動画の作成2

担当: 佐藤

計算機実習III

第4回

sato@ise.aoyama.ac.jp

2019/5/7

座標系の移動・回転・伸縮・保存と復元

sato@ise.aoyama.ac.jp

2019/5/7

translate()

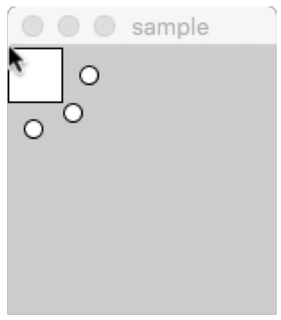
- translate(): 座標系を移動する関数
  - 引数: 移動後の座標系の原点の座標
  - draw()の中で使用➔フレームごとにリセット(次のフレームが始まった時点で原点は(0, 0)に戻る)

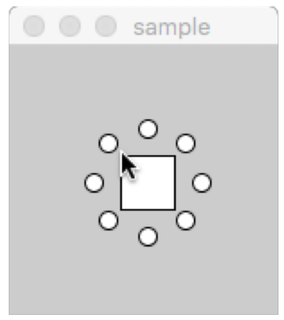
例

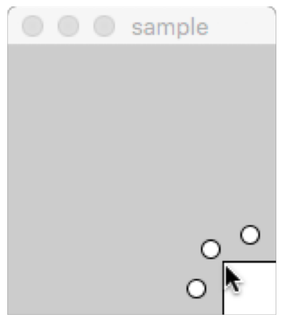
```
float diam = 30;

void setup() {
  size(150, 150);
}

void draw() {
  background(204);
  float x = min(width - diam, mouseX);
  float y = min(height - diam, mouseY);
  translate(x, y);
  rect(0, 0, diam, diam);
  translate(diam / 2, diam / 2);
  drawSatellite(diam);
}
```

```
void drawSatellite(float radius) {
  for (int i = 0; i < 8; i++) {
    float x = radius * cos(i * QUARTER_PI);
    float y = radius * sin(i * QUARTER_PI);
    ellipse(x, y, diam / 3, diam / 3);
  }
}
```

同じフレームにおいて効果は重なる

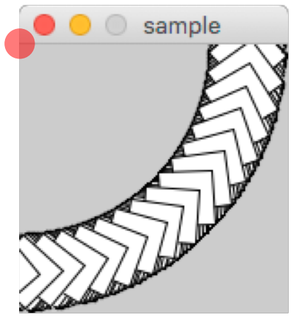


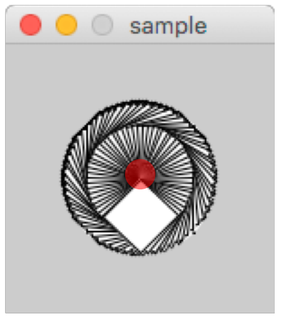




rotate()

- rotate(): 座標系を回転する関数
  - 引数: 回転角[rad]
  - 回転の中心は原点





例

```
float angle = 0;

void setup() {
  size(150, 150);
}

void draw() {
  rotate(angle);
  translate(width / 2, height / 2);
  rect(0, 0, 30, 30);
  angle = (angle + 0.1) % TWO_PI;
}
```

例

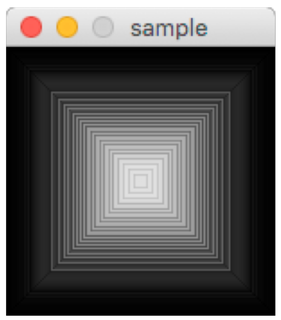
```
float angle = 0;

void setup() {
  size(150, 150);
}

void draw() {
  translate(width / 2, height / 2);
  rotate(angle);
  rect(0, 0, 30, 30);
  angle = (angle + 0.1) % TWO_PI;
}
```

scale()

- scale(): 座標系を伸縮する関数
  - 引数: 伸縮度合い
    - 0(0%)⇔1(100%)(デフォルト)⇔2(200%)...
  - draw()の中で使用➔フレームごとにリセット(次のフレームが始まった時点でスケールは元の1(100%)に戻る)

例

```
float angle = 0;

void setup() {
  size(150, 150);
  rectMode(CENTER);
}

void draw() {
  translate(width / 2, height / 2);
  scale(sin(angle));
  rect(0, 0, 150, 150);
  angle = (angle + 0.01) % TWO_PI;
}
```



