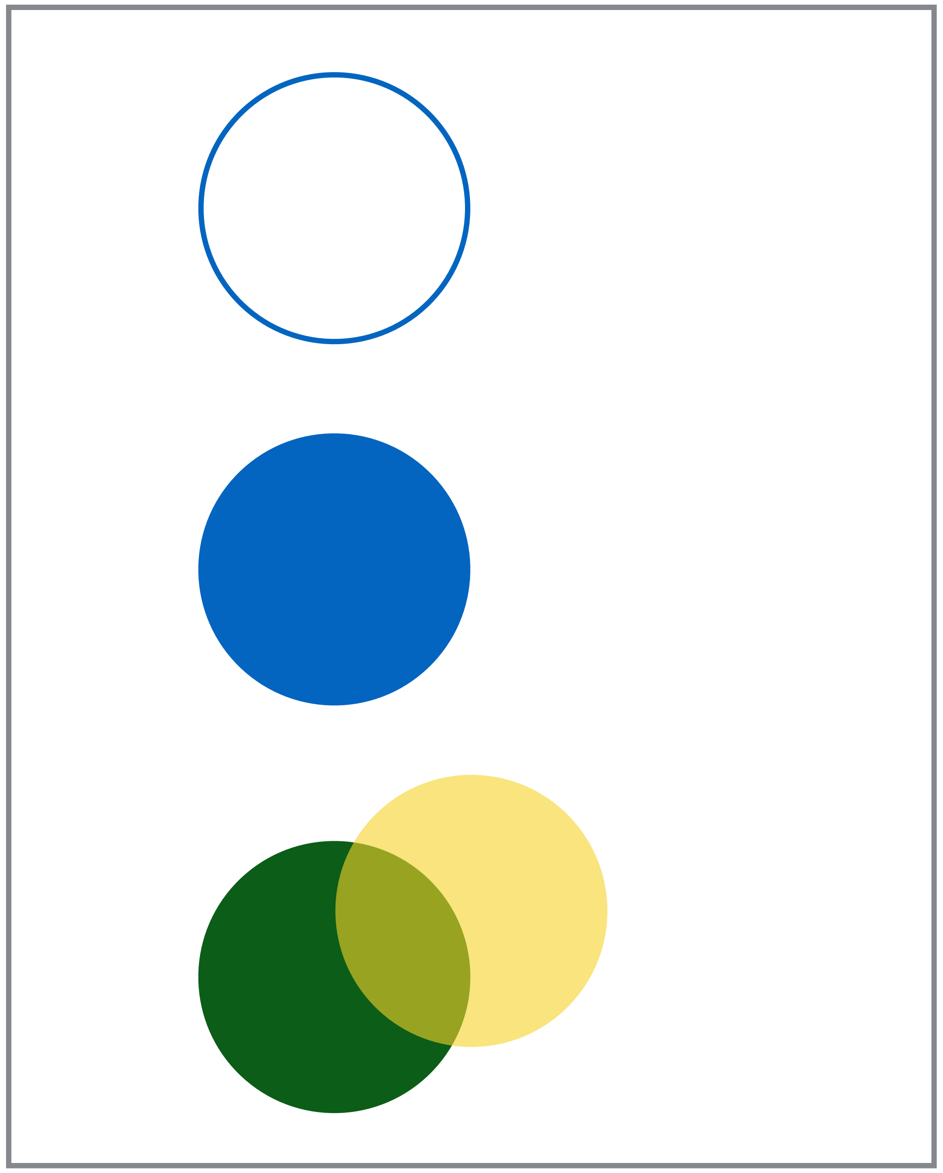
```
context.strokeStyle = 'blue';
```

  - fillStyle

```
context.fillStyle =
    'rgb(0, 255, 0)';
```

- Using transparency

```
context.fillStyle =
    'rgba(0, 255, 0, 0.6)';
```



Drawing

- Drawing with colors
- Rendering text

```
context.font = '40px Verdana';

context.fillText(
  'Some Text', x, y
);
```

- Measuring text

```
var measures =
  context.measureText('Some Text');

measures.width;
// measured width in pixels
```

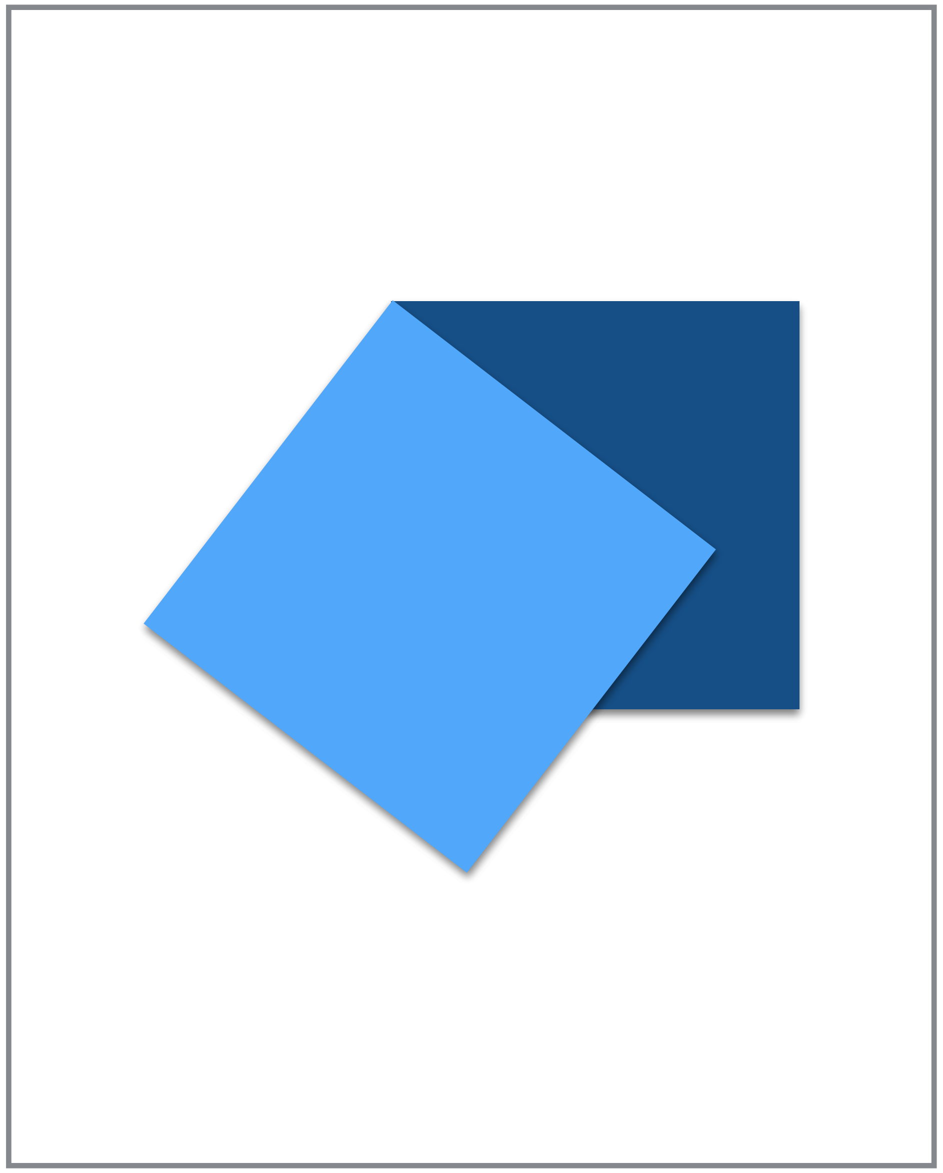
Saving and Restoring

- The state of a canvas can be stored and restored
- Useful to go back to a previous state
  - We do not need to apply or undo style/translation/… changes

```
context.strokeStyle = 'red';
context.strokeRect(
  100, 100, 200, 100
);

context.save();
context.strokeStyle = 'blue';
context.strokeRect(
  125, 125, 200, 100
);

context.restore();
context.strokeRect(
  150, 150, 200, 100
);
```

Transformations

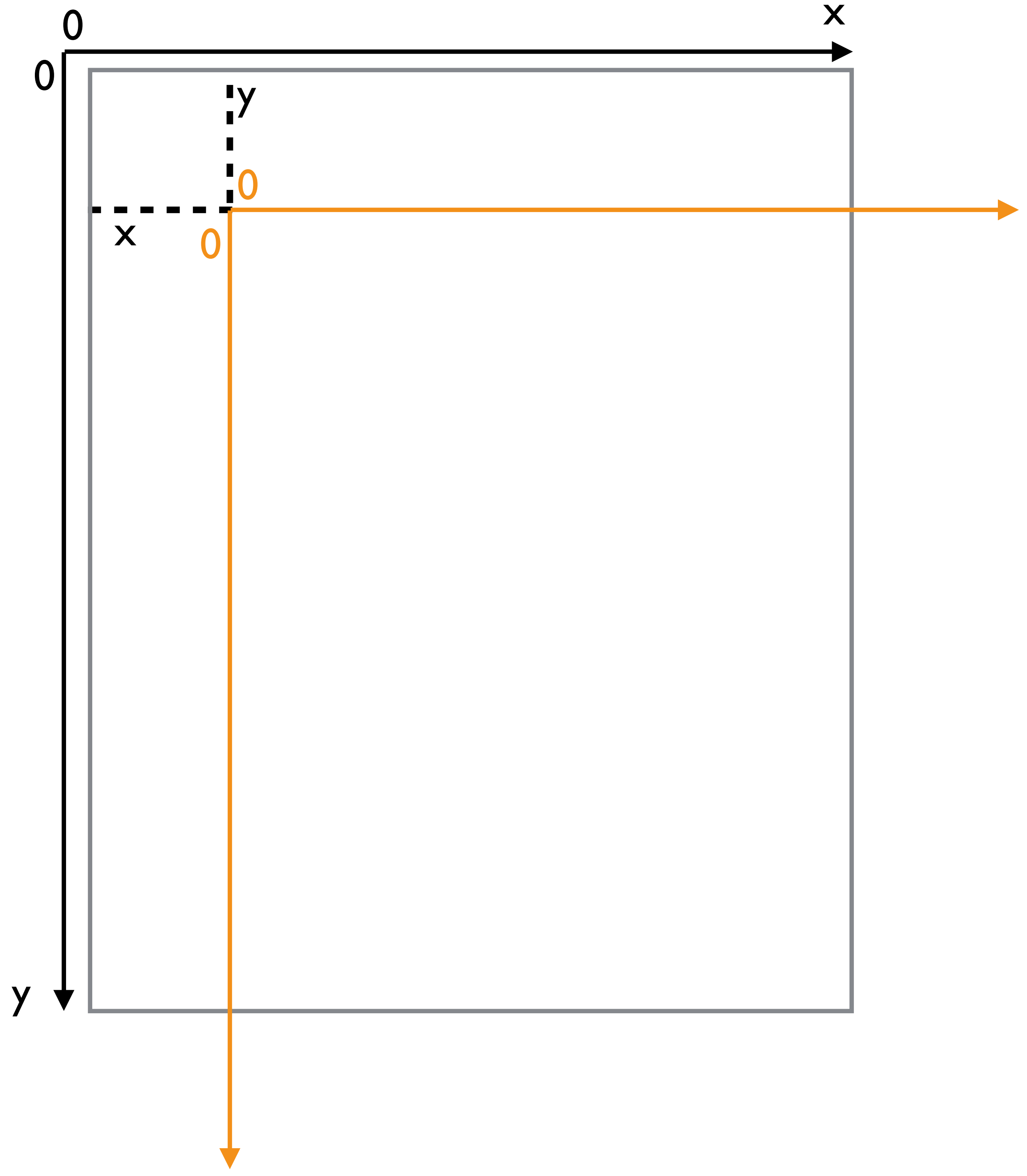
- Transformations do change the canvas origin not already drawn content
- Rotating, Translating, Scaling
- Good practice is to store canvas before



Translating

- Moves the Canvas and its origin on the grid
- Easy moving of complex drawing
- ⚠️ Possible to move outside the grid

