

Part1. プログラミングテクニック

ゲームの共同開発のために必要なプログラミングのテクニックたち。

1. レポジトリ
2. 命令を作る
3. クラス
4. テキストの取り扱い
5. キー入力
6. 画像ファイルの取り扱い
7. ピクセル単位の編集
8. XML ファイルの取り扱い

1. レポジトリ

プログラムの共同開発をするとき、皆で作ったデータを貯めておくところをレポジトリという。本格的に使うのはPart2からだが、とりあえずここでアカウントを作成して、いくつかの教材をダウンロードしてみよう。

- ・アカウントを取得しよう

<https://codebreak.com/>から登録する。登録したらアカウント名を先生に教えてほしい。

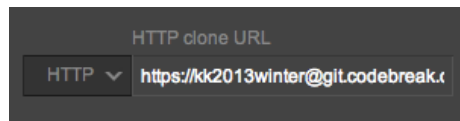
- ・教材を取得してみよう

登録したアカウントでサインインする。ページが英語かも知れないが、右下の方に「日本語」というボタンがあってこれを押すと日本語になる。

無事に進めば、左の方にリポジトリ一覧という表示(下図、左)が出るはずである。共同作業の部分をクリックして、下に出てくる

matty/kyouzai.git
というのにアクセスしよう。

すると、右上の方に下図の右のような欄が現れるので、ここに出てきたアドレスをコピーしよう。



次に、デスクトップに自分の名前のフォルダを作り、さらにその中に「kyouzai」という名前のフォルダを作ろう。そのフォルダに入ったら、<https://codebreak.com/ja/contents/guide/gui/>の中の「コードブレイクで利用する場合の設定はこちら」に従ってリポジトリを設定する。Step5のURLに、さっきコピーしたアドレスを用いる。

最後に、フォルダの中で右クリックし、tortoiseGit→プルを押して、OKを押していけば、このフォルダに教材が取得されるはずである。

※Processing の予備知識について

- ・ 基本文法

変数、if、for、多重ループ、一次元配列、（多次元配列）

- ・ 図形の描画

rect, ellipse, line, point, text
fill, stroke, background

- ・ 基本的な命令

println
random
sqrt, dist, max, min

- ・ アニメーションの方法

setup, draw, frameRate

- ・ マウス操作

mousePressed, mouseX, mouseY

Processing の経験があまりない（他の言語はある）人は入門編テキストの次のところをやるべし。

- ・ Part1 の 1-1 から 1-3 を読み、1-4 の練習問題を解く
- ・ Part2 の 2-2, 2-3, 2-6 の問題を解く
- ・ Part3 を 3-7 のテニスゲームを自力で作れるようになるまでやる
- ・ Part4 を 4-3 の (5) の避けるだけシューティングが作れるようになるまでやる

2. 命令を作る

(Processing で命令を作る方法を知っている人は飛ばして良い)

・基本

次のプログラムを実行してみよう (Part1_2_maru)。これはマウスの位置に○を描画するプログラムである。

```
void setup(){  
  size(300, 300);  
}
```

```
void draw(){  
  maru(mouseX, mouseY);  
}
```

```
void maru(float x, float y){  
  ellipse(x, y, 10, 10);  
}
```

これからファイル名が出てきたら
それを開くこと

自作命令の呼び出し

自作命令の定義

draw ブロックの中には ellipse 命令はなく、代わりに maru という名前の命令が呼び出されている。これが3つ目のブロック(void maru ...)で作られている「自作命令」である。


maru(mouseX, mouseY)の mouseX, mouseY は、maru 命令に渡す引数という。maru 命令の定義で void maru(float x, float y)... となっており変数 x, y が作られているように見えるが、この変数 x に渡した引数 mouseX が、y に mouseY が代入され、自作命令の中でこれらの値を使うことが出来る。

draw 命令の中に

```
maru(30, 60);
```

という行を書き足して、本当に(30, 60)の位置に円が描かれるか確認しよう。

次の空欄を埋めて(x0, y0), (x1, y1), (x2, y2)を頂点とする三角形を描く命令 sankaku を作れ。それを draw の中で呼び出せ。

```
void sankaku(float x0, float y0, float x1, float y1, float x2, float y2){  
    
}
```

・ 戻り値

命令の中にはその呼び出し元に「返事」をするものがある。たとえば今まで使ってきたルートを求める命令 `sqrt(x)` は、`x` のルートを返事として返していた。これによって、

```
float rx = sqrt(x);
```

のような処理(`sqrt(x)` の返事を変数 `rx` に代入)が可能だったのである。このような返事として返される値のことを戻り値という。

次のプログラム (Part1_2_3_saikoro) を動かしてみよう。

```
void setup(){
  size(300, 300);
}

void draw(){
  int m = saikoro();
  println(m);
}

int saikoro(){
  return (int)random(1, 7);
}
```

ここでは `saikoro` という名前の自作命令が作られている。この命令は 1 から 6 までのランダムな整数を戻り値として返す。先ほどの `maru` 命令と比較してみると、上の赤で囲った部分が `void` でなく `int` となっていることがわかる。この場所には戻り値が何の型であるかを書くことになっていて、`saikoro` 命令は整数を返したいので `int` である。`void` は戻り値なしという意味である。実際に返す値は、`return` の後に続けて書く。`return` のある行が実行されるとその命令は、`return` の次にかかれた値を戻り値として返して呼び出し元に帰還する。ちなみにこの命令は引数をとらないので、カッコの中身が空である。

次の二つの命令の意味を考えてから、実際に使って確かめてみよ。

```
int sqx(int x){
  return x*x;
}

int mx(int x, int y){
  if(x>y){
    return x;
  }else{
    return y;
  }
}
```

・まとめ

命令を自分で作ることが出来る。何度も行う処理などを自作命令に書けば、プログラムがすっきりする。

命令は、他の場所から呼び出すことでその中身が実行される。

命令の中では、引数として与えた変数を使うことが出来る。

命令から、呼び出し元に戻り値として値を返すことが出来る。

今まで使ってきた `println(...)` とか `ellipse(...)` とかも命令である。これらは自作ではなく、あらかじめ用意してある命令である。

今まで書いてきた `void setup() { ... }` とか `void draw() { ... }` とか `void mousePressed() { ... }` とかはみんな命令の定義である。ただしこれらは「名前が定められた命令」で、この名前の命令は特別な役割を持つ。setup なら起動時に実行、draw なら次々と実行、mousePressed ならマウスが押された時のみ実行される。

構文 命令を作る

作る命令の名前
`void hoge(){`
...
`}`
引数なし、戻り値なし
`void` は戻り値なしを表す

引数の型
引数名
`void hoge(int a, int b, ...){`
...
`}`
引数あり、戻り値なし
小カッコのなかに引数を列挙する

戻り値の型
`int hoge(){`
...
`return ...;`
`}`
引数なし、戻り値あり
`return` の後ろに戻り値を書く

`int hoge(int a, int b, ...){`
...
`return ...;`
`}`
引数あり、戻り値あり

・練習

次の命令たちの意味を考えて、実際に呼び出してみてそれを確認せよ。

```
int r1(){
    return 1;
}

void hello(){
    println("Hello!");
}

void nhello(int n){
    for(int i=0; i<n; i++){
        hello(); // 命令の中で他の命令を呼び出しても良い
    }
}
```

次のような命令を作れ。

- ・整数の引数を3つとり、そのなかで最も大きいものを戻り値として返す命令
- ・整数の引数 n をとり、 n 個の円をランダムな位置に描く命令
- ・整数の二つの引数 x , y をとり、戻り値として x の y 乗を返す命令

