2018 年度 コンテンツ工学システム

Processing 入門用　テキスト

目次

1. はじめに

命令の基本的な書き方

1 つの命令に対して、いくつかの数値を指定する 1 文で 1 つの図形を描画します。

命令 (命令に渡すデータ) セミコロン
例 `point(40, 40);`

プログラムは上から順に実行されるため、後で書いたもので上書きされるということを考えて、記述する順番にも気を付けましょう。

最後のセミコロンは忘れないようにしましょう。

2. 画面を表示して点と線を書いてみよう

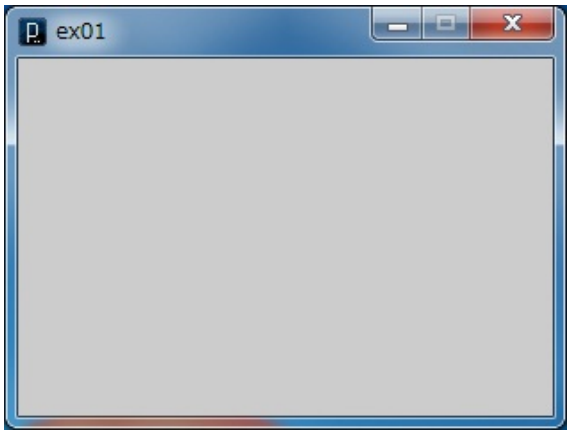
2.1 ウインドウの大きさ指定

以下のプログラムを書き込んで実行してみましょう。

// を先頭に書かれた行や、// の右に書かれた文字は書かなくて構いません。

```
// ウインドウの大きさの設定
size(300, 200);          // size(横幅, 縦幅);
```

実行するとグレーに塗りつぶされたウインドウが表示されます。

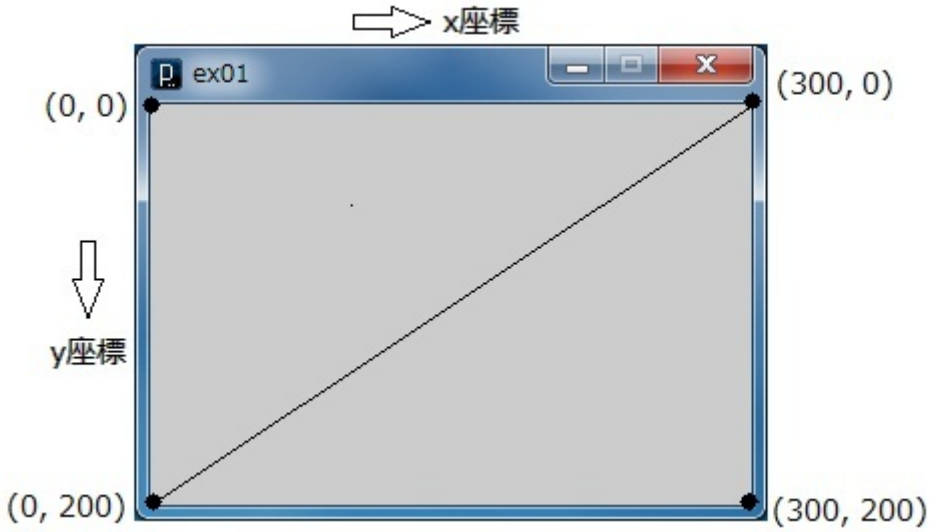


2.2 点と線の描画

以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
// ウインドウサイズの設定
size(300, 200);     // size(横幅,縦幅);

// 点の描画
point(100, 50);          // point(x 座標, y 座標);

// 線の描画
line(300, 0, 0, 200);
                    // 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);
```

実行すると点と線が描画されたウインドウが表示されます。

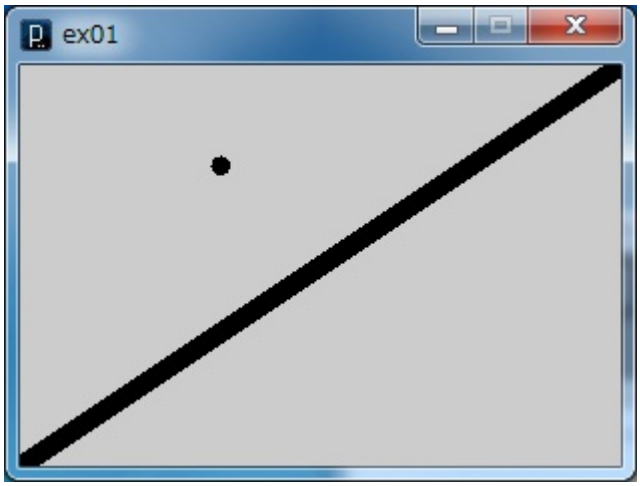


座標は左上を(0, 0)とし、下方向に y が増加し、右方向に x が増加します。

このままでは点が見えにくいため線の太さを変える命令を使いましょう。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
size(300, 200);     // size(横幅,縦幅);

// 線の太さの設定
strokeWeight(10);        // strokeWeight(線の太さ);

// 点の描画
point(100, 50);     // point(x 座標, y 座標);

// 線の描画
line(300, 0, 0, 200);
                  // 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);
```

実行すると以下の様に点が大きくなり、線は太くなります。



・size 命令を 2 つ書いてプログラムが上から順番に実行されることを確かめましょう。

例　　size(300, 200);

　　size(400, 500);

下に書いた命令で指定したサイズで表示されます。

早く終わった方

・横幅 700、縦幅 500 以下の範囲でウインドウサイズの変化をさせましょう。

・点や線の座標の数値を変更して描画位置の変更や、命令を増やして点や線を増やしてみましょう。

例　line(0, 0, 150, 100);　　を追加する等

また、線の描画を使って四角形(4 本の線を描画)や三角形(3 本の線を描画)を作ってみましょう。

命令の復習：

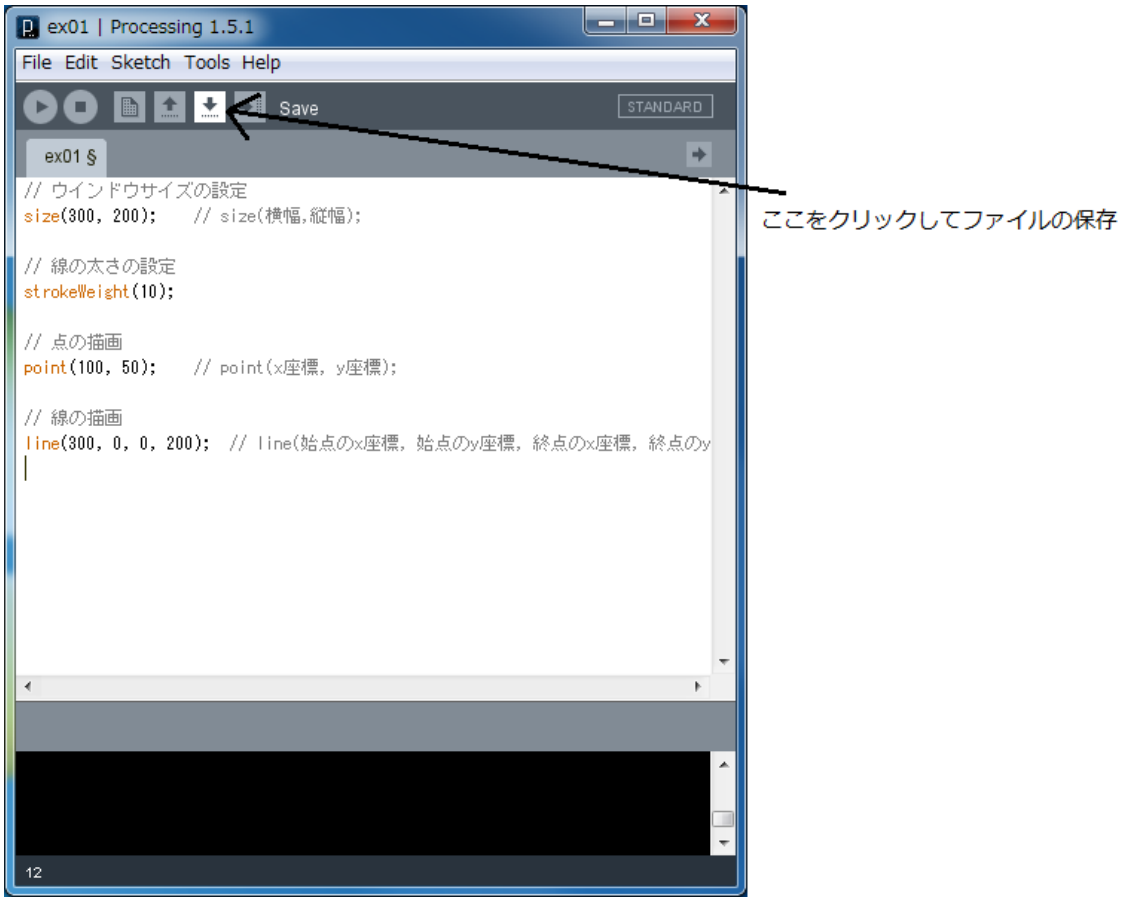
ここで使った命令

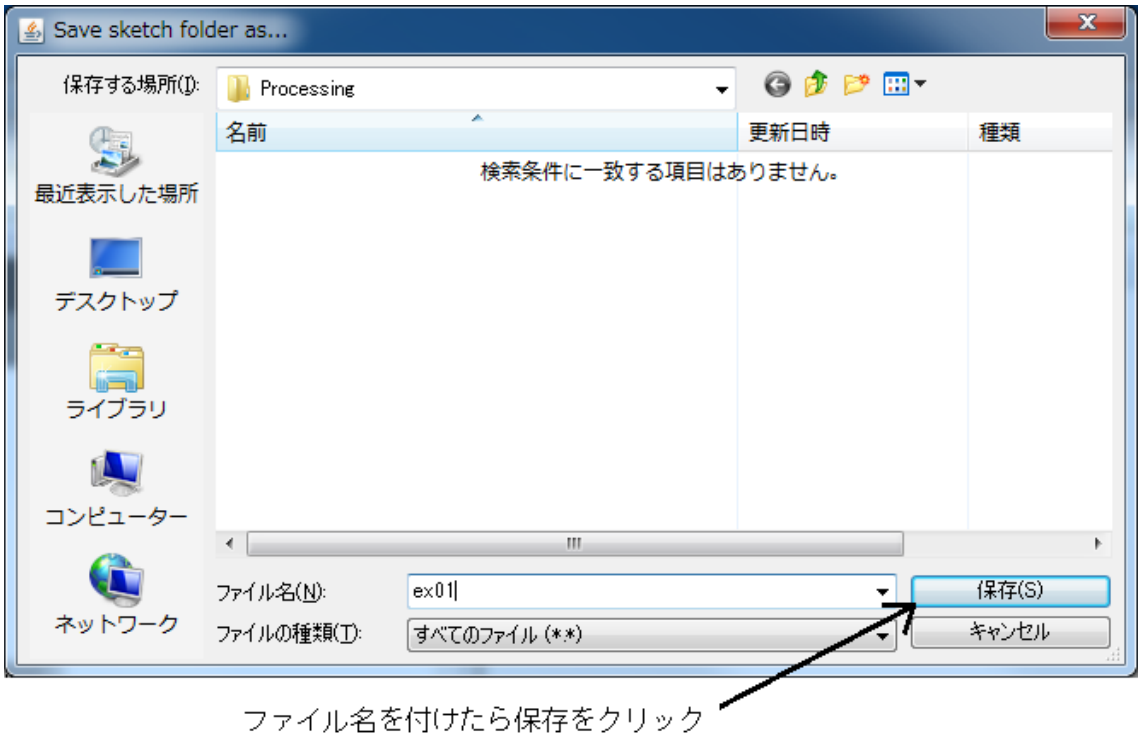
- size(横幅, 縦幅);
- point(x 座標, y 座標);
- line(始点の x 座標, 始点の y 座標, 終点の x 座標, 終点の y 座標);
- strokeWeight(線の太さ);

