2D transformations: changing the coordinate system

INTRODUCTION

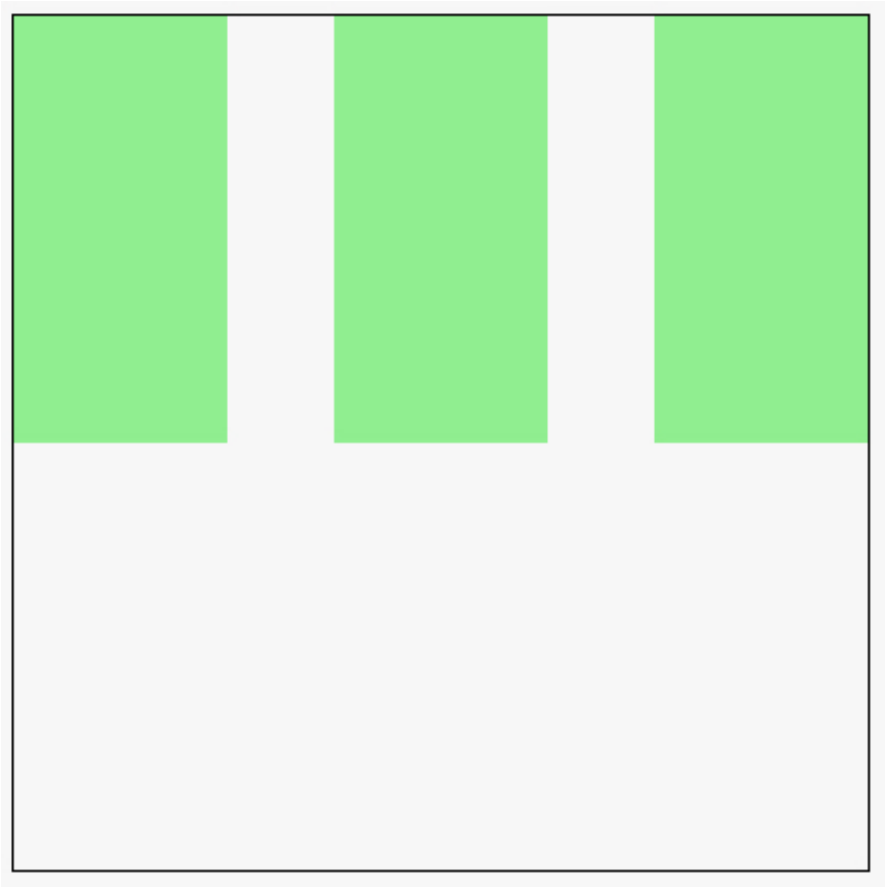
In this part of the course, we introduce the basics of 2D transformations, a powerful tool that will make things easier as soon as you have to:

- Draw complex shapes at given positions, with given orientations and sizes,
- Draw shapes relative to one another.

Let's start with some simple examples before looking at how we use 2D transforms.

LET'S DRAW THREE RECTANGLES!

If we draw three rectangles of size 100x200 in a 400x400 canvas, one at (0, 0) and another at (150, 0), and a third at (300, 0), here is the result and the corresponding code:



JavaScript code extract (see the online example at JS Bin for the complete running code)

```
function drawSomething() {
    ctx.fillStyle='lightgreen';

    ctx.fillRect(0,0,100,200);
    ctx.fillRect(150,0,100,200);
    ctx.fillRect(300,0,100,200);
}
```

