情報メディア基盤ユニット用資料（2014 年 4 月 15 日分）

Processing 言語による情報メディア入門

プログラムを使って絵を描く

神奈川工科大学情報メディア学科　佐藤尚

Processing とは？

Processing とは、アメリカのマサチューセッツ工科大学の BenFry さんと CaseyReas さんによって作られた視覚デザインのためのプログラミング言語と開発環境のことです。Processing の公式サイトは http://www.processing.org です。ここから Processing のプログラムなどをダウンロード出来ます。情報メディア基盤ユニットでは、Processing 言語を利用して、情報メディア系でのプログラミングに必要とされる基本的な考え方を修得することを目指します。

Processing は以下のような特徴を持っています。

- C 言語や Java 言語を利用するよりも簡単にインタラクティブかつビジュアルなプログラムを作成することができる。
- OpenGL などの機能も利用できるので、3 次元表現を伴うようなプログラムを作成できる。
- Java の機能を利用して機能を拡張することができる。
- Windows, MacOSX, Linux で実行できる。
- Android 用のプログラムも作ることが出来る。iPhone でも、Processing 言語のプログラムを作ることが出来る。

Examples の中に色々なサンプルプログラムが入っているので、実行してみると Processing でどんなことが出来るかがわかります。

Processing を使ってみる

プログラム作成するためのテキストエディタエリア、ツールバー、コンソールエリア、メッセージエリアから出来ています。実行ボタン (Run) を押すと、プログラムが実行されます。

Run ボタン : プログラムを実行する際に利用します。

Stop ボタン : プログラムを停止させる際に利します。

New ボタン : 新しいファイル（スケッチ）を作成する。Processing では、プログラムを書いたファイルなどをまとめてスケッチ (sketch) と呼んでいます。

Open ボタン : 既存のスケッチを読み込む。 このボタンをクリックすると、別なウィンドウが開き、そこから保存されているスケッチを読み込みます。

Save ボタン : 表示されているスケッチに名前をつけて保存する際に利用します。

Export ボタン : 表示されているスケッチ を Java アプレットとして保存します。その 際には、Java アプレットを表示するため に最低限必要な HTML ファイルも作成さ れます。

Processing を用いて作られるプログラムは、スケッチ (sketch)

テキストエディタ (Text Editor) とは？

基本的に文字情報のみからなるファイルを作成するために利用するソフトウェアのこと。

Processing の起動画面

と呼ばれています。保存をすると、ドキュメントフォルダの中の Processing というフォルダ内に新しくフォルダを作り、その中にスケッチを構成するプログラムやデータを保存します。

Processing プログラムの基本形その 1

プログラミングの基本にあるのは命令文です。これは、私たちの使っている言語に対応させれば、文に相当するものです。命令文は処理内容を表現したものです。普通の文の終わりに句読点を置くのと同じように、命令文の終わりには ;(セミコロン) を置きます。普通の文にも色々な文が存在するように、Processing の命令文にも様々な種類のものが存在しています。興味のある人は、http://www.processing.org/reference/index.html を見ると、どのような命令文があるのかがわかります。

Processing のプログラムでは、大文字と小文字を区別します。例えば、Size と size は異なったものとして扱われます。命令文と命令文の間の半角スペースは無視されます。ただし、全角のスペースを使うと、エラーとなるので気をつけてください。また、半角の " と全角の " も別なものとして扱われますので、気をつけてください。簡単に言うと、「全角文字を使うときには気をつけましょう！！」です。

Processing を起動して、テキストエディタエリアに以下の命令文を打ち込んで下さい。

最初のプログラム 1-1

```
ellipse(50,40,80,70);
```

この命令文の意味は、「(50,40) を中心に、横方向 80、縦方向 70 の楕円を描け」です。この命令文を入力し終わったら、"Run" ボタンをクリックして下さい。

