

Processing

Processingに関する本チュートリアルは、Josh NimoyによりITPで学ぶ学生を対象に作成されたものを日本語に翻訳したものである。

Language: [English](#) , [Español](#), Japanese , [Korean](#)

Workshop Teacher

Josh Nimoy
contact: jn429 at nyu dot edu
[website](#)

Software Creators

Processing is an open project
initiated by
[Ben Fry](#) and [Casey Reas](#)

Japanese Translation by

[Hironobu Fujiyoshi](#) and Ayako
Takabatake
contact: hf@cs.chubu.ac.jp

Korean Translation by

[Koo-Chul Lee](#)
contact: kclee [at] phya [dot] snu
[dot] ac [dot] kr

Spanish Translation by

[Gerald Kogler](#) & Angela Precht
contact: gerald [at] yuri [dot] at

Description

Processingは電子メディアより実現されるコンセプチュアルな空間を作り出す環境である。Processingは電子的な芸術作品を作成する過程を通じてコンピュータプログラミングの基礎を学ぶための環境であり、またアイデアを描き試すためのスケッチブックでもある。Processingはインタラクティブな画像マルチメディアのプログラミング環境として有名なJAVAを使用している。システムはブラウザに埋め込まれたJAVAアプレットのようにWEB上で動作する作品を出力することができる。また、このシステムは、本来のJAVAと[教育を目的としたグラフィックプログラミング環境](#)とのギャップを埋める架け橋となるべく設計されており、ProcessingはJAVA等の練習台としても使用できる。www.Proce55ing.net

このチュートリアルの目標はProcessingという環境をMacromedia FlashとDirectorの利用者に紹介することである。Macromedia社の製品より得た知識は、Processingに必要なものに依じて分割すれば簡単に真似ることが出来るはずであると考え。その際ユーザはMacromedia製品に関する基本的な理解が必要となる。しかしこのチュートリアルを読み終えるころには自らが作成するProcessing(またはJAVA)の作品を創造できるようになるだろう。また、シリアルポートや[BX-24 chip](#)を用いた通信も可能となるだろう。

索引

[Introduction](#)
[Obtaining the Processing Software](#)
[A tour of the interface](#)
[Lower Level Media Manipulation](#)
[Syntax Structure](#)
[Static 2D Drawing](#)
[Time and Motion](#)
[Mouse & Keyboard](#)
[Presentation / Exporting](#)
[Drawing Image Files](#)
[3D Form](#)
[Pixels](#)
[Typography](#)
[Serial](#)
[The Future](#)

Introduction

コンピュータによるインタラクティブデザインの教材としてFlashやDirectorが既に多く用いられている。学生は幾何学的かつ動的な要素を含むデザインを又複雑なアルゴリズムを用いて作成するようになった。以前ITPのある教室にて次のような実験が行われた。DirectorLingoに取り組んでいる際一週間JAVAに関する講義を行った。Lingo以外の言語に触れさせ、新しい言語やシステムの異なる点や見方について学ばせた。基本となるテンプレートと簡単なリファレンスを学生らに与えプログラムを変更させた。この一週間に及ぶJAVAの講義の後、何人かの学生はJAVAを学ぶことに関心を抱いていた。しかし、本来大学におけるJAVAの講義はテキストコンソールを用いており、グラフィックアプレットを対象としていない。

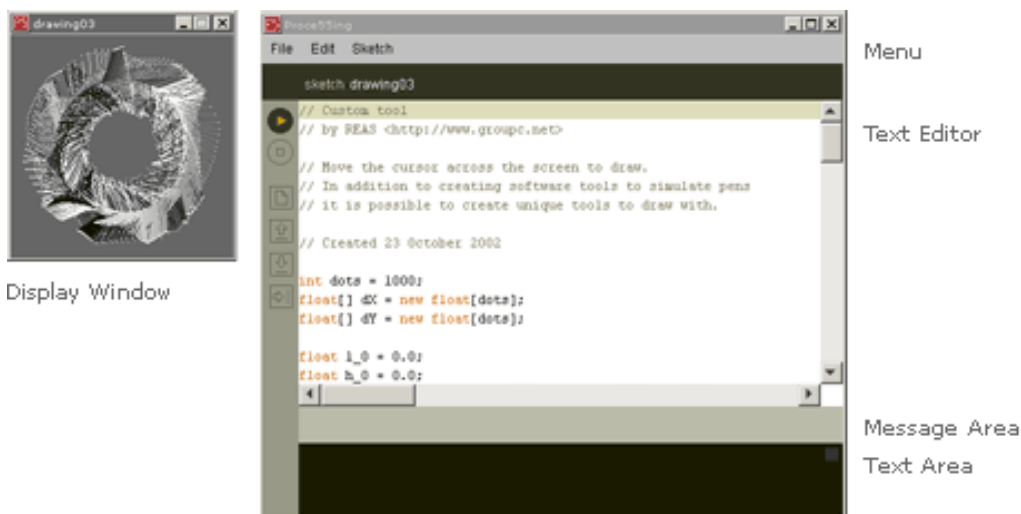
以下のレッスンを通してProcessingがプログラムとグラフィックデザイン間の差を乗り越える手助けとなることを期待している。これは決してJAVAの講義の代役として学ぶことを意味してはいない。ProcessingはJAVAの深いニュアンスを伝える以前の段階を学ぶ上でのサプリメントとして使用してもらいたい。また、ProcessingとJAVAはMacromediaの製品ほど高レベルでは無いが、他の低レベルなシステムよりはよいプログラミングと実行環境を提供してくれる。柔軟な考えをもつインストラクターが居れば、JAVAの講義を受講している者が提出物を作成する際にProcessingを使用できるかもしれない。このチュートリアルはCasey ReasとBen Fryの作成したProce55inのサイトより得られた情報を筆者独自の観点よりまとめたものである。

Obtaining the Processing Software

Processingは現在開発段階ではあるが無料で配布されており、完成した後も無料配布が予定されている。このソフトウェアは現在開発・改良の必要なALPHA段階にある。バグは直され、新たな機能が追加されている。開発テストの協力者、又ダウンロードを希望する者は一度開発者にメールで問い合わせてもらいたい。将来的にはProcessing websiteにてダウンロードを予定している。またa版のテスト協力者間のメッセージボードが存在する。参加は登録制であるが、そのコミュニティーに参加することは質問や疑問を解決する最もよい方法であるといえる。ここでは他のユーザを含め製作者側にも直接声を投げかけることが出来る。方法としてはProcessingのサイトにてDiscourseを選び登録してもらいたい。Processingとそのサイトは定期的に更新されている。Referenceにて更新機能を確認してもらいたい。

