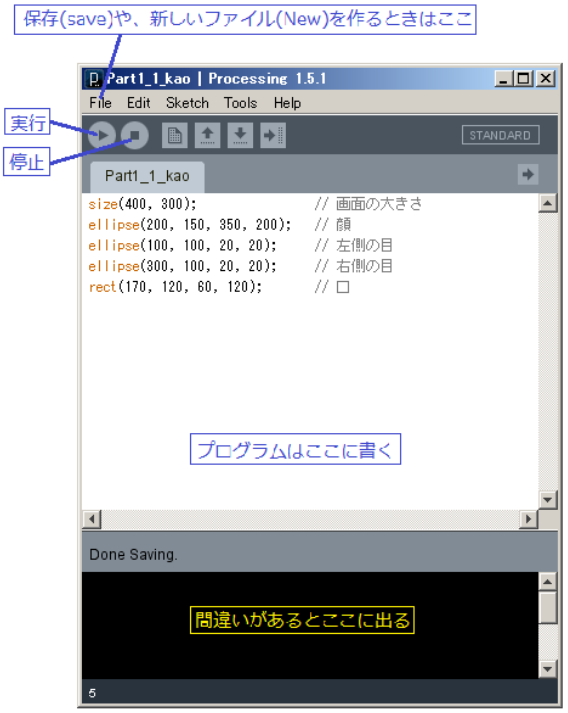
Part1　プログラムの書き方

目次

1 プログラムで絵を描く

教材の中の Part1_1_kao フォルダにある Part1_1_kao.pde を開こう。



このような画面が開くはずだ。さて、いくつかの基本操作を練習しよう。

・実行ボタンを押す。(° д °)みたいな顔が表示されるはずである。

・停止ボタンを押して終わる。

・左上の File メニューから Save As（名前をつけて保存）を選択して、
ファイル名を「Part1_1_kao2」として保存しよう。

・5 行目の rect...の次の行に、英語の「a」を書こう。実行ボタンを押すと

unexpected token: a

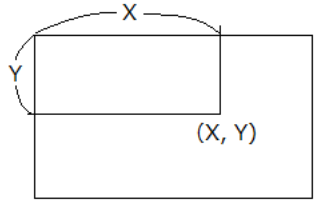
というメッセージが出るはずだ。これは「a とかいう変な文字がある」という意味で、
プログラムにミスがある（**バグ**という）とこのようにメッセージが表示される。
プログラムは一つでもミスが入っていると動かないので、a は消しておこう。

次に、キー操作の練習をしよう。

・５行目の rect...の手前に「 // 」を書こう。（//rect...のようになる）
　こうすると口が消えるはずである。実はこの rect は口を描いている命令文で、
　「//」はそこから行の終わりまでを無視するという意味である（**コメント**という）。
・４行目を**コピー**して（Edit メニューの Copy)、
　最後の行の次の行に**貼りつけ**（Edit メニューの Paste）よう。
　コピー、貼りつけがわからない人は手を上げてほしい。
　そして、次のように書き換えて実行してみよう。

```
ellipse(200, 170, 80, 120);
```

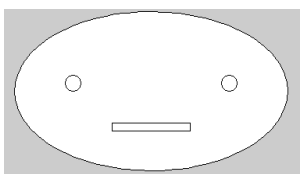
ここで、プログラムの意味を説明する。
まず、Processing ではウィンドウ上の位置を２つの数字のペアで表す。
（X, Y)はウィンドウ左端から X、上端から Y の位置である。



このプログラムの全ての行は命令文である。
１行目の size(400, 300);は**ウィンドウの大きさ**を
横幅 400、縦幅 300 にしろという命令である。
ellipse(x, y, w, h);は**だ円**を描く命令で、(x, y)はだ円の**中心**の位置を、(w, h)はだ円の横の長さと縦の長さを表す。
rect(x, y, w, h);は**長方形**を描く命令で、(x, y)は長方形の**左上の頂点**の位置を、(w, h)は長方形の横の長さと縦の長さを表す。

プログラムは上に書いた命令から順に実行される。次の操作をやってみよう。

・２行目と３行目を入れ替えてみよう。左の目が消えるはずである。これは、左の目を描いた**上から重ねて**顔のだ円を描いてしまったからである。
・次の顔の形になるようにプログラムを書き換えよう。



1-1　図形を描く

図形を描く命令は次のとおりである。

size(W, H)命令

画面の横幅を W,縦幅を H にする。

point(X, Y)命令

(X, Y)に点を書く

line(X0, Y0, X1, Y1)

(X0, Y0)から(X1, Y1)に線を引く。

rect(X, Y, W, H)

（X, Y)を左上の頂点とした、横 W、高さ H の長方形を描く。

ellipse（X, Y, W, H)

(X, Y)を中心とした、横 W、高さ H のだ円を描く。

画面は一種の「黒板」だと思うと良い。図形を描く命令を実行するたびに、上に重ねて図形が描かれていく。そして消さない限り残り続ける。

※セミコロンについて

プログラムのあらゆる行の終わりにセミコロン「；」が付いているのがわかるだろう。

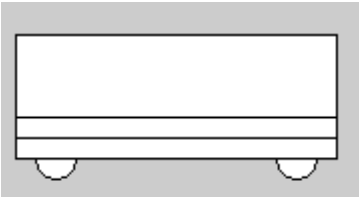
今日これから学ぶ if 文や for 文などを除いて、すべての行の終わりには「；」をつける。

課題「Part1_1_zukei」

新しくプログラムを作る。

File メニューから New を選んで、「Part1_1_zukei」で作成しよう。

・下のような電車の絵を表示せよ。余裕があったら窓などをつけてみよう。



1-2　色をつける

次に図形に色をつける方法を勉強する。

色の指定は次の４つの命令で行う。りんかく線と内部の塗りつぶしを別々に決める。

stroke(R, G, B) ・・・りんかく線の色をR,G,Bで指定

fill(R, G, B)・・・塗りつぶしの色をR,G,Bで指定

noStroke()・・・りんかくなし

noFill()・・・塗りつぶしなし(りんかくだけになる)

R,G,Bは0~255の数を入れる。Rが大きいほど赤色に、Gが大きいほど緑色に、Bが大きいほど青色になる。

これらの命令は、**それ以降の命令全てに影響する**ことに注意しよう。

例えば、stroke(255, 0, 0);を実行すると、次にstrokeかnoStroke命令が実行されるまで、描かれる図形のりんかく線は全て赤色になる。

課題

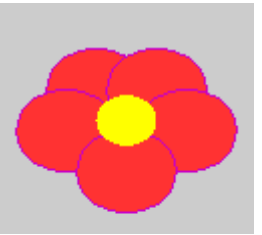
・教材の「Part1_1_France」を参考にして、ドイツの国旗を描いてみよう。

　ファイル名は「Part1_1_German」とする。

・スイカと梅を描く

ここから先、課題のファイル名は適当につけてください。

半円は円を長方形で切る。梅は自信のある人はやってみよう。



2　計算

```
size(200, 200);

int cx, cy;
cx = 100;
cy = 100;
int d = 50;

noFill();

ellipse(cx, cy, d, d);
ellipse(cx, cy, d*2, d*2);

d = d+100;
ellipse(cx, cy, d, d);
```

教材の「Part1_2_example1」を開こう。
実行すると３重の円が描かれるはずだ。
ここでのポイントは、**変数**と**計算**である。

このプログラムにはellipse命令が３つある。
さっきまではellipseの中には数字が入っていたのだが、
今度は英語や記号が入っている。

1つ目のellipseを見ると、横幅がcx、縦幅がcy、縦横の長さがdとなっていることが分かる。これらの英語は**変数**といって、数をひとつ**記憶する**役割を持っている。
3行目ではcx, cyという名前の変数を作っている。**int**というのは整数(..., -1, 0, 1, 2, …)を記憶する変数を作るという意味で、あとに来るcxとcyが作られる変数の名前である。
4, 5行目では、cxとcyにどちらも100という数を記憶させている。「**=**」記号は、左側の変数に右側の値を記憶するという意味である。これを**代入**という。
6行目はdという変数を作って、50をそこに代入している。

以上より、1つ目のellipseはcx, cy, dがそれぞれ100, 100, 50なので、(100, 100)の位置に直径50の円を描く。
2つ目のellipseでは、縦横の高さがd*2となっている。「**＊**」は掛け算の記号で、×は英語のxとまぎらわしいので＊記号を使う。つまり、直径は50*2の100となる。
3つ目のellipseは1つ目と中身が同じだが、その手前でd=d+100;という命令が実行されている。これは、dにd+100を記憶しなおせという意味で、dは今まで記憶していた50を忘れて、50+100の150に変わる。したがって、ここでは直径150の円が描かれる。

課題

・３つの円の中心を右に50移動させよ。
・３つの円の直径を、30, 60, 90にせよ。

引き続いて「Part1_2_example2」を開こう。
このプログラムのポイントは**if文**である。
しかし実行ボタンを押すと

というメッセージが出てくる。
これは「bがないよ」というエラーである。
たしかにfill(r, g, b)などでbという名前の変数を
使っているのに、それを作っていない。
rやgに習ってbという変数を作り、エラーを解決しよう。

```
size(200, 200);

int r = 0;
int g = 50;

fill(r, g, b);
if(r+g+b < 200){
  ellipse(100, 100, 100, 100);
}else if(r+g+b < 400){
  rect(50, 50, 100, 100);
}else{
  ellipse(100, 100, 200, 200);
}
```

さて、fill(r, g, b)を見るに、塗りつぶしの色を赤がr、緑がg、青がbに指定している。
そして、ellipse, rect, ellipseがあるので、円と長方形と円がこの順に描かれそうに見える。しかしながら、このプログラムではこの３つのうちの１つしか描かれない。
それは、if文による**分岐**が起こるからである。

if(r+g+b < 200){ … }というのは、ifの後ろのカッコの中身が「はい」だったら … を実行しろという意味になる。r+g+b < 200は、r+g+bが200より小さければ「はい」、そうでなければ「いいえ」だ。
それに続くelse if(r+g+b < 400){ … }は、前にあるif文が「いいえ」で、このカッコの中身が「はい」ならば … を実行する。
次のelseは、前のif文が「いいえ」なら中カッコの中身を実行する。

