コンピュータ基礎演習

第5回

理工学部 情報科学科 隅田 康明

sumida@ip.kyusan-u.ac.jp

この授業は当面遠隔授業を継続します

- パソコン教室は、感染リスクが高くなる場所です
  - 室内に多くの学生が集まる
    - 教室に入室人数制限がかかっている
      - そもそも、大学の規則からして全員集めての実施は不可能
  - 不特定多数の人が触るPCを使う
  - 窓も１面しかなく、換気も不十分になる
- 遠隔でも出来る授業は、遠隔で実施していきます
  - 感染防止対策が不要になったら、別ですが
- 対面での質問受付などは、状況を見ながら検討
  - 仮に実施出来たとしても、人数制限は必須になります
    - 事前予約制などにしないと無理
- まだ予断を許さない状況なので、
  それぞれで感染防止対策はしっかりしていきましょう

大学の自習室等の利用について

- 6月1日から、制限付きで自習室などが利用可能になります
  - 総合情報基盤センター等のPCを利用可能に
    - https://www.cnc.kyusan-u.ac.jp/aboutus/003560.html
    - 大学のPCにはProcessing、PowerPointがインストール済み（全部ではないですが）
- 通信環境の問題で、授業の受講が困難な場合も、自習室などで遠隔授業を受けることができます
  - https://www.kyusan-u.ac.jp/news/coronavirus20200519-2/
- 自宅で授業を受けられない場合には検討しましょう

  ※大学で作業することを推奨しているわけではありません

講義用HPについて

- 講義資料や、講義に必要な情報は、
  この授業用のホームページに掲載しています
  - K'sLifeの通知にも添付していますが、HPの方がアクセスはしやすいはずです
    - K'sLifeに接続しにくい状況が続いています
  - 念の為、K'sLifeにもアップロードはしますが、
    基本的には講義HPを見るようにしましょう
- 下記のHPから、講義資料、動画のURL、
  Zoomの招待リンクなどを確認できます。

http://www.is.kyusan-u.ac.jp/~sumida/class/pckiso/

  - だんだんMoodleに移行してきます
    - Moodleに行けば全部ある！の方が楽ですよね

遠隔授業期間中の質問

- **困ったら早めに質問・相談！！**
  - 学生側から質問されないと、
    誰が困っているのか、何が分からないのか、分かりません
- メール：やり取りに時間はかかるが一番確実
- Zoom：授業時間中限定
  - 時間は限られるが、作業中の画面を見ながら教えられるので、問題を短時間で解決出来る可能性が高い
- Line OpenChat：授業時間中限定
  - 文字だけのやり取りに限定（画像アップロードは禁止）
  - 質問内容が他の学生にも分かるので注意すること
    - プログラムの全文貼り付け等は厳禁！

授業についての質問メールについて

[授業名(曜日時限)]についての質問	件名
～先生	誰宛か
[授業名(曜日時限)]を受講しています、 20AA999の九産太郎です。	何者か
（質問内容） --	用件
20AA999 九産太郎 九州産業大学　芸術学部　○○学科 1年	署名

2日返信がなければもう一度送って下さい（なるべく見落としたくない）

各回のレポートの確認（必ず確認しておくこと）

提出ステータス

提出ステータス	評定のために提出済み
評定ステータス	未評定
終了日時	2020年 05月 25日(月曜日) 00:00
残り時間	4 日 2 時間
最終更新日時	2020年 05月 20日(水曜日) 21:19