3. クラス

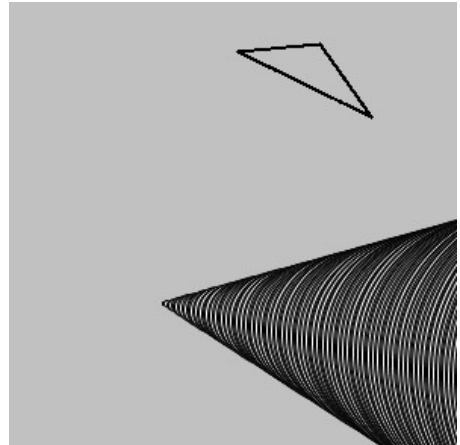
プログラムを複数人で協力して開発するときはどうやって役割分担をするかが重要であるが、クラスというものを使うことで、プログラムに含まれる変数や自作命令たちを、役割ごとに分割することが出来る。

実際に例で見てみよう。次のプログラム (Part1_3_1_class) は、ランダムな位置に三角形を描き、また、徐々に大きくなりながらあるランダムな方向に進んでいく円を描画するプログラムである。

```
float rx, ry, rr;
float rvx, rvy;
float tx0, ty0, tx1, ty1, tx2, ty2;

void setup(){
  size(300, 300);
  rx = 100;
  ry = 200;
  rvx = random(-5, 5);
  rvy = random(-5, 5);
  rr = 0;
  tx0 = random(0, width);
  ty0 = random(0, height);
  tx1 = random(0, width);
  ty1 = random(0, height);
  tx2 = random(0, width);
  ty2 = random(0, height);
}

void draw(){
  ellipse(rx, ry, rr, rr);
  rx += rvx;
  ry += rvy;
  rr += 2;
  line(tx0, ty0, tx1, ty1);
  line(tx1, ty1, tx2, ty2);
  line(tx2, ty2, tx0, ty0);
}
```



変数がたくさんあってごちゃごちゃしているのがわかる。rx と tx0 がどっちが円でどっちが三角形に関する変数なのか分からなくなりそうだし、途中でどこか一行忘れていたりしても気づかなさそうだ。

これをクラスを使って書き直すと、次のようになる (Part1_3_1_class2)。

```
Maru maru;
Sankaku sankaku;

void setup(){
    size(300, 300);
    maru = new Maru(100, 200);
    sankaku = new Sankaku();
}

void draw(){
    maru.move();
    maru.draw();
    sankaku.draw();
}
```

メインクラス

そして次の二つの「クラス」を新たに定義する。

Maru クラス

```
class Maru{
    float x, y;
    float vx, vy;
    float r;

    Maru(float x0, float y0){
        x = x0;
        y = y0;
        r = 0;
        vx = random(-5, 5);
        vy = random(-5, 5);
    }

    void move(){
        x += vx;
        y += vy;
        r += 2;
    }

    void draw(){
        ellipse(x, y, r, r);
    }
}
```

Sankaku クラス

```
class Sankaku{
    float x0, y0, x1, y1, x2, y2;

    Sankaku(){
        x0 = random(0, width);
        y0 = random(0, height);
        x1 = random(0, width);
        y1 = random(0, height);
        x2 = random(0, width);
        y2 = random(0, height);
    }

    void draw(){
        line(x0, y0, x1, y1);
        line(x1, y1, x2, y2);
        line(x2, y2, x0, y0);
    }
}
```

プログラムの量は長くなってしまっているものの、ごちゃごちゃ感は減って、何をしたいのかが分かりやすくなったと思わないだろうか？そして、プログラムが3つに別れたことで、3人で分担して開発することもできそうである。

ここからはクラスの文法を解説する。

・クラス定義の概要

クラス名

```
class Sankaku{
    float x0, y0, x1, y1, x2, y2;

    Sankaku(){
        x0 = random(0, width);
        y0 = random(0, height);
        x1 = random(0, width);
        y1 = random(0, height);
        x2 = random(0, width);
        y2 = random(0, height);
    }

    void draw(){
        line(x0, y0, x1, y1);
        line(x1, y1, x2, y2);
        line(x2, y2, x0, y0);
    }
}
```

Sankakuクラスが持っている変数
(メンバ変数)

Sankakuクラスが持っている命令
(メンバ命令)

ただしSankaku()はコンストラクタと呼ばれる特殊な命令
(戻り値がなく、new時に呼ばれる)

classから始まる中括弧内が、そのクラスの定義

クラスには変数と命令を書くことが出来る。これらはクラスのメンバ変数、メンバ命令と呼ばれて、メンバ命令はメンバ変数を操る命令となる。

・クラスの使い方の概要

クラス名と同じ名前

宣言 Sankaku sankaku;

new sankaku = new Sankaku();

アクセス sankaku.draw();

sankakuというのがSankakuクラスのメンバ変数、メンバ命令を持った変数のようなもの(インスタンスという)になる

new文
インスタンスが実体を持つ
class定義内のSankaku()命令が呼び出される

メンバ変数やメンバ命令は
ドットを介してアクセスできる

クラスを使うには上の三段階が必要である。宣言は `int[] a;` とかと同じノリで、このように宣言された `sankaku` は `Sankaku` クラスのインスタンスと呼ばれる。これは配列のように `new` をしないと使うことが出来ない。`new` した後は、メンバ変数、メンバ命令にドットを使ってアクセスすることが出来る。

- ・クラスを使ってみよう

次のプログラム(Part1_3_2_tennis)はテニスゲームのプログラムである。

```

宣言 float x, y, vx, vy;
      float px, py;

void setup(){
  size(200, 300);
  x = random(0, width);
  y = random(0, 150);
  vx = random(2, 4);
  vy = random(2, 4);
}

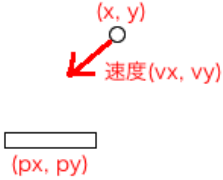
void draw(){
  background(255);

移動  x += vx;
      y += vy;
      px = mouseX;
      py = 280;

反射  if(x<0 && vx<0) vx = -vx; // 左端
      if(x>width && vx>0) vx = -vx; // 右端
      if(y<0 && vy<0) vy = -vy; // 上端
      if(y>py-10 && y<height && x>px-30 && x<px+30 && vy>0) vy = -vy; // バ-

描画  ellipse(x, y, 10, 10);
      rect(px-30, py, 60, 10);
}

```



反射の条件式が若干込み入っているが、他は特に分かりにくいところはないだろう。変数 px, py はバーの位置を表していて、py は常に 280 で固定、px はマウスに追従するようになっている。

