Modelling gravity and friction

Now, let's try to make our forces a little bit more true to the real world, by improving the gravity of the last example and introducing a friction force.

Gravity on Earth

You may have noticed something woefully inaccurate about this last example. The smaller the circle, the faster it falls. There is a logic to this; after all, we just stated (according to Newton's second law) that the smaller the mass, the higher the acceleration. But this is not what happens in the real world. If you were to climb to the top of the Leaning Tower of Pisa and drop two balls of different masses, which one will hit the ground first? According to legend, Galileo performed this exact test in 1589, discovering that they fell with the same acceleration, hitting the ground at the same time. Why is this? As we will see later in this course, the force of gravity is calculated relative to an object's mass. The bigger the object, the stronger the force. So if the force is scaled according to mass, it is cancelled out when acceleration is divided by mass. We can implement this in our sketch rather easily by multiplying our made-up gravity force by mass.

```
for (var i = 0; i < movers.length; i++) {
    var wind = new PVector(0.01, 0);
    var gravity = new PVector(0, 0.1 * movers[i].mass);
    movers[i].applyForce(wind);
    movers[i].applyForce(gravity);
    movers[i].update();
    movers[i].display();
```

```
    movers[i].checkEdges();
}
```

While the objects now fall at the same rate, because the strength of the wind force is independent of mass, the smaller objects still accelerate to the right more quickly.

```
var Mover = function(m, x, y) {
  this.mass = m;
  this.position = new PVector(x, y);
  this.velocity = new PVector(0, 0);
  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.acceleration.mult(0);
};

Mover.prototype.display = function() {
 stroke(0);
 strokeWeight(2);
```

```
  fill(255, 255, 255, 127);
  ellipse(this.position.x, this.position.y, this.mass*16, this.mass*16);
};

Mover.prototype.checkEdges = function() {
  if (this.position.x > width) {
    this.position.x = width;
    this.velocity.x *= -1;
  } else if (this.position.x < 0) {
    this.velocity.x *= -1;
    this.position.x = 0;
  }
  if (this.position.y > height) {
    this.velocity.y *= -1;
    this.position.y = height;
  }
};

var movers = [];

for (var i = 0; i < 20; i++) {
    movers[i] = new Mover(random(0.1, 5), 0, 0);
}

var draw = function() {
  background(50, 50, 50);
```

```
  for (var i = 0; i < movers.length; i++) {
    var wind = new PVector(0.01, 0);
    var gravity = new PVector(0, 0.1*movers[i].mass);
    movers[i].applyForce(wind);
    movers[i].applyForce(gravity);
    movers[i].update();
    movers[i].display();
    movers[i].checkEdges();
  }
};
```

Making up forces will actually get us quite far. The world of ProcessingJS is a pretend world of pixels and you are its master. So whatever you deem appropriate to be a force, well by golly, that's the force it should be. Nevertheless, there may come a time where you find yourself wondering:

"But how does it really all work?"

Open up any high school physics textbook and you will find some diagrams and formulas describing many different forces—gravity, electromagnetism, friction, tension, elasticity, and more. Or, hey, browse the physics lessons on Khan Academy. In this section, we're going to look at two of those forces—friction and gravity. The point we're making here is not that friction and gravity are fundamental forces that you always need to have in all your ProcessingJS programs. Rather, we want to evaluate these two forces as case studies for the following process:

- Understanding the concept behind a force
- Deconstructing the force's formula into two parts:
- How do we compute the force's direction?
- How do we compute the force's magnitude?

- Translating that formula into ProcessingJS code that calculates a `PVector` to be sent through our `Mover`'s `applyForce()` function.

  If we can follow the above steps with two forces, then hopefully if you ever find yourself Googling “atomic nuclei weak nuclear force” at 3 a.m., you will have the skills to take what you find and adapt it for ProcessingJS programs.