提出すると「評定のために提出済み」になる

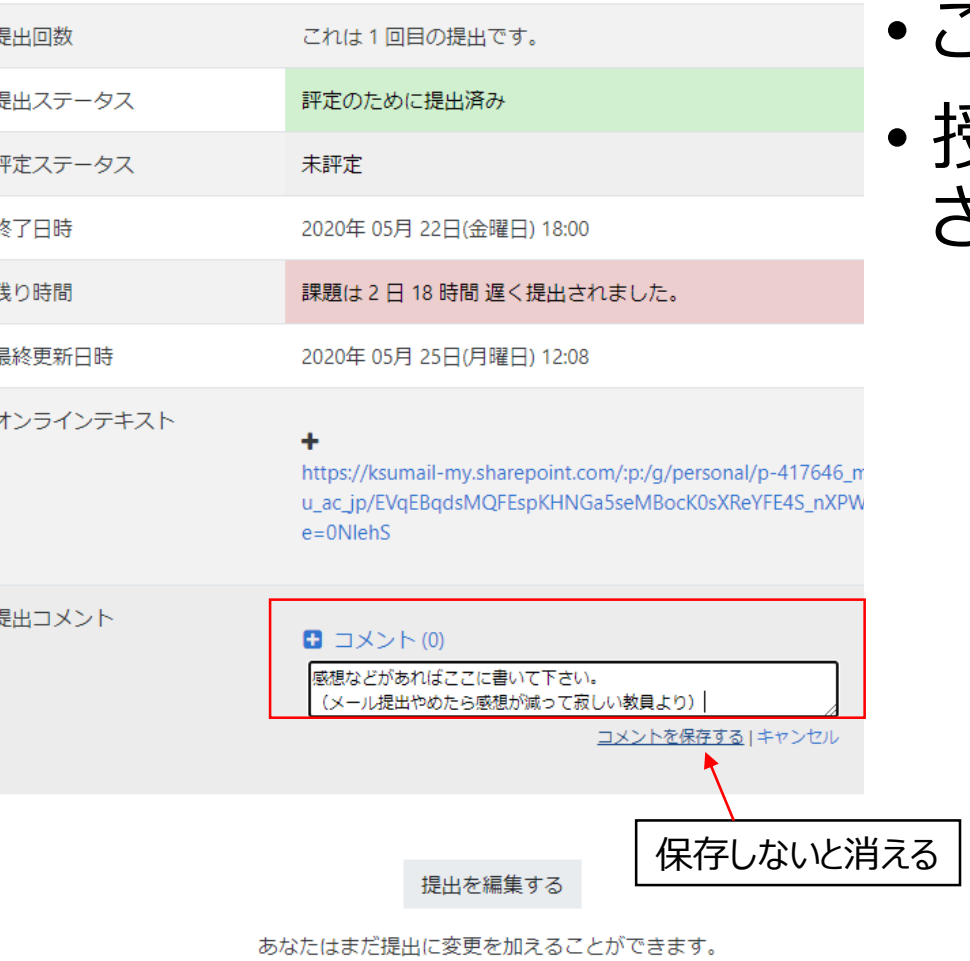
採点されると、「評定済み」になる

採点＝教員（私）が手動で点数を付ける

さらに下に、評点とフィードバックコメント（教員からのコメント）

毎回、何点だったかを確認しておこう（評点アップの交渉も遠慮せずにどうぞ）
（1回と2回はそのうち追加します(余裕が出来れば)）

Moodleのコメントについて



- こんな感じでコメント書けます。
- 授業の感想などはここに書いて下さい。
  - ただし、コメントへの返信は期待しないで下さい。
    - 返信欲しいことはメールで聞きましょう。
- 質問もメール等で！
  - レポートの採点はすぐに出来る訳ではありません（それなりに大変で時間もかかります）

今回の授業内容

- 動きのあるプログラム(アニメーション)の作成
  - 図形を動かしたり変形させてみる
  - setup と draw を使った構造を理解する
  - 変数の宣言と代入を理解する

- 発展：静止画を背景にする
  - 前回のプログラム(静止画)の上で図形を動かす
    - あるいは静止画の一部を動かす（高難易度）
  - 余裕のある人だけ

レポート（PowerPoint）の注意

- スライドにアニメーションを付けないこと：採点が大変になるので
  - 最終回に提出するレポートは別
- 「プログラム」スライドの目的
  1. 採点のため：動かないプログラムは評価出来ない
  2. バックアップのため
     - どこに保存したか分からない　→　Moodleにあります
  3. プログラムの再利用
     - 前回までのプログラムをコピペで応用
- コピー＆ペーストでProcessingで動かせることが大事
  - Moodleからレポートをダウンロードし、プログラムをコピー＆ペーストで動かせればOK
  - Web版のOffice365でスライドを開くと、
    コピペで動かない場合があるので、一度ダウンロードして、
    アプリ版のPower Pointで開いてコピペすること

「プログラム」スライドの例（こんなのでもOK）

はみ出してしまった（**問題なし**）

字が読めないほど小さい（**問題なし**）

Moodleからファイルをダウンロード→アプリ版のPowerPointで開く
→　全選択してコピー　→　Processingに貼り付け　で動けばOK

「プログラム」スライド、駄目な例

sketch_12_3D_enkei

```
float angle = 0;

void setup() {
  size(1000, 1000, P3D);
  colorMode(HSB, 360, 100, 100, 100);
}

void draw() {
  background(255);
  translate(width/2, 0);
  scale(0.5);
  for (int x = 0; x < 10; x++) {
    translate(100,100);
    rotateY(radians(angle));
    angle += 0.5;
    enkeiMethod(0, 0);
    rotateY(radians(30));
    enkeiMethod2(0, 0);
    rotateY(radians(30));
    enkeiMethod3(0, 0);
    rotateY(radians(30));
    enkeiMethod3(0, 0);
    rotateY(radians(30));
    roseCurve(0, 0, 360);
  }
}




void enkeiMethod(float x, float y) {  //メソッドの名前は分かり
  pushMatrix();  //現時点の座標系を保存
  translate(x, y);
```

- 画像として貼り付けている
- これは点数を付けません。
  出し直しを指示しているはずです。
  - 出し直さなければ0点になります。
    - 最終回までに出し直せば、減点もなしで受け付けますが。

「プログラム」スライド、駄目な例（分割してしまっているケース）

```
float angle = 0;
void setup() {
 size(1000, 1000, P3D);
 colorMode(HSB, 360, 100, 100, 100);
}

void draw() {
 background(255);
 translate(width/2, 0);
 scale(0.5);
 for (int x = 0; x < 10; x++) {
  translate(100,100);
  rotateY(radians(angle));
  angle += 0.5;
  enkeiMethod(0, 0);
  rotateY(radians(30));
  enkeiMethod2(0, 0);
  rotateY(radians(30));
  enkeiMethod3(0, 0);
  rotateY(radians(30));
  enkeiMethod3(0, 0);
  rotateY(radians(30));
  roseCurve(0, 0, 360);
 }
}
```

```
void enkeiMethod(float x, float y) { //メソッドの名前は分かりやすい名前にする
 pushMatrix(); //現時点の座標系を保存
 translate(x, y);
 float size = 50;
 for (int i= 0; i< 10; i++) { //10回繰り返す
  stroke(100, 100, 100, 50);
  fill(50, 100, 100, 50);
  rect(size, size, 10, 10);
  ellipse(size, size, 10, 10);
  rotate(radians(36)); //360度/繰り返し回数になるようにすると円形になる
 }
 popMatrix(); //保存した座標系に戻す
}
```

```
void enkeiMethod2(float x, float y) { //メソッドの名前は分かりやすい名前にする
 pushMatrix(); //現時点の座標系を保存
 translate(x, y);
 float size = 80;
 for (int i= 0; i< 10; i++) { //10回繰り返す
  stroke(300, 100, 100, 50);
  fill(200, 100, 100, 50);
  if (i % 2 == 0) { //もしも、i を2で割った余りが0なら（iが偶数なら）
   rect(size, size, 10, 10);
  } else { //そうでないなら（iが奇数なら）
   ellipse(size, size, 10, 10);
  }
  rotate(radians(36)); //360度/繰り返し回数になるようにすると円形になる
 }
 popMatrix(); //保存した座標系に戻す
}
```