この次はもう少し長い例です。同じようにエディタに入力し、入力が終わったら、"Run" ボタンをクリックして下さい。

2 番目のプログラム 1-2

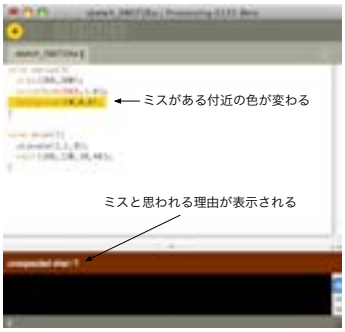
```
size(400,400);
ellipse(200,200,80,80);
ellipse(50,50,50,50);
ellipse(300,350,80,80);
```

Processing プログラムの基本形その 2

New ボタンを押して、新しいスケッチを作り、テキストエディタエリアに以下の命令を打ち込んで下さい。打ち込み終わったら、"Run" ボタをクリックして下さい。

命令文：プログラミング言語の種類によっては別な言い方をするかもしれません。

英語は苦手という人は、少し古いバージョンの物ですが、http://www.technotype.net/processing/reference/index.html に日本語訳があります。



エラー発生時の画面

基本形その 2 のプログラム 1-3

```
void setup(){
  size(640,480);
  smooth();
}

void draw(){
  if(mousePressed){
    fill(0);
  }else{
    fill(255);
  }
  ellipse(mouseX,mouseY,80,80);
}
```

適当な場所に空白や空行を入れることで読みやすいプログラムを作ることが出来ます。特に、行の開始位置を下げることで、命令文の塊を明らかにすることをインデント（indent）または字下げと呼びます。

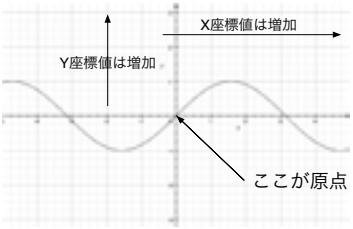
Processing のプログラムは、単純に命令文を一列に並べたものと、いくつかの塊に構造化して並べたものの 2 種類に分けることが出来ます。後者の場合には、基本的に setup と draw という 2 つの塊から成り立っています。setup には、最初だけ実行する命令文を書き、draw にはそうでない部分（プログラムの本体とでも言うべき部分）を書きます。setup の部分は実行開始時に 1 回だけ実行されますが、draw の部分は定期的に呼び出されて、何度も実行されます。Processing 言語では、この塊のことを関数と呼んでいます。少し複雑なプログラムになると、setup と draw 以外の塊（関数）を利用します。

正確には、Processing 言語では、setup や draw などを関数と呼びます。

今日の授業では、基本形 1 のような単純に命令文が一列並んだタイプのプログラムを作って行きます。

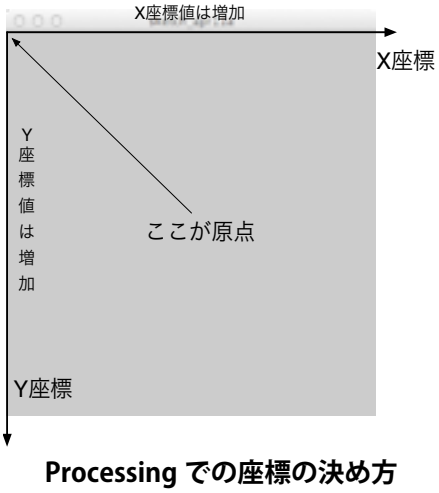
図形の描画

Processing 言語のプログラムを作る上で、おそらく最もよく使われる命令（関数）は size です。これは、横 x 画素、縦 y 画素の大きさのウィンドウを表示する命令です。



数学での座標の決め方

Processing 言語で図形を描く場合には、座標を利用して位置の指定を行います。つまり、X 座標値と Y 座標値があれば、平面上の点の位置を決めることか出来ます。そこて、2 つの値を利用して点の位置を決めます。数学では左下に原点を置きますが、Processing 言語では基本的に左上か原点となります。このため、X 軸方向は、左から右に移動するにつれて、座標値は大きくなりますが、Y 軸法では、上から下に移動するにつれて、



