Processing ノート

黒石商業高校「Java 言語を利用したゲームアプリケーションソフトウェアの開発」

2020 年 9 月 2 日(水)、3 日(木)、4 日(金)

1. println()関数

1.1 説明

機能: コンソールに 1 行表示

意味: print line プリントライン

使い方: println(表示したい内容);

1.2 コンソールに「おはよう」と表示


```
println("おはよう");
```

1.3 コンソールに「おはよう」「こんにちは」と 1 行ずつ表示


```
println("おはよう");
println("こんにちは");
```

1.4 コンソールに 1～5 までの数字を 1 行ずつ表示


```
for (int i = 1; i < 6; i++) {
```

```
  println(i);
}
```

2.size()関数

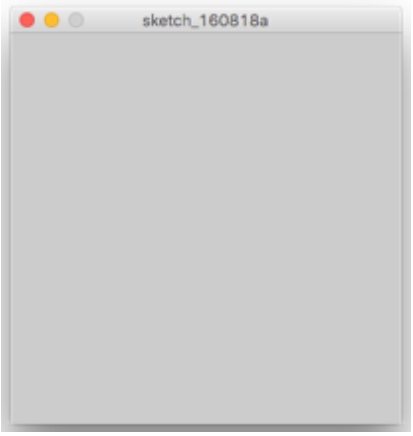
2.1 説明

機能: ディスプレイ・ウィンドウの大きさを指定

意味: size サイズ

使い方: size(幅, 高さ);

2.1 ディスプレイ・ウィンドウのサイズを 320x320 に




```
size(320, 320);
```

2.2 ディスプレイ・ウィンドウのサイズを 480x240 に


```
size(480, 240);
```

3. 描画関数

3.1 point()関数

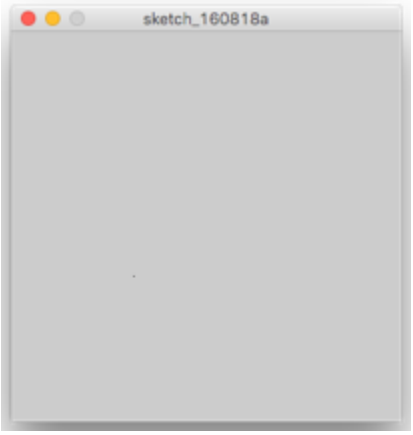
3.1.1 説明

機能: 点を描画

意味: point ポイント

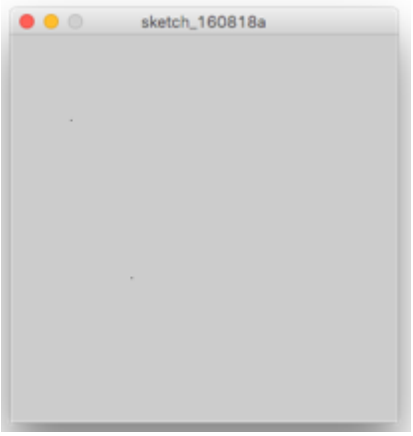
使い方: point(x, y);

3.1.2 点を(100, 200)に描画。ディスプレイ・ウィンドウのサイズは 320x320




```
size(320, 320);
point(100, 200);
```

3.1.3 点を(100, 200)と(50, 70)に描画。ディスプレイ・ウィンドウのサイズは 320x320




```
size(320, 320);
point(100, 200);
point(50, 70);
```

3.2 line()関数

3.2.1 説明

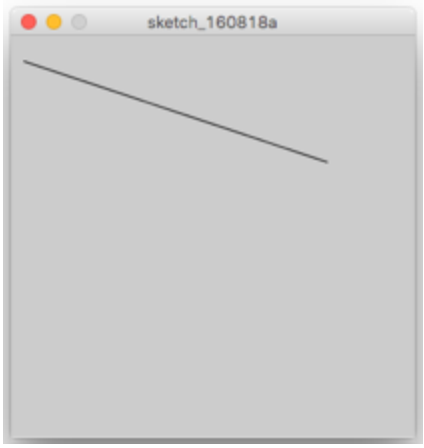
機能: 線を描画

意味: line ライン

使い方: line(x1, y1, x2, y2);

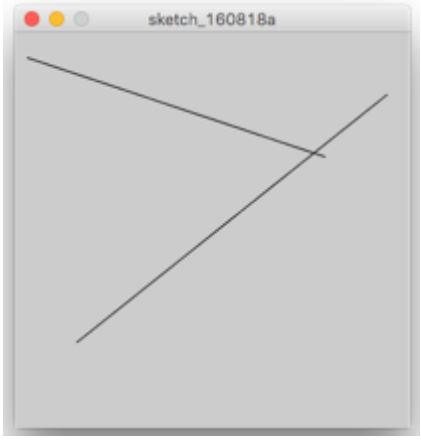
3.2.2 線を(10, 20)から(250, 100)に描画。ディスプレイ・ウィンドウのサイズは 320x320。

以降、特に断らない限り、ディスプレイ・ウィンドウのサイズは 320x320 とする。




```
size(320, 320);
line(10, 20, 250, 100);
```

3.2.3 線を 2 つ描画。1 つは(10, 20)から(250, 100)、もう一つは(300, 50)から(50, 250)。




```
size(320, 320);
line(10, 20, 250, 100);
```

```
line(300, 50, 50, 250);
```

3.3 rect()関数

3.3.1 説明

機能: 長方形を描画

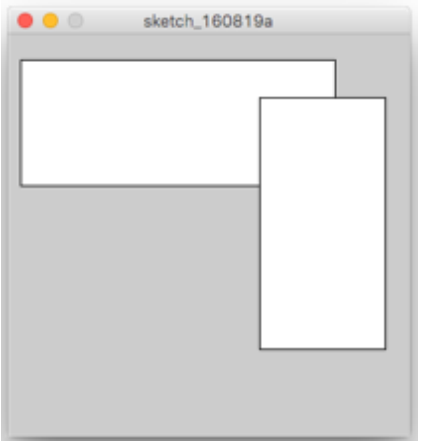
意味: rectangle レクタングル

使い方: rect(x, y, 幅, 高さ);

3.2.2 長方形を描画。左上が(10, 20)で、幅 250 高さ 100。


```
size(320, 320);
rect(10, 20, 250, 100);
```

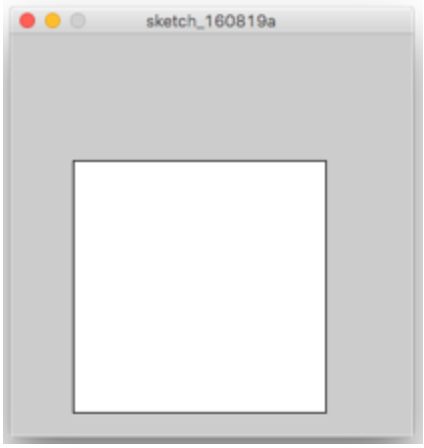
3.3.3 長方形を 2 つ描画。1 つは左上が(10, 20)で、幅 250 高さ 100。もう 1 つは左上が(200, 50)で、幅 100 高さ 200。




```
size(320, 320);
```

```
rect(10, 20, 250, 100);
rect(200, 50, 100, 200);
```

3.3.4 正方形を描画。左上が(50, 100)で、1 辺の長さ 200



3.4 ellipse()関数

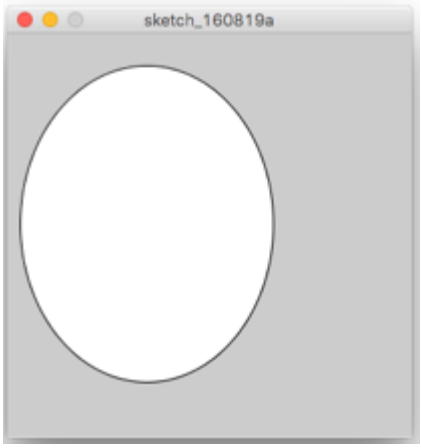
3.4.1 説明

機能: 楕円を描画

意味: ellipse エリプス

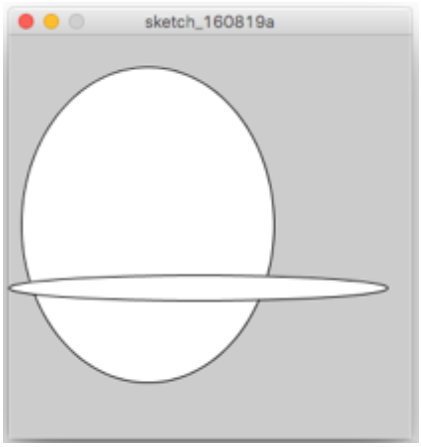
使い方: ellipse(x, y, 幅, 高さ);

3.4.2 楕円形を描画。中心が(110, 150)で、幅 200 高さ 250。



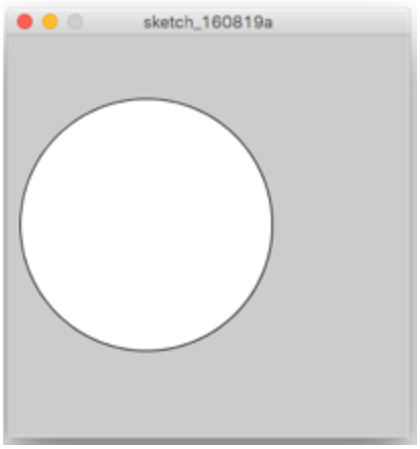

```
size(320, 320);
ellipse(110, 150, 200, 250);
```

3.4.3 楕円形を 2 つ描画。1 つは中心が(110, 150)で、幅 200 高さ 250。もう一つは中心が(150, 200)で、幅 300 高さ 20。




```
size(320, 320);
ellipse(110, 150, 200, 250);
ellipse(150, 200, 300, 20);
```

3.4.4 円を描画。中心が(110, 150)で、直径が 200。




```
size(320, 320);
ellipse(110, 150, 200, 200);
```

3.5 background()関数

3.5.1 説明

機能: 背景を描画

意味: background バックグランド

使い方: background(色);

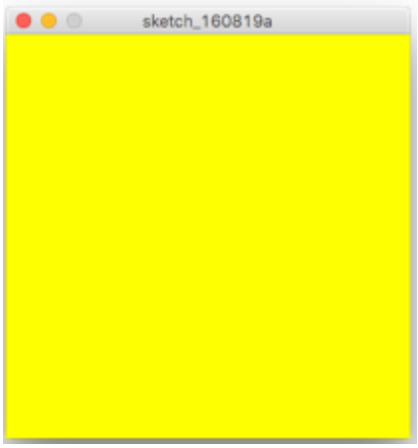
3.5.2 背景を白に。


```
size(320, 320);
background(255);
```

