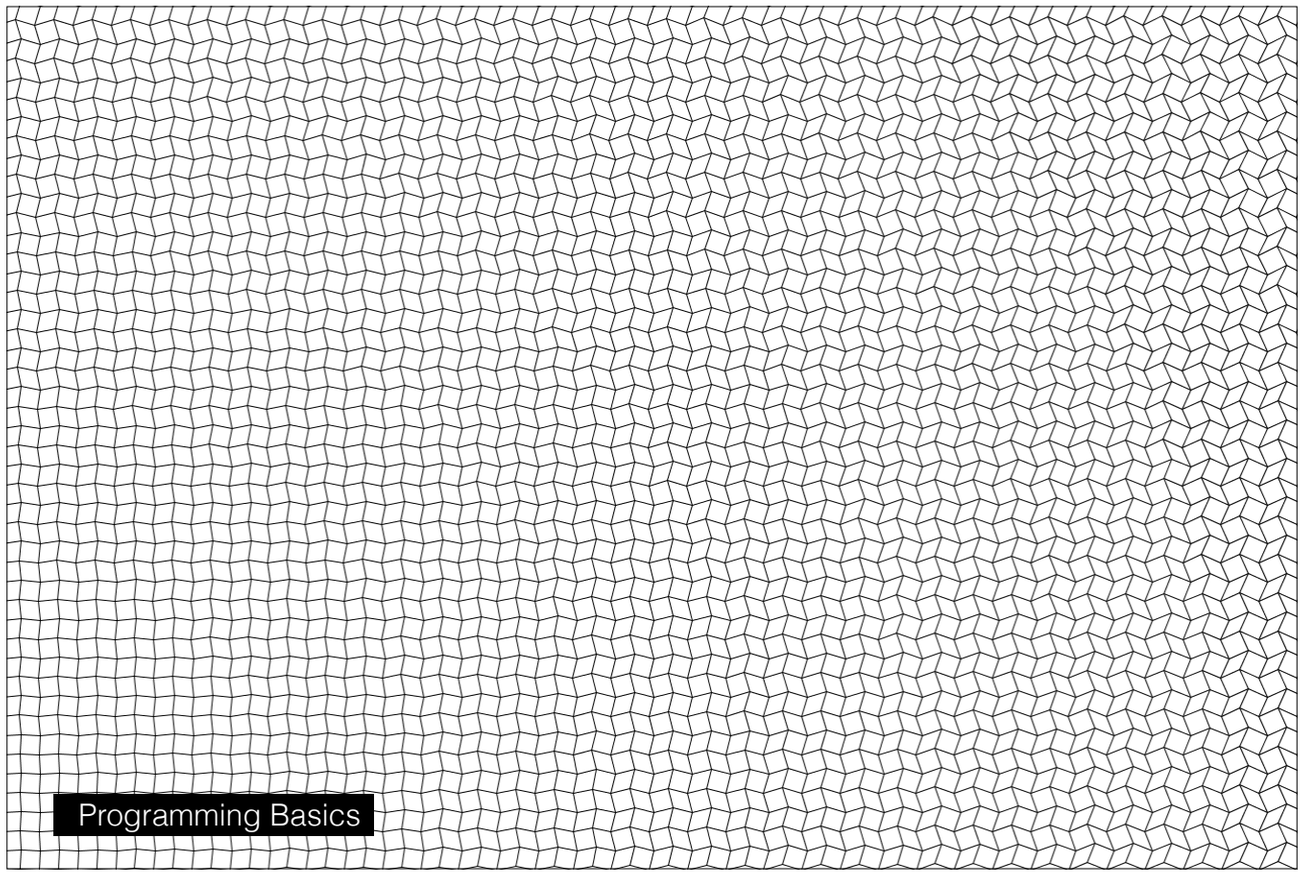
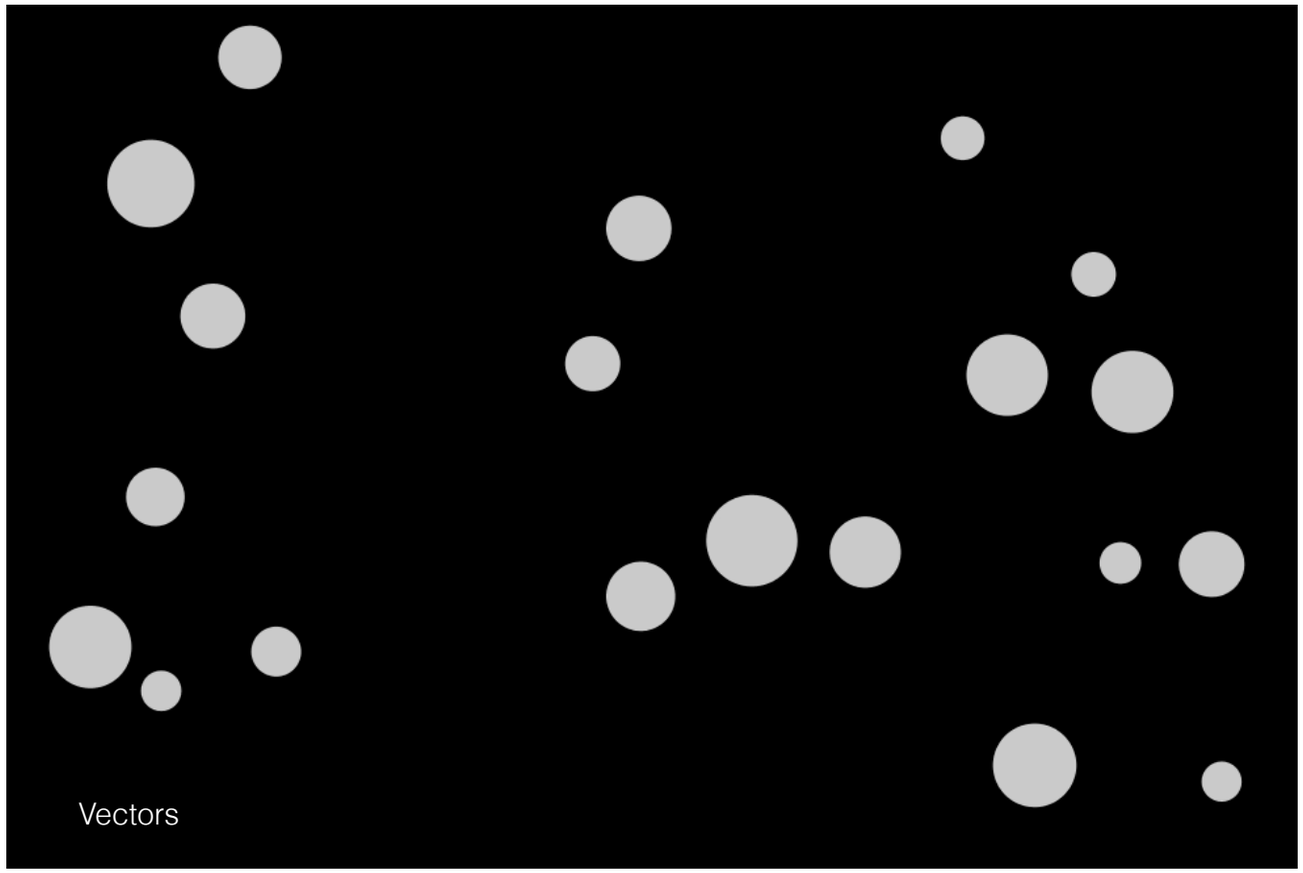
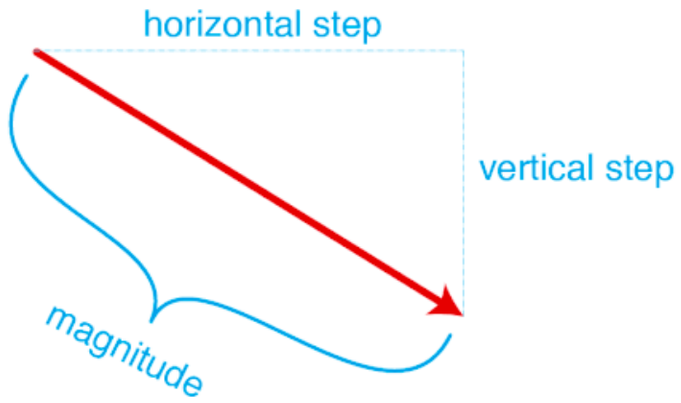

