Angular velocity

Remember all this stuff?

location = location + velocity
velocity = velocity + acceleration

The stuff we dedicated almost all of the last two sections to? Well, we can apply exactly the same logic to a rotating object.

angle = angle + angular velocity
angular velocity = angular velocity + angular acceleration

In fact, the above is actually simpler than what we started with because an angle is a scalar quantity—a single number, not a vector!

Using the answer from the previous challenge, let's say we wanted to rotate a baton in ProcessingJS by some angle. We would have code something like:

```
translate(width/2, height/2);

rotate(angle);

line(-50, 0, 50, 0);

ellipse(50, 0, 8, 8);

ellipse(-50, 0, 8, 8);
```

After we add in our principles of motion, we have the program below. The baton starts onscreen with no rotation and then spins faster and faster as the angle of rotation accelerates:

```
angleMode = "radians";

var angle = 0;

var aVelocity = 0;

var aAcceleration = 0.0001;
```

```
draw = function() {
   background(220, 220, 220);
      resetMatrix();
   translate(width/2, height/2);
   rotate(angle);

   stroke(0, 0, 0);
   strokeWeight(2);
   fill(127, 127, 127);
      line(-60, 0, 60, 0);
   ellipse(60, 0, 16, 16);
   ellipse(-60, 0, 16, 16);
      angle += aVelocity;
   aVelocity += aAcceleration;
};
```

This idea can be incorporated into our `Mover` object. For example, we can add the properties related to angular motion to our `Mover` constructor.

```
var Mover = function(m, x, y) {
    this.position = new PVector(x, y);
    this.mass = m;

    this.angle = 0;
    this.aVelocity = 0;
    this.aAcceleration = 0;

    this.velocity = new PVector(random(-1, 1), random(-1, 1));
    this.acceleration = new PVector(0, 0);
};
```

And then in `update()`, we update both the position and angle according to the same algorithm!

```
Mover.prototype.update = function () {


    this.velocity.add(this.acceleration);

    this.position.add(this.velocity);


    this.aVelocity += this.aAcceleration;

    this.angle += this.aVelocity;


    this.acceleration.mult(0);

};
```

Of course, for any of this to matter, we also would need to rotate the object when displaying it.

```
Mover.prototype.display = function () {

    stroke(0, 0, 0);

    fill(175, 175, 175, 200);

    rectMode(CENTER);


    // pushMatrix and popMatrix are needed so that the shape rotation

    //  doesn't affect the rest of the world

    pushMatrix();

    // Set the origin at the shape's location

    translate(this.location.x, this.location.y);

    // Rotate by the angle

    rotate(this.angle);

    rect(0, 0, this.mass*16, this.mass*16);
    popMatrix();
};
```

Now, if we were to actually go ahead and run the above code, we wouldn’t see anything new. This is because the angular acceleration (this a Acceleration = 0;) is initialized to zero. For the object to rotate, we need to give it an acceleration! Certainly, we could hard-code in a different number:

```
this.aAcceleration = 0.01;
```

Here's what a program looks like with the above logic, with a force calculated based on a central attractor:

```
angleMode = "radians";


var Attractor = function() {
   this.position = new PVector(width/2, height/2);
   this.mass = 20;
   this.G = 1;
};


Attractor.prototype.calculateAttraction = function(m) {
   // Calculate direction of force
   var force = PVector.sub(this.position, m.position);
   // Distance between objects
   var distance = force.mag();
   // Limiting the distance to eliminate "extreme" results for very close or very far objects
   distance = constrain(distance, 5, 25);
   // Normalize vector (distance doesn't matter here, we just want this vector for direction)
```

```
    force.normalize();
    // Calculate gravitional force magnitude
    var strength = (this.G * this.mass * m.mass) / (distance * distance);
    // Get force vector --> magnitude * direction
    force.mult(strength);
    return force;
};

Attractor.prototype.display = function() {
    ellipseMode(CENTER);
    strokeWeight(4);
    stroke(0);
    ellipse(this.position.x, this.position.y, this.mass*2, this.mass*2);
};

var Mover = function(m, x, y) {
    this.position = new PVector(x, y);
    this.mass = m;

    this.angle = 0;
    this.aVelocity = 0;
    this.aAcceleration = 0.01;

    this.velocity = new PVector(random(-1, 1), random(-1, 1));
    this.acceleration = new PVector(0, 0);
};
```

```
Mover.prototype.applyForce = function(force) {
   var f = PVector.div(force, this.mass);
   this.acceleration.add(f);
};

Mover.prototype.update = function () {
   this.velocity.add(this.acceleration);
   this.position.add(this.velocity);

   this.aVelocity += this.aAcceleration;
   this.aVelocity = constrain(this.aVelocity, -0.1, 0.1);
   this.angle += this.aVelocity;

   this.acceleration.mult(0);
};

Mover.prototype.display = function () {
   stroke(0, 0, 0);
   fill(175, 175, 175, 200);
   rectMode(CENTER);
   pushMatrix();
   translate(this.position.x, this.position.y);
   rotate(this.angle);
   rect(0, 0, this.mass*16, this.mass*16);
   popMatrix();
};
```

```
var mover = new Mover(10, 200, 200);


draw = function() {
    background(255, 255, 255);
    mover.update();
    mover.display();
};


var movers = [];
var attractor = new Attractor();


for (var i = 0; i < 20; i++) {
    movers.push(new Mover(random(0.1, 2), random(width), random(height)));
}


draw = function() {
  background(0, 204, 255);


  attractor.display();


  for (var i = 0; i < movers.length; i++) {
    var force = attractor.calculateAttraction(movers[i]);
    movers[i].applyForce(force);


    movers[i].update();
    movers[i].display();
  }
};
```

