プログラミング応用

シミュレーション

山辺真幸（masaki@allianceport.jp）

PVectorオブジェクト

PVectorオブジェクト

Processingにあらかじめ用意されているオブジェクト
ベクトルを扱うためのもの

PVectorクラス

情報（データ）

x, y, z

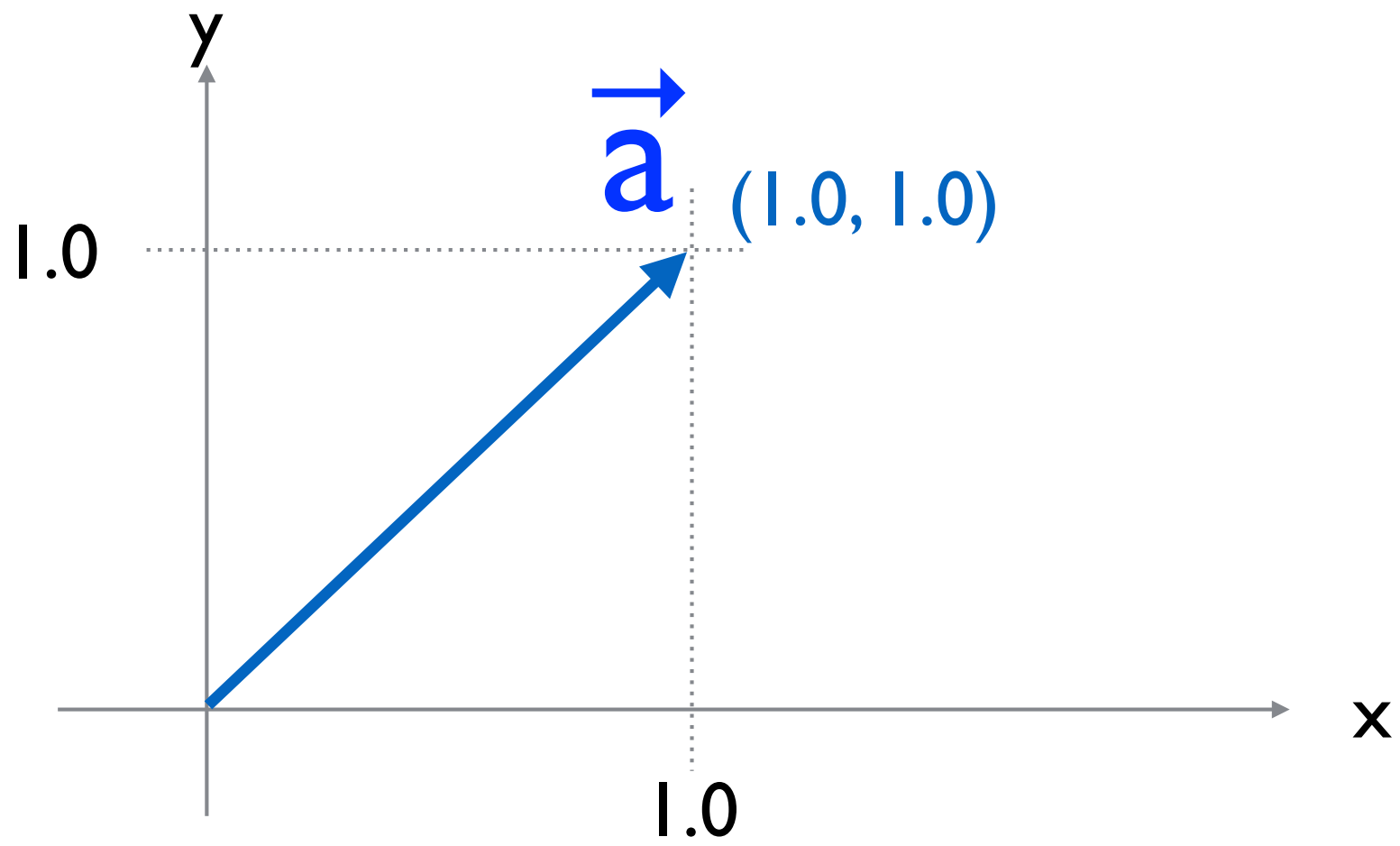
機能（関数）

add()	ベクトル同士を足す
sub()	ベクトル同士を引く
mul()	ベクトルにスカラーをかける（ベクトルの長さを変える）
normalize()	長さ１のベクトルにする
mag()	長さを返す

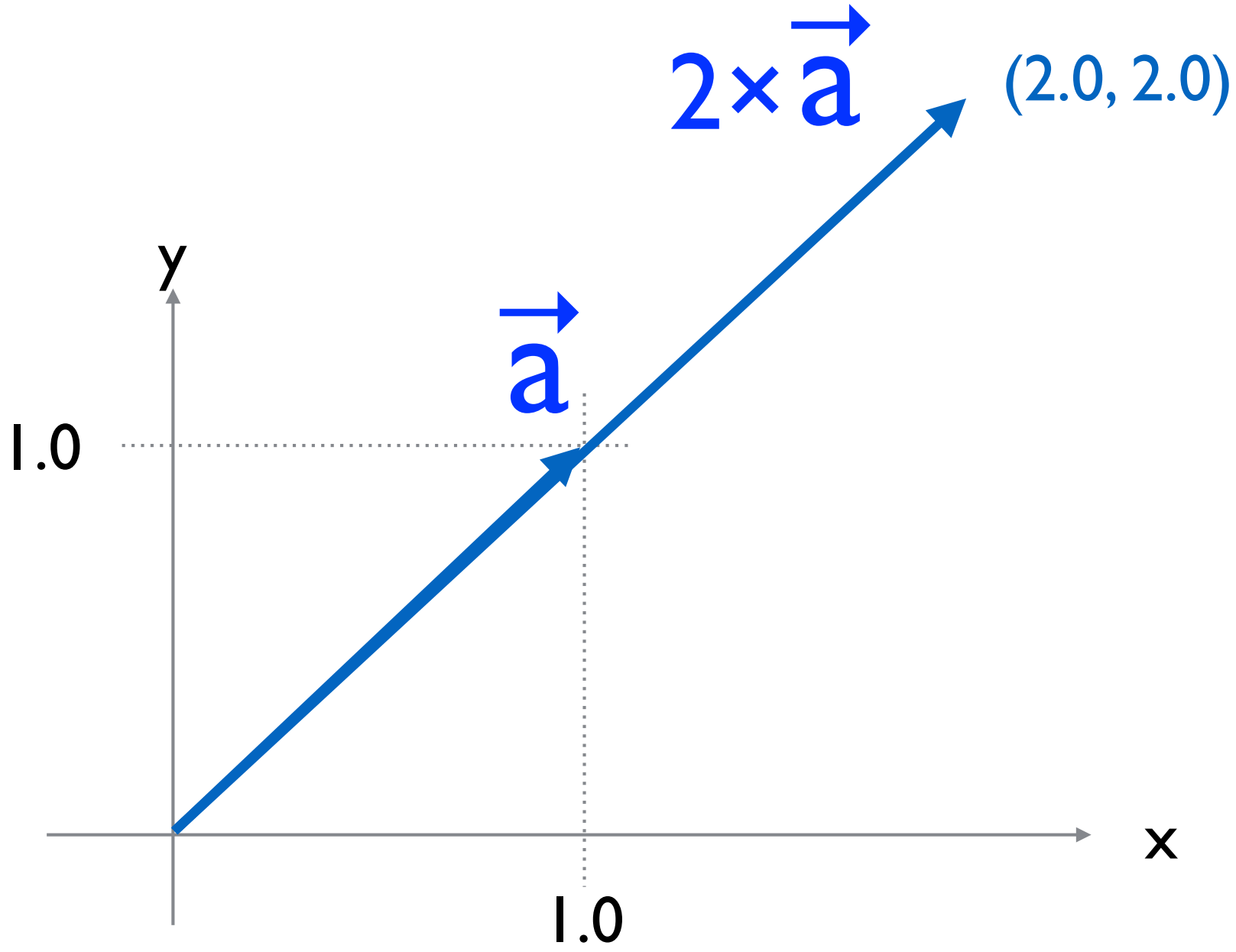
他にもあります。
https://processing.org/reference/PVector.html