Programming Basics


Vectors



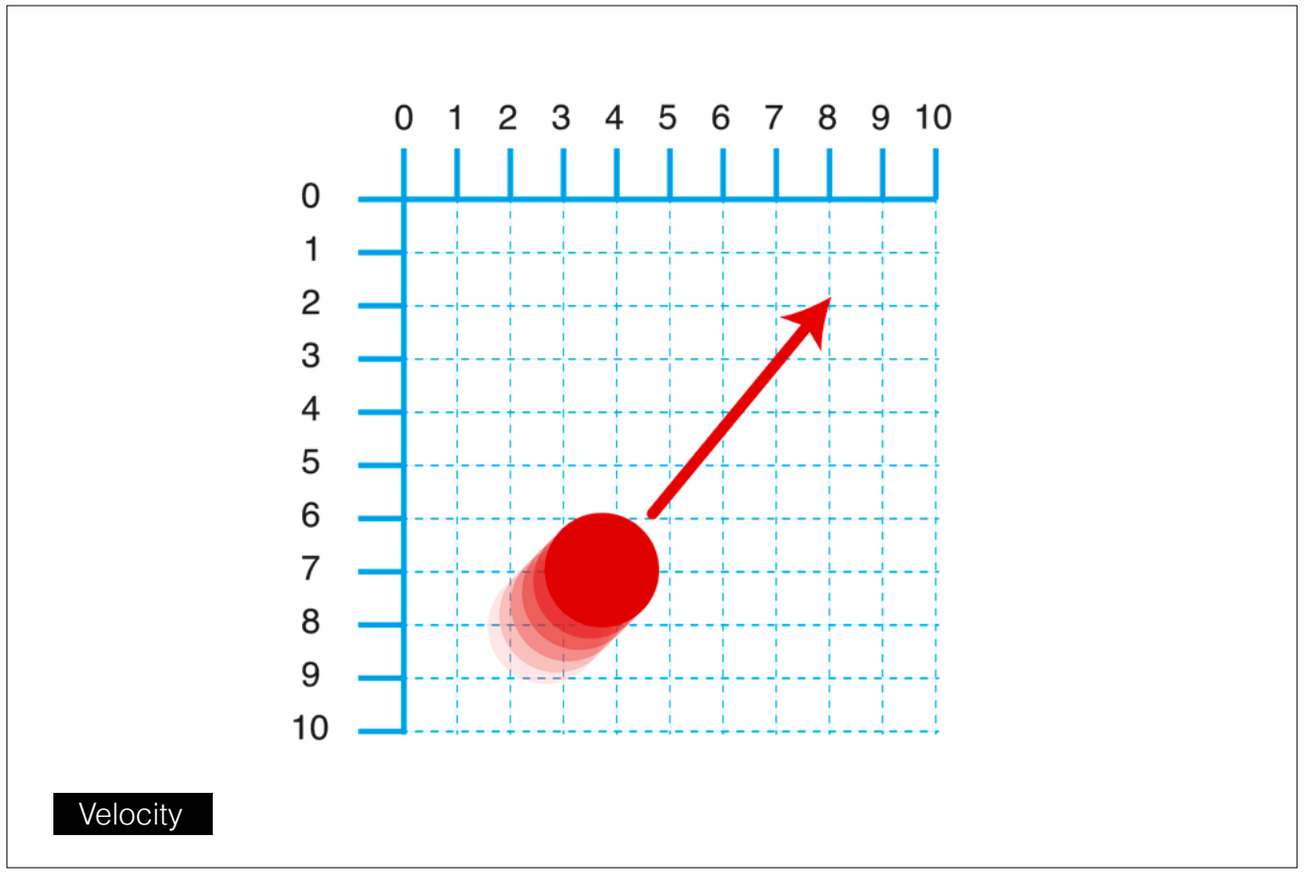
`(x, y)`

`Vector in 2D space`

Vectors

Vectors are enormously useful geometric object in maths and physics. Vectors express a direction and a magnitude (or length), often pictured as an arrow in 2D or 3D space.

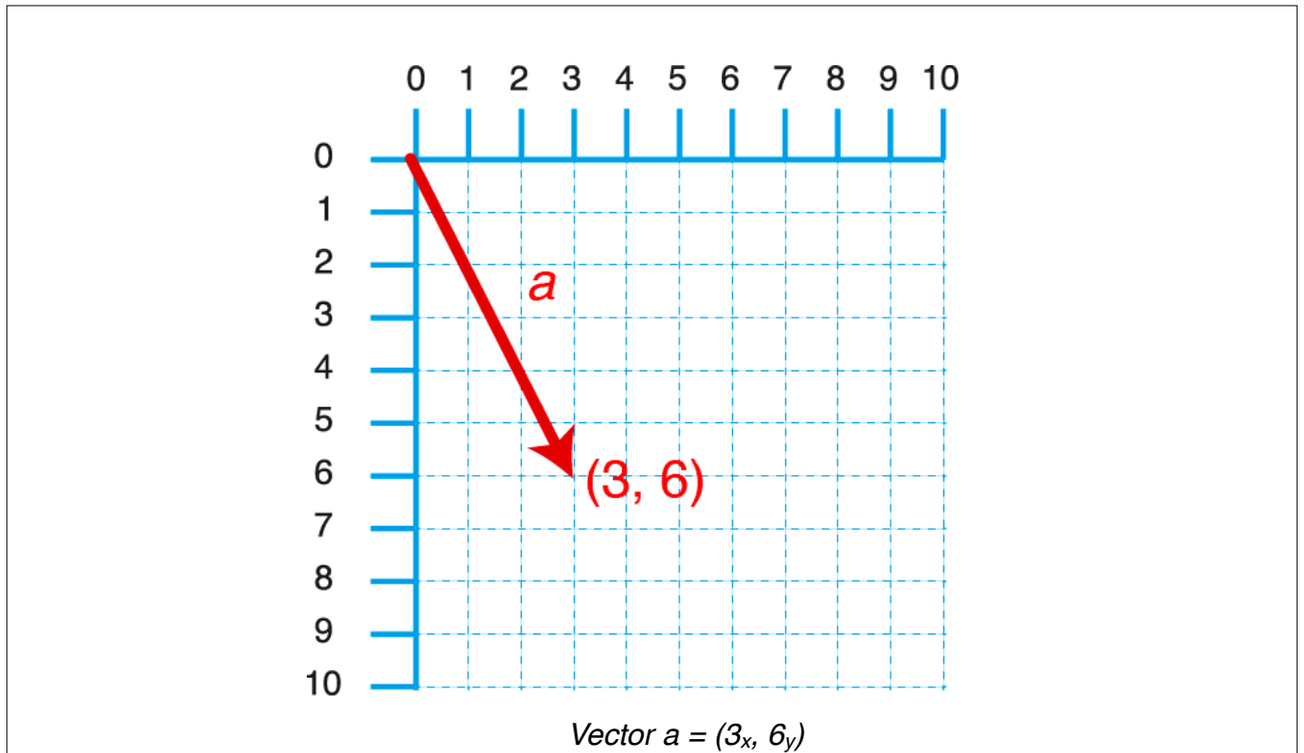
Vectors are key concept not only for producing geometry through code, but also for motion and interactive animation. One of the main uses of vectors are to express velocity, which is both speed and direction of travel.


