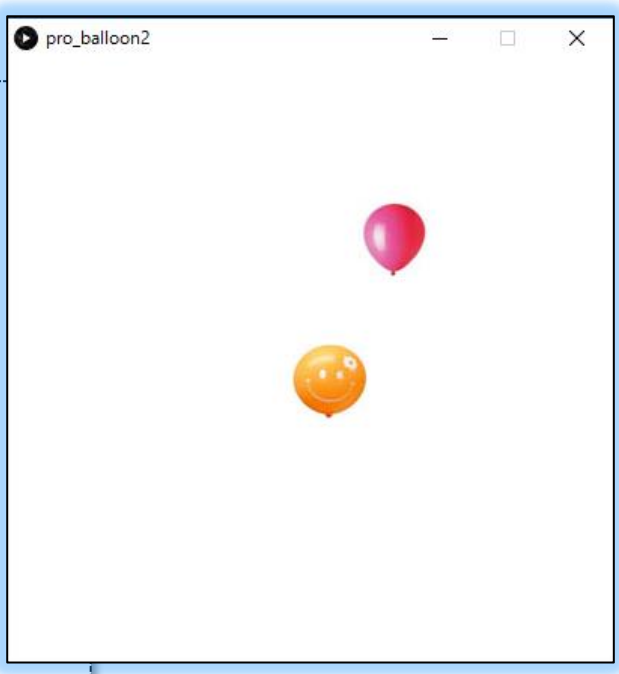
プログラミング入門

Processing の魅力

```
class Balloon{
   PImage p;
   private int x;
   private int y;
   public Balloon(){
      this(170,170);
   }
   public Balloon(int x,int y){
      this.x = x;
      this.y = y;
      p = loadImage("balloon1.jpg");
   }
```



プログラミング学習に最適な言語～Processing～

「BASIC」のように親しみやすく、オブジェクト指向など高度なプログラムにも挑戦できる Java ベースの言語で、フリーで使用できます。

ビジュアルデザインの機能も豊富です。

目　　次

第１章（基礎編）

第２章（ビジュアルデザイン編）

第３章（アニメーション編）

第４章（オブジェクト指向編）

第１章（基礎編）

1　プログラミングの環境を作る

①　Processing をインストールする

「Processing」（プロセッシング）はビジュアルデザイン用に開発された言語ですが、プログラミングの入門としても魅力的な言語で、フリーで使用できます。

「Processing」の公式ホームページから、ダウンロードします。

「http://processing.org/」に接続し、Download より自分のパソコンにあったものをダウンロードし、zip 形式のファイルを適当な場所に解凍して完了です。

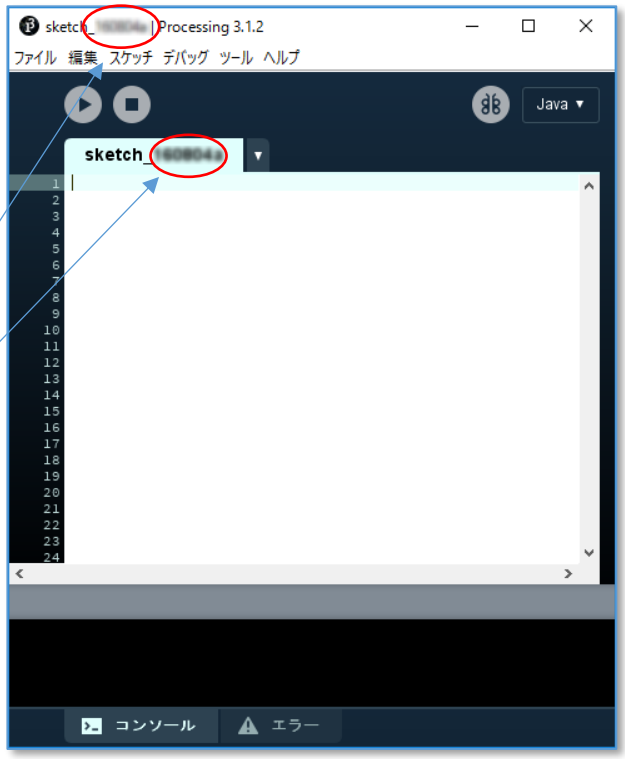
Point Windows の 32 ビット形式のダウンロードファイルは、「processing-3.3-windows32.zip」　です。

これを解凍すると、「processing-3.3」というフォルダができます。このフォルダを適当な場所に配置しておきます。（バージョン 3.3 の場合）　※2017.2.24 現在のバージョン

②　Processing を起動する

「processing-3.1.2」というフォルダの中にある次のようなアイコンのファイルをダブルクリックして起動します。

processing ←起動用ファイル processing. exe



Point 最初に起動すると、スケッチ（Processing ではプログラムのことをスケッチという）を保存するフォルダを指定する画面になります。通常はドキュメントの中に「Processing」というフォルダを作り、その中に保存します。

日付をもとに仮のスケッチ名が付けられています。保存するときは、分かり易い名前を付けて保存します。

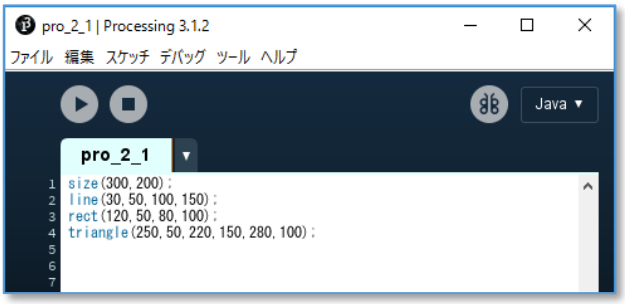
2　プログラム(スケッチ)を入力してみる

まずは簡単なプログラムを入力し、実行してみましょう。

①　直線、四角、三角を描く

右図のように、プログラムを入力し、▶　の実行ボタンをクリックすると、実行されます。

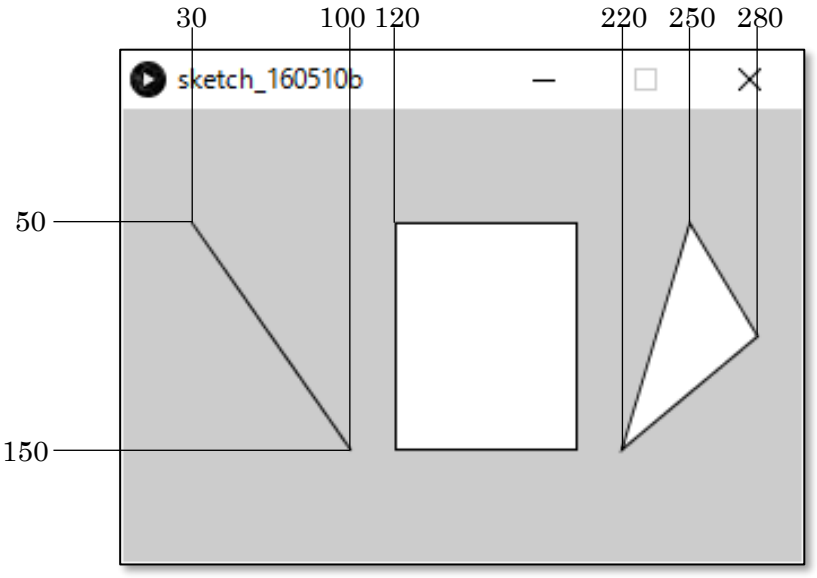
１行目に、size(300,200);
と入力し、▶　をクリックしてみましょう。
パソコンの画面に、300×200　の大きさのウィンドウが表示されます。
以下、２行目～４行目を入力、実行してみましょう。



各行末はセミコロンを付けます

```
size(300,200);
line(30,50,100,150);
rect(120,50,80,100);
triangle(250,50,220,150,280,100);
```

横 300 ドット、縦 200 ドットのディスプレイウィンドウを表示
座標 (30, 50) と (100, 150) を結ぶ直線を描画
座標 (120, 50) を左上とする横 80, 縦 100 の四角
座標 (250, 50), (220, 150), (280, 100) を結ぶ三角を描画



※作成したスケッチは、"pro_2_1" として保存しておきます。

② いろいろな色で描く

描く図形や背景の色を変えてみます。

色の指定は、赤、緑、青の３つの要素の強弱を０～２５５の数字で指定します。

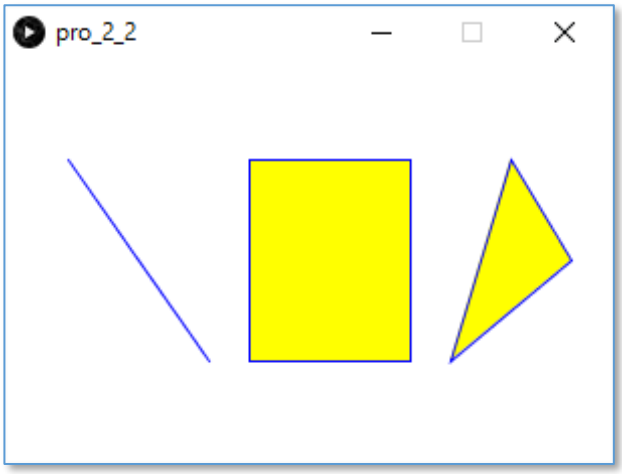
	命　　令	使　　用　　例
背景色	background(r, g, b)	background(0, 0, 0)　背景を黒にする
線の色	stroke(r, g, b)	stroke(255, 0, 0)　線の色を赤くする
塗りつぶし	fill(r, g, b)	fill(0, 255, 0)　図形の中を緑色で塗る

次のプログラムを入力し、実行してみましょう。

```
size(300,200);
background(255,255,255);
stroke(0,0,255);
fill(255,255,0);
line(30,50,100,150);
rect(120,50,80,100);
triangle(250,50,220,150,280,100);
```