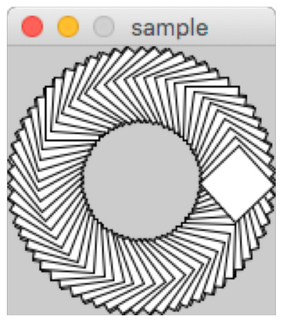
pushMatrix()，popMatrix()

- 「pushMatrix();」と「popMatrix();」の間で設定された座標系はこの間に限定される
  - 対象となる関数: 座標系を変更するもの
    - translate()，rotate()，scale()

例

```
float angle = 0 ;
float r = 60 - 15 * (sqrt(2) - 1);
float x = 0, y = 0;
void setup() {
  size(150, 150);
  rectMode(CENTER);
  noLoop();
}
void draw() {
  translate(width / 2, height / 2);
  rotate(angle);
  translate(r, 0);
  pushMatrix();
  rotate(QUARTER_PI);
  rect(0, 0, 30, 30);
  popMatrix();
  angle = (angle + 0.1) % TWO_PI;
}
```

```
void mousePressed() {
  redraw();
}
```



動き，方向，スピード

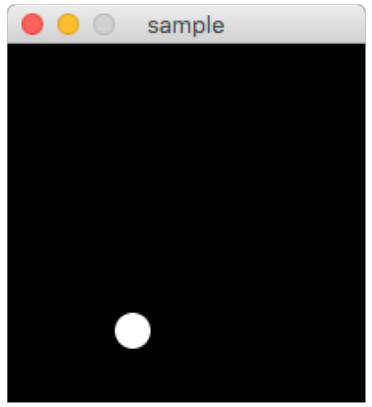
sato@ise.aoyama.ac.jp

2019/5/7

等速直線運動(1)

例

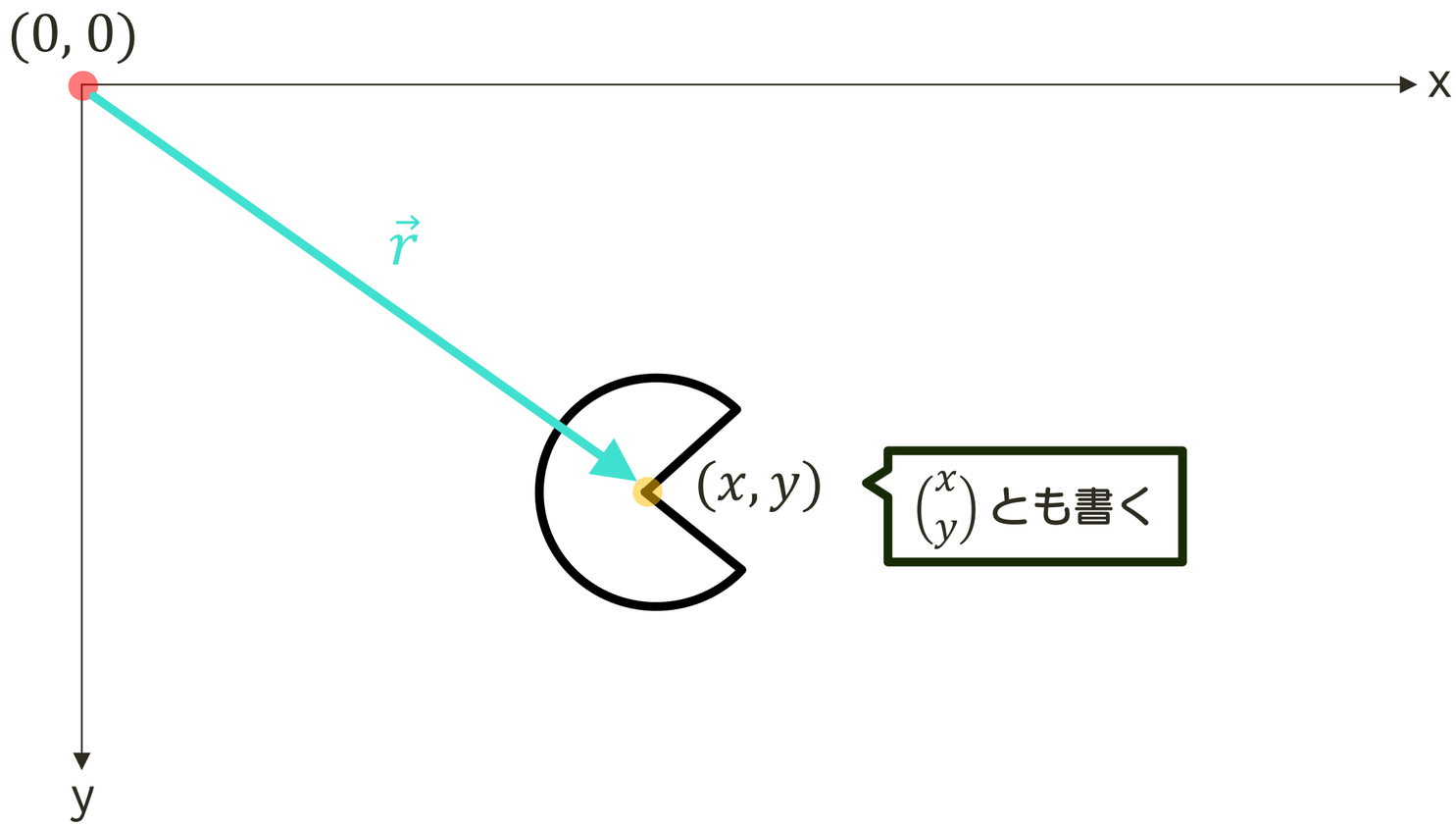
```
float ballX;
float ballY;
float ballSpeedX = 3;
float ballSpeedY = 3;
float ballDiameter = 20.0;
float ballRadius = ballDiameter / 2;

void setup() {
  size(200, 200);
  ballX = ballRadius;
  ballY = height / 2;
}
```




