

the basic of processing

Processing Tutorial 1

INDEX

- 3 Processingとは
- 4 Processingと他言語
- 5 目標
- 6 Processingの文法
- 7 記述サンプル
- 8 基本的な書き方
- 9 Processing Referenceの使い方
- 10 変数
- 12 配列
- 14 ファンクションとサブルーチン
- 16 ループ
- 18 条件分岐
- 20 レファレンス

Processingとは

ProcessingはMITのCasey ReasとBen Fryが開発したJavaをベースとした開発環境です。そのインタラクティブ性と簡単に覚ええることからメディアアートで多用されています。このレジュメは教本の行間を埋める目的で使ってください。

何ができるのか

- ・ インタラクティブアート
- ・ 情報の可視化
- ・ ゲーム
- ・ 音の波形の操作
- ・ スワームシミュレーション
- ・ 物理シミュレーション
- ・ ハードウェアのリモート操作
- ・ ファブリケーションデータの作成
- ・ 他、・・・

長点

- ・ 簡単
- ・ クロスプラットフォーム
- ・ コミュニティが大きい
- ・ 無料
- ・ 外部アプリケーションとの連携
- ・ Javaのライブラリを使える
- ・ さまざまな入力機器が使える

欠点

- ・ 少々遅い
- ・ 日本のコミュニティが少ない

Processingと他言語

現在Processingはその学習の容易さから、さまざまな分野で使われ始めています。美術、情報処理、経済、建築等。

いわゆるプログラミングには大きくわけて二種類あります。ひとつは、実行するためにコンパイルする必要がある言語。コンパイルとは、コードを一回アプリケーションという形に組み立てることです。アプリケーションという、それ単体で動くものを作るための言語ともいえます。

もうひとつはスクリプト言語。コンパイルする必要がない言語です。書いたコードを素早く試せるため、簡単なプログラムを書くのに適しています。

プログラミング言語例：

C++、Java、Visual Basic、openFrameworks等

スクリプト言語例：

Processing、Rhinoscript、Javascript、Python、Ruby、Perl、PHP等

Processingは、簡単にいうとJavaをスクリプト言語として使えるように改造したものです。Javaのプログラミング言語を、簡単に使えるコマンドを含んだ新たな層でコーティングしていると想像すると分かりやすいかもしれません。その新たな層は翻訳機のようなものです。簡単なProcessing言語を使ってコードを書き、実行すると、そのコードはまず翻訳機であるその新たな層の中で、Java本来のプログラミング言語へと翻訳されます。そして、その翻訳されたコードがProcessingの層からもっと深いJava本来の層へと浸透します。そして、そのコードがJava本来の層へ到着したとき初めてそれを実行します。欠点として、その翻訳する一手間が、Processingの動きを鈍重にすることがあります。

似たことができるものにopenFrameworksというプログラミング言語があります。こちらはC++で直接書くプログラミング言語です。そのため、実行する時点でより深い層、C++の層からはじめるため、翻訳する手間がありません。つまり、実行速度がとても速いのです。実際、ProcessingとopenFrameworksを比べると、openFrameworksの方が10倍速いといわれています。その代わり、openFrameworksの欠点として、C++本来の書き方でコードを書かなければいけないため、書き方が難しく、敷居が高いです。ただ、Processingでの書き方に慣れれば、移行は容易であると思います。



目標

それぞれ、自分のテーマを決めてもらって、そのテーマを元にProcessingを使って表現してもらうのが、上達の早道だと思います。作っている過程のものと最終的にできあがるもの、全てをオープンソースサイトであるopenProcessingにアップロードして頂きたいと思います。出来のいいものは世界中の人からフィードバックを得られるので、非常に速いスピードで上達できます。あと、質問をするためにオフィシャルのフォーラムも積極的に使ってください。

テーマ例（これ以外のテーマでももちろんOKです）：

人工知能

ライフゲーム
Patch dynamics
Cellular Automata
Flow (video game)
Spore Prototypes

<http://ja.wikipedia.org/wiki/ライフゲーム>
http://en.wikipedia.org/wiki/Patch_dynamics
<http://www.stephenwolfram.com/publications/articles/ca/>
[http://en.wikipedia.org/wiki/Flow_\(video game\)](http://en.wikipedia.org/wiki/Flow_(video_game))
<http://www.spore.com/comm/prototypes>

情報の可視化

Data Mining
House of Cards
ReWiring
Stream Graph

<http://www.shiffman.net/teaching/a2z/mining/>
<http://code.google.com/intl/ja/creative/radiohead/>
<http://www.moma.org/interactives/exhibitions/2008/elasticmind/>
<http://leebryon.com/else/streamgraph/>

パッキング・ネスティング

Sphere Packing
2D Packing

http://en.wikipedia.org/wiki/Sphere_packing
<http://www.csc.liv.ac.uk/~epa/surveyhtml.html>

自動ペインティング

Generator X
Generative Art

<http://www.generatorx.no/>
<http://levitated.net/daily/index.html>

フラクタル（分割）

八分木
Fractal Cities
African Fractal

<http://ja.wikipedia.org/wiki/八分木>
<http://www.fractalcities.org/>
http://www.ted.com/index.php/talks/ron_eglash_on_african_fractals.html

フラクタル（炎）

Fractal Flame
Technical Paper
Electric Sheep

http://en.wikipedia.org/wiki/Fractal_flame
<http://flam3.com/index.cgi?&menu=math>
<http://www.electricsheep.org/>

スプリングネットワーク

Traer Physics
Gaudi Catenary
CatenaryCAD
Urban Grid
Graphs

<http://www.cs.princeton.edu/~traer/physics/>
<http://designexplorer.net/projectpages/cadenary.html>
http://designexplorer.net/newscreens/cadenary/final_paper.pdf
<http://eecs.oregonstate.edu/library/files/2007-41/TensorStreetModeling.pdf>
http://en.wikipedia.org/wiki/Graph_%28mathematics%29

都市シミュレーション

Complex System
Scalable City

http://www.casa.ucl.ac.uk/working_papers/paper131.pdf
<http://www.scalablecity.net/>

パスファインディング

Path Finding <http://en.wikipedia.org/wiki/Pathfinding>

Processingの文法

Processingの書き方は非常に簡単です。でも、文法は文法なので、体に書き方が染み付くまで何回も何回も書いてください。最初は本を読みながらじっくりと。そのうち空で書けるようになります。下の図は、Processingでスケッチを書きはじめる際のベースとなるものです。まずはこれから覚えてください。

The screenshot shows the Processing 1.2.1 software interface. The title bar reads "sketch_nov09a | Processing 1.2.1". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with various icons. The main workspace displays the following code:

```
void setup(){
    //set window size
    size(300,300);
}

void draw(){
    //set background color
    background(0);
}
```

The code consists of two functions: `setup()` and `draw()`. The `setup()` function sets the window size to 300x300 pixels. The `draw()` function sets the background color to black. The Processing environment is shown with its characteristic dark gray and black color scheme.

