

デザインとプログラミング

関数、オブジェクトの利用法

山辺真幸 (masaki@allianceport.jp)

関数について

関数について

数学の関数は

$$y = f(x) = 2 \times x$$

$$x = 0 \quad \rightarrow \quad y = 0$$

$$x = 1 \quad \rightarrow \quad y = 2$$

$$x = 2 \quad \rightarrow \quad y = 4$$

xを関数に入力すると、結果はyとして得られる

関数 $f(x)$ は xを2倍して結果を返している。

関数について

Processingで同じことを実行するには

```
float y;
```

```
void setup(){  
  y = func(10); // 「func」という関数に10を入力すると y に20が代入される。  
  println(y);  
}
```

```
float func(float x){ // 「func」という関数の機能を定義している。  
  return x * 2;  
}
```

関数の定義の仕方

結果の型	関数名	引数
float	func	(float x) {
	return	x * 2;
}		返す結果

関数の定義の練習1

2つの値の平均を返す関数を作ってみよう

```
float y;
```

```
void setup(){  
    y = avg(50.0, 100.0); // y に 平均値が入るように関数「avg」を定義する  
    println(y);  
}
```



関数の定義の練習1

2つの値の平均を返す関数を作ってみよう

```
float y;  
  
void setup(){  
    y = avg(50.0, 100.0); // y に 平均値が入るように関数「avg」を定義する  
    println(y);  
}  
  
float avg(float a, float b){  
    return (a + b)/2;  
}
```

関数の定義の練習2

現在の秒数から色を返す関数を作ってみよう。

```
int sec;  
color cl;
```

```
void setup(){
  size(400,400);
  noStroke();
  colorMode(RGB,60);
}
```

```
void draw(){
    sec = second();
    cl = timeToColor(sec);
    fill(cl);
    rect(0, 0, width, height);
}
```

//secには現在の秒数 (0 … 60) が入る

```
color timeToColor(int s){
```


関数の定義の練習2

現在の秒数から色を返す関数を作ってみよう。

```
int sec;  
color cl;  
  
void setup(){  
  size(400,400);  
  noStroke();  
  colorMode(RGB,60);  
}  
  
void draw(){  
  sec = second();  
  cl = timeToColor(sec);  
  fill(cl);  
  rect(0, 0, width, height);  
}  
  
color timeToColor(int s){  
  float r = float(s);  
  float g = float(60 - s);  
  return color(r, g, 0);  
}
```

関数の定義の仕方2

結果を何も返さない関数も定義できる。

```
void setup(){  
    avg(50.0, 100.0);  
}
```

//結果を代入しない

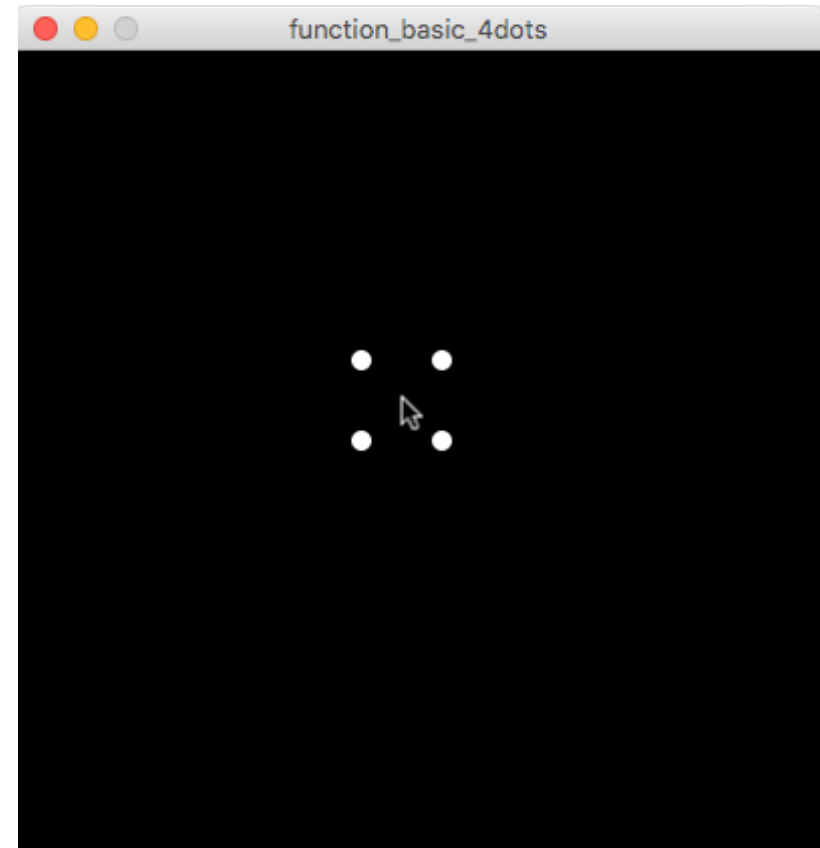
```
void avg(float a, float b){  
    float result = (a + b)/2;  
    println(result);  
}
```

