

Processingのインストール

<http://processing.org/download/index.html>
このサイトから最新のバージョンがダウンロード可能

記号の読み方

/ スラッシュ	' シングルクオート	\ バックスラッシュ (日本語環境では¥で表示)	; セミコロン
// ダブルスラッシュ	" ダブルクオート	~ チルダ	{ 左中カッコ
* アスタリスク	& アンド	@ アットマーク	} 右中カッコ
, カンマ	. ピリオド	: コロン	< 大なり
； セミコロン	! エクスクラメーション	バー	> 小なり

プログラム実行方法

3通りのプログラム実行方法があります

○テストで実行する

画面左上の「RUN」ボタンを押す。

○インターネット上で動作するアプリケーションを制作する

「Export」ボタンを押すと保存フォルダ内に「applet」フォルダが作られる。その中のファイルをアップロードすることにより、インターネット上で閲覧することができる。

○単体で動作するアプリケーションを制作する

メニュー「file」から「Export Application」を選択すると、ファイルの保存フォルダ内に「application.linux」「application.macosx」「application.windows」というフォルダが制作される。このフォルダ内のファイルはmac os, windows, linuxのOSに最適化されており、Processingのソフトがなくても単体で動作させることができる。

基本命令

1. 画面のサイズを指定する

size(横のピクセル数, 縦のピクセル数);

例 / 縦に480ピクセル、横に480ピクセルの画面サイズを設定する

```
size(640, 480);
```

2. 点を描く

point(x座標, y座標);

例 / 座標(50, 10)に点を描く

```
point(50, 10);
```

3.コメントをつける

// 以下任意の文章

例

```
//テスト
```

ダブルスラッシュ以下はプログラムに影響しない。制作者のメモや一時的にプログラムを無効にする際に使用

4.直線を描く

line(始点の x 座標, 始点の y 座標, 終点の x 座標, 終点の y 座標);

例 / 座標(200,0) から座標(0,200)まで直線を引く

```
line( 200, 0, 0, 200);
```

5.長方形を描く

rect(x座標, y座標, 幅, 高さ);

例 / 座標(100,10)に幅20高さ30の長方形を描く

```
rect( 100, 10, 20, 30);
```

6.円を描く

ellipse(x座標, y座標, 横の直径, 縦の直径);

例 / 座標(100,10)に横20縦30の楕円を描く

```
ellipse( 100, 10, 20, 30);
```

7.三角形を描く

triangle(x1, y1, x2, y2, x3, y3);

例 / 座標(100,10)と(50,100)と(100,50)を頂点とする三角形を描く

```
triangle(10, 10, 50, 100, 100, 50);
```

8.四角形を描く

quad (x1, y1, x2, y2, x3, y3, x4, y4);

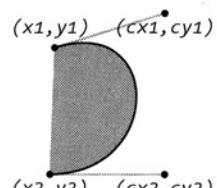
例 / 座標(10,10)と(70,5)と(90,50)と(10,100)を頂点とする四角形を描く

```
quad(10, 10, 70, 5, 90, 50, 10, 100);
```

9.ベジエ曲線を描く

bezier (x1, y1, cx1, cy1, cx2, cy2, x2, y2);

例 / 座標(32,20)と(80,5)のコントロールポイント、
座標(80,75)と(30,75)のコントロールポイントを結ぶベジエ曲線を描く



```
bezier (32, 20, 80, 5, 80, 75, 30, 75);
```

10. 背景に色をつける

background(明度);

または

background(red, green, blue);

数値は0から255の値の範囲内で指定する

例

```
background(0); //黒色背景
```

```
background(255,165,0); //オレンジ色
```

```
background(255); //白色背景
```

色の指定方法について

processingの標準設定では色を0から255までの数値で指定します。

メニューバーの「Tools→ColorSelector」を使用し、色の選択を行って下さい。

11. 線に色をつける

stroke(明度);

または

stroke(red, green, blue);

数値は0から255の値の範囲内で指定する

例1 / オレンジ色の線を描く

```
stroke(255,165,0); //オレンジ色  
line(10, 10, 60, 50);
```

例2 / オレンジ色と緑色の線を描く

```
stroke(255,165,0); //オレンジ色  
line(10, 10, 60, 50);  
stroke(0,165,100); //緑色色  
line(50, 50, 10, 50);
```

12. 線の太さを変える

strokeWeight(線の太さ);

線の太さはピクセル数で指定

例/ 5ピクセルの太さで直線を描く

```
strokeWeight(5);  
line(10, 10, 60, 50);
```

13. 線を描かない

noStroke();

例/ 輪郭線を描かずに円を描く

```
noStroke();  
ellipse(40, 40, 50, 50);
```

14. 色を塗りつぶす

fill(明度);

または

fill(red, green, blue);

例/ オレンジ色で塗りつぶした円を描く

```
fill(255,165,0);
ellipse(40, 40, 50, 50);
```

15. 色を塗りつぶさない (輪郭線のみ描く)

noFill();

例/ 輪郭線のみの円を描く

```
noFill();
ellipse(40, 40, 50, 50);
```

16. 半透明にする

fill(red, green, blue, 透明度);

透明度の項目に0から255までの透明度を設定する。数値が小さいほど透明度が高い。

透明度の指定はfillの他、strokeなど他の命令でも指定可能。

例：一方の円を半透明にして重ねる

```
fill( 0, 88, 255);
ellipse(40, 40, 50,50);
fill( 230, 188, 0, 70);
ellipse(80, 80, 100,100);
```

17. 図形のエッジをスムーズにする

smooth();

例：スムーズの有無を二種類の円で比較する

```
smooth();
ellipse(40, 40, 50,50);
noSmooth();
ellipse(60, 60, 50,50);
```

基本命令その2（複数の行で表現する命令）

18. 多角形を描く（連続した線を描く）

```
beginShape();  
vertex(x座標, y座標);  
endShape();
```

曲線を描くためにはvertex命令をbeginShapeとendShape命令の間に記述する必要があります。

```
size(200, 200);  
beginShape();  
vertex( 20, 20);  
vertex( 60, 60);  
vertex( 180, 20);  
vertex( 20, 180);  
vertex( 20, 20);  
endShape();
```

19. 連続した曲線を描く

```
beginShape();  
curveVertex(x座標, y座標);  
endShape();
```

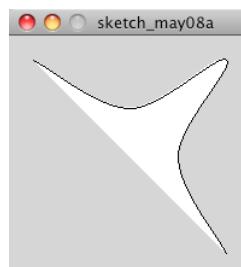
曲線を描くためにはcurveVertex命令をbeginShapeとendShape命令の間に記述する必要があります。

Vertex内の引数を通る滑らかな曲線を描きます。

また、始点と終点の座標は二回ずつ指定します。

例 / キャンバスのサイズを横200、縦200に設定し、各座標を通過する曲線を作成する

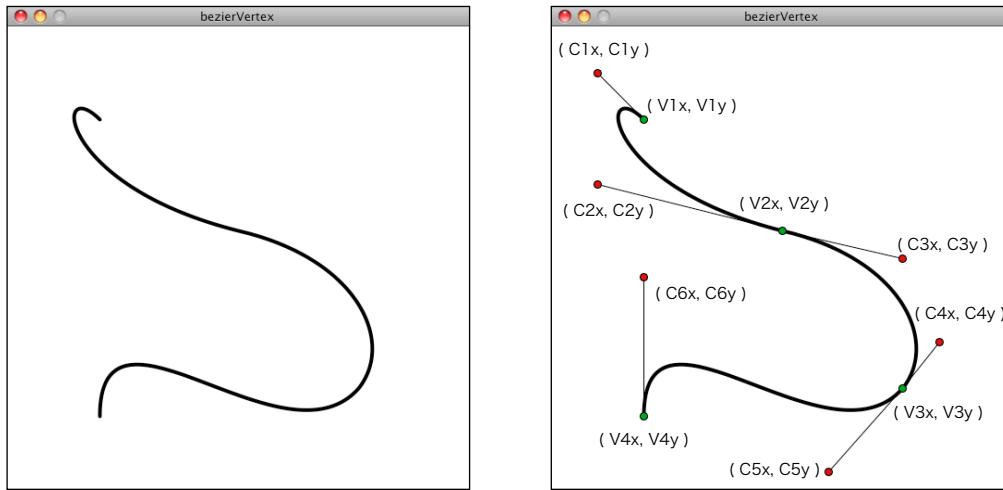
```
size(200, 200);  
beginShape();  
curveVertex( 20, 20);  
curveVertex( 20, 20);  
curveVertex( 100, 60);  
curveVertex( 180, 20);  
curveVertex( 140, 100);  
curveVertex( 180, 180);  
curveVertex( 180, 180);  
endShape();
```



20. 連続したベジエ曲線を描く

```
beginShape();  
bezierVertex(x座標, y座標);  
endShape();
```

ベジエ曲線を連続して描くためにはbezierVertex命令をbeginShapeとendShape命令の間に記述します。



```
size(500,500);  
background(255);  
noFill();  
beginShape();  
vertex(100, 100);  
bezierVertex(50, 50, 50, 170, 250, 220);  
bezierVertex(380, 250, 420, 340, 380, 390);  
bezierVertex(300, 480, 100, 270, 100, 420);  
endShape();
```

```
beginShape();  
vertex(V1x, V1y);  
bezierVertex(C1x, C1y, C2x, C2y, V2x, V2y);  
bezierVertex(C3x, C3y, C4x, C4y, V3x, V3y);  
bezierVertex(C5x, C5y, C6x, C6y, V4x, V4y);  
endShape();
```



RGB以外の色彩指定方法

色指定の際、標準ではRGB (Red,Green,Blue)で指定を行うが、HSB(色相Hue・彩度Saturation・輝度Brightness) での指定も可能

○HSBでの色指定

```
colorMode(HSB);
```

この命令を宣言後、色指定は（色相Hue、彩度Saturation、輝度Brightness）の数値で指定を行う

```
colorMode(HSB);  
fill( 0, 188, 100);  
ellipse(80, 80, 100,100);
```

○RGBでの色指定

```
colorMode(RGB);
```

この命令を宣言後、色指定は（ Red,Green,Blue）の数値の指定を行う

```
colorMode(RGB);  
fill( 0, 188, 100);  
ellipse(80, 80, 100,100);
```

