INTRO TO CANVAS

WHAT IS CANVAS?

- Canvas is a drawing surface
- We can use it to draw anything
- Things from graphs to games
- You'll likely be using it for your projects

THE CANVAS ELEMENT

- We can create a canvas using the <canvas> tag
- We usually include 4 properties
  - id: The name we will refer to the canvas by in JavaScript
  - Width: The width of the Canvas
  - Height: the height of the Canvas
  - style: We use it to give the Canvas a border to so we know where it is on the page.

```
<canvas id="drawingSurface"
    style="border-style: solid" width="600px"
    height="600px"></canvas>
```

THE RENDERING CONTEXT

- Before we can draw onto the canvas we must get the rendering context
- We get the context from the Canvas, which we have called drawingSurface using the id property

```
let drawingSurface =

document.getElementById("drawingSurface");
let ctx = drawingSurface.getContext("2d");
```

So we are getting the context with the getContext method, and storing it in the variable ctx

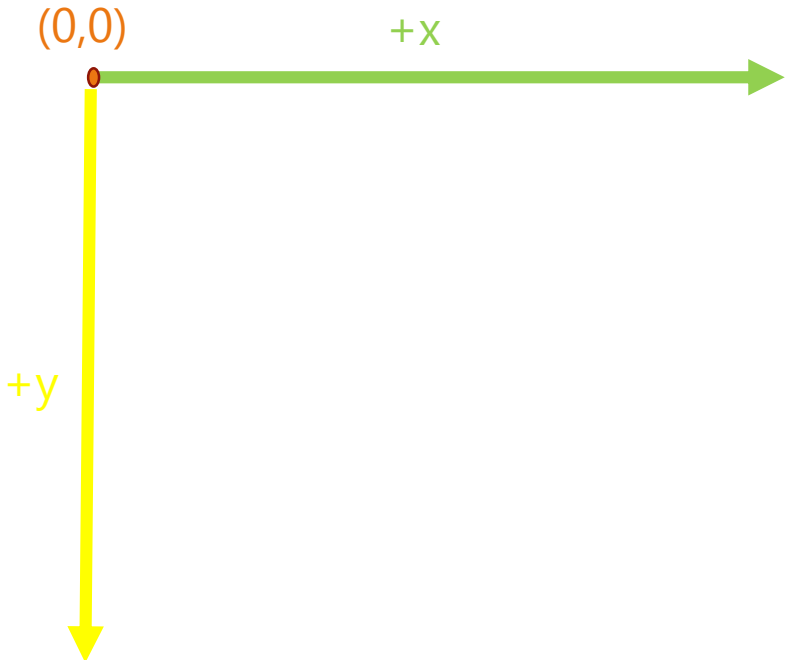
CANVAS COORDINATE SYSTEM

- There are two coordinate system, the local coordinate system and global coordinate system.
- All commands are relative to the local coordinate system
- The global coordinate system and local coordinate system are initially the same.

CANVAS COORDINATE SYSTEM

The Canvas coordinate system:

- (0,0) starts at the upper left hand corner
- Positive x points to the right
- Positive y points down



CTX.FILLRECT

- draws a rectangle

```
ctx.fillRect(x,y,W,H);
```

- x is the x-coordinate of the upper left corner of the rectangle
- y is the y-coordinate of the upper left corner of the rectangle
- W is the width in pixels
- H is the height in pixels

CTX.FILLSTYLE

- fillStyle is not a method but a property
- We can us it to set the color of the fill
- **`ctx.fillStyle = “red”;`**

EXAMPLE

```
let canvas =
document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.fillStyle = "cyan";
ctx.fillRect(0, 0, 100, 100);
```

- On a canvas that is 300px by 300px
- draws a cyan rectangle starting at (0,0) with width and height 100px
- Note that the color is set first, then the rectangle is drawn

DRAWING WITH PATHS

- Canvas has the ability to do line drawings.
- The Drawing Context has an internal list of points/commands.
- Commands that modify this list of points
  - `beginPath();`
  - `lineTo();`
  - `moveTo();`
- Commands that draw a line through the points in the list
- `stroke()`

TO DRAW A PATH

- We start by calling

```
ctx.beginPath()
```

- This call empties the internal list of points and commands inside the drawing context

- Next we can call lineTo() or moveTo() to add a point to the list

CTX.LINETO()

- ctx.lineTo() adds a point to the internal list, and the command to draw a line from the last point to the current point.
- No line is actually drawn yet
- If lineTo() is called immediately after beginPath(), then only the point is added to list, not drawing command is added because there is no point to connect to

- `ctx.lineTo(x,y);`
  - x is the x-coordinate in pixels in the local coordinate system
  - y is the y-coordinate in pixels in the local coordinate system

CTX.MOVETO() & STROKE()

- Move to adds the the point to the list, but does not add any drawing commands like the lineTo()

- **`ctx.moveTo(x,y);`**
  - x is the x-coordinate in pixels in the Local coordinate system
  - y is the y-coordinate in pixels in the local coordinate system

- **`ctx.stroke();`**
  - stoke actually draws the what is described in the internal list that we have been building up.

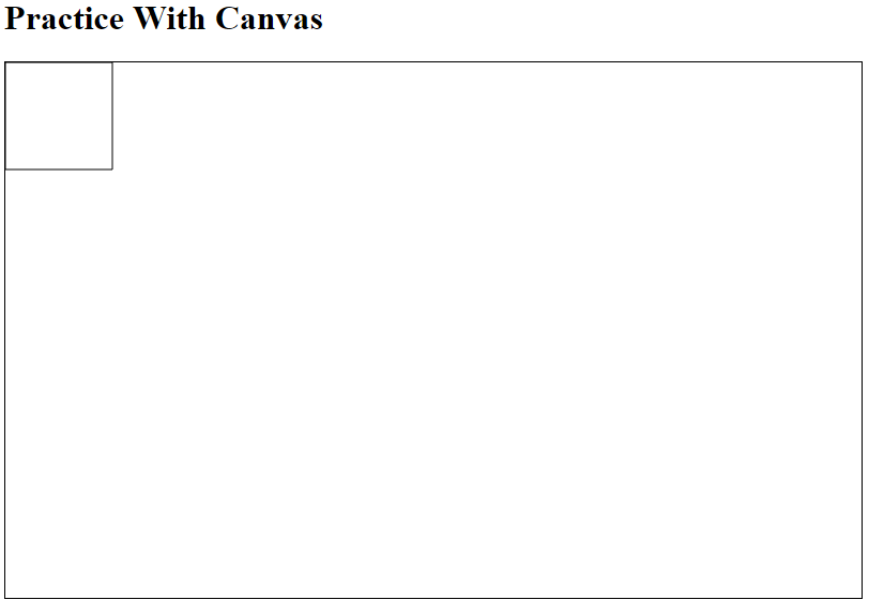
EXAMPLE

```
1. ctx.beginPath();
2. ctx.lineTo(0,0);
3. ctx.lineTo(100,0);
4. ctx.lineTo(100,100);
5. ctx.lineTo(0,100);
6. ctx.lineTo(0,0);
7. ctx.stroke();
```

line	coordinate List
1	[]
2	[(0,0)]
3	[(0,0),(100,0)]
4	[(0,0),(100,0),(100,100)]
5	[(0,0),(100,0),(100,100),(0,100)]
6	[(0,0),(100,0),(100,100),(0,100),(0,0)]
7	[(0,0),(100,0),(100,100),(0,100),(0,0)]

EXAMPLE CONT.

```
ctx.beginPath();
ctx.lineTo(0,0);
ctx.lineTo(100,0);
ctx.lineTo(100,100);
ctx.lineTo(0,100);
ctx.lineTo(0,0);
ctx.stroke();
```

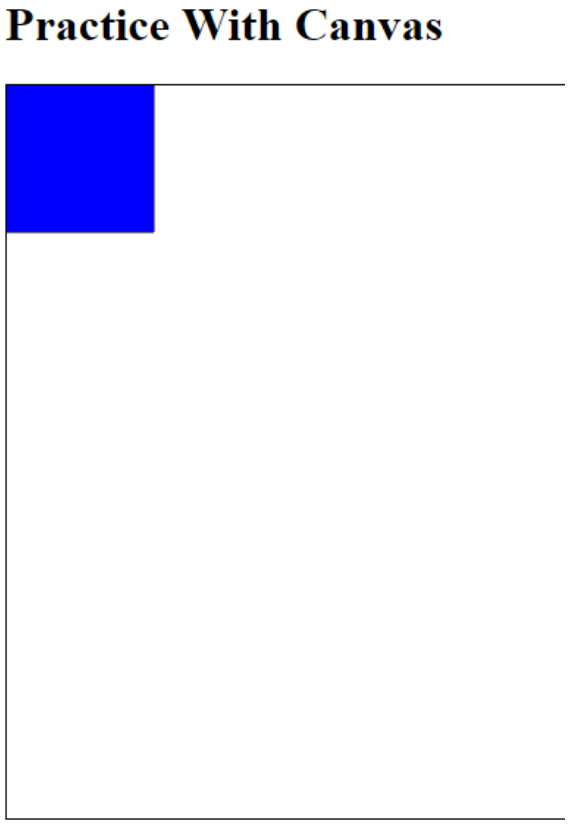


- And this square starting at (0,0) is drawn

EXAMPLE CONT.