Processing での座標の決め方

座標値が大きくなります。Processing 言語での座標の決め方に気をつけてください。

ウィンドウの表示

命令名（関数名）	意味
size(x,y);	横 x 画素、縦 y 画素の大きさのウィンドウを表示する。

基本的な図形の描画に関連する命令（関数）

命令名（関数名）	意味
line(x1,y1,x2,y2);	点 (x1,y1) と点 (x2,y2) の間に線分を描く。
ellipse(x,y,w,h);	基本的には、(x,y) を中心として、幅 w、高さ h の楕円を描く。
triangle(x1,y1,x2,y2,x3,y3);	3 点 (x1,y1)、(x2,y2)、(x3,y3) を頂点とする三角形を描く。
rect(x,y,w,h);	基本的には、(x,y) を左上の頂点とする幅 w、高さ h の長方形を描く。
quad(x1,y1,x2,y2,x3,y3,x4,y4);	4 点 (x1,y1)、(x2,y2)、(x3,y3)、(x4,y4) を頂点とする四角形を描く。
arc(x,y,w,h,s0,s1);	基本的には、(x,y) を中心として、幅 w、高さ h で角度 s0 から角度 s1 までの半円を描く。
point(x,y);	位置 (x,y) に点を描く。
radians(theta);	度で表された角度を弧度法（ラジアン）に変換する。

複数の点を描くプログラム 1-4

```
size(400,400);

point(100,200);
point(100,100);
point(399,399);
```

線の描画するプログラム 1-5

両端の点の位置を指定すると線分が決まることを思い出して下さい。

```
size(480,120);
line(20,10,460,110);
```

円の描画プログラム 1-6

英語では、楕円のことを ellipse と言います。円は楕円の特別な場合なので、楕円を描くことが出来れば、円も描くことが出来ます。

```
size(400,200);
ellipse(280,-100,400,400);
ellipse(120,100,110,110);
ellipse(360,100,18,18);
ellipse(250,180,200,60);
```

長方形の描画プログラム 1-7

長方形のことを矩形と呼ぶことあります。英語では、rectangle と言います。
座標軸に平行な辺を持つ長方形は左隅の頂点の位置と幅と高さを指定すれば決まることを思い出して下さい。

```
size(480,120);
rect(20,10,450,100);
```

三角形と四角形の描画プログラム 1-8

```
size(400,400);
triangle(250,30,380,100,300,300);
triangle(140,30,220,380,110,350);
quad(100,100,200,80,240,300,150,200);
```

円弧の描画（その 1）プログラム 1-9

```
size(400,400);
arc(200,200,300,300,radians(30),radians(330));
```

Processing 言語では角度の大きさの指定には弧度法（ラジアン、radian）を利用します。

円弧の描画（その 2）プログラム 1-10

```
size(480,120);

arc(90,60,80,80,0,HALF_PI);
arc(190,60,80,80,0,PI+HALF_PI);
arc(290,60,80,80,PI,TWO_PI+HALF_PI);
arc(390,60,80,80,QUARTER_PI,PI+QUARTER_PI);
```

角度の大きさを指定するの弧度法を利用する場合には、円周率の値が使えると便利です。そのため、円周率πに関連する値を表す特別な名前が用意されています。

弧度法を扱うのに便利な名前（定数）

名前	意味	値
PI	円周率πの値を表す。	3.14159265358979323846
TWO_PI	2 πの値を表す。	6.28318530717958647693
HALF_PI	円周率の半分の値を表す。	1.57079632679489661923
QUARTER_PI	円周率の 4 分の 1 の値を表す。	0.78539816339744830961

PI のように特別な値を表す名前のことを定数 (constant) と呼びます。

描画の順番による結果の違い

Processing 言語では、図形の描画命令を実行する順番を変えると、描かれる画像が変化することがあります。次のサンプルプログラムを実行して、結果の違いを見て下さい。

描画命令を並び替えると（円→長方形）プログラム 1-11

```
size(400,200);
ellipse(140,0,190,190);
// The rectangle draws on top of the ellipse
// because it comes after in the code
rect(100,30,260,20);
```

