

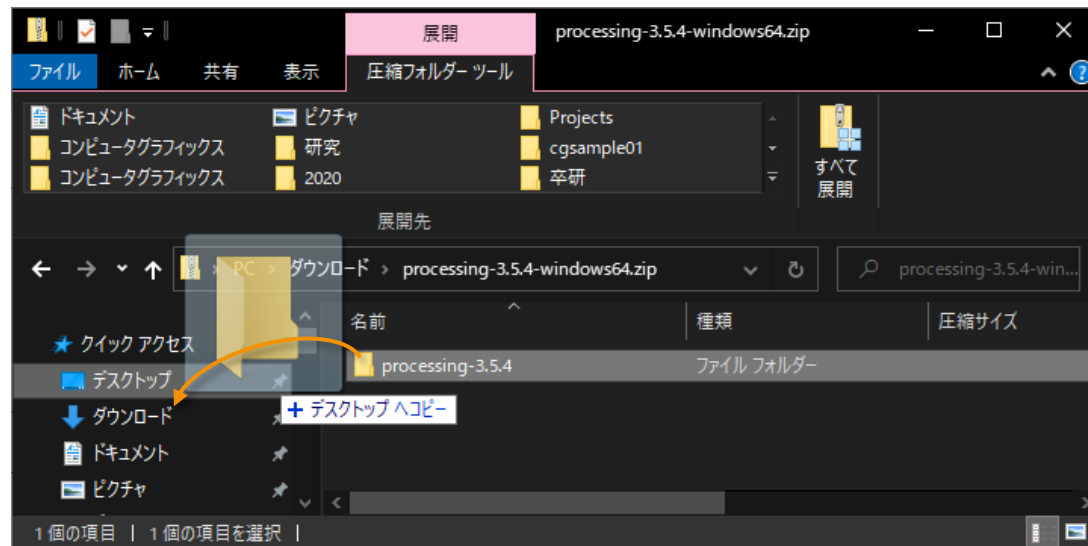
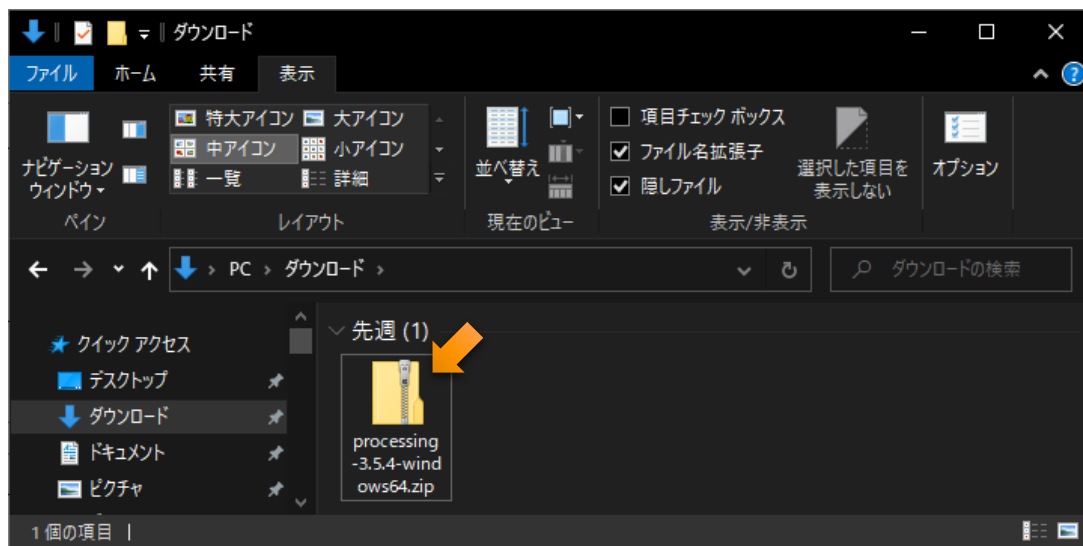
メディアデザインメジャー メジャー体験演習