3.5.3 背景を赤に。


```
size(320, 320);
background(255, 0, 0);
```

3.5.4 背景を黄色に。




```
size(320, 320);
background(255, 255, 0);
```

3.6 fill()関数

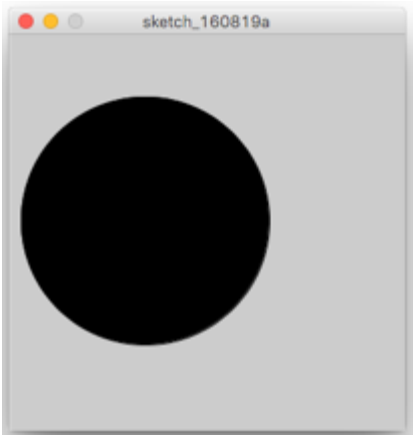
3.6.1 説明

機能: 塗りの色を指定

意味: fill フィル

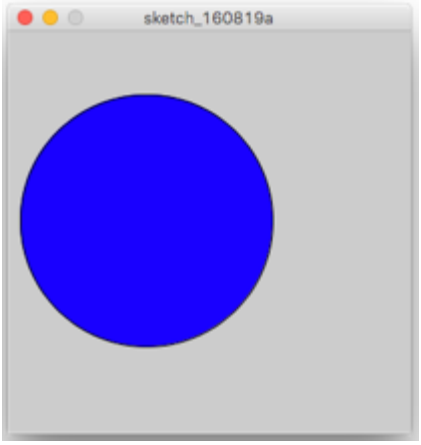
使い方: fill(色);

3.6.2 円を描画。中心が(110, 150)で、直径が200、塗りが黒。




```
size(320, 320);
fill(0);
ellipse(110, 150, 200, 200);
```

3.6.3 円を描画。中心が(110, 150)で、直径が200、塗りが青。




```
size(320, 320);
fill(0, 0, 255);
ellipse(110, 150, 200, 200);
```

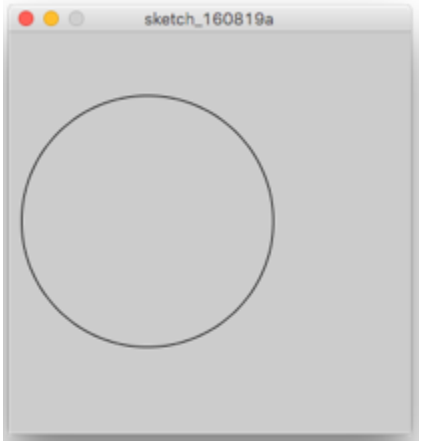
3.7 noFill()関数

3.7.1 説明

機能: 色を塗らない

使い方: noFill();

3.7.2 円を描画。中心が(110, 150)で、直径が200、塗らない。




```
size(320, 320);
noFill();
ellipse(110, 150, 200, 200);
```

3.8 strokeWeight()関数

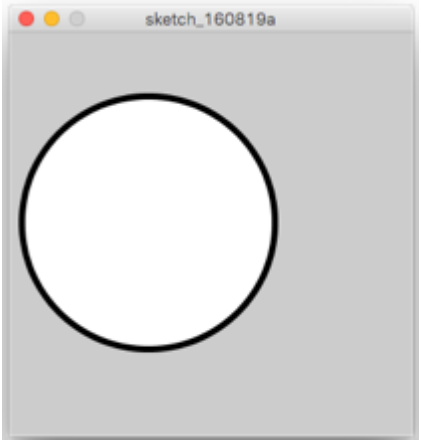
3.8.1 説明

機能: 線の太さを指定

意味: stroke weight ストローク ウェイト

使い方: strokeWeight(太さ);

3.8.2 円を描画。中心が(110, 150)で、直径が200、線の太さ5。




```
size(320, 320);
strokeWeight(5);
ellipse(110, 150, 200, 200);
```

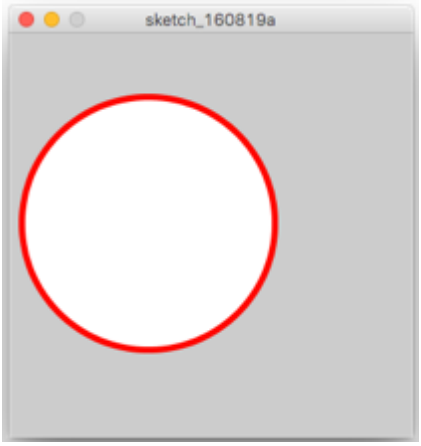
3.9 stroke()関数

3.9.1 説明

機能: 線の色を指定

使い方: stroke(色);

3.9.2 円を描画。中心が(110, 150)で、直径が 200、線の色が赤、線の太さ 5。




```
size(320, 320);
stroke(255, 0, 0);
strokeWeight(5);
ellipse(110, 150, 200, 200);
```

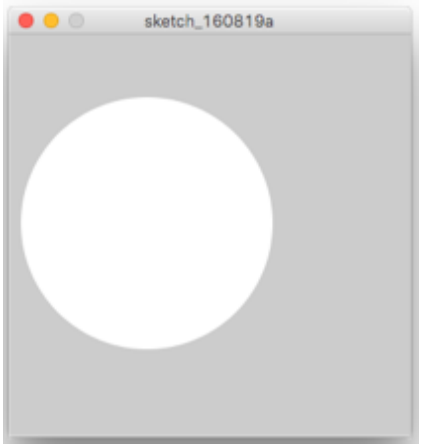
3.10 noStroke()関数

3.10.1 説明

機能: 線を描かない

使い方: noStroke();

3.10.2 円を描画。中心が(110, 150)で、直径が 200、線を描かない。



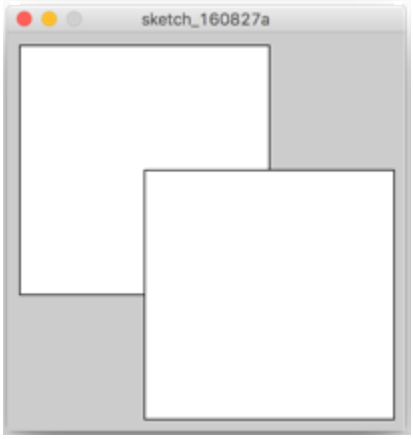

```
size(320, 320);
noStroke();
ellipse(110, 150, 200, 200);
```

3.11 順次

3.11.1 説明

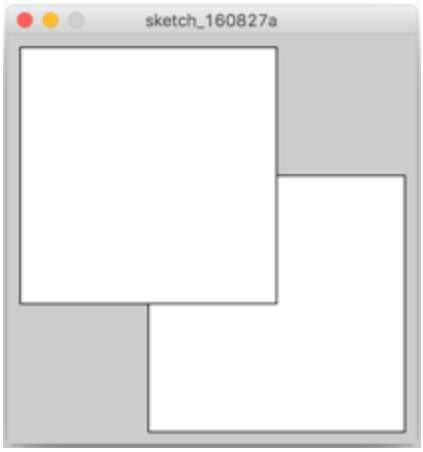
プログラムは上から順に実行される。

3.11.2 正方形を 2 つ描画。1 つめは左上が(10, 10)で、1 辺 200。2 つめは左上が(110, 110)で、1 辺 200。



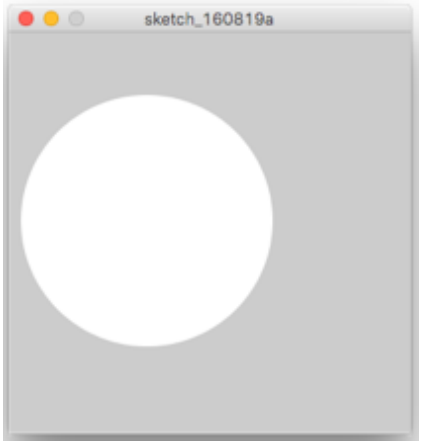

```
size(320, 320);
rect(10, 10, 200, 200);
rect(110, 110, 200, 200);
```

3.11.3 正方形を 2 つ描画。1 つめは左上が(110, 110)で、1 辺 200。2 つめは左上が(10, 10)で、1 辺 200。



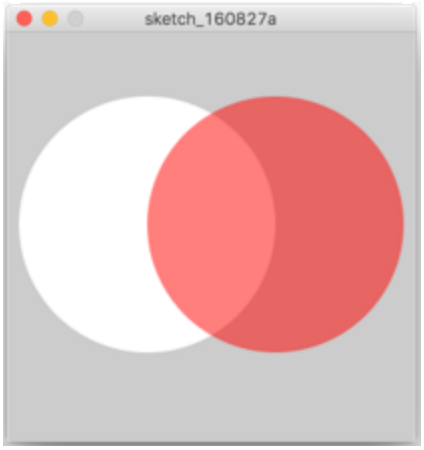
```
size(320, 320);
rect(110, 110, 200, 200);
rect(10, 10, 200, 200);
```

3.11.4 円を描画。中心が(110, 150)で、直径が 200、線を描かない。




```
size(320, 320);
noStroke();
ellipse(110, 150, 200, 200);
```

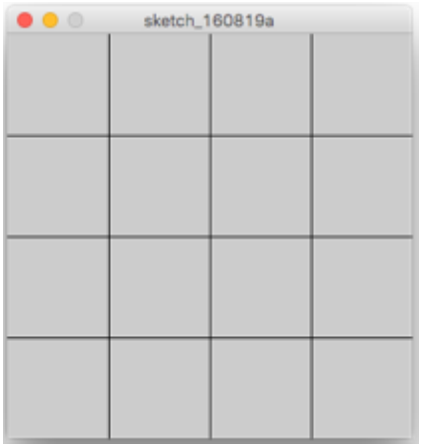
3.11.5 円を 2 つ描画。1 つめは中心が(110, 150)で、直径が 200、線を描かない、塗りが白(デフォルト)。2 つめは中心が(210, 150)で、直径が 200、線を描かない、塗りが赤で半透明。




```
size(320, 320);
noStroke();
```

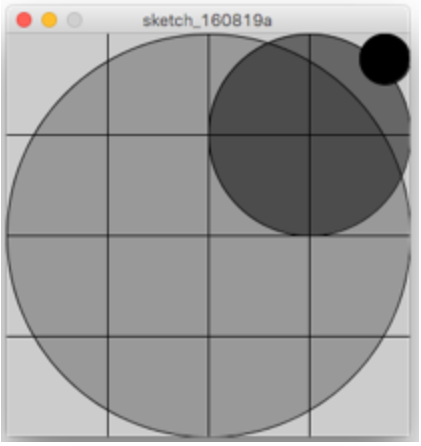
```
ellipse(110, 150, 200, 200);
fill(255, 0, 0, 127);
ellipse(210, 150, 200, 200);
```