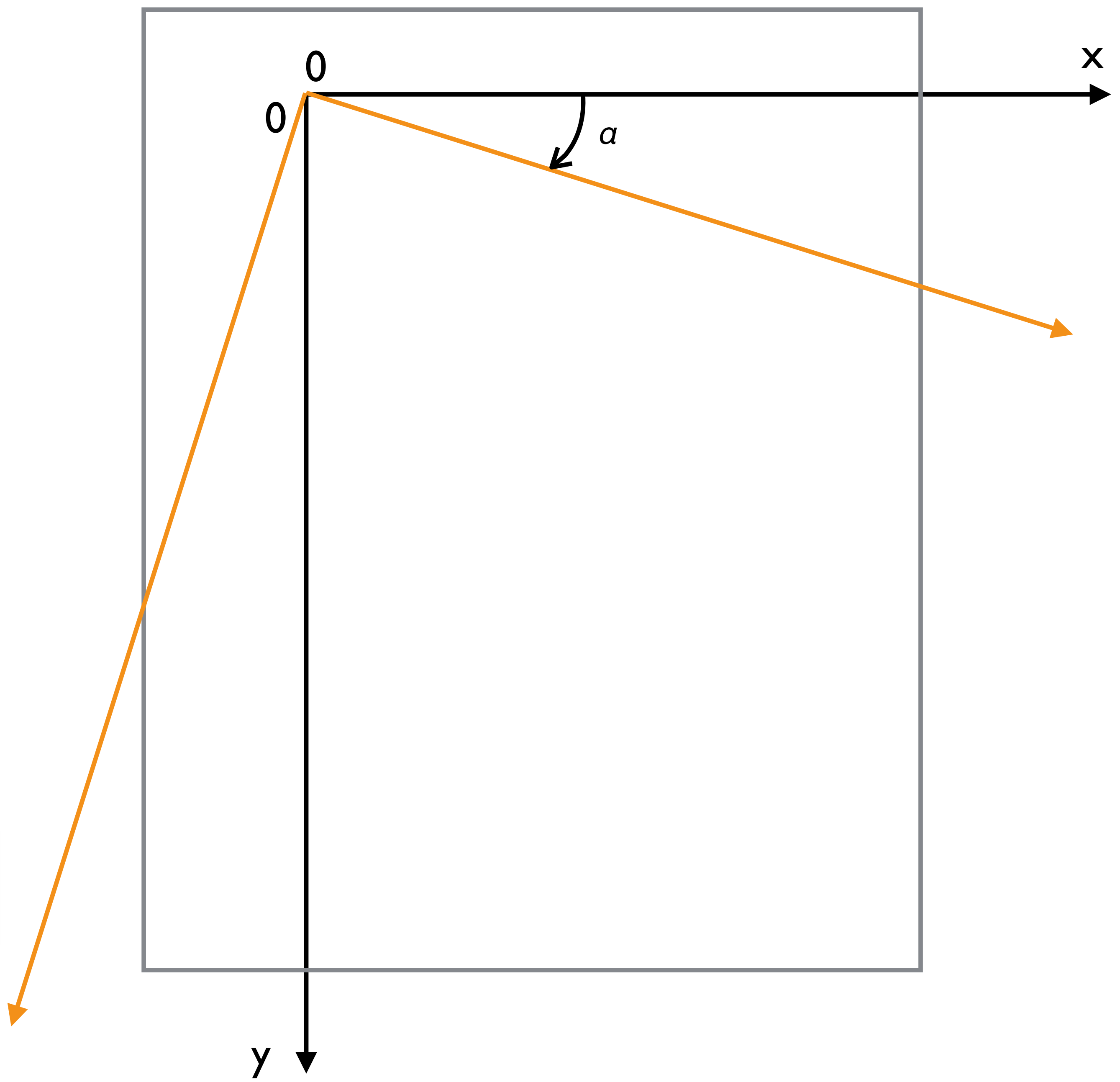
```
context.translate(10, 30);
```



Rotating

- Rotates the canvas clockwise around its origin
- Angle unit in radiant - remember using `arc()`?
- Change rotation center by using `translate()`

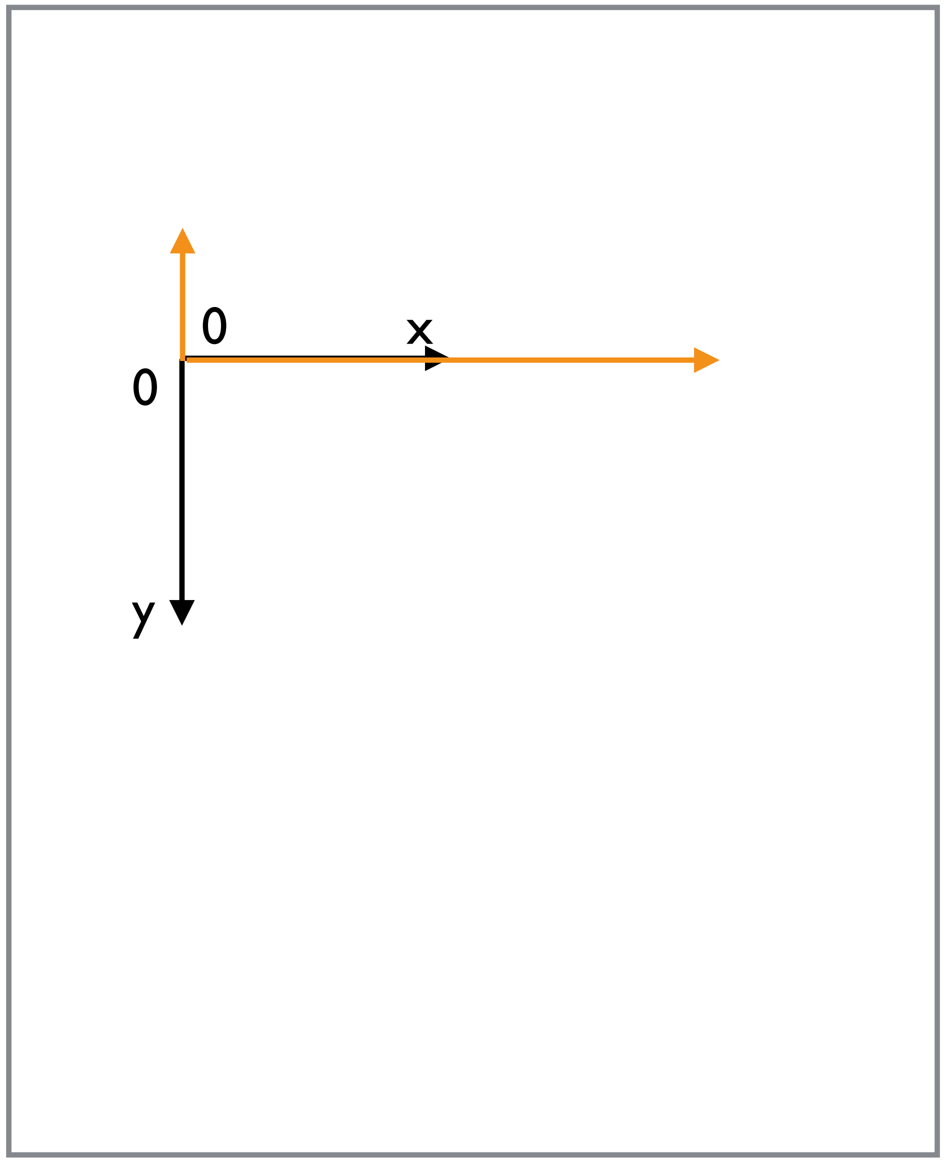
```
context.rotate(Math.PI / 2);
```



Scaling

- Changes the scale of the canvas grid
- Negative values allowed (resulting in mirroring)
- Can be used to scale down or up drawings by changing grid's pixel size

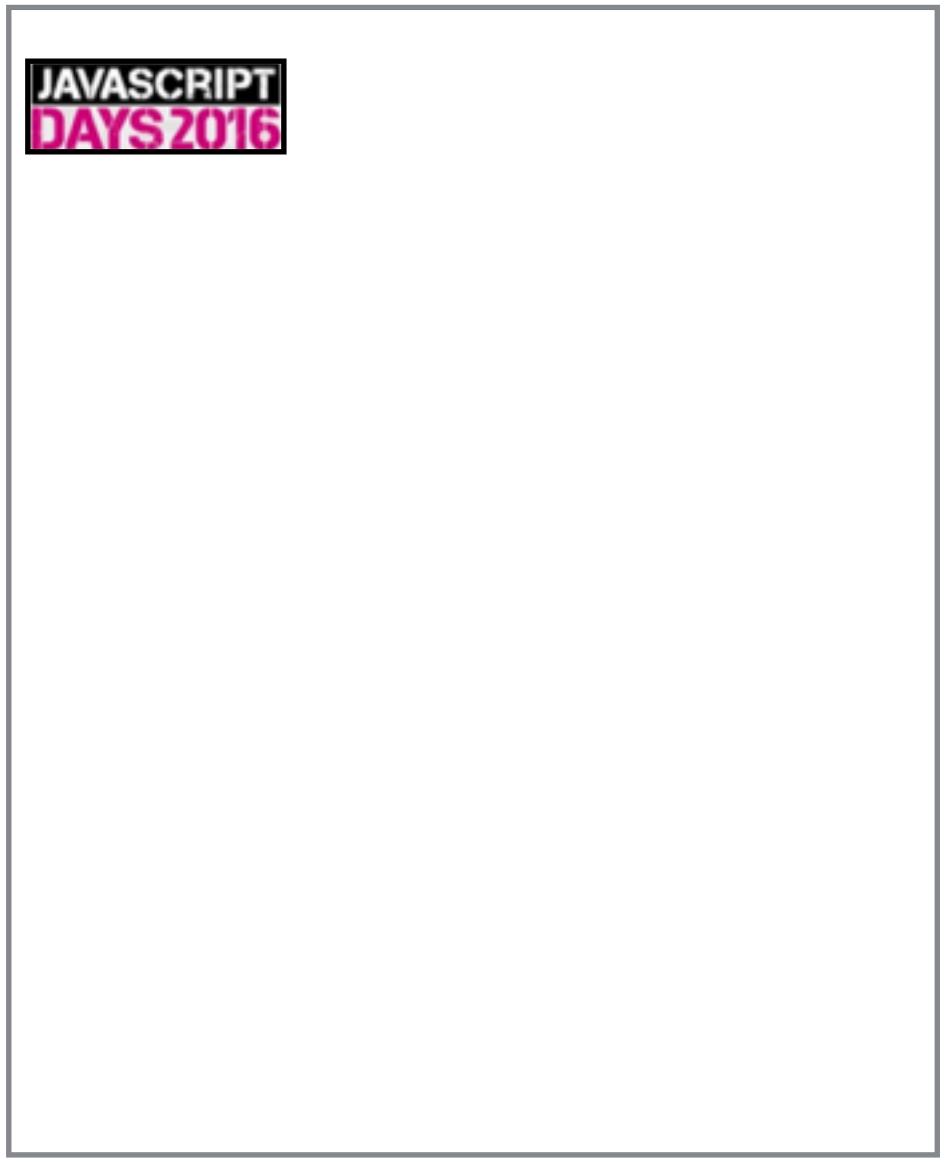
```
context.scale(2.0, -0.5);
```



Images

- Loading images
  - from `<img>`
  - from other `<canvas>`
  - frame from `<video>`
- Can be drawn to the current canvas
- Can be scaled and sliced

```
context.drawImage(img, 10, 50);
```



Manipulating pixels

- The `ImageData` object
  - width, height
  - data rgba-Pixelwerte[]

```
var imgData = context.getImageData(
    0, 0,
    width, height
);

// coloring first pixel red
imgData.data[0] = 255; // r
imgData.data[1] = 0;   // g
imgData.data[2] = 0;   // b
imgData.data[3] = 255; // a

context.putImageData(imgData, 0, 0);
```

Manipulating pixels

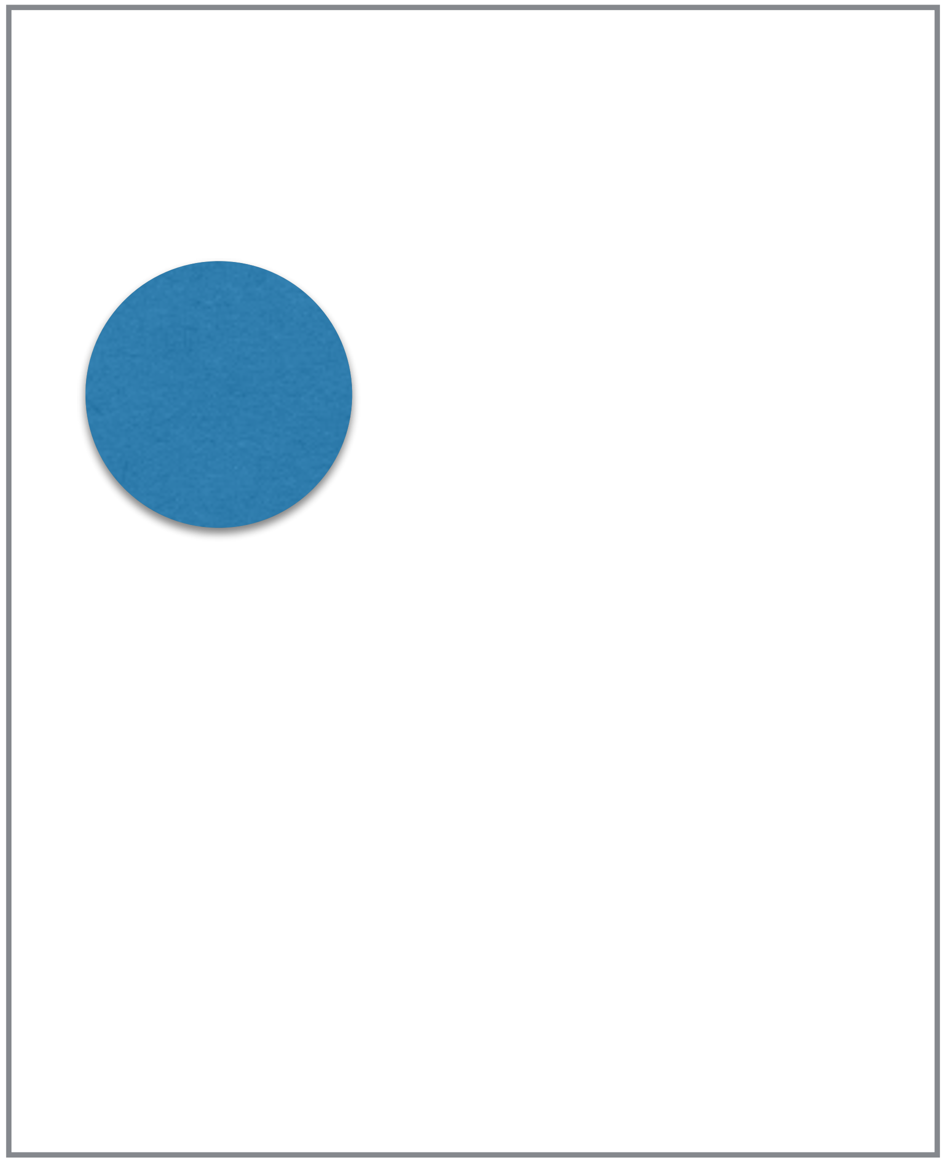
- Can be used to implement simple filters
- Grayfiltering by building the average over RGB-values

```
var img = document.createElement('img');
img.src = 'image.png';

img.onload = function() {
  context.drawImage(img, 0, 0);

  var imgData = context.getImageData(…);
  // manipulating data
  context.putImageData(…);
}
```

