情報メディア基盤ユニット用資料（2014 年 4 月 22 日分）

Processing 言語による情報メディア入門

変数、setup と draw

神奈川工科大学情報メディア学科　佐藤尚

プログラムとは？

一般には学芸会などの各種行事の進行表をプログラムと呼ぶことがあります。また、テレビ番組のことは、英語で ”TV program” と呼んでいます。つまり、プログラム（program）とは、1) あらかじめ決められている、2) ある順番やタイミングで行う処理（操作、作業）のことと考えることができます。つまり、プログラムを作るとは、自分以外のものに対して、自分が意図した動作を行うようにすることです。この ” 自分の意図した動作 ” のことがプログラムです。特に、コンピュータに対してプログラムを作成することをプログラミング（programming）と呼んでいます。

"What Most Schools Don't Teach" というビデオがあるので、是非見て下さい。URI は https://www.youtube.com/watch?v=nKIu9yen5nc です。

コンピュータは人間（プログラマ）の意図した動作を馬鹿正直に実行しようとします。ですので、プログラミングを学ぶことで、

1. 論理的な考え方
2. 論理的に情報を分析し、利用する方法
3. 正確に、なるべく早く処理をこなす工夫を考える力

などを身につけることが出来ます。また、グループで作業をする機会も多いので、他人と知識を共有する方法や議論を行い結果をまとめる力などもに見つけることが出来ます。

コンピュータのプログラミングとは？

この辺りの話は、IT 基礎で扱われる筈です。

コンピュータは、機械語と呼ばれるコンピュータに固有の命令のみを実行することが出来ます。しかし、機械語は整数値で表されている命令のため、機械語でプログラムを作成することは、非常に困難です。そこで、機械語の命令に人間がわかりやすい名前を有り当てたアセンブラを作成することが行われています。しかし、アセンブラでもプログラムを作成することはかなり面倒です。さらに、コンピュータ毎に機械語の命令は異なっているので、一つのプログラムで、別な種類のコンピュータでも動作させることの出来るプログラムを、機械語やアセンブラで記述することは絶望的に難しい（不可能な）作業です。このため、機械語やアセンブラは低水準言語 (low level language) と呼ばれることがあります。

この「わかりやすい」名前のことをニーモニック (mnemonic) と呼んでいます。

低水準言語という言葉あるということは、高水準言語と呼ばれるものも存在しています。高水準言語は、人間にとってわかりやすく命令を記述できるようになっています。また、一つのプログラムで、別な種類のコンピュータでも動作させることの出来るプログラムを

例えば、マイクロソフトが販売している Surface RT は見た目は、Window 8 です。しかし、Tegra3 というプロセッサを使っているので、一般的な Window8 用のプログラムを実行させることは出来ません。

作ることが容易となっています。高水準言語で作成した命令列のことをソースコード (source code) と呼ぶことがあります。

ソースコードを用いてコンピュータに処理を行わせるためには、通常、コンパイル（compile）またはインタープリット（interpret）という処理を行います。コンパイルは、作成したプログラムを機械語に変換し、その結果をファイルに保存します。こうして作成されたファイルは、オブジェクトコード (object code) や実行形式（executable）と呼ばれます。実行形式のファイルをコンピュータのメモリに読み込み、実行することで処理を行って行きます。なお、ソースコードを機械語に変換するプログラムのことをコンパイラ (compiler) と呼んでいます。これに対してインタープリットでは、ソースコードの命令を一つずつ読み込み、その命令に対応した処理をインタープリンタ (interpreter) というプログラムに実行させることが処理を進めて行きます。こうして出来たプログラムをコンピュータで実行させることで処理を行っています。

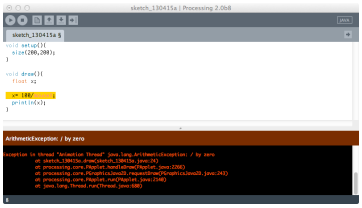
この授業で使っている Processing 言語はちょっと代わった方法を使って、ソースコードで書かれた処理を実行しています。まず、ソースコードをバイトコード (byte code) と呼ばれる仮想的な機械語に変換します。そのバイトコードを仮想機械 (virtual machine) と呼ばれるインタープリンタが読み込み、実行をしていきます。バイトコード呼ばれる命令はシンプルな命令なので、インタープリットが高速に行うことができます。この仮想機械は一般的なインタープリンタよりも単純な構造になっているので、作成が容易です。

