
関数
基礎ユニット
PROCESSING FRIENDS

中間試験のお知らせ

試験日：7月2日(火)2限
試験範囲：6月28日（金）までの授業範囲
主な内容：図形・文字等の描画、変数、分岐処理、
繰り返し処理、座標変換，関数、1次元配列

形式：マークシート
持ち込み：プリント、ノート、ノートPC、本は可。人は不可。
注意事項：持ち込んだノートPCをネットに接続することは認めません。カンニング、相談などもダメです。

最終課題制作

Processingでプログラムをつくる
発表会を実施（最後の金曜日、7月26日）

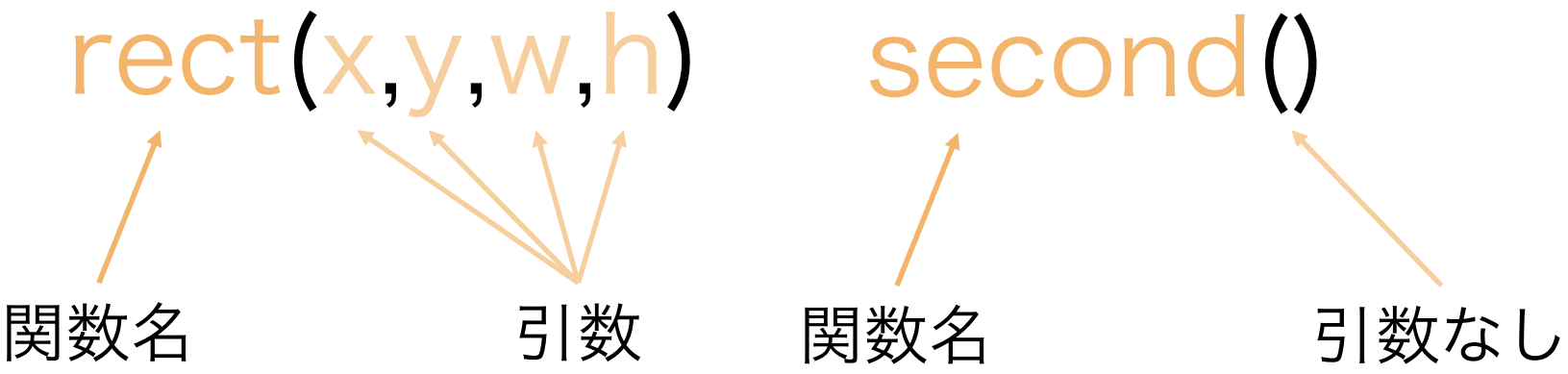
今日の内容

関数（続き）

前回の内容：関数の分類

	引数がない	引数がある
戻り値がある	randomなど	mapなど
戻り値がない void型	noFillなど	rectなど

手続き



関数の分類

戻り値がない

何らかの処理を
ひとまとめにしたもの

手続き、Procedure

戻り値がある

何らかの処理の結果、
何かの答え（戻り値）が
求まるもの

戻り値：リターンバリュー,return value

関数の宣言
（引数無し、戻り値なし）

```
void 関数名(){
    関数処理の内容を書きます。
    変数なども使うことができます。
}
```

setup関数やdraw関数は、このタイプの
関数の例となっている

関数の宣言
（引数無し、戻り値なし）

戻り値はvoid型の値

引数がないので空欄

関数名は変数名と同じように決めることができる

```
void drawPackMan(){
  pushMatrix();
  float angle=PI/6;
  arc(0, 0, 40, 40, angle, 2*PI-angle);
  rotate(angle);
  line(0, 0, 20, 0);
  rotate(-2*angle);
  line(0, 0, 20, 0);
  popMatrix();
}
```