2.3 ファイルの保存

書いたプログラムはこまめに保存しましょう。

① 左上にある下矢印をクリック





② ファイル名を決定して保存をクリック

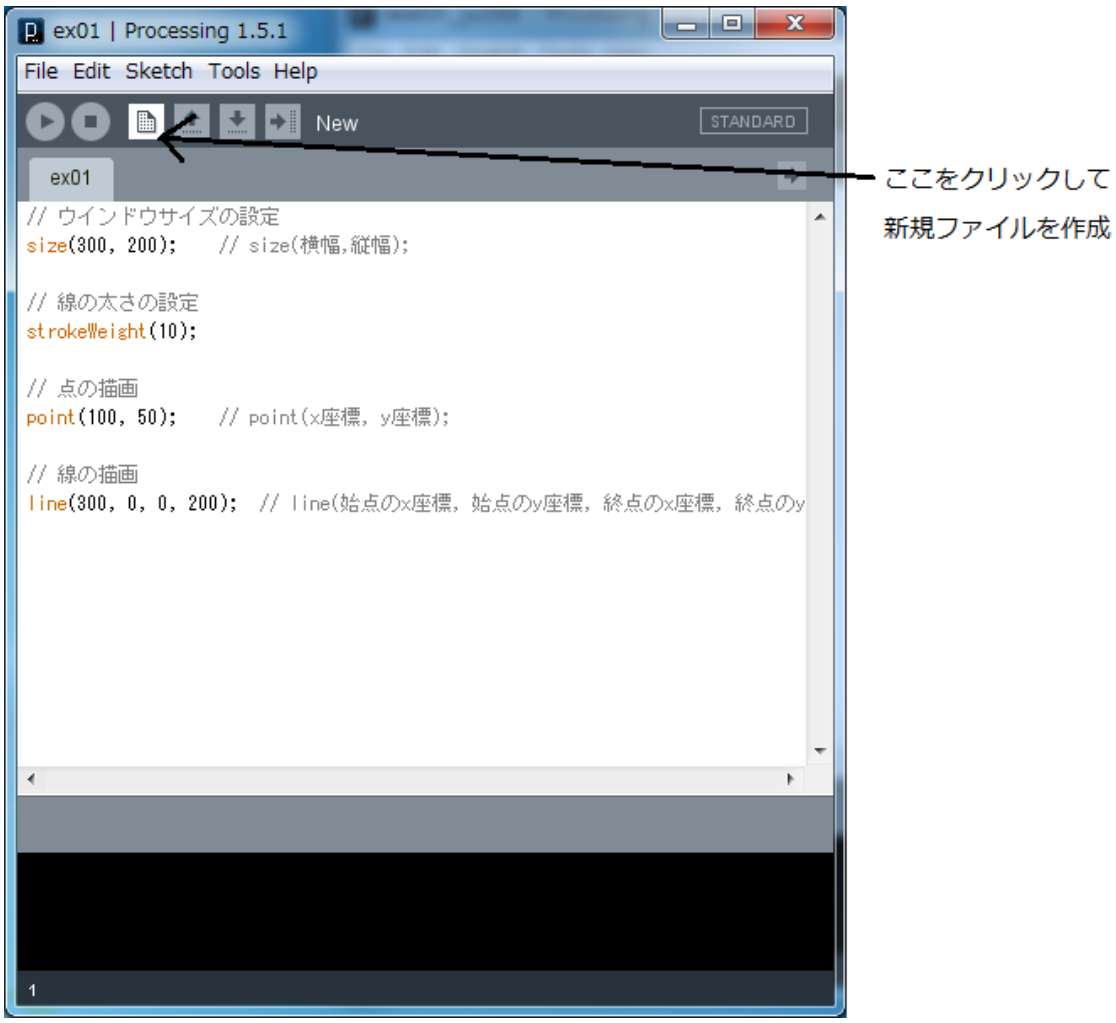
今回は ex01 として保存しましょう。

マイドキュメントの中の Processing という名前のフォルダに保存されます。

保存した場所につけた名前(ex01)のフォルダが作られ、その中につけた名前(ex01.pde)のプログラムが保存されます。

3. 図形を書いてみよう1

図の紙のようなボタンをクリックして新規ファイルを作成します。

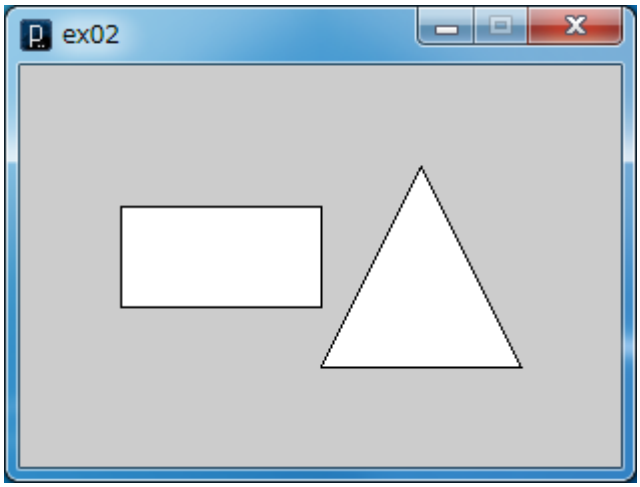


3.1 四角形、三角形の描画

以下のプログラムを１つずつ書き加えながら実行してみましょう。

```
size(300, 200);

// 四角形の描画
rect(50, 70, 100, 50);  // rect(左上頂点の x 座標, 左上頂点の y 座標, 横幅, 縦幅);

// 三角形の描画
triangle(200, 50, 150, 150, 250, 150); // triangle(頂点１の x 座標, 頂点１の y 座標,
                                       //          頂点２の x 座標, 頂点２の y 座標,
                                       //          頂点３の x 座標, 頂点３の y 座標);
```

実行すると次のようになります。

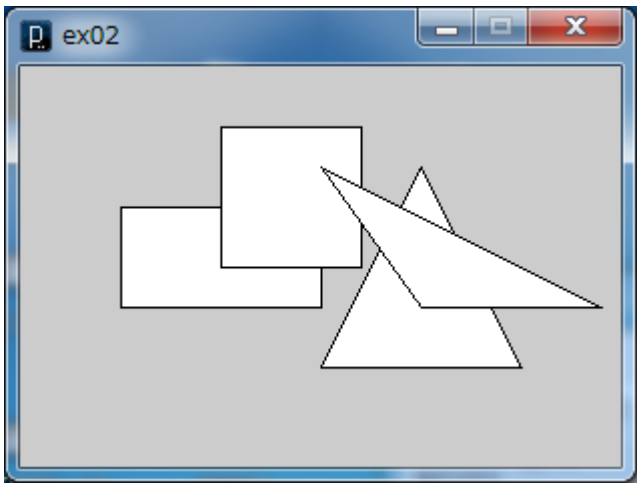


3.2 重ね描き

図形が重なるような描画命令をされたとき、後から描画されたものが上になるのを
確認しましょう。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
size(300, 200);

// 四角形の描画
rect(50, 70, 100, 50);  // rect(左上頂点の x 座標, 左上頂点の y 座標, 横幅, 縦幅);

// 三角形の描画
triangle(200, 50, 150, 150, 250, 150); // triangle(頂点 1 の x 座標, 頂点 1 の y 座標,
                                       //          頂点 2 の x 座標, 頂点 2 の y 座標,
                                       //          頂点 3 の x 座標, 頂点 3 の y 座標);

rect(100, 30, 70, 70);  // 左上頂点の座標が(100,30)の 70×70 の正方形

triangle(200, 120, 150, 50, 290, 120);
                    // 頂点 1(200, 120), 頂点 2(150, 50), 頂点 3(290, 120)の三角形
```

実行すると以下のようになります。
追加した四角と三角が上に描画されています。



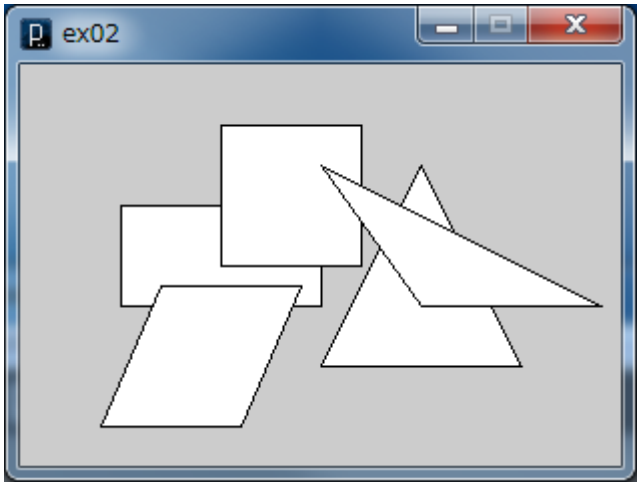
完成したら名前を ex02 として保存しましょう。

早く終わった方

自由に四角形を描画できる quad を追加して平行四辺形を作ってみましょう。

```
// 平行四辺形の描画
quad(70, 110, 140, 110, 110, 180, 40, 180);// quad(頂点 1 の x 座標, 頂点 1 の y 座標,
                                            //      頂点 2 の x 座標, 頂点 2 の y 座標,
                                            //      頂点 3 の x 座標, 頂点 3 の y 座標,
                                            //      頂点 4 の x 座標, 頂点 4 の y 座標);
```

実行すると以下のようになります。



余裕があれば台形や、ひし形も作ってみましょう。

命令の復習：

ここで使った命令

- rect(左上頂点の x 座標, 左上頂点の y 座標, 横幅, 縦幅);
- triangle(頂点 1 の x 座標, 頂点 1 の y 座標,
  頂点 2 の x 座標, 頂点 2 の y 座標,
  頂点 3 の x 座標, 頂点 3 の y 座標);
- quad(頂点 1 の x 座標, 頂点 1 の y 座標, 頂点 2 の x 座標, 頂点 2 の y 座標,
  頂点 3 の x 座標, 頂点 3 の y 座標, 頂点 4 の x 座標, 頂点 4 の y 座標);

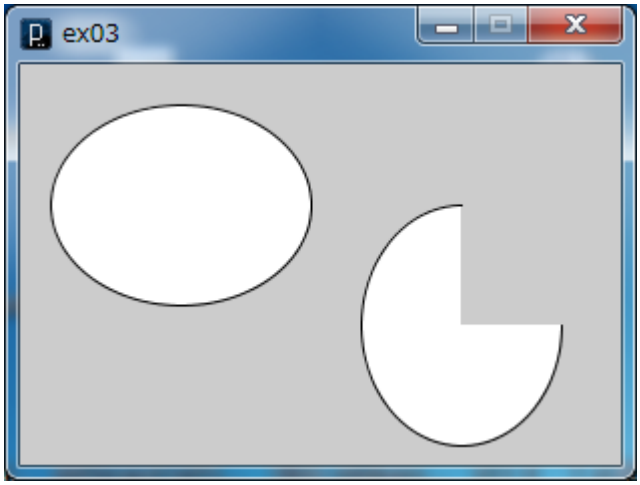
4. 図形を書いてみよう2

ファイルを新規作成します。

以下のプログラムを 1 つずつ書き加えながら実行しましょう。