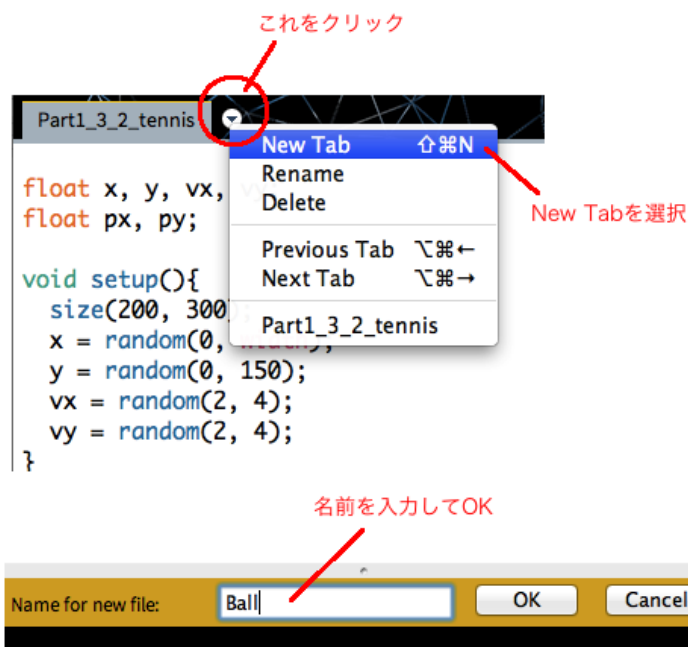
このプログラムをクラスを使って書きなおしていこう。

まず、上のプログラムで赤枠で囲ったところがボールに関わる部分である。これらをクラス Ball として分離する。

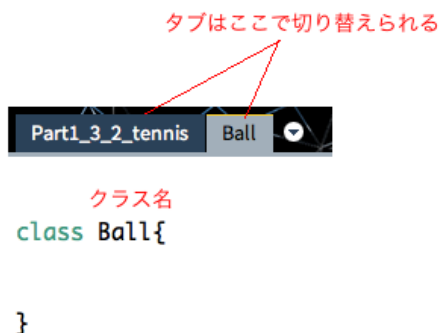
(1) タブを開く

新しいクラスを作るときは、同じプログラムに続けて書いても良いのだが、新しいタブを開いて別のプログラムに分けるのが良い。

次の図のように▽ボタンをクリックして、New Tab を選択し、下の方に出てくる欄に Ball と入れて OK を押そう。



(2) クラスの定義



Ball タブに移動したら、上のように `class Ball{ }` と書こう。この中括弧の内側が、Ball クラスの内容となる。

(3) メンバ変数

ボールに関する変数は、ボールの位置(x, y)と速度(vx, vy)であった。これらの変数を、次のように Ball クラスの定義の中に書こう。

```
class Ball{
    float x, y;
    float vx, vy;
}
```

(4) コンストラクタ

クラスのインスタンスは

```
b = new Ball();
```

のように、new することによって利用可能となる。この new がされた時に一度だけ実行される命令はコンストラクタと呼ばれ、主にクラスのメンバ変数の初期化に使われる。

ボールの変数の初期化に関わる文を、元のタブからコピーしてこよう。

```
class Ball{
    float x, y;
    float vx, vy;

    Ball(){
        x = random(0, width);
        y = random(0, 150);
        vx = random(2, 4);
        vy = random(2, 4);
    }
}
```

コンストラクタの命令定義であるが、
intとかvoidとか戻り値は書かない。
また、命令名は必ずクラス名と同じにする。
引数はあってもいい。

(5) 移動命令と描画命令

次はボールの移動と描画に関わる命令をクラス内に記述しよう。

元のタブにあるボールの移動と反射に関する行をまとめて、move という命令を作る。

```
void move(){
    x += vx;
    y += vy;

    if(x<0 && vx<0) vx = -vx; // 左端
    if(x>width && vx>0) vx = -vx; // 右端
    if(y<0 && vy<0) vy = -vy; // 上端
    if(y>py-10 && y<height && x>px-30 && x<px+30 && vy>0) vy = -vy; // バー
}
```

しかし、ここで問題なのは○で囲った部分にバーの位置を表す変数 px, py が使われているということである。クラスのメンバ命令はメンバ変数しか見ることが出来ないので、Ball クラス内に無い px, py を知ることが出来ないのである。

そこで、move 命令に引数を追加してやって、px, py は引数として呼び出し時に渡すことにする。そうすると Ball クラスは次のようになる。ついでに描画の命令 draw も追加した。

```
class Ball{
    float x, y;
    float vx, vy;

    Ball(){
        x = random(0, width);
        y = random(0, 150);
        vx = random(2, 4);
        vy = random(2, 4);
    }

    void move(float px, float py){
        x += vx;
        y += vy;
        if(x<0 && vx<0) vx = -vx;
        if(x>width && vx>0) vx = -vx;
        if(y<0 && vy<0) vy = -vy;
        if(y>py && y<height && x>px-30 && x<px+30 && vy>0) vy = -vy;
    }

    void draw(){
        ellipse(x, y, 10, 10);
    }
}
```

px, py は引数として渡す

(6) メインクラスの変更

これでとりあえず Ball クラスは完成した。次は元のタブ(メインクラスと呼ぼう)を編集する。

```
宣言 Ball b;
float px, py;

void setup(){
  size(200, 300);
  初期化 b = new Ball();
}

void draw(){
  background(255);

  px = mouseX;
  py = 280;

  移動・反射 b.move(px, py);
  描画 b.draw();

  rect(px-30, py, 60, 10);
}
```

Ball クラスに移行した部分は上のように変更される。

新たに作ったクラスは、int とか float のようなあたかも変数の型のようにして使うことが出来る。従って Ball b; と書くと、Ball 型の変数 b を宣言しているということになる。このような b は実は変数とは言わず、インスタンスという。

インスタンスと普通の変数との違いは、new しなければ使えないという点である。new を行くと、インスタンス b が、クラスで定義したメンバ変数 x, y, vx, vy やメンバ命令 move, draw などを「持つ」ことになる。これらのメンバ変数は、

b. x

b. vy

のようにドットを介して扱う。メンバ命令も

b. move(px, py)

b. draw()

のようにインスタンス名のあとにドットをつけて呼び出す。

将来的にはクラスの中にクラスが入っていたりして

a. b. c. hoge()

みたいなのも使うことになる。

・今度はプレイヤーのバーに関する部分を新しいクラス Player に移行する。今までの手順を真似して自分でやってみよう。

```

Ball b;
宣言 float px, py;

void setup(){
    size(200, 300);
    b = new Ball();
}

void draw(){
    background(255);

移動 px = mouseX;
    py = 280;

    b.move(px, py);
    b.draw();

    描画 rect(px-30, py, 60, 10);
}
    
```

今回、プレイヤーに関する変数 px, py は初期化をしていない。このような場合、Player クラスにはコンストラクタを書いても書かなくても良い。しかし、new をする必要はある。

・二つのクラス

同じクラスのインスタンスは複数作ることが出来る。右のプログラム(Part1_3_2_tennis4)は、Ball クラスの2つのインスタンス b1 と b2 を使っている。

このとき、b1 と b2 はそれぞれ独自のメンバ変数

b1. x, b1. y, b1. vx, b1. vy

b2. x, b2. y, b2. vx, b2. vy

を持つ。つまり、b1. x と b2. x は別の変数である。どちらかを変更してももう片方は変わらない。

従って、このプログラムを実行するとテニスゲームのボールが2つになる。

```

Ball b1, b2;
Player p;

void setup(){
    size(200, 300);
    b1 = new Ball();
    b2 = new Ball();
    p = new Player();
}

void draw(){
    background(255);

    p.px = mouseX;
    b1.move(p.px, p.py);
    b2.move(p.px, p.py);
    p.draw();
    b1.draw();
    b2.draw();
}
    
```


・ クラスの配列

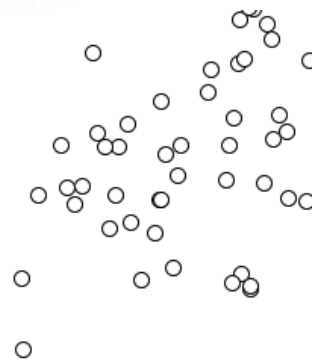
二つのインスタンスを作れるなら、もっとたくさんも可能である。配列を使うと非常に沢山のインスタンスをまとめて扱うことが出来る (Part1_3_2_tennis5)。

```
Ball[] bs; _____ インスタンスの配列を宣言
Player p;
```

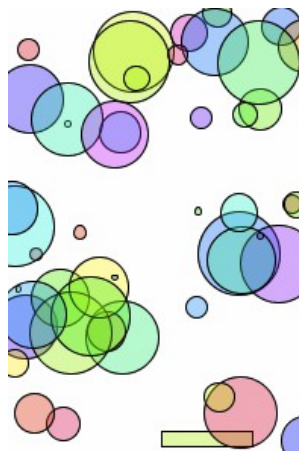
```
void setup(){
  size(200, 300);
  bs = new Ball[100]; _____ これは配列のnew
  for(int i=0; i<100; i++){
    bs[i] = new Ball(); _____ こっちがインスタンスのnew
  } _____ for文で全部をnewしてやる
  p = new Player();
}
```

```
void draw(){
  background(255);

  p.px = mouseX;
  for(int i=0; i<100; i++){
    bs[i].move(p.px, p.py); _____ 全てを移動
  }
  p.draw();
  for(int i=0; i<100; i++){
    bs[i].draw(); _____ 全てを描画
  }
}
```