A tour of the interface

次の画像はwww.Proce55ing.net上にあるものである。関連事項はReferenceから Environmentを押して参照してもらいたい。



「なんて単純なインターフェイスなんだろう、これが本当にFlashやDirectorのような優れた機能を持つのか」と画面を見た瞬間思うかもしれない。DirectorやFlashには多くの入力や共用の編集機能が付属している。Processingではこれらの機能は既に他のプログラムやJAVAにより作成されている。例として、Flashは簡易Illustratorを、またDirectorは簡易Photoshopをそれぞれ提供している。Processingにおいては(JAVAにおいても同様)ユーザは自らベクターパスのリストやgifファイルを作成し、それらのデータをプログラムにより描画する。ユーザはプログラムを用いることによりもっと直接的にスクリーン上のピクセルを自らのつくり出した構文や形式によって制御することが出来る。また、各ユーザが独自に作成した構文や形式の提供があればProcessingは更に利用しやすいシステムとなるだろう。

次にインターフェイス左部の6つのボタンについて説明する。



実行(play)ボタンはDirectorやFlashと同様にプログラムを実行する際に使用。



停止(**stop**)ボタンはDirectorやFlashと同様にプログラムを停止する際に使用。



新規作成(**new**)ボタン新しいファイルのことをProcessingではスケッチと呼ぶ。(他にもアプレット、プログラム、インタラクティブ素材と呼ばれることもある)一方DirectorやFlashではこれをムービー(*movies*)と呼ぶ。



読み込み(**Opens**)ボタン スケッチを読み込む際ポップメニューが横に表示されるため、そこから保存された作品を選択する。また、あらかじめ登録されている例を読み込み表示させ、プログラムや動作を勉強することも可能である。



保存(**Saves**)ボタン インターフェイス上に表示しているスケッチに名前をつけて保存する際に使用。(別名で保存したい場合はファイルメニューより"save As"を選択)



出力(**Exports**)ボタン 表示されているスケッチをJAVAアプレットとして出力する。またその際にJAVAアプレットを表示するために必要な最低限のHTMLタグを書き出す。

Processingに関する他の環境や命令については[Processing Environment reference](#)参照。

Lower Level Media

Manipulation

Directorでは、キャスト(*cast*)にメディアの読み込み又は作成を行い、次にステージ(*stage*)と呼ばれる画面部にスプライト(*sprite*)を生成・配置する。Flashではライブラリ(*library*)にメディアの読み込み又は作成を行い、Directorと同じようなステージにムービークリップ(*movieclips*)としてインスタンス化する。一方のProcessing(そしてJAVA)においてメディアの読み込みはコードにより既に終了しており、これはHTMLの働きと似ている。さらにどのようなメディアであれ(ベクターシステム、DNAデータ、フィルムからのカラーサンプル、Fargo)JAVA言語の一部として埋め込まれている。その結果外部からの画像や音等を1つのファイルにまとめることが何の制限も無しに実現できる。何故なら画像のピクセルもプログラムコードの一部として変換され、音データも巨大なデータ配列として格納されるためである。ライブラリやキャストをもつことの利点としてフォーマット時の制御の良さ、順序良くディスク(メモリ)の空き領域に保存できる等が挙げられる。またスプライトやムービークリップを持つ利点として視覚的であること、スクリーン上に見える物体であるため簡単にボタン、ゲームのキャラクタ、独立したグラフィック素材を作成することが可能であること、視覚的に制御可能かつ明確な空間上の素材等が挙げられる。Processingでは複雑なレイヤーは実現されておらず、基本的な描画ルーチンとマウス、キーボード、シリアルポート接続によるイベントのみを提供している。入力と時間の調整により、ある1シーンを繰り返し描画することもできる。問題としては自らが個々のスプライトやムービークリップを**描かなくてはならない**が、Processingでは必要とされていない。他の方法を発明することはアーティストとして便利なものを作ることになる。更によくスポットを当てるMacromedia社の関連物に対して芸術的に深い変化を生じる機会を与える。

この先のセクションではまずスクリーンに画像を描画する方法を、次に時間とアニメーションについて、最後にマウス、キーボード、シリアルポートを用いたインタラクションについて紹介していく。世の中にはユーザの行う作業を順に創造してくれるを高度なソフトウェアがあるが、それらの作業を自らが組み立てていくことを頭におけば、JAVAを用いて作業することもすぐに受け入れることができるだろう。

Syntax Structure

FlashMXを使用したことのある者にとっては復習になるが、下記の例はskeches内にあるstructure00である。

```
// 命令文とコメント文
// by REAS

// 命令文はプログラムを構成する要素の1つである。
// ";"セミコロンは命令文の末尾につき、これを"ステートメントターミネータ"と呼ぶ。
// コメント文はユーザがよりプログラムを理解しやすいようにメモをとる際に用いる。
// コメントは二本のスラッシュより始まる。

// 作成日 1 September 2002

// size関数はコンピュータに表示画面の大きさを決定する。
// 各関数には0または何個かの引数が必要となる。
// 引数とは他の関数に渡されるデータのことであり、
// コンピュータの動作を数値により決定していく。
size(200, 200);

// background関数はコンピュータに背景色を決定する。
background(102);
```

またJAVAの変数については次に示す。

```
int x = 0;
println(x);
x=x+1;
println(x);
x=x+1;
println(x);
```

実行結果:

```
0
1
2
3
```

Flashユーザへ：Processingには変数(var)という概念が無い。
変数についての詳細情報は公式の[JAVAチュートリアル](#)を参照すること。

if-then文について。

```
int a = 1;
int b = 2;

if(a==b){
  println("same");
}else{
  println("different");
}
```

実行結果:

```
different
```

条件分岐における比較部分の記述方法がLingoと若干異なる。Processingのプログラムは一つの"="を変数の値を代入する際に使用する。また、二つの"=="を双方の値を比較して同じ値か否かを調べる際に使用する。このとき同じ値ではないという記号は "<>"ではなく"!="と記載されることに注意してほしい。その他の比較記号("<",">","<=",">=")の使用方法はLingoと同様である。比較条件に関する詳細情報は[consult the authority](#)を参照すること。

繰り返し文について。

```
for(int i=0 ; i<5 ; i++){
  println(i);
}
```

実行結果:

```
0
1
2
3
4
```

Lingoユーザへ：このプログラムは"repeat with i = 0 to 4"と同じ意味を持つ。()の中身は三つの特別な構文がセミコロンで分けられて並んでいる。一つ目の構文は一時的な変数を作成。二つ目の構文は何度繰り返し処理を行うかの状態を定める。変数iが5を超えた大きな数値となった後に繰り返し処理が終了する。三つ目の構文はiの変化量を定めることができる。"i++"は"i = i + 1"の省略した形である。JAVAを提供するSunならば [繰り返し文に関するさらに多くの知識](#)を与えてくれる。

While文はif-then分と似ている。

```
while(6!=2){
  println("muhahaha!");
}
```

無限ループに陥いるため
実行しないこと...

