



プログラミング演習(5)

関数

中村、小林、橋本、辻野



- 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...
 - $h(3,1) = 3^2 + 2 \cdot 3 + 1 + 3 \cdot 1 + 1 = 20$
 - $h(1,2) = 1^2 + 2 \cdot 1 + 2 + 1 \cdot 2 + 1 = 8$
 - $h(0,0) = 0^2 + 2 \cdot 0 + 0 + 0 \cdot 0 + 1 = 1$



- 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));
```

今まで使ってきた関数



- 色んな関数を使ってきましたよね？
 - 距離を計算する: `dist(x1, y1, x2, y2)`
 - 正弦や余弦を求める: `sin(角度)`, `cos(角度)`
 - 背景を塗りつぶす: `background(赤, 緑, 青)`
 - 描画頻度の設定: `frameRate(1秒の描画回数)`
 - 現在の時間を取得する: `hour()`, `minute()`
 - 起動してからのミリ秒を取得する: `millis()`
 - 線を描画しない: `noStroke()`
 - 塗りつぶさない: `noFill()`
- などなど

関数(メソッド)は4種類



何か入力して
何か出力される



何か入力されるが
何も出力されない



何も入力してないけど
何か出力される



何も入力してないし
何も出力されない



今まで使ってきた関数



- 入力：あり 出力：あり
 - 距離を計算する： `dist(x1, y1, x2, y2)`
 - 正弦や余弦を求める： `sin(角度)`, `cos(角度)`



- 入力：あり 出力：なし
 - 背景を塗りつぶす： `background(赤, 緑, 青)`
 - 描画頻度の設定： `frameRate(1秒の描画回数)`



- 入力：なし 出力：あり
 - 現在の時間を取得： `hour()`, `minute()`
 - 起動してからのミリ秒を取得： `millis()`



- 入力：なし 出力：なし
 - 線を描画しない： `noStroke()`
 - 塗りつぶさない： `noFill()`

今まで使ってきた関数



- 引数：あり 返り値：あり
 - 距離を計算する： `dist(x1, y1, x2, y2)`
 - 正弦や余弦を求める： `sin(角度)`, `cos(角度)`



- 引数：あり 返り値：なし
 - 背景を塗りつぶす： `background(赤, 緑, 青)`
 - 描画頻度の設定： `frameRate(1秒の描画回数)`



- 引数：なし 返り値：あり
 - 現在の時間を取得： `hour()`, `minute()`
 - 起動してからのミリ秒を取得： `millis()`



- 引数：なし 返り値：なし
 - 線を描画しない： `noStroke()`
 - 塗りつぶさない： `noFill()`

関数の定義



関数の戻り値の型 関数名 (引数 (入力) 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 (出力) がない
場合は戻り値の型は `void`

例1: 関数の定義



- ある座標に二重の円を描く関数 `circleW`
 - ある点を中心として、直径と直径の半分の条件で、二重の円を描く
 - 出力はないので関数`circleW`は `void` で定義する

```
void circleW
```

- 引数は、`()`でくくって列挙し、中心座標を表す整数型の変数`(x, y)`と、直径を表す整数型の変数`d`で受け取るようにする

```
void circleW(int x, int y, int d)
```

- `int x / int y / int d` をカンマで繋いで書く

例1: 関数の定義



- ある座標に二重の円を描く関数 `circleW`
 - 中心座標を表す変数 (x, y) と、直径を表す変数 d を受け取って、それを内部で利用する

```
void circleW(int x, int y, int d)
{
    circle(x, y, d);
    circle(x, y, d / 2);
}
```

- `circle`関数を使って描画している

例2: 関数の定義



- ある半径で求まる円の面積を求める関数 `menseki`
 - 出力は実数値なので関数 `menseki` は `float` で定義する