背景を白色に、線の色を青色に、塗り色を黄色に指定しています。

※括弧の中の数字をいろいろと変えて、色の変化を確認しましょう。



※作成したスケッチは、"pro_2_2" として保存しておきます。

Point　図形の中を色で塗らない場合は、「 noFill(); 」と記述しておきます。(それより下に記述された命令では色が塗られなくなります。

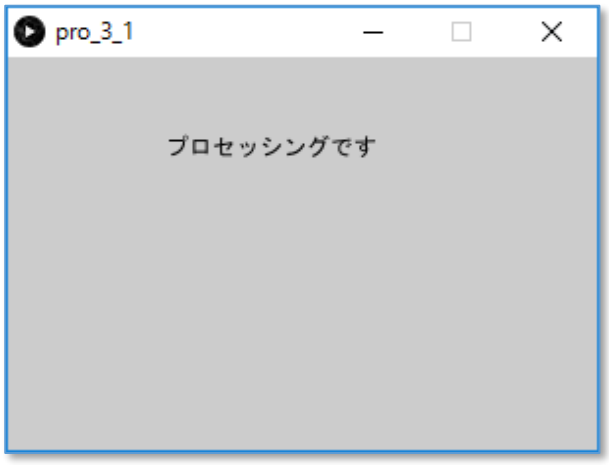
3　文字を書く

表示されているウィンドウの中に文字を表示してみましょう。

①　ウィンドウの中に文字を書く

次のスケッチを入力し、実行してみましょう。

```
size(300,200);
fill(0,0,0);
text("プロセッシングです",80,50);
```



Point　fill(0,0,0)で黒色を指定し、(80,50)の位置に文字を書いています。
「text」はディスプレイウィンドウに文字を表示する命令です。

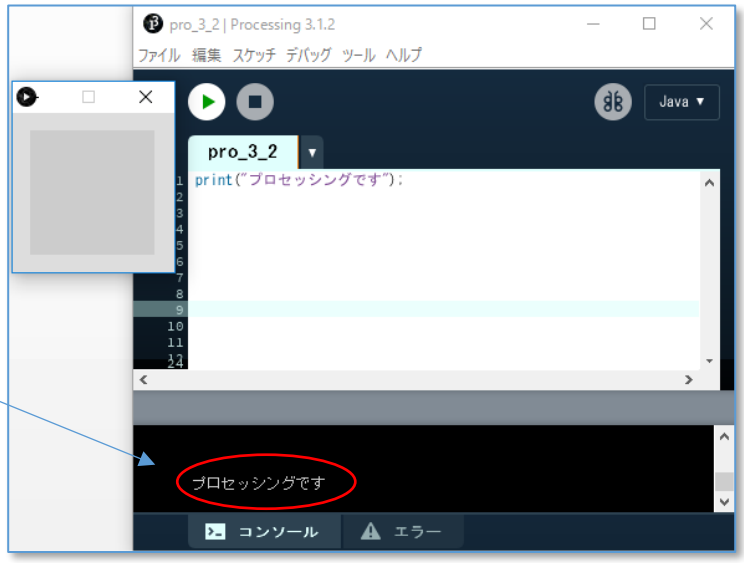
※作成したスケッチは、"pro_3_1" として保存しておきます。

②　テキストエリアに文字を書く

前のスケッチを削除し、下の一行を入力して、実行してみましょう。

```
print("プロセッシングです");
```

小さなディスプレイウィンドウも表示されますが、文字はテキストエリアに表示されています。



Point「print」はテキストエリア（画面下）に文字を表示する命令です。

※作成したスケッチは、"pro_3_2" として保存しておきます。

4　変数を使う

変数とは電卓のメモリーのようなもので、数字や文字を記録(記憶）しておいて、スケッチ（プログラム）の中でそれを使うことができます。データを入れておく箱のイメージです。

電卓のメモリーは一つしかありませんが、プロセッシングの中にはたくさんのメモリー（変数）を作っておくことができます。

変数は、中に入れるデータの種類により、”型”を決めておきます。

また、名前を自由に付けることができます。

（例）ｉｎｔ　ａ；　　ａという名前の***整数***を入れることができる変数を準備する

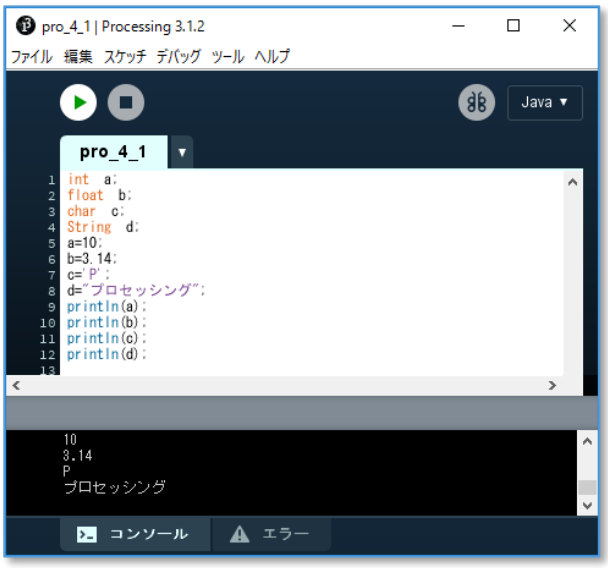
ｆｌｏａｔ　ｂ；　　ｂ　〃　***小数***　〃

ｃｈａｒ　ｃ；　　ｃ　〃　***英数字１文字***　〃

Ｓｔｒｉｎｇ　ｄ；　　ｄ　〃　***文字列***　〃

次のスケッチを入力し、実行してみましょう。

```
int  a;
float  b;
char  c;
String  d;
a=10;
b=3.14;
c='P';
d="プロセッシング";
println(a);
println(b);
println(c);
println(d);
```



Point　println( ) はテキストエリアに表示した後、改行する命令です。

変数の名前は、seisuu などと自由に、分かりやすい名前を付けることができます。

```
int  seisuu;
seisuu = 10;
println(seisuu);
```

などとします。

※作成したスケッチは、"pro_４_１" として保存しておきます。

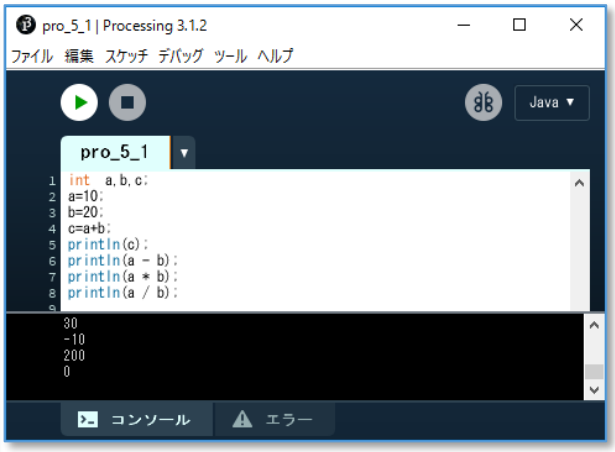
5 計算してみる

変数に入っているデータは、変数の名前を使って中身を計算することができます。

① 数字を計算する

次のスケッチを入力し、実行してみましょう。

```
int   a,b,c;          //変数 a,b,c を宣言
a=10;                 //a に 10 を代入
b=20;                 //b に 20 を代入
c=a+b;                //足し算
println(c);
println(a - b);       //引き算
println(a * b);       //掛け算
println(a / b);       //割り算
```



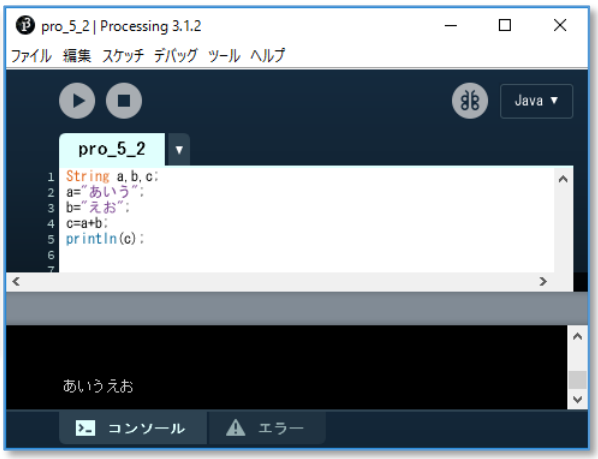
Point //の後にはスケッチの説明など、メモ書きをすることができます。
整数型の変数の計算結果は整数になります。a/b は 0.5 ですが結果は 0 と表示されています。
int の部分を double に書き換えてみましょう。

※作成したスケッチは、"pro_5_1" として保存しておきます。

② 文字を結合する

文字列についても足し算ができます。次のスケッチを入力し、実行してみましょう。

```
String a,b,c;         //変数 a,b,c を宣言
a="あいう";           //a に”あいう”を代入
b="えお";             //b に”えお”を代入
c=a+b;                //文字列の結合
println(c);           //結合した文字の表示
```



※作成したスケッチは、"pro_5_2" として保存しておきます。

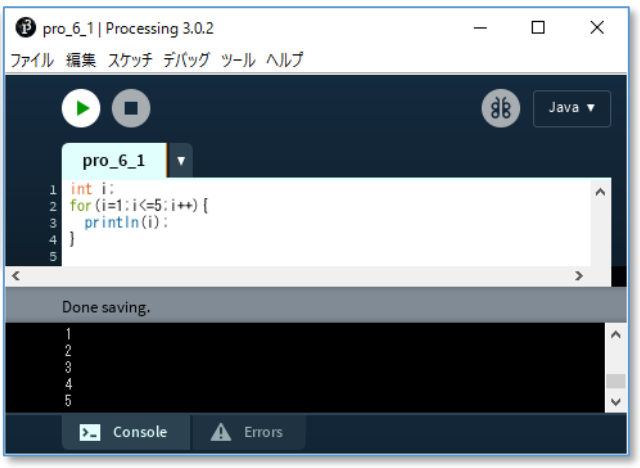
6　処理を繰り返す

スケッチの中で同じような処理を繰り返し実行するときに便利な命令があります。

①　for 文による繰り返し

次のスケッチを入力し、実行してみましょう。

```
int  i;
for(i=1;i<=5;i++){
   println(i);
}
```



１～５の数字がテキストエリアに表示されます。

ｆ ｏ ｒ文は、初期処理、判定、更新処理を括弧の中に記述します。

```
ｆ ｏ ｒ （ 初期処理 ； 判定 ； 更新処理 ）{
        繰り返す命令
}
```

