コンピュータ基礎演習
追加課題(4) クラスとオブジェクト

理工学部 情報科学科 隅田 康明
sumida@ip.kyusan-u.ac.jp

追加課題について

- 本来の授業内容から削った内容をいくつか出題
  - 14回授業が13回になったことの補填
  - 遠隔授業により削減した内容の提示

- 制作課題まで提出して、余裕があれば取り組みましょう
  - あくまで、やる気のある学生向けの内容

- いくつか出題する追加課題のうち、1つを提出すれば加点
  - 追加課題の提出で最大+5点
  - ただし、合計の評点は100点を上限とする

想定動作端末

- 追加課題は大学のパソコン教室等での動作を想定
  - 所持端末によっては動作しないものもあるので、動作するものを選んで作業すること

追加課題４
クラスとオブジェクト

はっきり言って難しいないようですが、理解出来れば、プログラミングの幅が大きく広がります

（今後もプログラミングを続けるなら必ず出てくる内容）

変数をまとめる（クラス化）

- 図形に複雑な動きをさせる場合、
  変数が沢山必要になり、管理が大変になる

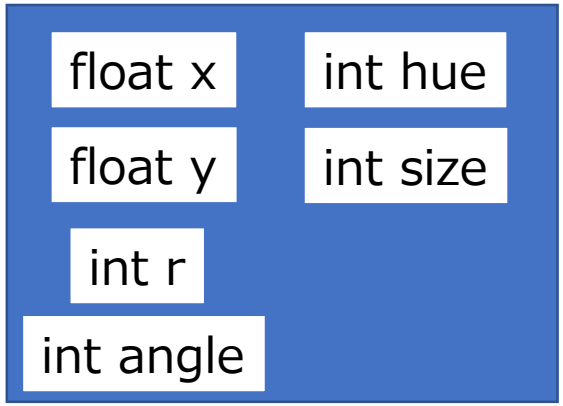
- 回転しながら、上下左右に跳ね返りながら、色が変わりながら、大きさも変わる図形のために必要な変数

```
float x = 0;
float vx = 1;
float y = 0;
float vy = 1;
int angle = 0;
int hue = 0;
int size = 0;
```

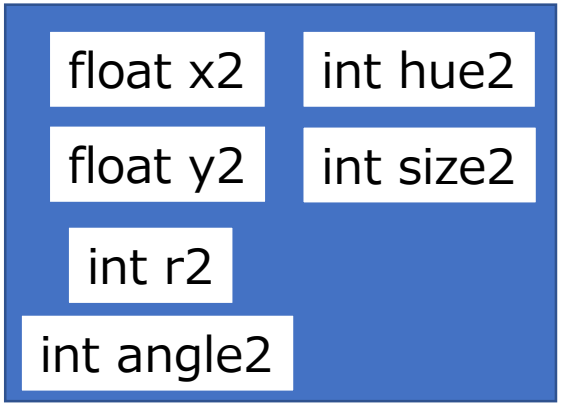
動く図形を増やす

- 図形と変数は１セット、図形が増えると変数も増える

周る四角形の変数



もう一つ増やすと



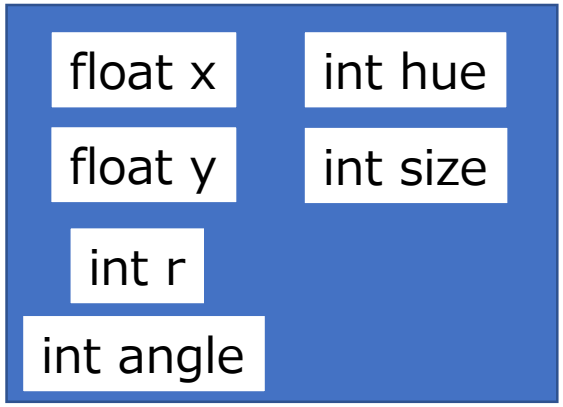

```
float x = 0;
float y = 0;
int r = 0;
int angle = 0;
int hue = 0;
int size = 0;
```

```
float x = 0;
float y = 0;
int r = 0;
int angle = 0;
int hue = 0;
int size = 0;
float x2 = 0;
float y2 = 0;
int r2 = 0;
int angle2 = 0;
int hue2 = 0;
int size2 = 0;
```