Animations

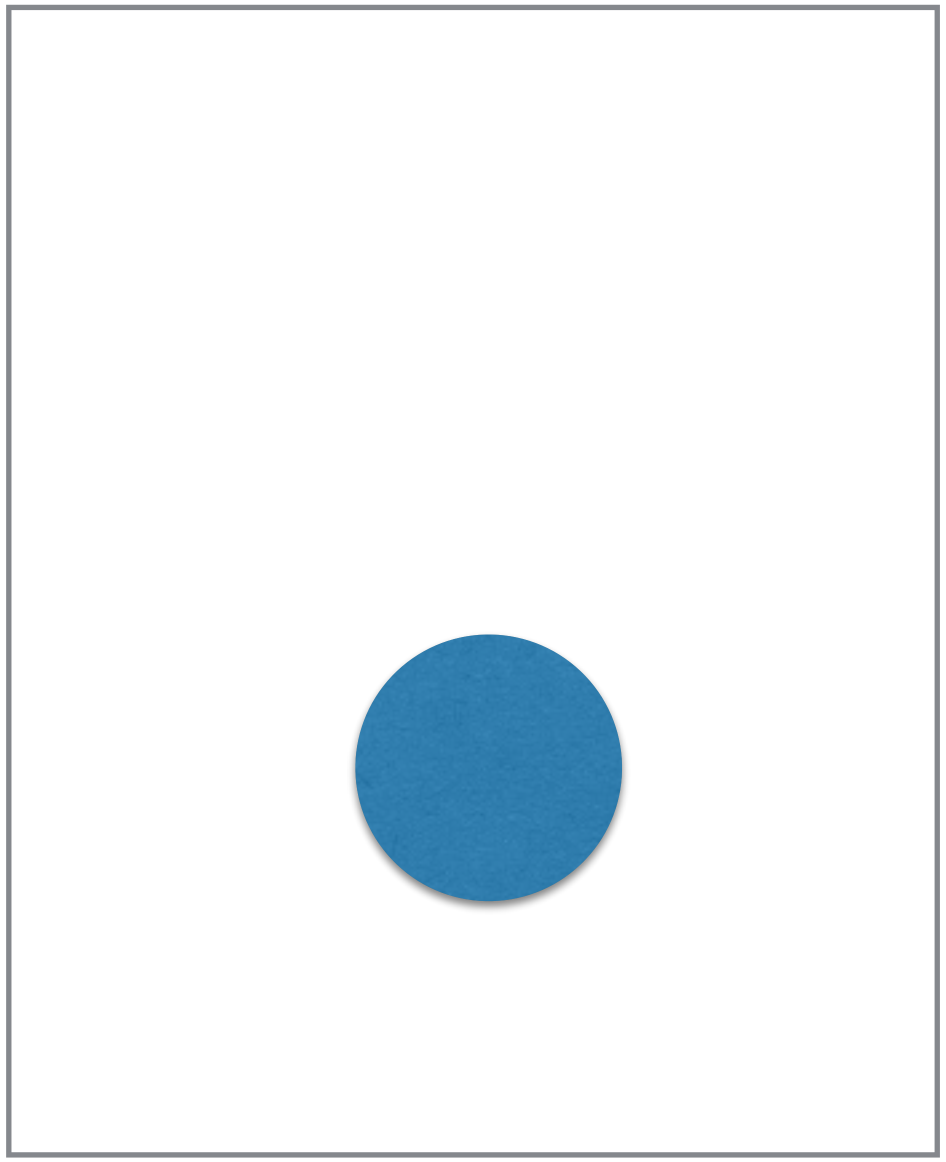
- Basic animation steps in a loop

1. Clearing the scene
2. Drawing the scene
3. Restoring the Canvas



Animations

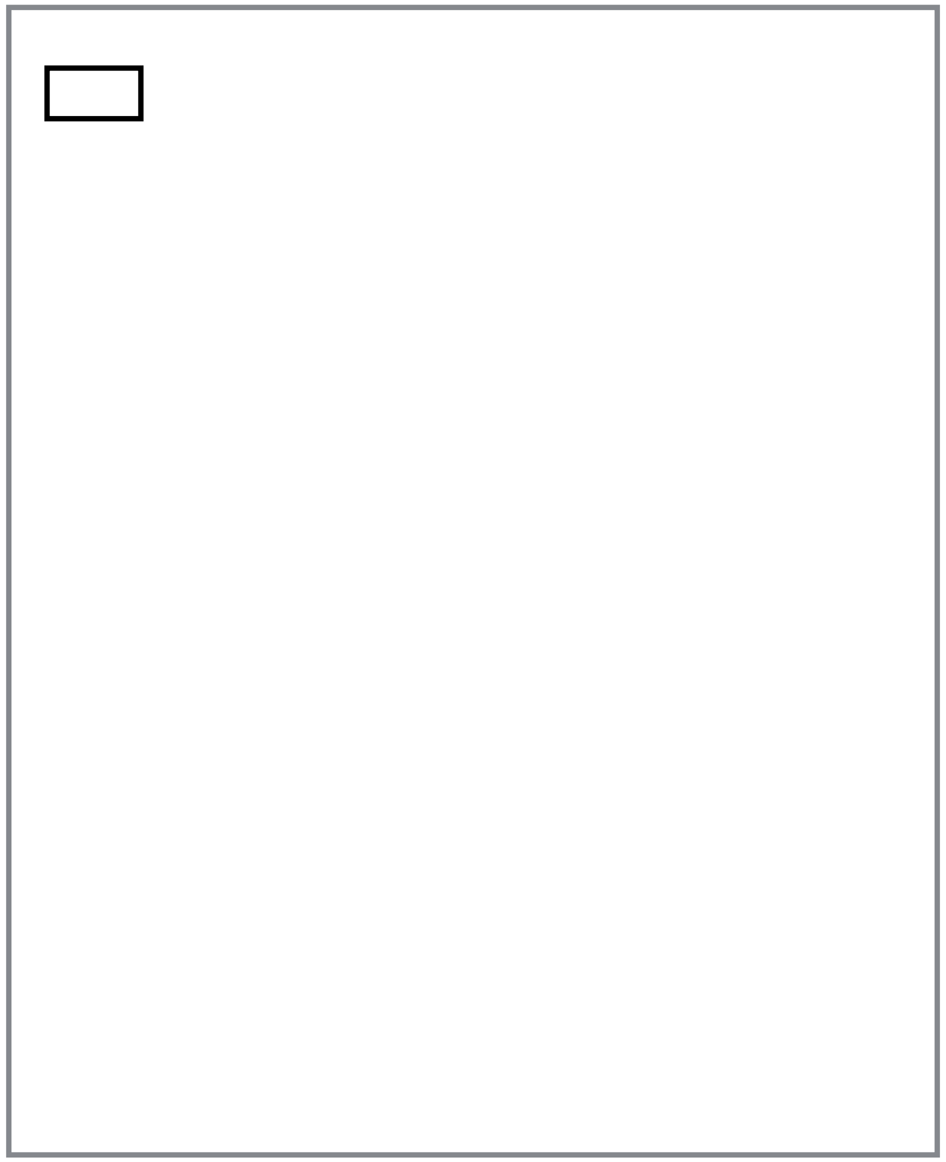
- Basic animation steps in a loop
  1. Clearing the scene
  2. Drawing the scene
  3. Restoring the Canvas



Animations

- Achieving loops by getting an animation frame
- encapsulate drawing in a method

```
var x = 20;
var vSpeed = 3;
var reqAf = window.requestAnimationFrame;

var draw = function() {
  clear();
  x += vSpeed;
  context.strokeRect(x, 100, 10, 5);

  animFrame = reqAf(draw);
};

animFrame = reqAf(draw);
```



Animations

- Achieving loops by getting an animation frame
- encapsulate drawing in a method

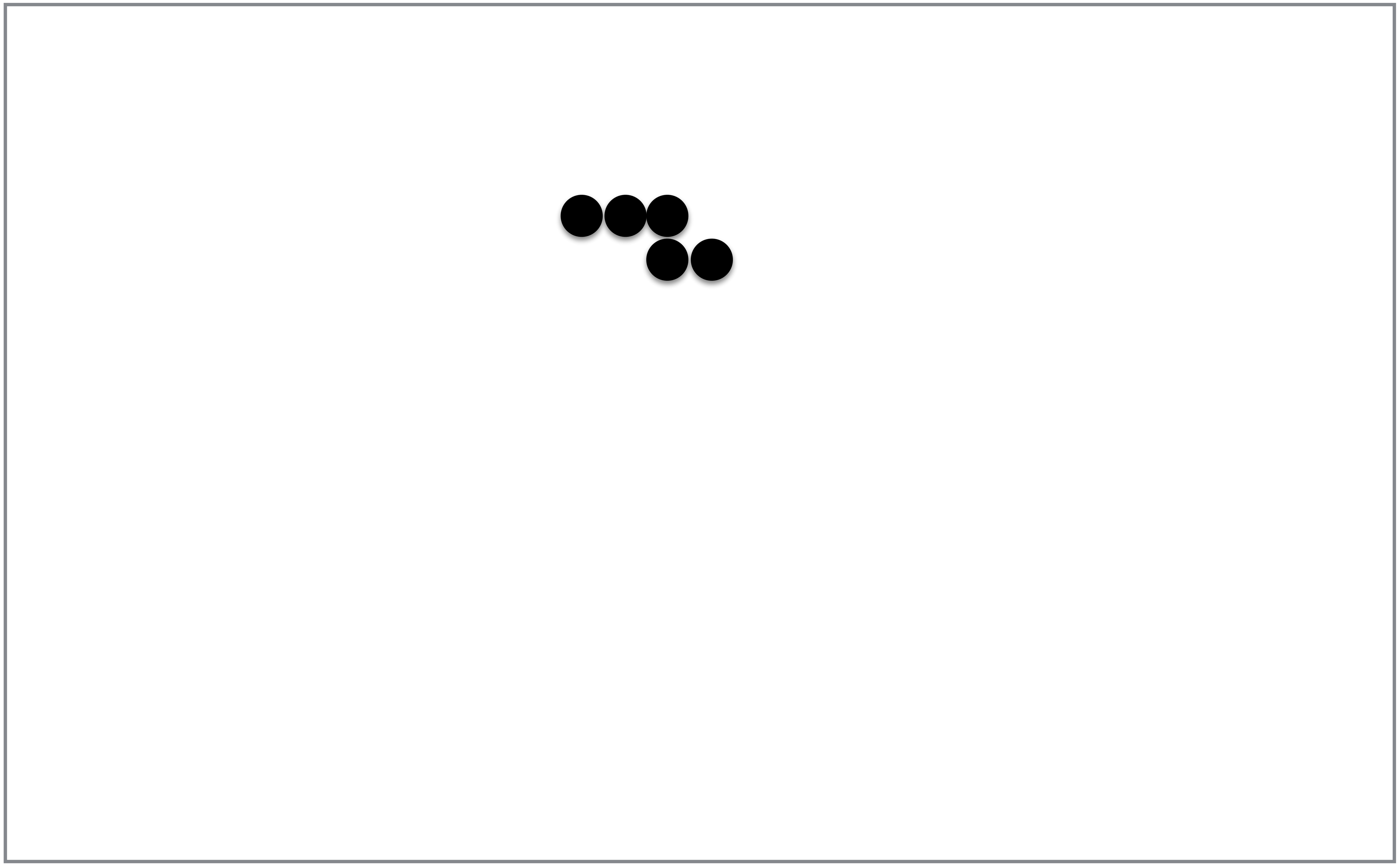
```
var x = 20;
var vSpeed = 3;
var reqAf = window.requestAnimationFrame;

var draw = function() {
  clear();
  x += vSpeed;
  context.strokeRect(x, 100, 10, 5);

  animFrame = reqAf(draw);
};

animFrame = reqAf(draw);
```

Animations

- Interacting with keyboard by just adding keyup Listener

```
var x = 20;
var vSpeed = 3;
var reqAf = window.requestAnimationFrame;

window.addEventListener('keyup',
  function(event) {
    switch (event.code) {
      case 'ArrowLeft':
        vSpeed = -3;
        break;
      case 'ArrowRight':
        vSpeed = 3;
        break;
      case 'Space':
        vSpeed = 0;
        break;
    }
});

var draw = function() {
  clear();
  x += vSpeed;
  context.strokeRect(x, 100, 10, 5);

  animFrame = reqAf(draw);
};

animFrame = reqAf(draw);
```

Let’s build a moving snake



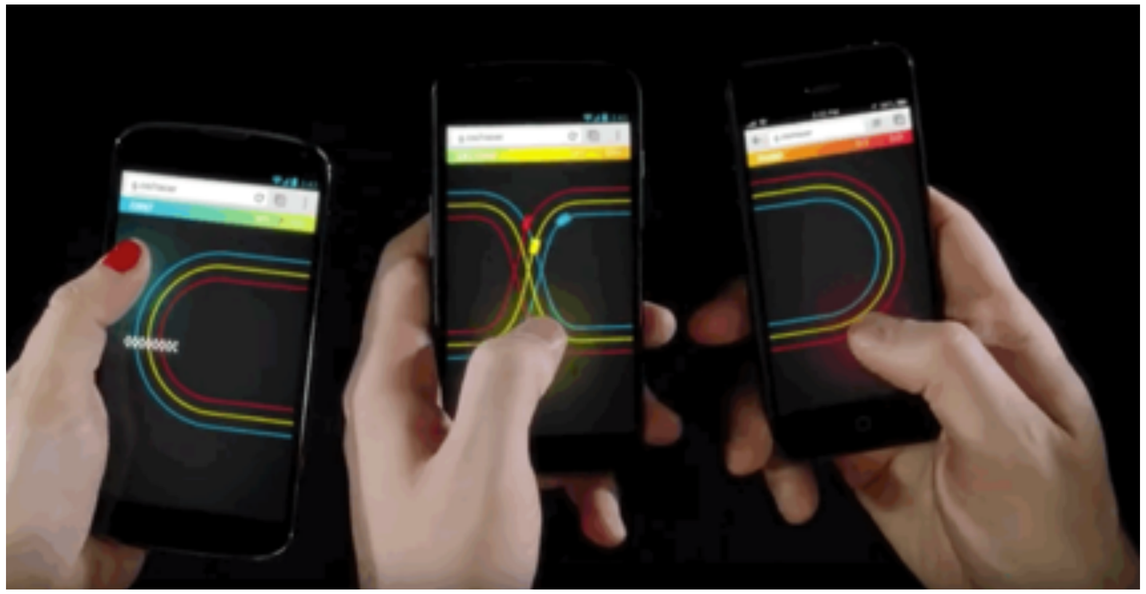
paper.js

What is paper.js?

