Simulating Natural Phenomena

Lec.2

Vector Processing

High Diploma 2nd Course

2023-2024

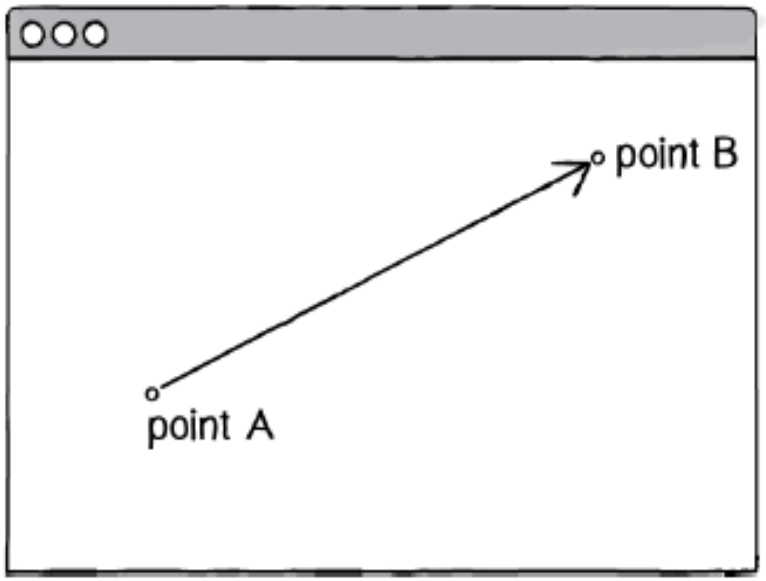
Nyan Dawood

Outlines

- **Vector Processing**
- **Bouncing ball with no vector**
- **Vector Addition**
- **Vector Subtraction**
- **Bouncing ball with vector**
- **Vector multiplication , Division or (scaling)**
- **Magnitude & Normalization**
- **Vector motion**

Vector Processing

A vector (drawn as an arrow) has magnitude (length of arrow) and direction (which way it is pointing).