```
float menseki
```

- 引数は、`()`でくくって列挙し、半径を表す実数型の変数 `r`で受け取るようにする

```
float menseki(float r)
```

- 返り値があるので、`r`を利用した計算結果を実数値で返すようにする

```
return rを利用した面積の値
```

例2: 関数の定義



- ある半径で求まる円の面積を求める関数 `menseki`
 - 引数で指定している変数 `r` を利用して半径を求めて変数 `area` に代入し、その値を返す

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

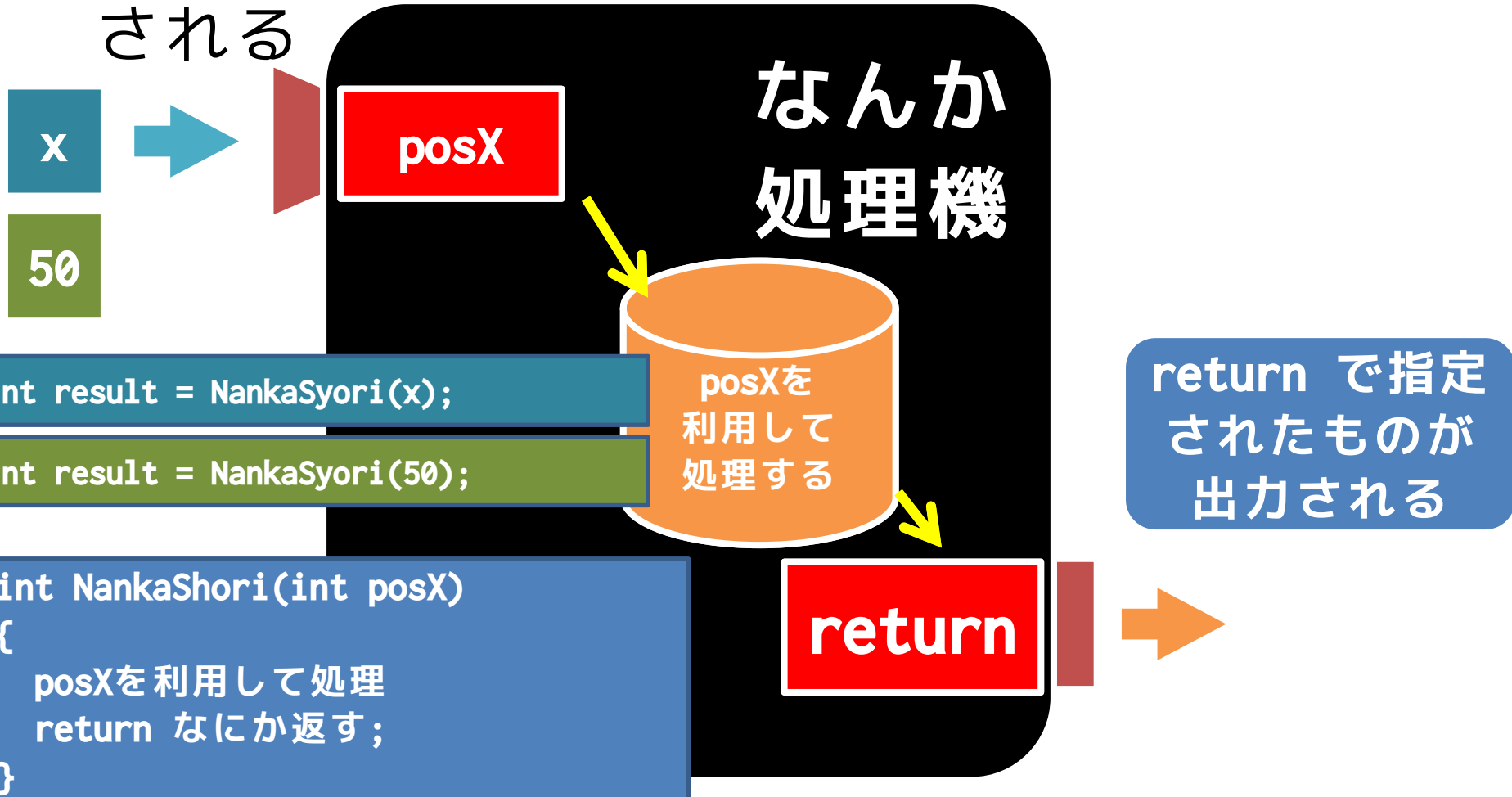
- 計算だけなら直接返してもOK

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