0 1 2 3 4 5 6 7 8 9 10
0
1
2
3
4
5
6
7
8
9
10
Velocity

(x, y)
Vector in 2D space

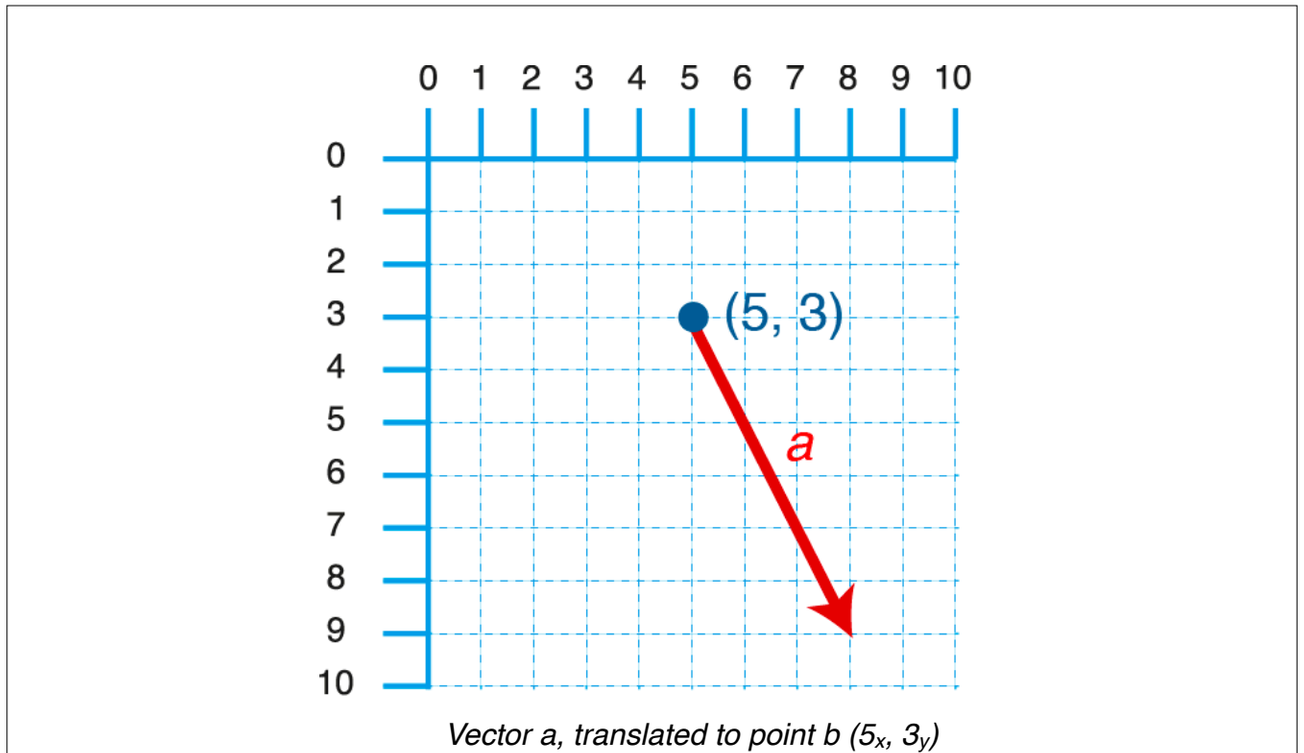
(x, y)
Point in 2D space

Points vs Vectors



Vector A

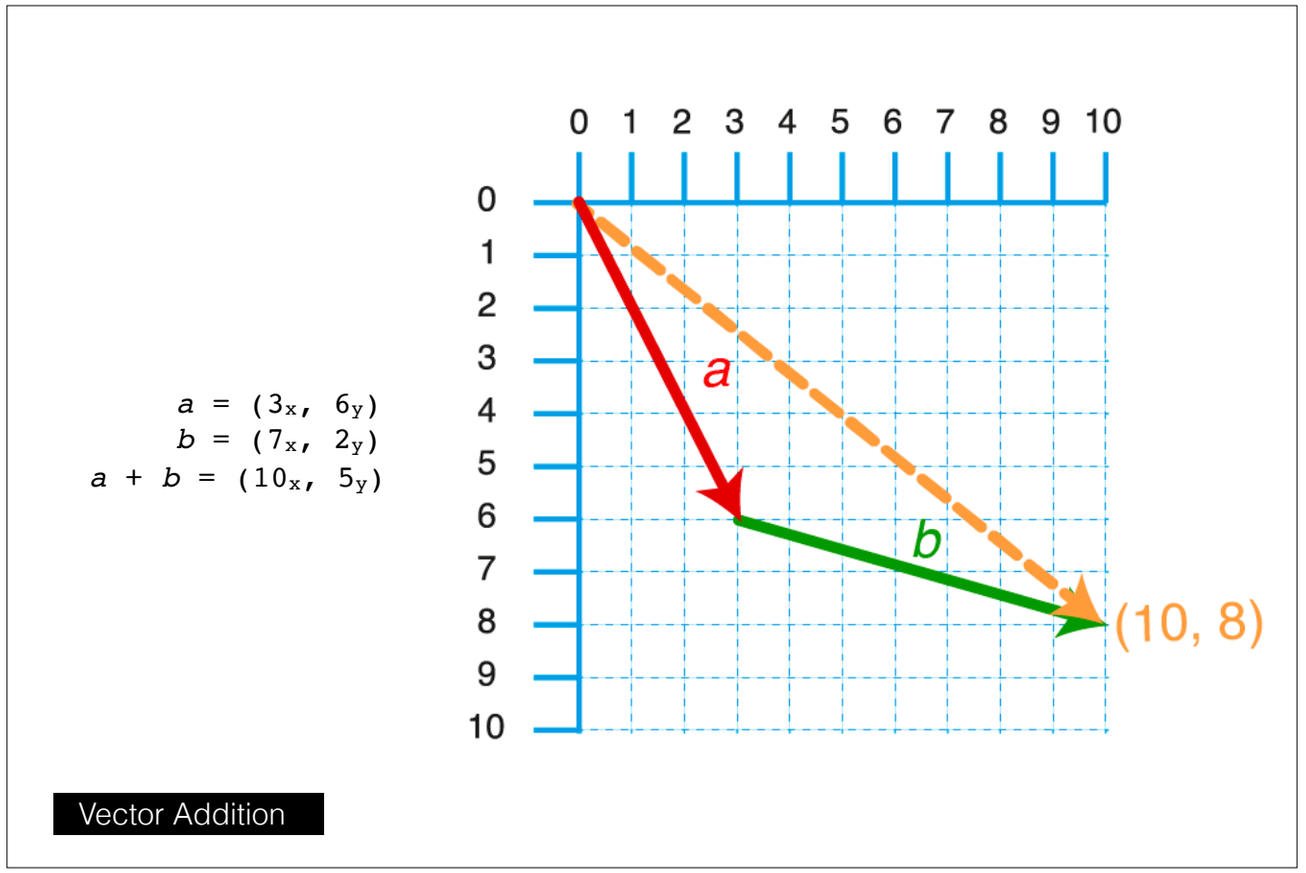
The key thing to remember is that vectors represent direction and magnitude without a location. For this reason vectors are often combined with a coordinate. 