描画命令を並び替えると（長方形→円）プログラム 1-12

```
size(400,200);
rect(100,30,260,20);
/*
 The ellipse draws on top of the rectangle
 because it comes after in the code
*/
ellipse(140,0,190,190);
```

順々に上書きされて描かれていくので、後から描いた図形が優先されます。一般的に、コンピュータのプログラムでは命令文を並べる順番を変更すると、実行結果が変わります。

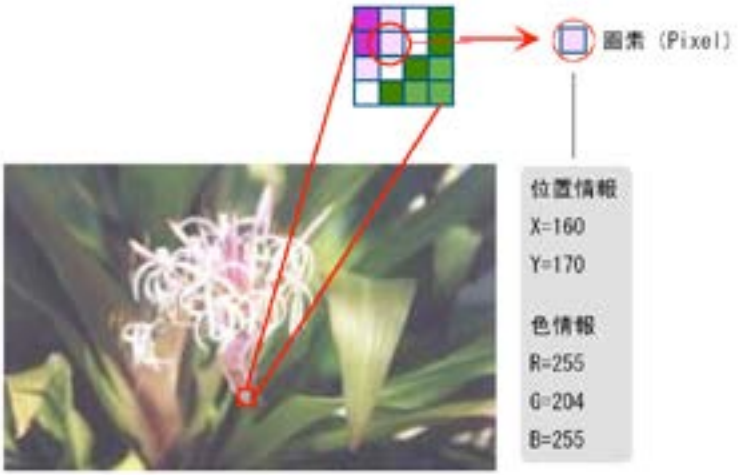
コメントとは？

プログラム内に書いた、プログラムの説明などをコメント (comment) と呼びます。Processing では、「//」をと書くと、これ以降行末まではコメントして扱われます。コメントは単なる説明なので、プログラムの動作には影響を与えます。複数行にわたるコメントを書く場合には、/* ～ */ という形式のコメントを使用することもあります。

図形の属性を変更する

基本的に、デジタル画像は画素と呼ばれる色の付いた小さい板の集まりとして記憶されています。そのため、デジタル画像では、画素の色とそれをどこの場所に置くかの情報を決める必要があります。

画素の色は、赤 (Red)、緑 (Green)、青 (Blue) の割合によって決めることが一般的です。Processing 言語では、色を指定する際には、特に指定をしない限り、この 3 つの値 (RGB) を 0 から 255 までの数字を用いて色を表現します。この RGB 以外にも、不透明度 (アルファ値) の情報を加えて 4 つの値 (RGBA) を用いることもあります。



デジタル画像のイメージ

RGB 以外にも HSB と呼ばれる、画素の色指定の方法があります。これは、色味を表す色相（Hue)、色の明るさを表す彩度 (Saturation)、色の鮮やかさを表す輝度 (Brightness) の 3 つの数値で色を指定するものです。Processing 言語のデフォルトでは、色相の情報は 0 ～ 360、色相と彩度の情報は 0 ～ 100 の数値で指定します。

自分の欲しい色を RGB の数値データとして表すことは、少し難しい作業です。そこで、Processing では Tools > Color Selector と呼ばれる機能が用意されています。これを利用することで，視覚的に自分の欲しい色の数値データを確認することが出来ます。Tools>Color Selector を選択すると、次のような Color Selector ウィンドウが開きます。

Color Selector の呼び出し方

右上の細長い四角形の色を数値データとして表したものが、HSB や RGB の部分の数字となって表示されています。細長い四角形をクリックすると色味（色相）を選択することが

リアルなデジタル画像？

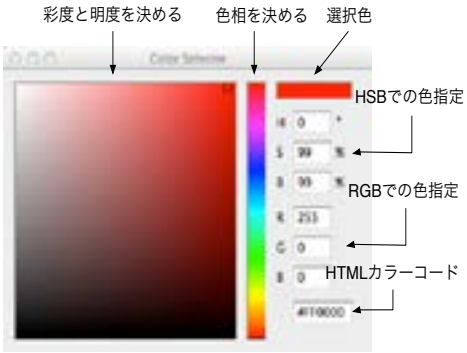
画素のことをピクセル (pixel) と呼ぶこともあります。

色の三原色

画素のことをピクセル (pixel) と呼ぶこともあります。色相の情報は 0 ～ 360 で表されているので、色相の異なる色を円周上に並べることが出来ます。これを色相環と呼ぶことがあります。