メソッドの内部処理



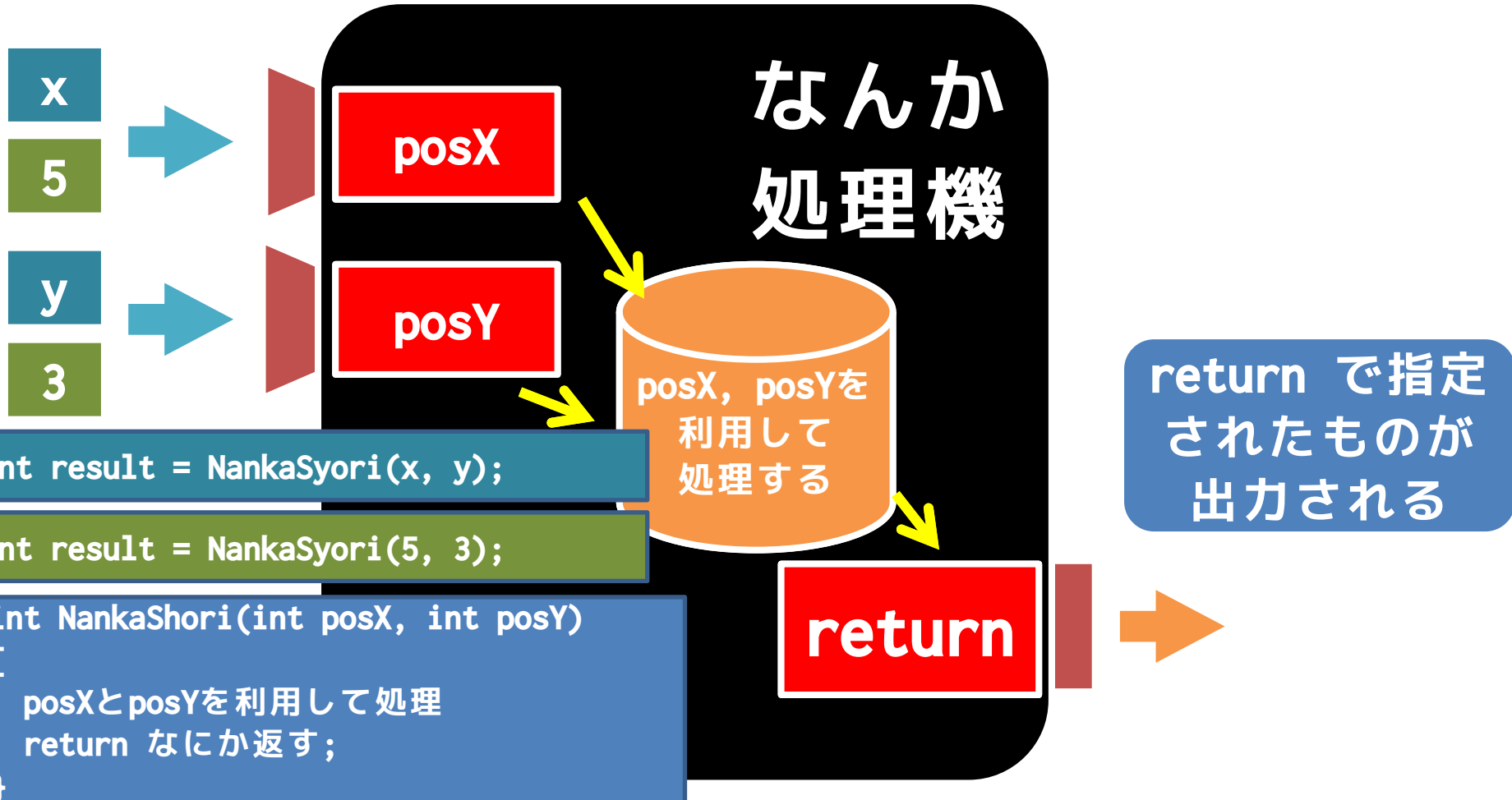
- 引数として取得した値は、引数で指定された変数名を利用して処理。returnで何か返される



メソッドの内部処理



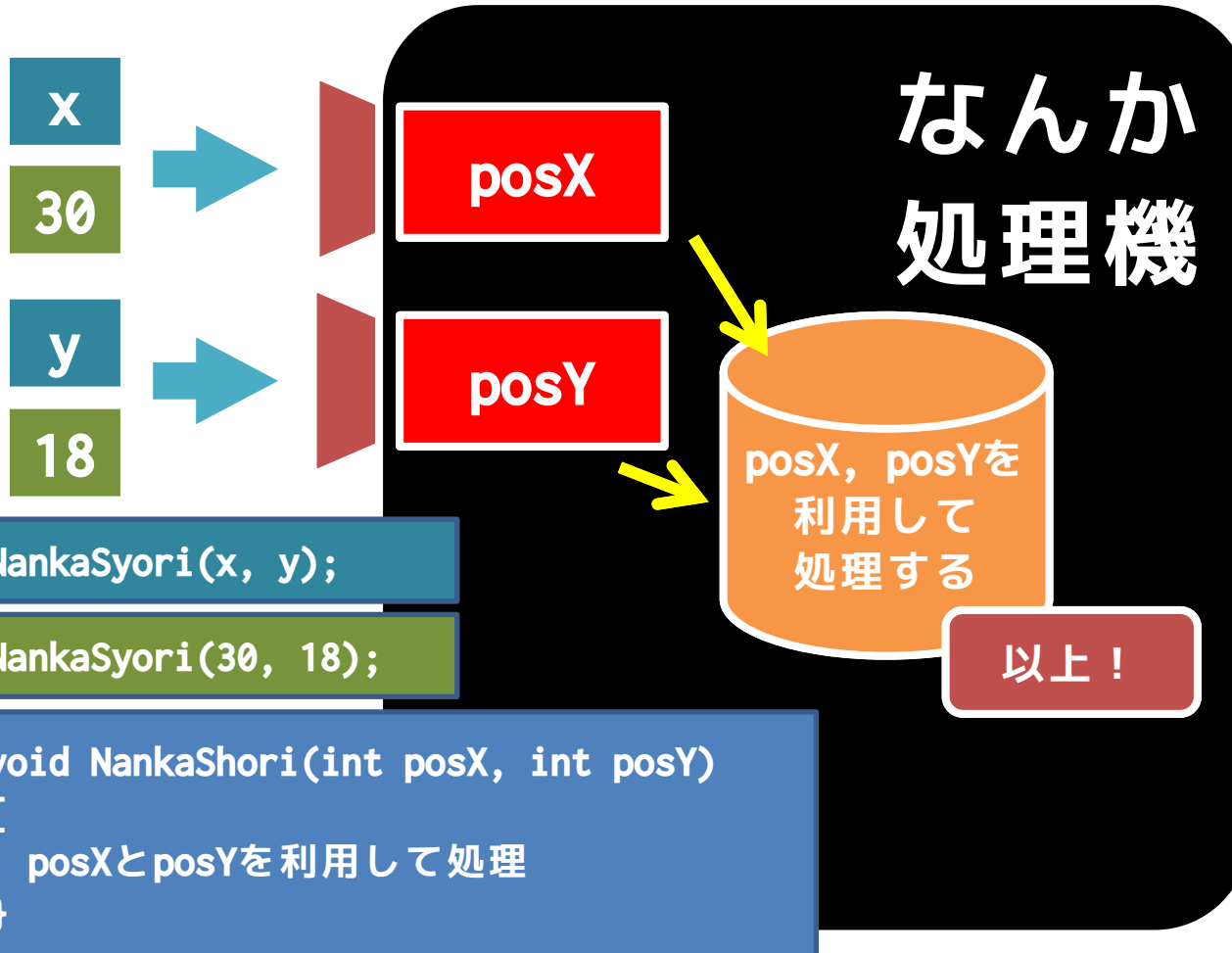
- 引数は増えるけれど返り値（returnされる値）は増えない