LET'S MODIFY THE CODE SO THAT WE CAN DRAW THESE RECTANGLES AT ANY X AND Y POSITION

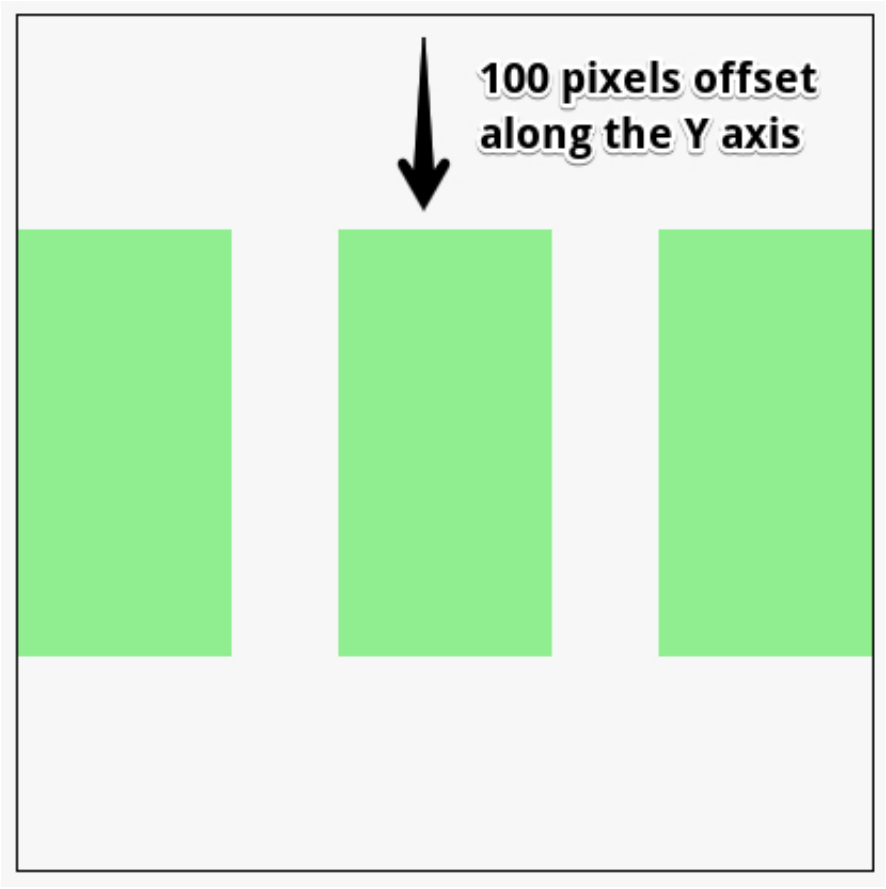
What if we wanted to draw these 3 rectangles at another position, as a group? We would like to draw all of them a little closer to the bottom, for example... Let's add some parameters to the function: the X and Y position of the rectangles.

The full JavaScript code is (see online running example):

```
var canvas, ctx;
function init() {
    // This function is called after the page is loaded
    // 1 - Get the canvas
    canvas =document.getElementById('myCanvas');
    // 2 - Get the context
    ctx=canvas.getContext('2d');
    // 3 - we can draw
    drawSomething(0, 100);
}
function drawSomething(x, y) {
    // draw 3 rectangles
    ctx.fillStyle='lightgreen';
    ctx.fillRect(x,y,100,200);
    ctx.fillRect(x+150,y,100,200);
    ctx.fillRect(x+300,y,100,200);
}
```

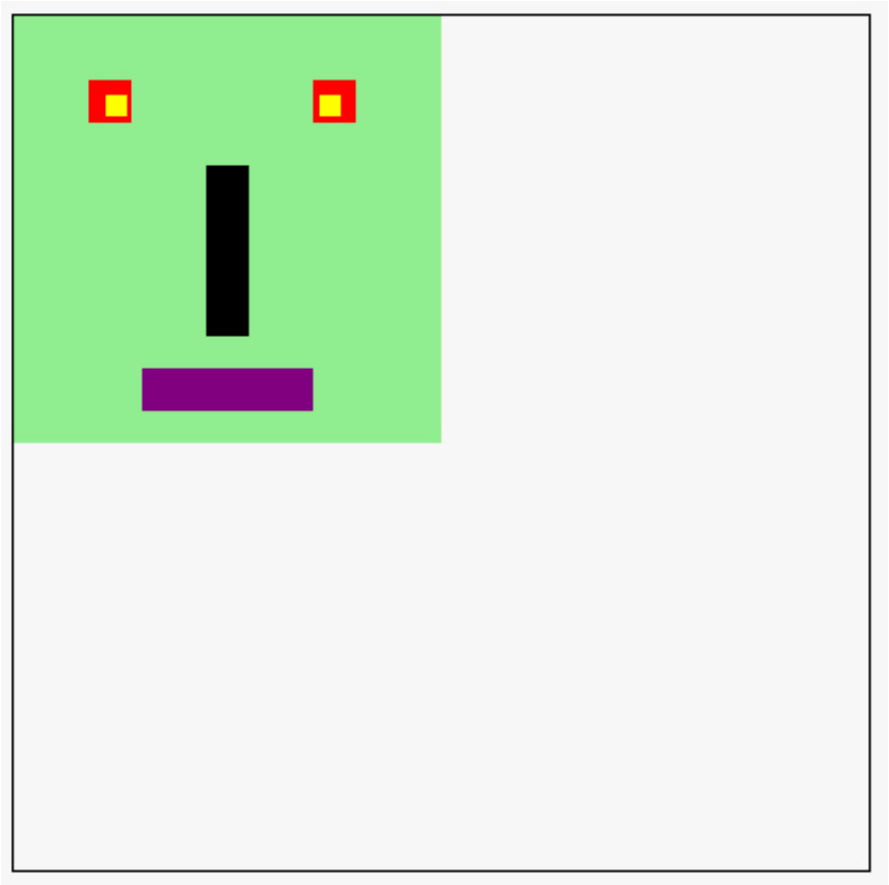
At line 10, we called the `drawSomething(...)` function with 0 and 100 as parameters, meaning "please add an offset of 0 in X and 100 in Y directions to what is drawn by the function...