上の例では、ｉが１から始まって５と等しくなるまで、ｉの値が１ずつ増えながら繰り返します。
ｉ＋＋　はｉを１ずつ増やす、という意味です。
{　　}で繰り返す命令を囲っていますが、命令の数が少ない（複数行にならない）ときは、

```
for(i=5;i<=5;i++)  println(i);
```

と１行で書くこともできます。

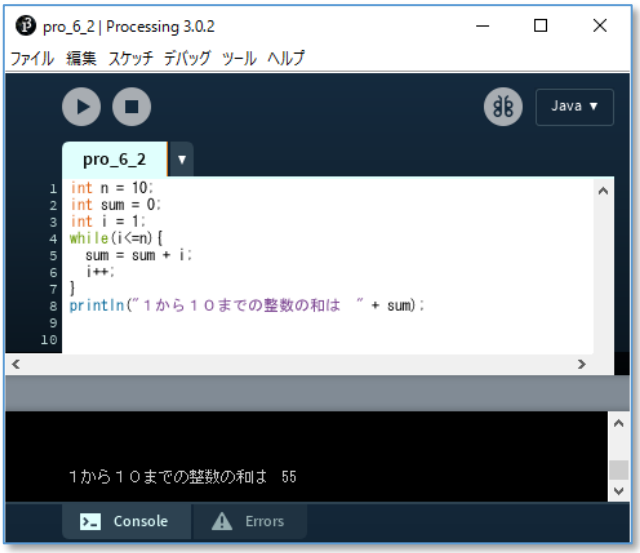
Point　println(i)の部分を、println(i*i)　とすると、ｉの２乗が表示されます。

※作成したスケッチは、"pro_６_１" として保存しておきます。

② while 文による繰り返し

ｆｏｒ文と同じように、繰り返し処理を行う命令に、"ｗｈｉｌｅ"があります。
次のスケッチを入力し、実行してみましょう。

```
int  n = 10;
int  sum = 0;
int  i = 1;
while(i<=n){
   sum = sum + i;
   i++;
}
println("１から１０までの整数の和は　"
+ sum);
```




```
ｗｈｉｌｅ（判定）{
        繰り返す命令
}
```

ｗｈｉｌｅ文は、括弧内の条件式が成り立っている間、処理を繰り返します。

ｓｕｍ　は合計を入れる整数型の変数で、最初は０になっています。
ｗｈｉｌｅのループを回るたびに、ｓｕｍにｉの値が足されていきます。
ｉ＋＋　はｉの値を１増やす命令で、１から１０まで変化します。
最後に１～１０までの合計が、テキストエリアに表示されます。

１から１０までの整数の和は、
println(1+2+3+4+5+6+7+8+9+10); としても表示できますが、数が大きくなると書くことが大変です。ｗｈｉｌｅループを上手く使うことで、短いスケッチにすることができます。

※ ***ｎ＝１０　を　ｎ＝１００ と書き換えて実行してみましょう。***
（println の中の文も変更しておきます。）

※作成したスケッチは、"pro_６_２" として保存しておきます。

7 Active モードと Static モード

① Active モードとは

これまで入力したスケッチは、上から順次、下へ実行されるスケッチ（プログラム）で、プログラムの基礎を理解したり、静止画を描くことに適していました。この状態を、Static モードによるスケッチの実行と呼びます。

Active モードでは、プログラムは永遠に実行され、動きのある画像などを描くことに適しています。このモードでスケッチを記述することにより、より高度な Processing の機能を使うことができます。

Active モードでは

```
void setup( ) {
    //最初に 1 回だけ実行される処理
}
void draw( ) {
    //繰り返し実行される処理
}
```

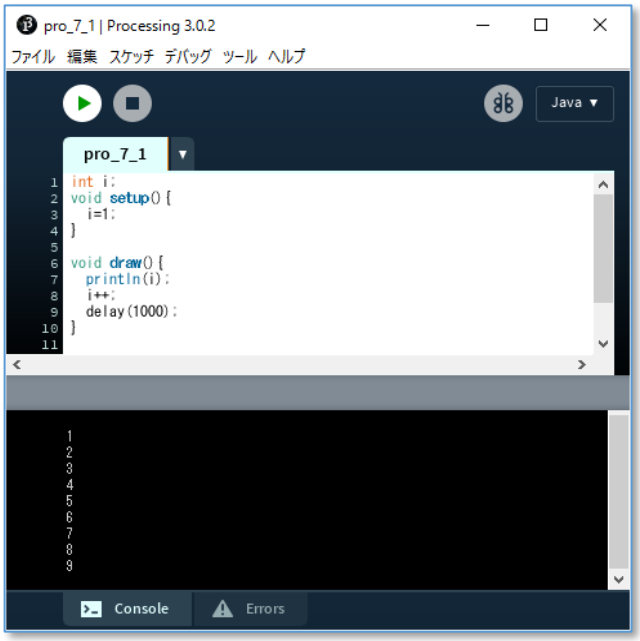
のようにスケッチを記述します。（書き方の変更だけです）

setup( ) に続く{ }の中はプログラムがスタートして最初に 1 回だけ実行したい内容を書いておきます。draw( ) に続く{ }の中には、繰り返し実行する命令を書いておきます。

② 数字を数え続けるスケッチ

次のスケッチを入力し、実行してみましょう。

```
int  i;
void setup( ){
   i=1;
}

void draw( ){
   println( i );
   i++;
   delay(1000);
}
```



整数型の i という変数は、setup と draw の両方の中で使うので、最初に宣言しておきます。

setup の中で i の値に 1 をセットし、draw の中でそれをテキストエリアに表示させています。

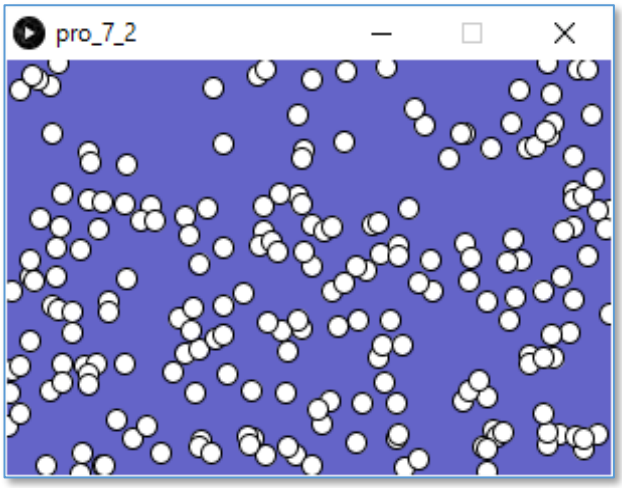
delay(1000) は 1000 ミリ秒プログラムを一時停止させる命令です。ほぼ 1 秒ごとに数字が表示されていきます。

※作成したスケッチは、"pro_ 7_ 2" として保存しておきます。

③ 円を描き続けるスケッチ

次の、円が無数に描かれるスケッチを入力し、実行してみましょう。

```
void setup( ) {
   size(300,200);
   background(100,100,200);
}

void draw( ) {
ellipse(random(300),random(200)
,10,10);
}
```



background(100,100,200) で薄い青色を背景にしています。

ellipse は楕円を描く命令です。縦、横とも 10 ドットを指定しているので、円になります。

random は 0～中の数字までの数が、ランダムに得られので、横と縦の位置を乱数で指定しています。

※作成したスケッチは、"pro_ 7_ 3" として保存しておきます。

Point Active モードと Static モードを混在して記述するとエラーになります。

```
size(300,200);
background(100,100,200);

void draw(){
      ellipse(random(300),random(200),10,10);
}
```

Static モードによる記述

混在不可

Active モードによる記述

8　キーボードから入力する

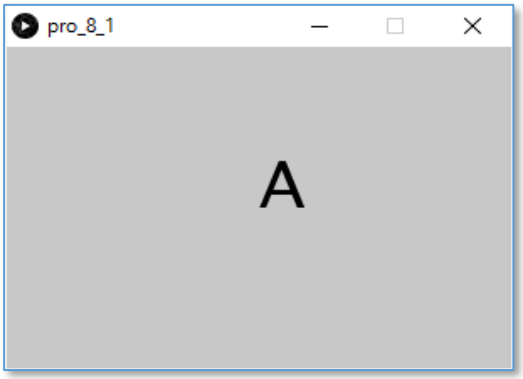
キーボードが押されたかどうか調べるものに、keyPressed というものがあります。キーボードが押されると、key に押されたキーのコードがセットされます。

①　押されたキーをウィンドウに表示する

キーボードで押されたキーをディスプレイウィンドウに表示するスケッチを入力してみましょう。
（ //の後のコメントは入力しなくてかまいません）

```
char   k;                                  //文字コードを入れる変数を準備

void setup( ) {
   size(300,200);
   fill(0,0,0);                            //文字の色を黒にする
   textSize(40);                           //文字の大きさを指定
   k=63;                                   //？の文字コード
}

void draw( ) {
   background(200,200,200);                //背景色を薄い灰色にする
   text(k,150,100);                        //(150,100)の位置に文字を表示
}

void keyPressed( ){                        //キーが押された時に呼び出される命令
   k=key;                                  //押されたキーのコードを k にセット
}
```

Point　char　k; を int　k; に書き換えて、実行してみましょう。文字に代わって文字コードが表示されることを確認します。



※作成したスケッチは、"pro_８_１" として保存しておきます。

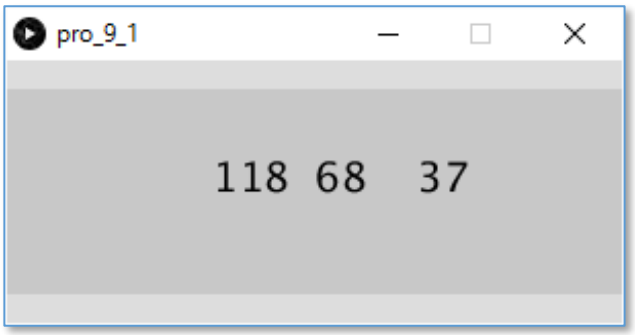
9 マウスを操作する

マウスの移動やクリックの情報をディスプレイウィンドウに表示してみましょう。

mouseX ----- ディスプレイウィンドウ内のＸ座標
mouseY ----- ディスプレイウィンドウ内のＹ座標
mouseButton --- マウスの押されたボタンに対応した数値