メソッドの内部処理



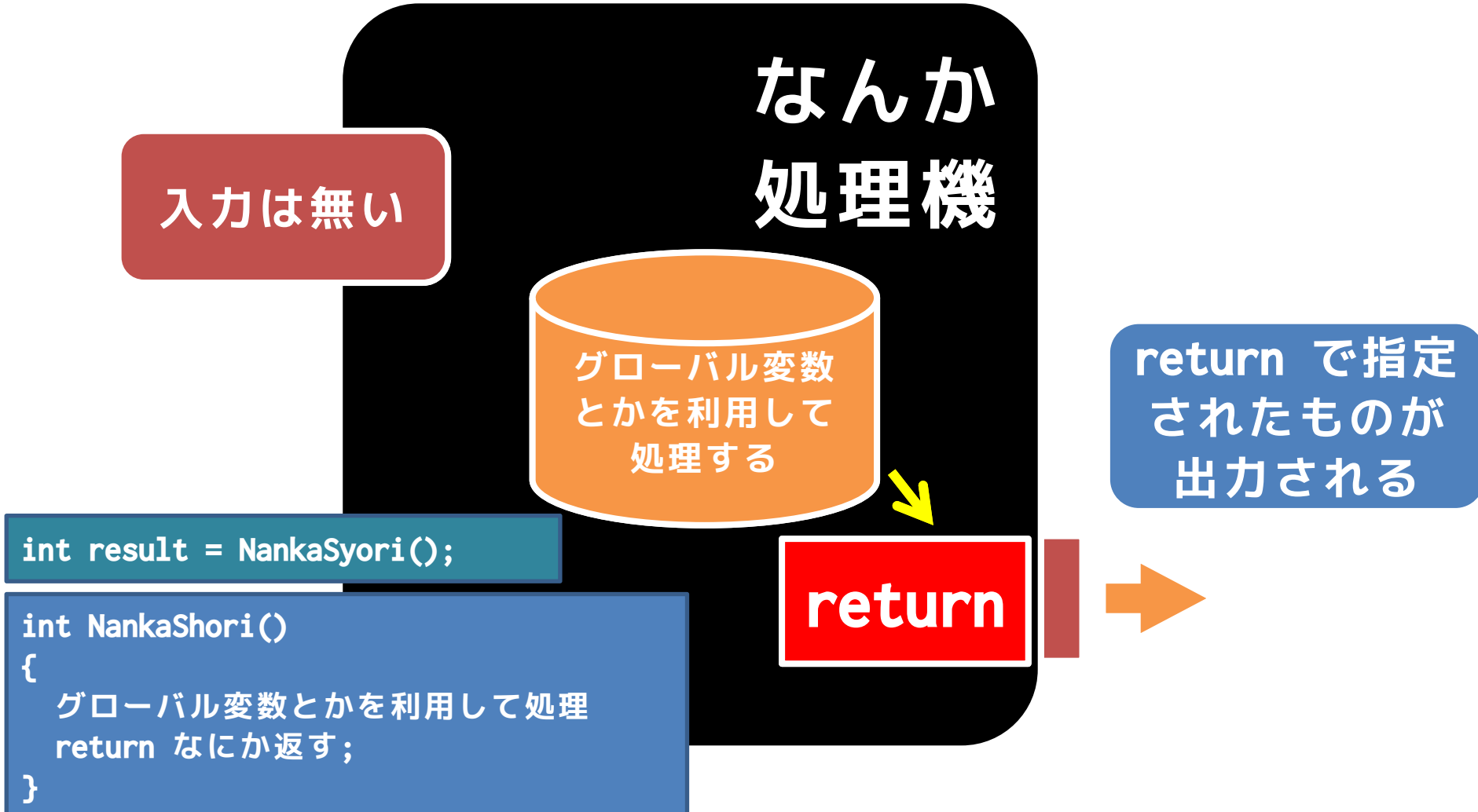
- 引数は増えるけれど返り値（returnされる値）は増えない



メソッドの内部処理



- 返り値だけのケース（引数がない場合）



メソッドの内部処理



- 入力も出力もないメソッド

入力は無い

なんか
処理機

グローバル変数とか
を利用して処理する

以上！

