

Arduino & Processing に関する参考資料

作成者：藤本 健司

(神戸市立工業高等専門学校 電子工学科)

■ Arduino の使用方法

(1)Arduino の起動

デスクトップ上の Arduino のアイコン  をダブルクリックして起動します。

(2)Arduino の画面

起動させると、図1のような画面が出てきます。

マイコンボードへの書き込み

検証 (コンパイル)

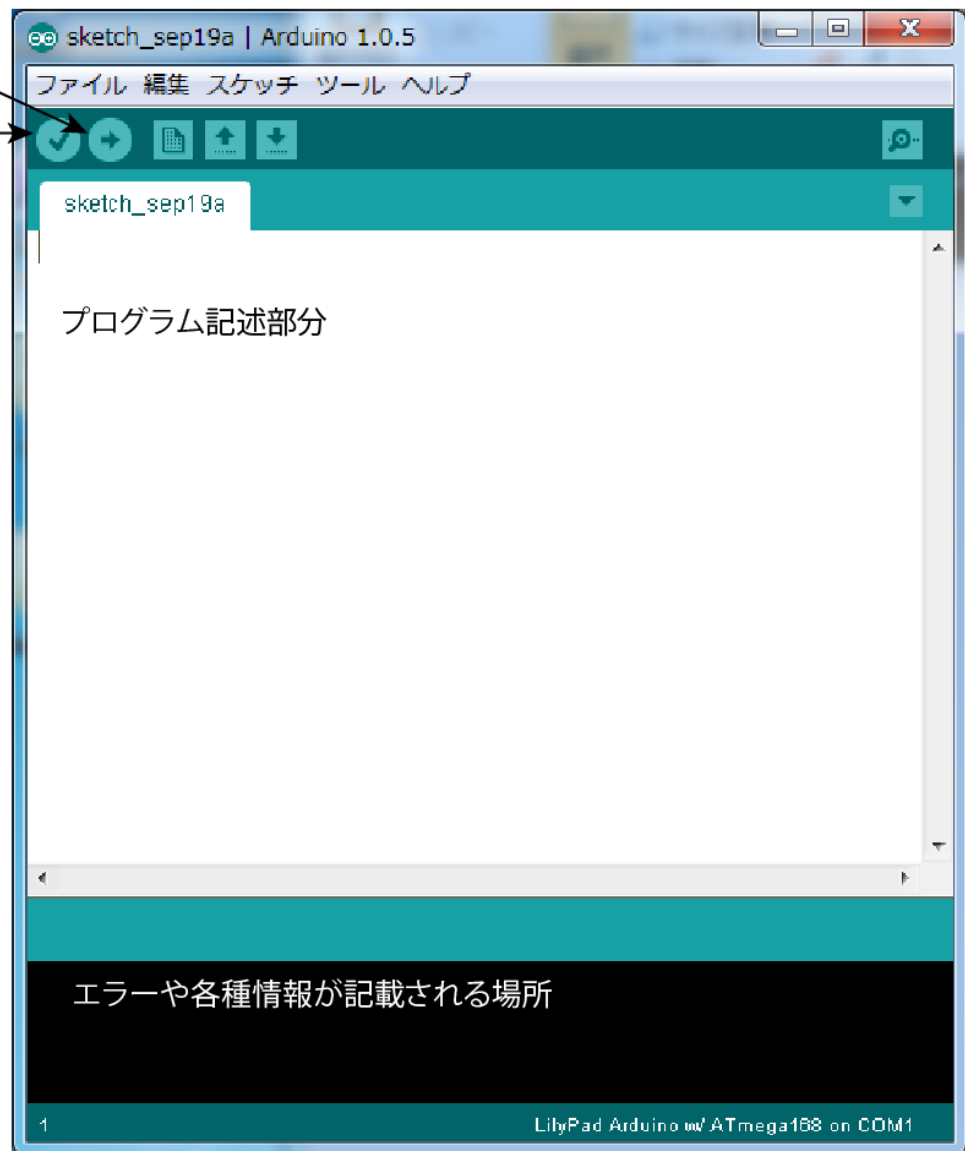


図1. 起動画面

起動画面を見ると、上部にメニューがあり、それぞれ左から、「ファイル」、「編集」、「スケッチ」、「ツール」、「ヘルプ」となっている。

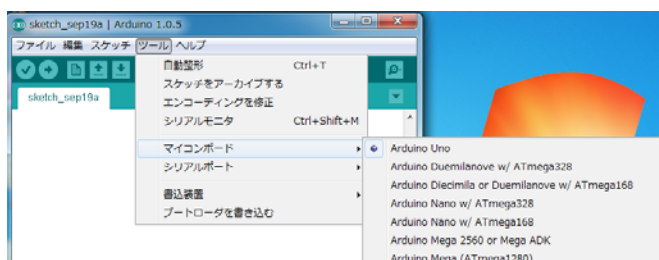
「ファイル」では、新規ファイルの作成や、保存などを行うことができます。

「編集」では、コピーやペーストなどを行うことができます。

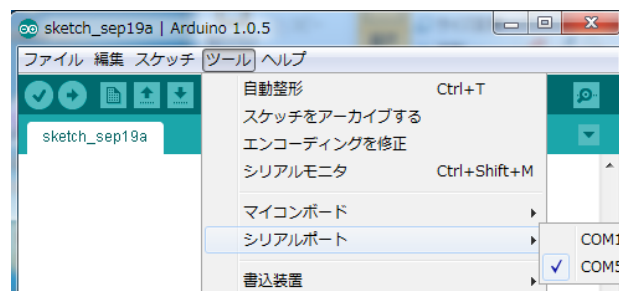
「スケッチ」では、検証・コンパイルなどが行えます。なお、検証・コンパイルは専用のボタンも

用意されています（図I参照）。

「ツール」では、シリアルモニタやマイコンボード、シリアルポートなどがあります。シリアルモニタはシリアル通信を利用する場合にはプログラムの挙動を確認するのに便利です。また、使用するマイコンボードの設定やシリアルポートの設定が間違っているとエラー「**avrdude:stk500_getsync(): not in sync: resp=0x00**」が出ます。今回使用するマイコンボードの種類は、Arduino Uno(図II参照)で、使用するポート(図III参照)は各自で調べる必要がありますが、最初は何もいじらなくてもかまいません(マイコンボードへの書き込みの際に、間違っていたら図IVのようなエラーが出ます。そのときには他のポートを選んで再度書き込んでください)



図II. マイコンボードの設定



図III. ポートの設定

```
コンパイル後のスケッチのサイズ: 2,426バイト (最大容量14,336バイト)  