コンピュータ関連業界（?）では、ユーザが特に指定しない場合に、あらかじめ設定されている値また動作条件のことをデフォルト (default) と呼んでいます。

出来ます。左側の大きな四角形をクリックすると、色の明るさや鮮やかさやを変更することが出来ます。



Color Selector の機能

今までのプログラムで描かれた円や楕円を見ると、少しガタガタしているように見えます。もっと綺麗な円や楕円を描きたい時には、smooth 命令を使います。

楕円の描画滑らかにする

命令名（関数名）	意味
smooth():	楕円の描画を滑らかにする。
noSmooth();	楕円を滑らかに描画しないようにする。

楕円をもっと綺麗に描きたい（smooth と noSmooth）プログラム 1-12

```
size(400,400);
smooth(); // Turns on smoothing
ellipse(100,100,180,180);
noSmooth(); // Turn off smoothing
ellipse(300,300,180,180);
```

ellipse などで描画される図形は、外側の枠と内側の塗りつぶされる領域に分かれています。この外側の枠を示す線分の太さを変更することが出来ます。このために利用される命令が strokeWeight です。

枠線の太さを変更する

命令名（関数名）	意味
strokeWeight(width);	枠線の太さを width にする。

枠線の太さを変えたい（strokeWeight）プログラム 1-13

```
size(400,120);
smooth();
ellipse(60,60,90,90);
strokeWeight(8); // Stroke weight to 8 pixels
ellipse(180,60,90,90);
strokeWeight(20); // Stroke weight to 20 pixels
ellipse(300,60,90,90);
```

枠線は太さを変えるだけなく、表示をしないようにすることも出来ます。

この目的のためには、noStroke 命令を使用します。

枠線を表示しないようにする

命令名（関数名）	意味
noStroke();	枠線を表示しないようにする。

枠線を描かない（noStroke）プログラム 1-14

```
size(400,120);

smooth();
ellipse(60,60,90,90);
noStroke();  // without stroke
ellipse(180,60,90,90);
ellipse(300,60,90,90);
```

枠線の太さだけはなく、色を変更することも出来ます。枠線の色を変更するためには、stroke 命令を利用します。

枠線の色を変更する

命令名（関数名）	意味
stroke(x,y,z);	値 x,y,z で指定される色で枠線を描画するようになる。

Processing では枠線だけなく、図形内部の塗りつぶされる色などの変更をすることが出来ます。この目的のためには、fill,noFill 命令が使用されます。

塗り色などを変更する

命令名（関数名）	意味
fill(x,y,z);	値 x,y,z で指定される色で図形の内部を塗りつぶすようになる。
noFill();	図形の内部を塗りつぶさないようになる。

塗りつぶし色を変更したい（fill）プログラム 1-15

```
size(400,400);
smooth();
fill(255,0,0); // Red
ellipse(140,140,200,200); //  Red circle
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
fill(0,0,255); // Blue
ellipse(280,280,200,200); // Blue circle
```

図形を塗りつぶしたくない（noFill）プログラム 1-16

```
size(400,400);
background(255,255,255); // White
smooth();
noFill(); // Turn off filling
ellipse(140,140,200,200); //  Outline circle
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
fill(0,0,255); // Blue
ellipse(280,280,200,200); // Blue circle
```