:)

If you are この現象について興味のある者はJAVAチュートリアル内に記述されている [the part about flow-control](#)についてを参照するとよい。

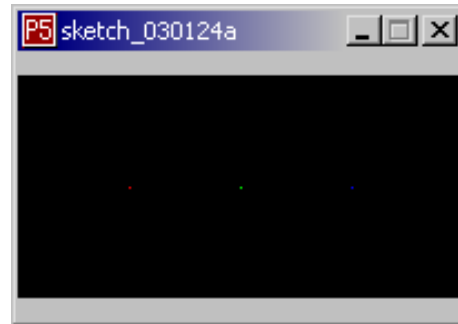
次に基本的な関数についての話を述べる前に、描画関数について触れておく。全てに目を通す必要はないが、描画機能を使いこなすためには必要なものである。

命令文に関する他のプログラム言語との[言語比較](#)並びに[Processingプログラムコード例](#)の詳細に関しては各リンク先を参照すること。

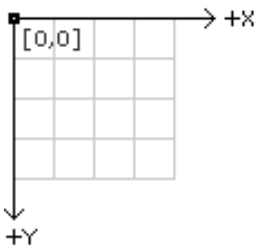
Static 2D Drawing

```
size(200,100);
background(0,0,0);
stroke(255,0,0);
point(50,50);
stroke(0,255,0);
point(100,50);
stroke(0,0,255);
point(150,50);
```

このコードを実行すれば次のような黒く横に長いスクリーンに三つの色付きピクセル(赤、青、緑)が出力されるはずである。



次にこのコードを行ごとにかけて解説していく。



基礎知識として、スクリーンはそれぞれの位置情報を持つピクセルと言う集合からできている。その原点は長方形の左上を(0,0)とする。Y方向に値を増やせば下方向に、X方向に値を増やせば右方向に動く。これはボードゲームのBattleShipと良く似ている。また、FlashやDirectorも同様である。

`size(200,100);`

`size`関数は画面上のキャンバスの大きさを設定する。この場合は縦200ピクセル、横100ピクセルのキャンバスが出来上がる。この関数をはじめにコールしない場合、デフォルト値としてサイズは100x100ピクセルとなる。

```
background(0,0,0);
```

background関数はキャンバス全体の色を決定する。Directorではステージカラー、Flashではドキュメントの背景を指す。また、0,0,0は黒色を表す。背景を決定しなかった場合デフォルトは灰色となる。

```
stroke(255,0,0);
```

stroke関数は描く色を決定する際に用いる。この関数の後に続く描画コマンドは先にstroke関数で示された色で描画する。描画色を指示しない場合黒色がデフォルトとなる。

```
point(50,50);
```

point関数はその与えられたアドレス(50,50)に前行のstrokeで設定された色(この場合は赤色)のピクセルを描く。

```
stroke(0,255,0);
```

```
point(100,50);
```

このコードはキャンバスの中心に緑色の点を描く。

```
stroke(0,0,255);
```

```
point(150,50);
```

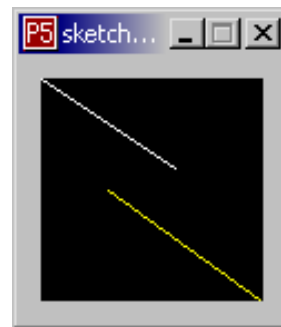
このコードはキャンバスの右方に青色の点を描く。

以上の解説より、使用されている命令コードはLingoの**draw**関数や、FlashActionScriptの描画関数にとっても似ている。ユーザはこれらの描画命令を理解して、目の前に表示されているスクリーンをデジタルキャンバスとして作品を創作することが可能となる。更に複雑な形の描画方法をいかに示す。

```
background(0,0,0);
stroke(255,255,255);
line(0,0,60,40);
stroke(255,255,0);
line(30,50,100,100);
```

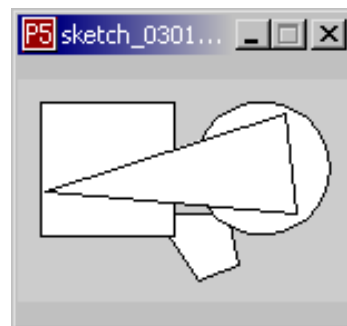
ここには二本の線が描かれている。一本目は白色、二本目は黄色である。

line関数は初めの二つのパラメータが始点のx-y座標を、また残りの二つのパラメータが線の終点座標を表す。



次に組み立て式の形を紹介する。

```
size(150,100);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



triangle関数 は三つの頂点を持つ図形を描く。これには六つのパラメータがあり、パラメータ1,2は一番目の頂点のx-y座標、パラメータ3,4は二番目の頂点のx-y座標、パラメータ5,6は三番目の頂点のx-y座標を示す。

```
triangle(x1, y1, x2, y2, x3, y3);
```

quad関数 will は四つの頂点を持つ図形を描く。これには八つのパラメータが存在し、その記述方法は四番目頂点の頂点のx-y座標を示すパラメータ7,8が追加される以外はtriangle関数の場合と同じである。

```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

rect関数は長方形を描く関数である。パラメータ1,2は長方形を描く開始位置を決定し、パラメータ3,4で高さと幅を決定する。

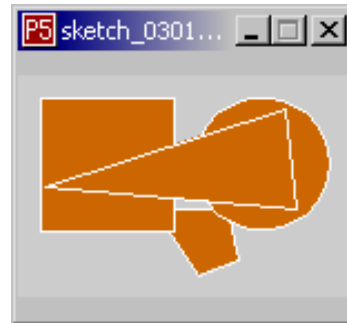
```
rect(x, y, width, height);
```

ellipse関数は楕円を描く関数である。パラメータの設定方法はrect関数の場合と同じである。

```
ellipse(x, y, width, height);
```

ここから新しい命令を紹介する。新たに表記されているものは**オレンジ**で記載されている。

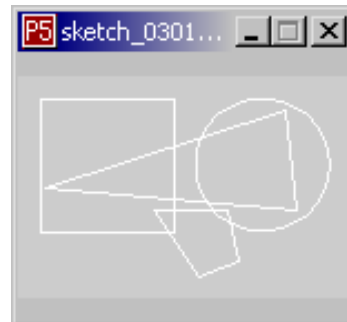
```
size(150,100);
fill(#CC6600);
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



(注意：色の設定法は前述までのRGB法では無くHTMLの表記方法で行うこともできる)

Fill関数はstroke関数と同じような意味を持ち、図形の中を塗りつぶす。図形を囲む線については先に学んだstroke関数を用いて設定する。またfill関数のデフォルトは白色と設定されている。

