

---

# プログラミング演習 (1)

## イントロダクション

---

中村, 高橋  
小林, 橋本

# 何故 Processing を？

---

(Q) 何故 HSP をやめるの？

(Q) 他の大学では Java や C をやっているけど、FMSでは Java とか C をやらないの？

(Q) 何故 Processing をやるの？

# 何故 HSP をやめるのか？

---

- HSP はちょっとしたコードで多くのことを表現可能で楽しいのだけれど、プログラミングの基礎を勉強するには不適切（gotoの多用など）
- プログラムの基礎を学ぶために Processing を勉強する！

(Q) HSP は無駄だったの？

(A) いつか困ったときに帰ってこれる言語！

# 何故 Java や C じゃないの？

---

- プログラミング嫌いを量産したくないから
  - 最初にJavaやCを学んだ学生さんの多くが挫折し、プログラミング自体を嫌いになってしまう
  - 苦手意識を払拭できず、コンピュータ自体も嫌いに
  - 一度嫌いになったら、そんじょそこらの教え方ではプログラミングできるようになりません...
- 例えば、プログラミングでウィンドウを出すこと自体が大変で、心が折れてしまうものです

# JavaやC言語でウィンドウ

```
import java.awt.*;
```

Java

```
public class Window01 {
    public static void main(String
args[]){
        WindowTest windowText=new
WindowTest();
    }
}

class WindowTest{
    WindowTest(){
        Frame frame;
        frame=new Frame("Window Test");
        frame.setSize(240, 240);
        frame.setVisible(true);

        Label label;
        label=new Label("Hello Window");
        frame
    }
}
```

```
#include <windows.h>
#define WNDCLASSNAME TEXT("SampleClass")
LRESULT CALLBACK WindowProc(HWND hWnd, UINT msg,
{
    switch (msg) {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
    }
    return DefWindowProc(hWnd, msg, wp, lp);
}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, PSTR lpCmdLine, int
nCmdShow)
{
    HWND hWnd;
    WNDCLASS wc;
    MSG msg;
    wc.style          = CS_HREDRAW | CS_VREDRAW;
    wc.lpfWndProc    = WindowProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInst;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = WNDCLASSNAME;

    if (!RegisterClass(&wc)) return 0;
    hWnd = CreateWindow(
        WNDCLASSNAME,
        TEXT("SampleWindow"),
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
```

C言語

正直心が折れます

# もちろん

---

- Java や C言語の良いところは沢山
  - 高速に動作する
  - 色々な環境での開発に利用される
    - Windows, Apple, Linux, Web, SmartPhone, ...
  - ライブラリ（サポートしてくれるもの）が豊富
  - 多くの人が開発に利用している
  
- 2年次にJavaは扱う予定

# 何故 Processing か

---

- 視覚的なプログラム作成が容易
- プログラムの基礎を学ぶのに適している
- 最先端の研究でも実を言うと使われている
- Processing は Java の上で動作しているものであり, Java にととても似ている
- Processing を勉強した後, Java を勉強するのはとても簡単!

# 超重要

---

- EP演習とは異なり、ウェブ上のプログラムや他人のプログラムをコピーして使うことは厳禁！
  - プログラミング演習はプログラムが書けるようになることを目的としておりEP演習とは異なります！
  - 写すと時間とお金の無駄です！！
- 他人のプログラムを写しているような行為が発見された場合は、提供した人も、コピーした人も両者をカンニング扱いとします
  - コピーする人は本当に稚拙な書き換えしかしませんので要注意！！
- 「見せない」「写させない」が重要です



# EP演習とP演習

---

- EP演習（≡ 動画制作）
  - プログラミングという行為の体験とプログラム制作におけるオリジナリティを重視
  - 制作行為と制作物が重要であり，模倣は推奨
- P演習（≡ 数学の問題解決）
  - 課題解決のために，どうプログラミングするかを学ぶというもの
  - 課題をどう解決するかが重要であり，他人の解法を見ることはカンニングに相当

# 提供してもらわないように！

---

- みなさんがコピーしてしまうことにより、コピー提供者も迷惑を被ります
- この講義を前提に、秋学期、2年のプログラミング系の講義が設計されているため、逃げたらずっと苦しむだけです。勉強にもなりませんのでしんどいなりに立ち向かいましょう。
  - これまで英語や数学とかを勉強してきたときどうしてます？ どれくらいの時間をかけてきました？

# 提供しないように！

---

- 良かれとしてやりたくなる気持ちはわかります
- 教えるのが面倒でこれみとけって気分になることもあるでしょう
  - でも、同一レポート（課題）の提出があった場合、提供された側が正直に申告しない場合は提供元・提供先の両方が処罰の対象となります
  - そんなしょうもないことで人生を狂わせてはもったいないので、くれぐれも注意してください
- 他の講義でも問題になりつつあります。絶対にレポートはコピーさせないこと！

# 昨年度来た質問

---

- 教えたら類似するのでは？
  - 多少は類似するでしょう。ただ、積み木でお城を作るという課題で同じ形、同じ色の積み木を同じように積むことはあり得るでしょうか？
  - 見せなければ一致しない
- 教えることを禁止するのか？
  - ぜひとも教えて下さい、ただコピーが発生するような行為は慎んでください
- 他人のプログラムを見てはだめなのか？
  - 教えるときは見ざるを得ませんので、その時は見て下さい。ただ、見せて教えないで下さい。

# 進め方

- 講義資料は事前にPDFで配布
  - PDFは必ず予習しておくこと
  - 資料は <http://nkmr.io/lecture/>
- 講義ではまず小テストを実施（20分）
  - 小テストに出す問題はスライドに とマークを付けているものと、前回の基本課題から（小テ **小テスト** で実施します）
- 小テストの後に小テスト＋内容の解説（20～30分）
- 課題を配布
  - 基本課題と発展課題の両方を紙で配布
  - 説明の後、各自課題に取り組む
- 課題を共有フォルダに提出
  - 教員とTAで課題のチェックを実施
  - 基本課題が終わるまでは帰ってはだめ！
  - 発展課題は講義中にはチェックしませんので確実に動作しているかどうかを周りのひとなどと確認しましょう！

# 講義の流れ

---

- 13:30-13:50 小テスト
- 13:50-14:20 解説
- 14:20-14:30 課題提示
- 14:30-16:30 課題を解く + 提出する
- 16:30-17:00 課題解説

# 評価

---

- 小テスト： 35点
  - 講義資料の予習と前回課題の復習で大丈夫です
- 基本課題： 35点
  - 基本課題が終わるまでは帰れません
- 発展課題： 15点
  - 発展課題を時間内に完成させると加点されます
- 最終課題（発表）： 15点
  - **2019年7月??日（？）**にホールで発表会を開きます
  - **19時から**の予定なので、いつもと時間が異なります
  - 成果物＋プレゼン＋ソースコードで評価

# 今回の目標

---

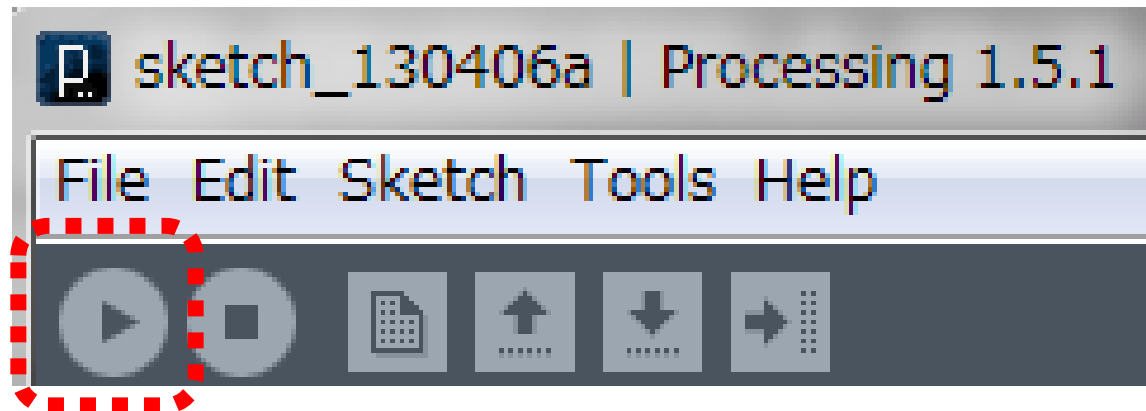
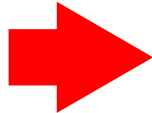
- 課題の提出方法を習得する
- HSPとProcessingの違いを学ぶ
- Processing の基本を学ぶ
- Processing に慣れ親しむ
- Processing で簡単な絵を描く



# Processing とは

- Processing プロジェクトは2001年春に開始
- アート, デザイン界向けにJava拡張として
- プログラムすることをスケッチするという
- 多くのライブラリを導入することで各種の処理が可能に
- プログラムの実行は再生ボタンを押すだけ

再生ボタン



# インストール方法

---

1. Processingを下記のURLよりダウンロード

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

Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below.



3.3.7 (13 March 2018)

[Windows](#) 64-bit

[Windows](#) 32-bit

[Linux](#) 64-bit

[Linux](#) 32-bit

[Linux](#) ARMv6hf

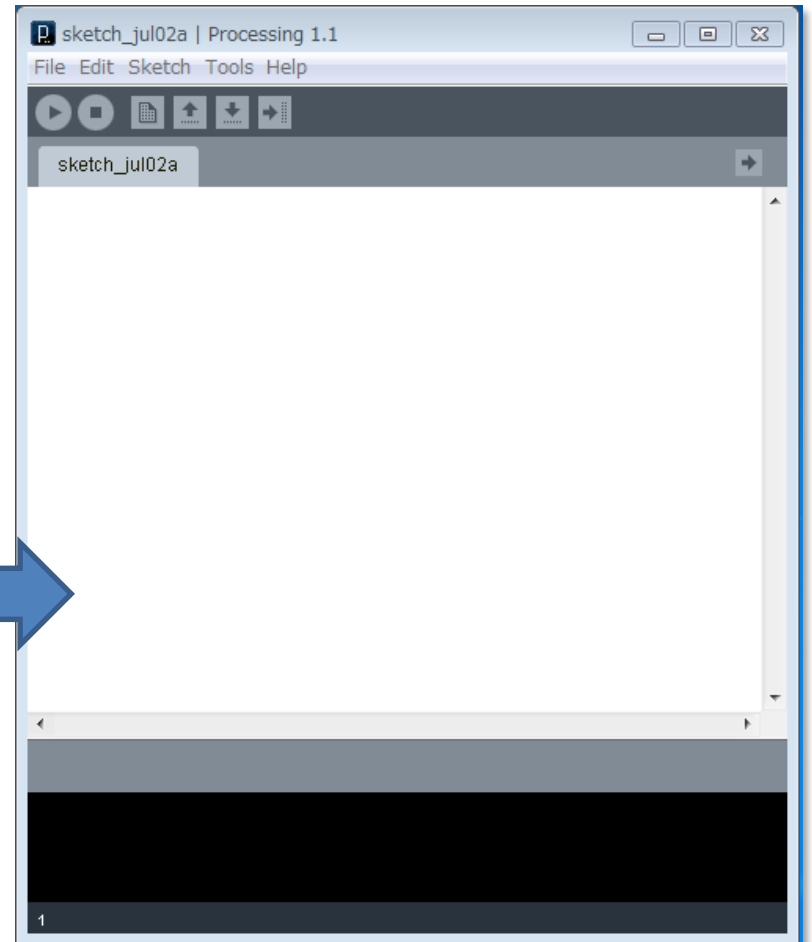
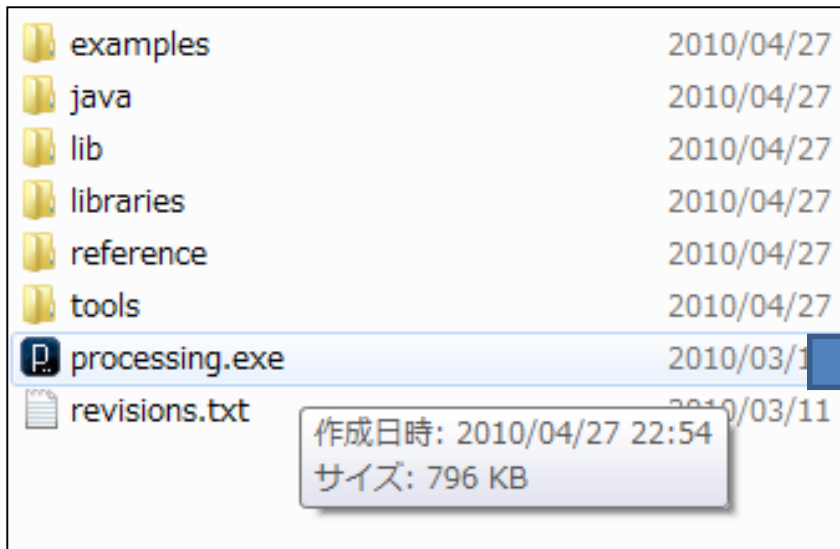
[Mac OS X](#)