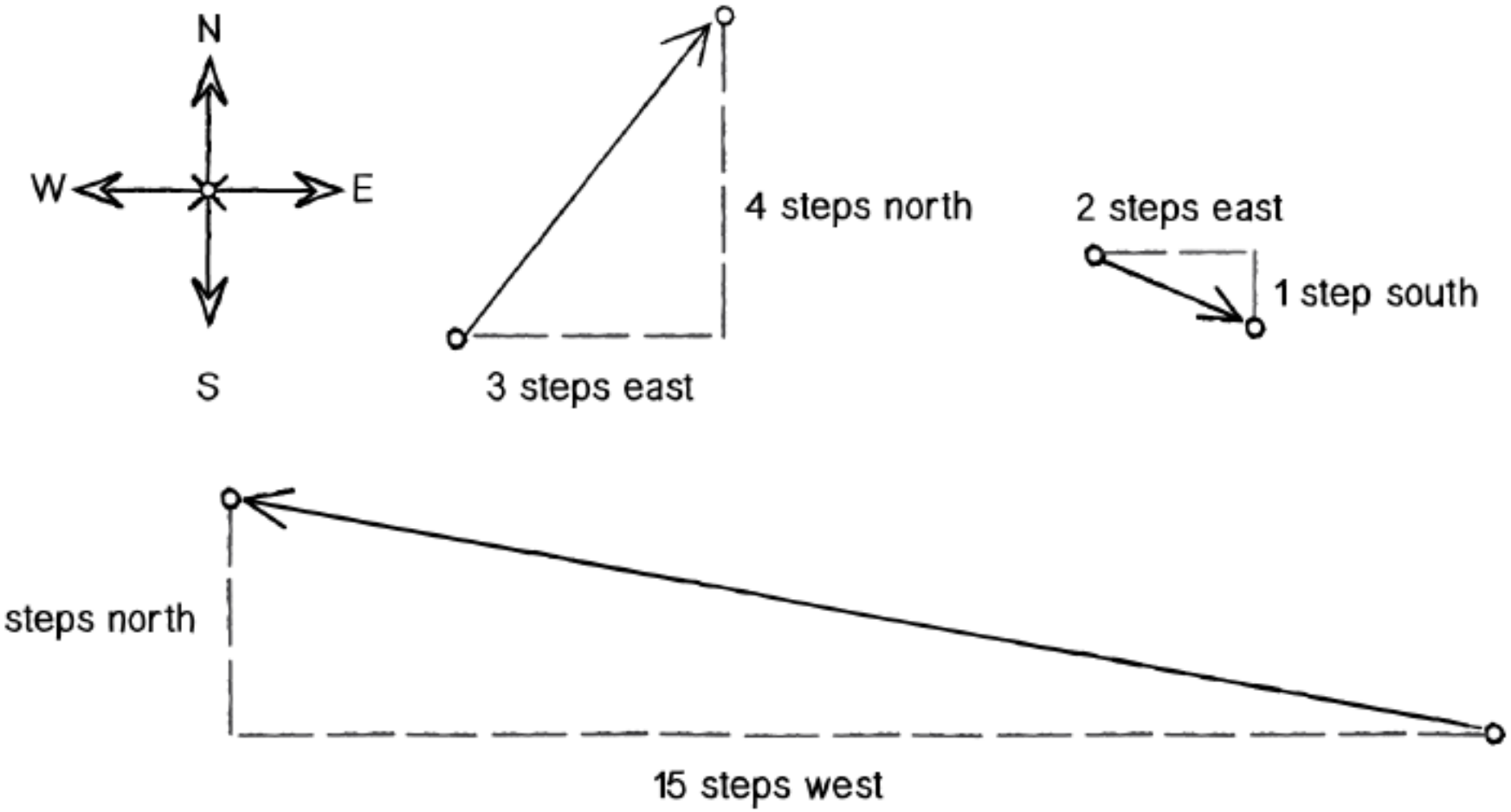


Bouncing ball with no vector

```
float x = 100;       //Variables for location and speed of ball.
float y = 100;
float xspeed = 1;
float yspeed = 3.3;
void setup() {
    size(200,200);
    smooth();
    background(255);}
 void draw() {
    background(255);
    x = x + xspeed;    //Move the ball according to its speed.
    y = y + yspeed;
```

```
if ((x > width) || (x < 0)) {                    //Check for bouncing.
xspeed = xspeed * -1;
}
```