ビジュアルグループ 1日目

ZIP ファイルを展開する

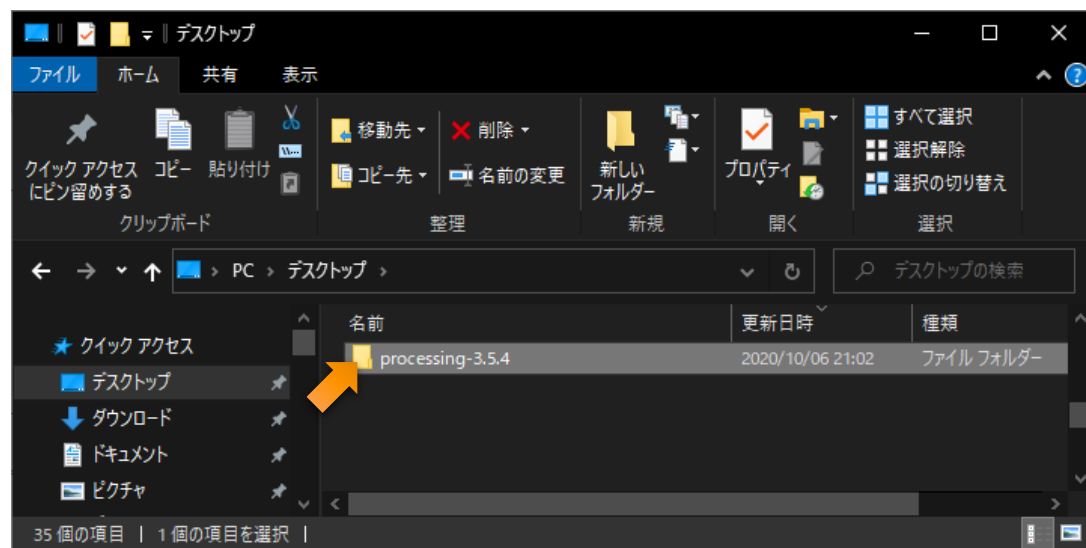
ZIP ファイルをエクスプローラーで開く（他のアーカイバも使用可）

中身をデスクトップに移動する（すべて展開したものを移動しても可）

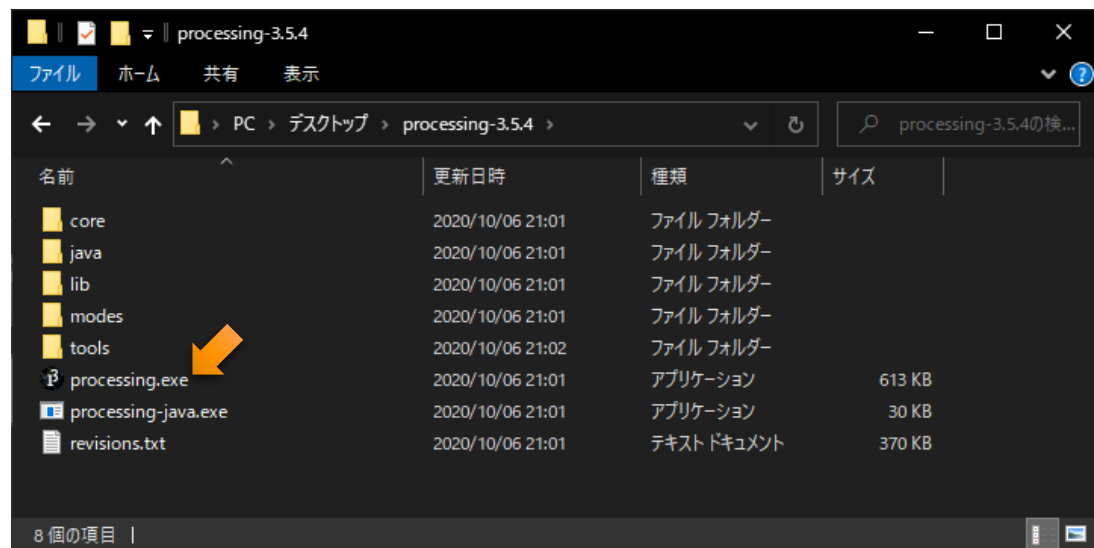


Processing を起動する

フォルダを開く

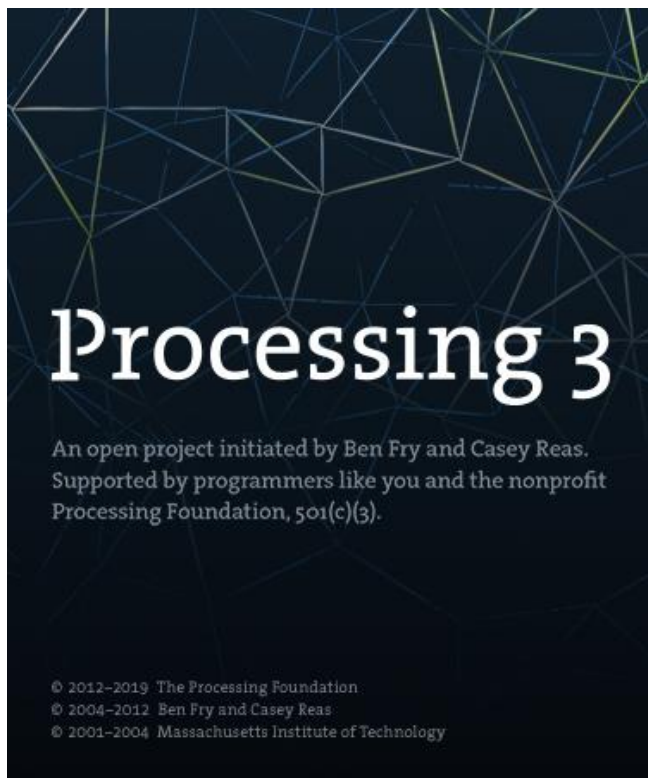


processing.exe をダブルクリックする

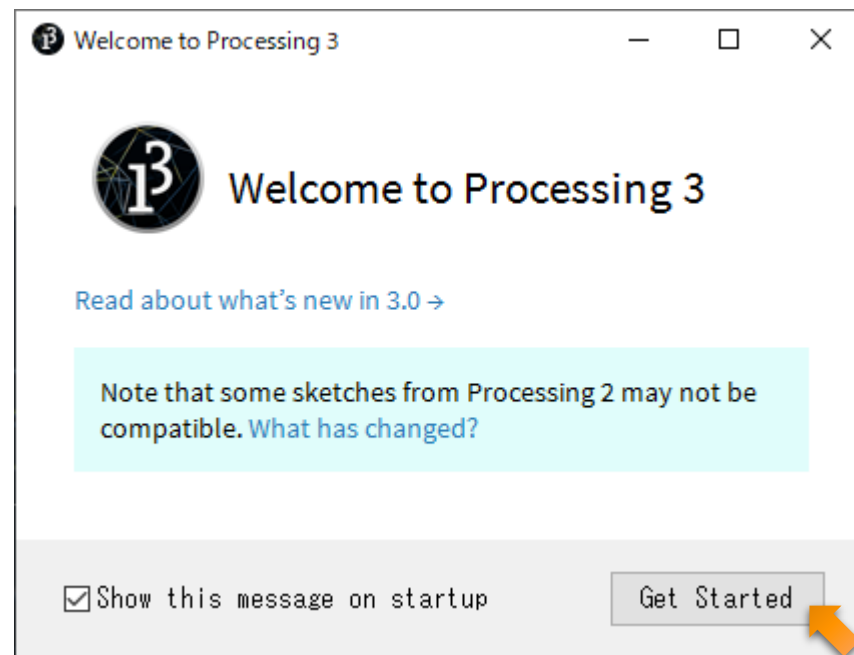


起動中

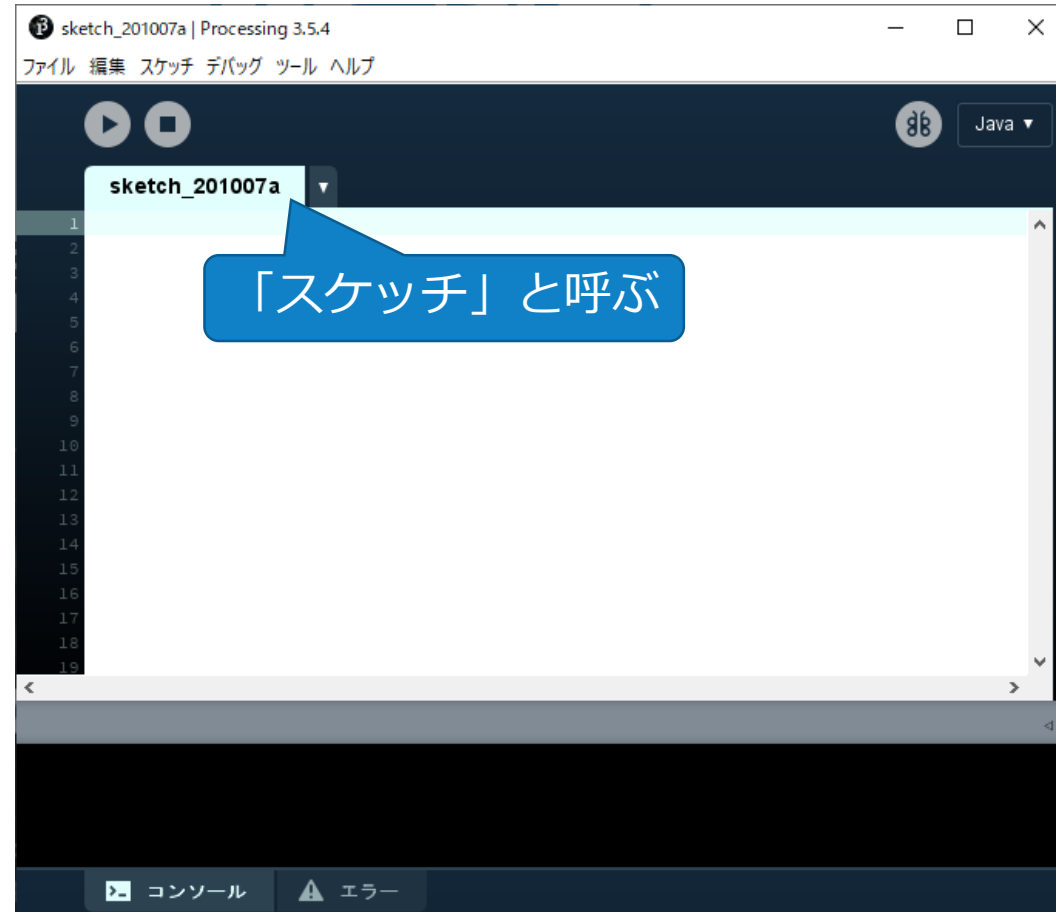
しばらく待つ



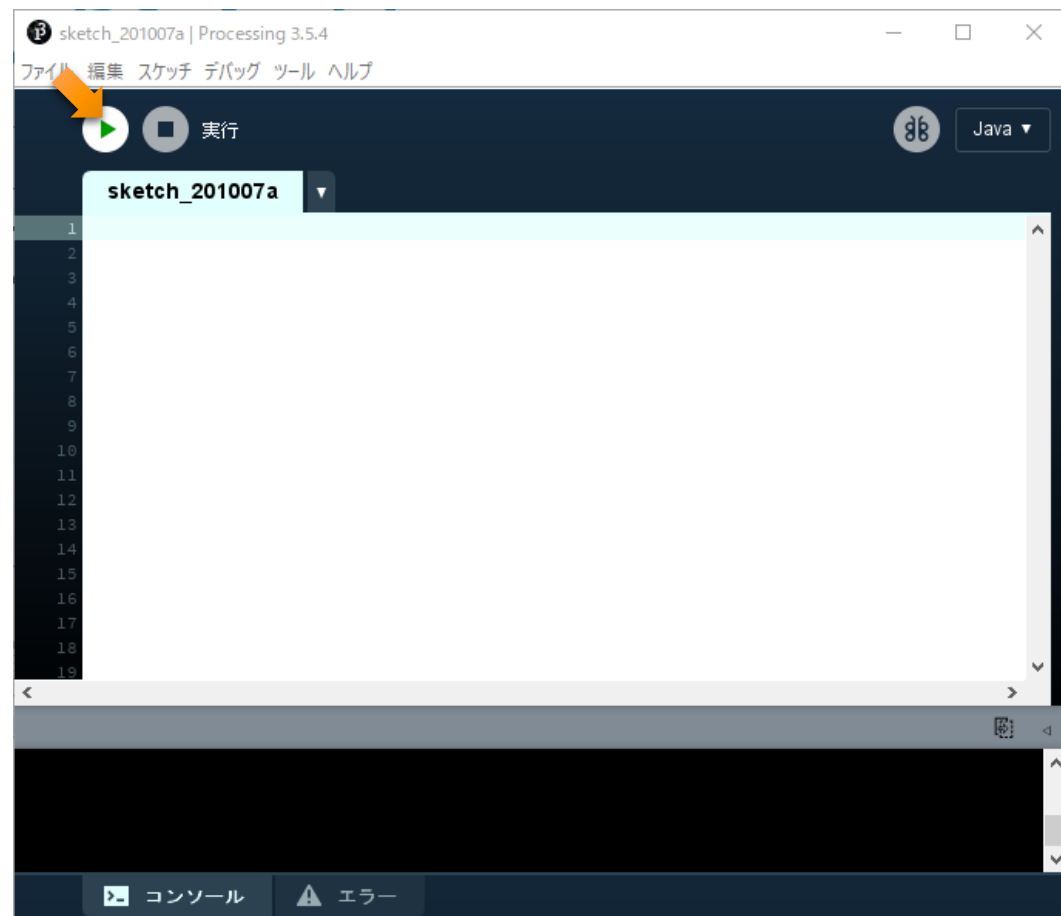
Get Started をクリックする



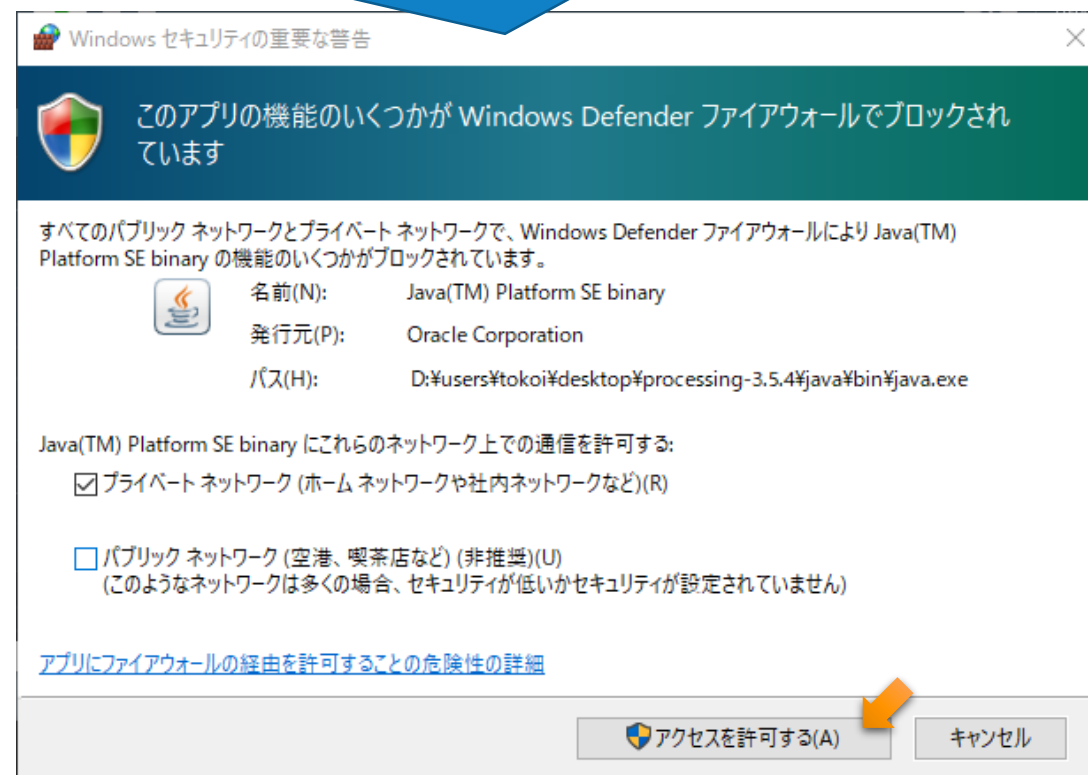
Processing のウィンドウ



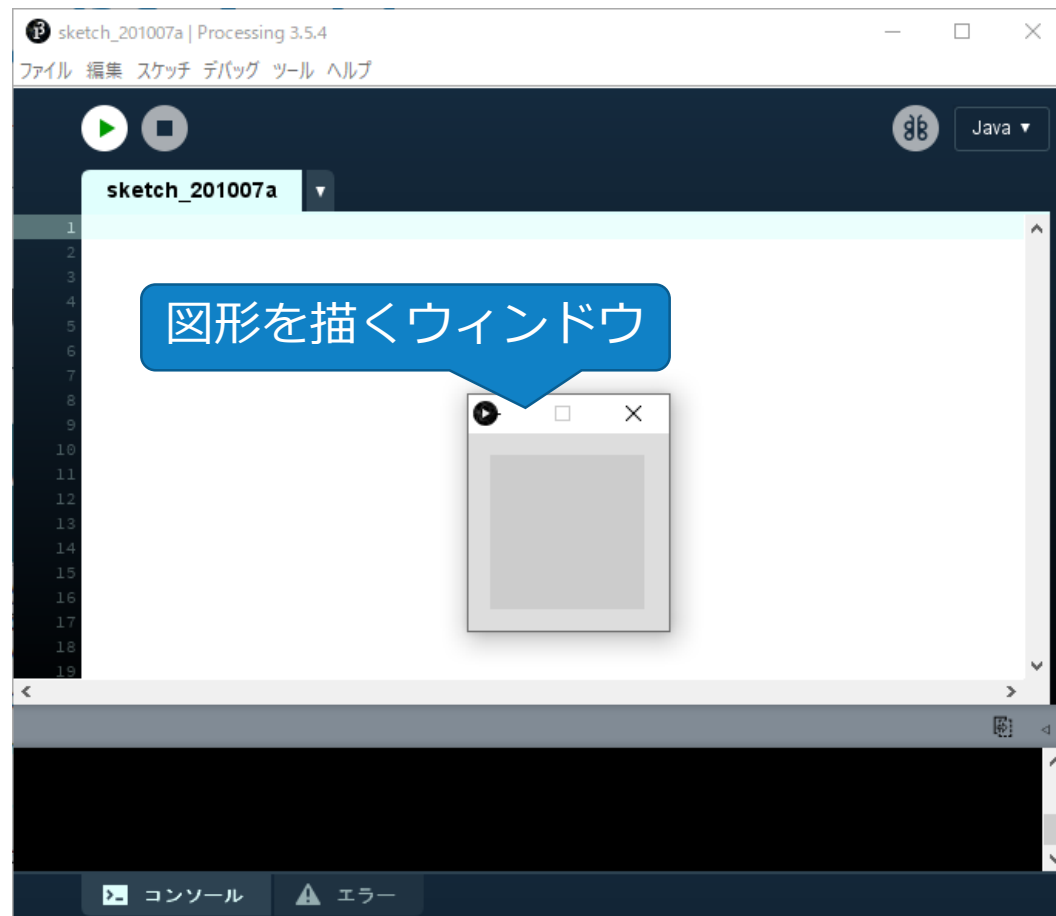
実行する



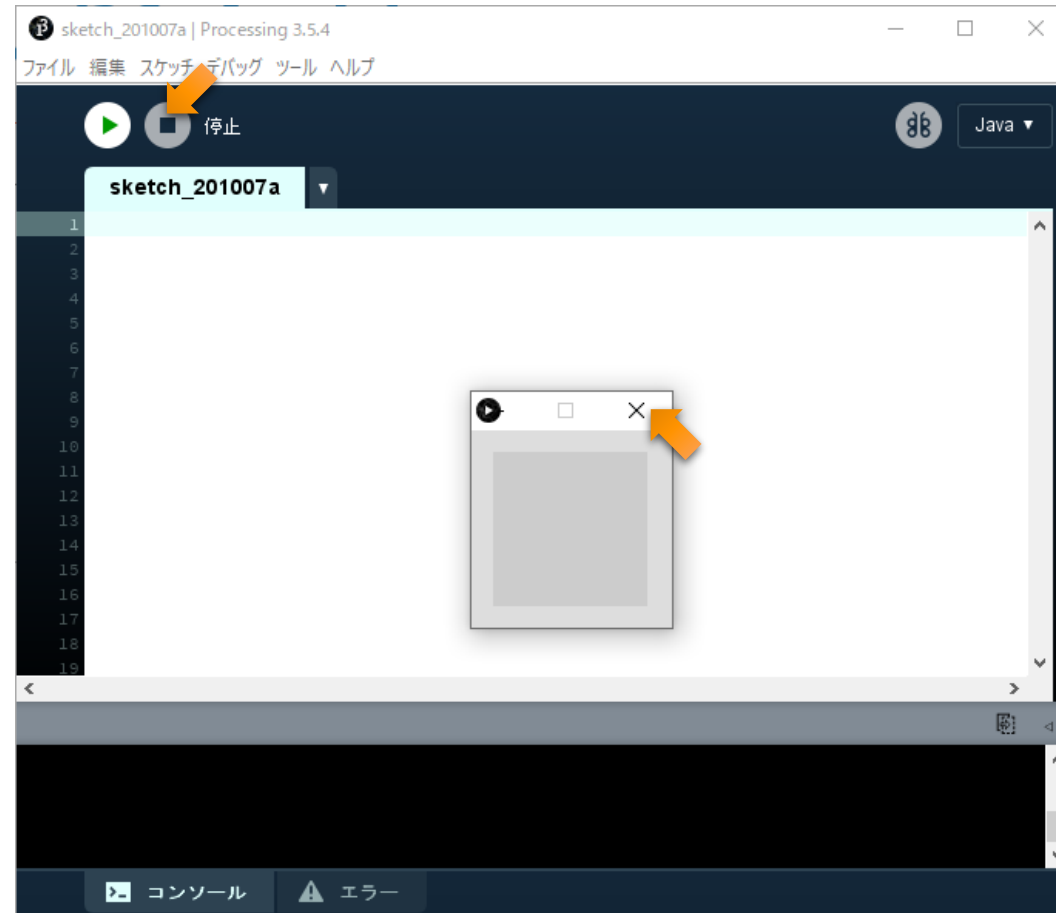
Processing を始めて使うときに出ることがある



実行中



停止する

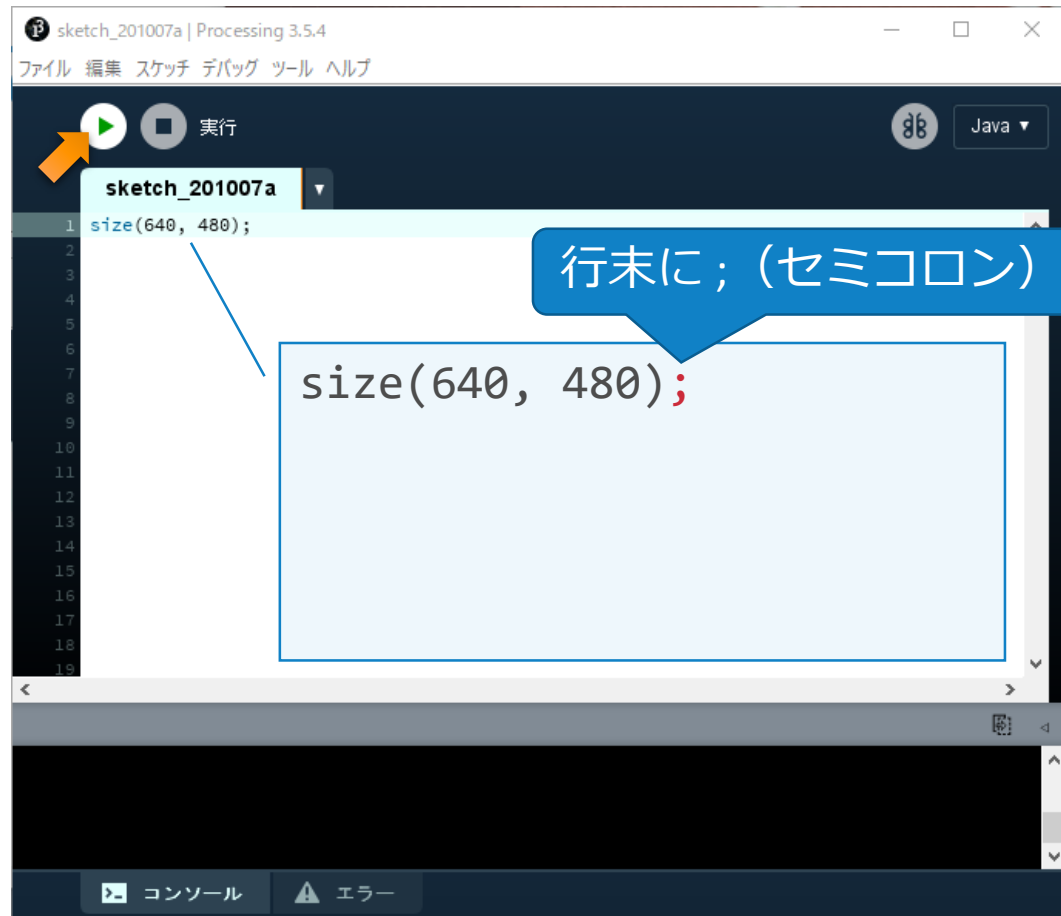


プログラムで絵を描く

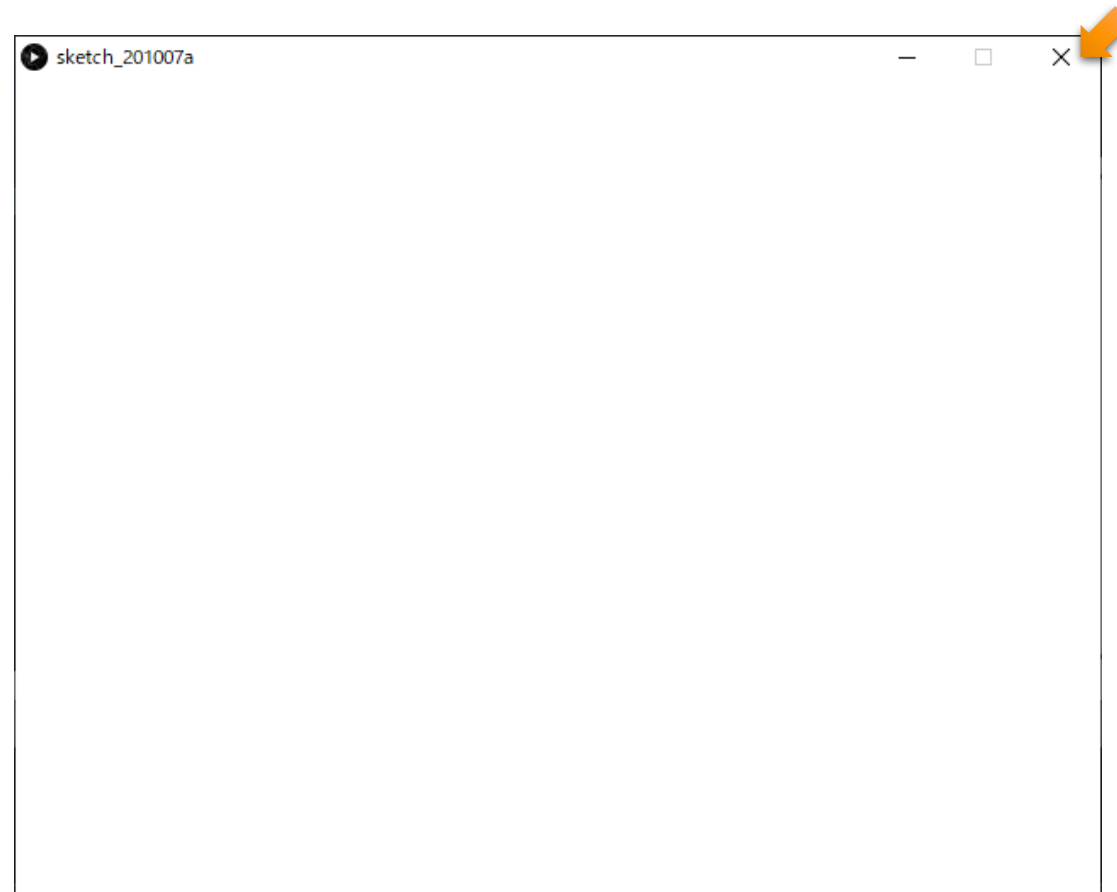
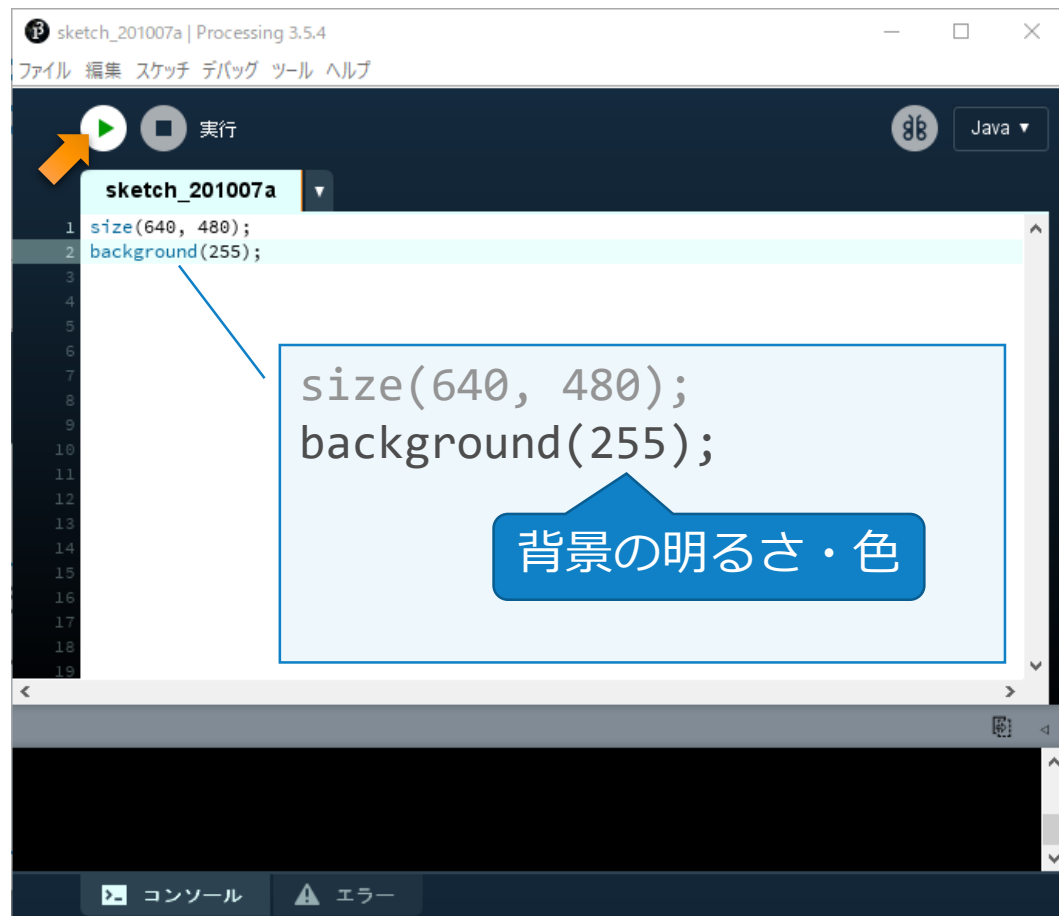
“Processing was designed to be a flexible software sketchbook.”

(Processing のリファレンスマニュアル <https://processing.org/reference/> より)


サイズを 640×480 に設定する





背景を白に設定する





色の指定方法


`background(0);` 


`background(255);` 


`background(128);` 

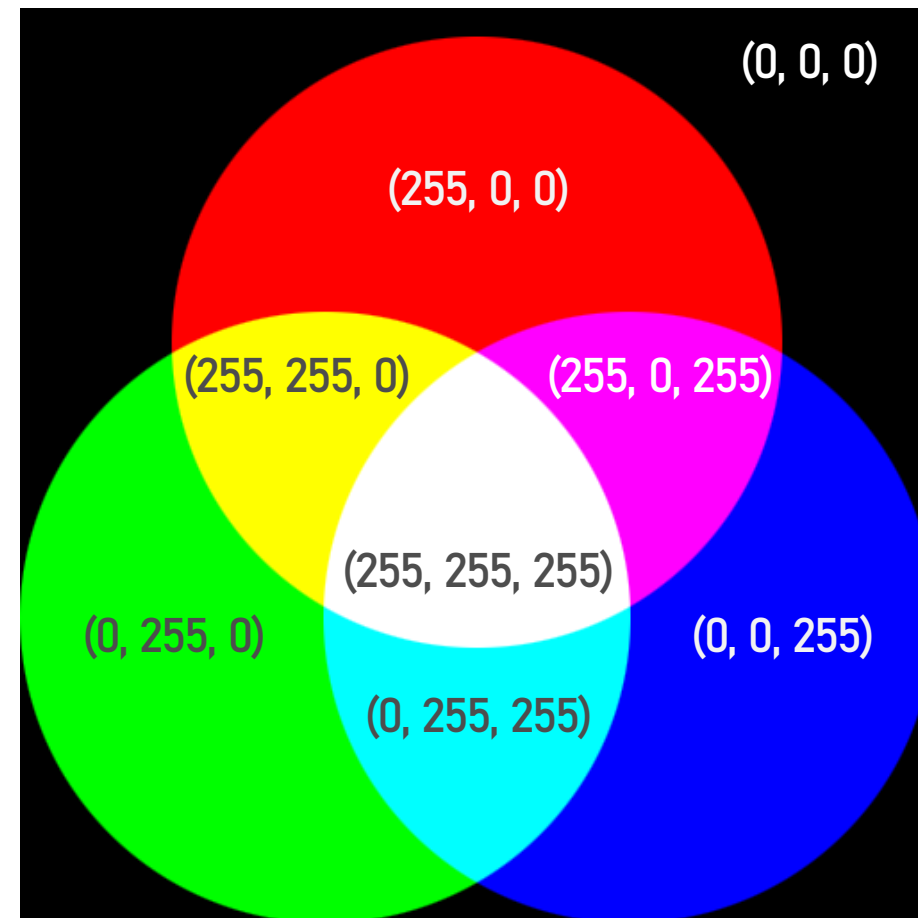
`background(255, 0, 0);` 

`background(0, 255, 0);` 

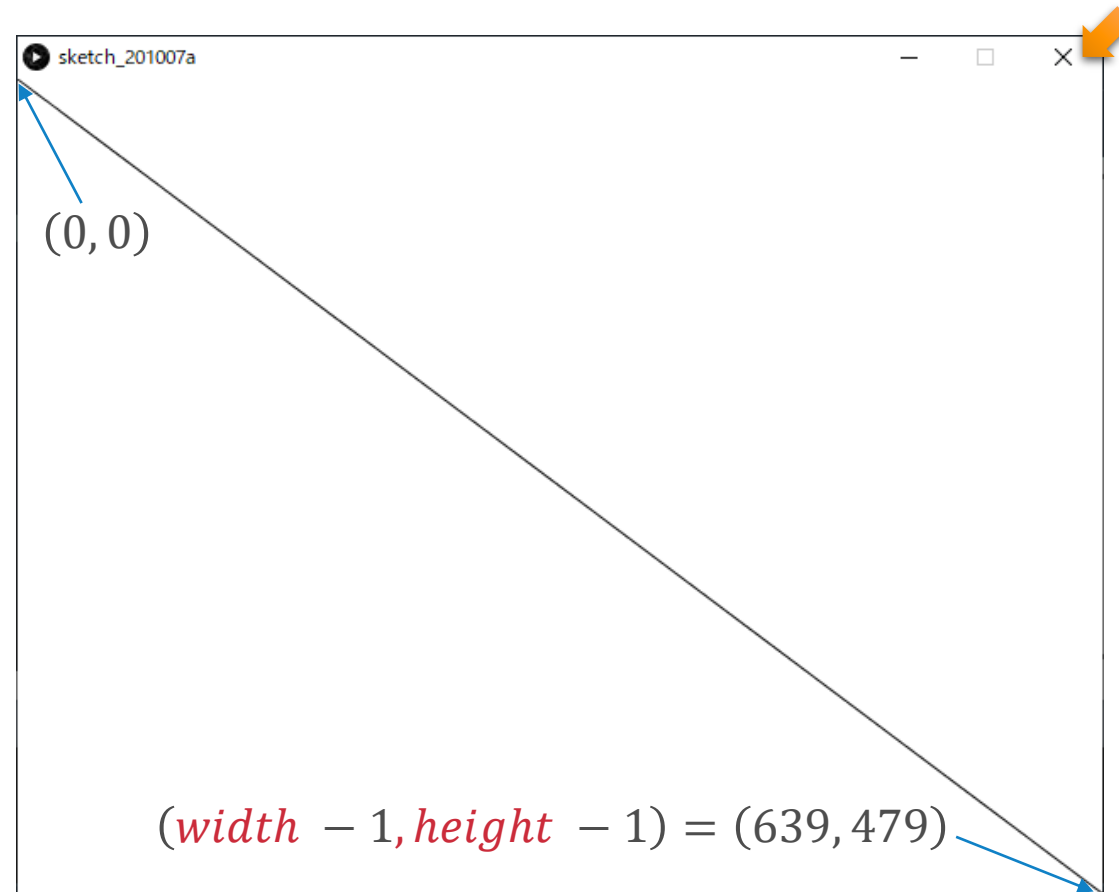
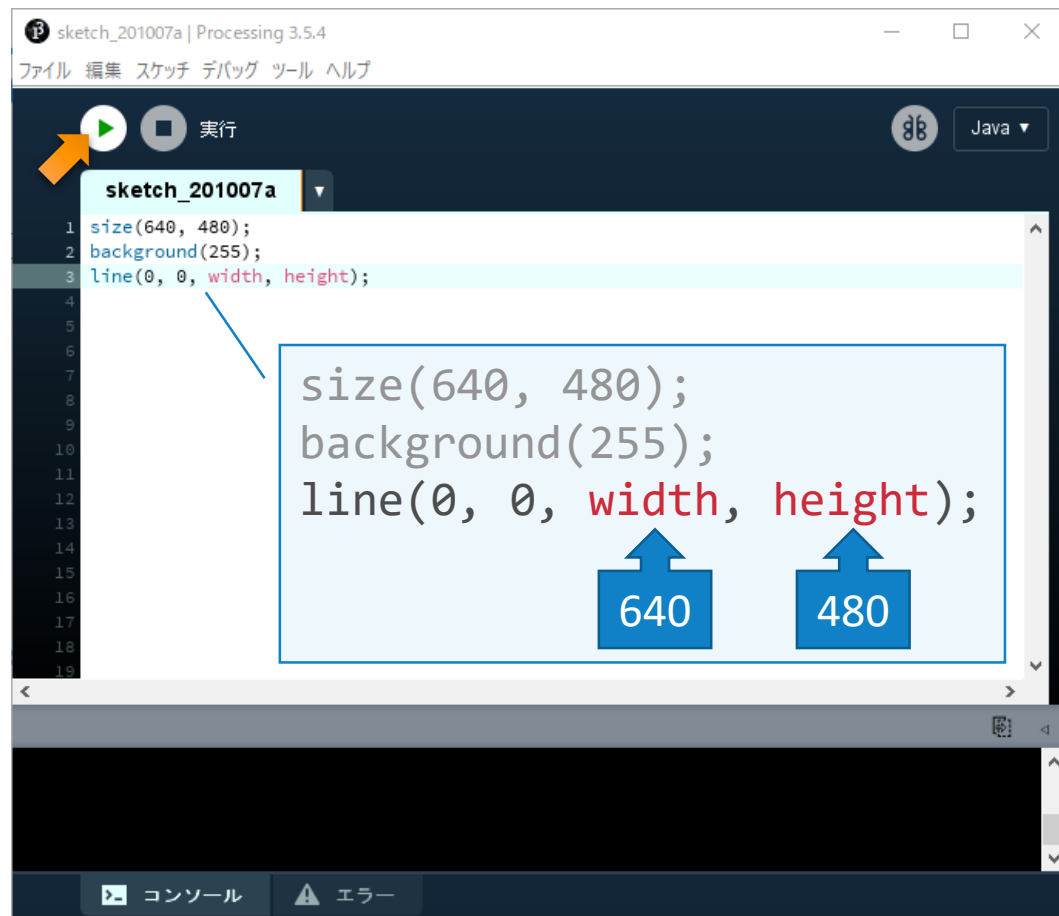
`background(0, 0, 255);` 

`background(255, 192, 0);` 

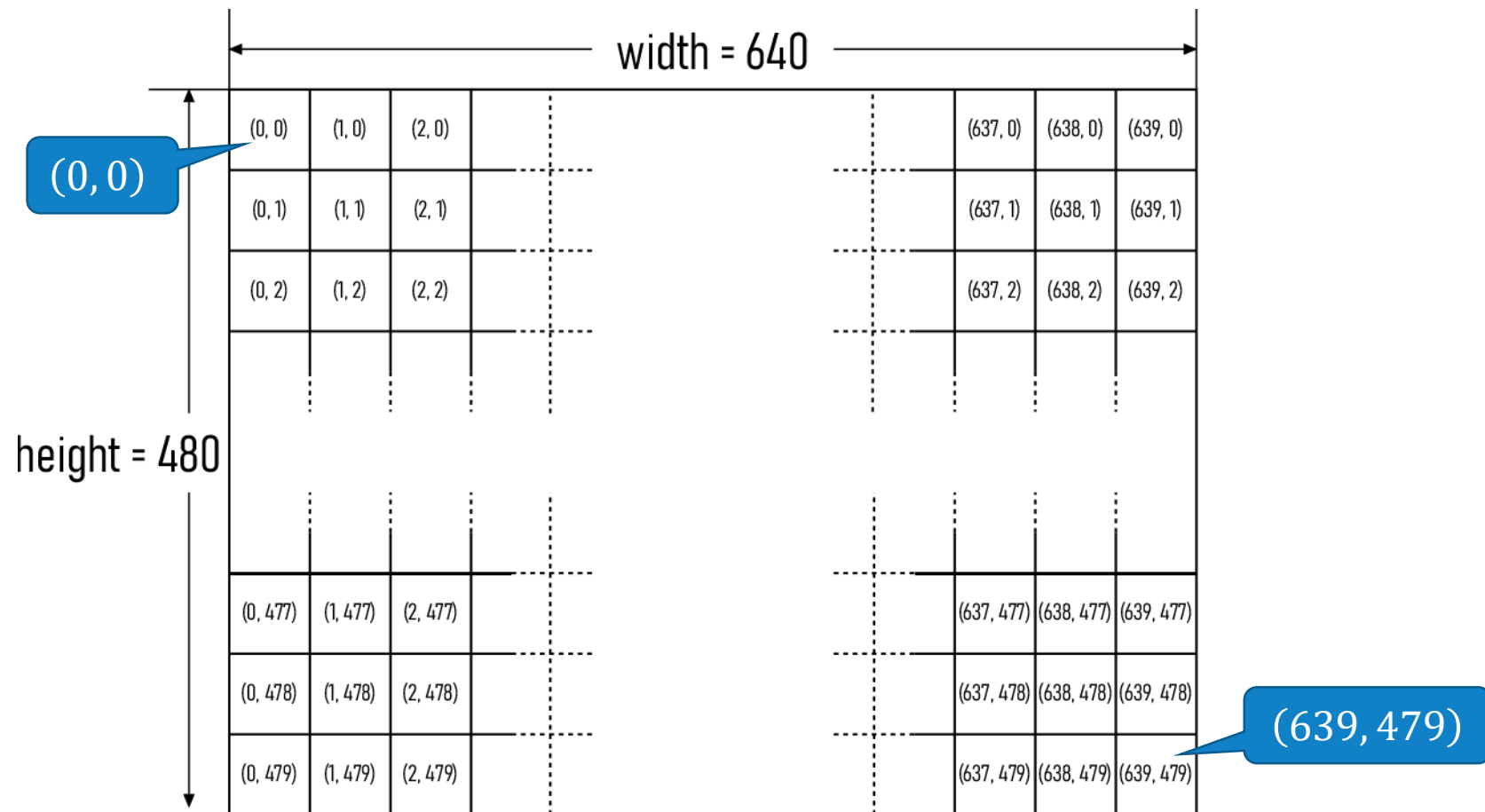
`background(255, 32, 128);` 



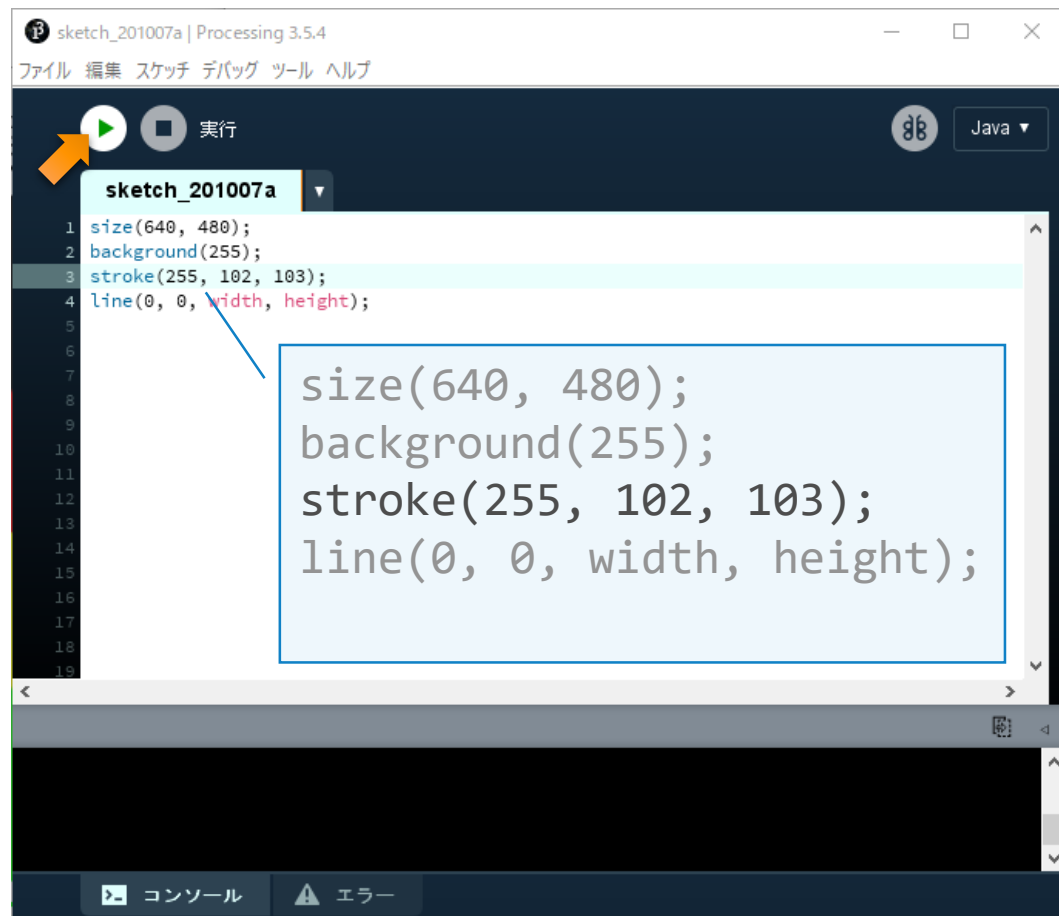
線分を描画する



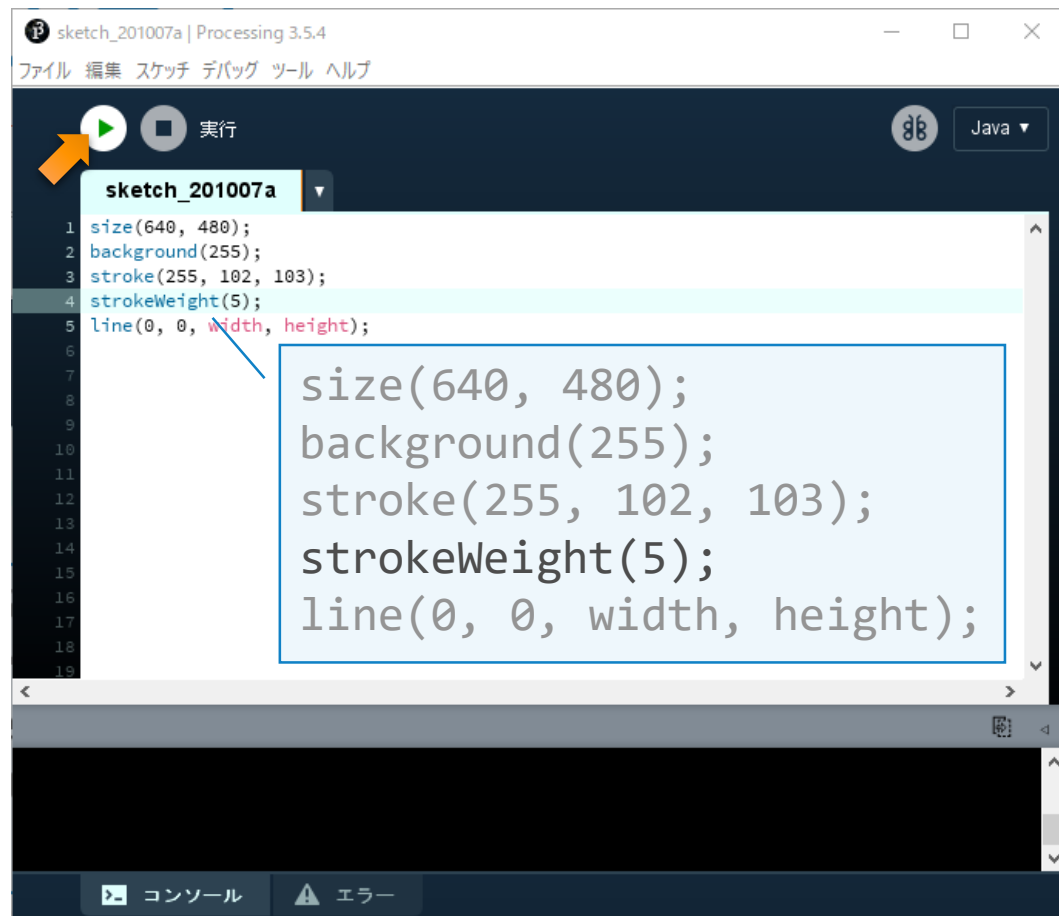
座標系



線の色を指定する

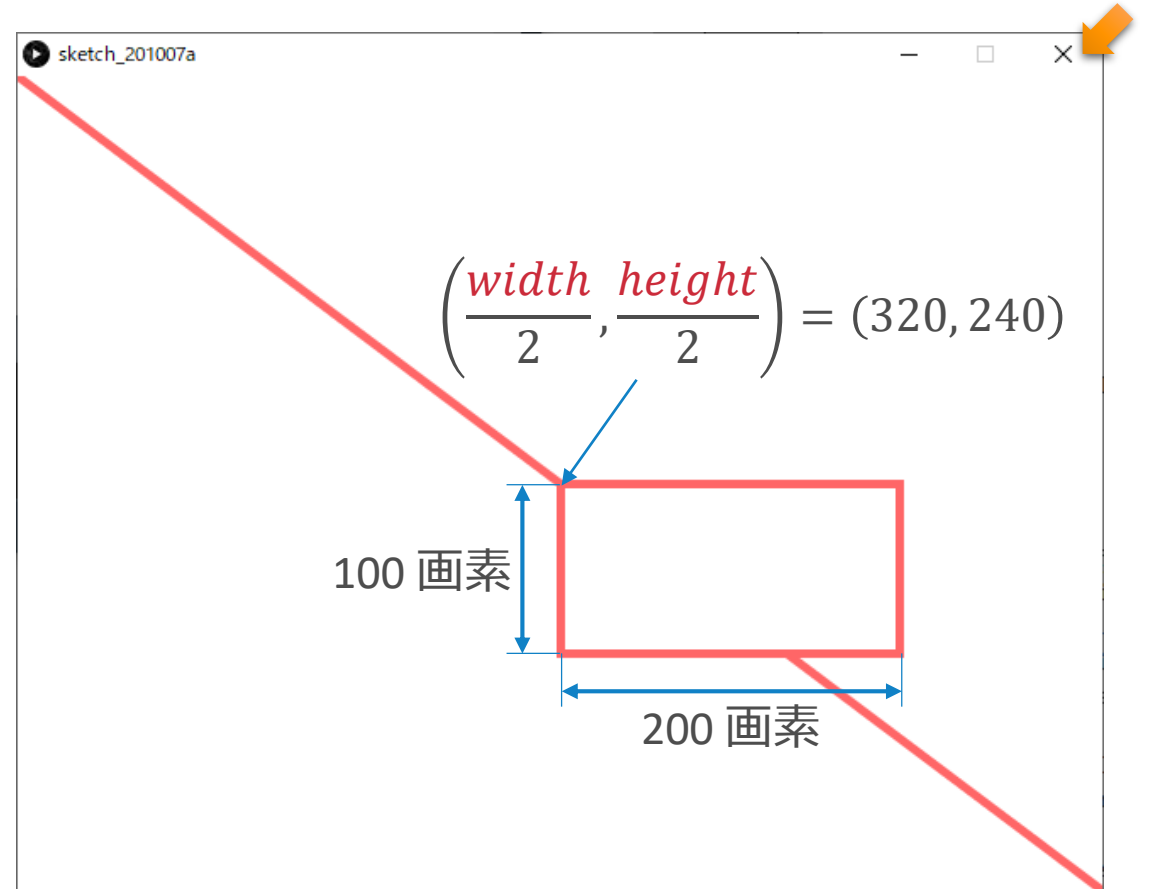


線の太さを指定する



矩形を描画する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
rect(width / 2, height / 2, 200, 100);
```



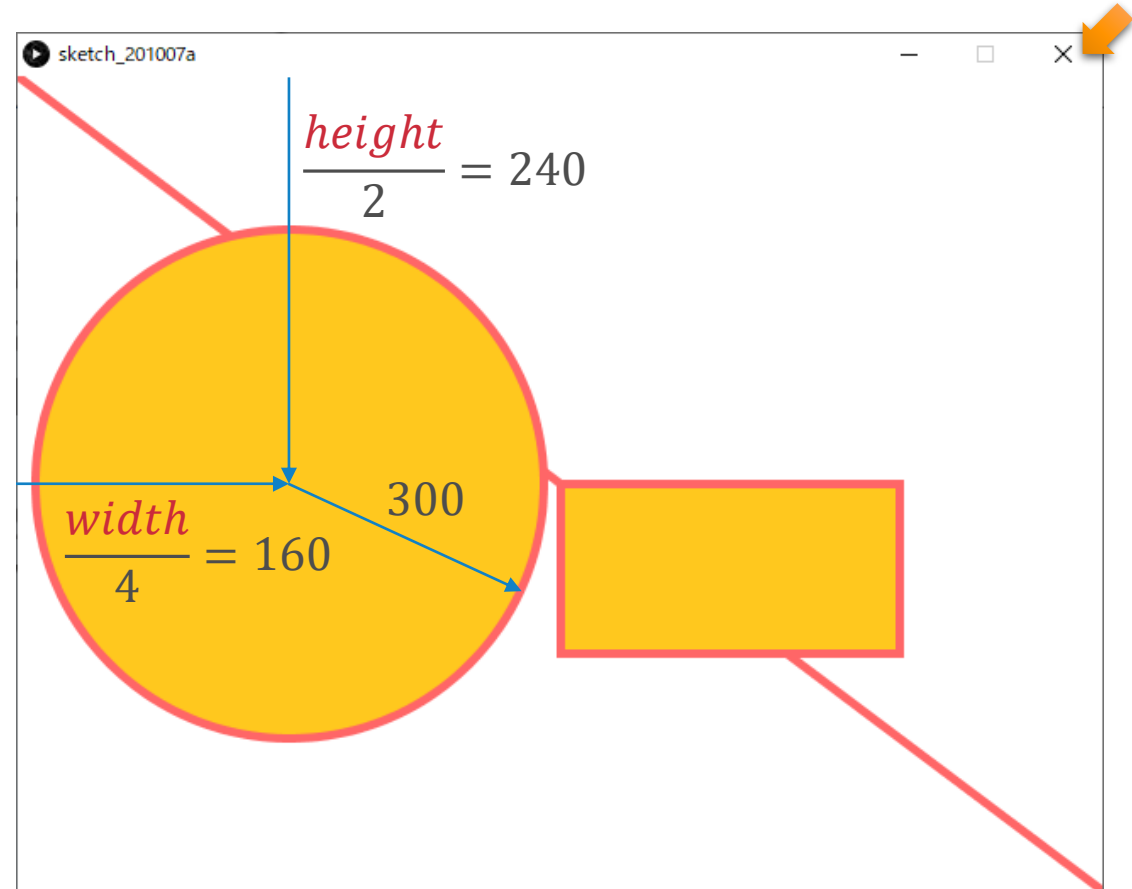
矩形の塗りつぶし色を指定する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);
```