---

# インストール方法

2. ダウンロードしたファイルを解凍（展開）
3. Processing.exe を起動（Windowsの場合）

Mac は Processing を実行  
Linux は `% sh processing`



# 想像してみましよう

---

- 画面にウィンドウを作るのってどうする？
- 国旗のようなものはどうやって作る？
- どうやったら沢山の平行な線を描ける？
- キャラクタはどうやって描画する？
- キャラクタを動かすにはどうしたら良い？

# まず最初に

- 下記のコード（プログラム）を入力して，再生ボタン（実行ボタン）を押してみましょう

400x300のウィンドウを作る

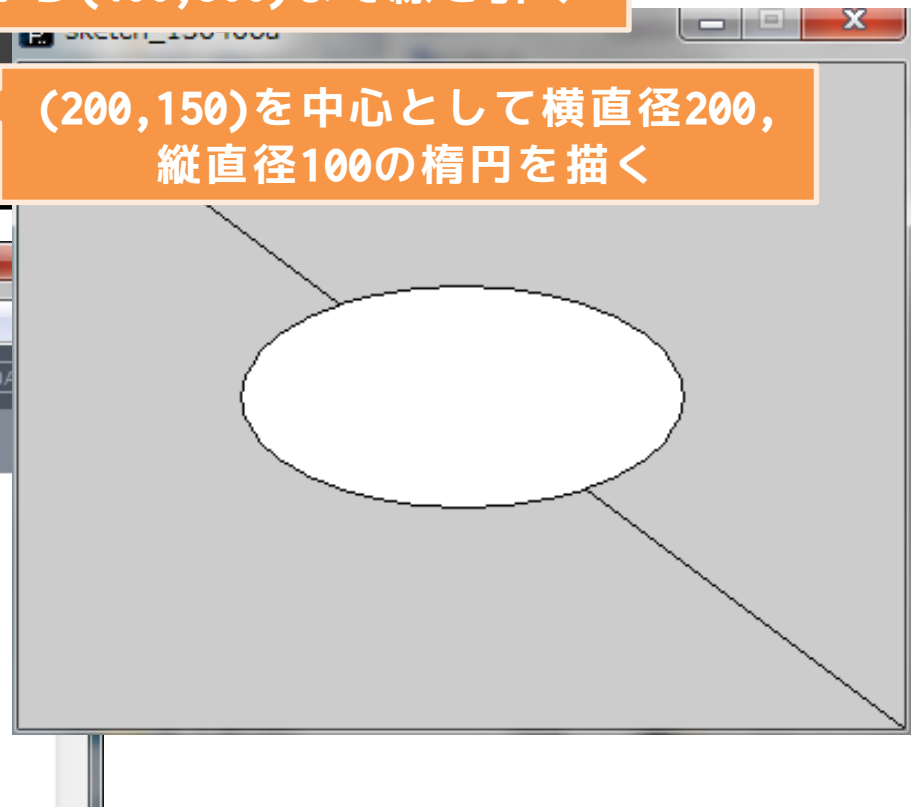
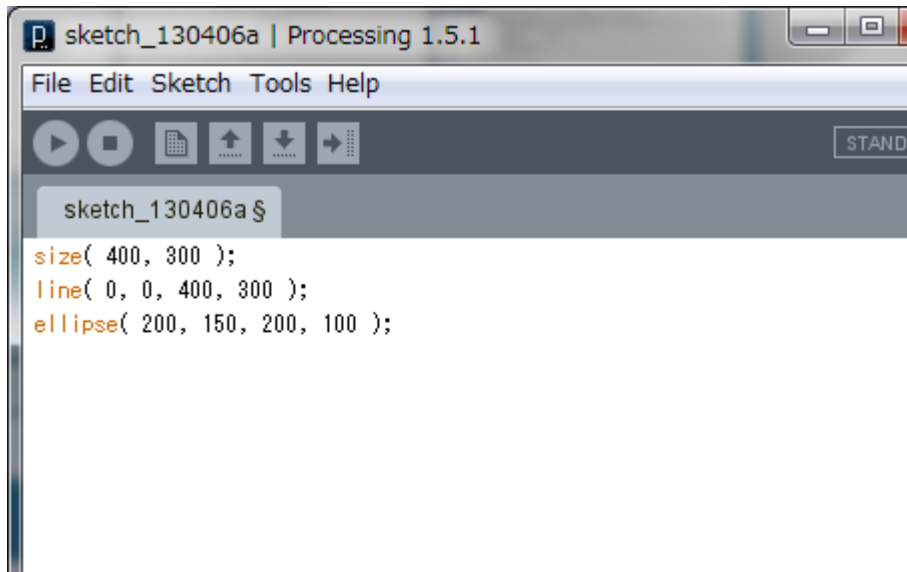
```
size(400, 300);
```

(0,0)から(400,300)まで線を引く

```
line(0, 0, 400, 300);
```

(200,150)を中心として横直径200，  
縦直径100の楕円を描く

```
ellipse(200, 150, 200, 100);
```



# もうひとつ

- 下記のコード（プログラム）を入力して，再生ボタン（実行ボタン）を押してみましょう

```
size(400, 300);
```

400x300のウィンドウを作る

```
background(255, 255, 255);
```

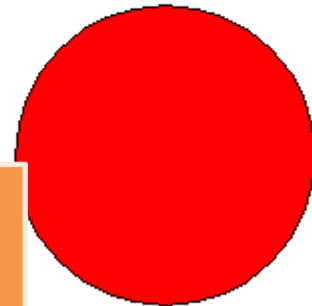
背景色を白色(255,255,255)にする

```
fill(255, 0, 0);
```

塗りつぶしの色を赤色(255,0,0)にする

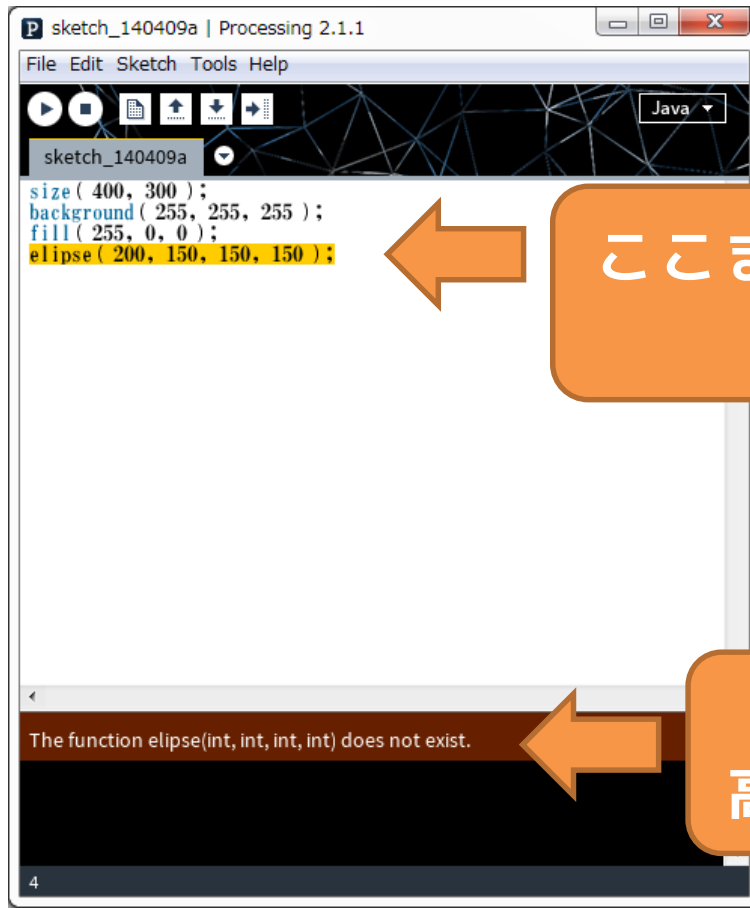
```
ellipse(200, 150, 150, 150);
```

(200,150)を中心として横直径と縦直径150の円を描く  
塗りつぶしの色は最後に指定した赤色



# 動かない??

- 何かプログラムが読み込めないエラーが発生した時, Processingはハイライトしてそこら辺がおかしいということを教えてくれます



ここまたは, ココらへんがおかしい  
という意味

エラーメッセージ  
高校レベルの英語なので読もう

# 動かない??

The function `ellipse(int, int, int, int)` does not exist.  
(訳) `ellipse(int,int,int,int)`という関数はないよ！