ボールの色や大きさがランダムで決まるようにしてみよう。



4. テキストの取り扱い

ここでは文章（特に日本語）を Processing でどう扱うかを学ぶ。

※Processing のバグにより、Processing 画面に日本語を打つと表記がおかしくなる。日本語を含む行だけおかしくなるので、そのような行は出来る限り簡潔なものにしておいたほうが良い。

あまり良くない例

```
text(“あいうえお”, random(0, 100), random(0, 100));
```

このように String 型(文字列クラス)を使って分割すると良い

```
String str;  
str = “あいうえお”;  
text(str, random(0, 100), random(0, 100));
```

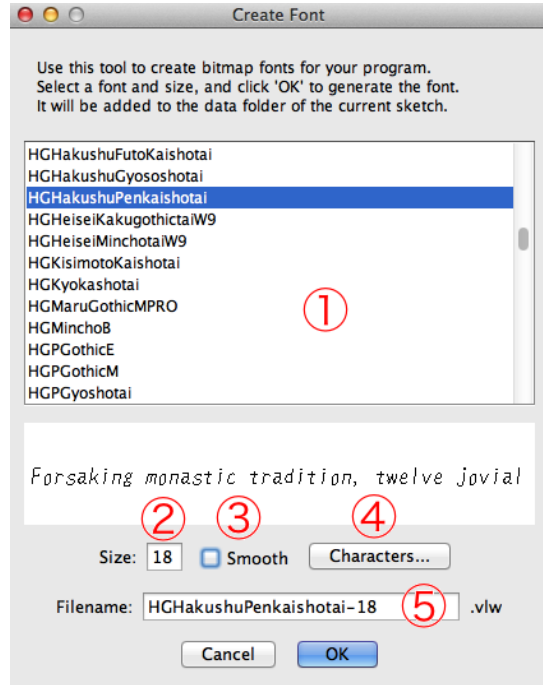
・フォントの作成

日本語を望みの書体（フォント）、大きさで表示するには一手間かかる。新しいプログラムを作り、上のメニューから Tools→Create Font を選ぶ。

右のような画面が現れるので

- ① フォントを選ぶ。非常に沢山あるが、HG で始まるのが日本の会社であるリコーが作ったものなので、それを選ぶと良い。
- ② Size を 20 程度の数にする（そうしないと結構重い）
- ③ Smooth をオフにする（これも軽くするため）
- ④ Characters ボタンを押して、出てきた画面で「All Characters」を選択する
- ⑤ この欄をコピーしておく

そして OK を押し、1分程度待つ。
すると選んだフォントが Processing で表示できるようになる。



・フォントの指定

プログラムを次のように書こう。二行目の HGKyoukashotai-18 というところは、自分が選んだフォントの、さっきコピーしたところ（⑤番の欄）.vlw を用いる。

```
size(500, 200);
PFont font = loadFont("HGKyoukashotai-18.vlw");

textFont(font); // text命令で書かれる文字のフォントを指定
textSize(18);   // 文字の大きさ
fill(0);        // 文字の色

String s = "あいうえお";
text(s, 100, 100);
```

あいうえお

「あいうえお」が選んだフォントで表示されただろうか？

このように、PFont クラスのインスタンス font に loadFont(“フォント名”)の結果を入れてやって、text 命令の前で textFont(font); を実行すれば text で書かれる文字列のフォントを指定できる。

また、textSize(大きさ); 命令で文字の大きさを指定できるが、さっきの②で指定した以上の大きさを使うとぼやけてしまう。これを回避するにはフォントの作成段階でもっと大きな Size で作らないといけないが、5 分とか待たされるのでどうしてもやりたいたいときだけやると良い。

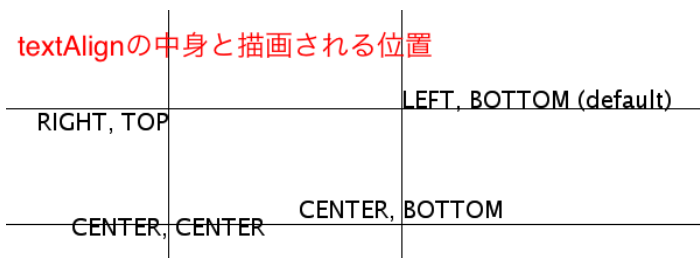
テキストに関する文法

```
PFont font = loadFont("フォント名");

textFont(フォント);
textSize(大きさ);
textAlign(指定子);
fill(色);

text(文字列, x, y);
```

text(s, x, y) 命令は、デフォルトでは (x, y) を左下の点とするような文字列を描画する。textAlign(CENTER, CENTER); 命令を書くと、次のように文字を書く基準点が変わる。



K 会冬期講習 2013 情報講座

・ ファイルからのロード

右のように、loadStrings 命令を使うと、ファイルから文字列を取得することが出来る
(Part1_4_3_loadstrings)。

この命令の戻り値は String 型の配列であり、配列の i 番目の要素は、ファイルの i 行目の文章に対応する。

逆に、String 型の配列をファイルに書き出す saveStrings(“ファイル名”, ss) という命令もある。

```
background(255);  
  
size(1000, 600);  
PFont font = loadFont("HGKyokashotai-18.vlw");  
  
textFont(font); // text命令で書かれる文字のフォントを指定  
textSize(18); // 文字の大きさ  
fill(0); // 文字の色  
  
String[] ss = loadStrings("../akebono.txt");  
for(int i=0; i<ss.length; i++){  
    text(ss[i], 100, 100+20*i);  
}
```

春はあけぼの。
やうやう白くなり行く、山ぎは少しあかりで、紫だちたる雲の細くたなびきたる。

夏は夜。
月のころはさらなり。
やみもなほ、ほたるの多く飛びちがひたる。
また、ただ一つ二つなど、ほのかにうち光りて行くもをかし。
雨など降るもをかし。

秋は夕暮。
夕日のさして山の端いと近うなりたるに、鳥の寝どころへ行くとして、
三つ四つ、二つ三つなど飛びいそぐさへあはれなり。
まいて雁などのつらねたるが、いと小さく見ゆるはいとをかし。
日入りはてて、風の音、虫の音など、はたいふべきにあらず。

冬はつとめて。
雪の降りたるはいふべきにもあらず。
霜のいと白きも、またさらでも、いと寒きに、
火などいそぎおこして、炭もてわたるもいとつきづきし。
昼になりて、ぬるくゆるびもて行けば、火桶の火も白き灰がちになりてわろし。

春はあけぼの。
やうやう白くなり行く、山ぎは少しあかりで、紫だちたる雲の細くたなびきたる。

夏は夜。
月のころはさらなり。
やみもなほ、ほたるの多く飛びちがひたる。
また、ただ一つ二つなど、ほのかにうち光りて行くもをかし。
雨など降るもをかし。

秋は夕暮。
夕日のさして山の端いと近うなりたるに、鳥の寝どころへ行くとして、
三つ四つ、二つ三つなど飛びいそぐさへあはれなり。
まいて雁などのつらねたるが、いと小さく見ゆるはいとをかし。
日入りはてて、風の音、虫の音など、はたいふべきにあらず。

冬はつとめて。
雪の降りたるはいふべきにもあらず。
霜のいと白きも、またさらでも、いと寒きに、
火などいそぎおこして、炭もてわたるもいとつきづきし。
昼になりて、ぬるくゆるびもて行けば、火桶の火も白き灰がちになりてわろし。

元のテキストファイル (akebono.txt)

Processing による表示

テキストファイルの入出力

```
String[] ss = loadStrings("ファイル名")  
saveStrings("ファイル名", ss)
```