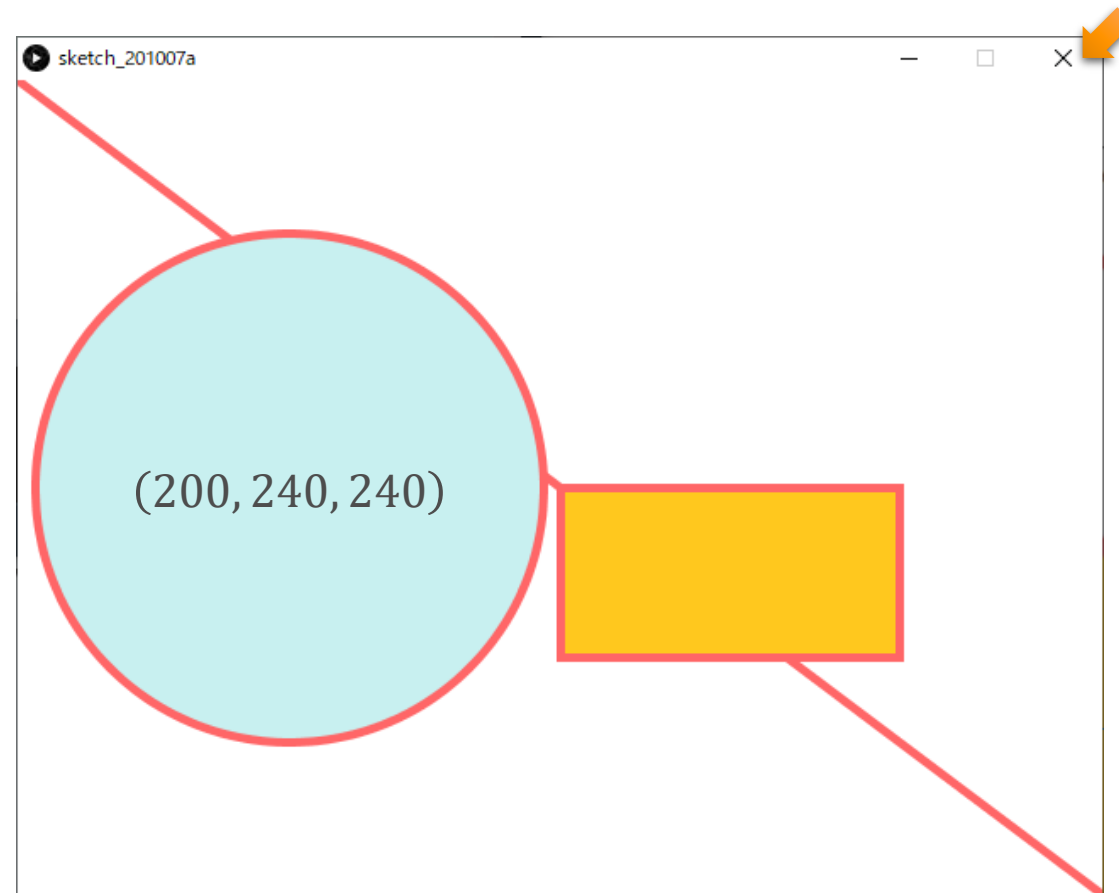
円を描画する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
circle(width / 4, height / 2, 300);
```



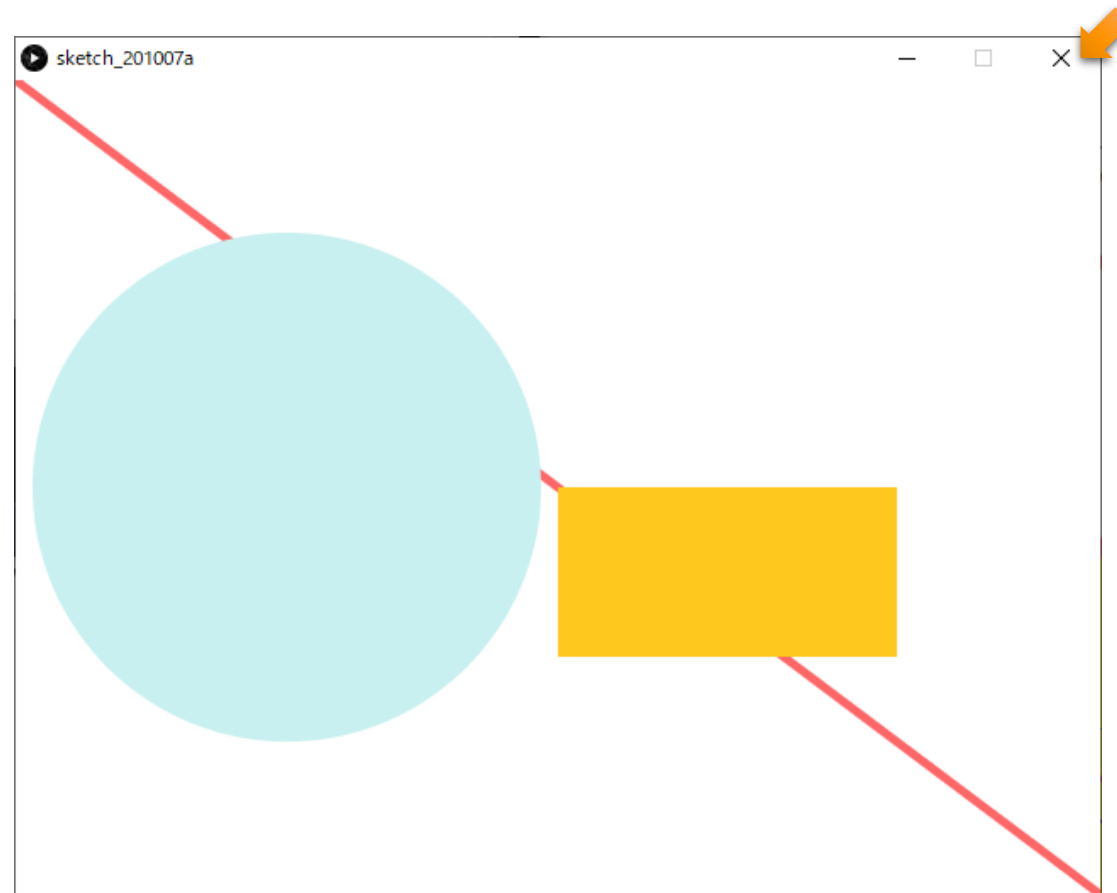
円の塗りつぶし色を指定する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
fill(200, 240, 240);  
circle(width / 4, height / 2, 300);
```



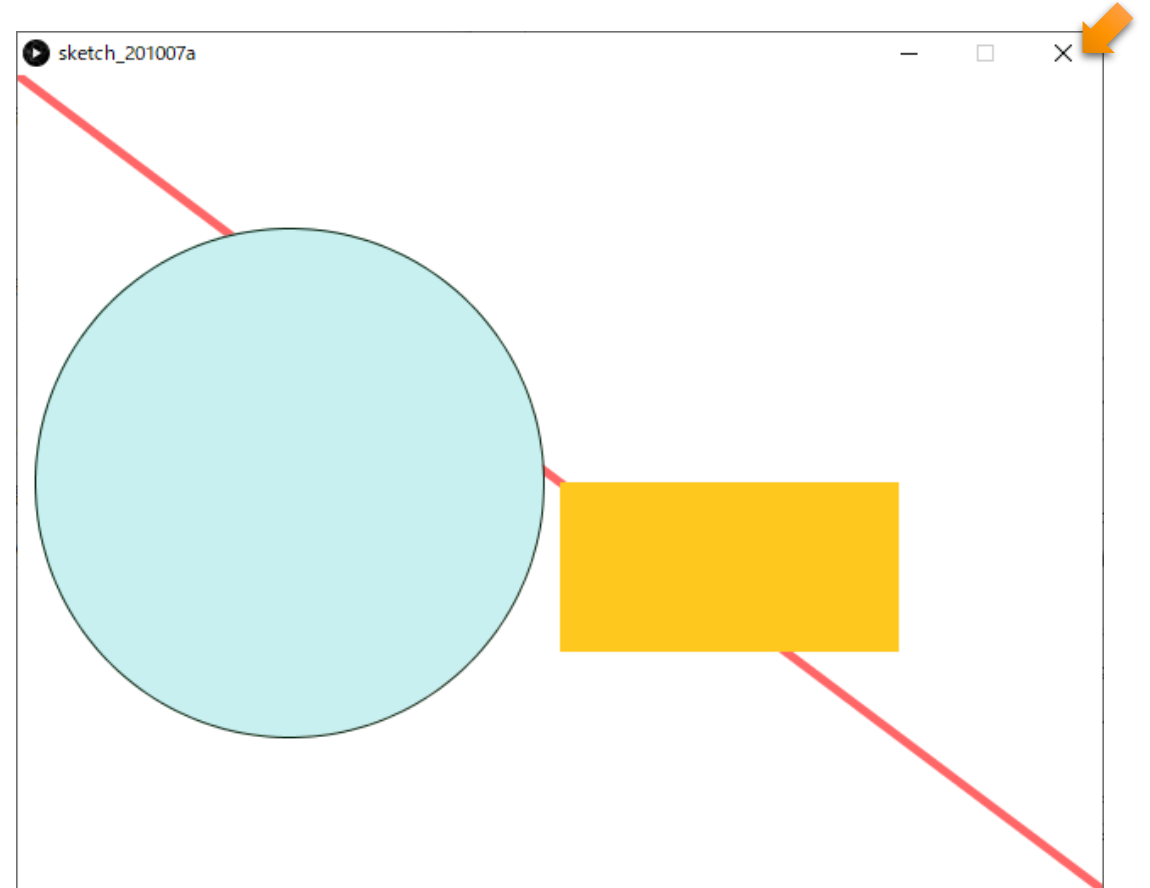
境界線を描かないようにする

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
noStroke();  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
fill(200, 240, 240);  
circle(width / 4, height / 2, 300);
```



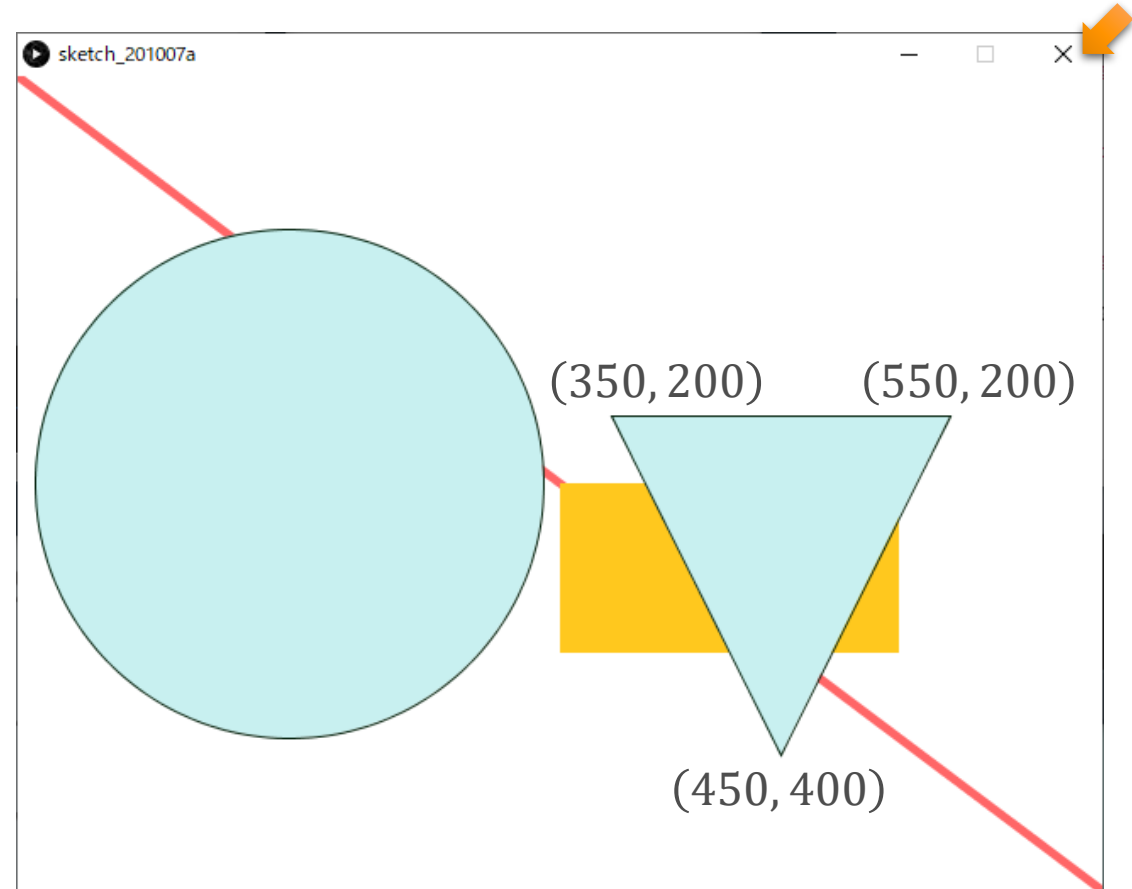
円の境界線の太さと色を設定する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
noStroke();  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
stroke(10, 40, 20);  
strokeWeight(1);  
fill(200, 240, 240);  
circle(width / 4, height / 2, 300);
```



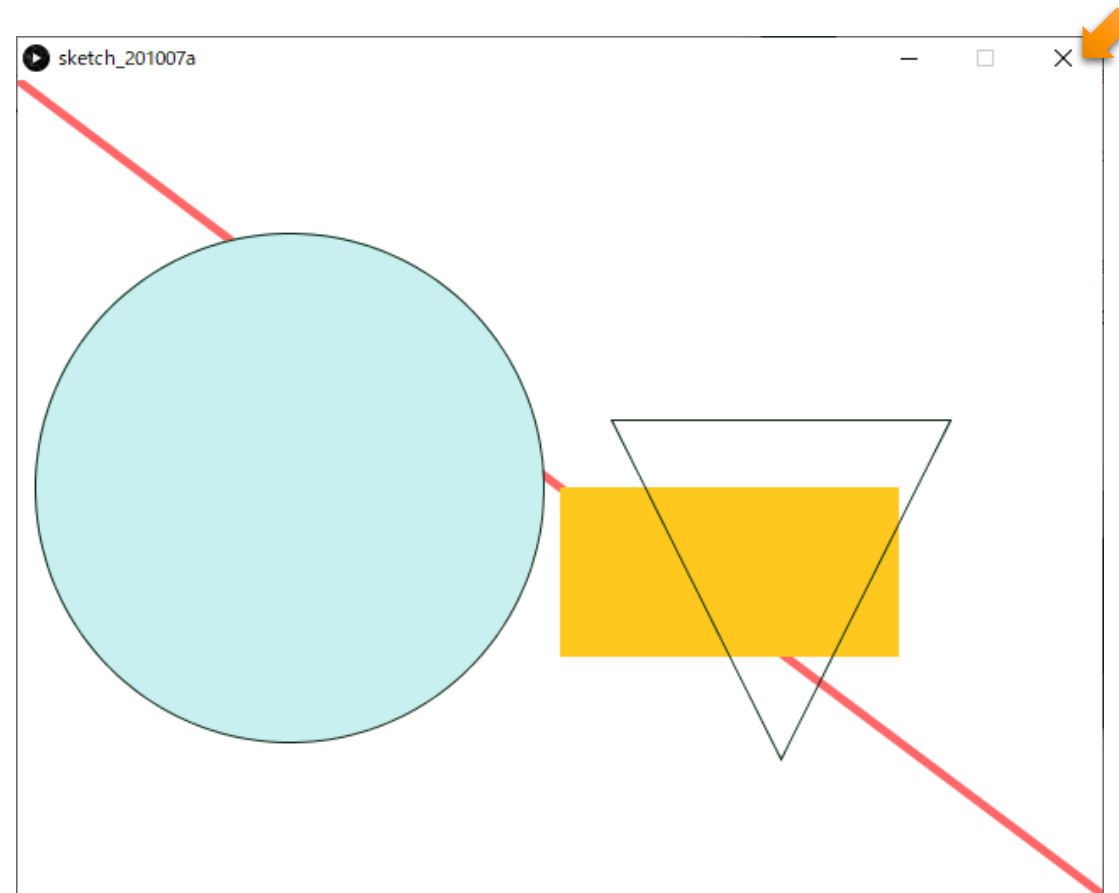
三角形の描画

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
noStroke();  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
stroke(10, 40, 20);  
strokeWeight(1);  
fill(200, 240, 240);  
circle(width / 4, height / 2, 300);  
triangle(350, 200, 450, 400, 550, 200);
```



三角形を塗り潰さないように設定する