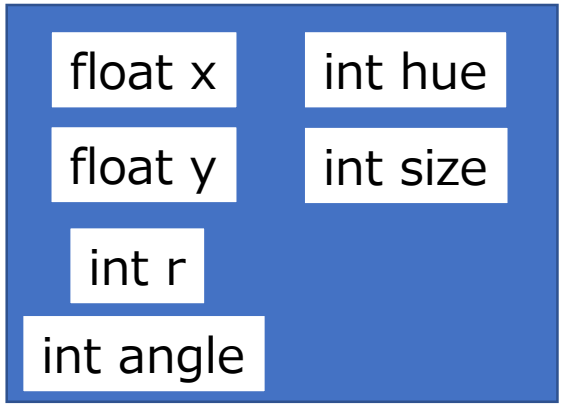
変数を一纏めにして管理する

- 使う変数の意味が同じなら、同じ名前で管理したい

図形１の変数（図形１のx）



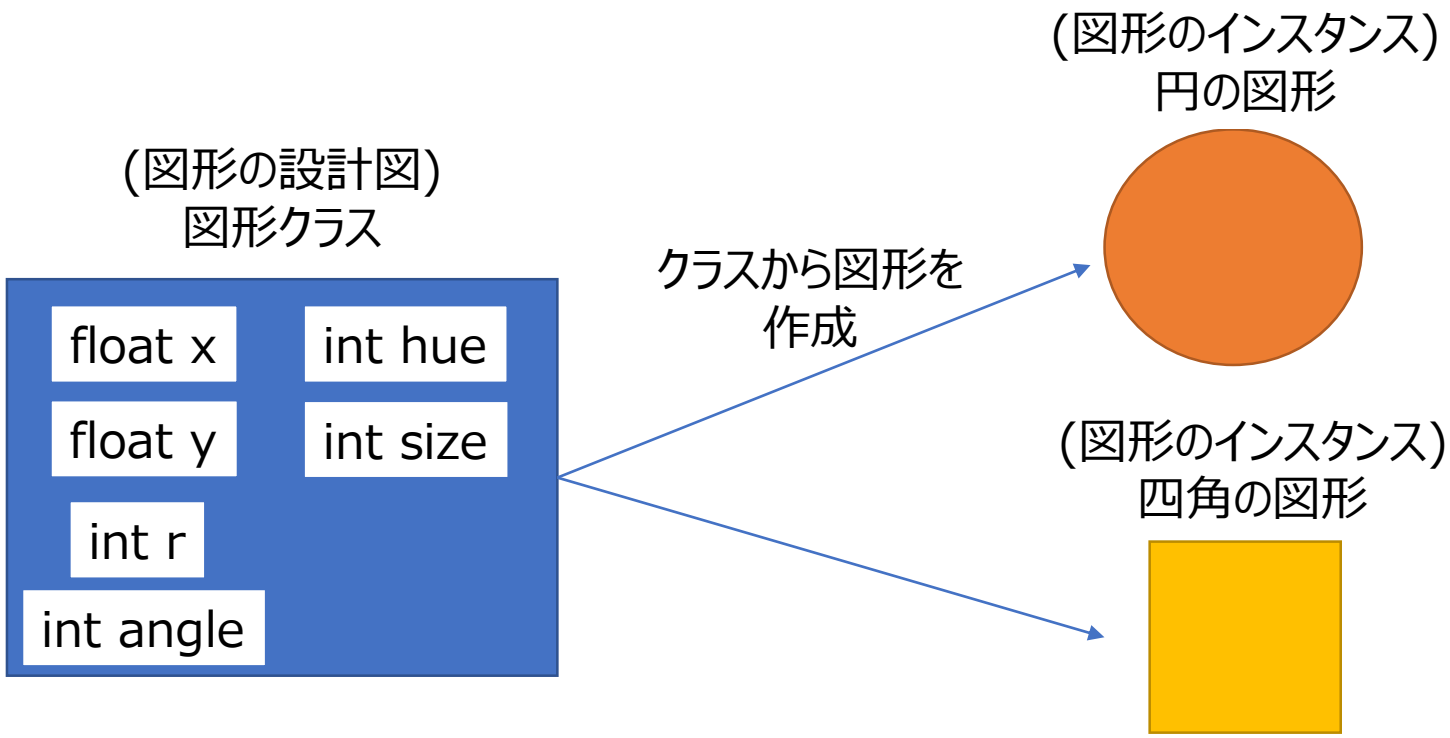
図形２の変数（図形２のx）



- クラスを作ると、
  同じような変数を宣言する手間を省ける

クラスとインスタンス

- クラス：物体の設計図
- インスタンス：クラス(設計図)を元に作る物体



図形クラス

- クラスの名前も自分で好きに付けてよい
  - 取り合えず変数を管理するだけのクラスを作ってみる

```
class Zukei{
  float x ,y;  //座標
  int cx, cy = 0;  //中心座標
  int angle = 0;  //角度
  int hue = 0;  //色相
  int size = 0;  //サイズ
  int red, green,blue;  //RGB(赤緑青)
}
```