○色指定の範囲の変更

```
colorMode(RGB, 色の範囲);
```

標準では色の範囲を0から255までの値で指定するが、任意の色の範囲に変更することも可能。

たとえば、colorMode(RGB, 100);とすると0から100までの色指定に変更することが可能である。

変数・計算

1. 計算する

演算子を使用して計算する

例 / 引数内の計算

```
rect(200-100, 20*5, 120-20, 120/2);
```

計算記号について

+	足す (和)
-	引く (差)
*	掛ける (積)
/	割る (商)
%	割り算の余り (剰余)

2. 変数

変数とは数値（データ）を納める入れ物である

①変数を宣言する（代入も同時に使う）

例

int abc = 10;	変数abcを作り10を代入する
float b = 10.4;	変数bを作り、10.4を代入する
color c1 = color(30,60,99);	変数c1を作り、色の値(30,60,90)を代入する
strings s = "tsukuba";	変数sを作り文字を代入する

変数を使用する前に、必ず定義すること

変数の名称は英数字であれば任意に設定出来る。大文字小文字は区別される。

②変数へ代入する

例

b = 10;	変数bに10を代入する
b = 10*2;	変数bに 10×2 を計算した値を代入
b = b + 1;	変数bの今の中身に1を足して代入し直す

備考 特殊な計算式

```
x++; (x = x + 1; と同義)  
x--; (x = x - 1; と同義)  
x += 10; (x = x + 10; と同義)  
x -= 10; (x = x - 10; と同義)
```

③変数を利用する

例

rect(50, 50, a, a);	座標(50,50)に一边がaの正方形を描く
fill(c1);	変数c1のカラーで塗りつぶす

③小数と整数の相互変換

Processingでは小数と整数は明確に区別されているので扱いに注意すること。小数と整数の変数は相互に代入出来ないので、下記の手順が必要となります

たとえば、

```
int a = 4;  
float def = 10.4;  
a = def +1;
```

ではエラーになります。そこで

```
int a = 4;  
float def = 10.4;  
a = (int)(def +1);
```

このようにすれば、aには整数に変換された値が代入されます。

3. システム変数

width

→画面の横幅

height

→画面の縦幅

上記の変数に数値を代入することはできません。数値を利用することだけが可能です。

繰り返し

for文 指定した回数繰り返す

```
for( 繰り返し変数をつくる ; ループを行う条件 ; 変数の変化 ){
    繰り返しの処理の内容
}
```

例 / {}の間のline文が10回繰り返される

```
for (int i=0; i<10 ; i=i+1){
    line ( i*4 , 10, i*4 + 40, 50); // ここの部分が繰り返される
}
```

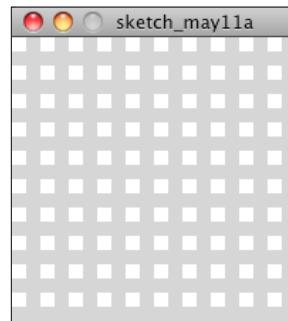
「変数iを0にして、毎回1を足しながら繰り返して、iが10になったら次に進む」

繰り返しを繰り返す

for文の中に、さらにfor文を入れることでさらに複雑な繰り返し処理が可能である

例 / {}の間のrect文が100回繰り返される

```
size(200,200);
noStroke();
for (int y=0; y<10 ; y++){
    for (int x=0; x<10 ; x++){
        rect(x*20, y*20, 10,10);
    }
}
```



一定の条件になるまで繰り返す-while文

```
while(ループする為の条件){
    繰り返しの処理の内容
}
```

例 / {}の間のrect文が15回繰り返される

```
int a = 0;
while( a < 15){
    a++;
    rect( a*5, a*5, 10, 10 );
}
```

条件式

条件によって処理の内容を変える- if ~ else 文

```
if( 条件式 ){
    条件式がtrueの場合の処理
} else{
    条件式がfalseの場合の処理
}
```

備考: elseは省略可能

例 / for文によって10本の線を引くが、前半の5本と後半の5本で線の太さを変える

```
for (int i=0; i<10 ; i=i+1){

    if ( i < 5){
        strokeWeight(1); //iが5以下の時
    }else{
        strokeWeight(1.3); //iが5以上の時
    }
    line ( i*8 , 10, i*8 + 40, 50);

}
```

条件の記述方法

```
if( i > 5) {処理} //iが5より大きい場合
if( i < 5) {処理} //iが5より小さい場合

if( i >= 5) {処理} //iが5以上の場合
if( i <= 5) {処理} //iが5以上場合

if( i == 5) {処理} //iが5場合
```

条件分岐にはif文の他にswitch case文がある

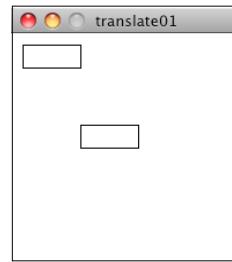
座標命令

1. translate 座標を移動させる

translate(Xの移動量、Yの移動量)；

例) 同じ座標にrect命令で矩形を描くが、間にtranslate命令を入れることで原点を移動することが可能

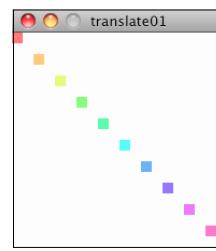
```
size(200,200);
rect(10,10,50,20);
translate(50,70);
rect(10,10,50,20);
```



例) forを使用し複数回translateを実行する

```
size(200,200);
colorMode(HSB, 100);
background(99);
noStroke();

for(int i = 0; i < 10 ; i++){
    fill(i*10, 60, 99);
    rect(0,0,10,10);
    translate(20, 20);
}
```



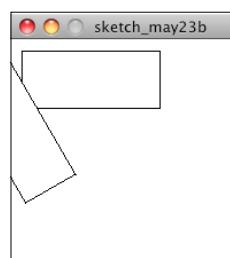
2. rotate 座標を回転させる

rotate(角度のラジアン値)； もしくは rotate(radians(角度))；

ラジアンと角度の関係: 2π ラジアン = 360°

例) rotateを使用し、矩形を回転させる

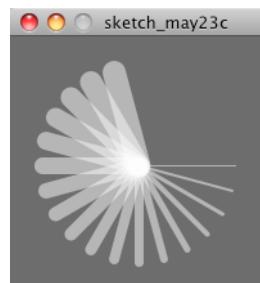
```
size(200,200);
rect(10,10,120,50);
rotate(radians(60));
rect(10,10,120,50);
```



例) rotateとforを使用し、線を回転と同時に線を太くしてゆく（原点は中心で固定）

```
void setup(){
    size(200,200);
    smooth();
    noStroke();
    noLoop();
}

void draw(){
    background(90);
    stroke(255, 120);
    translate(100,100);
    for( int i = 1; i < 19; i++){
        strokeWeight(i);
        line(0 ,0 ,75, 0);
        rotate(radians(15));
    }
}
```



例) 原点を移動させながら回転させる

```
size(200,200);
colorMode(HSB, 120);
background(120);
noStroke();

int angle = 30;
int margin = 40;

translate(120, 30);

for(int i = 0; i < 12; i++){
    fill(i*10, 100, 119, 60);
    rect(0, 0, 30, 30);

    rotate(radians(angle));
    translate(margin, 0);
}
```



3. pushMatrix, popMatrix 座標を戻す

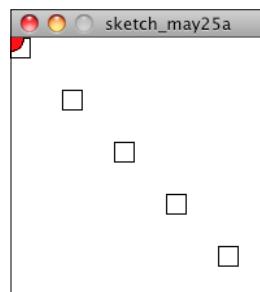
pushMatrix(); 座標をセーブする
popMatrix(); セーブした座標に戻る

例) pushMatrixで現在の座標をセーブし、popMatrixでセーブ時の座標に戻す

```
size(200,200);
background(255);
noFill();
pushMatrix();

for(int i = 0; i < 5; i++){
    rect(0,0,15,15);
    translate(40,40);
}

popMatrix();
fill(255,0,0);
ellipse(0,0,20,20);
```



例) pushMatrix、popMatrixの応用例

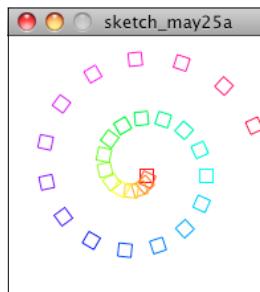
```
size(200,200);
colorMode(HSB, 300);
background(300);
noFill();
smooth();

int angle = 24;
int x = 3;

translate(width/2, height/2);

for(int i = 0; i < 30 ; i++){
    stroke(i*10,299,299);

    pushMatrix();
    rotate(radians(i*angle));
    translate(i*x, 0);
    rect(0,0,10,10);
    popMatrix();
}
```



ランダム

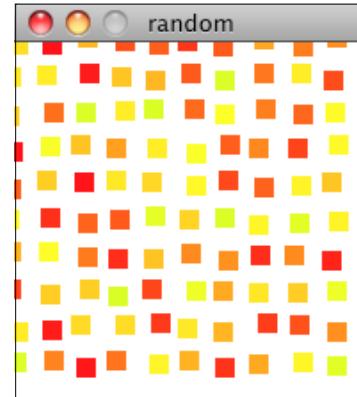
random(最大値); 0から最大値までの範囲で乱数をつくる
random(最小値,最大値); 最小値から最大値までの範囲で乱数をつくる

例) randomの応用例

```
size(200, 200);
colorMode(HSB, 255);
background(255);
noStroke();
rectMode(CENTER);

int range = 3;

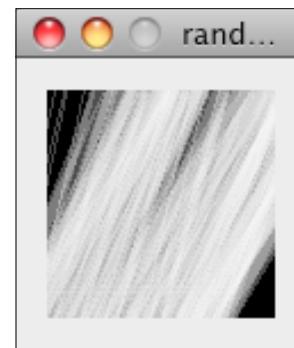
for(int y=0 ; y<10 ; y++){
    for(int x=0; x<10 ; x++){
        fill(random(50), 255, 255);
        rect(x*20 + random(-range, range),
             y*20 + random(-range, range), 10,10);
    }
}
```



例) randomの応用例

```
background(0);
stroke(255, 60);
float r;
float offset;

for (int i = 0; i < 100; i++){
    r = random(10);
    strokeWeight(r);
    offset = r * 5.0;
    line(i-20, 100, i+offset, 0);
}
```