```
NankaSyori();
```

```
void NankaShori()  
{  
    なんかグローバル変数を利用して処理  
}
```



最初に登場した

- `void setup(){ ... }`
- `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！

関数のポイント



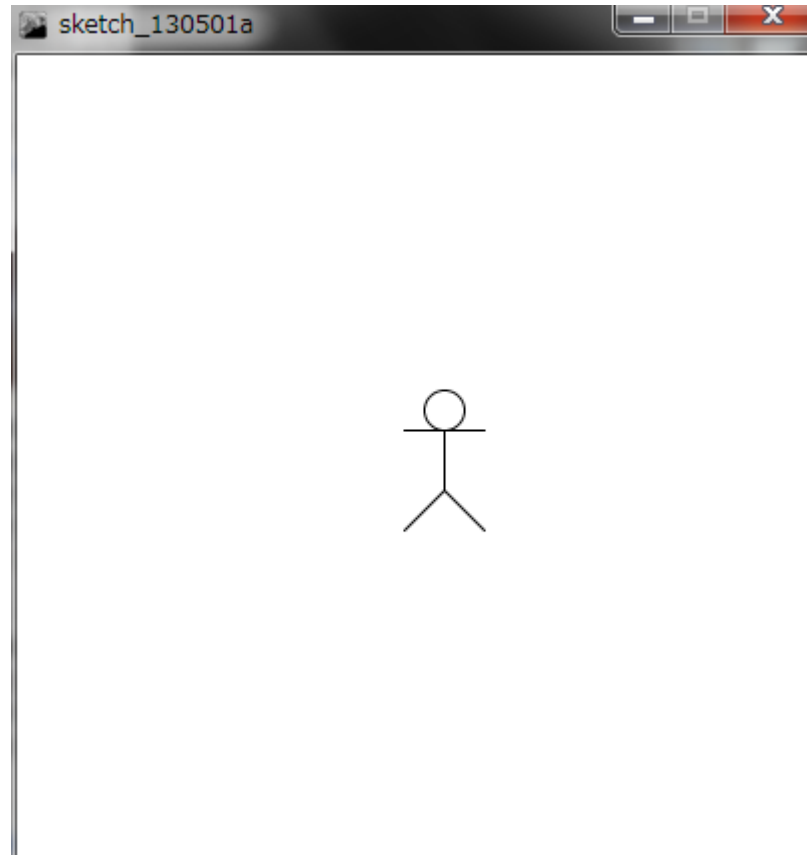
- 関数の引数リスト（どういう値を受け取るか）を受け取る変数を用意し、その変数のみを利用して結果を返すのがポイント
 - 引数で定義される変数はローカル変数
 - 描画であれば入力された x , y からの相対座標
 - 計算であれば入力された変数同士の計算
- 関数の利用用途
 - 色々な描画、計算、判定機能（判定は次回以降登場します）

自分専用の便利関数をたくさん作ろう！

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！

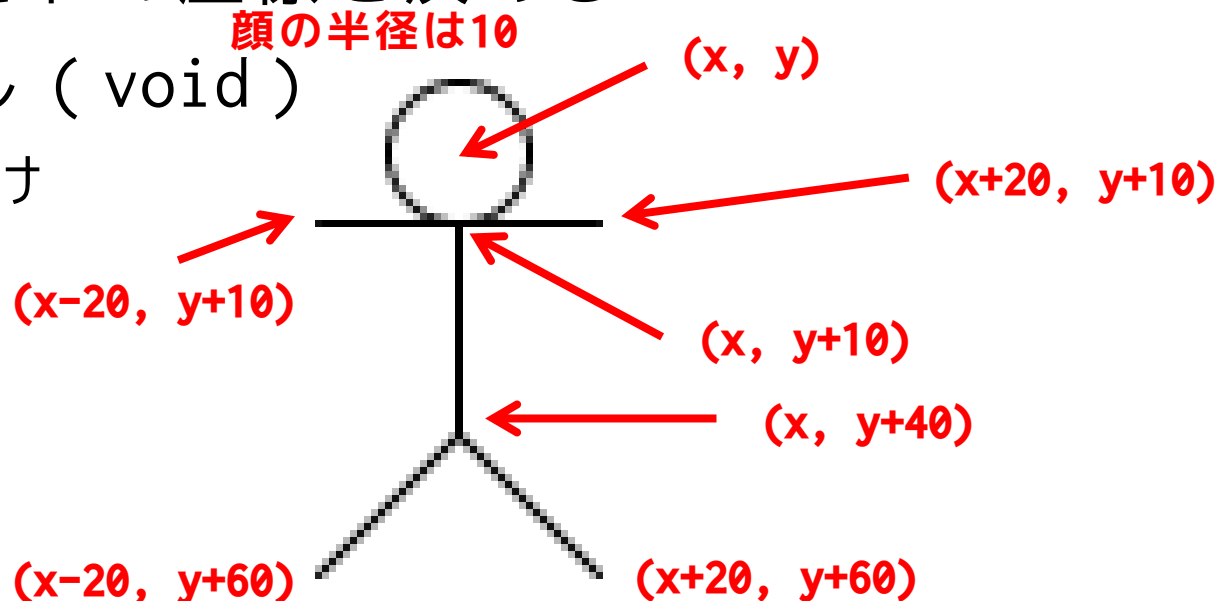




• 考え方

- 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整数型の (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){
  circle(x, y, 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`
- 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！）

予習問題

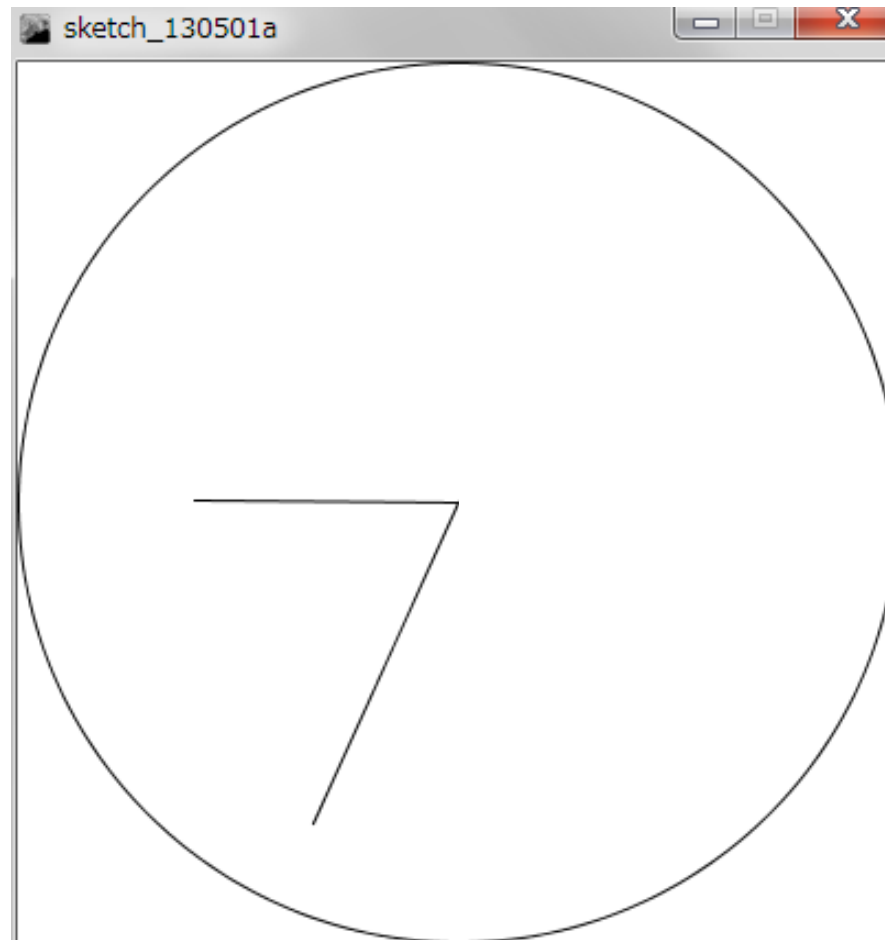


- マウスのX座標だけを考慮して目を動かしているが、マウスが下にある時、上にある時で目を上下に動かしてみましよう（8方向）
 - （ヒント） y と my の関係を利用
- 棒人間を左から右に移動し、その移動に応じて黒目の動きを変えましよう
 - （ヒント） 棒人間の座標を引数にする！
- （挑戦）もう少し目の動きをスムーズにしてみましよう！

アナログ時計を描く



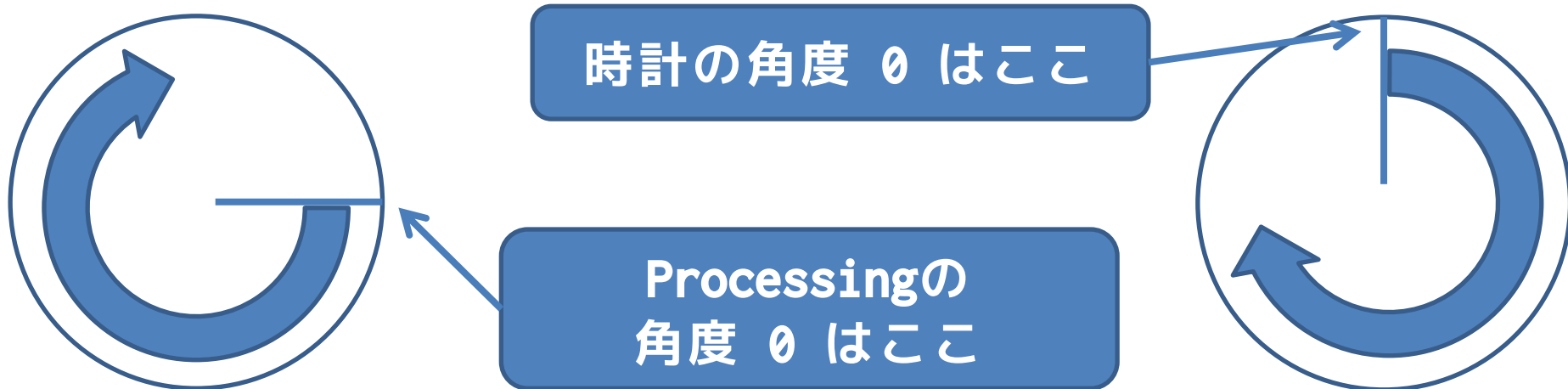
(Q) 400x400の画面上に現在の時間に合わせてアナログ時計を表示するプログラムを作れ



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()` で取得可能
 - 長針、短針は、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 ($2\pi = 2 * \text{PI}$)、短針は60分で1周 ($2\pi = 2 * \text{PI}$) 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 $2 * \text{PI} * \text{minute} / 60$

– $\pi/2$ 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

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

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

– 長針、短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)
{
    circle(x, y, r * 2);
    float a_h = 2 * PI * hour() / 12.0 - PI / 2;
    float a_m = 2 * PI * minute() / 60.0 - PI / 2;
    float l_h = r * 0.5;
    float l_m = r * 0.7;
    line(x, y, x + l_h * cos(a_h), y + l_h * sin(a_h));
    line(x, y, x + l_m * cos(a_m), y + l_m * sin(a_m));
}

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 h, int m)
{
    circle(x, y, r * 2);
    float a_h = 2 * PI * h / 12.0 - PI / 2;
    float a_m = 2 * PI * m / 60.0 - PI / 2;
    float l_h = r * 0.5;
    float l_m = r * 0.7;
    line(x, y, x + l_h * cos(a_h), y + l_h * sin(a_h));
    line(x, y, x + l_m * cos(a_m), y + l_m * sin(a_m));
}

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

もう少し発展



- 現在のアナログ時計は時針が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 2改