**ellipse は ellipse の間違い！**

**注意してみると ellipse は青色に  
ellipse は黒色になっており違う！**



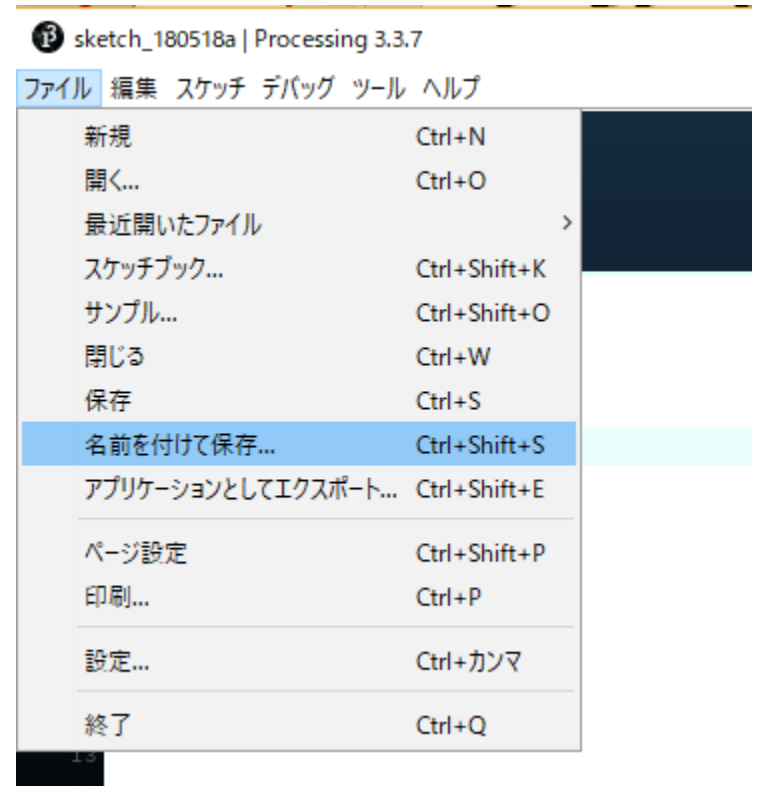
# エラーは最初に注目

---

- エラーメッセージは沢山表示されます
  - 一部おかしいところがあると、そこから他の部分もおかしいと判断されることがよくあります
  - 表示されるエラーメッセージは、最初にどんなメッセージが表示されているか注目しましょう
- ここにエラーがある！と完璧に推定することはコンピュータには難しい
  - その行または、その前後の行におかしなところがないかをチェックしよう！
  - 例えば、セミコロンが抜けている場合に、次の行でエラーが出ます

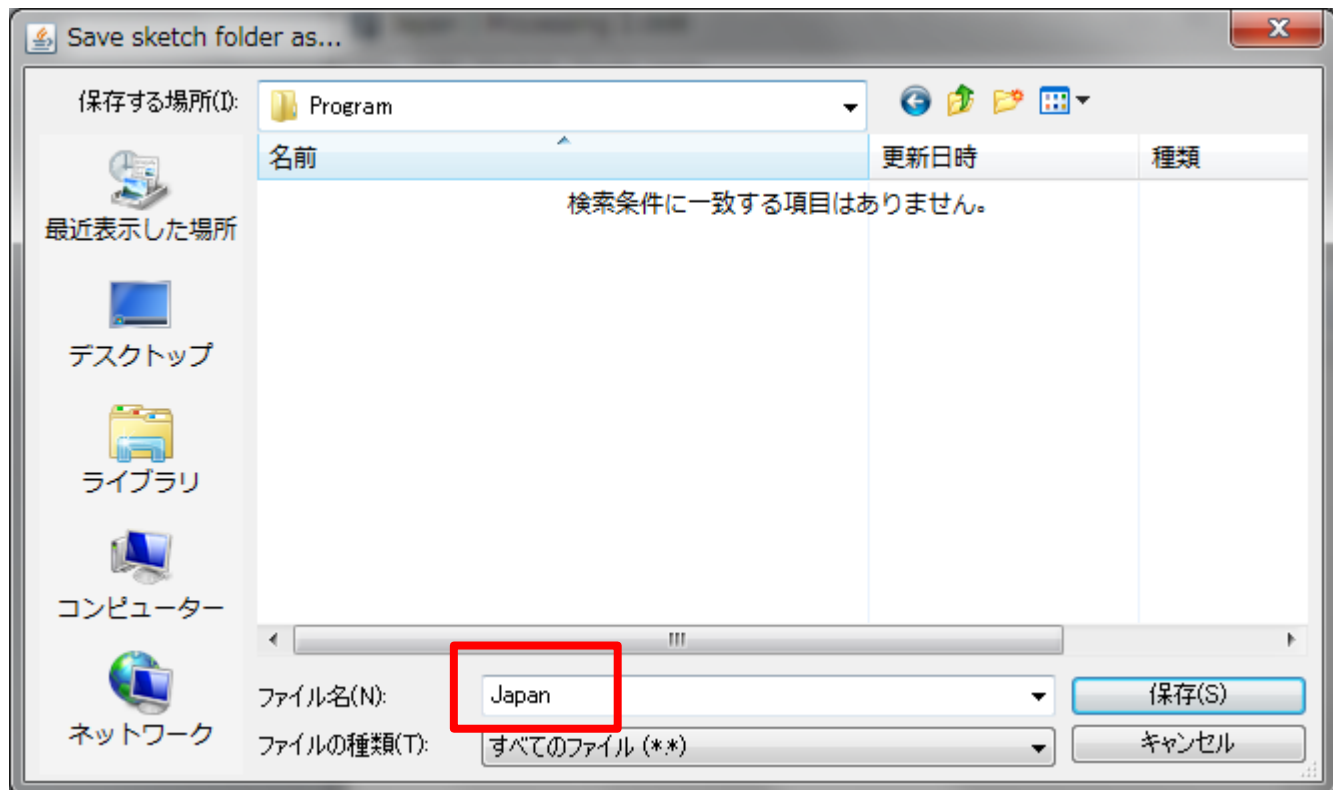
# 保存しましょう

- ファイル → 名前を付けて保存 で保存
  - 名前をつけて保存
  - どんどん保存しましょう
- ファイル → 新規
  - プログラムを新規作成



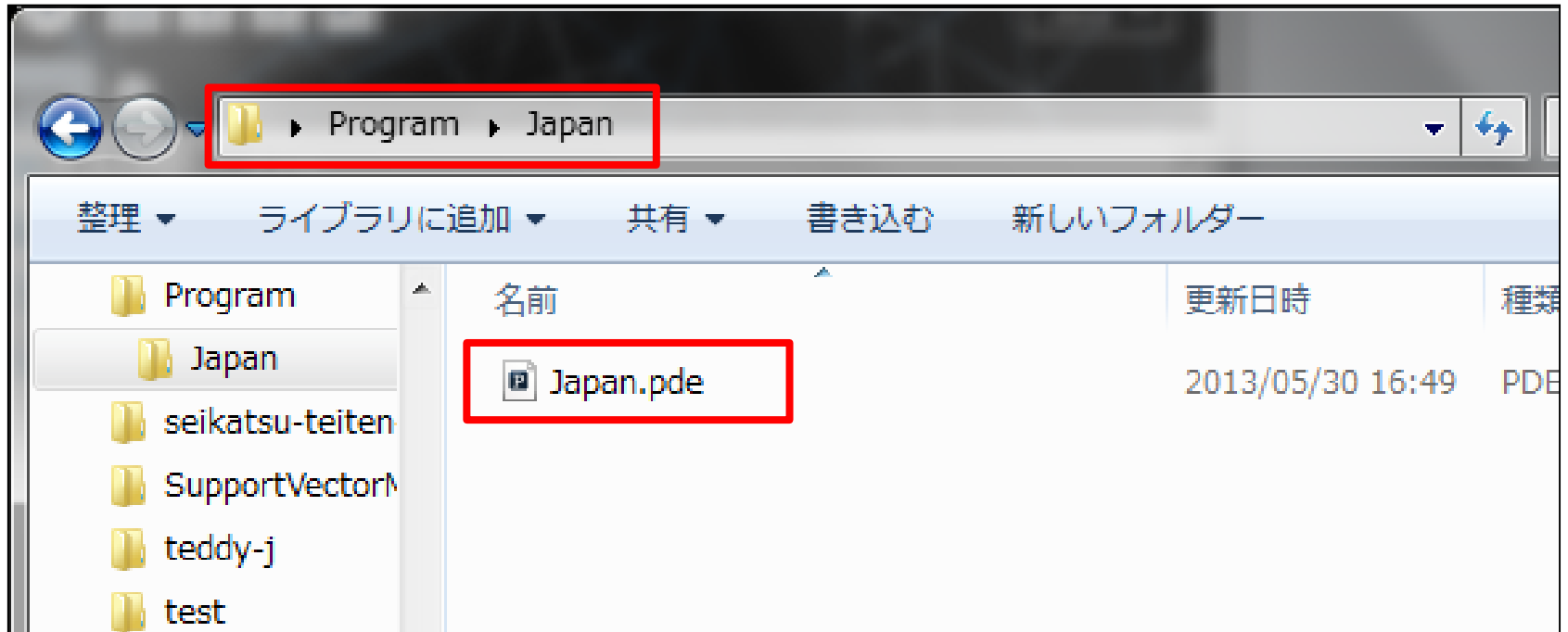
# フォルダ構造

- Processingのプログラムは、フォルダとセット
  - (例) スケッチ (プログラム) を、デスクトップに Program というフォルダを作成し、「Japan」という名前で保存する場合 (WindowsもMacも同じ)



# フォルダ構造

- 「Japan」という名前で保存すると．．．
  - 下図のように，名前が「Japan」というフォルダの下に，「Japan.pde」というプログラムが保存される
  - Japanというフォルダは自動生成される



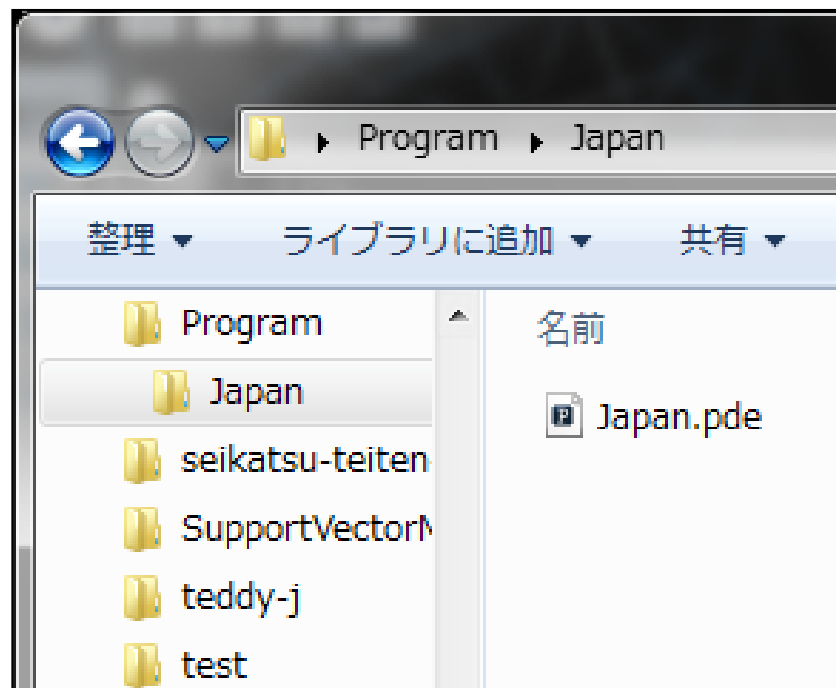
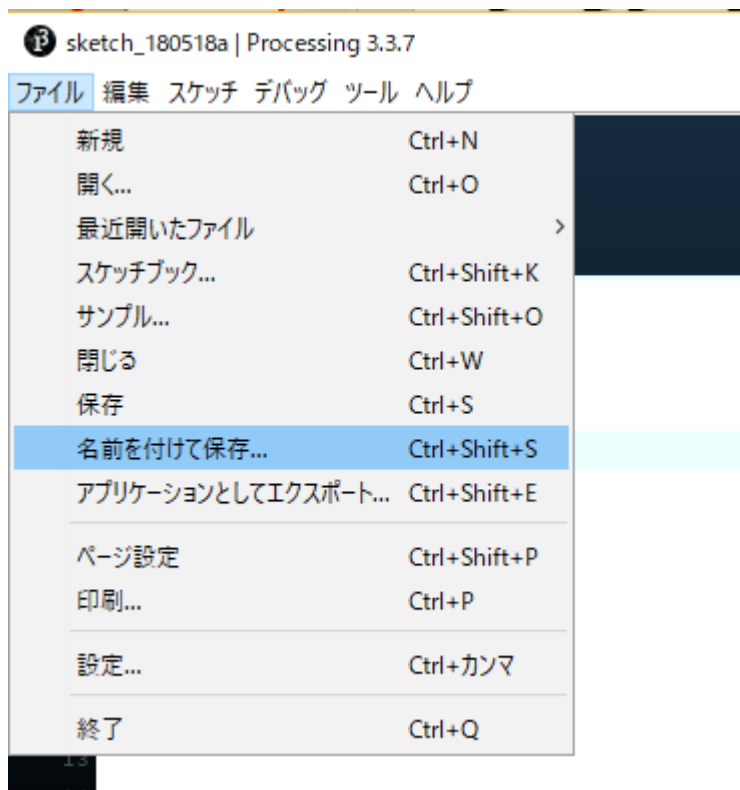
# 注意点