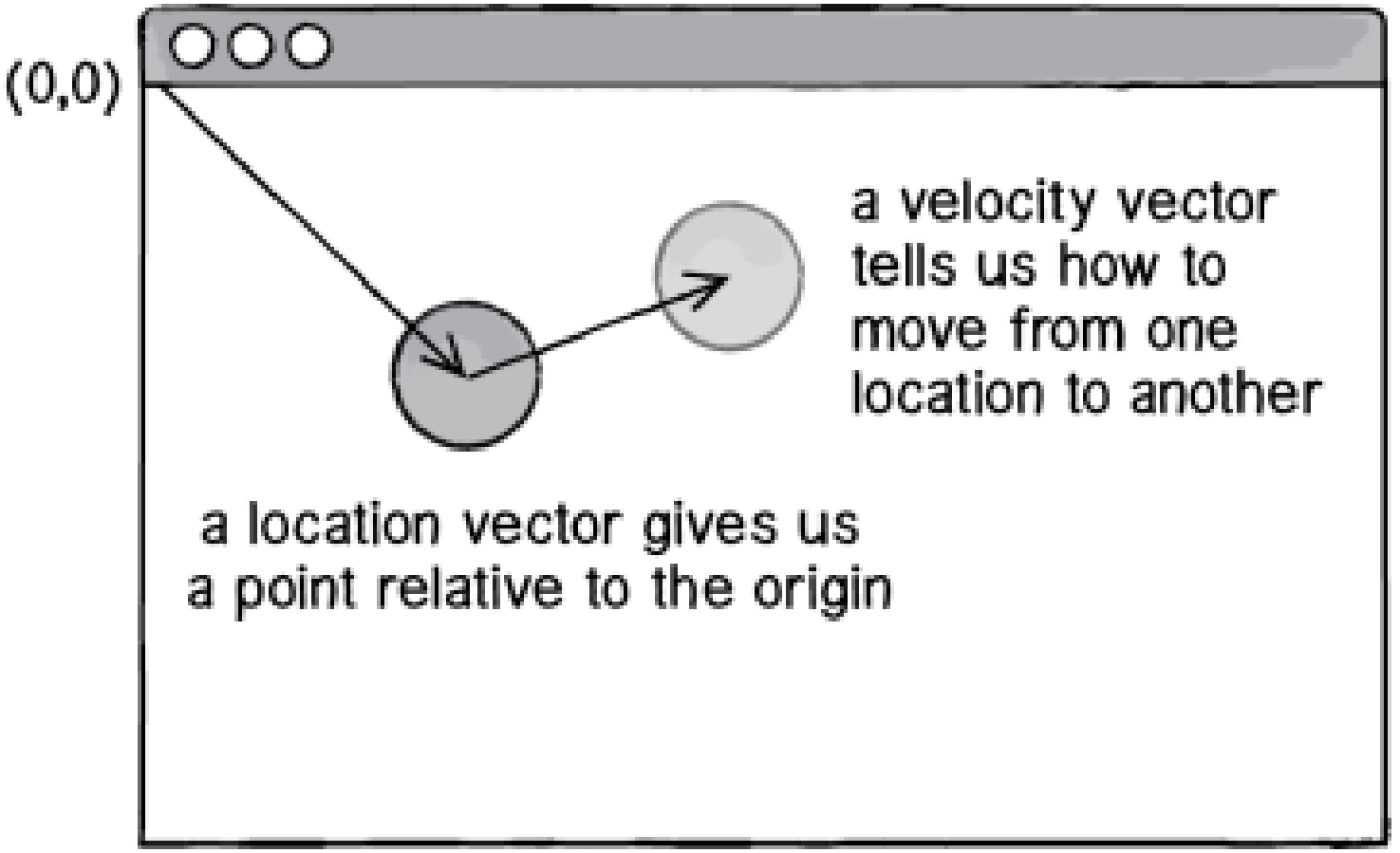
```
if ((y > height) || (y < 0)) {
      yspeed = yspeed * -1;
}
stroke(0);
fill(175);
ellipse(x,y,16,16);                    //Display the ball at the location (x,y).
}
```



(-15, 3)	*Walk fifteen steps west; turn and walk three steps north.*
(3, 4)	*Walk three steps east; turn and walk five steps north.*
(2, -1)	*Walk two steps east; turn and walk one step south.*

Vector Implementation

new location = velocity applied to current location



Instead of:

float x;

float y;

float xspeed;

float yspeed;

We use vector definition :

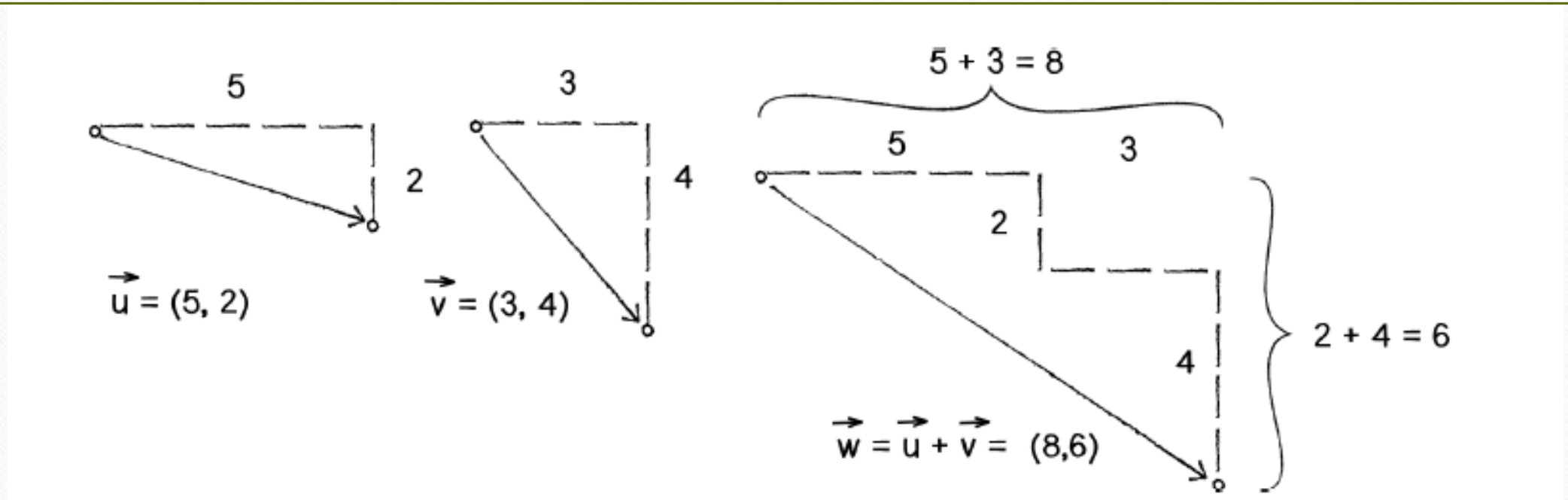
Vector location;

