



プログラミング演習(10)

関数

中村, 高橋
小林, 橋本



- Processing で関数に挑戦！
 - 機能をどんどん作ってみよう！
 - 円とか四角形だけじゃなくて、色々な図形描画を関数にしてしまおう！
 - 判定も関数で！
 - 関数を使わないとプログラムがどんどん汚くなっていく！
 - ここはしっかり動く！という部分を分離して関数化していこう！
 - drawやsetupの中身は極力短くする！



- $y=f(x)$ みたいなものを数学でよく見る
 - $f(x) = x^2 + 2x + 1$
 - $g(x) = x^3 + 3x^2 + 2x + 5$
 - これは, x に何らかの値を代入すると, その結果が返ってくるというもの
- x に何らかの値を代入すると...
 - $f(3) = 3^2 + 2 \cdot 3 + 1 = 16$
 - $f(1) = 1^2 + 2 \cdot 1 + 1 = 4$
 - $f(0) = 0^2 + 2 \cdot 0 + 1 = 1$
 - $g(2) = 2^3 + 3 \cdot 2^2 + 2 \cdot 2 + 5 = 29$



- $z = h(x, y)$
- $h(x, y) = x^2 + 2x + y + xy + 1$
 - x と y に値を代入すると...
 - $h(3, 1) = 3^2 + 2 \cdot 3 + 1 + 3 \cdot 1 + 1 = 16$
 - $h(1, 2) = 1^2 + 2 \cdot 2 + 2 + 1 \cdot 2 + 1 = 8$
 - $h(0, 0) = 0^2 + 2 \cdot 0 + 0 + 0 \cdot 0 + 1 = 1$



- プログラムにおける関数も同様
 - 何らかの値を代入すると、その結果を返してくれるというものでしかない

```
float f(float x)
{
    return x * x + 2 * x + 1;
}

float g(float x)
{
    return x * x * x + 3 * x * x + 2 * x + 5;
}

float h(float x, float y)
{
    return x * x + 2 * x + y + x * y + 1;
}
```

関数の呼び出し



- プログラムにおける関数も同様
 - 何らかの値を代入すると，その結果を返してくれるというものでしかない

```
println(f(3));  
println(f(1));  
println(f(0));  
println(g(2));  
println(h(3, 1));  
println(h(1, 2));  
println(h(0, 0));
```

関数の定義



関数の戻り値の型 関数名 (引数リスト)

{

関数内のいろいろな処理

(戻り値がある場合は) `return` 戻り値;

}

戻り値がない場合は `void`

`x, y` に半径 `r` の円を描く関数

```
void circle(int x, int y, int r)
{
    ellipse(x, y, r * 2, r * 2);
}
```

半径 `r` の円の面積を求める関数

```
float menseki(float r)
{
    return r * r * 3.14;
}
```

今まで使ってきた関数



- 背景を塗りつぶす: `background(色);`
- 円を描く: `ellipse(x座標, y座標, 縦直径, 横直径);`
- 線を描く: `line(x1, y1, x2, y2);`
- 四角形を描く: `rect(x座標, y座標, 横幅, 縦幅);`
- 距離を計算する: `dist(x1, y1, x2, y2);`

などなど, これまでに色々な機能をもつ関数を利用してきた.

こういった関数を作ることが今回の目的

関数について



- 最初に登場した「`void setup(){ ... }`」や
、 「`void draw(){ ... }`」は、`setup`や`draw`
には返り値が無いという事を意味している
– ただ準備、ただ描画をするだけなので！



- return があるとなそこで関数を抜け出す！

```
int hogeHogeFunc(int nya, int piyo)
{
    // 色々な処理

    if( nya == ?? )
    {
        // 色々な処理
        return 2;
    }

    // 色々な処理
    return 3;
}
```

この if 文の中に入ったら
return 2 して終わり！

↑の if 文の判定で含まれ
なかったらここに来る！



- 関数の引数リスト（どういう値を受け取るか）に入力とする変数を用意し，その変数のみを利用して結果を返すのがポイント
 - 描画であれば入力された x ， y からの相対座標
 - 計算であれば入力された変数同士の計算
- 関数の利用用途
 - 色々な描画機能
 - 色々な計算機能
 - 色々な判定機能

自分専用の便利関数をたくさん作ろう！

約数の数を求める関数



(Q) ある入力された数字の約数の数を求める関数をどう作るか？ また、その関数を使って数字と約数の数のペアを出力しよう

```
17989: 2  
17990: 16  
17991: 6  
17992: 16  
17993: 4  
17994: 8  
17995: 8  
17996: 12  
17997: 8  
17998: 4  
17999: 4  
18000: 60
```