つまり、このプログラムはr, g, bの合計が200未満なら小さい円を、そうでなくて400未満なら長方形を、それ以外なら大きい円を描く。

課題
・r, g, bの値を変えて、200未満、200以上400未満、400以上で描かれる図形が違うか試してみよ。

2-1　変数と計算

変数と計算式を使ってみよう。

・変数の使い方

(1) 変数を作る　　int a;（整数）　　float b;（小数）

(2) 変数に中身を代入する　a = 1;　b = 3.14;

(3) 数として使う

※変数の名前は、intやifのような別の意味がある単語でなければなんでもいい。

a1, ABC_2のような数字や__が混ざった名前も変数名として使える。

※同じ名前の変数を２つ作ることはできない。

※整数を整数で割り算すると切り捨てされる。それが困るときはfloatを使う。

・計算記号

足し算：＋

引き算：-

かけ算：＊

わり算：/　　（分数のイメージ。1/2 は２分の１っぽく見える）

あまり：%　　（7/2 は３で、7%2 は１になる。）

かっこ：（ ）

代入　　：x = 1;

加算代入：x += 3;　（x=x+3;の略。-*/もある）

１加算　：x++;　　（x+=1;の略。--もある）

課題

println(x);という命令を使うと、x を下の画面に表示することができる。

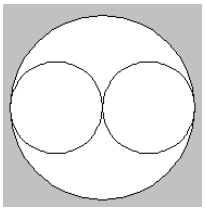
試しに x という名前の整数の変数を作り（int x;）、x に 100 を代入し(x = 100;）、x を表示してみよう（println(x);）。100 が表示されるはずである。

これができたら、

・12345 を 67 で割った商と余りを求めて println で表示せよ。
(変数は int で作る。「int 型の変数」という)

・12345/67 を小数で求めよ(float 型を使う)

・変数 d を用意し、外側の円の横幅が d である次のような絵を表示せよ。
内側の円の直径は d/2 で求める。



2-2　if 文

if 文の書き方と、条件式の書き方について整理しよう。

・if 文の文法

```
if(条件式 A){
        条件式 A が「はい」のときに実行される命令たち
}else if(条件式 B){
        A が「いいえ」で B が「はい」のとき
}else{
        A も B も「いいえ」
}
```

※else if が多段階に続いたり、あるいは else がなかったりしても大丈夫。

・条件式の記号

X == Y　:等しい
X != Y　:等しくない
X < Y　:X が Y より小さい
X <= Y　:X が Y 以下
X > Y　:X が Y より大きい
X >= Y　:X が Y 以上
(X==Y) && (Y==Z)　:X が Y と等しく、**かつ**、Y が Z と等しい
(X==Y) || (Y==Z)　:X が Y と等しい、**または**、Y が Z と等しい

課題

教材のPart1_2_example2と上の説明を参考にして、次のプログラムを書こう。

・２つの変数 a, b を用意して、なにか数を代入する。
　もし a が b 以下なら赤い丸を、そうでないなら青い丸を表示せよ。
・println を使って a と b の大きい方を片方だけ表示せよ。

デバッグ（プログラムの間違い探し）のポイント

・文の終わりにセミコロン「；」を忘れていないか？

・日本語が混じっていないか？（プログラムは普通は全て英語）

・開いたカッコが閉じているか？

() {} []

これは見た目の問題だが、中カッコを一つ開くと、次の行からは、その中カッコが閉じるまで先頭にタブ（Tab キーで入力）を一つ加える。だから次のようになる。

```
void draw(){
        …
        if( … ){
                …
        }
        …
}
```

クイズ

・次の計算式の答えは何になるか？

1+2*3

(1+2)*3

10/3

10.0/3

・次の条件式は「はい」か「いいえ」か？

5!=6

((5>0) && (5<10)) || ((15>0) && (15<10))

答え合わせは自分でプログラムに書いて実行してみよう。

3　アニメーション

動画の作り方と、マウス操作の方法を学ぶ。
教材の「Part1_3_example1」を開こう。

実行すると、円が小さくなって、また大きく広がっていくのが見えるはずだ。
プログラムをひとつずつ説明していこう。

```
int d;
```
外側

```
void setup(){
  size(300, 300);
  d = 300;
  frameRate(10);
}
```
setup命令

```
void draw(){
  background(255, 255, 255);
  ellipse(150, 150, d, d);
  d-=10;
}
```
draw命令

・プログラムの構成
プログラムの中カッコ {} でくくられた領域を、**ブロック**という。このプログラムには setup 命令ブロックと、draw 命令ブロックがある。前の章で出てきた if や else もブロックを作っていた。
ブロックの中身は一つの小プログラムとなっていて、ある条件が満たされた時に中身が実行される。例えば if 文なら条件式が「はい」のとき実行されるし、setup 命令ブロックなら、setup という命令が呼び出された時に実行される。
ブロックの中で作った変数は、そのブロック内でしか使えないことは重要である。
外側で作った変数は、どちらのブロック内でも使える。

・setup と draw
setup、draw 命令ブロックは、これらの命令が呼び出された時に実行される。このプログラム内で両者の命令を呼び出している文は無いように見えるが、setup と draw に関しては、何も書かなくても勝手に呼び出される「お約束」である。
アニメーションは、**静止画を連続して表示する**ことで動いているように見せる。
プログラムが起動すると setup 命令が一回実行され、その後 1 秒間に frameRate 命令で指定した回数ずつ draw 命令が実行される。draw では毎回少しずつ大きさの違う円を描いているので、円の大きさが変化しているように見えるのである。

・background(R, G, B)命令
画面全体を色(R, G, B)で塗りつぶす命令である。

まとめると、このプログラムは次のような流れで動作する。

(1) 外側に書いた変数 d が作られる

(2) setup 命令。画面の大きさと frameRate が設定され、d には 300 が入る

(3) draw 命令。画面全体を白で塗りつぶし、ウィンドウ中心を中心に直径 d の円を描く。そして d を 10 小さくする。

(4) 3 をくり返す。

課題

・frameRate の中身を 1 や 100 にして、表示速度の変化を調べる。とくに 1 のときは、1 秒に一回画面が描き変わっている（draw 命令が実行されている）のがわかるだろう。

・background の行の先頭に「//」をつけてコメントにして、何が変わるかを見る。
これがないと、前の絵が消されないまま重ねて次の絵を描く事になる。

・if 文を使って、d が 10 以上の時だけ d-=10;の文を実行するようにせよ。
これで円が小さくなった後大きくならずに停止するようになる。

・変数をもう一つ作り、円が横方向に毎秒 100 のスピードで動くようにせよ。
変数 d の真似をするとよい。

※いままで命令を呼び出すことは ellipse などでやってきたが、
void setup(){ … }は、いわば「命令を作る命令」で、setup という名前の命令を自作しているといえる。
命令の作り方については後で詳しくやる。

3-1　アニメーションの作り方

アニメーションの作り方を確認すると、

(1) 外側：変数を作る

(2) setup ブロック：画面などの設定と、変数への初期値の代入を行う

(3) draw ブロック：変数を使って絵を描いて、変数の値を少し変える

実行順序は(1)→(2)→(3)→(3)→(3)→(3) … であり、(3)が呼び出される頻度は frameRate 命令で指定する。

draw 命令は自動的には「黒板を消さない」から、前の絵が残ったまま次の絵を描く事になる。これを防ぐためには、画面全体を background 命令で同じ色に塗りつぶす。チョークで黒板を真っ白に塗りつぶしてキレイにするようなものである。強引ですね。rect などの別の図形を描く命令で上書きして消しても問題はないです。

課題

教材の Part1_3_example1 を参考にしよう。

・画面全体の色が白（255, 255, 255）からだんだん黒（0, 0, 0）に変わっていくアニメーションを作れ。

・白から黒になったらまた白に戻っていって、白になったら黒に戻っていって…とくり返すアニメーションにせよ。

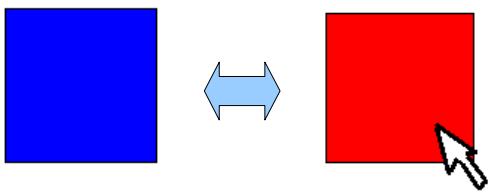
3-2　マウス操作

マウスで動画をコントロールできるようにしよう。そのためには、「マウスが今どこにいるか」を知る必要がある。
それには **mouseX**, **mouseY** という名前の変数を使う。これらの変数は何もしなくても用意されていて、**自分で作る必要はない**。これはマウスが（mouseX, mouseY）にいるということを表していて、その値はマウスが動くたびに更新される。

課題

・次の方針で、マウスにくっついてくる円を描こう。
　(1) setup ブロックで size 命令と frameRate 命令を実行する。
　(2) draw ブロックで ellipse 命令を実行し、円を描く位置を(mouseX, mouseY)にする。

・ボタン
　画面に四角形を表示し、マウスがその四角形の中に入ったら四角形の色を変えて、マウスが外に出たら戻るようにしよう。
　if 文をうまく使う。



・ボタン 2
　上の問題を四角形でなく円でやってみよう。dist(x1, y1, x2, y2)という命令を使うと二点(x1, y1)と(x2, y2)との距離を求めることができる。

4　ループ

さきほどの draw 命令は、なんども繰り返される命令だった。
これを使えば、直線をたくさん引いたり、四角形をたくさん描いたりするのにその本数だけ line 命令などを書かなくても、
中に入る変数の値を少しずつ変えていけば（毎回画面を消さないことにすれば）たくさんの図形を描くことが出来るだろう。

しかし、それをやりたいだけであればわざわざアニメーションを使う必要はない。
for 文という便利な構文があるからである。
直線を等間隔でたくさん引くことを考える。その x 座標（左端からの距離）は

5, 15, 25, 35, … , 185, 195

のような等差数列になる。ここで大事になる値は、**最初の値** 5, **最後の値** 195, **間隔** 10 の 3 つだ。これをプログラムで表現するには、変数 x を作って、最初の値として 5 を代入し、繰り返しのたびに x に 10 を足し、最大値を超えたら終わる、というふうにする。

教材の「Part1_4_example1」を開こう。
ここのポイントは、

for(x=5; x<200; x+=10){ … }

というブロックだ。この中身は何度も繰り返されるのだが、

```
int x;

size(200, 200);
background(255, 255, 255);

for(x=5; x<200; x+=10){
  line(x, 0, x, 200);
}
```