まとめたい処理内容を書く

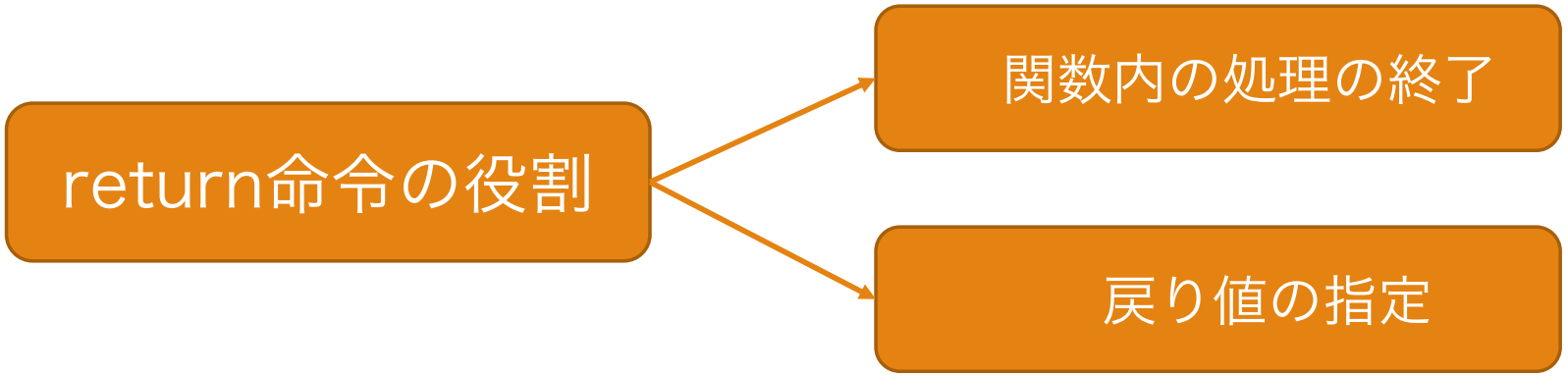
関数の宣言
（引数有り、戻り値なし）

```
void 関数名(データ型名1　引数名1,
            データ型名2　引数名2…){
    関数処理の内容を書きます。
    変数なども使うことができます。
}
```

引数名の有効範囲は関数の中だけ
rectなどのこのパターン

関数の宣言
（引数無し、戻り値あり）

戻り値のデータ型 関数名(){
　　関数処理の内容を書きます。
　　どこかに、return命令が必要です。
　　変数なども使うことができます。
}



関数の宣言
（引数無し、戻り値あり）

戻り値はString型の値　　引数がないので空欄

```
String today(){
  String msg = year() +”/”+month() + “/”+day();
  return msg;
}
```