減点まではしないけど、なるべく止めて下さい。
（これをする人が増えたら、この形式は禁止にします）

プログラム：2枚目

```
void enkeiMethod3(float x, float y) { //メソッドの名前は分かりやすい名前にする
  pushMatrix(); //現時点の座標系を保存
  translate(x, y);
  float size = 20;
  for (int i= 0; i< 10; i++) { //36回繰り返す
    stroke(i*50, 100, 100, 50);
    fill(i*10, 100, 100, 50);
    if (i % 2 == 0) { //もしも、i を2で割った余りが0なら（iが偶数なら）
      rect(size, size, 10, 10);
    } else { //そうでないなら（iが奇数なら）
      ellipse(size/2, size/2, 10, 10);
    }
    rotate(radians(36)); //360度/繰り返し回数になるようにすると円形になる
  }
  popMatrix(); //保存した座標系に戻す
}
```

```
void roseCurve(float cx, float cy, int loop) {
  pushMatrix();
  translate(cx, cy);
  strokeWeight(10);
  float r1 = 0, r2 = 0;
  for (int i = 0; i < loop; i++) {
    float a = sin(radians(r2)) * 0.8 + 1;
    float x =  a * cos(radians(r1)) * 50;
    float y =  a * sin(radians(r1)) * 50;
    ellipse(x, y, 1, 1);
    r1 += 1;
    r2 += 7;
  }
  strokeWeight(1);
  popMatrix();
}
```

コピペの時に面倒ですよね（私も面倒です＝採点が遅れます）。

どうしても、プログラムスライドを綺麗に整形したい人は・・・（やっても点数はあげません）

- **二度手間になるし、評点も上がらないので、スライドが汚くて我慢ならない！！という場合に限る**

- オンラインテキストにもプログラムを貼り付けておく
  - これでもOK。
  - プログラムスライドもちゃんと作っておくこと
    - 作品とプログラムが一つのPowerPointファイルで完結していると、後から見直すのが簡単
      - あのプログラムどこ行ったっけ・・・はありがち、ファイル名付けてもすぐ忘れます
      - PowerPointなら、エクスプローラー(MacならFinder）のプレビュー機能で絵からプログラムを探しやすくなる
      - 今はスマホで作業していても、そのうちPCを買ったときにプログラムを移せる
  - ちゃんと理由があって指示しているので、指示に従ってレポートを出すこと
    - 気になるならメールで聞いて下さい

プログラミングについての注意

- プログラミングは**最初は上手く行かなくて当たり前**です
  - 何度も失敗して、修正して、段々完成に近づけていきます
    - プログラムにエラーが出る、思ったとおりに動かない、は、日常茶飯事、ごく普通のこと
- 難しい内容も出てきますが、**最初は分からなくて当たり前**です
  - 試しに動かして、少し数値等を変えてみて、また試して、少し変えてみて、繰り返して理解していきます
  - 最初分からなくても、何となく動くものを作って行くうちに、だんだんと理解していくものです
- よく理解できない、自分にはプログラミングは向いていない…
  - そんなことないから、細かいことを気にする前に、プログラムを書いて、試してを繰り返しましょう
  - 最初は、**よく分かってないけど、書いたら動く、面白い！**でOK
    - 深く理解するのは、その後で

iPhoneでのプログラミングについて

- 前回も少しスライドで書きましたが、
  iPhoneのiProcessing Compilerでは、
  プログラムの間違いを指摘してくれません
  - 他は赤い波線で警告したり、
    何行目でエラーなど教えてくれます。

- iPhoneでエラーが出て困っているときには、
  OpenProcessingのエディターを利用する方法があります
  - HPかMoodleに別で資料を上げるのでそれを見て下さい
    - まだ早いかと思ってましたが、
      前回の時点ですごく頑張ってプログラミングをしてきた学生がいたので予定繰り上げ

Processingの基本形

- まずはこれを書いてからプログラミングスタート
  - 絶対に書く決まり事と思っておけば良い

```
void setup() {


}

void draw() {


}
```

setupメソッド
{} の中に最初に1度だけ
実行する内容を記述する
動かさないならここだけ使えばOK

drawメソッド
{} の中に**何度も繰り返し**
実行する内容を記述する
アニメーションを作るときに使う

これを書かないと、図形を動かせない。アニメーションでは必須！！

背景の書き方の違い

- setup(){　} 内に背景のプログラムを書く場合
  - 背景は一度しか描かれない
    - 最初から背景が描かれている紙に絵を描いていく
  - 完全に背景扱い（背景の上に絵を書いていく）
- draw(){ }内に背景のプログラムを書く場合
  - 背景が毎回描かれる
    - 常に上書きされる背景の上で絵を描く
      - 実際には背景もアニメーションの一部になっている
  - draw(){　} 内の一行目に background() を書くと、軌跡を残さないプログラムになる

setupとdrawについて注意

- setupメソッド
  - プログラム実行時に最初に実行する
  - 最初に1度だけ実行する命令を記述する場所

- drawメソッド
  - 何度も繰り返し実行する命令を記述する場所
  - setupの後に実行される

drawの{ }内の命令は何度も実行される
⇒　動きのあるプログラムが作れる！

今回の、演習の進め方

- まず、資料を読む、必要なら動画も見る
- 資料に書いてあるプログラムを書いて動かしてみる
- ある程度仕組みを理解できたら、ひな形をコピー
- コピーしたひな形をProcessingに貼り付ける
  - レポートの原型になる
- ひな形の穴を埋めてプログラムが動くようにする
  - **ひな形は、最初はエラーが出て動かない**
- レポートに取り組む
- レポート提出、ミニテスト受験

setupとdraw

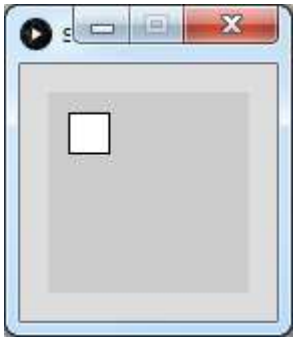
- setup(){　} と draw(){　}は、プログラムの中に、それぞれ一つしか書いてはいけない
  - setupが2つある、drawが2つあるは、プログラムが動かない