例) randomの応用例

```
size(200, 200);
colorMode(HSB, 100);
background(100);
noFill();
smooth();

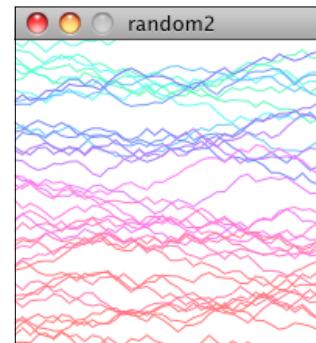
int range_color = 10;
int range_y = 5;
float fluc_color = 50;
float fluc_y;

for(int y =0; y < height ; y+=5){
    fluc_color +=random(-range_color,
                         range_color);
    stroke(fluc_color, 60, 99);
    fluc_y = 0;

    beginShape();

    for(int x = 0; x<=width; x +=5){
        fluc_y +=random(-range_y, range_y);
        vertex(x, y+ fluc_y);
    }

    endShape();
}
```



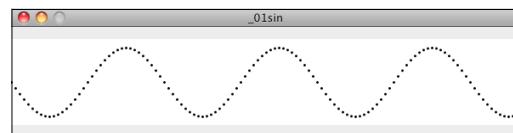
数式による描写

sin(角度のラジアン値); もしくは sin(radians(角度));

例) sinの例1

```
size(600, 100);
background(255);
smooth();
strokeWeight(3);
float angle = 0.0;

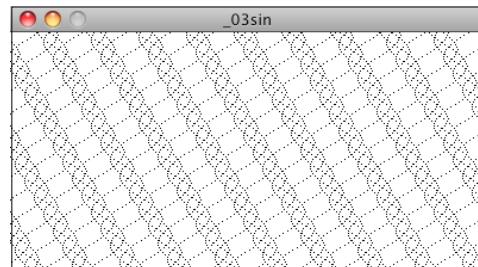
for (int x = 0; x <= width; x +=5){
  float y = 50 + (sin(radians(angle)) * 40.0);
  point(x,y);
  angle += 10;
}
```



例) sinの例2

```
size(400, 200);
background(255);
fill(255, 20);
noFill();
float angle = 0.0;
float scaleVal = 18.0;
float angleInc = 10;

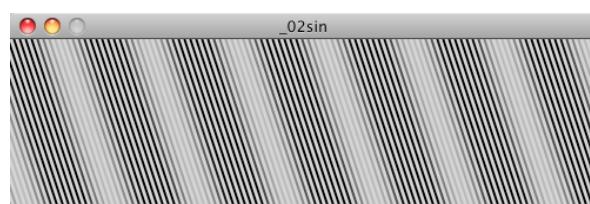
for (int offset = -10; offset < width+10; offset +=8){
  for(int y = 0; y <= height; y +=2){
    float x = offset + (sin(radians(angle))* scaleVal);
    noStroke();
    stroke(0);
    point(x, y);
    angle += angleInc;
  }
}
```



例) sinの例3

```
size(500, 150);
smooth();
strokeWeight(2);
float angle = 0.0;
float scaleVal = 126.0;
float angleInc = 0.42;

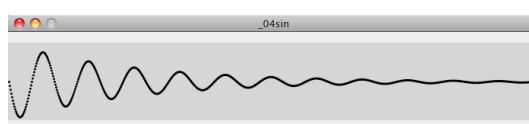
for (int x = -52; x <= width; x +=5){
  float y = 50 + (sin(angle) * scaleVal);
  stroke(y);
  line(x, 0, x+50, height);
  angle += angleInc;
}
```



例) sinの例4

```
size(700, 100);
smooth();
float angle = 0.0;
float amplitude = 50;
float x = 0, y = 0;
float xSpeed = 1;
float frequency = 6.0;
float damping = .994;
strokeWeight(3);

for (int i = 0; i<width; i+=xSpeed){
  x+=xSpeed;
  y = height/2 + sin(radians(angle))*amplitude;
  point(x,y);
  amplitude*=damping;
  angle+=frequency;
}
```



印刷用データの書き出し

例) PDFファイルへ静止画を描き出す (制作: 志原彩未)

```
import processing.pdf.*;

size(750,350, PDF, "kakidasi.pdf");
colorMode(RGB,100);
background(100,100,100);

float a=0.01;
noFill();
strokeWeight(a);

smooth();
stroke(random(50,100),10,10);

for(int i=0; i<100; i=i+1){

beginShape();

curveVertex(random(0,400),0);
curveVertex(random(0,400),0);
curveVertex(500,80);
curveVertex(random(0,80),350);
curveVertex(random(0,80),350);

curveVertex(random(680,750),0);
curveVertex(random(680,750),0);
curveVertex(100,270);
curveVertex(750,random(400,680));
curveVertex(750,random(400,680));

curveVertex(0,random(280,350));
curveVertex(0,random(280,350));
curveVertex(random(250,310),random(250,350));
curveVertex(random(480,680),350);
curveVertex(random(480,680),350);

curveVertex(750,random(280,350));
curveVertex(750,random(280,350));
curveVertex(random(250,310),random(250,350));
curveVertex(random(480,680),350);
curveVertex(random(480,680),350);

endShape();
}

exit();
```

1学期提出課題

「実社会で起きている状態や現象をprocessingを用い抽象的な静止画で表現しなさい」

講評会 7月1日(12:00~)

出力フォーマット：A3プリントアウト(画像の比率は自由)

提出枚数：2枚以上

参考サイト

<http://www.processing.org/>

<http://www.openprocessing.org/>

アニメーション

アニメーションの為のsetup関数とdraw関数

(参考書籍：Build with Processing 101～108ページ)

```
void setup(){
    //ここに初期化のプログラム
}

void draw(){
    //無限に繰り返すプログラム
}

frameRate(フレーム数); 一秒間の描写回数を決める (setup関数の中に記述すること)
```

[animation_sample1] 円を1秒間に10回ランダムな位置に描く

```
float radius;

void setup(){
    size(200,200);
    frameRate(10);
    noStroke();
    background(255);
}

void draw(){
    fill(random(255));
    radius = random(20);
    ellipse(random(200),random(200),radius,radius);
}
```

[animation_sample2] 円が動くアニメーション

```
int x = 0;

void setup(){
    size(200,200);
    frameRate(30);
}

void draw(){
    background(90);
    ellipse(x,100,20,20);
    x=x+2;
}
```

[animation_sample3] 円が動くアニメーション
(画面外に円が出ると位置がリセット)

```
int x = 0;

void setup(){
    size(200,200);
    frameRate(30);
}

void draw(){
    background(90);
    ellipse(x,100,20,20);
    x=x+2;
    if(x>width){
        x=0;
    }
}
```

[animation_sample4] 複数の円が動くアニメーション

```
int x1 = 0;
int x2 = 0;
int x3 = 0;

void setup(){
    size(200,200);
    frameRate(30);
}

void draw(){
    background(90);
    ellipse(x1,50,20,20);
    x1=x1+2;
    if(x1>width){
        x1=0;
    }
    ellipse(x2,100,20,20);
    x2=x2+1;
    if(x2>width){
        x2=0;
    }
    ellipse(x3,150,20,20);
    x3=x3+3;
    if(x3>width){
        x3=0;
    }
}
```

[animation_sample5] 円の半径の変化によるアニメーション

```
float d =20.0;
float speed = 1.0;

void setup(){
    size(100,100);
    frameRate(30);
    noStroke();
}

void draw(){
    background(0);
    fill(255,120);
    ellipse(0, 50, d, d);
    ellipse(100, 50, d, d);
    ellipse(50, 0, d, d);
    ellipse(50, 100, d, d);
    d = d + speed;
    if ((d > width) || (d < width/10)){
        speed = speed * -1;
    }
}
```

複雑なアニメーション

複雑なアニメーションと物理法則の表現

(参考書籍：Build with Processing 108～122ページ)

[animation_sample6] ポールの跳ね返り

```
float y = 50.0;
float speed = 1.0;
float radius = 15.0;

void setup(){
    size(100,100);
    smooth();
    frameRate(30);
    noStroke();
    ellipseMode(RADIUS);
}

void draw(){
    background(0);
    fill(255);
    ellipse(33, y, radius, radius);
    y = y + speed;
    if ((y > height-radius) || (y < radius)){
        speed = speed * -1;
    }
}
```

[animation_sample7] ポールの跳ね返り（2次元）

```
float x = 50.0;
float y = 50.0;
float speedX = 2.0;
float speedY = 3.4;
float radius = 15.0;

void setup(){
    size(100,100);
    smooth();
    frameRate(30);
    noStroke();
    ellipseMode(RADIUS);
}

void draw(){
    background(0);
    fill(255);
    ellipse(x, y, radius, radius);

    x = x + speedX;
    if ((x > width-radius) || (x < radius)){
        speedX = speedX * -1;
    }

    y = y + speedY;
    if ((y > height-radius) || (y < radius)){
        speedY = speedY * -1;
    }
}
```

[animation_sample8] 減速

```
float x = 0.0;
float y = 0.0;
float targetX = 70.0;
float targetY = 80.0;
float easing = 0.04;

void setup(){
    size(100,100);
    frameRate(30);
    smooth();
}

void draw(){
    background(0);
    float d = dist(x, y, targetX, targetY);

    if(d>1.0){
        x = x + (targetX - x) * easing;
        y = y + (targetY - y) * easing;
    }
    ellipse(x,y,20,20);
}
```

[animation_sample9] 重力表現1

```
float x = 100;
float y = 0.0;
float speed = 0;
float gravity = 0.1;

void setup(){
    size(200,200);
    frameRate(30);
    smooth();
}

void draw(){
    background(0);

    rectMode(CENTER);
    rect (x,y,10,10);

    y=y+speed;
    speed = speed + gravity;

    if(y>height){
        speed = speed * -0.85;
    }
}
```