If you look at the code of the modified function, you will see that each call to `fillRect(...)` uses the x and y parameters instead of hard coded values. In this way, if we call it with parameters (0, 100), then all rectangles will be drawn 100 pixels to the bottom (offset in y). Here is the result:



OK, NOW LET'S DRAW A SMALL MONSTER'S HEAD WITH RECTANGLES

Now we can start having some fun... let's draw a monster's head using only rectangles (online version):



An extract of the JavaScript source code is:

```
function drawMonster(x, y) {
  // head
  ctx.fillStyle='lightgreen';
  ctx.fillRect(x,y,200,200);
  // eyes
  ctx.fillStyle='red';
  ctx.fillRect(x+35,y+30,20,20);
  ctx.fillRect(x+140,y+30,20,20);
  // interior of eye
  ctx.fillStyle='yellow';
  ctx.fillRect(x+43,y+37,10,10);
  ctx.fillRect(x+143,y+37,10,10);
  // Nose
  ctx.fillStyle='black';
  ctx.fillRect(x+90,y+70,20,80);
  // Mouth
  ctx.fillStyle='purple';
```

```
    ctx.fillRect(x+60,y+165,80,20);
  }
```

As you can see, the code uses the same technique, becomes less and less readable. The Xs and Ys at the beginning of each call makes understanding the code harder, etc.

However, there is a way to simplify this => 2D geometric transformations!

GEOMETRIC TRANSFORMATIONS: CHANGING THE COORDINATE SYSTEM

The idea behind 2D transformations is that instead of modifying all the coordinates passed as parameters to each call to drawing methods like `fillRect(...)`, we will keep all the drawing code "as is". For example, if the monster of our previous example was drawn at (0, 0), we could just translate (or rotate, or scale) the original coordinate system.

Let's take a piece of code that draws something corresponding to the original coordinate system, located at the top left corner of the canvas:

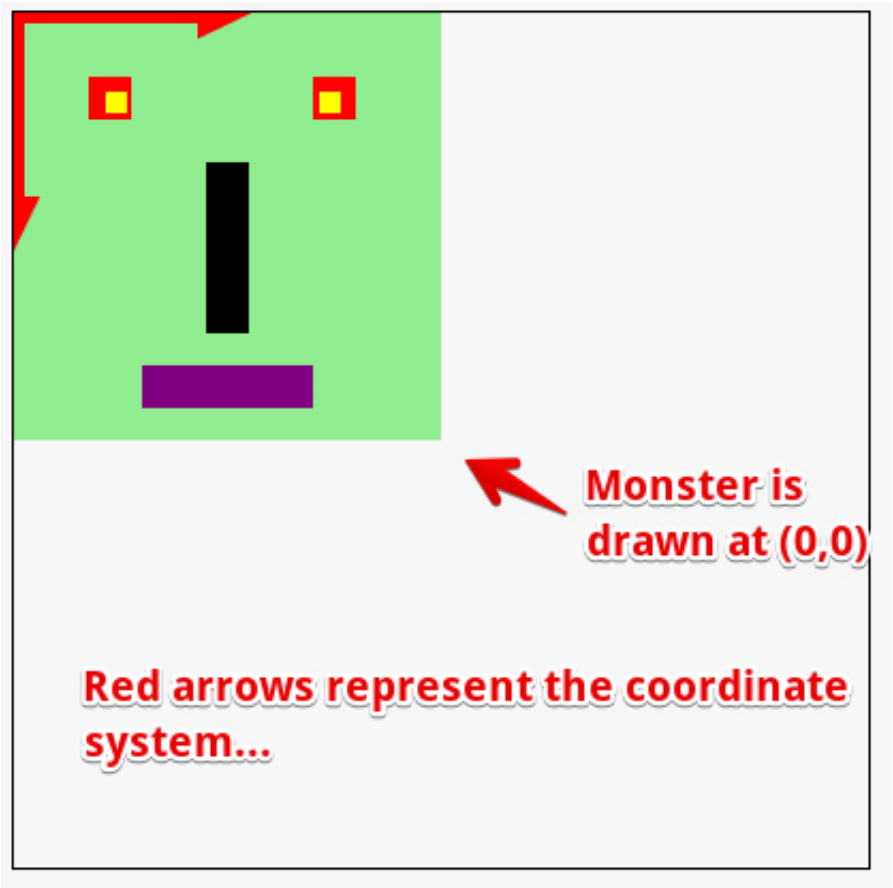
```
  function drawMonster(x, y) {
    // head
    ctx.fillStyle='lightgreen';
    ctx.fillRect(0,0,200,200);
    // eyes
    ctx.fillStyle='red';
    ctx.fillRect(35,30,20,20);
    ctx.fillRect(140,30,20,20);

    // interior of eye
    ctx.fillStyle='yellow';
    ctx.fillRect(43,37,10,10);
    ctx.fillRect(143,37,10,10);
    // Nose
    ctx.fillStyle='black';
    ctx.fillRect(90,70,20,80);
    // Mouth
```

```
    ctx.fillStyle='purple';
    ctx.fillRect(60,165,80,20);
    // coordinate system at (0, 0)
    drawArrow(ctx, 0, 0, 100, 0, 10,'red');
    drawArrow(ctx, 0, 0, 0, 100, 10,'red');
}
```