```
size(150,100);
noFill();
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



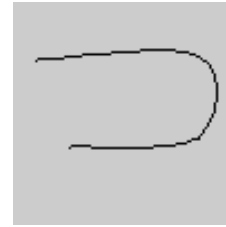
noFill関数を使用すれば図形の中を塗りつぶさない線分のみのアウトライン図形を描くことができる。また、これと同様にstrokeにもnoStroke関数が存在する。noStroke関数は図形のアウトラインを無くした状態の図形を描画できる。

曲線を描くには、直線よりも少し複雑になる。Processingではcurve関数とbezier関数を提供している。

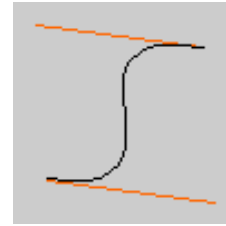
```
curve(84, 91, 68, 19, 21, 17, 32, 100);
```



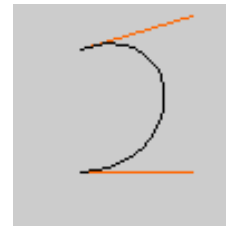
```
curve(10, 26, 83, 24, 83, 61, 25, 65);
```



```
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);
```



```
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);
```



```
curve(x1, y1, x2, y2, x3, y3, x4, y4);
bezier(x1, y1, x2, y2, x3, y3, x4, y4);
```

curve関数のパラメータ1,2は線の始点を表し、パラメータ7,8は線の終点を表す。間に挟まれた各パラメータの対はカーブを定義する際に通過する点を表す。

bezier関数のパラメータ1,2,3,4は **curve**関数のそれらと同じものを示す。また、間に挟まれたパラメータの対は曲線を定義する際に必要な制御点を表す。

上記の例において表示されている図形のオレンジ色の線分は隠れた制御点を現している。

以上のように予め定義された基本的な図形描画関数は存在するが、ユーザは自らの好きな形を構築することも可能である。

beginShape関数と**endShape**関数、これらの関数は複雑な形状の図形を描画する際に鍵となる。**beginShape**関数は図形の頂点を記録を順次行う。一方、**endShape**関数はその記録を終了する。**beginShape**関数には記録していく頂点の値を使用してどのような図形を描くかのパラメータを引数として設定する必要がある。このときに記すパラメータは**LINES**, **LINE_STRIP**, **LINE_LOOP**, **TRIANGLES**, **TRIANGLE_STRIP**, **QUADS**, **QUADS_STRIP**, そして**POLYGON**のいずれか一つである。**beginShape**関数の後には**vertex**関数を並べる必要がある。図を描くのを終了するときは**endShape**関数を入れる。**vertex**関数はパラメータの数により2Dまたは3Dの頂点を定めることができる。2Dの頂点座標を設定する際はx-yの座標値を、一方3Dの頂点座標を設定する際はx-y-zの座標値を定義する。各図形は予め**stroke**関数で定義されていた色でアウトラインが縁取られ、**fill**関数で塗りつぶされる。

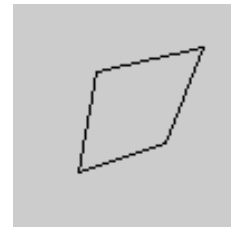
また、Processingの取り扱いについて以下の注意が必要である:

Processingは現在凸面の多角形のみ描画可能な状態である。しかし、将来的には凹面の多角形の描画も可能となるように現在取り組んでいる。

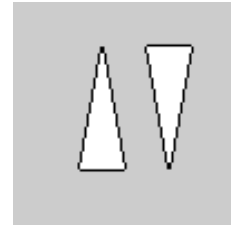
凸面多角形の作品例またそれらのプログラムはサイト内に展示してあるものを見てもらいたい。

以下にProce55ing.net上に展示されている作品を作成例として紹介する。

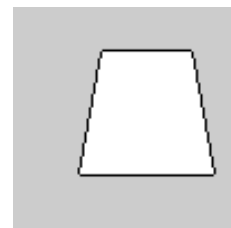

```
beginShape(LINE_LOOP);  
vertex(30, 20, -50);  
vertex(85, 20, 0);  
vertex(85, 75, -80);  
vertex(30, 75, 0);  
endShape( );
```



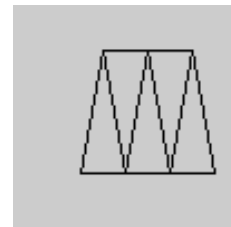
```
beginShape(TRIANGLES);  
vertex(30, 75);  
vertex(40, 20);  
vertex(50, 75);  
vertex(60, 20);  
vertex(70, 75);  
vertex(80, 20);  
vertex(90, 75);  
endShape( );
```



```
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( );
```



```
noFill( );  
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( );
```



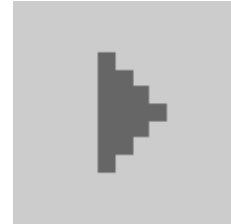
```
noStroke( );  
fill(153, 153, 153);  
beginShape(TRIANGLE_STRIP);  
vertex(30, 75);  
vertex(40, 20);  
vertex(50, 75);  
vertex(60, 20);  
vertex(70, 75);  
vertex(80, 20);  
vertex(90, 75);  
endShape( );
```



```

noStroke( );
fill(102);
beginShape(POLYGON);
vertex(38, 23);
vertex(46, 23);
vertex(46, 31);
vertex(54, 31);
vertex(54, 38);
vertex(61, 38);
vertex(61, 46);
vertex(69, 46);
vertex(69, 54);
vertex(61, 54);
vertex(61, 61);
vertex(54, 61);
vertex(54, 69);
vertex(46, 69);
vertex(46, 77);
vertex(38, 77);
endShape( );

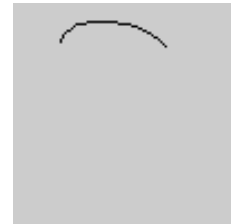
```



```

beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
endShape( );

```



```

beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
curveVertex(32, 100);
endShape( );

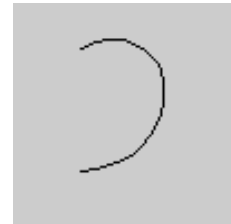
```



```

beginShape(LINE_STRIP);
bezierVertex(30, 20);
bezierVertex(80, 0);
bezierVertex(80, 75);
bezierVertex(30, 75);
endShape( );