```
size(300, 200);

smooth();  // 曲線を描画する際に、線を滑らかにするためにつける

// 円の描画
ellipse(80, 70, 130, 100);  // ellipse(中心の x 座標, 中心の y 座標, 横直径, 縦直径);

// 円弧の描画
arc(220, 130, 100, 120, radians(0), radians(270));
                                        // arc(中心の x 座標, 中心の y 座標,
                                        //      横直径, 縦直径,
                                        //     書き始めの角度, 書き終わりの角度);
```

2 つとも書き終えて実行すると次のようになります。



完成したら名前を ex03 として保存しましょう。

早く終わった方

円の直径や、円弧の表示する角度を変えましょう。

例　`ellipse(80, 80, 100, 100);`

　　`arc(220, 130, 100, 100, radians(-90), radians(90));`

命令の復習：

ここで使った命令

- ellipse(中心の x 座標, 中心の y 座標, 横直径, 縦直径) ;
- arc(中心の x 座標, 中心の y 座標, 横直径, 縦直径,
  書き始めの角度, 書き終わりの角度);

5. 色を変えよう

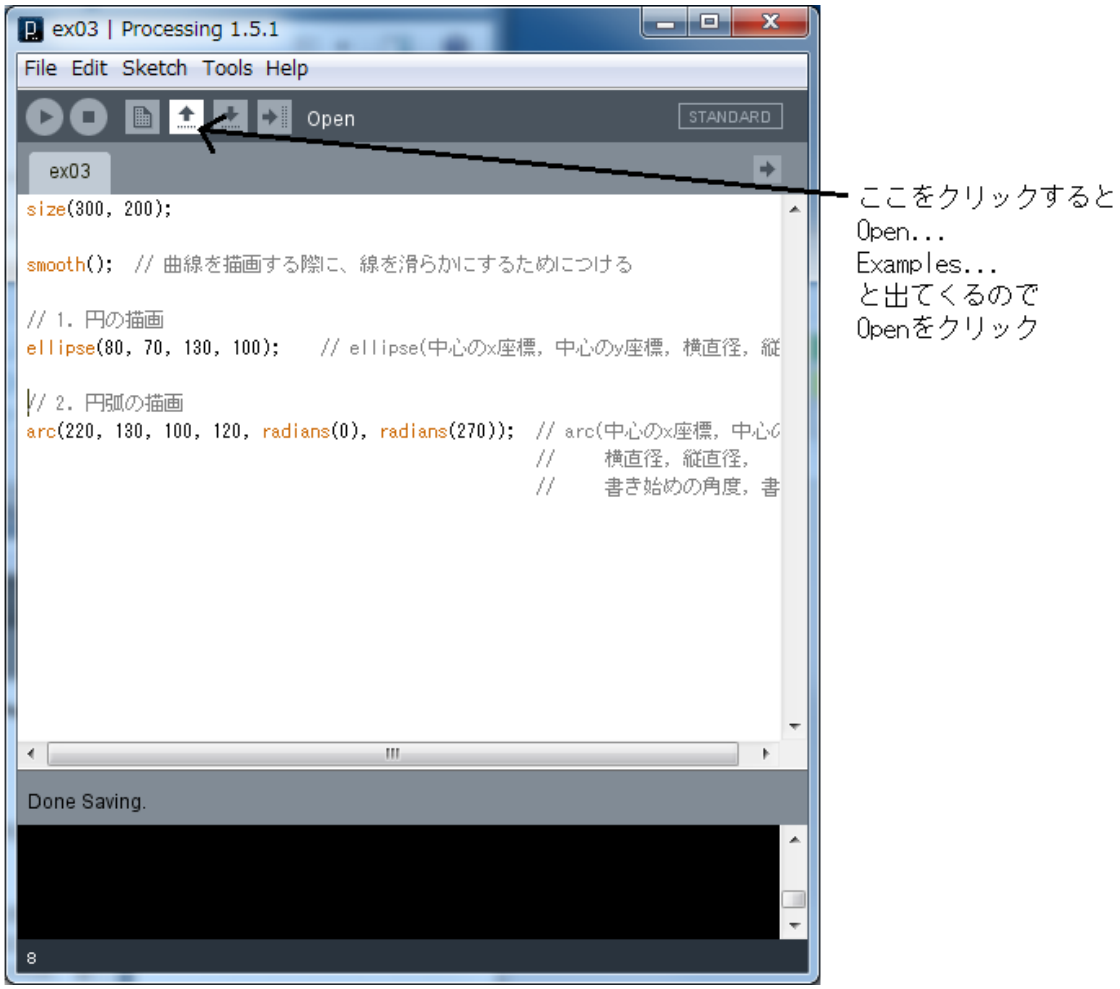
Processing では背景の色、図形の枠の色、図形の中の色を付けることができます。

5.1 背景色の変更

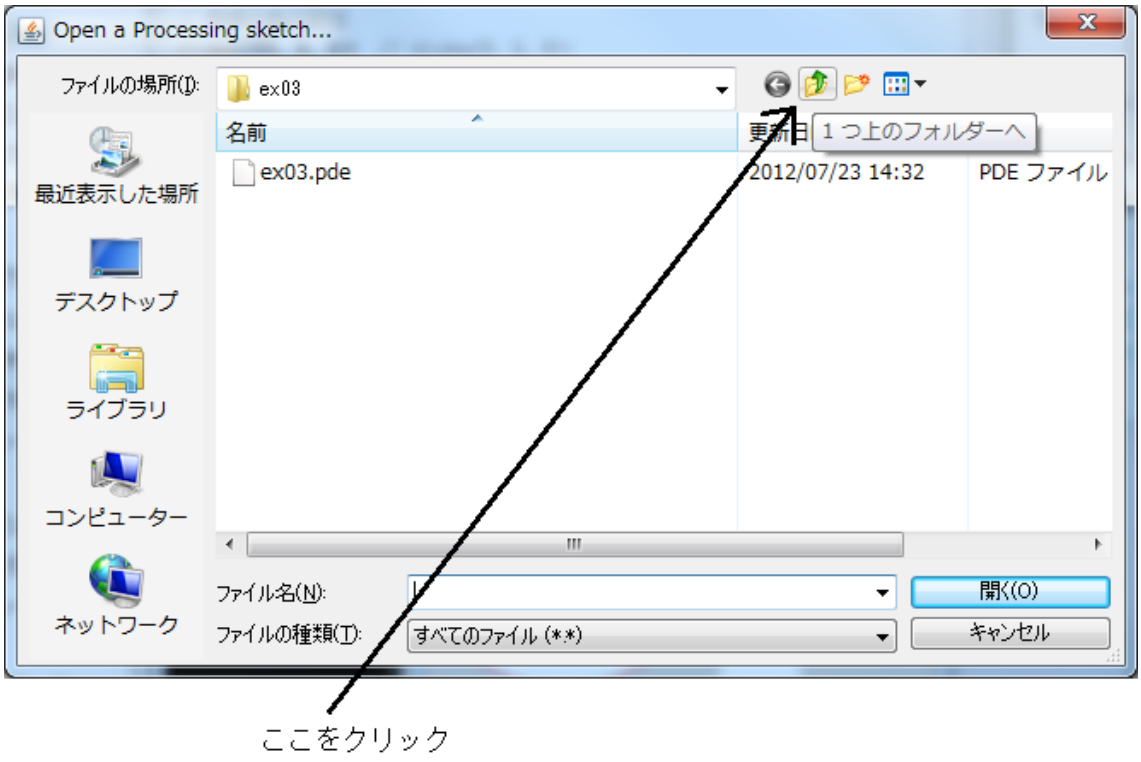
3 章で作った ex02 を開きましょう。

ファイルは以下のようにして開きます。

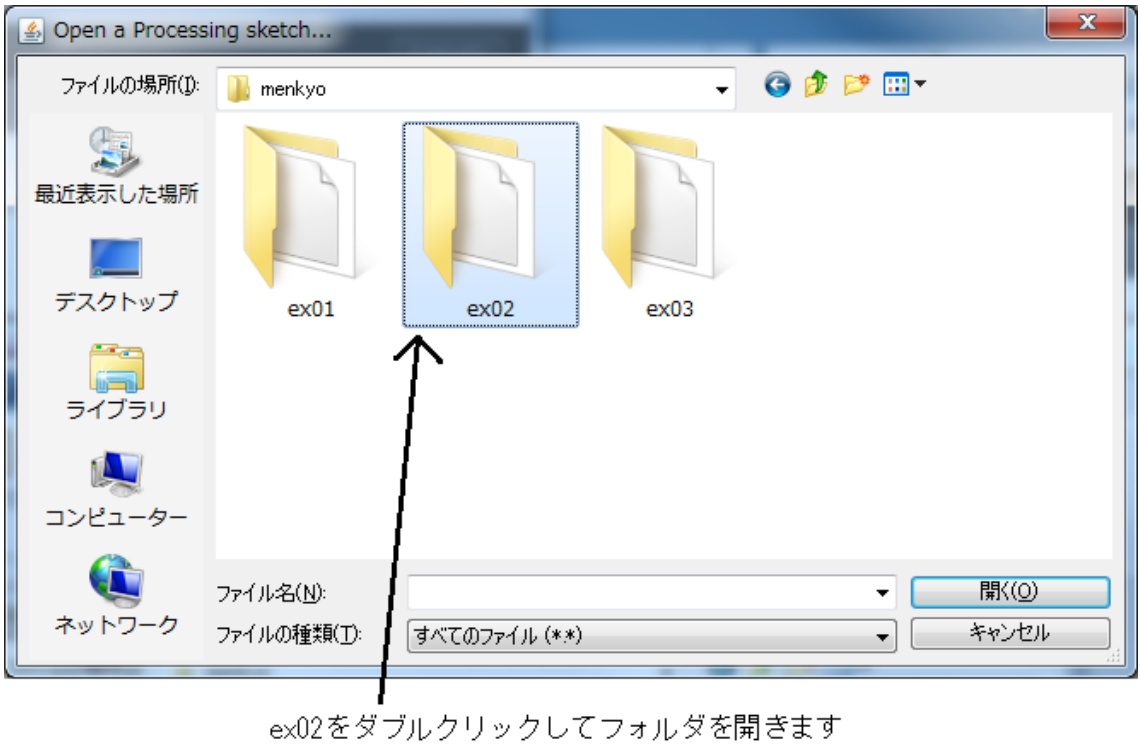
① 図の上向き矢印のボタンをクリック



② 以下のような画面が表示されるので 1 つ上のフォルダーへ移動するボタンをクリックします。



③ 以下のような画面に代わるので ex02 のフォルダを開きます。

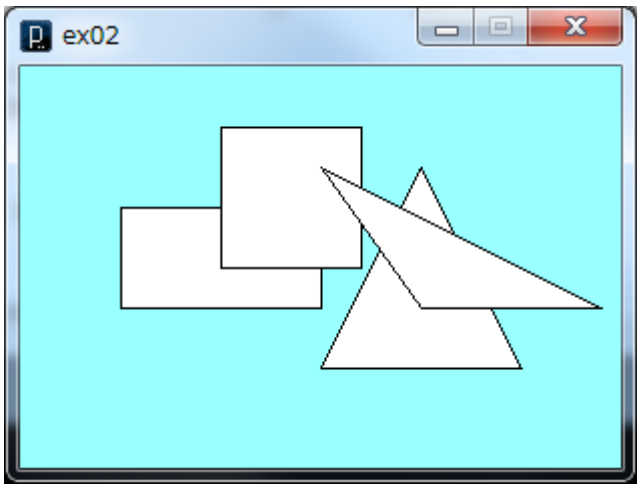


④　ex02.pde という名前のファイルが表示されるのでそれをダブルクリック、またはクリックして右下の開くボタンを押すことでファイルが開かれます。

以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。
色は光の三原色(赤:R、緑:G、青:B)のそれぞれの 0~255 の数字によって決まります。
数字が大きいほどその色の明るさが大きくなります。