- „The Swiss Army Knife of Vector Graphics Scripting“
- Scene Graph for vector graphics
- Support for Vector calculations

paper.js Showcases

- Foursquare's timemachine
  - visualizing checkins locations
- Google Chrome Experiment: Racer



Basic Types

- Simple „data“ structures:
  - Point
  - Size
  - Rectangle
- Not drawn to the view
- Needed for geometric calculations

```
var dot, size, rect;

dot = new paper.Point(10, 20);
console.log(dot);
// {x: 10, y: 20}

size = new paper.Size(15, 30);
console.log(size);
// {width: 15, height: 30}

rect = new paper.Rectangle(
    dot, size
);
console.log(rect);
// {x: 10, y: 20, width: 15,
//   height: 30}
```

A simple Path

- Creating a Path object

```
var path = new paper.Path();
```

A simple Path

- Creating a Path object

```
var path = new paper.Path();
```

- Moving to start coord

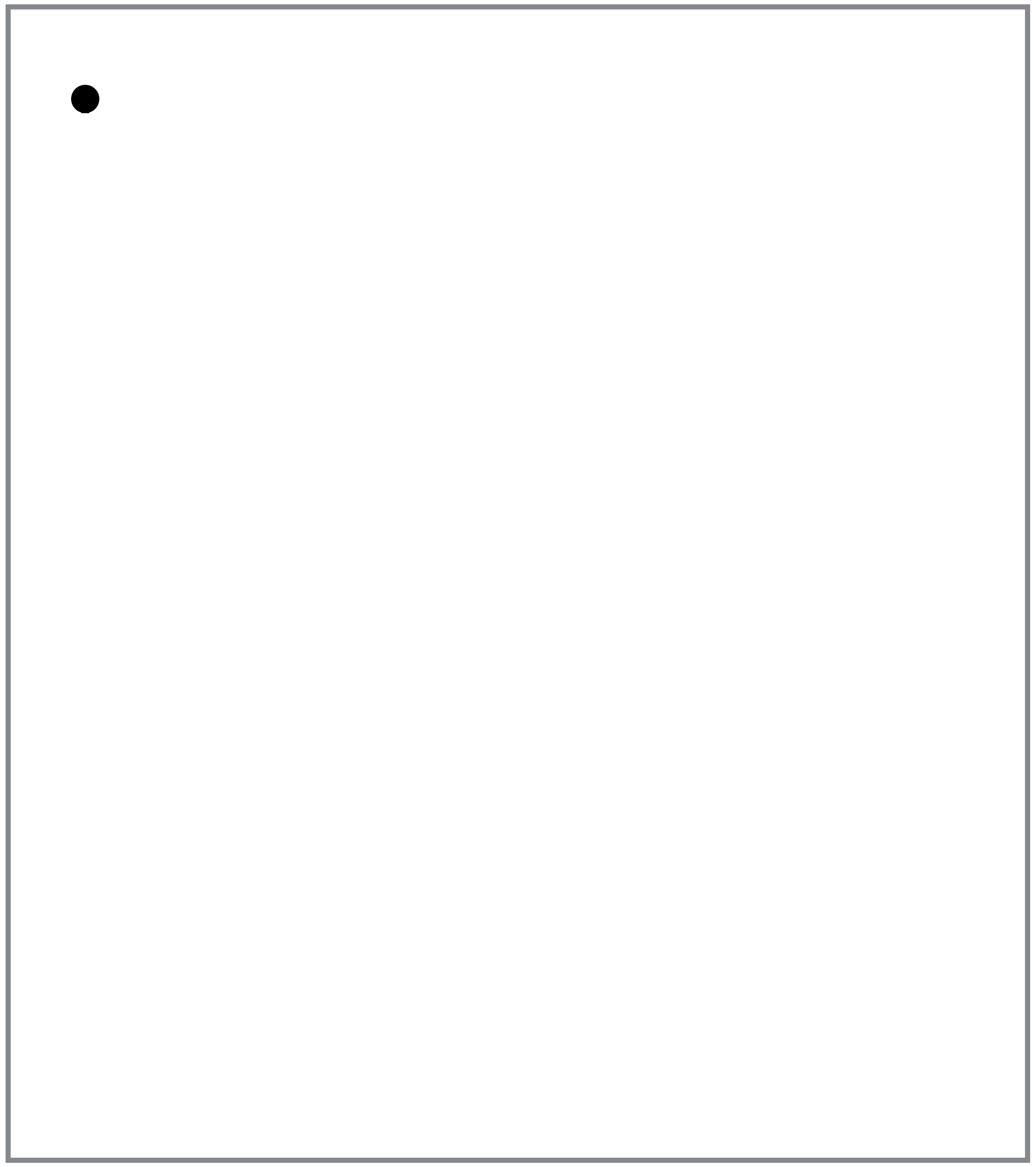
A simple Path

- Creating a Path object

```
var path = new paper.Path();
```

- Moving to start coord

```
    path.moveTo(10, 20);
```



A simple Path

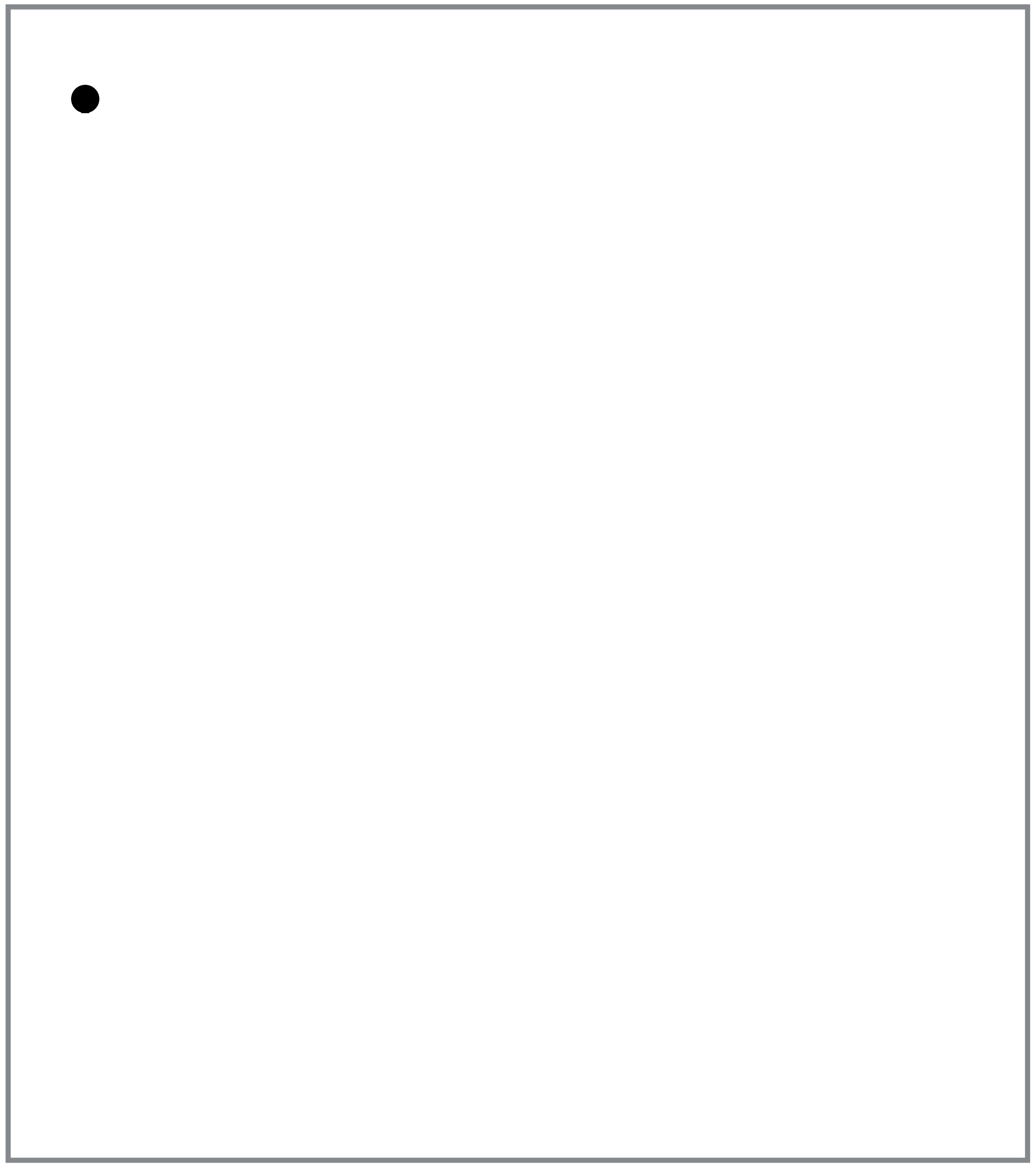
- Creating a Path object

```
var path = new paper.Path();
```