```



ベクター描画に関するの詳細は[Processing Form Examples](#)、[Processing Shape reference](#)を参照してもらいたい。

ここに紹介した以外の描画方法もProcessingに用意されているが、次にインタラクティブアクションやアニメーションの作成方法について説明する。その後に残りの描画方法について解説する。

Time and Motion

Directorにおいてはスコアが存在し、頭出しや先送りを行いコマを補完していく。ビデオ、FlashSWF、QTVRsや音はそれぞれの時間軸を持つ。更なる動的なアニメーション作成に取り組む場合、一つのフレームとコードをExitFrameやPrepareFrameイベントで使用する可能性もある。Flashではタイムラインが与えられ、Directorよりも更に複雑な制御が可能である。通常ActionScriptを用いて作業する場合、ユーザは二つのフレームを使用する。一つは繰り返されるフレームを呼ぶセット用、もう一つは永久にループさせるためのものである。更にActionScriptはonClipEventなるマウス操作に対応する関数も存在する。しかし、Processingにはタイムラインもスコアも補完機能も無いため、LingoやActionScriptのようなスクリプトとなるコードをユーザが構築しなければならない。Processingは連続するフレームに応答するユーザ独自の関数を取り扱うことが出来る。今まで解説してきたものはBasic Modeと呼ばれる段階である。このモードは静的な画像を描くためのもので単に順序立てて並べていくに過ぎない。Processingにはbasic, standard, そしてadvancedからなる三つのモードが存在する。advancedモードは通常のJAVAであり、JAVA熟練者用である。時間と動作を実現する上で、次のステップであるstandardモードに移行する。ここに簡単な例を示す:

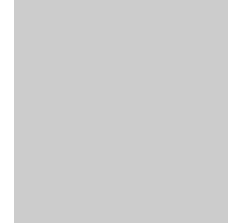
```
int x = 0;
```

```
void setup(){
  noStroke();
}
```

この例では白い長方形が左から右へ一度だけ移動する。

```
void loop(){
  rect(x, 0, 5, 100);
  x=x+1;
}
```

この動作をアニメーションGIFにしたものを右に示す。



setup関数に囲まれた部分はプログラムの開始後一度だけ実行される。loop関数に囲まれた部分はプログラムが終了するまで永久に繰り返される。setup関数はLingoのbeginSpriteやstartMovieに、またloop関数はExitFrameやPrepareFrameに似ている。Flashにおいてsetup関数は実行後一度しか動作しない最初のアニメーションフレームに似ている。setup, loopは共に関数である。また、複雑な関数を集約したり組織化するような **ユーザ独自の関数を作成**することも出来る。JAVAで新しい関数を作成する際の情報源としてはSunの[JAVAチュートリアル](#)を参照すること。

初めの関数を記載したとたんにProcessingのモードはstandardモードに移行する。関数でまとめられた部分以外に何かを記載しても、変数宣言以外の構文は実行ボタンを押した際には無視される。コードはsetup関数、もしくはloop関数の中に入れること。グローバル変数を必要とする場合、両関数の間では無くプログラムの先頭に記述すること。例に用いたコードではxがグローバル変数として定義されている。

Processingにおいてはframerate(n)関数がスケッチ全体の動作スピードを調節するのに利用できる。しかし、ものをそれぞれ**異なるスピードで動かす**ことも変数の変動具合により調節できる。一方で小数値を用いてわずかな変化を生じさせることも可能である。更に長い期間や微小な時間制御のために、Processingでは日付や時間等へのアクセスも可能である。

Processingにはコンピュータより得られる日付けや時間を取得する様々な方法が存在する。

year関数 // 現在の西暦を返す(i.e. 2002, 2003, etc.)

month関数 // 現在の月を返す(1..12)

day関数 // 現在の日にちを返す(1..31)

hour関数 // 現在の時数を返す(0..23)

minute関数 // 現在の分数を返す(0..59)

second関数 // 現在の秒数を返す(0..59)

特殊な関数で**millis**関数がある。これはアプレット開始後のミリ秒(1000分の1秒)を返す。主にアニメーションのタイミングを計る際に使用する。

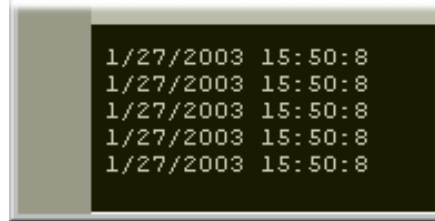
millis() // アプレット開始後のミリ秒(1000分の1秒)を返す。

It is 更に遅延関数を用いることによりアプレットを待機させることも可能である。delay関数を用いてフレームの動作を遅延させることが出来る。

delay(40); // takes a nap for 40 milliseconds

```
void loop() {
  print(month() + "/" );
  print(day() + "/" );
  print(year() + " ");
  print(hour() + ":" );
  print(minute() + ":" );
  println(second());
}
```

ここにあるサンプルは Processingの下方のウィンドウ上に現在の日付と時間を出力するものである。



これは良い例とは言えないが、簡単に理解することが出来る。更に複雑な例については次の作品を見ることを推奨する。[時計、 by Mescobosa](#)、[Milliseconds、 by REAS](#) また、アニメーション動作のための[サンプル](#)も挙げておく。

Mouse & Keyboard

マウスとキーボードを利用する場合、FlashやDirectorとの違いは少ない。Lingoではマウスは"the mouseLoc", "the mouseH", そして"the mouseV"というコマンドで状態を得る。更にonMouseDownイベントハンドラでマウスのクリック等も認識できる。FlashではonClipEventが存在し、これもマウスの状態や座標を得ることが出来る。これらに対しProcessingではマウスがクリックされる度にmousePressed関数を呼び出され、マウスボタンを離す度にmouseReleased関数が呼び出される。そのためマウスの動作に対して何かをさせたい場合はloop関数同様に、動作を関数の中に組み込めば良い。

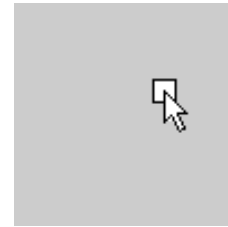
```
void loop() {
  rect(mouseX-5, mouseY-5, 10, 10);
}

void mousePressed() {
  fill(0);
}

void mouseReleased() {
  fill(255);
}
```

この単純なサンプルではマウスの動きに追従する様に白い正方形が描かれている。

また、クリックしたときに正方形が黒色に変わる。

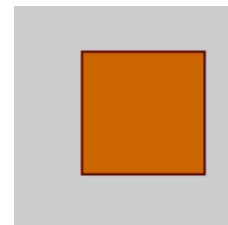


マウスに関する命令の詳細情報は[Processing Mouse reference](#)に、また、興味深い[サンプル集](#)のチェックも忘れずに。

キーボード入力についてはDirectorやFlashと全くかわらない。

```
void loop() {
  if(keyPressed) {
    fill(102, 0, 0);
  } else {
    fill(204, 102, 0);
  }
  rect(30, 20, 55, 55);
}
```