次のスケッチを入力し、実行してみましょう。

```
void setup( ) {
   size(300,100);
}

void draw( ){
   background(200,200,200);
   fill(0,0,0);              //文字色を黒
   textSize(20);
   text(mouseX,100,50);
   text(mouseY,150,50);
   text(mouseButton,200,50);
}
```



マウスを動かすと、ウィンドウ内の数値が変化します。
左右のボタンをクリックすると、それに対応した数値が表示されます。

※作成したスケッチは、"pro_９_１" として保存しておきます。

Point 上のスケッチで、マウスの左ボタンをクリックすると"３７"が、右ボタンをクリックすると"３９"が表示されます。

Processing のシステムの中では、"LEFT" という文字列に３７が、"RIGHT" という文字列に３９という数値がふられています。print(LEFT); と入力し実行すると、テキストエリアに 37 と表示されることからわかります。

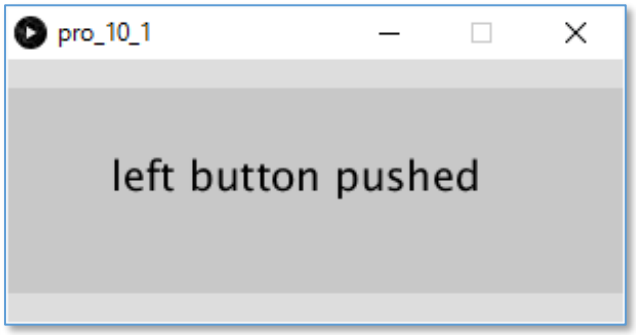
10　条件分岐をする

draw( ) の中では、上から順にプログラムが実行されていきますが、条件によってプログラムの流れを変えたい時は、if 文を使用します。

```
if（条件式）{
    条件式が成り立つ時に処理する内容
}
```

次のスケッチを入力し、実行してみましょう。

```
void setup( ) {
  size(300,100);
}

void draw( ){
  background(200,200,200);
  fill(0,0,0);
  textSize(20);
  if(mouseButton == 37){
    text("left button pushed"
    ,50,50);
  }
  if(mouseButton == 39){
    text("right button pushed"
    ,50,50);
  }
}
```



左ボタンが押された時に実行される内容

右ボタンが押された時に実行される内容

37　の代わりに LEFT、39　の代わりに RIGHT　と記述しても同じ結果になります。

＝＝　は左右の２つのものが等しいかどうかを判定しています。等しい時だけ、後の{　}に続く内容が処理されます。

※作成したスケッチは、"pro_１０_１" として保存しておきます。

（参 考）

【Processing の基本的な命令（抜粋）】

基本的な関数	size( ) ウィンドウサイズの設定	データ型	int　整数
	noLoop( ) 処理の一時停止		float　浮動小数点数
	loop( ) 処理の再開		boolean　論理値
	redraw( ) 1 回だけ draw( )を実行		char　文字
	exit( ) スケッチを終了する		String　文字列
	delay( ) 指定した時間停止する		color　色
スケッチの情報	width 描画ウィンドウの幅		PImage　画像
	height 描画ウィンドウの高さ		PFont　VLW フォント
	screenWidth スクリーンの幅	図形の描画（二次元）	point(x,y)　座標(x,y)に打点
	screenHeight スクリーンの高さ		line(x1,y1,x2,y2)　直線
	frameRate( ) フレームレートの設定		triangle(x1,y1,x2,y2,x3,y3)　三角形
	frameRate フレームレートの取得		rect(x,y,w,h)　長方形
	frameCount 累計のフレーム数		quad(x1,y1,x2,y2,x3,y3,x4,y4) 四辺形
条件分岐と繰り返し	if～else 条件分岐		arc(x,y,h,start,stop) 円弧
	for 繰り返し		ellipse(x,y,w,h)　楕円
	while 繰り返し	マウス	mouseX　マウスカーソルの水平位置
	switch( )～case 分岐処理		mouseY　マウスカーソルの垂直位置
	break ブロックからの脱出		mousePressed ボタンが押されているか
	continue 処理をスキップする		mouseButton　押されているボタン
色	background( ) 背景色を設定する	キーボード	keyPressed　キーが押されているか
	colorMode( ) カラーモードの設定		key　最後に押されたキーを表す文字
	stroke( ) 線の色を指定する		keyCode　最後に押されたキーコード
	noStroke( ) 線や輪郭を描画しない		keyPressed( ) キー押下による呼ばれる
	fill( ) 塗り色を指定する		keyReleased( ) 離れたときに呼ばれる
	noFill( ) 図形内部の色を塗らない	座標	translate(x,y) 位置をずらす
	color( ) 色を作成する		rotate(a) 回転させる
文字の出力	PFont クラス フォントを扱うクラス		scale(x,y) 拡大と縮小
	loadFont( ) フォントを読み込む	出力	print( ) コンソールへ出力
	textFont( ) 使用フォントを選択する		println( ) コンソールへ出力し改行
	createFont( ) フォントを動的に作成	保存	saveFrame( ) 連続するフレームの保存
	text( ) 文字を表示する		save( ) 現在のフレームを保存
	textSize( ) 文字の大きさを設定する	乱数	random( )　乱数の生成
	textAlign( ) 文字の揃え方を設定する		randomSeed( ) 乱数の種を設定
	textLeading( ) 行間を設定する	画像	loadImage( ) 画像を読み込む
	textWidth( ) 文字の幅を計算する		image( ) 画像を表示する

第2章（ビジュアルデザイン編）

11　色を指定する

①　色指定の方法

Processing では、次の４つの方法で色を指定します。

・グレイスケール
　黒～白への単色で色を指定。０～２５５の数値で、０は黒、２５５は白色になります。
・ＲＧＢ
　光の三原色である赤(Red)、緑(Green)、青(Blue)に色を分解し、それぞれの色の濃さを０～２５５の数値で指定します。
・ＨＳＢ
　色を色相、彩度、輝度という成分で指定します。
・不透明度（アルファチャンネル）
　色の不透明度を小さくすることにより、下の図形が透けて見えるように塗ることができます。

②　カラーモードの指定

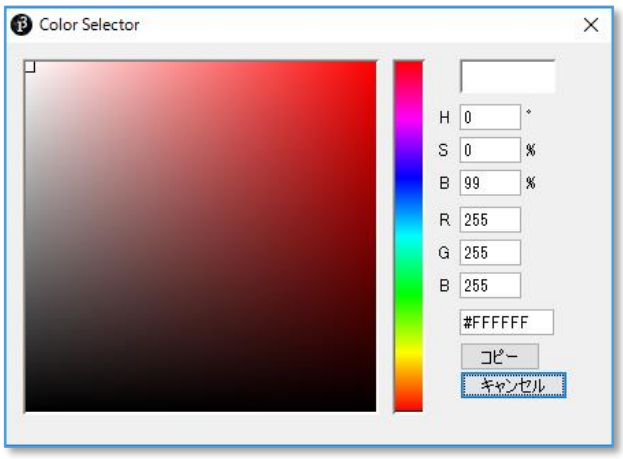
ＲＧＢ、ＨＳＢのカラーモードを指定する命令

colorMode(RGB);　　以降のカラーモードをＲＧＢ指定にする（初期値）

colorMode(HSB);　　以降のカラーモードをＨＳＢ指定にする

Processing には、プログラムで使う色を選ぶために、カラーセレクターが用意されてます。

（ツール）→（色選択）



各数値を参考にして、色を指定することができます。また、16 進数カラーコード（＃から始まる 16 進数 6 桁のコード)を利用して色を指定することもできます。

（例）background(#00FF00);　背景を緑色

Point　colorMode(HSB,360,100,100); のように、色指定の範囲を数値で指定し、何段階で色を指定するかを変更することもできます。（ＨＳＢでの指定は、通常このように指定します）

12　色を指定して図形を描く

①　グレイスケールで図形を描く

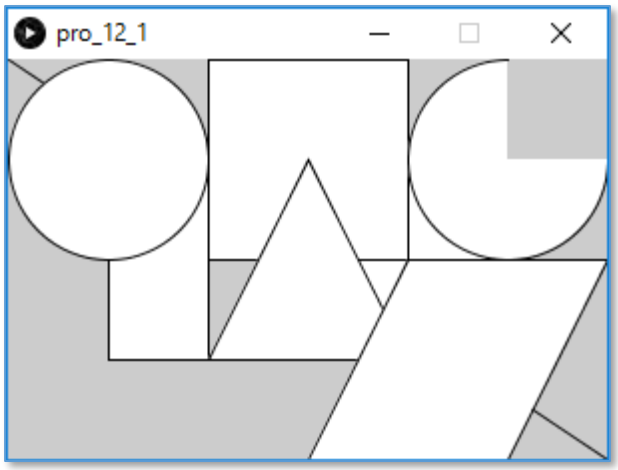
次のスケッチを１行ずつ入力して実行し、描かれる図形を確認してみましょう。

※//以下はコメントなので、入力の必要はありません。

```
size(300, 200);          //300×200 のディスプレイウィンドウを表示
point(50, 50);           //(50,50)の位置に点を表示
line(0, 0, 300, 200);          //直線
rect(150, 50, 100, 100);       //(150,50)を左上とする縦、横 100 の正方形
rectMode(CENTER);        //rectMode を CENTER に設定
rect(150, 50, 100, 100);       //中心が(150,50)で縦、横 100 の正方形
rectMode(CORNERS);       //rectMode を CORNERS に設定
rect(50, 50, 100, 150);        //左上(50,50),右下(100,150)の正方形
triangle(150, 50, 100, 150, 200, 150);              //三角形
quad(200, 100, 150, 200, 250, 200, 300, 100);              //四角形
ellipse(50, 50, 100, 100);                          //楕円（この場合は円）
arc(250, 50, 100, 100, radians(0), radians(270));       //扇形
```