約数の数を求める関数



- 考え方
 - 引数は整数型の `num` にする
 - 約数を数える整数型の変数 `count` を用意
 - 整数型の変数 `i` (1から`num`まで1ずつ増やす) を用意し, `num` が `i` で割り切れたら `count` を1追加する
 - 最後に `count` の値を返す
 - `return count;`
 - `println` で数と, 返って来た値を表示する

```
int getNumberOfDivisor(int num)
```

```
{  
    int i = 1;  
    int count = 0;  
    while(i <= num)  
    {  
        if(num%i == 0)  
        {  
            count++;  
        }  
        i++;  
    }  
    return count;  
}
```

戻り値 (この値が
呼び出し元に返る)

```
void setup()
```

```
{  
    for(int i = 1; i < 100000; i++)  
    {  
        println(i + ": " + getNumberOfDivisor(i));  
    }  
}
```

ここに戻る



素数かどうかを判定



(Q) 関数の引数として入力した素数かどうかを判定する関数を作ろう。素数ならtrueを、素数じゃなければfalseを返すようにしよう

- 引数は整数型の num にする
- 1とその数以外で約数があったら素数！なので、falseと返す。それ以外の場合はtrueと返す
- 関数の戻り値がtrueだったら素数！と表示する

素数かどうかを判定



```
boolean isPrimeNumber(int num)
{
    int i = 2;
    while(i <= num-1)
    {
        if(num % i == 0)
        {
            // 割り切れたらfalse
            return false;
        }
        i++;
    }
    // 何もなかったらtrue
    return true;
}
```

```
void setup()
{
    for(int i = 1; i < 100000; i++)
    {
        if(isPrimeNumber(i) == true)
        {
            println(i+"は素数です!");
        }
    }
}
```

ちなみに、約数の数を数える
プログラムをそのまま使うだけでもOK

階乗の計算を行う関数



(Q)入力した値の階乗の計算を行う関数を作り、その計算結果を返すようにせよ。また、1から10までについて階乗の結果を標準出力せよ

- ある自然数 n の階乗 ($n!$) は、1から n までの数字を掛けあわせた値
- 引数は整数型の `num` にする
- 繰り返しを使って計算する！

階乗を計算



```
int factorial(int num)
{
    int i = 1;
    int total = 1;
    while(i <= num)
    {
        total = total * i;
        i++;
    }
    return total;
}
```

```
void setup()
{
    int i = 1;
    while(i <= 10)
    {
        println(i + "の階乗は" + factorial(i) + "です!");
        i++;
    }
}
```

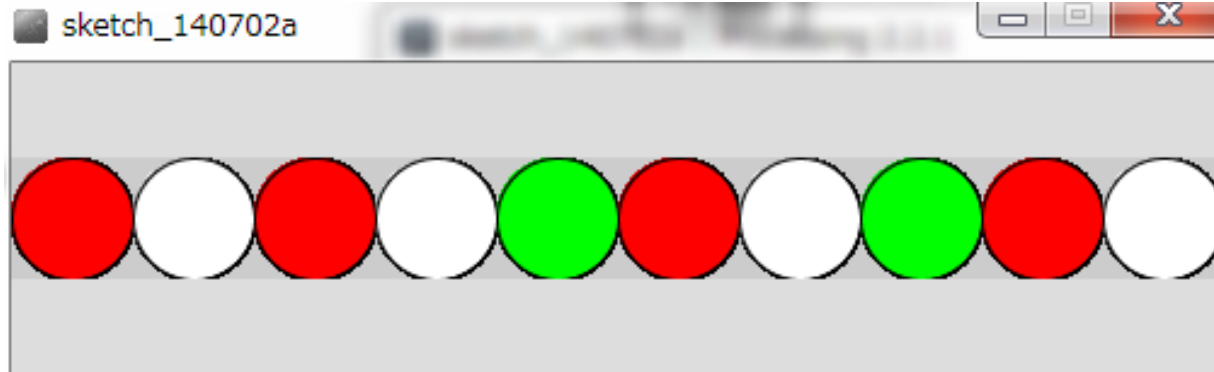
予習問題



- 入力された数字が完全数かどうかを判定する関数を作ってみましょう
- 四角形と円の当たり判定を行う関数を作ってみましょう
- 四角形同士の当たり判定を行う関数を作ってみましょう