Vector speed;

```
class PVector {
      float x;
      float y;
      PVector(float x_, float y_) {
            x = x_;
            y = y_;
             }
}
```

Vector Addition

Each vector has two components, an x and a y. To add two vectors together, we simply add both x's and both y's.



```
class PVector {

  float x;
  float y;

  PVector(float x_, float y_) {
    x = x_;
    y = y_;
  }

  void add(PVector v) {
    y = y + v.y;
    x = x + v.x;
  }
}
```

New! A function to add another PVector to this PVector. Simply add the *x* components and the *y* components together.

Bouncing ball with vector

```
PVector location;
PVector velocity;
```

Instead of a bunch of floats, we now just have two PVector variables.

```
void setup() {
  size(200,200);
  smooth();
  location = new PVector(100,100);
  velocity = new PVector(2.5,5);
}

void draw() {
  background(255);

  location.add(velocity);
```

```
  if ((location.x > width) || (location.x < 0)) {
    velocity.x = velocity.x * -1;
  }
  if ((location.y > height) || (location.y < 0)) {
    velocity.y = velocity.y * -1;
  }

  stroke(0);
  fill(175);
  ellipse(location.x,location.y,16,16);
}
```

We still sometimes need to refer to the individual components of a PVector and can do so using the dot syntax: location.x, velocity.y, etc.

Mathematical function commonly used with Pvector

- add() — add vectors
- sub() — subtract vectors
- mult() — scale the vector with multiplication
- div() — scale the vector with division
- mag() — calculate the magnitude of a vector
- setMag() - set the magnitude of a vector

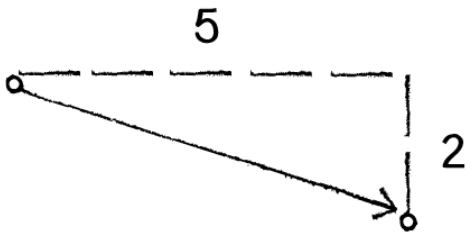
…..

Vector Subtraction

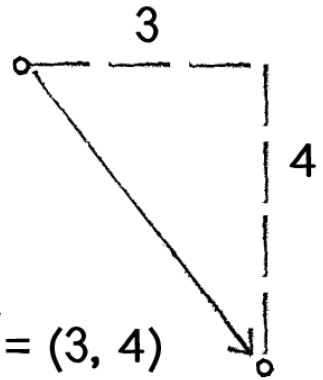
$$\vec{w} = \vec{u} - \vec{v}$$

can be written as:

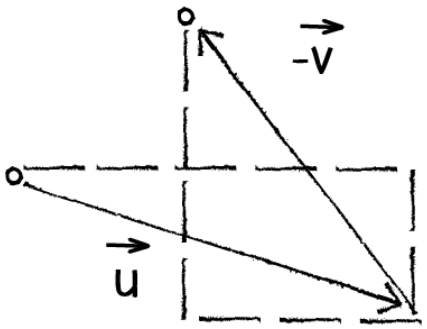
$$w_x = u_x - v_x$$
$$w_y = u_y - v_y$$



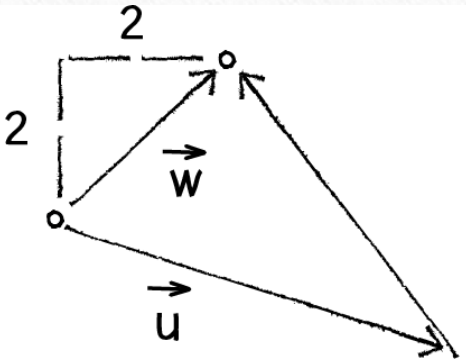
$\vec{u} = (5, 2)$



$\vec{v} = (3, 4)$



$\vec{w} = \vec{u} - \vec{v} = (2, 2)$



Vector Subtraction

And the function inside PVector look like :

```
void sub(PVector v) {
x = x - v.x;
y = y - v.y;
}
```