高水準言語の例とは、C 言語、C++ 言語、Java 言語、Perl、Ruby,PHP, javascript,Common Lisp、Haskell, Prolog など色々な種類のものがあります。

バグとデバッグ

作成したプログラムが意図した通りに実行されない場合が多々あります。このようなプログラムの誤りのことをバグ (bug) と呼びます。バグを修正する作業のことをデバッグ（debug）と呼びます。バグには次の 3 つのものがあります。

1 つ目は、構文エラー (syntax error) と呼ばれるものです。ソースコードに記述される命令には、厳密な文法が決められています。構文エラーが見つかると、コンパイラやインタープリンタは処理を停止してしまいます。日常的に使っている日本語などは文法のミスや書き間違い対して、非常に寛容です。少しぐらい誤りがあっても、文の意味を取ることができます。しかし、コンパイラやインタープリンタは、この種のミスに対して、非常に不寛容です。一般的に言って、3 種類のバグの中では一番ミスを発見しやすいエラーです。
Processing 言語では、構文エラーが発生した場合には、前回のプリントで説明した場所にメッセージが表示されます。



2 つ目は、実行時エラー（run-time error）です。これは、プログラムの実行時に何らかの不都合が発生したことを意味するエラーです。例えば、0 で割り算をしたなどの場合に発生します。Processing

言語では、実行時エラーが発生した場合には、ウインドウの下部に実行時エラーの種類などに関するメッセーが表示され、実行時エラーが発生した場所がハイライトで示されます。

３つ目は、論理エラーです。論理エラーを持ったプログラムでは、構文エラーも実行時エラーも発生しません。しかし、作成したプログラムが意図通りに動作しないというものです。論理エラーの原因を突き止めることは、かなり面倒な作業です。自分の意図した動作を行わせるための手順（アルゴリズム）を考え直したり、プログラムの動作を見直したりなどの作業が必要となる場合もあります。一般的に言って、3 種類のバグの中では一番やっかいなものです。

変数

Processing 言語に限らず、コンピュータでプログラムを作るときは、変数 (variable) という考え方が出てきます。変数はコンピュータのメモリに名前をつけて、その場所に値を保存したり、読み出したりして利用します。変数には名前をつけて区別します。変数につけた名前のことを変数名と呼びます。

variable とは、どんな意味かわかりますか？

Processing 言語では、変数を使う際にどのような種類のデータを保存するのかを指定する必要があります。コンピュータの中では、全ての情報が数値（整数値）に置き換えて、記憶されています。従って、どんな種類のデータかの情報がないと、うまく情報を読み出すことや保存することが出来ません。変数にしまうデータの種類をデータ型 (data type) と呼びます。

連続量（アナログ量）を、整数値のような飛び飛びの量（デジタル量）に変換することを量子化やデジタル化と呼びます。連続的に変化している量を一定の間隔をおいて測定することを標本化やサンプリングと呼びます。この辺の詳しい話は、「情報理論とデジタル信号処理」で学習します。

Processing 言語では以下のような種類のデータ (Primitive data types) を扱うことが出来ます（一部ですが)。

Processing 言語で使用できる代表的なデータ型

データ型名	説明
boolean	true と false という 2 つの状態を表すのに使用します。
char	‘a’ や ’b’ などのような一文字（CHARacter）を表します。
String	“riho” などような文字列を表す。文字列のはじめと終わりを " で囲って表します。
int	-2147483648 から 2147483647 の範囲の整数 (INTeger) を表すときに使用します。
float	3.14159 のような実数を表すときに使用します。
PImage	jpeg や png 形式の画像情報を Processing 言語の中で利用するときに使用します。
PFont	ウィンドウに表示する文字の形状情報を保存するときに利用します。

float 型のような実数（小数点付きの数や int 型では表せない範囲の数値など）は、内部では浮動小数点形式と呼ばれる方法で数値データを記憶しています。詳しくは、IT 基礎の教科書を見て下さい。

プログラムを作る人が、自分なりのデータ型を新たに作り出すことも出来ます。

クラスと呼ばれている仕組みです。

変数の使用例その 1：同じ値の再利用

プログラムの中に同じ数字が出てくることがあります。これは偶然同じ値が出てくることもありますが、何らかの理由があって同じに値になっていることがあります。それをハッキリさせたときには、変数を使うと便利です。