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

  // draw the ball
  fill(255);
  noStroke();
  ellipse(ballX, ballY, ballDiameter, ballDiameter);

  // update the location of the ball
  ballX += ballSpeedX;
  ballY += ballSpeedY;

  // bouncing at the horizontal edges of the screen
  if (ballX < ballRadius ||
      ballX > width - ballRadius) {
    ballSpeedX = -ballSpeedX;
  }

  // bouncing at the vertical edges of the screen
  if (ballY < ballRadius ||
      ballY > height - ballRadius) {
    ballSpeedY = -ballSpeedY;
  }
}
```

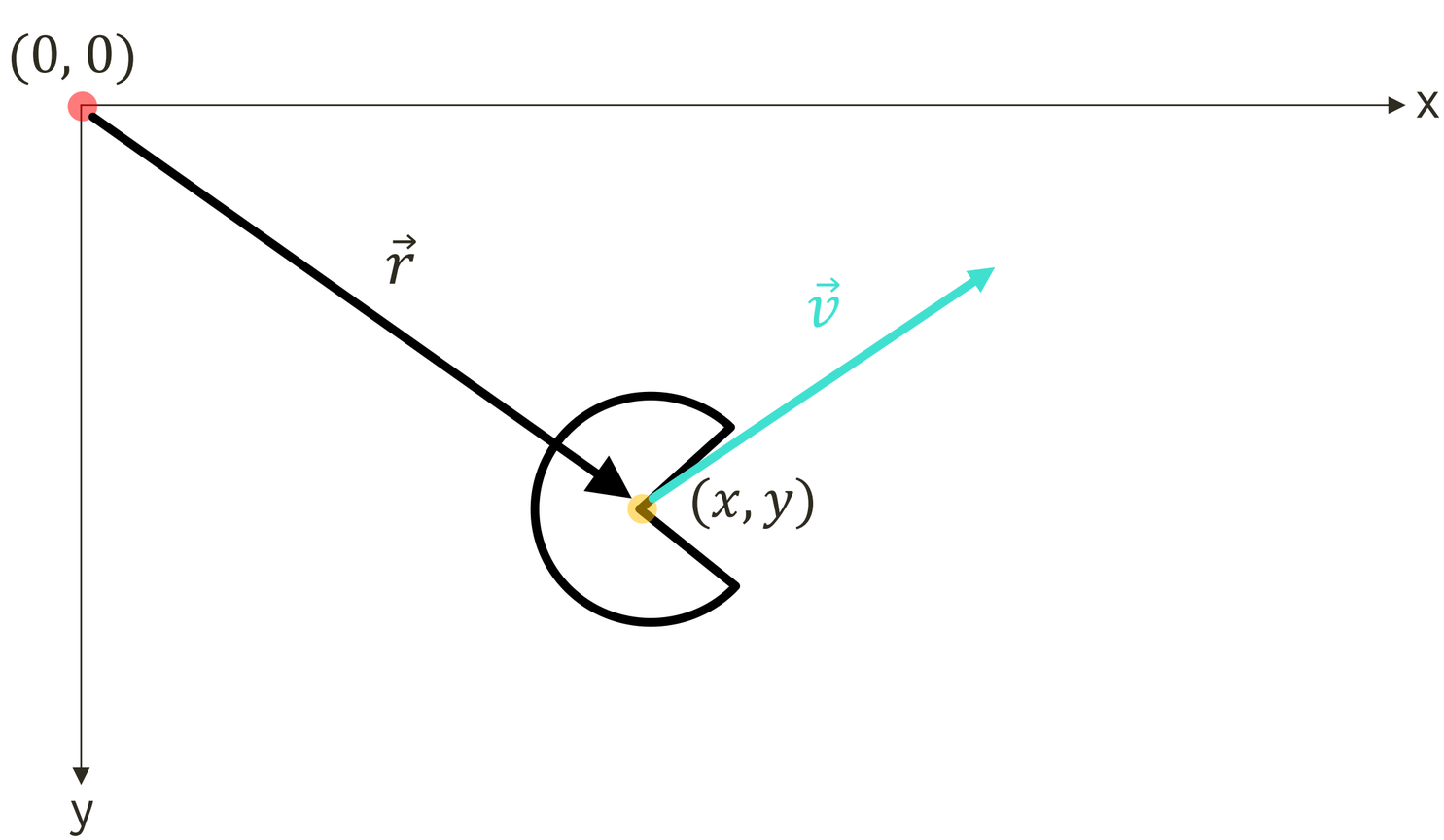
位置

- ベクトル(幾何ベクトル): 2点間の差を表す大きさと方向を持った量
  - 矢印(有向線分)で表される
    - 大きさ: 矢印の長さ
    - 方向: 矢印の向き
- 位置: 原点からある点に向かうベクトル



速度

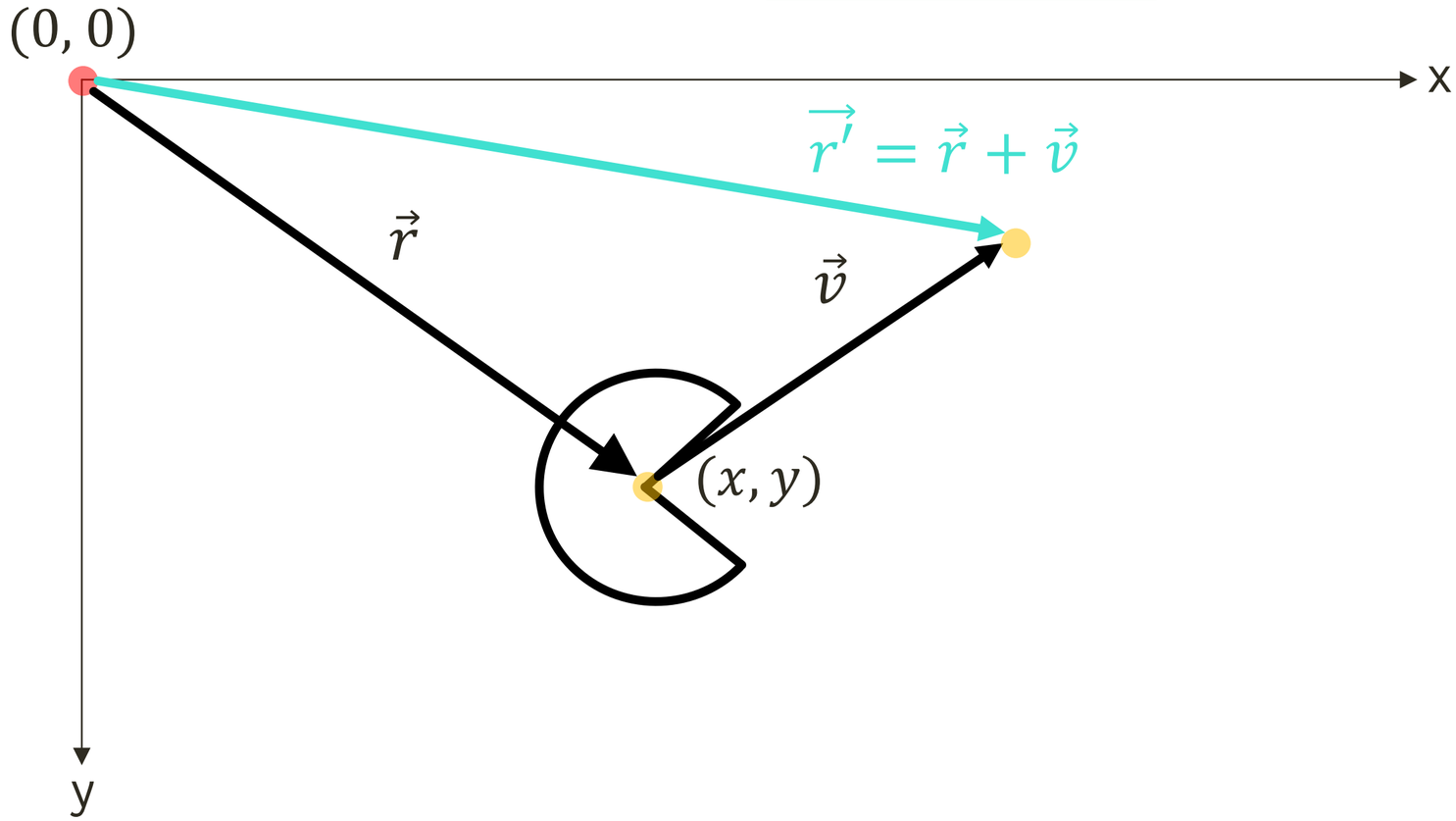
- 速度: 行き先を表すベクトル
  - 厳密には，変位を時間微分したものが速度．すなわち，速度はある瞬間から次の瞬間までに移動した位置の差．フレームレートが高ければ，大きさの小さい変位は速度と同一視できる
- 現在位置($\vec{r}$)と速度($\vec{v}$)から次の位置が定まる