2D and 3D coordinates are ofter recorded in the same format as vectors, with just two or three numbers.



Vector a, translated to point b (5_x, 3_y)

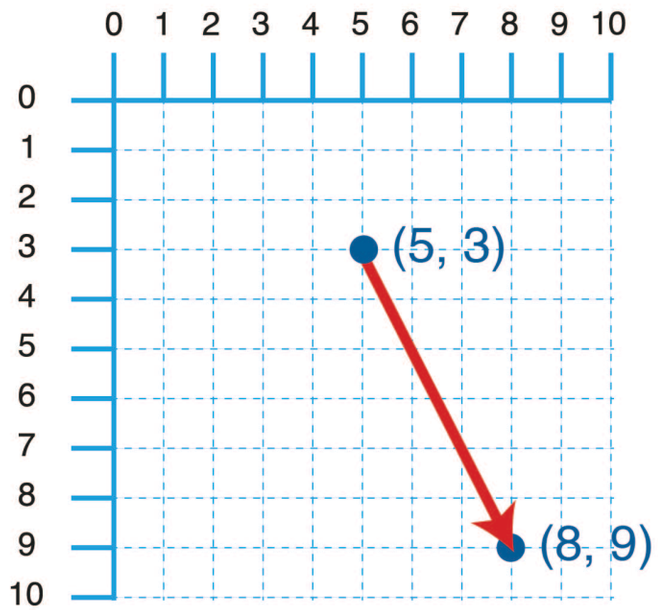
Vector A Translated

However, coordinates are normally visually represented as points rather than arrows.