この単純なサンプルではキーボード上のキーが押されたときに四角形の色が濃いオレンジに変化する。



またキーボード入力はイベントハンドル用関数としても利用できる。

```

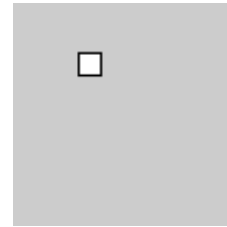
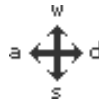
int x = 50;
int y = 50;

void loop( ){
  rect(x,y,10,10);
}

void keyPressed( ){
  if(key=='w' || key=='W'){
    y--;
  }else if(key=='s' || key=='S'){
    y++;
  }else if(key=='a' || key=='A'){
    x--;
  }else if(key=='d' || key=='D'){
    x++;
  }
}


```

このサンプルでは指定されたキーを押すと画面上の正方形が移動する。




キーボード入力に関する機能については[Processing Keyboard reference](#)を参照、また、興味深い[サンプル集](#)のチェックも忘れずに。

Presentation / Exporting

 FlashとDirectorにおいてはキーコントロールとメニューバーよりディスプレイ全体を使用してプログラムを実行させることができる。フルスクリーンモードは調整時や発表時にとっても有効である。ProcessingでもメニューのsketchからPresentを選択するか、Ctrl + P (apple + p on Mac)を押すことによりフルスクリーンモードにすることが出来る。(⌘+P on a Mac)更に試してもらいたいことがある。SHIFTキーを押しながらplayボタンを押してもらいたい。スクリーン全体が濃いグレーに変わり、中央に描き出したものが表示される。元の状態に戻す際はescキーか画面左下のstopボタンを押して終了となる。



 Processingで作成されたものはすべてJAVAアプレットとして出力することが出来る。はじめに自分のsketchが保存されていることを確認した後にFileをメニューから選び、Export to Webを選択する。もしくはCtrl + Eが画面左方のExportボタンを押す。すぐにメッセージウィンドウに出力していると言う意のメッセージが表示され、終了したときにはその旨を表示する。こうして出力されたwebファイルはProcessingフォルダの中のsketchbook内にある自分が名付けたファイルの下のappletフォルダに格納されている。Processingにより生成されたデフォルトのindex.htmlは是非ユーザに練習がてら編集してもらいたい。ただappletフォルダ内にそのhtmlファイルに関係のあるファイルをまとめておかない。他のhtmlフォルダについてもlinkを切らないために同様のことが言える。

Processing Folder/

```

sketchbook/
  default/
    your_sketch_name/
      applet/
        your_sketch_name.java
        your_sketch_name.class
        your_sketch_name.jar
      index.html

```

ScreenGrab()

また、先ほどとは逆に出力先をインタラクティブな動作の無い形式にすることも可能である。screenGrab関数を利用することにより表示されている画面を.tif形式にすることができる。loop関数の末尾にこの関数を配置することによりスクリーン上のイメージを保存する。この関数は複数回呼ばれた際は次のような画像の連鎖を生成する。(screen-0001, screen-0002, ...)これらは簡単にQuicktimeや他のビデオプログラムに移しProcessingの生きたドキュメントとして動作させることも出来る。しかし未だProcessingの画像保存関数は簡単なものである。将来的には他のフォーマットでも出力できるように開発を進める。ここに一例としてAdobe Illustrator形式で出力するものがある。

Drawing Image Files

他のイメージをProcessing上に表示するのは簡単である。JAVAはjpg又はgif形式の画像のみを受け付けるため、他の形式の場合はこの何れかに加工する必要がある。ファイルシステムと数行のコードを使用することにより、他の画像を簡単にsketch上に表示できる。はじめにsketchを保存する。そしてProcessingのプログラムフォルダの中を見ていくとsketchbook と名付けられたファイルがある。そのフォルダの中にはexampleフォルダともう一つdefaultフォルダがある。このdefaultフォルダの中に先ほど名前を受けて保存したsketchと同じフォルダが存在する。簡単に画像を利用するためにはこの中のdataフォルダに外部より持ち込む画像を入れる必要がある。実例としては以下のファイルとなる:

```
Processing Folder/  
  sketchbook/  
    default/  
      your_sketch_name/  
        data/  
          your_imagefile.gif
```

例えばimage_example_1というsketchがあるとすると、これに外部より持ち込む画像としてtwombly.jpg (Twombly作)が存在する。

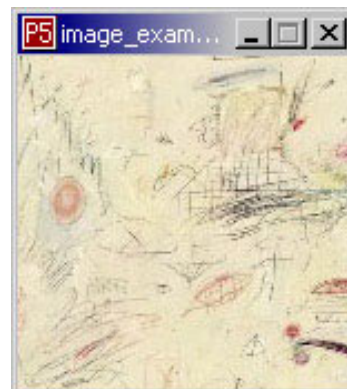


ユーザがtwombly.jpgを以下の場所にコピー終わるとコードを打ち込む準備が出来る。:

```
Processing Folder/  
  sketchbook/  
    default/  
      image_example_1/  
        data/  
          twombly.jpg
```

次に以下のコードを記述する。

```
size(150,150);  
BImage b = loadImage("twombly.jpg");  
image(b,0,0,150,150);
```



Bimageは事項されているファイルを描き終わるまで保持するオブジェクトである。bはこの画像につけた適当な名前である。そしてimage関数が実際に画像をスクリーンに描画する。

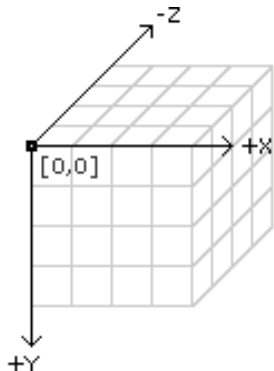
```
image(BImage, x, y, width, height);
```

画像のサイズを決める高さと横幅については省略することも可能である。省略した場合は元の画像のサイズが適用される。

直に画像ファイルをフォルダの下に置く以外の方法としてURLの利用も可能である。

既存の外部ファイルを読み込む場合に関する詳細については[Loading_and_Displaying](#)を参照すること。より多くの知識を得たい場合に[sequential images \(video footage\)](#)を参考にすることもできる。更なる情報源として[Processing Image Examples](#)を参考にするのもよい。