Processing 言語で変数を利用する際には、どのようなデータ型のどんな名前の変数（変数名）にするかを決めて、Processing に伝える必要があります。これを変数宣言と呼んでいます。変数宣言は、次のような形になります。

変数宣言の形式
データ型　変数名 ;

変数名は、アルファベットまたは _（アンダースコアー）または $ で始まり、アルファベット、数字、_、$ を組み合わせて作られる単語です。ただし、_ で始まる変数名は特別な用途で使用されることがあります。そのため、_ で始まる _test などのような変数名は使用しないことをおすすめします。

高校で物理を勉強した人は、より

$$4.9t^2$$

の方がわかり易いですね。これ

$$\frac{1}{2}gt^2$$

も、変数 (g) の使用例です。

変数には名前とデータ型が絶対に必要です。

命令文の時と同じで、最後には ; を置きます。

単語と言っても、英単語である必要ありません。ただし、予約語と呼ばれている単語は、変数名として使用することは出来ません。例えば、int,float,string,draw,setup などの単語は予約語なので、変数名として、使用できません。簡単に言うと、予約語とは Processing 言語が使うことになっている単語です。

当然、大文字と小文字は区別しますので、test と Test は異なる変数名です。

declare の意味はわかりますか？

変数を使用プログラム例その 1 サンプル 2-01

```
size(480,120);
smooth();

int y; // Declare y as an int value
y = 60; // Assign a value to y
int d; // Declare d as an int value
d = 80; // Assign a value to d

ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
```

一般的に、変数に値を保存するためには、= を利用して、

変数名 = 値 ;

のように書きます。変数に値を保存することを代入するということもあります。　変数の代入には、= 記号を使用しますが、数学での = とは少し役割が異なることに注意して下さい。変数の中に保存されている値を読み出す（取り出す）ためには、変数名を書けば OK です。

「代入するを」英語で言うと、assign です。

変数を使用プログラム例その 2 サンプル 2-02

```
size(480,120);
smooth();

int y; // Declare y as an int value
y = 100; // Assign a value to y
```

ちょっとわかりづらいですが、サンプル 2-01 とは、変数 y と変数 d に代入している値が異なっています。

```
int d; // Declare d as an int value
d = 130; // Assign a value to d

ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
```

プログラムの中で、変数の値を変更することも出来ます。

変数を使用プログラム例その 3 サンプル 2-03

```
int d; // Declare d as an int value
d = 80; // Assign a value to d

ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
y = 180;
ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
```

ここで、変数 y の値を変えています。

サンプル 2-01 では、変数 y のデータ型は int 型として宣言をしています。従って、サンプル 2-04 のように、実数（小数点付きの数）を代入しようとすると、エラーとなります。

変数を使用プログラム例その 4 サンプル 2-04

```
size(480,120);
smooth();
int y; // Declare y as an int value
y = 60.5; // Assign a value to y
int d; // Declare d as an int value
d = 80; // Assign a value to d
ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
```

ここで、変数 y に 60.5 という数値を代入していますが、変数 y のデータ型は int 型(整数）なので、エラーとなります。

数値の値としては、60 と 60.0 は同じ値ですが、60 は整数 (int 型)、60.0 は小数点付きの数（float 型）なので、次のような場合にもエラーとなります。

変数を使用プログラム例その 5 サンプル 2-05

```
size(480,120);
smooth();
```

```
int y; // Declare y as an int value
y = 60.0; // Assign a value to y
int d; // Declare d as an int value
d = 80; // Assign a value to d
ellipse(75,y,d,d); // Left
ellipse(175,y,d,d); // Middle
ellipse(275,y,d,d); // Right
```

ここで、変数 y に 60.0 という数値を代入していますが、数値の値としては 60 と同じなのですが、小数点がついているため、実数と判断されて、エラーとなります。

変数の使用例その 2：簡単な計算を利用

数値を保存している変数の場合には、簡単な計算式を利用することが出来ます。例えば、サンプル 2-01 を下のように変更するとプログラムの意図がよりハッキリします。

変数を使用したプログラム例その 6 サンプル 2-06

```
size(480,120);
smooth();

int y; // Declare y as an int value
y = 60; // Assign a value to y
int d; // Declare d as an int value
d = 80; // Assign a value to d
int x; // Declare x as an int value
x = 75;

ellipse(x,y,d,d); // Left
x = x+100;
ellipse(x,y,d,d); // Middle
x = x+100;
ellipse(x,y,d,d); // Right
```