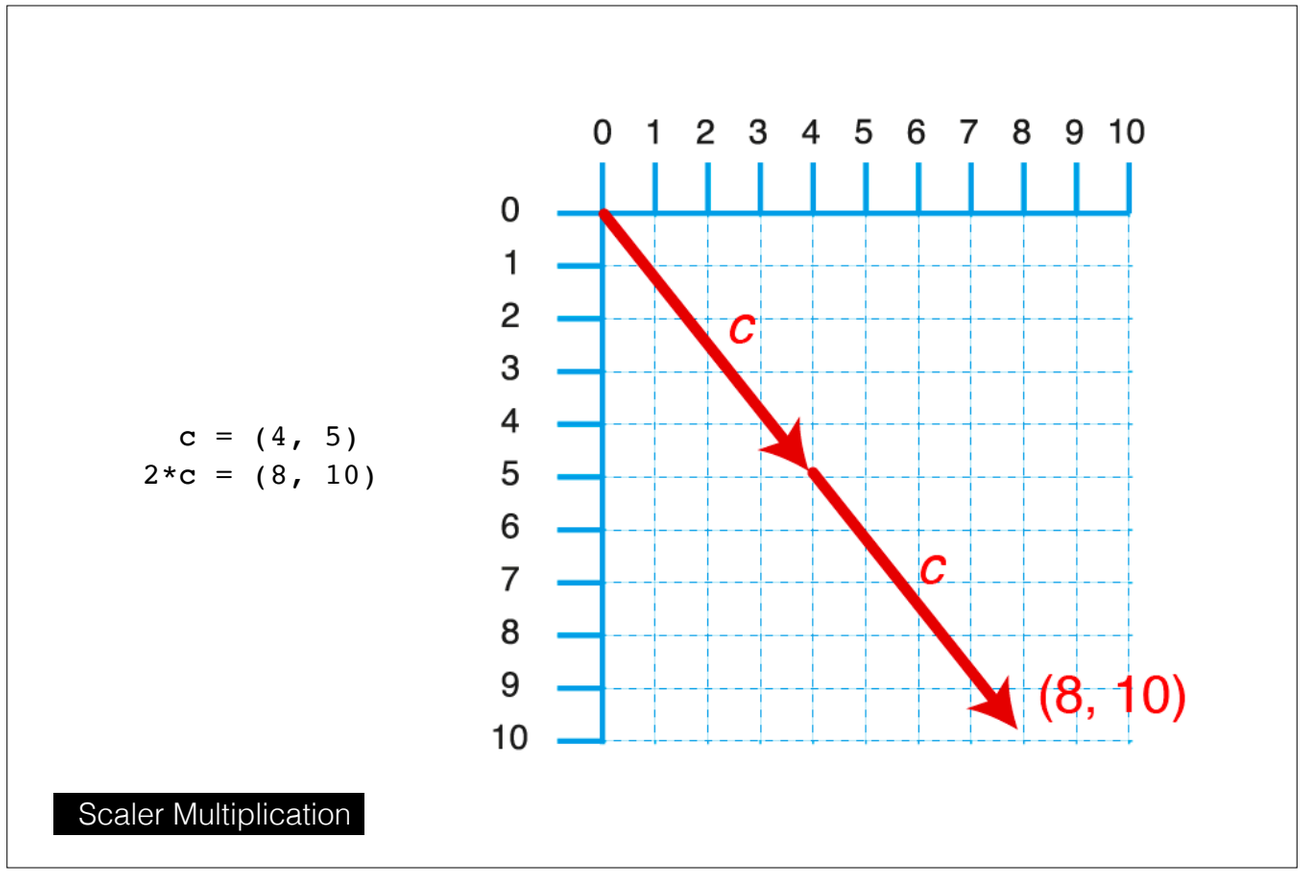
If you add to vectors together, the resulting vector is the equivalent of stacking the vectors end to end. Adding vectors together can be used to simulate simple physics like the effect of wind or gravity, so for every step forward a game character might have a wind vector added to their movement vector. If a character in a game jumps while moving forward, then we also want to add the vectors of the upward movement with the forward motion, and then in reverse when they fall back down (or by adding a gravity vector which points down).

$$a = (5_x, 3_y)$$
$$b = (8_x, 9_y)$$
$$b - a = (3_x, 6_y)$$

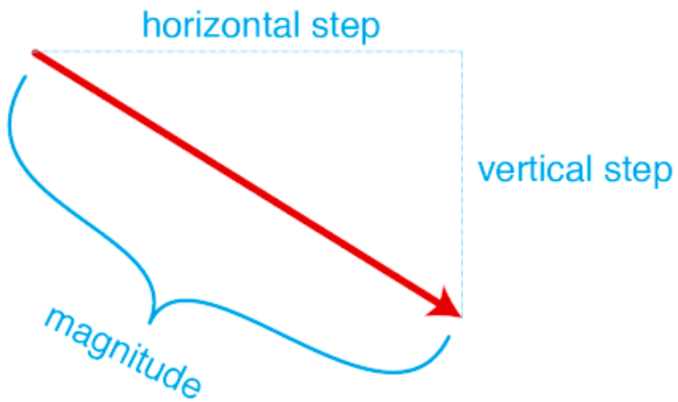


Subtraction

If you add to vectors together, the resulting vector is the equivalent of stacking the vectors end to end. Adding vectors together can be used to simulate simple physics like the effect of wind or gravity, so for every step forward a game character might have a wind vector added to their movement vector. If a character in a game jumps while moving forward, then we also want to add the vectors of the upward movement with the forward motion, and then in reverse when they fall back down (or by adding a gravity vector which points down).