基本的な書き方

基本フレーム

基本的には、二つのサブルーチン（後で説明します）の中にプログラムを書いていきます。

ひとつは

`void setup(){}`

これは、初期起動プログラムです。Processingの実行ボタンを押したときに、どのコードよりも一番最初に、一回だけ実行するプログラムです。初期設定として使うことが多いです。{}の中に実行したいプログラムを書きます。

もうひとつは

`void draw(){}`

これは、フレームごとに毎回実行するループプログラムです。時間の経過とともに動くものなどを書きたい場合は、この中にプログラムを書きます。これも{}の中にプログラムを書きます。

```
void setup(){
    //ここにプログラムを書きます。何個改行しても構いません。
}
```

```
void draw(){
    //ここにプログラムを書きます。好きなだけ改行してください。
}
```

コメント

コードを書くときに重要なのは、自分が何を書いているのか理解することと、後で読み返して何が書いてあるのか瞬時に理解できることです。そのために、慣例として書いたコードの前にそれが一体どういった働きをするのかコメントを書く場合があります。ただ、そのコメントはプログラムコードの一部では無いということをエディタに理解させる必要があります。

その書き方は一行だけのコメントなら、コメントの直前に `//` を書きます。

複数行にまたぐ場合は、`/*` と `*/` でコメントを

はさみます。

```
//このコメントはプログラムとして認識されません
```

```
/*
このコメントは
プログラムとして
認識されません
*/
```

size()

`setup()`の中でしか起動しないコマンドがあります。それが、ウィンドウのサイズを決定する `size()` です。

使い方 (Syntax) は、

`size(width, height)`

という風になります。

`width`に幅の値を、`height`には高さの値をいれます。するとそのサイズのウィンドウが生成されます。実際に書くと、

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

となります。

行の終わり

上で書かれたコードを見ると、`size()` の後に「`;`」が来ているのが分かると思います。Processing及びJavaの文法では、行ごとに何かしらのアクションが行われます。その一つ一つのアクションの終わりには常にこのセミコロンが書かれている必要があります。例えば `size()` であると、ウィンドウのサイズを決定しているというアクションを起こしています。

ただ、{}の後には書く必要はありません。`{}`は複数のコードをラップしてひとまとめにしてますよ、という意味で、実際に何か具体的なアクションを起こしているわけではないからです。

記述サンプル

実際にどういった手順で書くのか、簡単に下の動画で追ってみてください。

The screenshot shows the Processing 1.2.1 software interface. The title bar reads "sketch_nov09a | Processing 1.2.1". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu is a toolbar with various icons. The main area is a code editor containing the following Pseudocode:

```
void setup(){
    //set window size
    size(300,300);
}

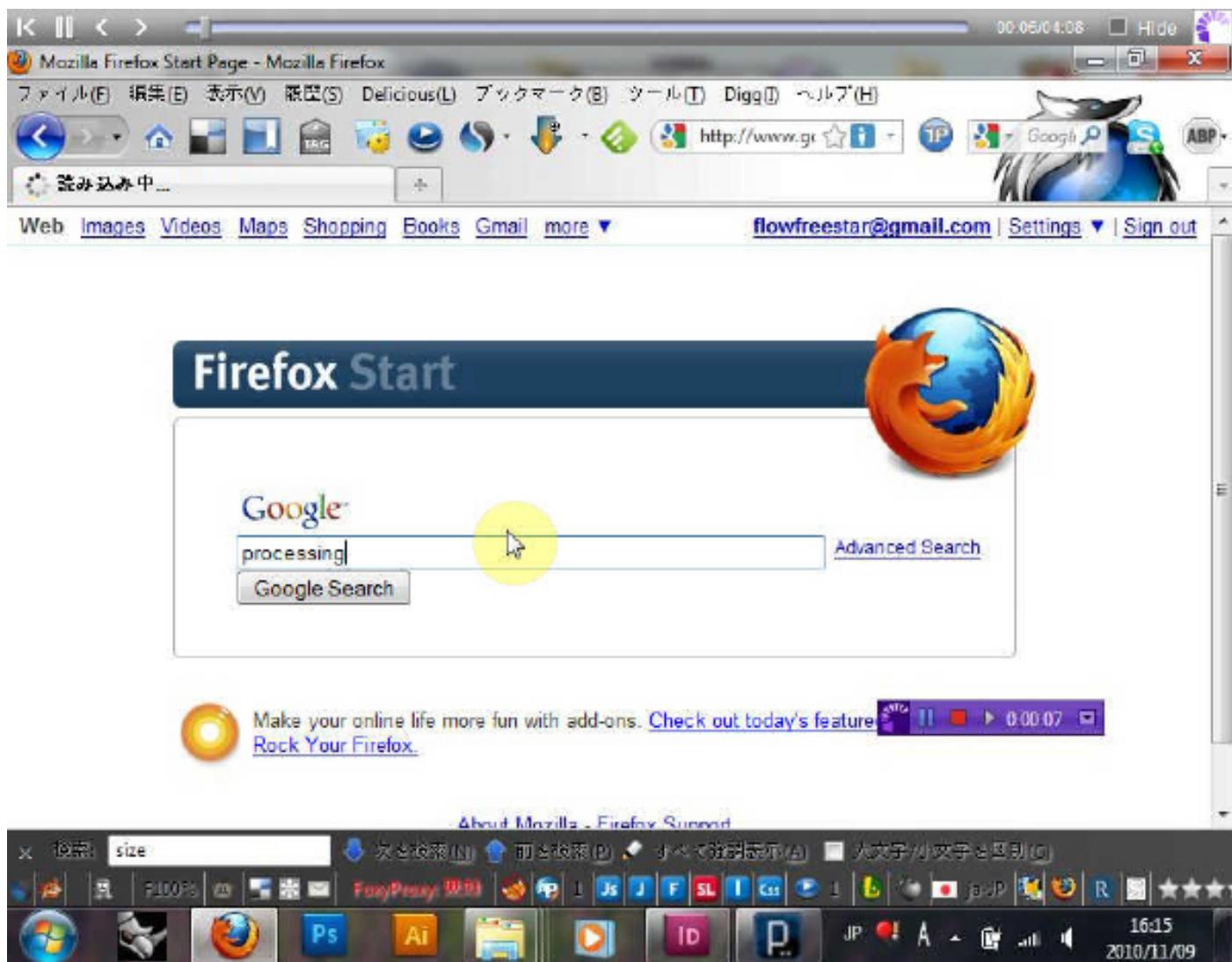
void draw(){
    //set background color
    background(0);
}
```