This code is the just the same as in the previous example except that we removed all Xs and Yx in the code. We also added at the end (lines 25-26) two lines of code that draw the coordinate system. The `drawArrow(startX, startY, endX, endY, width, color)` function is a utility function that we will present later. You can see it in the source code of the complete online example on JS Bin: look in the JavaScript tab.

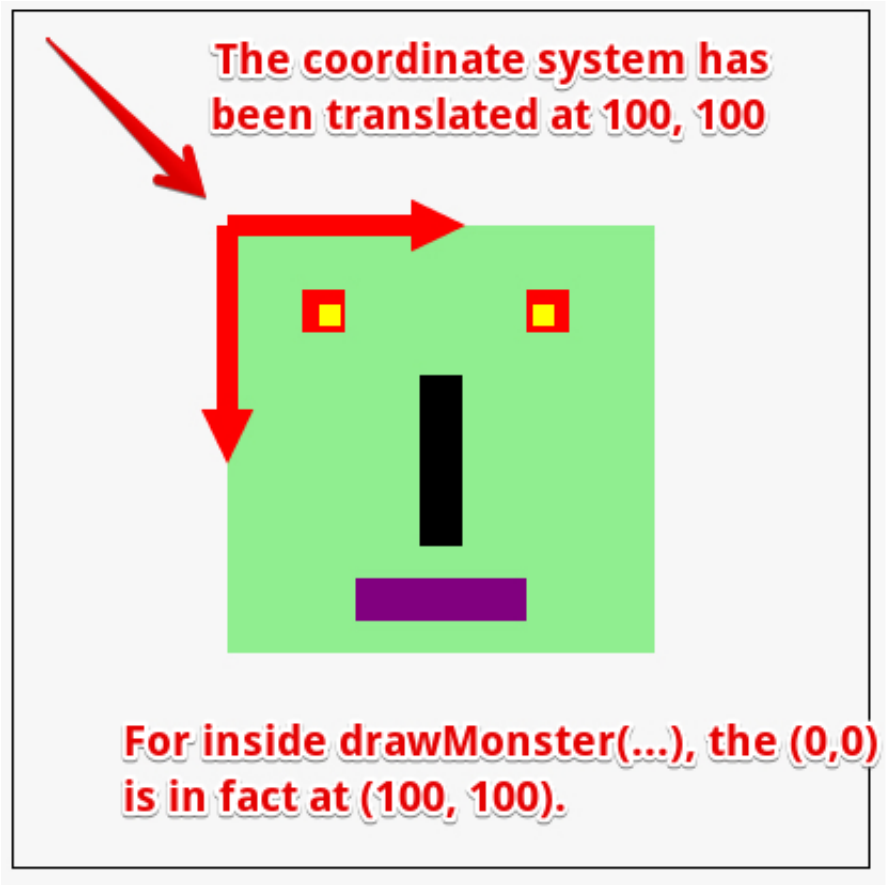
Note that the X and Y parameters are useless for now...