最後まで入力し、実行すると右のようになります。

先に描いた図形は、後からの図形で隠れています。



正方形の描画モードは、CORNER,CENTER,CORNERS, RADIUS の４つがあります。初期値は、CORNER です。

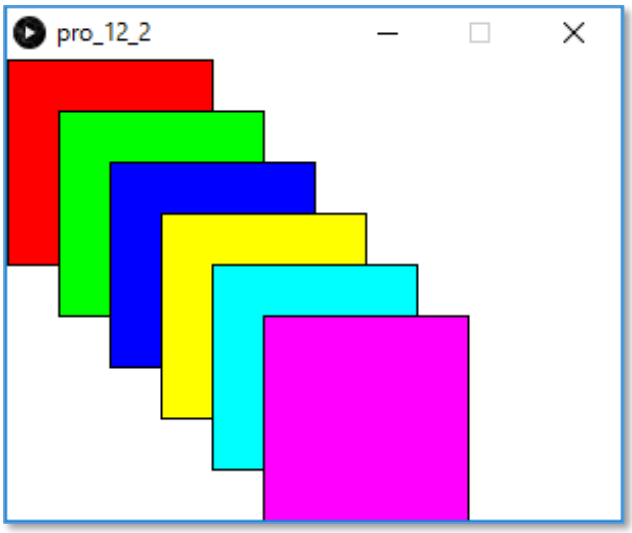
※作成したスケッチは、"pro_１２_１" として保存しておきます。

Point　background( ) 記述しない場合、background(200); の指定と同じになります。（初期値）

② 色を変えて正方形を描く

ＲＧＢカラーモード（初期値）で、いろいろな色の正方形を描いてみましょう。

```
size(300,225);
background(255);   //背景色白
fill(255,0,0);     //赤色
rect(0,0,100,100);
fill(0,255,0);     //緑色
rect(25,25,100,100);
fill(0,0,255);     //青色
rect(50,50,100,100);
fill(255,255,0);   //黄色
rect(75,75,100,100);
fill(0,255,255);   //水色
rect(100,100,100,100);
fill(255,0,255);   //紫色
rect(125,125,100,100);
```

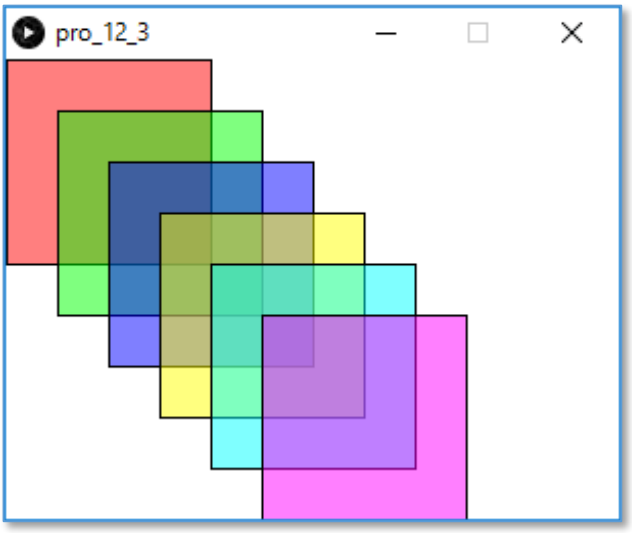


※作成したスケッチは、"pro_１２_２" として保存しておきます。

③ 不透明度（アルファチャンネル）を指定して正方形を描く

不透明度を指定して、正方形を描きます。

```
size (300,225);
background(255);
fill(255,0,0,128);
rect(0,0,100,100);
fill(0,255,0,128);
rect(25,25,100,100);
fill(0,0,255,128);
rect(50,50,100 size,100);
fill(255,255,0,128);
rect(75,75,100,100);
fill(0,255,255,128);
rect(100,100,100,100);
fill(255,0,255,128);
rect(125,125,100,100);
```



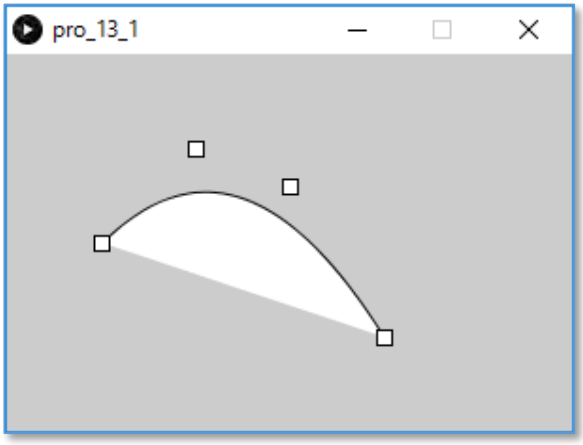
※作成したスケッチは、"pro_１２_３" として保存しておきます。

13　ベジェ曲線を描く

ベジェ曲線は、コンピュータ上で滑らかな曲線を書く場合に用いられる手法で、Processing では、bezier( ); という命令が用意されています。

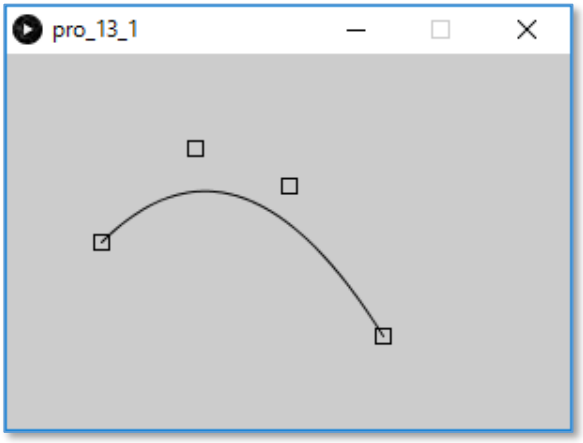
下のスケッチを入力し、ベジェ曲線を描いてみましょう。

```
size(300, 200);
//noFill( );
bezier(50, 100, 100, 50, 150, 70,
200, 150);

//指定した点の表示
rectMode(CENTER);
rect(50, 100, 8, 8);
rect(100, 50, 8, 8);
rect(150, 70, 8, 8);
rect(200, 150, 8, 8);
```



小さな□は、指定した点の位置を示しています。

noFill( ); を指定すると、線だけが表示されます。（右図）



※作成したスケッチは、"pro_１３_１" として保存しておきます。

14　beginShape( )命令を使う

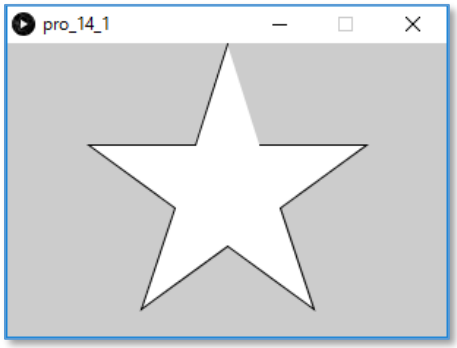
beginShape( )～endShape( ) の間に複数の点を指定することで、それらを結ぶ多角形や曲線を描くことができます。

① 直線による複雑な図形

vertex( ) は、直線で結ぶ点を指定する命令です。
次のスケッチを入力し、星を描いてみましょう。

```
//beginShape 命令

size(300,200);
//fill(0,255,255);

beginShape( );
   vertex(150,0);
   vertex(128,69);
   vertex(55,69);
   vertex(114,112);
   vertex(91,181);
   vertex(150,138);
   vertex(209,181);
   vertex(186,112);
   vertex(245,69);
   vertex(172,69);
endShape(CLOSE);
```

CLOSE は図形を閉じる指定です。endShape(); とすると最後の位置の直線は描画されません。



※作成したスケッチは、"pro_１４_１" として保存しておきます。

② 曲線による複雑な図形

①で作成したスケッチをもとに、曲線による複雑な図形を作成します。

vertex( ) を curveVertex( ) に変更するとともに、始点と終点では、それぞれ前後の点の座標を書き加えます。

```
//beginShape 命令

size(300,200);
//fill(0,255,255);

beginShape();
   curveVertex(172,69);
   curveVertex(150,0);
   curveVertex(128,69);
   curveVertex(55,69);
   curveVertex(114,112);
   curveVertex(91,181);
   curveVertex(150,138);
   curveVertex(209,181);
   curveVertex(186,112);
   curveVertex(245,69);
   curveVertex(172,69);
   curveVertex(150,0);
   curveVertex(128,69);
endShape(CLOSE);
```

fill(0,255,255) を指定すると、図形の中を指定の色で塗ることができます。

※作成したスケッチは、"pro_１４_２" として保存しておきます。

15　座標を変換する

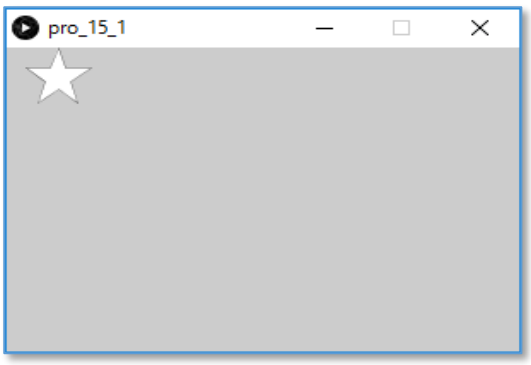
①　座標を移動する

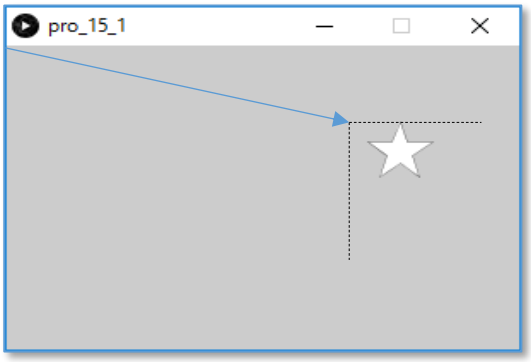
translate(x,y) は座標そのものを移動する命令で、それ以後の描画の命令は移動した座標で実行されます。

scale( ) によって座標軸の比率を変え、図形を拡大・縮小することができます。

星を描く pro_14_1 をもとに、次のスケッチを実行してみましょう。

```
size(300,200);
//translate(200, 50);
scale(0.2);

beginShape( );
   vertex(150,0);
   vertex(128,69);
   vertex(55,69);
   vertex(114,112);
   vertex(91,181);
   vertex(150,138);
   vertex(209,181);
   vertex(186,112);
   vertex(245,69);
   vertex(172,69);
endShape(CLOSE);
```





scale(0.2) によって、図形の縮尺が１／５になり、星が小さくなります。

translate(200,50) によって、座標軸が横 200、縦 50　移動した位置に設定されています。

注）上図は、scale(0.2) だけを指定した画面。下図は、translate(200,50) も実行している。

※作成したスケッチは、"pro_１５_１" として保存しておきます。

Point 座標系を初期化するためには、resetMatrix(); 命令で行います。この命令が記述されていると、それ以降の描画は初期化された座標系で行われます。