(x=5; x<200; x+=10)の部分で繰り返しの回数を指定している。
１つ目の x=5 は、繰り返しの前に実行する命令を表す。ここでは x の初期値 5 を代入している。
２つ目の x<200 は、x が 200 未満であるかぎり繰り返すという条件である。x が 200 を超えたら繰り返しを終えて、ループからは脱出する。
3 つ目の x+=10 は、一回繰り返すたびに実行する命令である。ここでは x に 10 を足すことで、5 の次は 15, その次は 25 … のように x の値を 10 ずつ増やしていく。

課題

・この for 文を真似して、上端からの距離が 10, 30, 50, … , 190 の位置に横線を引け。
・45 度の斜め線を 20 本引け。

次に教材の「Part1_4_example2」を開こう。
これは左から右に行くにしたがって、少しずつ色を明るくしながら長方形を描くプログラムだ。

```
size(400, 100);
background(255, 255, 255);

noStroke();
for(int i=0; i<20; i++){
  fill(i*10, i*10, i*10);
  rect(i*20, 0, 20, 100);
}
```

さっきのプログラムとの違いのひとつは、
(int i=0; i<20; i++) の部分で変数 i を作っているということだ。この書き方もよく使うので見たら分かるようにしてほしい。
「ブロックの中で作った変数はそのブロックでしか使えない」というルールによって、変数 i はこの for ループの外側では使うことができない。

example1 と同様に考えると、変数 i の初期値は 0 で、20 以上になるまで 1 を足しながら繰り返すので、この for ループは 20 回実行され、そのときの i の値は、

0, 1, 2, … 18, 19

となる。
fill の中では i*10 の形で使っているので、fill に入る色は

(0, 0, 0), (10, 10, 10), … , (190, 190, 190)

となる。rect によって描かれる長方形のの左端からの距離は、

0, 20, 40, … 380

である。

課題

・長方形の横幅を半分に、数を 2 倍にして変化がなめらかになるようにせよ。
・青から赤に色が変化するようにせよ。

4-1　for 文

for 文を使うと面白い絵が描けるようになる。いくつか練習してみよう。

・for 文の文法

```
for(A; B; C){
    D
}
```

実行順序は

A→if(B)→D→C→if(B)→D→C→if(B)→D→C→ …

B が「いいえ」になったところで繰り返しを終える

課題

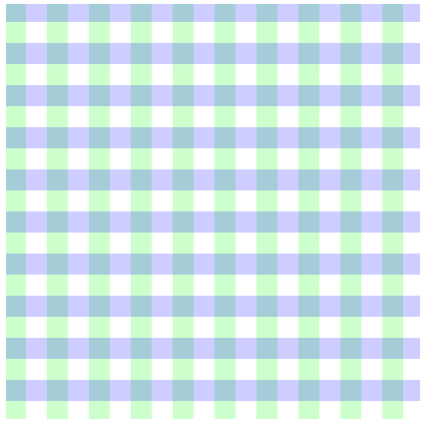
・次の絵を描くプログラムを作れ。

まず等間隔に緑の縦長の長方形を並べて描く。その上に青の横長長方形を並べて描くと左の図のようになる。

右の図は、色として透明色を用いたものである。

透明色とは、今までの色指定で用いた(R, G, B)に**透明度** A を加えて(R, G, B, A)で色を決めるものである。A が小さいほど色が透明になり、右図のように下が透けて見えるようになる。A の値は 0~255 である。

使用例：fill(255, 0, 0, 50) … （薄い赤の半透明）



・乱数と for 文

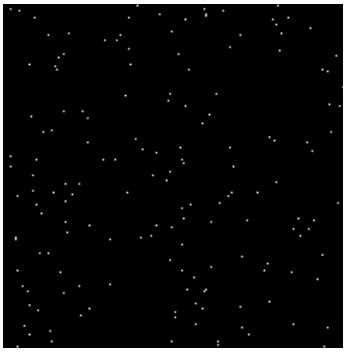
random(a, b)という命令は、a 以上 b 未満のランダムな小数を得る命令である。

X = random(0, 200);

と書くと、変数 X には実行するたびに違う 12.3, 143.2, 33.0, … のような 0 から 200 までの小数が入る。これを乱数という。

例えば、point(random(0, 200), random(0, 200));と命令すると、左端からの距離が 0~200 までのランダムな値、上端からも 0~200 のランダムな値が入り、つまり 200*200 の四角形内部のランダムな位置に点を打てる。

乱数は実行するたびに結果が変わるので、for 文で連続実行すると、毎回違うことをしてくれる。



200*200 の黒い四角形内部にランダムに 200 個の白点を打って、星空のようにせよ。

・アニメーションの draw ブロックの中で for 文を用いて次のようなプログラムを書け。

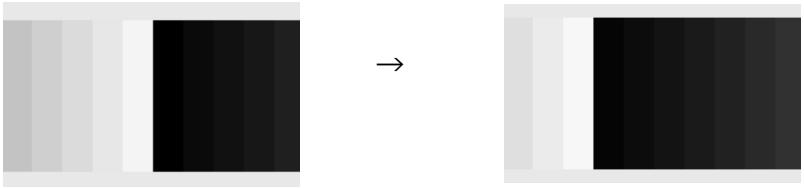
少しずつ色を変えた四角形を並べる。時間が経つごとに各四角形の色を少しずつ変化させる。こうすると下の絵のように黒い部分が左に動いているように見える。

ヒント 1 ：変数 j を作って、draw 命令が呼ばれるたびに 1 を加算していく

ヒント 2 ：色の指定の時に 256 以上の値を指定すると強制的に 255 にされてしまう。

%(割ったあまり)か if 文をうまく使って、

255 を超えたら色の指定に 0 が入るようにしよう。



4-2　二重ループ

for 文は何重にもすることができる。

```
for(int x=0; x<200; x+=20){
    for(int y=0; y<200; y+=20){
        rect(x, y, 18, 18);
    }
}
```

このように書くと、一番内側の rect では(x, y)が

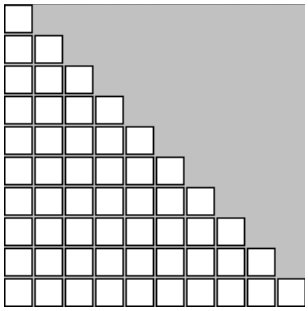
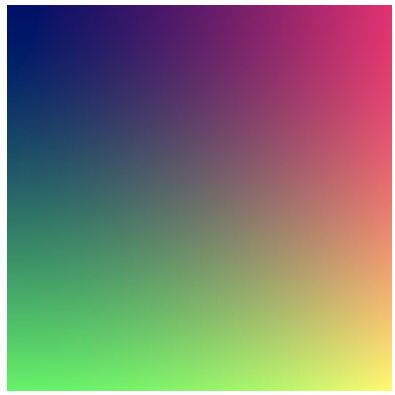
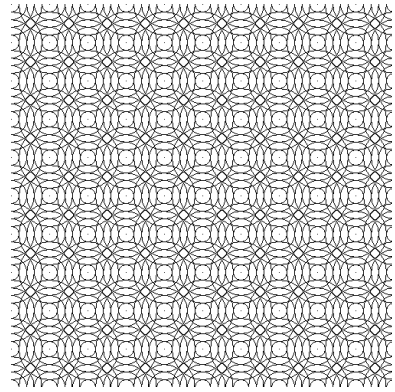
(0, 0)→(0, 20)→ … →(0, 180)
→(20, 0)→(20, 20)→ … →(20, 180)
→ … →
→(180, 0)→ … →(180, 180)

と変化する。これによって、マス目状に四角形を描くことができる。

課題

・上の例を実装せよ。

・次のような絵を描いてみよ。

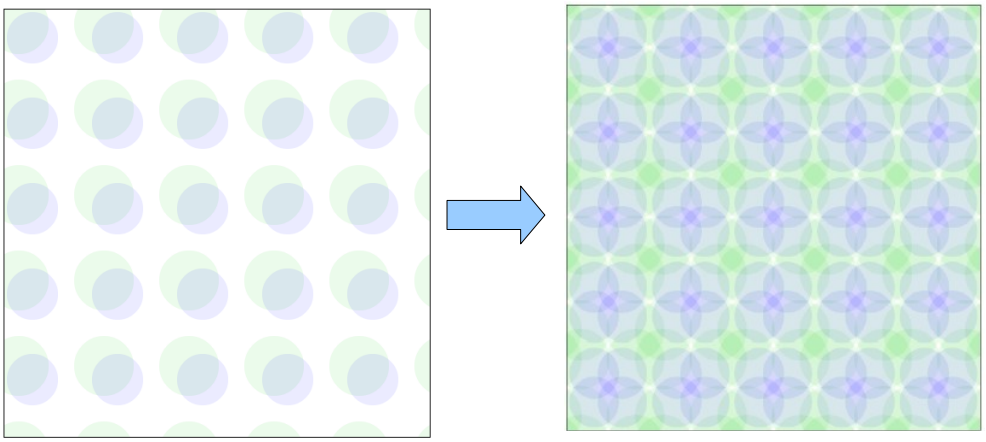
左：y の初期条件を x で決める。

中：point を使って点を打つ。Point の色を指定するには fill でなく stroke を使う。

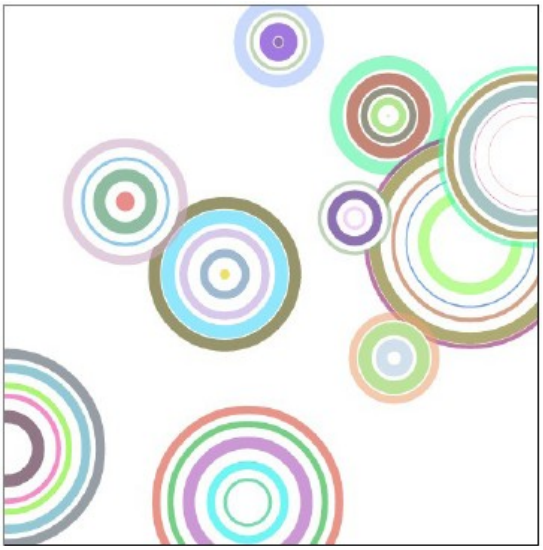
右：３重のループを使う。

・二重ループを使った難しい課題

半透明の二色の円を縦横等間隔に二重ループで描く（左図）。
円の種類を増やすと右図のようなきれいな模様が描ける。



for 文で 10 回のループをする。1 回のループにつき、乱数で決まった(x, y)を中心とする同心円たちを描画する。
同心円は for ループで外側の円から描いていき、色と半径の変化は乱数で決める。