---

- Processingの名前は英数字のみ（ただし、1文字目は英文字のみ）
  - Processingでは、日本語の名前でプログラムを保存することはできません
  - Processingでは、プログラム名に空白や記号は使えません

# 開き方

- メニューの ファイル→ 開く から選んで開く！
- 関連付けを行うとダブルクリックで開けるようになる！（Windows8だと少し苦戦する場合も）



# 基本的な事

```
命令名(命令の詳細, 命令の詳細, ...);
```

- 例: size, background, line, ellipse, ...
- **すべて半角英数字**
  - 日本語はダメ！大文字小文字に注意！
- 命令の詳細は括弧の中に！
  - 複数あるときはカンマで区切る
- **最後はセミコロン！**
- プログラムは上から順に実行される

# キャンバスを設定する

---

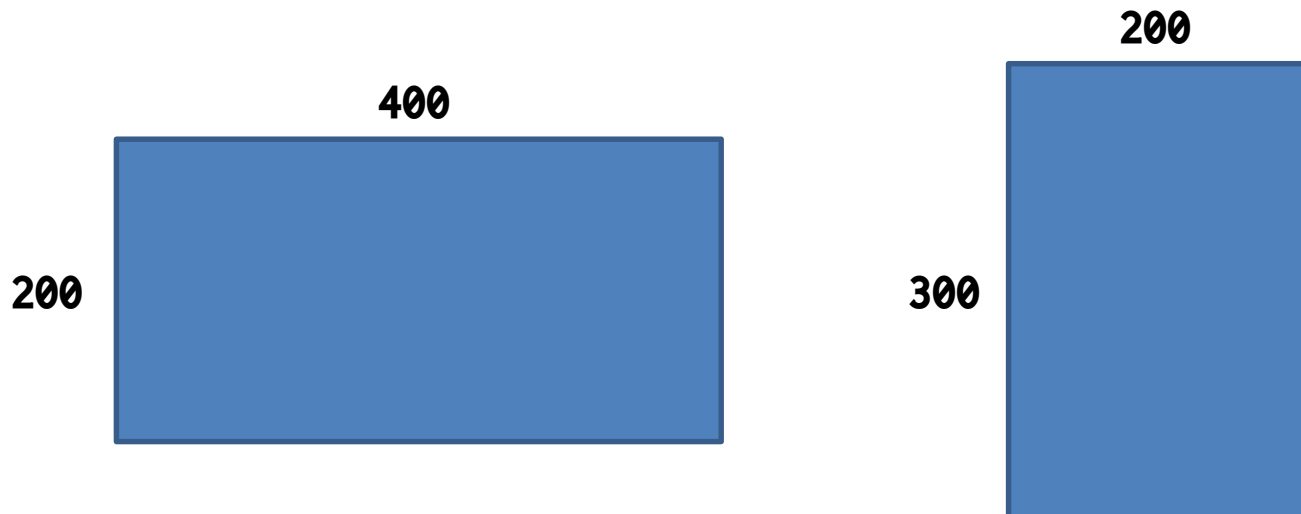
キャンバス（ウィンドウ）のサイズを設定

**size(横幅, 縦幅);**

size(400, 200);      400x200のウィンドウを作る

size(200, 300);      200x300のウィンドウを作る

※ 単位はピクセル（1つの描画単位）





# 点を描く

---

## 点を描画する場所を指定

**point(x座標, y座標);**

point(400, 200);    x=400, y=200に点を描画

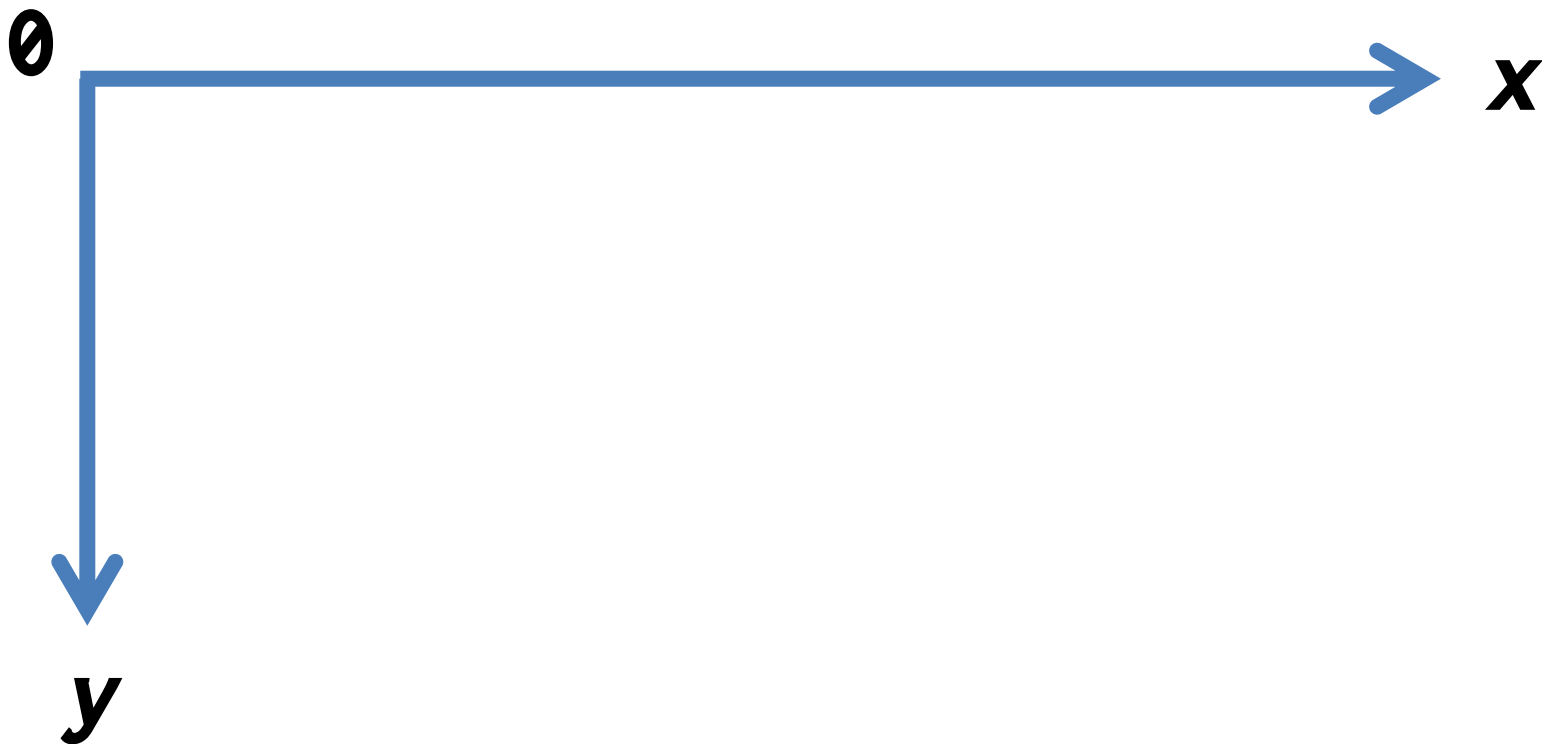
point(200, 300);    x=200, y=300に点を描画

※ 単位はピクセル（1つの描画単位）

# 座標軸

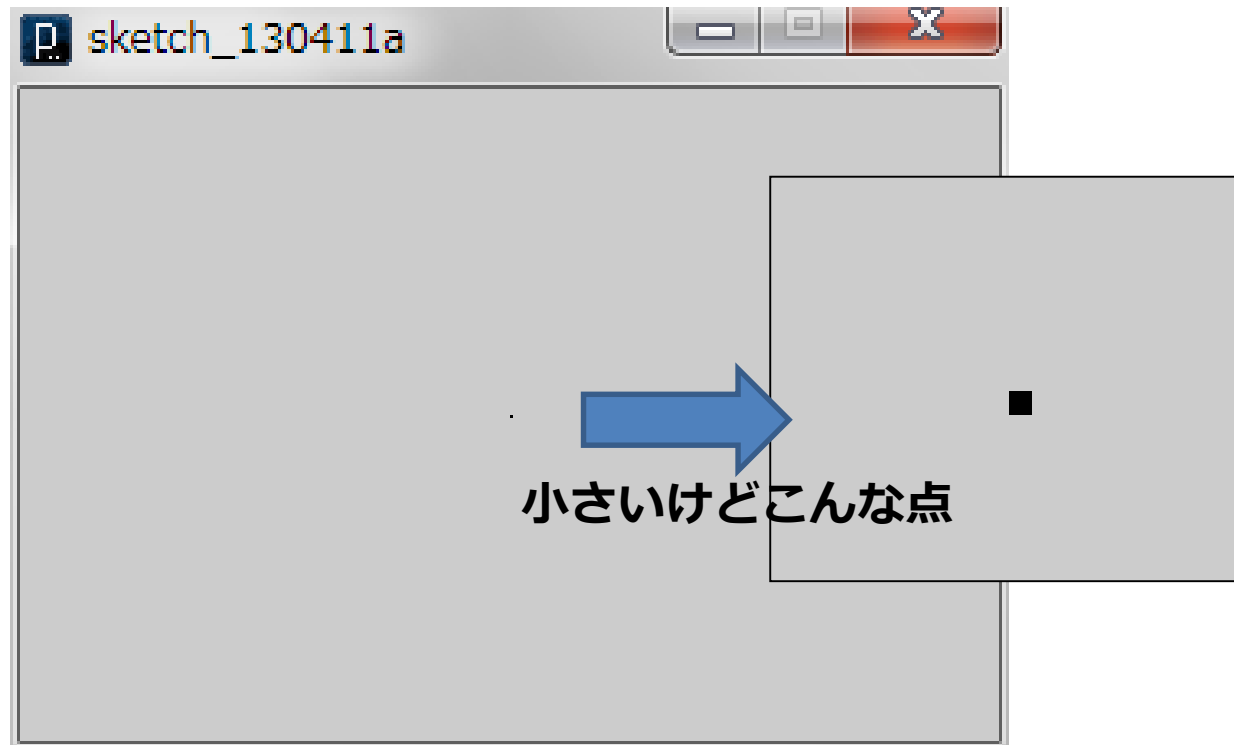
---

- 左上が  $(0,0)$  で、 $x$  が大きくなると右へ、 $y$  が大きくなると下へ（ $y$  が下方方向というのがちょっと慣れないけれど）



# 点を描く

(Q) 横幅300×縦幅200ピクセルのウインドウの中心に点を描画したい。どうする？

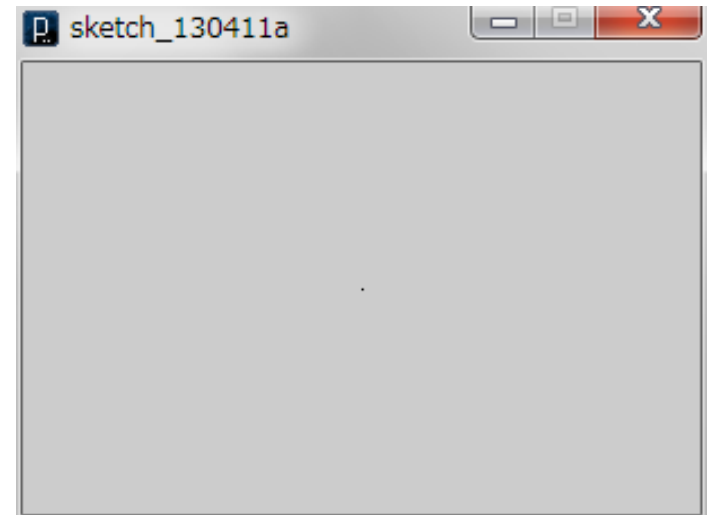


# 点を描く

(A) 横幅300x縦幅200ピクセルのウィンドウの中心に点を描画したい。どうする？

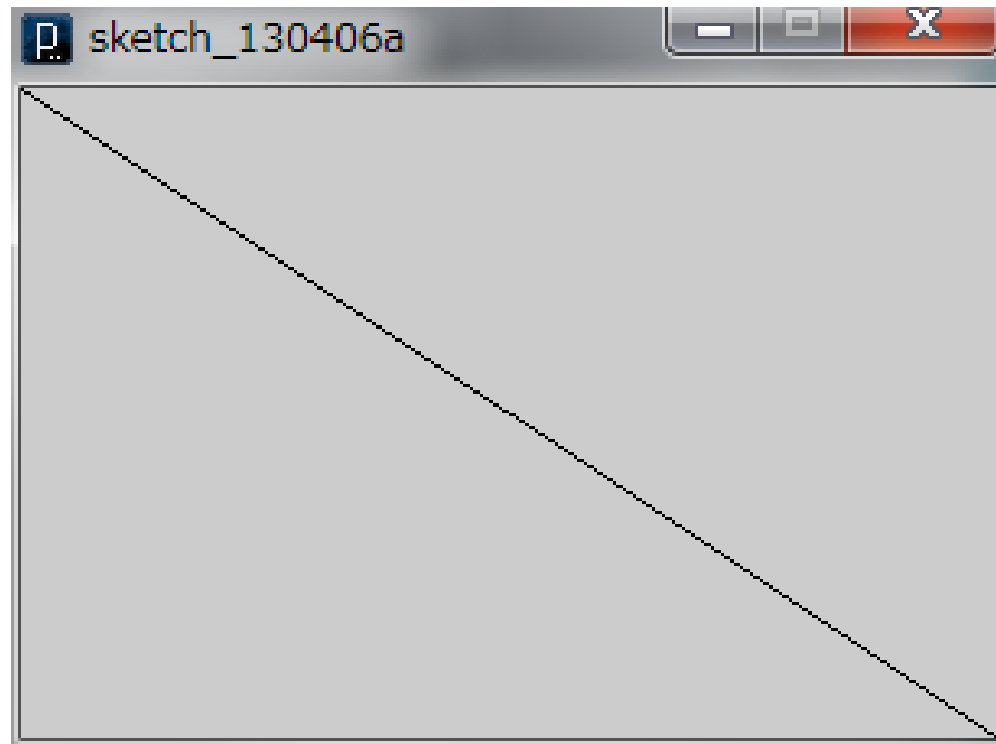