- Now if we modify this example slightly by adding two extra lines

```
ctx.beginPath();
ctx.lineTo(0,0);
ctx.lineTo(100,0);
ctx.lineTo(100,100);
ctx.lineTo(0,100);
ctx.lineTo(0,0);
ctx.stroke();
ctx.fillStyle = "blue";
ctx.fill();
```



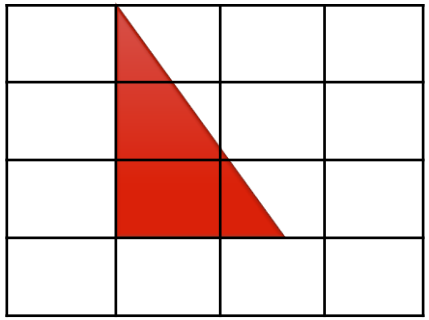
CTX.STROKESTYLE

- In the previous example you may have noticed the final box was filled with blue, but still had a black outline
- We can change the color of the stroke
- stokeStyle is a property of the rendering context

- **`ctx.strokeStyle = “blue”;`**

DRAWING SHAPES

- If we're going to draw anything, we need to know the coordinates of the points
- Easiest way to do this is to superimpose a grid on top of your drawing
- Then you know exactly what points you need to construct your drawing



ANGLES IN CANVAS

- Any time we need to specify and angle we must specify it in radians
- Sorry
- Going to have to convert degrees to radians
- Don’t panic! This is easy in JavaScript

$$q_{rad} = q_{\deg} \frac{p}{180}$$

```
radians = degrees *(Math.PI/180);
```

DRAWING CIRCLES

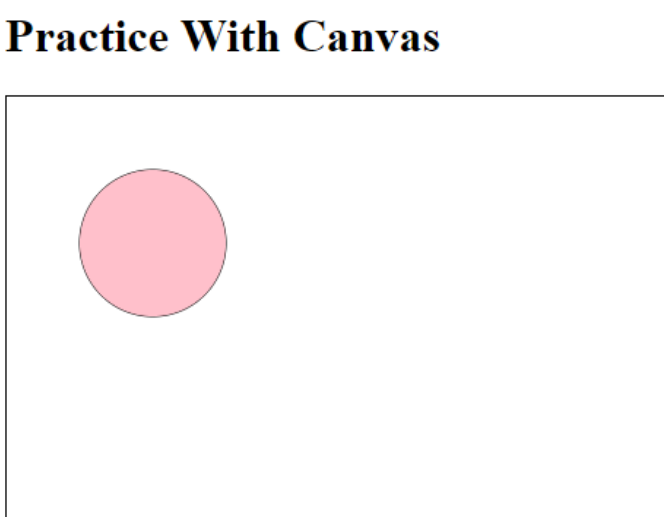
- `ctx.arc(x, y, r, start, stop);`

- X and Y give the location of the CENTER of the circle
  - x is the x-coordinate in pixels
  - y is the y-coordinate in pixels
- r is the radius of the circle you want to draw
- start is the angle you want to start drawing your circle at (in radians)
- stop is the angle you want to stop drawing your circle at (in radians)

MY FIRST CIRCLE

```
ctx.fillStyle = "pink";
ctx.beginPath();
ctx.arc(100,100,50,0,2*Math.PI);
ctx.stroke();
ctx.fill();
```

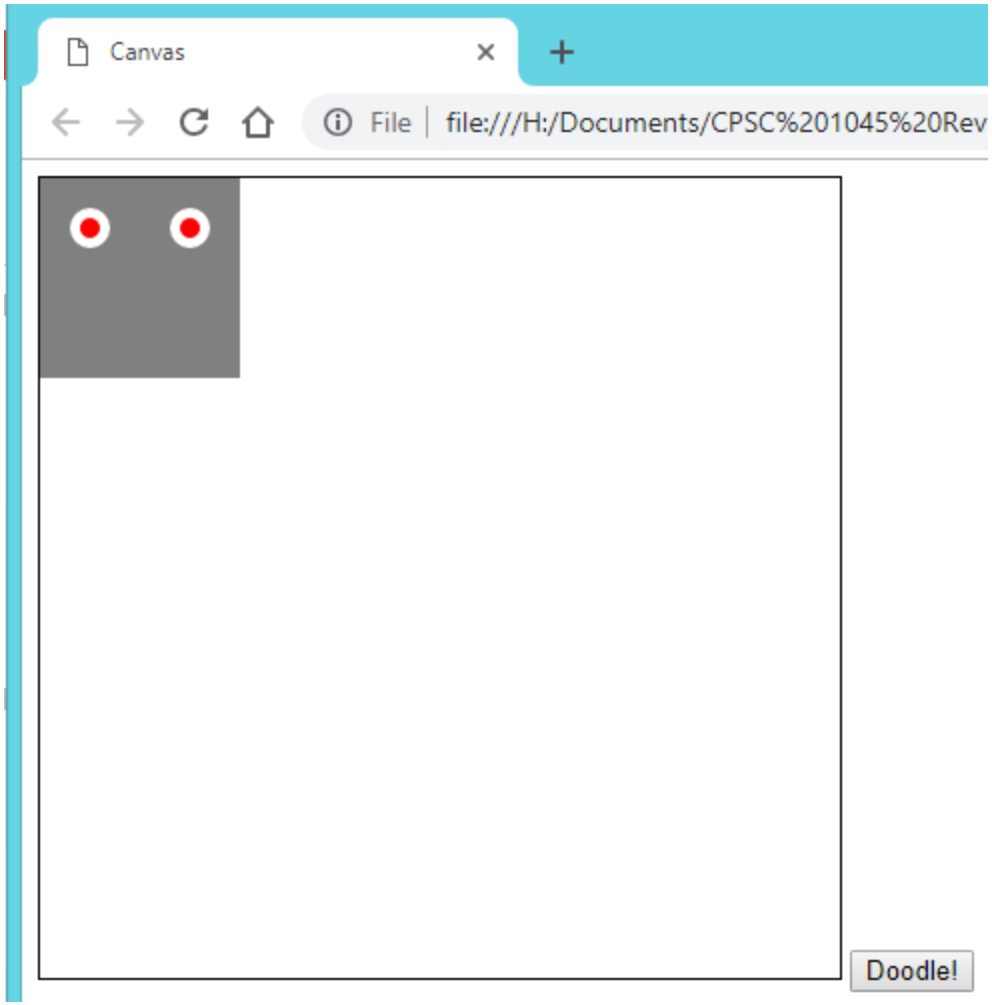
- If you want to draw a full circle, you want to start at 0 degrees and go a full 360 degrees around
- Converting 360 degrees to radians is 360*(Math.PI/180) which is 2*Math.PI



EXERCISE

```
function draw(){
        ctx.fillStyle = "gray";
        ctx.fillRect(0,0,100,100);
        ctx.fillStyle = "white";
        ctx.beginPath();
        ctx.arc(25,25,10,0, 2*Math.PI);
        ctx.fill();

        ctx.beginPath();
        ctx.arc(75,25,10,0, 2*Math.PI);
        ctx.fill();

        ctx.fillStyle = "red";
        ctx.beginPath();
        ctx.arc(25,25,5,0, 2*Math.PI);
        ctx.fill();

        ctx.beginPath();
        ctx.arc(75,25,5,0, 2*Math.PI);
        ctx.fill();
}
```