- 直径50の円を横に10個並べたボードを作成せよ。円の内部をクリックする度に、その円の色が【白→赤→緑→黄→青→白】と変化させるようにすること。関数を作って次頁のプログラムを短くしよう！



```
int[] status;
void setup()
{
  size(500, 50);
  status = new int[10];
  for(int i = 0; i < 10; i++)
  {
    status[i] = 0;
  }
}
void draw()
{
  for(int i = 0; i < 10; i++)
  {
    if(status[i] == 0)
    {
      fill(255, 255, 255);
    }
    else if(status[i] == 1)
    {
      fill(255, 0, 0);
    }
    else if(status[i] == 2)
    {
      fill(0, 255, 0);
    }
    else if(status[i] == 3)
    {
      fill(0, 0, 255);
    }
    else if(status[i] == 4)
    {
      fill(255, 255, 0);
    }
    ellipse(i * 50 + 25, 25, 50, 50);
  }
}
```

```
void mousePressed()
{
  for(int i = 0; i < 10; i++)
  {
    if(dist(mouseX, mouseY, i*50+25, 25) <= 25)
    {
      status[i] ++;
      if(status[i] > 4)
      {
        status[i] = 0;
      }
    }
  }
}
```

ここが長すぎる！

関数を作る



- `status`の配列の値に応じて色を切り替えた円を描画する関数を作って利用する！
- 必要な情報
 - `status` の値
 - 表示する座標



関数で短くす

全体としては短くなって無くてでも部分的に短くすることが重要！

```
int[] status;

void setup()
{
  size(500, 50);
  status = new int[10];
  for(int i = 0; i < 10; i++)
  {
    status[i] = 0;
  }
}

void draw()
{
  for(int i = 0; i < 10; i++)
  {
    myEllipse(i * 50 + 25, status[i]);
  }
}
```

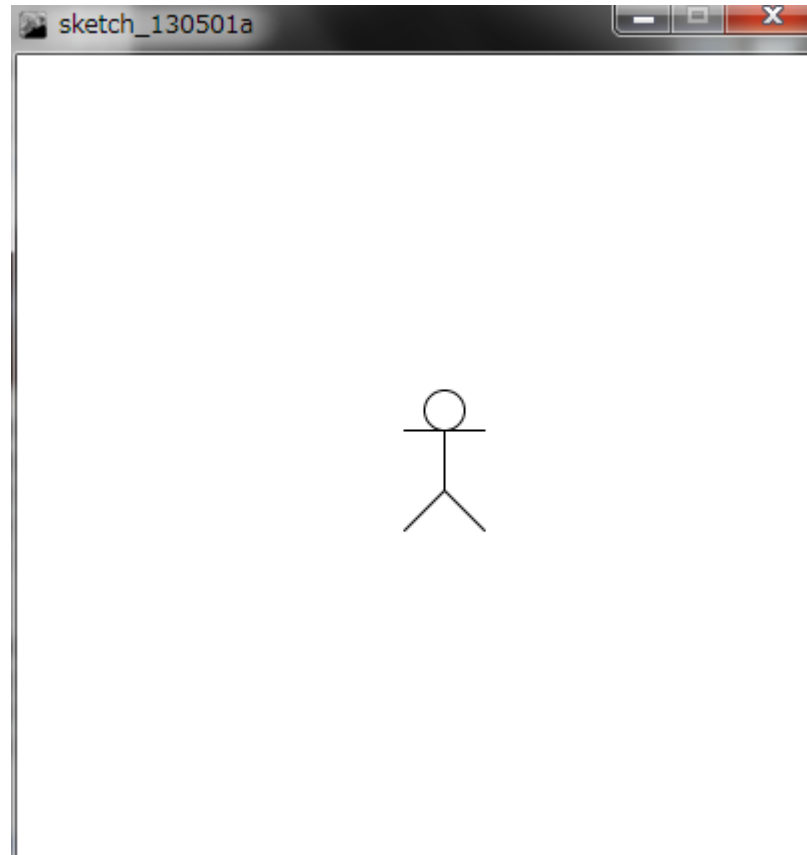
短くなった！

```
void myEllipse(int posX, int showStatus)
{
  if(showStatus == 0)
  {
    fill(255, 255, 255);
  }
  else if(showStatus == 1)
  {
    fill(255, 0, 0);
  }
  else if(showStatus == 2)
  {
    fill(0, 255, 0);
  }
  else if(showStatus == 3)
  {
    fill(255, 255, 0);
  }
  else if(showStatus == 4)
  {
    fill(0, 0, 255);
  }
  ellipse(posX, 25, 50, 50);
}
```