5　配列

for 文を使うことでたくさんのものを描くことができた。

さて、今まで変数は自分で 1 つずつ作ってきたが、これをまとめてたくさん作ることができる。これを**配列**といって、たくさんの数を記憶することができるようになる。

教材の「Part1_5_example1」を開こう。

今までの変数は次の手順で使っていたことを思い出そう。

作る：float x;

代入：x = 100;

使う：ellipse(x, …

配列は次の 4 つの手順で使う。

作る：float[] x;

準備：x = new float[100];

代入：x[2] = 100;

使う：ellipse(x[2], …

変数と違うのは、作ると代入の間に準備がいることである。

x = new float[100];と書くことで、x という名前の配列が 100 個の要素を持つことになる。

それらは x[0], x[1], x[2], … , x[99]という名前になる。

1 から 100 までではなく、**0 から** 99 までであることに注意。

```
float[] x, y;
float[] vx, vy;

void setup(){
  int i;
  size(200, 200);
  frameRate(100);
  x = new float[100];
  y = new float[100];
  vx = new float[100];
  vy = new float[100];
  for(i=0; i<100; i++){
    x[i] = random(0, 200);
    y[i] = random(0, 200);
    vx[i] = random(-2, 2);
    vy[i] = random(-2, 2);
  }
}

void draw(){
  int i;
  background(255,255,255);
  for(i=0; i<100; i++){
    ellipse(x[i], y[i], 10, 10);
  }
  for(i=0; i<100; i++){
    x[i] += vx[i];
    y[i] += vy[i];
  }
}
```

x[2]の[2]の部分を配列の**添字**（そえじ）という。

配列の便利なところは、添字として別の変数を使うことができる点である。これによって、for 文と組み合わせてたくさんのものをまとめて操ることができるようになる。

今までは 5 つの変数に全部 0 を代入したければ

a=0; b=0; c=0; d=0; e=0;

のようにそれぞれ代入文を書かないといけなかったが、配列を使えば

for(int i=0; i<5; i++) x[i] = 0;

のように for 文と 1 つの代入文で書くことができる。

課題

このプログラムをいじって、次のことをやってみよう。

・描画するものを円ではなくて長方形にしよう。

・今はすべての長方形がランダムな位置からスタートするが、
　これを全部(100, 100)からスタートするようにしよう。

・長方形の数を今は 100 だが、300 にしよう。（修正は 7 カ所）

・このプログラムのなかでわからない行はないか確認しよう。
　この時点で不明点があると先に進めないので、遠慮なく手を上げて質問して。

・このプログラムを、弾が右端ではね返るようにする。
　if 文を使って、x[i]が 200 を超えたものについて、vx[i]を-vx[i]にする。

・全部の端ではね返るようにせよ。

・図形にそれぞれランダムな色をつけよう。

(1) float 型の配列 r, g, b を作る。ここに図形の色を記録する

```
float[] r, g, b;
```

(2) 準備する

```
r = new float[300];　など
```

(3) それぞれ 0~256 のランダムな数を入れる

```
r[i] = random(0, 256);　など
```

(4) i 番目の図形の色を r[i], g[i], b[i]で指定する

```
fill(r[i], g[i], b[i])
```

5-1　1 次元配列

たくさんの変数をまとめて作って、まとめて扱う方法が配列である。配列の要素には「添字」を使ったアクセスができて、添字の数が一つのものを 1 次元配列という。

(1) 配列を宣言する

配列を使うには、まず最初に配列の名前と型を決める。
この時点ではそれらが決まるだけで、中身は作られない。
setup と draw 命令両方から使われることが多く、**外側で宣言**することが多い。

（例）　float[] x;　　int[] r, g, b;

(2) 配列を new する

配列の int や float といった変数との違いは、名前を決めた後にそれを new しないといけないという点である。これは配列が「オブジェクト」の一つである（オブジェクトについては Part2 で学ぶ）からである。

new すると何が起こるのかというと、パソコンのメモリ上にこの配列の記憶領域が確保されて、配列の要素にアクセスが可能になる。

new する際に配列の要素の数を決めて、[]の中に書く。これは変数でもよい。

new は普通は最初に 1 回しか行わないので、**setup 命令の中**に書くことが多い。

（例）　x = new float[100];　　r = new int[N];

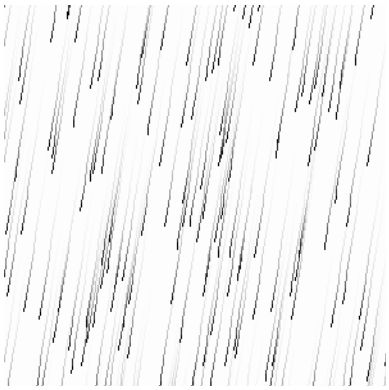
(3) 配列の要素にアクセスする

配列の要素には添字を使ってアクセスする。**添字は 0 から**はじまる。
上の例の配列 x なら、x[0], x[1], … x[99]が int 型の変数として使える。
配列 r なら、r[0], r[1], … r[N-1]が使える。
x[100]や r[N]を使おうとするとエラーになるので注意。
for 文と組み合わせて配列全体を操作したり、複数の配列の同じ添字番号の要素を組みにして使ったりするテクニックがある。

（例）　for(int i=0; i<100; i++) { x[i] = i; }
　　fill(r[3], g[3], b[3]);

課題

雨が降るアニメーションを作れ。



以下の手順で行うと良い。

(1) 要素数がたくさん（1000 くらい）ある
　配列 x, y を宣言し、new する。
　これらには雨つぶの位置を記憶する。

(2) int 型の変数 n を作り、ここには雨つぶの数を記憶することにする。最初は 0 を入れておく。

(3) Draw 命令の中では次の処理を行う

1. 新しい雨つぶを作る。添字が n の要素が新しい雨つぶである。
　x[n]には雨つぶの初期位置の x 座標を入れたいので、乱数で決定する。
　y[n]には y 座標を入れたくて、初期位置は一番上にしたいので 0 とする。
　雨つぶがひとつ増えるので代入が終わったら n には 1 を足す。
2. for 文を使って、配列の全要素について、x,y 座標を加算する。
　ここでは雨つぶを下に移動して降らせたいので x[i], y[i]にそれぞれ一定の数を足すと良い。
3. for 文を使って、すべての雨つぶを描く。

・上の図のように軌跡（残像）を残すには draw 命令の最初に background 命令を実行する代わりに、半透明の白い四角形で画面全体を覆う。こうすると前に描かれていたものが少し薄くなり、残像として残ってくれる。
・draw 命令で雨つぶを同時に 5 つくらい作ると上の図のような雨つぶの量になる。
・n が増えていって、配列の添字の上限を超えるとプログラムは止まってしまう。
　これには次のような解決策がある。
　・new するときにもっとたくさんの要素数を確保する（10000 とか）
　・外に出た雨つぶを消す。
　　y[i]が画面の縦幅を超えたら、x[i], y[i]に x[n-1], y[n-1]を代入して n--;する。

6 命令

いままで様々な命令を使ってきたが、ここで命令について整理しよう。
命令には次のようなものがあった

引数なし、戻り値なし：noFill()
引数４つ、戻り値なし：ellipse(x, y, w, h)
引数２つ、戻り値あり：x = random(0, 100)

引数というのは命令を呼び出すときにカッコの中にカンマ「,」で区切って入れる数のことで、戻り値というのは、命令それ自体が変数のように使えるかということ。
たとえば random(0, 100)は、それ自体が 0~100 までの数として使うことができる。
このように、命令というのは**引数**として与えたデータをもとに何かをして、**戻り値**を返すものと考えることができる。引数、戻り値はないこともある。

さて、命令は自分で作ることもできる。よく使う処理の流れを一つの自作命令としてまとめておくことによって、プログラムをわかりやすくすることができる。

(1)引数も戻り値もない命令の作り方
「Part1_6_example1」を開いてみよう。
これはマウスをクリックするたびに丸がランダムな位置に移動し、ランダムな速度で動いていくというものだ。
このプログラムには shokika と idou という自作命令がある。どちらも引数、戻り値はない。
shokika 命令は 3~8 行目で作っていて、変数 x, y, vx, vy にランダムな値を入れる。
setup 命令の中と、mousePressed 命令の中で呼ばれているので、最初とマウスがクリックした時に実行される。
idou 命令は 10~13 行目で作っていて、位置(x, y)を(x+vx, y+vy)に移動するものである。これは draw 命令の中で使われているので、一秒間に 100 回実行されることがわかる。

shokika 命令を作っているところを見ると、void shokika(){ … }という形をしている。

void というのは戻り値が無いことを表していて、()は引数が無いことを表す。続く中カッコの中身が命令の内容になる。引数があるとこのカッコの中に引数の情報が入る。これは次の例で学ぶ。

課題

・10~13 行目について、何をしているのか説明せよ。

・byouga という命令（引数、戻り値なし）を作り、そのなかで

ellipse(x, y, 10, 10)

を実行するようにせよ。draw 命令の中では、ellipse 命令の代わりに byouga 命令を呼ぶようにせよ。命令の作り方は、idou や shokika を真似すると良い。

(2)引数がある命令の作り方

次に「Part1_6_example2」を実行しよう。これはクリックすると自作命令 sankaku（引数 2 つ、戻り値なし）が実行され、クリックした位置に三角形を描く。

sankaku 命令は 1~5 行目で作っている。一行目の(float x, float y)は引数として 2 つの float 型の数をとり、それらを x, y という変数に代入するという意味である。(1)の例でこの部分が()になっていたのは、引数がなかったからである。これらの変数 x, y は sankaku 命令の中だけで使える。

課題

・draw 命令の中で for 文を使って、ランダムな位置に 100 個の三角形を描け。

・sankaku 命令の引数を一つ増やし、その引数によって描かれる三角形の大きさが変わるようにせよ。

(3)戻り値がある命令の作り方

「Part1_6_example3」を見よう。ここには beki という名前の２引数、戻り値ありの命令が作られていて、これは引数 a（float）と n（整数）をとり、a の n 乗（a を n 回かけたもの）が戻り値として返されるものである。