Result:



Translation using `ctx.translate(offsetX, offsetY)`

Now, instead of simply calling `drawMonster(0, 0)`, we will call first `ctx.translate(100, 100)`, and look at the result (online code: http://jsbin.com/yuhamu/2/edit)



JavaScript code extract:

```
ctx.translate(100, 100);
drawMonster(0, 0);
```

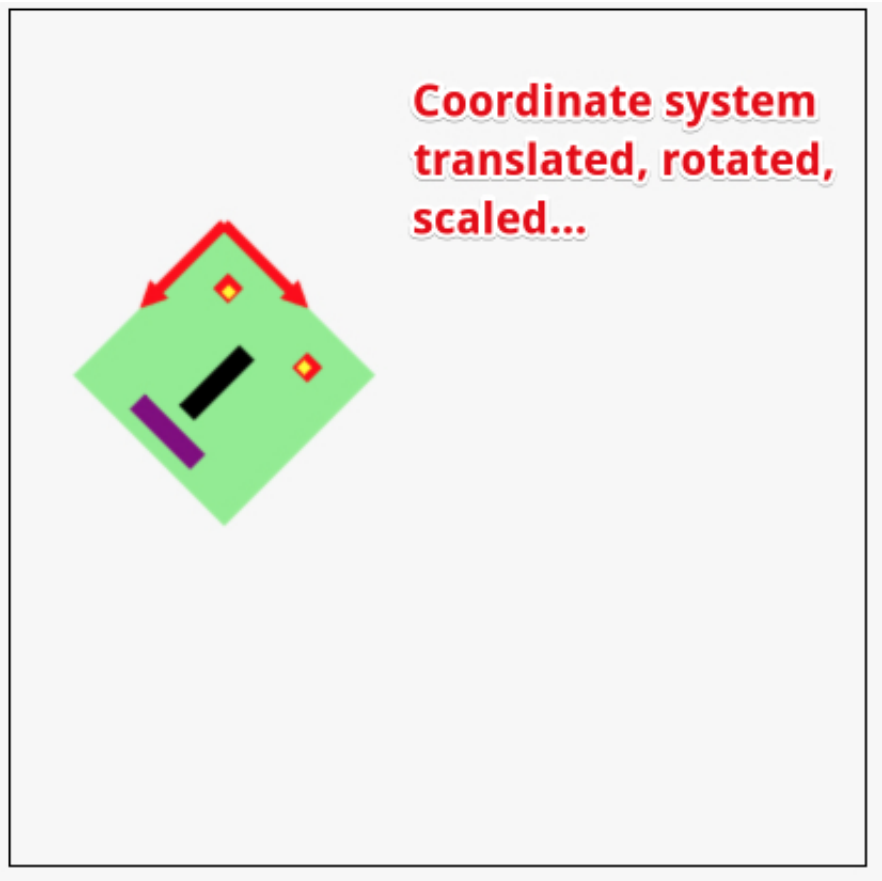
Line 1 changes the position of the coordinate system, line 2 draws a monster in the new translated coordinate system. All subsequent calls to drawing methods will be affected and will work in this new system too.

OTHER TRANSFORMATIONS: ROTATE, SCALE

There are other transformations available:

- `ctx.rotate(angle)`, with angle in radians. Note that the order of transformations is important: usually we translate, then rotate, then scale... If you change this order, you need to know what you are doing...
- `ctx.scale (sx, sy)`, where `scale(1, 1)` corresponds to "no zoom", `scale(2, 2)` corresponds to "zooming 2x" and`scale(0.5, 0.5)` corresponds to zooming out to see the drawings half as big as before. If you do not use the same values for `sx` and `sy`, you do "asymmetric scaling", you can distort a shape horizontally or vertically. Try changing the values in the source code of the next online examples.

Here is the previous example, but this time we translated the coordinate system, then rotated it with an angle equal to PI/4 , then we scaled it so that units are half as big (see the example online):