棒人間を描く



(Q) x, y 座標を指定すると棒人間を描いてくれる関数を作成せよ！

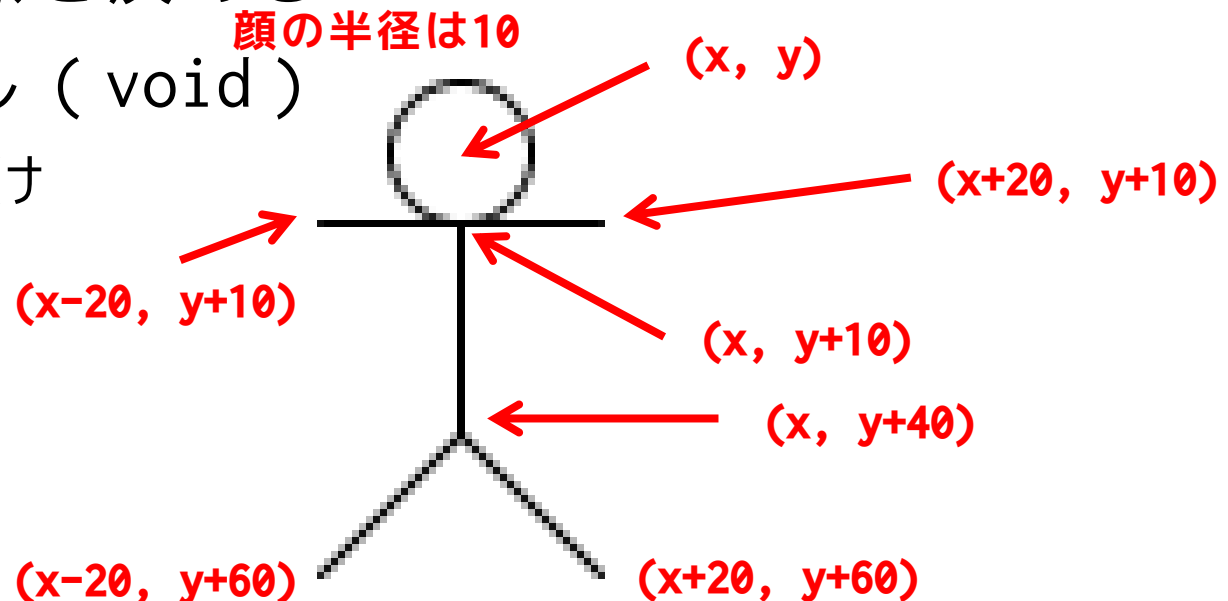




• 考え方

- 棒人間は，顔の中心の座標 (x, y) を与えると，勝手に体と手と足を描くものにする
- 棒人間の中心の座標を (x, y) としたときのそれぞれの座標を決める
- 返り値はなし (void)

- 描画するだけ





- マウスカーソルの場所に棒人間を描く

```
void setup(){
  size(400, 400);
}
void drawHuman(int x, int y)
{
  ellipse(x, y, 20, 20);
  line(x, y+10, x, y+40);
  line(x-20, y+10, x+20, y+10);
  line(x, y+40, x-20, y+60);
  line(x, y+40, x+20, y+60);
}
void draw()
{
  background(255);
  drawHuman(mouseX, mouseY);
}
```