実行すると下の画面に 3 の 4 乗である 81 が表示される。

１行目では(1)、(2)では void となっていた部分が float となっている。これはこの命令の戻り値が float 型であり、beki 命令の結果が float 型の変数のようにして使えることを表している。

７行目の return ret;は、戻り値として変数 ret を返すということを表している。

return 10;と書けば、もちろん毎回 10 が返ってくるようになる。

課題

・5 の 6 乗を計算せよ。

・float 型の引数２つをとって、そのうちの大きい方を戻り値として返す命令を作れ。

命令の作り方まとめ

```
<戻り値の型> 命令の名前(引数たち){
        命令の内容
        return 戻り値;
}
```

戻り値がないときは、戻り値の型はvoid。

Part2　ブロック崩し

(1) いろいろな動き
(2) 配列の操作
(3) 当たり判定
(4) 状態管理
(5) キー操作
(6) 二次元配列

1　いろいろな動き

・用語

画面内の物体の位置を変数 x, y で表す。

物体の速度は変数 vx, vy で表す。速度とは、一回 draw 命令が実行された時に x, y に足される数である。

(1) 直線運動

教材の Part2_1_example1 を開こう。実行すると一つの物体が画面の中を飛び回る。

ここには settei, idou, byouga の 3 つの自作命令がある。

setup 命令と draw 命令の中を見ると、settei は setup で使われているので、今は最初に一回だけ実行される。idou, byouga は draw の中で使われているので、繰り返し呼ばれることが分かる。

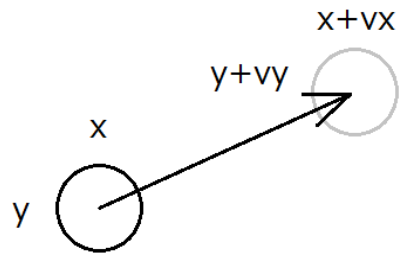
settei 命令は 4 つの小数の引数をとり、その 4 つの数を x, y, vx, vy の変数に順に代入する。つまり、これは位置と速度を設定する命令である。

idou 命令は、x, y を vx, vy 進めて物体を移動する命令である。画面の端で反射する処理も書いてある。

byouga 命令は、物体を描画する命令である。今は x, y の位置に○を描いている。

この直線運動こそが、あらゆる物体の動きの基本である。

1 ステップ（一回の draw 命令）での物体の動きを図にすると、右のようになる。



課題

・物体の移動方向をわかりやすくするために、byouga 命令を書き換えて円ではなく直線で物体を描画しよう。直線は始点が x, y で、進行方向に平行な直線にする。

・直線を描く手前で strokeWeight(W);という命令を実行すると、直線の太さが W になる。W を 1 にすると一番細くなる。直線を太くして見やすくしよう。

・settei 命令に渡す引数を変えて最初の位置と速度を変えてみよう。

(2) カ

(1)のプログラムをベースに改良していく。

直線運動では idou 命令で x, y を x+vx, y+vy に移動していた。Vx, vy は反射の処理で -1 倍されて、それ以外では一定だった。Vx, vy が一定ならば、物体は直線運動する。

Vx, vy を idou 命令の中で更新すると、物体に曲線運動させることが出来る。Vx, vy を vx+ax, vy+ay に更新したとすると、**(ax, ay)方向に力を受けたような運動をする**。

（(ax, ay)方向というのは、(0, 0)から(ax, ay)に伸ばした直線方向ということ）

さて、物体に重力を加えてみよう。

重力は下方向なので、y が大きくなる方向に働く。

そこで、idou 命令の中で

vy を vy+0.1 に更新しよう。

うまくいくと右のようなボールを投げた時のような軌道を描く。

次に、画面の中央に引き寄せられるようにしよう。

物体が(x, y)にいる時に画面中央(150, 150)は、物体から見る(物体の位置を(0, 0)とする)と(150-x, 150-y)の位置に見える。だから速度の変化量(ax, ay)は(150-x, 150-y)方向と平行にしたい。

しかし、vx, vy を vx+150-x, vy+150-y に変えるのは速度の変化量が大きすぎておかしなことがおこってしまう。そこで、変数 k を用いて vx+k*(150-x), vy+k*(150-y) と速度を更新する。(150-x, 150-y)方向と(k*(150-x), vy+k*(150-y))方向は平行だからである。k は小さな小数にする。

k の値を調節して、うまくいくと画面中央の周りをぐるぐると回るようになる。画面端での反射があるとうまくいかないことがあるので、反射は消してしまって良い。

課題

・マウスに引き寄せられる（マウスの周りを回る）ようにしよう。

(3) 抵抗

(2)では画面中央やマウス方向に力は受けていたものの、その場所までは行ってくれなかった。それは「抵抗」を加えることで実現できる。

抵抗は idou 命令内で速度に 0.98 や 0.95 のような 1 より少し小さい数を掛け算する

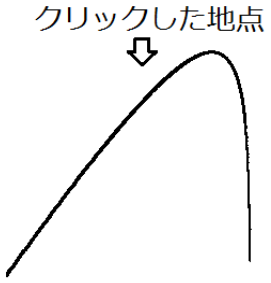
```
vx *= 0.98; vy *= 0.98;
```

ことで実現できる。こうすると 1 ステップごとに速度がすこしずつ遅くなっていく。

課題

・マウスに引き寄せられるプログラムに抵抗を加えよう。

・重力のプログラムに抵抗を加えよう。こうすると空気抵抗の大きなボールを投げた時の軌道になる。それができたら、マウスがクリックされた時に物体が(0, 300)からマウスの方向の速度で飛んで行くようにしよう。settei 命令を使って移動と速度の変更を行う。



(4) 反発

マウスと物体との距離を f としよう。

```
f = dist(x, y, mouseX, mouseY);
```

f がある値（たとえば 20）よりも近づいたら、物体がマウスから反発するようにしたい。

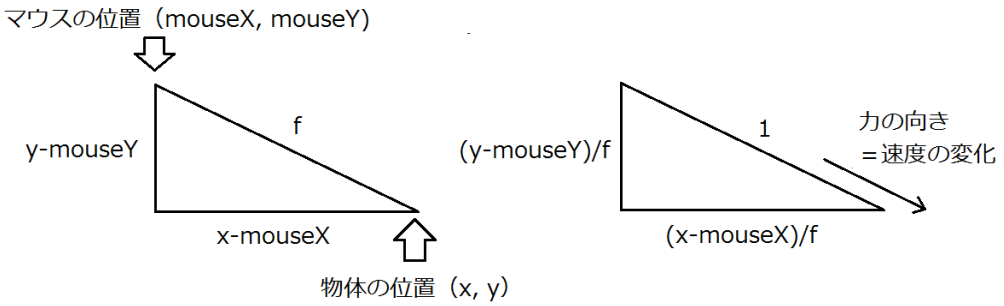
```
if(f<20){
    反発処理
}
```

さて、物体にかかる力により、物体の速度は力の方向に変化を受ける。

反発力の力の向き（速度の変化方向）は、マウスから物体に向かう直線の向きである。

そして、f が 20 よりも小さくなればなるほど反発力が強くなるようにするとキレイに動くようになるので、速度は次のように変化させる。

```
vx += k*(20-f)*(x-mouseX)/f;
vy += k*(20-f)*(y-mouseY)/f;
```



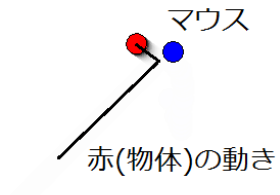
k は反発力の強さを表す小さな小数で、(20-f)は f が 20 より小さくなればなるほど大きくなる。(x-mouseX)/f と(y-mouseY)/f は、斜辺の長さが 1 となる直角三角形の 2 辺の長さになっていて、力の向きを表す。課題

・k を調節して、マウスを近づけたら物体がキレイに反発するようにせよ。

・物体とマウスを直径 20 の円にすると、ちょうど円同士がぶつかって反射したようにみえる。

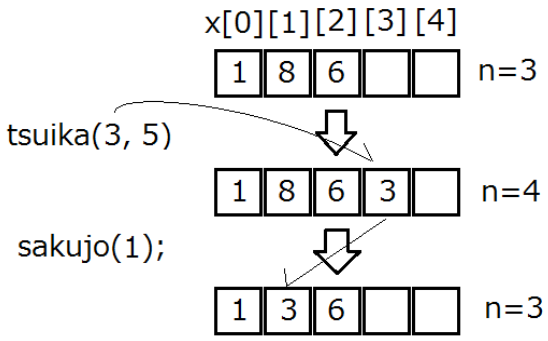
・これを使ってなにかゲームを作ろう。

（物体が画面の外に出たらゲームオーバーとか）



2　配列の操作

配列はたくさんのものを管理するのに使うのだった。例えば、画面内にいる敵の一覧とか、持ってるアイテムのリストといった使い道がある。
Part2_2_example1 を開こう。この章ではこのプログラムをベースにプログラミングしていく。
このプログラムにはまず配列 x,[] y[]がある。これらは物体たちの位置を記録する配列である。
自作命令は tsuika と sakujo があり、それぞれ次の図のような働きをする。



整数 n は、配列に入っている物体の数を表している。
配列 x, y には、tsuika を実行するたびに、引数で指定した値が順番に詰まっていく。その都度 n には 1 が足される。
sakujo が実行されると、引数で指定した添字の配列の要素を削除する。具体的には配列の末尾の要素を削除指定された要素に上書きして移して、n から 1 を引く。

このプログラムでは、draw 命令の中で tsuika を読んでいるので、draw が実行されるたびに物体が 1 つずつ増えていく。
setup の中の new 文で確保している配列の要素数は 1000 なので、物体の数が 1000 を超えたところでエラーとなり止まってしまう。

課題

・物体数が 999 を超えたら sakujo 命令を実行して n が 1000 を超えないようにせよ。

・このプログラムと、さっきの Part2_1_example1 を組み合わせよう。

(1) 配列 vx, vy を用意する

(2) setup で vx, vy を確保する

(3) tsuika, sakujo を vx, vy に対応させる

(4) settei, idou, byouga 命令を作る。その際引数として整数 i を追加し、たとえば idou 命令なら、添字が i の要素についての移動処理を行わせる。