② 座標を回転させる

rotate( ) は座標軸を回転させる命令です。次のスケッチでは、６０度回転させながら、６つの星を描いています。

pro_15_1 をもとに、命令を追加して、６つの星を描いてみましょう。

```
size(300,200);
translate(200, 50);
scale(0.2);

for (int n=0; n<6; n++) {
beginShape( );
   vertex(150,0);
   vertex(128,69);
   vertex(55,69);
   vertex(114,112);
   vertex(91,181);
   vertex(150,138);
   vertex(209,181);
   vertex(186,112);
   vertex(245,69);
   vertex(172,69);
endShape(CLOSE);
rotate(radians(60));
}
```

※作成したスケッチは、"pro_１５_２" として保存しておきます。

背景色を白にし、塗り色を乱数で指定し、回転させる角度を工夫すると、右図のような模様を描くことができます。

（追加する命令）

fill(random(255), random(255), random(255), 128);

第3章（アニメーション編）

16 イメージを表示する

写真や図の位置を変化させたり、表示するイメージを連続的に変化させることで、アニメーションを表現できます。写真や図はイメージオブジェクトとしてディスプレイウィンドウに表示されます。

イメージオブジェクトは

PImage （イメージオブジェクト名);

という形で宣言します。イメージオブジェクトは、イメージデータを入れておく入れ物のようなものです。

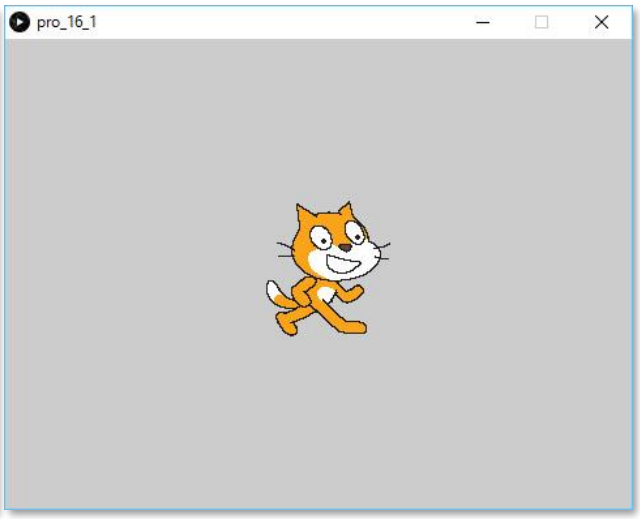
イメージオブジェクトが準備できたら、そこにイメージデータを読み込みます。

(イメージオブジェクト名) = loadImage(“イメージファイル名”);

最後に、image( ) 命令により、ディスプレイウィンドウに表示します。

プログラムの記述は以下のようになります。ここでは、Scratch で登場するキャラクター(cat) のイメージデータを読み込み、表示しています。

```
PImage cat;            //cat は任意の名前

void setup( ){
   size(480,360);
   cat = loadImage("cat1a.gif");
}

void draw( ){
   image(cat,200,120);
}
```



※作成したスケッチは、"pro_１６_１" として保存しておきます。

ねこのイメージデータ”cat1a.gif”を、プログラムを保存したフォルダにコピーします。

実行すると、ディスプレイウィンドウの中央にねこのイメージが表示されます。

※cat データのダウンロード先

http://ymdnet.cho88.com/processing/cat.zip

draw( ) の中は繰り返し実行されています。同じイメージを描き続けているので、cat は止まって見えています。image はイメージオブジェクトをディスプレイウィンドウに表示する命令です。

17「ねこが歩くアニメーション」を作成する

ねこが歩く動きを表すために、cat の位置を変数で指定し、変数の値を変化させることで移動する様子を表現します。

① cat を動かしてみる

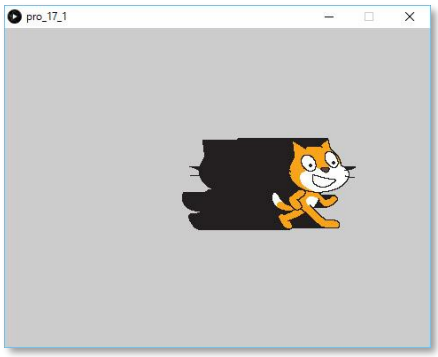
```
PImage cat;
int x,y;
void setup( ){
   size(480,360);
   x = 200;
   y = 120;
   cat = loadImage("cat1a.gif");
}

void draw( ){
   image(cat,x,y);
   x++;
}
```

cat の縦、横の位置を記録する変数

最初の位置を指定

変数に書き換える

x に＋１を加える
(cat は右へ移動)



【実行したイメージ】

draw( )はプログラムが停止するまで、動き続けています。cat は画面から飛び出して見えなくなります。また、動いた跡が黒くなってしまいます。

draw( )の中で、background(200); を指定すると、draw( )のループ１回ごとに、ウィンドウを灰色で描き直すので、黒い軌跡はなくなります。

② cat をウィンドウの中で動かし続ける

cat が画面からはみ出さないように、端まで来たら向きを変えるようにしてみます。

移動する方向を表す変数として houkou を定義し、条件判断をするために、if という命令を使います。

```
if(条件){
  //条件が成り立った時に実行
}
```

```
PImage cat;
int x,y;
int houkou;

void setup( ){
   size(480,360);
   x = 200;
   y = 120;
   houkou = 1;
   cat = loadImage("cat1a.gif");
}
void draw( ){
   background(200);
   image(cat,x,y);
   x = x + houkou;
   if (x>390 || x<0 ) {
      houkou =  −houkou;
   }
}
```

移動方向の変数として、houkou を定義

＋１で右方向、－１で左方向にする
（最初は右方向）

１のときは１ずつ増え、－１の時は減っていく

x が 390 より大きいか、または、 x が 0 より小さい場合
＋１と－１が交互に入れ替わる

朱文字の部分を書き加え、プログラムを実行してみましょう。

※作成したスケッチは、"pro_１７_２" として保存しておきます。

〖プログラムに挑戦！〗

（次ページのスケッチを参照）

左に移動する時は、左向きのねこのイメージ“cat2a.gif”が表示されるようにスケッチを変更してみましょう。

また、足の形が違うねこのイメージを交互に表示させることにより、足が動いているようにしてみましょう。（“cat1b.gif”，“cat2b.gif”を使用）

```
PImage cat1a,cat1b,cat2a,cat2b;
int x,y;          //cat の位置座標
int mx,my;        //マウスの座標
int houkou;       //cat が動く方向　1 で右　-1 で左
int ashi;         //足の形（0 と 1 で判定）
void setup(){
  size(480,360);
  background(255);  //背景は白色
  x = 200;
  y = 120;
  houkou = 1;  //最初は右へ
  ashi = 0;
  button_disp();  //スタート、ストップボタン表示

  //cat イメージの読み込み
  cat1a = loadImage("cat1a.gif");
  cat1b = loadImage("cat1b.gif");
  cat2a = loadImage("cat2a.gif");
  cat2b = loadImage("cat2b.gif");
  noLoop();
}

void draw(){
  background(255);
  button_disp();
  if(houkou == 1){
    if(ashi == 0){
      image(cat1a,x,y);
      ashi = 1;      //足の入れ替え
    }else{
      image(cat1b,x,y);
      ashi = 0;      //足の入れ替え
    }
  }
  if(houkou == -1){
    if(ashi == 0){
      image(cat2a,x,y);
      ashi = 1;      //足の入れ替え
    }else{
      image(cat2b,x,y);
      ashi = 0;      //足の入れ替え
    }
  }
  x = x + houkou * 10;   // 1 0 歩進む
  if(x<0 || x>390){
    houkou = -houkou;    //方向を反転
  }
  delay(100);                 //0.1 秒停止
}

void mousePressed(){
  mx = mouseX;
  my = mouseY;
  if(mx>430 && mx<445 && my>10 && my<22)
    loop();
  if(mx>452 && mx<468 && my>7 && my<23)
    noLoop();
}

//スタートとストップのボタン
void button_disp(){
  stroke(0);
  fill(0,255,0);
  rect(430,10,15,12);
  fill(255,0,0);
  ellipse(460,15,15,15);
}
```

イメージオブジェクトの宣言（左向き、右向きと、足の形の違い）

最初の位置を指定

スタート、ストップボタンは何度も表示するので、まとめて関数にしておく（関数を呼び出している）

cat1 は右向き

cat2 は左向き

最初は停止させておく（draw がループしない状態）

背景色を白色（画面が一旦消去される）
ボタンを描き直す

右向きに移動しているときの処理

※足の形が異なるイメージを交互に表示

左向きに移動しているときの処理

※足の形が異なるイメージを交互に表示

方向が＋1 で右方向へ、－1 で左方向へ移動
左端または右端まできたら方向を変える

マウスがクリックされたときに呼び出される関数
マウスの X 座標を変数 mx にセット
マウスの Y 座標を変数 my にセット
スタートボタンの位置がクリックされた