Vector Subtraction processing

```
PVector location;
PVector center;


void setup(){
 size(400,400);
 }


void draw(){
background(255);
location=new PVector(50,100);
center=new PVector(width/2,height/2);
location.sub(center);
translate(width/2,height/2);
line(0,0,location.x,location.y);
}
```

Vector multiplication –or scaling

$\vec{w} = \vec{u} * n$

can be written as:

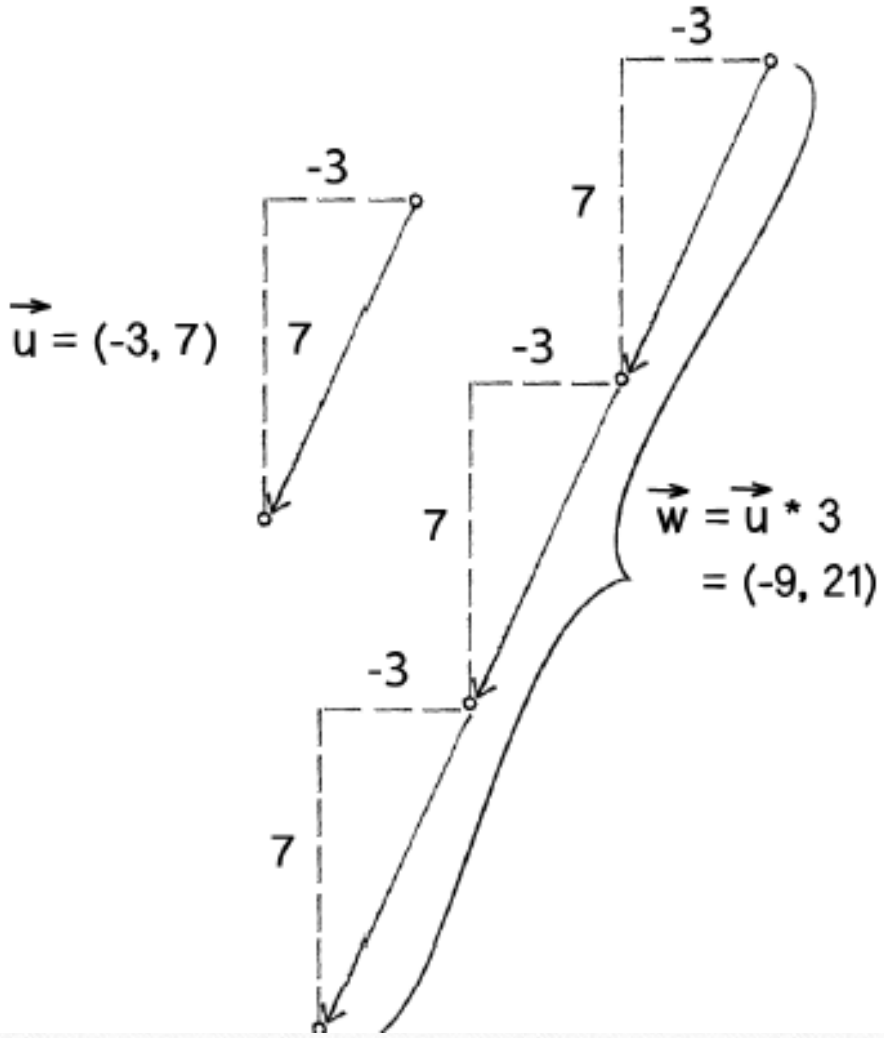
$w_x = u_x * n$
$w_y = u_y * n$

Let's look at an example with vector notation.