Scaler Multiplication only effects the magnitude of the vector, leaving the direction unchanged. However, if the vector is multiplied by a negative, then the direction is reversed! Scaler Multiplication can be used to control the acceleration of space ship or to simulate wind resistance or drag. If an object in a game collides with wall for example, we could multiply the objects vector by -1, to reverse it's direction so that it bounces off the surface.



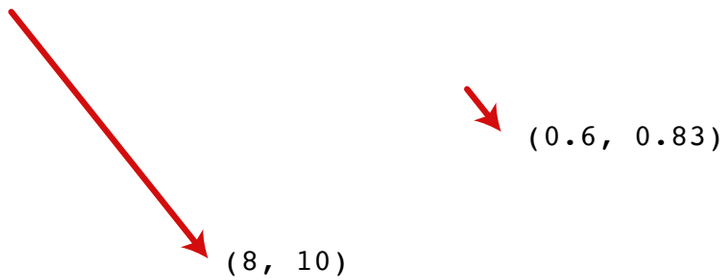

```
c = (4, 4)

Magnitude of c =  √4² + 4²


The Pvector class in processing
has a method that returns this value: mag()
```

Finding the Magnitude

You can find the magnitude (the length) using the Pythagorean theorem. In a car game, you might need to find this value show the speed of an object on the HUD display.



Normalising a vector:
1. calculate the magnitude
2. Divide each vector components by the magnitude

