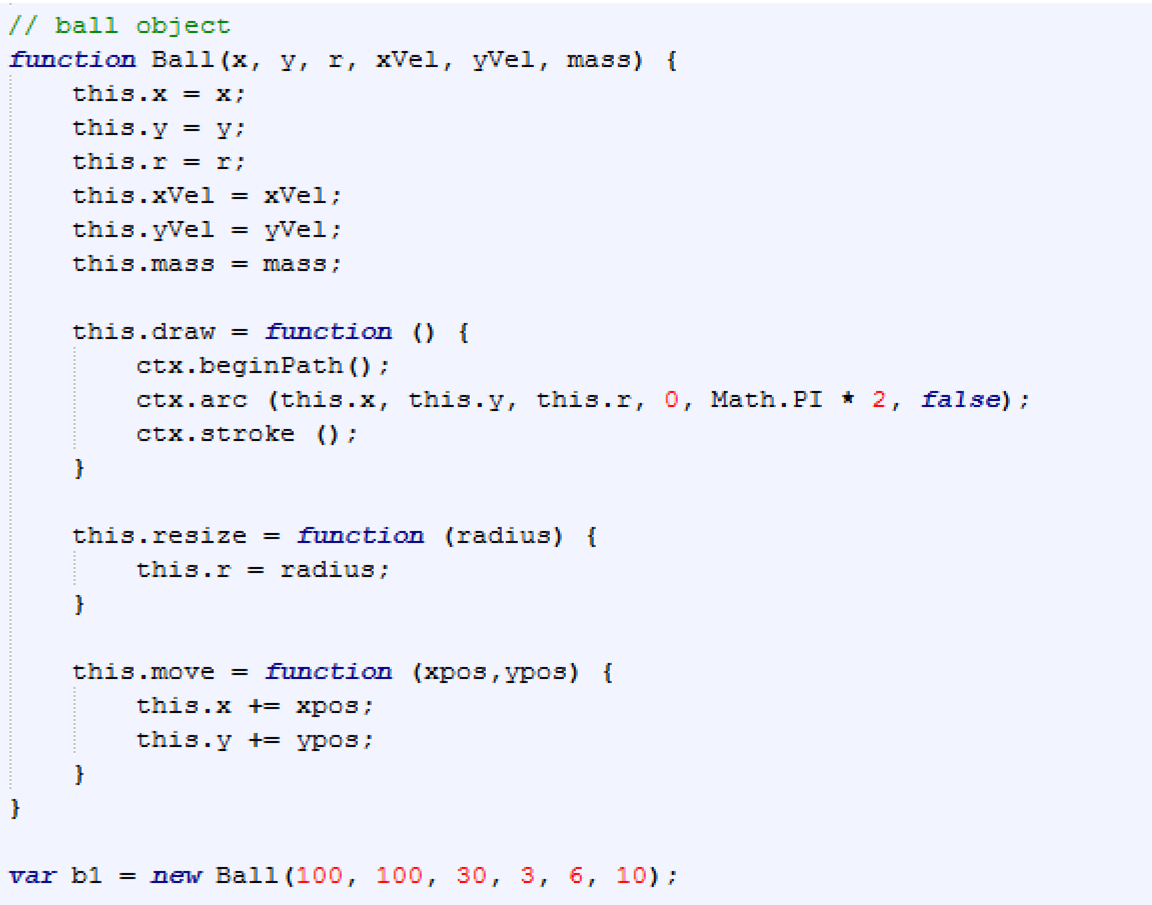
Lecture 6

Object Oriented JavaScript



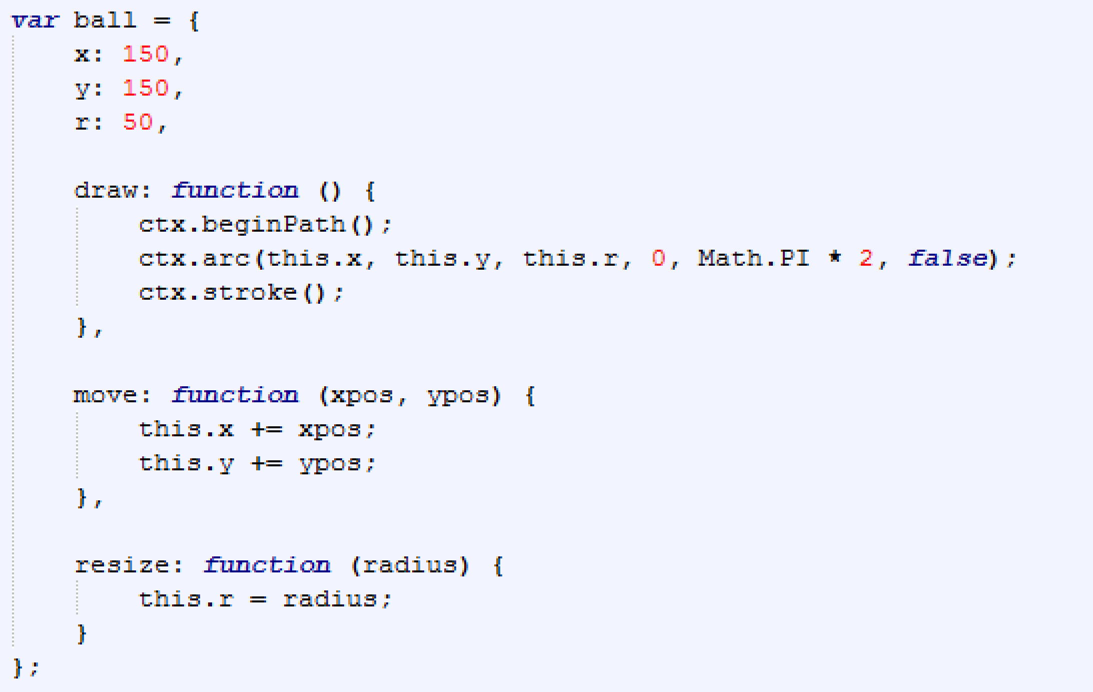
```javascript
// ball object
function Ball(x, y, r, xVel, yVel, mass) {
    this.x = x;
    this.y = y;
    this.r = r;
    this.xVel = xVel;
    this.yVel = yVel;
    this.mass = mass;

    this.draw = function () {
        ctx.beginPath();
        ctx.arc (this.x, this.y, this.r, 0, Math.PI * 2, false);
        ctx.stroke ();
    }

    this.resize = function (radius) {
        this.r = radius;
    }

