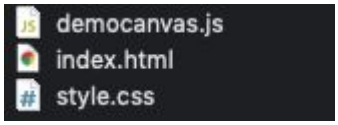
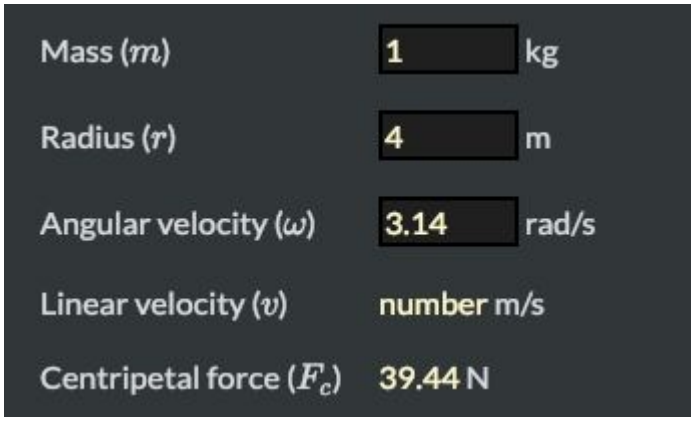
Criterion C: Development

The index.html file opens a webpage with three different 2D simulations of first-year uniform circular motion scenarios. The democanvas.js file draws the simulations on HTML canvases. The style.css file affects padding, font, font color, and background color. All 3 files should be in the same folder.

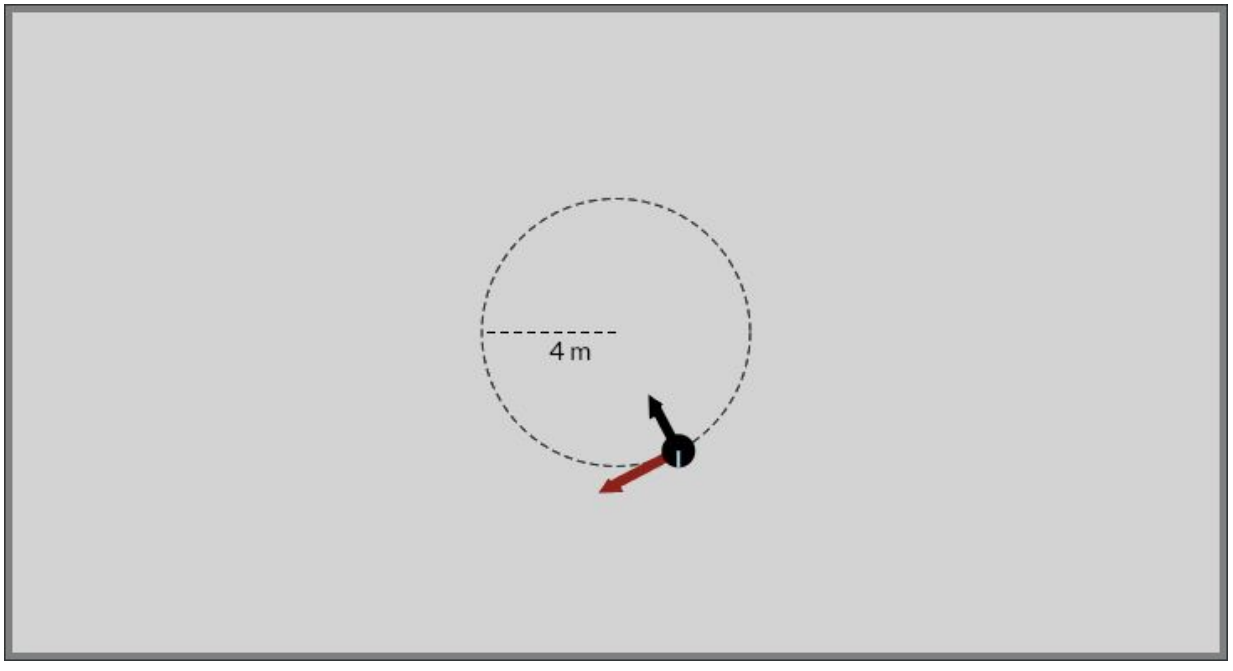


Webpage Structure using Simulation 1 as an Example

The webpage has three simulations that follow the same structure. The numeric values of the simulations are displayed in a table above the canvas; the independent values that can be edited are input text fields that change the simulation every time they are changed, and the dependent values are displayed as text that updates with each edit. This is the first table.