動く図形に必要そうな変数をまとめたクラス

Zukeiクラスの変数宣言とインスタンス化

- 他の変数と同じように、作成したクラスも変数に出来る

```
int x;      //int(整数)型の変数xを宣言
float y;        //float(小数)型の変数yを宣言
Zukei sikaku = new Zukei();//Zukei型の変数sikakuを生成
Zukei maru   = new Zukei();//Zukei型の変数maruを生成
```

インスタンス化の構文：**クラス名 変数名 = new クラス名();**

半角スペース

インスタンス内の変数を使う

- インスタンスにまとめた変数を使うには、
  変数名.x　変数名.y　のように. (ピリオド) で繋ぐ
- sikakuの x に10を代入
  - sikaku.x = 10;
- sikakuの x の位置に円を描画
  - ellipse(sikaku.x, 10, 10, 10);
- sikakuの x を1増やす
  - sikaku.x = sikaku.x + 1;

クラスの使い方まとめ

1. クラスを作る

```
class Zukei{
  float x = 0;
  ・・・
}
```

2. 変数を作る

```
Zukei sankaku = new Zukei();
```

同じクラス名

3. 使用

```
sankaku.x = 10;
```

パターンを覚えれば難しくない

参考：跳ね返る図形

- 質問返しの続き（物足りない人用）

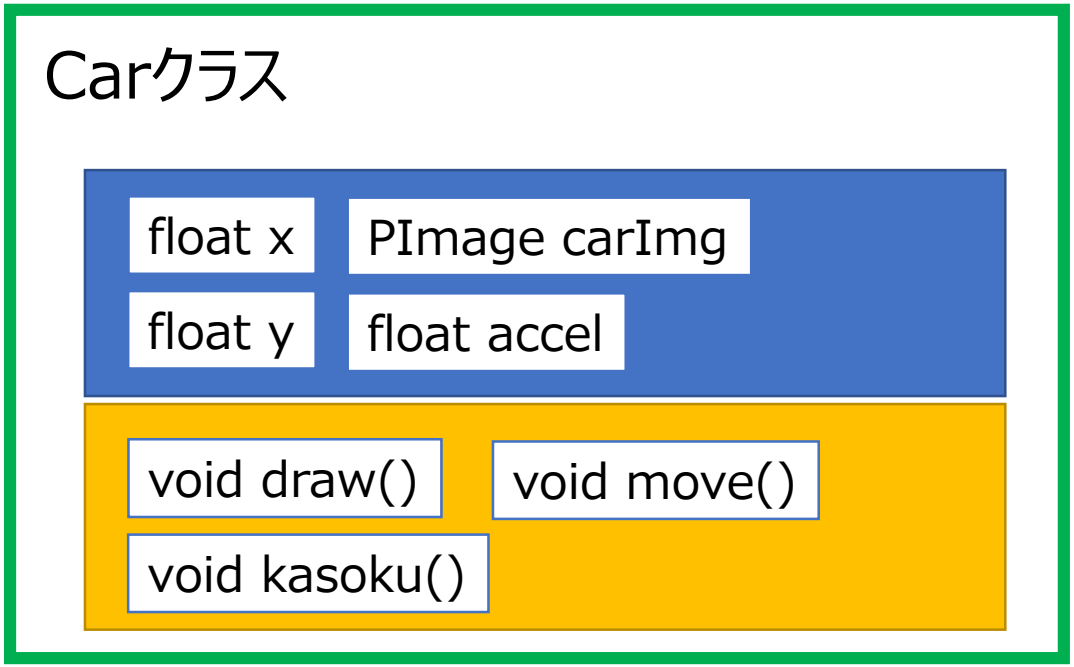
```
int x = 20;
int vx = 1;  //x方向の移動速度
void setup(){
 size(400,400);
}
void draw(){
  background(255);

  ellipse(x, 50, 10, 10);
  if(x >= width || x < 0) { //もしも、xがwidth以上か 0未満ならば
    vx = vx * -1;  //x方向の移動速度を反転（逆方向に移動）
  }
  x += vx;  //xにx方向の移動速度を足す
}
```

参考：動くタイミングをずらす

```
int x = 20;
int x2 = 40;
int count = 0;
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  ellipse(x, 50, 10, 10);
  if(count >= 100){ //もしも、countが100以上なら
    x += 1;  //xを1ずつ増やす
  }
  ellipse(x2, 60, 10, 10);
  if(count >= 200){//もしも、countが200以上なら
    x2 += 1; //x2を1ずつ増やす
  }
  count = count + 1;  //毎フレーム１ずつ増やす(時間を計る)
}
```