EXERCISE



CHANGING THE LOCAL COORDINATE SYSTEM

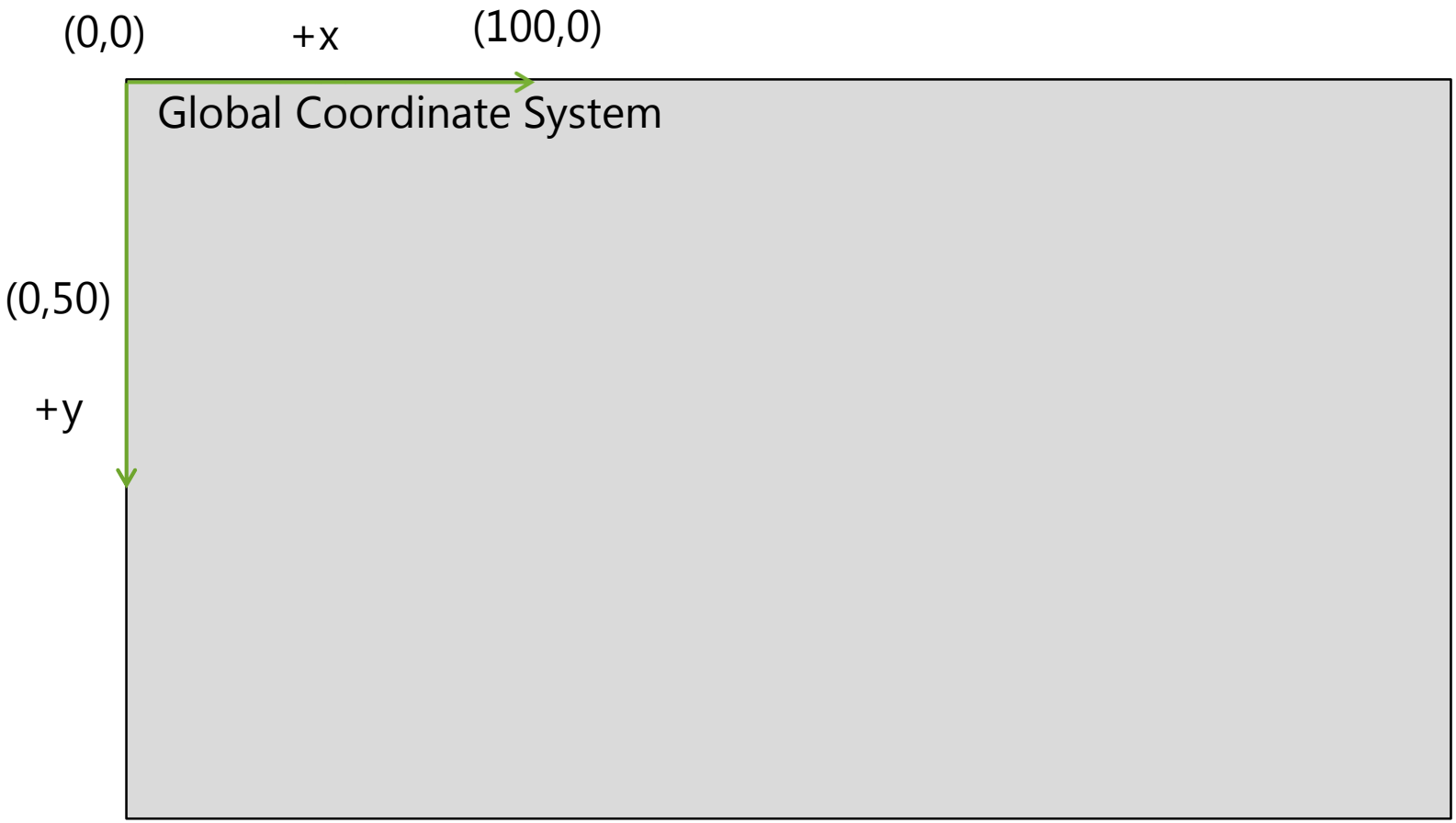
- We can change the local coordinate system
- This is convenient, since we can draw everything around the origin (0,0) and then just move them to the right place

- **`ctx.translate(deltaX, deltaY);`**
  - deltaX : The amount to move in the x-direction relative to the local coordinate system.
  - deltaY: The amount to move in the y-direction relative to the local coordinate system.

EXAMPLE

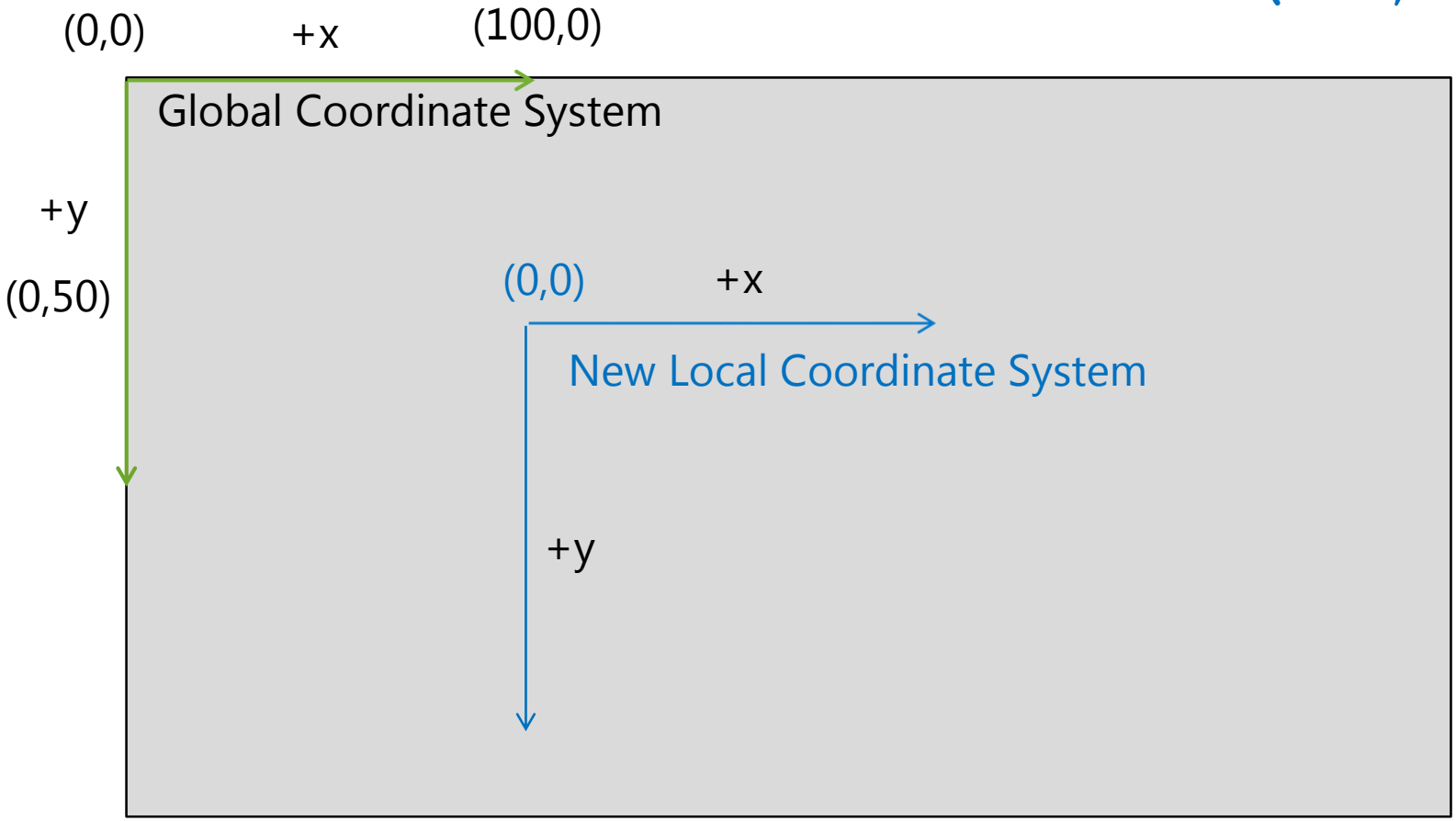
- **`ctx.translate(100,50);`**
- will move the local coordinate system, and all subsequent commands will use the new local coordinate system.
- The above command will move
  1. Move 100 in the current local x-direction
  2. Move 50 in the current local y-direction
- After the command we have a new local coordinate system.

