6.3: Using Computation to Simulate Centripetal Force

Let's use trinket to simulate centripetal forces. Below is a trinket that will create ball on the x-axis at a distance $R = 1$ with an initial velocity along the y-axis equal to $v = 1$. The program will also create an arrow to show the direction of the force being applied to the ball. Click the run button to see that these objects are created.

Now, let's simulate what happens in the absence of an applied force. Notice the ball has attributes `.pos`, `.vel`, and `.acc` for the kinematic variables. We will use the kinematic equations and the attributes `ball.pos` and `ball.vel`. Add the line below in the loop on line 17 after the `rate(1000)` line.

```
ball.pos = ball.pos + ball.vel * dt
```

You should see the ball moves vertically at constant speed according to Newton's First Law. Next, we will update the velocity using the initial acceleration. Then, we apply a force with an acceleration corresponding to $a = v^2/R$. We aim to make the ball go in a circle around the origin. To do this the force will always point toward the origin. Therefore, we define the acceleration as the unit vector pointing from the ball to the origin and multiply it by a magnitude v^2/R.

```
ball.vel = ball.vel + ball.acc * dt
ball.acc = hat(vec(0,0,0) - ball.pos) * v**2/R
```

The next time through the loop the position will be updated using the new velocity and acceleration, but we need to revisit the position calculation and add the acceleration term. Change line 17 to

```
ball.pos = ball.pos + ball.vel * dt + 0.5 * ball.acc * dt**2
```

This should be enough to make the ball move in a circle. However, we'd like the arrow to show us the acceleration at each step. Add a line after the ball.vel calculation to draw the arrow at the ball and pointed in the direction of the acceleration.

```
force.pos = ball.pos
force.axis = ball.acc
```

trinket _Code _Run _Help Share

main.py

Remix

To test whether circular motion really depends on a centripetal force with net acceleration v^2/R, try editing the magnitude of the acceleration. For example, lower the acceleration to a magnitude of `0.1*v**2/R` in the `ball.acc` calculation. What happens? Next, try a larger acceleration such as `5*v**2/R`. What does this do to the motion?

Return your calculation to `v**2/R.` You can also check to see that the relationship holds for any radius or speed. To do this, change `v` and/or `R` at the beginning of the program.