```
size(300, 200);

// 背景色の設定
background(153, 255, 255);  // background(R, G, B);

// 四角形の描画
rect(50, 70, 100, 50);  // rect(左上頂点の x 座標, 左上頂点の y 座標, 横幅, 縦幅);
        :
        :
```

実行すると背景の色が水色に変わります。

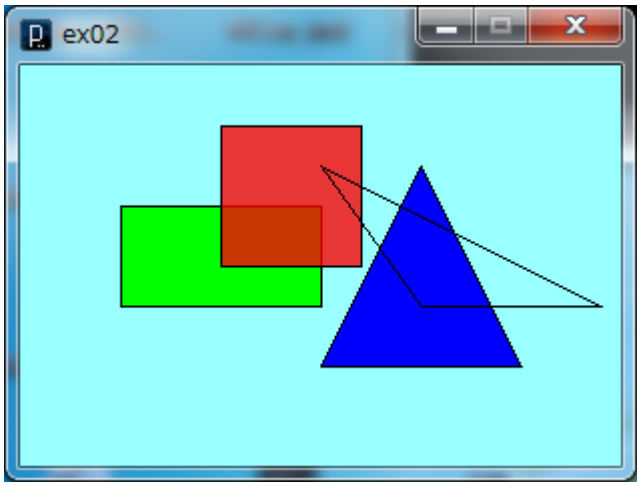


5.2 図形の塗りつぶし

図形の塗りつぶしをする色を設定します。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
size(300, 200);

// 背景色の設定
background(153, 255, 255);  // background(R, G, B);

// 塗りつぶし色の設定
fill(0, 255, 0);      // fill(R, G, B);

// 四角形の描画
rect(50, 70, 100, 50);  // rect(左上頂点の x 座標, 左上頂点の y 座標, 横幅, 縦幅);

// 塗りつぶし色の設定
fill(0, 0, 255);      // fill(R, G, B);

// 三角形の描画
triangle(200, 50, 150, 150, 250, 150); // triangle(頂点 1 の x 座標, 頂点 1 の y 座標,
                                       //          頂点 2 の x 座標, 頂点 2 の y 座標,
                                       //          頂点 3 の x 座標, 頂点 3 の y 座標);

// 透明度を含めた塗りつぶし色の設定
fill(255, 0, 0, 200);  // fill(R, G, B, 不透明度);

rect(100, 30, 70, 70);

noFill();       // 塗りつぶしをなくす

triangle(200, 120, 150, 50, 290, 120);
```

すべて書き加えて実行すると以下のようになります。



完成したら保存のボタンをクリックして保存しましょう。
先ほどのプログラムを更新して保存になるため、ボタンをクリックするだけで保存できます。

早く終わった方

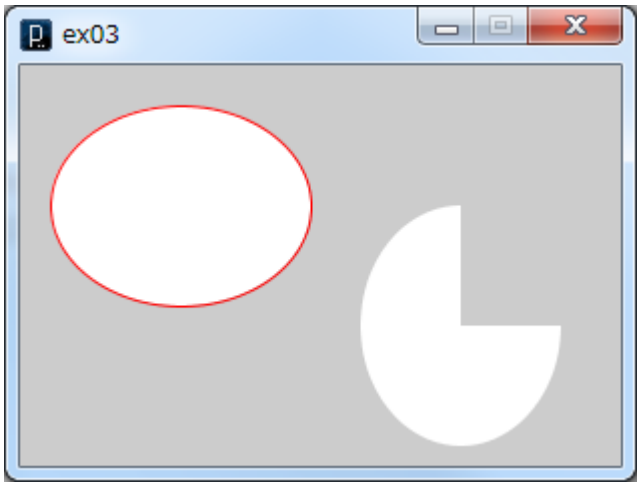
R, G, B の値を変更して、どのような値でどのような色になるのか見てみましょう。
例　`fill(200, 100, 100, 100);`

5.3 図形枠線の変更

ex03 を開きましょう。
図形の枠線の色の設定と枠線の無効化をします。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
size(300, 200);

smooth();  // 曲線を描画する際に、線を滑らかにするためにつける

// 枠線の色の設定
stroke(255, 0, 0);  // stroke(R, G, B);

// 円の描画
ellipse(80, 70, 130, 100);     // ellipse(中心の x 座標, 中心の y 座標, 横直径, 縦直径);

// 枠線をなくす
noStroke();        // かっこの中身は必要ありません

// 円弧の描画
arc(220, 130, 100, 120, radians(0), radians(270));
```

すべて書き加えて実行すると以下のようになります。



完成したら保存しましょう。

早く終わった方

枠線の色を変更してみましょう。

strokeWeight();を追加して太くした線の色を変えてみましょう。

例　strokeWeight(3);

　　stroke(200, 200, 100);

命令の復習：

ここで使った命令

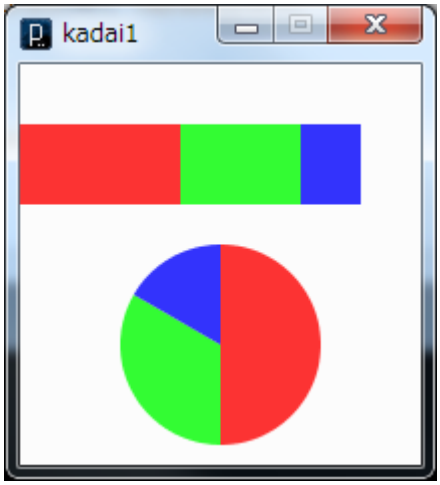
- background(R, G ,B);
- fill(R, G ,B);
- noFill();
- stroke(R, G, B);
- noStroke();

製作課題1　棒グラフと円グラフを作ってみよう

新しいファイルを開いて次のプログラムを書き込んで実行してみましょう。

```
size(200, 200);

colorMode(RGB, 100);
background(99);

smooth();
noStroke();

fill(99, 20, 20);
rect(0, 30, 90, 40);
arc(100, 140, 100, 100, radians(-90), radians(90));

fill(20, 99, 20);
rect(80, 30, 60, 40);
arc(100, 140, 100, 100, radians(90), radians(210));

fill(20, 20, 99);
rect(140, 30, 30, 40);
arc(100, 140, 100, 100, radians(210), radians(270));
```

実行すると以下のようになります。



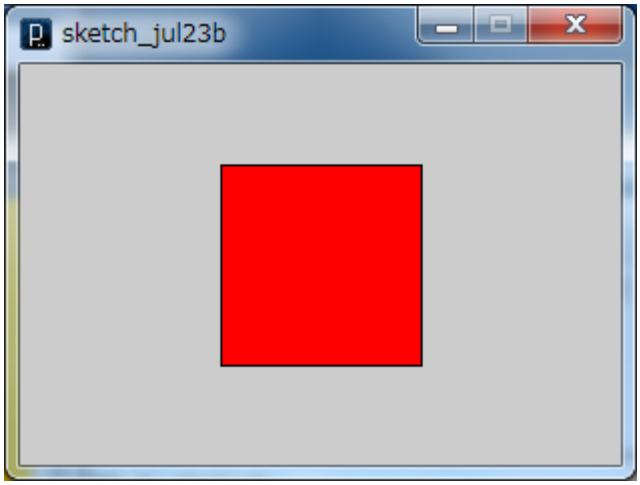
6. 変数と計算を使ってきれいに書こう

6.1 変数

変数と呼ばれる箱のようなものに数字などのデータを入れて扱います。
新しいファイルを開いて次のプログラムを書き込んで実行してみましょう。

```
size(300, 200);

// 変数の設定
int x = 100;          // 変数 x を整数として設定、100 を入れる
int y = 50;           // 変数 y を整数として設定、100 を入れる
int range = 100;      // 変数 range を整数として設定、100 を入れる

// 変数の使用
fill(255, 0, 0);
rect(x, y, range, range);  // rect(100, 50, 100 ,100);と同じ
```

実行すると以下のようになります。

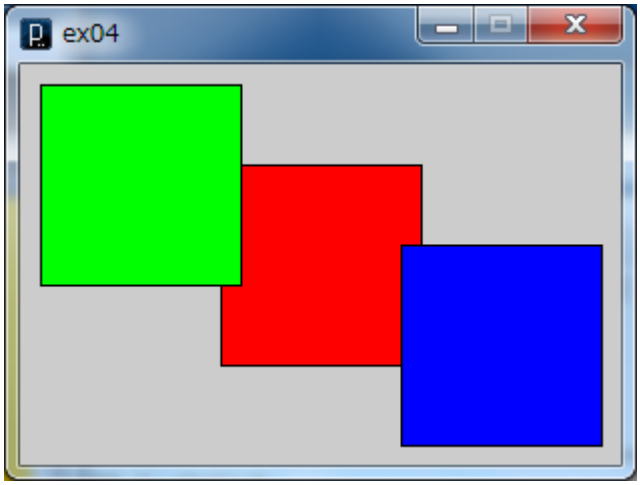


6.2 計算

＋は足し算、－は引き算を表し、計算を行って変数の値を変化させたり、計算式を命令の数値に使うことができます。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
size(300, 200);

// 変数の設定
int x = 100;          // 変数 x を整数として設定、100 を入れる
int y = 50;           // 変数 y を整数として設定、100 を入れる
int range = 100;      // 変数 range を整数として設定、100 を入れる

// 変数の使用
fill(255, 0, 0);
rect(x, y, range, range);  // rect(100, 50, 100 ,100);と同じ

// 計算値の使用
fill(0,255,0);
rect(x-90, y-40, range, range);  // rect(10, 10, 100, 100)と同じ

fill(0,0,255);
rect(x+90, y+40, range, range);  // rect(190, 90, 100, 100)と同じ
```

実行結果は以下のようになります。

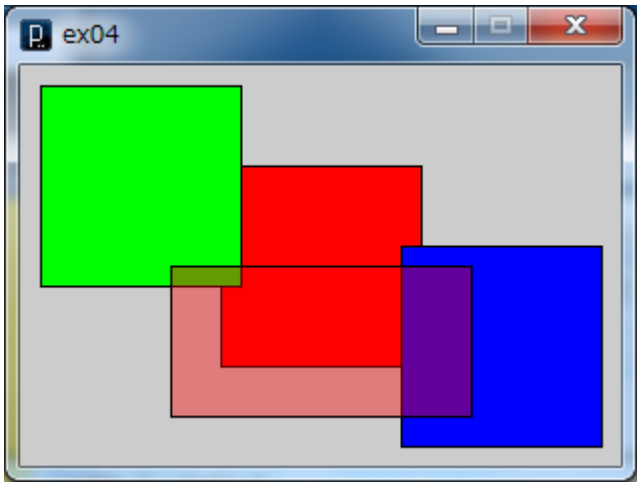


6.3 変数の値の更新

変数へ新たな値を入れることで変数の中身を変えます。
＊は掛け算、/ は割り算を表し、＋やーと同様に計算を行って変数の値を変化させたり、計算式を命令の数値に使うことができます。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
        :
        :