そもそもベクトルとは

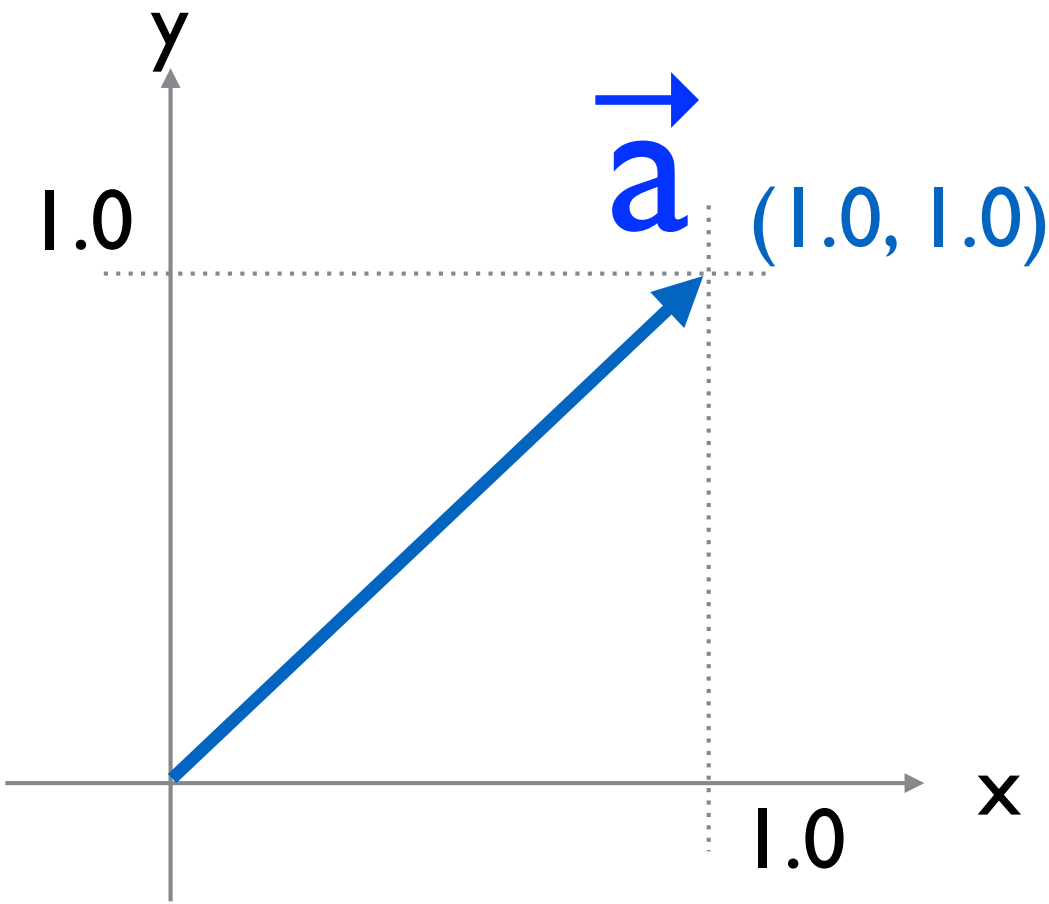
ベクトル　＝　２つの量と大きさを持つもの

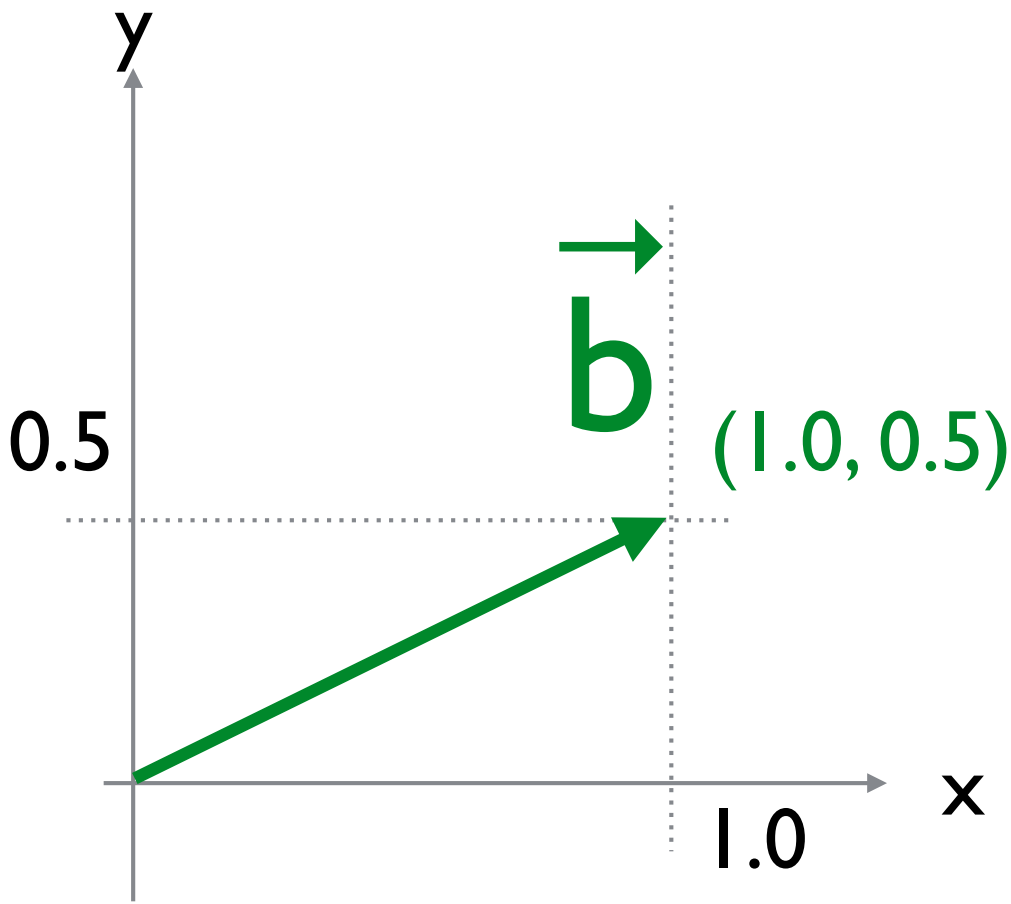


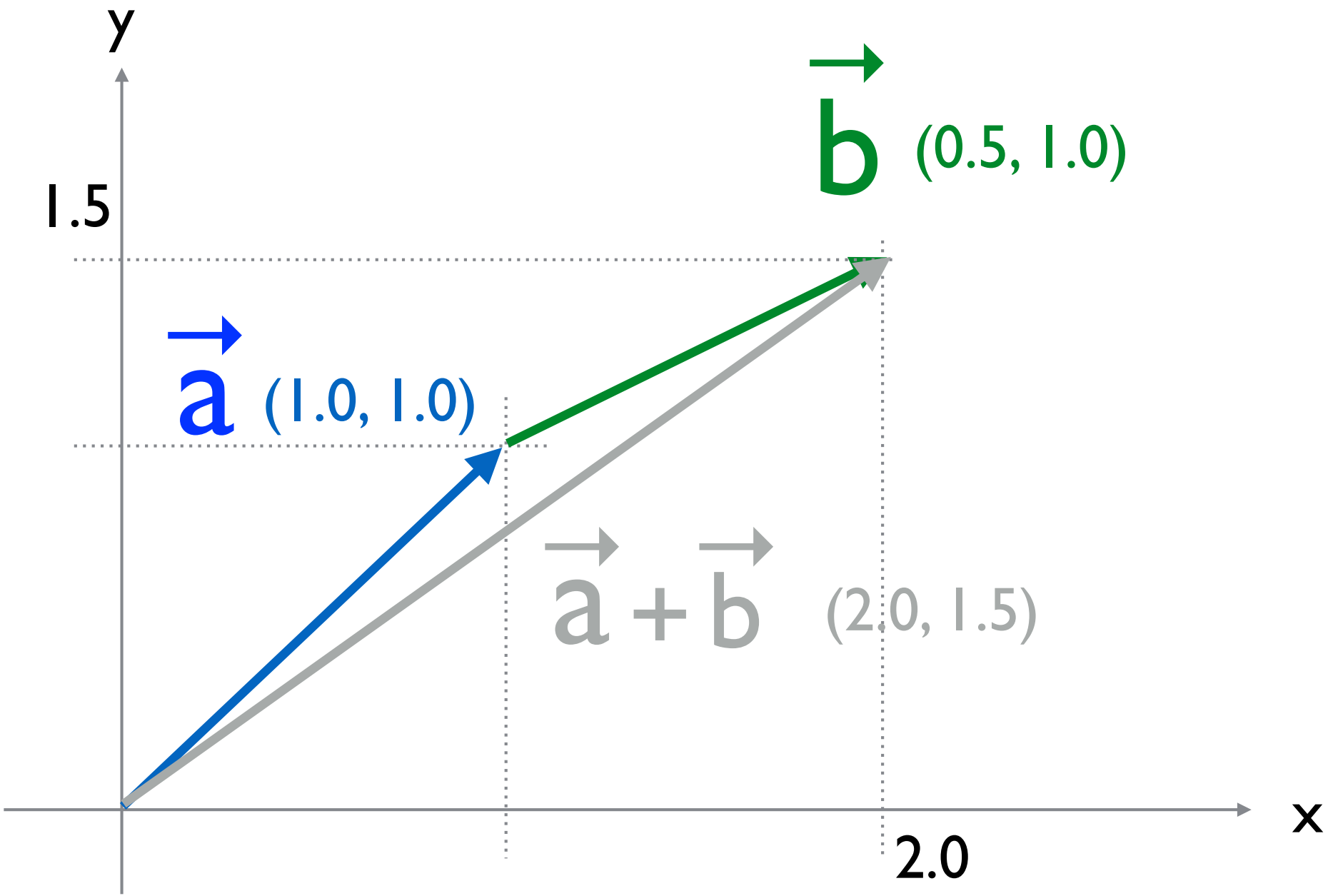
$\vec{a}$ の大きさ（長さ）＝ $\sqrt{2}$



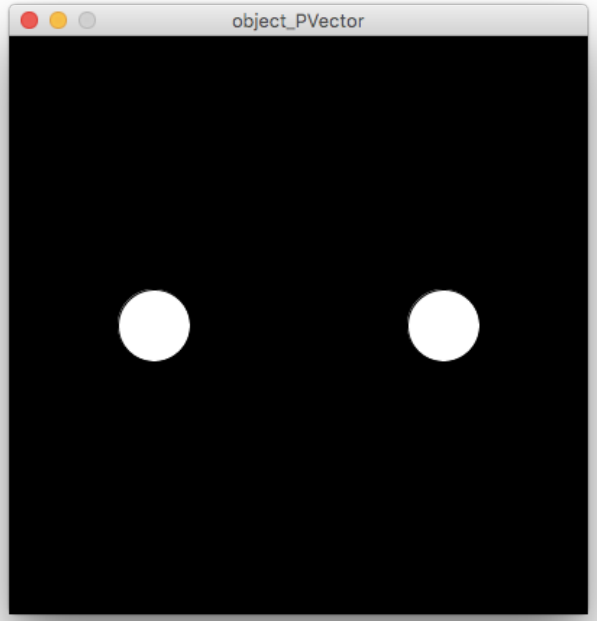
$2\times\vec{a}$ の大きさ（長さ）＝ $\sqrt{2}$


y
1.0
$\vec{a}$
(1.0, 1.0)
1.0
x


y
0.5
$\vec{b}$
(1.0, 0.5)
1.0
x


y
1.5
$\vec{b}$ (0.5, 1.0)
$\vec{a}$ (1.0, 1.0)
$\vec{a} + \vec{b}$ (2.0, 1.5)
x
2.0

