```
// Ramp simulation with Potential and Kinetic Energy
// shared under Creative Commons license
// https://creativecommons.org/licenses/by-nc-sa/3.0/
// Created by: Jason Galbraith -- December 2017

// Last Updated 10/28/25
//Converted to a p5 Javascript Simulation

/*
**************************************************************
Change the numbers below to see how it changes the simulation!
**************************************************************
 */
//This is the height of the ramp in centimeters, range of 0.1 - 24.5
//Your ramp is only 25 cm long, so calulations are incorrect above that!
let height = 5;//cm
//This is the mass of the ball object
let mass = 5;//kg
//Yes, you can alter gravity
let gravity = 9.8;//m/s^2
//This is a delay in microseconds per step for the simulation
let delay = 50;

//These are all the colors in the simulation. They are hex colors.
//You can find new ones at https://www.w3schools.com/colors/colors_picker.asp
let backgroundColor = "#CCCCCC";
let rampColor = "#000000";
let ballColor = "#2DBCBA";
let velocityColor = "#0000FF";
let massColor = "#8C6632";
let heightColor = "#000000";
let gravityColor = "#666666";
let peColor = "#FF0000";
let keColor = "#227C40";
let defaultColor = "#000000";

/*
****************************************************************
Below here is the actual simulation. You probably shouldn't mess
with this section. Or at least, you should expect weird results.
****************************************************************
 */
let rampScale = 0;
let graphScale = 0;
```

```
let rampLength = 25;//cm
let rampBase = Math.sqrt(rampLength*rampLength-height*height);
let pixgravity = gravity * (100.0/1.0) * (30.0/1.0) * (1.0/100000.0);
//                m/s^2       cm/m        pix/cm      s^2/hundredThousandthSeconds (approximate
refresh rate)
let rampHeight = Math.round(height * (30.0/1.0));
//                    cm       pix/cm
let rampBottom = 200;
let rampTop = rampBottom - rampHeight;
let rampLeft = 10;
let rampRight = rampLeft+(rampBase*(30.0/1.0));//510
let ballAcc = 0;
let ballX = 0;
let ballY = 0;
let ballVX = 0;
let ballVY = 0;
let ballCenterX = 0;
let ballCenterY = 0;
let radius = 10;
let rampSlope = 0;
let rampAngleRadians = Math.atan2(rampHeight, rampRight-rampLeft);
let formulaFont;
let graphFont;
let ke = [];
let pe = [];
let eCount = 0;
let eHistory = 1200;

//Initial setup of the simulation
function setup () {
  createCanvas(1000, 500);
        if (rampHeight > 200) {
                        rampScale = 200 / rampHeight;
                rampHeight = rampHeight * rampScale;
                rampRight = rampRight * rampScale;
                radius = radius * rampScale;
                rampTop = rampBottom - rampHeight;
        }
        else {
                        rampScale = 1.0;
        }
        ellipseMode(CENTER);
        ballX = rampLeft;
        ballY = rampBottom-rampHeight;
```

```
        rampSlope = (rampBottom-rampTop)/(rampRight-rampLeft);
        rampPerpendicular = -pow(rampSlope, -1);
        //Set up graph arrays, fonts, and scale
        //formulaFont = loadFont(("Impact",26);
        //graphFont = loadFont(("Impact",20);
        ke = [eHistory];
  pe = [eHistory];
        for (let a = 0; a < eHistory; a++) {
                ke[a] = -1;
            pe[a] = -1;
        }
        graphScale = 120.0/(mass*pixgravity*(rampBottom-ballY)*(1.0/100.0));
}

//Updates the physics of acceleration, velocity, and position
function update() {
        ballAcc = pixgravity*sin(rampAngleRadians);
        ballVX += cos(rampAngleRadians)*ballAcc;
        ballVY += sin(rampAngleRadians)*ballAcc;
        ballX += ballVX;
        ballY += ballVY;
        if (ballY > rampBottom) {//exact final calculation
            ballY = rampBottom;
        }
}
//Drawing loop; draws every step
function draw() {
        //As long as the ball has not gone off the ramp...