// 計算値の使用
fill(0,255,0);
rect(x-90, y-40, range, range);  // rect(10, 10, 100, 100)と同じ

fill(0,0,255);
rect(x+90, y+40, range, range);  // rect(190, 90, 100, 100)と同じ

// 変数の値の更新
range = range + 50;     // range に range+50 (100+50) を入れる
fill(255,0,0,100);
rect(x-25, y*2, range, range/2);  // rect(75, 100, 150, 75)と同じ
```

実行結果は以下のようになります。



ここまで完成したら ex04 として保存しましょう。

早く終わった方

変数の値を変えて図形全体がまとまって移動することを確認してみましょう。
x の値を大きくすれば右に移動し、y の値を大きくすれば下へ移動します。
range の値を大きくすればすべての四角形が大きくなります。

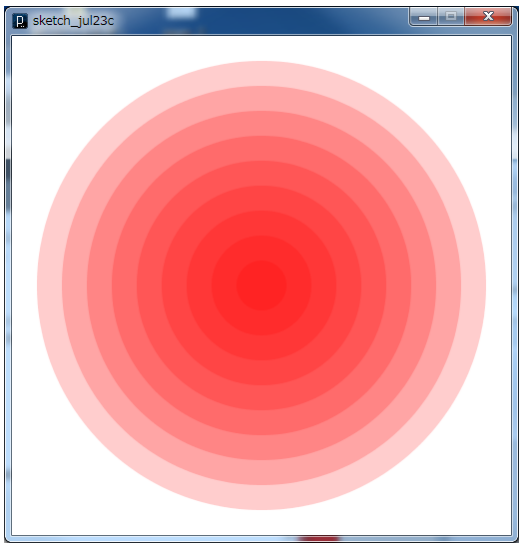
7. 繰り返し命令、条件命令

7.1 繰り返し命令 —for 文

{}で囲まれた中身を繰り返し実行します。
新しいファイルを開きましょう。
以下のプログラムを書き込んで実行してみましょう。

```
size(500, 500);

background(255);  // 背景を白に設定

smooth();
noStroke();
fill(255, 0, 0, 50);              // 透過させた赤色を設定

// 中心点から円の直径を大きくしながら描画
// 変数 i を設定し 1 を入れる; i が 10 以下の間繰り返す; 繰り返しの最後に i を 1 増やす
for(int i=1; i<10; i++){
  ellipse(250, 250, i*50, i*50);  // 円の直径を大きくしながら描画
}
```

実行結果は以下のようになります。
中心から描画されるため、重ねて描画されるたびに色が濃くなります。

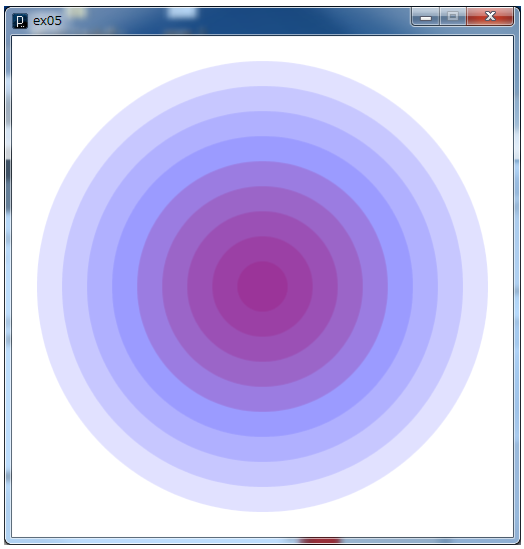


7.2 条件命令 —if 文

()の中の条件に当てはまれば{}の中の命令を実行します。
else はその条件外での実行内容です。
以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
          :
          :
// 中心点から円の直径を大きくしながら描画していく
// 変数 i を設定し 1 を入れる; i が 10 以下の間繰り返す; 繰り返しの最後に i を 1 増やす
for(int i=1; i<10; i++){

  // 変数 i が 5 より大きければ青色を設定
  // そうでなければ赤色を設定する
  if(i > 5){
    fill(0, 0, 255, 30);
  }else {
    fill(255, 0, 0, 50);
  }

  ellipse(250, 250, i*50, i*50);  // 円の直径を大きくしながら描画
}
```

実行結果は以下のようになります。
最初は赤色の円が描画され、途中から青色の円に変わります。



完成したら ex05 として保存しましょう。

早く終わった方

繰り返しの回数や、条件の値を変えてみましょう。

例
```
for(int i=1; i<50; i++){
    if(i > 20){
     :
    ellipse(250, 250, i*10, i*10);
```

7.3 二重の繰り返し命令

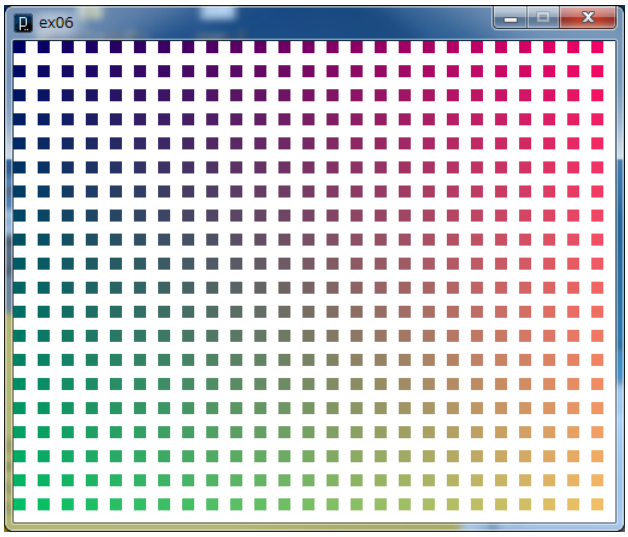
for 文の中に for 文を書いて for 文を繰り返し実行します。

新しいファイルを開きましょう。

以下のプログラムを書き込んで実行しましょう。

```
size(500, 400);

background(255);

noStroke();

for(int y=0; y<20; y++){
  for(int x=0; x<25; x++){
    fill(x * 10, y * 10, 100);
    rect(x * 20, y * 20, 10, 10);
  }
}
```

実行結果は以下のようになります。



完成したら ex06 として保存しましょう。

早く終わった方

for や fill、rect の中の値を変えて実行してみましょう。

例　　rect(x * 20, y * 20, 10, 20);

これでストライプにすることができます。

rect 内の値を調整するとボーダーにしたり、画面全体を塗りつぶすこともできます。

ellipse に変えるとまた異なる結果が現れます。

命令の復習：

ここで使った命令

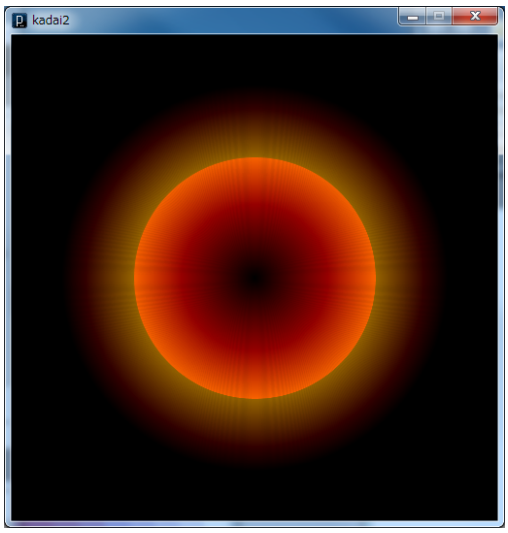
- for(初期設定，繰り返し条件，変数の更新){
  実行する命令
  }
- if(条件){
  実行する命令
  }else {
  実行する命令
  }

製作課題2　きれいな画像を作ってみよう

Processing ではコンピュータならではの絵を描くことができます。
for 文と if 文を使ってきれいな絵を描画してみましょう。
新しいファイルを開いて以下のプログラムを書き込み、実行してみましょう。

```
size(500, 500);
background(0);
smooth();
noFill();
for(int i=0; i<500; i++){
  if(i<250){
    stroke(i, i-150, 0);
    ellipse(250, 250, i, i);
  }else {
    stroke(400-i, 350-i, 0);
    ellipse(250, 250, i, i);
  }
}
```

実行すると以下のようになります。



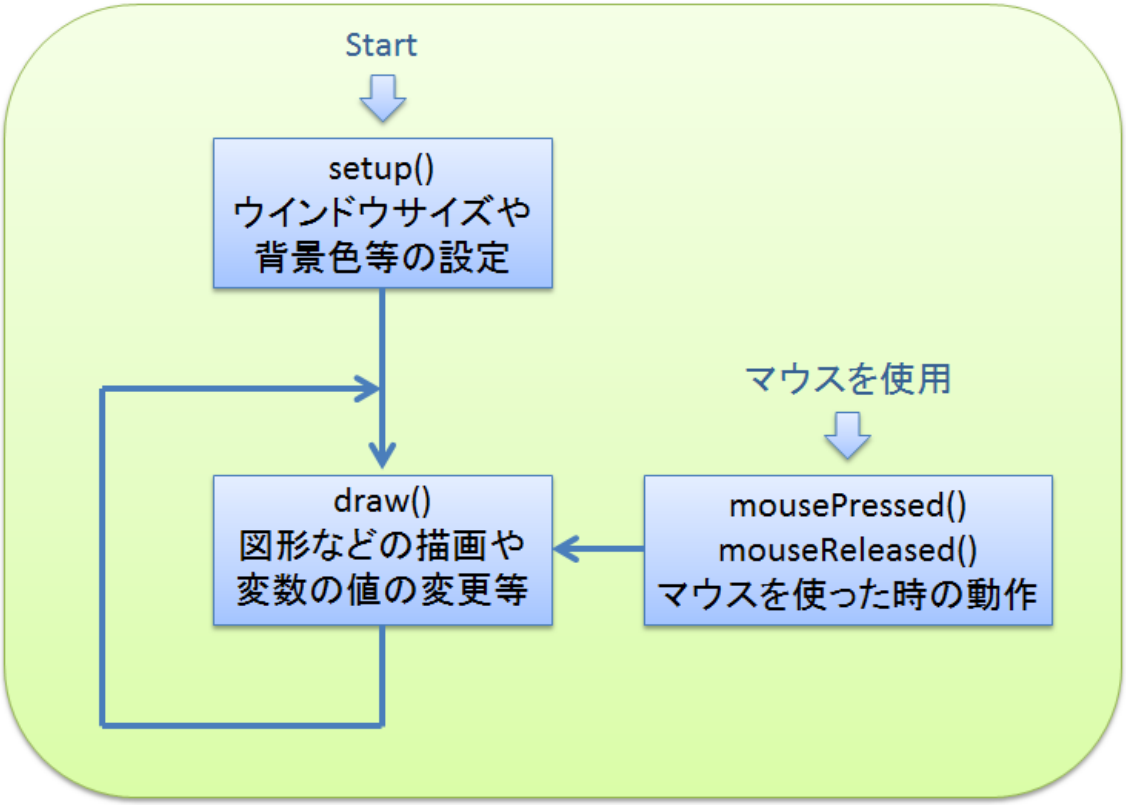
完成したら名前を kadai2 として保存しましょう。

早く終わった方

色や図形を変更してみましょう。
if の条件の数値や stroke の数値、ellipse の数値を変えるだけでも異なる画像ができます。

8. アニメーションの作成

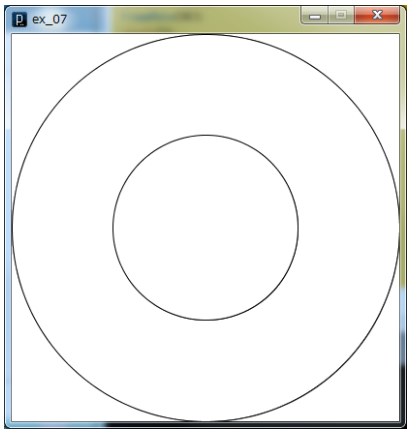
void setup(){}と void draw(){}の{}の中にそれぞれ初期設定(ウインドウサイズ、背景色等)と、描画するものを書くことで簡単にアニメーションを作成することができます。最初にsetup()内が実行され、以後は draw()内が繰り返し実行されます。