EXAMPLE



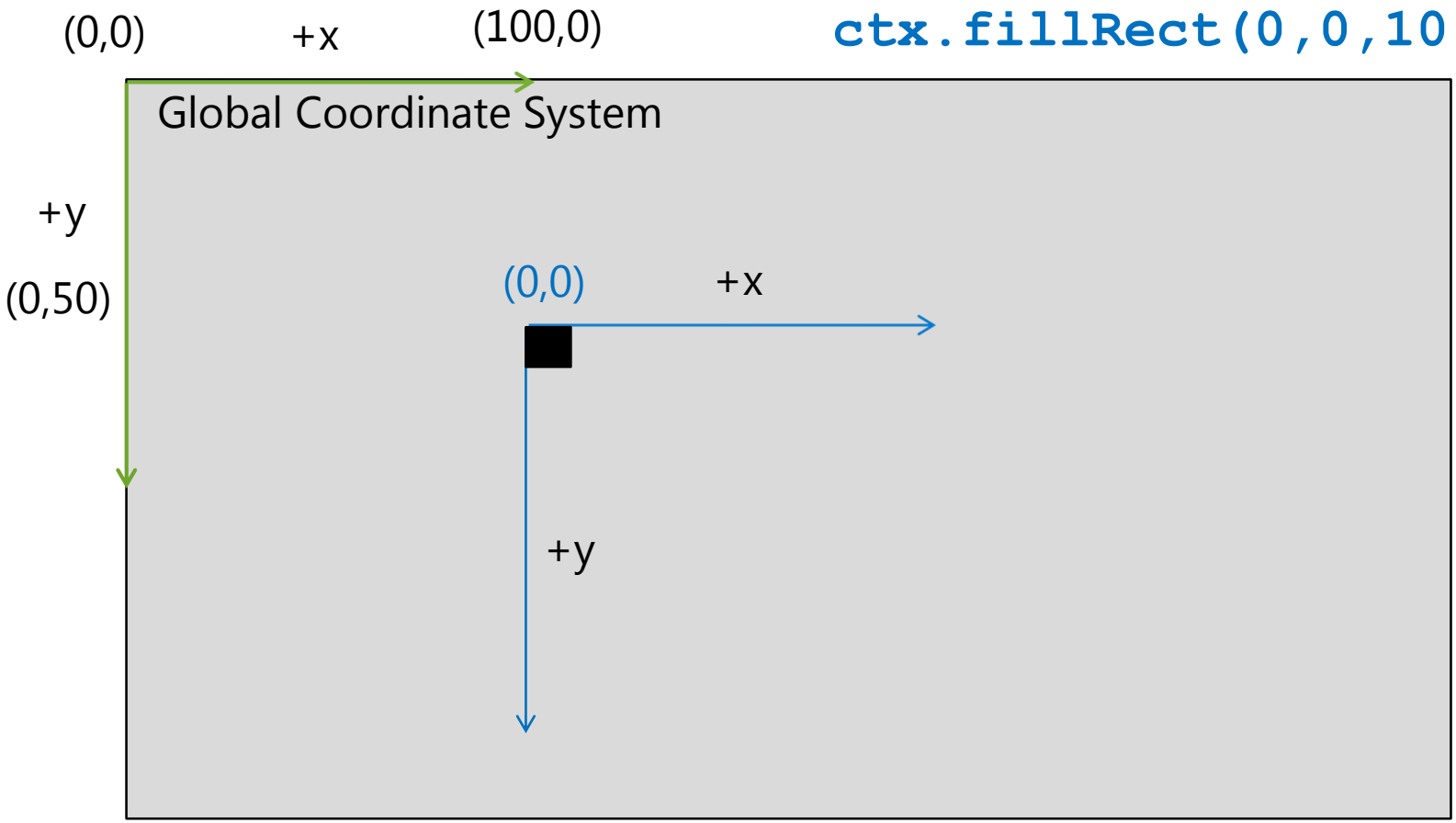
EXAMPLE PART 1

```
ctx.translate(100,50);
```



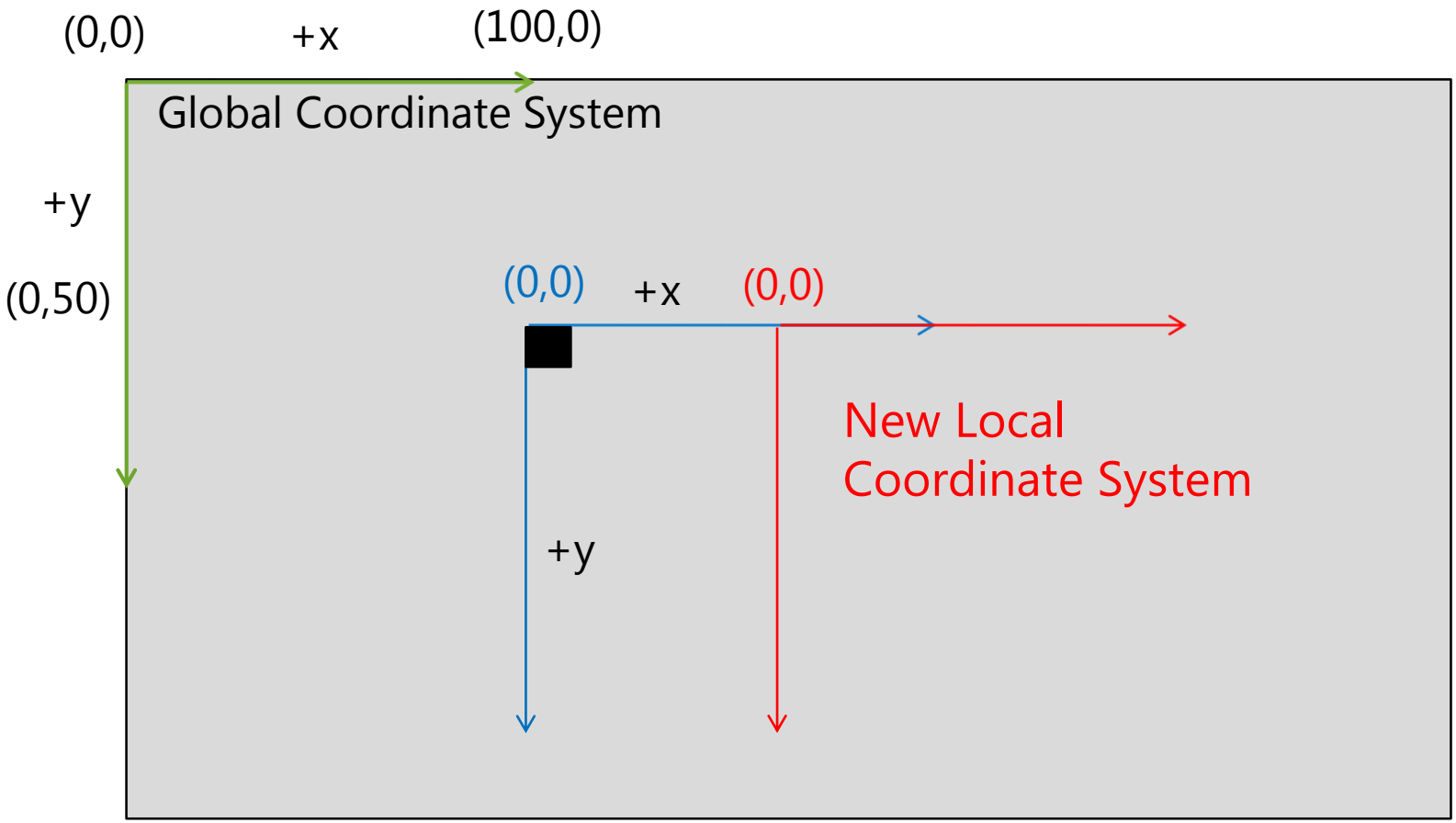
EXAMPLE PART 1

```
ctx.translate(100,50);
ctx.fillRect(0,0,10,10);
```