3.11.6 次のように描画させる。




```
size(320, 320);
line(80, 0, 80, 320);
line(160, 0, 160, 320);
line(240, 0, 240, 320);
line(0, 80, 320, 80);
line(0, 160, 320, 160);
line(0, 240, 320, 240);
```

3.11.7 次のように描画させる。




```
size(320, 320);
fill(128, 128);
ellipse(160, 160, 320, 320);
fill(64, 192);
ellipse(240, 80, 160, 160);
fill(32);
ellipse(300, 20, 40, 40);

line(80, 0, 80, 320);
line(160, 0, 160, 320);
line(240, 0, 240, 320);
line(0, 80, 320, 80);
line(0, 160, 320, 160);
line(0, 240, 320, 240);
```

3.11.8 日の丸を作る。ウィンド・サイズは 300x200。

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

3.11.9 フランス国旗を作る。ウィンド・サイズは 300x200。

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

```
background(255);
noStroke();
fill(0, 0, 255);
rect(0, 0, 100, 200);
fill(255, 0, 0);
rect(200, 0, 100, 200);
```

3.11.10 ドイツ国旗を作る。ウィンド・サイズは 300x200。

```
size(500, 300);
background(0);
noStroke();
fill(255, 0, 0);
rect(0, 100, 500, 100);
fill(255, 255, 0);
rect(0, 200, 500, 100);
```

4.変数と型

4.1 変数の宣言と型

4.1.1 説明

```
変数の宣言方法
変数の型 変数名;

型の種類
整数 int
小数 float
真偽値 boolean
文字 char
文字列 String
```

4.1.2 整数の変数ｉの宣言

```
int i;
```

4.1.3 小数の変数 x の宣言

```
float x;
```

4.2 変数への値の代入

4.2.1 説明

代入の方法

変数名 = 値;

4.2.2 整数の変数 i を宣言し、10 を代入し、コンソールに表示する。

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

4.2.3 小数の変数 x を宣言し、1.3 を代入し、コンソールに表示する。

```
float x;
x = 1.3;
println(x);
```

4.3 変数の初期化

4.3.1 説明

変数宣言と代入を同時に行う方法

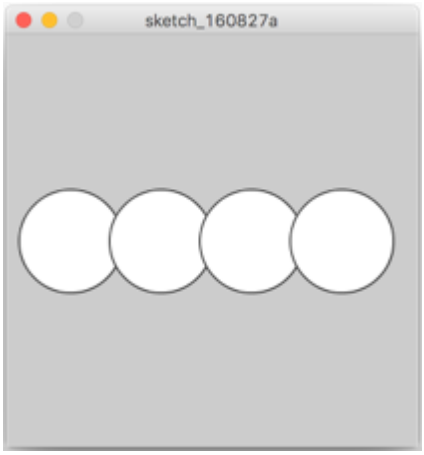
型 変数名 = 値;

4.3.2 整数の変数 i を 10 で初期化し、コンソールに表示する。

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

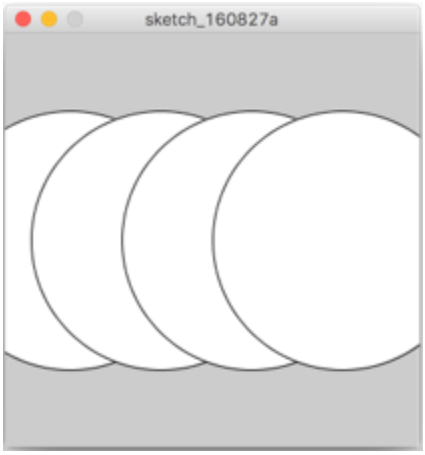
4.4 変数を活用した描画

4.4.1 円を 4 つ描く。直径はすべて 80。中心(50、160)、(120、160)、(190、160)、(260、160)。



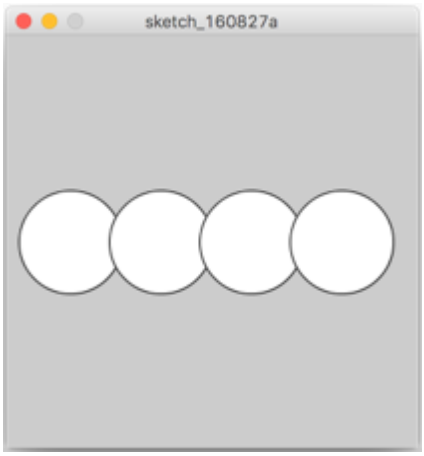

```
size(320, 320);
ellipse(50, 160, 80, 80);
ellipse(120, 160, 80, 80);
ellipse(190, 160, 80, 80);
ellipse(260, 160, 80, 80);
```

4.4.2 円を 4 つ描く。直径はすべて 200。中心(50、160)、(120、160)、(190、160)、(260、160)。



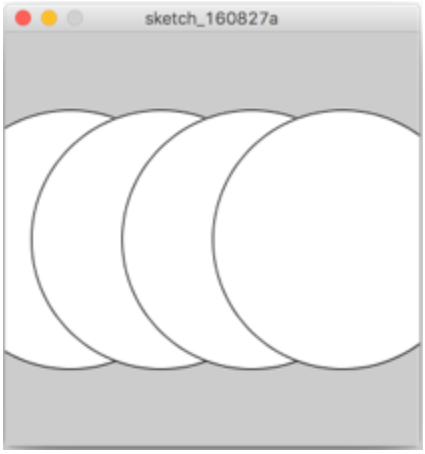

```
size(320, 320);
ellipse(50, 160, 200, 200);
ellipse(120, 160, 200, 200);
ellipse(190, 160, 200, 200);
ellipse(260, 160, 200, 200);
```

4.4.3 円を 4 つ描く。直径には変数 d を指定し、その値は 80。中心(50、160)、(120、160)、(190、160)、(260、160)。




```
size(320, 320);
int d = 80;
ellipse(50, 160, d, d);
ellipse(120, 160, d, d);
ellipse(190, 160, d, d);
ellipse(260, 160, d, d);
```

4.4.4 円を 4 つ描く。直径には変数 d を指定し、その値は 200。中心(50、160)、(120、160)、(190、160)、(260、160)。




```
size(320, 320);
int d = 200;
```

```
ellipse(50, 160, d, d);
ellipse(120, 160, d, d);
ellipse(190, 160, d, d);
ellipse(260, 160, d, d);
```

4.5 ディスプレイ・ウィンドウの変数

4.5.1 説明

プロセッシングがあらかじめ用意している変数

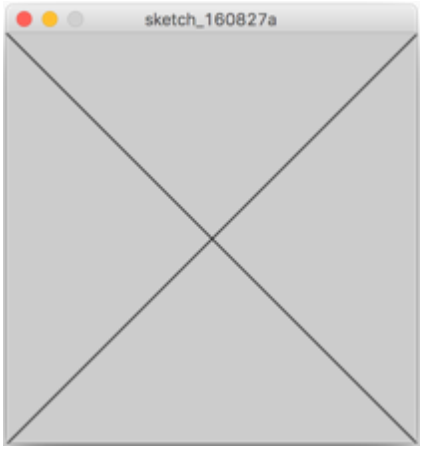
ディスプレイ・ウィンドウの幅: width ウィドゥス

ディスプレイ・ウィンドウの高さ: height ハイト

4.5.2 ディスプレイ・ウィンドウのサイズを 100x300 に設定し、ディスプレイ・ウィンドウの幅と高さをコンソールに表示する。

```
size(100, 300);
println(width);
println(height);
```

4.5.3 ディスプレイ・ウィンドウの変数を用いて以下のように描画する。




```
size(500, 300);
line(0, 0, width, height);
line(width, 0, 0, height);
```

5.演算子

5.1 さまざまな演算子

5.1.1 説明

代入 =

足す、文字列を連結する +

引く -

かける *

割る /

割ったあまり %

足して代入 +=

引いて代入 -=

5.1.2 コンソールに、12 足す 3 の結果を表示する。

```
println(12 + 3);
```

5.1.3 整数の変数を i と j を用意し、それぞれ 12 と 3 を代入し、足した結果をコンソールに表示する。

```
int i = 12;
int j = 3;
println(i + j);
```

5.1.4 整数の変数 i を 12 で初期化する。それをコンソールに表示する。次に 3 増やし、コンソールに表示する。

```
int i = 12;
println(i);
i += 3;
println(i);
```

5.1.5 コンソールに"hello"と"world"をつなげた文字列を表示する。

```
println("hello" + " world");
```

5.2 自動型変換と演算子の優先順位

5.2.1 説明

整数 と 整数 の演算結果は 整数

小数 と 小数 の演算結果は 小数

小数 と 整数 の演算結果は 小数

文字列 と 数値 の演算結果は 文字列

5.2.2 コンソールに、100 割る 3 の結果を表示する。

```
println(100 / 3);
```

5.2.3 コンソールに 100.0 割る 3.0 の結果を表示する。

```
println(100.0 / 3.0);
```

5.2.4 コンソールに 100.0 割る 3 の結果を表示する。

```
println(100.0 / 3);
```

5.2.5 以下のプログラムを実行すると、数値の部分が 123 と表示される。正しく表示するように修正せよ。

```
println("合計は" + 12 + 3 + "です。");
```

```
println("合計は" + (12 + 3) + "です。");
```

5.3 代入の制限と型変換

5.3.1 説明

小数 の変数に 整数 を代入すると小数に変換される

整数 の変数に 小数 を代入することはできない。int()で変換する。

5.3.2 小数の変数 x を 12 で初期化する。それをコンソールに表示する。

```
float x = 12;
println(x);
```

5.3.3 次のプログラムを実行しようとするとエラーになる。型変換を利用し、エラーが出なくなるようにせよ。

```
float x = 100.0;
float y = 3;
int i = x / y;
println(i);
```

```
float x = 100.0;
float y = 3;
int i = int(x / y);
println(i);
```

6.くりかえし

6.1 for 文

6.1.1 説明

```
for ( 初期化; 繰り返し条件; 増分 ) {
  繰り返したい処理;
}
```

条件式

等しい　　==

等しくない !=

より大きい >

以上　　>=

以下　　<=

より小さい <

論理式

かつ　&&

または ||

でない !