Processing Referenceの使い方

Processingでコードを書く際に重要なのは、まずどのようなコマンドが実装されているのかということを理解することです。そのためには、Processingのオフィシャルサイトに行って、Referenceページを見てください。そこにProcessing独自のコマンドが全てのっています。

コマンドごとにまたページが分かれています。そこにはそのコマンドの使いかたやシンタックス、サンプルが載っているので、まずはいくつかのサンプルをコピーペーストして自分で動かしてみてください。一つ一つのコマンドは小さなアクションしか起こせませんが、それらを組み合わせることで、複雑な挙動も可能となってきます。ずっと書いていれば、そのうち感覚的に、こういったエフェクトにはこのコマンドの組み合わせで、、、と想像できるようになります。まずはなれてください。

以下の動画は、レファレンスとProcessingを行き来する例です。



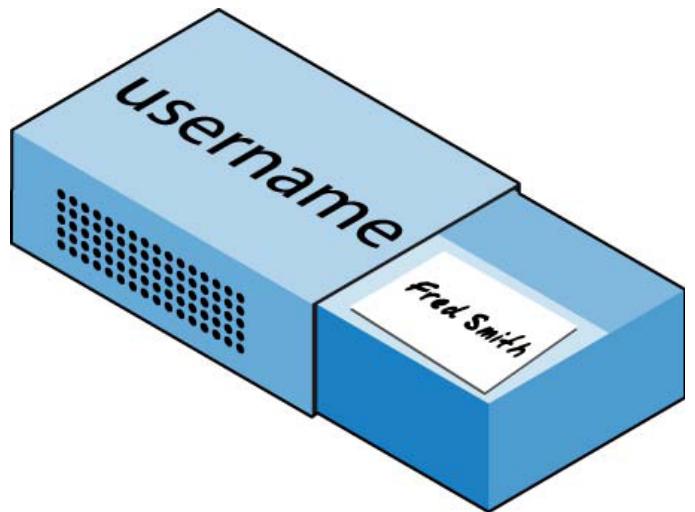
変数

ここからはProcessingに限らず、どのプログラミングにおいても重要なことを覚えてもらいます。一つは変数(Variable)と呼ばれるものです。

変数とは、なにかしらの値が入っている容器物であると考えてください。右図のような、マッチボックスを想像してもらえると分かりやすいと思います。どのマッチボックスにも名前がついていて、それが変数の名前となります。そしてその名前のついたマッチボックスに、数字やら文字列やらといったデータを入れていくのです。

よく「変数を宣言する」といいますが、それはつまり、まだ中に何も入っていない空のマッチボックスを作る、というように覚えておいてください。

ここで気をつけなければいけないのは、Processingにおいて変数を宣言するとき、最初にそのマッチボックスがどういった種類の値を入れられるものなのかということを決めておく必要があるということです。例えば、整数を入れるためのマッチボックスを作ったとします。その中には、後でどんな整数の値を入れても構いません。でも、その中には文字列や不動小数といった値を入れることはできなくなります。入れようするとエラーが出るはずです。



変数とはものを入れたマッチ箱のようなもの
(参照 : Learning PHP, MySQL & JavaScript)