```
size(640, 480);  
background(255);  
stroke(255, 102, 103);  
strokeWeight(5);  
line(0, 0, width, height);  
noStroke();  
fill(255, 200, 30);  
rect(width / 2, height / 2, 200, 100);  
stroke(10, 40, 20);  
strokeWeight(1);  
fill(200, 240, 240);  
circle(width / 4, height / 2, 300);  
noFill();  
triangle(350, 200, 450, 400, 550, 200);
```

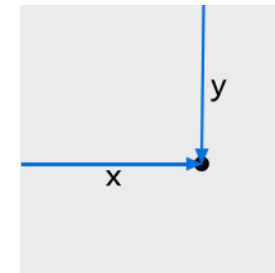


二次元の基本図形

■ `point(x, y)`

- x, y : 点の位置

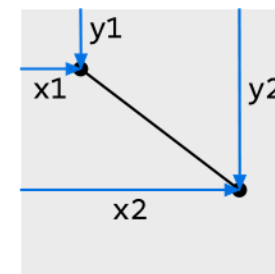
`point(x, y)`



■ `line(x1, y1, x2, y2)`

- $x1, y1$: 線分の始点の位置
- $x2, y2$: 線分の終点の位置

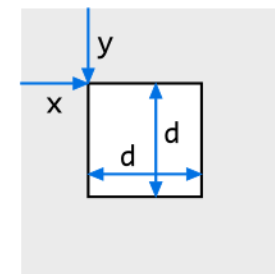
`line(x1, y1, x2, y2)`



■ `square(x, y, d)`

- x, y : 正方形の左上隅の位置
- d : 正方形の1辺の長さ

`square(x, y, d)`



二次元の基本図形

■ `rect(x, y, w, h)`

- x, y : 矩形の左上隅の位置
- w, h : 矩形の幅と高さ

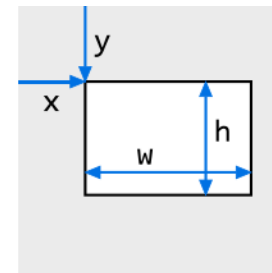
■ `circle(x, y, d)`

- x, y : 円の中心位置
- d : 円の直径

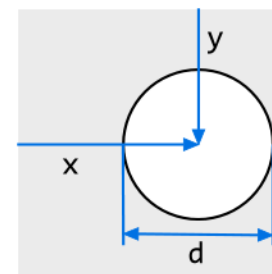
■ `ellipse(x, y, w, h)`

- x, y : 楕円の中心位置
- w, h : 楕円の幅と高さ

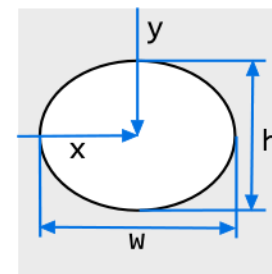
`rect(x, y, w, h)`



`circle(x, y, d)`



`ellipse(x, y, w, h)`



二次元の基本図形

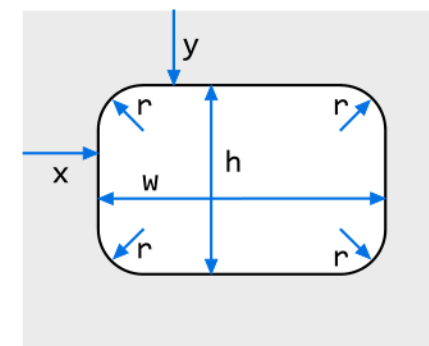
■ `rect(x, y, w, h, r)`

- `x, y` : 矩形の左上隅の位置
- `w, h` : 矩形の幅と高さ
- `r` : 矩形の四隅の半径

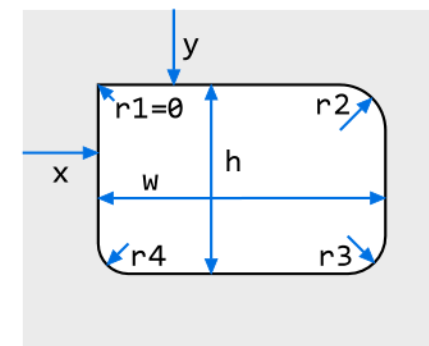
■ `rect(x, y, w, h, r1, r2, r3, r4)`

- `x, y` : 矩形の左上隅の位置
- `w, h` : 矩形の幅と高さ
- `r1` : 左上, `r2` : 右上, `r3` : 右下, `r4` : 左下の角の半径

`rect(x, y, w, h, r)`



`rect(x, y, w, h, r1, r2, r3, r4)`



二次元の基本図形

■ `arc(x, y, w, h, start, stop)`

- `x, y` : 扇形の中心位置
- `w, h` : 扇形の幅と高さ
- `start, stop` : 開始角度と終了角度

■ `arc(x, y, w, h, start, stop, mode)`

- `x, y` : 扇形の中心位置
- `w, h` : 扇形の幅と高さ
- `start, stop` : 開始角度と終了角度
- `mode` : OPEN, CHORD, PIE

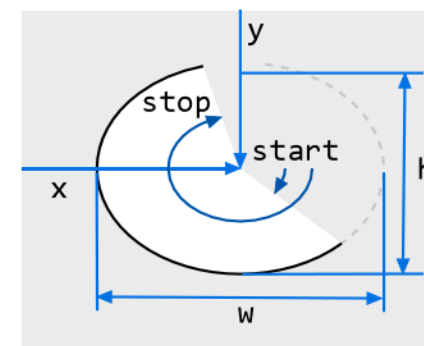
`arc(x, y, w, h, start, stop)`

start, stop の単位は
ラジアン

$\pi \rightarrow \text{PI}$

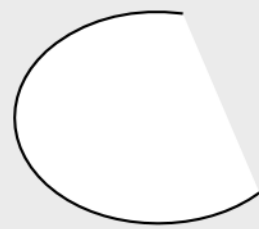
$\pi/2 \rightarrow \text{HALF_PI}$

$2\pi \rightarrow \text{TWO_PI}$



`arc(x, y, a, b, start, stop, mode)`

mode=OPEN



mode=CHORD



mode=PIE

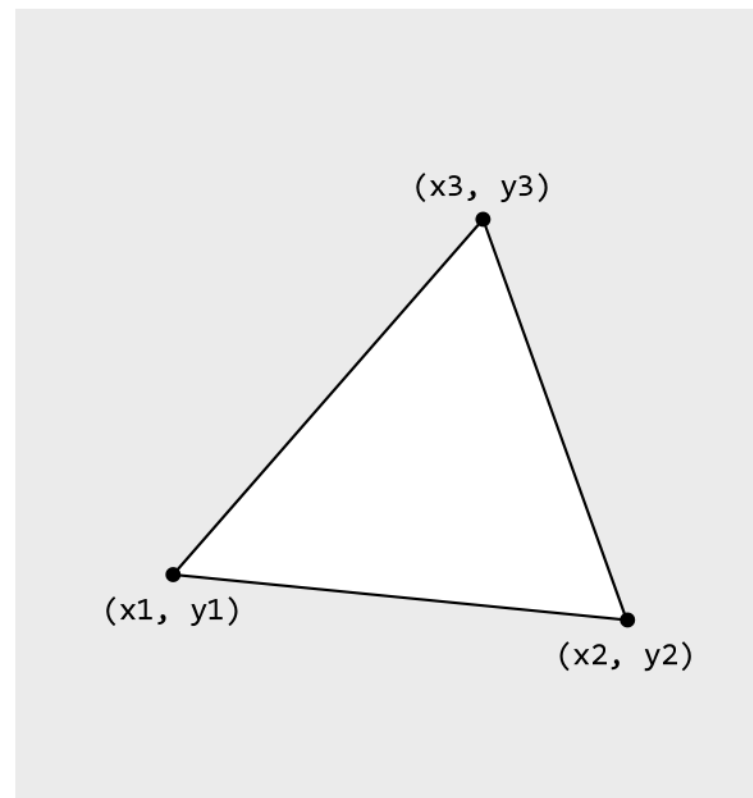


二次元の基本図形

■ `triangle(x1, y1, x2, y2, x3, y3)`

- `x1, y1` : 三角形の1つ目の頂点の位置
- `x2, y2` : 三角形の2つ目の頂点の位置
- `x3, y3` : 三角形の3つ目の頂点の位置

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

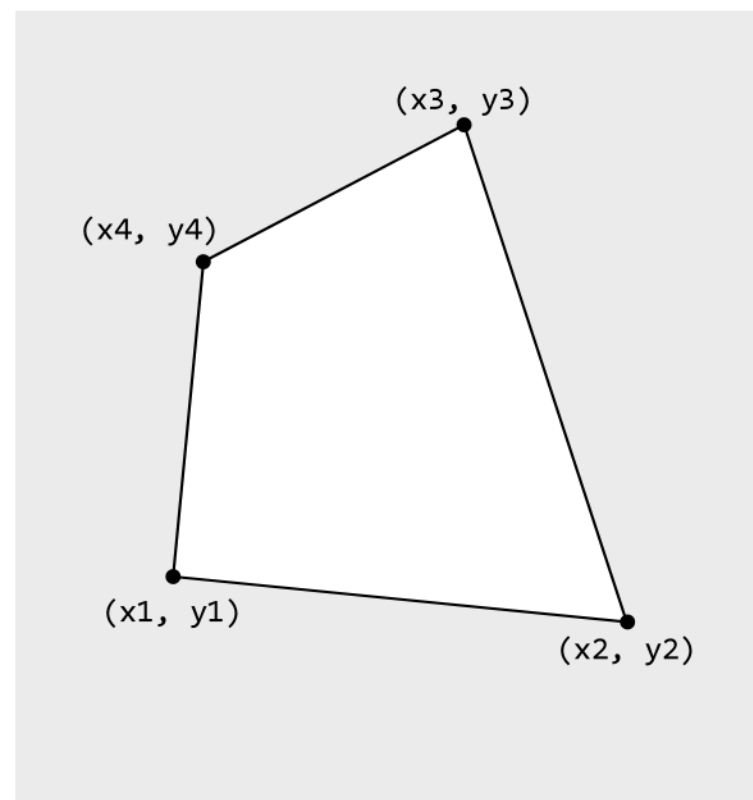


二次元の基本図形

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

- `x1, y1` : 四角形の1つ目の頂点の位置
- `x2, y2` : 四角形の2つ目の頂点の位置
- `x3, y3` : 四角形の3つ目の頂点の位置
- `x4, y4` : 四角形の4つ目の頂点の位置

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



各種設定

■ stroke(gray)

- gray : 点や線の色 (白黒)

■ stroke(gray, alpha)

- gray : 点や線の色 (白黒)
- alpha : 点や線の不透明度 (0~255)

■ stroke(r, g, b)

- r, g, b : 点や線の色

■ stroke(r, g, b, alpha)

- r, g, b : 点や線の色
- alpha : 点や線の不透明度 (0~255)

■ noStroke()

- 点や線を描かない

■ fill(gray)

- gray : 塗りつぶす明るさ (白黒)

■ fill(gray, alpha)

- gray : 塗りつぶす明るさ (白黒)
- alpha : 塗りつぶす不透明度 (0~255)

■ fill(r, g, b)

- r, g, b : 塗りつぶす色

■ fill(r, g, b, alpha)

- r, g, b : 塗りつぶす色
- alpha : 塗りつぶす不透明度 (0~255)

■ noFill();

- 塗りつぶさない

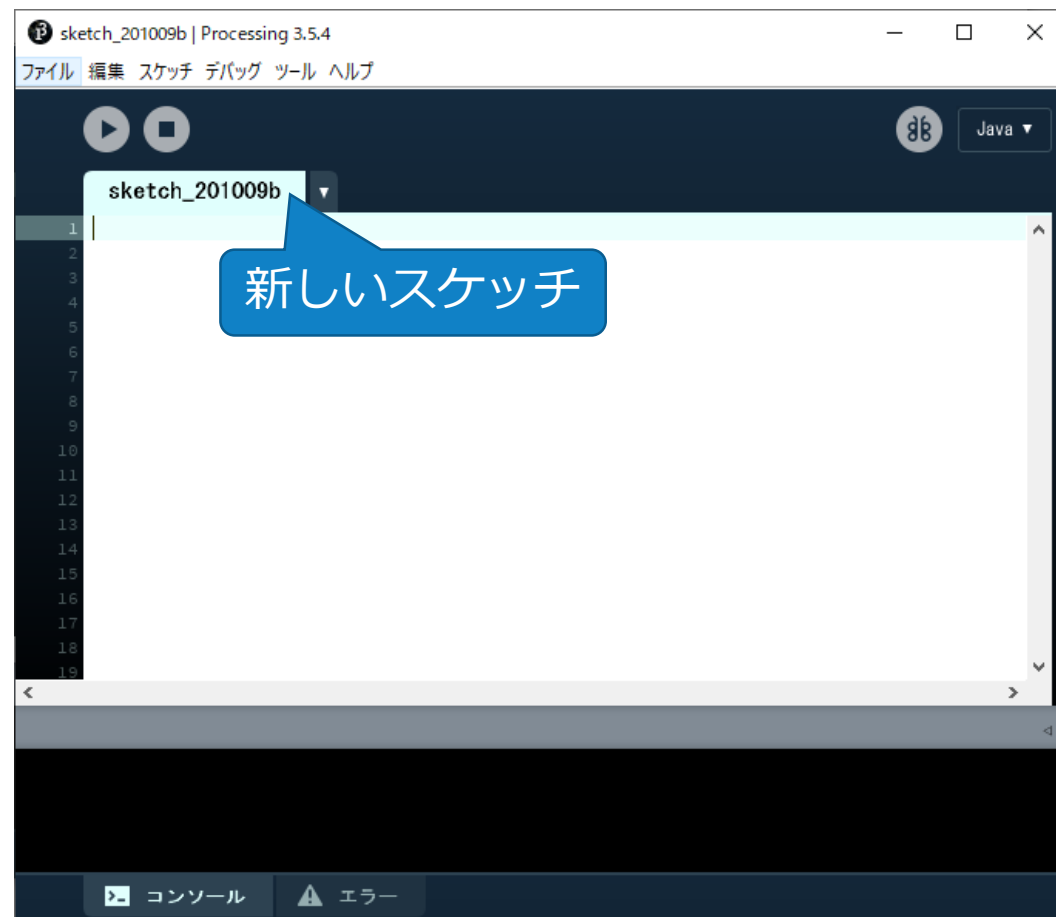
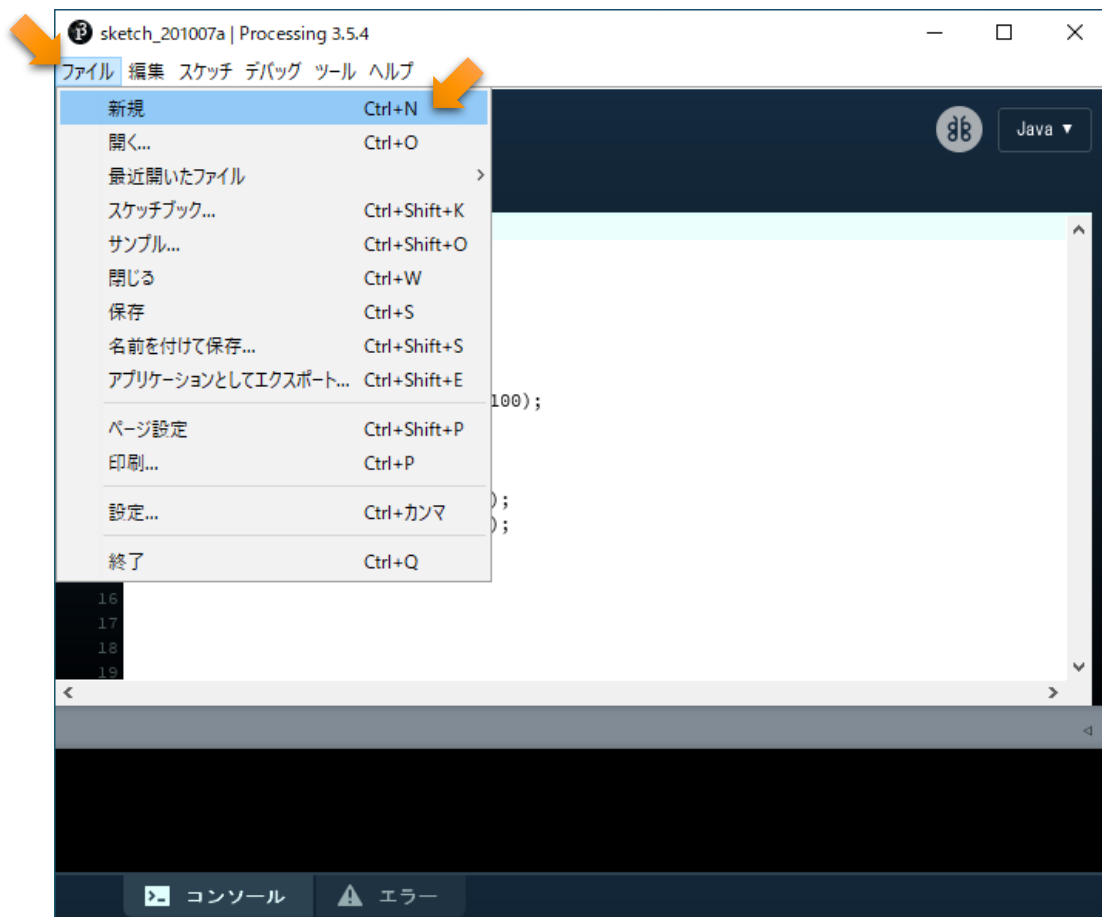
各種設定

- `background(gray)`
 - `gray` : 背景の明るさ（白黒）
- `background(gray, alpha);`
 - `gray` : 背景の明るさ（白黒）
 - `alpha` : 背景の不透明度
- `background(r, g, b)`
 - `r, g, b` : 背景の色
- `background(r, g, b, a)`
 - `r, g, b` : 背景の色
 - `a` : 背景の不透明度

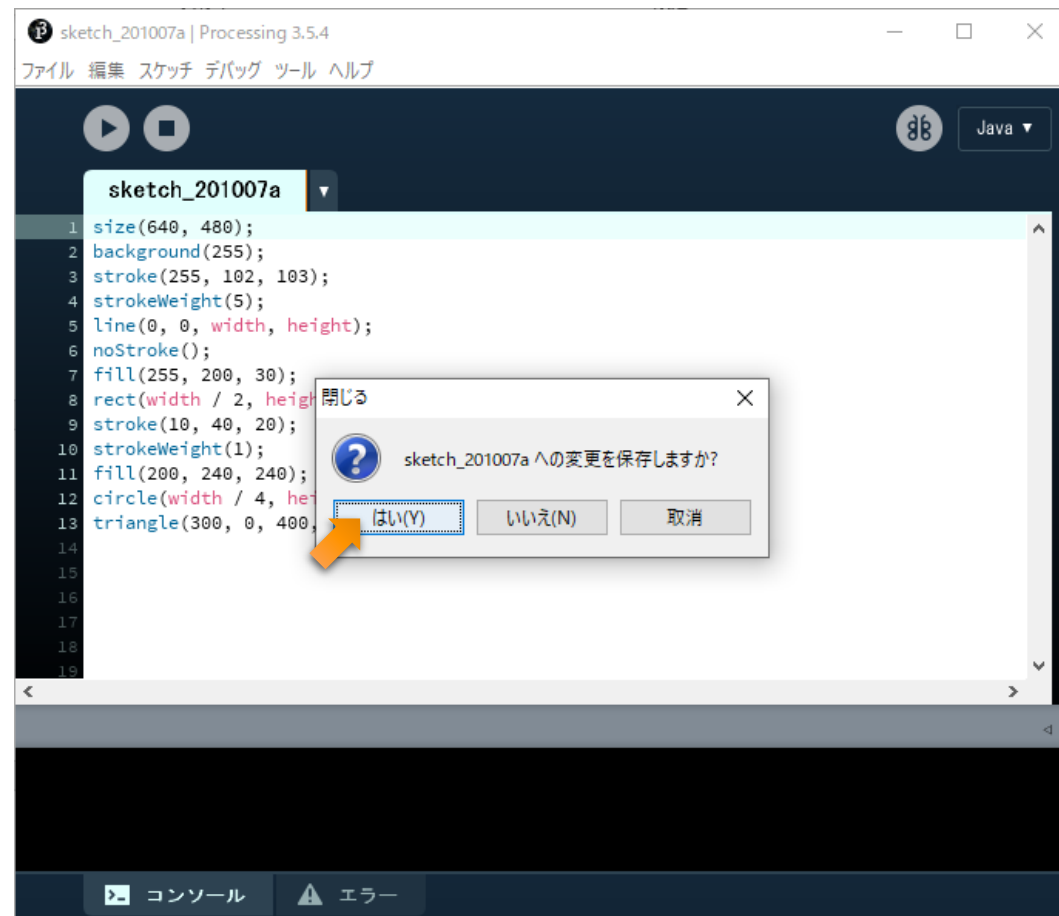
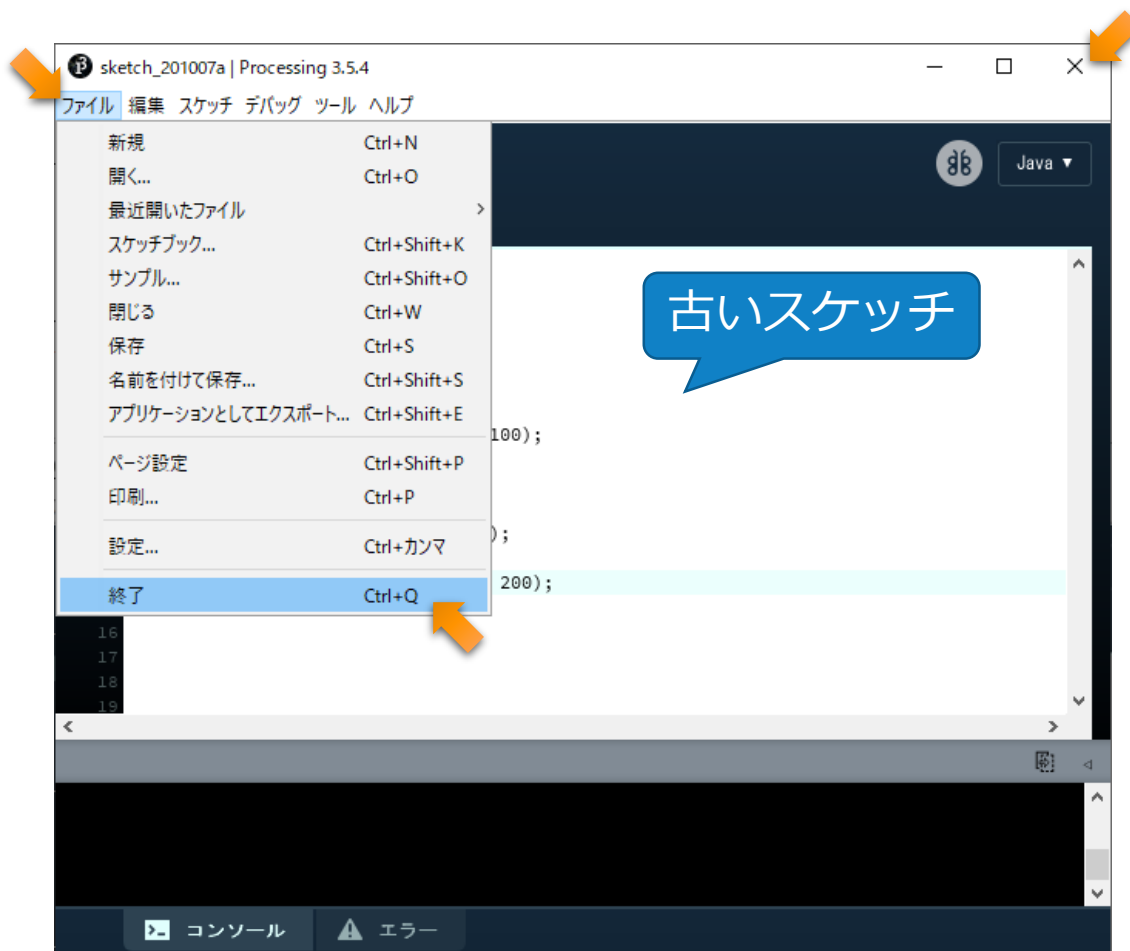
課題 1

プログラムで図形を描く

新しいスケッチを作成する

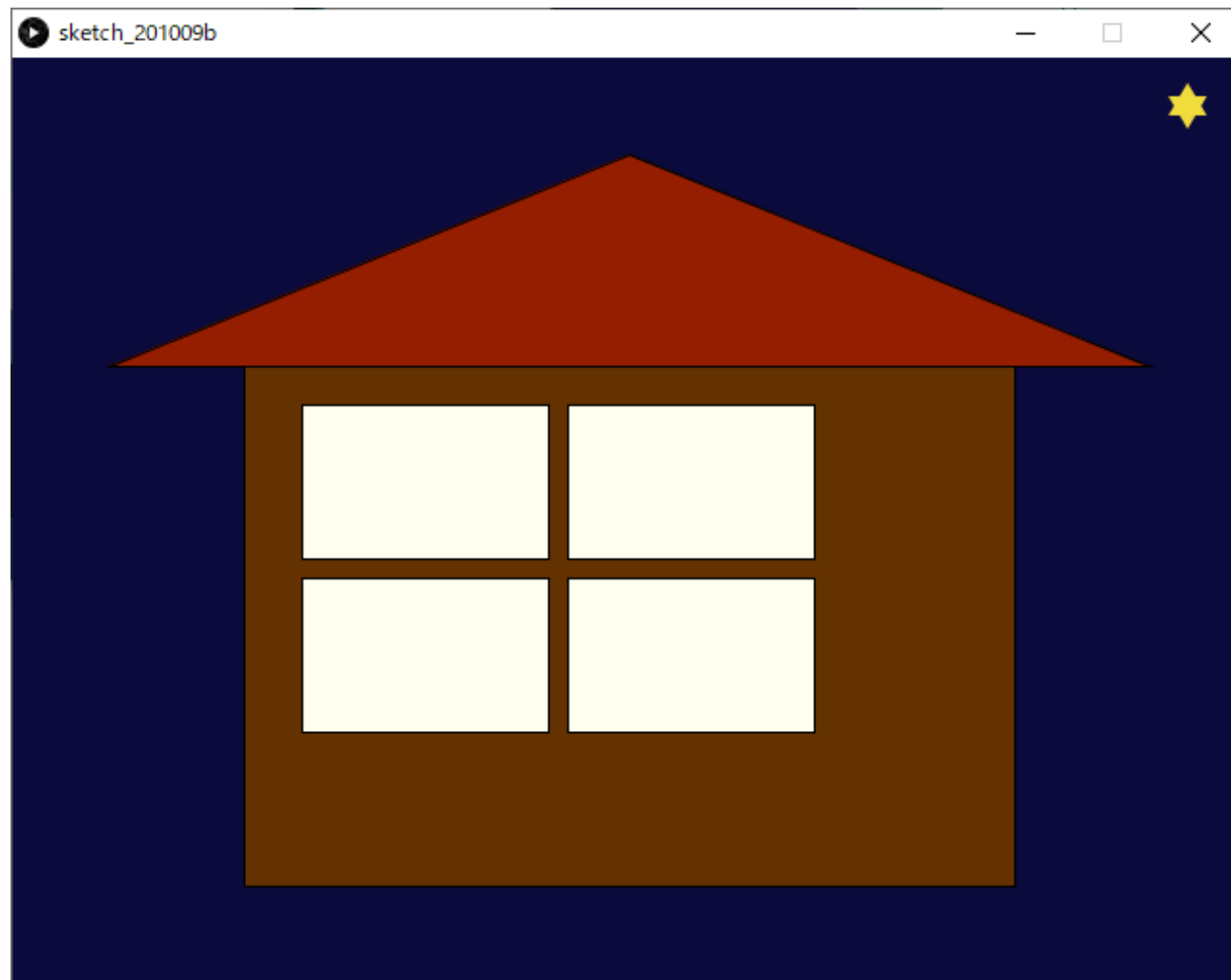


今まで使っていたスケッチは終了する



こんな感じの絵を
描いてください

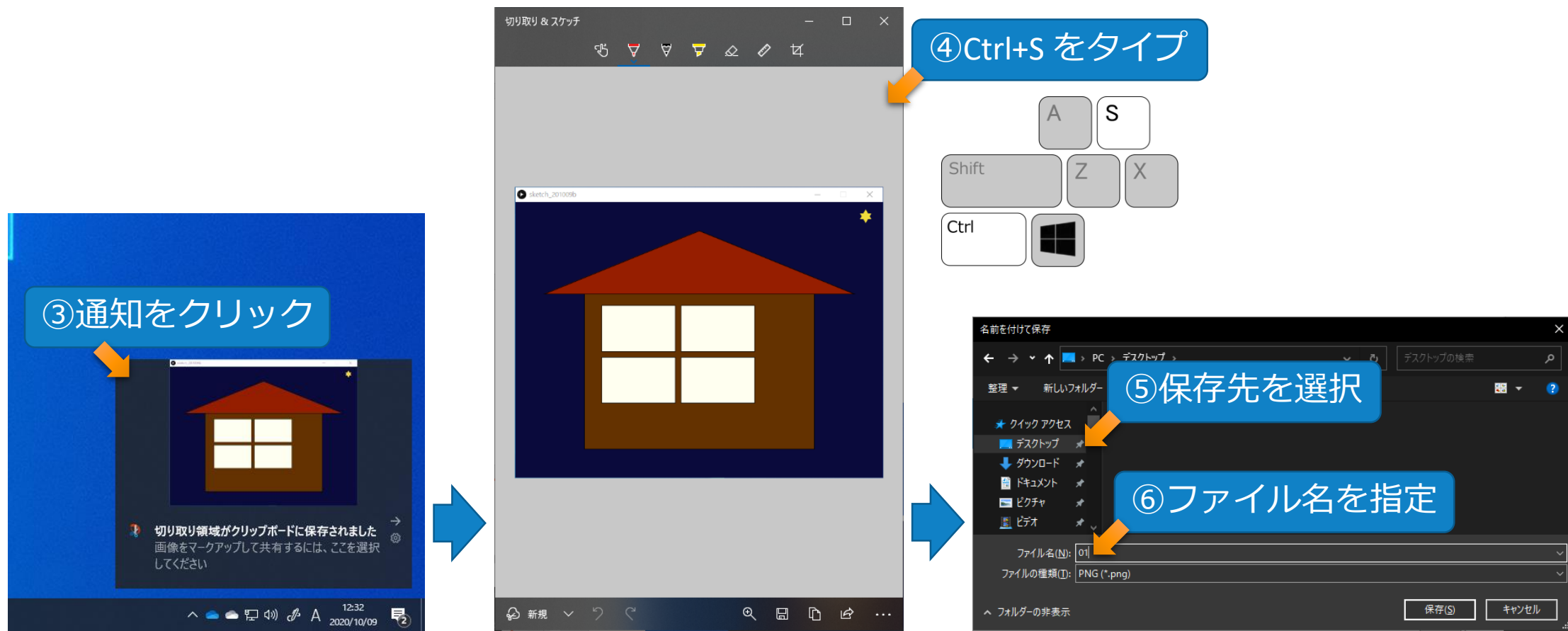
形や色はだいたいこれに似せてください



表示図形のスクリーンショットを撮る



“01.png” というファイル名で保存する



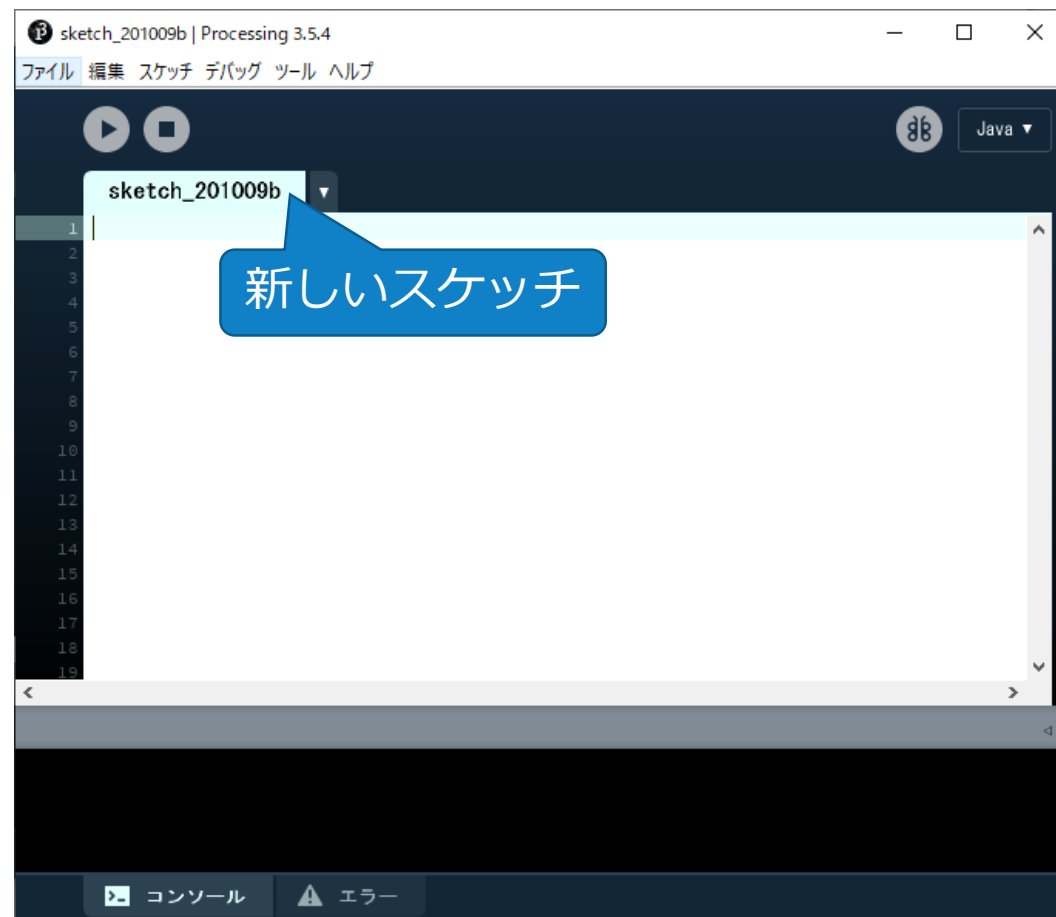
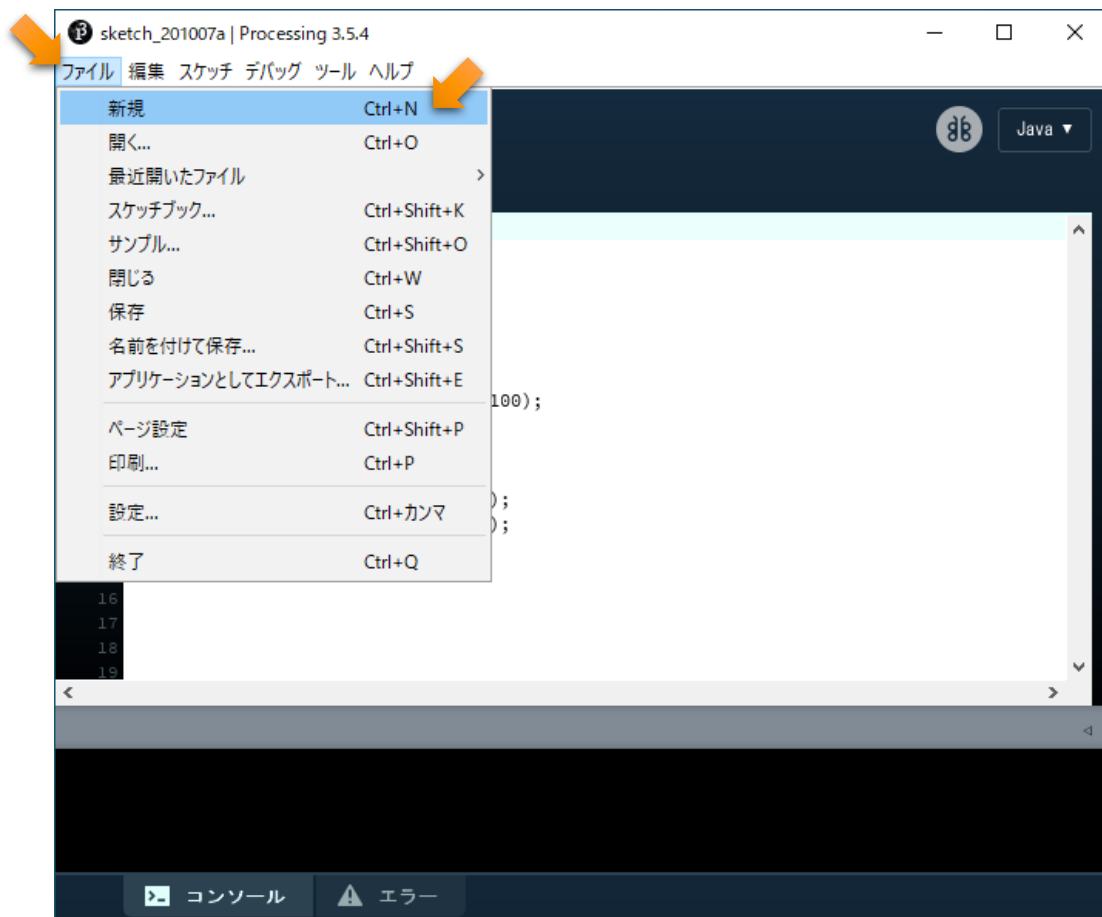
スクリーンショットをアップロードする

- 実行結果が表示されているウィンドウのスクリーンショットを **01.png** というファイル名で保存して、Moodle の「スクリーンショットのアップロード」からアップロードしてください。

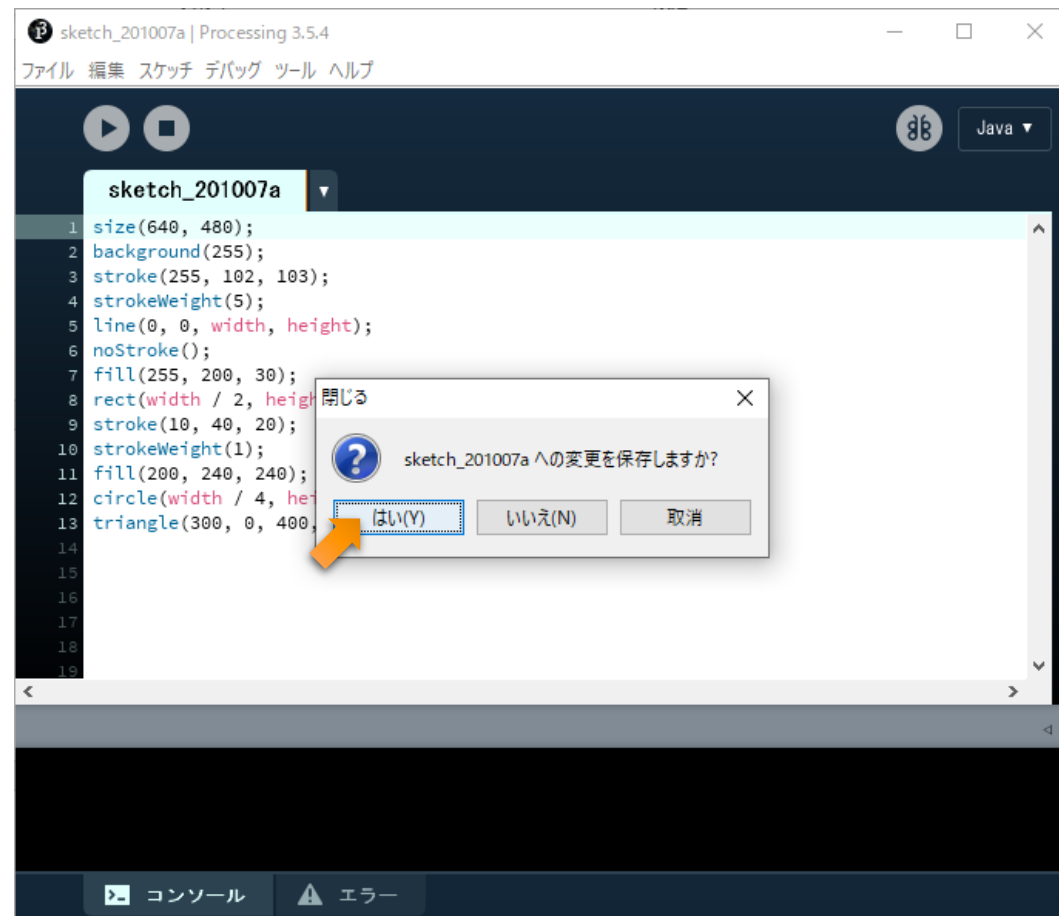
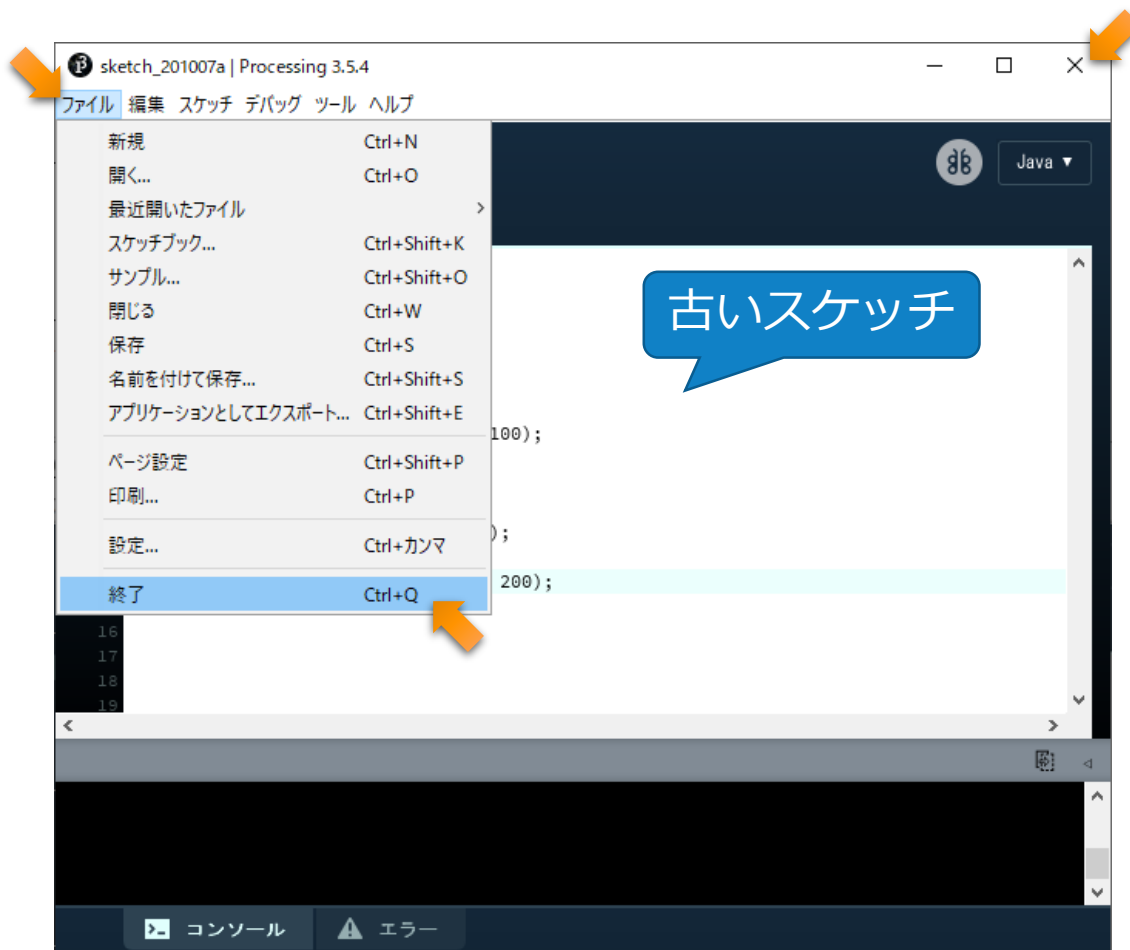
マウス操作

マウスで絵を描く

新しいスケッチを作成する

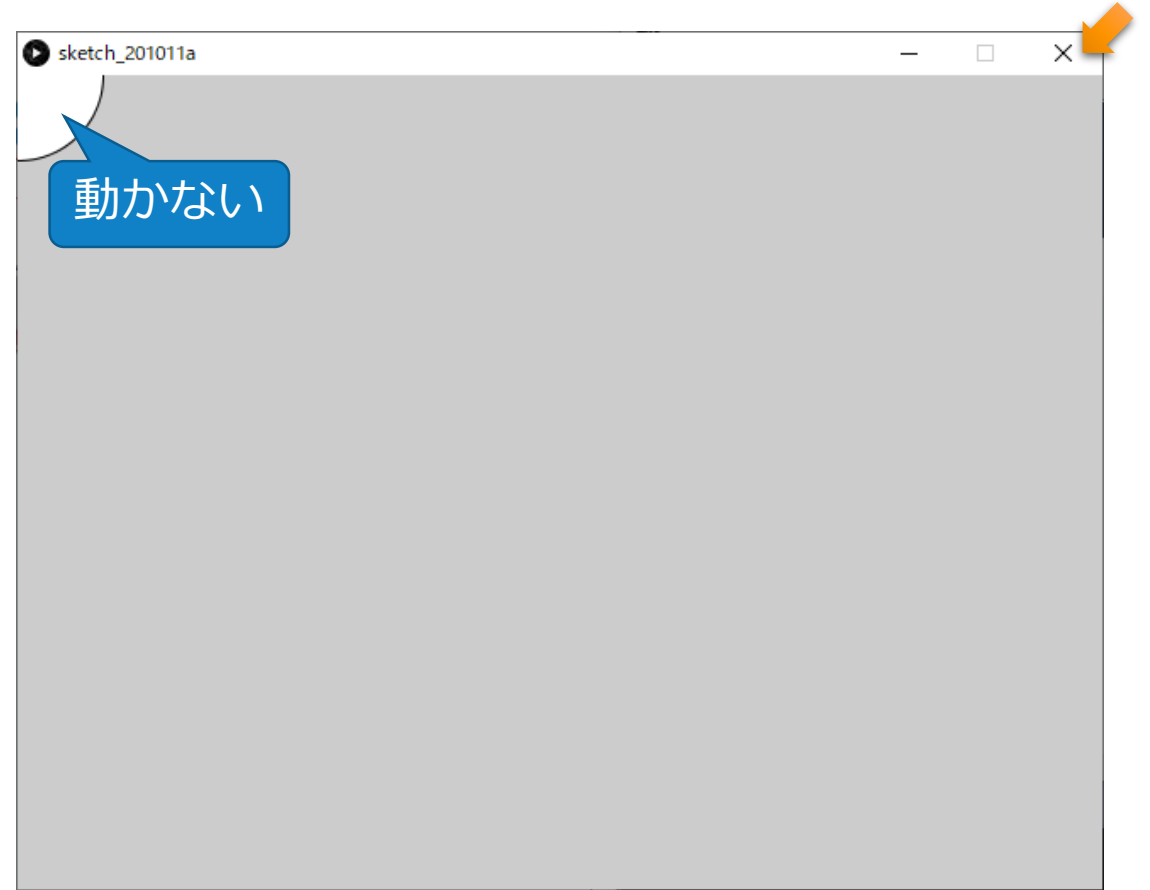


今まで使っていたスケッチは終了する



mouseX, mouseY はマウスカーソルの位置