描画色を変更する命令を実行すると、新たに描画色を変更する命令を実行しない限り、描画色の指定は変更されません。このような挙動をするプログラムなどは状態機械 (state machine) と呼ばれることがあります。

noFill 命令と noStroke 命令を一緒に実行すると、何も描画されなくなります。注意して下さい。

色の指定はデフォルトの RGB による方法が使用されています。

表示する図形の色だけではなく、background 命令を利用することで、背景の色を変更することが出来ます。

background という英語の単語の意味を知っていますか？

背景を指定した色で塗りつぶす

命令名（関数名）	意味
background(x,y,z);	値 x,y,z で指定される色で背景を塗りつすぶ。

背景色の変更（background）プログラム 1-17

```
size(400,400);
smooth();
background(100,100,100); // Gray
fill(255,0,0); // Red
ellipse(140,140,200,200); //  Red circle
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
fill(0,0,255); // Blue
ellipse(280,280,200,200); // Blue circle
```

複数しての組み合わせ プログラム 1-18

```
size(400,400);
background(255,255,255); // White
smooth();
noFill(); // Turn off filling
ellipse(140,140,200,200); //  Outline circle
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
fill(0,0,255); // Blue
ellipse(280,280,200,200); // Blue circle
```

複数指定の組み合わせ プログラム 1-19

```
size(400,400);
smooth();
background(100,100,100); // Gray
fill(255,0,0); // Red
ellipse(140,140,200,200); //  Red circle
noStroke();
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
noFill();
ellipse(280,280,200,200); // Doesn't droaw!!
```

複数指定の組み合わせ プログラム 1-20

```
size(400,400);
smooth();
background(100,100,100); // Gray
fill(255,0,0); // Red
ellipse(140,140,200,200); //  Red circle
noStroke();
fill(0,255,0); // Green
ellipse(200,40,200,200); // Green circle
noFill();
ellipse(280,280,200,200); // Doesn't droaw!!
```

描画命令を組みあせた例その 1

描画命令を組み合わせた例 1-21

```
// Learning Processing by Daniel Shifffman のサンプルを改変
size(400,400);
background(255,255,255);

// body
stroke(0,0,0);
fill(150,150,150);
rect(180,100,40,200);

// head
fill(255,255,255);
ellipse(200,140,120,120);

// eyes
fill(0,0,0);
ellipse(162,140,32,64);
ellipse(238,140,32,64);

// legs
stroke(0,0,0);
line(180,300,160,320);
line(220,300,240,320);
```

少し複雑な図形を描く

三角形 (triangle) や四角形 (rect,quad) を描く命令以外にも、多角形を描く方法が用意されています。これは描きたい多角形の頂点を vertex 命令で順番に指定していきます。どこからが描きたい多角形の頂点指定が始まっているか示すために beginShape 命令を、描きたい多角形の頂点指定の終了を示すために endShape 命令を利用します。2 つのサンプルを実行して違いを見て下さい。

vertex という英語の単語の意味を知っていますか？

便宜的に、vertex 命令、beginShape 命令、endShape 命令などと呼んでいますが、本来は、vertex 関数、beginShape 関数、endShape 関数です。

塗り色などを変更する

命令名（関数名）	意味
beginShape();	多角形の描き描きはじめを指定する。
endShape();	多角形の描き終わりを指定する。引数を CLOSE とすると、枠線を閉じて描く。
vertex(x,y);	頂点の位置を (x,y) にする。

endShape() の場合　例 1-22

```
size(400,200);
beginShape();
vertex(350,100);
vertex(290,50);
vertex(290,80);
vertex(50,80);
vertex(50,120);
vertex(290,120);
vertex(290,150);
endShape();
```

endShape(CLOSE) の場合　例 1-23

```
size(400,200);
beginShape();
vertex(350,100);
vertex(290,50);
vertex(290,80);
vertex(50,80);
vertex(50,120);
vertex(290,120);
vertex(290,150);
endShape(CLOSE);
```