- Moving to start coord

```
    path.moveTo(10, 20);
```

- Draw a line



A simple Path

- Creating a Path object

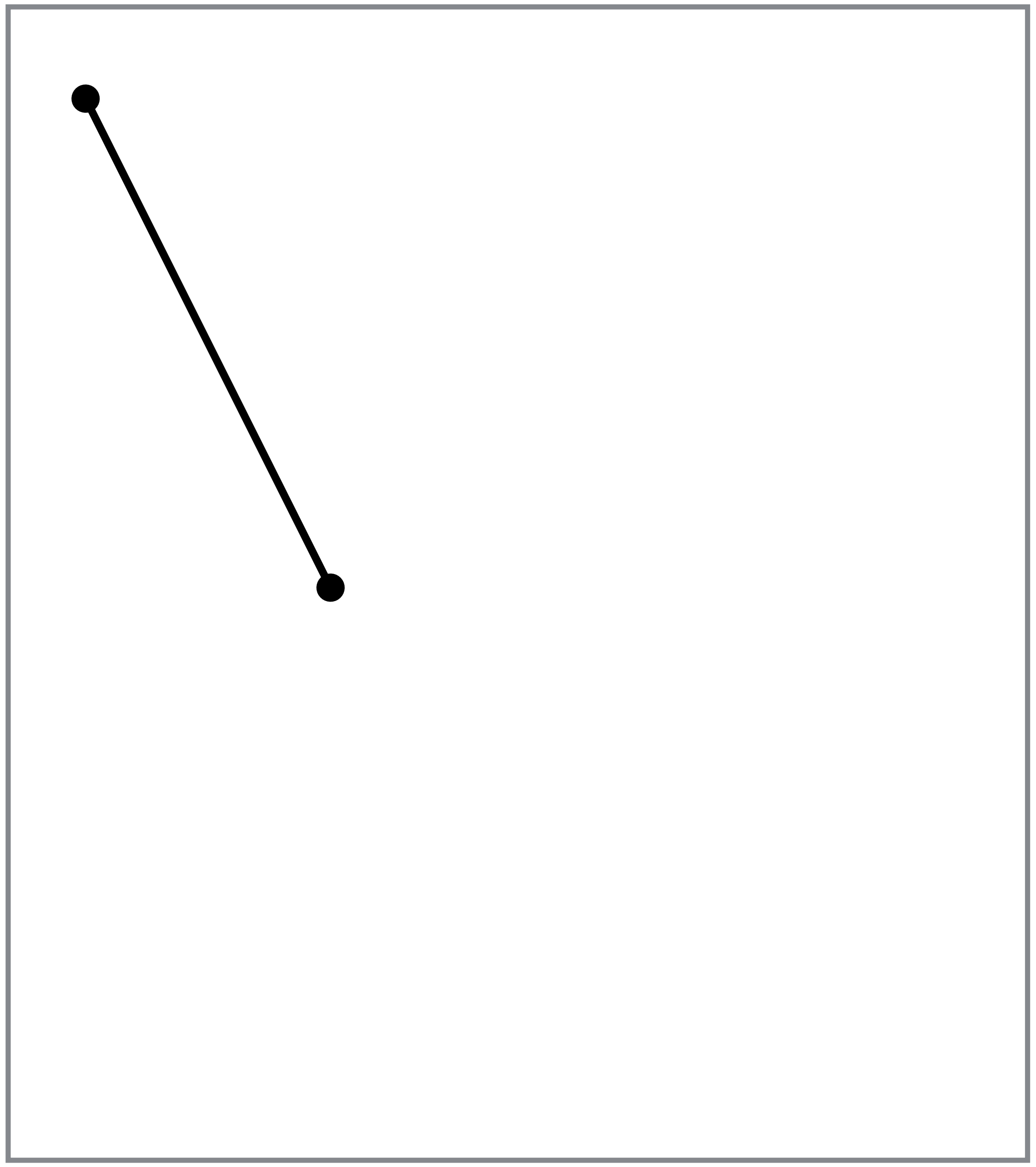
```
var path = new paper.Path();
```

- Moving to start coord

```
path.moveTo(10, 20);
```

- Draw a line

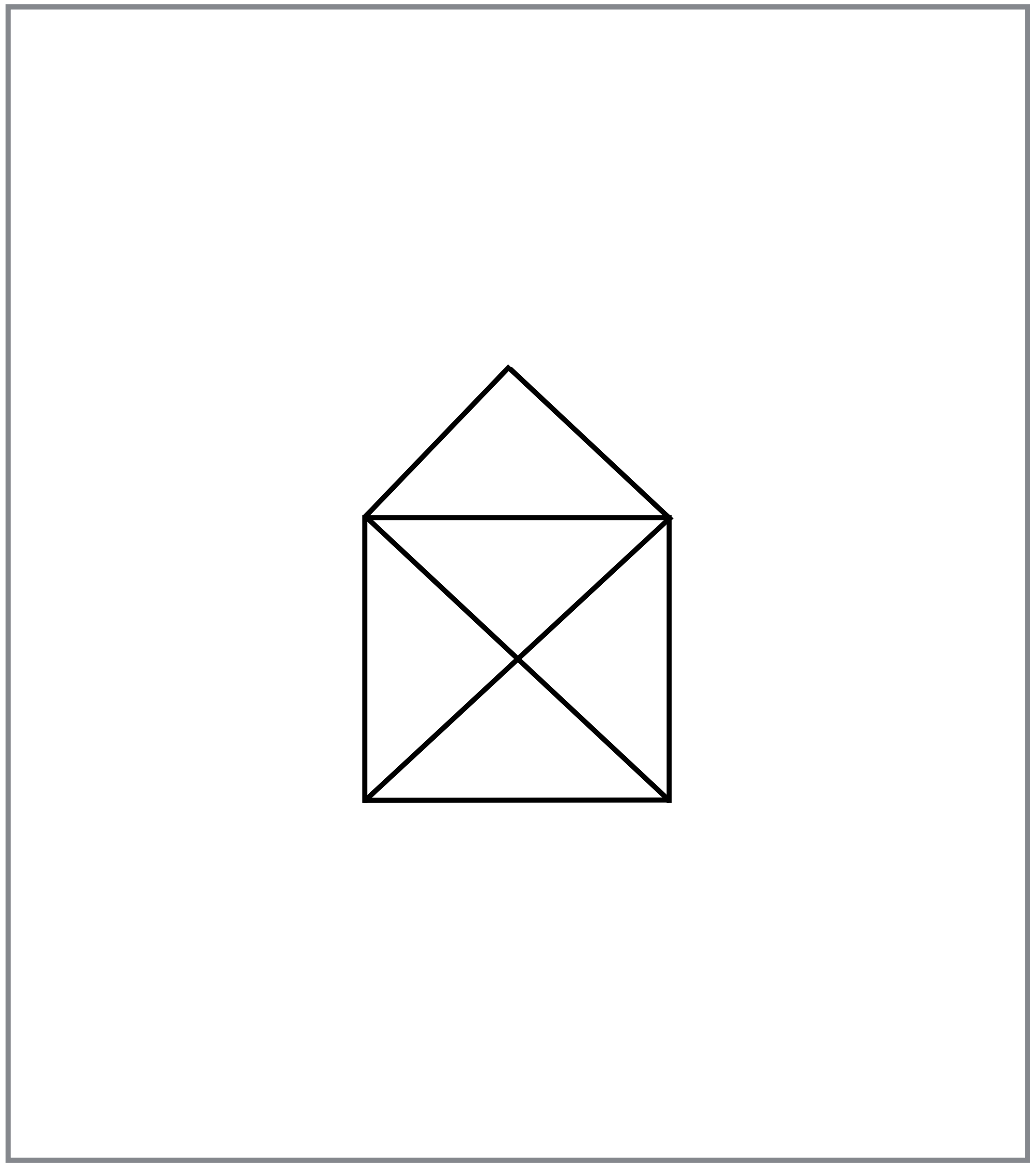
```
path.lineTo(100, 200);
```



Let’s draw a house

Things you’ll be needing:

```
// Creating a path
var path = new paper.Path();

// moving the "cursor" to a
// start position
path.moveTo(0, 0);
// drawing a line from „cursor"
// position to end position
path.lineTo(100, 200);
// additional: choosing a stroke
// color (before calling lineTo)
path.strokeColor = 'black';
```



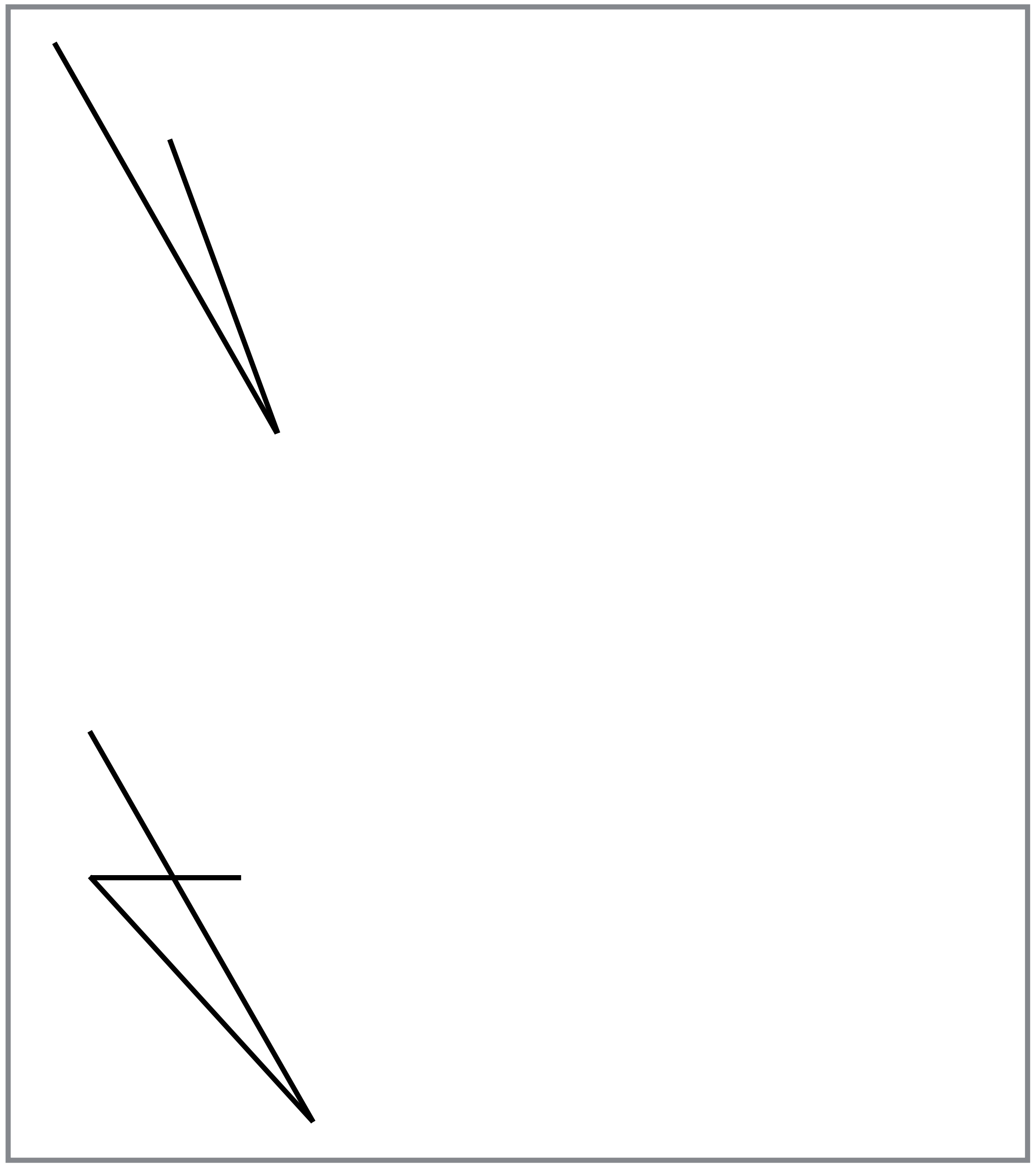
Complex Paths

- Adding Segments

```
var path = new paper.Path();

path.strokeColor = ‚black';

path.add(10, 10);
path.add(100, 200);
path.add(50, 50);
```

- Inserting Segments

```
path.insert(
  2,
  new paper.Point(10, 50)
);
```



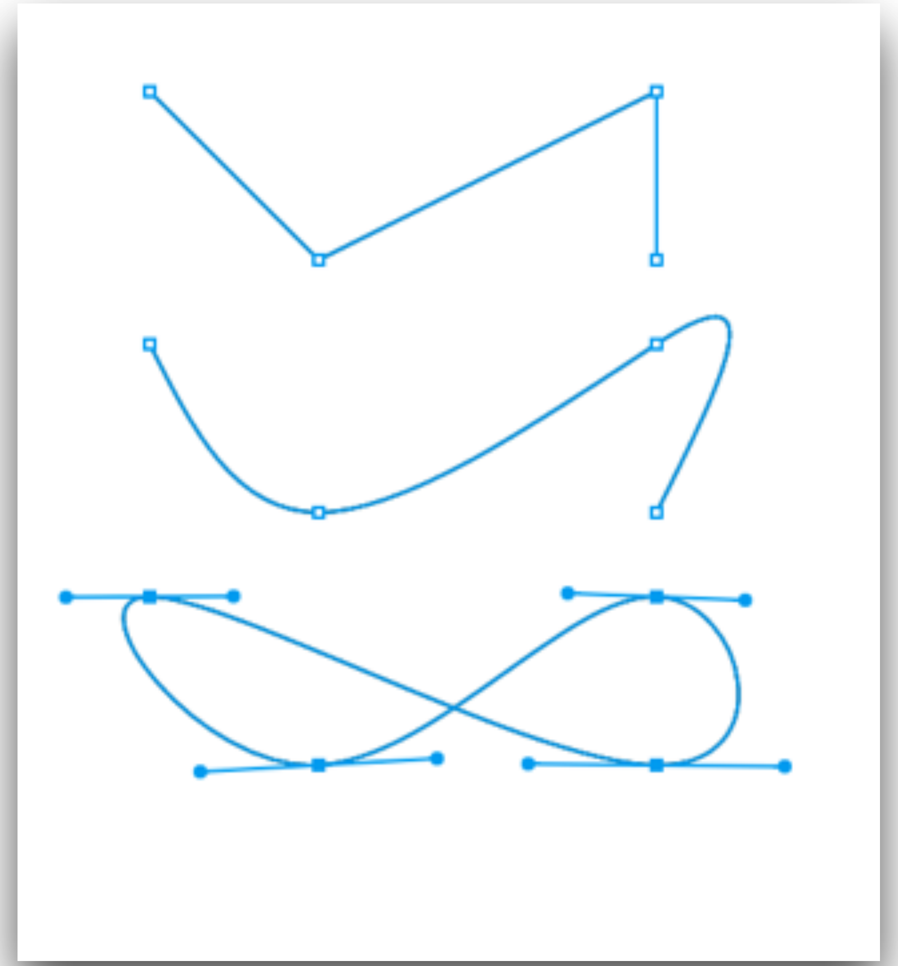
Path Segments

- Path consists of Segments
    - forming the Path

```
// add segment
path.add(10, 200);
path.insert(
  0, new paper.Point(10, 5)
);
path.removeSegment(0);
```

- Path can be smoothed around Segments

```
// automatically smoothing
path.smooth();
```



Path Segments

- can be selected

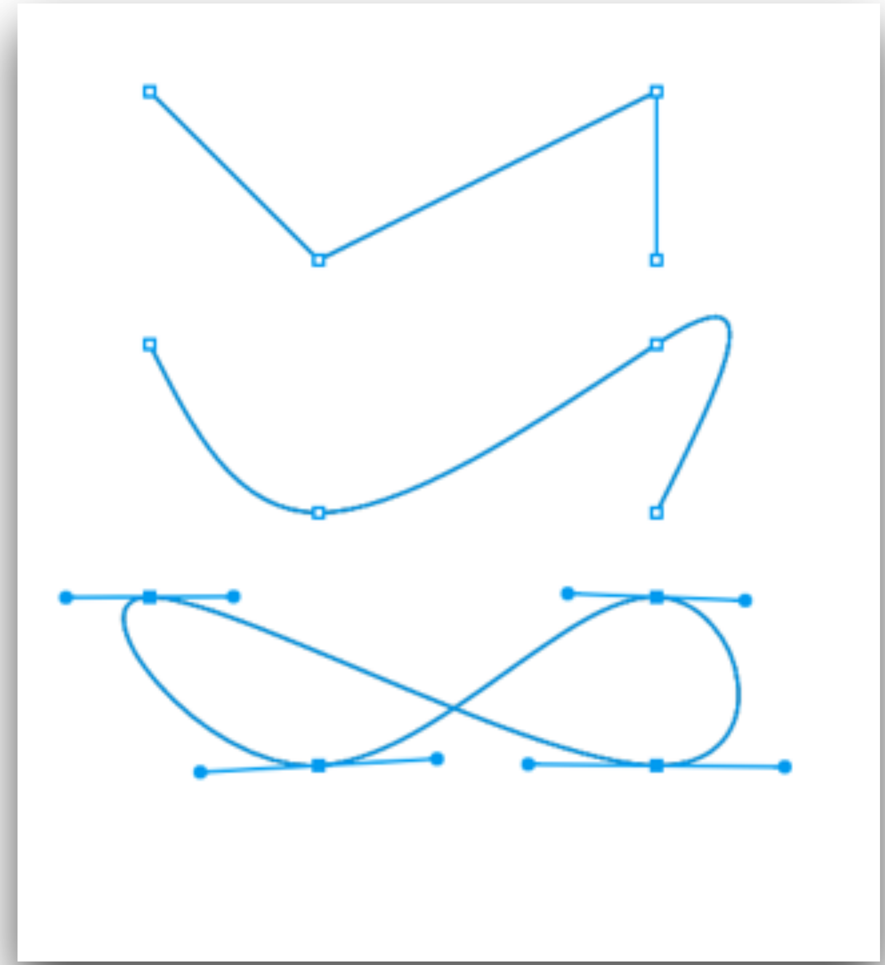
```
path.selected = true;
```

- Path could be automatically closed

```
path.closed = true;
path.fullySelected = true;
```