```
size(640, 480);  
circle(mouseX, mouseY, 100);
```



アニメーションに対応する

```
void setup() {  
  size(640, 480);  
}  
  
void draw() {  
  circle(mouseX, mouseY, 100);  
}
```



setup() と draw()

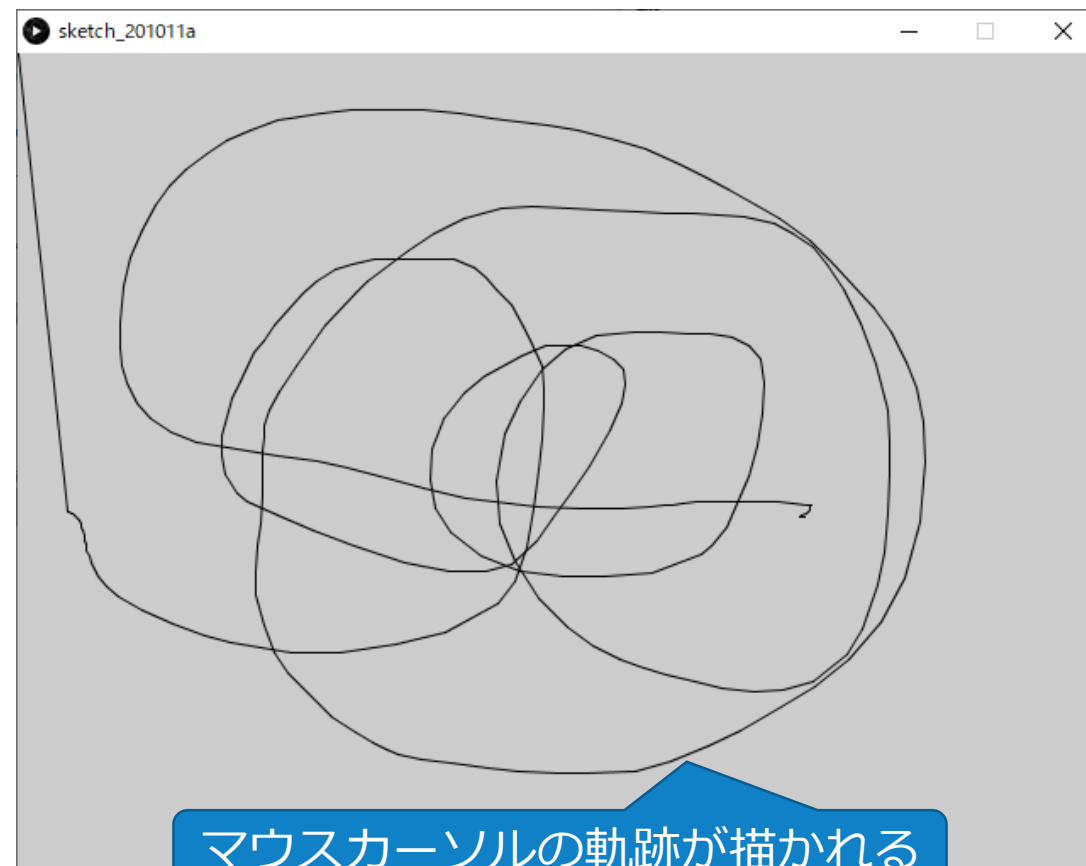
```
void setup() {  
    // ここに書いたことはプログラムを動かしたときに  
    // 一度だけ実行される  
}  
  
void draw() {  
    // ここに書いたことは setup() を実行したあと  
    // 繰り返し実行される  
}
```

マウスカーソルの軌跡を描く

```
void setup() {  
  size(640, 480);  
}  
  
void draw() {  
  circle(mouseX, mouseY, 100);  
  line(pmouseX, pmouseY, mouseX, mouseY);  
}
```

削除

追加



マウスカーソルの軌跡が描かれる

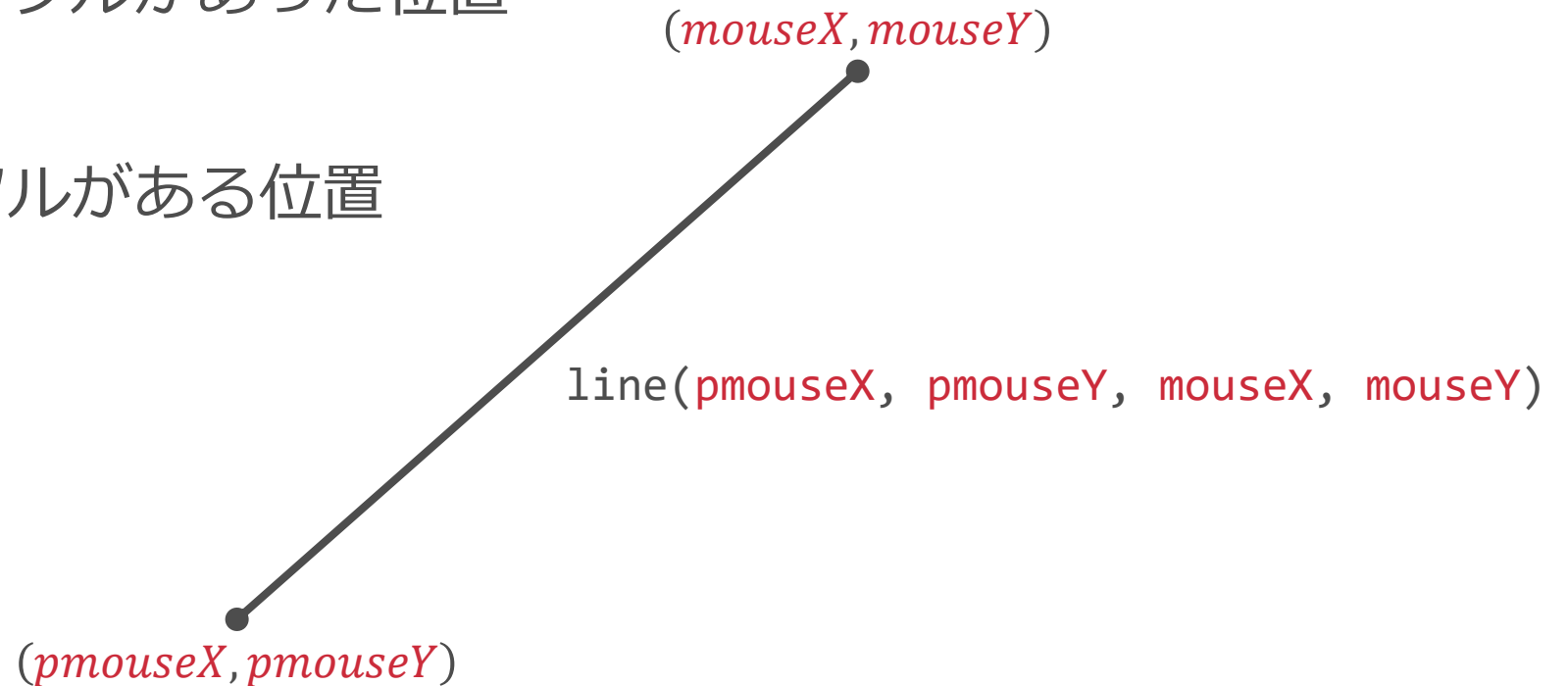
`line(pmouseX, pmouseY, mouseX, mouseY)`

■ `pmouseX, pmouseY`

- 直前にマウスカーソルがあった位置

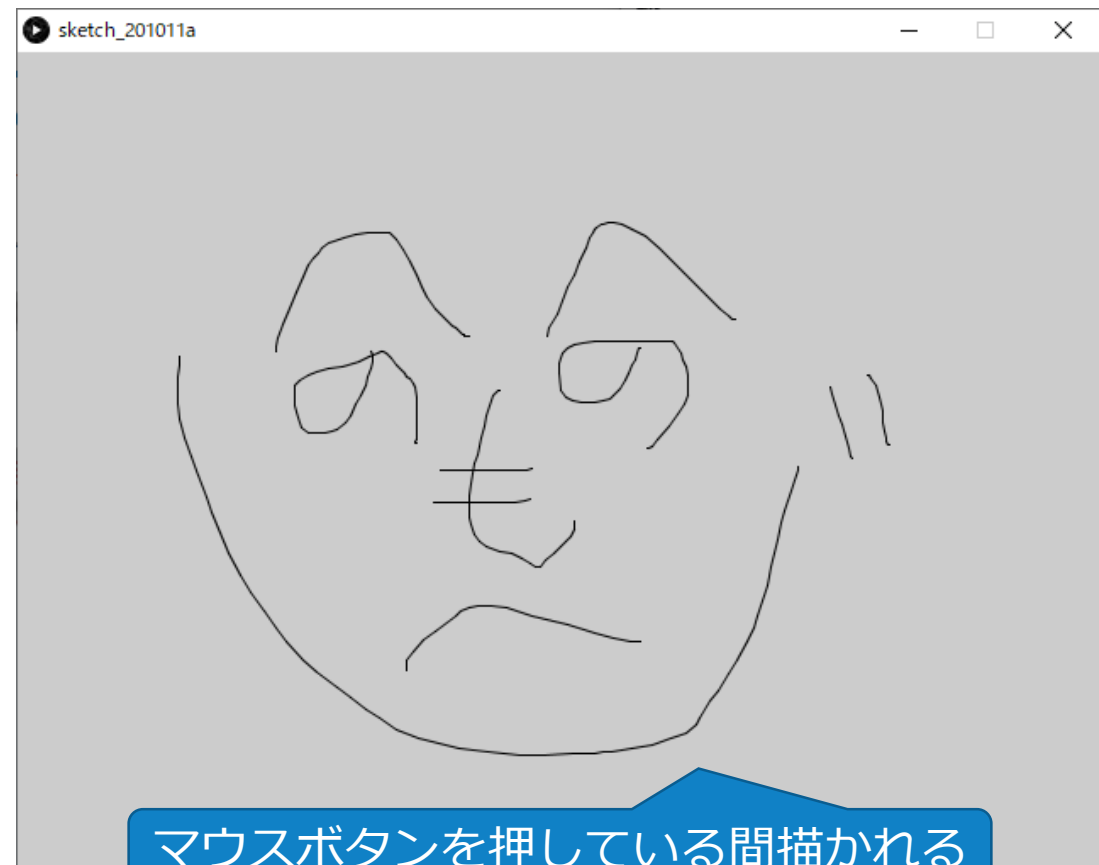
■ `mouseX, mouseY`

- 現在マウスカーソルがある位置



マウスボタンを押している間だけ描く

```
void setup() {  
  size(640, 480);  
}  
  
void draw() {  
  if (mousePressed) {  
    line(pmouseX, pmouseY, mouseX, mouseY);  
  }  
}
```



if 文

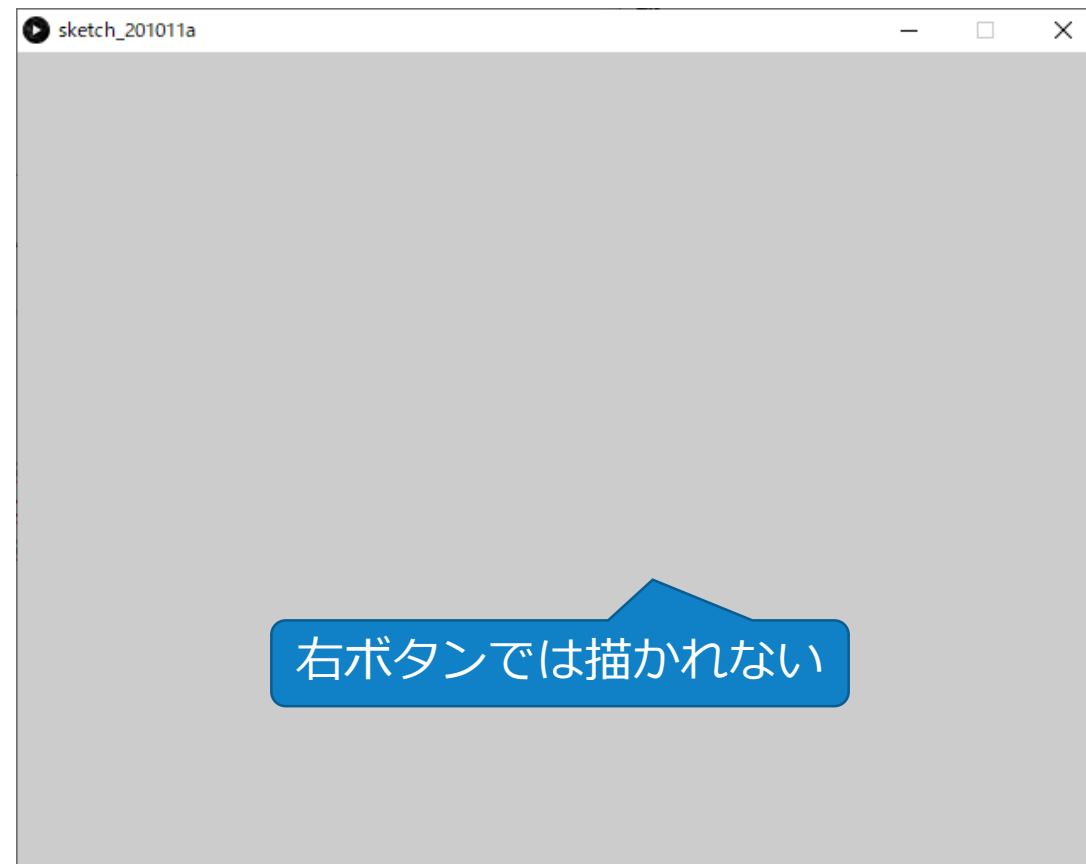
```
if (条件) {  
    // ここに書いたことは条件が真の時だけ実行される  
}
```

■mousePressed

- マウスボタンが押されているとき真になる

左ボタンを押している間だけ描く

```
void setup() {  
  size(640, 480);  
}  
  
void draw() {  
  if (mousePressed) {  
    if (mouseButton == LEFT) {  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
  }  
}
```



比較演算子

■ 値1 == 値2

- 値1 と 値2 が等しければ
値1 == 値2 は真

■ 値1 != 値2

- 値1 と 値2 が等しくなければ
値1 != 値2 は真

■ 値1 < 値2

- 値1 が 値2 より小さければ
値1 < 値2 は真

■ 値1 > 値2

- 値1 が 値2 より大きければ
値1 > 値2 は真

■ 値1 <= 値2

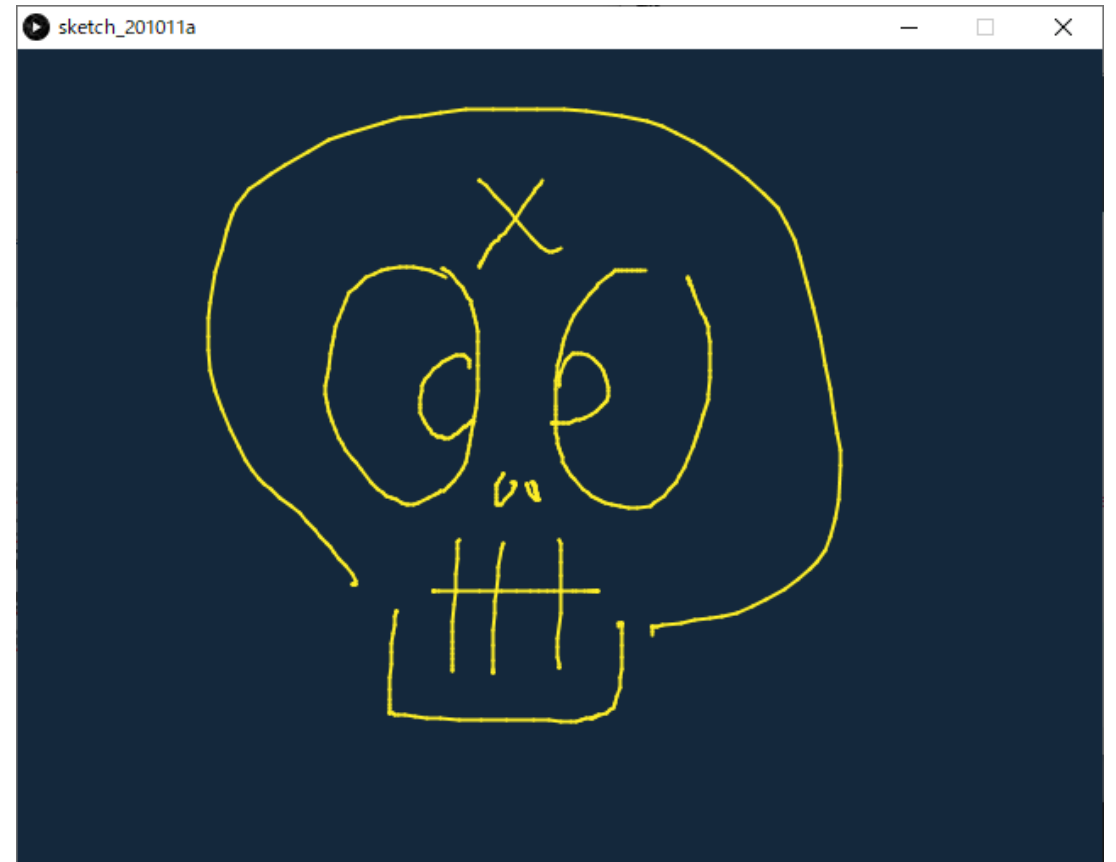
- 値1 が 値2 以下なら
値1 <= 値2 は真

■ 値1 >= 値2

- 値1 が 値2 以上なら
値1 >= 値2 は真

背景色と線の色・太さを指定する

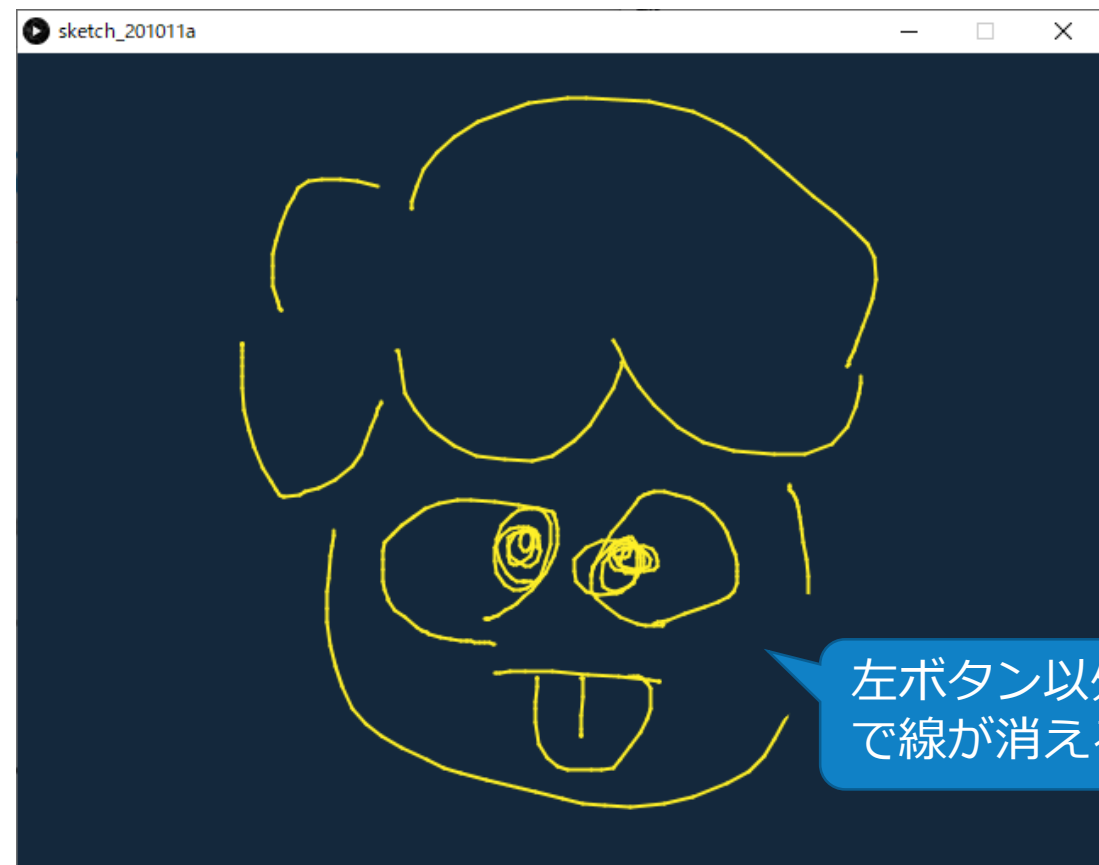
```
void setup() {  
  size(640, 480);  
  background(20, 40, 60);  
}  
  
void draw() {  
  if (mousePressed) {  
    if (mouseButton == LEFT) {  
      stroke(255, 240, 40);  
      strokeWeight(2);  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
  }  
}
```



左ボタン以外で消す

```
void setup() {  
  size(640, 480);  
  background(20, 40, 60);  
}  
  
void draw() {  
  if (mousePressed) {  
    if (mouseButton == LEFT) {  
      stroke(255, 240, 40);  
      strokeWeight(2);  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
    else {  
      stroke(20, 40, 60);  
      strokeWeight(20);  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
  }  
}
```

背景と同じ色



左ボタン以外
で線が消える

if ～ else 文

```
if (条件) {  
    // ここに書いたことは条件が真の時だけ実行される  
}  
else {  
    // ここに書いたことは条件が偽の時だけ実行される  
}
```

右ボタンで消す

```
void setup() {  
  size(640, 480);  
  background(20, 40, 60);  
}  
  
void draw() {  
  if (mousePressed) {  
    if (mouseButton == LEFT) {  
      stroke(255, 240, 40);  
      strokeWeight(2);  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
    else if (mouseButton == RIGHT) {  
      stroke(20, 40, 60);  
      strokeWeight(20);  
      line(pmouseX, pmouseY, mouseX, mouseY);  
    }  
  }  
}
```



if ～ else if ～ else 文