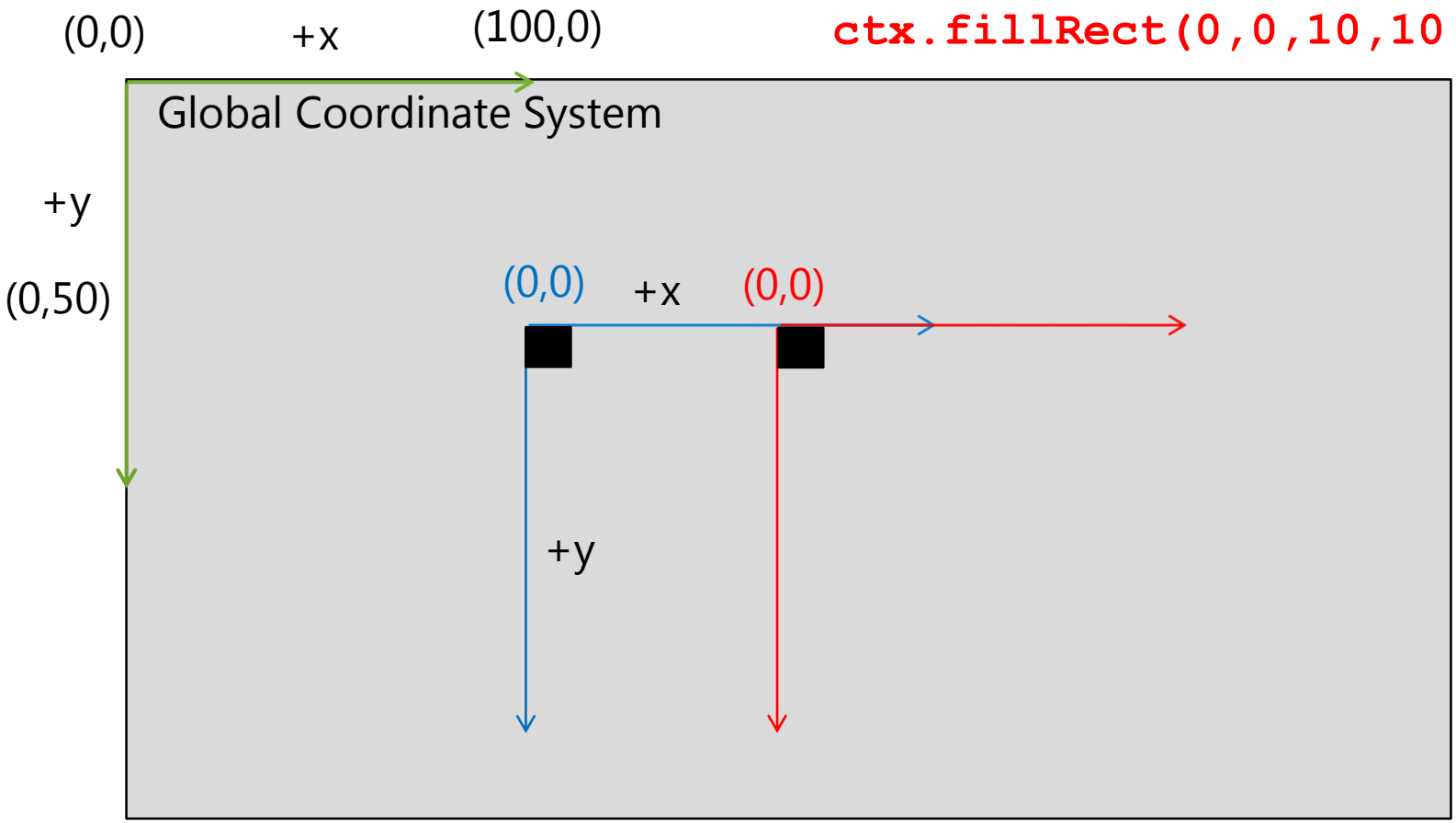
EXAMPLE PART 2

```
ctx.translate(100,50);
ctx.fillRect(0,0,10,10);
ctx.translate(50,0);
```



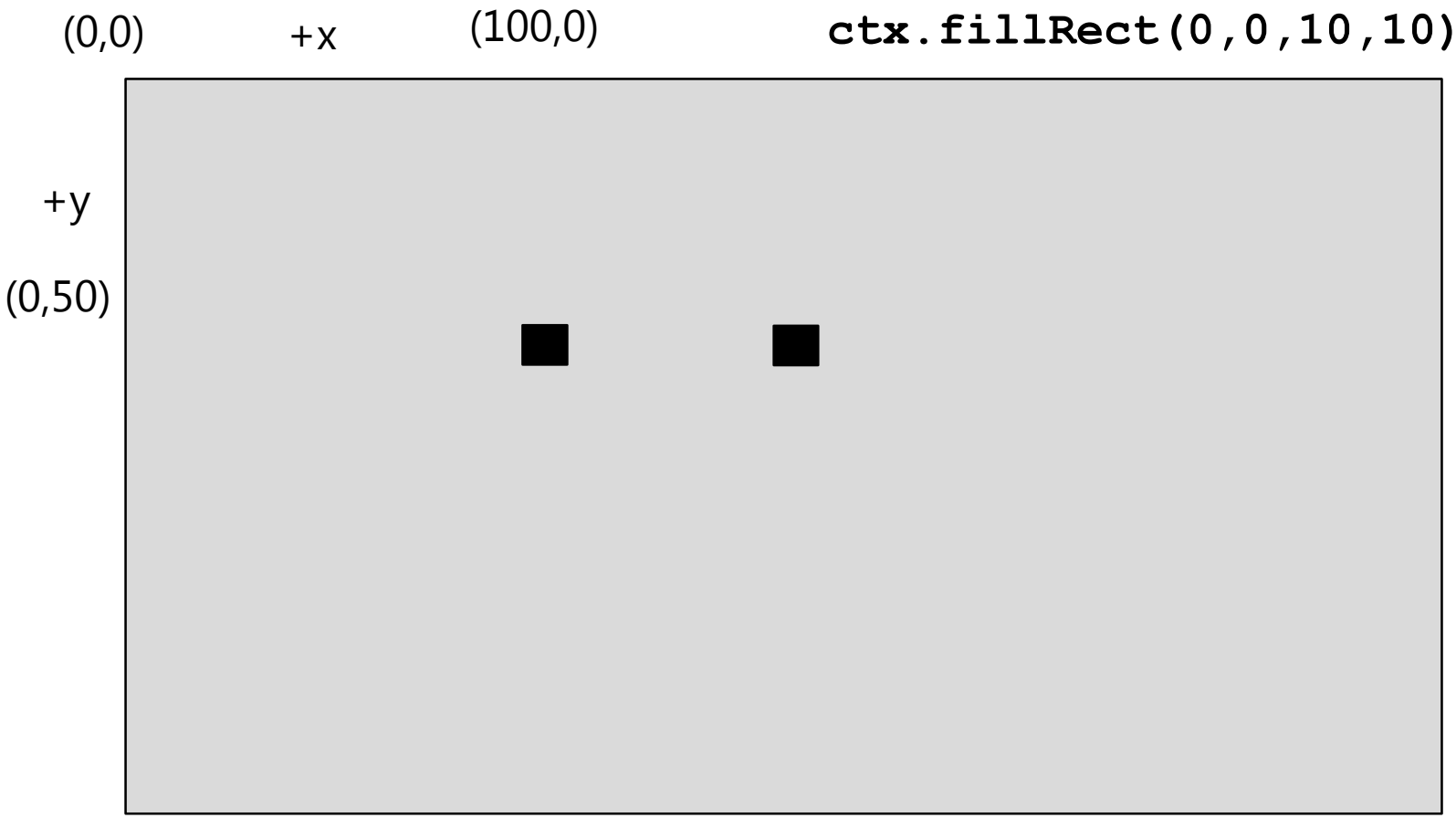
EXAMPLE PART 2

```
ctx.translate(100,50);
ctx.fillRect(0,0,10,10);
ctx.translate(50,0);
ctx.fillRect(0,0,10,10
```



EXAMPLE PART 2

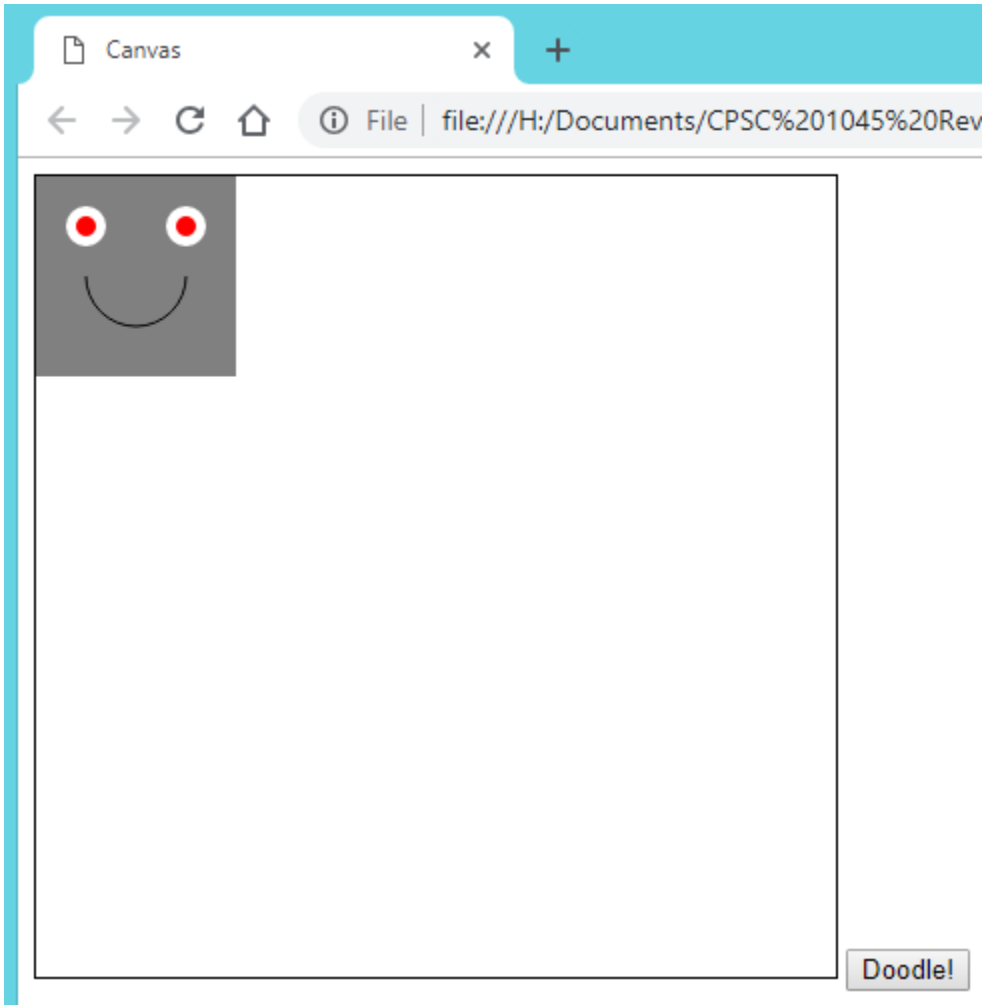
```
ctx.translate(100,50);
ctx.fillRect(0,0,10,10);
ctx.translate(50,0);
ctx.fillRect(0,0,10,10);
```