ストップボタンの位置がクリックされた
draw( ) を停止

緑のスタートボタンを描画

赤のストップボタンを描画

18 フレームの保存とムービーの作成

① フレームを保存する

ディスプレイウィンドウのイメージを保存する命令として、save( ) と saveFrame( ) があります。

save( ) ---- 現在のフレームを保存

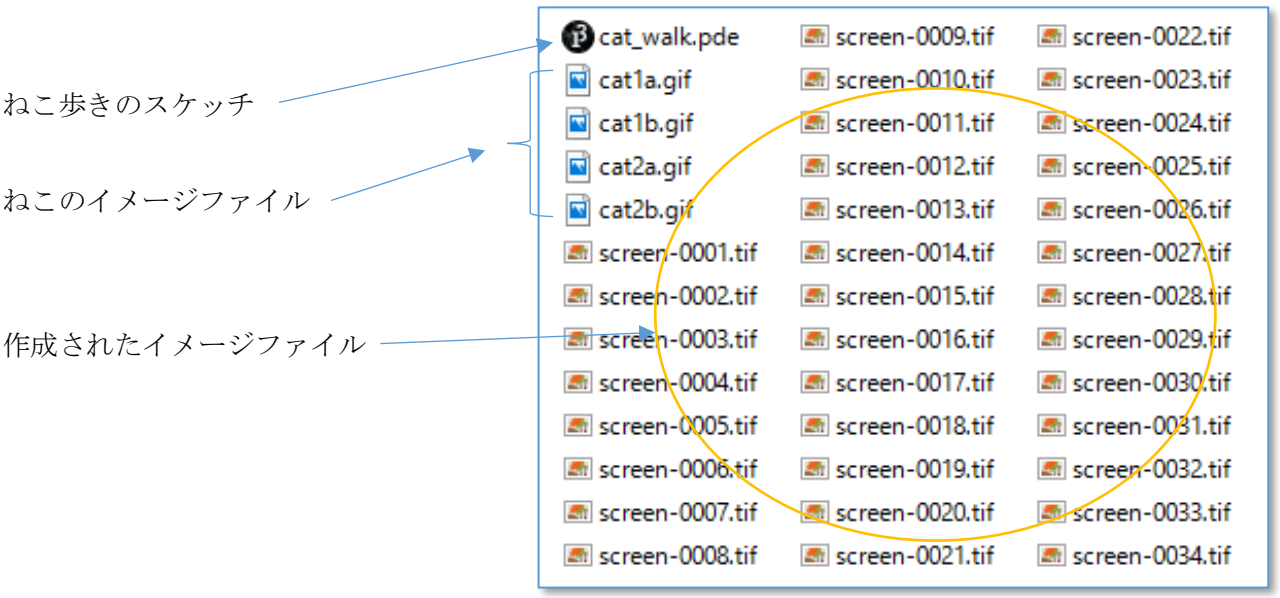
saveFrame( ) ---- 連続するフレームを保存（自動的に連番が振られます）

ファイル形式は、tiff , jpeg , png 形式を指定することができます。（初期値：tiff）

ねこが歩くアニメーションの Processing のプログラム（スケッチ）から、ムービーを作成するために、連続したフレームを保存します。

draw( ) ループの最後に、次の一行を追加し、実行すると、スケッチが保存されているフォルダの中に、次々とイメージが保存されていきます。

```
saveFrame( );
```



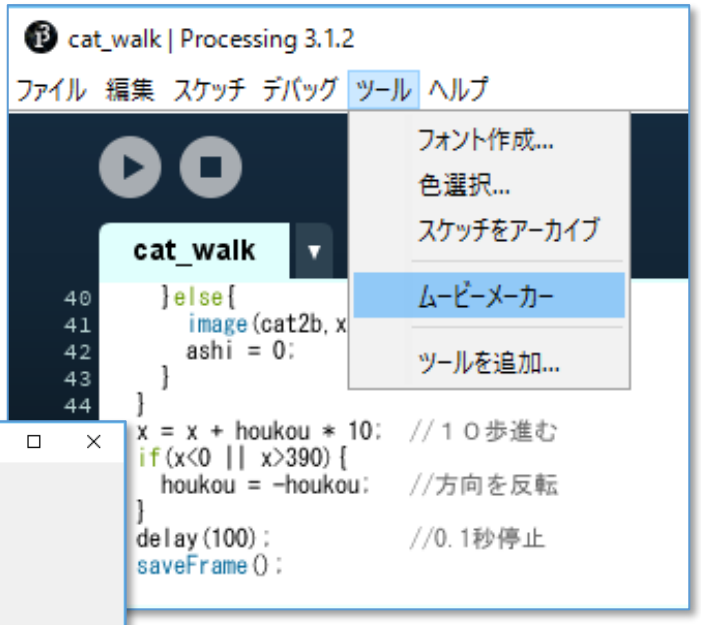
ファイル名を指定してフレームを保存する場合は次のようにスケッチに記述します。

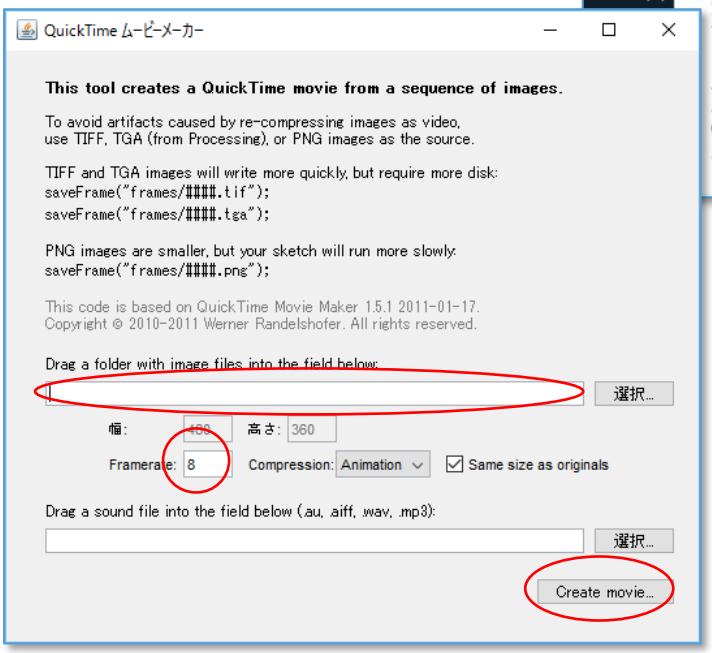
```
saveFrame("cat-#####.png")
```

png 形式のファイルが、cat-00001.png , cat-00002.png , cat-00003.png のように作成されます。

② ムービーを作成する

連続したスクリーンショットからムービーを作成します。
（ツール）→（ムービーメーカー）





［選択］ボタンをクリックし、イメージファイルが入っているフォルダを指定します。

Framerate に適当な数値（ここでは８）を入力し、［Create movie］ボタンをクリックして、ムービーを作成します。

※作成されるファイルは、mov 形式のファイルです。

※mov 形式を wmv 形式に変換した動画の掲載サイト
http://ymdnet.cho88.com/processing/cat_walk.wmv

第４章（オブジェクト指向編）

19　風船割りゲームを作る

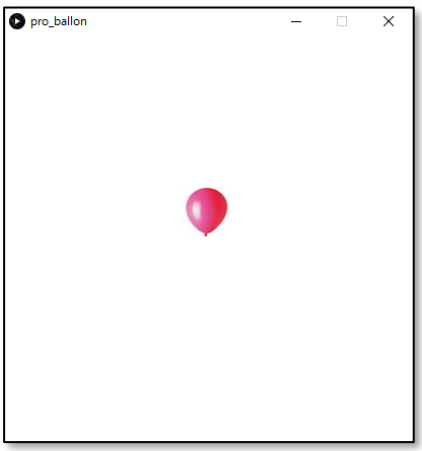
画面に表示される風船をマウスでクリックして割るゲームを、オブジェクト指向の考え方で作成していきます。

まず、オブジェクト指向の考え方を使わない「Ballon1」を作成します。

（ゲームの流れ）

縦、横 400 のウィンドウが開き、中央に風船が表示される。風船をマウスのクリックで割り、ポイントをためていく。風船は、２秒毎に位置が変わるので、マウスを素早く操作してクリックする。

20 秒でゲームは終了し、ポイントが表示される。



割れた風船を表すイメージ　→　 

（必要な命令）

・イメージ（JPEG ファイル）の表示

Pimage photo　(photo は任意の名前)	※イメージを扱う部品を準備
photo = loadImage("ファイル名")	※イメージファイルを読み込む
image(photo,x,y)	※x,y の位置に表示

・フレームの表示間隔と表示回数

frameRate(数字)	※１秒間に draw がフレームを描く回数 framerate(30) が初期値
frameCount	※フレームが draw によって表示された回数

（スケッチ作成の考え方）

draw は for や while のように繰り返し実行される processing 独特の命令です（もともと、ビジュアルデザイン用の言語なので、繰り返し描画する機能として備わっている）。frameRate(5) とすると、0.2 秒に１回ループすることになるので、２秒は 10 回のループになる。これを数える変数（次頁のスケッチでは"n"）により、２秒毎に風船の位置を変化させる。

風船の大きさは、50×50 で作成してある。

スケッチの例は次頁にあるが、実際の動きをサンプルファイルをダウンロードし、確かめてみよう。

風船割りゲームダウンロード　http://ymdnet.cho88.com/processing/ballon.zip

【風船割りゲーム「Ballon1」プログラムリスト】

```
PImage b1,b2,b3;                          //風船、割れた風船を入れる部品を準備
int x,y;
float kyori;                              //風船とマウスの間の距離を入れる変数
int n;
int tokuten;                              //得点を入れる変数

void setup(){
  size(400,400);
  background(255,255,255);
  b1 = loadImage("ballon1.jpg");
  b2 = loadImage("bomb.jpg");
  b3 = loadImage("clear.jpg");
  n=0;
  tokuten=0;
  x=170;
  y=170;
  frameRate(5);
}

void draw(){
  background(255,255,255);
  image(b1,x,y);                          //風船をｘ、ｙの位置に表示
  n++;                                    //draw が回った回数を数える
  if(n>10){
    x=(int)random(340);                   //(int) は実数を整数型に変換して、
    y=(int)random(340);                   //ｘ、ｙに入れるためのもの
    n=0;                                  //１０回数えたら０に戻す
  }
  if(frameCount>100){
    fill(255,0,0);
    textSize(30);
    text("game over   "+tokuten+" point get!",10,50);
    noLoop();                             //draw のループを止める命令
  }
}

void mousePressed() {                     //マウスがクリックされた時に実行される
  kyori=dist(mouseX,mouseY,x+25,y+25);
  if(kyori<25){
    image(b2,x,y);                        //割れた風船を表示
    tokuten++;                            //ポイントを加算
  }else{
    image(b3,x,y);                        //風船を消す（はずれ）
  }
  x=(int)random(350);
  y=(int)random(350);
  n=0;
}
```

