```
var DIAMETER_BALL =         0.04267; //m
var AREA_BALL =      0.00143; //m^2
var GRAVITY = 9.81; // m/s^2
var MASS_BALL = 0.0457; //kg
var DELTA_T = 0.03; //s   <- change in time
var RHO = 1.2041; //kg/m^3
var NU = 0.0000146; //kinematic viscosity of air -> m^2/s
var SPIN_DECAY = 0.04; //spin decay per second
var pi = 3.14159265359

function start(){
   var V_BALL = readFloat("What is the ball speed?(mph) "); //mph
   var BALL_VELOCITY = V_BALL*1609.34/60/60; //m/s
   var launchAngle = readFloat("What is the launch angle?(deg) "); // deg
   var LAUNCH_ANGLE = launchAngle*pi/180; //radians
   var RPM = readFloat("What is the spin rate?(rpm) "); //rpm
   var OMEGA = RPM*2*pi/60; // rad/sec

   //ask user if they want to know the carry distance, max height, or both. User will input initial
ball velocity, launch angle, and spin rate as well as the form of data they want to see.
   var question = readLine("Would you like to know the carry distance or view a graph? ");

   calculateDistance(BALL_VELOCITY, LAUNCH_ANGLE, RPM, question);
}

/*this function calculates the Reynold's Number when
given the velocity of the ball
the equation also uses the diameter of the ball and the kinematic
viscosity of air
*/
function findRE(Velocity){
   var RE = Velocity*DIAMETER_BALL/NU;
   return RE;
}

/*
This function is used to calibrate later equations (that use the drag coeficient)
to empirical data to provide accurate results.
refrence: https://blog.trackmangolf.com/trackman-average-tour-stats/
*/
function dragMultiplier(RPM){
   var result = -0.000000019*(RPM-6750)*(RPM-6750)+1.52;
   return result;
}
```

```
/*
Because balls traveling at different speeds have different drag coeficients,
this function uses equations based off of empirical data to return a drag
coeficient for a given Reynold's Number
refrence for empirical data:
https://www.semanticscholar.org/paper/Aerodynamics-of-Golf-Balls-in-Still-Air-Lyu-Kensrud/595
d3d1f91e240cb47e153b8d0ef586b8caa67c4/figure/3
*/
function dragCoeficient(RE, DRAG_MULTIPLIER){
  var result = 0;
  if(RE<81207.187){
     //low speed
     result = 0.000000000129*RE*RE-0.0000259*RE+1.5;
  }else if(RE<141361.257){
     //high speed
     result = (1.91)*(0.00000000001)*RE*RE-0.0000054*RE+0.56;
  }else{
     //"higher" speed
     result =0.178;
  }
  var finalResult = result*DRAG_MULTIPLIER;
  return finalResult;
}

/*
This function finds the force of drag on the ball.
R=1/2 *Cd*rho*A*V^2
The unit of the result is Newtons.
For the parameters, v represents ball velocity and Cd represents the coeficient
of drag.
*/
function airResistance(V, Cd){
    var result = 0.5*Cd*RHO*AREA_BALL*V*V;
    return result;
}

/*
As a ball moves through the air, its spin rate changes over time
For the purposes of this project, the rate of decay is assumed to be 4%

reference:
https://www.semanticscholar.org/paper/Aerodynamics-of-Golf-Balls-in-Still-Air-Lyu-Kensrud/595
d3d1f91e240cb47e153b8d0ef586b8caa67c4
```

```
go to section "3.3. Golf Ball Trajectory Simulation" (of reference article)

This function returns the angular velocity of the ball for a given amount of time
the parameter t represents time
*/
function angularVelocity(t, OMEGA){
    var result = OMEGA *(1- SPIN_DECAY*t)
    return result;
}

/*
To find the force of lift exerted on the ball, we first need to find the
coeficient of lift. The coeficeint of lift can be determined based off of a
spin factor (S)
reference:https://www.semanticscholar.org/paper/Aerodynamics-of-Golf-Balls-in-Still-Air-Lyu-Ke
nsrud/595d3d1f91e240cb47e153b8d0ef586b8caa67c4/figure/4

where S = (angular velocity)*(radius of ball)/(velocity of ball)
*/
function spinFactor(omega, velocity){
    var s = omega * DIAMETER_BALL /2 /velocity;
    return s;
}
// S represents the spin factor and liftMult represents the lift multiplier
function coeficientLift(S, liftMult){
    if(S<0.2145){
        var result = -2.4*S*S+1.8*S;
    }else{
        var result = 1/1.68 * Math.sqrt(S);
    }
    result = result * liftMult;
    return result;
}

//used to calibrate drag multiplier and lift multiplier functions to empirical data (from Trackman
reference)
function findRPM(omega){
    var result = omega/2/pi *60;
    return result;
}

function liftMultiplier(rpm){
    var result = -0.0000000047*rpm*rpm+1.05;
    return result;
```

```
}

/*
This function is used to calculate the force of lift exerted on the ball
Cl represents the coeficient of lift and V represents the velocity of the ball.