6.1.2 コンソールに 0 から 10 まで 1 つずつ表示せよ。

```
for (int i = 0; i <= 10; i++) {
  println(i);
}
```

6.1.3 コンソールに 1 から 10 まで 1 つずつ表示せよ。

```
for (int i = 1; i <= 10; i++) {
  println(i);
}
```

6.1.4 コンソールに 0、1、2、…と 10 個(9 まで)の数字を表示せよ。

```
for (int i = 0; i < 10; i++) {
  println(i);
}
```

6.1.5 コンソールに 7、8、9、…、13 と表示せよ。

```
for (int i = 7; i <= 13; i++) {
  println(i);
}
```

6.1.6 コンソールに 10、9、…、0 と表示せよ。

```
for (int i = 10; i >= 0; i--) {
  println(i);
}
```

6.1.7 コンソールに 1、3、5、…、19 と表示せよ。

```
for (int i = 1; i < 20; i += 2) {
  println(i);
}
```

6.1.8 コンソールに 2、4、8、…、128 と表示せよ。

```
for (int i = 2; i < 200; i *= 2) {
  println(i);
```

```
}
```

6.2 for 文による描画

6.2.1 図のように描画せよ。

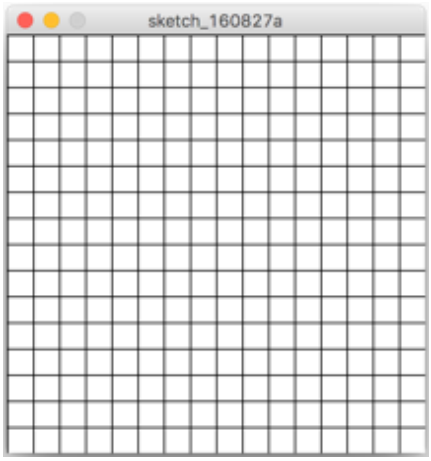

```
size(320, 320);
for (int x = 0; x < width; x += 20) {
  rect(x, 0, 20, 20);
}
```

6.2.2 図のように描画せよ。


```
size(320, 320);
for (int x = 0; x < width; x += 20) {
  //rect(x, 0, 20, 20);
  ellipse(x + 10, 0 + 10, 20, 20);
}
```

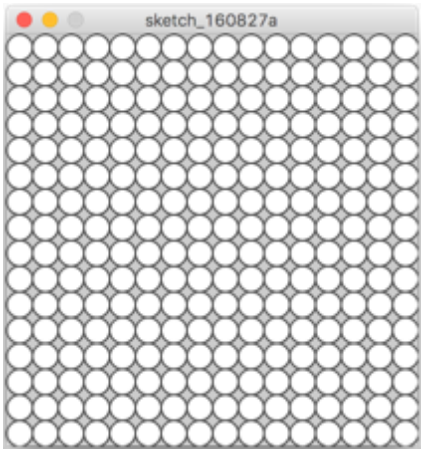
6.3 2 重ループ

6.3.1 図のように描画せよ。




```
size(320, 320);
for (int y = 0; y < height; y += 20) {
  for (int x = 0; x < width; x += 20) {
    rect(x, y, 20, 20);
    //ellipse(x + 10, 0 + 10, 20, 20);
  }
}
```

6.3.2 図のように描画せよ。




```
size(320, 320);
for (int y = 0; y < height; y += 20) {
  for (int x = 0; x < width; x += 20) {
```

```
    //rect(x, y, 20, 20);
    ellipse(x + 10, y + 10, 20, 20);
  }
}
```

7.アニメーション

7.1 説明

円を移動させるアニメーション

```
// 全体で使う変数
int x = 100;

// 最初に一回だけ実行される処理
void setup() {
  size(320, 320);
}

// 毎フレーム実行する処理
void draw() {
  background(255);
  ellipse(x, 100, 50, 50);
  x += 1;
}
```

7.2 フレーム・レートの設定

7.2.1 説明

1秒間に切り替えられるフレームの数

フレームレートの値の設定　　frameRate(数値);

フレームレートの現在の値　　frameRate

7.2.2 最初にフレーム・レートを 20 に設定し、実測値をコンソールに毎フレーム表示する。


```
void setup() {
  frameRate(20);
}

void draw() {
  println(frameRate);
}
```

7.3 経過時間

7.3.1 説明

プログラムを起動してからの経過時間(ms)

7.3.2 経過時間(ミリ秒)をコンソールに毎フレーム表示する。


```
void setup() {
}

void draw() {
  println(millis());
}
```

7.3.3 経過時間(秒)をコンソールに毎フレーム表示する。


```
void setup() {
}

void draw() {
  println(millis()/1000);
}
```

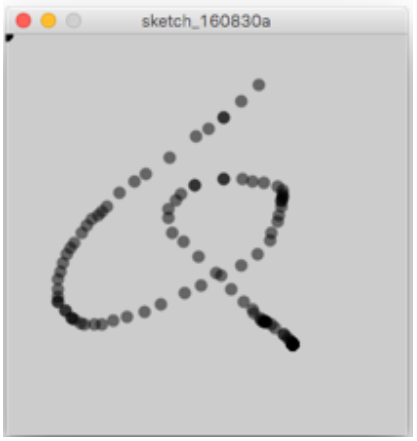
7.4 マウス変数

7.4.1 説明

マウスの x,y 座標: mouseX, mouseY

前のフレームのマウスの x,y 座標: pmouseX, pmouseY

7.4.2 マウスの位置に点を描画する。描線の太さは 10、色は黒の半透明。




```
void setup() {
  size(320, 320);
  strokeWeight(10);
  stroke(0, 127);
}
```

```
void draw() {
  point(mouseX, mouseY);
}
```

7.4.3 マウスの現在の位置と 1 フレーム前の位置を結ぶ線を描画する。描線の太さは 10。

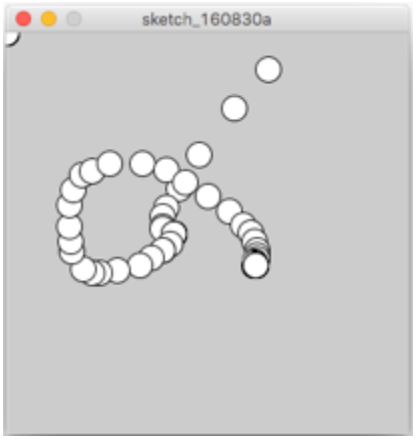

```
void setup() {
  size(320, 320);
  strokeWeight(10);
  stroke(0, 127);
}

void draw() {
  //point(mouseX, mouseY);
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

7.5 残像と background()関数

7.5.1 説明

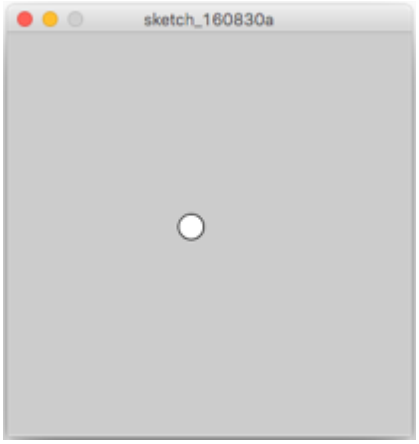
毎フレーム、ウィンド全体を背景色で塗りつぶす

7.5.2 マウスの位置に直径 20 の円を描く。




```
void setup() {
  size(320, 320);
}

void draw() {
  ellipse(mouseX, mouseY, 20, 20);
}
```

7.5.3 マウスの位置に直径 20 の円を描く。残像が出ないようにする。




```
void setup() {
  size(320, 320);
}

void draw() {
```

```
  background(192);
  ellipse(mouseX, mouseY, 20, 20);
}
```

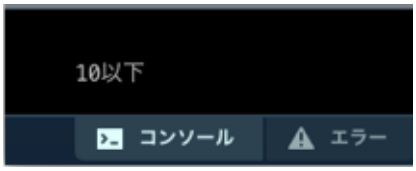
8.条件分岐

8.1 if 文

8.1.1 説明

```
if（条件文）{
  条件が真のとき実行する処理
}

if（条件文）{
  条件が真のとき実行する処理
} else {
  条件が偽のとき実行する処理
}
```

8.1.2 整数の変数 i を適当な値で初期化する。i の値が 10 以下のときに「10 以下」とコンソールに表示




```
int i = 10;
if (i <= 10) {
  println("10 以下");
}
```

8.1.3 整数の変数 i を適当な値で初期化する。i の値が 10 以下のときに「10 以下」、そうで

ないときに「10 より大きい」とコンソールに表示


```
int i = 17;
if (i <= 10) {
  println("10 以下");
} else {
  println("10 より大きい");
}
```

8.1.4 整数の変数 i を適当な値で初期化する。i の値が 0 以上 10 以下のときに「0 以上 10 以下」、そうでないときに「0 より小さいか 10 より大きい」とコンソールに表示


```
int i = 5;
if (0 <= i && i <= 10) {
  println("0 以上 10 以下");
} else {
  println("0 より小さいか 10 より大きい");
}
```

9. マウス

9.1 マウス変数 ※7.3 で説明しなかったもの

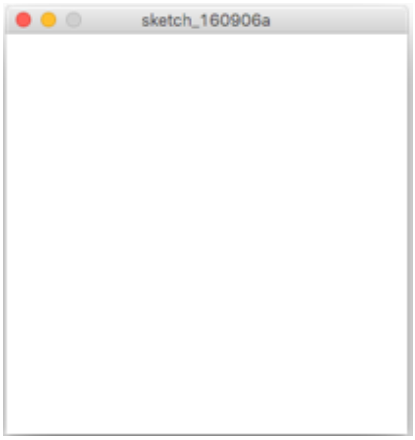
9.1.1 説明

マウスが押されたかどうか調べる変数: mousePressed … true/false

押されたマウスのボタンを調べる変数: mouseButton … LEFT/RIGHT/CENTER

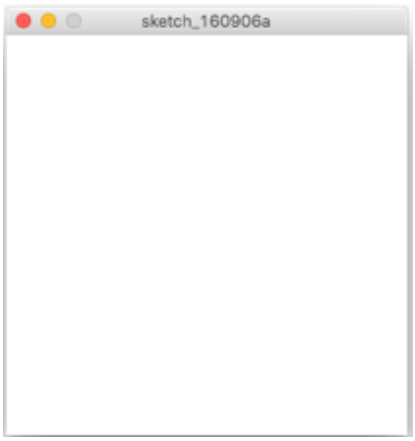
マウスが押されたとき一度だけ呼ばれる関数: mousePressed()

9.1.2 マウスを押したとき背景を黒に、押していないとき背景を白に



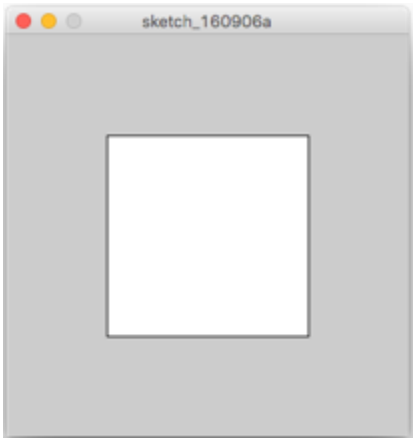

```
void setup() {
  size(320, 320);
}

void draw() {
  if (mousePressed) {
    background(0);
  } else {
    background(255);
  }
}
```

9.1.2 マウスを押したとき左ボタンなら背景を赤に、それ以外のボタンなら黒に、押していないとき背景を白に