avrdude: stk500_getsync(): not in sync: resp=0x00
```

図IV. ポートを間違えたときのエラー (例外もあります)

基本的に以下のような流れで、進んでいきます。

1. 起動
2. プログラム作成
3. 保存
4. 検証・コンパイル (もしエラーが出たら2に戻って修正、エラーが出なければ5へ)
5. マイコンボードへの書き込み (エラーが出なければ、Arduino にプログラムが組み込まれます)
6. 確認 (シリアルモニタや実際の回路を見て確認)

慣れれば非常に簡単に作業することができます。また、コンパイルやボードへの書き込みはメニューから選ぶよりも専用ボタンをクリックしたほうが便利です。

1. LED を用いた実験

Arduino にはデジタル入出力端子として 0 番～13 番までの 14 本のピンが割り当てられています。また、アナログ入力ピンとしてアナログ 0 番～5 番の 6 ピン分が用意されています（デジタルとして割り当てたい場合、アナログ 0 番～5 番までをデジタルの 14 番～19 番として使用可能）。

LED を光らせるためには、デジタル入出力を行う必要があります。Arduino では、以下の 3 つの関数や命令を利用すれば簡単に実現することができます。

(a) ピンの入出力モード設定

```
pinMode(pin, mode);
```

ピンモードを入力、もしくは、出力に変更する命令です。

pin にはピン番号を、mode には「INPUT」、「OUTPUT」を入れることで、モードを入力モード、出力モードに切り替えることができます（モードは大文字で指定すること）。

デフォルトでは、デジタル 0 番～13 番は INPUT となっています。出力で使用する場合には、OUTPUT を指定してやる必要があります。ここで例の最後にセミコロン(;)がついていますが、これは Arduino のプログラムでは必ず文の最後ではセミコロン(;)をつけるようになっているからです。こうすることで、Arduino IDE で作成したプログラムを Arduino 上で使用できるようにコンパイル（翻訳）するとき、どこまでが命令かということが分かるようになっています。