- 300x200のウィンドウは `size(300, 200);`
- 300と200の中心は, `x=150, y=100`
- 点を描画する命令は `point(x座標, y座標);`

```
size(300, 200);  
point(150, 100);
```



# 線を描く

(Q) 横幅300×縦幅200ピクセルのウインドウの中で左上から右下まで線を引きたい



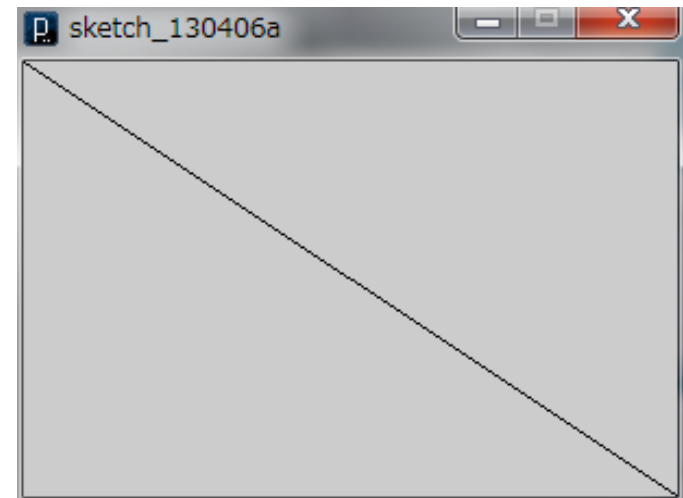
**point** で沢山点を描画するのは大変

# 線を描く

(A) 横幅300x縦幅200ピクセルのウインドウの中で左上から右下まで線を引きたい

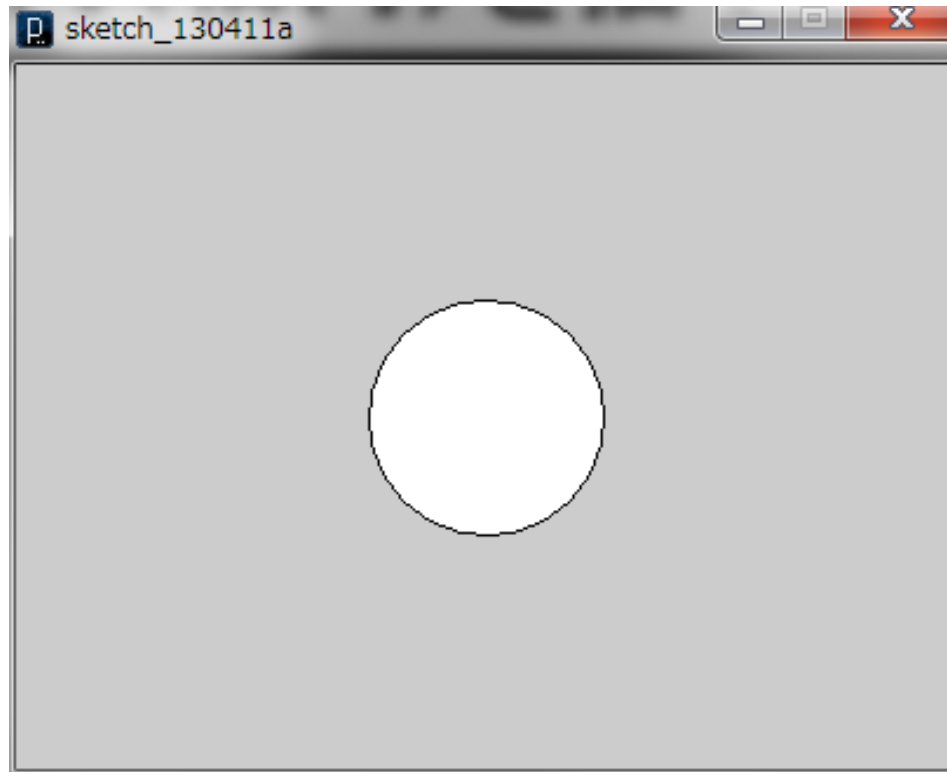
- 300x200のウインドウは `size(300, 200);`
- 左上のxy座標は(0, 0), 右下は(300, 200)
- 線を描画する命令は `line(x1, y1, x2, y2);`

```
size(300, 200);  
line(0, 0, 300, 200);
```



# 円（楕円）を描く

(Q) 横幅400x縦幅300のウィンドウの中央に直径100ピクセルの円を描きたい



**point で円を描画するのは大変**

# 円（楕円）を描く

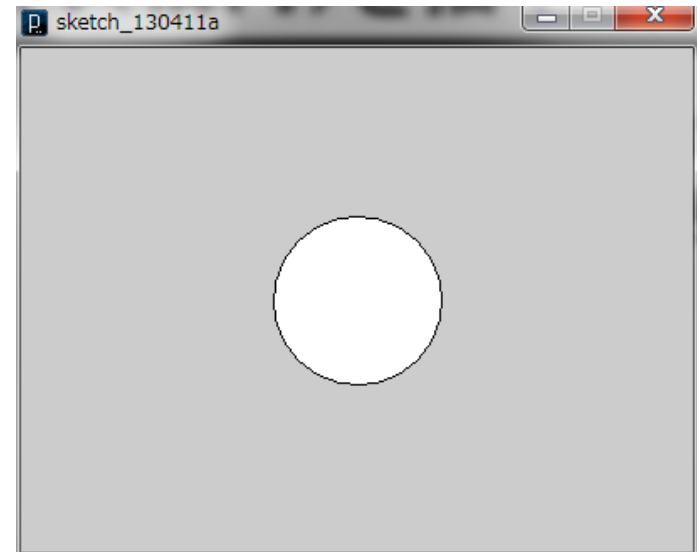
(A) 横幅400x縦幅300のウィンドウの中央に直径100ピクセルの円を描きたい

- 400x300のウィンドウは `size(400, 300);`
- 円の中央のxy座標は (200, 150)
- `ellipse(中心x, 中心y, 縦直径, 横直径);`

```
size(400, 300);  
ellipse(200, 150, 100, 100);
```

v3.5から `circle` という記述もOKに

```
size(400, 300);  
circle(200, 150, 100);
```





# 色々 と 描画する

## 点を描く

```
point(x, y);
```

## 線を描く

```
line(始点X, 始点Y, 終点X, 終点Y);
```

● point(500,300)

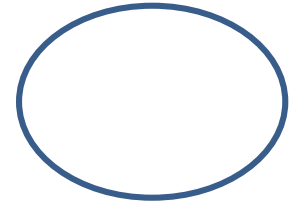
(100,150)

line(100,150,300,350);

(300,350)