```
void setup() {
  size(320, 320);
}

void draw() {
  if (mousePressed) {
    if (mouseButton == LEFT) {
      background(255, 0, 0);
    }
    else {
      background(0);
    }
  }
  else {
    background(255);
  }
}
```

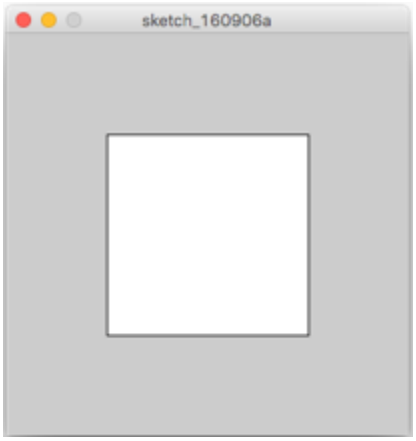
9.1.3 1 辺 160 の正方形を中央に描く。




```
void setup() {
  size(320, 320);
}

void draw() {
```

```
  rect(80, 80, 160, 160);
}
```

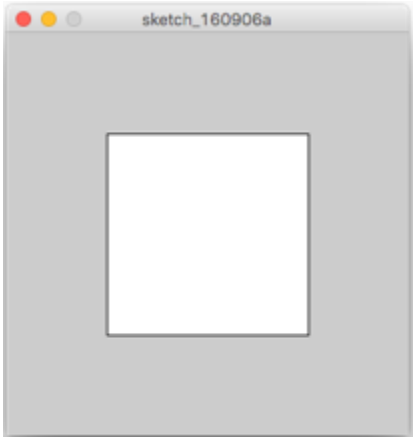
9.1.4 1 辺 160 の正方形を中央に描く。マウスの x 座標が 80 以上なら色を黒く、そうでなければ白く塗る。




```
void setup() {
  size(320, 320);
}

void draw() {
  if (mouseX >= 80) {
    fill(0);
  } else {
    fill(255);
  }
  rect(80, 80, 160, 160);
}
```

9.1.5 1 辺 160 の正方形を中央に描く。マウスの x 座標が 80 以上、y 座標が 80 以上なら色

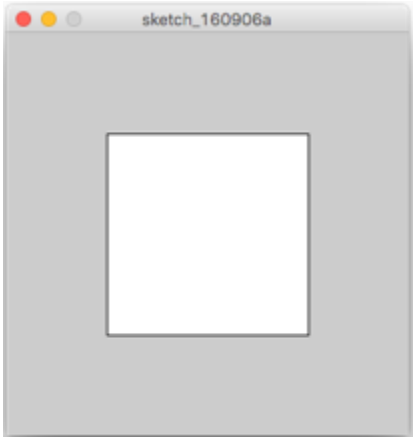
を黒く、そうでなければ白く塗る。




```
void setup() {
  size(320, 320);
}

void draw() {
  if (mouseX >= 80 && mouseY >= 80) {
    fill(0);
  } else {
    fill(255);
  }
  rect(80, 80, 160, 160);
}
```

9.1.6 1 辺 160 の正方形を中央に描く。マウスの x 座標が 80 以上 240 以下、y 座標が 80 以

上 240 以下なら色を黒く、そうでなければ白く塗る。




```
void setup() {
  size(320, 320);
}

void draw() {
  if (mouseX >= 80 && mouseY >= 80 && mouseX <= 240 && mouseY <= 240) {
    fill(0);
  } else {
    fill(255);
  }
  rect(80, 80, 160, 160);
}
```

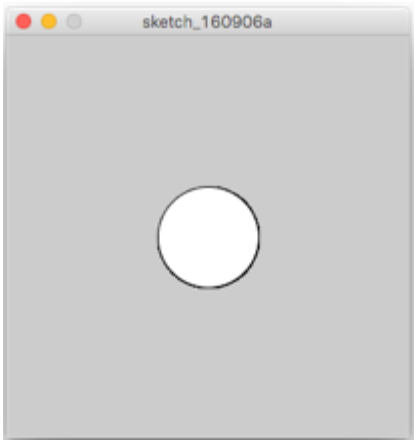
9.2 円形の当たり判定と dist 関数

9.2.1 説明

2 点間の距離を求める： dist(x1, y1, x2, y2)

9.2.2 マウスと画面の中心の間の距離をコンソールに表示

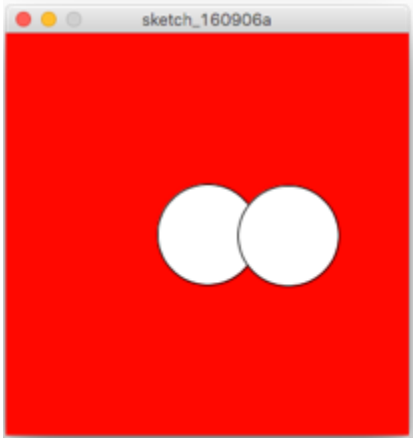
9.2.3 画面中央に半径 40(直径 80)の円を描く。マウスが円の中に入っていれば黒、そうでなければ白で円を塗る。




```
void setup() {
  size(320, 320);
}

void draw() {
  if (dist(mouseX, mouseY, 160, 160) <= 80) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(160, 160, 160, 160);
}
```

9.2.4 画面中央とマウスを中央に半径 40(直径 80)の円をそれぞれ描く。2 つの円が交差し

ていれば背景を赤、そうでなければ灰色にする。



10. キー

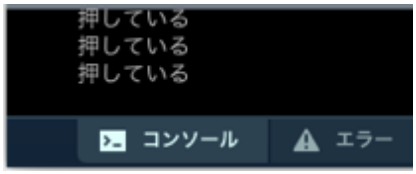
10.1 キー変数

10.1.1 説明

キーボードが押されたか調べる変数: keyPressed

押されたキーの種類を調べる変数: key

10.1.2 キーを押しているとコンソールに「押している」、そうでなければ「押していない」と表示




```
void setup() {
}

void draw() {
  if (keyPressed) {
    println("押している");
  }
}
```

10.1.3 キーを押しているとコンソールにその値を、そうでなければ「押していない」と表示

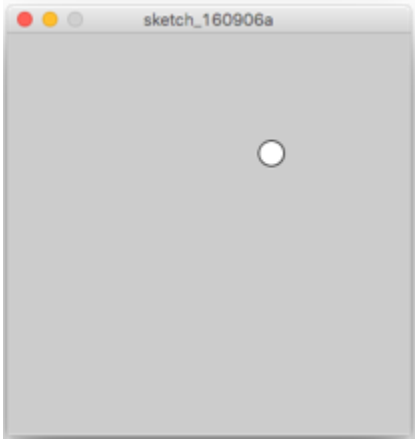

```
void setup() {
}

void draw() {
  if (keyPressed) {
    println(key);
  }
}
```

10.1.4 修飾キー以外のキーを押しているとコンソールにその値を、上下キーは、それぞれ「上」「下」を、そうでなければ「押していない」と表示。


```
void setup() {
}

void draw() {
  if (keyPressed) {
    if (key != CODED) {
      println(key);
    }
    else {
      if (keyCode == UP) {
        println("上");
      }
      else if (keyCode == DOWN) {
```

```
        println("下");
      }
      else {
        println("押していない");
      }
    }
  }
}
```

10.1.5 画面中央に直径 20 の円を表示する。上下左右キーで、上下左右に移動させる。




```
float x, y;

void setup() {
  size(320, 320);
  x = width / 2;
  y = height / 2;
}

void draw() {
  background(192);
  int v = 5;
  if (keyPressed) {
    if (key == CODED) {
      if (keyCode == UP) {
```

```
      y -= v;
    }
    else if (keyCode == DOWN) {
      y += v;
    }
    else if (keyCode == RIGHT) {
      x += v;
    }
    else if (keyCode == LEFT) {
      x -= v;
    }
  }
 }
 ellipse(x, y, 20, 20);
}
```

11.画像

11.1 PImage クラスと image()関数

11.1.1 説明

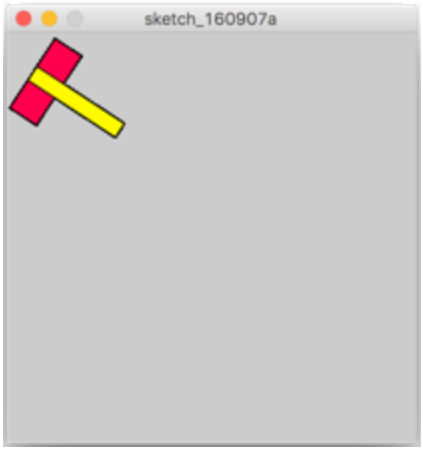
(1) 画像変数宣言:　PImage 変数名;

(2) 画像を読み込み:　変数名 = loadImage(ファイル名);

(3) 画像を表示する:　image(変数名, x, y);

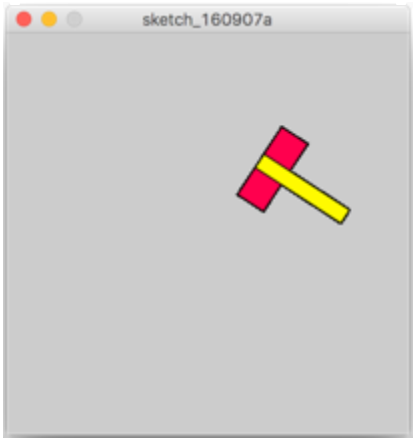
11.1.2 画像 hammer.png を表示せよ。具体的には、画像を表す変数 hammer を用意し、画像

を読み込み、画像の左上の座標を(0, 0)にして表示すること。




```
PImage hammer;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
}

void draw() {
  background(192);
  image(hammer, 0, 0);
}
```

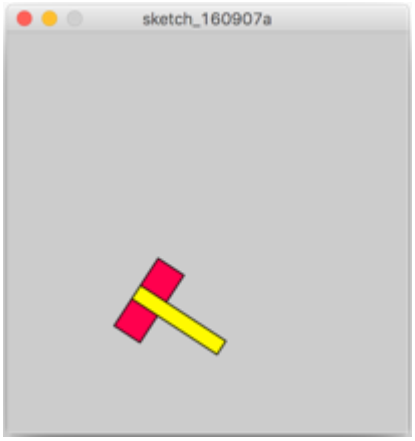
11.1.3 画像 hammer.png を、画像の左上の座標を(180, 70)にして表示