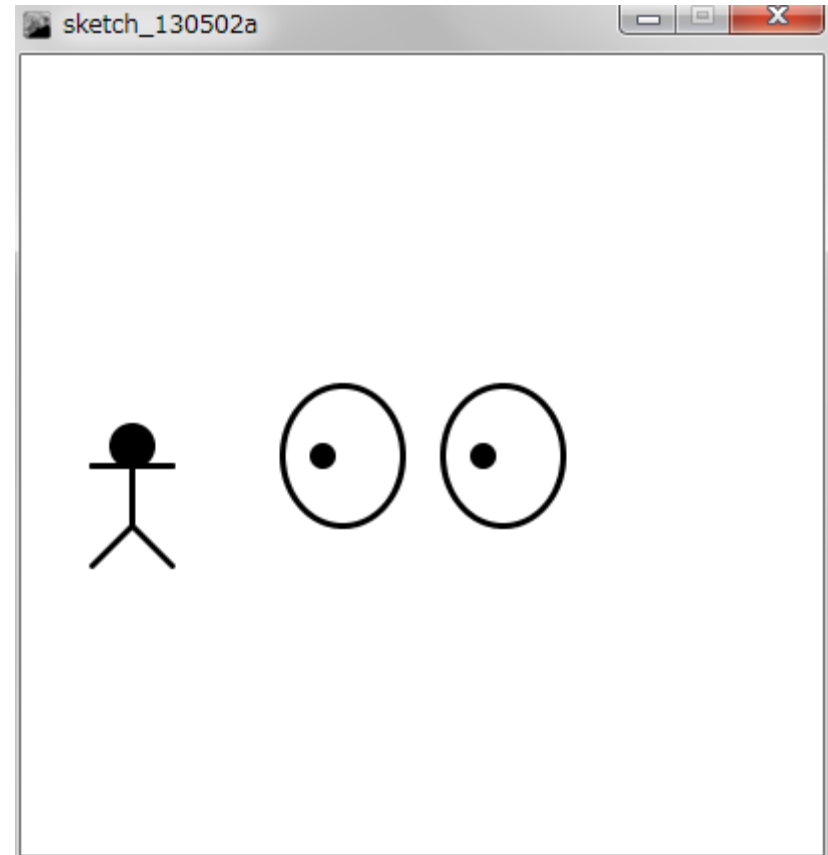
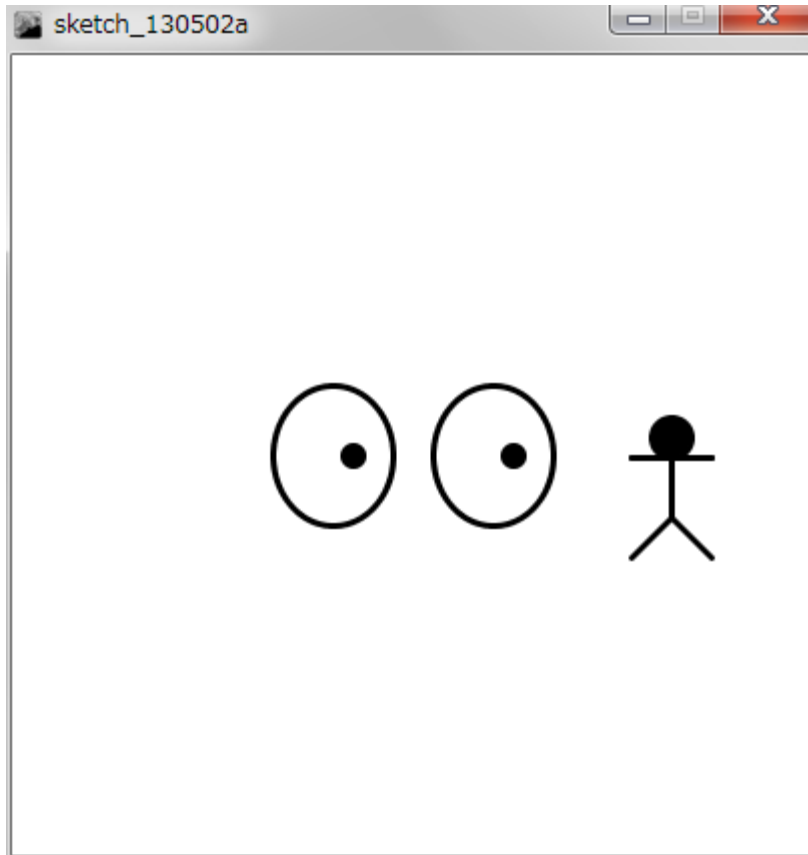


- マウスクリックされた場所に棒人間を描くプログラムを書いてみましょう
 - mousePressed で drawHuman
- 配列と繰り返しを使ってマウスカーソルに追尾する棒人間を100人描いてみましょう
 - 前回の資料を参考に！
- マウスカーソルに追尾する棒人間100人を、
どんどん薄くしましょう（同上）
- 棒人間じゃない何かを描画する関数を作り
ましょう（ロボットでもOK！）

関数でアニメーション



(Q) マウスの位置に応じて黒目を動かすプログラムを作ってみよう！ (目玉で1つの関数)





• 考え方

- 目玉の場所 (x, y) と、黒目が向く方向を決めるための座標 (mx, my) を引数にする
- mx, my はマウス座標を入力
- 白目を (x, y) を中心とし適当な楕円で描く
- x と mx の位置関係で黒目の座標を決め黒目を描く
 - $mx < x - 10$ なら黒目を左へ (10は適当な値)
 - $mx > x + 10$ なら黒目を右へ
 - そうでなければ黒目を真ん中へ
- drawEye を2つdraw()の中に書く！
- drawHuman は前に作ったのを使う