3D Form



3Dが固有の2D環境へ取り入れられる過程には多くの論争を巻き起こしていた。Flashにおいては多くの外部ツールが開発され、Directorの場合は近年3DVectorGraphicSpriteが改良された。システムとしてはとても複雑でその難しさ故に学び始めることすら難しい状態である。

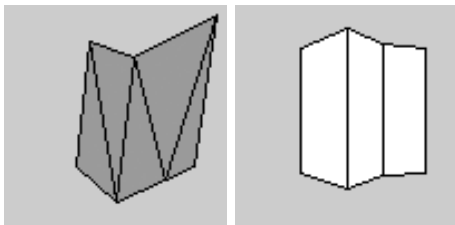
そのため、現段階のProcessingにおいて3Dとはz軸を追加しただけものとする。

```
vertex(x, y, z);
```

```
line(x1, y1, z1, x2, y2, z2);
```

```
bezierVertex(x, y, z);
```

```
curveVertex(x, y, z);
```

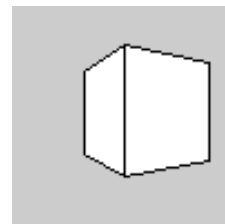


```
box(size);
```

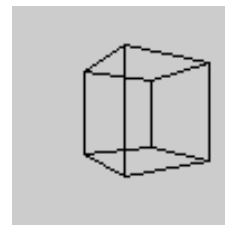
```
box(width, height, depth);
```

```
sphere(size);
```

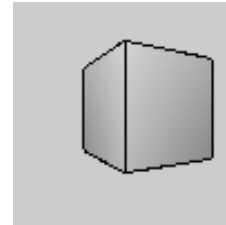
```
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



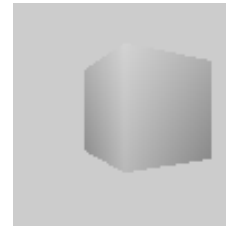
```
noFill( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



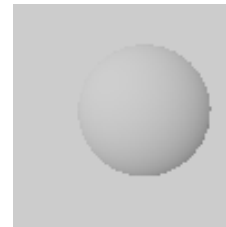
```
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
noStroke( );
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
noStroke( );
lights( );
translate(58, 48, 0);
sphere(28);
```



注意しなければならないのは`box`関数や`sphere`関数はその配置位置を指定するのではなく、動かす際は`translate`関数や`rotate`関数を使用する必要があるということである。更に`scale`関数、対になっている`push`,`pop`関数が移動させた情報等をきれいに整頓し記録してくれる。上手な描画方法については[Processing Transform Reference](#)、[Processing Transform Examples](#)を参考にして欲しい。一方でこれらの方法を利用しない3Dの描画方法も存在する。これについては[always a solution](#)を参考にしてもらいたい。

また、`lights`関数と`noLights`関数にも注意してもらいたい。`light`関数を使用することにより3Dをシェーディングした状態を得ることが出来る。光源の詳細については[Processing Lights Reference](#)を参照すること。

"何? 3Dについてはこれだけで終わり!?"

と思われるかもしれないが、3Dに対するプログラムは現在はいずれもこれだけである。将来的にはより多くの機能も期待できるだろう。しかし今現在あるだけの機能でここまで素晴らしい作品をつくり出すことも出来ることを知ってもらいたい。[Processing Software](#).

Pixels

ピクセル毎に操作を行うことはFlashでは難しい処理であった。同操作を可能とする機能`setPixel`と`getPixel`が最近Directorに追加されたばかりである。しかしながら、このDirectorの低速なアドレッシングシステムに向かうのは難しいとも考えられる。ITPの学生は[Danny Rozin](#)氏の講義である[The World - Pixel by Pixel](#)を受講し、プログラムをCを用いて作成し始めた。なぜならそれがLingoやMAXと言う選択肢よりも彼等の概念を完成させる最も高速な言語であると判明したためである。ITPの学生はDannyのC言語の講義を受講するようになった。Processingにおいてピクセルを扱う場合、Lingoを使用するよりもCのほうが遥かに簡易かつ処理が高速であった。しかしその早さはJAVAには遠く及ばなかった。

```
getPixel(x, y); // Returns an integer
setPixel(x, y, color);
pixels[index]; // Array containing the display window
```

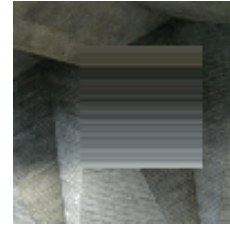


```

int width = 100;
int height = 100;
BImage b; // declare variable "b" of type BImage
b = loadImage("basel.gif");

image(b, 0, 0);
for (int i=30; i<(width-15); i++) {
  for(int j=20; j<(height-25); j++) {
    color here = getPixel(30, j);
    setPixel(i, j, here);
  }
}

```



ピクセルを制御する際ユーザは独自の関数を作成することも出来る。詳細は [transparency](#) を参照すること。

ピクセルの操作を行うサンプルとしては [Processing Image reference](#)、[Processing Image Examples](#) を参照にするのも良い。

Typography

Processingの文字レンダリングシステムは主に特有のフォントファイルフォーマットを利用している。開発者はよりフレキシブルで自由にカスタマイズできるシステムを目指していた。以前は開発スタッフのプログラマの用意と選ぶべき豊富なフォントの用意をしていた。ここに利用できるフォントの一覧が設置してある。これらのイメージはすべてビットマップ形式で保存されている。ここにテキストをスケッチに描画するととても簡単な方法がある。まずはエラーを覚悟した上で実行してもらいたい。

```

size(200,100);
hint(SMOOTH_IMAGES);
background(#FFFFFF);
fill(#000000);
BFont f = loadFont("Bodoni-Italic.vlw.gz");
setFont(f, 50);
text("handglove", 14, 60);

```



エラーはBondni-Italic.vlw.gzが見つからないと言うはずである。なぜならまだフォントのファイルをデータフォルダ上に移動させていないためである。好きなフォントを選んだ後にProcessing内の fonts フォルダを開いてもらいたい。選んだフォントをスケッチのデータフォルダにcopyすればプログラムは正常に動作するはずである。

hint(SMOOTH_IMAGES); 文字の描画を少しぼかして表示。

BFont f = loadFont("Bodoni-Italic.vlw.gz");フォントファイルを変数fに格納。

setFont(f, size); テキストの種類とその大きさを指定、テキストを描画する前に宣言。

text("handglove", x, y);テキストを指定位置に描画。

この例は回転と簡単なfor文により作成される。

```

size(200,100);
hint(SMOOTH_IMAGES);
noStroke( );
BFont f = loadFont("Univers66.vlw.gz");
setFont(f, 50);
fill(#FFFFFF);
ellipse(-50,-55,150,150);
fill(#CC6600);
for(int i=0;i<20;i++){
  rotateZ(0.2);
  text("dizzy", 90,0);
}