20 Balloon クラスを作る

「Processing プログラミング入門」の第１２章で取り上げた「風船割りゲーム」を、オブジェクト指向の考え方を取り入れたプログラムで作成していきます。

このプログラムは、２個の風船をマウスのクリックで割って、ポイントしていくゲームで、約２０秒で終了し、得点が表示されます。(表紙の挿絵参照)

まず、完成した Windows(32)用の exe ファイルをダウンロードし、作成するプログラムのイメージをつかんで下さい。

風船割りゲーム(balloon2)　http://ymdnet.cho88.com/processing/balloon2.zip

①　Balloon クラスの設計

Balloon クラスには、最低限、以下の機能をもたせることにします。

＜フィールド＞

・Balloon の位置の座標（横:x 縦:y）

・Balloon のイメージファイル名（jpeg ファイル等）

＜メソッド＞

・Balloon を表示する

```
class Balloon{
   PImage p;
   private int x,y;
   public Balloon(int x,int y,String s){
      this.x = x;
      this.y = y;
      p = loadImage(s);
   }
   public void show(){
      image(p,x,y);
   }
}
```

・イメージオブジェクトの宣言

・フィールド（メンバ変数）

・コンストラクタ

・メソッド

Point クラスとはオブジェクトを作るための設計図にあたります。

② Balloon のインスタンスの作成

Balloon クラスを使って、風船を表示するためには、そのインスタンスを作成する必要があります。インスタンス作成の基本的な手続きは、

```
Ｂａｌｌｏｏｎ　ｂａｌ；
ｂａｌ　＝　ｎｅｗ　Ｂａｌｌｏｏｎ（１７０，１７０，”ｂａｌｌｏｏｎ１．ｊｐｇ”）；
```

という形になります。インスタンス名”ｂａｌ”は任意の名前です。複数のインスタンスを作る場合は、”ｂａｌ１”,”ｂａｌ２”などとします。

作成した風船を表示するには、ｓｈｏｗメソッドを使用します。

```
ｂａｌ．ｓｈｏｗ（）；
```

Processing のスケッチとしては、以下の様になります。入力し、風船が１個表示されることを確かめましょう。

```
Balloon bal1;

void setup(){
  size(400,400);
  background(255);              //背景を白色
  bal1 = new Balloon(170,170,"balloon1.jpg");
}

void draw(){
  bal1.show();                  //bal1 をウィンドウに表示
}

class Balloon{
  PImage p;
  private int x,y;

  public Balloon(int x,int y,String s){
    this.x = x;
    this.y = y;
    p = loadImage(s);
  }

  public void show(){
    image(p,x,y);
  }
}
```

Point balloon2 として保存し、同じ場所に"balloon1.jpg"のファイルを入れておきます。

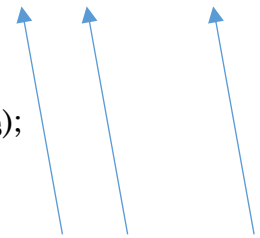
インスタンスとはクラスをもとに作成されたオブジェクトの実体のことです。

③　コンストラクタの役割

public Balloon( )は Balloon クラスのコンストラクタといいます。コンストラクタは、そのクラスを使ってインスタンスを作成する時に、最初に呼び出されて実行されるもので、クラス名と同じ名前がついています。

Balloon クラスのコンストラクタが呼び出される時の動きです。

```
public Balloon(int x,int y,String s){
   this.x = x;
   this.y = y;
   p = loadImage(s);
}

bal1 = new Balloon(170,170,"balloon1.jpg");
```



170 の数値(画面のほぼ中央の位置) が、コンストラクタの x と y に渡されます。風船の絵の jpeg ファイル名が、文字列を格納する s に入ります。

引数の x, y と Balloon クラスの x, y はここでは同じ記号を使っているので、区別するために this が使われています。this は現在処理を行っているオブジェクトを示すキーワードです。

引数は仮の名前なので、x, y を使わずに他の文字を用いた場合は、区別するための this は必要なくなります。

```
public Balloon(int a,int b,String s){
   x = a;
   y = b;
   p = loadImage(s);
}
```

p はイメージオブジェクトとして startup()の中で定義されているので、balloon1. jpg がセットされます。(ファイルが同じフォルダにある必要があります)

Point 引数に指定するデータの型は、コンストラクタの引数の型と同じ必要があります。

170.0　などとするとエラーになります。

21 コンストラクタのオーバーロード

① 複数のコンストラクタを作る

引数が異なるコンストラクタを作っておくと、インスタンスを作成する時に便利です。

前頁の例では、座標 x と y、画像ファイル名を必ず引数として指定する必要がありますが、引数の無いコンストラクタや座標だけのコンストラクタを作成しておくこともできます。

```
public Balloon(){
  x = 170;
  y = 170;
  p = loadImage("balloon1.jpg");
}
```

引数が無い場合は、初期値として、座標(170,170)、イメージファイルは balloon1.jpg となる。

```
public Balloon(int x,int y){
  this.x = x;
  this.y = y;
  p = loadImage("balloon1.jpg");
}
```

引数が x と y だけの場合は、イメージファイルを balloon1.jpg とする。

上記のコンストラクタを Balloon クラスに追加しておくと、

```
bal1 = new Balloon();
bal1 = new Balloon(170,100);
```

という記述で、Balloon のインスタンスを作成することができます。

同じ名前で、引数の数や型が異なるコンストラクタを複数作成することを、コンストラクタのオーバーロードといいます。

Point 引数の無いコンストラクタは、次の様に記述することもできます。

```
public Balloon(){
    this(170,170);
}
```

※現在処理を行っているオブジェクトの、引数が２つあるコンストラクタを呼び出している。

② 2つの風船を異なるコンストラクタで作る

次のスケッチを入力し、Balloon クラスによって作成した風船を２つ表示してみましょう。
（赤文字は追加する部分を示しています）

```
Balloon bal1,bal2;

void setup(){
  size(400,400);
  background(255);
  bal1 = new Balloon();          //170,170,balloon1.jpg で作成される
  bal2 = new Balloon(170,100,"balloon2.jpg");
}

void draw(){
  bal1.show();
  bal2.show();                   //風船２を表示
}

class Balloon{
  PImage p;
  private int x,y;

  public Balloon(){              //引数なしのコンストラクタ
    this(170,170);
  }

  public Balloon(int x,int y){   //引数が２つあるコンストラクタ
    this.x = x;
    this.y = y;
    p = loadImage("balloon1.jpg");
  }

  public Balloon(int x,int y,String s){
    this.x = x;
    this.y = y;
    p = loadImage(s);
  }

  public void show(){
    image(p,x,y);
  }

}
```

22 Balloon クラスに機能を追加する

① 位置を変えるメソッドを追加する

Balloon クラスの中に、以下のメソッドを追加します。

random(数字)は０～数字の間の乱数を発生させます。Balloon の座標として整数(int 型)を使っていますので、(int)を付けることによって、整数に変換しています。

```
public void move(){
   x=(int)random(350);
   y=(int)random(350);
}
```

② マウスの位置との距離を返すメソッドを追加する

Balloon クラスの中に、以下のメソッドを追加します。

マウスの位置は、processing では、mouseX と mouseY で得られます。これを引数として、Balloon との距離を返すメソッドを追加します。

mx, my はマウスの位置を受け取る変数です。

```
public double distance(int mx,int my){
   double kyori = dist(mx,my,x,y);
   return kyori;
}
```

Point dist(x1,y1,x2,y2) は Processing の関数で、座標(x1,y1)と(x2,y2)の距離を返す関数です。

23 風船割りゲーム「Balloon2」を完成する

以下のスケッチは、２個の風船をマウスでクリックして割り、ポイントを得るゲームの全リストです。時間は約２０秒間で、終了と同時にポイントが表示されます。

```
//Balloon クラスを用いた風船割りゲーム（風船２個）

PImage bomb;                //風船が破裂したイメージ
int n;                      //draw()のループ回数
int tokuten;
Balloon bal1,bal2;

void setup(){
  size(400,400);
  background(255);
  bal1 = new Balloon();
  bal2 = new Balloon(170,100,"balloon2.jpg");
  bomb = loadImage("bomb.jpg");
  n=0;
  tokuten=0;
  frameRate(5);             //draw()を１秒間に５回ループ
}

void draw(){
  background(255);
  bal1.show();
  bal2.show();
  n++;
  if(n>10){                 //２秒で風船は移動
    bal1.move();
    bal2.move();
    n=0;
  }
  if(frameCount>100){       //２０秒で終了
    fill(255,0,0);
    textSize(30);
    text("game over  "+tokuten+" point get!",10,50);
    noLoop();
    //exit();
  }
}

void mousePressed() {
  int mx = mouseX-25;       //風船の大きさは 50×50,描画左上の座標に変換
  int my = mouseY-25;

  if(bal1.distance(mx,my)<25){      //マウスと風船の位置が近い
    image(bomb,bal1.x,bal1.y);
    tokuten++;
    bal1.move();
    n=0;
```

```
    }
    if(bal2.distance(mx,my)<25){
      image(bomb,bal2.x,bal2.y);
      tokuten++;
      bal2.move();
      n=0;
    }
  }


  //*********** class 定義 ************
  class Balloon{
    PImage p;
    private int x;
    private int y;

    public Balloon(){
      this(170,170);
    }

    public Balloon(int x,int y){
      this.x = x;
      this.y = y;
      p = loadImage("balloon1.jpg");
    }

    public Balloon(int x,int y,String s){
      this.x = x;
      this.y = y;
      p = loadImage(s);
    }

    public void show(){
      image(p,x,y);
    }

    public void move(){
      x=(int)random(350);
      y=(int)random(350);
    }

    public double distance(int mx,int my){
      double kyori = dist(mx,my,x,y);
      return kyori;
    }

}
```