```
if (条件1) {  
    // ここに書いたことは条件1が真の時だけ実行される  
}  
else if (条件2) {  
    // ここに書いたことは条件2が真の時だけ実行される  
}  
else {  
    // ここに書いたことは条件1も条件2も偽の時だけ実行される  
}
```


課題 2

マウスで図形を描く

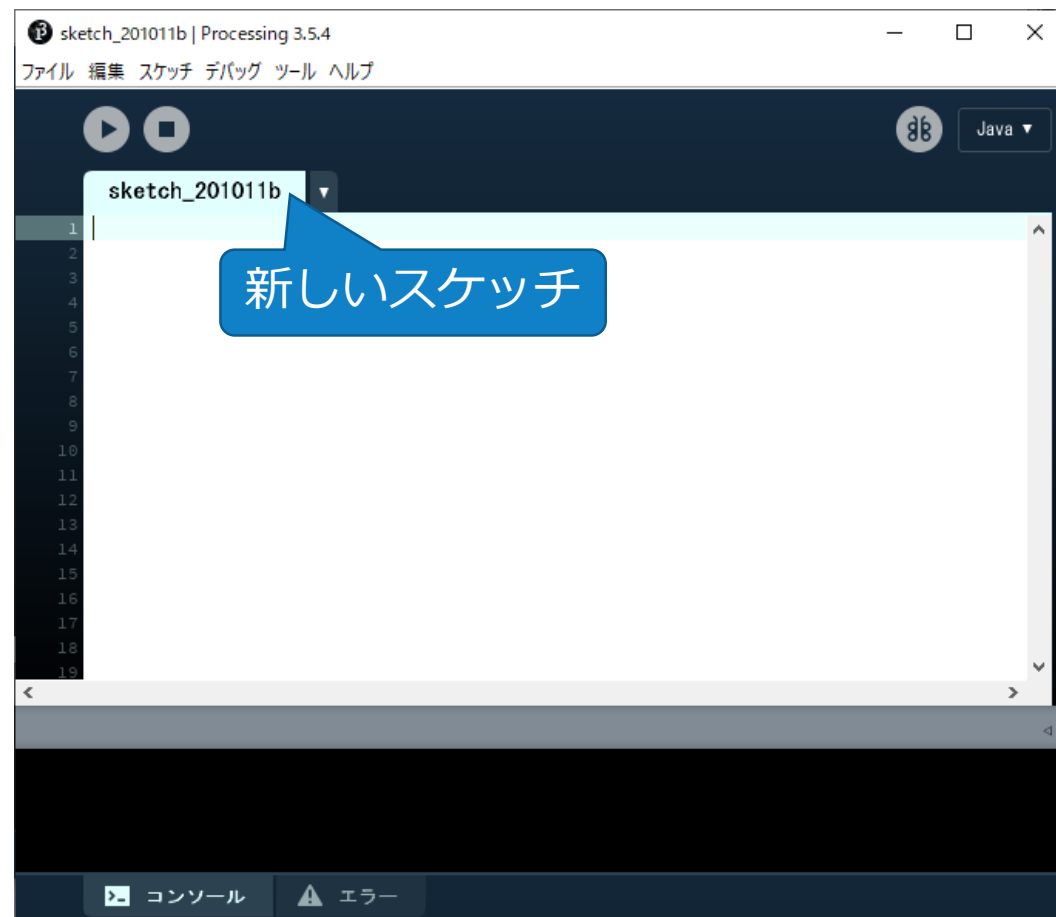
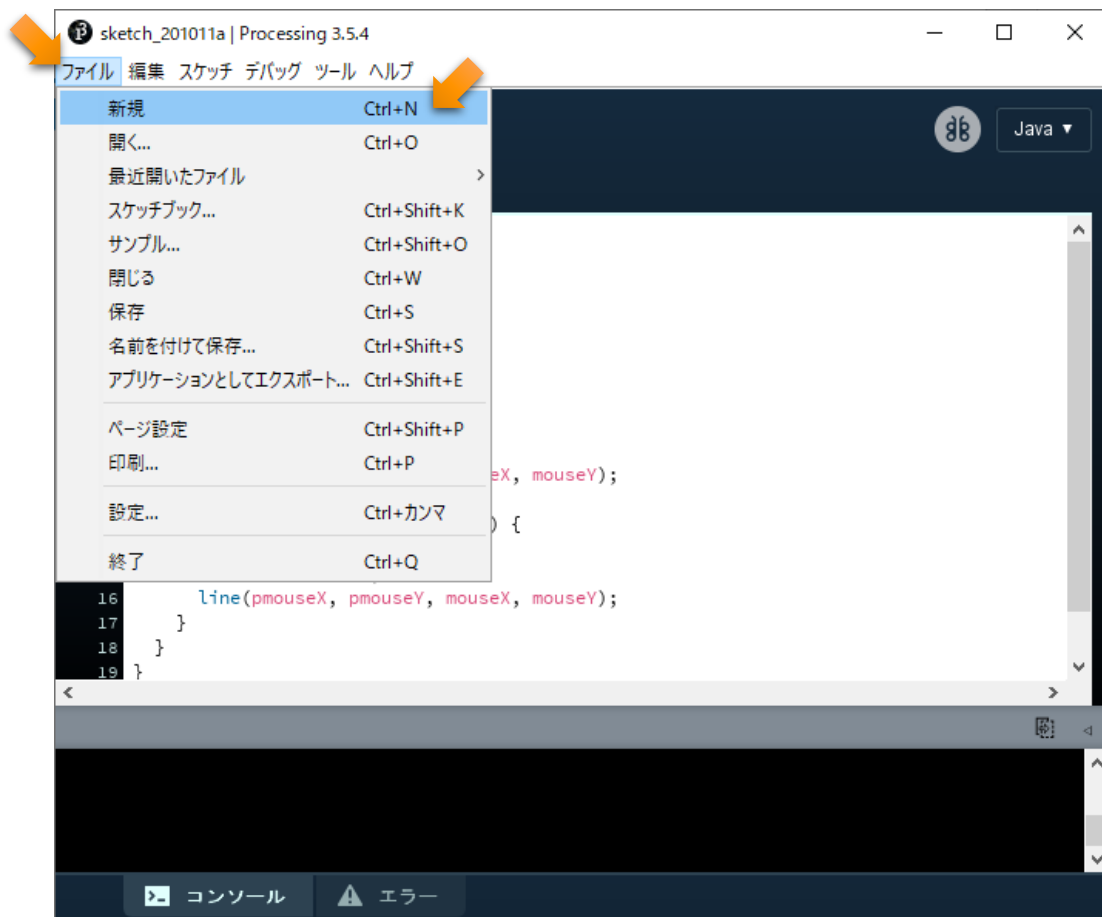
スクリーンショットをアップロードする

- 実行結果のウィンドウにマウスで何か適当な絵を描き、そのスクリーンショットを撮って **02.png** というファイル名で保存して、Moodle の「スクリーンショットのアップロード」からアップロードしてください。

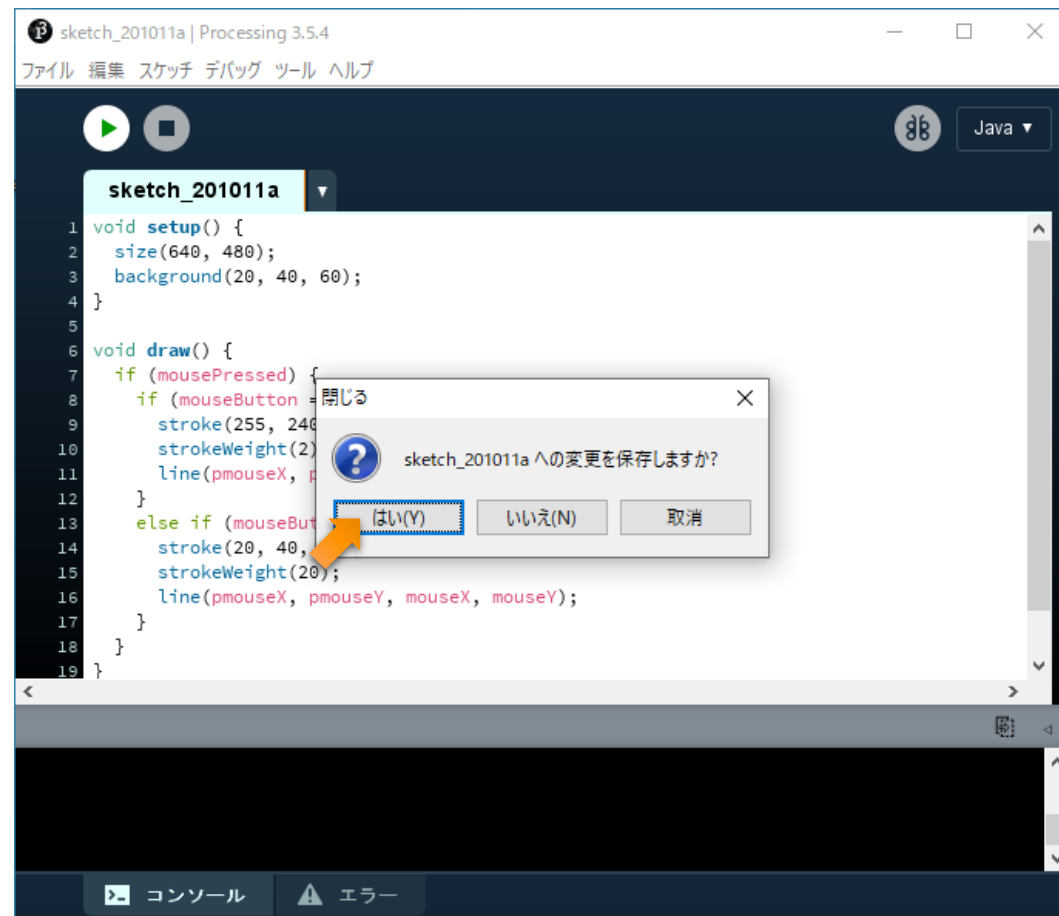
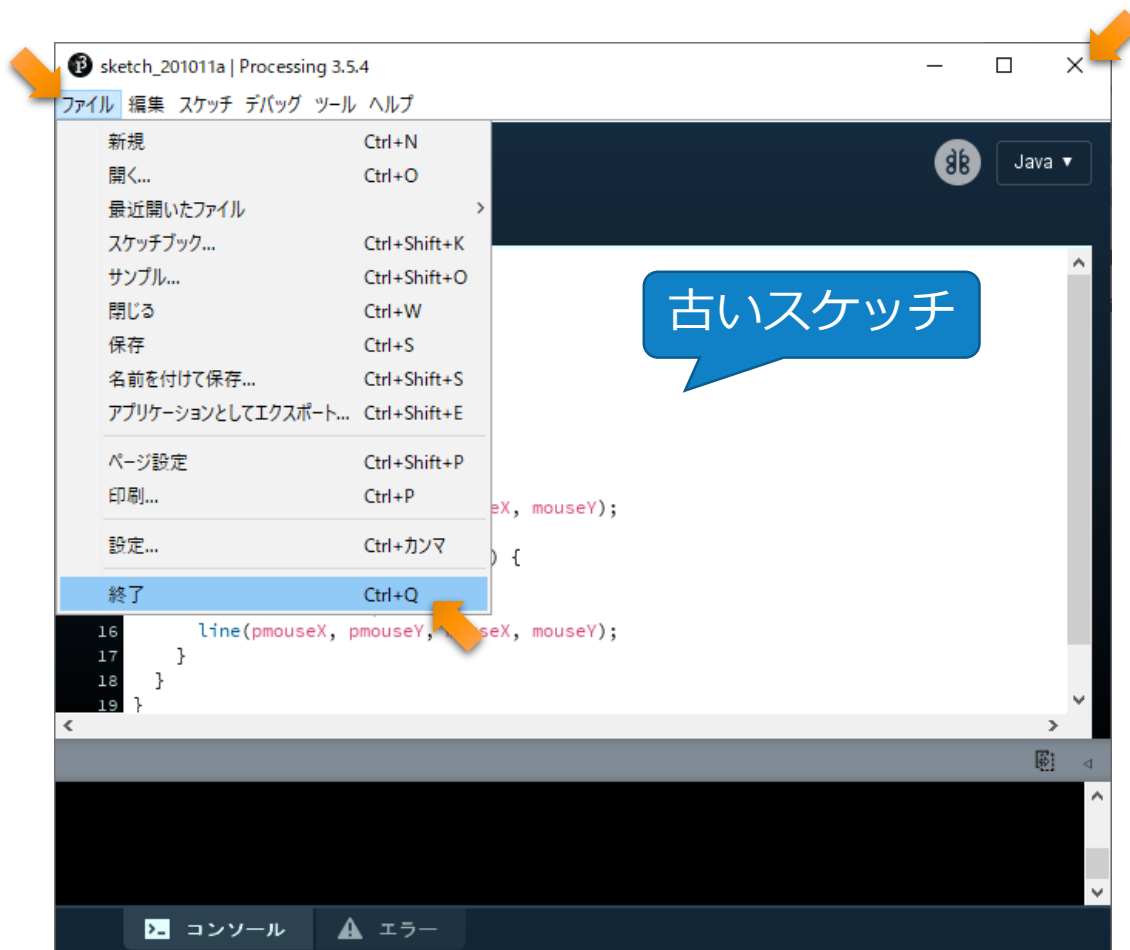
アニメーション

図形を動かす

新しいスケッチを作成する

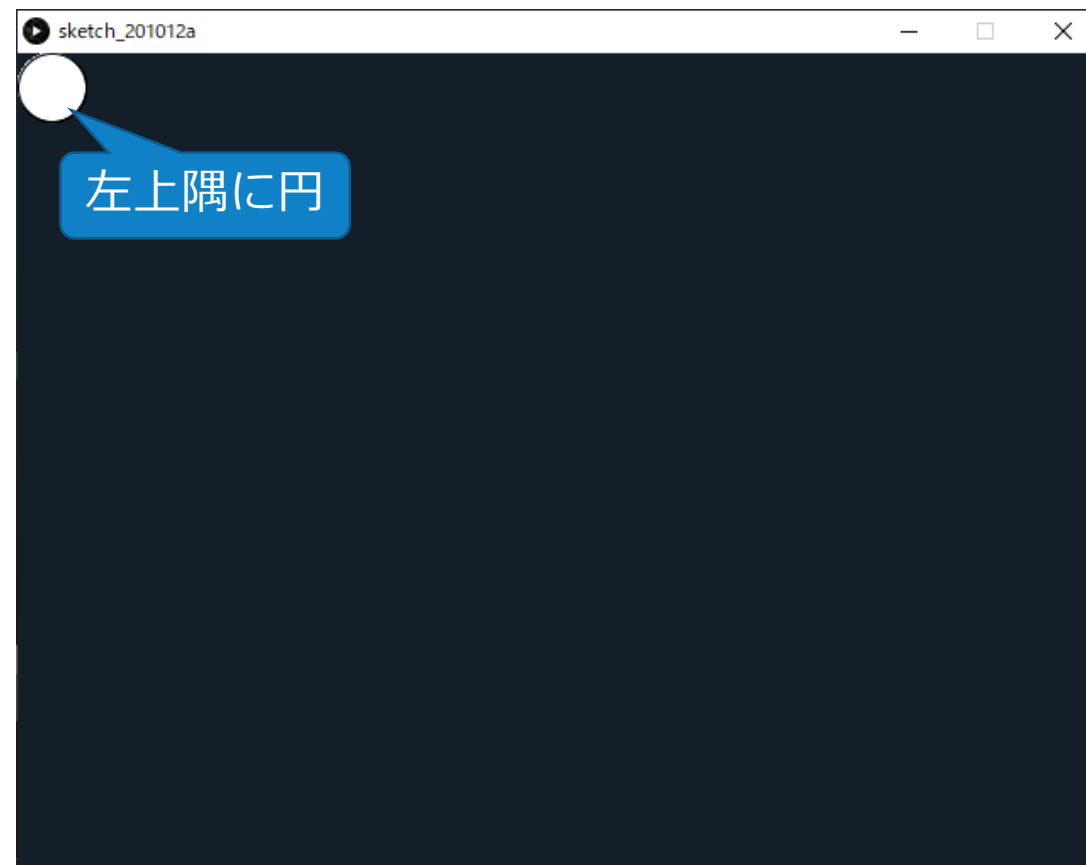


今まで使っていたスケッチは終了する



左上隅に円を描く

```
void setup() {  
  size(640, 480);  
  background(20, 30, 40);  
}  
  
void draw() {  
  circle(20, 20, 40);  
}
```



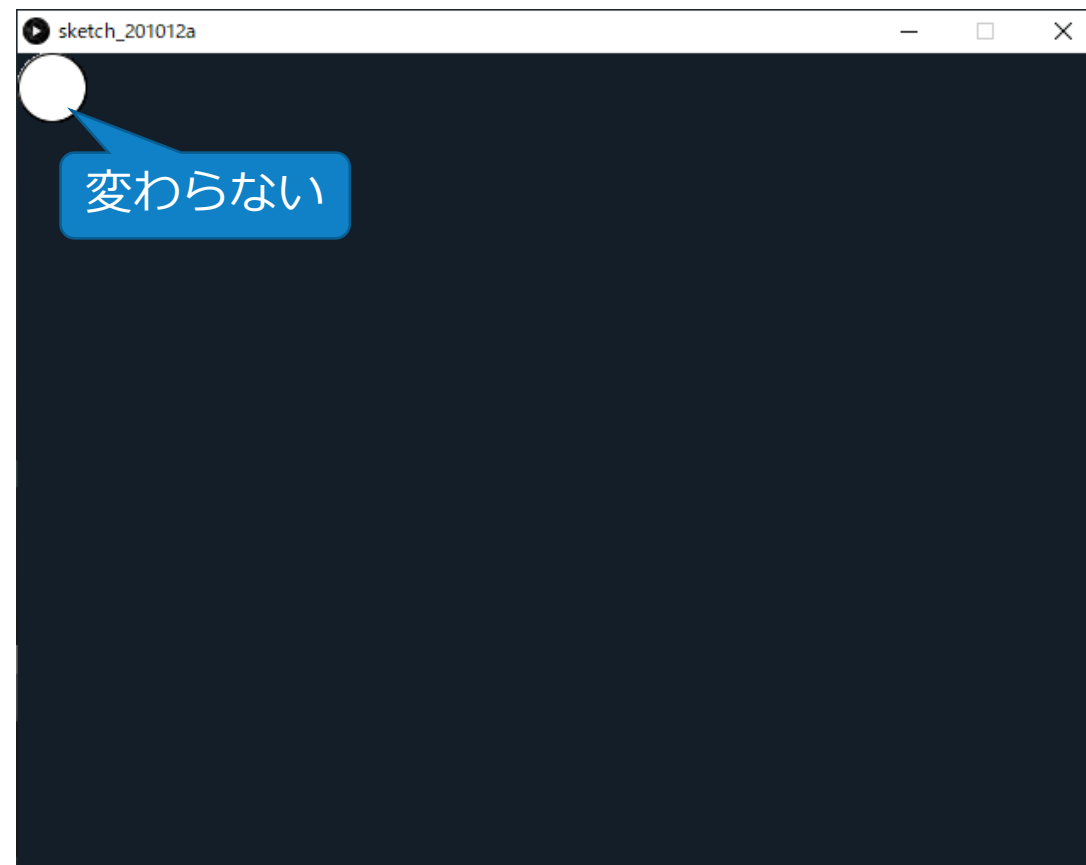
位置を変数で指定する

```
float position_x, position_y;

void setup() {
  size(640, 480);
  background(20, 30, 40);
  position_x = 20;
  position_y = 20;
}

void draw() {
  circle(position_x, position_y, 40);
}
```

↑ 20 ↑ 20



変数

- データに名前を付けて覚えておくもの

```
float position_x, position_y;
```

- `float` 型の二つの変数 `position_x` と `position_y` を準備する (変数宣言)

```
position_x = 20;
```

```
position_y = 20;
```

- `position_x` に 20、`position_y` に 20 を格納する (代入)

```
circle(position_x, position_y, 40);
```

- `position_x` を `circle` の `x` 座標値、`position_y` を `circle` の `y` 座標値に使う

データ型

■float

- 実数型、浮動小数点
- 小数点以下が表現できる
- 非常に大きな数や小さな数が表現できる
- 数値計算に使う

■int

- 整数型、固定小数点
- 整数のみ、小数点以下は表現できない
- 番号や数を数えることに使う

円を動かす

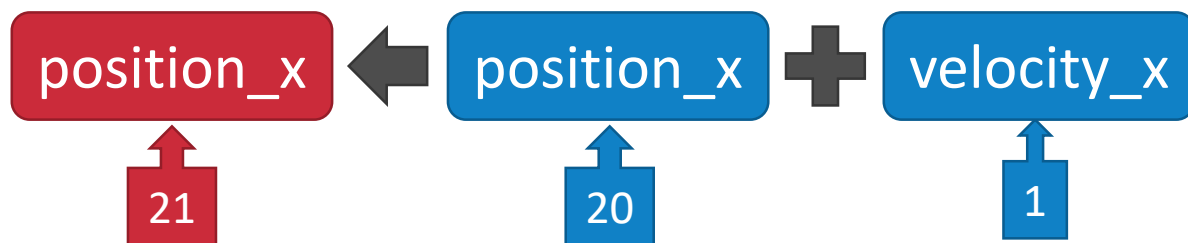
```
float position_x, position_y;  
float velocity_x, velocity_y;  
  
void setup() {  
  size(640, 480);  
  background(20, 30, 40);  
  position_x = 20;  
  position_y = 20;  
  velocity_x = 1;  
  velocity_y = 0;  
}  
  
void draw() {  
  circle(position_x, position_y, 40);  
  position_x += velocity_x;  
  position_y += velocity_y;  
}
```



`+=` 演算子

`position_x += velocity_x;`

- `position_x` に `velocity_x` の内容を加算する
 - `position_x` には 20 が入っている
 - `velocity_x` には 1 が入っている



`a += b`

■ `a ← a + b`

`a -= b`

■ `a ← a - b`

`a *= b`

■ `a ← a * b`

`a /= b`

■ `a ← a / b`

円を動かす

```
float position_x, position_y;  
float velocity_x, velocity_y;  
  
void setup() {  
  size(640, 480);  
  background(20, 30, 40);  
  position_x = 20;  
  position_y = 20;  
  velocity_x = 1;  
  velocity_y = 0;  
}  
  
void draw() {  
  background(20, 30, 40);  
  circle(position_x, position_y, 40);  
  position_x += velocity_x;  
  position_y += velocity_y;  
}
```

削除

追加



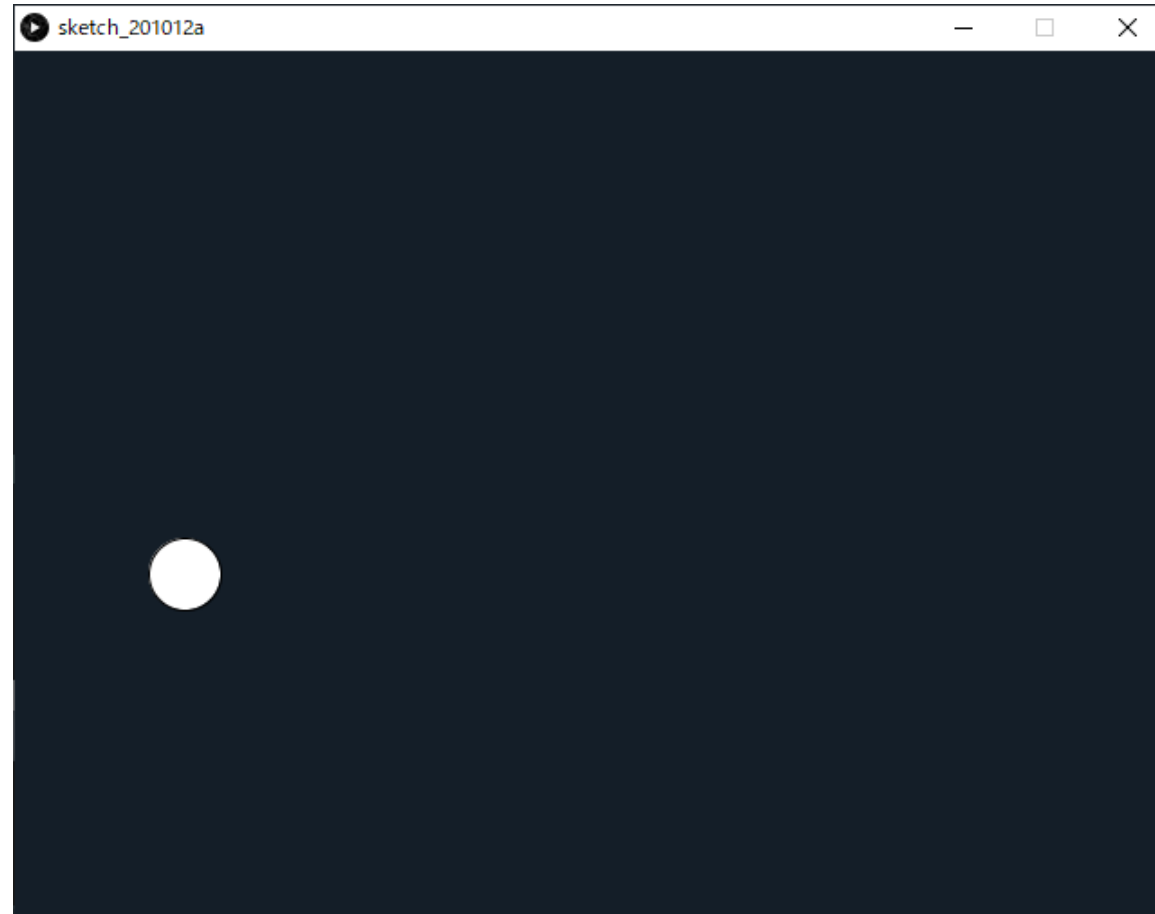
加速度を設定する

```
float position_x, position_y;
float velocity_x, velocity_y;
float acceleration_x, acceleration_y;

void setup() {
    size(640, 480);
    position_x = 20;
    position_y = 20;
    velocity_x = 1;
    velocity_y = 0;
    acceleration_x = 0;
    acceleration_y = 0.1;
}
```

```
void draw() {
    background(20, 30, 40);
    circle(position_x, position_y, 40);
    position_x += velocity_x;
    position_y += velocity_y;
    velocity_x += acceleration_x;
    velocity_y += acceleration_y;
}
```

下に落ちる



クラスを使って書き直す

```
class Vector {  
    float x, y;  
  
    Vector(float x, float y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    void add(Vector v) {  
        this.x += v.x;  
        this.y += v.y;  
    }  
}  
  
// 右に続く
```

← こちらの this. は省略可

```
Vector position, velocity, acceleration;  
  
void setup() {  
    size(640, 480);  
    position = new Vector(20, 20);  
    velocity = new Vector(1, 0);  
    acceleration = new Vector(0, 0.1);  
}  
  
void draw() {  
    background(20, 30, 40);  
    circle(position.x, position.y, 40);  
    position.add(velocity);  
    velocity.add(acceleration);  
}
```

クラス

- 複数のデータを一つにまとめる (メンバ)
- データのまとまりに対する処理を定義する (メソッド)

```
class Vector {  
    float x, y;
```

- Vector というクラスは x, y の2つの float 型のメンバを持つ

```
    Vector(float x, float y) { ... }
```

- コンストラクタ

```
    void add(Vector v) { ... }
```

- メソッド

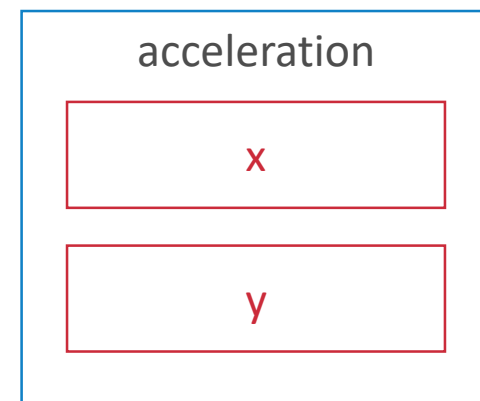
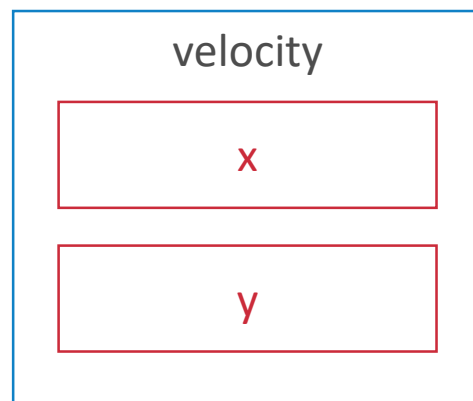
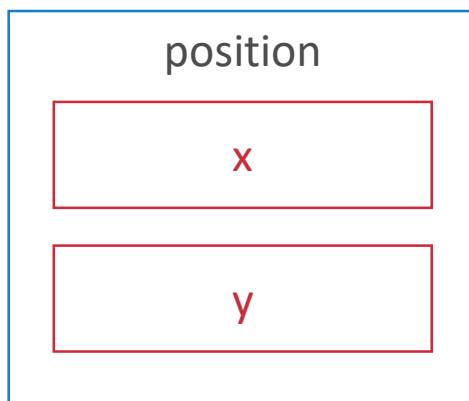
```
}
```


変数宣言

■データの入れ物を用意する

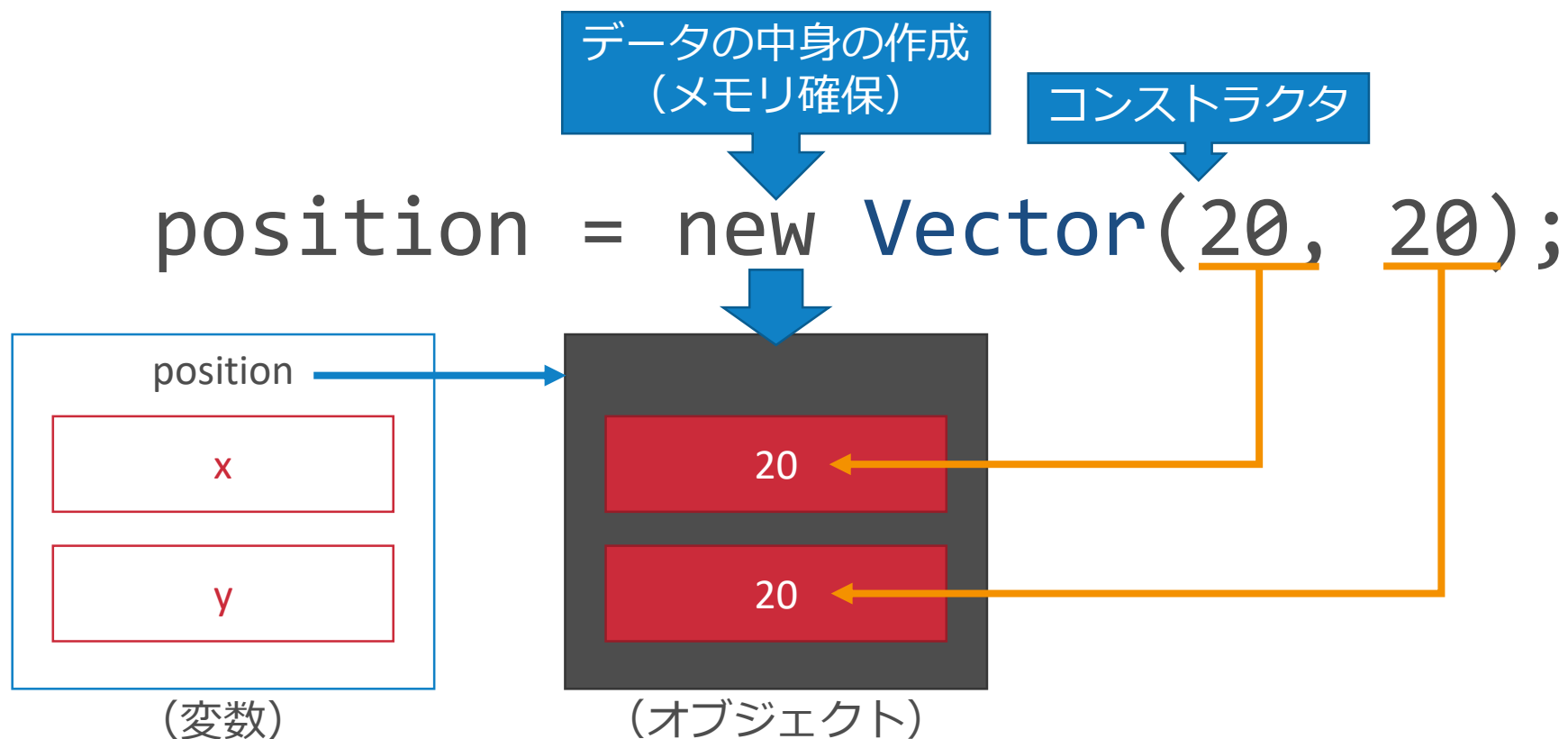
Vector position, velocity, acceleration;

- position（位置を入れるつもり）、velocity（速度を入れるつもり）、acceleration（加速度を入れるつもり）の3つの入れ物を用意する



オブジェクト生成

- データの中身を作って変数に入れる



コンストラクタ

- クラスのデータ（オブジェクト）を作成するときに実行される

```
Vector(float x, float y) {  
    this.x = x;  
    this.y = y;  
}
```

- this

- new で生成したオブジェクト自身

メソッド

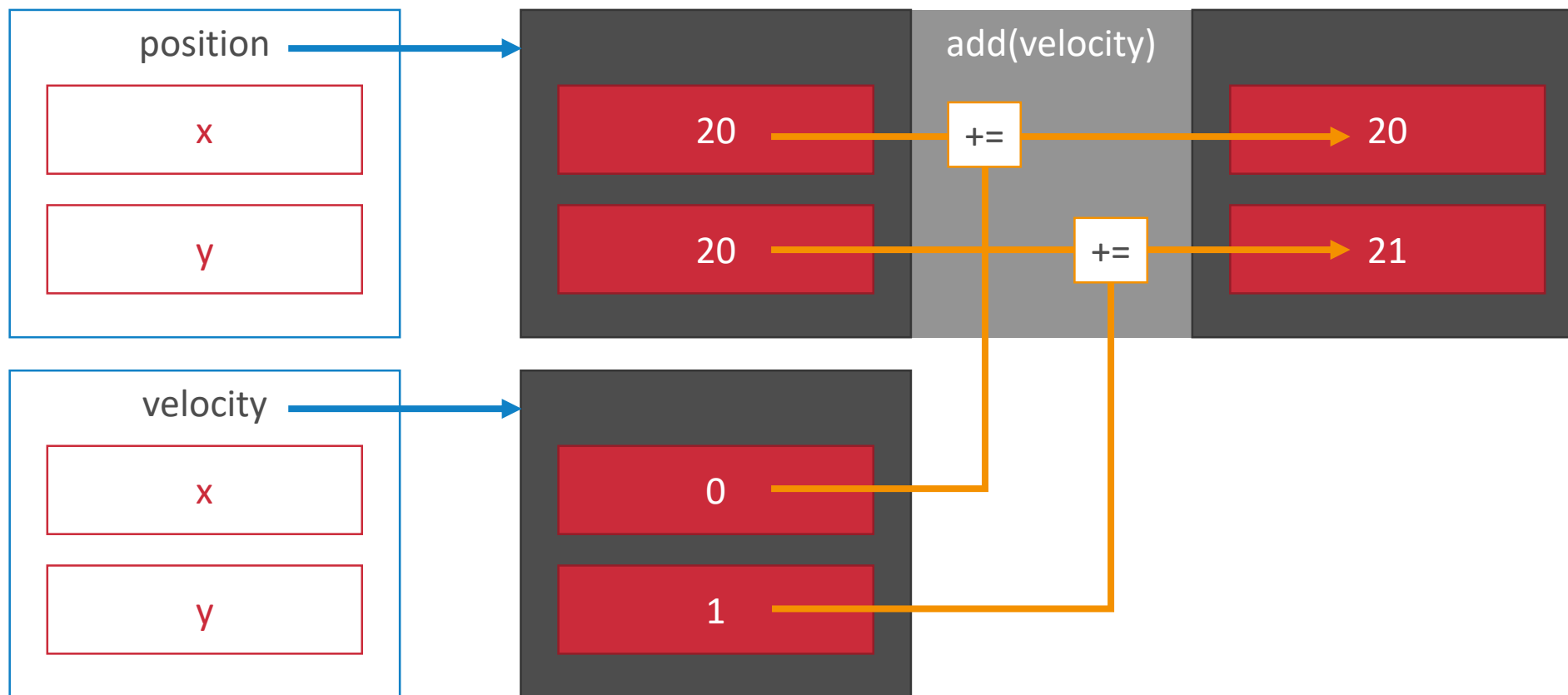
- オブジェクトのデータを操作する