変数だけでなく、動きもクラスで管理

- 物体に合わせた動きを、一纏めにして定義しておく



クラスを定義する（作る）

- 放物線を描いて落ちるボールクラス

```
class FreeFallBall{  //名前は自分で好きに決める(最初は大文字が良い)
  float x = random(width), y = random(height);
  float vx = random(3)-1.5, vy = -random(5)-1;
  float g = 0.25; //重力加速度
  float sx = random(width), sy = random(height);
  //必要な変数があれば追加で宣言していく
  void paint(){ //このボールを描画するためのメソッド
    //このボールの動きに関係するプログラムだけを書く(背景等はここには書かない)
    ellipse(x, y, 10, 10);
    x += vx;   y += vy;
    vy+=g;  //vy(速度)を少しずつ早くすることで放物線の動きになる
    if(y > height){  //画面外に出たら
      vy = -random(5)-1; //y方向の速度を初期化
      x = mouseX; y = mouseY;  //初期値をマウス座標に
    }
  }
}
```

クラスの中のメソッドを呼び出す

```
FreeFallBall fb1 = new FreeFallBall();  //インスタンス変数の宣言
void setup(){//省略 }
void draw(){
  background(255);
  fb1.paint();  //変数の参照と同じ様に, . で繋いで呼び出す
}
class FreeFallBall{  //中身は省略
 //変数宣言省略
  void paint(){ //このボールを描画するためのメソッド
    ellipse(x, y, 10, 10);
    x += vx;    y += vy;
    vy+=g;  //vy(速度)を少しずつ早くすることで放物線の動きになる
    //省略
  }
}
```

何が便利か？

- 例えば、跳ね返るボールを描くプログラム

```
float x = 0; y = 50; vx = 1; vy = 1;
void setup(){
  size(400,600);
}
void draw(){
  background(255);
  ellipse(x, y, 10, 10);
  if(x > width || x < 0){
    vx *= -1;
  }
  if(y > height || y < 0){
    vy *= -1;
  }
}
```

• これに放物線を描いて落ちるボールを追加

```
float x = 0, y = 50, vx = 1, vy = 1;
float x2 = 200, y2 = 100, vx2 = 1, vy2 = 5, g = 0.25;
void setup() {
  size(400, 600);
}
void draw() {
  background(255);
  ellipse(x, y, 10, 10);
  x+=vx;
  if (x > width || x < 0) {
    vx *= -1;
  }
  y+=vy;
  if (y > height || y < 0) {
    vy *= -1;
  }
  ellipse(x2, y2, 10, 10);
  x2 += vx2;
  y2 += vy2;
  vy2+=g;
  if (y2 > height) {
    vy2 = -random(5)-1;
    x2 = 200;
    y2 = 100;
  }
}
```

跳ね返るボールの処理

放物線ボールの処理

drawの中がごちゃごちゃして分かりにくい

- クラス内メソッドを利用すると

```
BoundBall bb = new BoundBall();
FreeFallBall fb = new FreeFallBall();
void setup(){
  size(400,600);
}
void draw(){
  background(255);

  bb.paint();

  fb.paint();

}

class BoundBall{

}
class FreeFallBall{

}
```

それぞれの図形の描画は一行だけ

跳ね返るボールの処理

放物線ボールの処理

図形ごとの処理がまとまって分かりやすい

クラスアレンジ例：噴き出すボール


```
class FreeFallBall {
  int ballNum = 200;
  float g = 0.25; //重力加速度
  float[] x = new float[ballNum];   float[] y = new float[ballNum];   float[] sx = new float[ballNum];
  float[] sy = new float[ballNum];   float[] vx = new float[ballNum];   float[] vy = new float[ballNum];
  //初期化のためのメソッド,setup()内でfb1.init(200,100); のようにして呼び出す
  void init(float x2, float y2) {
    for (int i = 0; i < ballNum; i++) {
      x[i] = x2;
      y[i] = y2;
      sx[i] = x2;
      sy[i] = y2;
      vx[i] = random(3)-1.5;
      vy[i] = -random(5)-2.5;
    }
  }
  void paint() { //このボールを描画するためのメソッド
    for (int i = 0; i < ballNum; i++) {
      ellipse(x[i], y[i], 5, 5);
      x[i] += vx[i];
      y[i] += vy[i];
      vy[i]+=g;  //vy(速度)を少しずつ早くすることで放物線の動きになる
      if (y[i] > height) {  //画面外に出たら
        vy[i] = -random(5)-1; //y方向の速度を初期化
        x[i] = sx[i];
        y[i] = sy[i];  //座標を基点座標にリセット
      }
    }
  }
}
```

追加課題４のレポート

- 雛型をアレンジたプログラムを作ってレポートを提出
  - 出来れば面白いけれど、それなりに難しいですよ

- FreeFallBallクラスをアレンジ
  - 別の性質を持つクラスに改造する
    - 吹き出す物体を丸から変える
    - 跳ね返る動作などを付けてみる
  - FreeFallBallオブジェクトを動かしてみる
    - 跳ね返してみる
    - マウス座標から噴き出すようにしてみる