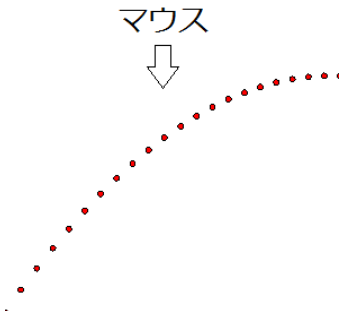
(5) draw 命令の中で for ループを回して、idou、byouga を呼び出す

他にも次のような改造をしてみよう。

・反射をなくして、画面外に出た物体を sakujo 命令で消す

・マウスに向かって弾を撃ち続ける

重力も入れてみた↓



3　当たり判定

図形と図形が接触しているかどうかを判定するのが当たり判定である。
敵とぶつかったら死ぬ、敵をクリックして攻撃、などゲームには不可欠。

(1) マウスとの当たり判定

マウスの位置は点(mouseX, mouseY)だから、点と図形との接触判定ができればよい。

・円と点

点(mx, my)と、中心が(cx, cy)で直径が d の円とが接触しているかどうかを考える。
もし点が円内にあるなら、点と円の中心との距離は d/2 以下である。
したがって、if(dist(mx, my, cx, cy)<d/2) return 1;とすれば、円内に点がある時に戻り値 1 が返るようになる。

Part2_3_example1 を見てみよう。これはマウスが触れている円だけ色を変えるプログラムである。円は draw が実行されるたびに 1 つずつ追加されて、ランダムな位置と速度を持つ。画面外に出た円は削除される。
当たり判定の命令は inCircle 命令である。これは上述の 5 つの引数をとり、点が円内にあれば 1 を、円外ならば 0 を返す。
この命令を使っているのは byouga 命令の中で、fill で指定する色を inCircle の戻り値によって切り替えている。

課題

・inCircle 命令をまねして長方形との当たり判定をする命令を作成せよ。
引数は float 型で点（mx, my）と長方形の左上の点(rx, ry)と横幅 w、縦幅 h とする。
戻り値は、点が長方形の内部にあれば 1、そうでなければ 0 とする。
できたら、この example1 の円を全て長方形に変えて試してみよ。
・図形の出現速度が速すぎるので、数回に一回だけ図形を出現させるようにせよ。
・マウスでクリックされた図形が消えるようにせよ。

(2) 物体同士の当たり判定

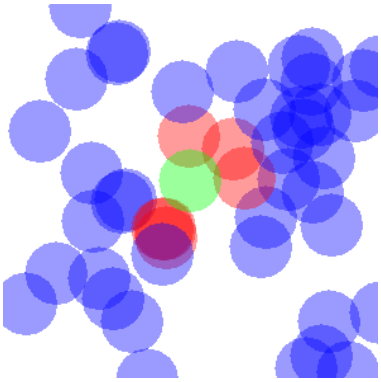
・円と円

円と円との当たり判定は簡単である。円と円とが接触しているなら、両者の中心の間の距離は、両者の半径を足したものよりも短いはずである。

円１の中心と直径が x1, y1, d1 で、円２が x2, y2, d2 ならば、

dist(x1, y1, x2, y2) < (d1+d2)/2

が「はい」のときに両者は接触している。



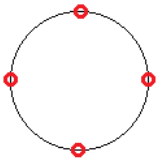
課題

右の図のように、マウスを中心とする円（緑の円）に接触している円だけ色が変わるようなプログラムを作れ。

・小さい円と長方形

大きな円と長方形が接触しているか判定するには、難しい数学の公式が必要である。しかし、シューティングゲームの弾丸のような小さい円と長方形との当たり判定は簡単に行うことができる。

右図のように円を４つの点の集まりとみる。

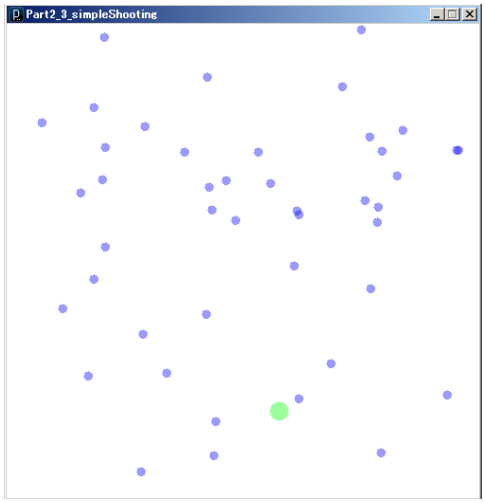


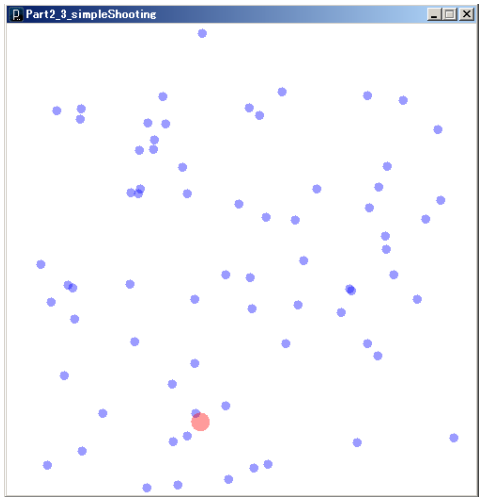
それで、前のページの課題の長方形と点との当たり判定を４回行えば良い。

どの点がぶつかったかによって戻り値を変えれば、上下左右どの辺に衝突したのかも判断できる。これはブロック崩しを作るときに必要になる。

ここまでで作れるゲームの例

・簡単なシューティング（避けるだけ）



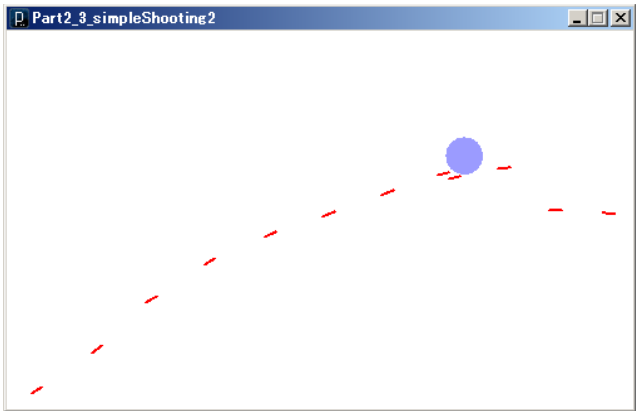


(1) 前のページの円と円の当たり判定の課題を改造する。

(2) ランダムに出現し、ランダムな方向に進んでいく円から逃げる。

(3) 接触したらゲームオーバー。生き延びた時間をスコアとして表示

・簡単なシューティング（撃つだけ）



(1) 35 ページの課題のプログラムと、点と円との当たり判定を組み合わせる。

(2) 弾はマウスの方向に撃ち続ける。敵は青い円。

(3) 弾と敵との当たり判定ができたら、敵の hp 記録する変数を用意。

(4) 弾が敵に当たったらダメージ。敵の hp が 0 になったら勝利

・ブロック崩し

(1) ブロックたちを配列 x, y, (vx, vy)で位置を記録。ブロックが動かないなら vx, vy はなくてもよい

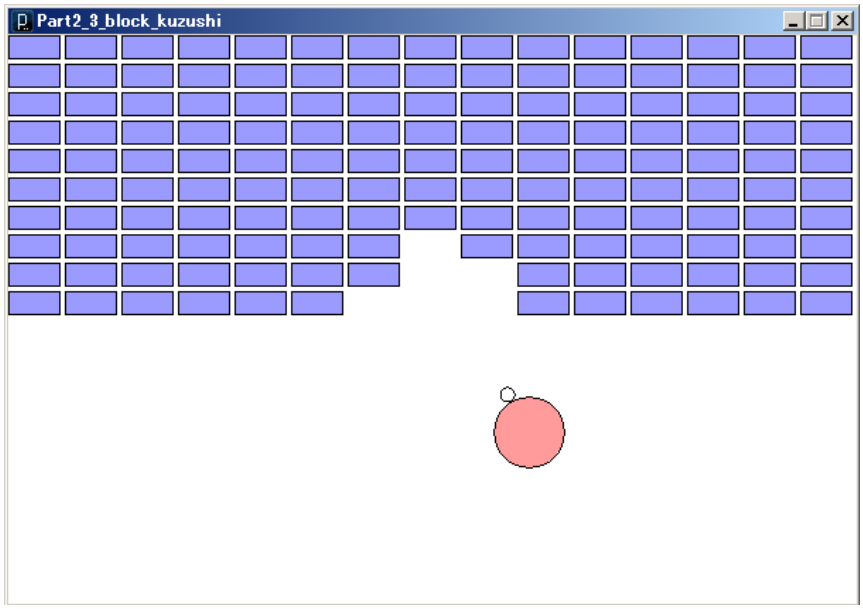
(2) 弾の位置を変数 cx, cy、速度を cvx, cvy などで記録。

(3) 弾とマウスは Part2_1 の反発の手法を使う

(4) 弾とブロックは、小さい円と長方形との当たり判定を使う

(5) 壁にあたった弾は反射させる。下の壁だけ通過させる

(6) ブロックに当たった弾は、どの面からぶつかったかによって反射方向を変える



ブロック崩しの改造（時間のある人向け）

・弾が下から落ちたら Game Over と表示しよう。文字の表示は

text(“moji”, x, y);

で、これは位置(x, y)を左下にして moji という文字を表示する。

・ブロックを全部消したら Game Clear と表示しよう。

・３回までなら落ちても大丈夫なようにする

・スコアを作る

などなどいろんな改造があります。

・大幅に改造して大量の弾が撃てて、敵も動いたり弾を撃ったりするようにしたらシューティングゲームになる。

4　状態管理

タイトル画面、ゲーム画面、ゲームオーバー画面などが切り替わるようにする。

まず現在の状態を記録する変数 state を作る。
state の値は
タイトル：0
ゲーム：1
ゲームオーバー：2
などとして、draw 命令の中で if 文を使って state の値によって処理を切り替える。
タイトルのときにクリックされると state を 1 にして、
ゲームオーバーになったら state を 2 に、
ゲームオーバー画面でクリックされたら state を 0 に戻す。