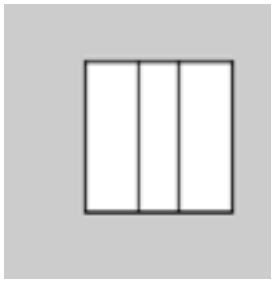
close という英語の単語の意味を知っていますか？

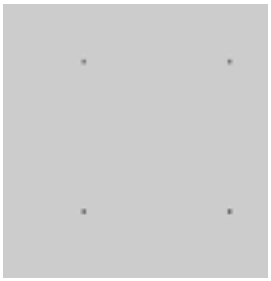
実は beginShape にも引数を指定することが出来ます。指定する引数により色々な多角形を描くことが出来ます。ここでは、Processing のマニュアルに出ている例を載せておきます。

beginShape に引数指定した場合　例 1-24

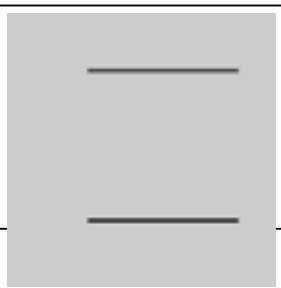
(Processging のマニュアルより)

プログラム例	描画結果

`beginShape(QUAD_STRIP);`
`vertex(30, 20);`
`vertex(30, 75);`
`vertex(50, 20);`
`vertex(50, 75);`
`vertex(65, 20);`
`vertex(65, 75);`
`vertex(85, 20);`
`vertex(85, 75);`
`endShape();`
	

`beginShape(POINTS);`
`vertex(30, 20);`
`vertex(85, 20);`
`vertex(85, 75);`
`vertex(30, 75);`
`endShape();`
	

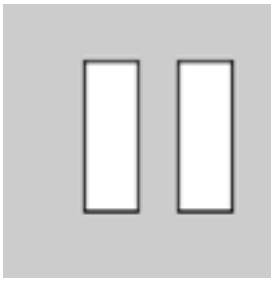
`beginShape(TRIANGLES);`
`vertex(30, 75);`
`vertex(40, 20);`
`vertex(50, 75);`
`vertex(60, 20);`
`vertex(70, 75);`
`vertex(80, 20);`
`endShape();`
	

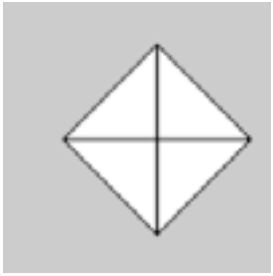
`beginShape(LINES);`
`vertex(30, 20);`
`vertex(85, 20);`
`vertex(85, 75);`
`vertex(30, 75);`
`endShape();`
	

実は、size 命令がない場合には、小さなウィンドウが開いて、描画が行われます。

プログラム例	描画結果

beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape();
	

beginShape(QUADS);
vertex(30, 20);
vertex(30, 75);
vertex(50, 75);
vertex(50, 20);
vertex(65, 20);
vertex(65, 75);
vertex(85, 75);
vertex(85, 20);
endShape();
	

beginShape(TRIANGLE_FAN);
vertex(57.5, 50);
vertex(57.5, 15);
vertex(92, 50);
vertex(57.5, 85);
vertex(22, 50);
vertex(57.5, 15);
endShape();
	

灰色系の色の指定

関数 fill などで色を指定する際に、白色、灰色、黒色の場合には、RGB の 3 つの値が同じ値となります。そこで、同じ数字を 3 つ並べて書くか代わりに 1 つで代用することが出来ます。

つまり、fill(255,255,255) と fill(255) は同じ意味になります。次のサンプルでは、このことを利用して色指定を行っています。

RGB を利用して色を表したときに、3 つの値が同じになるような色を無彩色と呼びます。そうでない色は有彩色と呼びます。

灰色系色の簡易指定　例 1-25

```
size(480,120);
smooth();
background(255);  // background(255,255,255);

// Left creature
fill(200);  // fill(200,200,200);
beginShape();
vertex(50,120);
vertex(100,90);
vertex(110,60);
vertex(80,20);
vertex(210,60);
vertex(160,80);
vertex(200,90);
vertex(140,100);
vertex(130,120);
endShape();
fill(0);   // fill(0,0,0);
ellipse(155,60,8,8);

// Right creature
fill(128);   // fill(128,128,128);
beginShape();
vertex(480-50,120);
vertex(480-100,90);
vertex(480-110,60);
vertex(480-80,20);
vertex(480-210,60);
vertex(480-160,80);
vertex(480-200,90);
vertex(480-140,100);
vertex(480-130,120);
endShape();
fill(0);   // fill(0,0,0);
ellipse(480-155,60,8,8);
```