- Existing predefined Paths

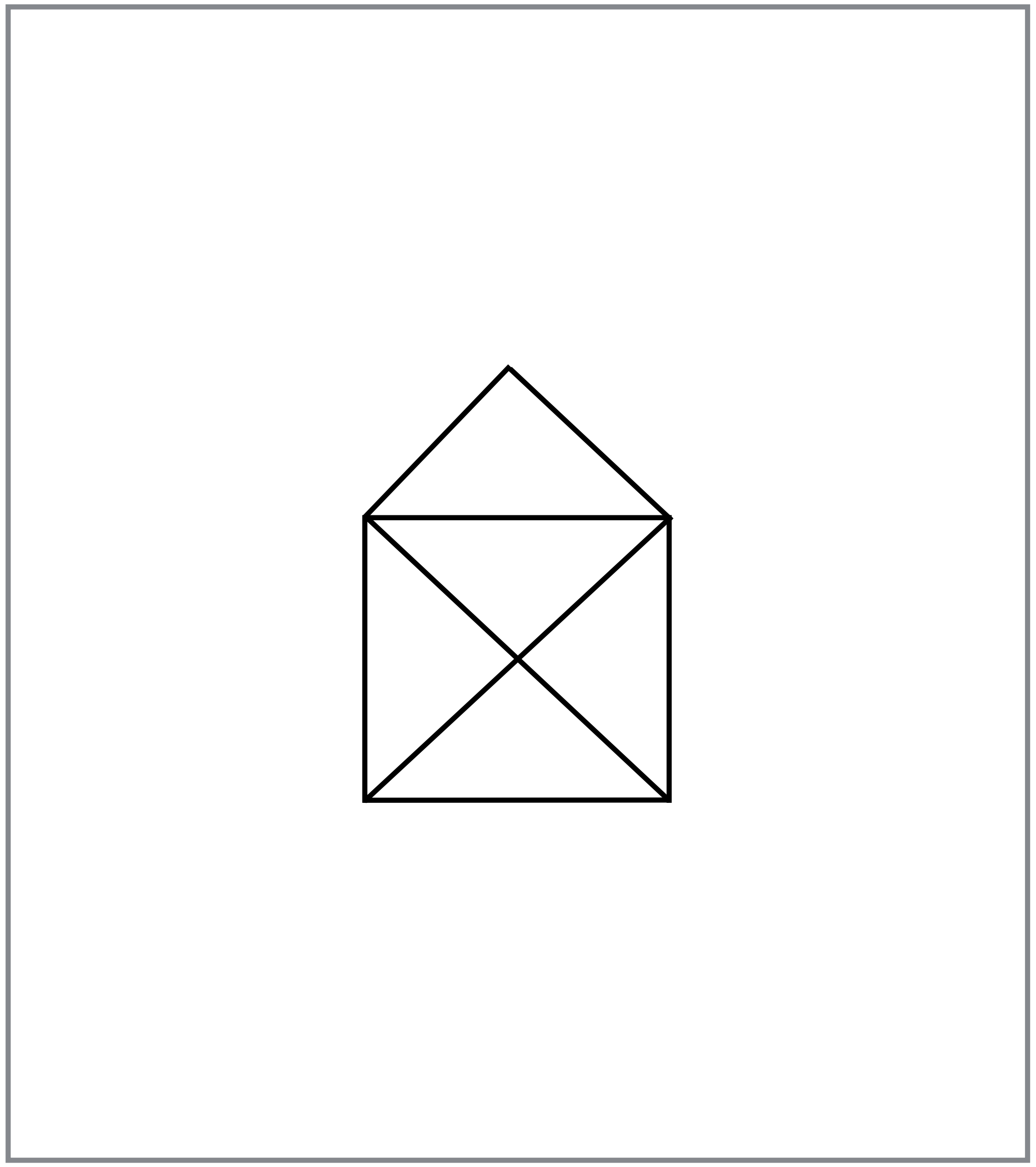
```
new paper.Path.Circle(point, r);
new paper.Path.Rectangle(rect);
new paper.Path.RoundRectangle(
  rect, cornerSize
);
new paper.Path.RegularPolygon(
  point, numSides, r
);
```



Let’s draw a house again

Things you’ll be needing:

```
var rect = new paper.Rectangle(
  100, 100, 200, 200
);
var path = new paper.Path.Rectangle(
  rect
);

// inserting segments
path.add(100, 100);
path.insert(
  3, new paper.Point(100, 100)
);
// removing existent segments
path.removeSegment(1);
// not closing the path
path.closed = false;
// additional: choosing a stroke
// color (before calling lineTo)
path.strokeColor = 'black';
```



Shapes

- Predefined shapes
  - Circle

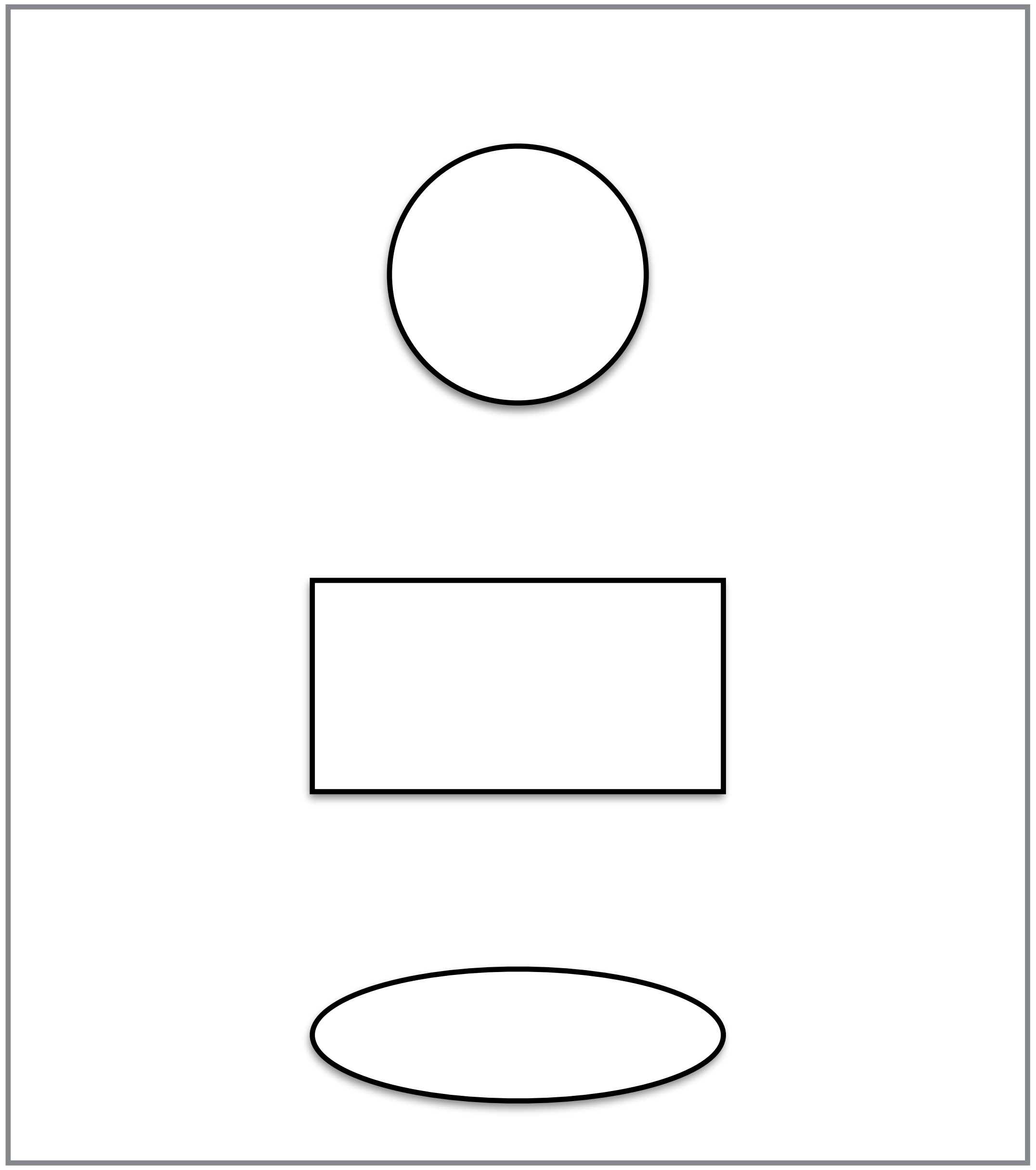
```
var point = new paper.Point(200, 200),
  radius = 100;
new paper.Shape.Circle(point, radius);
```

  - Rectangle

```
var point = new paper.Point(200, 200),
    size = new paper.Size(200, 100);
new paper.Shape.Rectangle(
    point,
    size
);
```

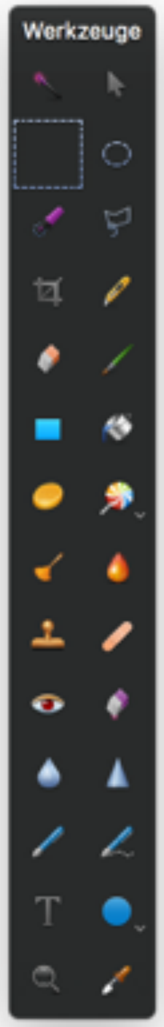
  - Ellipse

```
new paper.Shape.Ellipse(
  new paper.Rectangle(point, size)
);
```



Tools

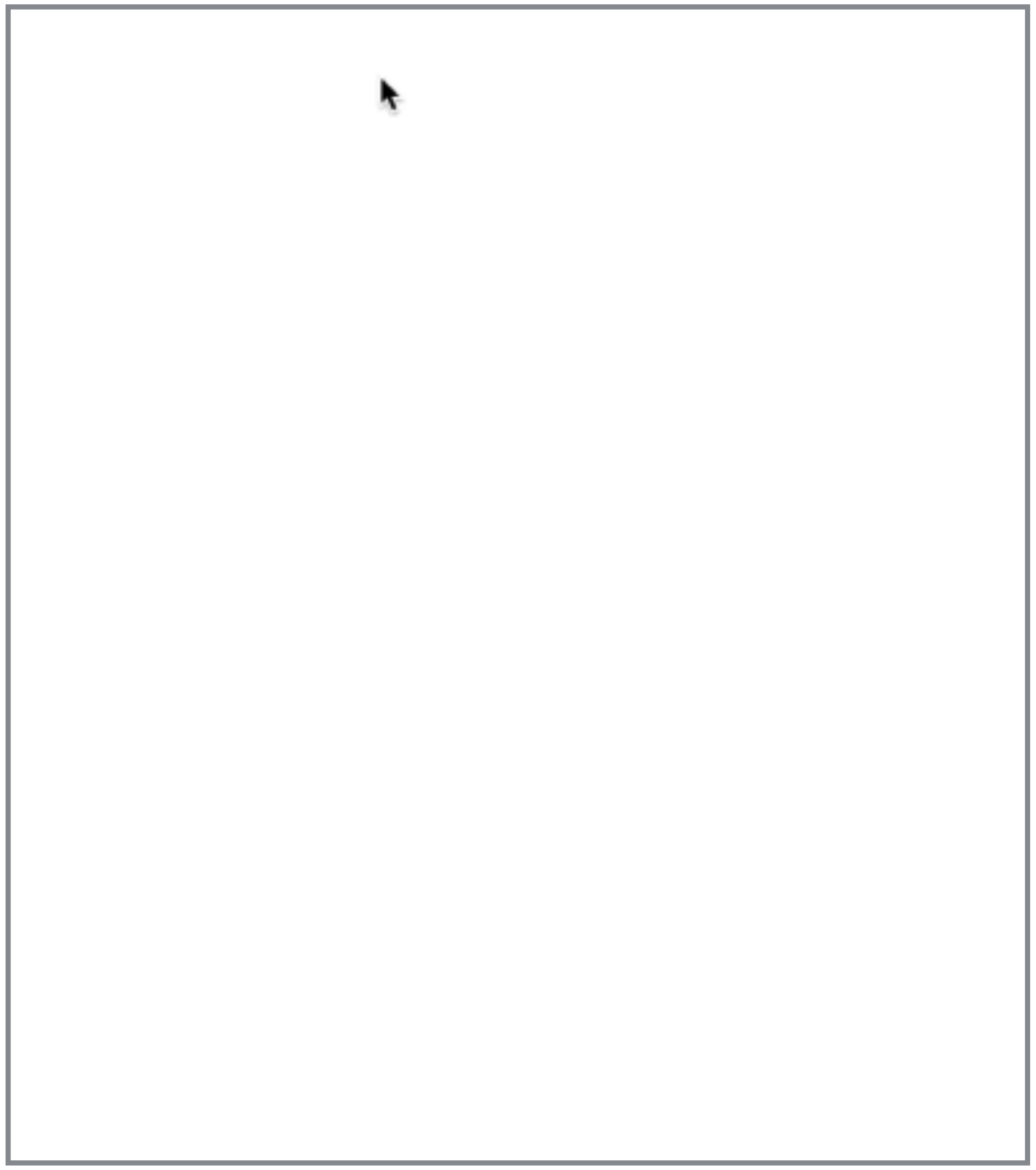
- Used to react to Events (Mouse, Keyboard)
- Only one tool can be activated at the same time per view



Bildquelle: Pixelmator Toolwindow

Tools

- How to write them?