PVectorオブジェクト



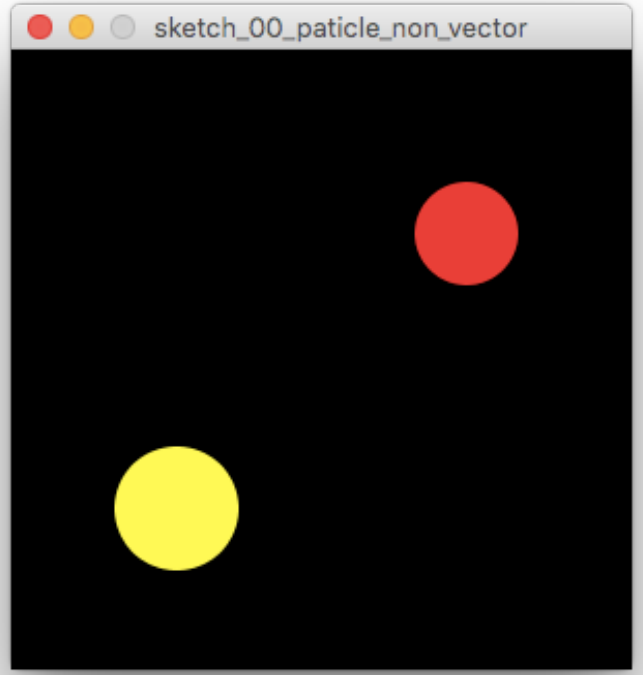
PVectorオブジェクトの使い方

```
float x;
float y;
PVector p;              // PVectorの宣言

void setup(){
  size(400,400);
  x = 100;
  y = 200;

  p = new PVector();    // PVectorをコンストラクト
  p.x = 300;            // PVector のxとyに代入
  p.y = 200;
}

void draw(){
  background(0);
  ellipse(x,y,50,50);           // 変数x, yで描いた円
  ellipse(p.x, p.y, 50,50);     // PVector を介して描いた円
}
```

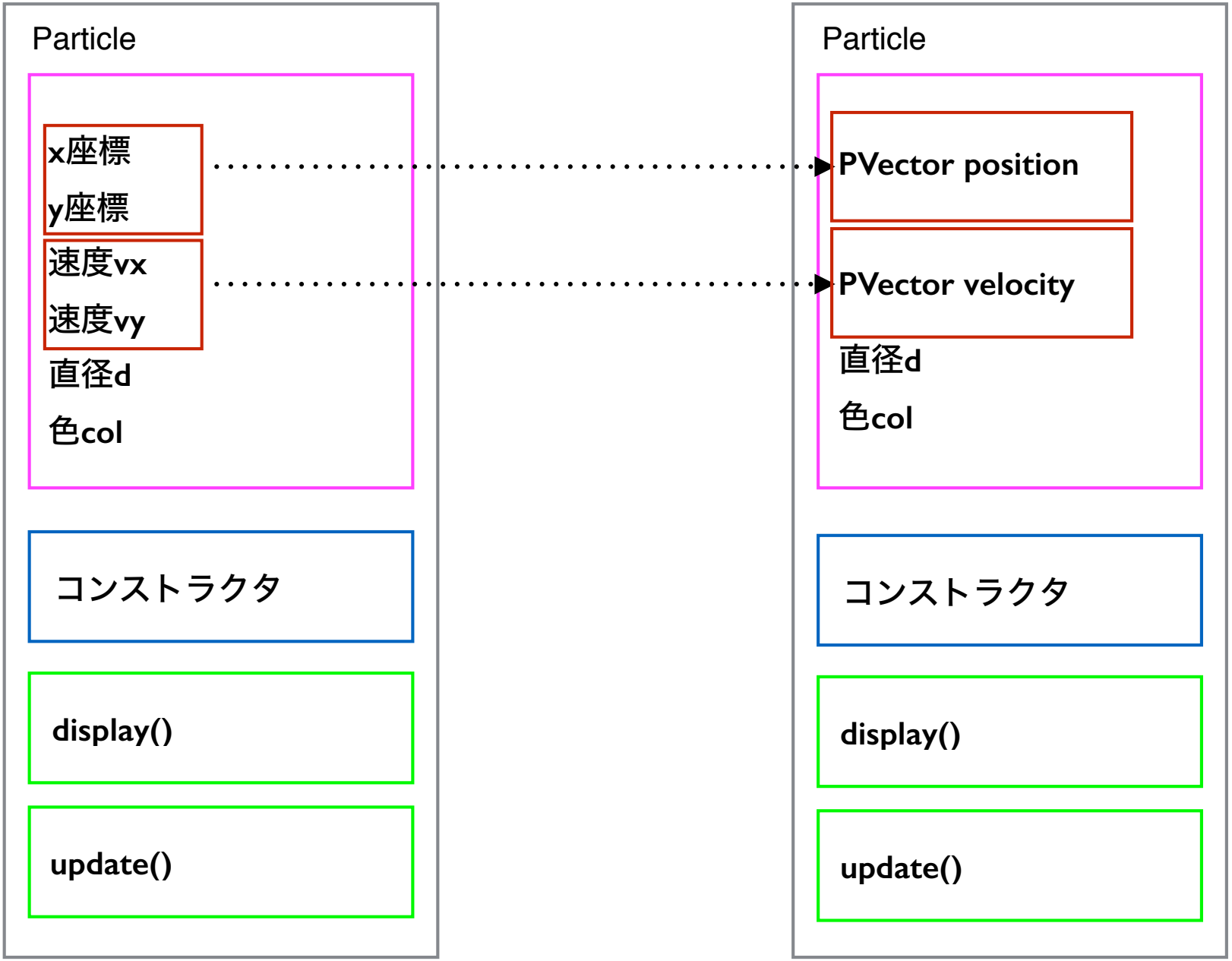
PVectorクラス

sketch_00_paticle_non_vector.pdeの

ParticleクラスをPVectorで書き直してみよう



PVectorクラスにする変数は



PVectorクラスにする変数は

変更前

```
class Particle{
 float x;
 float y;
 float d;
 float vx;
 float vy;
 color col;

 Particle(float _x, float _y, float _d, float _vx, float _vy, color _col){
   x = _x;
   y = _y;
   d = _d;
   vx = _vx;
   vy = _vy;
   col = _col;
 }
```

変更後

```
class Particle{
 PVector position;
 PVector velocity;
 float d;
 color col;

 Particle(float _x, float _y, float _d, float _vx, float _vy, color _col){
   position = new PVector(_x, _y);
   velocity = new PVector(_vx, _vy);
   d = _d;
   col = _col;
 }
```

PVectorクラスにする変数は

変更前

```
void display(){
  fill(col);
  ellipse(x, y, d, d);
}
```

変更後

```
void display(){
  fill(col);
  ellipse(position.x, position.y, d, d);
}
```

PVectorクラスにする変数は

変更前

```
void update(){
  x = x + vx;
  y = y + vy;

  if(x < d/2){
    vx *= -1;
    x = d/2;
  }
  if(x > width - d/2){
    vx *= -1;
    x = width - d/2;
  }
  if(y < d/2){
    vy *= -1;
    y = d/2;
  }
  if(y > height - d/2){
    vy *= -1;
    y = height - d/2;
  }
}
```

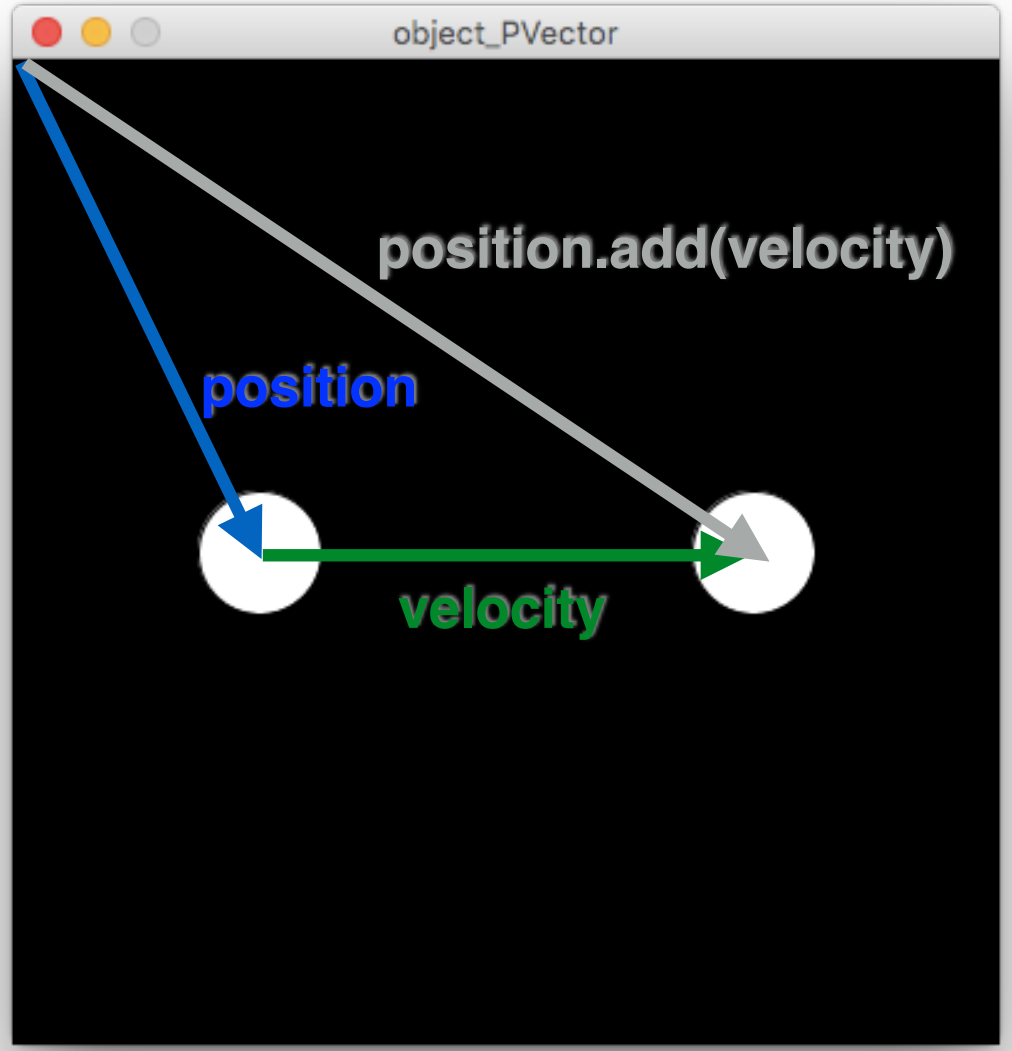
変更後

```
void update(){
  position.add(velocity);

  if(position.x < d/2){
    velocity.x *= -1;
    position.x = d/2;
  }
  if(position.x > width - d/2){
    velocity.x *= -1;
    position.x = width - d/2;
  }
  if(position.y < d/2){
    velocity.y *= -1;
    position.y = d/2;
  }
  if(position.y > height - d/2){
    velocity.y *= -1;
    position.y = height - d/2;
  }
}
```