真ん中の円の中心は、左の円の中心より X 軸方向に 100 だけ移動した場所に表示します。右の円の中心は、真ん中の円の中心より X 軸方向に 100 だけ移動した場所に表示します。

計算式を作る際には、次の表のような演算子が使用できます。

演算子のことを、英語では operator と呼びます。

Processing での演算子

演算子	意味	演算子	意味
+	足し算	*	かけ算
-	引き算	/	割り算
=	代入	%	剰余

割り算をしたときの、余りを求める計算のことを剰余を求める呼びます。剰余のことを英語では、modulo と呼びます。

変数を使用プログラム例その 7 サンプル 2-07

```
size(480,120);
smooth();
int y; // Declare y as an int value
y = 60; // Assign a value to y
```

```
int d; // Declare d as an int value
d = 80; // Assign a value to d
int x; // Declare x as an int value
x = 75;

ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
```

サンプル 2-6 より、描く円の数が増えただけです。

このサンプル 2-07 は、次のように書き換えることが出来ます。

変数を使用プログラム例その 8 サンプル 2-08

```
size(480,120);
smooth();

int y = 60; // Declare y as an int value and assign a value to y
int d = 80; // Declare d as an int value and assign a value to d
int x = 75; // Declare x as an int value and assign a value to d

ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
x = x+100;
ellipse(x,y,d,d);
```

このサンプルのように、変数の宣言と、最初の値の代入は同時に行うことが出来ます。最初の値を代入することを「初期化する (initialize)」と呼びます。

ただし、次の使い方はエラーとなります。

```
int x; // Declare x as an int variable
int x = 12; // ERROR!! Can't two variables called x here
```

同じ名前を同時に使おうとすると、混乱しますよね。AKB の大島さんでは、大島優子さんか大島涼花さんかの区別が出来ないですよね。

もう一つ変数を利用したサンプルを示します。

変数を使用プログラム例その 9 サンプル 2-09

```
size(480,120);
int x = 25;
int h = 20;
int y = 25;
rect(x,y,300,h); // Top
x = x + 100;
rect(x,y+h,300,h); // Middle
x = x -250;
rect(x,y+2*h,300,h); // Bottom
```

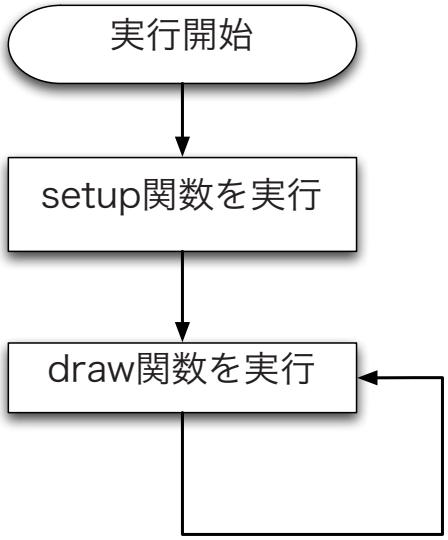
setup と draw

今までは、静止画の表示のみを行ってきましたが、動きのある画像を作り出すことも出来ます。そのために、必要となるが setup 関数と draw 関数です。

静止画を表示するだけであれば、単純に上から順番に Processing 言語の命令を実行すれば、表示が行えます。ところが、動きのある画像を表示するためには、短い時間に少しずつ異なった画像を表示する必要があります。映画では毎秒 24 枚の画像を表示していますし、テレビゲームなどでは毎秒 60 枚程度の画像を表示しています。人間は短い時間に少しずつ動いている画像を見ると、なめらかに動いているように感じます。

仮現現象と呼ばれています。

Processing 言語で動きのある画像を表示する際には、はじめに 1 度だけ実行すればよい命令と、画像を表示するために何度も繰り返し実行する必要のある命令に分けることができます。例えば、size 関数ははじめに 1 度だけ実行すれば OK です。そこで、この区別を Processing 言語に知らせる役割を担っているのが、setup 関数と draw 関数です。setup 関数と draw 関数を含んだ Processing 言語で書かれたプログラムが実行される際には、右図のような形で命令の実行が進んでいきます。