関数でアニメ



引数は4つ

描画だけなので
返り値はなし

```
void setup()
{
  size(400, 400);
}

void drawEye(int x, int y, int mx, int my)
{
  strokeWeight(3);
  fill(255, 255, 255);
  ellipse(x, y, 60, 70);
  fill(0, 0, 0);
  if(x > mx + 10)
  {
    ellipse(x - 10, y, 10, 10);
  }
  else if(x < mx - 10)
  {
    ellipse(x + 10, y, 10, 10);
  }
  else
  {
    ellipse(x, y, 10, 10);
  }
}

void draw()
{
  background(255);
  drawHuman(mouseX, mouseY);
  drawEye(160, 200, mouseX, mouseY);
  drawEye(240, 200, mouseX, mouseY);
}
```

x と mx の位置で
条件分岐

予習問題

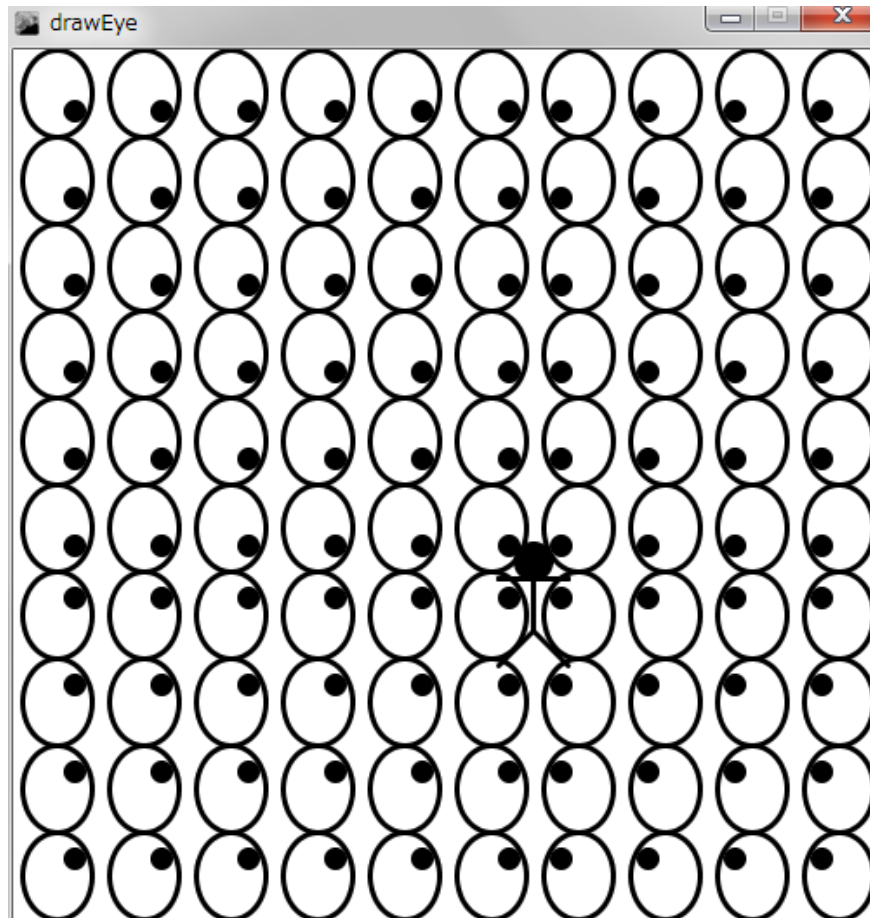


- マウスのX座標だけを考慮して目を動かしているが，マウスが下にある時，上にある時で目を上下に動かしてみましよう（8方向）
 - －（ヒント） y と my の関係を利用
- 棒人間を左から右に移動し，その移動に応じて黒目の動きを変えましよう
 - －（ヒント） 棒人間の座標を引数にする！
- （挑戦）もう少し目の動きをスムーズにしてみましよう！