文字列
処理の基礎

文字列+文字列–＞文字列の連結
“kait”+”DeNA” -> “kaitDeNA”
数字+文字列、文字列+数字->文字列の連結

関数の宣言
（引数あり、戻り値あり）

引数（ひきすう）

戻り値のデータ型 関数名(データ型名1　引数名1,
データ型名2　引数名2…){

関数処理の内容を書きます。
どこかに、return命令が必要です。
変数なども使うことができます。
引数は自由に使うことができる。

}

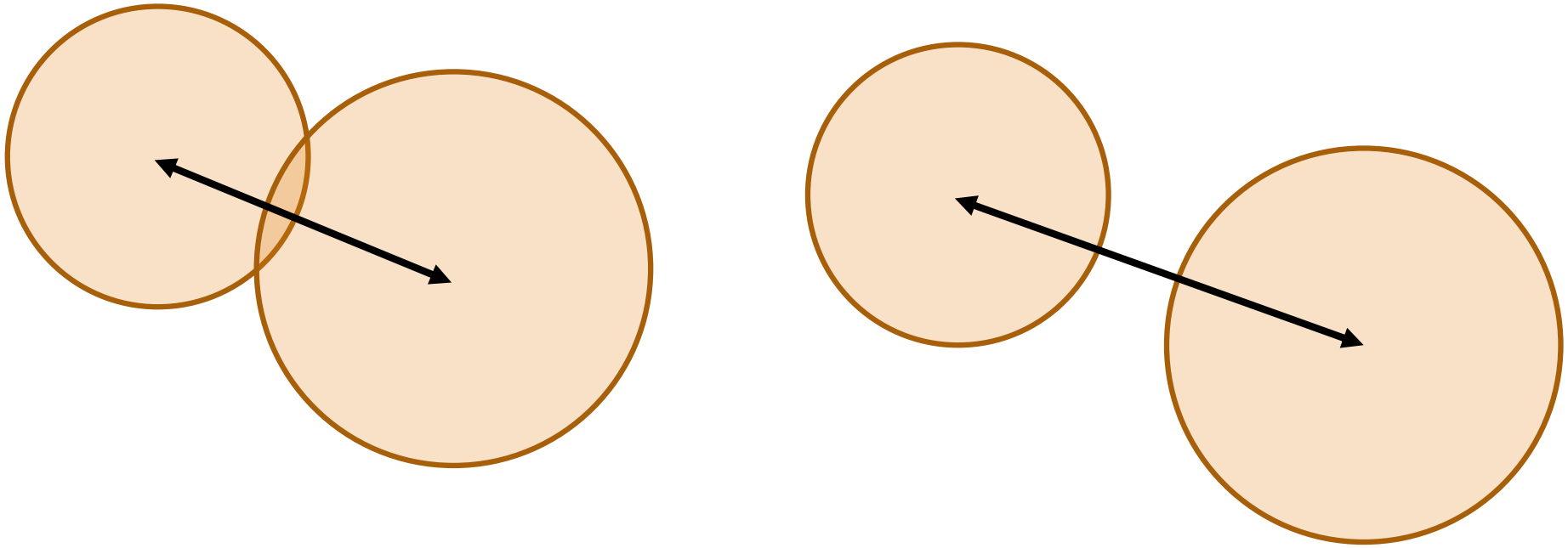
引数：関数の処理に利用するデータ（値）を与える

dist(x0,y0,x1,y1)
2点（x0,y0）と(x1,y1)の距離を求める組み込み関数
abs(x)
xの絶対値を求める組み込み関数

絶対値

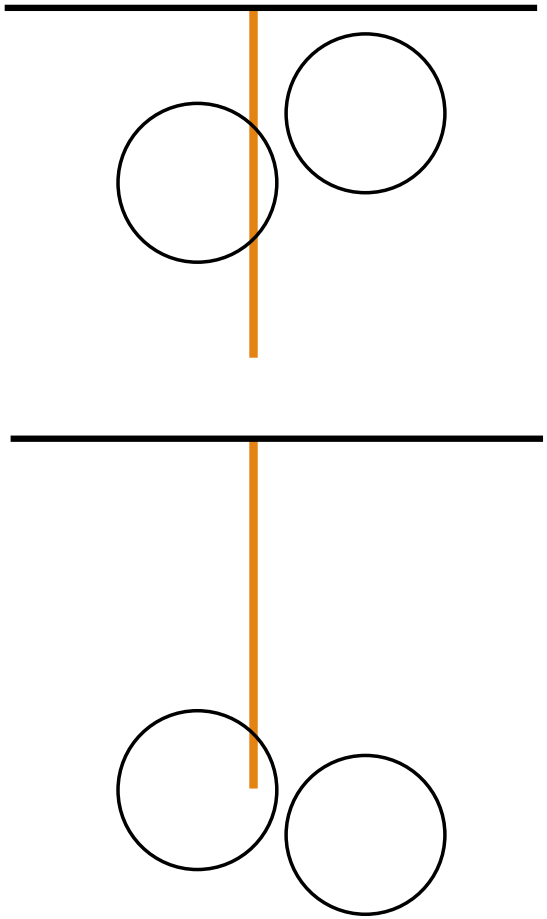
```
float abs(float x){
  if(x > 0){
    return x;
  }else{
    return –x;
  }
}
```

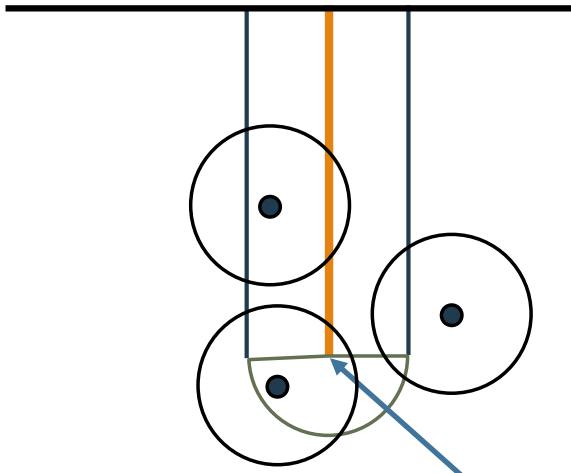
円と円の接触



2つの円の中心の間の距離がわかれば判定できる
dist関数を利用する

垂直な線分と円の接触





円の中心がこの中に入ると、線分と円が接触する

ここの座標は
(xTip,yTip)