//結果の型はvoid（何も無いの意味）

関数の定義の練習3

マウスの周りに4つのドットを描く関数を作ってみよう

```
void setup(){  
  size(400,400);  
  noStroke();  
}  
  
void draw(){  
  background(0);  
  fill(255);  
  draw4dots(mouseX, mouseY);  
}
```



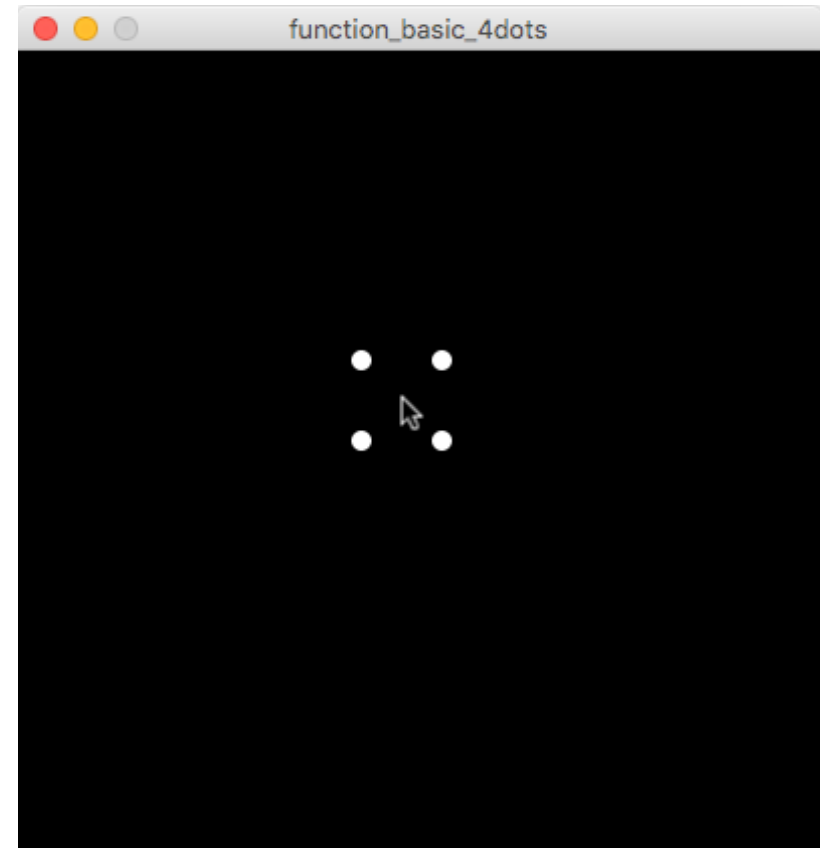
関数の定義の練習3

マウスの周りに4つのドットを描く関数を作ってみよう

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

void draw(){
  background(0);
  fill(255);
  draw4dots(mouseX, mouseY);
}

void draw4dots(int x, int y){
  ellipse(x-20, y-20, 10, 10);
  ellipse(x-20, y+20, 10, 10);
  ellipse(x+20, y-20, 10, 10);
  ellipse(x+20, y+20, 10, 10);
}
```



グローバル変数とローカル変数

グローバル変数=どの関数からでもアクセスできる。

ローカル変数=宣言した関数の中でのみアクセスできる。

グローバル変数の例

```
int global = 100;

void setup(){
  println(global); // 100
  func();
  print(global); // 110
}

void func(){
  global = global + 10;
}
```

ローカル変数の例

```
void setup(){
  int local = 100;
  println(local);
  func();
  print(local);
}

void func(){
  local = local + 10;
}
```

「local」にアクセスできない（宣言されていない）のでエラーになる。

オブジェクト指向プログラミング

オブジェクト指向とは

プログラミングする時に、これまでは変数や関数を組み合わせて手順を考えて実行させてきた。例えば、画面にひとつの円を描くときは、

プログラム