具体的な宣言の仕方は以下のようになります。

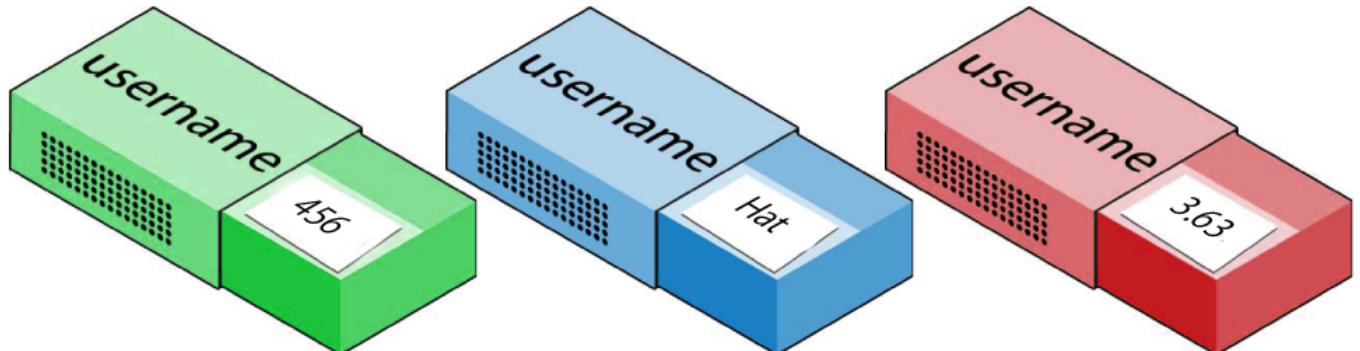
```
//seisuという名前の、整数を入れるための変数を作る
int seisu;

//fudoshosuという名前の、不動小数を入れるための変数を作る
float fudoshosu;

//mojiretsuという名前の、文字列を入れるための変数を作る
String mojiretsu;

そして、それぞれの箱の中に値を入れるに以下のように書きます。

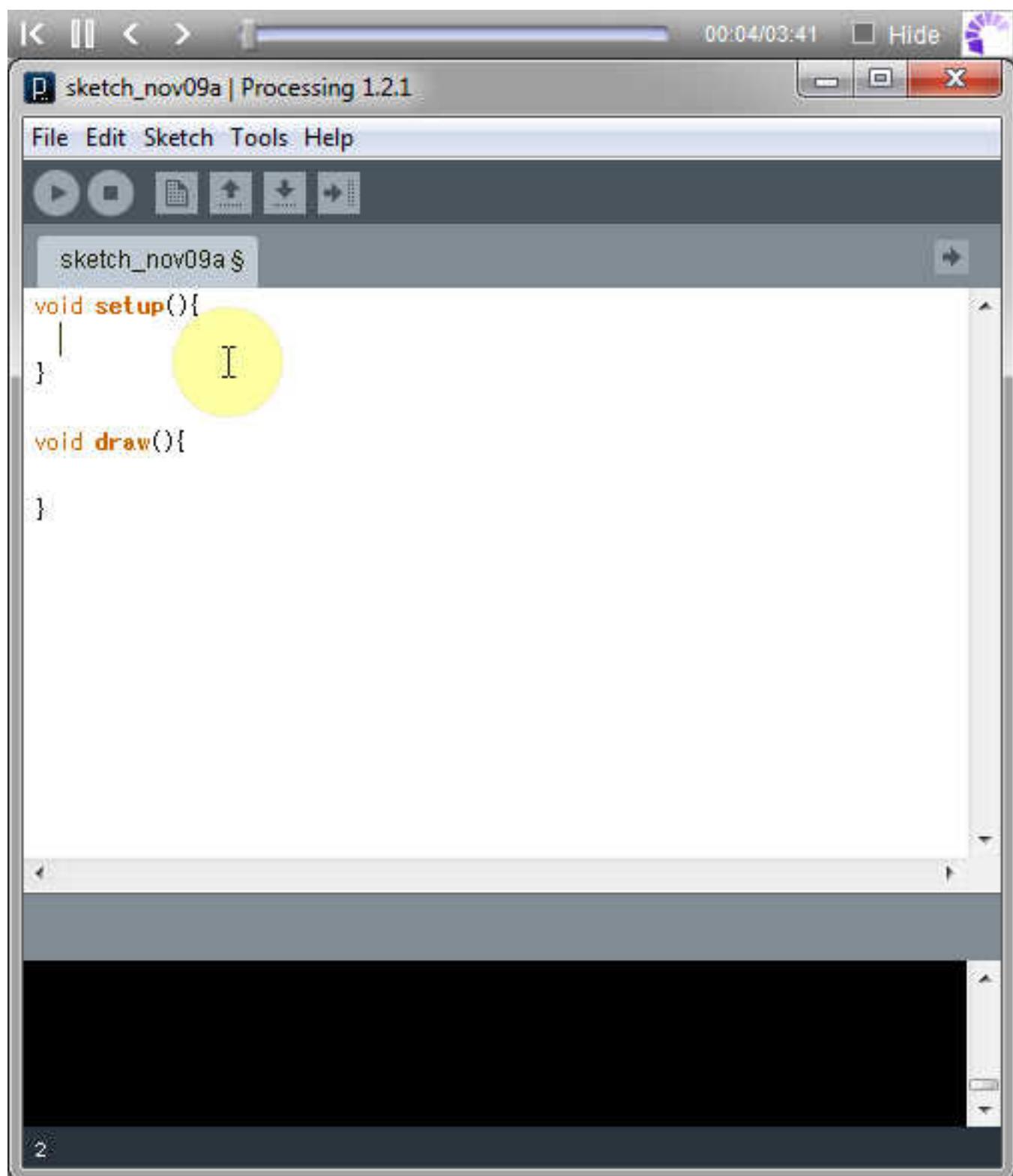
seisu = 456;
fudoshosu = 3.63;
mojiretsu = Hat;
```



中に入れるものの種類に合わせて、変数の種類を決めて作ってあげる必要がある。

変数

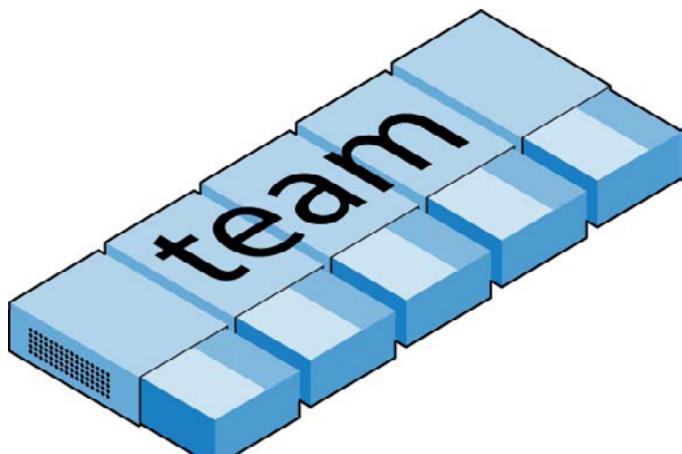
変数を宣言したり、変数の中に入りする例を示した動画です。ここでは `println()` というコマンドを使ってコマンドラインに変数の中身を表示させています。



配列

配列（Array）もまた、プログラミング全般において非常に重要なものです。できるだけ早めに使い方を覚えて活用してください。

変数がなにかしらの値が入っている容器であるとすると、配列はその容器を一まとめにしてくっつけたものと考えられます。



配列とは複数のマッチ箱をくっつけたようなもの
(参照 : Learning PHP, MySQL & JavaScript)

これは、上図のようなマッチボックスを横にくっつけたものをイメージすると分かりやすいと思います。どのマッチボックスにも名前がついていたように、マッチボックスをまとめたグループそれ自体にも名前がついています。それが配列の名前となります。

また別の言い方をすれば、複数のマッチボックスを入れることができる箱ということにもなります。

配列も変数と同じように、まず空の配列を作る（宣言する）わけですが、変数のときは名前と入れられるデータの種類だけ指定すればよかったですのに対し、配列ではそれに加えどれだけのマッチボックスを入れられるかという箱の大きさも指定してあげる必要があります。