新しいファイルを開いて、以下のプログラムを書き込み実行してみましょう。

```
int range;

// setup()内をまず実行する
void setup(){
  size(400, 400);    // ウインドウサイズの設定
  background(255);   // 背景色を白に設定
  frameRate(30);     // 一秒間に何回 draw を実行するか設定
  smooth();          // 線を滑らかにする
  range = 1;         // 変数 range に 1 を入れる
}

// setup が実行された後、draw を frameRate();にしたがって繰り返し実行する
void draw(){
  if(range>400){       // range が 400 より大きくなった場合
    range = 1;         // range に 1 を入れる
  }
  ellipse(200, 200, range, range);     // ウインドウの中心を中心とする円を描画
  range = range +1;                    // range に 1 を加える
}
```

実行すると広がる円のアニメーションが表示されます。



完成したら ex08 として保存しましょう。

早く終わった方

色の変更や、書き出しの位置などを変えて実行してみましょう。

例　`rect(0, 0, range, range);`

次第に色が変化するような命令を入れると面白いと思います。

9. マウスを使ってみよう

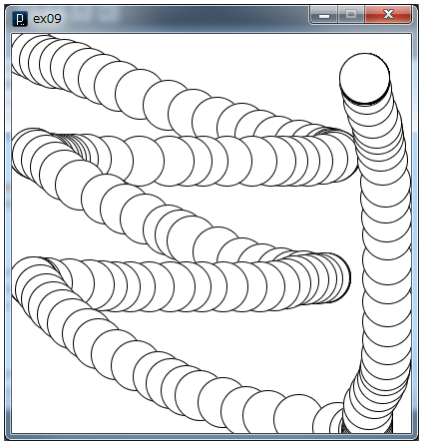
9.1 マウスの位置

マウスポインタのある位置を描画に使用します。

新しいファイルを開いて、以下のプログラムを書き込んで実行してみましょう。

```
void setup(){
  size(400, 400);
  background(255);
  smooth();
  frameRate(30);
}

void draw(){
  ellipse(mouseX, mouseY, 50, 50);   // mouseX はマウスの矢印が示す場所の x 座標
                                     // mouseY はマウスの矢印が示す場所の y 座標
}
```

実行してウインドウ内でマウスを動かすと、その場所に円が描画されます。

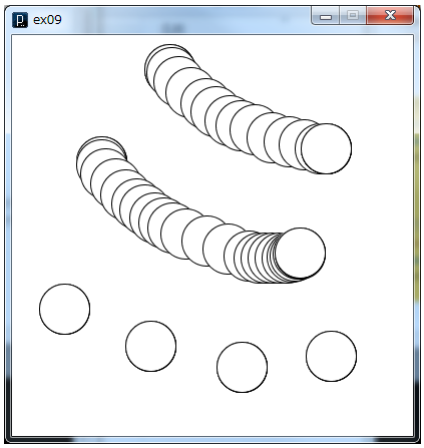


9.2 マウスクリック

以下のようにプログラムの網掛け、太字の部分を書き加えて実行してみましょう。

```
int write;        // write の数値が 1 か 0 かでマウスのボタンが押されているか判断する

void setup(){
  size(400, 400);
  background(255);
  smooth();
  frameRate(30);
  write = 0;       // write を 0 にしておく
}

void draw(){
  if(write==1){           // write が 1 の場合
    ellipse(mouseX, mouseY, 50, 50);  // mouseX はマウスの矢印が示す場所の x 座標
                                      // mouseY はマウスの矢印が示す場所の y 座標
  }
}

// マウスのボタンが押されたときに実行する
void mousePressed(){
  if(mouseButton == LEFT){     // 左ボタンであれば
    write = 1;              // write を 1 にする
  }
}

// マウスのボタンが離されたときに実行する
void mouseReleased(){
  if(mouseButton == LEFT){     // 左ボタンであったならば
    write = 0;              // write を 0 にする
  }
}
```

実行すると、マウスの左ボタンを押している間だけ円が描画されるようになります。

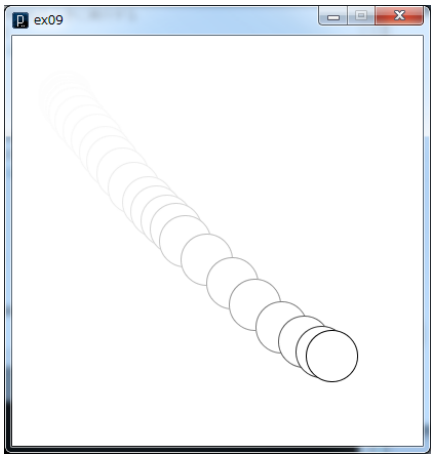


9.3 緩やかな変化

描画された円が次第に消えていくようにしましょう。
以下のプログラムの網掛け、太字の部分を追加して実行しましょう。

```
        :
        :
void draw(){
  fadeToWhite();  // 下に書かれた void fadeToWhite()の{ }の中を実行する
  stroke(1);     // fadeToWhite で枠線の設定が変更されたので再設定
  fill(255);     // fadeToWhite で塗りつぶしの設定が変更されたので再設定
  if(write==1){           // write が true だった場合
    ellipse(mouseX, mouseY, 50, 50);  // mouseX はマウスの矢印が示す場所の x 座標
                                      // mouseY はマウスの矢印が示す場所の y 座標
  }
}
// 白い透過色で画面全体を塗りつぶすことにより次第に消えていくようにする
void fadeToWhite(){
  noStroke();
  fill(255, 100);
  rect(0, 0, 500, 500);
}
        :
        :
```

実行すると以下のようになります。



完成したら名前を ex09 として保存しましょう。

早く終わった方

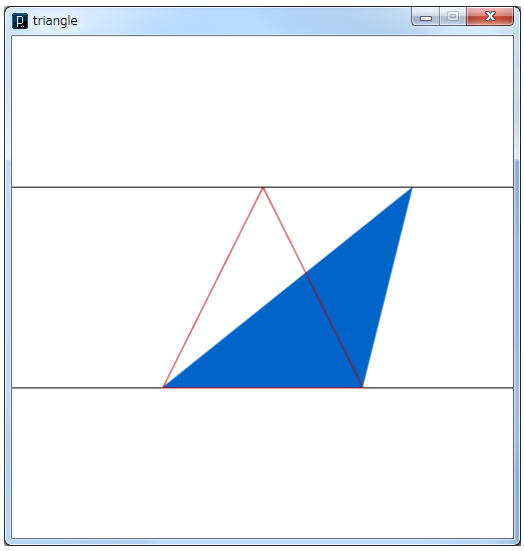
図形を変更したり色を付けたりしてみましょう。複数の図形を描くこともできます。

選択課題

どれかを選択してはじめてください。
早く終われば他のものも作ってみてください。

1 三角形の面積　　難易度 ★☆☆

三角形の頂点をマウスで平行線上を移動させるプログラムです。



プログラム

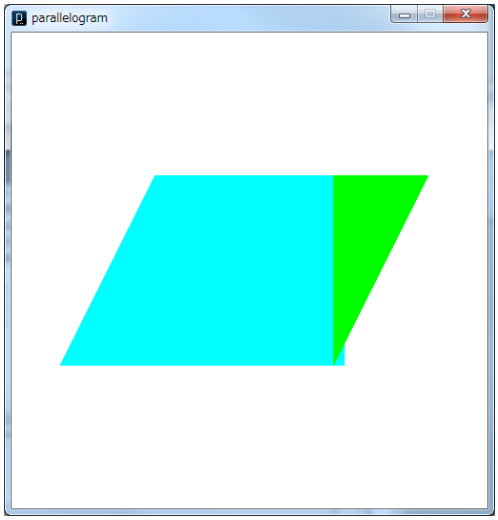
```
int clicked;     // マウスのボタンが押されているか判定するために使用する
int x1, y1;      // 三角形の頂点 1 の座標
int x2, y2;      // 三角形の頂点 2 の座標
int x3, y3;      // 三角形の頂点 3 の座標




void setup(){
  size(500, 500);
  background(255);
```

```
  smooth();
  frameRate(30);
  x1 = 250;    // 各頂点の座標の設定
  y1 = 150;
  x2 = 150;
  y2 = 350;
  x3 = 350;
  y3 = 350;
}

void draw(){
  background(255);       // 画面を更新するために全体を城で塗りつぶす
  stroke(0);
  line(0, 150, 500, 150);       // 平行線の描画
  line(0, 350, 500, 350);

  if(clicked == 1){       // 頂点１の座標がクリックされている間は
    x1 = mouseX;          // マウスポインタのｘ座標を頂点１のｘ座標とする
  }

  if(x1 >500){           // 頂点が画面外に出ないように調整
    x1 = 500;
  }else if(x1 < 0){
    x1 = 0;
  }
  noStroke();
  fill(0, 100, 200);
  triangle(x1, y1, x2, y2, x3, y3);       // 三角形の描画

  stroke(200, 0, 0);
  noFill();
  triangle(250, 150, x2, y2, x3, y3);    // 最初の位置での三角形の枠線を描画
}
void mousePressed(){
  if(mouseButton == LEFT){                        // マウスの左ボタンが押された時
    if(mouseX > (x1 - 20) && mouseX < (x1 + 20) &&     // マウスポインタの座標が
```

```
    mouseY > (y1 - 20) && mouseY < (y1 + 20)){          // 頂点 1 の付近だった場合
      clicked = 1;                                      // clicked を 1 にする
    }
  }
}
void mouseReleased(){
  clicked = 0;          // マウスのボタンが離されたら clicked を 0 にする
}
```