```
PImage hammer;
```

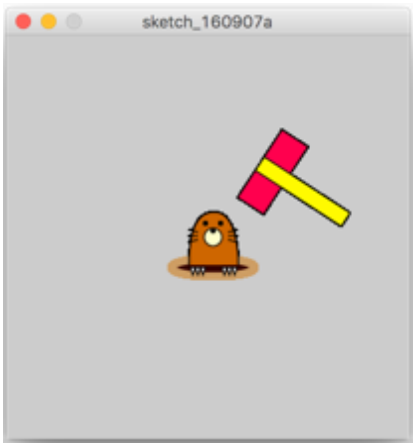
```
void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
}

void draw() {
  background(192);
  image(hammer, 180, 70);
}
```

11.1.4　画像 hammer.png を、画像の左上の座標をマウスの中心にして表示。残像は消すこと。




```
PImage hammer;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
}

void draw() {
  background(192);
  image(hammer, mouseX, mouseY);
}
```

11.1.5 画像 hammer.png を、画像の左上の座標を(180, 70)にして表示。画像 mole.png を、画像の左上の座標を(128, 128)にして表示。



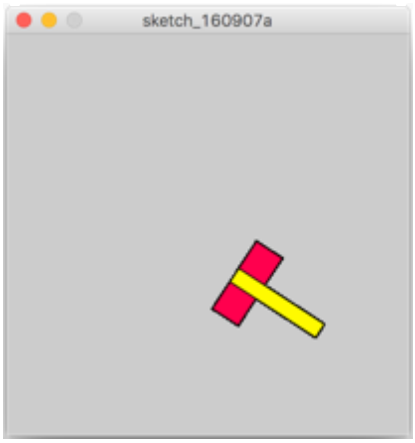

```
PImage hammer;
PImage mole;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
  mole = loadImage("mole.png");
}

void draw() {
  background(192);
  image(mole, 160, 160);
  image(hammer, mouseX, mouseY);
}
```

11.2 imageMode()関数

11.2.1 説明

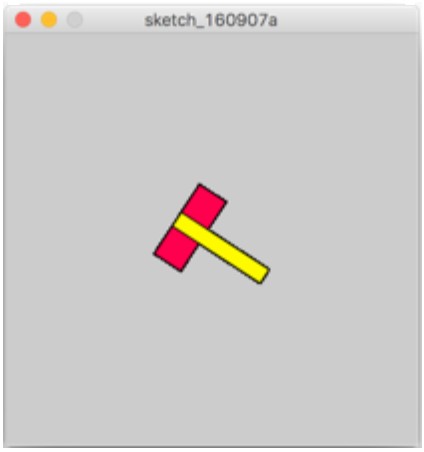
11.2.2 画像 hammer.png を、画像の左上の座標をディスプレイ・ウィンドウの中心にして

表示



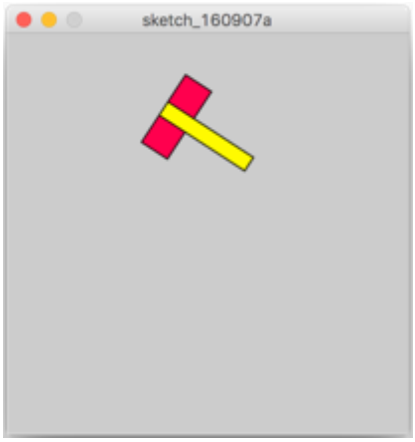

```
PImage hammer;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
}

void draw() {
  background(192);
  image(hammer, width/2, height/2);
}
```

11.2.3 画像 hammer.png を、画像の中心の座標をディスプレイ・ウィンドウの中心にして表示



```
PImage hammer;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
  imageMode(CENTER);
}

void draw() {
  background(192);
  image(hammer, width/2, height/2);
}
```

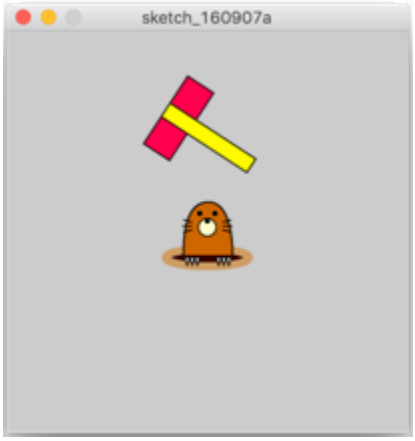
11.2.4　画像 hammer.png を、画像の中心の座標をマウスの中心にして表示。残像は消すこと。




```
PImage hammer;

void setup() {
  size(320, 320);
  hammer = loadImage("hammer.png");
  imageMode(CENTER);
}

void draw() {
```

```
  background(192);
  image(hammer, mouseX, mouseY);
}
```

11.2.5　画像 hammer.png を、画像の中心の座標をマウスの中心にして表示。画像 mole.png を、ディスプレイ・ウィンドウの中心に表示。残像は消すこと。




```
PImage hammer;
PImage mole;

void setup() {
  size(320, 320);
  imageMode(CENTER);
  hammer = loadImage("hammer.png");
  mole = loadImage("mole.png");
}

void draw() {
  background(192);
  image(mole, 160, 160);
  image(hammer, mouseX, mouseY);
}
```

12. フォントと文字列の表示

12.1 PFont クラスと text()関数

12.1.1 説明

(1) フォント変数宣言:　　PFont 変数名;

(2) フォントを生成:　　変数名 = createFont(フォント名, サイズ);

(3) フォント指定:　　textFont(変数名);

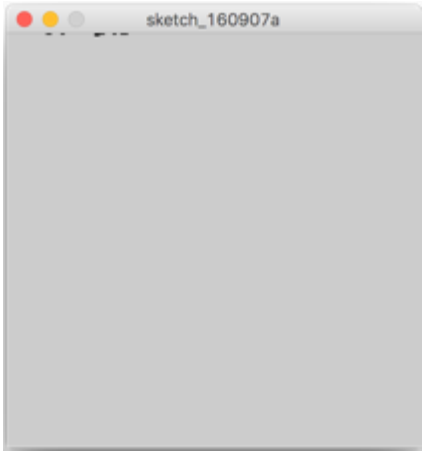
(4) 文字を表示する:　　text(文字列, x, y);

12.1.2 サイズが 20 ポイントの「Osaka」フォントを使い、「こんにちは」と座標(20, 20)に表示


```
PFont font;

void setup() {
  size(320, 320);
  font = createFont("Osaka", 20);
}

void draw() {
  textFont(font);
  fill(0);
  text("こんにちは", 20, 20);
}
```

12.1.3 サイズが 20 ポイントの「Osaka」フォントを使い、「こんにちは」と座標(0, 0)に表

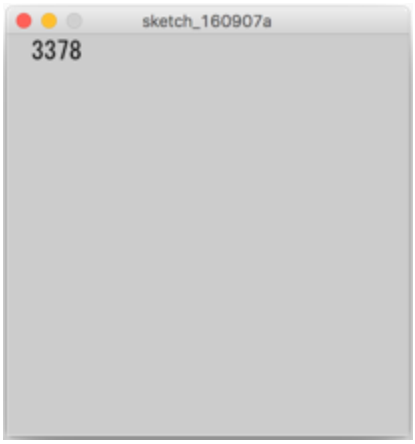
示




```
PFont font;

void setup() {
  size(320, 320);
  font = createFont("Osaka", 20);
}

void draw() {
  textFont(font);
  fill(0);
  text("こんにちは", 0, 0);
}
```

12.1.4 サイズが 20 ポイントの「Osaka」フォントを使い、アプリが実行を開始してからの

時間(単位はミリ秒)を座標(20, 20)に表示。残像は消すこと。




```
PFont font;

void setup() {
  size(320, 320);
  font = createFont("Osaka", 20);
}

void draw() {
  background(192);
  textFont(font);
  fill(0);
  text(millis(), 20, 20);
}
```

13. 乱数

13.1 説明

0 以上、値未満のでたらめな数: random(値)

下限以上、上限未満のでたらめな数: random(下限, 上限)

13.2 0 から 10 の間の乱数をコンソールに表示(10 は含まない)


```
println( random(10) );
```

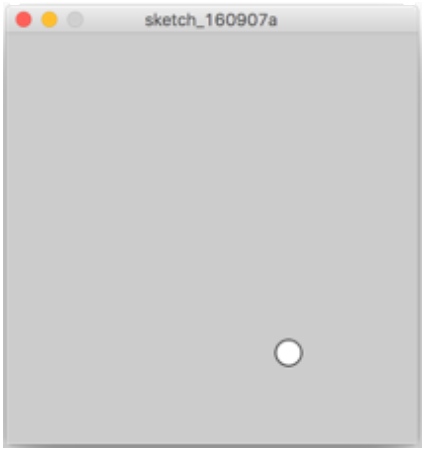
13.3 サイコロをふって出た目の数を表示


```
println((int)random(1, 7));
```

13.3 サイコロをふって出た目の数を 5 つ表示

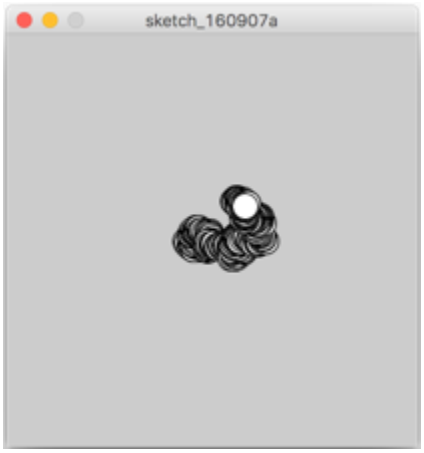

```
for (int i = 0; i < 5; i++) {
  println((int)random(1, 7));
}
```

13.4 画面の中のランダムな位置に直径 20 の円を表示




```
ellipse(random(width), random(height), 20, 20);
```

13.5　画面の中央に直径 20 の円を表示。毎フレーム、その円をランダムに移動させる(上下左右に最大 2)。




```
float x, y;

void setup() {
  size(320, 320);
  x = width / 2;
  y = height / 2;
}

void draw() {
  ellipse(x, y, 20, 20);
  x += random(-2, 2);
  y += random(-2, 2);
}
```

14. 関数

14.1 説明

関数の使い方

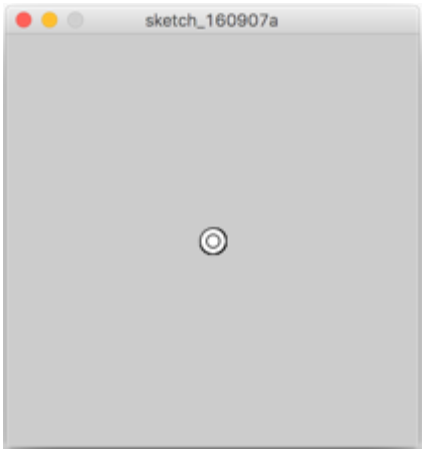
(1) 関数の定義