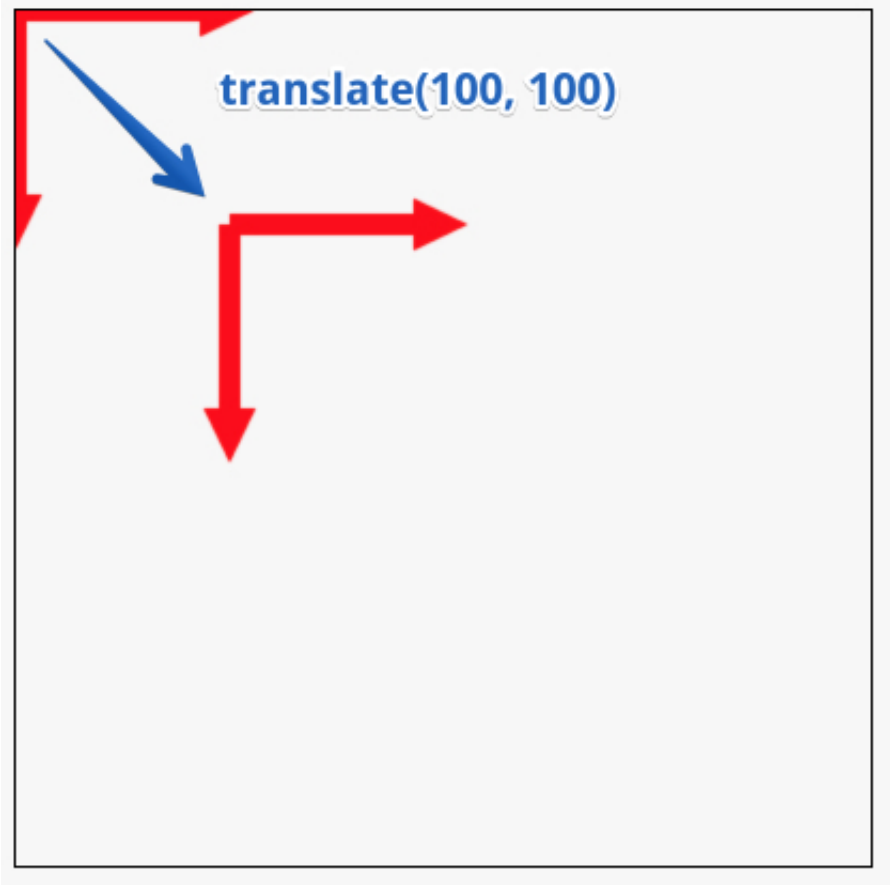


And here is the code of the transformations we used, followed by the call to the function that draws the monster:

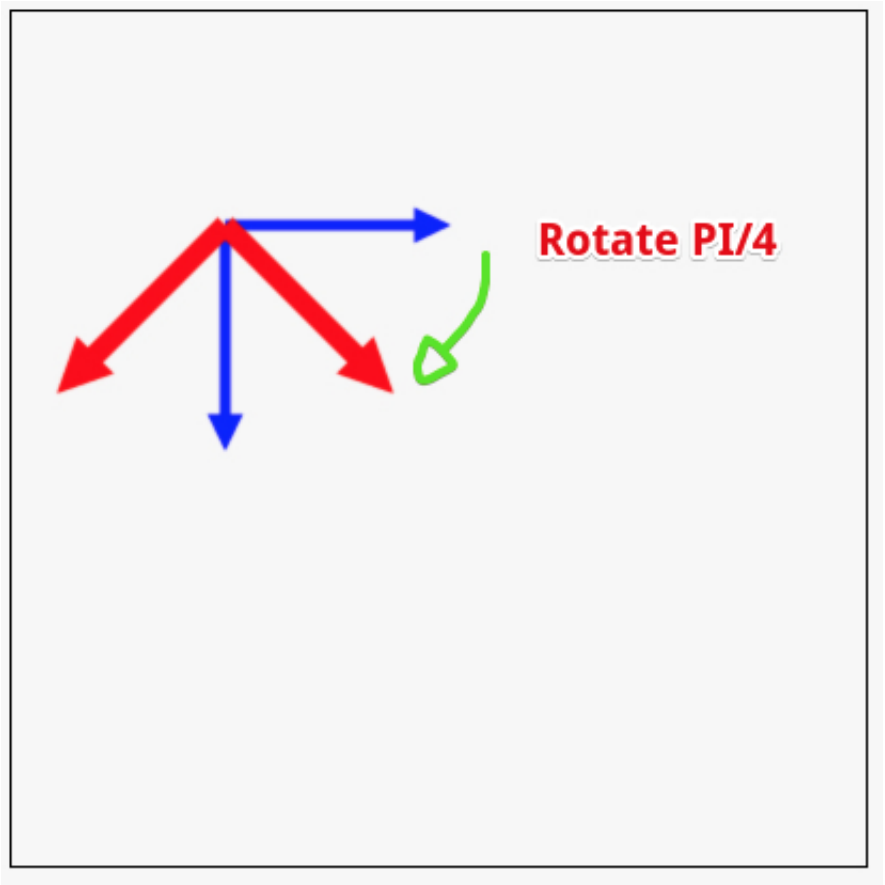
```
    ctx.translate(100, 100);
    ctx.rotate(Math.PI/4);
```

```
ctx.scale(0.5, 0.5);

drawMonster(0, 0);
```