- setup(){　}の中に、或いは draw(){　}の中に、setup(){　} や draw(){　} を書いてはいけない
  - { から　} までで一つの括りになる
  - { 波括弧 を閉じ忘れないように気を付けること
    - **カッコは開いたら必ず閉じる！！**

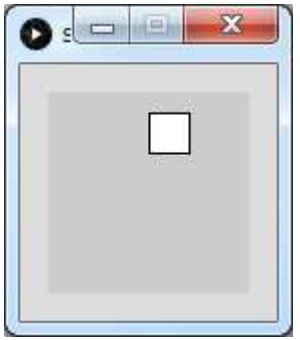
描いた図形を動かすには？

- 例えば、長方形を描く命令
  rect(10,10,20,20);

  座標(10,10)を左上に、
  幅20, 高さ20の長方形を描く



- 命令の座標を増やしてみると・・・
  rect(50,10,20,20);
  右に移動する

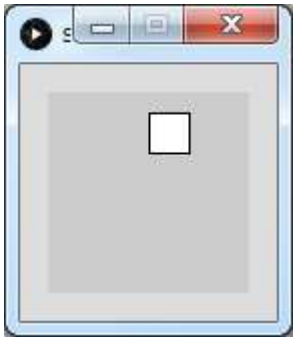


座標の数値を変えれば、図形を動かせる

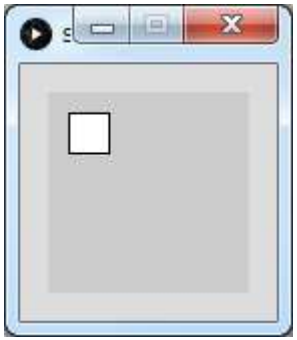
動かした図形を元に戻すには？

- さっき変えた命令
  rect(50,10,20,20);

  座標(50,10)を左上に、
  幅20, 高さ20の長方形を描く



- 命令の座標を減らしてみると・・・
  rect(10,10,20,20);
  左に移動する



座標の数値を増やせば右に、
減らせば左に移動する

座標を変えるには？

- これまでの書き方だと、座標は固定されたまま
  図形は毎フレーム同じ場所に描かれる
  （上書きされている）
  rect(50,10,20,20);

- 座標の数値を変えなければ、図形を動かせない

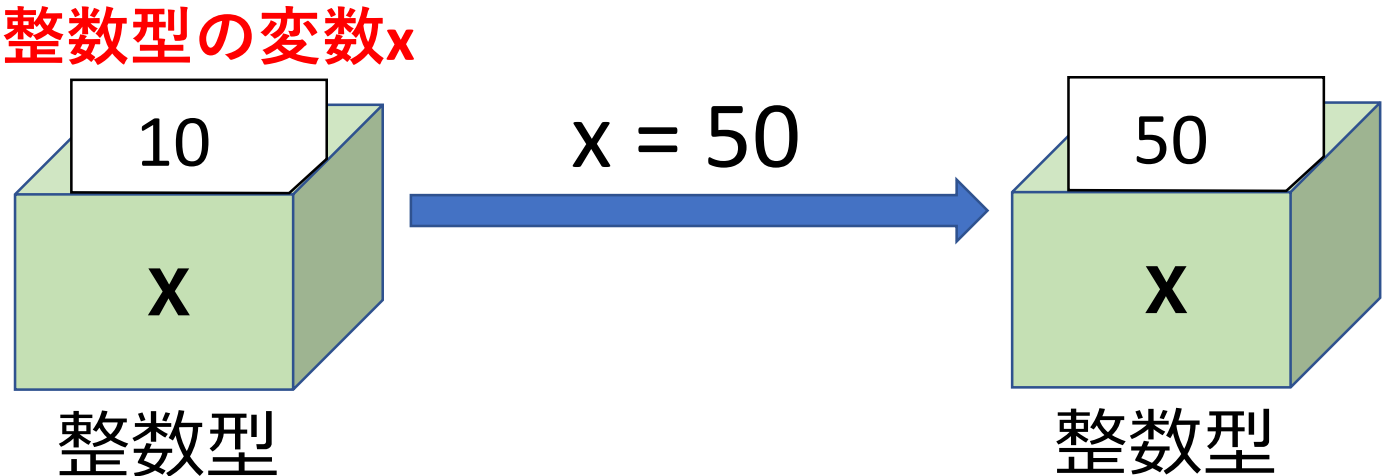
- 変数　：「変」わる「数」で変数

思い出してみよう（mouseXとmouseY）

- 2回と3回に使った命令
  - ellipse(mouseX, mouseY, 10, 10);
  - マウスを動かすと、円が描かれる場所が変わった
- mouseXとmouseYは、最初から用意されている変数
  - mouseX：マウスのX座標が入っている
  - mouseY：マウスのY座標が入っている
- マウスを右に動かすと、mouseXの値は大きくなる
  - mouseXの値が大きくなるので、図形は右の方に
- マウスを左に動かすと、mouseXの値は小さくなる
  - mouseXの値が小さくなるので、図形は左の方に

変数：値を保存できる箱のようなもの

- 変数:数値や文字列を格納（保存）しておける
- 変数には型と名前がある
  - 変数の型　：　格納できるデータの種類
  - 変数の名前　：　変数を区別するための名前
- 変数の中身は変えることが出来る