変数の初期化と同じに様に、最初に 1 回だけ実行される処理のことを初期化処理と呼びます。「初期化処理を、setup 関数で行う」というような使い方をします。

システム変数

Processing 言語にはシステム変数と呼ばれる、宣言をすることで使用できる変数が用意されています。代表的なものを表にまとめておきます。このシステム変数は値を読み出すことは出来ますが、プログラムの作成者が値を変更することは出来ません。

システム変数は、プログラムの作成者が値を変更することが出来ないので、変数と呼ぶことには、少し違和感があるかも知れません。

代表的なシステム変数

変数名	変数が保持している値の意味
width	表示しているウインドウの横幅。
height	表示しているウインドウの縦幅。
mouseX	現在のカーソル位置の X 座標の値。
mouseY	現在のカーソル位置の Y 座標の値。
pmouseX	直前のカーソル位置の X 座標の値。
pmouseY	直前のカーソル位置の Y 座標の値。
frameCount	何回目かの描画かを記録している変数。
key	どのキーが押されたかを記憶している変数。

pmouseX とか pmouseY の p は previous の頭文字の p だと思います。所で、previous の意味は大丈夫ですか？

変数名	変数が保持している値の意味
keyPressed	キーが押されているかどうかを示す変数、true か false。
mousePressed	マウスボタンが押されているかどうかを示す変数、true か false。
mouseButton	どのマウスボタンが押されているかを示す変数。RIGHT,CENTER,LEFT のどれかの値となります。

つまり、システム変数 keyPressed, mousePressed, mouseButton は boolean 型変数です。

このシステム変数を利用したプログラムのサンプルを示します。このプログラムは、ウィンドウの中央を中心とする直径がウィンドウの横幅の 4 分の 1 の円を描くプログラムです。

システム変数を使用プログラム例その 1 サンプル 2-10

```
int diameter;  // 直径

void setup(){
  size(400,400);
  smooth();
  diameter = width/4;
}
void draw(){
  background(255);
  fill(150);
  ellipse(width/2,height/2,diameter,diameter);
}
```

diameter の意味がわかりますか？

{ と } で囲まれている部分が一つの塊（ブロック）を作っています。

このプログラムはサンプル 2-11 のように書いても同じ実行結果となります。

サンプル 2-11

```
int x = 200;          // 円の中心の X 座標値
int y = 200;          // 円の中心の Y 座標値
int diameter = 100;
void setup(){
  size(400,400);
  smooth();
}
void draw(){
  background(255);
  fill(150);
  ellipse(x,y,diameter,diameter);
}
```

サンプル 2-10 とサンプル 2-11 の「size(400,400);」の数字を色々変更して、プログラムを実行してみて下さい。違いがわかりますか？

しかし、サンプル 2-10 の方が変更に強い（？）プログラムとなっています。

システム変数と setup&draw を組み合わせると、簡単な対話的（？）なプログラムを作成することが出来ます。下のサンプルは点

(mouseX,mouseY) を中心に、直径 80(=diameter) の円を描画するプログラムです。draw 関数の部分は、定期的に呼び出され、draw 関数内部に書かれている命令が実行されます。実行される度に、mouseX や mouseY の値は異なる（マウスカーソルが動いていれば）ので、その度に円が描かれる位置が変わります。そのため、動いているように見えます。

システム変数を使用プログラム例その 2 サンプル 2-12
mouseX と mouseY

ウィンドウからはみ出してしまう部分は描画されません。

```
int diameter = 80;

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

void draw(){
  background(255);
  fill(150);
  ellipse(mouseX,mouseY,diameter,diameter);
}
```

ところで、サンプル 2-12 を次の様に書き換えるとどのような動作になるでしょうか？また、なぜこのような動作になるかわかりますか？

サンプル 2-13

サンプル 2-12 とサンプル 2-13 では、「background(255);」の場所が異なっています。background はどのような動作をするかを思い出して下さい。

所で、「background(255);」は「background(255, 255, 255);」と同じ意味です。大丈夫ですね？

```
int diameter = 80;
void setup(){
  size(400,400);
  smooth();
  background(255);
}
void draw(){
  fill(150);
  ellipse(mouseX,mouseY,diameter,diameter);
}
```