[animation_sample10] 重力表現2

```

float x, y, w, h;
float speedX, speedY;
float gravity;

void setup(){
    size(400,400);
    frameRate(20);
    smooth();
    x = width/2;
    w = 20;
    h = w;
    speedX = 2;
    gravity = 0.7;
}

void draw(){
    background(0);
    rect(x,y,w,h);
    x= x + speedX;
    speedY = speedY + gravity;
    y = y + speedY;

    if(x>width-w){
        x = width-w;
        x = width - w;
        speedX = speedX * -1;
    }
    else if(x<0){
        x = 0;
        speedX = speedX * -1;
    }
    else if(y > height-h){
        y = height - h;
        speedY = speedY * -1;
    }
    else if(y<0){
        y = 0;
        speedY = speedY * -1;
    }
}

```

[animation_sample11] 衛星軌道の表現

```

float r= 75;
float theta = 0;

void setup(){
    size(200,200);
    frameRate(30);
    smooth();
}

void draw(){
    background(0);
    float x = r *cos(theta);
    float y = r *sin(theta);
    ellipse(x+width/2, y+height/2, 16, 16);
    theta += 0.05;
}

```

[animation_sample12] バネの表現

```

float theta = 0.0;

void setup(){
    size(200,200);
}

void draw(){
    background(255);
    float x = (sin(theta)+1) * width/2;

    theta +=0.05;
    fill(0);
    stroke(0);
    line(width/2,0,x,height/2);
    ellipse(x,height/2,16,16);
}

```

animation_sample13] バネの表現 (減衰)

```

float K = 0.02;
float FRICTION = 0.98;
float SpringLength;
float X,Y;
float Spx, Spy;
float Acx, Acy;
int Radius = 10;

void setup(){
    size(200, 200);
    colorMode(HSB,100);
    frameRate(20);
    SpringLength = height /2;
    X = width/2;
    Y = 0;
    Spx = Spy = 0;
    Acx = Acy = 0;
}

void draw(){
    background(99);
    float distY = Y - SpringLength;
    Acy = -K * distY;
    Spy = FRICTION * (Spy + Acy);

    X = X + Spx;
    Y = Y + Spy;

    stroke(0, 99, min(abs(Acy) * 50, 99));
    line(width/2, Y, width/2, height);

    fill(0);
    noStroke();
    ellipse(X, Y, Radius*2, Radius*2);
}

```

空白

マウスとのインタラクション

[mouse_sample01] マウスポインタの位置に図形を描写

```
void setup(){
    size(200,200);
    noStroke();
    smooth();
}

void draw(){
    background(126);
    ellipse(mouseX,mouseY,33,33);
    ellipse(mouseX+20,50,33,33);
    ellipse(mouseX-20,84,33,33);
}
```

[mouse_sample02] マウスポインタの位置によって、図形の位置と円の半径を変更

```
float x;
float y;
float ix;
float iy;

void setup(){
    size(200,200);
    noStroke();
    smooth();
}

void draw(){
    x=mouseX;
    y=mouseY;
    ix=width-mouseX;
    iy=mouseY-height;
    background(126);
    fill(255,150);
    ellipse(x,height/2,y,y);
    fill(0,159);
    ellipse(ix,height/2,ix,iy);
}
```

[mouse_sample03] マウスポインタの x 座標の位置によって描写する図形を変更

```
void setup(){
    size(200,200);
    noStroke();
    smooth();
    fill(0);
}

void draw(){
    background(204);
    if(mouseX<100){
        rect(0,0,100,200);
    }else{
        rect(100,0,100,200);
    }
}
```

[mouse_sample04] マウスのクリックによって色を塗りつぶす

```
void setup(){
    size(200,200);
    noStroke();
    smooth();
    fill(0);
}

void draw(){
    background(204);

    if(mousePressed == true){
        fill(255);
    }else{
        fill(0);
    }
    rect(25,25,50,50);
}
```

[mouse_sample05] マウスクリックによって背景の明度を明るくする

```
float gray=0;

void setup(){
    size(200,200);
}

void draw(){
    background(gray);
}

void mouseReleased(){
    gray+=20;
}
```

[mouse_sample06] マウスポインタの位置へ、イメージング

```
float x=0;
float y=0;
float easing = 0.05;

void setup(){
    size(200,200);
    noStroke();
    smooth();
    fill(0);
}

void draw(){
    background(0);
    float targetX = mouseX;
    float targetY = mouseY;
    x += (targetX - x) * easing;
    y += (targetY - y) * easing;
    fill(153);
    ellipse(mouseX,mouseY,20,20);
    fill(255);
    ellipse(x,y,40,40);
}
```

[mouse_sample07] マウスポインタの移動速度に応じて、円の半径が変化

```
void setup(){
    size(200,200);
    noStroke();
    smooth();
}

void draw(){
    background(0);
    float speed =
        dist(mouseX, mouseY, pmouseX, pmouseY);
    float diameter = speed * 3.0;
    ellipse(100, 100, diameter, diameter);
}
```

[mouse_sample08] マウスポインタの方角を指示示す図形

```
float x = 100;
float y1 = 66;
float y2 = 122;

void setup(){
    size(200,200);
    noStroke();
    smooth();
}

void draw(){
    background(0);
    float angle = atan2(mouseY-y1, mouseX-x);
    pushMatrix();
    translate(x, y1);
    rotate(angle);
    triangle(-20, -8, 20, 0, -20, 8);
    popMatrix();

    pushMatrix();
    float angle2 = atan2(mouseY-y2, mouseX-x);
    translate(x,y2);
    rotate(angle2);
    triangle(-20, -8, 20, 0,-20.8);
    popMatrix();
}
```

[mouse_sample09] マウスのオーバーロールとクリック、ドラッグによる図形の変化

```
float bx;
float by;
int bs = 20;
boolean bover = false;
boolean locked = false;
float bdifx = 0.0;
float bdify = 0.0;

void setup()
{
    size(200, 200);
    bx = width/2.0;
    by = height/2.0;
    rectMode(RADIUS);
}

void draw()
{
    background(0);

    // Test if the cursor is over the box
    if (mouseX > bx-bs && mouseX < bx+bs &&
        mouseY > by-bs && mouseY < by+bs) {
        bover = true;
        if(!locked) {
            stroke(255);
            fill(153);
        }
    } else {
        stroke(153);
        fill(153);
        bover = false;
    }

    // Draw the box
    rect(bx, by, bs, bs);
}

void mousePressed() {
    if(bover) {
        locked = true;
        fill(255, 255, 255);
    } else {
        locked = false;
    }
    bdifx = mouseX-bx;
    bdify = mouseY-by;
}

void mouseDragged() {
    if(locked) {
        bx = mouseX-bdifx;
        by = mouseY-bdify;
    }
}

void mouseReleased() {
    locked = false;
}
```

配列

扱う変数が多い時は配列を利用する

配列の宣言方法

```
int [] data = new int[5];
```

→data[0]からdata[4]までの配列を作る

```
int [] x = new int [19,20,1,50,125,10,10,0];
```

→new[0]からnew[7]までの配列を作り、それぞれに値を代入する。

配列の使用方法

```
data[0] = 562;
```

→配列data[0]に562を代入する。

[arrays_sample01] 配列を使用した座標の指定



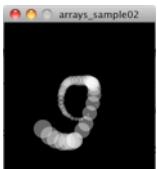
```
int [] x = { 50,61,83,69,71,50,29,31,17,39 };
int [] y = { 18,37,43,60,82,73,82,60,43,37 };

beginShape();
for (int i = 0; i < x.length; i++){
  vertex(x[i], y[i]);
}
endShape(CLOSE);
```

解説

x.lengthはxの配列の個数を表す。
このプログラムでは、"x.length"は10である。

[arrays_sample02] マウスポインタの位置を配列に記録し、残像として表示

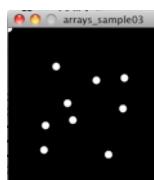


```
int num = 50;
int[] x = new int[num]; //配列xを50個定義
int[] y = new int[num]; //配列yを50個定義

void setup(){
  size(200,200);
  noStroke();
  smooth();
  fill(255, 102);
}

void draw(){
  background(0);
  for(int i = num-1; i>0; i--){ //配列を一つ後ろにずらす
    x[i] = x[i-1];
    y[i] = y[i-1];
  }
  x[0] = mouseX;
  y[0] = mouseY;
  for(int i = 0;i<num;i++){
    ellipse(x[i],y[i],i/2.0,i/2.0);
  }
}
```

[arrays_sample03] クリックの座標を配列に記録し、その位置に図形を表示



```
int num = 10;
int[] x = new int[num];
int[] y = new int[num];
int count = 0;

void setup(){
  size(400,400);
  noStroke();
  smooth();
  fill(255);
}

void draw(){
  background(0);

  for(int i = 0;i<num;i++){
    ellipse(x[i],y[i],10,10);
  }
}

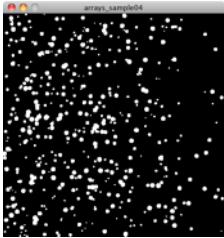
// クリックによって配列xとyに現在のマウスの座標を代入
void mousePressed(){
  y[count] = mouseY;
  x[count] = mouseX;
  count++;
}

if(count>=num){
  count = 0;
}

}
```

[arrays_sample04]

配列によって複数の図形を描写（その1）



```

int ballCount = 500;
int ballSize = 8;
int ballSpeed = 3;
float[] xspeed = new float[ballCount];
float[] yspeed= new float[ballCount];
float[] xpos = new float[ballCount];
float[] ypos = new float[ballCount];
float[] wdth = new float[ballCount];
float[] ht = new float[ballCount];

void setup(){
  size(400, 400);
  background(0);

  //initialize values for all balls
  for (int i= 0; i< ballCount; i++){
    // set varied ball speed
    xspeed[i] = random(1, ballSpeed);
    yspeed[i] = random(-ballSpeed, ballSpeed);

    // ball varied ball sizes
    wdth[i]= random(1, ballSize);
    ht[i]= wdth[i];

    // set initial ball placement
    xpos[i] = width/2+random(-width/3, width/3);
    ypos[i] = height/2+random(-height/3, height/3);
  }

  noStroke();
  frameRate(30);
}

void draw(){
  background(0);

  for (int i=0; i<ballCount; i++){
    //draw balls
    ellipse(xpos[i], ypos[i], wdth[i], ht[i]);

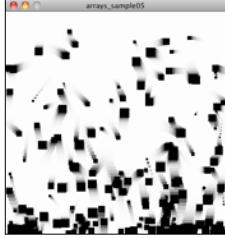
    //upgrade position values
    xpos[i]+=xspeed[i];
    ypos[i]+=yspeed[i];

    /*conditionals:
    detects ball collision with sketch window edges
    accounting for ball thickness.
    */
    if (xpos[i]+wdth[i]/2>=width || xpos[i]<=wdth[i]/2){
      xspeed[i]*=-1;
    }
    if (ypos[i]+ht[i]/2>=height || ypos[i]<=ht[i]/2){
      yspeed[i]*=-1;
    }
  }
}