## 楕円を描く

```
ellipse(中心X, 中心Y, 横直径, 縦直径);
```



## 円弧を描く (角度はラジアンで与える)

```
arc(中心X, 中心Y, 横直径, 縦直径, 開始角, 終了角);
```

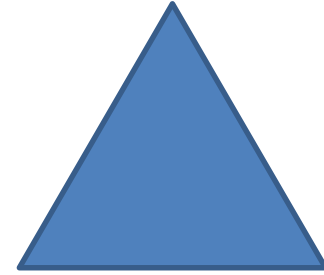


# 色々 と 描画する

---

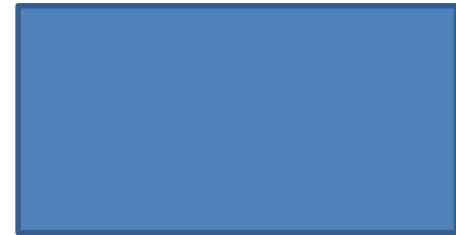
三角形を描く

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



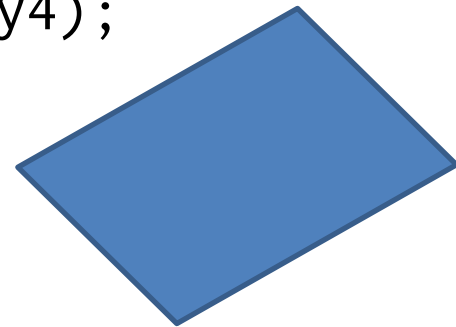
長方形を描く

```
rect(左上X, 左上Y, 横幅, 縦幅);
```



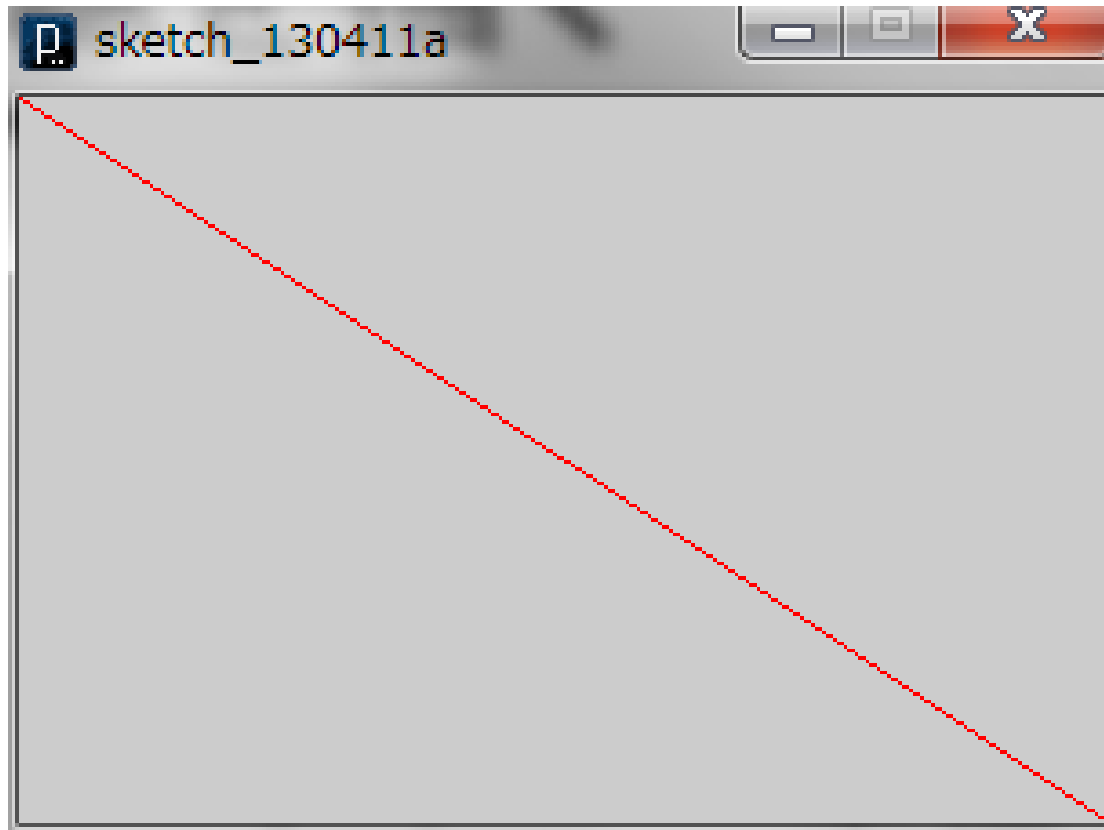
四角形を描く

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



# 色を付けたい

(Q) 300x200のウィンドウで左上から右下まで描画した線を赤色にしたい



# 色を付けたい

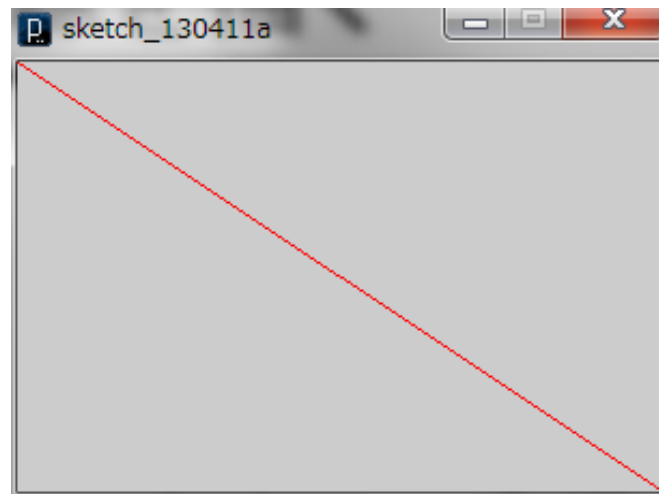
(A) 300x200のウィンドウで左上から右下まで描画した線を赤色にしたい

- 左上から右下までの線は`line(0, 0, 300, 200);`
- 線の色を変える命令は

`stroke(赤の強さ, 緑の強さ, 青の強さ);`

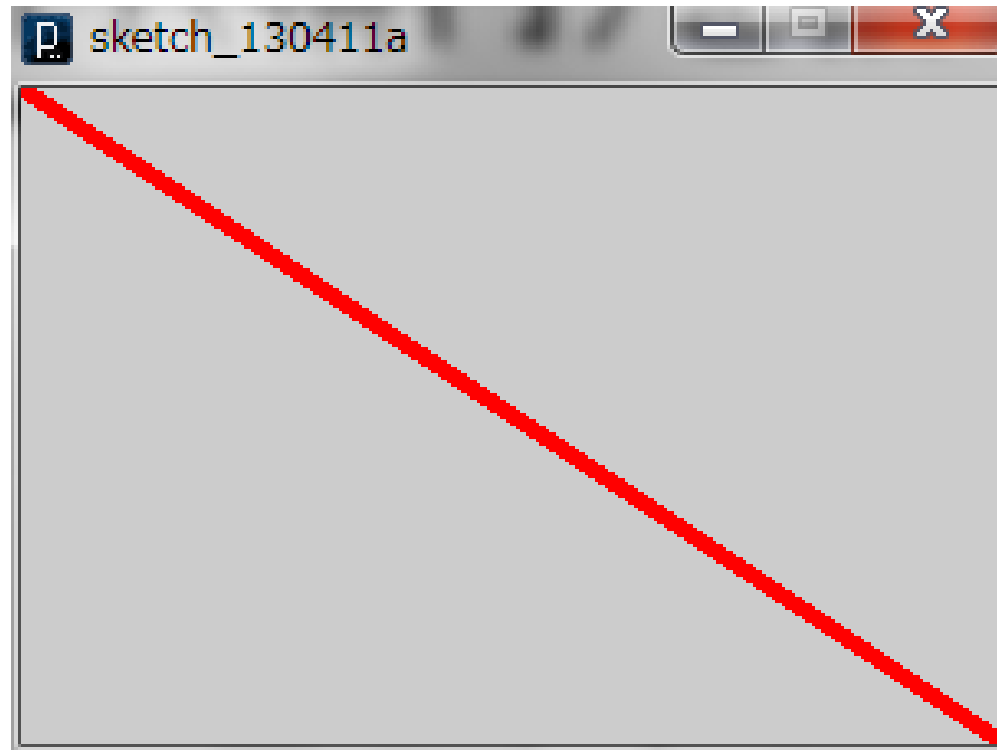
強さは 0 から 255 までの値

```
size(300, 200);  
stroke(255, 0, 0);  
line(0, 0, 300, 200);
```



# 太くしたい

(Q) 300x200のウィンドウで左上から右下まで描画した赤色の線を太くしたい



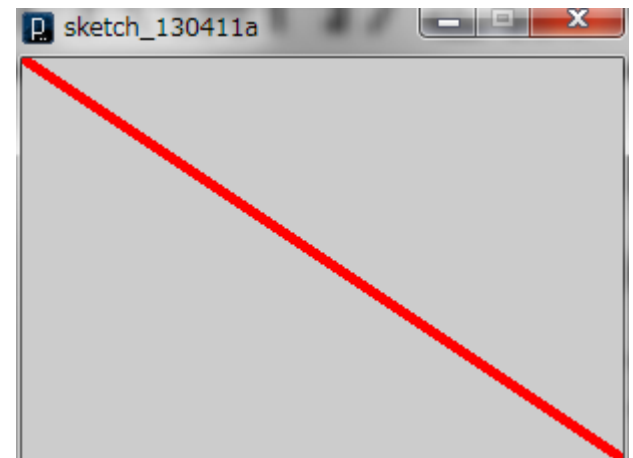
少しずつ線をずらすのは大変

# 太くしたい

(A) 300x200のウィンドウで左上から右下まで描画した赤色の線を太くしたい

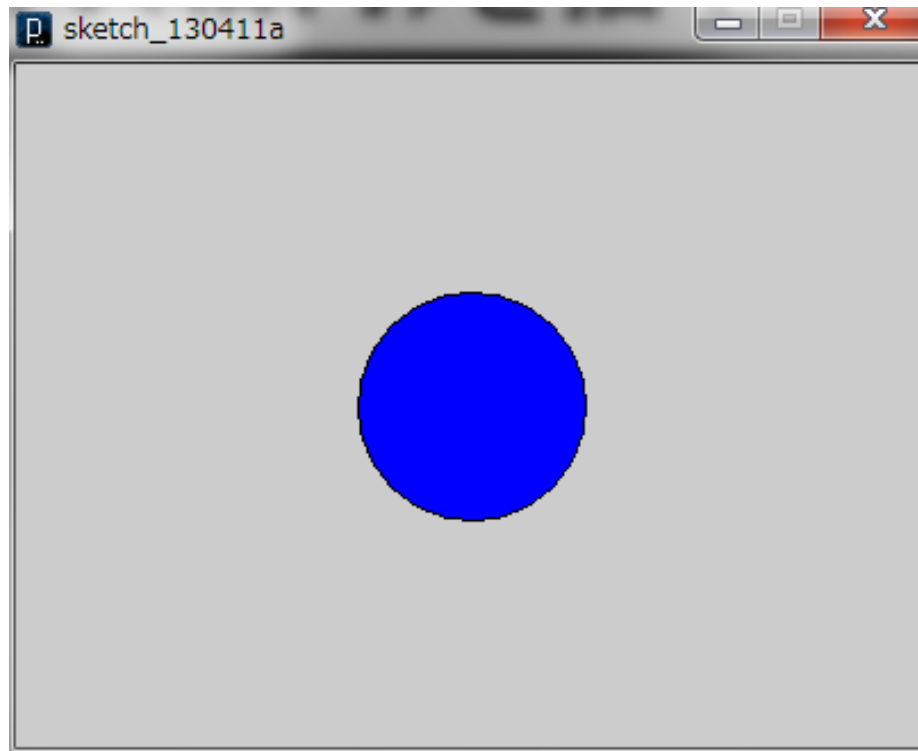
- 左上から右下までの線は  
`line(0,0,300,200);`
- 赤色の線は `stroke(255,0,0);`
- 線を太くする命令は `strokeWeight(太さ);`

```
size(300, 200);  
stroke(255, 0, 0);  
strokeWeight(5);  
line(0, 0, 300, 200);
```