$\vec{u} = (-3, 7)$
$n = 3$

$\vec{w} = \vec{u} * n$
$w_x = -3 * 3$
$w_y = 7 * 3$

$\vec{w} = (-9, 21)$



```
void mult(float n) {
x = x * n;                    //vector components multiplied by n
y = y * n;
}
```

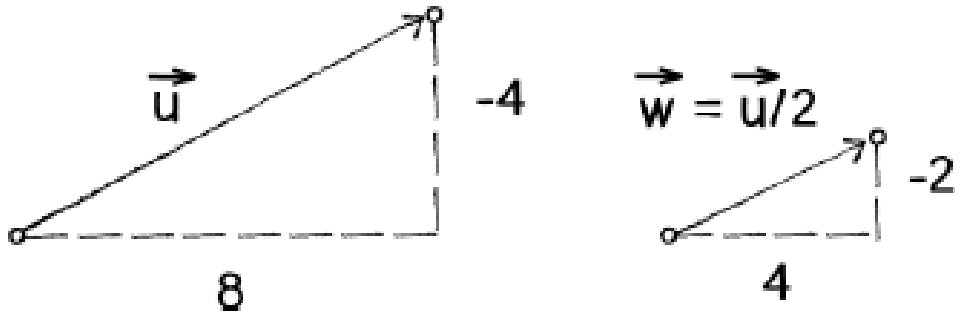
```
PVector u = new PVector(-3,7);
u.mult(3);                    // the vector will be (-9,21)
```

```
void setup() {
    size(200,200);
    smooth();
}
void draw() {
    background(255);
    PVector mouse = new PVector(mouseX,mouseY);
    PVector center = new PVector(width/2,height/2);
    mouse.sub(center);
    mouse.mult(0.5);                    //The vector is now half its original size

    translate(width/2,height/2);
    line(0,0,mouse.x,mouse.y);
}
```

Vector division

Division works just like multiplication—we simply replace the multiplication sign (asterisk) with the division sign (forward slash).




```
void div(float n) {
x = x / n;
y = y / n;
}
PVector u = new PVector(8,-4);
u.div(2);                                   // u vector will be (4,-2)
```

Vector Magnitude

The Pythagorean theorem is a squared plus b squared equals c squared.

$$\| \vec{v} \| = \sqrt{v_x * v_x + v_y * v_y}$$

```
float mag() {
return sqrt(x*x + y*y);
}
```

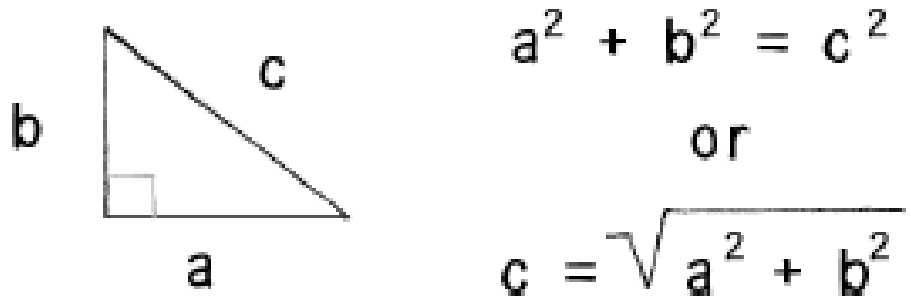


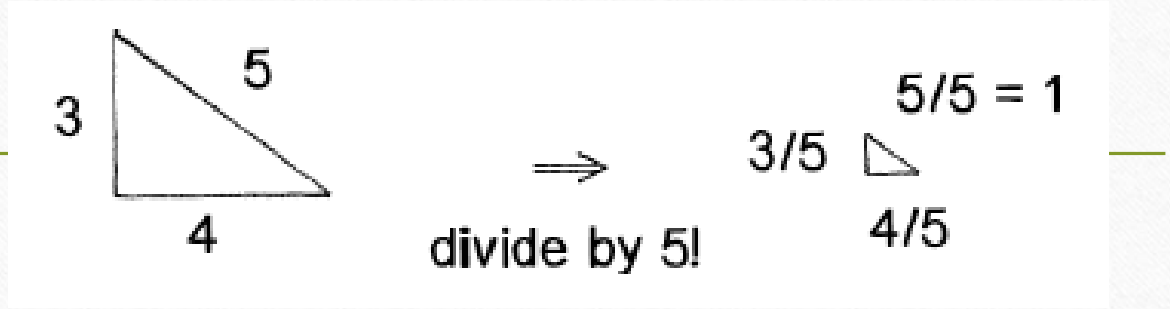
Figure 1.11: The Pythagorean Theorem

```
void setup() {
        size(200,200);
        smooth();
}
void draw() {
        background(255);
        PVector mouse = new PVector(mouseX,mouseY);
        PVector center = new PVector(width/2,height/2);
        mouse.sub(center);
                        //The magnitude (i.e. length) of a vector can be accessed via the mag() func.
                       // the width of a rectangle drawn at the top of the window.
float m = mouse.mag();
fill(0);
rect(0,0,m,10);
translate(width/2,height/2);
line(0,0,mouse.x,mouse.y);
}
```