```
float x = 133.0;
float y = 150.0;
float d = 50.0;

void setup(){
  size(300,300);
  noStroke();
}

void draw(){
  background(0);
  ellipse(x, y, d, d);
}
```

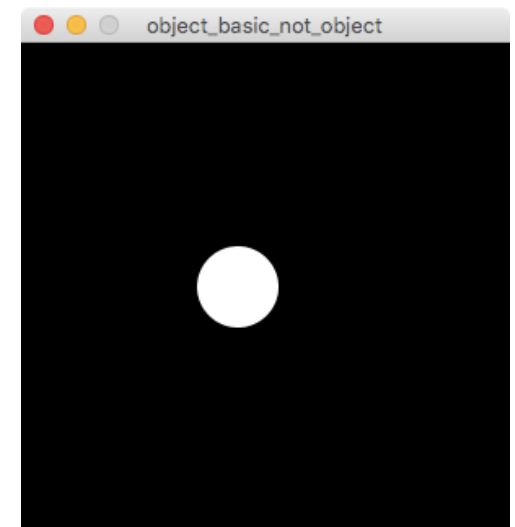
手順

変数x、y、dを宣言する。



座標(x,y)に直径dの円を描く

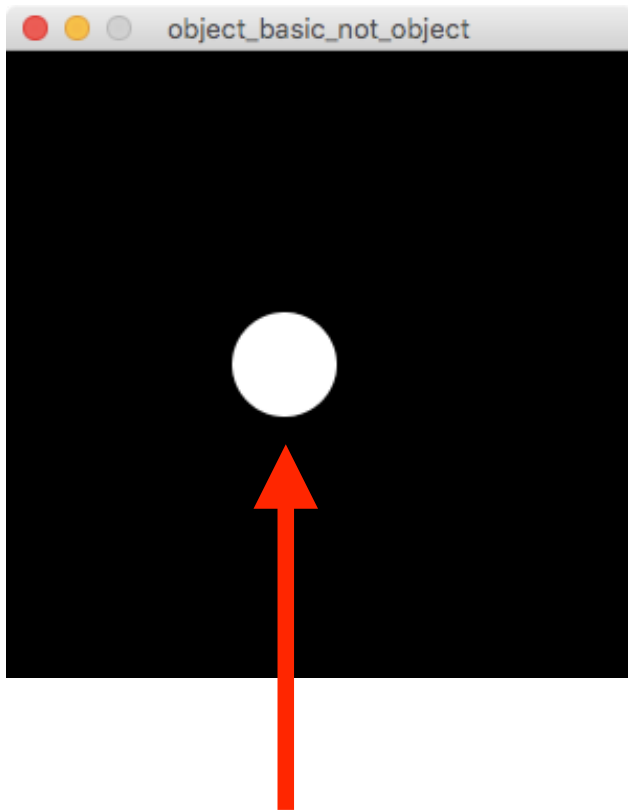
結果



円を描くための手順を組み立てている

オブジェクト指向とは

「円を描く」ではなく、「粒子」という概念を中心に考え直す。



「粒子」という「物」

「粒子」は情報を持っている

現在地 (x、y) 直径d

「粒子」は機能を持っている

現在地に円を描く

オブジェクト指向とは

クラス=物の概念をプログラムで表現したもの。

クラスは、変数と関数で構成されている。

```
class Particle{
```

```
    float x;
```

```
    float y;
```

```
    float d;
```

```
    void display(){
```

```
        ellipse(x, y, d, d);
```

```
    }
```

```
}
```

クラス名

情報（変数）

機能（関数）

Particle

現在地(x,y) 直径d

現在地に自分を描く
関数 display()

ここまでで、「粒子」という物の概念を定義できた。

クラスはいわば物（オブジェクト）の設計図であり、実体を伴わない。

オブジェクト指向とは、物を実体化させ、物に情報を与え、機能させることでプログラミングする手法のことである。

クラスからオブジェクトを生成して使用するまで

```
Particle p; //オブジェクト宣言 Particleクラスのオブジェクトpを宣言

void setup(){
  size(300,300);
  noStroke();
  p = new Particle(); // pに Particleクラスからオブジェクトを生成（コンストラクト）。
  p.x = 133.0; // pの変数x,y,dに値を入力
  p.y = 150.0; // (pの後にドットをつけて変数にアクセスする。)
  p.d = 50.0;
}