Tools

- How to write them?
  - Simple!

Tools

- How to write them?
  - Simple!

```
var tool = new paper.Tool();

tool.onMouseUp = function(event)
{
  var path = new paper.Path();

  path.add(event.downPoint);
  path.add(event.point);
};
```

Mouse Event object

- Special paper event containing useful information:
    - point
    - delta
    - middlePoint
    - downPoint

Mouse Event object

- Special paper event containing useful information:
  - point
  - delta
  - middlePoint
  - downPoint

```
var tool = new paper.Tool();

var path = new paper.Path();
path.selected = true;
path.strokeColor = 'black';

tool.onMouseDown = function(event) {
    path.add(event.point);

    var middleCirc = new paper.Shape.Circle(
        event.middlePoint, 5
    );
    middleCirc.fillColor = 'red';
};
```

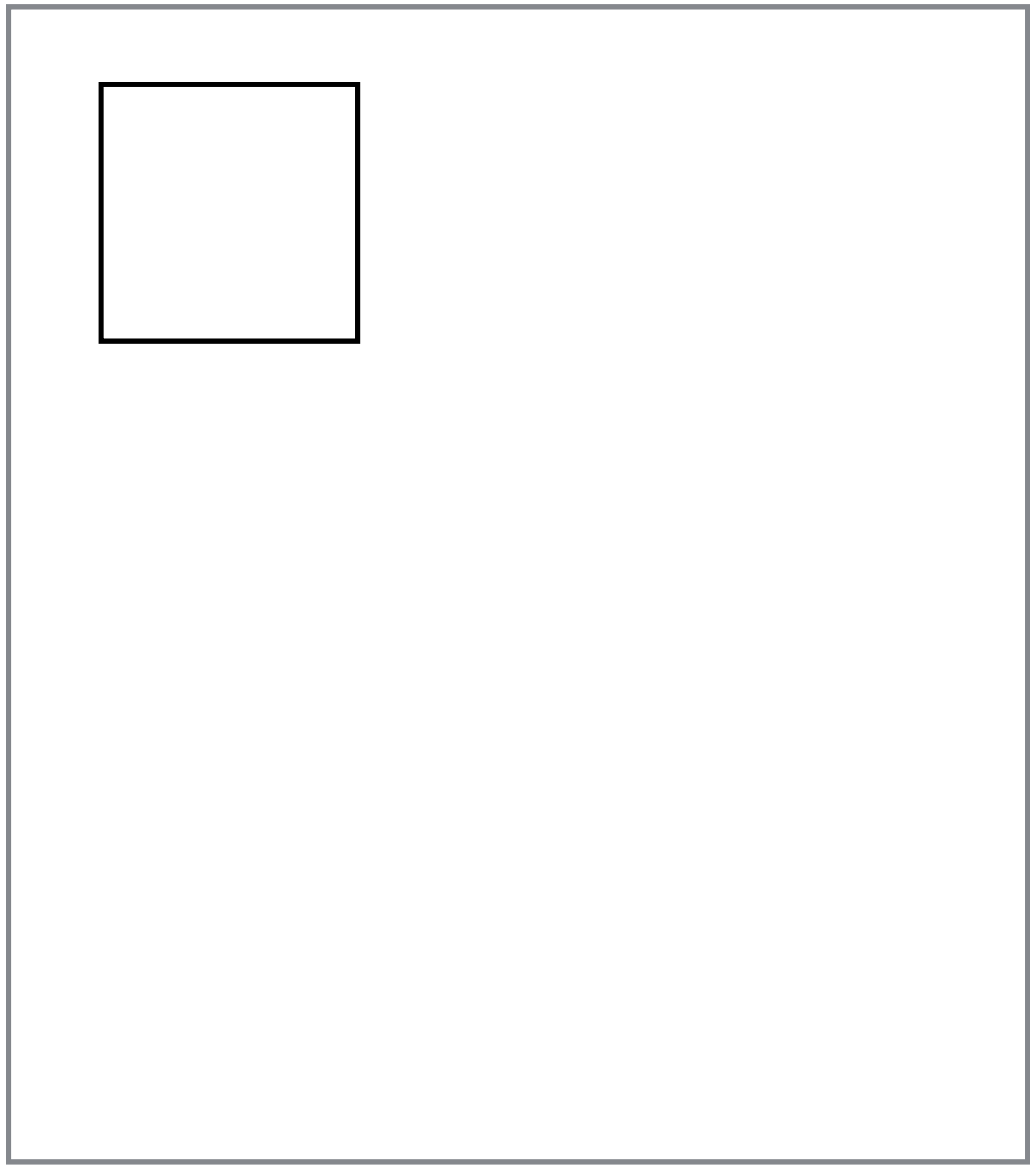
Mouse Event object

- Special paper event containing useful information:
  - point
  - delta
  - middlePoint
  - downPoint

event.point:
event.delta:
event.middlePoint:
event.downPoint:

Simple draw tool

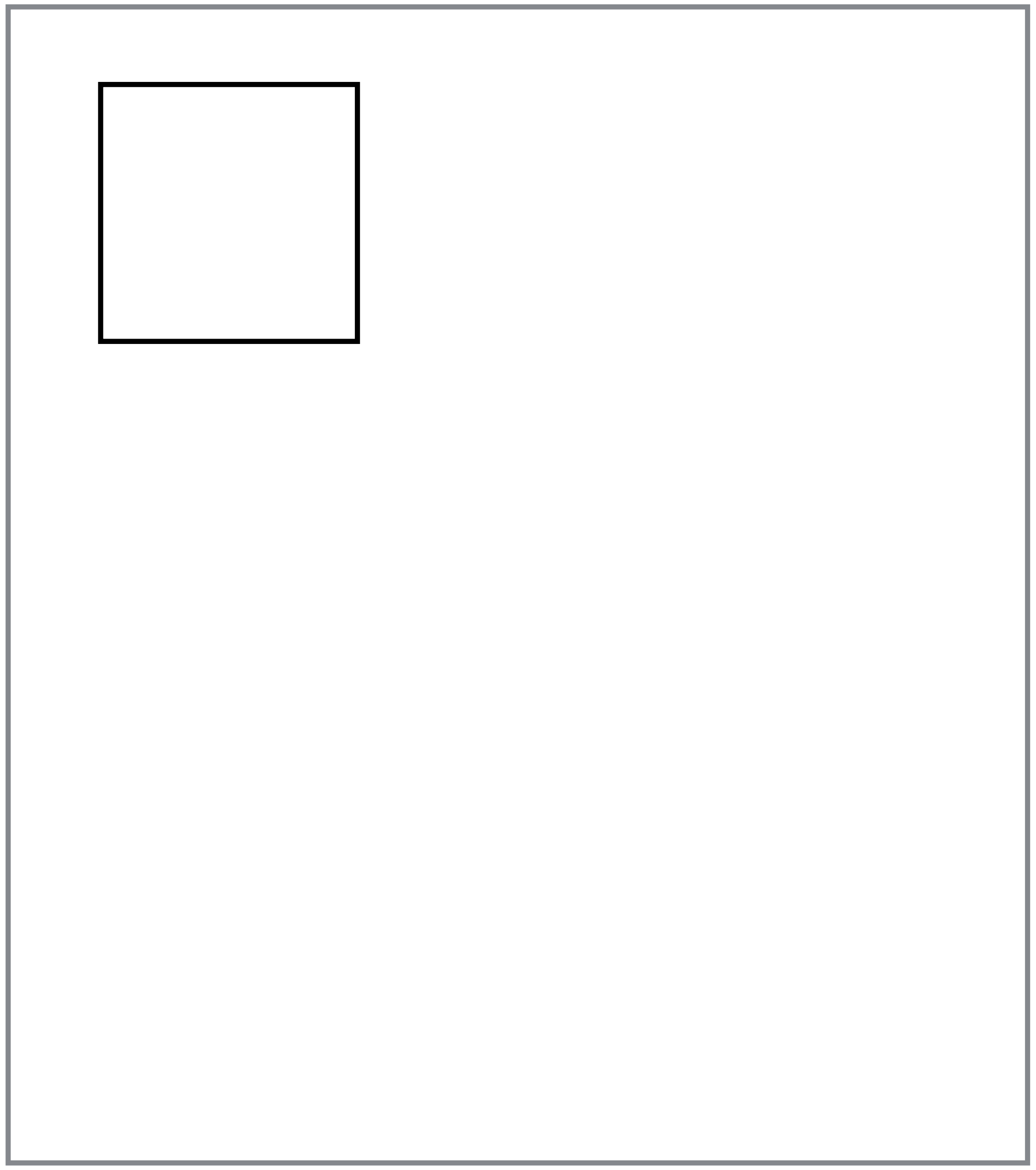
- Drawing tool to draw rectangles
- Move path segments



Simple draw tool

- Drawing tool to draw rectangles
- Move path segments

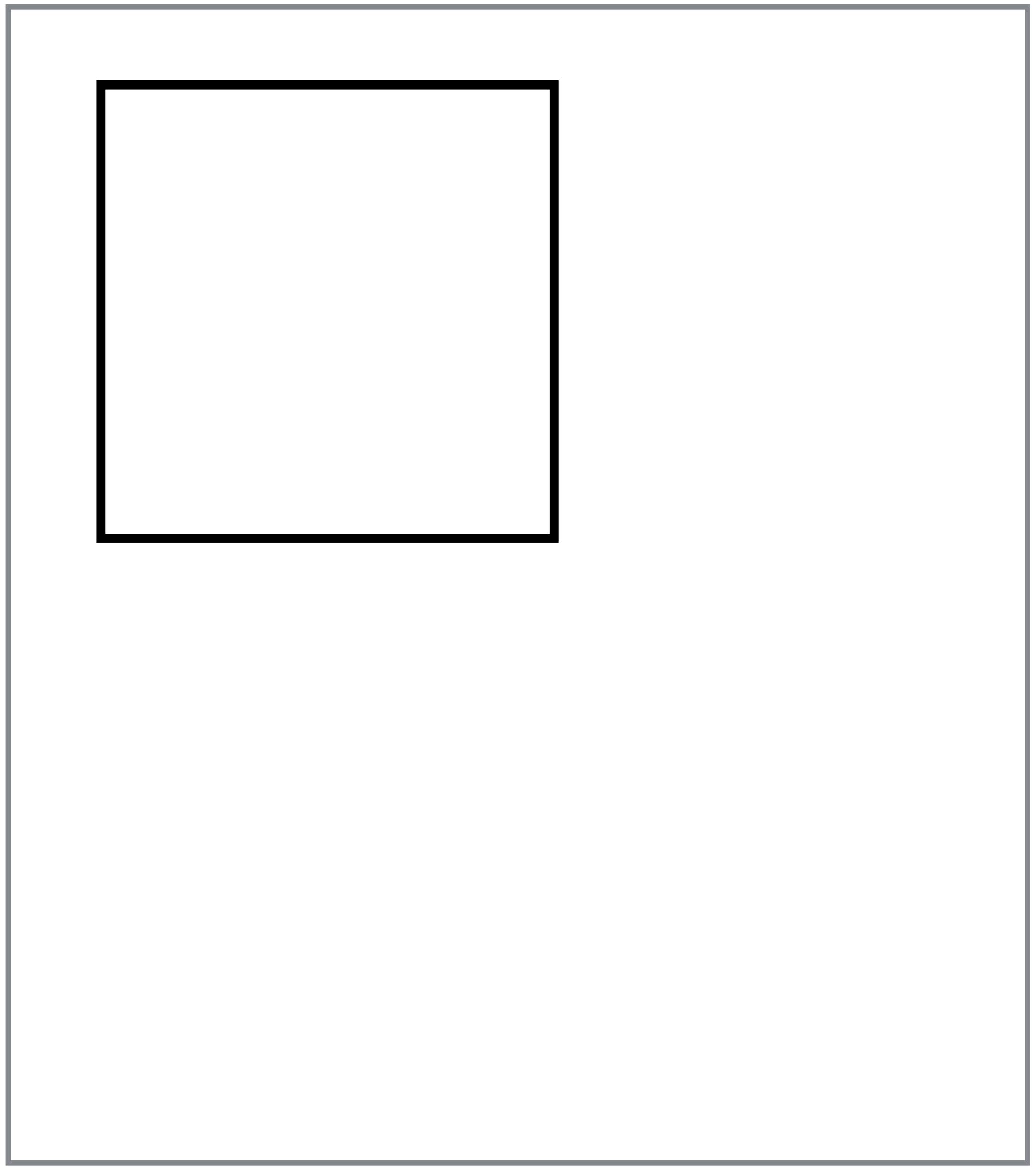
```
var tool = new paper.Tool();

tool.onMouseDown =
function(event) {};

tool.onMouseDrag =
function(event) {};

tool.onMouseUp =
function(event) {};
```