# 色を塗りたい

(Q) 400x300のウィンドウ中央に描いた直径100ピクセルの円を青で塗りつぶしたい



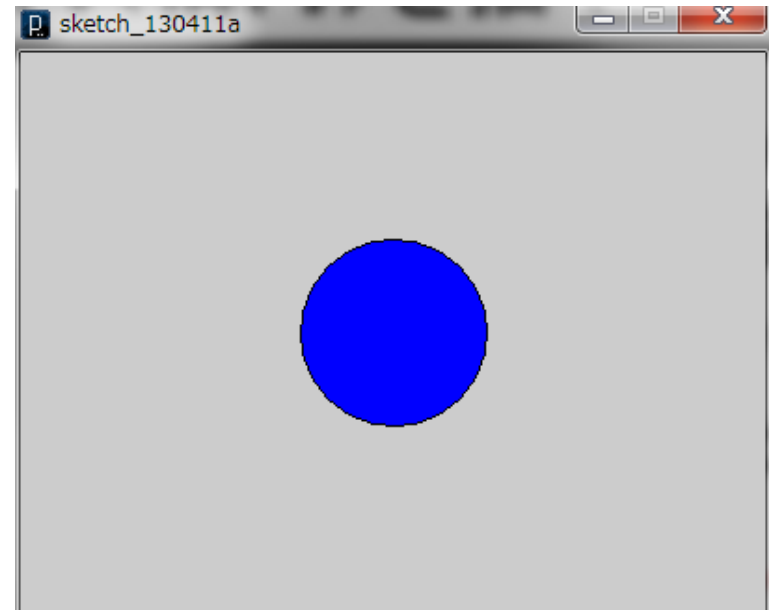
**pointで塗りつぶすのは大変**

# 色を塗りたい

(A) 400x300のウィンドウ中央に描いた直径100ピクセルの円を青で塗りつぶしたい

- 円は `ellipse(200, 150, 100, 100);`
- 塗りつぶし命令は `fill(赤色, 緑色, 青色);`

```
size(400, 300);  
fill(0, 0, 255);  
ellipse(200, 150, 100, 100);
```





# 色を設定する

背景色を設定（色は0-255）

```
background(赤, 緑, 青);
```

線の太さを設定（数字が大きくなるほど太い）

```
strokeWeight(太さ);
```

線の色を設定（色は0-255）

```
stroke(赤, 緑, 青);
```

線を描画しない

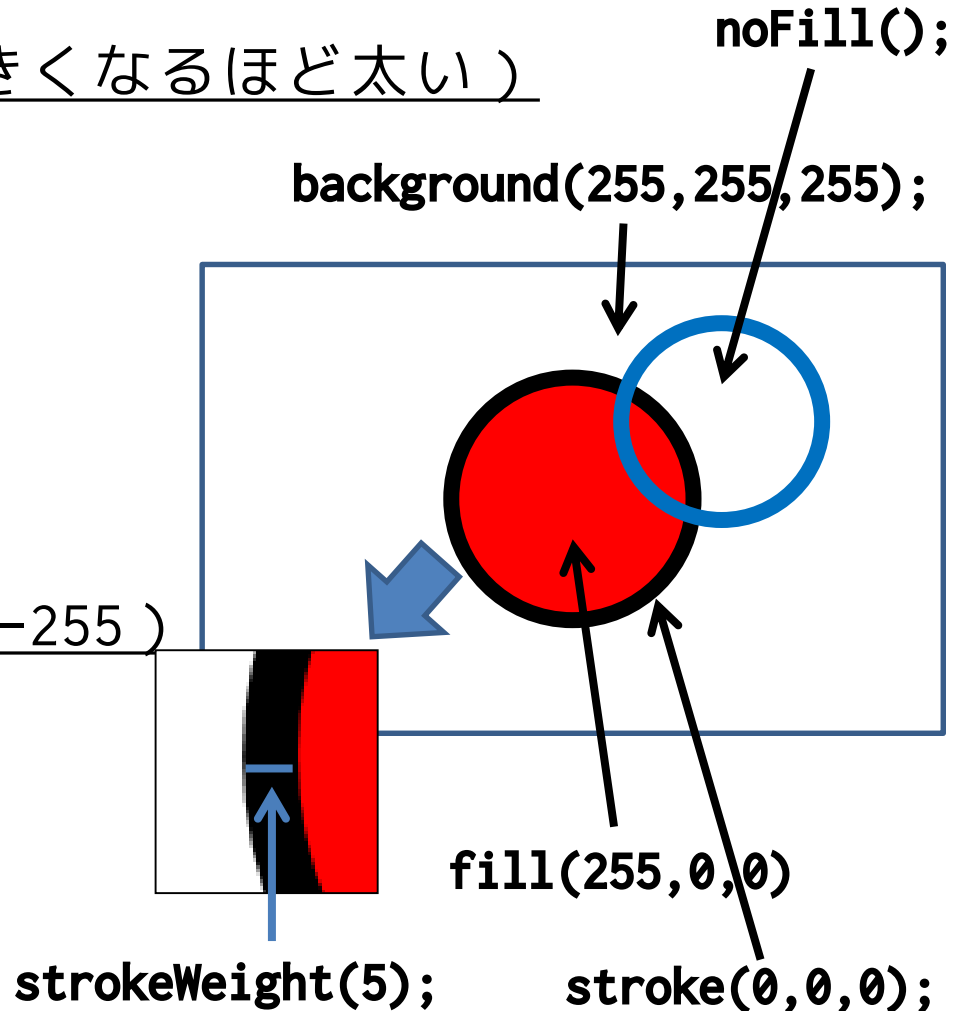
```
noStroke();
```

塗りつぶす色を設定（色は0-255）

```
fill(赤, 緑, 青);
```

塗りつぶさない

```
noFill();
```



# 色の設定はどこまで有効？

- 色や線の太さの設定は，次に他の設定がなされるまでずっと有効
  - stroke, noStroke, strokeWeight, fill, noFill

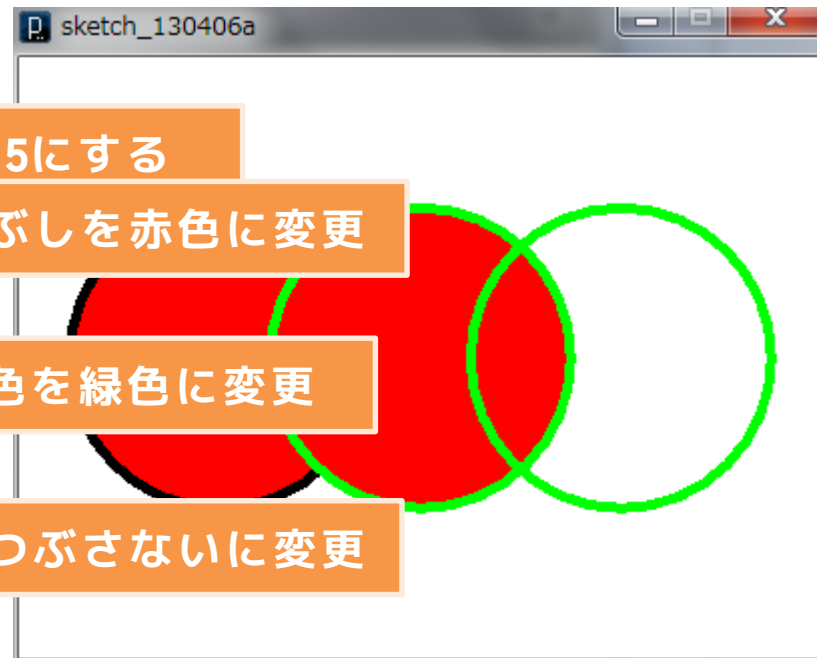
```
size(400, 300);  
background(255, 255, 255);  
strokeWeight(5);  
fill(255, 0, 0);  
ellipse(100, 150, 150, 150);  
stroke(0, 255, 0);  
ellipse(200, 150, 150, 150);  
noFill();  
ellipse(300, 150, 150, 150);
```

線の太さを5にする

塗りつぶしを赤色に変更

線の色を緑色に変更

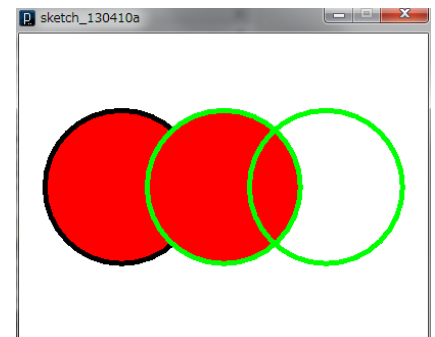
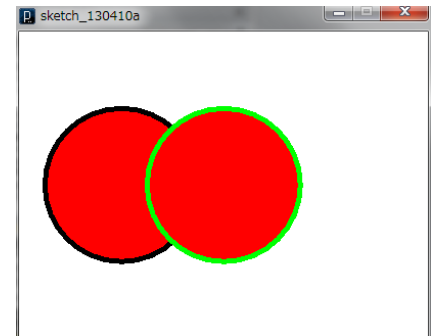
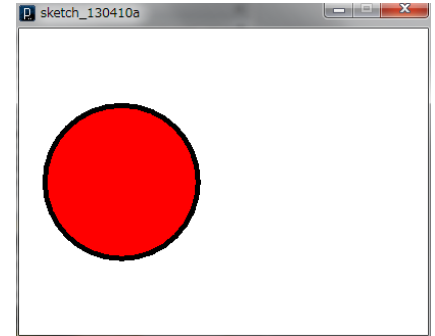
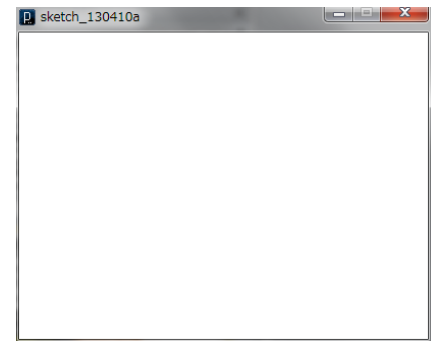
塗りつぶさないに変更



# 色の設定はどこまで有効？

```
size(400, 300);  
background(255, 255, 255);  
strokeWeight(5);  
fill(255, 0, 0);  
ellipse(100, 150, 150, 150);  
stroke(0, 255, 0);  
ellipse(200, 150, 150, 150);  
noFill();  
ellipse(300, 150, 150, 150);
```

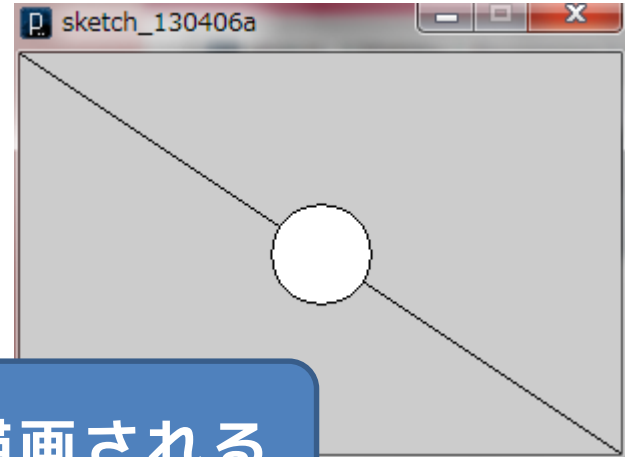
stroke	(0, 255, 0)
strokeWeight	5
fill	(255, 0, 0)