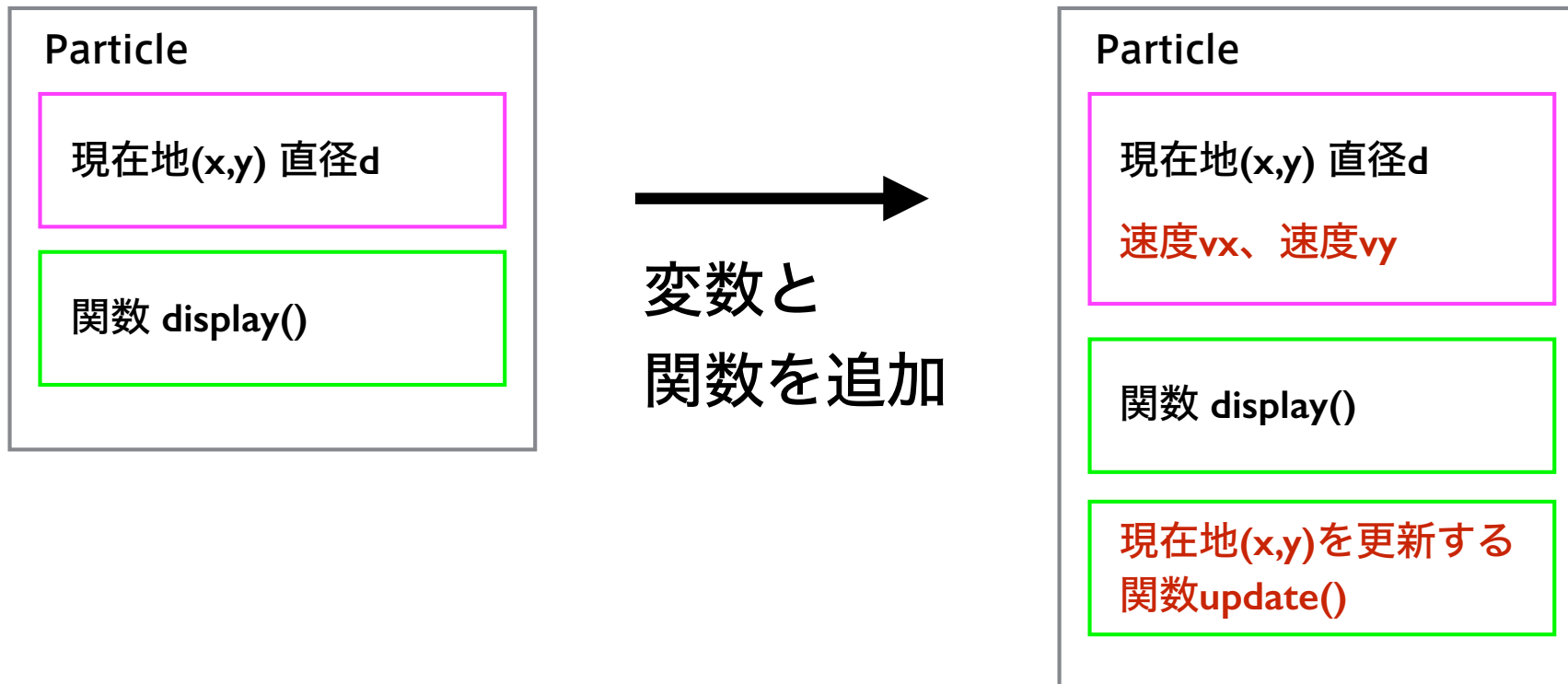
void draw(){
  background(0); //pの関数display()を実行させ、現在地に描画させる
  p.display(); // (pの後にドットをつけて関数を実行する。)
}

class Particle{
  float x;
  float y;
  float d;
  void display(){
    ellipse(x, y, d, d);
  }
}
```

Particleクラスを拡張する

粒子が動くようにするにはどうしたらよいか？

まず、クラスを見直す。



Particleクラスを拡張する

クラスに、変数vx、vy、関数update()を追加

```
class Particle{  
    float x;  
    float y;  
    float d;  
    float vx;  
    float vy;  
    void display(){  
        ellipse(x, y, d, d);  
    }  
    void update(){  
        x = x + vx;  
        y = y + vy;  
    }  
}
```

Particle

現在地(x,y) 直径d

速度vx、速度vy

関数 display()

現在地(x,y)を更新する
関数update()

Particleクラスを拡張する

生成したオブジェクトに追加した速度に値を代入、追加した関数を実行