もう一つ別なサンプルを示します。今度は、mouseX と mouseY だけでなく、pmouseX と pmouseY というシステム変数を利用しています。

システム変数を使用プログラム例その 3 サンプル 2-14
mouseX,mouseY,pmouseX,pmouseY

```
void setup(){
  size(400,400);
  stroke(255,0,0);
}
```

```
void draw(){
  background(255);
  line(pmouseX,pmouseY,mouseX,mouseY);
}
```

このプログラムを下のように書き換えると、どのような動作になるかわかりますね。

サンプル 2-15

```
void setup(){
  size(400,400);
  stroke(255,0,0);
  background(255);
}
void draw(){
  line(pmouseX,pmouseY,mouseX,mouseY);
}
```

物体が移動する簡単なプログラム

少しずつ表示する位置を変えながら、描画を行うことで、アニメーションを表示することが出来ます。サンプル 2-12 では、マウスカーソルを動かすことで、アニメーションのような表示を実現していました。「マウスカーソルを動かす」と同じようなことをプログラムで実現できれば良いわけです。ここでは、とても簡単なサンプルを示します。

次のサンプルでは、幾つかの円を表示するものです。

サンプル 2-16

```
int d = 20;// 円の直径
int y = 0;// 円の中心の Y 座標
int dy = 40;// 一度の移動量
size(100,200);
background(255);
fill(0,0,255);
smooth();
int x = width/2;/ 円の中心の X 座標
ellipse(x,y,d,d);
y = y+dy;
ellipse(x,y,d,d);
y = y+dy;
ellipse(x,y,d,d);
y = y+dy;
ellipse(x,y,d,d);
y = y+dy;
ellipse(x,y,d,d);
y = y+dy;
ellipse(x,y,d,d);
```

この円の移動量を小さくして、一度の一個だけ円を表示するよう

本当は、変数の有効範囲という話をしないといけないのですが、今は触れないことにしています。このことは、もう少し後で、説明します。

本当は、「int dy=40;」の直ぐ後に、「int x = width/2;」という行を持ってきたいのですが、予期した動作をしません。その理由が予想できますか？

にすれば、円が移動するアニメーションとなるはずです。そこで、setup 関数と draw 関数の組み合わせが登場します。

動く円を表示するその 1 サンプル 2-17

```
int d = 20; // 円の直径
int y = 0;  // 円の中心の Y 座標
int dy = 1; // 一度の移動量
int x;      // 円の中心の X 座標
void setup(){
  size(180,200);
  smooth();
  fill(0,0,255);
  x = width/2;
}
void draw(){
  background(255);
  ellipse(x,y,d,d);
  y = y+dy;
}
```

サンプル 2-17 を変更して、もっとゆっくり移動するようにしたものを次に示します。このサンプルでは、変数 dy に 1 より小さな正の数を指定したいので、変数 y や変数 dy のデータ型を変更しています。それ以外は、同じになっています。

動く円を表示するその 2 サンプル 2-18

```
int d = 20; // 円の直径
float y = 0;  // 円の中心の Y 座標
float dy = 0.5; // 一度の移動量
int x;      // 円の中心の X 座標

void setup(){
  size(180,200);
  smooth();
  fill(0,0,255);
  x = width/2;
}

void draw(){
  background(255);
  ellipse(x,y,d,d);
  y = y+dy;
}
```

変数値の表示

プログラムを作っていると、変数の値を調べたくなります。このような目的のために、Processing では、println という命令文

println は「print line」の略だと思います。

（関数）が用意されています。プログラム中で「println(変数名);」や「println(式);」という文を加えると、変数の値や式を計算した結果がメッセージエリアに表示されます。

「式を計算した結果」のことを、式の値を呼ぶことがあります。

println では文字を表示することも出来ます。

println の使用例 サンプル 3-13

```
void setup(){
  size(300,300);
  smooth();
  fill(51);
}
void draw(){
  background(255);
  ellipse(mouseX,mouseY,20,20);
  println(mouseX); // システム変数 mouseX の値を表示
}
```

論理エラーを持っているプログラムのデバッグを行うさいに、変数の値を表示することで、意図していない動作を起こしている場所を探すことがあります。途中の計算結果を変数に代入しておくことで、デバッグがやりやすくなる場合があります。