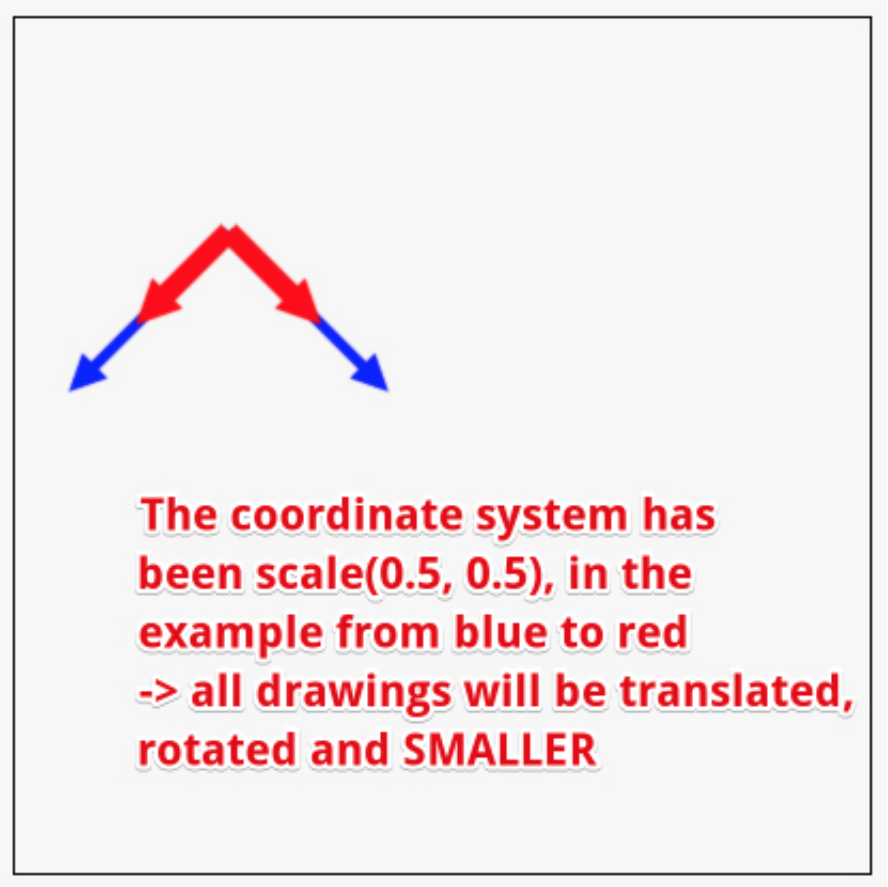
Line 1 does:



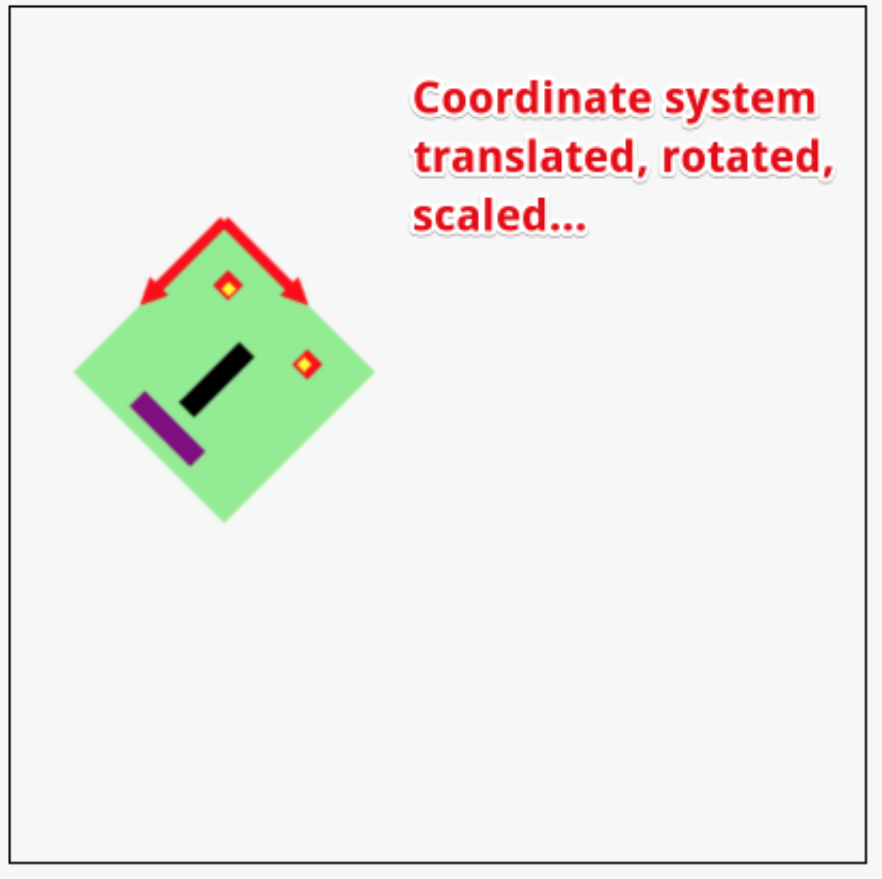
Line 2 does:



