Pointing towards movement

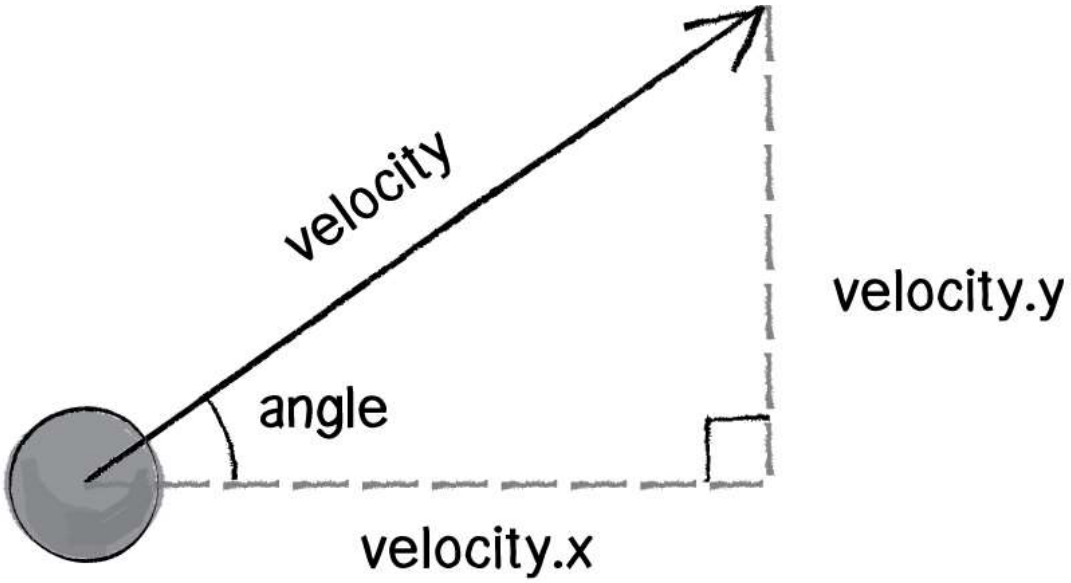
Let’s go all the way back to one of our first examples, the one where a `Mover` object accelerates towards the mouse.

```
var Mover = function() {
   this.position = new PVector(width/2, height/2);
   this.velocity = new PVector(0, 0);
   this.acceleration = new PVector(0, 0);
};

Mover.prototype.update = function() {
   var mouse = new PVector(mouseX, mouseY);
   var dir = PVector.sub(mouse, this.position);
   dir.normalize();
   dir.mult(0.5);
   this.acceleration = dir;
   this.velocity.add(this.acceleration);
   this.velocity.limit(5);
   this.position.add(this.velocity);
};

Mover.prototype.display = function() {
   stroke(0);
   strokeWeight(2);
   fill(127);
   ellipse(this.position.x, this.position.y, 48, 48);
};
```

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

  if (this.position.x > width) {
    this.position.x = 0;
  } else if (this.position.x < 0) {
    this.position.x = width;
  }

  if (this.position.y > height) {
    this.position.y = 0;
  } else if (this.position.y < 0) {
    this.position.y = height;
  }
};

var mover = new Mover();

var draw = function() {
  background(255, 255, 255);

  mover.update();
  mover.checkEdges();
  mover.display();
};
```

You might notice that almost all of the shapes we've been drawing so far are circles. This is convenient for a number of reasons, one of which is that

we don’t have to consider the question of rotation. Rotate a circle and, well, it looks exactly the same. However, there comes a time in all motion programmers’ lives when they want to draw something on the screen that points in the direction of movement. Perhaps you are drawing an ant, or a car, or a spaceship. And when we say "point in the direction of movement," what we are really saying is “rotate according to the velocity vector.” Velocity is a vector, with an x and a y component, but to rotate in ProcessingJS we need an angle. Let’s draw our trigonometry diagram one more time, with an object’s velocity vector:



tangent(angle) = velocity.y/velocity.x

OK. We know that the definition of tangent is:

tangent(angle)=\frac{velocity_y}{velocity_x}*tangent*(*angle*)=*velocity**x**velocity**y*t, a, n, g, e, n, t, left parenthesis, a, n, g, l, e, right parenthesis, equals, start fraction, v, e, l, o, c, i, t, y, start subscript, y, end subscript, divided by, v, e, l, o, c, i, t, y, start subscript, x, end subscript, end fraction

The problem with the above is that we know velocity, but we don’t know the angle. We have to solve for the angle. This is where a special function known as inverse tangent comes in, sometimes referred to as arctangent or tan-1. (There is also an inverse sine and an inverse cosine.)

If the tangent of some value a equals some value b, then the inverse tangent of b equals a. For example:

| if | tangent(a) = b | | then | a = arctangent(b) |

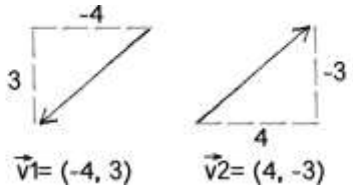
See how that is the inverse? The above now allows us to solve for the angle:

| if | tangent(angle) = velocity_y / velocity_x | | then | angle = arctangent(velocity_y / velocity_x) |

Now that we have the formula, let’s see where it should go in our mover’s display() function. Notice that in ProcessingJS, the function for arctangent is called `atan()`. JavaScript also provides `Math.atan()` natively (as well as all the basic trig functions), but we'll stick with the ProcessingJS provided functions.