例えば、最大で三つの変数を入れるのであれば、箱の大きさは三であると宣言する必要があります。

記述方法は、

//まずnumbersという名前の整数の配列を作り、その大きさは3であると宣言

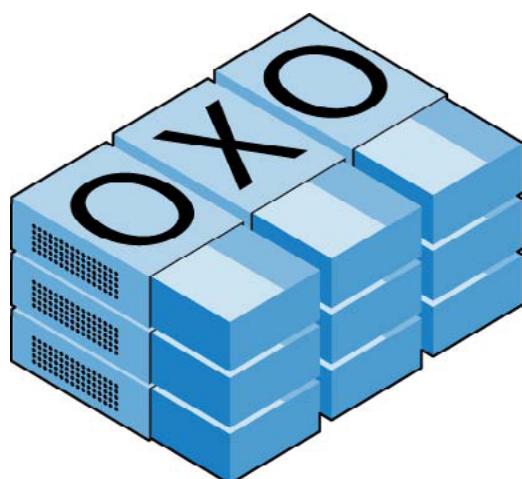
```
int[] numbers = new int[3];
```

//numbersという配列の0番目に90という値を入れる
numbers[0] = 90;

となります。

ここで応用編です。配列ですが、実は配列の中に配列をさらに入れることができます。イメージでいうと、ダンボールの中に複数の小さいダンボールがまた入っていて、それぞれの箱の中には数百個のマッチ箱が入っています。そしてそのマッチ箱それぞれに、いろいろな種類のもの（プログラミングでいうところの変数の種類）が入っています。

あるいは別の考え方として、マトリックス上に配列が並んでいると考えてもらっても大丈夫です。複数のマッチボックスをくっ付けた配列を一次配列だとすると、その配列をさらに他の配列たちとくっつけたものが二次配列、そしてそれを繰り返すと三次、四次、、、となります。



二次配列の例

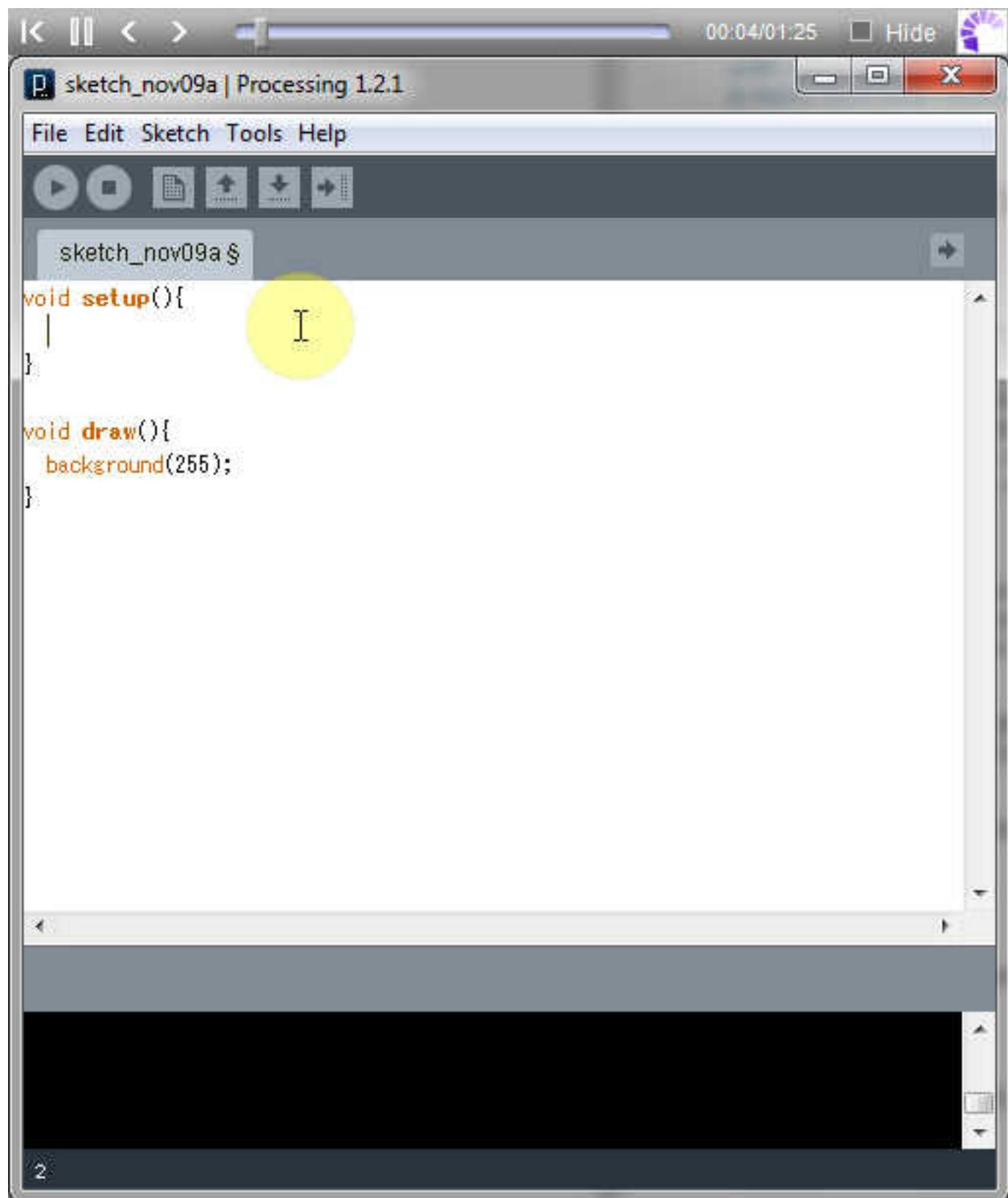
記述方法ですが、二次配列の場合

```
int[][] numbers = new int[3][2];  
numbers[0][1] = 90;
```

のように書きます。

配列

配列の基本的な使い方を示した動画です。ループと組み合わせると強力なツールとなります。あと、ArrayListと呼ばれるJava特有の配列もあります。動的配列と呼ばれるもので、今はそういうものがあるのだと覚えておいてください。



```
sketch_nov09a | Processing 1.2.1
File Edit Sketch Tools Help
sketch_nov09a $ void setup(){}
void draw(){
    background(255);
}
```