入力
出力

どちらもssはString型の配列であり、
配列のi番目がi行目に対応する

5. キー入力

多くのゲームはキーボードで操作をする。ここではその方法を扱う。

(1) 基本

右のプログラム(Part1_5_1_keyinput)は、キーボードの a, d, w, s キーを使って○を移動させるプログラムである。なんで adws なのかというと、ちょうどキーボードの左の方にあって、a キーに左手の薬指を、s キーに中指を、d キーに人差し指を乗せると動かしやすいからである。

`keyPressed` 命令を作ると、ここはキーが押された時だけ実行されるようになる。押されたキーは `key` 変数に記録される。もし a キーが押されて `keyPressed` が呼ばれたなら、`key` は 'a' と等しくなるので `if(key=='a')` で判定ができる。

しかしこのプログラムの挙動を観察すると、いくつか問題がある。

- ・ 英語や数字キーはこれで良さそうだが、Shift とか矢印とかはどうするんだろう？
- ・ a と w を同時押ししても片方しか反応しない

これらの問題を解決しよう。

(2) 特殊キー

特殊キーが押された時は、`key` 変数は CODED という特殊な値になる。このとき、`keyCode` という名前の別の変数にその中身が記録されていて、UP なら上、DOWN なら下、同様に RIGHT, LEFT, SHIFT, ENTER, ESC, CONTROL などもある。

これを真似して、矢印キーでも円を移動できるようにしてみよう。

```
float x, y;

void setup(){
  size(300, 300);
  x = width/2;
  y = height/2;
}

void draw(){
  background(255);
  ellipse(x, y, 10, 10);
}

void keyPressed(){
  if(key=='a'){
    x -= 5;
  }else if(key=='d'){
    x += 5;
  }else if(key=='w'){
    y -= 5;
  }else if(key=='s'){
    y += 5;
  }
}
```

```
if(key==CODED){
  if(keyCode==UP){
    y -= 5;
  }else if(keyCode==DOWN){
    y += 5;
  }
}
```

(3) 同時押し

同時押しの判定は実はそこそこ複雑である。

入力を読み込みたい各キーごとに、それが押されているかどうかを表す変数を用意して、そのキーが押されたらそれを1に、離されたらそれを0にすることで対応する。keyPressed 命令はキーが押された時に呼び出されるが、同じようにキーが離された時に呼び出される `keyReleased` 命令があるので、これを用いる。

新しいタブを開いて、KeyState クラスを作る。このメンバ変数として、どのキーが押されているかを記録することにする。次のプログラム(Part1_5_3_keyinput)では、上と左矢印キーに関して実装されている。

メインクラス

```
float x, y;
KeyState ks;
    KeyStateクラスのインスタンスksを宣言
void setup(){
    ks = new KeyState();    new
    size(300, 300);
    x = width/2;
    y = height/2;
}
    ksのメンバ変数を見ることで
    今のキーの状態がわかる
void draw(){
    background(255);

    if(ks.left==1) x-=5;
    if(ks.up==1) y-=5;

    ellipse(x, y, 10, 10);
}
    キーが押された時
void keyPressed(){
    ks.pressed();
}
    キーが離された時
void keyReleased(){
    ks.released();
}
```

KeyStateクラス

```
class KeyState{
    int up;
    int left;

    KeyState(){
        up = 0;
        left = 0;
    }
    コンストラクタ
    最初はキーが離された状態(0)に

    void pressed(){
        if(key==CODED){
            if(keyCode==UP) up = 1;
            else if(keyCode==LEFT) left = 1;
        }
    }

    void released(){
        if(key==CODED){
            if(keyCode==UP) up = 0;
            else if(keyCode==LEFT) left = 0;
        }
    }
}

key, keyCodeはどのクラスからも見られる特別な変数である。
width, height, mouseX, mouseYとかもそう。
```

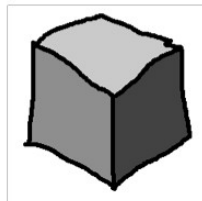
このプログラムの意味を理解したら、実際に実行してみて、上と左の同時押しで○が斜めに進むかを確認しよう。さらに、右と下矢印についても動くようにしよう。

6. 画像ファイルの取り扱い

Processing では `rect` とか `ellipse` とかの命令で基本的な図形を描くことができるが、本格的な絵を扱うには、別に画像ファイルを用意しておいて、それを読み込んで表示するという方法を使う。

・画像の作成

画像編集ソフトを使って適当な絵を描こう。tekitou.png という名前で保存する。



適当な画像
(tekitou.png)

・画像のロードと表示

次のプログラム (Part1_6_1_image) は、この画像を Processing で読み込み、表示するものである。

```

PImage img;

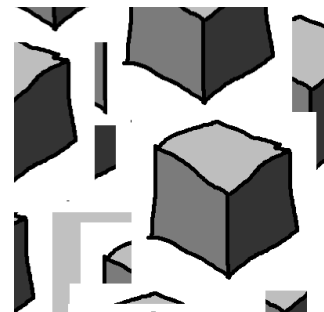
void setup(){
  size(400, 400);

  img = loadImage("tekitou.png");
}
// 画像の読み込み
// imgという名前のPImageクラスのインスタンスに格納される

void draw(){
  image(img, 30, 30);
}
// 画像の左上の角の場所
// 表示する画像のインスタンス
    
```

`PImage` という、予め用意されたクラスを用いる。そのインスタンス `img` に画像を読み込み、`image` 命令で表示する。上のプログラムでは `img` を `new` していないが、`loadImage` 命令の中に `new` が隠されているので OK。

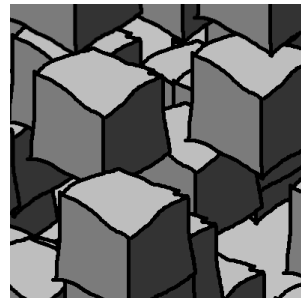
右のようにたくさん重ねて表示してみよう。



・透明色

たくさん重ねて表示すると、画像の背景の白い部分が上に重ねられてしまう。

画像編集ソフトで画像の背景を透明にして、右のようにちゃんと重なるようにしよう。



・ imageMode

imageMode 命令を使うことで、image(img, x, y)と書いた時に、x, y を左上の角とするか、画像の中心とするかを変えることが出来る。

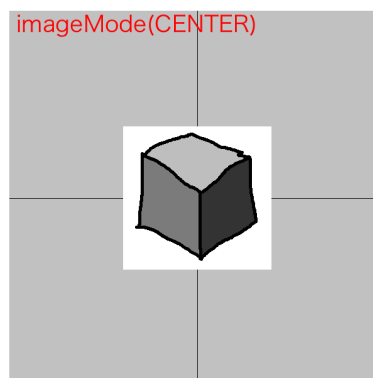
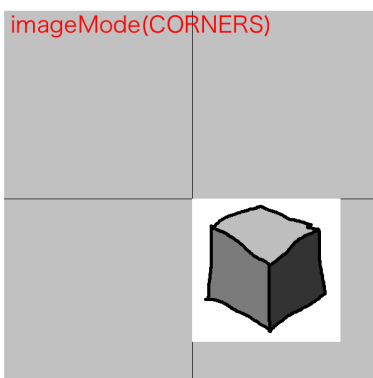
```
void draw(){
  line(width/2, 0, width/2, height);
  line(0, height/2, width, height/2);

  imageMode(CENTER);
  image(img, width/2, height/2);
}
```

画像の読み込みと表示

```
PImage img;
img = loadImage("画像名");
image(img, x, y);

imageMode(指定子);
指定子: CENTER, CORNERS
         中心   左上の角
```