```
    戻り値 関数名(引数の型 引数名, … ) {
        実行させたい内容
        return 戻り値;
    }
```

(2) 関数の呼び出し

変数名 = 関数名(関数に渡す値, …);

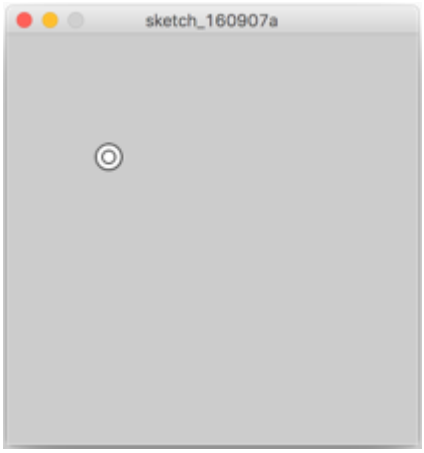
14.2 画面の中央に◎を表示する関数 bullseye()を宣言し、呼出す。大きい円の直径 20、小さい円の直径 10 とする。




```
void setup() {
  size(320, 320);
}

void draw() {
  bullseye();
}

void bullseye() {
  ellipse(width / 2, height / 2, 20, 20);
  ellipse(width / 2, height / 2, 10, 10);
}
```

14.3 引数で指定した座標を中心に◎を表示する関数 bullseye()を宣言する。そして、引数としてマウスの座標を指定して呼出す。大きい円の直径 20、小さい

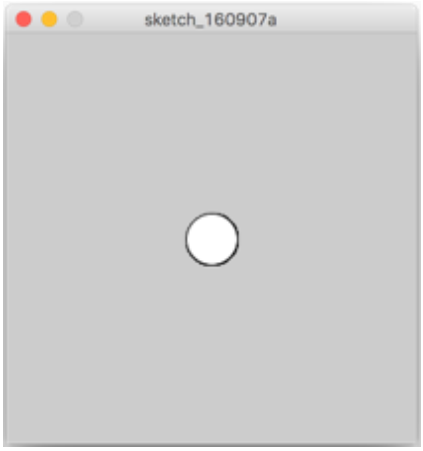
円の直径 10 とする。残像は消すこと。




```
void setup() {
  size(320, 320);
}

void draw() {
  background(192);
  bullseye(mouseX, mouseY);
}

void bullseye(float _x, float _y) {
  ellipse(_x, _y, 20, 20);
  ellipse(_x, _y, 10, 10);
}
```

14.4 画面の中央に直径 40 の円を表示する。この円の中にマウスが入ると true、入っていないと false を返す関数 inside()を宣言する。inside()の戻り値をコ

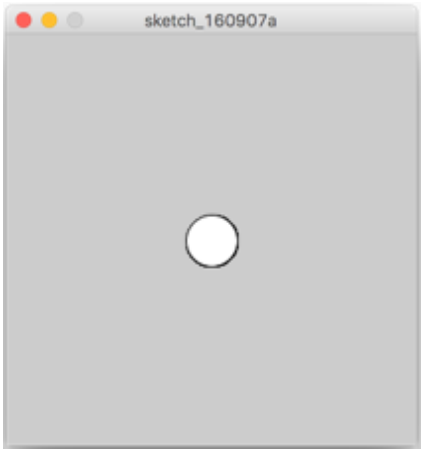
ンソールに表示




```
void setup() {
  size(320, 320);
}

void draw() {
  ellipse(width/2, height/2, 40, 40);
  println(inside());
}

boolean inside() {
  if (dist(mouseX, mouseY, width/2, height/2) <= 40) {
    return true;
  }
  else {
    return false;
  }
}
```

14.5 画面の中央に直径40の円を表示する。この円の中にマウスが入るとtrue、

入っていないと false を返す関数 inside()を宣言する。inside()の戻り値の値を利用して、マウスが円の中に入ると円を黒く、そうでなければ白く塗りつぶす




```
void setup() {
  size(320, 320);
}

void draw() {
  if (inside()) {
    fill(0);
  }
  else {
    fill(255);
  }
  ellipse(width/2, height/2, 40, 40);
}

boolean inside() {
  if (dist(mouseX, mouseY, width/2, height/2) <= 40) {
    return true;
  }
  else {
    return false;
  }
}
```

15.オブジェクト

15.1 クラスとインスタンス

15.1.1 説明

オブジェクト: プログラムで扱う「もの」を一まとめに定義、部品として再利用できるようにする。

オブジェクト = 変数(フィールド) + 処理(メソッド)

クラス: オブジェクトの設計図

インスタンス: クラスから生成される実体 ← これをプログラムで操作

(1) クラス(設計図)を定義

```
class クラス名 {
  フィールド … オブジェクトで使う値のための変数
  コンストラクタ … クラスからインスタンス(実体)が生成されたときに
                   最初に一度だけ実行されるメソッド
  メソッド … オブジェクトで行う処理(関数)
}
```

(2) インスタンス変数の宣言

```
クラス名 変数名;
```

(3) インスタンス(実体)生成

```
インスタンス変数 = new クラス名();
```

(4) インスタンス操作

```
インスタンス変数.フィールド名
インスタンス変数.メソッド名()
```

15.1.2 ハンマーを表すクラス Hammer を宣言

```
class Hammer {
}
```

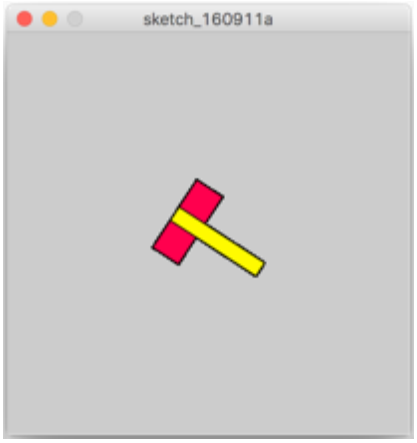
15.1.3 クラス Hammer に、画像 image、中心の座標 x、y をフィールドとして用意する

```
class Hammer {
  PImage image;
  float x, y;
}
```

15.1.4 クラス Hammer に、画面に表示する display()メソッドを用意する

```
class Hammer {
  PImage image;
  float x, y;

  void display() {
    image(image, x, y);
  }
}
```

15.1.5 クラス Hammer を使用するメイン・プログラムを作る。メイン・プログラムとクラス Hammer は別のタブにする。メイン・プログラムでは、インスタンスを生成し、画像 hammer.png と中心の座標を画面の中心に設定し、それを表示する。なお、imageMode(CENTER)を使い、画像の中心の座標を指定して表示するモードにすること。




```
Hammer h;

void setup() {
  size(320, 320);
```

```
  imageMode(CENTER);
  h = new Hammer();
  h.image = loadImage("hammer.png");
  h.x = width / 2;
  h.y = height / 2;
}

void draw() {
  h.display();
}
```

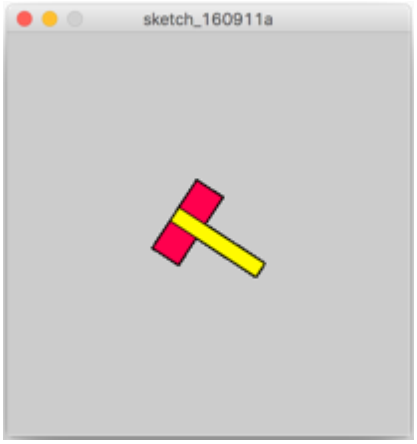
15.2 コンストラクタと this キーワード

15.2.1 説明

コンストラクタ: クラスからインスタンス生成時に一度だけ呼ばれる関数(初期化)

this: インスタンス内でインスタンス自身を指定

15.2.2 クラス Hammer にコンストラクタを用意し、それを利用してインスタンスを生成する。コンストラクタの中で image フィールドに loadImage("hammer.png")を代入して画像を初期化、座標 x, y をウィンドの中心に設定せよ。

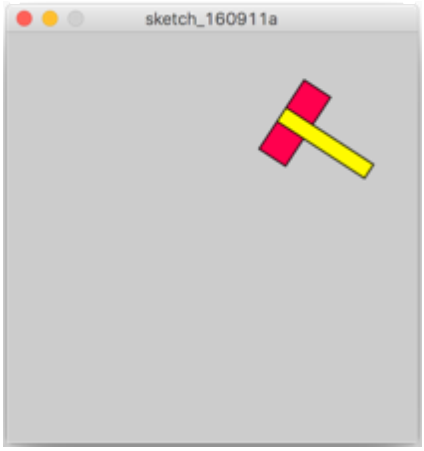



```
Hammer h;

void setup() {
  size(320, 320);
```

```
  imageMode(CENTER);
  h = new Hammer();
}

void draw() {
  h.display();
}

class Hammer {
  PImage image;
  float x, y;

  Hammer() {
    image = loadImage("hammer.png");
    x = width / 2;
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }
}
```

15.2.4 クラス Hammer に、マウスの座標の位置に移動させるメソッド move()を用意する。

それを用いて、マウスの位置に画像を表示する。残像は消すこと。



[メインタブ]

```
Hammer h;

void setup() {
  size(320, 320);
  imageMode(CENTER);
  h = new Hammer();
}

void draw() {
  background(192);
  h.move();
  h.display();
}
```

[Hammer タブ]

```
class Hammer {
  PImage image;
  float x, y;

  Hammer() {
    image = loadImage("hammer.png");
    x = width / 2;
```

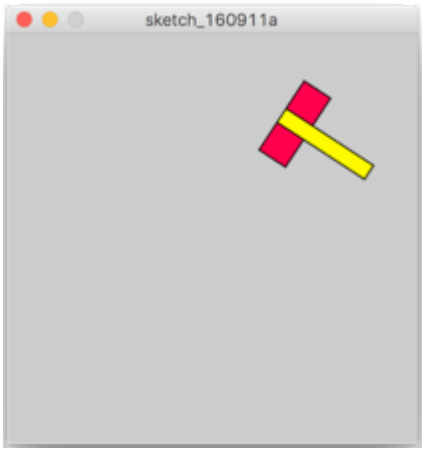
```
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }

  void move() {
    x = mouseX;
    y = mouseY;
  }
}
```

15.3 オーバーロード

15.3.1 説明

クラス内に同じ名前の関数を作ることができる(引数の数や型で区別する)

15.3.2 クラス Hammer にオーバーロードを用いて、もう一つのコンストラクタを用意する。もう 1 つのコンストラクタでは座標 x、y の 2 つを引数として受け取り、それを初期値に設定する。




```
class Hammer {
  PImage image;
  float x, y;
```

```
  Hammer() {
    image = loadImage("hammer.png");
    x = width / 2;
    y = height / 2;
  }