PVector同士を足す

position.x = position.x + velocity.x;
position.y = position.y + velocity.y;
と同じ

ベクトルの足し算




```
position.add(velocity);
```

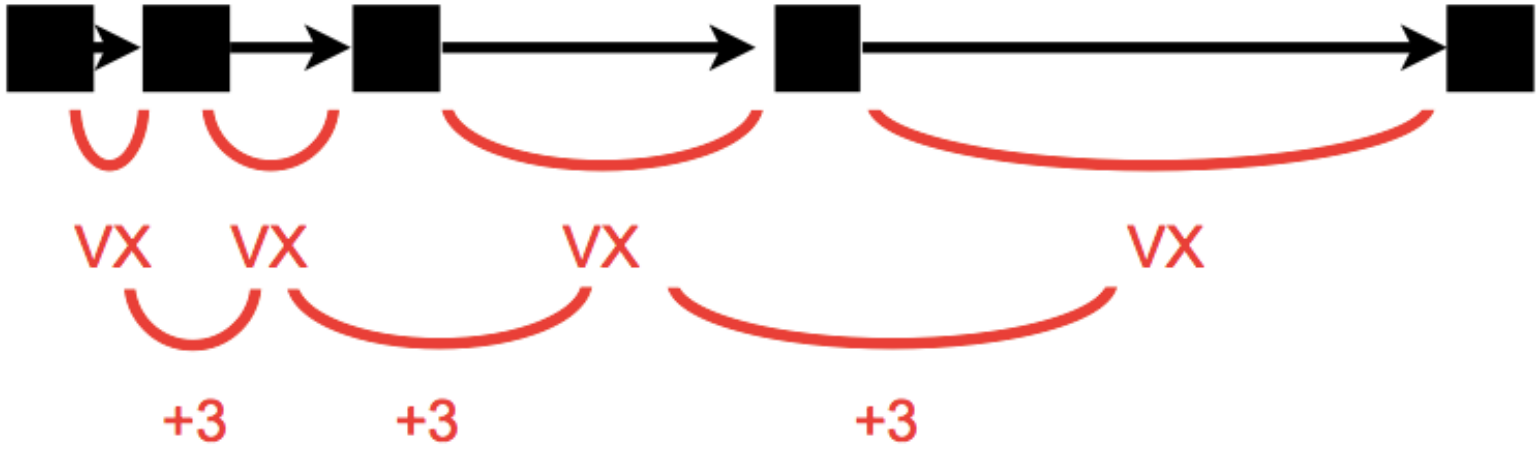
PVector同士を足す

position.x = position.x + velocity.x;
position.y = position.y + velocity.y;
と同じ

「力」のシミュレーション1　重力

「力」と速度の関係

力が加わると速度が一定に変化する
常に下向きの力（重力）が加わると跳ねる動きになる



VX = VX +3;

重力をPVectorを使って組み込んでみる

Particle

PVector position
PVector velocity
直径d
色col

コンストラクタ

display()

update()

Particle

PVector position
PVector velocity
PVector force
直径d
色col

コンストラクタ

display()

update()

加速度で速度を更新する
applyForce()

重力をPVectorを使って組み込んでみる

Particle

PVector position
PVector velocity
PVector force
直径d
色col

```
class Particle{
 PVector position;
 PVector velocity;
 PVector force;
 float d;
 color col;
```

コンストラクタ

```
Particle(float _x, float _y, float _d, float _vx,
  position = new PVector(_x, _y);
  velocity = new PVector(_vx, _vy);
  force = new PVector(0,1);
  d = _d;
  col = _col;
}
```

display()

update()

加速度で速度を更新する
applyForce()

```
void applyForce(){
      velocity.add(force);
}
```

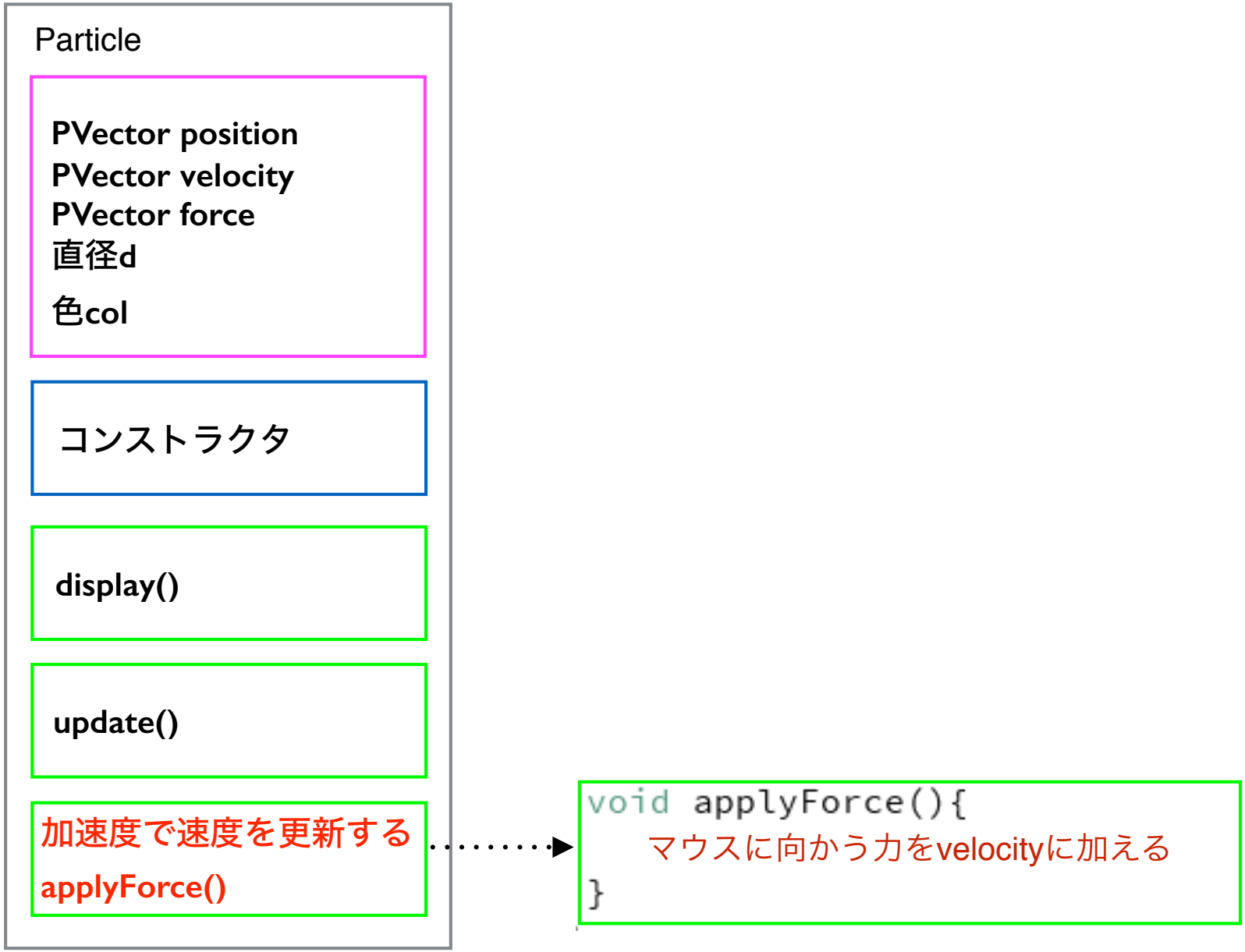
重力をPVectorを使って組み込んでみる

すべてのparticleでapplyForceを実行

```
void draw(){

  background(0);
  p.applyForce();
  p2.applyForce();
  p.update();
  p2.update();
  p.display();
  p2.display();

}
```