変数の型と名前

- 変数の型：数値や文字など色々な型がある
  - 整数：int
  - 小数：float
  - 文字列：string
- 変数の名前：自分で好きに付けられる
  - 付けられない変数名
    - 数字から始まる名前
    - _(ｱﾝﾀﾞｰｽｺｱ)以外の記号を含む名前
- 変数名も大文字、小文字はきっちり区別
  - **int X; と int x; は全く別の変数**になる

変数の宣言

- 変数を作ることを、変数の宣言という
- 宣言の仕方

変数の型　変数の名前;

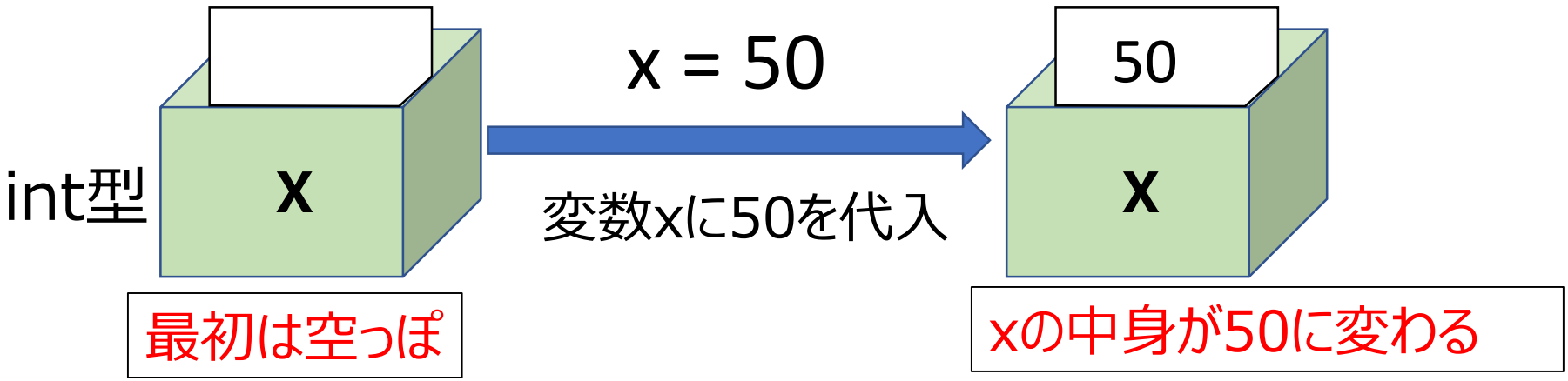
- 整数型の変数 x を宣言
  int x;
- 小数型の変数 yとz を宣言
  float y, z;

型が同じなら、複数まとめて宣言できる

代入

- 変数に値を入れることを代入という
- 代入の仕方

変数の名前　＝ 値;



※正確には、空っぽではなく初期値がはいる (int の場合は0)

変数に式を代入

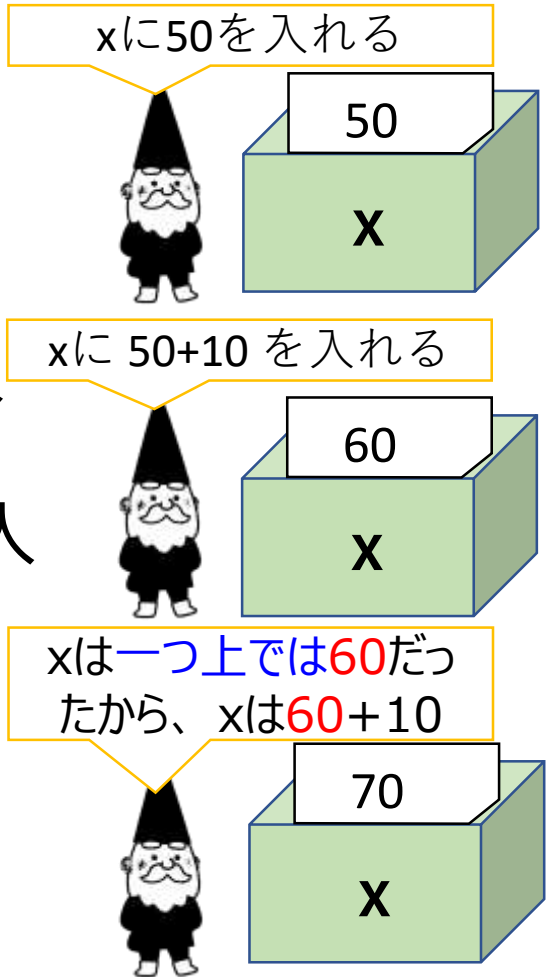
- 数値型の変数には、数だけでなく式も代入できる

```
int x;      //整数型の変数xを宣言
x = 50;  //xに50を代入

x = 50 + 10; //xに(50+10)を代入

x = x + 10; //xに(元のx+10)を代入
```

代入する式にその変数を使うと、
変数の数を増やしたり減らしたり出来る



宣言と代入を一緒にすることも出来る

```
int x;      //整数型の変数xを宣言
x = 50;  //xに50を代入
```

int x = 50;

この2つは同じ結果になる
（上のプログラムを2行続けて書いた場合に限る）

演算子

- 基本的な演算子

演算名	演算子	例
足し算	+	x = 10 + 1;
引き算	-	x = 10 - 3;
掛け算	*	x = 5 * 4;
割り算	/	x = 10 / 2;
割った余り	%	x = 2 % 1;

- 資料では、この演算子のみで説明する
- 次ページの演算子は、使うと少しだけ楽になる

便利な演算子

- 覚えておくと便利な演算子

演算名	演算子	例	別の書き方
加える	+=	x += 5;	x = x + 5;
引く	-=	x -= 2;	x = x – 2;
かける	*=	x *= 3;	x = x * 3;
割る	/=	x /= 10;	x = x / 10;
インクリメント（1増やす）	++	x++;	x = x + 1;
デクリメント（1減らす）	--	x--;	x = x – 1;
負数	-	x = -x;	x = x * -1;