一度ゲームオーバーになってからもう一度ゲームを始めるとき、前のゲームの敵の位置などの情報が残っているとうまくいかない。ゲームを始める前にこれらの変数の値を全てリセットする必要がある。

5　キー操作

今まではマウスによる操作のみを行なってきたが、ここでキーボードで操作する方法を紹介する。

(1) 一つのキー

２キー以上のの同時押しをしないのならばキー操作は簡単である。
Key と keyCode という名前の変数が最初から用意されており、
ここに押されたキーの情報が入り、キー関連の命令の中で使うことが出来る。
キーが押されたタイミングでは keyPressed という命令が呼ばれる。
キーが離されたタイミングでは keyReleased という命令が呼ばれる。

キーを押しっぱなしにすると、keyPressed はなんども繰り返し呼ばれる。

英語のキーが押されると、key の値が例えば a キーなら'a'になる。
上下左右キーが押されると、key の値が CODED になり、
keyCode の値が UP, DOWN, LEFT, RIGHT になる。

Part3_5_example1 には英語キーの使い方が（z, x, c キー）
Part3_5_example2 には左右キーの使い方が書かれている。

課題

・example2 で、上下キーで○が上下に移動するようにせよ。
・キーを押しっぱなしにしたり、２つ同時に押すと何が起こるか試してみよ。

(2) 同時押し

同時押しは、キャラクターの斜め方向への移動や、移動しながら攻撃
といった場面で使う必要がある。

キーを押しっぱなしにすると、keyPressed 命令が繰り返し呼ばれるが、
このとき変数 key には最後に押されたキーの情報しか入っていない。
例えば a キーと b キーを同時に押して離すと、次のようなことが起こる。
ここで、a キーのほうが押すタイミングと離すタイミングが
わずかに速かったとしよう。

(1) key に'a'が入り、keyPressed 命令が呼び出される
(2) key に'b'が入り、keyPressed 命令が呼び出される
(3) 押してる間中 keyPressed 命令が繰り返し呼び出される。key は'b'のまま。
(4) key に'a'が入り、keyReleased 命令が呼び出される
(5) key に'b'が入り、keyReleased 命令が呼び出される

ここでは同時押しした時でも、keyPressed 命令と keyReleased 命令は、押した瞬間と離した瞬間については両方のキーについて呼び出されるという性質を使う。

Part3_5_example3 を見てみよう。これは example2 を同時押しに対応したものであり、斜めに移動できることがわかるはずだ。
このプログラムでは、kup, kdown, kright, kleft という変数を作ってあり、
もし上、下、右、左キーが押される（keyPressed 命令が呼ばれる）とこれらの変数は 1 になり、離されると 0 になるようになっている。そして、draw 命令の中で、これらの変数の値を元に○の運動を決定している。

課題

・example3 で、z キーを押すとなにか起こるようにせよ。同時押しにも対応せよ。
・前に作ったゲームにキー操作を加えよ。

6　二次元配列

二次元配列は、その名の通り二次元の配列である。今まで使ってきた 1 次元の配列を、表のような形にしたものだといえる。

宣言：float[][] a;

new　：a = new float[100][100];　　...100*100 の配列を作る

使う：a[3][4] = 0;

このように、 1 次元の配列と比較して添字が一つ増えただけである。

将棋や囲碁のようなマス目があるゲームや、RPG のマップのような平面的に色々なものがあるときに役に立つ。

添字が 2 つあるので、この配列の要素全体にアクセスするときは、二重ループを使う。

Part2_6_example1 を見てみよう。これはクリックしたところの四角形の色が変わるというものである。クリックすると、その場所に対応する配列の要素が、 0 ならば 1 に、 1 ならば 0 になる。

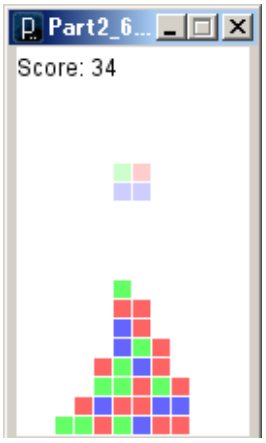
課題

・クリックすると、そのマスとその上下左右のマスの色が反転する。すべてのマスの色が揃ったらクリアというゲームを作ろう。

　右端のマスには右のマスがないというのが厄介だが、配列を上下左右に一つ余分に確保しておいて、一番端を表示しなければ、この問題を解決できる。

　見本は Part2_6_example2 にある。これはまだ全部の色が揃ったらクリアという判定をしていないので、それを書き加えよう。

・右のような落ち物ゲームも作れる



Part 3　シューティングゲーム

目次

1　クラスとは

Processing におけるクラス（class）とは、学校のクラスと同じ「組」という意味だ。何の組なのかというと、「同じ種類のもの」に関する変数と命令を組にしたものである。

Part2 では配列を使って敵の位置や速度を表し、命令を作って移動や追加、削除などをした。しかし敵、自機、弾など登場人物の種類が増えてくるに従ってだんだんプログラムがごちゃごちゃになっていったのではないだろうか。敵のｘ座標 ex, 弾のｘ座標 bx、自機のｘ座標 px、...のようにそれぞれに別の変数名を割り当てないといけなくて、プログラムが読みにくくなってしまう。

クラスというのは、**登場人物の種類ごとに変数と命令をまとめて**、プログラムをすっきりさせようというものである。

例えば、シューティングゲームの敵ならば、

位置(x, y), 速度(vx, vy), HP, 攻撃パターン, …

といった変数たちと、

設定, 移動, 描画, ダメージを受ける, 攻撃する, …

といった命令たちを一つのクラスにまとめる。

クラスは「オブジェクト」の設計図である。クラスの文では「このクラスはこういう変数たちとこういう命令たちを持つ」と書いておいて、そのクラスを new することで、実際にその変数と命令を備えたオブジェクトを生成する。オブジェクトが持っている変数や命令には obj.x のようにドット「.」を使ってアクセスする。オブジェクトがオブジェクトを持つこともできて、そのときは a.b.c のようにドットが連鎖することもある。

同じクラスから　敵１、敵２、敵３、・・・と多数のオブジェクトを生成することができる。敵１.x には敵１のｘ座標が、敵２.y には敵２のｙ座標が・・・という感じでオブジェクトごとに変数が作られる。命令についても、敵１.idou()とすると、敵１だけが移動するし、敵２.byouga()とすれば敵２が画面上に描かれるというように、オブジェクトごとに操作を行う。

・クラスの使い方

(1)定義

クラスは変数と命令の組であり、次のようにその構成を記述する。

これはいわば設計図である。

```
        class クラス名{
                変数たち
                命令たち
        }
例：     class Ball{
                float x, y;
                float vx, vy;
                void idou(){
                        x+=vx;
                        y+=vy;
                }
        }
```

(2)作る

「int[]」や「float[]」で配列を作る時と同じようにする。

```
        クラス名 オブジェクト名;
        例：Ball b;
```

(3) new する

配列と同じで new しないと使えない。new すると、(1)の定義に基づいてオブジェクトが生成され、メンバーの変数たちがメモリ上に出現する。

実は「クラスが配列と似ている」というより、「配列がクラスの一種」である。

```
        オブジェクト名 = new クラス名();
        例：b = new Ball();
```

(4)使う

オブジェクトの変数や命令には、ドットを使ってアクセスする。

オブジェクト名.変数名

例:b.x　　b.idou()

・クラスの例

「Part3_class_1」と「Part3_class_2」を比較しましょう。

1 がクラスを使わない今までの方法、2 がクラスを使った方法である。

```
float[] x, y;
float[] vx, vy;
int n;

void tsuika(float x0, float y0, float vx0, float vy0){
  x[n] = x0;
  y[n] = y0;
  vx[n] = vx0;
  vy[n] = vy0;
  n++;
}

void sakujo(int i){
  x[i] = x[n-1];
  y[i] = y[n-1];
  vx[i] = vx[n-1];
  vy[i] = vy[n-1];
  n--;
}

void settei(int i, float x0, float y0, float vx0, float vy0){
  x[i]=x0;
  y[i]=y0;
  vx[i]=vx0;
  vy[i]=vy0;
}

void idou(int i){
  x[i] += vx[i];
  y[i] += vy[i];
}

void byouga(int i){
  ellipse(x[i], y[i], 10, 10);
}

void setup(){
  size(300, 300);
  frameRate(30);
  x = new float[1000];
  y = new float[1000];
  vx = new float[1000];
  vy = new float[1000];
}

void draw(){
  background(255, 255, 255);
  for(int i=0; i<n; i++){
    idou(i);
    byouga(i);
  }
  for(int i=n-1; i>=0; i--){
    if(x[i]<0 || x[i]>300 || y[i]<0 || y[i]>300){
      sakujo(i);
    }
  }
}

void mousePressed(){
  tsuika(mouseX, mouseY, random(-1, 1), random(-1, 1));
}
```

```
class Ball{
  float x, y;
  float vx ,vy;

  Ball(float x0, float y0, float vx0, float vy0){
    x = x0;
    y = y0;
    vx = vx0;
    vy = vy0;
  }
  void idou(){
    x += vx;
    y += vy;
  }
  void byouga(){
    ellipse(x, y, 10, 10);
  }
}

Ball[] bs;
int n;

void tsuika(float x, float y, float vx, float vy){
  Ball b = new Ball(x, y, vx, vy);
  bs[n] = b;
  n++;
}
void sakujo(int i){
  bs[i] = bs[n-1];
  n--;
}



void setup(){
  size(300, 300);
  frameRate(30);
  bs = new Ball[100];
}

void draw(){
  background(255, 255, 255);
  for(int i=0; i<n; i++){
    bs[i].idou();
    bs[i].byouga();
  }
  for(int i=n-1; i>=0; i--){
    if(bs[i].x<0 || bs[i].x>width || bs[i].y<0 || bs[i].y>height){
      sakujo(i);
    }
  }
}

void mousePressed(){
  tsuika(mouseX, mouseY, random(-1, 1), random(-1, 1));
}
```

・コンストラクタ

new するとオブジェクトが生成されるが、それと同時にオブジェクトの変数の初期化もついでに行うのがコンストラクタである。前のページの例では、

```
Ball b = new Ball(x, y, vx, vy);
```