        if (ballY < rampBottom) {
                update(); //Update the physics
                background(backgroundColor);
                //Draw the ramp
                fill(rampColor);
                stroke(rampColor);
                triangle(rampLeft, rampTop, rampLeft, rampBottom, rampRight, rampBottom);
                //Draw the ball
                fill(ballColor);
                ballCenterX = ballX+radius*cos(rampAngleRadians+(PI/2));
                ballCenterY = ballY-radius*sin(rampAngleRadians+(PI/2));
                ellipse(ballCenterX, ballCenterY, radius*2, radius*2);
                //Draw velocity vector
                fill(velocityColor);
                stroke(velocityColor);
```

```
                drawArrow(ballCenterX, ballCenterY, ballCenterX+(ballVX*10),
ballCenterY+(ballVY*10));
                //Draw the gravity vector
                fill(gravityColor);
                stroke(gravityColor);
                drawArrow(ballCenterX, ballCenterY, ballCenterX, ballCenterY+50);
                //Draw normal vector
                fill(heightColor);
                stroke(heightColor);
                drawArrow(ballCenterX, ballCenterY, ballCenterX+sin(rampAngleRadians)*50,
ballCenterY-cos(rampAngleRadians)*50);
                //Starting height and ramp length labels
                //textFont(formulaFont,26);
                fill(heightColor);
      textSize(26);
                text("Start Height:"+sigPre(height, 2)+" cm",600,30);
                fill(heightColor);
                text("Ramp Length:"+sigPre(rampLength, 2)+" cm",600,60);
                //Starting mass label
                fill(massColor);
                text("Mass:"+sigPre(mass, 2)+" kg",600,90);
                //Acceleration label
                fill(gravityColor);
                let acc = ballAcc * (1.0/100.0) * (1.0/30.0) * (100000.0/1.0);
  //          m/s^2     m/cm        cm/pix  hundredThousandthSeconds/s^2 (approximate refresh
rate)
                text("Acceleration:"+sigPre(acc, 2)+" m/s^2",600,120);
                //Velocity label
                fill(velocityColor);
                let ballVel = sqrt(ballVX*ballVX+ballVY*ballVY);
                let vel = ballVel * (1.0/100.0) * (1.0/30.0) * sqrt(100000.0/1.0);
  //          m/s       m/cm        cm/pix  hundredThousandthSeconds/s (approximate refresh
rate)
                text("Velocity:"+sigPre(vel, 2)+" m/s",600,150);
                pe[eCount] = (mass*pixgravity*(rampBottom-ballY)*(1.0/100.0));
                //Constants for drawing the PE/KE formulas
                let formulaLeft = 0;
                let theoryPEBottom = 200+45;
                let formulaPEBottom = 200+70;
                let theoryKEBottom = 200+105;
                let formulaKEBottom = 200+130;
                //PE formula
                fill(peColor);
                text("PE:", formulaLeft, theoryPEBottom);
```

```
                text(sigPre(mass*pixgravity*(rampBottom-ballY)*(1.0/100.0), 2) + " J",
formulaLeft, formulaPEBottom);
                text("=", formulaLeft+150,theoryPEBottom);
                text("=", formulaLeft+150,formulaPEBottom);
                fill(massColor);
          text("mass", formulaLeft+200, theoryPEBottom);
                text(sigPre(mass, 1)+" kg", formulaLeft+200, formulaPEBottom);
                fill(peColor);
                text("*", formulaLeft+300, theoryPEBottom);
                text("*", formulaLeft+300, formulaPEBottom);
                fill(gravityColor);
                text("gravity", formulaLeft+350, theoryPEBottom);
                text(sigPre(gravity, 1)+" m/s^2", formulaLeft+350, formulaPEBottom);
                fill(peColor);