unit for result in Newtons
*/
function forceLift(Cl, V){
   var result = 0.5*Cl*RHO*AREA_BALL*V*V;
   return result;
}

//this function calculates the force of gravity exerted on the ball
//unit for result in Newtons
function forceGravity(){
   var result = MASS_BALL*GRAVITY;
   return result;
}

//this function returns the angle of the direction of the velocity for an inputed x velocity and y
velocity
function findTheta(Vx, Vy){
   return (Math.atan(Vy/Vx));
}

/*
This function calculates the acceleration of the ball in the y direction by suming the forces acting
on the ball and dividing by the mass
*/
function accelerationY(theta, lift, drag, gravity){
   var a = (lift*Math.cos(theta) - drag*Math.sin(theta) - gravity)/MASS_BALL;
   return a;
}

//acceleration in the x direction
function accelerationX(theta, lift, drag){
   var a = (-lift*Math.sin(theta) - drag*Math.cos(theta)) /MASS_BALL;
   return a;
}

//functions to find x velocity and y velocity at a given time
//Vf=Vi +at
//where Vi is initial velocity and a is acceleration
function findVx(Vix, ax){
    var result = Vix + ax*DELTA_T;
    return result;
}
function findVy(Viy, ay){
    var result = Viy + ay*DELTA_T;
    return result;
}

//y1+ ay*t+0.5ay t^2
//y1 is the previous y value
// the first values of x and y are 0 because the ball starts at (0,0)
//findY and findX return the ball's position relative to the orgin (0,0)
function findY(y1, Vy, ay){
    var result = y1 + Vy*DELTA_T + 0.5*ay*DELTA_T*DELTA_T;
    return result;
}
function findX(x1, Vx, ax){
    var result = x1 + Vx*DELTA_T + 0.5*ax*DELTA_T*DELTA_T;
    return result;
}

//this function finds the total velocity of the ball for a given x velocity and a given y velocity
//the total velocity is needed to calculate the Reynolds number and to calculate forces
function findVelocity(Vx, Vy){
    var result = Math.sqrt(Vx*Vx + Vy*Vy);
    return result;
}




/*
To calculate the trajectory of the ball, we must use a numerical solution.
To do this, we repeat steps (ex. find magnitude of each force, then combine forces, then find
acceleration, then find velocity, then find distance)
This function will combine the previous functions used to carry out the smaller steps.
arrays will contain the x and y position of the ball for each value of time.

The inputs will be the user inputed values for ball speed, launch angle, and spin rate //the
inputed values should be in the correct units
userStr represents the type of information that the user asked for          ex. carry distance or
graph
*/
function calculateDistance(ballSpeed, LAUNCH_ANGLE, RPM, userStr){
    //first, set x and y positions to zero because the ball starts at the position (0,0)
    var x = 0;
    var y = 0;
    var theta = LAUNCH_ANGLE;
    var OMEGA = RPM*2*pi/60;
    var distanceX = [];
    var distanceY = [];
    var COUNT = 400; //count at 400 for high ball speeds and/or to ensure enough data points
    var rpm = RPM;
    var Vball = ballSpeed;
    var dragMult = dragMultiplier(RPM);
    var liftMult = liftMultiplier(RPM);
    var time = 0;
    var RE = findRE(Vball);
    var Cd = dragCoeficient(RE, dragMult);
    var omega = angularVelocity(time, OMEGA);
    var S = spinFactor(omega, Vball);
    var Cl = coeficientLift(S, liftMult);
    var R = airResistance(Vball, Cd);
    var Fl = forceLift(Cl, Vball);
    var Fg = forceGravity();
    var ax = accelerationX(theta, Fl, R);
    var ay = accelerationY(theta, Fl, R, Fg);
    var vx = Vball*Math.cos(theta);
    var vy = Vball*Math.sin(theta);
    x = findX(x, vx, ax)
    y = findY(y, vy, ay);

    //for loop to repeat steps(finding forces...) for numerical solution
    //the value of COUNT does not mater but it should be large enough so that the ball crosses
the x axis again to calculate the carry distance
    for(var i = 0; i < COUNT; i++){
        theta = findTheta(vx, vy)
        RE = findRE(Vball);
        Cd = dragCoeficient(RE, dragMult);
        omega = angularVelocity(time, OMEGA);
        rpm = findRPM(omega);
        S = spinFactor(omega, Vball);
        Cl = coeficientLift(S, liftMult);
        R = airResistance(Vball, Cd);
        Fl = forceLift(Cl, Vball);
        ax = accelerationX(theta, Fl, R);
        ay = accelerationY(theta, Fl, R, Fg);
        vx = findVx(vx, ax);
        vy = findVy(vy, ay);
        x = findX(x, vx, ax);
        y = findY(y, vy, ay);
        Vball = findVelocity(vx, vy);

        //push values of x and y into arrays
        distanceX.push(x);
        distanceY.push(y);
        //increase time for next loop
        time += DELTA_T;
    }

    //return values that the user asked for
    if(userStr == "carry distance"){
        //carry distance is how far in the x direction the ball traveled in the air (before hiting the
ground)
        //use distanceY array to find the maximum height of the ball
        var maxY = 0;
        for(var i = 0; i < distanceY.length; i++){
            if(distanceY[i] > maxY){
                maxY = distanceY[i]
            }
        }
        var maxX = 0;
        for(var i = 0; i < distanceX.length; i++){
            if(distanceY[i] > 0){
                if(distanceX[i] > maxX){
                    maxX = distanceX[i];
                }
            }
        }
        //convert from meters to yards
        maxY = maxY*1.09361
        maxX = maxX*1.09361

        println("Carry Distance: "+ maxX + " yards");
        println("Max Height: "+maxY+ " yards");
    }else{
        //provide graph of data
        //first conform x values so that the graph only shows positive x and positive y values
        for(var i = 0; i < distanceY.length; i++){
            if(distanceY[i] < 0){
                distanceY.remove(i);
                distanceX.remove(i);
            }
        }
        for(var i = 0; i < distanceY.length; i++){
            if(distanceY[i]>= -2){
                var circle = new Circle(1);
                circle.setPosition(distanceX[i], 100 - distanceY[i]);
                circle.setColor(Color.blue);
                add(circle);
            }
        }
    }
}
```