    this.move = function (xpos,ypos) {
        this.x += xpos;
        this.y += ypos;
    }
}

var b1 = new Ball(100, 100, 30, 3, 6, 10);
```



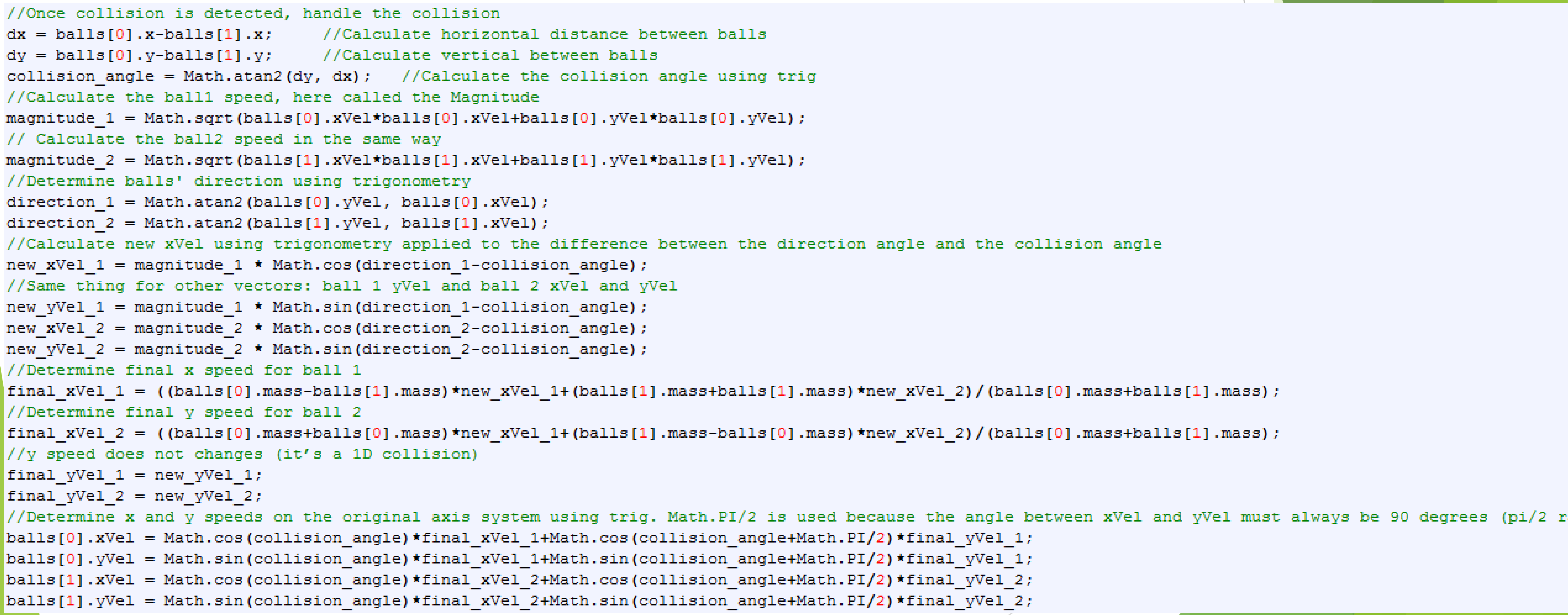
```javascript
var ball = {
    x: 150,
    y: 150,
    r: 50,

    draw: function () {
        ctx.beginPath();
        ctx.arc(this.x, this.y, this.r, 0, Math.PI * 2, false);
        ctx.stroke();
    },

    move: function (xpos, ypos) {
        this.x += xpos;
        this.y += ypos;
    },

    resize: function (radius) {
        this.r = radius;
    }
};
```