(例)12 番ピンを出力として使用する場合

```
pinMode(12, OUTPUT);
```

(b) 入力モードに設定したピンからの入力

```
変数 = digitalRead(pin); // 戻り値は int 型
```

各入力ピンから信号を入力します。ピンに高い電圧が入力されている時は変数に HIGH が、低い電圧が入力されているときには LOW が代入されます。

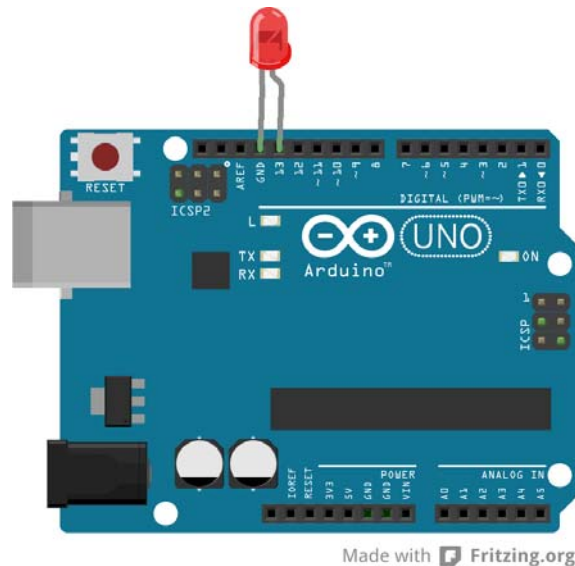
(c) 出力に設定したピンへの出力

```
digitalWrite(pin, value);
```

pin で指定した各出力ピンに電圧を出力します。value には、HIGH か LOW を指定します。HIGH なら通常は Vcc 電圧が、LOW なら 0 V が出力されます。ちなみに、今回使用する Arduino Uno では Vcc は 5V となります（設定を変更すれば 3.3V に変更することもできます）。

1.1 回路の作成

では、まずLEDを下図のようにArduinoに接続してみましょう。今回はデジタル13番ピンとGND（グラウンド）に接続します。今回13番ピンを選んだ理由は、GNDの隣にあるので接続しやすいという理由と1kΩの抵抗が入っているという理由です。アノードを13番ピンへ、カソードをGNDに接続します（アノードは、足の長い方ですが、もし長さが同じ場合には、LED内を見たときに大きい面積の板にくっついている方をカソード、小さいほうをアノードと覚えておくとうよいかと思います。但し、例外もあります）。



(注意点) もし、13番ピン以外を使用する際には保護抵抗として1kΩ程度の抵抗をはさんで下さい。

1.2 サンプルプログラム (LEDの点灯)

```
// 通常 setup のところに、初期条件を記載します。
// ここでは、13 番ピンを出力として使用することを宣言しています。
void setup() {
  pinMode(13, OUTPUT); //13 番ピンを出力として使用
}

// Arduino では、void loop()が本体になります。ここに書かれている命令が繰り返し行われます。
// 今回のケースでは、13 番ピンに HIGH を出力するようにしています。
void loop() {
  digitalWrite(13, HIGH); // 13 番ピンに HIGH (5V) を出力する
}
```

プログラム内で//で書かれている部分はコメントとして扱われます。実際のプログラムを実行するには、無視されます。

2. 点滅させるためには？

1.2のサンプルでLEDの点灯を行うことができました。このLEDを点滅させるには、一度点灯したLEDを消し、また点灯するといった動作を繰り返す必要があります。ちなみに、LEDを消すためには出力をHIGHからLOWにしてやる必要があります。

(ヒント 1) void loop()内に書かれた命令は繰り返し行われる。

(ヒント 2) LEDを消すには、13番ピンの出力をLOWにする必要がある。

プログラムが作成できた人は、**Arduino**に転送して実行してみてください。多分、**LED**の点灯、消灯を繰り返すようにしたのにもかかわらず、点灯したままになっているように見えていると思います。これは、処理速度が速すぎて点滅しているのが分からないためです。

■ delay 関数

速すぎて点滅が分からないという人は、この関数を利用してみましょう。**delay**関数は、

```
delay(数値);
```

と記載します。数値 **ms** (ミリ秒) だけ待つという働きがあります。この関数を、点灯後と、消灯後に入れてみてください。数値はあまり小さいとわかりにくいので最初は **1000** などの大きな値を入れてみてください。

2.1 いろいろな記述方法

1.2のサンプルプログラムでは、ピン番号を直接記載しましたが、実際には、接続ピンを変えることもよくあります。その際、ピン番号を直接記載していると、プログラム内のすべてのピン番号を書き換えないといけない必要があります。長いプログラムになってくるとこの作業は非常にミスが出やすいと書き換えの労力がかかります。そこで、以下のような記述方法もあります。

```
//整数の変数LEDpinに13を代入。以後、"LEDpin"と入力すると数字の13と同じ働きをする。
int LEDpin = 13;
void setup() { //ピンの設定などを行う void setup()
  pinMode(LEDpin, OUTPUT); //13番ピンをOUTPUT(出力)に指定
}
void loop() { //プログラム本体(ずっと繰り返す)
  digitalWrite(LEDpin, HIGH); //LEDpinにHIGH(5V)を出力
}
```