```

[arrays_sample05]

配列によって複数の図形を描写（その2）



```

// Bouncing Rectangles with Simple Physics III
int shapes = 200;
float[] w = new float[shapes];
float[] h = new float[shapes];
float[] x = new float[shapes];
float[] y = new float[shapes];
float[] xSpeed = new float[shapes];
float[] ySpeed = new float[shapes];
float[] gravity = new float[shapes];
float[] damping = new float[shapes];
float[] friction = new float[shapes];
//controls rate rects are born
float shapeCount;
float birthRate = .25;
// control width of spray when rects are born
float sprayWidth = 5;

void setup(){
  size(400, 400);
  noStroke();
  //initialize arrays with random values
  for (int i=0; i<shapes; i++){
    x[i] = width/2.0;
    w[i] = random(2, 17);
    h[i] = w[i];
    xSpeed[i] = random(-sprayWidth, sprayWidth);
    gravity[i] = .1;
    damping[i] = random(.7, .98);
    friction[i] = random(.65, .95);
  }
}

void draw(){
  //fade background
  fill(255, 100);
  rect(0, 0, width, height);
  fill(0);

  // shapeCount births rects over time
  for (int i=0; i<shapeCount; i++){
    rect(x[i], y[i], w[i], h[i]);
    x[i]+=xSpeed[i];
    ySpeed[i]+=gravity[i];
    y[i]+=ySpeed[i];

    //collision detection
    if (y[i]>=height-h[i]){
      y[i]=height-h[i];
      //bounce
      ySpeed[i]*=-1.0;
      // slow down vertical motion on ground collision
      ySpeed[i]*= damping[i];
      // slow down lateral motion on ground collision
      xSpeed[i]*=friction[i];
    }
    if (x[i]>=width-w[i]){
      x[i]=width-w[i];
      xSpeed[i]*=-1.0;
    }
    if (x[i]<=0){
      x[i]=0;
      xSpeed[i]*=-1.0;
    }
  }
  if (shapeCount<shapes){
    shapeCount+=birthRate;
  }
}

```

時間を扱う関数

millis()

プログラムがスタートしてからの時間を返す。単位はミリセカンド。(1000ミリ秒=1秒)

second() : 現時刻の秒を返す

minute() : 現時刻の分を返す

hour() : 現時刻の時間を返す

day() : 日付を返す

month() : 月を返す

year() : 西暦を返す

[time_sample01] プログラムを開始してから4秒経過するとアニメーションを開始

```
int x = 10;

void setup(){
  size(200,200);
}

void draw(){
  background(255);
  //プログラムを開始してから4秒以上経過場合
  if(millis() > 4000){
    x++;
  }
  line(x, 0, x, 200);
}
```

[time_sample03] 日時を扱う関数を使用し

```
void setup(){
  size(600,600);
  smooth();
}

void draw(){
  background(255);

  float s = map(second(), 0, 60, 0, 600);
  float m = map(minute(), 0, 60, 0, 600);
  float h = map(hour(), 0, 24, 0, 600);

  line(s, 0, s, 200);
  line(m, 200, m, 400);
  line(h, 400, h, 600);
}
```

[time_sample02] 時間の経過によりアニメーションを変化させる

map()関数について

ある値の範囲を他の値の範囲に変換する時にmap()関数を使用します。

例)

data = map(x, a, b, c, d);

aからbまでの範囲内でのxの位置をcからdまでの値の範囲内に置き換える。

```
float x = 0;

void setup(){
  size(200,200);
  smooth();
}

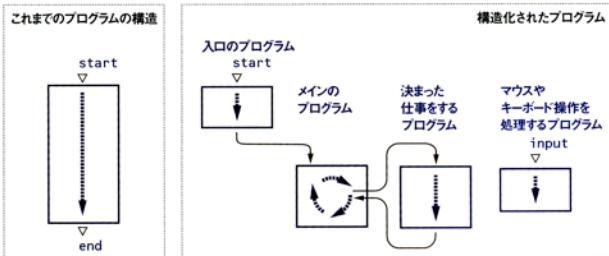
void draw(){
  background(255);

  if(millis() < 5000){ //5秒未満の場合
    x=x+0.1;
  }else if(millis() < 7000){ //5秒以上7秒未満の場合
    x++;
  }else{ //7秒以上の場合
    x=x+0.1;
  }

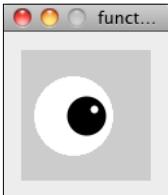
  line(x, 0, x, 200);
}
```

構造化「オリジナル関数の制作」

プログラムの構造化



[function_sample01] 関数の基本1

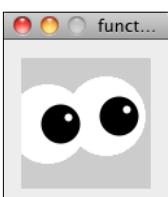


```
void setup(){
    size(100, 100);
    noStroke();
    smooth();
}

void draw(){
    eye();
}

void eye(){
    fill(255);
    ellipse(40, 50, 60, 60);
    fill(0);
    ellipse(50, 50, 30, 30);
    fill(255);
    ellipse(56, 45, 6, 6);
}
```

[function_sample02] 関数の基本2。引数の使用

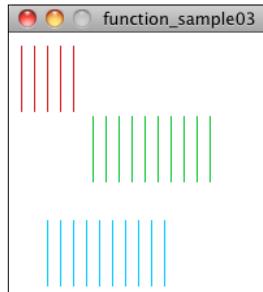


```
void setup(){
    size(100, 100);
    noStroke();
    smooth();
}

void draw(){
    eye(65, 44);
    eye(20, 50);
}

void eye(int x, int y){
    fill(255);
    ellipse(x, y, 60, 60);
    fill(0);
    ellipse(x+10, y, 30, 30);
    fill(255);
    ellipse(x+16, y-5, 6, 6);
}
```

[function_sample03] 関数の基本3。複数の引数を使用

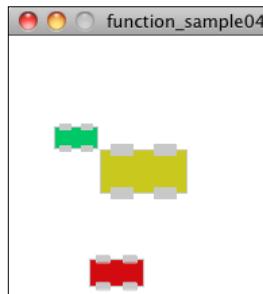


```
void setup(){
    size(200, 200);
    smooth();
}

void draw(){
    background(255);
    drawLines(65, 64, 10, color(20,200,30));
    drawLines(10, 10, 5, color(200,10,10));
    drawLines(30, 144, 10, color(20,200,300));
}

void drawLines(int x, int y, int num, color c){
    stroke(c);
    for (int i = 0; i < num; i++){
        line(x+i*10, y, x+i*10, y+50);
    }
}
```

[function_sample04] 関数で複雑な図形を描く



```
void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    drawCar(100, 100, 64, color(200,200,0));
    drawCar(50, 75, 32, color(0,200,100));
    drawCar(80, 175, 40, color(200,0,0));
}

void drawCar(int x, int y, int thesize, color c){

    int offset = thesize/4;
    rectMode(CENTER);
    stroke(200);
    fill(c);
    rect(x,y,thesize,thesize/2);
    fill(200);
    rect(x-offset, y-offset, offset, offset/2);
    rect(x+offset, y-offset, offset, offset/2);
    rect(x-offset, y+offset, offset, offset/2);
    rect(x+offset, y+offset, offset, offset/2);
}
```

オブジェクト指向

classの使用

↓ オブジェクト名を定義する

```
Car myCar1, myCar2;
```

```
void setup() {  
    size(200,200);      ↓ それぞれのオブジェクトを初期化  
    myCar1 = new Car(color(255,0,0),0,100,2);  
    myCar2 = new Car(color(0,0,255),0,10,1);  
}
```

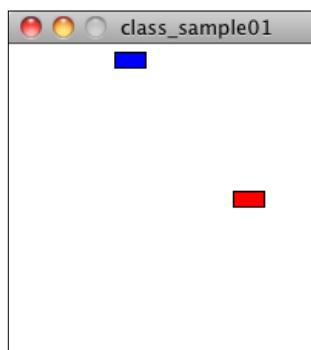
```
void draw() {  
    background(255);  ↓ それぞれのオブジェクトを操作する  
    myCar1.move();  
    myCar1.display();  
    myCar2.move();  
    myCar2.display();  
}
```

↓ クラスの定義

```
class Car {  
    color c;  
    float xpos;  
    float ypos;  
    float xspeed;  
  
    ← クラス内で使用する変数を定義  
  
    ↓ コンストラクタ（クラスの初期化）  
  
    Car(color tempC, float tempXpos,  
         float tempYpos, float tempXspeed) {  
        c = tempC;  
        xpos = tempXpos;  
        ypos = tempYpos;  
        xspeed = tempXspeed;  
    }  
  
    void display() {  
        stroke(0);  
        fill(c);  
        rectMode(CENTER);  
        rect(xpos,ypos,20,10);  
    }  
  
    ← クラス内の関数1  
  
    void move() {  
        xpos = xpos + xspeed;  
        if (xpos > width) {  
            xpos = 0;  
        }  
    }  
    ← クラス内の関数2
```

[class_sample01] classの基本1

Carクラスを定義し、2つの四角をアニメーションさせる



Car myCar1, myCar2; // 2つのオブジェクトを作る

```
void setup() {  
    size(200,200);  
    // オブジェクト myCar1 を初期化する  
    myCar1 = new Car(color(255,0,0),0,100,2);  
    // オブジェクト myCar2 を初期化する  
    myCar2 = new Car(color(0,0,255),0,10,1);  
}
```

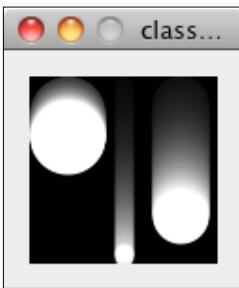
```
void draw() {  
    background(255);  
    myCar1.move(); // オブジェクト myCar1 を操作  
    myCar1.display(); // オブジェクト myCar1 を操作  
    myCar2.move(); // オブジェクト myCar2 を操作  
    myCar2.display(); // オブジェクト myCar2 を操作  
}
```