ベクトル加算

- **物体の運動は $\vec{r}$ と $\vec{v}$ のベクトル加算の繰り返しで表現できる**
  - $\overrightarrow{r_{n+1}} = \overrightarrow{r_n} + \overrightarrow{v_n}$
    - **(例)** $\vec{v} = (z, w)$ **ならば** $\overrightarrow{r'} = \vec{r} + \vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} z \\ w \end{pmatrix} = \begin{pmatrix} x+z \\ y+w \end{pmatrix}$

同一成分を加算



PVectorクラス

- **PVector**: ベクトルを記述するためのクラス
- PVectorクラスのオブジェクトの値の設定方法
  - ① オブジェクト生成時のコンストラクタで設定
    - デフォルトコンストラクタ→**すべての成分の値が0**

例
```
PVector v = new PVector(1, 2);
```

  - ② オブジェクト生成後に各成分ごとに直接代入

例
```
PVector v = new PVector();
v.x = 1;
v.y = 2;
```
v.xとv.yは0で初期化

  - ③ オブジェクト生成後にset()メソッドを利用

例
```
PVector v = new PVector();
v.x = 1;
v.set(1, 2);
```

PVectorクラスのベクトル加算

- Processingでは，加算演算子(+)はintやfloatなどの基本データ型を持つプリミティブ値にしか使えない
- 2つのPVectorオブジェクトを加算するには，PVectorクラスであらかじめ定義されている算術演算用メソッドの1つである**add()**を使えばよい

例