EXERCISE

```
function draw(){
        ctx.fillStyle = "gray";
        ctx.fillRect(0,0,100,100);
        ctx.fillStyle = "white";
        ctx.beginPath();
        ctx.arc(25,25,10,0, 2*Math.PI);
        ctx.fill();

        ctx.beginPath();
        ctx.arc(75,25,10,0, 2*Math.PI);
        ctx.fill();

        ctx.fillStyle = "red";
        ctx.beginPath();
        ctx.arc(25,25,5,0, 2*Math.PI);
        ctx.fill();

        ctx.beginPath();
        ctx.arc(75,25,5,0, 2*Math.PI);
        ctx.fill();

        ctx.translate(50,50);
        ctx.fillStyle = "black";
        ctx.beginPath();
        ctx.arc(0,0,25,0, Math.PI);
        ctx.stroke();
}
```

EXERCISE



CTX.ROTATE()

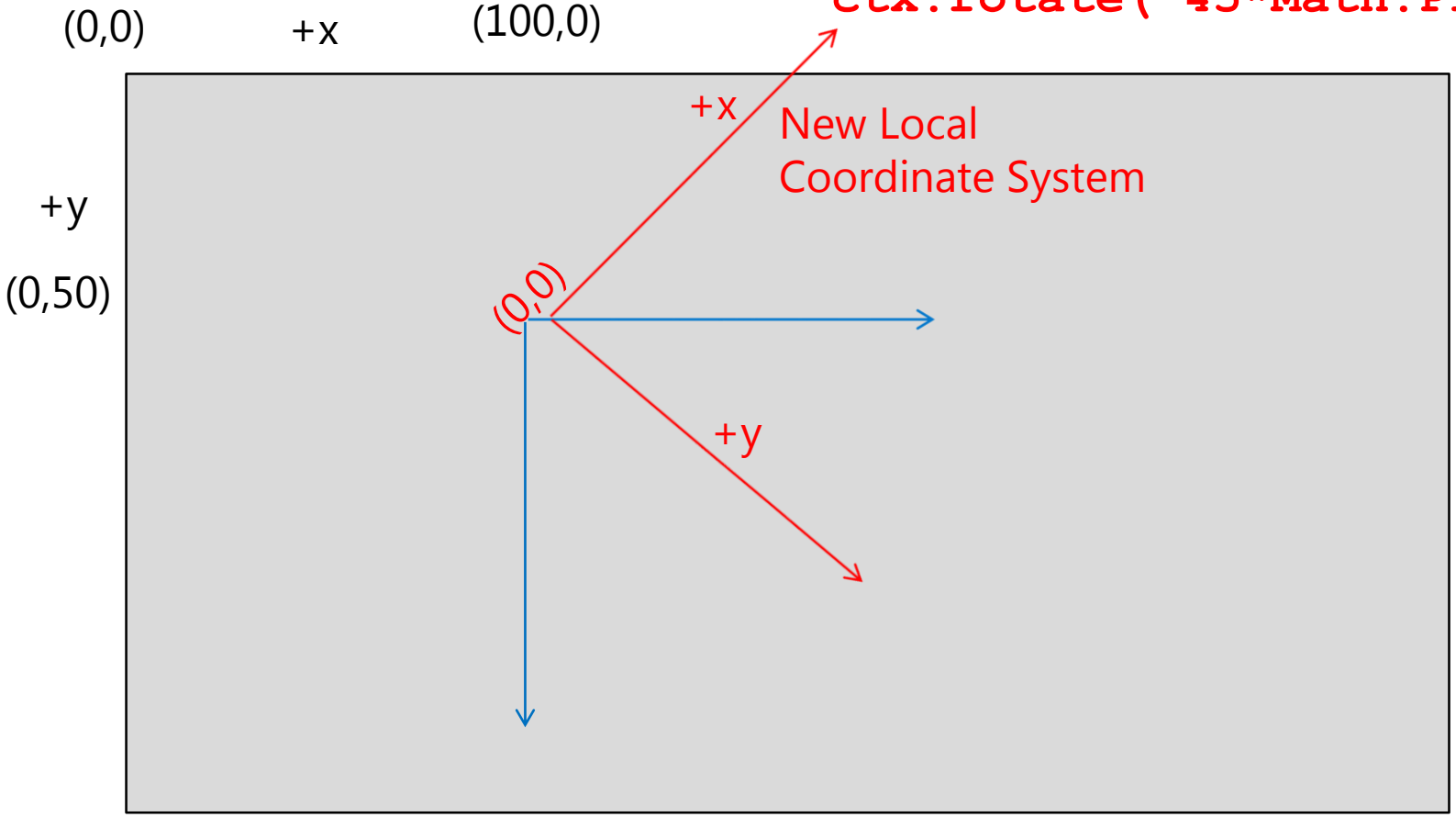
- ctx.rotate() rotates the coordinate system.
- The angle is specified in radians
- But that's okay

```
radians = degrees *(Math.PI/180);
```

$$q_{rad} = q_{\deg} \frac{p}{180}$$

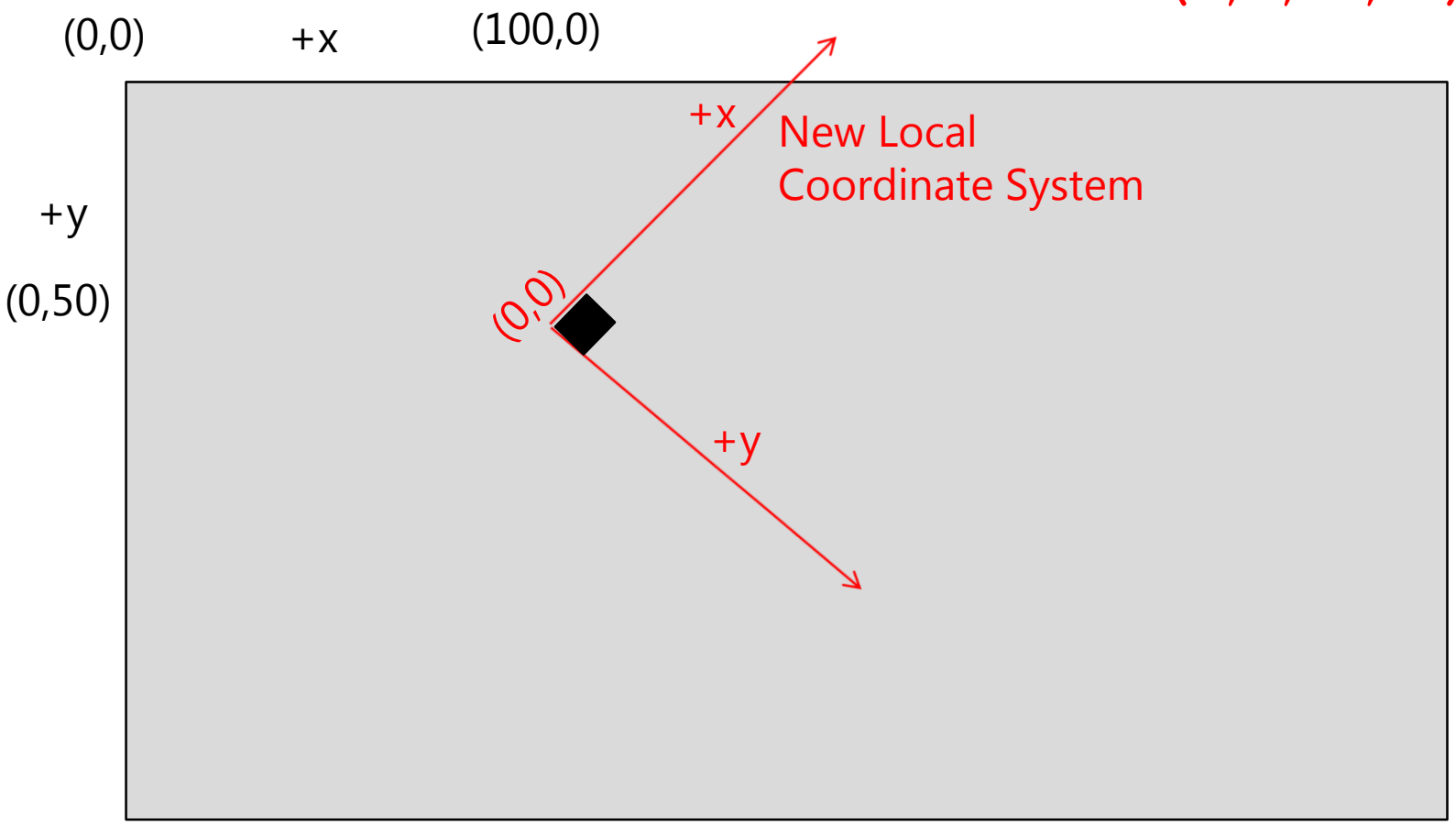
EXAMPLE PART 3

```
ctx.translate(100,50);
ctx.rotate(-45*Math.PI/180);
```