The HTML canvas draws the simulation 50 times per second using setInterval(function, delay), each time changing the position of the moving object based on the change in time. 50 updates per second made the animation smooth and didn't sacrifice a large amount of performance. In simulation 1, the canvas draws a circle moving in a circular path at a constant speed and shows the values in the table using the equation $F_C = \frac{mv^2}{r}$.



The second simulation involved a simulation of the friction force holding an object to a spinning surface. I drew ellipses using another object in order to create a sense of flatness and then moved them using the same processes as the circles in simulation 1, this time multiplying the vertical movement by a smaller value to achieve elliptical orbit. Simulation 3 simulates a tetherball scenario where an object is connected to a post by a string and spins around the post.

Input and Output on the Canvas
I used the MDN web docs and w3schools canvas tutorial to draw on the canvases.

In order to update text on the web page and receive user input from the input fields, I labeled each element with a unique id in the HTML file and used document.getElementByID() to get each one. After storing the elements in a variable, I could edit text by setting the element's innerHTML and I could get input values by getting the element's value.

```
var massInput1 = document.getElementById("mass1");
var radiusInput1 = document.getElementById("radius1");
var angVelInput1 = document.getElementById("angularVelocity1");
var cenForce1 = document.getElementById("cenForceText1");
var linVel1 = document.getElementById("linVelText1");
```

To initially draw the shapes, I created objects that took position and size values as parameters.

```
function circle(simAreaName, x, y, radius, startAngle, endAngle,
    pathRadius = 0, angularVelocity = 0, color = "black", mass = 1) {…
}
```

This allowed me to create new circles from a template in each simulation. In the start function, the object calls a function to draw circles using those values with context.beginPath() and context.arc. This draws a circle whose center is positioned at the values passed into the object as parameters.

```
//canvas1
simArea1.start();

circle1 = new circle("simArea1", simArea1.canvas.width/2, simArea1.canvas.height/2,
    10, 0, 2 * Math.PI, 80, 1, "black", 1);
```

To draw the shapes, the objects update() function needs to be called. The function finds the desired canvas by finding one of the simArea objects by name and getting its context. It then draws and fills in an arc using the passed-in values. Putting this code in a function in the circle object reduces the amount of code written in the update function of the page and isolates each shape.

```
    this.update = function() {
        ctx = window[simAreaName].context;
        ctx.fillStyle = color;
        ctx.beginPath();
        ctx.arc(this.x, this.y, this.radius, this.startAngle, this.endAngle);
        ctx.fill();
        ctx.closePath();
    }
}
```

Each update, the center is moved using sine and cosine function to model the motion of a circle. I increment the radians value of the circle object and get the sine and cosine of that number, then multiply it by the desired radius of the circle's path and subtract it from the initial position. This moves the circle in a circular path where the initial position is the center.

```
circle1.radians += circle1.angularVelocity / fps;

xpos = circle1.initialx - (Math.cos(circle1.radians) * circle1.pathRadius);
ypos = circle1.initialy - (Math.sin(circle1.radians) * circle1.pathRadius);

circle1.x = xpos;
circle1.y = ypos;
```

Input

To get user input, I took the value from the input fields and used parseFloat() to convert them to floating point numbers. The values needed to be floats in case the user inputs a non-integer value.

```
circle1.mass = parseFloat(massInput1.value);
circle1.pathRadius = parseFloat(radiusInput1.value) * 20;
circle1.angularVelocity = parseFloat(angVelInput1.value);
```