楽になるだけで、使えないと困るわけではない

動きのあるプログラムを作る下準備

- 動きのあるプログラムを作る下準備

```
void setup() {
   size(400,400);
}
void draw() {
 background(255);
  ellipse(10,10,20,20);
}
```

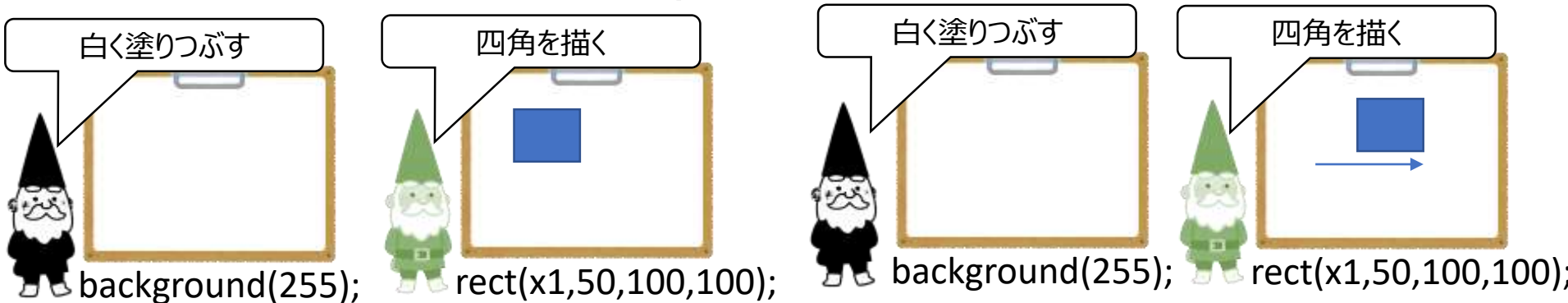
setup には画面サイズの設定などを記述しておく

背景を塗り潰す

丸を描く命令
繰り返し描かれるが、
同じ場所に上書きしている

アニメーションの仕組み

- 一見動いているように見えるが、
  実際には静止画を連続で表示している
- Processingでのアニメーション描画
  1. drawを実行し、図形を描画
  2. 変数の値を更新
  3. 画面全体を塗り潰す（background(255); 等）
  4. drawを実行し、図形を描画
     - これを繰り返すことで絵が動いているように見せる



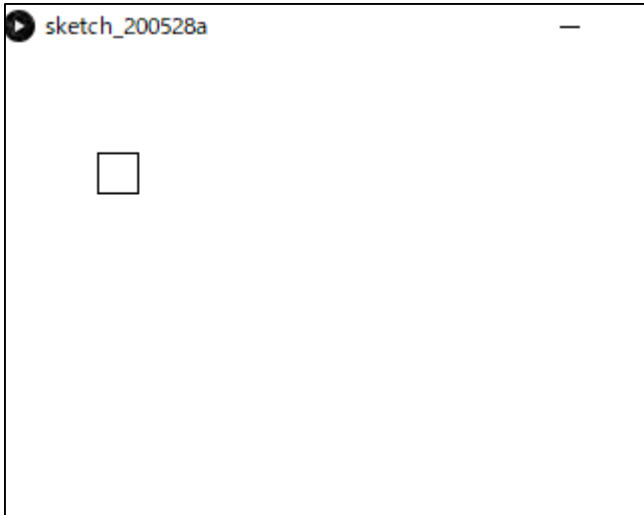
動く図形を作るときの手順（１）（右に動く図形を作る場合）

- プログラムの原型

```
void setup(){
 size(400,400);
}
void draw(){
  background(255);
}
```

- まず、動かない図形を作る（drawの中に書く）

```
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(50, 50, 20, 20);  //まだ動かない
}
```



動く図形を作るときの手順（2）（右に動く図形を作る場合）

- 動かしたい値の場所がどこかを考える
  - 右に動かしたい　⇨　横の座標を変える　⇨　x 座標
    - 図形描画の命令の数値を変えて試しておく

```
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(50, 50, 20, 20);  //まだ動かない
}
```

ここを変えると横に動く　→　ここを変数にする

まずは、どこの値を変えればいいか？　をはっきりさせる

動く図形を作るときの手順（３）（右に動く図形を作る場合）

- 動かすための変数を宣言する
  - 右に動かしたい　⇨　横の座標を変える　⇨　ｘ座標
    - 変数名：x1

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(50, 50, 20, 20);  //まだ動かない
}
```

同じ値にする（動き始める場所になる）

動く図形を作るときの手順（３）（右に動く図形を作る場合）

- 動かしたい値の場所を変数名と入れ替える

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20);  //まだ動かない
}
```