2 平行四辺形の面積計算の考え方　　難易度 ★☆☆

クリックすると平行四辺形の右側の部分を緑色の三角形として切り取り左側へ移動させ長方形にするプログラムです。



プログラム

```
int clicked;     // マウスのボタンが押されたか判定するために使用する

int x1, y1;     // 平行四辺形の頂点 1 の座標
int x2, y2;     // 平行四辺形の頂点 2 の座標
int x3, y3;     // 平行四辺形の頂点 3 の座標
int x4, y4;     // 平行四辺形の頂点 4 の座標

int x5, y5;     // 移動する三角形の頂点 1 の座標
int x6, y6;     // 移動する三角形の頂点 2 の座標
int x7, y7;     // 移動する三角形の頂点 3 の座標
```

```
void setup(){
  size(500, 500);
  background(255);
  frameRate(30);
  noStroke();



  x1 = 150;    // 平行四辺形の各頂点の座標の設定
  y1 = 150;
  x2 = 450;
  y2 = 150;
  x3 = 350;
  y3 = 350;
  x4 = 50;
  y4 = 350;
  x5 = x2;    // 移動する三角形のスタート時の各頂点の座標を設定
  y5 = y2;
  x6 = x3;
  y6 = y3;
  x7 = x3;
  y7 = y2;
}



void draw(){
  background(255);

  if(clicked == 0){         // まだクリックがされていないときに実行
    fill(0, 255, 255);
    quad(x1, y1, x2, y2, x3, y3, x4, y4);  // 平行四辺形の描画
  }else if(clicked == 1){  // クリックがされた後に実行
    fill(0, 255, 255);
    quad(x1, y1, x3, y2, x3, y3, x4, y4);  // 平行四辺形から三角形を
```

```
                                        // 切り離した台形を描画
    fill(0, 255, 0);
    triangle(x5, y5, x6, y6, x7, y7);       // 三角形を描画

    if(x6>x4){                              // 長方形になるような位置まで
      x5 = x5 - 3;                          // 各頂点を左に 3 づつ動かす
      x6 = x6 - 3;
      x7 = x7 - 3;
    }
  }
}

void mousePressed(){
  if(mouseButton == LEFT){     // マウスの左ボタンが押されたとき
    clicked = 1;               // clicked を 1 にする
  }
}
```

3 写真の描画　　難易度 ★☆☆

画像を読み込んでサイズや色を変更して描画するプログラムです。

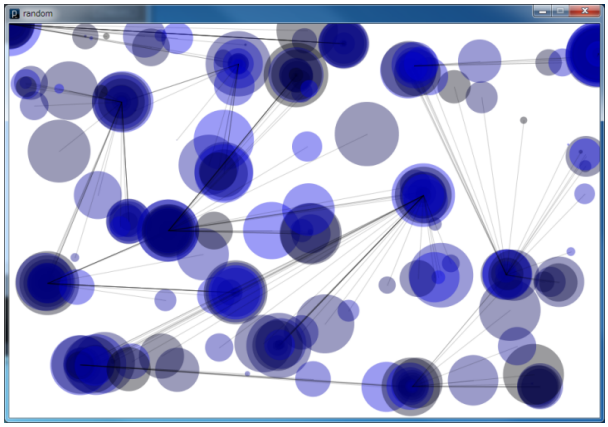
プログラム

```
size(500, 500);
```

```
background(255);

PImage img;       // 画像ファイル読み込みのための準備

// 画像ファイルの読み込み
img = loadImage("sakura.jpg");     // loadImage("描画する画像のファイル名");

// 画像ファイルの描画
image(img, 0, 0);  // imgae(描画する画像, 描画する x 座標, 描画する y 座標);

// 画像ファイルのサイズ指定描画
image(img, 10, 10, 200, 200);
  // image(描画する画像, 描画する x 座標, 描画する y 座標, 横の大きさ, 縦の大きさ);

// 画像の色の設定
tint(50, 80, 80);     // tint(R, G, B);
image(img, 150, 150, 200, 200);

tint(15, 80, 80, 200);     // tint(R, G, B, 不透明度);
image(img, 290, 290, 200, 200);
```

4 ランダムを使ったアニメーション　　難易度 ★☆☆

マウスの動きとクリックを使ってウインドウに絵を描画していくプログラムです。



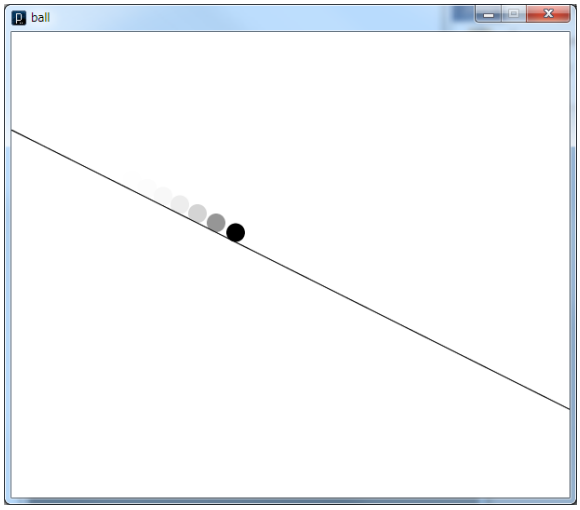
プログラム

```
int x, y;              // クリックされた点の座標を入れる変数
float range;           // 円の半径を入れる変数
```

```
void setup(){
  size(900, 600);
  background(255);
  frameRate(15);
  smooth();
  x = 0;          // 最初の円の位置座標を設定
  y = 0;
}
void draw(){
  stroke(0, 50);
  line(x, y, mouseX, mouseY);
                    // 前にクリックされた点と現在のマウスポインタの位置を線で結ぶ
  noStroke();
  fill(0, 0, random(256), 100);
                           // 塗りつぶしの色 B を 0~255 の間の数字からランダムで選択
  range = random(100);              // 円の半径を 0~99 の間からランダムで選択
  ellipse(mouseX, mouseY, range, range);  // 現在のマウスポインタの位置に円を描画
}
void mousePressed(){
  if(mouseButton == LEFT){      // マウスの左ボタンが押されると
    x = mouseX;                 // その座標を変数に入れる
    y = mouseY;
  }
  if(mouseButton == RIGHT){     // マウスの右ボタンが押されると
    background(255);            // キャンパスを白で塗りつぶす
  }
}
```

5 転がるボール　　難易度 ★★☆

ボールが斜面を下っていく様子を表したプログラムです。



プログラム

```
// int は整数しか使えないが float は小数点以下の数値を使うことができる
float x, y;           // ボールの位置の座標
float spx, spy;       // ボールの横方向の速度と縦方向の速度
float acx, acy;       // ボールの横方向の加速度と縦方向の加速度

void setup(){
  size(600, 500);
  background(255);
  smooth();
  frameRate(15);
  x = 10;             // ボールのスタート位置の設定
  y = 100;
  spx = spy = 0;      // ボールの初速度の設定
  acx = 1.0;          // ボールの横方向の加速度の設定
  acy = 0.5;          // ボールの縦方向の加速度の設定
}




void draw(){
  fadeToWhite();      // 残像が残るようにする
  spx = spx + acx;    // ボールの速度に加速度を加える
```

```
  spy = spy + acy;
  x = x + spx;       // ボールの位置に速度の値を加える
  y = y + spy;

  stroke(0);
  line(0, 105, 600, 405);     // ボールが転がる斜面を描画
  noStroke();
  fill(0);
  ellipse(x, y, 20, 20);     // ボールを描画
}
void fadeToWhite(){          // 次第に白く塗りつぶしていく命令
  noStroke();
  fill(255, 150);
  rect(0, 0, 600, 500);
}
```

6 バウンドするボール　　難易度 ★★☆

落下するボールがバウンドするプログラムです。

プログラム

```
float GRAVITY = 1;          // 重力による加速度の設定
float FRICTION = 0.6;       // 跳ね返りの際の速度の減衰率
```

```
float x, y;              // ボールの位置座標
float spx, spy;          // ボールの速度
int radius = 10;         // ボールの半径

void setup(){
  size(200, 200);
  background(255);
  frameRate(20);

  x = 100;           // ボールの初期位置を設定
  y = 10;

  spx = random(-5, 5);     // ボールの横方向の初速度を -5～5 の内からランダムで設定
  spy = 0;                 // ボールの縦方向の速度を設定
}



void draw(){
  fadeToWhite();         // 残像を残す

  spy = spy + GRAVITY;     // 下方向に重力を加える
  x = x + spx;             // 速度の値を位置座標に加える
  y = y + spy;

  bounce();    // バウンドの動作を行う

  noStroke();
  fill(0);
  ellipse(x, y, radius*2, radius*2);  // ボールの描画

}
void bounce(){      // バウンドの命令
  float bounceMinX = radius;
  float bounceMaxX = 200 - radius;
```

```
  float bounceMinY = radius;
  float bounceMaxY = 200 - radius;

  // 横方向の跳ね返り
  if(x < bounceMinX || x > bounceMaxX){ // 横の壁にぶつかったとき
    spx = -spx * FRICTION;              // 減衰率をかけて移動方向を反対にする

    if(abs(spx) < 1) spx = 0;           // 速度が 1 以下になれば動きを止める

    if(x < bounceMinX) x = bounceMinX - (x - bounceMinX);
                                                  // 画面外にボールが出ないように調整
    if(x > bounceMaxX) x = bounceMaxX - (x - bounceMaxX);
  }
  // 縦方向の跳ね返り
  if(y < bounceMinY || y > bounceMaxY){  // 上下の壁にぶつかったとき
    spy = -spy * FRICTION;              // 減衰率をかけて移動方向を反対にする

    if(abs(spy) < 1){                   // 速度が 1 以下になれば動きを止める
      spy = 0;
      GRAVITY = 0;
    }

    if(y < bounceMinY) y = bounceMinY - (y - bounceMinY);
                                                  // 画面外にボールが出ないように調整
    if(y > bounceMaxY) y = bounceMaxY - (y - bounceMaxY);
  }
}

void fadeToWhite(){    // 次第に白く塗りつぶしていく命令
  noStroke();
  fill(255, 150);
  rect(0, 0, 200, 200);
}
```