When the user edits an input field, they might erase the entire field, leaving an empty value with a type of NaN (not a number). The program attempts to parse the value into a float which results in an error. I used an if statement to check each input field using isNan() which returns true if the value is NaN. If it returns true, I set the value to a default number; in this case, I set circle1.mass to 0.

```
if (isNaN(circle1.mass)) {
    circle1.mass = 0;
    massInput1.value = 0;
}
```

Pausing

I also used boolean variables to pause each canvas. In the start function of the canvas objects, I created a boolean value isPaused and an onclick function that changed isPaused.

```
var simArea1 = {
    canvas : document.getElementById("canvas1"),
    isPaused : false,
    start : function() {…
    },
    clear : function() {
        this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
    }
}
```

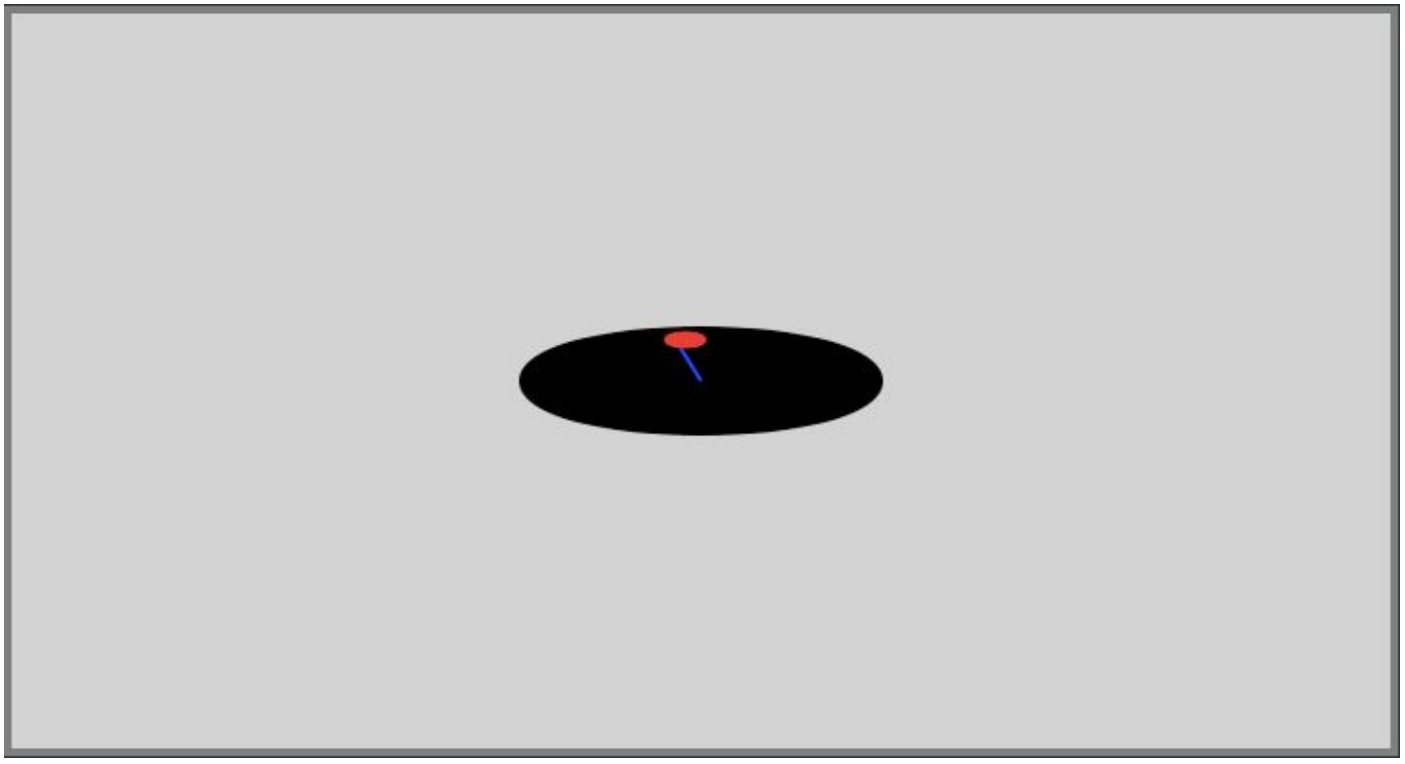
```
simArea1.canvas.onclick = function() {
    simArea1.isPaused = !simArea1.isPaused;
};
```