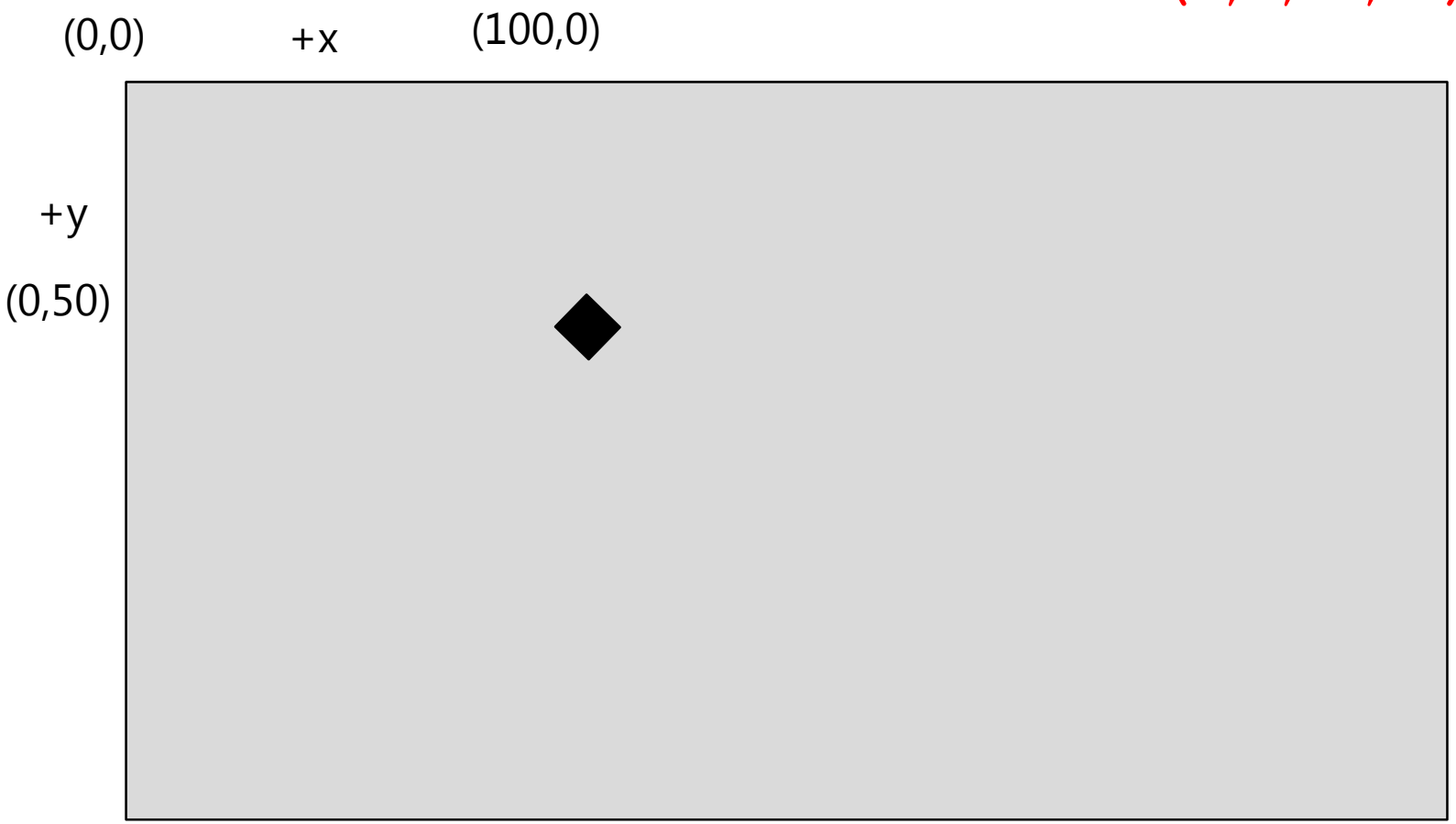
EXAMPLE PART 3

```
ctx.translate(100,50);
ctx.rotate(-45*Math.PI/180);
ctx.fillRect(0,0,10,10);
```



EXAMPLE PART 3

```
ctx.translate(100,50);
ctx.rotate(-45*Math.PI/180);
ctx.fillRect(0,0,10,10);
```



NOTE:

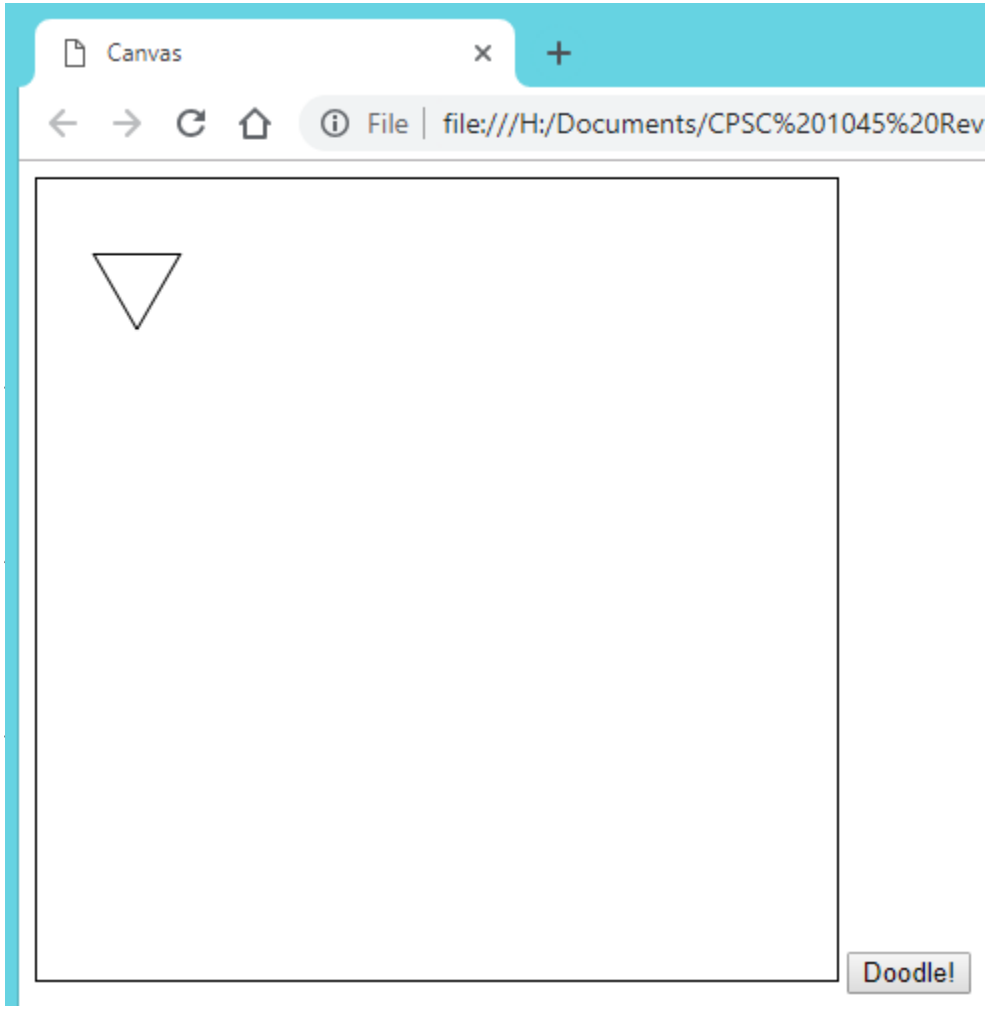
- All coordinates in the ctx list are stored in the global coordinate system
- So, calling translate or rotate does not effect existing points already on the list, just points added after the translate or rotate is called
- I.e., things drawn before rotate is called do not change