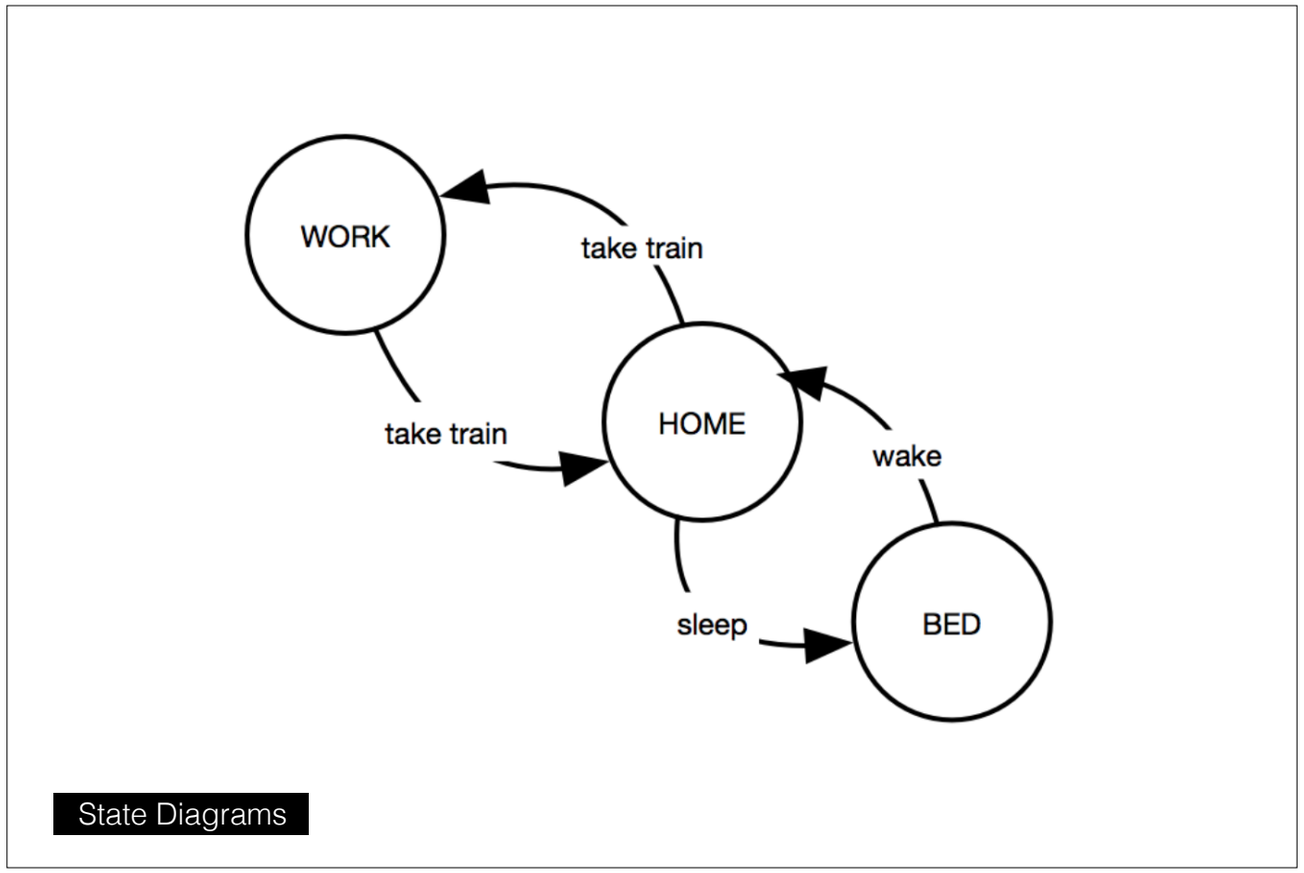
or

Use the normalize() function of the Pvector class

Normalising A Vector

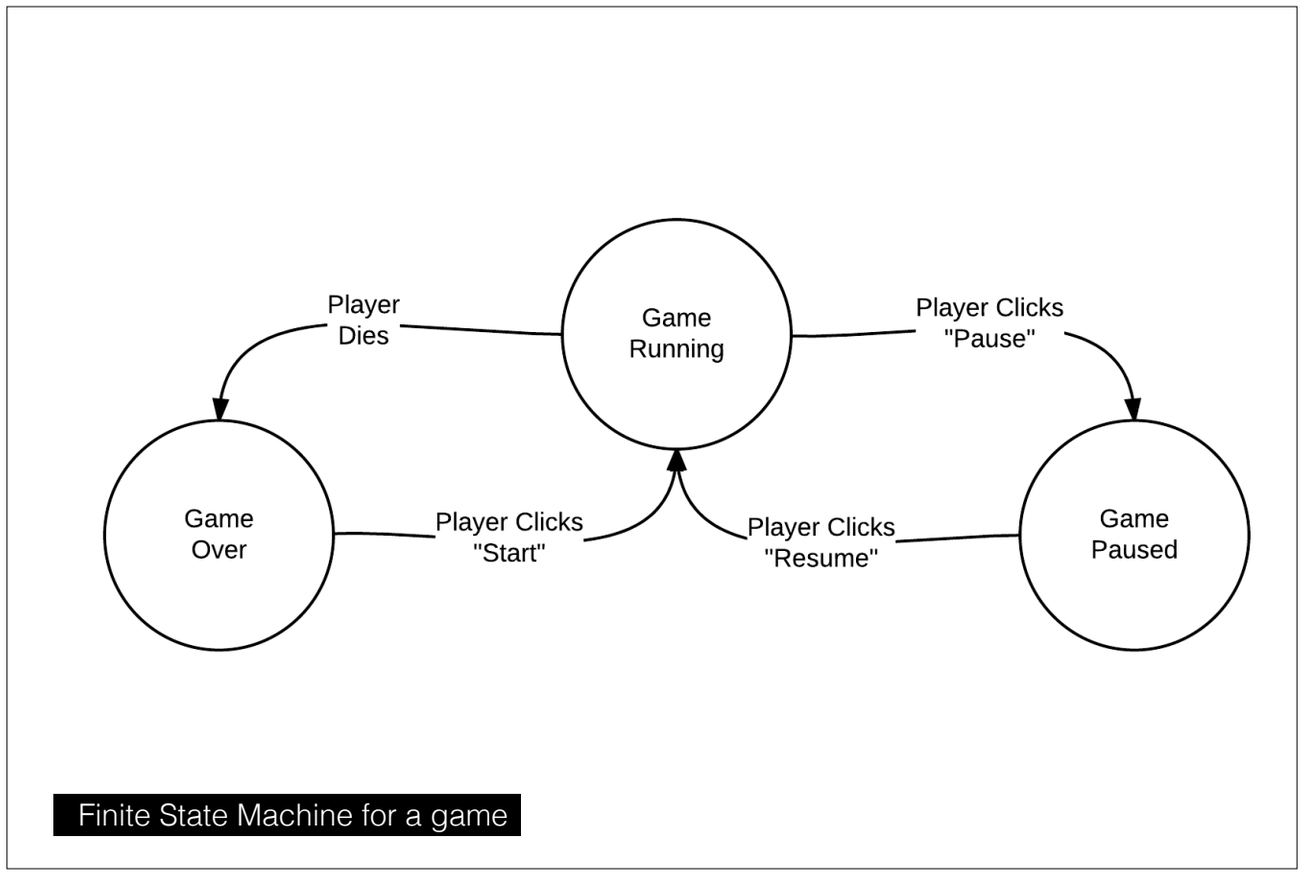
Normalising

A normalised vector or a unit vector has had its magnitude set to one, with the direction left unchanged. A unit vector shows us a direction alone, without a magnitude. Sometimes we are only interested in direction, and removing magnitude can make many calculations much simpler.



A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction.

Arrows indicate transitions
Circle indicate states



Finite State Machine for a game

```
enum State {
  GAME,
  MENU,
  QUIT
}
```

Enumerator

```
String state = "Start";
void draw() {
  if (state.equals("Start")) {
    // do something
  }
  // more states
}
```

States with Strings