In the update function, I checked if isPaused was true and then used return to prevent any code from being executed, essentially pausing the canvas simulation. I chose to use a variable to keep track of pausing because it permitted a simple if statement in the update function.

```
if (simArea1.isPaused == true) {

    ctx.font = "30px Lato";
    ctx.fillStyle = "#303639";
    ctx.fillText("Paused", simArea2.canvas.width - 115, 35);

    return;

} else {
    ctx.fillStyle = "lightgray";
    ctx.fillText("Paused", simArea1.canvas.width - 50, 10);

}
```

Simulation 2

Simulation 2 calculates the maximum friction force to keep an object from losing traction on a spinning surface. I achieved this by drawing a large ellipse using bezier curves (How do You Draw an Ellipse or Oval…) with a line segment that spins around the origin to simulate spinning, as well as drawing a smaller ellipse representing the object being held by friction.

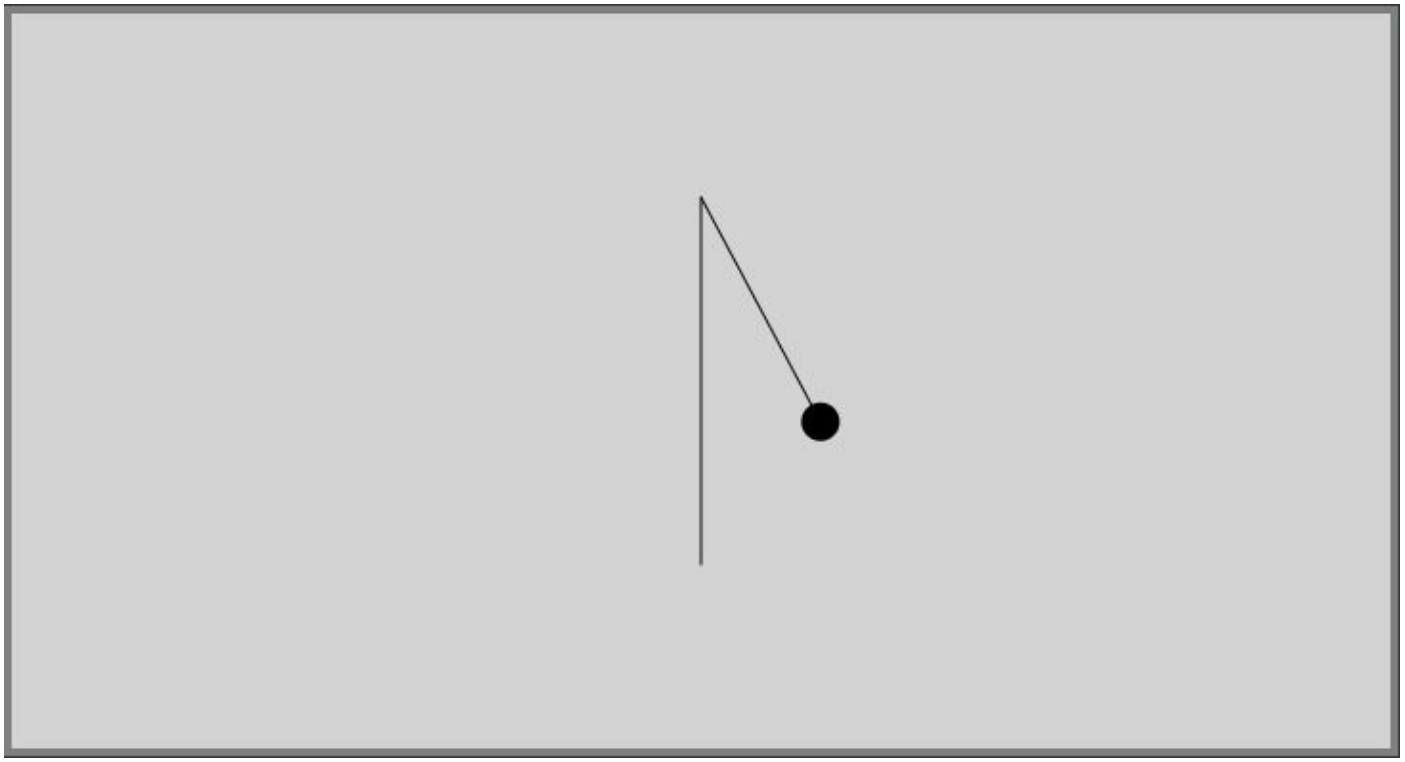


The table is similar to simulation 1, with input fields for the independent variables and text for the dependent variables. The program uses the same process to move the circle, incrementing the radians value and then finding the sine and cosine of that value. The forces are then calculated using physics equation and the independent variables specified by the user.

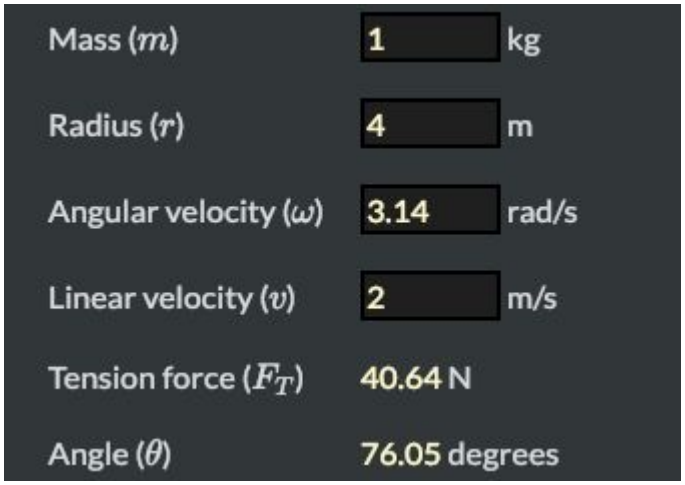
Mass (m)	1	kg
Radius (r)	4	m
Angular velocity (ω)	3.14	rad/s
Coefficient of static friction (μ)	0.5	
Centripetal force (F_c)	39.44 N	
Frictional force (F_f)	4.90 N	

Simulation 3

Simulation 3 also draws a circle moving in an elliptical orbit, but draws a line up the center of the canvas and connects the top of it with the circle using another line. This simulates a string exerting a tension force on the object to maintain uniform circular motion.



The object uses the same process of animating an elliptical path as simulation 2. The program calculates the vertical and horizontal tension of the string and uses the distance formula to calculate the total tension force. This formula involves a Math.sqrt (square root) function which is relatively inefficient compared to the other processes because it is completed every update. This could be a point of performance improvement in the future. The angle is calculated by using Math.atan() with the horizontal and vertical tension vectors. The table again takes user input and updates the text accordingly.



Word count: 1015

References

“Canvas Tutorial” *MDN web docs,* https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial

Malone, William. “HOW DO YOU DRAW A ELLIPSE OR OVAL ON HTML5 CANVAS?” *Williammalone*, www.williammalone.com/briefs/how-to-draw-ellipse-html5-canvas/.

“Canvas Tutorial” *w3schools*, https://www.w3schools.com