予習問題



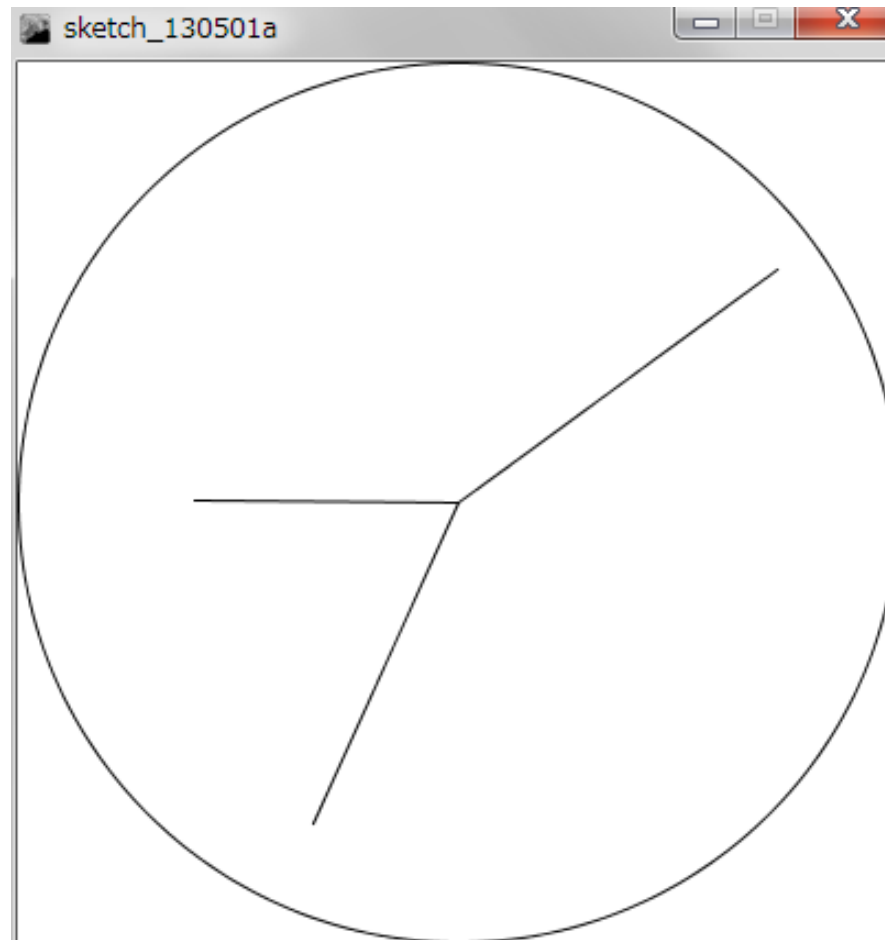
- 繰り返しを利用して妖怪百眼を作ろう
- x , y の2つのループを用意する！



アナログ時計を描く



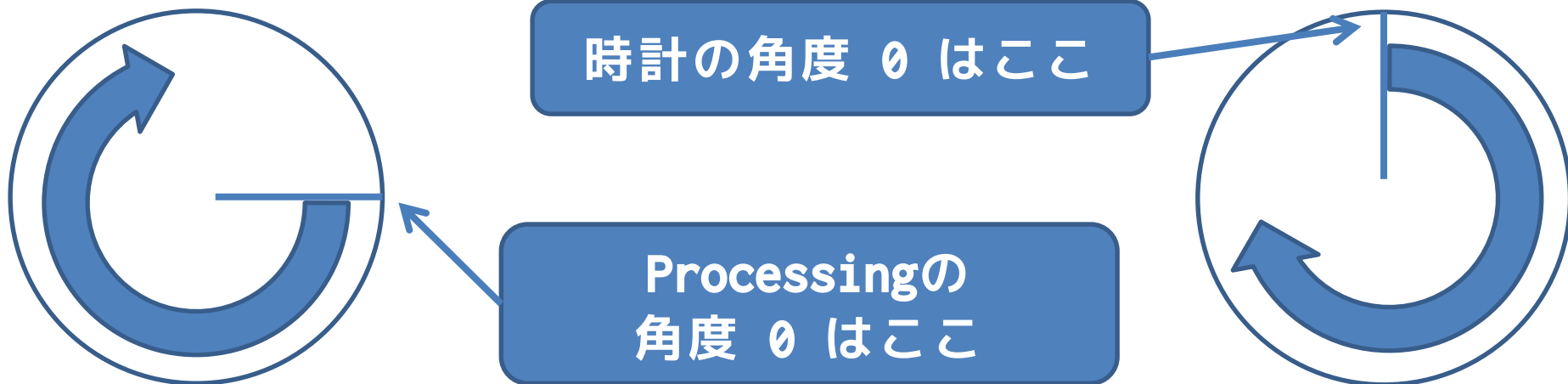
(Q) 400x400の画面上に現在の時間に合わせてアナログ時計を表示するプログラムを作れ



アナログ時計を描く



- 考え方
 - 現在の時間は `hour()` で、現在の分は `minute()` で、現在の秒は `second()` で取得可能
 - 長針, 短針, 秒針は, 角度 a , 半径を r としたとき, 画面の中央 (x, y) から $(x+r*\cos(a), y+r*\sin(a))$ までの線分を描くことで表現可能 (a は $0\sim 2\pi$ のラジアン)
 - 角度 a をどうやって求めるかが問題