# 順番が重要

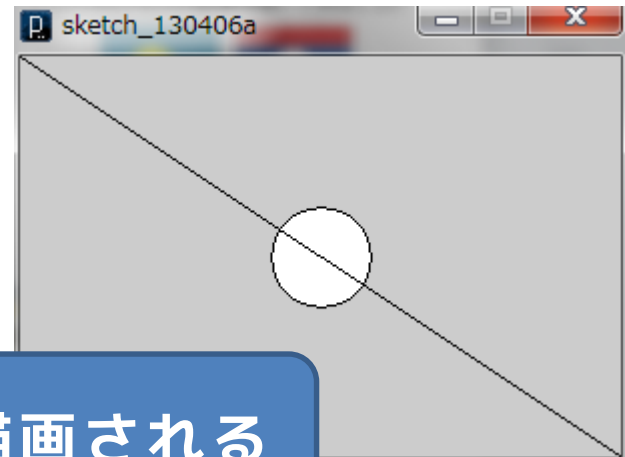
```
size(300, 200);  
line(0, 0, 300, 200);  
ellipse(150, 100, 50, 50);
```

線が描画されて円が描画される



```
size(300, 200);  
ellipse(150, 100, 50, 50);  
line(0, 0, 300, 200);
```

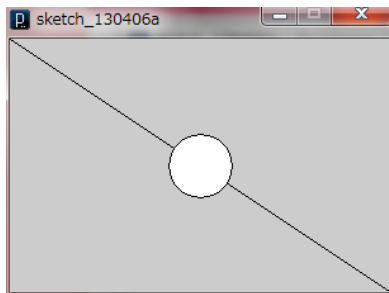
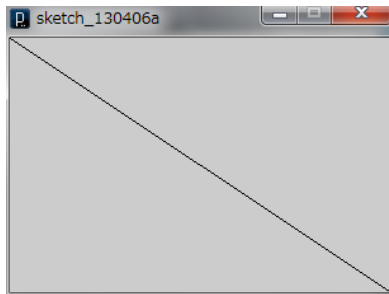
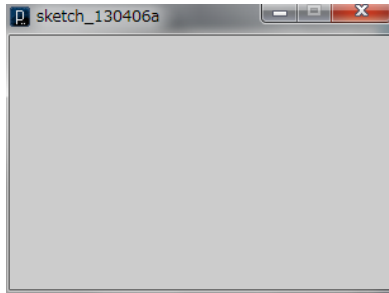
円が描画されて線が描画される



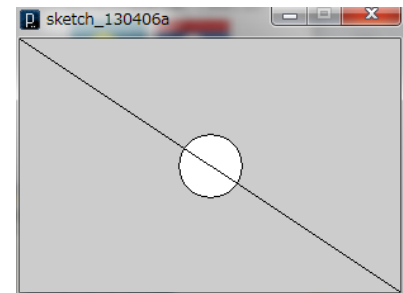
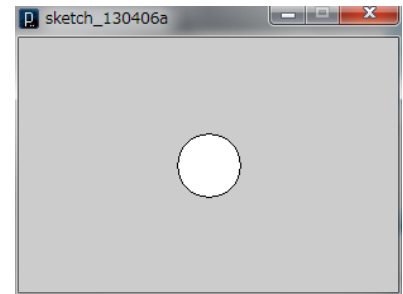
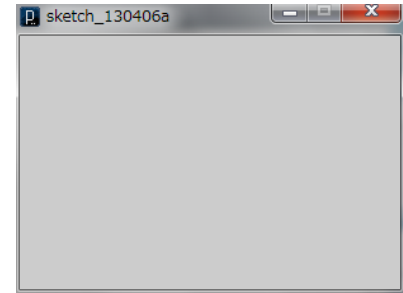
# 順番が重要

```
size(300, 200);  
line(0, 0, 300, 200);  
ellipse(150, 100, 50, 50);
```

```
size(300, 200);  
ellipse(150, 100, 50, 50);  
line(0, 0, 300, 200);
```

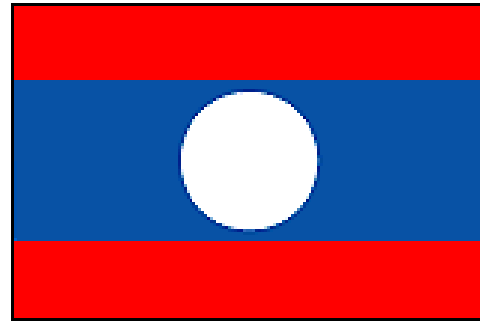
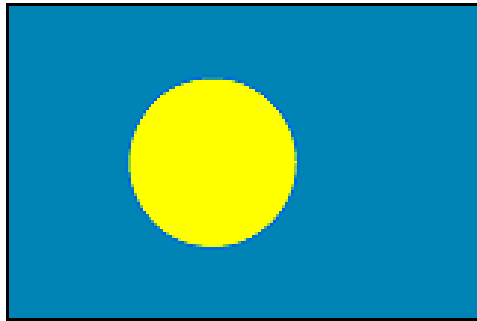


一瞬で描画されるため  
気づかないが本当は  
こんな感じで描画している

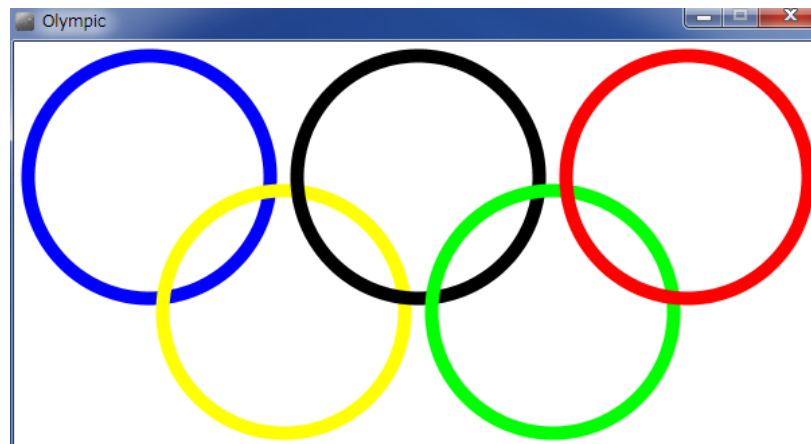


# 演習

- パラオとラオスの国旗を描いてみましょう



- 五輪の輪を描いてみましょう
  - 重なりの状態についてはある程度でOK



# コメント

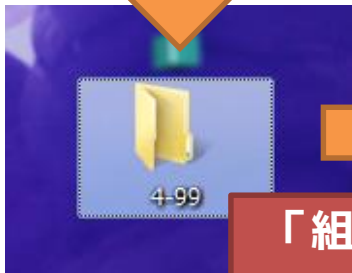
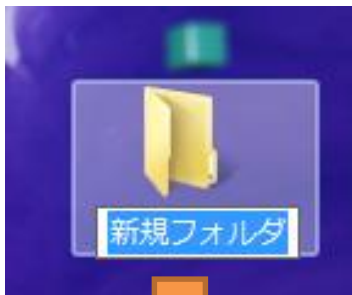
- プログラムとしては実行されない，人間用の説明文で，後で読むためにどんどん書く！
- コメントは「`//`」か「`/*`」と「`*/`」のペアを利用
  - 「`//`」は「`//`」以降行末までをコメントとして解釈
  - 「`/*`」と「`*/`」は，その中身を全てコメントとして解釈

```
// 背景を白色に設定
background(255, 255, 255);
fill(255, 0, 0); // 赤色で塗りつぶす
// fill(0, 0, 255); // 青色で塗りつぶす
/*
    ここから
    ここも
    ここまでもコメントですよー
*/
```

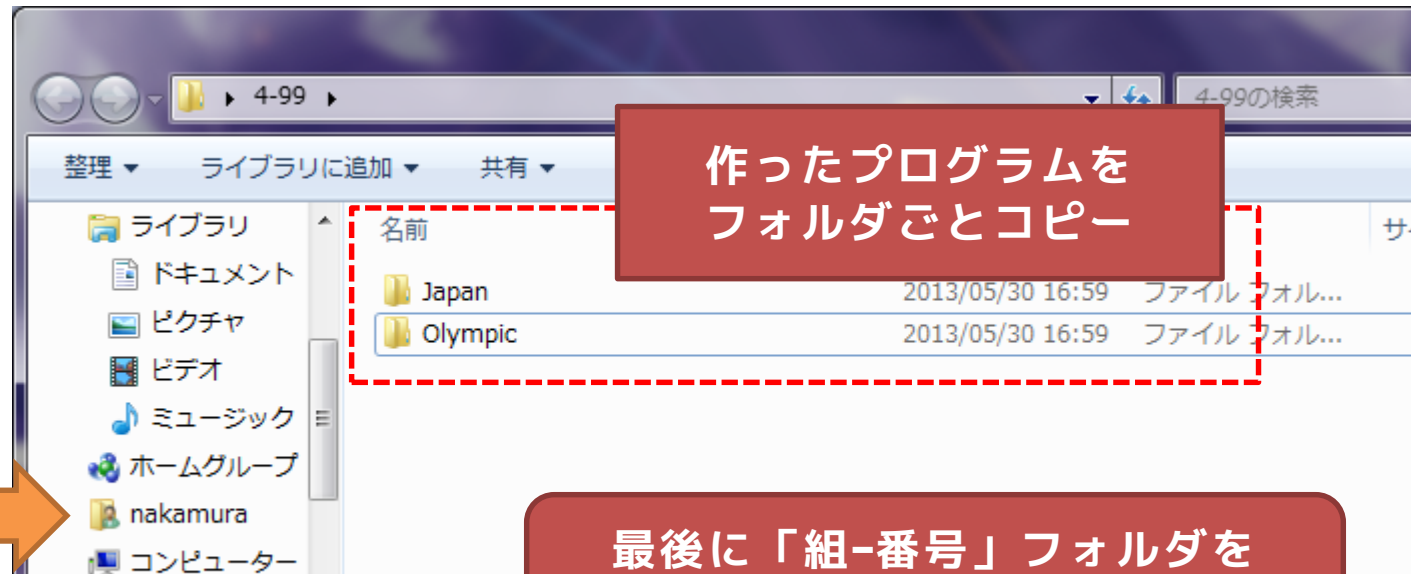
動かない時に消さずコメントアウトしよう！

# 課題の提出方法

- デスクトップに「組+番号」のフォルダを作る
- 「組-番号」のフォルダの中に、作成した Processing のプログラムをフォルダごとコピー
- 「組-番号」のフォルダを、共有フォルダに提出



「組-番号」のフォルダを作る  
(例) 4組99番



最後に「組-番号」フォルダを  
共有フォルダにコピーする！



# 参考

---

- Processing入門
  - <http://www.cp.cmc.osaka-u.ac.jp/~kikuchi/kougi/simulation2009/processing0.html>
- Processing リファレンス
  - <http://processing.org/reference/>
- yoppa.org
  - 田所淳先生のサイト（とても参考になります）
  - <http://yoppa.org>