7 黒板　　難易度 ★★☆

自由に描画することのできる黒板のプログラムです。

プログラム

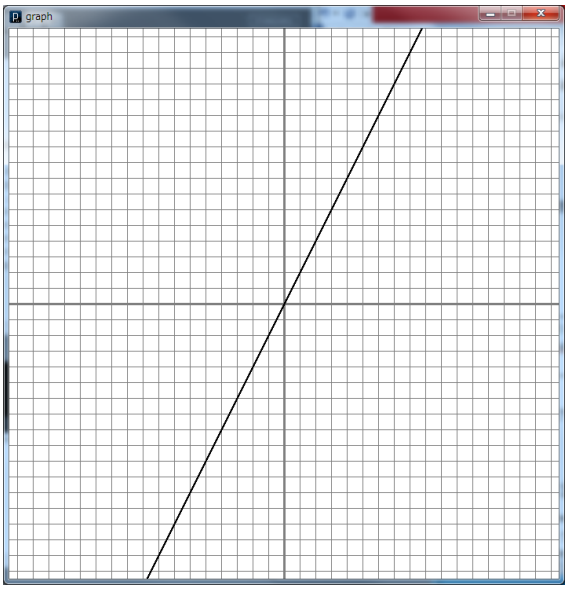
```
int r = 5;              // 線の太さの設定
color nowcolor;         // 色用の変数を設定
color mc[] = new color[5];  // 変数 mc を mc[0] ~ mc[4]までの 5 つ用意する（配列）
int mx = 750, my = 10;  // パレットの枠の位置座標の設定
int ms = 40;            // パレットの枠のサイズの設定

int isdrag = 0;     // マウスのボタンが押されているかの確認に使用する
int bx, by;         // 描画をする座標

void setup(){
  size(800, 600);
  background(0, 60, 20);

  mc[0] = color(250, 250, 250);     // 白のチョークの設定
  mc[1] = color(200, 0, 0);         // 赤のチョークの設定
  mc[2] = color(250, 200, 0);       // 黄のチョークの設定
  mc[3] = color(0, 100, 200);       // 青のチョークの設定
  mc[4] = color(0, 60, 20);         // 黒板の色の設定 （黒板消し）
  nowcolor = mc[0];                 // 現在選択している色の設定
}

void draw(){
  if(isdrag==1){                // マウスのボタンが押されているとき
    stroke(nowcolor);           // 選択されている色の設定
```

```
    strokeWeight(r);              // 線の太さの設定
    if(nowcolor == mc[4])         // 黒板消しの場合
      strokeWeight(r*5);          // 線をより太くする
    line(bx, by, mouseX, mouseY);  // ひとつ前のマウスポインタの座標と
                                  // 現在のマウスポインタの座標の間で線を描画する

    bx = mouseX;                  // 現在のマウスポインタの座標を bx, by に入れる
    by = mouseY;

  }else {                         // マウスのボタンが押されていないとき
    stroke(0, 0, 100);
    for(int i=0; i<5; i++){     // 白、赤、黄、青、黒板消しのパレットを描画する
      fill(mc[i]);
      strokeWeight(1);
      rect(mx, my+(ms+10)*i, ms, ms);
    }
  }
}

void mousePressed(){
  int i = 5;
  // パレットの色が選択されたときにどの色が選ばれたのかを判断する
  if(mouseX >= mx && mouseX <= mx+ms){
    for(i=0; i<5; i++){
      if(mouseY >= my+(ms+10)*i && mouseY <= my+(ms+10)*(i+1))  break;
    }
  }

  if(i<5){      // 選択された色を現在の色に設定する
    nowcolor = mc[i];



  // パレット以外の場所ではマウスポインタの座標を変数に入れ、線の描画の準備
  }else{
    isdrag = 1;
```

```
    bx = mouseX;
    by = mouseY;
  }
}

void mouseReleased(){
  isdrag = 0;    // マウスのボタンが離されたら isdrag を 0 にして描画をやめる
}
```

8 グラフ　　難易度 ★★★

定数や関数を設定してグラフを描画するプログラムです。



プログラム

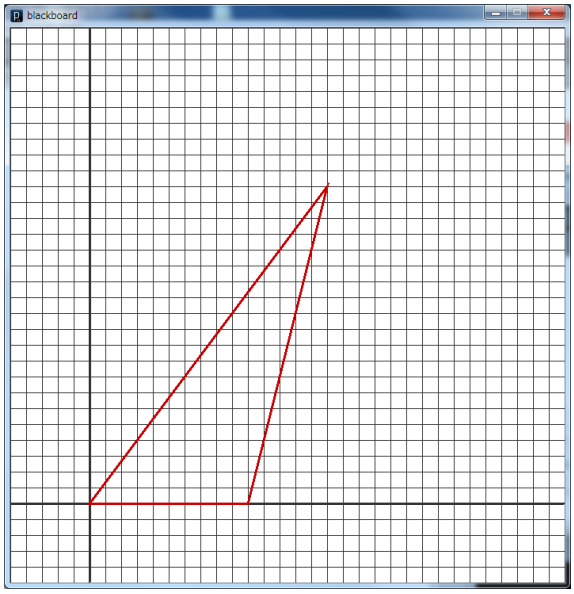
```
int unit;    // グラフのメモリの大きさ
int orgx;    // y 軸の位置
int orgy;    // x 軸の高さ
float a, b;     // 関数の定数
                // 用意している関数 // f(x) = ax + b
                                    // f(x) = a(1/x) + b
                                    // f(x) = ax^2 + b
int x1,  y1;  // ひとつ前の点を入れておく変数
int y;        // y の値を入れる変数
```

```
void setup(){
  size(700, 700);
  background(255);
  smooth();
  unit = 20;       // グラフのメモリの幅を 20 で設定
  orgx = 350;       // y 軸を中央に設定
  orgy = 350;       // x 軸を中央に設定
  drawOrgLine();    // グラフの描画

  a = 2;    // a を 2 に設定
  b = 0;    // b を 0 に設定
}

void draw(){
  stroke(0);
  for(int x = -700; x < 700; x++){
    // GYtoWY はグラフ座標系における y 座標の数値を
    // 描画ウインドウの座標系における y 座標の数値に変換する命令
    // WXtoGX は描画ウインドウの座標系における x 座標の数値を
    // グラフ座標系における x 座標の数値に変換する命令
    y = GYtoWY(WXtoGX(x) * a + b);    // f(x) = ax + b
    //if(x!=0) y = GYtoWY((1/WXtoGX(x)) * a + b);    // f(x) = a(1/x) + b
    //y = GYtoWY(pow(WXtoGX(x), 2) * a + b);  // f(x) = ax^2 + b
    // pow(x,2)は x の 2 乗(x^2)を計算する
    // 他にも sqrt(x)で√x、log(x)で log x の計算ができます

    // ひとつ前の x の値とその時の y の値を変数に入れておく
    if(x > -700)line(x1, y1, x, y);
    x1 = x;
    y1 = y;
  }
}

// グラフの描画
void drawOrgLine(){
  stroke(127);
```

```
  strokeWeight(3);
  line(0, orgy, width-1, orgy);       // x 軸の描画
  line(orgx, 0, orgx, height-1);      // y 軸の描画
  strokeWeight(1);
  for(int y=orgy-unit; y>=0; y-=unit){    // 縦線をメモリの幅ごとに描画する
    line(0, y, width-1, y);
  }
  for(int y=orgy+unit; y<=height-1; y+=unit){  // 縦線をメモリの幅ごとに描画する
    line(0, y, width-1, y);
  }

  for(int x=orgx-unit; x>=0; x-=unit){    // 横線をメモリの幅ごとに描画する
    line(x, 0, x, height-1);
  }
  for(int x=orgx+unit; x<=width-1; x+=unit){   // 横線をメモリの幅ごとに描画する
    line(x, 0, x, height-1);
  }
}

// 描画ウインドウ座標系の x 座標をグラフ座標系の x 座標に変換する
float WXtoGX(int wx){
  // 描画ウインドウ座標系の x 座標と x 軸の距離をグラフメモリの大きさで割る
  float gx = (float)(wx - orgx) / (float)unit;
  return gx;
}
// グラフ座標系の y 座標を描画ウインドウ座標系の y 座標に変換する
int GYtoWY(float gy){
  // グラフ座標系の y 座標にグラフメモリの幅をかけて y 軸との距離を計算する
  int wy = orgy - (int)(gy * unit);
  return wy;
}
```

9 三角形の面積計算　　難易度 ★★★

底辺の長さと上の頂点を指定してグラフ上に三角形を描画し、面積をエディタの下の黒い枠の箇所に表示させるプログラムです。



プログラム

```
int unit;    // グラフのメモリの大きさ
int orgx;    // y 軸の位置
int orgy;    // x 軸の高さ

float x1,y1;       // 三角形の頂点 1 の座標
float x2,y2;       // 三角形の頂点 2 の座標
float x3,y3;       // 三角形の頂点 3 の座標

float d1, d2, d3;    // 各辺の長さを入れる変数

float S;       // 三角形の面積を入れる変数









void setup(){
  size(700, 700);
  background(255);
  noFill();
```

```
  unit = 20;       // グラフのメモリの幅を 20 で設定
  orgx = 100;      // y 軸を左寄りに設定
  orgy = 600;      // x 軸を下寄りに設定
  drawOrgLine();     // グラフの描画
  stroke(250);

  x1 = 0.0;        // 頂点 1 は原点に合わせる
  y1 = 0.0;
  x2 = 10.0;       // 頂点 2 は高さ(y の値)だけを設定する
  y2 = 0.0;
  x3 = 15.0;       // 頂点 3 は好きな座標に設定する
  y3 = 20.0;

  S = (x2 * y3) / 2;  // 底辺×高さで面積を計算する
  print(S);           // 面積の値をエディタの下の黒いスペースに表示する
}

void draw(){
  strokeWeight(3);
  stroke(200, 0, 0);
  // 三角形の各頂点の座標をグラフ座標系から描画ウインドウの座標系の数値に変換し
て三角形を描画
  triangle(GXtoWX(x1), GYtoWY(y1), GXtoWX(x2), GYtoWY(y2), GXtoWX(x3), GYtoWY(y3));
}

// グラフの描画
void drawOrgLine(){
  stroke(127);
  strokeWeight(3);
  line(0, orgy, width-1, orgy);       // x 軸の描画
  line(orgx, 0, orgx, height-1);      // y 軸の描画
  strokeWeight(1);
  for(int y=orgy-unit; y>=0; y-=unit){     // 縦線をメモリの幅ごとに描画する
    line(0, y, width-1, y);
  }
  for(int y=orgy+unit; y<=height-1; y+=unit){  // 縦線をメモリの幅ごとに描画する
```

```
    line(0, y, width-1, y);
  }
  for(int x=orgx-unit; x>=0; x-=unit){    // 横線をメモリの幅ごとに描画する
    line(x, 0, x, height-1);
  }
  for(int x=orgx+unit; x<=width-1; x+=unit){   // 横線をメモリの幅ごとに描画する
    line(x, 0, x, height-1);
  }
}

// グラフ座標系の x 座標を描画ウインドウ座標系の x 座標に変換する
int GXtoWX(float gx){
  // グラフ座標系の x 座標にグラフメモリの幅をかけて x 軸との距離を計算する
  int wx = orgx + (int)(gx * unit);
  return wx;
}

// グラフ座標系の y 座標を描画ウインドウ座標系の y 座標に変換する
int GYtoWY(float gy){
  // グラフ座標系の y 座標にグラフメモリの幅をかけて y 軸との距離を計算する
  int wy = orgy - (int)(gy * unit);
  return wy;
}
```

10 自由製作

このほかに製作してみたいものや、作品のアイデアがあれば自由に作ってみましょう。

自分自身で学習したい方のために

本講習が終わった後，自宅にもどって Processing をさらに勉強してみたいと思う方に，Processing のインストール方法や参考書籍の情報を以下に掲載します．参考にしてください．

□ダウンロードについて

以下の URL

http://processing.org/download/

にアクセスしていただき，ご自身の環境（Mac, Windows など）にあわせて，ダウンロードしてみてください．ダウンロード後，フォルダを解凍すればインストール完了です．今回の講習で利用した Processing より，少し新しいバージョンになっていると思いますが，基本的な操作は同じですので安心してください．

□参考書について

(1) Built with Processing デザイン/アートのためのプログラミング入門 [単行本]
前川 峻志 (著), 田中 孝太郎 (著)
出版社: ビー・エヌ・エヌ新社 (2007/3/26)
ISBN-10: 4861004241
ISBN-13: 978-4861004247

(2) Processing アニメーションプログラミング入門 [単行本（ソフトカバー）]
田中　孝太郎 (著)
技術評論社 (2011/7/15)
ISBN-10: 477414715X
ISBN-13: 978-4774147154

(3) Processing プログラミング入門―Java ベースのオープンソース統合開発環境 [単行本]
田原 淳一郎 (著)
出版社: カットシステム (2010/10)
ISBN-10: 4877832475
ISBN-13: 978-4877832476