```
void add(Vector v) {  
    this.x += v.x;  
    this.y += v.y;  
}
```

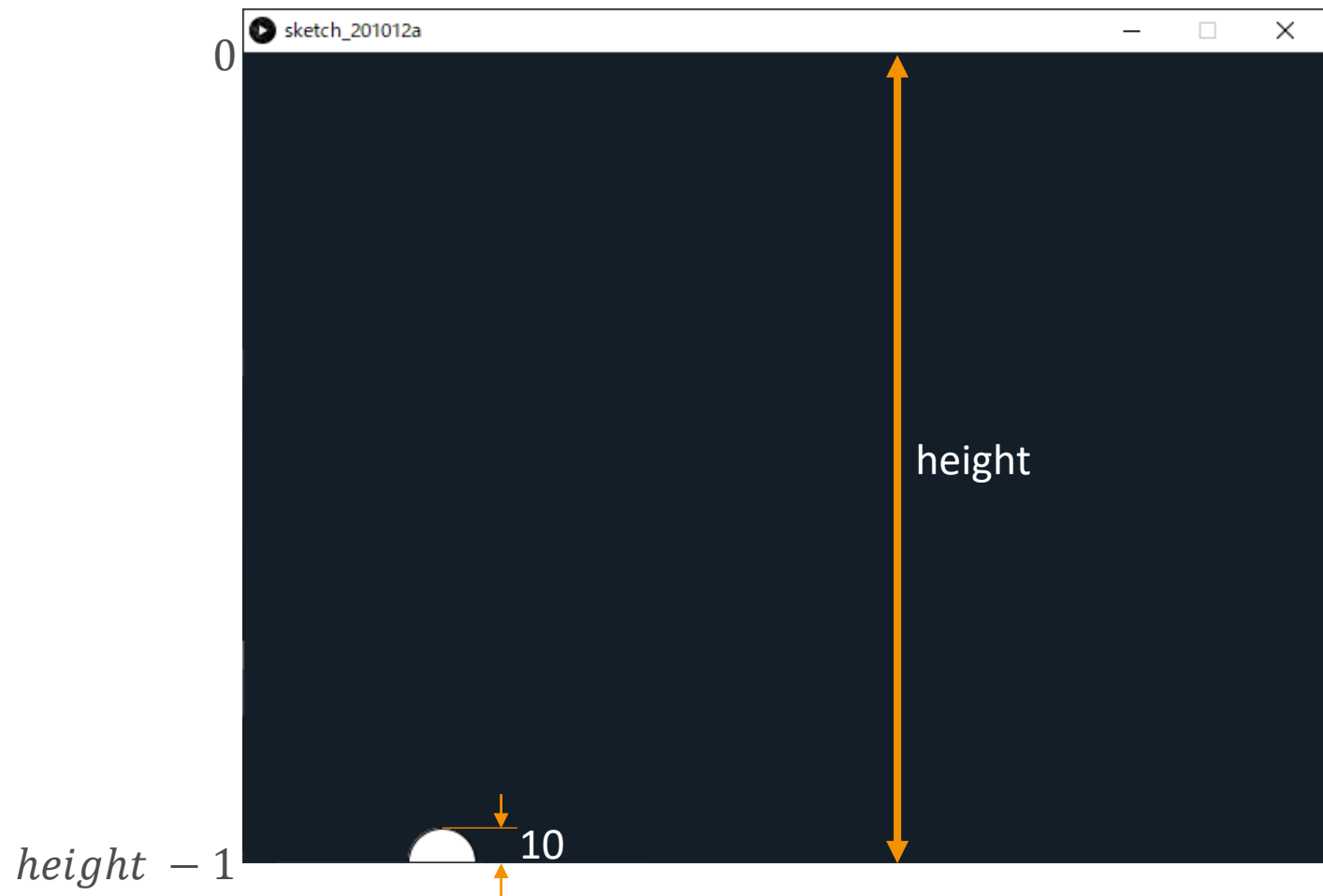
- このオブジェクト (this) のメンバ x, y のそれぞれに v のオブジェクトのメンバ x, y を加える

メソッドの実行

```
position.add(velocity);
```



下端で跳ね返るようにする



下端に接したら y 方向の速度を反転する

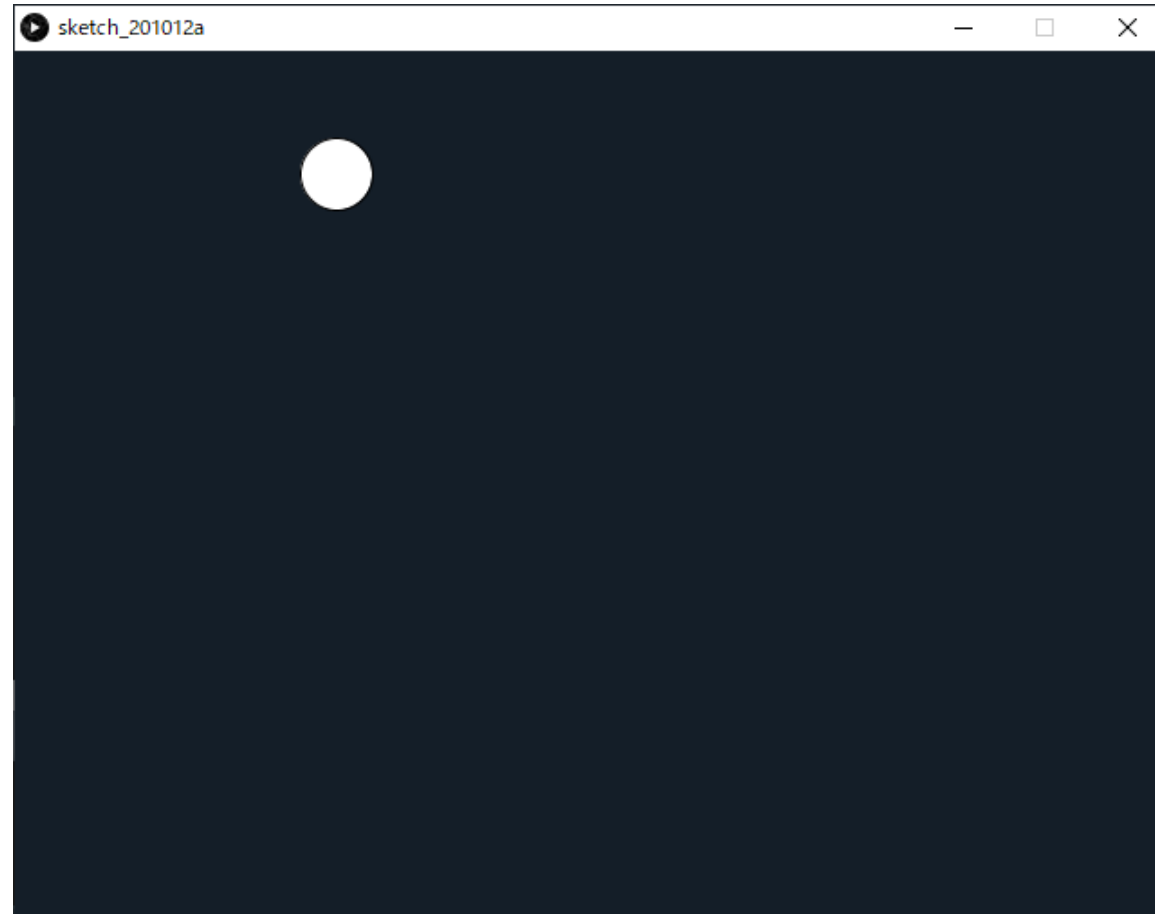
```
class Vector {  
    float x, y;  
  
    Vector(float x, float y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    void add(Vector v) {  
        this.x += v.x;  
        this.y += v.y;  
  
        if (this.y > height - 21) {  
            v.y = -v.y;  
            this.y = height - 21;  
        }  
    }  
}
```

下端からはみ出たとき

速度を反転して位置を境界に戻す

```
Vector position, velocity, acceleration;  
  
void setup() {  
    size(640, 480);  
    position = new Vector(20, 20);  
    velocity = new Vector(1, 0);  
    acceleration = new Vector(0, 0.1);  
}  
  
void draw() {  
    background(20, 30, 40);  
    circle(position.x, position.y, 40);  
    position.add(velocity);  
    velocity.add(acceleration);  
}
```

下端で跳ね返る



課題 3

右端でも跳ね返らせる

円が右端でも跳ね返るようにする

- 表示領域の横幅は `width` に入っています
- `add` メソッドに追加するのが楽だと思います

```
void add(Vector v) {  
    this.x += v.x;  
    this.y += v.y;  
    if (this.y > height - 21) {  
        v.y = -v.y;  
        this.y = height - 21;  
    }  
}
```

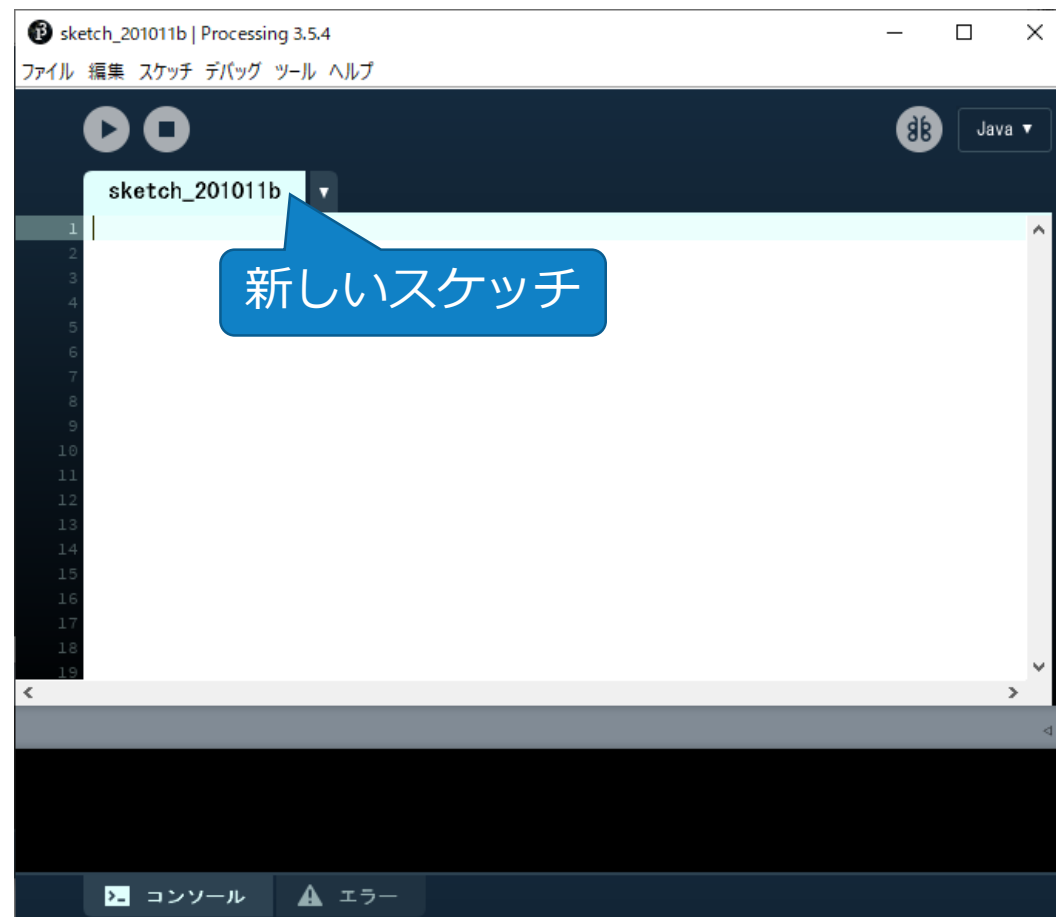
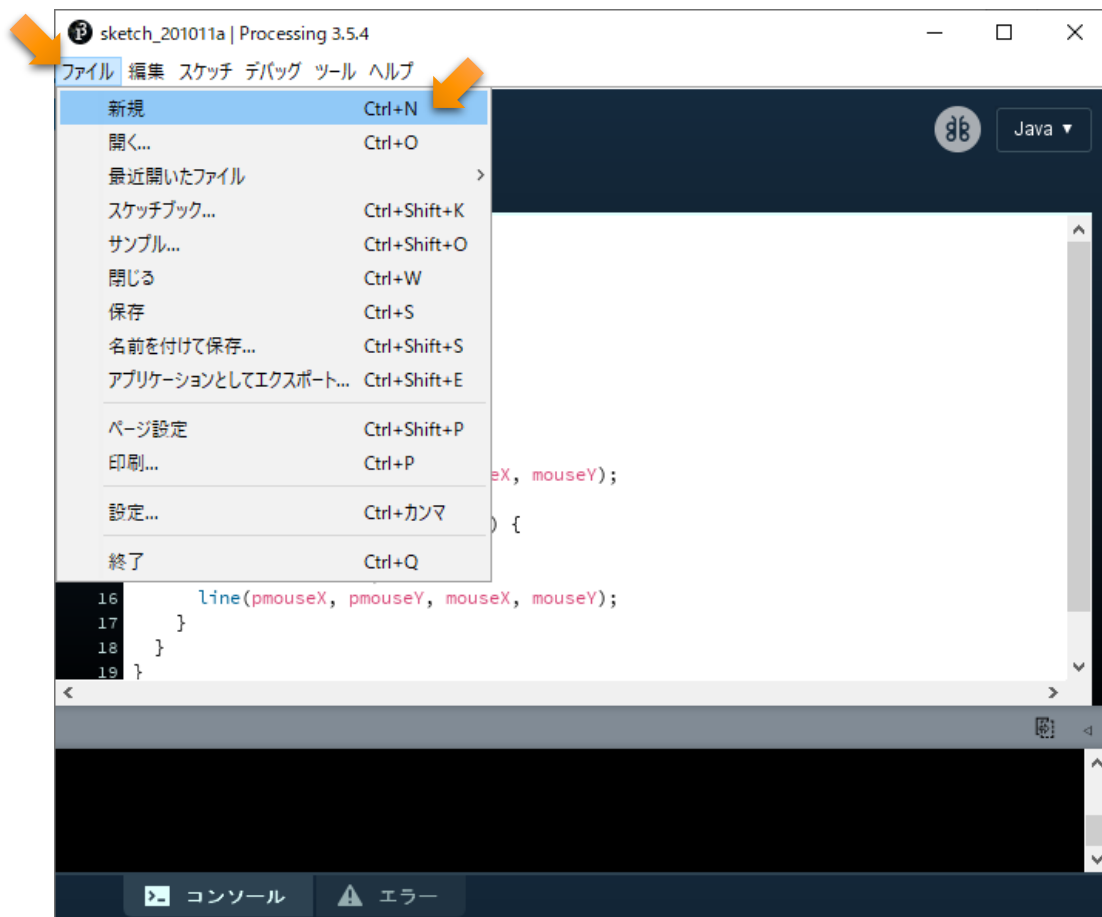
スクリーンショットをアップロードする

- 実行結果のウィンドウのスクリーンショットを適当なタイミングで撮って **03.png** というファイル名で保存して、Moodle の「スクリーンショットのアップロード」からアップロードしてください。

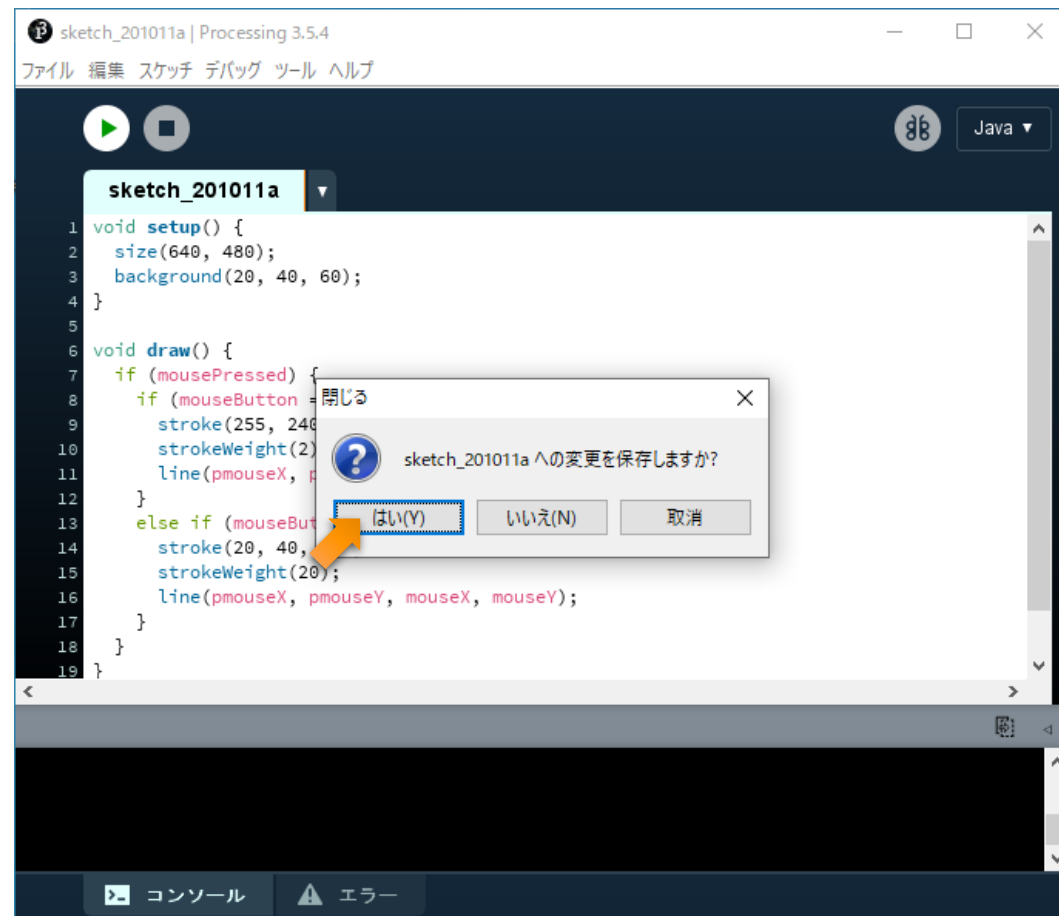
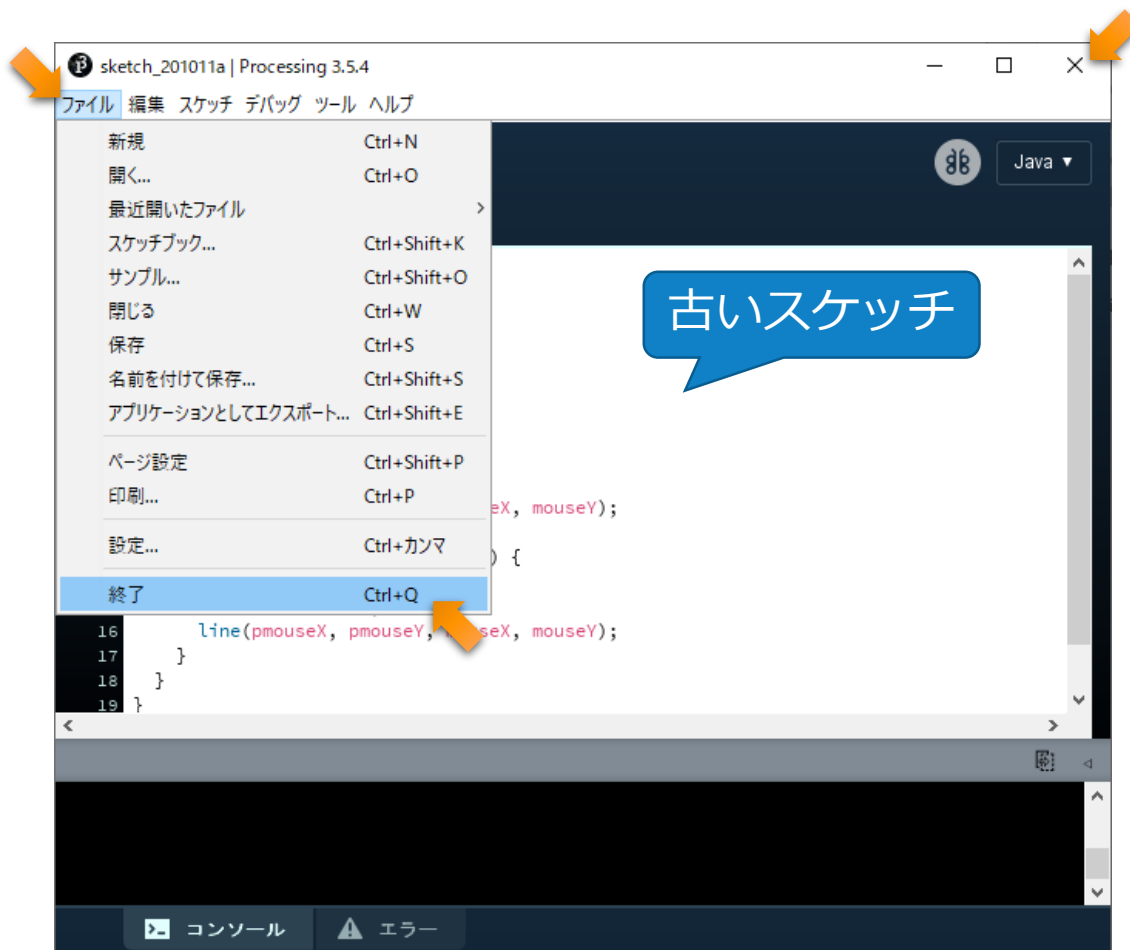
映像入力

パソコンに Web カメラが付いていなければ「画像入力」に進んでください

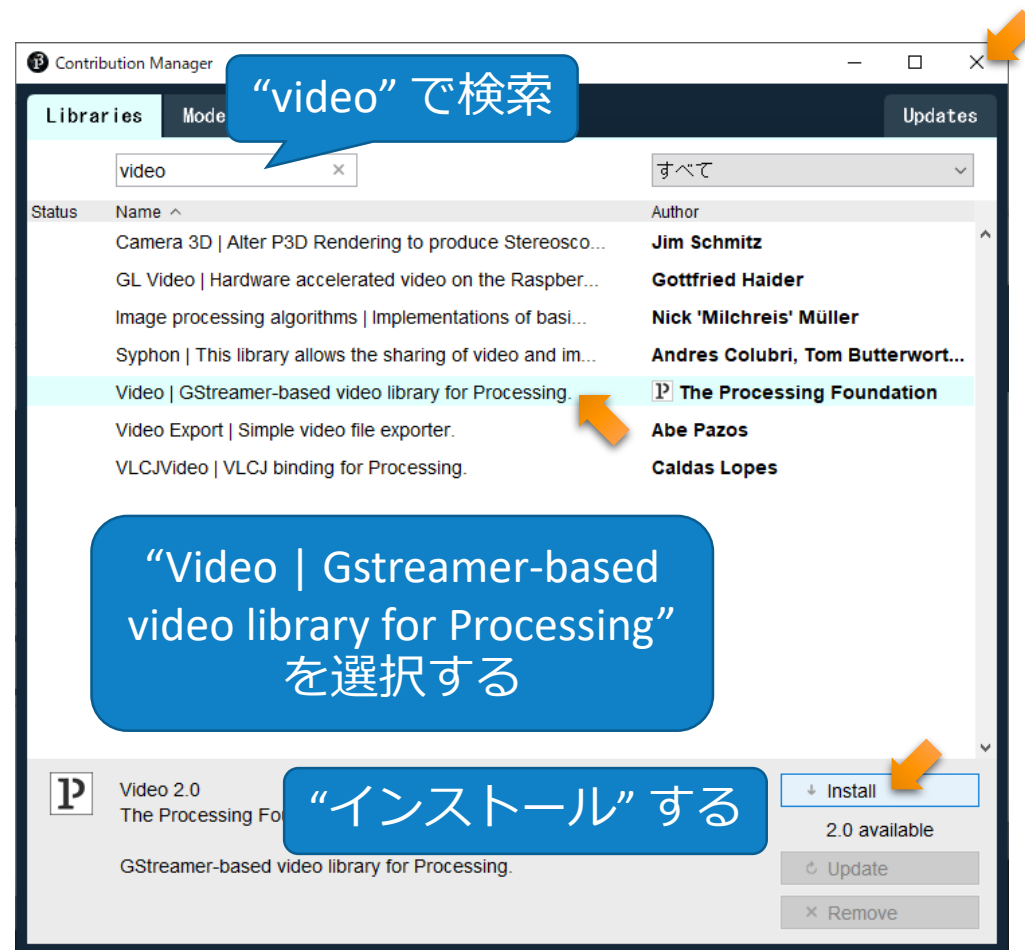
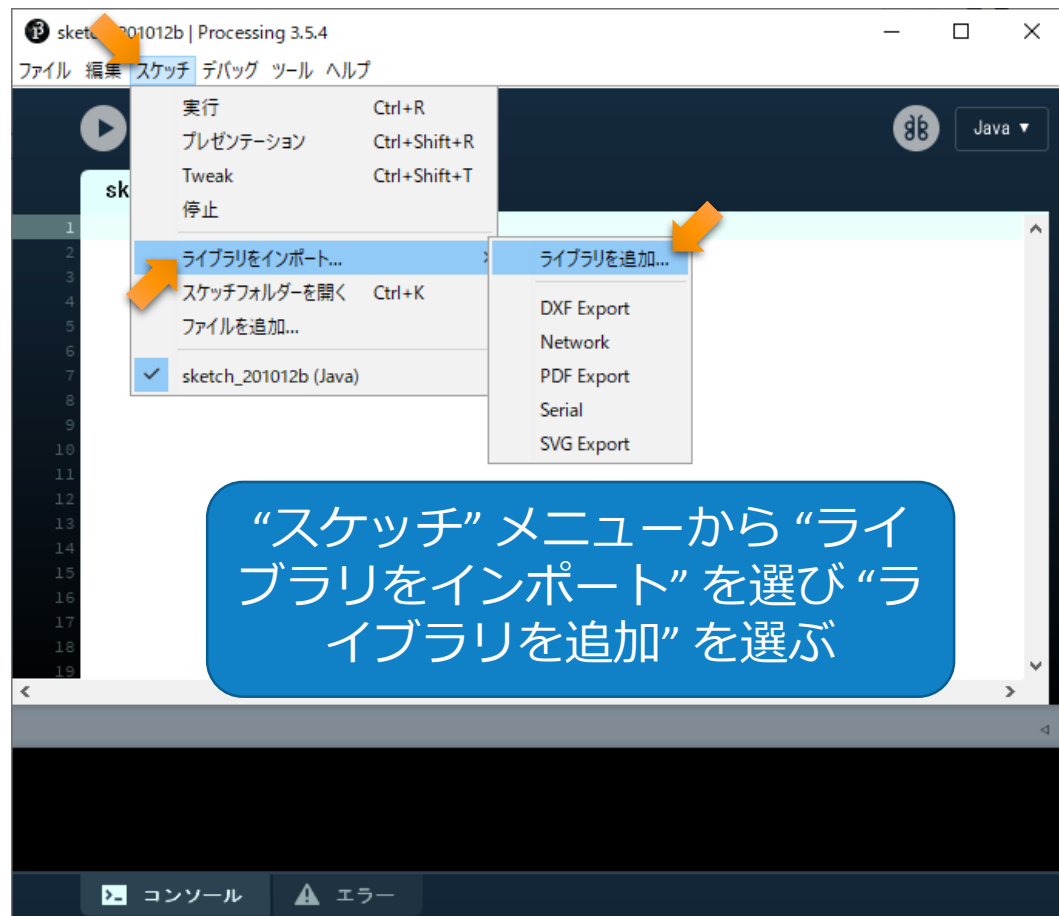
新しいスケッチを作成する



今まで使っていたスケッチは終了する



ビデオ入力ライブラリを追加する



Web カメラから入力した画像を表示する

```
import processing.video.*;
Capture img;

void setup() {
  size(640, 480);
  img = new Capture(this);
  img.start();
}

void draw() {
  if (img.available()) {
    img.read();
  }
  image(img, 0, 0, width, height);
}
```



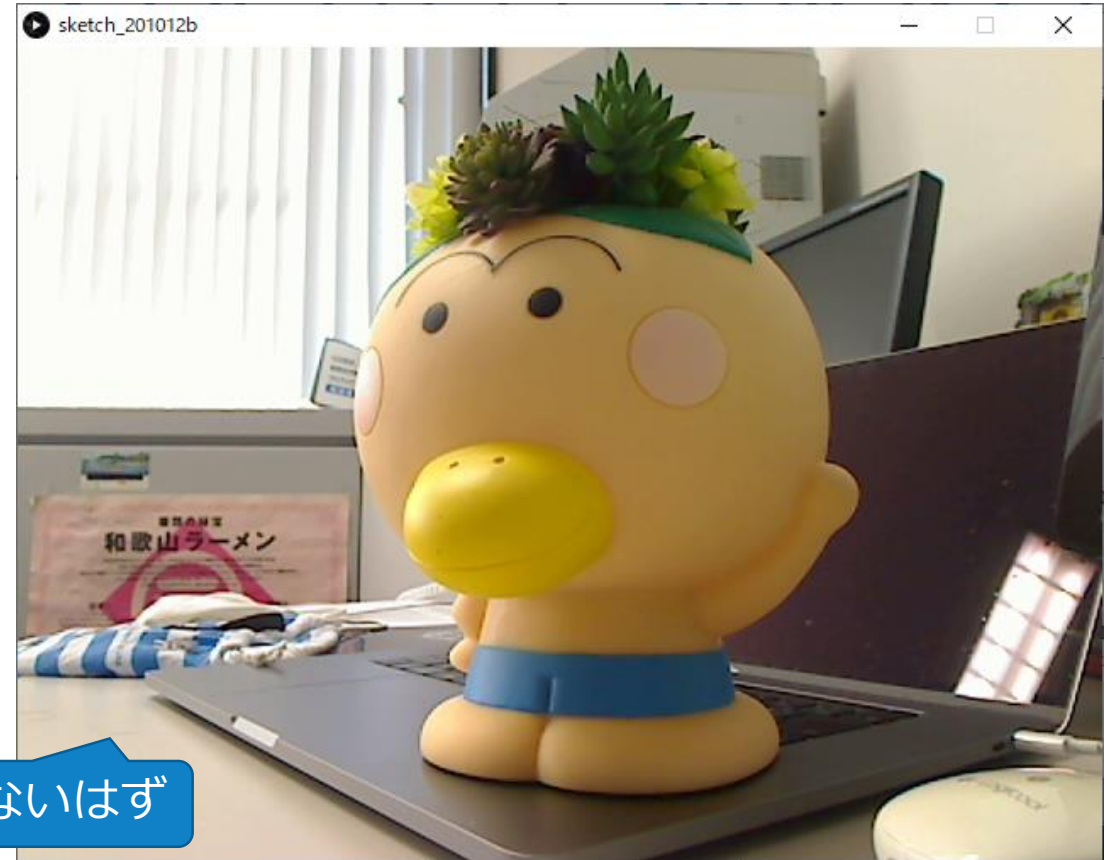
画像の解像度を表示サイズに合わせる

```
import processing.video.*;
Capture img;

void setup() {
  size(640, 480);
  img = new Capture(this);
  img.start();
}

void draw() {
  if (img.available()) {
    img.read();
    img.resize(width, height);
  }
  image(img, 0, 0, width, height);
}
```

変化はないはず



画像から100点ランダムに色を取得する

```
import processing.video.*;
Capture img;