ファンクションとサブルーチン

サブルーチン

プログラミング初心者にとってとっつきにくいものの一つが、サブルーチン (subroutine) とファンクション (function) の違いだと思います。

実は、サブルーチンに関しては、サンプルプログラムを書いたときにすでに使っているはずです。

それは、

```
void setup(){}
```

と

```
void draw(){}
```

です。

最初にvoidと書かれていることに注目してください。voidからはじまるものがサブルーチンであると思ってください。

ではサブルーチンとは何であるかですが、何をするかという目的が決まっている道具であるという風に考えてもらえると分かりやすいと思います。

例えば、釘を打つためのハンマー、ねじを回すためのドライバー、木を切るためののこぎりなどのようなものです。

たとえばハンマーですが、ハンマーには釘という材料を木材に打ち付けるための目的を持っているとします。ハンマーには、この目的を達成させるためのアクションを起こすプログラムがすでに記述されているわけです（実際に人間が使い方を予想して使うわけですが、、、）

この、どういった材料を用いてどういったアクションを起こすかというものが記述されているのが、サブルーチンという道具です。

ここで重要なのは、サブルーチンという道具の特徴は、道具は道具としてのみ使われ、その道具自身を材料としては使えないという点です。ハンマー自身を、材料として他の道具を用いて利用することはできないということです。

記述方法は、

```
void hammer(){  
    //釘を打つプログラム  
}
```

となります。道具にも変数と同じように名前をつける必要があります。

ファンクション

そこでファンクションというものが出てきます。ファンクションとは、端的に言ってしまうと、道具自体が材料にもなり得る道具です。消耗型の道具とでもいうのでしょうか（あくまでイメージです）

たとえばセロハンテープを想像してみてください。これはものとものを張り合わせるという目的をもった道具です。ですが、これは同時に他の道具（はさみやカッター）などで加工できる材料にもなり得ます。

processingというプログラミング言語において、この材料とはつまり変数のことをいっています。たとえば整数という材料になる道具、不動小数という材料になる道具等々。

記述方法は変数の宣言と似ていて、

```
int scotchtape()  
float draftingtape()  
String magictape()
```

となります。{}の中にプログラムを記述するわけです。具体的な使い方は後ほど。

ファンクションとサブルーチン

サブルーチンとファンクションの基本的な使い方を示した動画です。

The screenshot shows the Processing 1.2.1 software interface. The title bar reads "sketch_nov09a | Processing 1.2.1". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with various icons. The main code editor window contains the following Pseudocode:

```
//here are
void setup(){
    size(300,300);
    smooth();
}

void draw(){
    background(200);
}
```

ループ

ループもまたプログラミングにおいて重要な概念です。これから書くほぼ全てのプログラムで、このループを使うことになると思います。

プログラミングにおけるループをイメージする際に、下の図を常に思い浮かべるようにすると分かりやすいと思います。

プログラムを走らせて、今どの辺を走っているのかを車で表したとします。車はあるところでロータリーに入ります。このロータリー一周分がループされるプログラムの流れであると考えてください。

その車は、ある一定の条件を満たすまでずっとロータリーを周り続けます。つまり、ずっと同じプログラムが実行されるわけです。そしてある条件が満たされると、車はロータリーを抜け、次のコードへと向かいます。

Processingには大きく二種類のループが用意されています。一つはforループ、そしてもう一つはwhileループです。

forループ

forループの記述方法は以下のようにになります。

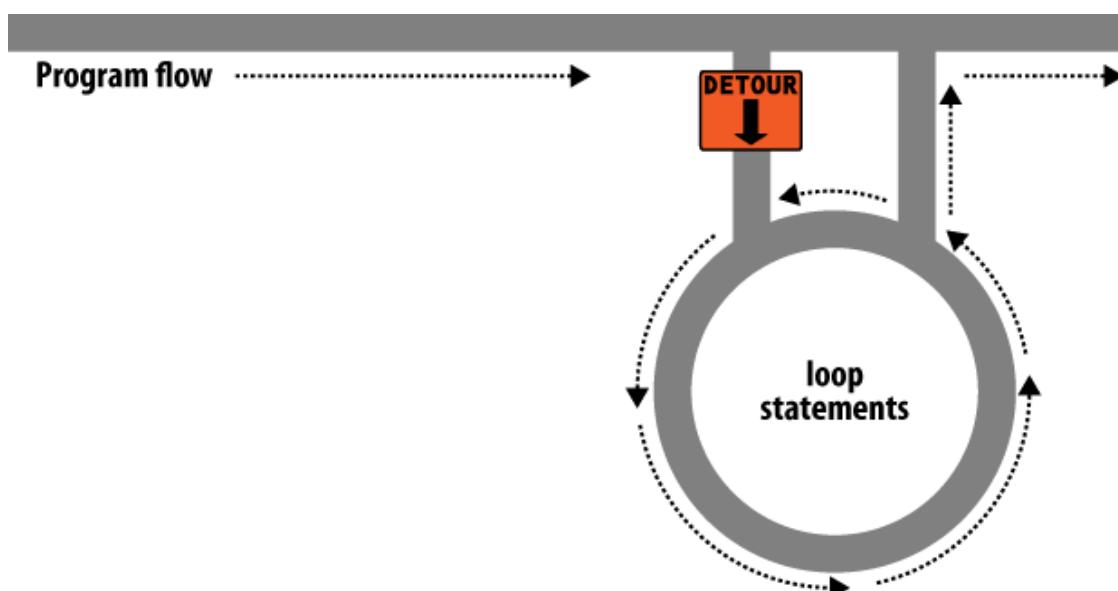
```
//何回ループさせたいか  
int num = 10;  
for(int i = 0; i < num; i++){  
    //ループさせたいプログラム  
}
```

forの括弧の中には、最初0という値を設定した変数iに、ループするごとに1ずつ足していく、その値が10になったときにループから出るという条件が書いてあります。これが基本的な書き方なので、まずはこれを暗記するまで自分で書いてください。