Simple draw tool

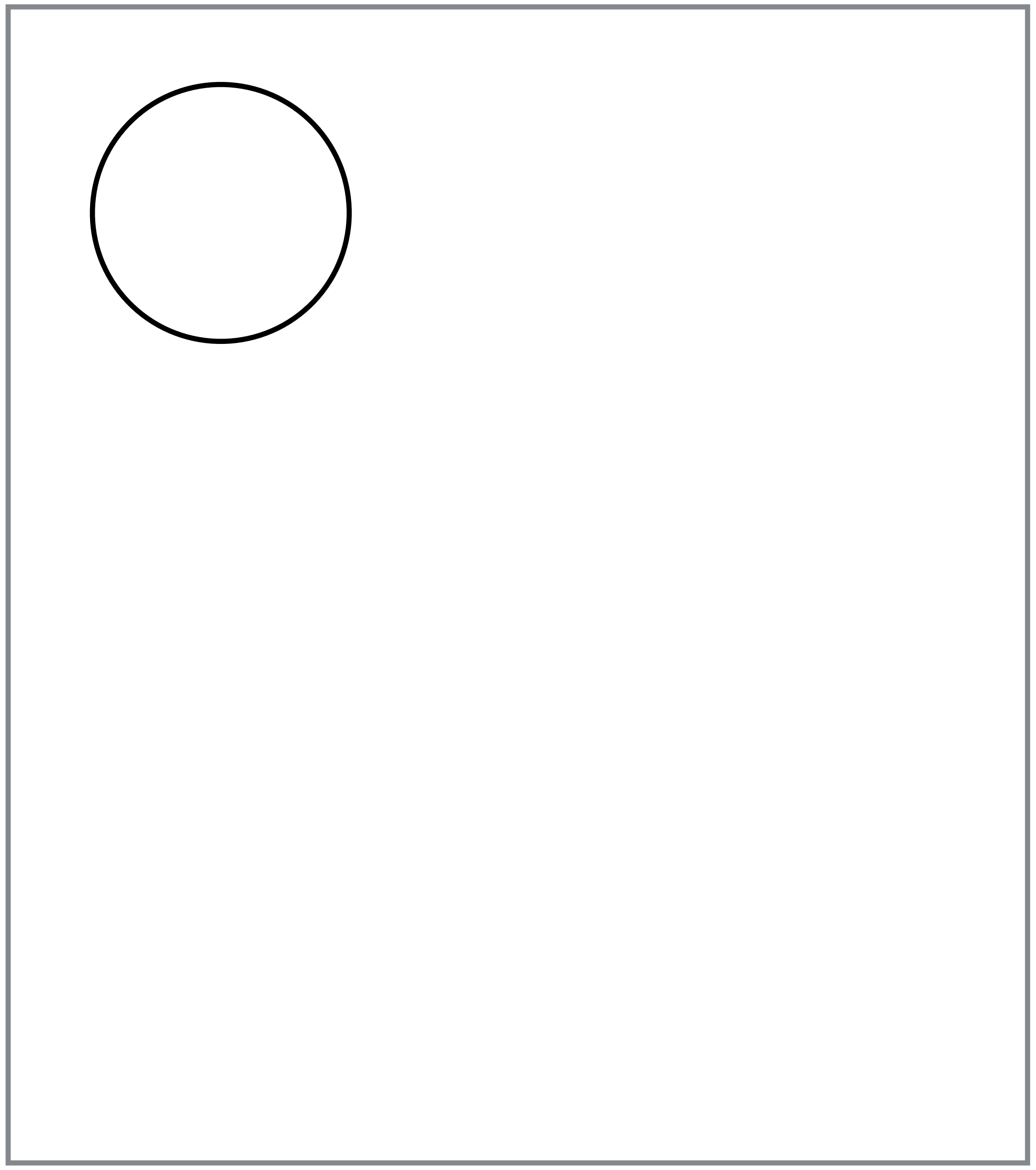
- Drawing tool to draw rectangles
- Move path segments

```
var tool = new paper.Tool();

tool.onMouseDown =
function(event) {};

tool.onMouseDrag =
function(event) {};

tool.onMouseUp =
function(event) {};
```



Multiple draw Tools

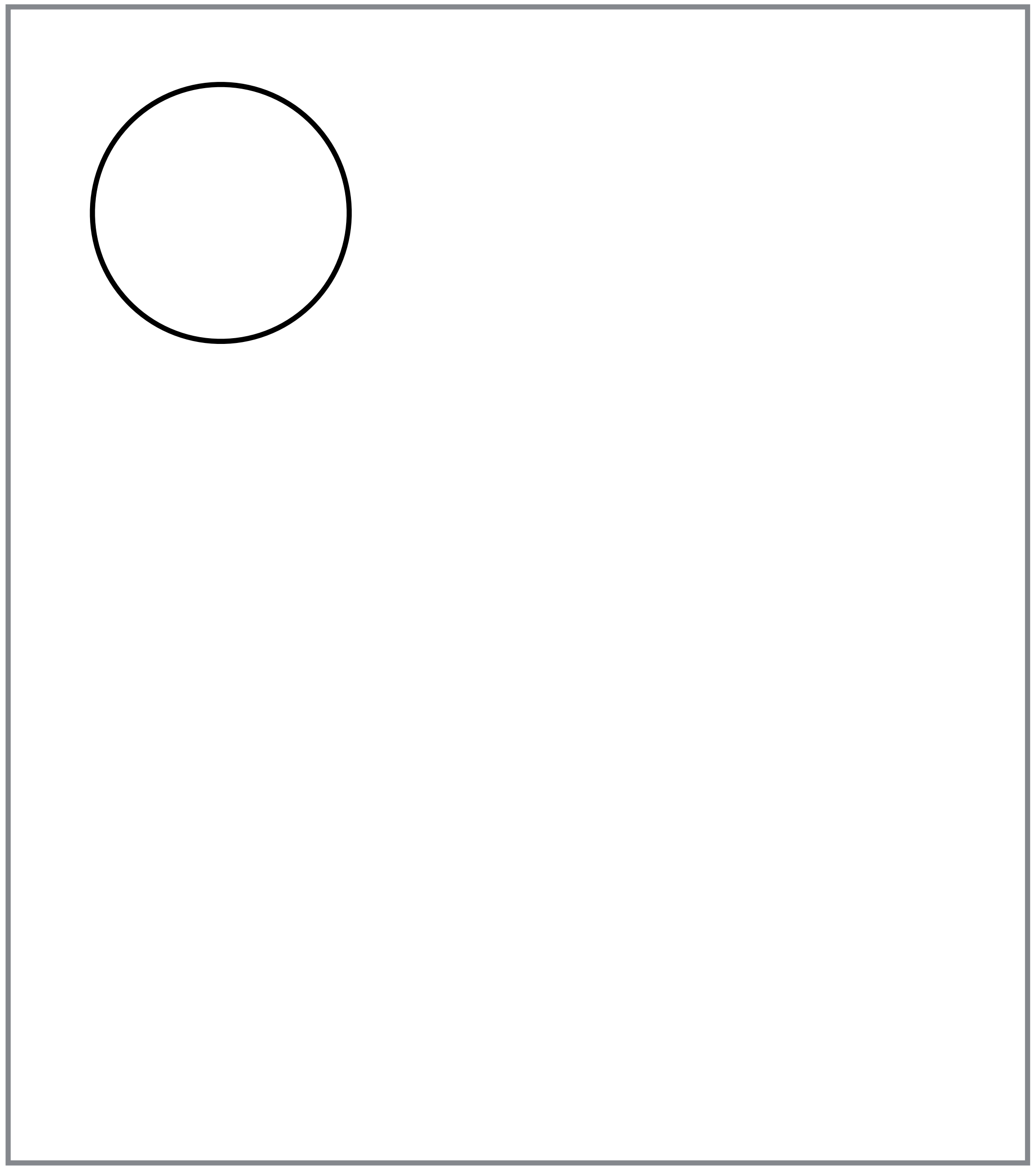
- Only one tool can be active at the same time
- Changing between tools by calling `activate()` on tool



Multiple draw Tools

- Only one tool can be active at the same time
- Changing between tools by calling `activate()` on tool

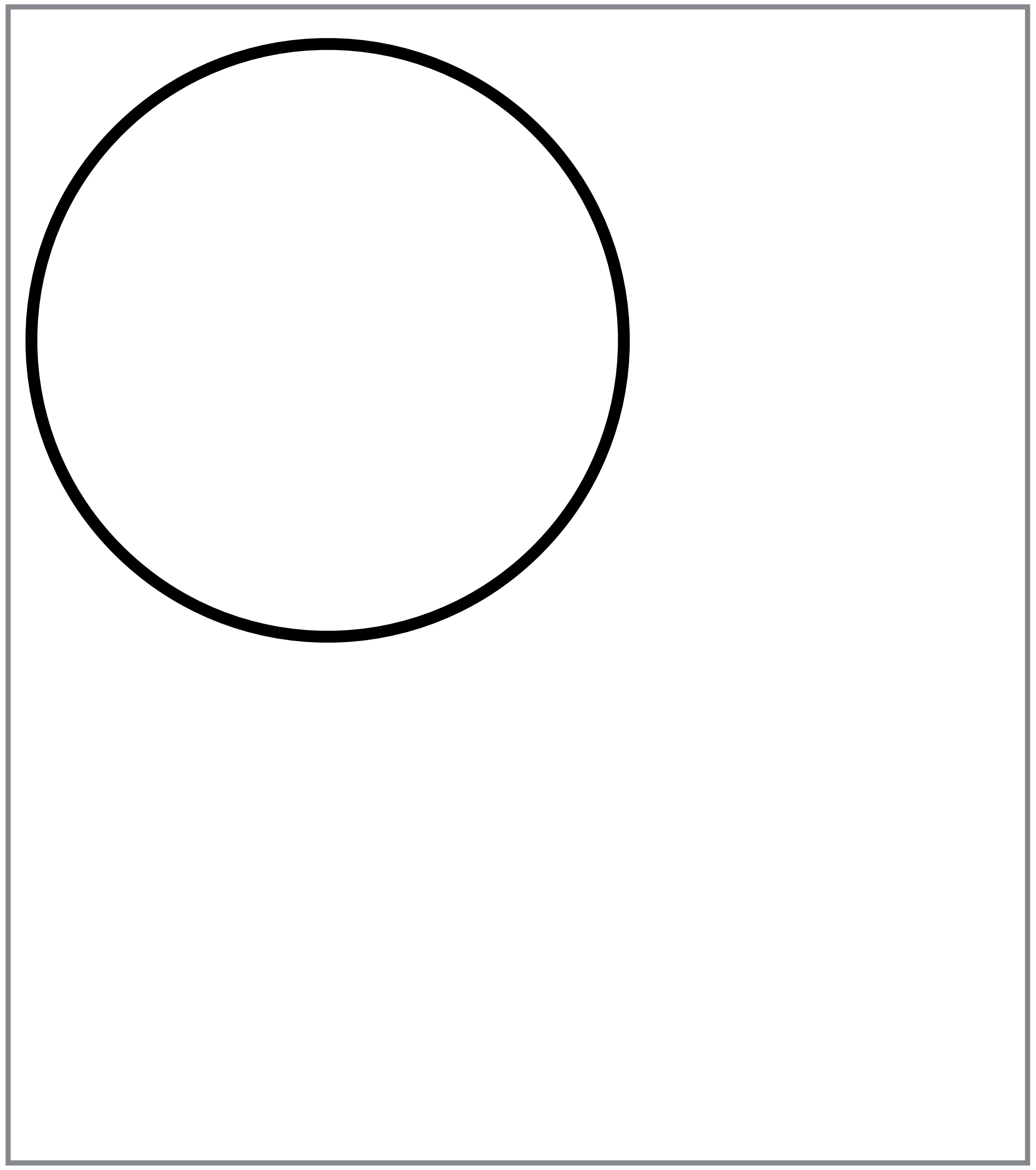
```
var tool = new paper.Tool();

tool.onMouseDown =
function(event) {};

tool.onMouseDrag =
function(event) {};

tool.onMouseUp =
function(event) {};
```



Multiple draw Tools

- Only one tool can be active at the same time
- Changing between tools by calling `activate()` on tool

```
var tool = new paper.Tool();

tool.onMouseDown =
function(event) {};

tool.onMouseDrag =
function(event) {};

tool.onMouseUp =
function(event) {};
```



HitTesting

- Method for testing if a object has been hit
- Things to match are configurable
- Considers layers

HitTesting

- Method for testing if a object has been hit
- Things to match are configurable
- Considers layers

```
var hit = paper.project.hitTest(
  event.point,
  {
    fill: true
  }
);

hit.type === 'fill'
// true for a filled shape

hit === null
// if no hit was found
```

HitTesting

- Implement a HitTest
- Select the matched shape
- Make it moveable by arrow keys

```
var tool = new paper.Tool();

// selecting a shape by mouse
tool.onMouseUp =
function(event) {};

// moving the shape
tool.onKeyDown =
function(event) {};
```

Animations

- paper has its own frame-handler
- FrameEvent:
  - count
  - time
  - delta

```
var path = new
paper.Path.Rectangle({
  point: [75, 75],
  size: [75, 75],
  strokeColor: 'black'
});

paper.view.onFrame =
function() {
  path.rotate(3);
};
```

Symbols

- Symbols = existing Paths that can be placed
- Reuse of existing complex or simple Paths

```
var path = new paper.Path.Circle({
  point: [75, 75],
  radius: 10,
  fillColor: 'black'
});

var symbol = new
paper.Symbol(path);

symbol.place(
  new paper.Point(20, 10)
);
```

Now it’s your turn

What we’ve learned

What we’ve learned

- „Down to the roots“ handling of `<canvas>`

What we’ve learned

- „Down to the roots“ handling of `<canvas>`
- Basic animation handling in the web

What we’ve learned

- „Down to the roots“ handling of `<canvas>`
- Basic animation handling in the web
- How to use paper.js for drawing things

What we've learned

- „Down to the roots“ handling of `<canvas>`
- Basic animation handling in the web
- How to use paper.js for drawing things
- And hopefully that using canvas is fun! :-)

Thanks!

Further questions?

@schaumiii

Useful resources

Useful resources

- paperjs.org Documentation
- Live sketchpad: http://sketch.paperjs.org
- Mozilla Developer Network: Canvas