PVector

XY 座標を 1 つにまとめて扱う型

```
PVector p = new PVector(100, 50);
```

PVector とは?

x と y（座標）を 1 つにまとめて持つオブジェクト

- x と y を 1 つの「点」としてまとめて扱える
- 座標だけでなく 速度・向き（ベクトル）も表せる
- add ・ dist などの計算を メソッドで行える

1 つの p に x と y が入る。 p.x ・ p.y で取り出す（次のスライド）。

```
// x, y を別々に持つと…
float x = 100;
float y = 50;

// PVector なら 1 つにまとめられる
PVector p =
      new PVector(100, 50);
```

1 作る・フィールド ― .x / .y で取り出す

```
PVector p = new PVector(100, 50);   // (x, y)

println(p.x);   // 100.0 （x フィールド）
println(p.y);   // 50.0  （y フィールド）
p.x += 5;   // x だけ動かす
```

PVector が持つフィールド

`p.x` … 横（float）
`p.y` … 縦（float）
`p.z` … 奥行き（3D 用）

3D は new PVector(x, y, z)

PImage の .width / .pixels と同じで、PVector も「.（ドット）」でフィールドにアクセスする。

color との違い – 「型」と「オブジェクト」

似ているが別物。特に new が要るか要らないかが違う

color 型・値

正体

基本型（中身は int ）

作り方 new 不要

```
color c = color(255,0,0);
```

中身の取得

```
red(c) / green(c) / blue(c)
```

代入したとき

値がコピー（別物になる）

PVector クラス・オブジェクト

正体

クラス（オブジェクト）

作り方 new が必要

```
PVector p = new PVector(…);
```

中身の取得

```
p.x / p.y
```
（フィールド）

代入したとき

参照がコピー（同じ実体）

覚えどころ： color は関数で「値」を作る（ new なし）／ PVector は new で「オブジェクト」を作る。代入も color は中身のコピー、 PVector は同じ実体を指す。

2 動かす — add で「位置 + 速度」

```
PVector pos = new PVector(100, 100);   // 位置
PVector vel = new PVector(2, -1);   // 速度

pos.add(vel);   // pos = pos + vel
// pos.sub(vel);  // 引く
```

位置ベクトル + 速度ベクトル
= 少し進んだ新しい位置。

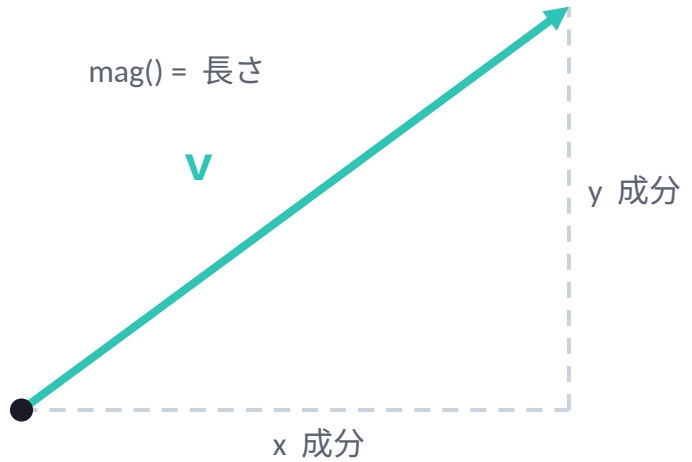
draw() の中で毎フレーム
`pos.add(vel)` すると、
点が動き続ける。

add / sub は「pos 自身」を書き換える。元を残したいときは PVector.add(a, b) の形（新しいベクトルを返す）を使う。

3 ベクトル演算 — 距離・長さ・向き

```
float d = PVector.dist(a, b);   // 2 点 a,b の距離
float len = v.mag();   // ベクトルの長さ

v.normalize();   // 長さを 1 に（向きだけ）
v.mult(3);   // 3 倍に伸ばす
```



dist / mag で距離や大きさ、 normalize / mult で向きと長さを調整する。

配列と組み合わせる

PVector の配列で「点の集まり」を管理する（配列 × PVector ）

```
PVector[] pts = new PVector[3];   // PVector の配列
pts[0] = new PVector(50, 50);
pts[1] = new PVector(120, 80);

for (int i = 0; i < pts.length; i++) {
  ellipse(pts[i].x, pts[i].y, 10, 10);   //  各点に円を描く
}
```