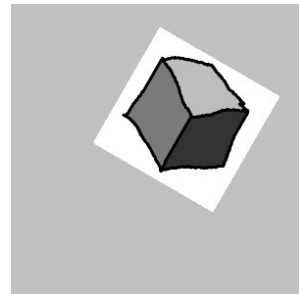


- ・ 回転、拡大、移動

画像を回転したり、拡大したりして表示したいこともある。そういうときは、次のようなプログラムを書くといい(Part1_6_4_image)。

```
void draw(){
    displayImage(img, 180, 120, PI/6, 0.6);
}

// (x, y)を中心とし、角度th回転し、s倍に相似拡大した画像imgを表示する
void displayImage(PImage img, float x, float y, float th, float s){
    pushMatrix();
    translate(x, y);
    rotate(th);
    scale(s);
    imageMode(CENTER);
    image(img, 0, 0);
    popMatrix();
}
```



ここで作っている displayImage 命令は、たくさんの新しい命令を含んでいるが、

translate(x, y): これ以降に描かれる画像を (x, y) 平行移動する

rotate(th): これ以降に描かれる画像を th ラジアン回転する

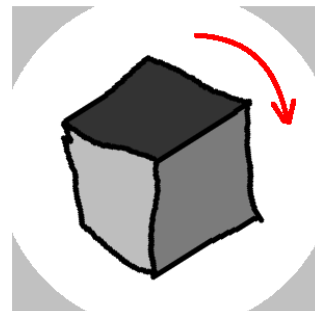
scale(s): これ以降に描かれる画像を s 倍拡大する

pushMatrix(), popMatrix(): この二つの間に書かれた translate, rotate, scale はこれより外側に効力を発揮しなくなる (これを書かないと、translate とかは、これ以降に書いた全ての画像を平行移動してしまう)。

という意味である。全体で、(0, 0) を中心にある画像を s 倍し、th ラジアン回転し、x, y 平行移動した画像を描くという意味になる。

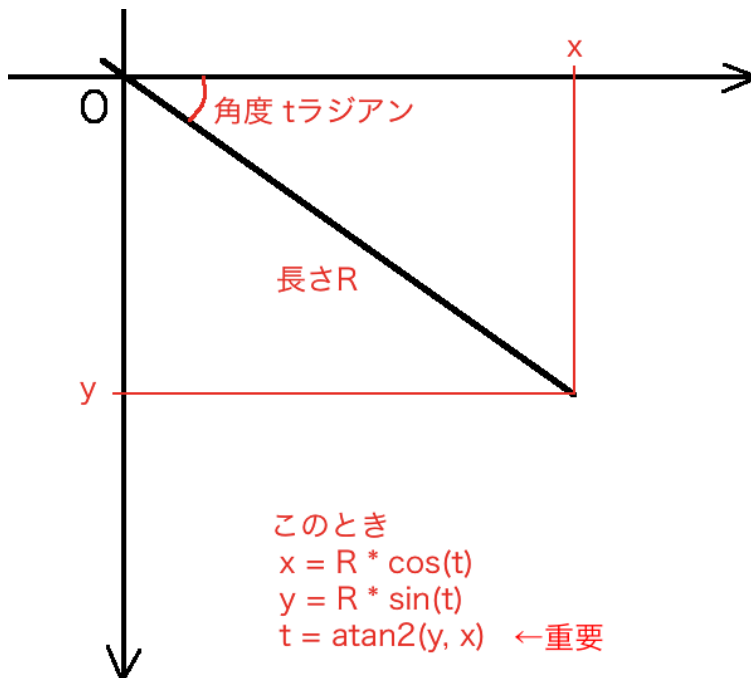
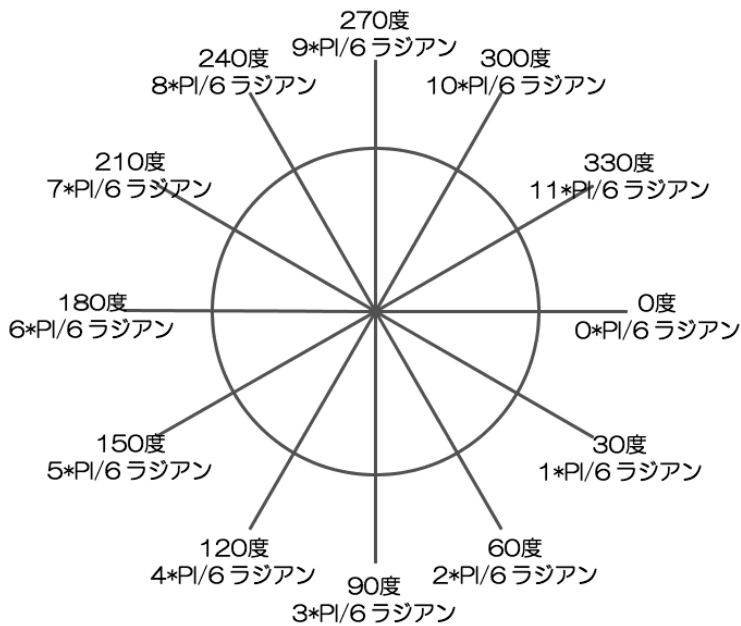
この displayImage 命令を利用して、画面の中心で描いた画像が回転するようにせよ。

以後、必要があればこの displayImage 命令をコピーして使うといいと思う。



補足：ラジアン、三角関数

※Processing の世界では y 軸が下方方向を向いているので数学とは上下逆になる。



7. ピクセル単位の編集

Processing では読み込んだ画像ファイルをピクセル単位で編集することが出来る。すなわち、この画像のこの場所の色は何ですか？とかこの場所を赤にし、とかができる。

・使い方

次のプログラム(Part1_7_1_pixels)は、画像を読み込んで、マウスをクリックした時、その場所のピクセルの色を取得し、明るさを出力欄に出力し、ピクセルをランダムな色にする。

```
PImage img;

void setup(){
  size(300, 300);
  img = loadImage("tekitou.png");
}

void draw(){
  imageMode(CORNERS);
  image(img, 0, 0);
}

void mousePressed(){
  img.loadPixels();

  // 画像imgの(mouseX, mouseY)の位置の色を取得
  color c = img.pixels[mouseY * img.width + mouseX];

  // その色の明るさ(brightness)を出力
  println(brightness(c));

  // その場所の色をランダムに変更
  img.pixels[mouseY * img.width + mouseX]
    = color(random(0, 255), random(0, 255), random(0, 255));

  // 変更を反映させる
  img.updatePixels();
}
```

まず、画像をクリックした時の反応を確認せよ。（よく見ると色がクリックした1点だけランダムに変わるのが分かる）

次に、画像の右端の外側をクリックして、その場所の点は変わらずに、画像上の違う場所の色が変わることを確認しよう。これは img.pixels という配列があくまで画像の内側の点しか記録していないことによる。

・解説

img.widthとかimg.loadPixels()とかを見て分かるように、PImageというのはクラスで、imgはそのインスタンスである。PImageクラスは次のような機能を持つ。

PImageクラスの使い方

PImage img;	宣言
img = loadImage("画像名");	画像ファイルの読み込み
img = createImage(w, h, ARGB);	空の画像データを作る(横w, 縦h)
img.width	横幅
img.height	縦幅
img.loadPixels();	ピクセル編集を可能にする
img.pixels[y*img.width + x]	画像の(x, y)の位置のピクセルの色
img.updatePixels();	ピクセル編集結果を適用する

今回のポイントはimg.pixels[]という配列で、ここにはimg.loadPixels()命令を実行することで、画像の全てのピクセルの色情報が一次元配列として記録される。

一次元配列であるが、画像は二次元なので、画像上の点x, yは

img.pixels[y * img.width + x]

という場所に格納される。つまり、画像上の各点はpixels配列の次のような値番目に格納される。

0	1	2	...	width-1
width+0	width+1	width+2	...	2*width-1
2*width+0	2*width+1	2*width+2	...	3*width-1
...				
(height-1)*width+0			...	height*width-1

また、colorという記述が何度か出てきたが、これは色を表す型で、img.pixels[]はcolor型の配列である。color型の変数からは、次のような命令を使ってその赤成分とか明るさとかを取り出すことが出来る。また、color(r, g, b)という命令で、色(r, g, b)を持ったcolor型の値を作り出せる。