```

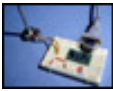


文字を描くのが嫌いなユーザにはもっと簡単に事を運ぶ方法も存在することを示しておく。Processingではテキストボックスを設けそれを用いて文字を入力することも予定している。実際にProcessingを用いて文字をモチーフに描かれた作品を挙げておく -[Processing typography examples](#)-。

Serial

シリアルポートを通じてコンピュータは手動操作される機械装置とコミュニケーションをとることができる。エレクトロニックアートの世界ではシリアルポートを利用しデバイスを利用することが知られている。Directorにおいても中継装置を用いれば可能である。しかしFlashではこのようなシリアルポートに関するサポートはなく、中継装置となるプラグインも存在しない。これらに対しProcessingはシリアルポートを使用できる環境が組み込まれている。以下の例では回転可能なつまみとProcessingスケッチとが相互に作用する。注意として、このセクションは前のセクションに対して基礎的な電気回路の知識が必要となる。

回路としては[BX-24](#), ITPにて又[他の似たような場所](#)でよく利用されているプロトタイプ集積回路を使用する。BX-24の設定に関する情報については [Tom Igoe's Physical Computing reference](#)、ITPの講義"Physical Computing"で使用された[lab assignments](#)を参照すること。



左図は今回使用した回路の写真である。他に+5v電源を使用しているが割合する。これは10K電位差、1K抵抗器、13ピンを使用している。作成方法については[ITP Intro to BX-24](#)を参照してもらいたい。

```

sub main()
  delay 0.5
  do
    debug.print cStr(getADC(13))
    delay 0.1
  loop
end sub

```

ここにつまみの回転角度をPCに返すプログラムがある。これはチップにダウンロードするためのものであり、BasicXのデバッグモニターにて数値を見れば どのような変化が生じているかを理解できる。

Processingではインターフェイス上のSketch -> Serial Portサブメニューを選び、どのシリアルポートを使用するかを選択を行う。

```
String buff = "";
int val = 0;

void setup() {
  beginSerial(19200);
}

void loop() {
  background(val,val,val);
}

void serialEvent() {
  if(serial!=10){
    buff += (char)serial;
  }else{
    buff = buff.substring(0,buff.length()-1);
    val = Integer.parseInt(buff)/4;
    buff = "";
  }
}
```

このProcessingプログラムは、つまみの回転角を調節することにより黒と白の間で背景色を変化させる。



シリアルに関する詳細については[Processing serial reference](#)を参照すること。

The Future

Processingは進化し続ける全く新しい環境であり、前述したサイトの更新を継続して確認することが大切である。新バージョンは簡単にインストール可能であり、また以前のバージョンで作成されていた sketchbookやdefaultフォルダはどのバージョンでもコピー後継続して使用することが出来る。著者がこのドキュメントを作成している時点のProcessingのバージョンはALPHA0050である。また、ここだけの話ではあるがProcessingを5を抜いたprocessingへバージョン名を変更することや、basic, standard, advancedモードの名前の変更等もあり得る。グラフィックエンジンはアンチエイリアスレンダリングを利用することにより、アルファ値の変化や多角形の塗りつぶし効果にも進化が見られるだろう。C言語を用いて作成されているFlashやDirectorとは異なり、Processingはユーザが入力する言語と同じもの(JAVA)により作成されている。将来的には音源の再生や作成をプログラムにより行うことが期待される。またtodo Listとしてインターネット上で皆とコミュニケーションをとる際にも有効となるであろう。新たにダウンロードする必要が出てくるかもしれないがTCP/IPを用いてtelnet、FTP、更にはFlashやDirectorのプログラムとも通信できるかもしれない。完成された環境開発における精神を持ちProcessingはカラーピッカー、ベジエ加工等更に多くの内部組み込みツールを追加していくことが期待される。ソフトに関する追記としてCaseyとBenはコミュニティーが広がることを期待している。集められたコードは中枢ディレクターとして確立し、Processing言語において便利なものとなっていくだろう。Processingはソースを公開している。そのため誰にでもProcessingを生成するプログラムをダウンロード、加工、コンパイルし、Processing新バージョン製作に貢献できる。

この方針より、Processingはユーザが既に習得、又は勉強中のMacromedia製品を使用する際の便利な機能を持つソフトとして利用されれば幸いである。そして作成されたProcessingのサンプルから製作側が何かを学ぶこともあるため是非出版、もしくは製作側にプログラムを送付してもらいたい。またオンライン上のProcessingサイトにあるコミュニティーでの談話に参加することも忘れないでほしい。それでは、良きクリエイションを!
Happy creating!

Josh Nimoy is a graduate student in the [Interactive Telecommunications Program](#) at New York University's Tisch School of the Arts. He creates and exhibits interactive media work concerned with vernacular digital interactivity, nature, and experimental typography systems. Nimoy values the effects of good teaching, good communication, and honest work. He holds a BA in Design and Media Arts from UCLA School of Arts and Architecture, specializing in digital cultures and technologies. Josh was a visiting undergraduate researcher at the MIT Media Laboratory in 1999 in the Aesthetics and Computation Group, led by John Maeda, where he worked with Ben Fry and Casey Reas. [website](#)

-

Benjamin Fry is a doctoral candidate at the MIT Media Laboratory. His research focuses on methods of visualizing large amounts of data from dynamic information sources. The work uses ideas from distributed and adaptive systems to form organic representations that react and respond to the input data. This work is currently directed towards Genomic Cartography which is a study into new methods to represent the data found in the human genome. At MIT, he is a member of the Aesthetics and Computation Group, led by John Maeda. Ben received an undergraduate degree from the School of Design at Carnegie Mellon University, with a major in Graphic Design and a minor in Computer Science. [website](#)

Casey Reas is an Associate Professor at the newly established Interaction Design Institute Ivrea in northern Italy. His work explores abstractions of biological and natural systems through diverse digital media including software art, digital prints, and responsive installations. In 2001, Casey received his M.S. degree in Media Arts and Sciences from the MIT Media Laboratory, where he was a member of John Maeda's Aesthetics and Computation Group (ACG). Casey has lectured and exhibited in Europe, Asia, and the United States. His work has recently been shown at the American Museum of the Moving Image, Ars Electronica, Interaction01 in Ogaki, New York Digital Salon, Museum of Modern Art, P.S.1, and Siggraph2000. [website](#)

-

-

Original tutorial : last updated October 31, 2005
日本語チュートリアル : last updated October 31, 2005