```
void setup(){
  size(300,300);
  noStroke();
  p = new Particle();
  p.x = 133.0;
  p.y = 150.0;
  p.d = 50.0;
  p.vx = 5.0;           // 追加したvxに値を代入
  p.vy = 2.0;           // 追加したvyに値を代入
}

void draw(){
  background(0);
  p.update();           // 追加したupdate()を実行
  p.display();
}
```

ここまでのコード全体

```
Particle p;

void setup(){
  size(300,300);
  noStroke();
  p = new Particle();
  p.x = 133.0;
  p.y = 150.0;
  p.d = 50.0;
  p.vx = 5.0;
  p.vy = 2.0;
}

void draw(){
  background(0);
  p.update();
  p.display();
}
```

```
class Particle{
  float x;
  float y;
  float d;
  float vx;
  float vy;
  void display(){
    ellipse(x, y, d, d);
  }
  void update(){
    x = x + vx;
    y = y + vy;
  }
}
```

さらにクラスを改良する

```
Particle p;  
  
void setup(){  
  size(300,300);  
  noStroke();  
  p = new Particle();  
  p.x = 133.0;  
  p.y = 150.0;  
  p.d = 50.0;  
  p.vx = 5.0;  
  p.vy = 2.0;  
}  
      この部分（初期値の設定）を  
      スッキリさせたい  
  
void draw(){  
  background(0);  
  p.update();  
  p.display();  
}
```

```
class Particle{  
  float x;  
  float y;  
  float d;  
  float vx;  
  float vy;  
  void display(){  
    ellipse(x, y, d, d);  
  }  
  void update(){  
    x = x + vx;  
    y = y + vy;  
  }  
}
```

さらにクラスを改良する

```
p = new Particle();  
p.x = 133.0;  
p.y = 150.0;  
p.d = 50.0;  
p.vx = 5.0;  
p.vy = 2.0;
```



クラスからオブジェクトを生成する時に、変数も渡してしまいたい

```
p = new Particle(133.0, 150.0, 50.0, 5.0, 2.0);
```


さらにクラスを改良する

クラスにコンストラクタを追加する

コンストラクタはオブジェクトを生成するときに実行されるコード

Particle

現在地(x,y) 直径d
速度vx、速度vy

コンストラクタ

display()

update()

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

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

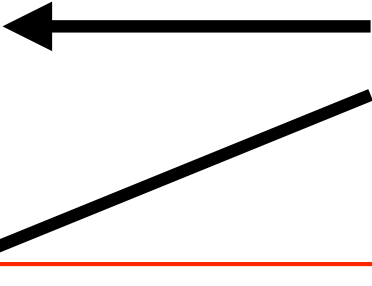
```
    void display(){  
        ellipse(x, y, d, d);  
    }  
    void update(){  
        x = x + vx;  
        y = y + vy;  
    }  
}
```

コンストラクタは特殊な関数（のようなもの）

- ・ 結果の型はない（voidとか）
- ・ パラメータは任意に定義できる
- ・ 必ずクラス名と同じ名前

```
class Particle{  
    float x;  
    float y;  
    float d;  
    float vx;  
    float vy;  
  
    Particle(float _x, float _y, float _d, float _vx, float _vy){  
        x = _x;  
        y = _y;  
        d = _d;  
        vx = _vx;  
        vy = _vy;  
    }  
  
    void display(){  
        ellipse(x, y, d, d);  
    }  
    void update(){  
        x = x + vx;  
        y = y + vy;  
    }  
}
```

同じ



ここまでのコード全体

```
Particle p;

void setup(){
  size(300,300);
  noStroke();
  p = new Particle(133.0, 150.0, 50.0, 5.0, 2.0);
}

void draw(){
  background(0);
  p.update();
  p.display();
}

class Particle{
  float x;
  float y;
  float d;
  float vx;
  float vy;

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

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

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

練習1

particleオブジェクトを増やして、画面上に大きさの違う2つ以上のparticleを表示させてみよう。

ヒント

- ・ オブジェクトの宣言をひとつ追加 Particle p2
- ・ p2の初期値を与える
- ・ p2のupdate()を実行する
- ・ p2のdisplay()を実行する

練習2

さらに改良して、粒子に色をつける。

ヒント

- ・ データに色colを加える。
- ・ display関数の中でfillを使う。

Particle

現在地(x,y) 直径d
速度vx、速度vy、色col

コンストラクタ

display()

update()

練習3

ここまでのコードを改良して、上下左右で衝突判定と跳ね返りを実行する。

ヒント

- ・ クラスの、update関数の中に衝突判定と跳ね返りを定義する。