// クラスの定義ここまで

```
class Car {  
    // クラス内で使用する変数を定義する  
    color c;  
    float xpos;  
    float ypos;  
    float xspeed;  
  
    // クラス内で使用する変数を初期化する  
    Car(color tempC, float tempXpos,  
         float tempYpos, float tempXspeed) {  
        c = tempC;  
        xpos = tempXpos;  
        ypos = tempYpos;  
        xspeed = tempXspeed;  
    }  
  
    // クラス内の関数その1（オブジェクトの画面表示を担当）  
    void display() {  
        stroke(0);  
        fill(c);  
        rectMode(CENTER);  
        rect(xpos,ypos,20,10);  
    }  
  
    // クラス内の関数その2（オブジェクトの座標移動を担当）  
    void move() {  
        xpos = xpos + xspeed;  
        if (xpos > width) {  
            xpos = 0;  
        }  
    }  
}
```

[class_sample02] classの基本2

Spotクラスを定義し、3つの円をアニメーションさせる



Spot sp1, sp2, sp3; // 3つのオブジェクトの定義

```
void setup() {
    size(100, 100);
    smooth();
    noStroke();
    sp1 = new Spot(20, 50, 40, 0.5);
    sp2 = new Spot(50, 50, 10, 2.0);
    sp3 = new Spot(80, 50, 30, 1.5);
}
```

```
void draw() {
    fill(0, 15);
    rect(0, 0, width, height);
    fill(255);
    sp1.move();
    sp2.move();
    sp3.move();
    sp1.display();
    sp2.display();
    sp3.display();
}
```

//クラスの定義ここから

```
class Spot {
    float x, y;
    float diameter;
    float speed;
    // Direction of motion (1 is down, -1 is up)
    int direction = 1;

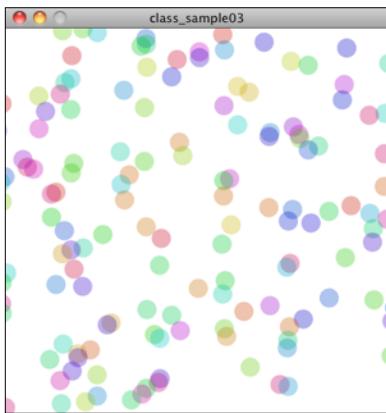
    Spot(float xpos, float ypos, float dia, float sp) {
        x = xpos;
        y = ypos;
        diameter = dia;
        speed = sp;
    }

    void move() {
        y += (speed * direction);
        if ((y > (height - diameter / 2)) || (y < diameter / 2)) {
            direction *= -1;
        }
    }

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

[class_sample03] classの基本3

クラスと配列の組み合わせで、膨大な数のオブジェクトを同時にアニメーションさせる。



```
int numBall = 200;
Ball[] myBall = new Ball[numBall];
```

```
void setup() {
    colorMode(HSB);
    size(400,400);
    smooth();
    frameRate(30);
    for(int i =0; i<myBall.length ; i++){
        myBall[i] = new Ball(color(random(255),255,200),
                             0,i*2,random(20));
    }
}
```

```
void draw() {
    background(255);
    for( int i = 0; i<myBall.length; i++){
        myBall[i].move();
        myBall[i].display();
    }
}
```

//クラスの定義ここから

```
class Ball {
    color c;
    float xpos;
    float ypos;
    float xspeed;

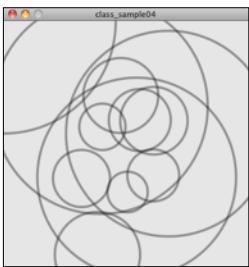
    Ball(color tempC, float tempXpos,
          float tempYpos, float tempXspeed) {
        c = tempC;
        xpos = tempXpos;
        ypos = tempYpos;
        xspeed = tempXspeed;
    }

    void display() {
        noStroke();
        fill(c,80);
        ellipse(xpos,ypos,20,20);
    }

    void move() {
        xpos = xpos + xspeed;
        if (xpos > width+100) {
            xpos = -100;
        }
    }
}
```

[class_sample04] classの基本4

マウスをクリックした場所に広がる円をそれぞれクラスで定義する。



```

Ring[] rings; // Declare the array
int numRings = 50;
int currentRing = 0;

void setup() {
    size(400, 400);
    smooth();
    rings = new Ring[numRings]; // Create the array
    for (int i = 0; i < numRings; i++) {
        rings[i] = new Ring(); // Create each object
    }
}

void draw() {
    background(230);
    for (int i = 0; i < numRings; i++) {
        rings[i].grow();
        rings[i].display();
    }
}

// Click to create a new Ring
void mousePressed() {
    rings[currentRing].start(mouseX, mouseY);
    currentRing++;
    if (currentRing >= numRings) {
        currentRing = 0;
    }
}

// クラスの定義ここから
class Ring {
    float x, y;
    float diameter;
    boolean on = false;

    void start(float xpos, float ypos) {
        x = xpos;
        y = ypos;
        on = true;
        diameter = 1;
    }

    void grow() {
        if (on == true) {
            diameter += 0.5;
            if (diameter > 400) {
                on = false;
            }
        }
    }

    void display() {
        if (on == true) {
            noFill();
            strokeWeight(4);
            stroke(15, 100);
            ellipse(x, y, diameter, diameter);
        }
    }
}

```

[class_sample05] classの基本5

マウスで描いた線の座標が配列に記録され、その線が時間と経過に伴い移動する。



```

int numLines = 500;
MovingLine[] lines = new MovingLine[numLines];
int currentLine = 0;

void setup() {
    size(400, 400);
    smooth();
    frameRate(20);
    for (int i = 0; i < numLines; i++) {
        lines[i] = new MovingLine();
    }
}

void draw() {
    background(204);
    for (int i = 0; i < currentLine; i++) {
        lines[i].display();
    }
}

void mouseDragged() {
    lines[currentLine].setPosition(mouseX, mouseY,
                                    pmouseX, pmouseY);
    if (currentLine < numLines - 1) {
        currentLine++;
    }
}

// クラスの定義ここから
class MovingLine {
    float x1, y1, x2, y2;

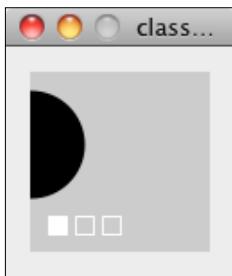
    void setPosition(int x, int y, int px, int py) {
        x1 = x;
        y1 = y;
        x2 = px;
        y2 = py;
    }

    void display() {
        x1 += random(-0.1, 0.1);
        y1 += random(-0.1, 0.1);
        x2 += random(-0.1, 0.1);
        y2 += random(-0.1, 0.1);
        strokeWeight(2);
        line(x1, y1, x2, y2);
    }
}

```

[class_sample06] classの応用

GUIの提案。左下のボタンをクリックすることで、画面のアニメーションの変化を容易に行うことが出来る。



```
Button button1, button2, button3;
```

```
int mode = 1;
```

```
//setup関数
```

```
void setup() {
```

```
size(100, 100);
```

```
smooth();
```

```
color gray = color(204);
```

```
color white = color(255);
```

```
color black = color(0);
```

```
button1 = new Button(10, 80, 10, gray, white, black);
```

```
button2 = new Button(25, 80, 10, gray, white, black);
```

```
button3 = new Button(40, 80, 10, gray, white, black);
```

```
}
```

```
//draw関数
```

```
void draw() {
```

```
background(204);
```

```
manageButtons();
```

```
noStroke();
```

```
fill(0);
```

```
if (mode == 1) {
```

```
ellipse(0, 40, 60, 60);
```

```
} else if (mode == 2) {
```

```
ellipse(50, 40, 60, 60);
```

```
} else if (mode == 3) {
```

```
ellipse(100, 40, 60, 60);
```

```
}
```

```
}
```

```
//manageButtons関数(オリジナルの関数)
```

```
void manageButtons() {
```

```
button1.update();
```

```
button2.update();
```

```
button3.update();
```

```
button1.display();
```

```
button2.display();
```

```
button3.display();
```

```
}
```

```
//mousePressed関数
```

```
void mousePressed() {
```

```
if (button1.press() == true) {
```

```
mode = 1;
```

```
}
```

```
if (button2.press() == true) {
```

```
mode = 2;
```

```
}
```

```
if (button3.press() == true) {
```

```
mode = 3;
```

```
}
```

```
}
```

```
//mouseReleased関数
```

```
void mouseReleased() {
```

```
button1.release();
```

```
button2.release();
```

```
button3.release();
```

```
}
```

//クラスの定義

```
class Button {  
    // The x- and y-coordinates  
    int x, y;  
    int size; // Dimension (width and height)  
    color baseGray; // Default gray value  
    // Value when mouse is over the button  
    color overGray;  
    // Value when mouse is over and pressed  
    color pressGray;  
    boolean over = false; // True when the mouse is over  
    // True when the mouse is over and pressed  
    boolean pressed = false;
```

//クラスのコンストラクタ

```
Button(int xp, int yp, int s, color b, color o, color p) {  
    x = xp;  
    y = yp;  
    size = s;  
    baseGray = b;  
    overGray = o;  
    pressGray = p;  
}
```

// Updates the over field every frame

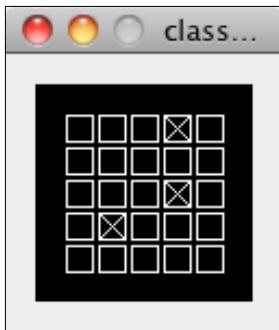
```
void update() {  
    if ((mouseX >= x) && (mouseX <= x + size) &&  
        (mouseY >= y) && (mouseY <= y + size)) {  
        over = true;  
    } else {  
        over = false;  
    }  
}
```

```
boolean press() {  
    if (over == true) {  
        pressed = true;  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
void release() {  
    // Set to false when the mouse is released  
    pressed = false;  
}
```

```
void display() {  
    if (pressed == true) {  
        fill(pressGray);  
    } else if (over == true) {  
        fill(overGray);  
    } else {  
        fill(baseGray);  
    }  
    stroke(255);  
    rect(x, y, size, size);  
}
```

[class_sample07] classの応用
チェックボックスをクラスで一括して作る。