同じする（図形のx座標が変数になる）

動く図形を作るときの手順（４）（右に動く図形を作る場合）

- 変数の値を変えて、座標を変える

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20);  //x1 の値が変わることで動き始める
  x1 = x1 + 1;
}
```

四角形のx座標（ x1 ）を１ずつ増やす ＝ 右に移動する

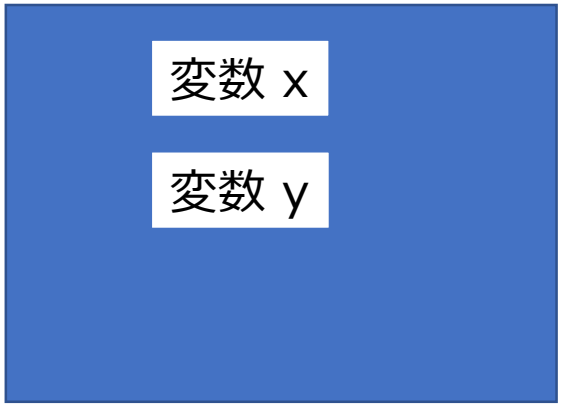
慣れるまでは、まず動かない図形を描いてから、どう動かすのかを考えよう

自分のやり易い方法で図形を動かせるなら、それでも良い

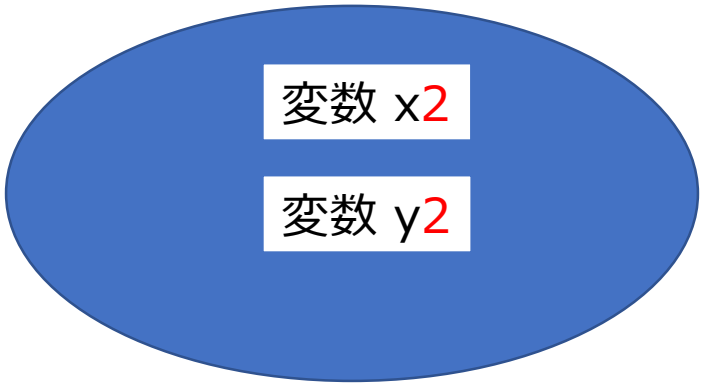
動く図形を増やす

- 図形と変数は１セット

四角の図形の変数



楕円の図形の変数



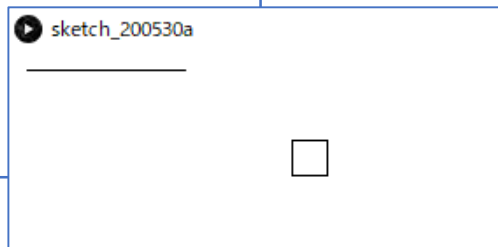
それぞれの図形用の変数を作らないといけない
同じ名前の変数は使えない

動く図形を増やす手順（１）

- 例：下に伸びる線を追加する場合
- まずは、動かない線を描く（動き始める前の状態）
  - どこを変えたらどう動くか、数値を変えて確かめておく

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20);  //x1 の値が変わることで動き始める
  x1 = x1 + 1;
  line(10, 10, 100, 10);  //(10,10) から (100,10)に線を引く

}
```



動く図形を増やす手順（２）

- 動かすための変数を宣言する
  - 下に伸ばしたい　⇨　終点のY座標を増やせばいい
    - 変数名：y2　(2つ目の図形なので2とつけておく)

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
int y2 = 10;
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20); //x1 の値が変わることで動き始める
  x1 = x1 + 1;
  line(10, 10, 100, 10); //(10,10) から (100,10)に線を引く

}
```

同じする（終点のy座標が変数になる）

ここを変えると下に伸びる　→　ここを変数にする

動く図形を増やす手順（３）

- 動かしたい値の場所を変数名と入れ替える

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
int y2 = 10;
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20);  //x1 の値が変わることで動き始める
  x1 = x1 + 1;
  line(10, 10, 100, y2);  //(10,10) から (100,10)に線を引く

}
```

同じする（終点のy座標を作った変数に置き換え）

動く図形を増やす手順（4）

- 変数の値を変えて、座標を変える

```
int x1 = 50;  //動かしたい図形の元の数値と同じ値を代入
int y2 = 10;
void setup(){
 size(400,400);
}
void draw(){
  background(255);
  rect(x1, 50, 20, 20);  //x1 の値が変わることで動き始める
  x1 = x1 + 1;
  line(10, 10, 100, y2);  //(10,10) から (100,10)に線を引く
  y2 = y2 + 1;
}
```

線の終点のY座標を1ずつ増やす → 線が下に伸びていく

変数を使って、図形を変化させよう

- 色を変える
  - stroke(255, 0, 0, 255);
  - 透明度を少しずつ減らすと・・・？
- 大きさを変える
  - ellipse(200, 200, 10, 10);
  - 幅と高さを増やしていくと・・・？
    - ellipse(200, 200, s3, s3);
      のように同じ変数を使うこともできる

今回のレポート

1. ひな形プログラムをコピーして、Processingに貼り付け
   - 穴を埋めて、プログラムが動くようにする
   - 左に動く丸のプログラムを完成させる
2. 図形に色を付ける（色の付け方は4回の内容を参照）
   - ellipse の前に色を付ける命令を追加
3. 動く図形を最低2個増やす（全部で3個）
   - または、ひな形にある図形の動きを追加する
     - 下に動くだけでなく、大きくなりながら下に動くなど
   - **右に動く図形、下に伸びる線はカウントしない**
4. 自由にアレンジ
   - ３までで最低基準はクリア
   - 後は自由にアレンジしよう
     - 変数を使った工夫の数で評点アップ

第5回ひな形：HPからもコピー可

```
int x1 = 350;   //動かしたい図形の元の数値と同じ値を代入
void setup(){
 size(400,400);
 colorMode(HSB, 360, 100, 100, 100);
}
void draw(){
  background(255);
  //色を付ける
  ellipse(??,150, 50, 70); //穴埋め：x座標には何が入るか？
  x1 =  ??;  //穴埋め：x1に数を足す、または引く
}
```

??を埋めて、左に動く丸を作る

動く図形を追加する（動かし方は自由）
最低2つ（合わせて3つ）の図形を動かそう

評点：シンプルにします

- 最低基準（チェックリスト参照）を満たせば3点
  - ここまでは同じ

- 加点項目(正確には加点とは違うが)
  - 取り組めば評点アップ
    （頑張ったレポートには相応に加点）

- 高度な内容（ここまでしなくても満点は取れる）
  - 物足りない学生向けの内容。
  - やる気があって、余裕がある場合にのみ挑戦しましょう。

レポートチェックリスト（第5回）

- □ミニテストを受験した(レポート提出の前でも後でも可)
- □雛形プログラムをコピーしてProcessingに貼り付けた
  - **●この時点ではプログラムは動かない**
- □雛形プログラムの？？を書き換えて、図形を動かした
  - □指定通りに動いた（エラーなく実行できた）
- □図形に色を付けた
- □動く図形を追加した（1つ目）
- □動く図形を追加した（2つ目、合わせて3個の動く図形）
- □PowerPointでレポートを作成した
  - □タイトル、作品紹介(工夫点)、プログラムの3枚
- □Moodleでレポートを提出した