```
PVector v1, v2, v3;

v1 = new PVector(10, 20);
v2 = new PVector(30, 40);
v3 = new PVector();
v2.add(v1);
v3.add(v2);
println("v2 = " + v2);
println("v3 = " + v3);
```

```
v2 = [ 40.0, 60.0, 0.0 ]
v3 = [ 40.0, 60.0, 0.0 ]
```

PVectorは3つの成分を持つ．2次元の場合，x成分とy成分のみを使い，z成分の値は0として扱う

staticメソッド

- 2つのベクトルを加算した結果を第3のベクトルに代入しようとする場合，次のように書くことはできない

例

```
PVector v = new PVector();
PVector u = new PVector(4, 5);
PVector w = v.add(u);
println("v = " + v);
println("u = " + u);
println("w = " + w);
```

```
v = [ 4.0, 5.0, 0.0 ]
u = [ 4.0, 5.0, 0.0 ]
w = [ 4.0, 5.0, 0.0 ]
```

vの成分が変更されてしまう!

- PVectorクラス自体が持つメソッド(staticメソッド)を使えばよい
  - 使い方: クラス名.メソッド名(…)

例

```
PVector v = new PVector();
PVector u = new PVector(4, 5);
PVector w = PVector.add(v, u);
println("v = " + v);
println("u = " + u);
println("w = " + w);
```