色

color cl = img.pixels[y * img.width + x];	変数clに画像imgの(x, y)の位置の色を代入
red(cl), green(cl), blue(cl)	clの赤成分、緑成分、青成分
hue(cl), saturation(cl), brightness(cl)	clの色調、彩度、明るさ
img.pixels[y * img.width + x] = color(r, g, b);	画像imgの(x, y)の位置の色を(r, g, b)にする

- ・ pixels の使い道その 1 : マップとして使う

例えばゲームで主人公が陸の上だけ移動できるという状態を考える。次のプログラム (Part1_7_2_map) は、マウスが陸の部分 (色の緑成分が 100 以上) にいるときは○の位置 x , y を変更し、そうでないときはそのままにすることで、○の移動を陸の上に制限するものである。

```
PImage img;
float x, y;

void setup(){
  size(300, 300);
  img = loadImage("land.png");

  x = width/2;
  y = height/2;
}

void draw(){
  imageMode(CORNERS);
  image(img, 0, 0);

  img.loadPixels();
  if(green(img.pixels[mouseY * img.width + mouseX]) > 100){
    x = mouseX;
    y = mouseY;
  }
  ellipse(x, y, 10, 10);
}
```



次のように、リアルな画像と、その移動可能範囲で色分けしたマスク画像の二枚を用意すると、表示用には元の画像、移動判定用にはマスク画像を使うことで、リアルな画像を使ったアクションゲームとかも作れる。

画像編集ソフトでリアル画像とマスク画像の 2 枚を用意し、この仕組みを実装してみよ。



マスク画像

・pixels の使い道その 2：アニメーションの読み込み

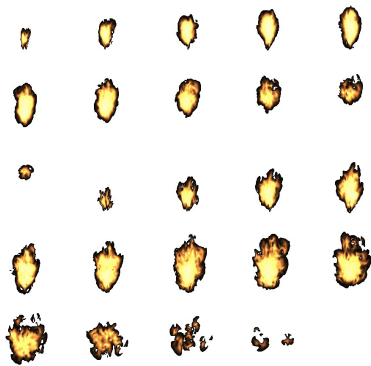
ゲームにおけるアニメーション（キャラの歩行とか、爆発とか）を実装するために、別の画像ファイルにアニメーションのコマを記録しておいて、それを連続で再生するというのを考える。

たとえば次のサイトには RPG で使えるような素材が置いてある（商用利用可能なコンテンツを探すのが大変だった…）。

<http://segment.nobody.jp/mat/index.html>

このなかに、次のような炎のアニメーション

(http://spiralwind.tuzikaze.com/main/material/animation/animation_fire001.html) があったので、これを使ってみよう。これは 1 枚の画像 (960*960) に 25 枚の小さな画像 (192*192) が並んでいて、左上から始まる一連の 11 枚、その後の 14 枚



```
PImage img_at(PImage img, int x, int y){
    int xx = 192 * x;
    int yy = 192 * y;
    PImage ret = createImage(192, 192, ARGB);
    img.loadPixels();
    ret.loadPixels();
    for(int i=0; i<192; i++){
        for(int j=0; j<192; j++){
            int id = (yy+i)*960 + (xx+j);
            ret.pixels[i*192+j] = img.pixels[id];
        }
    }
    ret.updatePixels();
    return ret;
}
```

がそれぞれ独立した火の玉が燃えるアニメーションになっているようだ。

このプログラム (Part1_7_4_anim) は、大きな画像 img の中の、左から x 番目、上から y 番目の小画像を取り出し、戻り値として返す命令 `img_at(PImage img, int x, int y)` である。

中でやっていることは、まず戻り値となる画像 ret を用意して、大画像のなかの、指定された位置の小画像に対応する全ピクセルを ret のピクセルにコピーし、それを戻り値として返すということである。変数 xx, yy は大画像の中の、指定された小画像の左上の角の位置、二重ループで使っている変数 i, j が小画像のなかのピクセルの位置、二重ループの内側にある変数 id が小画像中の位置 (i, j) に対応する、大画像での位置の `img.pixels` 配列上での番地を表す。

この命令 `img_at` を利用して、炎のアニメーションを再生するプログラムを書け。

8. XML ファイルの取り扱い

RPG を作るとしたら、アイテムの一覧とか魔法の一覧とかのデータが必要となる。このようなデータを記述し、プログラムに読み込んだり、あるいはセーブデータを作ったりするのに XML ファイルは適している。

・XML ファイルとは

XML というのは、次のような形式をした文章のことである。

```
<cities>
  <city>
    <name>東京</name>
    <place hokui="35.680909" toukei="139.767372" />
  </city>
  <city>
    <name>京都</name>
    <place hokui="35.009129" toukei="135.754807" />
  </city>
  <city>
    <name>青森</name>
    <place hokui="40.824589" toukei="140.755203" />
  </city>
  <city>
    <name>名古屋</name>
    <place hokui="35.154919" toukei="136.920593" />
  </city>
  <city>
    <name>大阪</name>
    <place hokui="34.702509" toukei="135.496505" />
  </city>
  <city>
    <name>福岡</name>
    <place hokui="33.579788" toukei="130.402405" />
  </city>
</cities>
```

XML は XML 要素の集まりに分解できる。

XML要素

<要素名 属性名="属性" 属性名2="属性2" ... >

内容

</要素名>

内容の部分には、さらにXML要素を含めることが出来る

<要素名 属性名="属性" ... /> 内容がない時 ">"の手前にスラッシュ

上の XML では、一番外側に cities という要素名の XML 要素があり、これは内容として 6 つの city 要素を持っている。一番上の city 要素は name 要素と place 要素を持っていて、その name 要素は内容として “東京” という文字列を持っている。place 要素は内容は持たず、二つの属性 hokui, toukei を持っている。

ちなみに要素名、属性名などは全て自由につけることが出来る。

・ Processing での XML の扱い

Processing では、XML クラスを使って XML 要素をロードすることが出来る。

XMLの読み込み		
戻り値	記法	説明
XML	XML xml; xml = loadXML("ファイル名");	XMLクラスの宣言 ファイルからxml要素を読み込む
XML[] XML	xml.getChildren("要素名") xml.getChild("要素名")	xml要素が内容として持つ、 指定した要素名の子xml要素を得る 複数の時(インスタンスの配列が返される) 単数の時(XMLクラスのインスタンスが返される)
String	xml.getName()	XML要素の要素名を得る
String	xml.getContent()	XML要素の内容を文字列として得る
String int float	xml.getString("属性名") xml.getInt("属性名") xml.getFloat("属性名")	XML要素の、指定した属性名に対する属性値を得る 文字列 整数 小数

プログラム例 (Part1_8_1_xml)

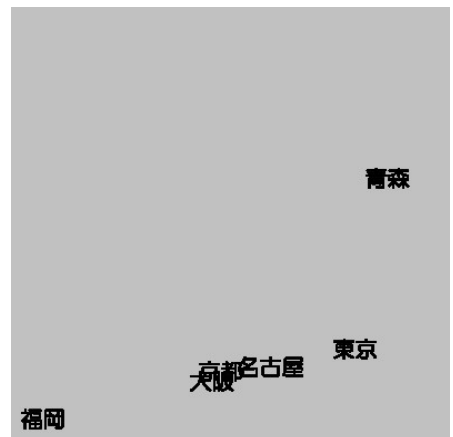
```
XML xml;
XML[] children;
PFont font;
```

```
void setup(){
  size(400, 400);
  xml = loadXML("../cities.xml");
  font = loadFont("HGMaruGothicMPRO-20.vlw");
  textFont(font);
}
```

```
void draw(){
  String name;
  float x, y;
```

```
  children = xml.getChildren("city");
  for(int i=0; i<children.length; i++){
    name = children[i].getChild("name").getContent();
    x = children[i].getChild("place").getFloat("toukei");
    y = children[i].getChild("place").getFloat("hokui");

    fill(0);
    text(name, 30*(x-130), height-30*(y-33));
  }
}
```