PowerPointでレポート作成

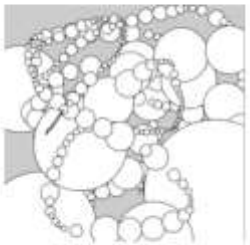
- 1枚目：タイトル

PC基礎 05

学籍番号　氏名

- 2枚目：実行画像

タイトル：（作品のタイトル）



- 説明
  - ～～をイメージして作った
  - マウスの動きに合わせて、丸の大きさが変わるように工夫した

どんなものを作ろうとしたのか？

どんな工夫をしたのか？

簡単でいいので説明を書く

- 3枚目：プログラム

プログラム

プログラムを全て貼り付ける

```
void setup(){
  size(400,400);
  colorMode(HSB,360,100,100,100);
}
void draw(){
  float d = dist(mouseX, mouseY, pmouseX, pmouseY);
  ellipse(mouseX,mouseY,d,d);
}
```

貼り付けたときに、文字が見えない程小さくなっても構わない

これは第3回のレポートの例

実行画面やプログラムは、
（当然）今回の内容にすること

必要事項が揃っていれば、レイアウトは自由

加点項目(最低基準点より上の点)

- 下記をすれば最低基準点にプラス（最高5点）
  - 動く図形をさらに追加した
  - 複数の図形を一塊にして移動させた
  - 斜めに移動しながら大きくなる円、など、より複雑な動きをさせた

  - 高度な内容：円の軌道を描く図形を追加
  - 高度な内容：前回のプログラムと合体
    - これを提出する場合、最低基準の左に動く丸と、大きさが変わる四角形はなくてもよい。ただし、３つ以上の図形を動かすこと
    - 背景の一部を動かす
    - 難易度が高い内容。やってみたい人だけ。
    - 詳しくは後のページを参照

よくありそうな質問

- 図形が画面の外に出て戻ってこない
  - その動きで正常です
  - 戻ってくるようにするには、下記のようにするといい

```
x1 = x1 + 1;
x1 = x1 % width;

y1 = y1 + 1;
y1 = y1 % height;
```

  - 左に行って右端に戻る、などは条件分岐が必要(第7回)
  - 跳ね返る動きも、条件分岐が必要

高度な内容：揺れながら落ちる

```
float x,y;
int angle;
float theta;
void setup() {
  size(400,400);
  frameRate(30);
  x = 100;
}
void draw() {
 background(255,255,255);
 theta = radians(angle);
 x = 20 * cos(theta) +100;
 ellipse(x,y,20,20);
 y = y + 1;
 angle += 5;
}
```

- ふらふら下に落ちる動き
- x座標を三角関数で計算

高度な内容：ぐるぐる回る図形

```
float rx, ry;
int angle;
float theta;
void setup() {
   size(400,400);
   frameRate(30);
}
void draw() {
  background(255,255,255);
  stroke(255,0,0);
  theta = radians(angle);
  rx = 50 * cos(theta) + 200;
  ry = 50 * sin(theta) + 200;
  angle = angle + 5;
  rect(rx,ry,10,10);
}
```

rx, ry ：円の座標
angle ：角度
theta：ラジアン単位の角度

thetaに角度をラジアン単位変換値を代入

rxとryに円周上の座標をそれぞれ代入
50 は円周の半径、200は円周の中心座標

角度を5度増やす

四角形を描画

挑戦的な課題：＋前回のプログラム

- 前回作った静止画の一部を動かす
  - やってみたい人向けの紹介です
    - ここまでしなくても満点になる
  - やる気があって、余裕がある場合に取り組みましょう

1. 前回のプログラムの静止画描画部分をコピーして貼り付け
   - void draw(){ }の中に静止画部分を貼り付ける
   - マウスで絵を描く部分は今回は使わない(使ってもいい)
2. 静止画の一部の図形を動くように修正する

例：

- 第4回のレポートのプログラムがこうだった場合

```
void setup(){
  size(400,400);
  stroke(0,0,0);
  fill(255, 255, 0);
  rect(100, 150, 150, 80);
  fill(0, 0, 0);
  ellipse(130, 230, 30, 30);
  ellipse(220, 230, 30, 30);
  fill(0, 0, 255);
  rect(225,160,20,40);
  fill(127, 255, 212);
  rect(200,160,15,20);
  rect(175,160,15,20);
}
void draw(){
}
```

赤字部分（背景を描く部分のみ）をコピー

｛　や　｝を余計にコピーする、
削除するなどがないように十分注意すること

```
void setup(){
  size(400,400);
}
void draw(){
  background(255):
  stroke(0,0,0);
  fill(255, 255, 0);
  rect(100, 150, 150, 80);
  fill(0, 0, 0);
  ellipse(130, 230, 30, 30);
  ellipse(220, 230, 30, 30);
  fill(0, 0, 255);
  rect(225,160,20,40);
  fill(127, 255, 212);
  rect(200,160,15,20);
  rect(175,160,15,20);

}
```

backgroundを追加(毎回塗りつぶし)

draw 内に貼り付け

draw の { } の対応が崩れていないか？
よく注意すること。

```
int x1 = 130;
void setup(){
  size(400,400);
  //ここまでの間にプログラムを書く
}
void draw(){
  background(255);
  stroke(0,0,0);
  fill(255, 255, 0);
  rect(100, 150, 150, 80);
  fill(0, 0, 0);
  ellipse(x1, 230, 30, 30);
  x1 = x1 +1;
  ellipse(220, 230, 30, 30);
  fill(0, 0, 255);
  rect(225,160,20,40);
  fill(127, 255, 212);
  rect(200,160,15,20);
  rect(175,160,15,20);

}
```

配置している図形を動くように、
変数を作って変えていく