```
int numChecks = 25;
Check[] checks = new Check[numChecks];

void setup() {
    size(100, 100);
    int x = 14;
    int y = 14;
    for (int i = 0; i < numChecks; i++) {
        checks[i] = new Check(x, y, 12, color(0));
        x += 15;
        if (x > 80) {
            x = 14;
            y += 15;
        }
    }
}

void draw() {
    background(0);
    for (int i = 0; i < numChecks; i++) {
        checks[i].display();
    }
}

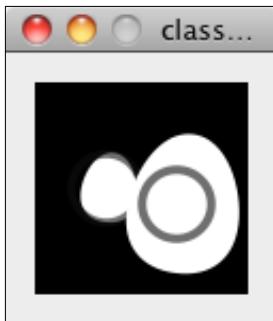
void mousePressed() {
    for (int i = 0; i < numChecks; i++) {
        checks[i].press(mouseX, mouseY);
    }
}
```

```
// クラスの定義
class Check {
    int x, y; // The x- and y-coordinates
    int size; // Dimension (width and height)
    color baseGray; // Default gray value
    // True when the check box is selected
    boolean checked = false;
    Check(int xp, int yp, int s, color b) {
        x = xp;
        y = yp;
        size = s;
        baseGray = b;
    }

    // Updates the boolean variable checked
    void press(float mx, float my) {
        if ((mx >= x) && (mx <= x + size)
            && (my >= y) && (my <= y + size)) {
            // Toggle the check box on and off
            checked = !checked;
        }
    }

    // Draws the box and an X inside
    void display() {
        stroke(255);
        fill(baseGray);
        rect(x, y, size, size);
        if (checked == true) {
            line(x, y, x + size, y + size);
            line(x + size, y, x, y + size);
        }
    }
}
```

[class_sample08] classの応用
複数のクラスの定義



```
EggRing er1, er2;

void setup() {
    size(100, 100);
    smooth();
    er1 = new EggRing(33, 66, 0.1, 33);
    er2 = new EggRing(66, 90, 0.05, 66);
}

void draw() {
    background(0);
    er1.transmit();
    er2.transmit();
}
```

```
//クラスの定義
class Egg {
    float x, y; // X-coordinate, y-coordinate
    float tilt; // Left and right angle offset
    float angle; // Used to define the tilt
    float scalar; // Height of the egg

    // Constructor
    Egg(int xpos, int ypos, float t, float s) {
        x = xpos;
        y = ypos;
        tilt = t;
        scalar = s / 100.0;
    }

    void wobble() {
        tilt = cos(angle) / 8;
        angle += 0.1;
    }

    void display() {
        noStroke();
        fill(255);
        pushMatrix();
        translate(x, y);
        rotate(tilt);
        scale(scalar);
        beginShape();
        vertex(0, -100);
        bezierVertex(25, -100, 40, -65, 40, -40);
        bezierVertex(40, -15, 25, 0, 0, 0);
        bezierVertex(-25, 0, -40, -15, -40, -40);
        bezierVertex(-40, -65, -25, -100, 0, -100);
        endShape();
        popMatrix();
    }
}
```

```
//クラスの定義
class EggRing {
    Egg ovoid;
    Ring circle = new Ring();

    EggRing(int x, int y, float t, float sp) {
        ovoid = new Egg(x, y, t, sp);
        circle.start(x, y - sp / 2);
    }

    void transmit() {
        ovoid.wobble();
        ovoid.display();
        circle.grow();
        circle.display();
        if (circle.on == false) {
            circle.on = true;
        }
    }
}
```

```
//クラスの定義
class Ring {
    float x, y; // X-coordinate, y-coordinate
    float diameter; // Diameter of the ring
    boolean on = false; // Turns the display on and off

    void start(float xpos, float ypos) {
        x = xpos;
        y = ypos;
        on = true;
        diameter = 1;
    }

    void grow() {
        if (on == true) {
            diameter += 0.5;
            if (diameter > 400) {
                on = false;
            }
        }
    }

    void display() {
        if (on == true) {
            noFill();
            strokeWeight(4);
            stroke(15, 153);
            ellipse(x, y, diameter, diameter);
        }
    }
}
```

[class_sample09] classの応用
クラスによるパーティクルの表現1



```
int numParticles = 200;
GenParticle[] p = new GenParticle[numParticles];

void setup() {
    size(100, 100);
    noStroke();
    smooth();
    for (int i = 0; i < p.length; i++) {
        float velX = random(-1, 1);
        float velY = -i;
        // Inputs: x, y, x-velocity, y-velocity,
        // radius, origin x, origin y
        p[i] = new GenParticle(width/2, height/2, velX, velY,
            5.0, width / 2, height / 2);
    }
}

void draw() {
    fill(0, 36);
    rect(0, 0, width, height);
    fill(255, 60);
    for (int i = 0; i < p.length; i++) {
        p[i].update();
        p[i].regenerate();
        p[i].display();
    }
}
```

```
//クラスの定義
class Particle {
    float x, y; // The x- and y-coordinates
    float vx, vy; // The x- and y-velocities
    float radius; // Particle radius
    float gravity = 0.1;

    Particle(int xpos, int ypos,
             float velx, float vely, float r) {
        x = xpos;
        y = ypos;
        vx = velx;
        vy = vely;
        radius = r;
    }

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

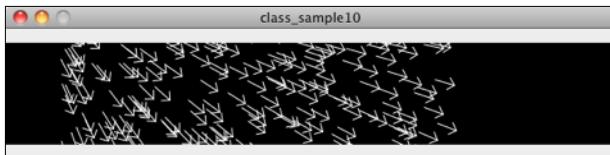
    void display() {
        ellipse(x, y, radius*2, radius*2);
    }
}
```

```
//クラスの定義
class GenParticle extends Particle {
    float originX, originY;

    GenParticle(int xln, int yln, float vxln,
               float vyln, float r, float ox, float oy) {
        super(xln, yln, vxln, vyln, r);
        originX = ox;
        originY = oy;
    }

    void regenerate() {
        if ((x > width + radius) || (x < -radius) ||
            (y > height + radius) || (y < -radius)) {
            x = originX;
            y = originY;
            vx = random(-1.0, 1.0);
            vy = random(-4.0, -2.0);
        }
    }
}
```

[class_sample10] classの応用
クラスによるパーティクルの表現2



```
int num = 320;
ArrowParticle[] p = new ArrowParticle[num];
float radius = 1.2;

void setup() {
    size(600, 100);
    smooth();
    for (int i = 0; i < p.length; i++) {
        float velX = random(1, 8);
        float velY = random(-5, -1);
        // Parameters: x, y, x-velocity, y-velocity, radius
        p[i] = new ArrowParticle(0, height / 2,
                               velX, velY, 1.2);
    }
}

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

//クラスの定義 (パーティクルの部分を担当)

```
class Particle {

    float x, y; // X-coordinate, y-coordinate
    float vx, vy; // X velocity, y velocity
    float radius; // Particle radius
    float gravity = 0.1;

    Particle(int xln, int yln, float vxln, float vyln, float r) {
        x = xln;
        y = yln;
        vx = vxln;
        vy = vyln;
        radius = r;
    }

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

    void display() {
        ellipse(x, y, radius*2, radius*2);
    }
}
```

//クラスの定義 (パーティクルの形状を担当)

```
class ArrowParticle extends Particle {
    float angle = 0.0;
    float shaftLength = 20.0;
    ArrowParticle(int ix, int iy, float ivx, float ivy, float ir) {
        super(ix, iy, ivx, ivy, ir);
    }

    void update() {
        super.update();
        angle = atan2(vy, vx);
    }

    void display() {
        stroke(255);
        pushMatrix();
        translate(x, y);
        rotate(angle);
        scale(shaftLength);
        strokeWeight(1.0 / shaftLength);
        line(0, 0, 1, 0);
        line(1, 0, 0.7, -0.3);
        line(1, 0, 0.7, 0.3);
        popMatrix();
    }
}
```

VJ発表のTips

キーボードによる画像やテンポの切替

[vj_sample01] キーボードによる画像表示1

任意のキーを押したときに画像を表示させる

```
boolean button1;  
//キーボードを押すことによって  
//trueとfalseが入れ替わる変数  
float b1;//図形の角度を変更する変数  
  
void setup(){  
  size(400,400);  
  smooth();  
  frameRate(30);  
}  
  
void draw(){  
  background(255);  
  
  if (button1){//ボタンが押された後の処理  
    pushMatrix();  
    b1 = b1+3;  
    if(360<b1)b1=0;  
    translate(width/2,height/2);  
    rotate(radians(b1));  
    noFill();  
    strokeWeight(10);  
    stroke(0,50);  
    rect(-100,-100,200,200);  
    rect(-200,-200,400,400);  
    popMatrix();  
  }  
}  
  
void keyPressed(){ //キーボードを押した瞬間の処理  
  button1 = true;  
}  
void keyReleased(){ //キーボードを離した瞬間の処理  
  button1 = false;  
}
```

[vj_sample02] キーボードによる画像表示2

1と2と3のキーを押した時に画像を表示させる

```
boolean button1, button2, button3;  
float b1,b2,b3;  
  
void setup(){  
  size(400,400);  
  smooth();  
  frameRate(30);  
}  
  
void draw(){  
  noStroke();  
  fill(255,100);  
  rect(0,0,width,height);  
  
  if (button1){//1のキーが押されたときの処理  
    drawButton1();  
  }  
  else{  
    b1=0;  
  }  
  if (button2){//2のキーが押されたときの処理  
    drawButton2();  
  }  
  if (button3){ //2のキーが押されたときの処理  
    drawButton3();  
  }  
}
```