```
Mover.prototype.display = function () {
  var angle = atan(this.velocity.y / this.velocity.x);


  stroke(0, 0, 0);
  fill(127, 127, 127);
  pushMatrix();
  rectMode(CENTER);
  translate(this.position.x, this.position.y);
  rotate(angle);
  rect(0, 0, 30, 10);
  popMatrix();
};
```

Now the above code is pretty darn close, and almost works. We still have a big problem, though. Let's consider the two velocity vectors depicted below.



Though superficially similar, the two vectors point in quite different directions—opposite directions, in fact! However, if we were to apply our formula to solve for the angle to each vector…

V1 ⇒ angle = atan(-4/3) = atan(-1.333...) = -0.9272952 radians = -53 degrees
V2 ⇒ angle = atan(4/-3) = atan(-1.333...) = -0.9272952 radians = -53 degrees

…we get the same angle for each vector. This can't be right for both; the vectors point in opposite directions! The thing is, this is a pretty common problem in computer graphics. Rather than simply using `atan()` along with a bunch of conditional statements to account for positive/negative scenarios, ProcessingJS (along with JavaScript and pretty much all programming environments) has a nice function called `atan2()` that does it for you.

```
Mover.prototype.display = function () {
  var angle = atan2(this.velocity.y, this.velocity.x);

  stroke(0, 0, 0);
  fill(127, 127, 127);
  pushMatrix();
  rectMode(CENTER);
  translate(this.position.x, this.position.y);
```

```
  rotate(angle);
  rect(0, 0, 30, 10);
  popMatrix();
};
```

To simplify this even further, the `PVector` object itself provides a function called `heading()`, which takes care of calling `atan2()` for you so you can get the 2D direction angle, in radians, for any `PVector`.

Here's what the program looks like, all together. Move your mouse over it and see how it rotates!

```
var Mover = function() {
  this.position = new PVector(width/2, height/2);
  this.velocity = new PVector(0, 0);
  this.acceleration = 0;
  this.topspeed = 4;
  this.xoff = 1000;
  this.yoff = 0;
  this.r = 16;
};


Mover.prototype.update = function () {
  var mouse = new PVector(mouseX, mouseY);
  var dir = PVector.sub(mouse, this.position);
  dir.normalize();
  dir.mult(0.5);
  this.acceleration = dir;


  this.velocity.add(this.acceleration);
```

```
    this.velocity.limit(this.topspeed);
    this.position.add(this.velocity);
};

Mover.prototype.display = function () {
    var angle = this.velocity.heading();

    stroke(0, 0, 0);
    strokeWeight(2);
    fill(127, 127, 127);
    pushMatrix();
    rectMode(CENTER);
    translate(this.position.x, this.position.y);
    rotate(angle);
    rect(0, 0, 30, 10);
    popMatrix();
};

Mover.prototype.checkEdges = function () {
    if (this.position.x > width) {
        this.position.x = 0;
    } else if (this.position.x < 0) {
        this.position.x = width;
    }

    if (this.position.y > height) {
        this.position.y = 0;
```

```
    } else if (this.position.y < 0) {
        this.position.y = height;
    }
};

var mover = new Mover();

var draw = function() {
    background(169, 230, 232);
    mover.update();
    mover.checkEdges();
    mover.display();
};
```