「力」のシミュレーション２ 引力

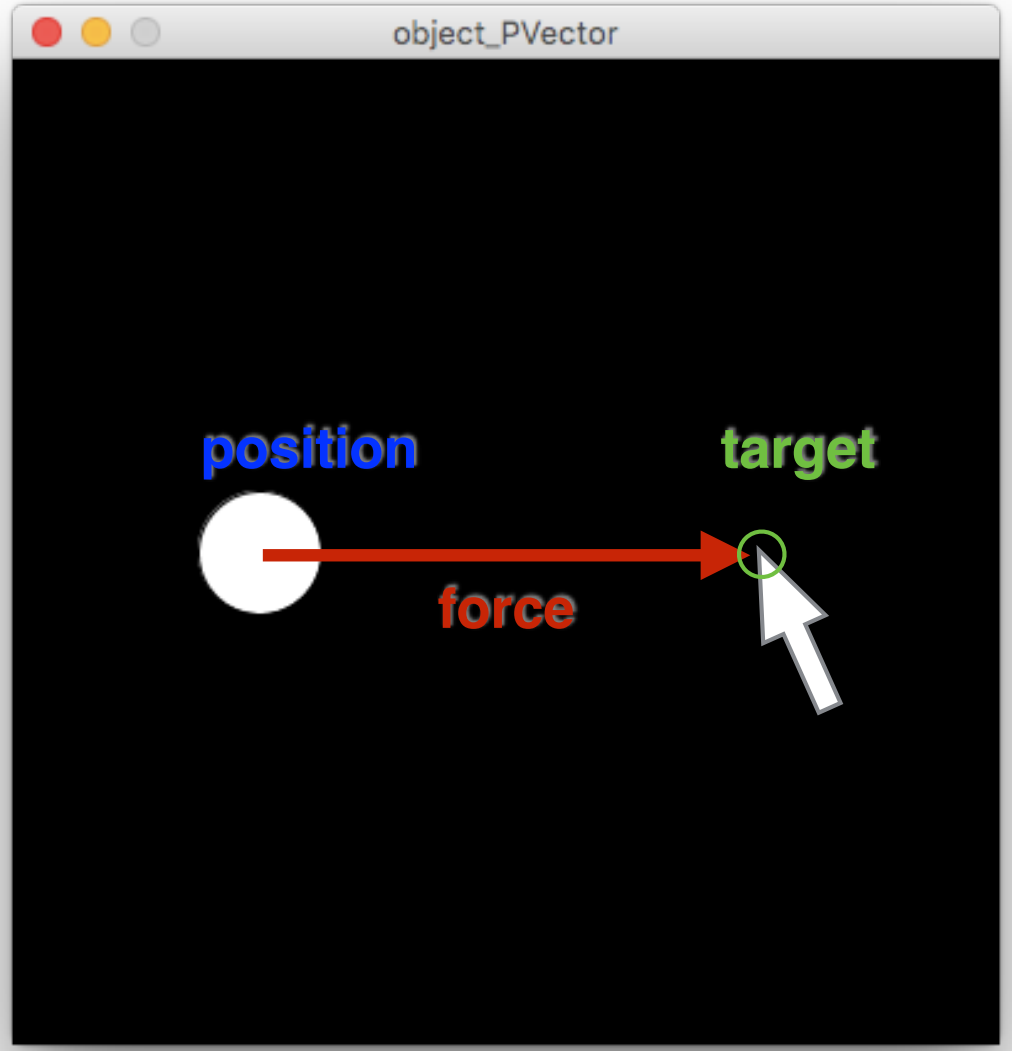
マウスの位置に向かって吸い込まれる力を加える



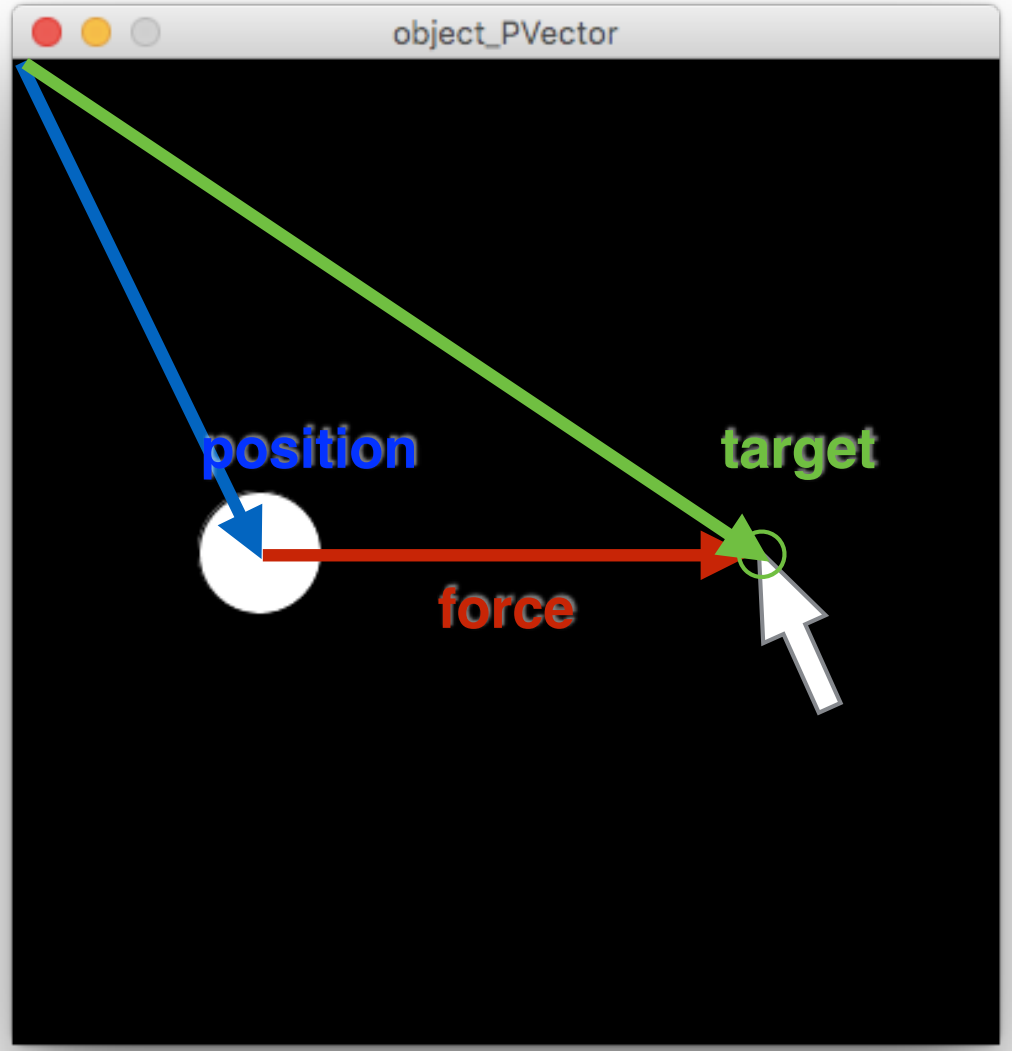
マウスの位置に向かって吸い込まれる力を加える

```
void applyForce(){

    PVector target = new PVector(mouseX, mouseY);
    force = PVector.sub(target,position);
    float distance = force.mag();
    if(20.0 < distance){
      force.normalize();
      force.mult(100.0/pow(distance,2));
      velocity.add(force);
    }
}
```

力の計算方法



力の計算方法

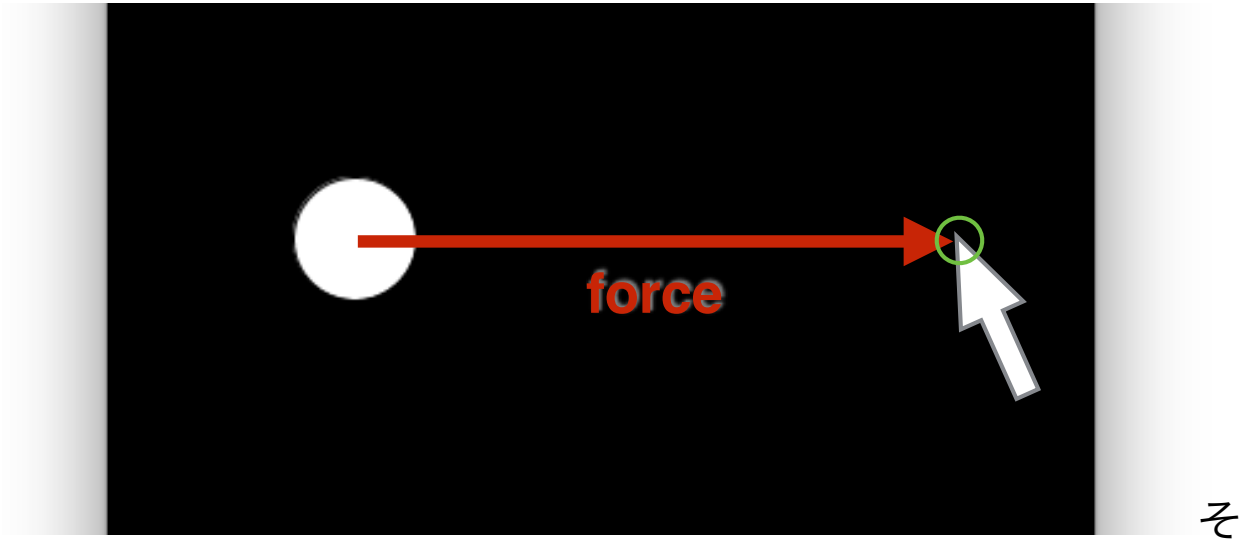


targetから positionを引くと

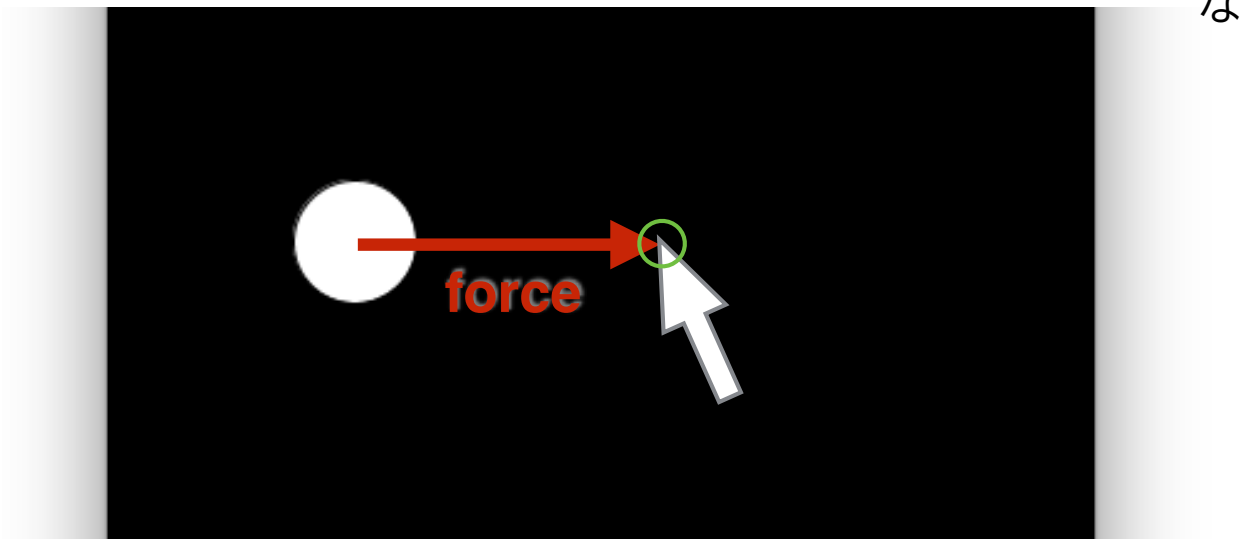
forceを求めることができる

```
force = PVector.sub(target,position);
```