(vj_sample02続き)

```
void drawButton1(){//1のキーが押されたときの処理内容  
  noFill();  
  strokeWeight(10);  
  stroke(0,50);  
  ellipse(width/2,height/2,b1,b1);  
  b1=b1+10;  
  if(b1 > height)b1=0;  
}  
  
void drawButton2(){//2のキーが押されたときの処理内容  
  pushMatrix();  
  b2 = b2+3;  
  if(360<b2)b2=0;  
  translate(width/2,height/2);  
  rotate(radians(b2));  
  noFill();  
  strokeWeight(10);  
  stroke(0,50);  
  rect(-100,-100,200,200);  
  rect(-200,-200,400,400);  
  popMatrix();  
}  
  
void drawButton3(){//3のキーが押されたときの処理内容  
  strokeWeight(10);  
  noFill();  
  stroke(0,50);  
  line(b3,0,b3,height);  
  b3=b3+5;  
  if(b3==width)b3=0;  
}  
  
void keyPressed(){//キーボードを押した瞬間の処理  
  switch(key){  
    case('1'): //キーが1の場合  
      button1 = true;  
      break;  
    case('2'): //キーが2の場合  
      button2 = true;  
      break;  
    case('3'): //キーが3の場合  
      button3 = true;  
      break;  
  }  
}  
  
void keyReleased(){//キーボードを離した瞬間の処理  
  switch(key){  
    case('1'): //キーが1の場合  
      button1 = false;  
      break;  
    case('2'): //キーが2の場合  
      button2 = false;  
      break;  
    case('3'): //キーが3の場合  
      button3 = false;  
      break;  
  }  
}
```

[vj_sample03] キーを押したとき順次画像が切り替わる
キー1を押すごとに順番に切り替わり、キー2は逆方向に
切り替わる。

```
boolean button1, button2, button3;  
float b1,b2,b3;  
int keyOrder=0;  
  
void setup(){  
  size(400,400);  
  smooth();  
  frameRate(30);  
}  
  
void draw(){  
  noStroke();  
  //残像を残す処理  
  fill(255,160);  
  rectMode(CORNER);  
  rect(0,0,width,height);  
  
  //keyOrder変数の値に応じて処理を変更する  
  switch(keyOrder){  
    case(0):  
      drawButton1();  
      break;  
    case(1):  
      drawButton2();  
      break;  
    case(2):  
      drawButton3();  
      break;  
    case(3):  
      drawButton1();  
      drawButton2();  
      drawButton3();  
      break;  
  }  
}
```

(vj_sample03続き)

```
void drawButton1(){//1のキーが押されたときの処理内容  
noFill();  
strokeWeight(3);  
stroke(0,80);  
ellipse(width/2,height/2,b1,b1);  
b1=b1+10;  
if(b1 > height)b1=0;  
}  
  
void drawButton2(){//2のキーが押されたときの処理内容  
pushMatrix();  
b2 = b2+3;  
if(360<b2)b2=0;  
translate(width/2,height/2);  
rotate(radians(b2));  
noFill();  
strokeWeight(3);  
stroke(0,80);  
rect(-100,-100,200,200);  
rect(-200,-200,400,400);  
popMatrix();  
}  
  
void drawButton3(){//3のキーが押されたときの処理内容  
strokeWeight(3);  
noFill();  
stroke(0,80);  
line(b3,0,b3,height);  
b3=b3+5;  
if(b3==width)b3=0;  
}  
  
//キーが押されたときに変数keyOrderをカウントアップ  
void keyPressed(){  
  if(key=='1'){  
    keyOrder++;  
    if(keyOrder > 3) keyOrder=0;  
  }  
}  
  
//キーが離されたときに変数keyOrderをカウントダウン  
void keyReleased(){  
  if(key=='2'){  
    keyOrder--;  
    if(keyOrder < 0) keyOrder=3;  
  }  
}
```

[vj_sample04] タップによるテンポの入力
キー”1”でテンポを入力し、キー”2”でテンポを再生

```
float angle = 0;
float time_old = 0;
float beat=0;//入力テンポを格納（単位ミリセカンド）
int clock = 0;
int clock_old = 0;
float[] tap = new float[5];
int x=0;
int start = 0;

void setup(){
  size(400,400,P3D);
  frameRate(30);
}

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

  countBeat();//オリジナル関数でテンポの計算を行う

  //图形描写
  lights();
  fill(255);
  translate(width/2,height/2);
  rotateX(radians(-30));
  rotateY(radians(angle));
  box(100,100,100);
  stroke(255);
  strokeWeight(2);
  line(-200,0,0,200,0,0);
}

void keyPressed(){ //キーボードを押した瞬間の処理

  if(key == '1'){ //キー1を押した場合
    start=0;

    tap[x]=(millis()-time_old);
    time_old =time_old+tap[x];
    x++;
    if(x>4)x=0;
    angle=angle+10;

    beat = tap[0]+tap[1]+tap[2]+tap[3]+tap[4];
    beat = beat/5;
  }

  if(key == '2'){//キー1を押した場合
    start=1;
    clock_old = millis();
  }
}

void countBeat(){ //テンポの計算
  clock = millis() - clock_old;
  if(clock>=beat){
    clock_old = clock + clock_old;
    if(start==1){
      angle=angle+10;//テンポによって角度を変更
    }
    if(angle>=360)angle=0;
  }
}
```

[vj_sample05] クラスの使用による図形の切替

```
//クラスの定義
int numBall = 200;
Ball[] myBall = new Ball[numBall];
Spot sp1, sp2;

boolean button1, button2, button3;
float b1,b2,b3;

void setup(){
  size(400,400);
  smooth();
  colorMode(HSB);
  frameRate(30);

  //クラスの初期化
  for(int i =0; i<myBall.length ; i++){
    myBall[i] = new Ball(color(random
      (255),255,200),0,i*2,random(20));
  }
  sp1 = new Spot(100, 50, 100, 0.5);
  sp2 = new Spot(250, 100, 200, 2.0);
}

void draw(){
  background(255);

  if (button1){ //キー1によるクラスの呼び出し
    sp1.move();
    sp2.move();
    sp1.display();
    sp2.display();
  }

  if (button2){//キー2によるクラスの呼び出し
    for( int i = 0; i<myBall.length; i++){
      myBall[i].move();
      myBall[i].display();
    }
  }
}

void keyPressed(){//キーボードを押した瞬間の処理
  switch(key){
    case('1'):
      button1 = true;
      break;
    case('2'):
      button2 = true;
      break;
  }
}

void keyReleased(){//キーボードを離した瞬間の処理
  switch(key){
    case('1'):
      button1 = false;
      break;
    case('2'):
      button2 = false;
      break;
  }
}
```

(vj_sample05続き)

```
class Ball {  
    color c;  
    float xpos;  
    float ypos;  
    float xspeed;  
  
    Ball(color tempC, float tempXpos,  
          float tempYpos, float tempXspeed) {  
        c = tempC;  
        xpos = tempXpos;  
        ypos = tempYpos;  
        xspeed = tempXspeed;  
    }  
  
    void display() {  
        noStroke();  
        fill(c,80);  
        ellipse(xpos,ypos,20,20);  
    }  
  
    void move() {  
        xpos = xpos + xspeed;  
        if (xpos > width+100) {  
            xpos = -100;  
        }  
    }  
}  
  
class Spot {  
    float x, y;  
    float diameter;  
    float speed;  
    int direction = 1;  
  
    Spot(float xpos, float ypos, float dia, float sp) {  
        x = xpos;  
        y = ypos;  
        diameter = dia;  
        speed = sp;  
    }  
  
    void move() {  
        y += (speed * direction);  
        if ((y > (height - diameter / 2)) || (y < diameter / 2)) {  
            direction *= -1;  
        }  
    }  
  
    void display() {  
        noFill();  
        strokeWeight(2);  
        stroke(90);  
        ellipse(x, y, diameter, diameter);  
    }  
}
```

[vj_sample06] 特定のクラスの図形のみ残像を残す
キー1で残像のある図形を表示

```
//クラスの定義  
Sample mySample1, mySample2;  
Spot sp1;  
  
void setup(){  
    size(400,400);  
    frameRate(30);  
//クラスの初期化  
    mySample1 = new Sample  
        (color(255,0,0,100),0,120,2);  
    mySample2 = new Sample  
        (color(255,220,0,100),0,250,8);  
    sp1 = new Spot(250, 100, 200, 6.0);  
    smooth();  
}  
  
void draw(){  
  
    background(255);  
  
//クラスの描写  
    sp1.move();  
    sp1.display();  
  
    mySample1.display();  
    mySample1.move();  
    mySample2.display();  
    mySample2.move();  
}  
  
void keyPressed(){//キーボードを押した瞬間の処理  
    if(key=='1'){  
        //クラス内のfadein関数を呼出  
        mySample1.fadeIn();  
        mySample2.fadeIn();  
    }  
}  
  
void keyReleased(){//キーボードを離した瞬間の処理  
    if(key=='1'){  
        //クラス内のfadeOut関数を呼出  
        mySample1.fadeOut();  
        mySample2.fadeOut();  
    }  
}
```

(vj_sample06続き)

```
class Sample {  
    color c;  
    float xpos;  
    float ypos;  
    float xspeed;  
    int f;  
    boolean fade;  
  
    Sample(color tempC, float tempXpos,  
           float tempYpos, float tempXspeed) {  
        c = tempC;  
        xpos = tempXpos;  
        ypos = tempYpos;  
        xspeed = tempXspeed;  
        f = 0;  
        fade = false;  
    }  
  
    void display() {  
  
        if(fade==true){  
            f=f+10;  
        }else{  
            f=f-10;  
        }  
        if(f<0)f=0;  
        if(f>255)f=255;  
  
        noStroke();  
        fill(c,f);  
        rectMode(CENTER);  
        rect(xpos,ypos,200,200);  
    }  
  
    void move() {  
        xpos = xpos + xspeed;  
        if (xpos > width+200) {  
            xpos = -200;  
        }  
    }  
  
    void fadeIn(){  
        fade = true;  
    }  
  
    void fadeOut(){  
        fade = false;  
    }  
}
```

```
class Spot {  
    float x, y;  
    float diameter;  
    float speed;  
    int direction = 1;  
  
    Spot(float xpos, float ypos, float dia, float sp) {  
        x = xpos;  
        y = ypos;  
        diameter = dia;  
        speed = sp;  
    }  
  
    void move() {  
        y += (speed * direction);  
        if ((y > (height - diameter / 2)) || (y < diameter / 2)) {  
            direction *= -1;  
        }  
    }  
  
    void display() {  
        noFill();  
        strokeWeight(2);  
        stroke(90);  
        ellipse(x, y, diameter, diameter);  
    }  
}
```