・プログラム解説

このプログラムは、右のXML ファイルを読み込み、ここに記載された各都市を、place 要素の属性として書かれた toukei, hokui の場所を書くプログラムである。

loadXML 命令を実行すると、XML ファイル全体のXML 要素が戻り値として得られる。右のファイルなら、cities 要素である。

メンバ命令 getChildren(“city”)によって、cities 要素が持つ city 要素を全て配列 children に格納する。これによって、XML のインスタンスの配列の 0 番目(children[0])は

```
<cities>
  <city>
    <name>東京</name>
    <place hokui="35.680909" toukei="139.767372" />
  </city>
  <city>
    <name>京都</name>
    <place hokui="35.009129" toukei="135.754807" />
  </city>
  <city>
    <name>青森</name>
    <place hokui="40.824589" toukei="140.755203" />
  </city>
  <city>
    <name>名古屋</name>
    <place hokui="35.154919" toukei="136.920593" />
  </city>
  <city>
    <name>大阪</name>
    <place hokui="34.702509" toukei="135.496505" />
  </city>
  <city>
    <name>福岡</name>
    <place hokui="33.579788" toukei="130.402405" />
  </city>
</cities>
```

```
<city>
  <name>東京</name>
  <place hokui="35.680909" toukei="139.767372" />
</city>
```

という子要素となる。

この要素に対して getChild(“name”)を実行すると

```
<name>東京</name>
```

という子要素がさらに得られる(今度は単数なので配列ではなく、そのままXML クラスのインスタンスが得られる)。

この子要素に対して getContent()を実行すると、この要素の内容である「東京」という文字列が得られることとなる。

プログラムでは、この二つをまとめて

```
children[i]. getChild(“name”). getContent()
```

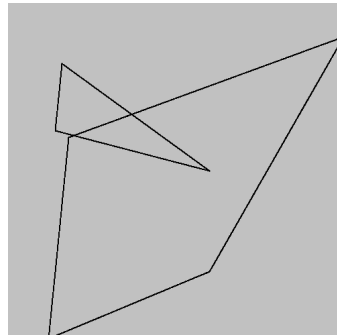
と書いていた。

北緯、東経は属性として書かれているので、getFloat(“toukei”)とgetFloat(“hokui”)命令で取得する。float なので小数が得られる。

・ 課題

次のような XML ファイルを読み込んで、右のように作図をするプログラムを書け (point で指定された点を直線で結ぶ)。getChildren を二段階行うことになる。

```
<figure>
  <zukei>
    <point x="300" y="400" />
    <point x="500" y="50" />
    <point x="90" y="200" />
    <point x="60" y="500" />
  </zukei>
  <zukei>
    <point x="300" y="250" />
    <point x="80" y="90" />
    <point x="70" y="190" />
  </zukei>
</figure>
```



・ XML ファイルの保存

次のプログラム (Part1_8_3_xmlsave) は、マウスをクリックすると、クリックした位置を順に xml に記録していき、なにかキーを押したところでそれを xml ファイルに出力するものである。

```
XML xml;

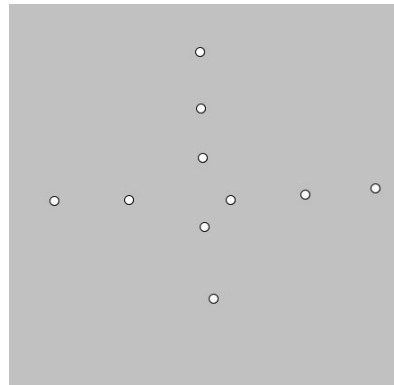
void setup(){
  size(500, 500);
  xml = new XML("points");
}

void draw(){
}

void mousePressed(){
  XML ch = new XML("point");
  ch.setInt("x", mouseX);
  ch.setInt("y", mouseY);
  xml.addChild(ch);

  ellipse(mouseX, mouseY, 10, 10);
}

void keyPressed(){
  saveXML(xml, "test.xml");
}
```



```
<?xml version="1.0" encoding="UTF-8"?>
<points>
  <point x="81" y="247"/>
  <point x="164" y="246"/>
  <point x="277" y="246"/>
  <point x="360" y="240"/>
  <point x="438" y="233"/>
  <point x="243" y="81"/>
  <point x="244" y="144"/>
  <point x="246" y="199"/>
  <point x="248" y="276"/>
  <point x="258" y="356"/>
</points>
```

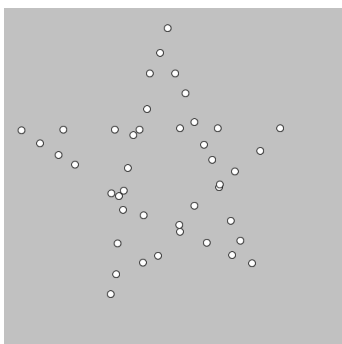
このプログラムは、次のような命令を利用している。

XMLのセーブに関する命令

<code>XML a = new XML("要素名");</code>	指定した要素名のXML要素を作る
<code>a.setInt("属性名", 属性値)</code> <code>a.setString("属性名", 属性値)</code> <code>a.setFloat("属性名", 属性値)</code>	XML要素に属性を追加する
<code>a.setContent("内容")</code>	XML要素に内容を追加する
<code>a.addChild(他のXML要素)</code>	XML要素の内容に、他のXML要素を追加する (XMLファイルでは内側に来る)
<code>saveXML(a, "ファイル名")</code>	XML要素をファイルに出力

このようなXML出力を使うと、プログラムの状態を比較的簡単にXMLファイルとして記録することが出来る。そしてそのようなXMLファイルは、先ほどのロードに関する命令達を使って読み込むことが可能である。これはゲームのセーブ&ロードに使える。

このプログラムを改良して、マウスの位置だけでなく、クリックした時間（プログラム開始からのステップ数）も記録するようにせよ。そして、これが出力したXMLファイルを読み込んで、自分がクリックした通りの位置、時間でその位置に円を書いていくプログラムを書け。



記録



```
<?xml version="1.0" encoding="UTF-8"?>
<points>
  <point step="60" x="86" y="177"/>
  <point step="74" x="161" y="177"/>
  <point step="80" x="197" y="177"/>
  <point step="100" x="256" y="175"/>
  <point step="115" x="311" y="175"/>
  <point step="137" x="402" y="175"/>
  <point step="184" x="373" y="208"/>
  <point step="198" x="336" y="238"/>
  <point step="209" x="313" y="261"/>
  <point step="223" x="277" y="288"/>
  <point step="235" x="255" y="316"/>
  <point step="283" x="224" y="361"/>
  <point step="298" x="202" y="371"/>
  <point step="318" x="155" y="417"/>
  <point step="344" x="163" y="388"/>
  <point step="359" x="165" y="343"/>
  <point step="391" x="173" y="294"/>
  <point step="406" x="174" y="266"/>
  <point step="425" x="180" y="233"/>
  <point step="444" x="188" y="185"/>
  <point step="472" x="208" y="147"/>
  <point step="491" x="212" y="99"/>
  <point step="506" x="227" y="65"/>
  <point step="522" x="238" y="29"/>
  <point step="542" x="249" y="99"/>
  <point step="557" x="264" y="124"/>
  <point step="572" x="277" y="166"/>
  <point step="586" x="291" y="199"/>
  <point step="619" x="303" y="221"/>
  <point step="634" x="314" y="257"/>
  <point step="649" x="330" y="310"/>
  <point step="664" x="344" y="339"/>
  <point step="682" x="361" y="372"/>
  <point step="712" x="332" y="360"/>
  <point step="728" x="295" y="342"/>
  <point step="743" x="256" y="326"/>
  <point step="757" x="203" y="302"/>
</points>
```

再生