このプログラムでは、円の半径は20。

仮引数

①ここに来ると処理がこの関数に移る、実引数の値が仮引数にコピーされる

```
boolean doesCollide(int x,int y,int xTip,int yTip){
  float d = dist(x,y,xTip,yTip);
  if(d <= 20){
    return true;
  }else if((abs(x - xTip) <= 20) && (y <= yTip)){
    return true;
  }else{
    return false;
  }
}
void setup(){
  size(400,400);
}
```

②ここに来ると戻り値が呼び出し元に戻される

```
void draw(){
  background(255);
  stroke(0);
if(doesCollide(mouseX,mouseY,width/2,height/3)){
    fill(255,10,10);
  }else{
    fill(100);
  }
  line(width/2,0,width/2,height/3);
  ellipse(mouseX,mouseY,2*20,2*20);
}
```

実引数

仮引数

```
boolean doesCollide(int x,int y,int xTip,int yTip){
  float d = dist(x,y,xTip,yTip);
  if(d <= 20){
    return true;
  }else if((abs(x - xTip) <= 20) && (y <= yTip)){
    return true;
  }else{
    return false;
  }
}
```

仮引数の有効範囲

仮引数の有効範囲は定義した関数の中だけ

関数が異なれば同じ名前の仮引数を使っても良い

```
戻り値のデータ型 関数名(データ型名1　引数名1,
                        データ型名2　引数名2…){
    関数処理の内容を書きます。
    どこかに、return命令が必要です。
    変数なども使うことができます。
    引数は自由に使うことができる。
}
```

組み込み関数

- 関数名と処理内容が既に決まっている
- 呼び出されるタイミングはプログラマが決める

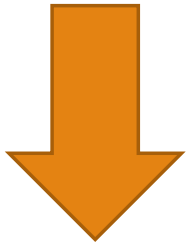
ユーザ定義関数

- 関数名と処理内容はプログラマが決める
- 呼び出されるタイミングはプログラマが決める

コールバック関数

- 関数名はProcessing側で決めている
- 処理内容はプログラマが決める
- 呼び出されるタイミングはProcessing側が決める

システム変数mousePressedなどを利用してクリックやドラッグの処理を書くのはかなり面倒



コールバック関数を利用する

コールバック関数の例

関数名	役割・呼び出されるタイミング
mousePressed	マウスボタンが押されたら
mouseClicked	クリックされたら
mouseMoved	マウスが移動したら
mouseDragged	マウスをドラッグしたら
mouseReleased	マウスボタンが離された
mouseWheel	マウスのホイールが回されたら
keyPressed	キーが押された（Shiftキーなどでも OK）
keyReleased	キーが押されなくなったら
keyTyped	キーが押されたら（Shiftキーなどはダメ）

基本的に戻り値は無し

押されたマウスボタンの種類：mouseButton変数でわかる
LEFT,CENTER,RIGHT

押されたキーの種類： key変数とkeyCode変数でわかる
key変数：どのキーが押されたかを保持している
key == ‘k’ ←小文字のkキーが押されたか？
key == ‘1’ ←1キーが押されたか？
key変数の値がCODEDの時は、特殊キーが押されたら、
どのキーが押されたかの情報がkeyCode変数に入っている

```
void mouseWheel(MouseEvent event) {
  float e = event.getCount();
  println(e);
}
void mousePressed(){
  println("mouse pressed "+mouseButton);
}
void mouseDragged(){
  println("mouse dragged");
}
void mouseReleased(){
  println("mouse released");
}
void mouseMoved(){
  println("mouse moved");
}void keyPressed() {
  println("pressed " + int(key) + " " + keyCode);
}
```

```
void keyTyped() {
  println("typed " + int(key) + " " + keyCode);
}
void keyReleased() {
  println("released " + int(key) + " " + keyCode);
}
void setup(){
  size(200,200);
}
void draw(){
  background(255);
}
```

スケッチ（プログラム）の提出に関して

キャリアポートフォリオ上で解答して下さい。
「スケッチをアーカイブ」で保存されるファイルをアップロードして下さい。

授業時に配布した資料

http://www.sato-lab.jp/imfu/index.html

においてあります。

データを与えて処理を行う

```
void 関数名(データ型名 引数名,…){
    処理内容を書く
}

関数名(引数に与えるデータ,…); // <-処理内容を実行できる
```