Whileループ

whileループは、forよりも単純な条件式を使います。下手するとループから出られず、プログラムがフリーズする可能性もあるので気をつけてください。

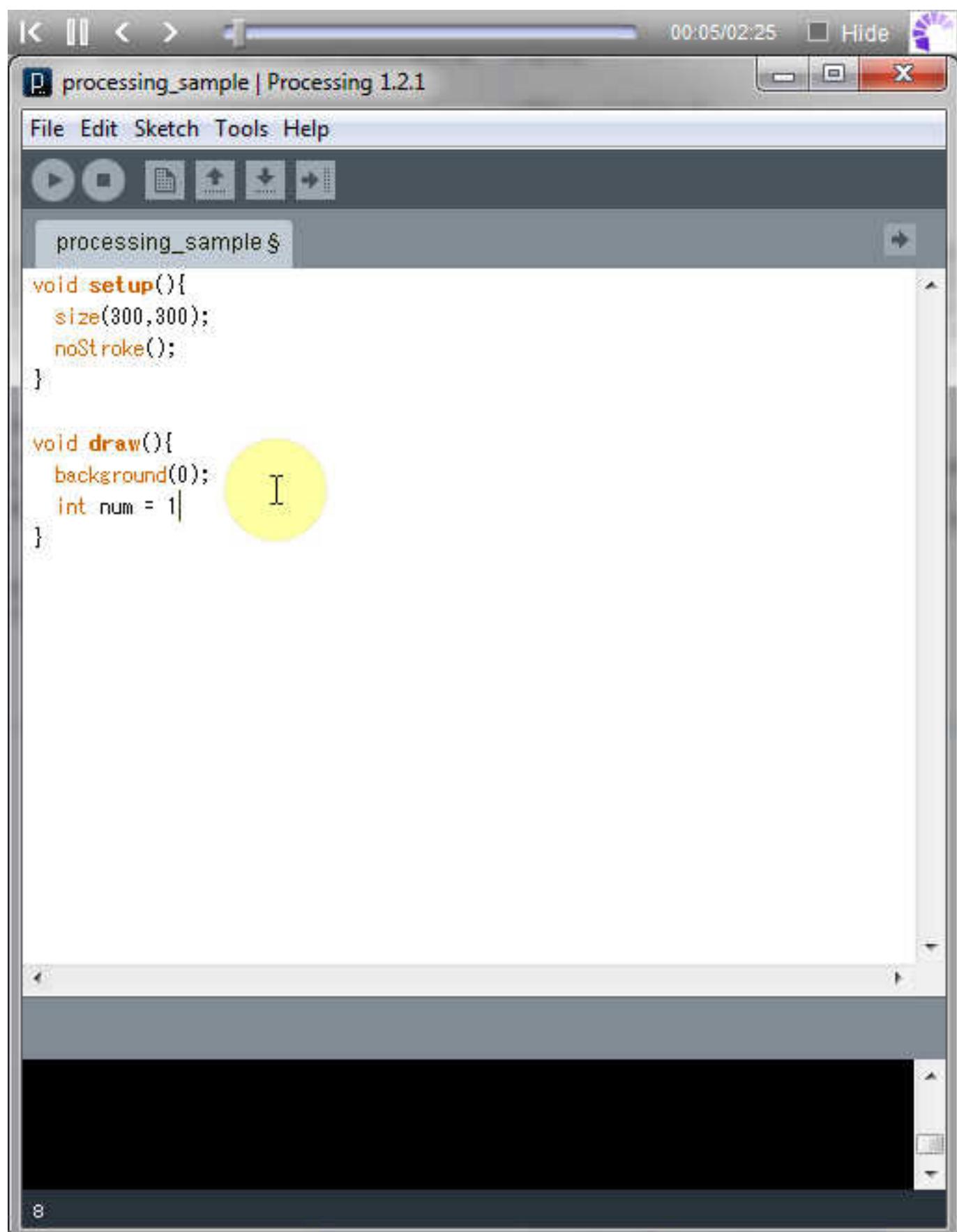
```
int i = 0;  
while(i<80){  
    //ループさせたいプログラム  
}
```



ループするコマンドとは、ロータリーを何周もする車のよう
(参照 : Learning PHP, MySQL & JavaScript)

ループ

ループの基本的な使い方を示した動画です。



```
P processing_sample | Processing 1.2.1
File Edit Sketch Tools Help
processing_sample §
void setup(){
    size(300,300);
    noStroke();
}

void draw(){
    background(0);
    int num = 1
}
```

条件分岐

条件分岐という方法も、プログラミングにおいて非常に重要なものです。ループと一緒に使い方を覚えてください。

条件分岐とは、ある条件が満たされたときに、通る道のことです。「もし・・・だったら、・・・をする」ということです。

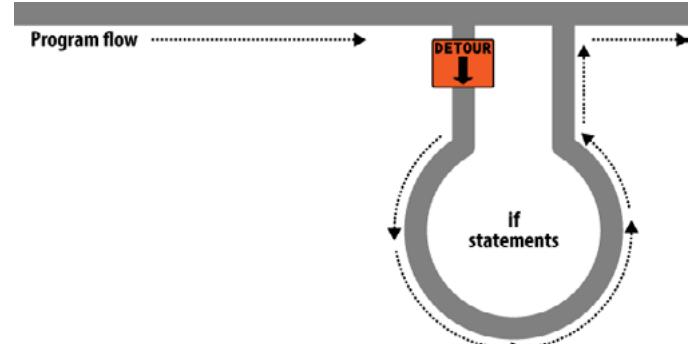
書き方は、

```
int i = 10;  
if(i < 20){  
    //条件が満たされたときに走らせるプログラム  
}else if(i<50){  
    //最初の条件が満たされず、今回の条件が満たされたとき走らせるプログラム  
}  
else{  
    //条件が満たされなかった時に走らせるプログラム  
}
```

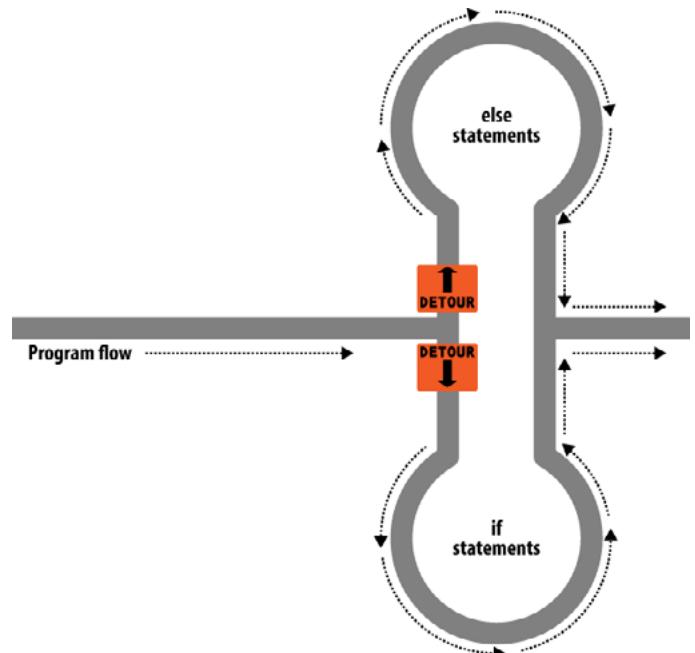
となります。

基本的にifで書き始め、()の中に条件を書きます。そして条件が満たされたときに実行するプログラムを{}の中に書きます。elseは、条件のそれ以外という意味で使います。