というところで、クラスの定義のところで書かれたコンストラクタ

```
Ball(float x0, float y0, float vx0, float vy0){
  x = x0;
  y = y0;
  vx = vx0;
  vy = vy0;
}
```

を呼び出して、その引数に new のところの引数 x, y, vx, vy を入れている。

コンストラクタの特徴は、**命令の名前がクラス名と同じで、戻り値が何も書かれていない**ということである。

課題

Part3_class_2 の Ball クラスに、描かれる○の直径を表す変数 d を追加して、コンストラクタでこれらの値も初期化出来るように、byouga でこの直径の○が描かれるようにせよ。

2　クラスを持つクラス

クラスの集まりを一つの集団として扱いたい時、それらをさらに別のクラスのなかに入れるのが良い。例えば東京 23 区が東京都クラスの中にあって、さらに 47 都道府県が日本クラスの中にあると考えることが出来る。下位のクラスのメンバーを使うときは

　　日本.東京都.新宿区

のように、ドットを連鎖させる。

先ほどの Part3_class_2 では bs という配列で Ball オブジェクトの集団を扱っていた。ここでもし bs1 と bs2 という 2 つの Ball の集団を扱わないといけないとすると、n1, n2, tsuika1, tsuika2, sakujo1, sakujo2 というように、これらの変数、命令も 2 つに増やさないといけない。これは面倒なので、Ball の集団を扱う Balls クラスを作って、そこに n, tsuika, sakujo を持ってくる。これが Part3_class_3 である。こうすることで Ball の集団が複数あった時にもプログラムをシンプルに保つことが出来る。

次のような点に注目して欲しい

・本体と Balls を別のタブに分離して、プログラムを見やすくしている。
　また、これらは別のファイルになるので、別のプログラムで Balls を使いたい時は
　これをコピーしてそのまま使うことが出来る。
・Balls の get という命令は、引数として整数 i をとって、bs に格納された
　i 番目の Ball オブジェクトを返す。このようにオブジェクトを返す命令も
　作ることが出来る。

課題

・この例を理解する
・前のページの課題の変数 d を、こっちのプログラムでも使えるようにせよ

Part3_class_3 § | Balls

```
Balls bs;

void setup(){
  size(300, 300);
  frameRate(30);
  bs = new Balls();
}

void draw(){
  background(255, 255, 255);
  for(int i=0; i<bs.n; i++){
    bs.get(i).idou();
    bs.get(i).byouga();
  }
  for(int i=bs.n-1; i>=0; i--){
    if(bs.get(i).isout()==1){
      bs.sakujo(i);
    }
  }
}

void mousePressed(){
  bs.tsuika(mouseX, mouseY, random(-1, 1), random(-1, 1));
}
```

Part3_class_3 § | Balls

```
class Balls{
  int n;
  Ball[] bs;
  Balls(){
    bs = new Ball[1000];
  }
  Ball get(int i){
    return bs[i];
  }
  void tsuika(float x, float y, float vx, float vy){
    Ball b = new Ball(x, y, vx, vy);
    bs[n] = b;
    n++;
  }
  void sakujo(int i){
    bs[i] = bs[n-1];
    n--;
  }
}

class Ball{
  float x, y;
  float vx ,vy;

  Ball(float x0, float y0, float vx0, float vy0){
    x = x0;
    y = y0;
    vx = vx0;
    vy = vy0;
  }
  void idou(){
    x += vx;
    y += vy;
  }
  void byouga(){
    ellipse(x, y, 10, 10);
  }
  int isout(){
    if(x<0 || x>width || y<0 || y>height) return 1;
    else return 0;
  }
}
```

3　命令にクラスを渡す

命令の引数としてクラスを使うことが出来る。このとき int や float の変数を渡すのとは少し挙動が違うので注意が必要である。

・変数の時

引数として渡された変数は、「コピー」である。

```
void test(int a){
        a = 0;
}
…
b=10;
test(b);
```

というプログラムを実行すると、b の中身 10 が test 命令の中で作られる変数 a にコピーされて a は 10 になり、a=0;により a は 0 になる。しかし、a はあくまで b のコピーなので、コピー元である**b は 10 のまま**である。

・クラスの時

引数としてオブジェクトを渡してそのオブジェクトに変更を加えると、**コピー元の方に変化がおよぶ**。

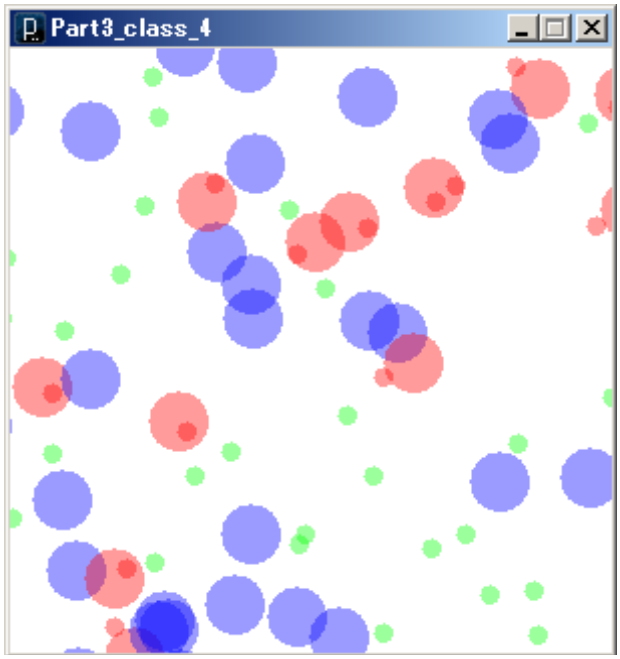
```
Void test2(Ball a){
        a.x = 0;
}
…
Ball b = new Ball(1, 2, 3, 4);
test2(b);
```

というプログラムを実行すると、test2 が実行された時に b のメンバー変数 x は 0 になっている。 ちなみに配列もオブジェクトの一種なので、命令の引数として配列を使っても同じことがおこる。

課題

・2 つの Ball オブジェクトを引数にとって、両者が接触していたら 1 を、していなければ 0 を返す命令を作れ。円と円との当たり判定を使えば良い。

・Ball クラスに変数 r, g, b を追加して、setColor 命令を作り、表示する Ball オブジェクトの色を設定できるようにせよ。
・Balls クラスを 2 つ作って二種類の Ball 集団を扱うことにする。それらの集団を a, b としよう。まず、a と b とで表示する色を分けよ。
・a のオブジェクトと b のオブジェクトで接触しているものがあれば、そこだけさらに色を変えよ。
　これはシューティングゲームの弾と敵、弾と自機との当たり判定に使える。



Part3_class_4 が解答。青と緑が接触していると、それらを赤で描いている↑

4　クラスの継承

シューティングゲームの敵、弾、自機のように、種類は違うけど中身はそんなに変わらないようなクラスたちを作るときは、共通する部分を持った**親クラス**を先に作って、それを**継承**することで**子クラス**を作り、そこには子クラス独自の変数と命令だけを書くようにすることができる。

Ball クラスを継承した子クラス Enemy を作るには、次のようにする。

```
class Enemy extends Ball{
        Enemy 独自の変数たち
        Enemy のコンストラクタ
        Enemy 独自の命令たち
}
```

コンストラクタについては親クラスのものが使えないので、新たに作る必要がある。

・子クラスは親クラスの持っている変数と命令をそのまま引き継ぐ

Enemy クラスから生成されたオブジェクトは、Ball クラスにある変数 x や命令 idou()をそのまま使うことが出来る。

```
Enemy e = new Enemy();
e.x = 1;
```

・親クラスと同じ名前の命令を作ったら、親クラスの命令は上書きされる

Enemy クラスの中で idou()命令を作ったら、Enemy クラスのオブジェクトで idou()命令を実行すると、Ball クラスの idou()ではなく、Enemy で新たに作った idou()の方が実行される。

・親が使える場面では、子を代わりに使うことができる

例えば Ball クラスのオブジェクトを格納するクラスである Balls には、Ball を継承している全ての子クラスも格納できる。

だから Enemy クラスのオブジェクトを tsuika(Ball b)命令で追加出来る。

クラスの継承を使った例（Part3_class_5）

Part3_class_5 §

```
class Oya{
  float x, y;
  Oya(){          //子のコンストラクタをつかうには親に空のコンストラクタが必要
  }
  Oya(float x0, float y0){   //引数が違えば同じ名前の命令を2つ作っても良い
    x = x0;
    y = y0;
  }
  void byouga(){
    ellipse(x, y, 10, 10);
  }
}

class Ko1 extends Oya{   //子クラス１
  float vx, vy;
  Ko1(float x0, float y0, float vx0, float vy0){
    x = x0;
    y = y0;
    vx = vx0;
    vy = vy0;
  }
  void idou(){
    x += vx;
    y += vy;
  }
}

class Ko2 extends Oya{    //子クラス２
  Ko2(float x0, float y0){
    x = x0;
    y = y0;
  }
  void byouga(){          //親クラスと同じ名前の命令を作って上書きしている
    rect(x, y, 10, 10);
  }
}

Oya[] k1, k2;

void setup(){
  size(200, 200);
  k1 = new Oya[100];
  k2 = new Oya[100];
  for(int i=0; i<100; i++){   //親クラスの配列に子クラスのオブジェクトを入れる
    Ko1 a = new Ko1(random(0, width), random(0, height), random(-1, 1), random(-1, 1));
    k1[i] = a;
    Ko2 b = new Ko2(random(0, width), random(0, height));
    k2[i] = b;
  }
}

void draw(){
  background(255);
  for(int i=0; i<100; i++){
    Ko1 a = (Ko1)k1[i];    //配列の型はOyaなので、Ko1に変換するために(Ko1)をつける
    a.idou();              //idou命令はKo1の命令なので、Oyaのままでは使えないから一行前でKo1に変換しているのだ
    a.byouga();
    k2[i].byouga();        //描画命令はOyaにもあるのでこのまま使える
  }
}
```

課題

シューティングゲームを作る。

・Ball クラスを継承して、Enemy, Tama, Jiki クラスを作る。

・hp 変数を加えたり、idou 命令を書き換えたりする。

・hp が 0 になったら消える。

・弾を撃つ命令を追加する。

・攻撃パターンのバリエーションを作ると面白い。