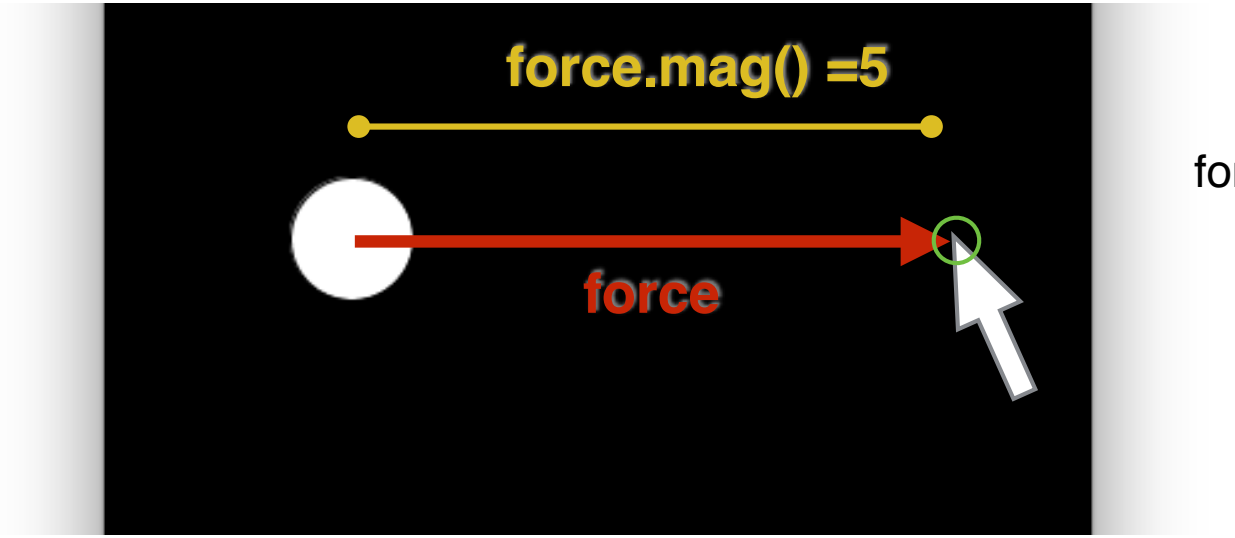
力の計算方法



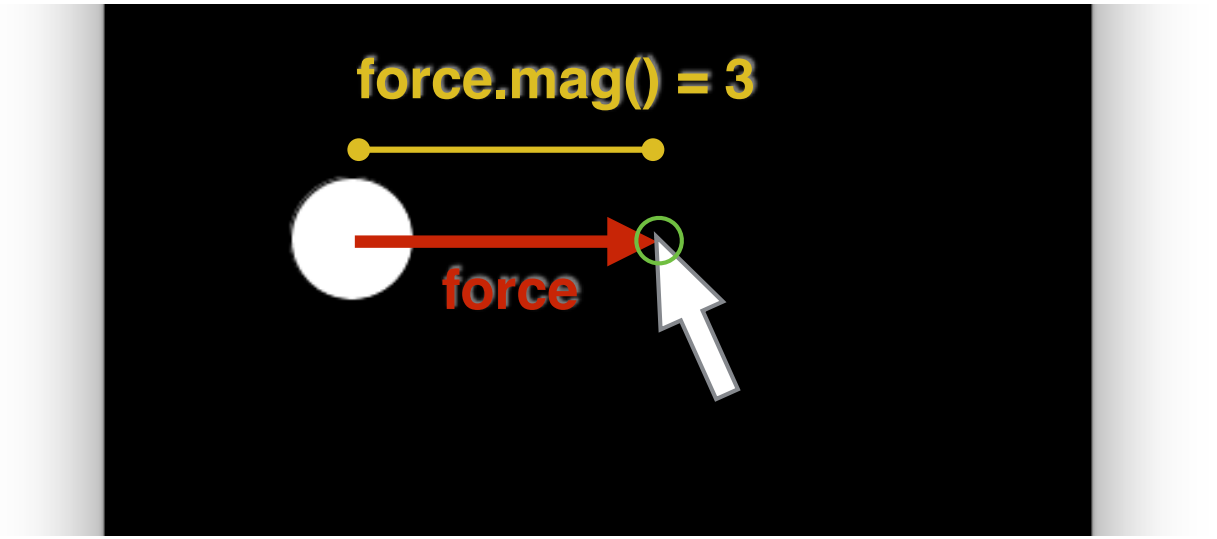
そのままforceを使うと問題が。。。
targetから離れているほどforceが大きくなってしまう。



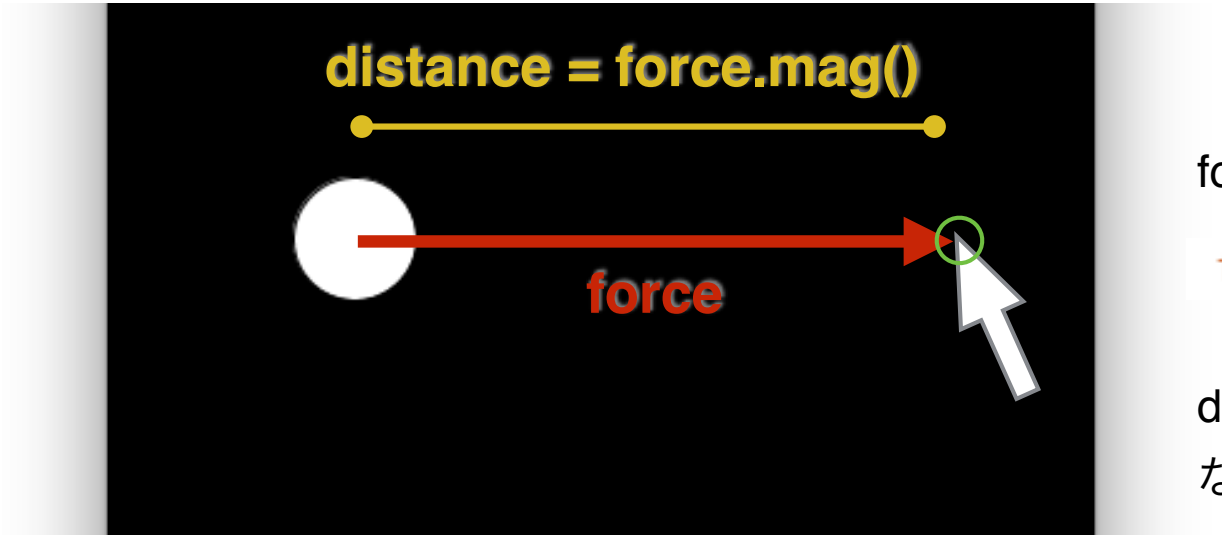
力の計算方法



force.mag()で、forceの大きさがわかる。



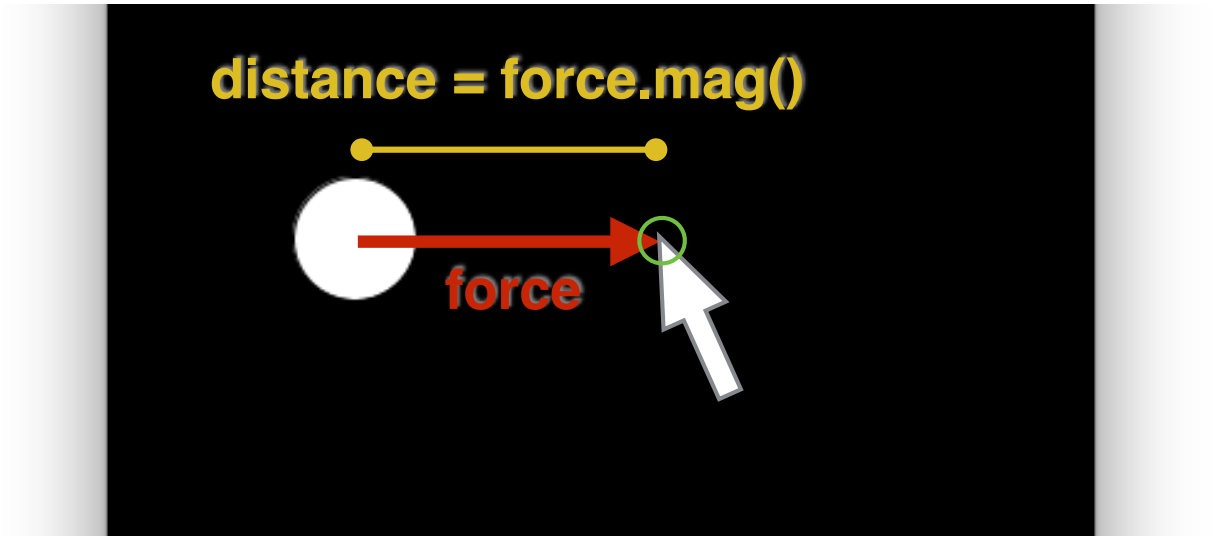
力の計算方法



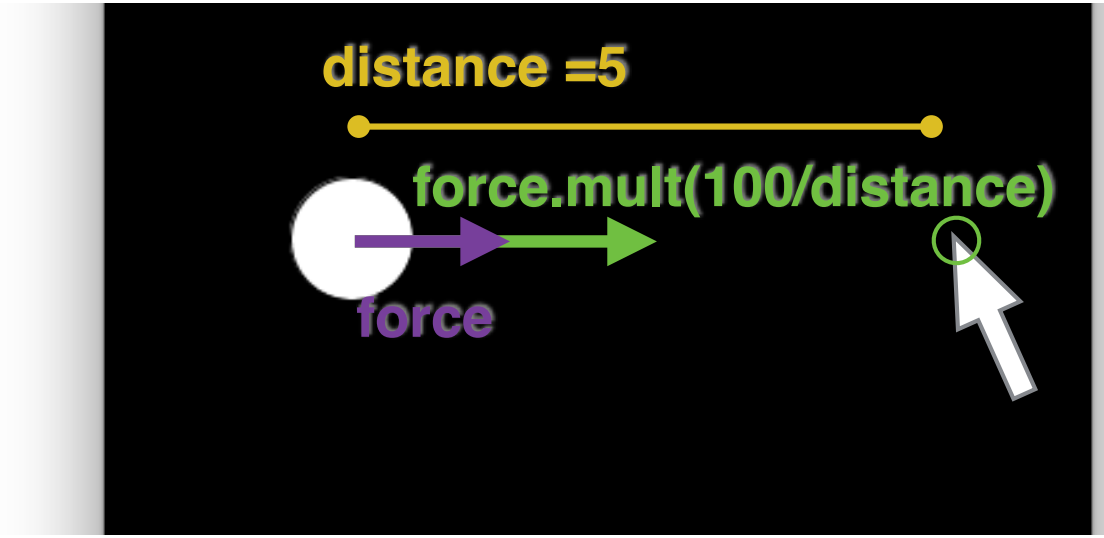
force.mag()で、forceの大きさがわかる。

```
float distance = force.mag();
```