```
void drawClock(int x, int y, int r, int h, int m)
{
    circle(x, y, r * 2);
    float a_h = 2 * PI * (h + m / 60.0) / 12.0 - PI / 2;
    float a_m = 2 * PI * m / 60.0 - PI / 2;
    float l_h = r * 0.5;
    float l_m = r * 0.7;
    line(x, y, x + l_h * cos(a_h), y + l_h * sin(a_h));
    line(x, y, x + l_m * cos(a_m), y + l_m * sin(a_m));
}

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

アナログ時計を描く 2改二

明治大学総合数理学部
先端メディアサイエンス学科
中村研究室



```
float getAngleHourHandle(int h, int m)
{
    return 2 * PI * (h + m / 60.0) / 12.0 - PI / 2;
}

float getAngleMinuteHandle(int m)
{
    return 2 * PI * m / 60.0 - PI / 2;
}

void drawClock(int x, int y, int r, int h, int m)
{
    circle(x, y, r * 2);
    float angle_h = getAngleHourHandle(h, m);
    float angle_m = getAngleMinuteHandle(m);
    float l_h = r * 0.5;
    float l_m = r * 0.7;
    line(x, y, x + l_h * cos(angle_h), y + l_h * sin(angle_h));
    line(x, y, x + l_m * cos(angle_m), y + l_m * sin(angle_m));
}
```



- アナログ時計に秒針を追加してみよう！
 - 秒針を追加するとしたらどうする？
- アナログ時計を改良し、もっとわかりやすくしてみましよう
 - 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ょ
- 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