Line 3 does:



Line 4 draws the monster in this new translated, rotated, scaled coordinate system:



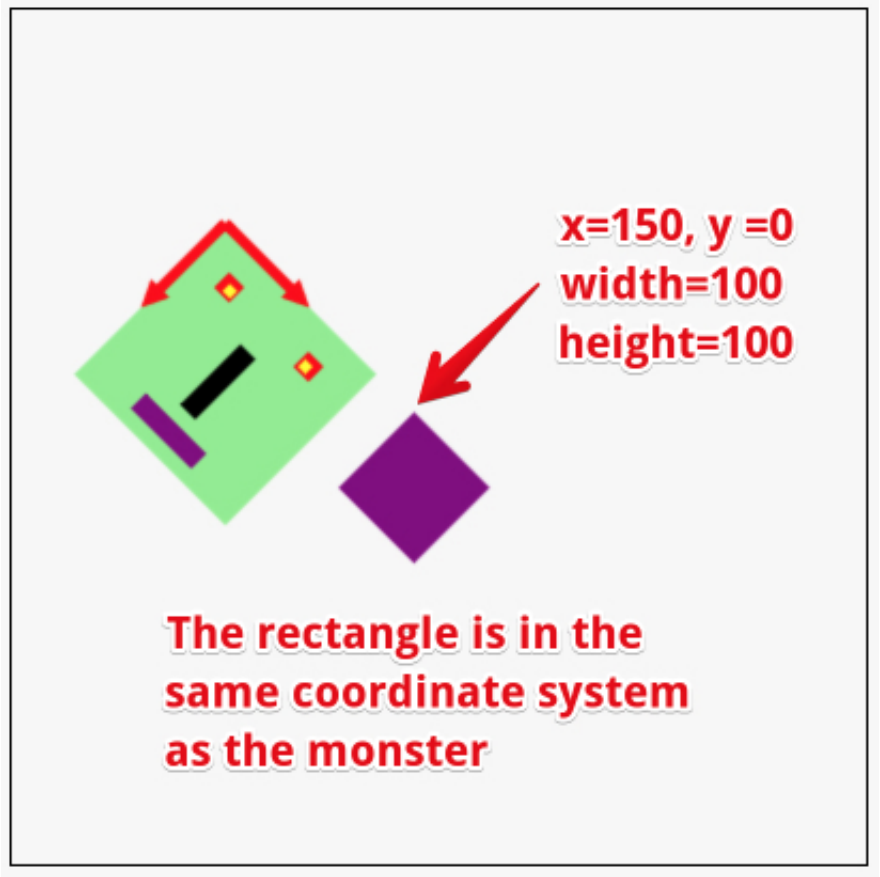
BEWARE: ALL DRAWINGS TO COME WILL BE IN THAT MODIFIED COORDINATE SYSTEM!

If we draw two shapes at two different positions, they will be relative to this new coordinate system.

For example, this code (online athttp://jsbin.com/yuhamu/4/edit):

```
ctx.translate(100, 100);
ctx.rotate(Math.PI/4);
ctx.scale(0.5, 0.5);
 // Draw the monster at (0, 0)
drawMonster(0, 0);
 // Draw a filled rectagle at (250, 0)
ctx.fillRect(250, 0, 100, 100);
```

... gives this result (there is an error in the red text below, it should say "x=250, y=0"):



HOW CAN WE RESET THE COORDINATE SYSTEM, HOW CAN WE GO BACK TO THE PREVIOUS ONE?

Aha, this is a very interesting question... we will give the answer very soon in the section on saving/restoring the context :-)

KNOWLEDGE CHECK 3.2.8 (NOT GRADED)

```
ctx.translate(100, 100);
ctx.scale(2, 2);
ctx.strokeRect(0, 0, 100, 100);
```

Where will the rectangle be drawn?

- x=100, y=100, twice the size

- x=0, y=0, twice the size
- x=200, y=200, size unchanged
- x=200, y=200, twice the size