これらが基本的なプログラミング作法です。ここまで書いたことはどんなに小さなプログラムを書くにしても使うことなので、絶対に覚えておいてください。



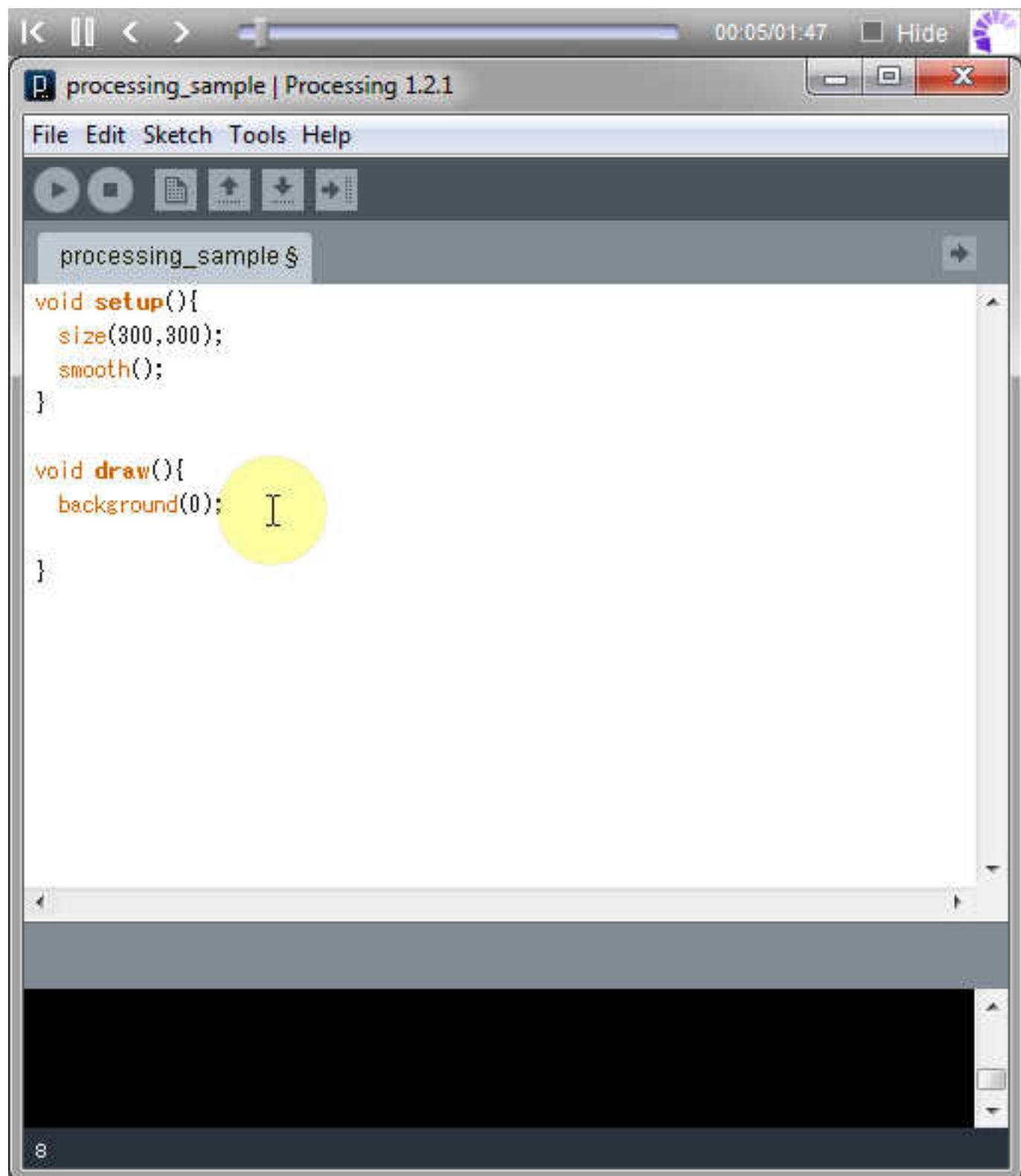
条件が満たされると、寄り道をする。満たされないとスキップ
(参照 : Learning PHP, MySQL & JavaScript)



条件が満たされると、寄り道をする。それ以外は他の道。
(参照 : Learning PHP, MySQL & JavaScript)

条件分岐

条件分岐の基本的な使い方を示した動画です。



```
processing_sample.p | Processing 1.2.1
File Edit Sketch Tools Help
processing_sample.p
void setup(){
  size(300,300);
  smooth();
}

void draw(){
  background(0);  [Yellow circle highlights this brace]
}
```

レファレンス

書籍

田中孝太郎・前川峻志「Built with Processing」
Processingの基礎が学べる日本の書籍。

Reas・Fry 「Processing a programming handbook for visual designers and artists」
Processing開発者が書いた本。コマンドごとに、チュートリアル形式で学べる。

Ben Fry 「ビジュアライジング・データ」
和訳された、情報をProcessingで可視化する方法を書いた本。

Reas・Fry 「Getting Started with Processing」
Processing開発者が書いた初心者のための本。英語。

ネット

Processing.org. <http://processing.org/>
オフィシャルサイト。ここでアプリケーションを手に入れる。

OpenProcessing <http://www.openprocessing.org/>
世界中の人がアップロードしたProcessingスケッチがあるオープンソースサイト。

Ben Fry <http://benfry.com/>
開発者のサイト。情報の視覚化の例が多くある。

REAS <http://www.reas.com/>
開発者のサイト。ここもprocessingの例が多くある。

daniel shiffman <http://www.shiffman.net/>
Processing関係の本を執筆している人のブログ。最新情報あり。

Processing Forum <http://forum.processing.org/>
オフィシャルフォーラム。ここで分からぬことを聞く。