Normalizing Vectors

Normalizing refers to the process of making something "standard".

$$\hat{u} = \frac{\vec{u}}{\| \vec{u} \|}$$




```
void normalize() {
float m = mag();
div(m);
}
```

Note : We can't divide by 0! Some quick error checking will fix that right up:

```
void normalize() {
float m = mag();
if (m != 0) {
div(m);}    }
```

```
void draw() {
    background(255);
    PVector mouse = new PVector(mouseX,mouseY);
    PVector center = new PVector(width/2,height/2);
    mouse.sub(center);
      //normalized vector multiplied by 50 so that it is viewable onscreen.
    mouse.normalize();
    mouse.mult(50);
    translate(width/2,height/2);
    line(0,0,mouse.x,mouse.y);
}
```

Vector Motion: Velocity

```
class Mover {
    PVector location;
    PVector velocity;
Mover() {                                   // constructor to initialize vectors values
    location = new PVector(random(width),random(height));
    velocity = new PVector(random(-2,2),random(-2,2));   }
void update() {                             //motion update
    location.add(velocity); }
void display() {                            //motion view
    stroke(0);
    fill(175);
    ellipse(location.x,location.y,16,16);
                }
}
```

```
void checkEdges() {
                                    //When it reaches one edge, set location to the other.
      if (location.x > width) {
      location.x = 0;}
       else if (location.x < 0) {
      location.x = width;}


      if (location.y > height) {
      location.y = 0;
      } else if (location.y < 0) {
      location.y = height;
}
                              // in main program
Mover mover;                          //object declaration
mover = new Mover();                  // in setup function  , object initialization
mover.update();                       // in draw function call class methods
mover.checkEdges();
mover.display();
```

Vector Motion: Acceleration

Acceleration Algorithms!

1. A constant acceleration

---

2. A totally random acceleration
3. Acceleration towards the mouse

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

coding : Constant Acceleration

```
class Mover {
      PVector location;
      PVector velocity;
      PVector acceleration;            //Acceleration is the key!
                                       //The variable top speed will limit the magnitude of velocity.
          float topspeed;
Mover() {
      location = new PVector(width/2,height/2);
      velocity = new PVector(0,0);
      acceleration = new PVector(-0.001,0.01);
      topspeed = 10;
}
void update() {
                                  //Velocity changes by acceleration and is limited by topspeed.
      velocity.add(acceleration);
      velocity.limit(topspeed);
```

```
location.add(velocity);
}
void display() {
                                    // same display() in velocity motion          }
void checkEdges() {       // same checkEdges()in velocity motion }
}
```

Question://

Write the limit() function for the PVector class.?

```
void limit(float max) {
if (_____ > _____) {
_____();
____(max);
}
}
```

Exercise :

Create a simulation of a car (or runner) that accelerates when you press the up key and brakes when you press the down key.

Program: Random acceleration

```
Class Mover{
  PVector location;
  PVector velocity;
  PVector acceleration;
  float topspeed;
  Mover() {
      location = new PVector(width/2, height/2);
      velocity = new PVector(0, 0);
      topspeed = 6;
  }
```

```
void update() {
    acceleration = PVector.random2D();
    acceleration.mult(random(2));
    velocity.add(acceleration);
    velocity.limit(topspeed);
    location.add(velocity);  }
  }
  void display() {
        //   the same }
  void checkEdges() {
       // the same  }
}
```