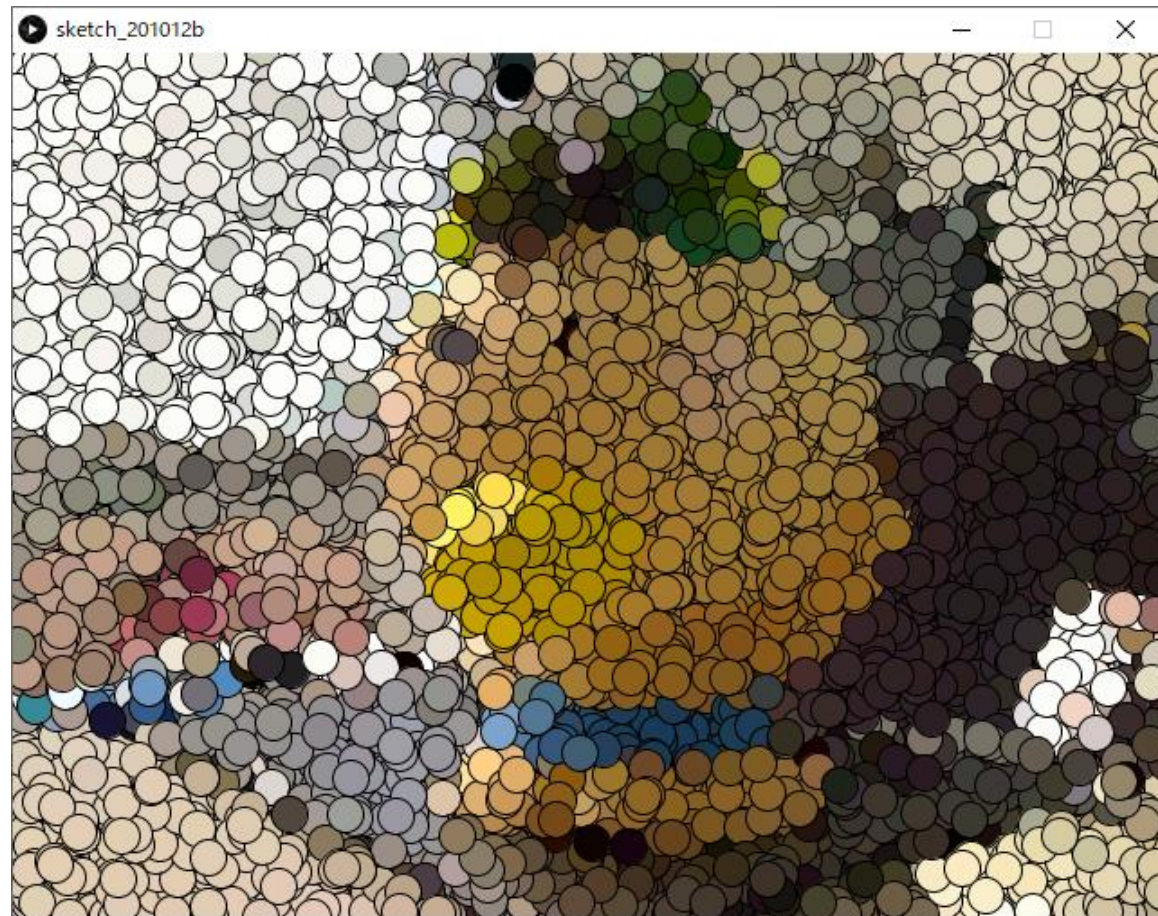
void setup() {
  size(640, 480);
  img = new Capture(this);
  img.start();
}

void draw() {
  if (img.available()) {
    img.read();
    img.resize(width, height);
  }

  // 右に続く
```

```
for (int i = 0; i < 100; ++i) {
  int x = int(random(width));
  int y = int(random(height));
  color c = img.get(x, y);
  fill(c);
  circle(x, y, 20);
}
```

点を取得した位置にその色で円を描く



ビデオ入力の準備

```
import processing.video.*;
```

- ビデオ入力ライブラリの読み込み

```
Capture img;
```

- ビデオ入力のクラス

```
img = new Capture(this);
```

- ビデオ入力のオブジェクトの作成

```
img.start();
```

- ビデオ入力の開始

ビデオ入力の処理はグラフィックス表示と並行動作する

フレームの取り込み

```
if (img.available()) {
```

- もしフレーム（ビデオの1枚の画像）が撮り込めていたら

```
    img.read();
```

- その1枚のフレームを取り出して

```
    img.resize(width, height);
```

- 幅 width、高さ height に拡大縮小する

```
}
```

画像中の100点を円に置き換えて描画する

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

- }までを100回繰り返す

```
    int x = int(random(width));  
    int y = int(random(height));
```

- random(width) は0以上 width 未満の float 型（実数型）の乱数を返す
- それを int 型に直して int 型（整数型）の変数 x に格納している（y についても同様）

```
    color c = img.get(x, y);
```

- 画像の (x, y) の位置の色を取り出して c に格納している（color は色のデータ型）

```
    fill(c);  
    circle(x, y, 20);
```

- 取り出した色 c を塗りつぶしの色にを使って (x, y) の位置に半径 20 の円を描く

```
}
```

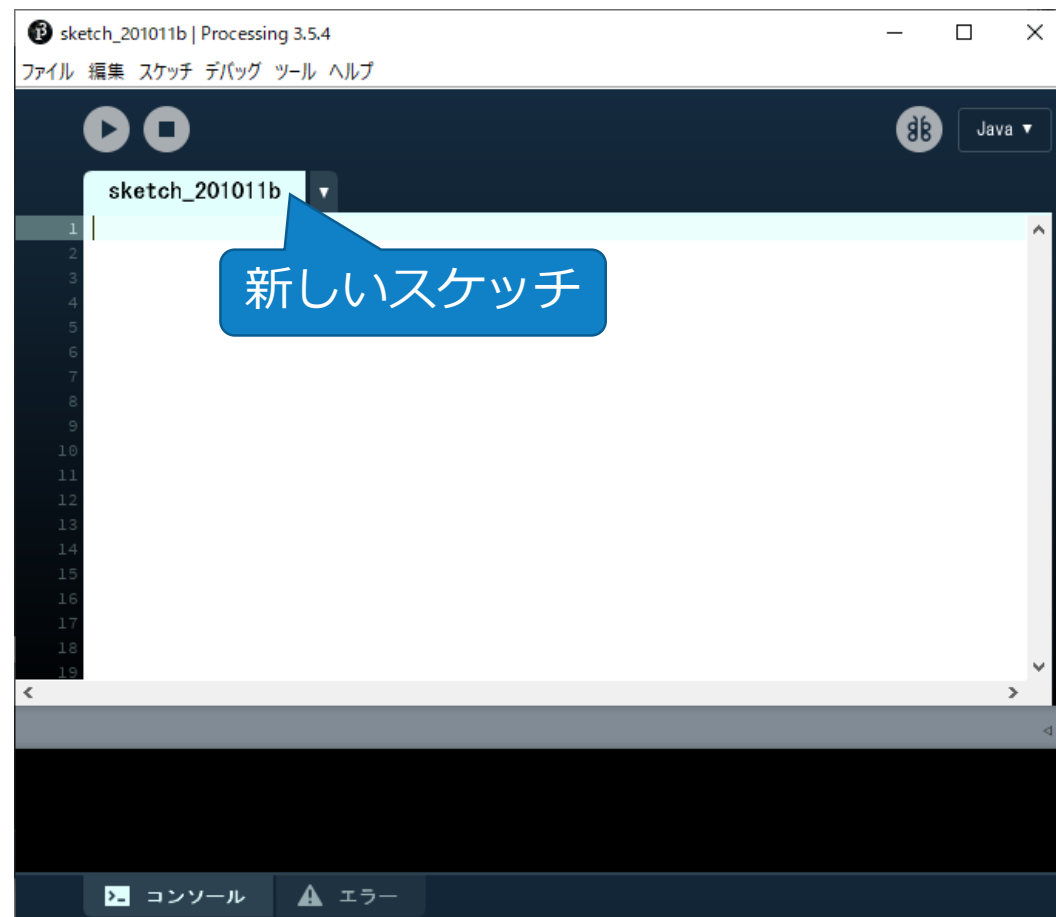
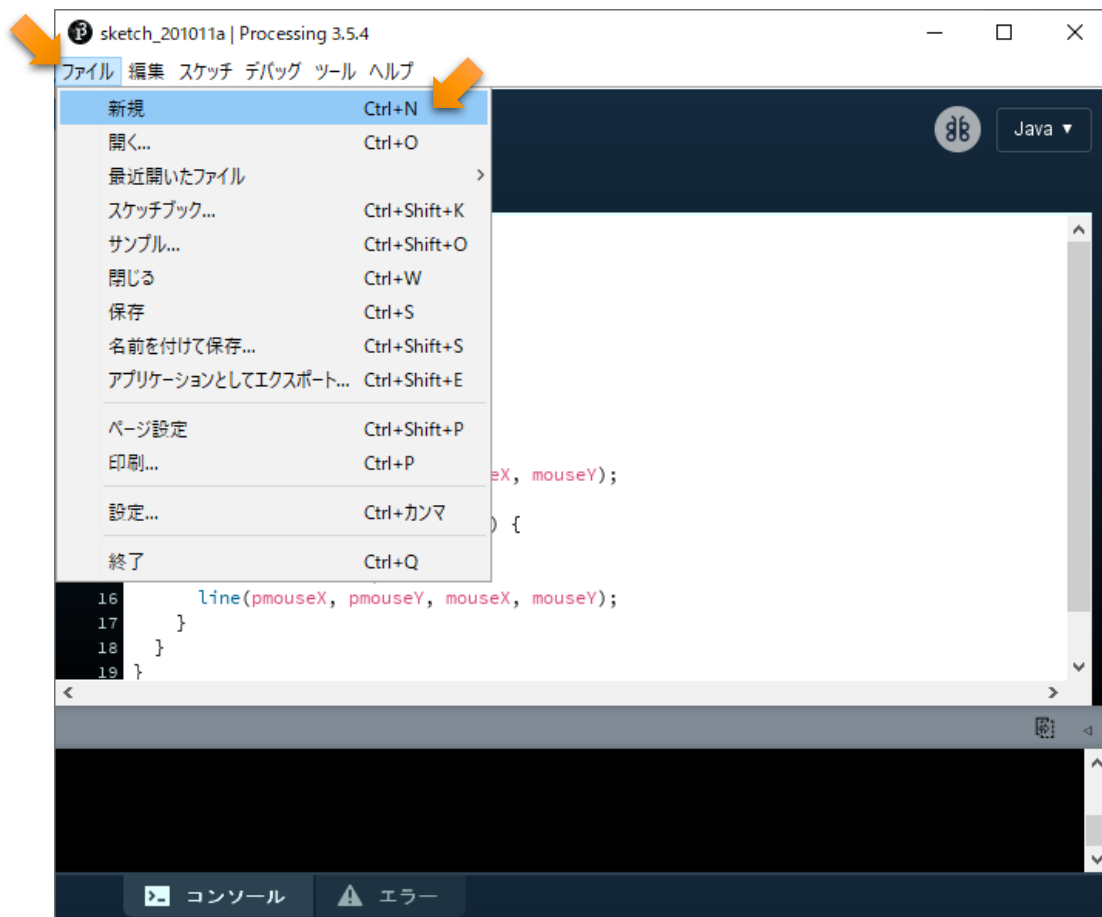
課題 4

に進んでください

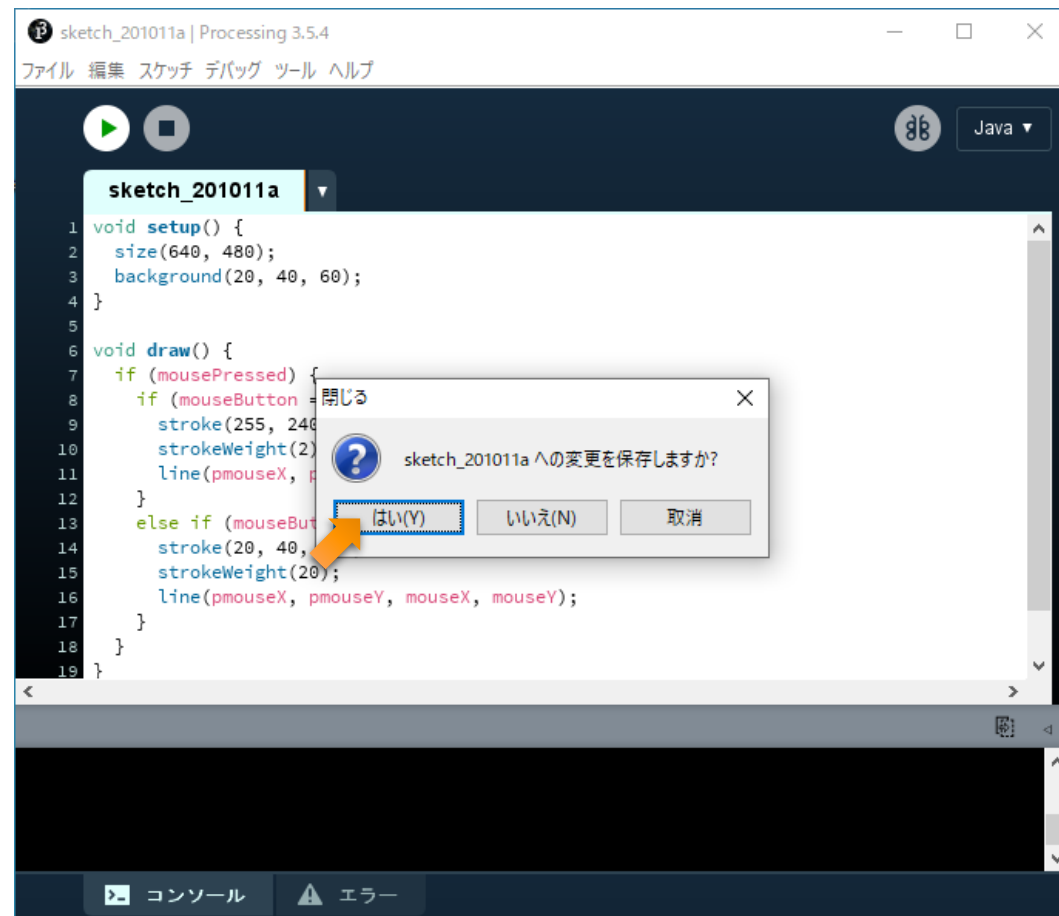
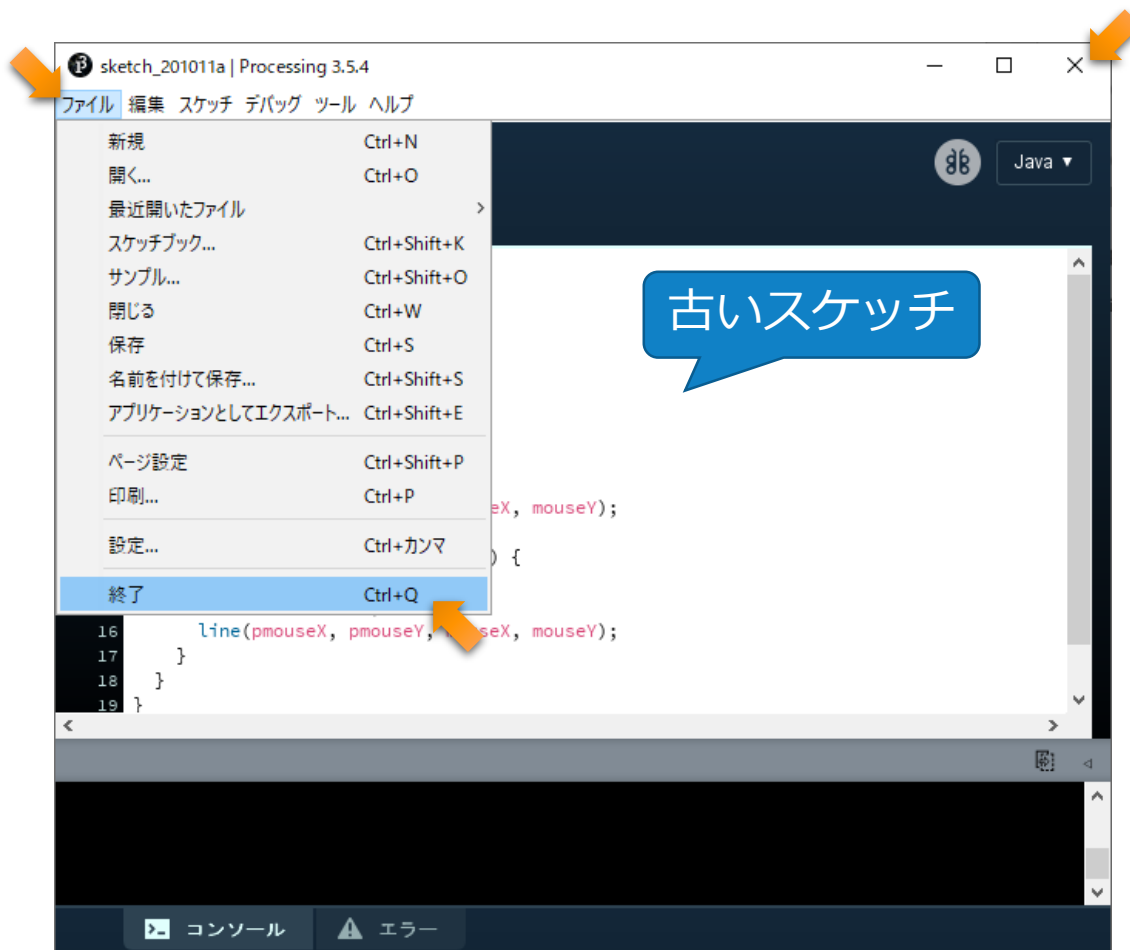
画像入力

画像を加工する（パソコンに Web カメラが付いていない場合）

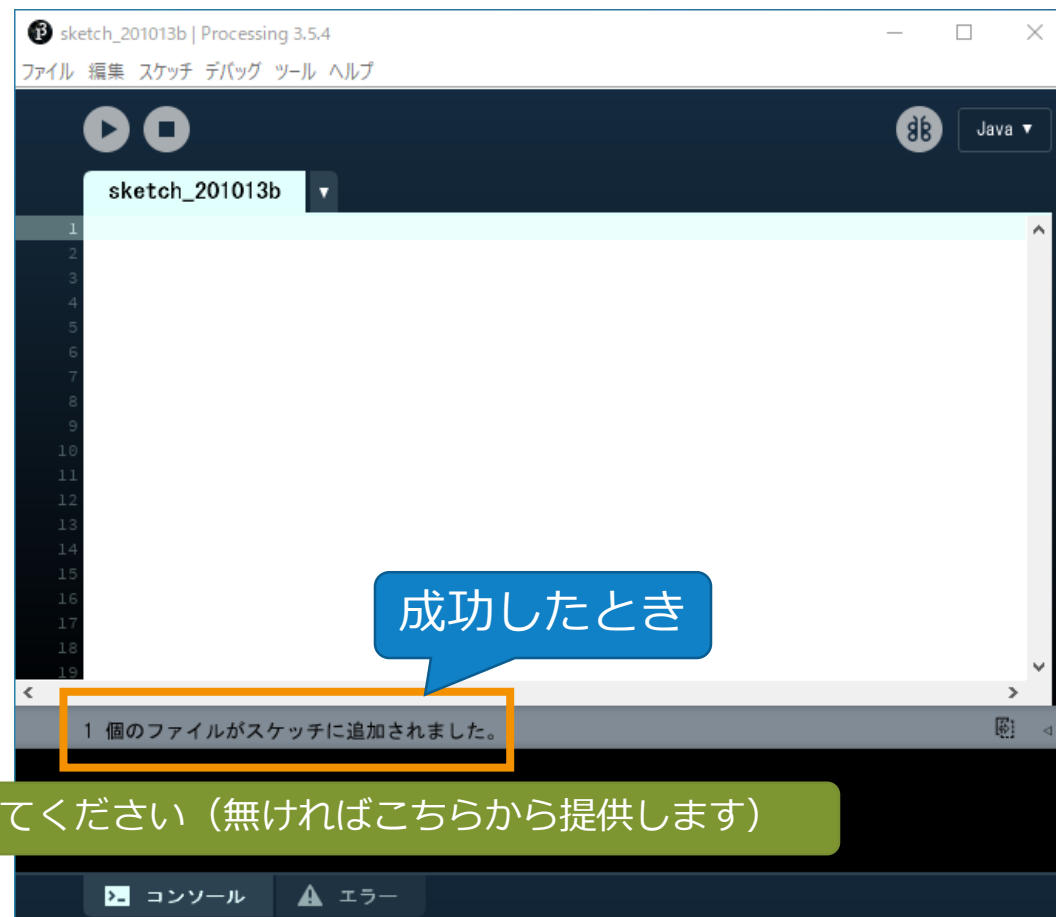
新しいスケッチを作成する



今まで使っていたスケッチは終了する



画像ファイルをスケッチに入れる

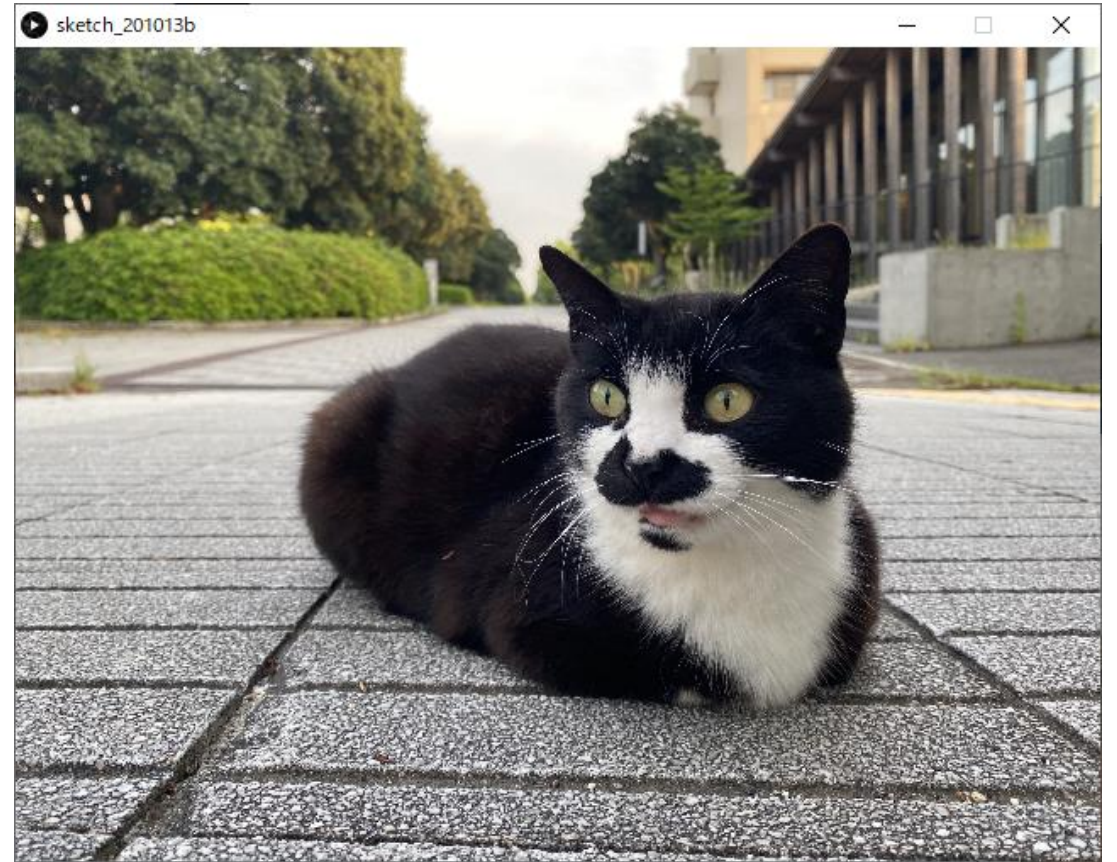


画像ファイルは手持ちのもの (jpg, png, gif) を使ってください (無ければこちらから提供します)

画像ファイルの内容を表示する

```
PImage img;  
  
void setup() {  
  size(640, 480);  
  img = loadImage("image.jpg");  
}  
  
void draw() {  
  image(img, 0, 0, width, height);  
}
```

追加した画像ファイル名



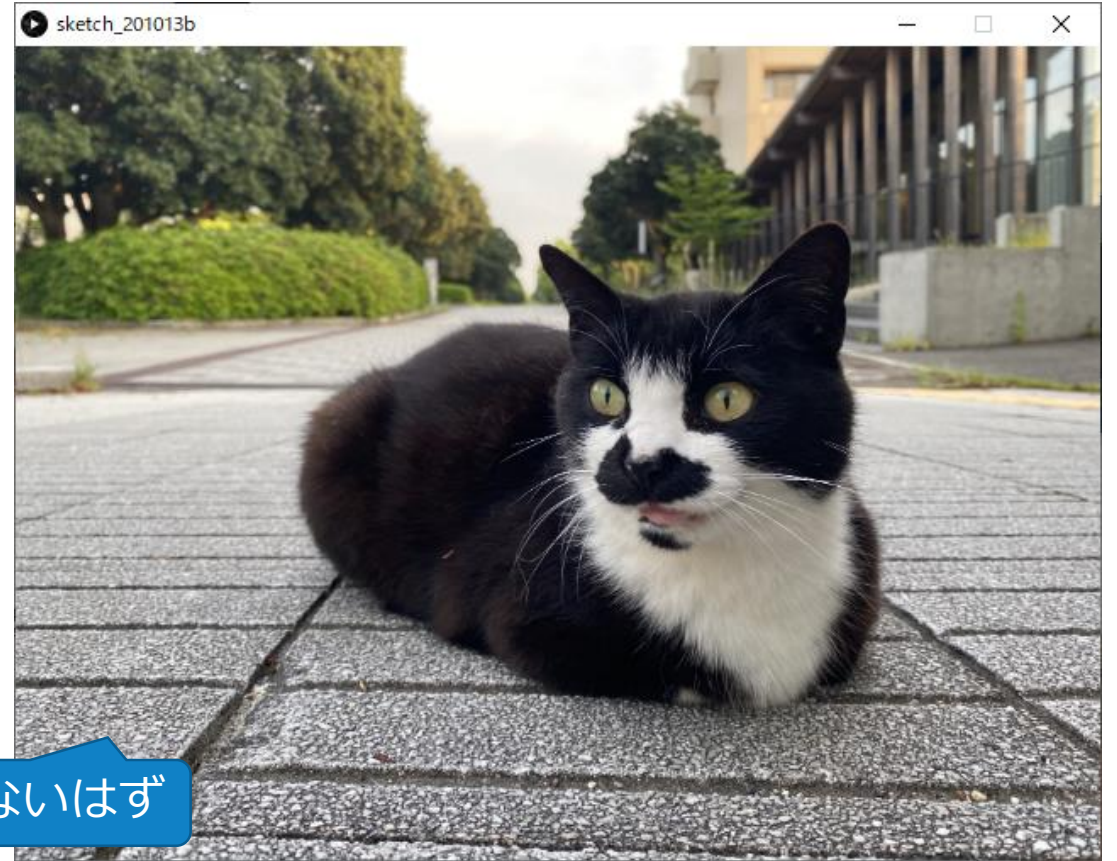
画像の解像度を表示サイズに合わせる

```
PImage img;

void setup() {
  size(640, 480);
  img = loadImage("image.jpg");
}

void draw() {
  img.resize(width, height);
  image(img, 0, 0, width, height);
}
```

変化はないはず



画像から100点ランダムに色を取得する

```
PImage img;

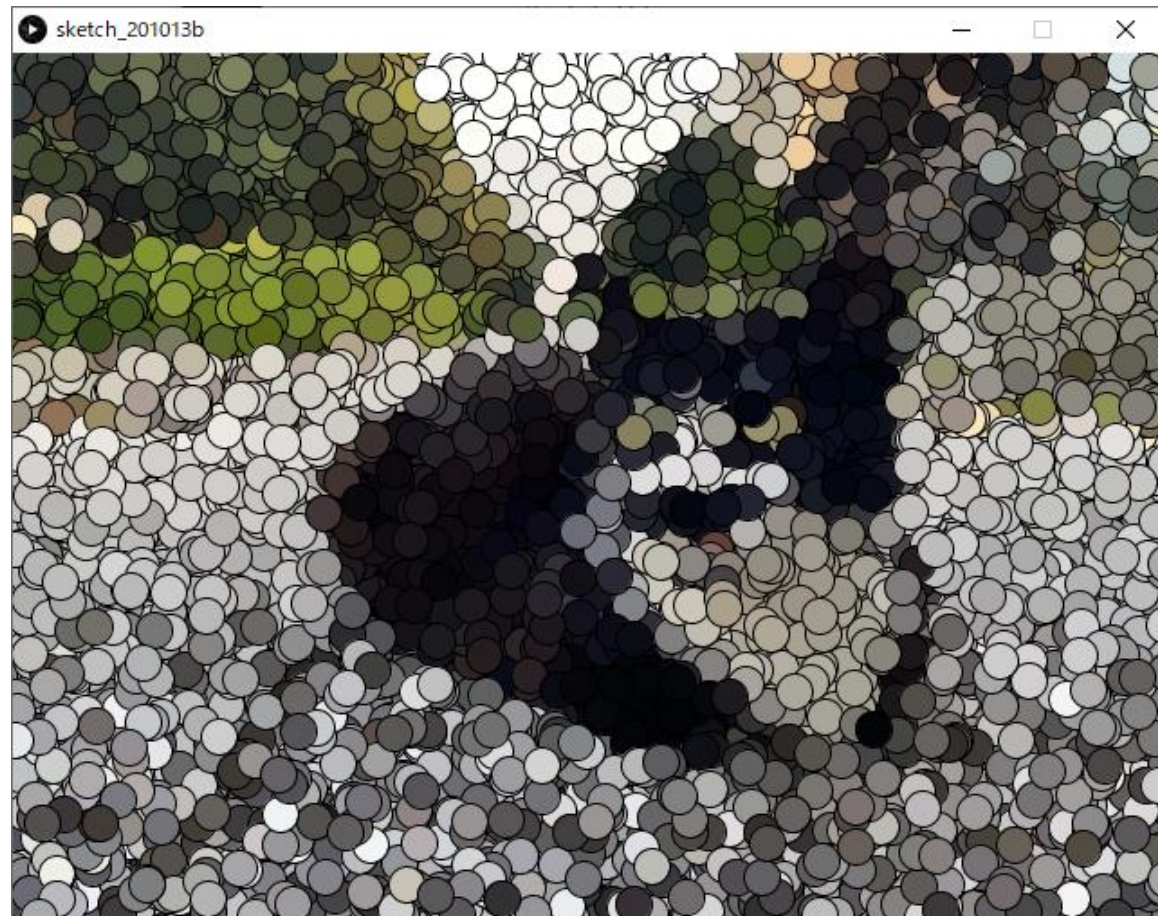
void setup() {
  size(640, 480);
  img = loadImage("image.jpg");
}

void draw() {
  img.resize(width, height);

  // 右に続く
```

```
for (int i = 0; i < 100; ++i) {
  int x = int(random(width));
  int y = int(random(height));
  color c = img.get(x, y);
  fill(c);
  circle(x, y, 20);
}
```


点を取得した位置にその色で円を描く



課題 4

画像中の点を円以外の図形に置き換える

スクリーンショットをアップロードする

- 画像中の点を円以外の図形に置き換えた実行結果のウィンドウのスクリーンショットを適当なタイミングで撮って **04.png** というファイル名で保存して、Moodle の「スクリーンショットのアップロード」からアップロードしてください。