                text("*", formulaLeft+500, theoryPEBottom);
                text("*", formulaLeft+500, formulaPEBottom);
                fill(heightColor);
                text("height", formulaLeft+550, theoryPEBottom);
                text(sigPre((rampBottom-ballY)*(1.0/30.0)*(1.0/100.0),3)+" m", formulaLeft+550,
formulaPEBottom);
                //KE formula
                fill(keColor);
                ke[eCount] = (0.5*mass*(ballVX*ballVX+ballVY*ballVY)*(1.0/100.0));
                text("KE:", formulaLeft, theoryKEBottom);
                text(sigPre(0.5*mass*(ballVX*ballVX+ballVY*ballVY)*(1.0/100.0), 2) + " J",
formulaLeft, formulaKEBottom);
                text("=", formulaLeft+150, theoryKEBottom);
                text("=", formulaLeft+150, formulaKEBottom);
                fill(defaultColor);
                text("const", formulaLeft+200, theoryKEBottom);
                text("0.5", formulaLeft+200, formulaKEBottom);
                fill(keColor);
                text("*", formulaLeft+300, theoryKEBottom);
                text("*", formulaLeft+300, formulaKEBottom);
                fill(massColor);
                text("mass", formulaLeft+350, theoryKEBottom);
                text(sigPre(mass, 1)+" kg", formulaLeft+350, formulaKEBottom);
                fill(keColor);
                text("*", formulaLeft+500, theoryKEBottom);
                text("*", formulaLeft+500, formulaKEBottom);
                fill(velocityColor);
                text("velocity squared", formulaLeft+550, theoryKEBottom);
                text(sigPre((ballVX*ballVX+ballVY*ballVY)*(1.0/100.0), 3)+" (m/s)^2",
formulaLeft+550, formulaKEBottom);
```

```
                //Update count, draw graph, and delay the frame
                eCount++;
                drawGraph();
                newtime = millis();
                while (millis() < newtime + delay) {}
        }
}

//Draws arrow from point (x1,y1) to (x2,y2), draws the arrow head at (x2,y2)
function drawArrow(x1, y1, x2, y2) {
        let arrowAngle = PI/6;
        let arrowLength = 6;
        line(x1, y1, x2, y2);
        let arrowRadians = atan2(y2-y1, x2-x1);
        let tx1 = x2;
        let ty1 = y2;
        let tx2 = x2-arrowLength*cos(arrowRadians-(arrowAngle));
        let ty2 = y2-arrowLength*sin(arrowRadians-(arrowAngle));
        let tx3 = x2-arrowLength*cos(arrowRadians+(arrowAngle));
        let ty3 = y2-arrowLength*sin(arrowRadians+(arrowAngle));
        triangle(tx1, ty1, tx2, ty2, tx3, ty3);
}

//Draws the PE/KE graph at the bottom of the simulations
function drawGraph() {
        graphWidth = 850 / eCount;
        //textFont(graphFont,20);
        stroke(peColor);
        fill(peColor);
        text("PE:"+sigPre(pe[0],2), 0, 500-round(pe[0]*graphScale));
        stroke(keColor);
        fill(keColor);
        text("KE:"+sigPre(ke[0],2), 0, 500-round(ke[0]*graphScale));
        lastIndex = 0;
        for (let a = 0; a < eHistory; a++) {
                if (pe[a] >= 0) {
                        stroke(peColor);
                        fill(peColor);
                        ellipse(a*graphWidth+70,490-round(pe[a]*graphScale),3,3);
                }
                if (ke[a] >= 0) {
                        stroke(keColor);
                        fill(keColor);
                        ellipse(a*graphWidth+70,490-round(ke[a]*graphScale),3,3);
```

```
                }
            }
            stroke(defaultColor);
            fill(defaultColor);
            text(sigPre(eCount/60.0, 2)+" seconds",850,350);
}

//This sets the precision of the floating point decimals
function sigPre(value, digits) {
             return round(value*pow(10, digits))/pow(10, digits);
}
```