動かない理由

無限ループ

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

void draw(){
  background(255);
  fill(128);
  int x = 0;
  while(x >= 0){
    x = x + int(random(10));
    ellipse(x,height/2,10,10);
  }
}
```

デバッガを使うとデバッグ(debug)が楽になる

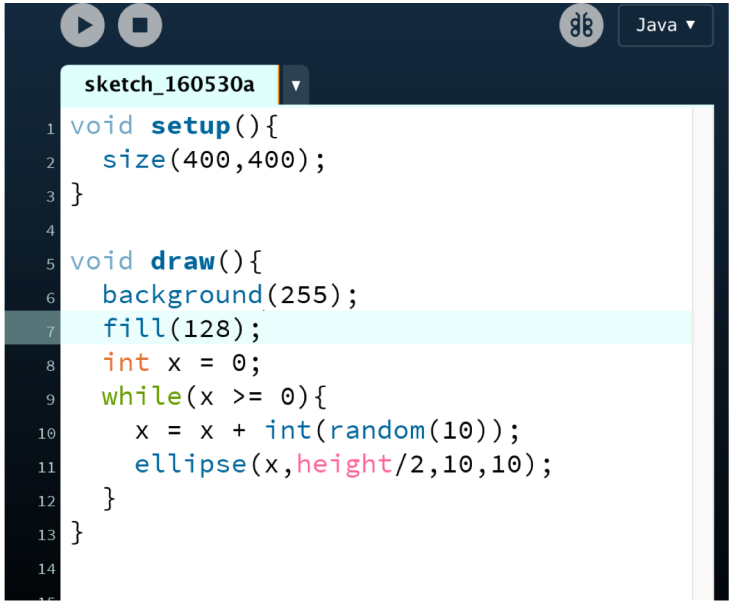
こんな時に便利

- 問題がありそうな処理のところで処理を止めて動きを確かめたい。
- 変数の値の変化を確認したい。
- 少しずつ動かしながら、動作を確認したい。

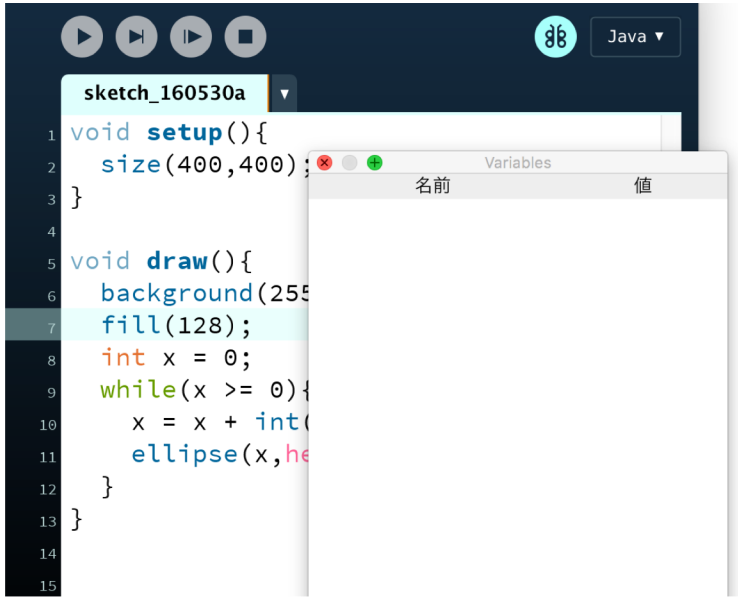
デバッグモードで出来ること

- 動きを確認したい処理のあたりにブレークポイントを設定し、実行。
  - ブレークポイント＝プログラムの動作を止めたい場所
- ステップ実行しながら、インスペクターで変数を確認しながら、期待の処理が行われている確認。

通常のウインドウ



デバッグ状態の ウインドウ



具体的には

1. ステップ実行：プログラムを1ステップ毎に実行できる機能
   - stepボタンを押すと、1ステップ毎に実行。(左端に実行行にマークがつく)
2. ブレークポイント:実行を停止したい行を設定できる機能
   - ブレークポイントの設定：設定したい行にカーソルを合わせて、
   - Runボタンを押すと、ブレークポイントを設定した行でプログラムが一時停止する。
3. インスペクター：変数名、値を確認できる機能
   - 変数名と値を確認できる

今日やった内容

関数の続き

コールバック関数

授業時に配布した資料

http://www.sato-lab.jp/imfu/index.html

においてあります。