ROTATE EXAMPLE

```
ctx.translate(50,50);
ctx.beginPath();
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180);
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180);
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180);
ctx.lineTo(0,25);
ctx.stroke();
```

ROTATE EXAMPLE

```
ctx.translate(50,50);
ctx.beginPath();
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180)
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180)
ctx.lineTo(0,25);
ctx.rotate(120*Math.PI/180)
ctx.lineTo(0,25);
ctx.stroke();
```

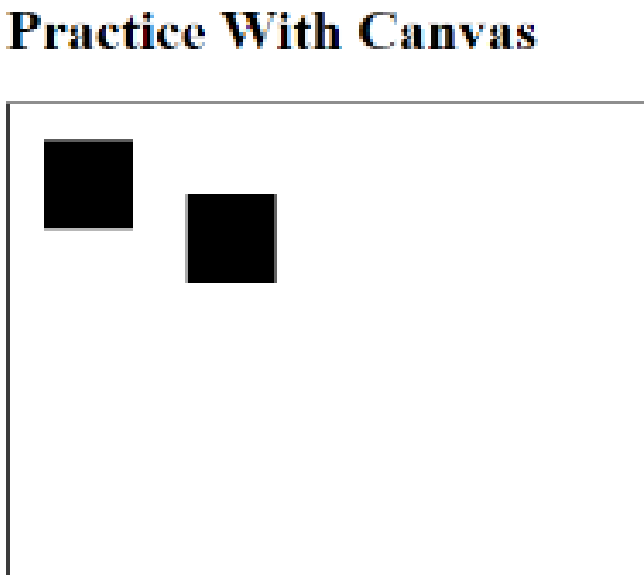


CTX.SAVE() & CTX.RESTORE()

- ctx.save() saves the current local coordinate system to a list
- ctx.restore() restores last local coordinate system saved to the list, and removes it from the list
- This allows us to isolate our effects of translate and rotate to a small section of code

EXAMPLE OF SAVE/RESTORE

```
ctx.save();
ctx.translate(100,50);
ctx.fillRect(0,0,50,50);
ctx.restore();
ctx.save();
ctx.translate(20,20);
ctx.fillRect(0,0,50,50);
ctx.restore();
```



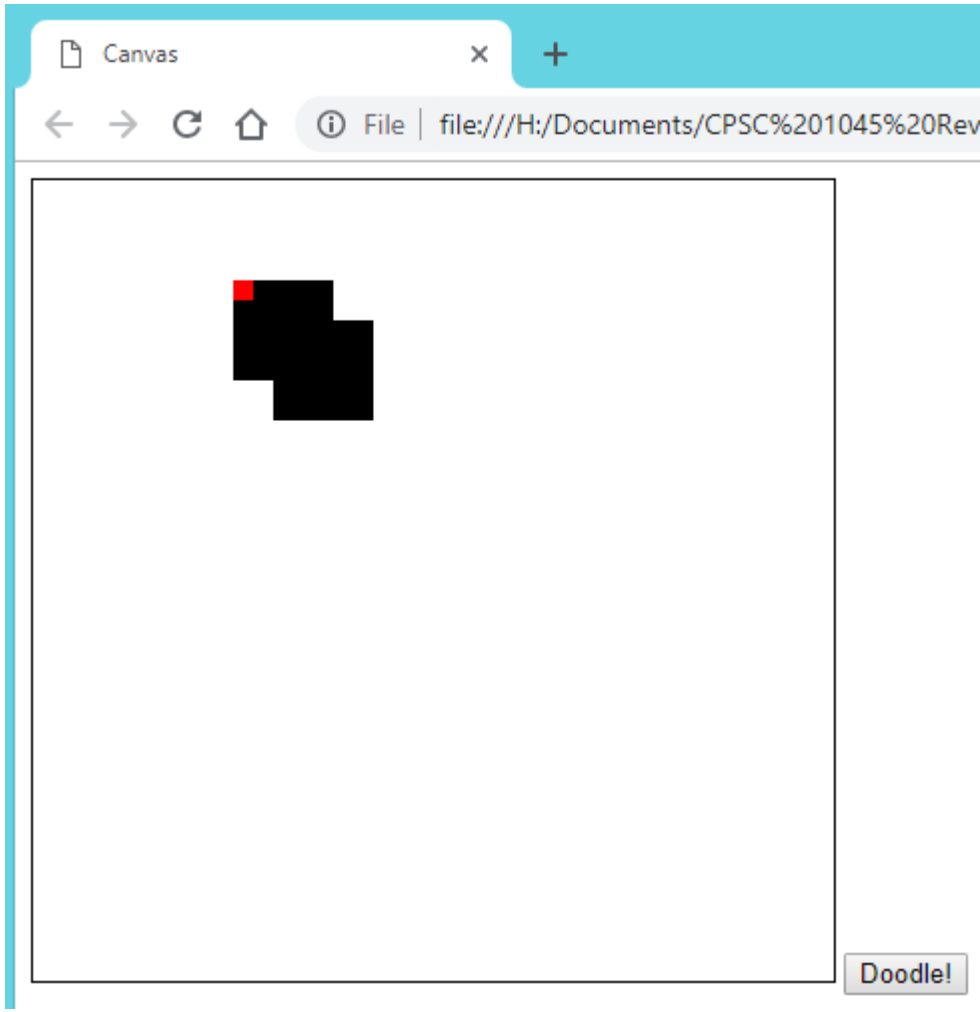
The above will draw two rectangles at different locations

EXERCISE

```
ctx.save();
ctx.translate(100,50);
ctx.fillRect(0,0,50,50);
ctx.save();
ctx.translate(20,20);
ctx.fillRect(0,0,50,50);
ctx.restore();
ctx.fillStyle = "red";
ctx.fillRect(0,0,10,10);
ctx.restore();
```

EXERCISE

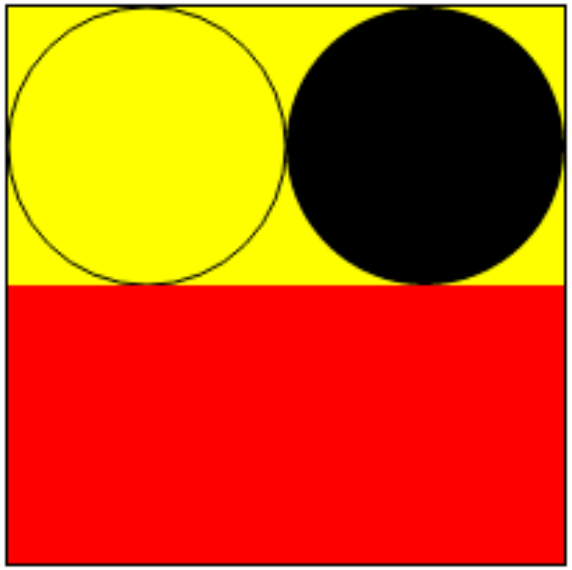
```
ctx.save();
ctx.translate(100,50);
ctx.fillRect(0,0,50,50);
ctx.save();
ctx.translate(20,20);
ctx.fillRect(0,0,50,50);
ctx.restore();
ctx.fillStyle = "red";
ctx.fillRect(0,0,10,10);
ctx.restore();
```



```
ctx.fillStyle = "yellow";
ctx.fillRect(0,0,200,200);
ctx.fillStyle = "black";

ctx.save();
ctx.beginPath();
ctx.translate(50,50);
ctx.arc(0,0,50,0, 2*Math.PI);
ctx.stroke();
ctx.restore();

ctx.save();
ctx.beginPath();
ctx.translate(150,50);
ctx.arc(0,0,50,0, 2*Math.PI);
ctx.fill();
ctx.restore();

ctx.fillStyle = "red";
ctx.fillRect(0, 100, 200, 100);
```

CANVAS IS 200 BY 200 WHAT IS DRAWN?

SOLUTION



LINE WIDTH

```
let canvas =
document.getElementById("canvas");
let ctx = canvas.getContext("2d");
ctx.strokeStyle = "blue";
ctx.strokeRect(5 , 5 , 50 , 50);
ctx.lineWidth = 5;
ctx.strokeRect(135 , 5 , 50 , 50) ;
```

TEXT

- **`ctx.fillText()`**
  - Draws letters filled with current fillStyle color
- **`ctx.strokeText()`**
  - Draws outline of letters


```
ctx.font = "40px Georgia ";
ctx.fillStyle = "orange";
ctx.fillText("I can draw text, too!",10,50);

ctx.strokeStyle = "purple";
ctx.strokeText("Fancy text is fancy",10,100);

ctx.fillStyle = "yellow";
ctx.strokeStyle = "blue";
ctx.fillText("EXTRA FANCY", 10, 150);
ctx.strokeText("EXTRA FANCY", 10, 150);
```