distance が大きいほど、forceは小さくなるようにしたい。



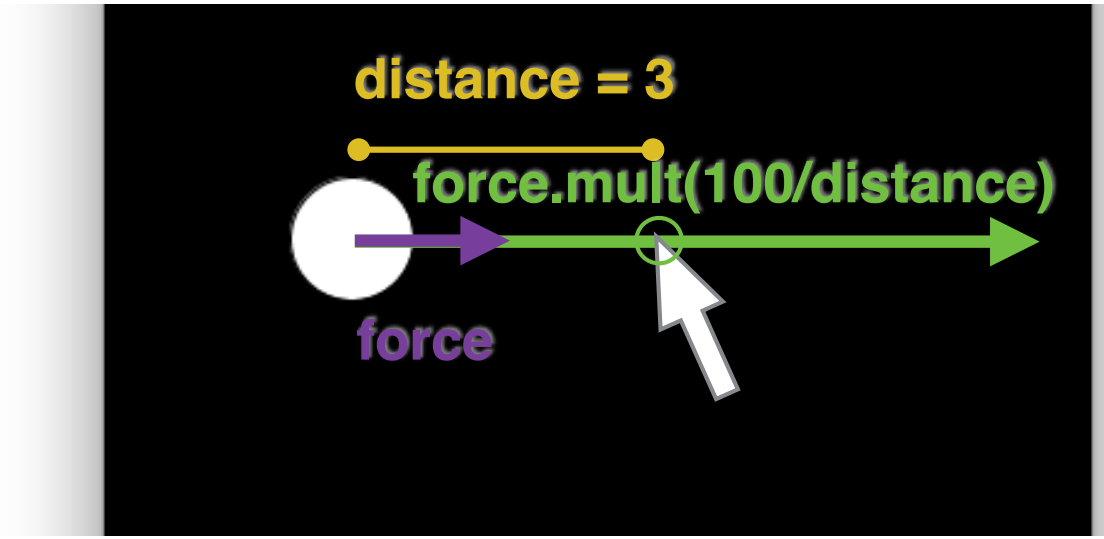
力の計算方法



forceにdistanceの逆数をかけると
distanceが小さいほど、大きなforceになる。

```
force.mult(100.0/pow(distance,2));
```

実際のコードでは、ditanceの2乗の逆数を
かけている。




```
velocity.add(force);
```

最後に、velocityにforceを足す。

演習1

・初期の速度を0にしてみる

演習2

・配列を使って、1000個のparticleを出現させる

```
int num = 1000;

Particle[] particles = new Particle[num];

void setup(){
 size(600,600);
 noStroke();
 for(int i = 0; i<num; i++){
   particles[i] =new Particle(random(0,width), random(0,height), 3,0,0,
                                    color(random(255), random(255), random(255)));
 }
}

void draw(){
 background(0);
 for(int i = 0; i<num; i++){
   particles[i].applyForce();
   particles[i].update();
   particles[i].display();
 }
}
```

完成版：sketch_04_particles

粒子の誕生、生存、消滅のシミュレーション

これまで

最初から決まった数のparticleが出現していた。

1000個のparticle

```
int num = 1000;                                          //particleの数1000

Particle[] particles = new Particle[num];                //1000個の配列を宣言、初期化

void setup(){
 size(600,600);
 noStroke();
 for(int i = 0; i<num; i++){                             // setupで1000個のparticleを生成
   particles[i] =new Particle(random(0,width), random(0,height), 3,0,0,
                              color(random(255), random(255), random(255)));
 }
}

void draw(){
 background(0);
 for(int i = 0; i<num; i++){
   particles[i].applyForce();                            // drawでparticleの位置や速度の更新
   particles[i].update();
   particles[i].display();
 }
}
```

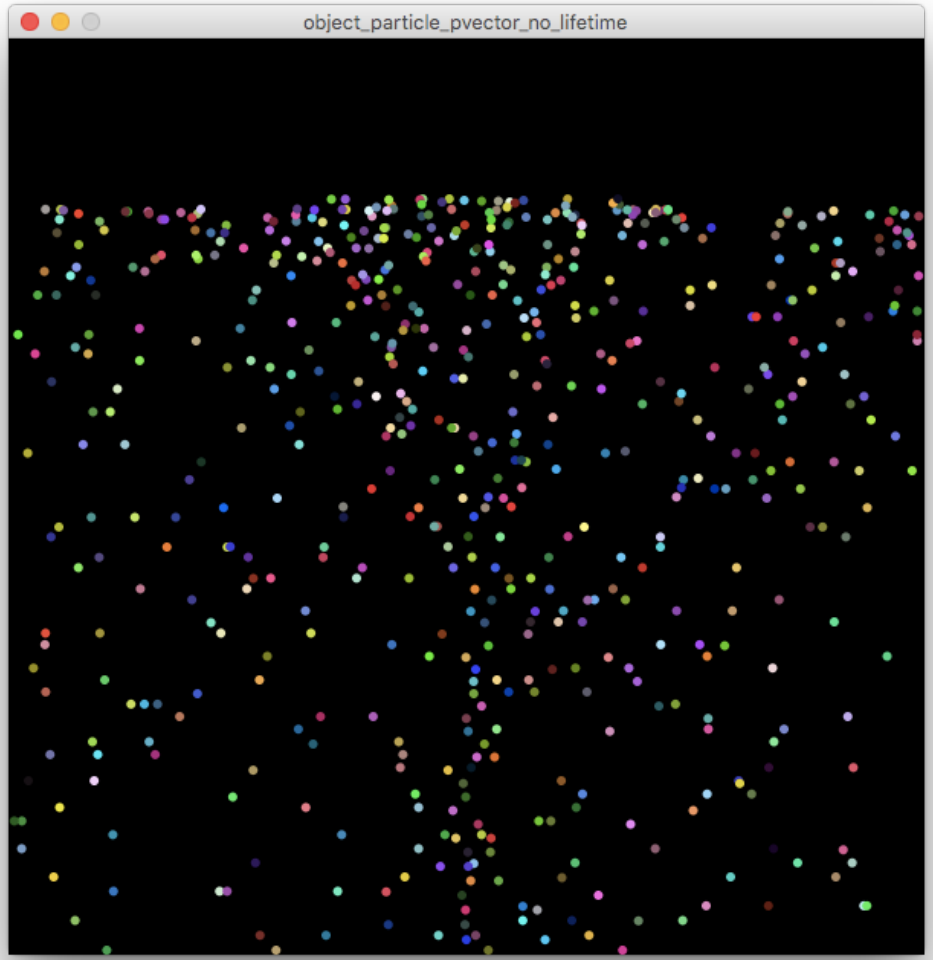
Particleの数が増える（決まっていない）

→可変型の配列ArrayListを使う。

```
ArrayList<Particle> particles;                  // ArrayListの宣言

void setup(){
 size(600,600);
 particles = new ArrayList<Particle>();         // ArrayListの初期化
 noStroke();
}

void draw(){
 background(0);

                                                // drawでparticleをひとつ生成
 Particle p = new Particle(width/2, height, 6,
                           random(-1,1),random(-10,-150),
                           color(random(255), random(255), random(255)));
 particles.add(p);          // arrayListの最後の要素に生成したParticleを加える

 for(int i = 0; i<particles.size(); i++){
   particles.get(i).applyForce();
   particles.get(i).update();
   particles.get(i).display();
 }
}
```

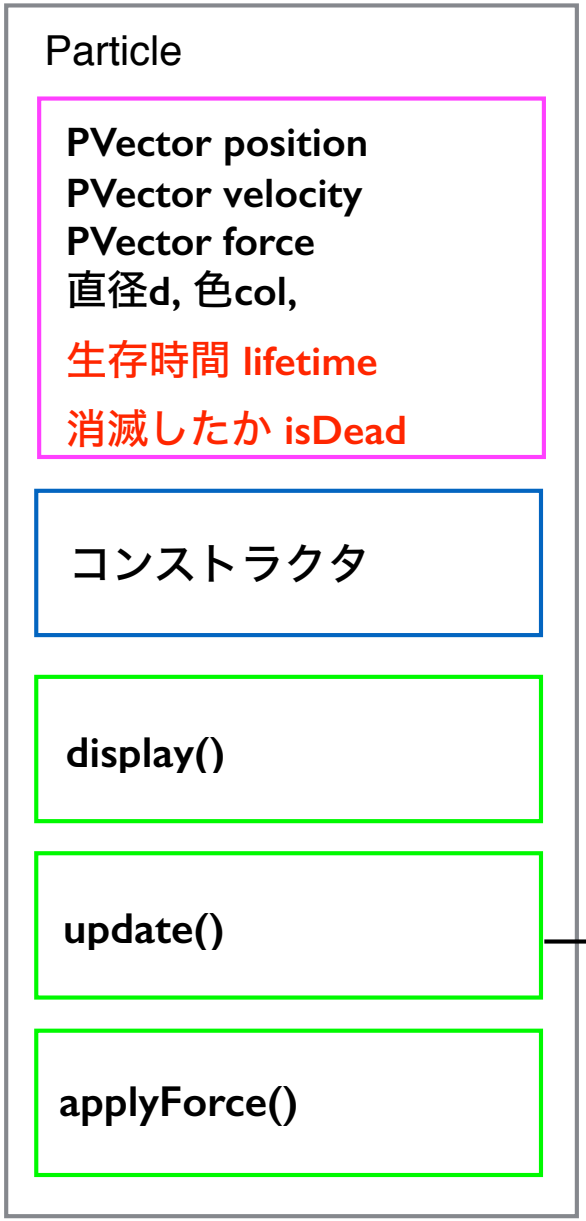
次々と粒子が誕生するサンプル

sketch_05_particles_arrayList



ただし、無限にParticleが増えるので問題がある

粒子の誕生、生存、消滅



updateするごとに、lifetimeを減らす。
lifetimeが0以下なら、isDead = true

draw時に、isDeadがtrueの粒子は
arrayListから削除する

Particle クラスの拡張

```
class Particle{
 PVector position;
 PVector velocity;
 PVector force;
 float d;
 float vx;
 float vy;
 color col;
 float lifetime;
 boolean isDead;

 Particle(float _x, float _y, float _d, float _vx, float _vy, color _col, float _lifetime){
   position = new PVector(_x, _y);
   velocity = new PVector(_vx, _vy);
   force = new PVector(0,0);
   d = _d;
   col = _col;
   lifetime = _lifetime;
   isDead = false;
 }
```