```
v = [ 0.0, 0.0, 0.0 ]
u = [ 4.0, 5.0, 0.0 ]
w = [ 4.0, 5.0, 0.0 ]
```

PVectorクラスのメソッド

メソッド名	機能
add()	ベクトル加算
sub()	ベクトル減算
mult()	乗算によるベクトルのスケーリング
div()	除算によるベクトルのスケーリング
mag()	ベクトルの大きさ
normalize()	ベクトルの正規化
limit()	ベクトルの大きさを制限
heading()	ベクトルの方向
rotate()	ベクトルの回転
lerp()	ベクトルの線形補間
dist()	ベクトル間の距離
angleBetween()	ベクトル間の角度
dot()	ベクトルの内積
cross()	ベクトルの外積
random2D()	ランダムな2Dベクトル
random3D()	ランダムな3Dベクトル

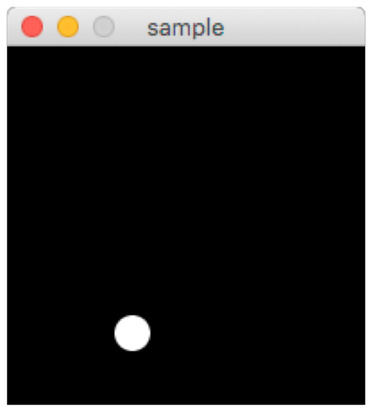
static版が用意されているメソッドもある
(リファレンス参照)

等速直線運動(2)

等速直線運動(1)のスケッチをPVectorクラスを使って書き直す

例

```
PVector location;
PVector velocity;
float ballDiameter = 20.0;
float ballRadius = ballDiameter / 2;

void setup() {
  size(200, 200);
  location = new PVector(ballRadius, height / 2);
  velocity = new PVector(3, 3);
}
```




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

  // draw the ball
  fill(255);
  noStroke();
  ellipse(location.x, location.y, ballDiameter, ballDiameter);

  // update the location of the ball
  location.add(velocity);

  // bouncing at the horizontal edges of the screen
  if (location.x < ballRadius ||
      location.x > width - ballRadius) {
    velocity.x *= -1; // velocity.x = -velocity.x;
  }

  // bouncing at the vertical edges of the screen
  if (location.y < ballRadius ||
      location.y > height - ballRadius) {
    velocity.y *= -1; // velocity.y = -velocity.y;
  }
}
```

位置に速度を加算

加速度

- 速度に変化をもたせることで現実の現象に近い運動になる
- 位置の変化は「位置+速度」で表された．同様に，速度の変化は「速度+加速度」で表現可能
  - $\overrightarrow{v_{n+1}} = \overrightarrow{v_n} + \overrightarrow{a_n}$
- **位置を更新する前に速度を更新**しておく

例

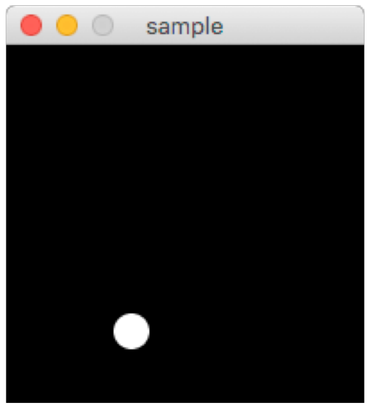
```
velocity.add(acceleration); // 追加
location.add(velocity);
```

等加速度運動

例

```
PVector location;
PVector velocity;
PVector acceleration;
float ballDiameter = 20.0;
float ballRadius = ballDiameter / 2;

void setup() {
  size(200, 200);
  location = new PVector(ballRadius, height / 2);
  velocity = new PVector(3, 3);
  acceleration = new PVector(0, 0.1);
}
```

下方向の加速度
（「重力」を表現）




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

  // draw the ball
  fill(255);
  noStroke();
  ellipse(location.x, location.y, ballDiameter, ballDiameter);

 // update the velocity of the ball
  velocity.add(acceleration);

  // update the location of the ball
  location.add(velocity);

  // bouncing at the horizontal edges of the screen
  if (location.x < ballRadius ||
      location.x > width - ballRadius) {
    velocity.x *= -1; // velocity.x = -velocity.x;
  }

  // bouncing at the vertical edges of the screen
  if (location.y < ballRadius ||
      location.y > height - ballRadius) {
    velocity.y *= -1; // velocity.y = -velocity.y;
  }
}
```

速度に加速度を加算