That's a good start, but we can produce a more interesting result by dynamically assigning an angular acceleration according to forces in the environment - since objects don't usually spin of their own accord! Now, we could head pretty far down this road, trying to model the physics of angular acceleration using the concepts of torque and moment of inertia. That level of simulation is beyond the scope of this course - but we'll still get a bit more complex.

For now, a quick and dirty solution will do. We can produce reasonable results by simply calculating angular acceleration as a function of the object's acceleration vector. Here's one such example:

```
this.aAcceleration = this.acceleration.x;
```

Yes, this is completely arbitrary. But it does do something. If the object is accelerating to the right, its angular rotation accelerates in a clockwise direction; acceleration to the left results in a counter clockwise rotation. Of course, it's important to think about scale in this case. The x component of the acceleration vector might be a quantity that's too large, causing the object to spin in a way that looks ridiculous or unrealistic. So dividing the x component by some value, or perhaps constraining the angular velocity to a reasonable range, could really help. Here's the entire `update()` method with these tweaks added.

```
Mover.prototype.update = function () {
    this.velocity.add(this.acceleration);
    this.position.add(this.velocity);

    // Calculate angular acceleration based on horizontal acceleration,
    //  and divide so it's not as strong
    this.aAcceleration = this.acceleration.x / 10.0;
    this.aVelocity += this.aAcceleration;

    // Use constrain to ensure velocity doesn't spin out of control
    this.aVelocity = constrain(this.aVelocity, -0.1, 0.1);
    this.angle += this.aVelocity;

    this.acceleration.mult(0);
};
```

Here's what the program looks like, with those changes incorporated:

```
angleMode = "radians";


var Attractor = function() {
    this.position = new PVector(width/2, height/2);
    this.mass = 20;
    this.G = 1;
};


Attractor.prototype.calculateAttraction = function(m) {
    var force = PVector.sub(this.position, m.position);
    var distance = force.mag();
    distance = constrain(distance, 5, 25);
    force.normalize();
    var strength = (this.G * this.mass * m.mass) / (distance * distance);
    force.mult(strength);
    return force;
};
```

```
Attractor.prototype.display = function() {
   ellipseMode(CENTER);
   strokeWeight(4);
   stroke(0);
   ellipse(this.position.x, this.position.y, this.mass*2, this.mass*2);
};

var Mover = function(m, x, y) {
   this.position = new PVector(x, y);
   this.mass = m;

   this.angle = 0;
   this.aVelocity = 0;
   this.aAcceleration = 0;

   this.velocity = new PVector(random(-1, 1), random(-1, 1));
   this.acceleration = new PVector(0, 0);
};

Mover.prototype.applyForce = function(force) {
   var f = PVector.div(force, this.mass);
   this.acceleration.add(f);
};

Mover.prototype.update = function () {
   this.velocity.add(this.acceleration);
   this.position.add(this.velocity);
```

```
        this.aAcceleration = this.acceleration.x / 10.0;
    this.aVelocity += this.aAcceleration;
    this.aVelocity = constrain(this.aVelocity, -0.1, 0.1);
    this.angle += this.aVelocity;

    this.acceleration.mult(0);
};

Mover.prototype.display = function () {
    stroke(0, 0, 0);
    fill(175, 175, 175, 200);
    rectMode(CENTER);
    pushMatrix();
    translate(this.position.x, this.position.y);
    rotate(this.angle);
    rect(0, 0, this.mass*16, this.mass*16);
    popMatrix();
};

var mover = new Mover(10, 200, 200);

draw = function() {
    background(255, 255, 255);
    mover.update();
    mover.display();
};
```

```
var attractor = new Attractor();

var movers = [];

for (var i = 0; i < 20; i++) {

   movers.push(new Mover(random(0.1, 2), random(width), random(height)));

}


draw = function() {

   background(44, 222, 98);


   attractor.display();


   for (var i = 0; i < movers.length; i++) {

      var force = attractor.calculateAttraction(movers[i]);

      movers[i].applyForce(force);


      movers[i].update();

      movers[i].display();

   }

};
```