  Hammer(float _x, float _y) {
    image = loadImage("hammer.png");
    x = _x;
    y = _y;
  }

  void display() {
    image(image, x, y);
  }

  void move() {
    x = mouseX;
    y = mouseY;
  }
}
```

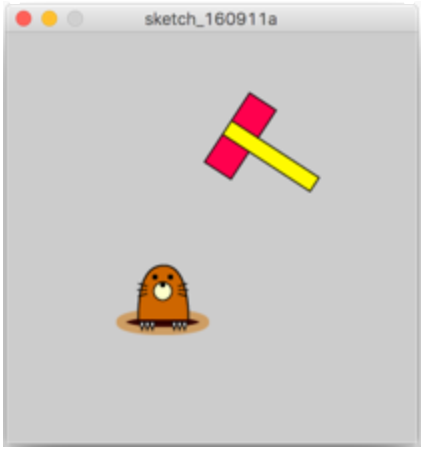
15.4 継承

15.4.1 説明

一つのクラスの性質(フィールド)や機能(メソッド)を引き継いで別のクラスを作り出すことができる。継承したクラスでさらに性質(フィールド)や機能(メソッド)を追加・拡張することができる。

15.4.2 画像 mole.png をランダムな位置に表示する Mole クラスを追加する。フィールドは

画像、x、y 座標。メソッドはランダムな位置への移動と表示が必要である。



[メインタブ]

```
Hammer h;
Mole m;

void setup() {
  size(320, 320);
  imageMode(CENTER);
  h = new Hammer();
  m = new Mole();
  m.move();
}

void draw() {
  background(192);
  m.display();
  h.move();
  h.display();
}
```

[Hammer タブ]

```
class Hammer {
  PImage image;
```

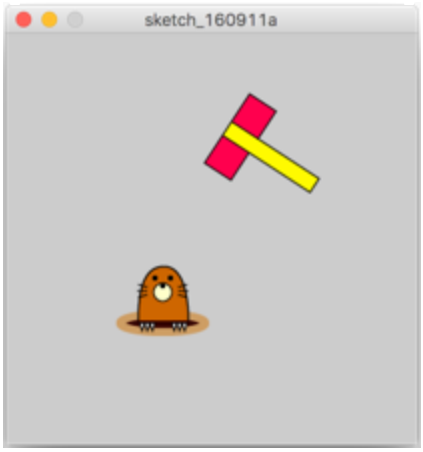
```
  float x, y;

  Hammer() {
    image = loadImage("hammer.png");
    x = width / 2;
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }

  void move() {
    x = mouseX;
    y = mouseY;
  }
}
```

[Moleタブ]

```
class Mole {
  PImage image;
  float x, y;

  Mole() {
    image = loadImage("mole.png");
    x = width / 2;
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }
```

```
  void move() {
    x = random(width);
    y = random(height);
  }
}
```

15.4.3 クラス Hammer と Mole には共通の部分がある。これを Sprite クラスとして取り出し、継承するようにする。



[メインタブ]

```
Hammer h;
Mole m;

void setup() {
  size(320, 320);
  imageMode(CENTER);
  h = new Hammer();
  m = new Mole();
  m.move();
}

void draw() {
  background(192);
  m.display();
  h.move();
```

```
  h.display();
}

[Hammer タブ]
class Hammer extends Sprite {
  Hammer() {
    image = loadImage("hammer.png");
    x = width / 2;
    y = height / 2;
  }

  void move() {
    x = mouseX;
    y = mouseY;
  }
}

[Mole タブ]
class Mole extends Sprite {
  Mole() {
    image = loadImage("mole.png");
    x = width / 2;
    y = height / 2;
  }

  void move() {
    x = random(width);
    y = random(height);
  }
}

[Sprite タブ]
class Sprite {
```

```
  PImage image;
  float x, y;

  void display() {
    image(image, x, y);
  }
}
```

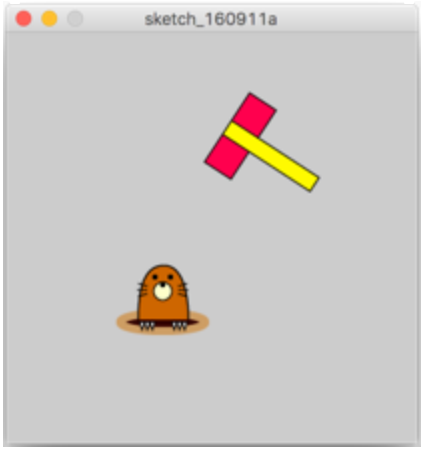
15.5 オーバーライドと super

15.5.1 説明

親クラスのメソッドを、子クラスの同じ名前のメソッドで上書きすることができる。

親クラスのメソッドを super.メソッド名()で呼び出すことができる。

15.5.2 クラス Hammer と Mole のコンストラクタには、座標を設定している部分がある。この部分について、Sprite クラスにコンストラクタを用意して、それを呼び出すように変更する。Hammer と Mole のコンストラクタでは、super を必ず最初に記述する必要があることに注意する。



[メインタブ]

```
Hammer h;
Mole m;

void setup() {
  size(320, 320);
  imageMode(CENTER);
```

```
  h = new Hammer();
  m = new Mole();
  m.move();
}

void draw() {
  background(192);
  m.display();
  h.move();
  h.display();
}
```

[Hammer タブ]

```
class Hammer extends Sprite {
  Hammer() {
    super();
    image = loadImage("hammer.png");
  }

  void move() {
    x = mouseX;
    y = mouseY;
  }
}
```

[Mole タブ]

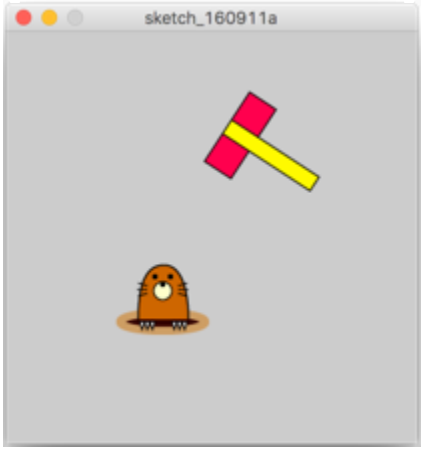
```
class Mole extends Sprite {
  Mole() {
    super();
    image = loadImage("mole.png");
  }

  void move() {
```

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

[Sprite タブ]

```
class Sprite {
  PImage image;
  float x, y;

  Sprite () {
    x = width / 2;
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }
}
```

15.5.3 クラス Hammer と Mole にはどちらも move()メソッドがある。Sprite クラスにも空の move()メソッドを用意する。



[メインタブ]

```
        (省略)
```

[Hammer タブ]

（省略）

[Mole タブ]

（省略）

[Sprite タブ]

```
class Sprite {
  PImage image;
  float x, y;

  Sprite () {
    x = width / 2;
    y = height / 2;
  }

  void display() {
    image(image, x, y);
  }

  void move() {
  }
}
```

15.6 多態性

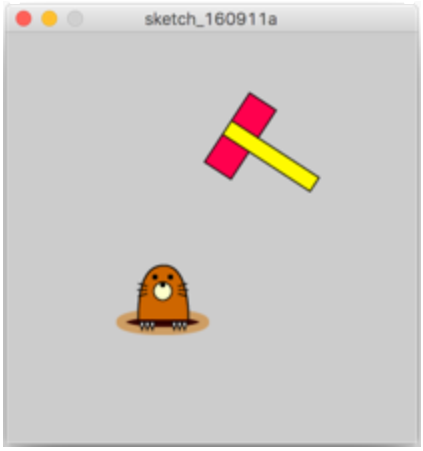
15.6.1 説明

親クラスの変数に子クラスのインスタンスを代入することができる。

操作が同じならば、どのインスタンスであるか知っている必要はない。

15.6.2 メイン・プログラムで、Hammer クラス、Mole クラスの変数を使用しているが、これ

らを Sprite クラスの変数に書き換えよ。



[メインタブ]

```
Sprite h, m;

void setup() {
  size(320, 320);
  imageMode(CENTER);
  h = new Hammer();
  m = new Mole();
  m.move();
}

void draw() {
  background(192);
  m.display();
  h.move();
  h.display();
}
```

[Hammer タブ]

（省略）

[Mole タブ]

（省略）

[Sprite タブ]

（省略）

16.配列

16.1 説明

配列の生成

型名[] 配列変数名= new 型名[要素数]

配列の参照

配列変数名[添字]

配列の初期化

配列変数名[] = {値, 値, 値, … }

16.2 整数型で 10 個の要素を持つ配列 a を用意し、0～9 までの値を代入し、コンソールに表示する。


```
int[] a = new int[10];
for (int i = 0; i < 10; i++) {
  a[i] = i;
}

for (int i = 0; i < 10; i++) {
  println(a[i]);
}
```

16.3 整数型で 10 個の要素を持つ配列 a を 0～9 までの値で初期化し、コンソールに表示する。


```
int[] a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

for (int i = 0; i < 10; i++) {
```

```
  println(a[i]);
}
```

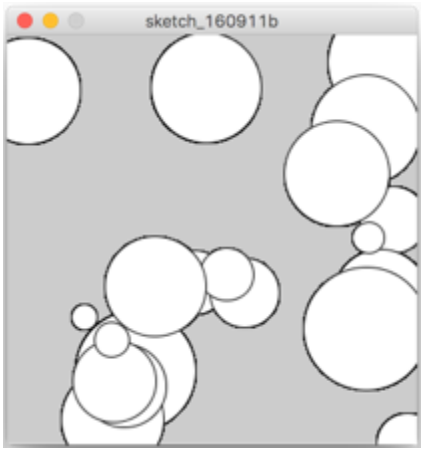
16.4 浮動小数点数型で 10 個の要素を持つ配列 a を用意し、100 未満の乱数を代入し、コンソールに表示する。


```
float[] a = new float[10];
for (int i = 0; i < 10; i++) {
  a[i] = random(100);
}

for (int i = 0; i < 10; i++) {
  println(a[i]);
}
```

16.5 円を表すクラス Ball を作る。Ball のフィールドは座標 x、y、直径 d。コンストラクタでは、ランダムな座標を x、y に設定し、直径は 100 未満のランダムな値を設定する。また、表示するメソッド display() を用意する。

```
class Ball {
  float x, y, d;

  Ball() {
    x = random(width);
    y = random(height);
    d = random(100);
  }

  void display() {
    ellipse(x, y, d, d);
  }
}
```

16.6 クラス Ball のインスタンスを 20 個生成し、表示するメイン・プログラムを作る。



[メインタブ]

```
Ball[] b = new Ball[20];

void setup() {
  size(320, 320);
  for (int i = 0; i < 20; i++) {
    b[i] = new Ball();
  }
}

void draw() {
  for (int i = 0; i < 20; i++) {
    b[i].display();
  }
}
```

[Ball タブ]

```
class Ball {
  float x, y, d;

  Ball() {
```

```
    x = random(width);
    y = random(height);
    d = random(100);
  }

  void display() {
    ellipse(x, y, d, d);
  }
}
```