生存時間 lifetime

消滅したか isDead

Particle クラスの拡張（続き）

```
void update(){
  lifetime -= 0.005;

  if(lifetime < 0){
    isDead = true;
  }

  position.add(velocity);

  if(position.x < d/2){
     velocity.x *= -1;
     position.x = d/2;
  }
  if(position.x > width - d/2){
```

updateするごとに、lifetimeを0.005減らす。

lifetimeが0未満なら、isDead = true

drawブロックの変更

```
void draw(){
 background(0);

 Particle p = new Particle(width/2, height, 6,
                           random(-1,1),random(-10,-150),
                           color(random(255), random(255), random(255)),1.0);
 particles.add(p);

 for(int i = 0; i<particles.size(); i++){
   if(particles.get(i).isDead){
    particles.remove(i);
   }
   particles.get(i).applyForce();
   particles.get(i).update();
   particles.get(i).display();
 }
}
```

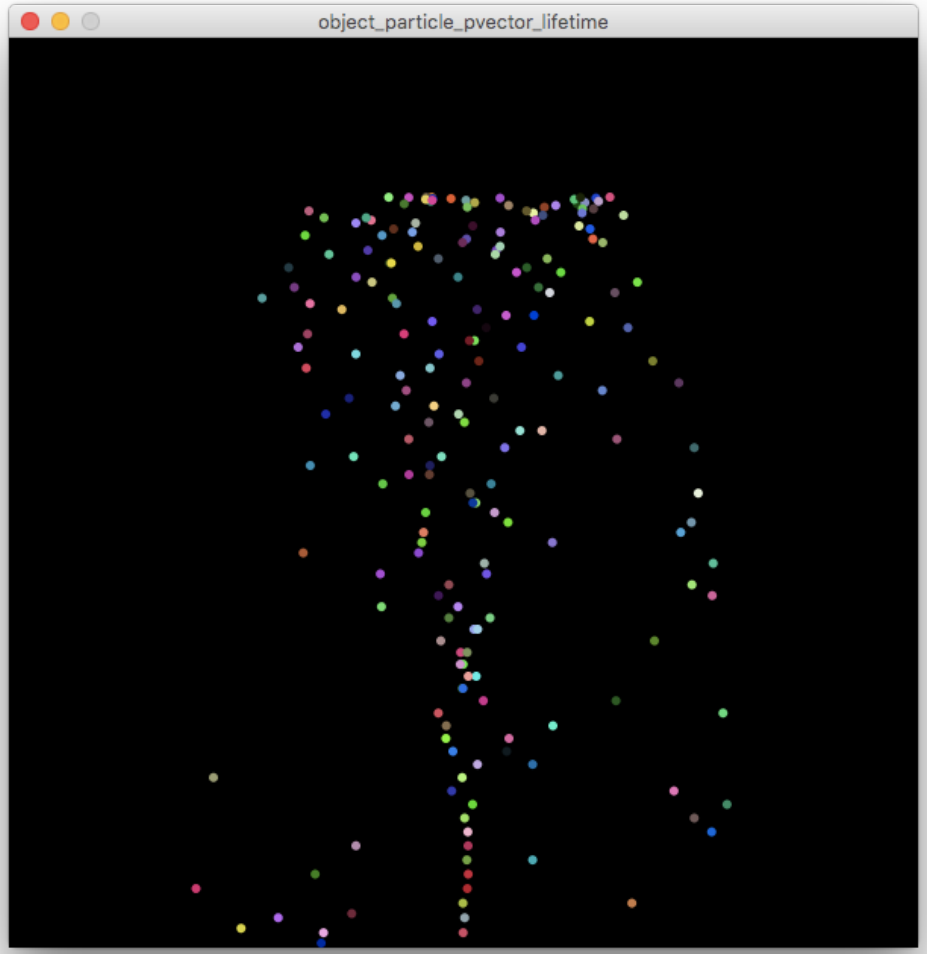
Particleを生成するとき、初期のlifeTimeは1.0

isDeadがtrueの粒子はarrayListから削除する

粒子の誕生、生存、消滅

一定時間が過ぎると、粒子が消滅するサンプル

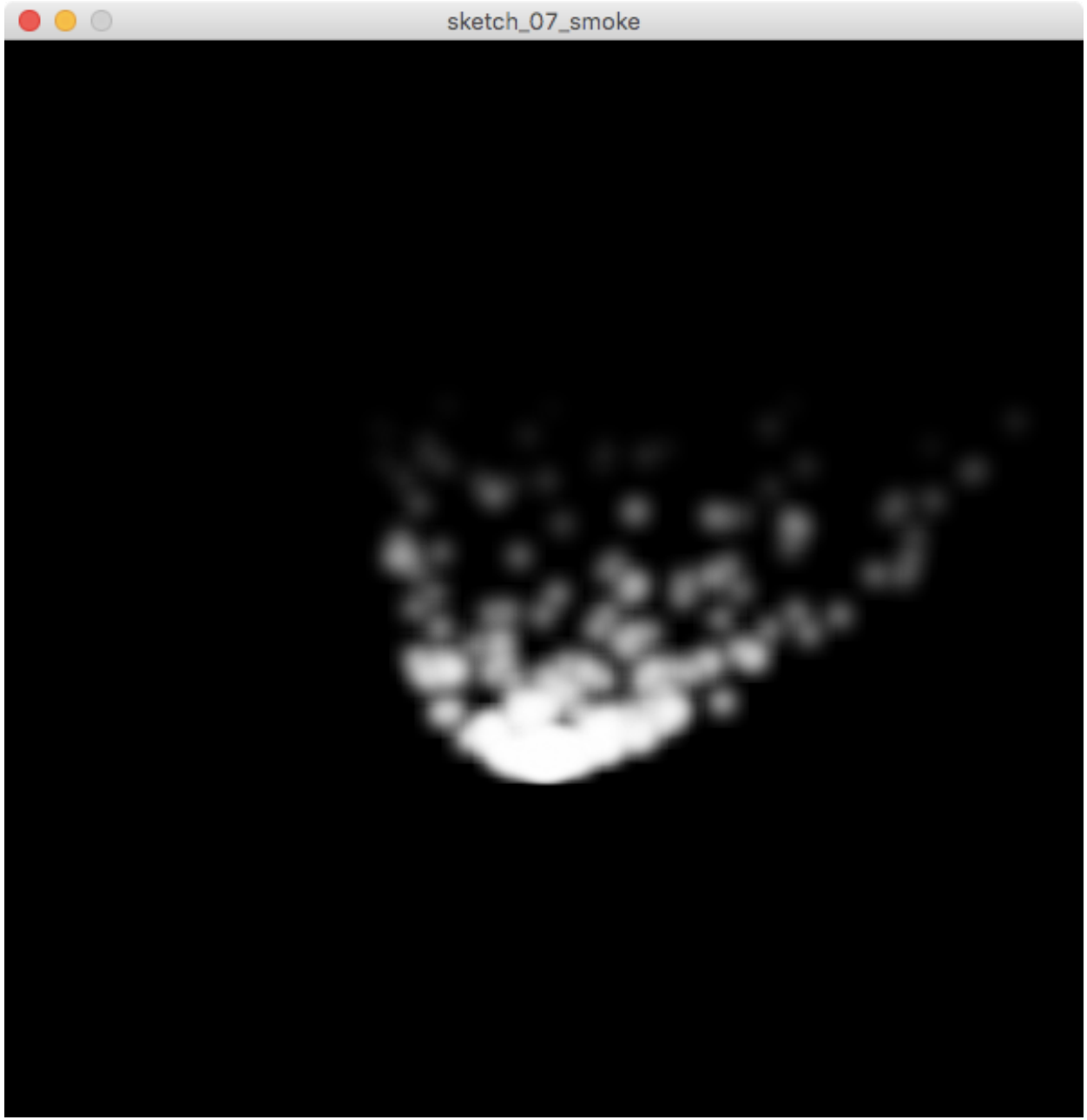
sketch_06_particles_life



演習

・lifetimeが少なくなると、色を薄くする。

画像を使って煙のシミュレーション



Particleクラスの拡張

Particle

PVector position
PVector velocity
PVector force
直径d, 色col,

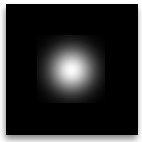
lifetime
isDead
PImage texture

コンストラクタ

display()

update()

applyForce()



dataフォルダにtexture.pngがある

display() → ellipse()ではなく、image()でtexture.pngを表示

Particle クラスの拡張

```
class Particle{
 PVector position;
 PVector velocity;
 PVector force;
 PImage texture;
 float d;
 float vx;
 float vy;
 color col;
 float lifetime;
 boolean isDead;

 Particle(float _x, float _y, float _d, float _vx, float _vy, float _lifetime, PImage _texture){
   position = new PVector(_x, _y);
   velocity = new PVector(_vx, _vy);
   force = new PVector(0,0);
   d = _d;
   lifetime = _lifetime;
   isDead = false;
   texture = _texture;
 }
```

Particle クラスの拡張（続き）

```
void display(){
  imageMode(CENTER);
  tint(255,255*lifetime);
  image(img,position.x,position.y,30,30);
}
```

lifetimeが少なくなると、
色を薄くする。

```
void applyForce(){
    force = new PVector(0.005, -0.01);
    velocity.add(force);
}
```

左下から右上に流れる
ような力の加え方

setup、drawの変更

```
ArrayList<Particle> particles;
PImage img;

void setup(){
 size(600,600);
 particles = new ArrayList<Particle>();
 img = loadImage("texture.png");
 noStroke();
}

void draw(){
 background(0);

 Particle p = new Particle(width/2, height-200, 6,
                           random(-1,1),random(-0.1,0.1),1.0, img);
 particles.add(p);

 for(int i = 0; i<particles.size(); i++){
   if(particles.get(i).isDead){
    particles.remove(i);
   }
   particles.get(i).applyForce();
   particles.get(i).update();
   particles.get(i).display();
 }
}
```

texture.pngを読み込む変数imgを宣言

texture.pngの読み込み

particle生成時にimgを指定

はじめのvelocityは小さめに設定