点の座標を配列でまとめ、 pts[i].x / pts[i].y でアクセス。軌跡・パーティクル・図形の頂点などに使える。

まとめ

1

作る

```
new PVector(x, y)
```

2

フィールド

```
p.x  /  p.y
```

3

演算

```
add / dist / mag
```

覚える形：　PVector　名前　= new PVector(x, y);

情報処理基礎 | 演習 4C | Processing

画像のピクセルに配列でアクセスする

読み込んだ画像を pixels[] で 1 画素ずつ扱う

```
color c = img.pixels[y * img.width + x];
```

画像は「色の配列」

2 次元の画像を 1 次元配列 pixels[] にならべて持つ（左上→右へ）

x →

y ↓

0	1	2	3
4	5	6	7
8	9	10	11

2 次元の座標 (x, y) を 1 次元のインデックスに変換する

```
index = x + y * width
```

例） x=2, y=1, width=4 → index = 2 + 1×4 = 6 （左の水色のマス）

width = 横のピクセル数（ 1 行に並ぶ要素数）。行が変わるたびに width ぶんだけ番号が進む。

1 画像を読み込む — loadImage

```
PImage img;    //  画像用の変数（型は  PImage ）

void setup() {
  size(400, 300);
  img = loadImage("photo.jpg");    //  読み込み
  img.resize(width, height);    // ★ 窓の大きさに合わせる
  image(img, 0, 0);    //  画面に表示
}
```

どんな解像度でも押し込む
`img.resize(width, height)` で、画像を ウィンドウ（ 400×300 ）にそろえる。

これで img.width / img.height も 400 ・ 300 になり、後のピクセル計算とズレない。

画像は **data フォルダ** に置く。

2 img が持つ「フィールド」 — PImage 型のオブジェクト

変数 img は、画像の情報を「フィールド」としてまとめて持つ。ドット . で取り出す。

`img` ← PImage 型の変数（オブジェクト）

`img.width` `int`	横のピクセル数
`img.height` `int`	縦のピクセル数
`img.pixels[]` `color[]`	全画素の色（配列）

```
img.loadPixels();   // 準備（関数）
color c = img.pixels[0];   // フィールド
int w = img.width;   // フィールド
```

.（ドット）は「img が持つメンバー」を指す。
() が付く `img.loadPixels()` は関数（メソッド）、付かない `img.width` や `img.pixels` は変数（フィールド）。

3 RGB を取り出す — red / green / blue

```
int x = 10, y = 5;
int index = x + y * img.width;   // 2 次元→1 次元
color c = img.pixels[index];   //  その位置の色

float r = red(c);   //  赤 0〜255
float g = green(c);   //  緑 0〜255
float b = blue(c);   //  青 0〜255
```

取り出しの流れ

座標 (x, y)
↓ index を計算
pixels[index]
↓ 色を取得
red/green/blue()
↓
R・G・B の数値

全ピクセルを走査する

二重 for 文で 端から端まで 1 画素ずつ処理する

```
img.loadPixels();
for (int y = 0; y < img.height; y++) {
  for (int x = 0; x < img.width; x++) {
    int index = x + y * img.width;
    color c = img.pixels[index];
    float r = red(c), g = green(c), b = blue(c);
    //  ここで各ピクセルを処理
  }
}
```

外側の for が縦 (y) 、内側が横 (x) 。 index = x + y × width の計算が 配列アクセスのカギ。

まとめ

1

読み込み

`loadImage()`
`loadPixels()`

2

アクセス

`pixels[x + y*width]`

3

成分を取得

`red/green/blue(c)`

応用：色を書き換えるとき

```
img.pixels[index] = color(255, 0, 0);   // 赤に変更
img.updatePixels();          // 変更を画像に反映
```