アナログ時計を描く



- 考え方（続き）

- 時計の長針は12時間で1周（ 2π ），短針は60分で1周（ 2π ），秒針は60秒で1周

- hour 時は，角度として $2 * \text{PI} * \text{hour} / 12$

- minute 分は，角度として $2 * \text{PI} * \text{minute} / 60$

- second 秒は，角度として $2 * \text{PI} * \text{second} / 60$

- $\pi/2$ だけProcessingの角度は進んでいるので， $\pi/2$ だけ角度をマイナスする

- $\text{kaku_h} = 2 * \text{PI} * \text{hour} / 12 - \text{PI} / 2;$

- $\text{kaku_m} = 2 * \text{PI} * \text{minute} / 60 - \text{PI} / 2;$

- 長針，短針，秒針は適当に半径を決定

アナログ時計を描く [1]



```
void drawClock(int x, int y, int r)
{
    ellipse(x, y, r * 2, r * 2);
    float kaku_h = 2 * PI * hour() / 12.0 - PI / 2;
    float kaku_m = 2 * PI * minute() / 60.0 - PI / 2;
    float kaku_s = 2 * PI * second() / 60.0 - PI / 2;
    float len_h = r * 0.5;
    float len_m = r * 0.7;
    float len_s = r * 0.9;
    line(x, y, x + len_h * cos(kaku_h), y + len_h * sin(kaku_h));
    line(x, y, x + len_m * cos(kaku_m), y + len_m * sin(kaku_m));
    line(x, y, x + len_s * cos(kaku_s), y + len_s * sin(kaku_s));
}

void draw()
{
    background(255);
    drawClock(200, 200, 190);
}
```

アナログ時計を描く [2]



```
void drawClock(int x, int y, int r, int hour, int min, int sec)
{
    ellipse(x, y, r * 2, r * 2);
    float kaku_h = 2 * PI * hour / 12.0 - PI / 2;
    float kaku_m = 2 * PI * min / 60.0 - PI / 2;
    float kaku_s = 2 * PI * sec / 60.0 - PI / 2;
    float len_h = r * 0.5;
    float len_m = r * 0.7;
    float len_s = r * 0.9;
    line(x, y, x + len_h * cos(kaku_h), y + len_h * sin(kaku_h));
    line(x, y, x + len_m * cos(kaku_m), y + len_m * sin(kaku_m));
    line(x, y, x + len_s * cos(kaku_s), y + len_s * sin(kaku_s));
}

void draw()
{
    background(255);
    drawClock(200, 200, 190, hour(), minute(), second());
}
```

もう少し発展



- 現在のアナログ時計は短針が1時間に1回，分針が1分に1回しか動かないが，本当はもっと細かく動くはず。それをプログラム！

アナログ時計を描く [1]



```
void drawClock(int x, int y, int r)
{
    ellipse(x, y, r * 2, r * 2);
    float kaku_h = 2 * PI * (hour() + minute()/60.0) / 12.0 - PI / 2;
    float kaku_m = 2 * PI * minute() / 60.0 - PI / 2;
    float kaku_s = 2 * PI * second() / 60.0 - PI / 2;
    float len_h = r * 0.5;
    float len_m = r * 0.7;
    float len_s = r * 0.9;
    line(x, y, x + len_h * cos(kaku_h), y + len_h * sin(kaku_h));
    line(x, y, x + len_m * cos(kaku_m), y + len_m * sin(kaku_m));
    line(x, y, x + len_s * cos(kaku_s), y + len_s * sin(kaku_s));
}

void draw()
{
    background(255);
    drawClock(200, 200, 190);
}
```

アナログ時計を描く [2]



```
void drawClock(int x, int y, int r, int hour, int min, int sec)
{
    ellipse(x, y, r * 2, r * 2);
    float kaku_h = 2 * PI * (hour + min / 60.0) / 12.0 - PI / 2;
    float kaku_m = 2 * PI * min / 60.0 - PI / 2;
    float kaku_s = 2 * PI * sec / 60.0 - PI / 2;
    float len_h = r * 0.5;
    float len_m = r * 0.7;
    float len_s = r * 0.9;
    line(x, y, x + len_h * cos(kaku_h), y + len_h * sin(kaku_h));
    line(x, y, x + len_m * cos(kaku_m), y + len_m * sin(kaku_m));
    line(x, y, x + len_s * cos(kaku_s), y + len_s * sin(kaku_s));
}

void draw()
{
    background(255);
    drawClock(200, 200, 190, hour(), minute(), second());
}
```




- アナログ時計を改良し，もっとわかりやすく，豪華にしてみましよう
 - 色を変えたり，太さを変えたり
- 面白い時計を作ってみましよう
 - 自分専用の時計を作ろう！

予習問題



- マウスカーソルの場所に応じてアナログ時計を動かしてみましよう
- LondonとTokyoとNew Yorkの時計を表示するアナログ時計を横に並べてみましょう（時差考慮）
 - サマータイムで日本とロンドン間の時差が+8時間，日本とNYの間の時差が-13時間