曲線を描く

Processing 言語では曲線を描くことも出来ます。下に曲線を描くための関数 bezier と curve を用いた例を載せておきます。

bezier と curve のサンプル (Processging のマニュアルより)
プログラム例

```
size(100,100);
noFill();
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
size(100,100);
noFill();
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20,  80, 5,  80, 75,  30, 75);
size(100,100);
noFill();
stroke(255, 102, 0);
curve(5, 26, 5, 26, 73, 24, 73, 61);
stroke(0);
curve(5, 26, 73, 24, 73, 61, 15, 65);
stroke(255, 102, 0);
curve(73, 24, 73, 61, 15, 65, 15, 65);
```

インデント

インデント（字下げ）は、プログラミングにおいてプログラムの構造を明らかにするために、命令文の一群（コードのブロック）を字下げすることでです。

大半のプログラミング言語では、字下げは必須の事項ではありません。字下げは、自分を含むプログラマにプログラムの構造を見やすく伝えるために行います。特に、条件分岐や繰り返しといった制御構造を明確にするために利用されます。短いプログラムでは、まだ理解しづらいとおもいますが、例 1-26 と例 1-27 は同じ内容のプログラムとなっていますが、例 1-26 の方がわかりやすいプログラムになっていると思います。何文字くらいを文をずらすかにもいつかの流儀があります。通常は 4 ～ 8 文字程度をずらすようです。

どのように字下げをするかに関しては、いくつかのスタイルがあります。例えば、命令文の塊（ブロック）の開始の中括弧を制御文と同じ行に置き、ブロック内の文を字下げして表し、ブロックを閉じる中括弧を制御文と同じ字下げ位置に戻して書く（つまり、その行

Python などのプログラミング言語では、括弧やキーワードではなく字下げで構造を記述するようになっている。これをオフサイドルールと呼ぶ。これらの言語ではインデントを行うことは必須となります。

K&R スタイルと呼ばれることがあります。

は中括弧が先頭になる）というものです。このスタイルで例 1-26 を書いたものが、例 1-28 となります。

インデントあり　例 1-26

```
void setup(){
  size(640,480);
  smooth();
}

void draw(){
  if(mousePressed){
    fill(0);
  }else{
    fill(255);
  }
  ellipse(mouseX,mouseY,80,80);
}
```

インデントなし　例 1-27

```
void setup(){
size(640,480);
smooth();
}

void draw(){
if(mousePressed){
fill(0);
}else{
fill(255);
}
ellipse(mouseX,mouseY,80,80);
}
```

インデントあり（K&R スタイル）　例 1-28

```
void setup()
{
  size(640,480);
  smooth();
}

void draw()
{
  if(mousePressed){
    fill(0);
  }else{
    fill(255);
  }
  ellipse(mouseX,mouseY,80,80);
}
```

K&R スタイルに似たものでろオールマンスタイルというものがあります。このスタイルでは、例 1-29 のようになります。制御文の後の中括弧を次の行に置き、制御文と同じ字下げ位置とするもので、ブ

ロック内の文はもう一段字下げをされます。

インデントあり（オールマンスタイル） 例 1-29

```
void setup()
{
  size(640,480);
  smooth();
}

void draw()
{
  if(mousePressed)
  {
    fill(0);
  }
  else
  {
    fill(255);
  }
  ellipse(mouseX,mouseY,80,80);
}
```

インデントは自分の好きなものをスタイルで書けば良いと思います。ただ、首尾一貫して同じスタイルで書くことが重要です。

プログラムのソースコードは、自分のやりたいことをコンピュータを含めた別の人に伝えるためのコミュニケーションの手段です。しばらくすると自分で書いたプログラムでさえも、どのような動作をするものかを理解することが難しくなります。なるべくわかりやすくプログラムを書くことは、未来（明日）の自分のためです。