Elastic collision – 2Balls – 2 dimensions



```
//Once collision is detected, handle the collision
dx = balls[0].x-balls[1].x;     //Calculate horizontal distance between balls
dy = balls[0].y-balls[1].y;     //Calculate vertical between balls
collision_angle = Math.atan2(dy, dx);   //Calculate the collision angle using trig
//Calculate the ball1 speed, here called the Magnitude
magnitude_1 = Math.sqrt(balls[0].xVel*balls[0].xVel+balls[0].yVel*balls[0].yVel);
// Calculate the ball2 speed in the same way
magnitude_2 = Math.sqrt(balls[1].xVel*balls[1].xVel+balls[1].yVel*balls[1].yVel);
//Determine balls' direction using trigonometry
direction_1 = Math.atan2(balls[0].yVel, balls[0].xVel);
direction_2 = Math.atan2(balls[1].yVel, balls[1].xVel);
//Calculate new xVel using trigonometry applied to the difference between the direction angle and the collision angle
new_xVel_1 = magnitude_1 * Math.cos(direction_1-collision_angle);
//Same thing for other vectors: ball 1 yVel and ball 2 xVel and yVel
new_yVel_1 = magnitude_1 * Math.sin(direction_1-collision_angle);
new_xVel_2 = magnitude_2 * Math.cos(direction_2-collision_angle);
new_yVel_2 = magnitude_2 * Math.sin(direction_2-collision_angle);
//Determine final x speed for ball 1
final_xVel_1 = ((balls[0].mass-balls[1].mass)*new_xVel_1+(balls[1].mass+balls[1].mass)*new_xVel_2)/(balls[0].mass+balls[1].mass);
//Determine final y speed for ball 2
final_xVel_2 = ((balls[0].mass+balls[0].mass)*new_xVel_1+(balls[1].mass-balls[0].mass)*new_xVel_2)/(balls[0].mass+balls[1].mass);
//y speed does not changes (it's a 1D collision)
final_yVel_1 = new_yVel_1;
final_yVel_2 = new_yVel_2;
//Determine x and y speeds on the original axis system using trig. Math.PI/2 is used because the angle between xVel and yVel must always be 90 degrees (pi/2 r
balls[0].xVel = Math.cos(collision_angle)*final_xVel_1+Math.cos(collision_angle+Math.PI/2)*final_yVel_1;
balls[0].yVel = Math.sin(collision_angle)*final_xVel_1+Math.sin(collision_angle+Math.PI/2)*final_yVel_1;
balls[1].xVel = Math.cos(collision_angle)*final_xVel_2+Math.cos(collision_angle+Math.PI/2)*final_yVel_2;
balls[1].yVel = Math.sin(collision_angle)*final_xVel_2+Math.sin(collision_angle+Math.PI/2)*final_yVel_2;
```

Lab

- As usual, create and resize a canvas
- Create two ball objects
    - draw, move, setColour, setMass methods
    - x, y, r, xVel, yVel, colour, mass attributes
- First implement collision detection and reaction between balls and walls
- Next implement inter-ball collision detection
- Once collisions can be successfully detected, implement ball reactions
- Advanced:
    - Experiment with different sphere masses
    - Implement collision system for several balls (scalable code if possible)

Nice demo

- https://scratch.mit.edu/projects/116144988/#player