3. 入力値を読み取るプログラム

連続的な入力値を読み取るためには、デジタル入力では無理です。そのため、今回はアナログ入力を用います。また、Arduino とパソコンはシリアル通信が行われており、これを利用することで Processing やその他の外部ツールと連携することができます (Arduino IDE には、シリアルモニタと呼ばれるモニタがあり、これを利用するとシリアル通信の結果を見ることができ、デバッグに役立ちます)。ここでは、アナログ入力取得とシリアル通信機能を利用して入力値を読み取るプログラムを作成します。まず、下準備として、アナログの 0 番ピンに直流安定化電源の+側を、GND に一側を接続してください。

3.1 アナログ入力取得

```
変数 = analogRead(analog_pin);
```

変数は基本的に int 型になります。analog_pin の部分には、アナログ入力のピン番号が入ります。デジタル入出力のときと同じく数字だけでかまいません。また、入力電圧(0~5V) を 0~1023 に変換して取得されます。

3.2 シリアル通信 (下準備)

シリアル通信を行うためには、前準備が必要になります。以前のプログラムでもあった void setup() の中に以下のような命令をまず追加します。

```
Serial.begin(数値);
```

この命令は、シリアル通信を行う速度(bps : bits per second)を表しています。使用する Arduino の種類によっても上下限はありますが、この値が大きいと通信が速く (情報をたくさん送れる)、小さいと通信が遅い (情報をあまり送れない) といったことになります。ただ、単純に大きければよいわけではなく、大きい場合には、Arduino に搭載されているチップが送られてくる情報を処理できないといったケースも起こります。なお、指定できる数値は、

300/1200/2400/4800/9600/14400/19200/28800/38400/57600/115200

から選択することができます。一般的には 9600 がよく使用されています。

3.3 シリアル通信 (データ出力)

```
(a) Serial.print(データ,フォーマット);
```

```
(b) Serial.println(データ,フォーマット);
```

データ出力には(a) と(b)のようなものがあります。違いは、(a)は改行なしのデータ出力、(b)は改行ありのデータ出力になっています。データには、出力したい整数型もしくは String 型 (文字) の変数が入ります。フォーマットは、データを変換する方法になります。例えば、データの変数名を val とす

ると、次のように設定できます。

(1) `Serial.print(val, DEC);`

このケースだと `val` の値を 10 進数の値で送信します。何も指定しないときは 10 進数になるようになっています。

(2) `Serial.print(val, HEX);`

このケースだと、`val` の値を 16 進数の値で送信します。

(3) `Serial.print(val, BIN);`

このケースだと、`val` の値を 2 進数の値で送信します。

`Serial.println` でも同様にフォーマットを変更することができます。

以下に `Serial.print` と `Serial.println` の実行例を記載してみます。自分で作成したプログラムの結果は、「ツール」→「シリアルモニタ」から確認してみてください。（但し、使用する関数などによっては、確認できないケースもあります）

Serial.print(data,format)の例

```
000120 . . .
```

Serial.println(data,format)の例

```
0
0
0
1
2
.
.
.
```

3.4 サンプルプログラム

```
int analn0; // 変数の宣言（アナログ入力の値を入れるための変数）

void setup(){
  Serial.begin(9600); // シリアル通信の設定 9600bps
}

void loop(){
  analn0=analogRead(0); // アナログ 0 番ピンの入力値を analn0 に代入
  Serial.println(analn0); // パソコンに analn0 の値を転送（改行あり）
  delay(100); // 100ms 待つ
}
```


100ms 待っているのは、delay 無しにすると送るデータが多くなりすぎるため、少し待ち時間を設定しています。今回は、1 入力のみで作成しましたが、複数の入力にも同じように作成すれば対応することができます（入力用の変数を入力分用意するか、配列などを利用して汎用化することも可能）。但し、その場合には、今回のケースでは、Serial.println を利用していますが、複数の入力を用いる場合には、そのままでは利用できません。少し考えてみてください。

（注意点）このケースでは、出力されるのは 0~1023 の数値になります。実際には 0~5V の範囲を測定したいので正しい値を出力するには変換してやる必要があります。

3.5 変換するためには？

実際の値にするためには、変換してやる必要があります。一番簡単な方法は、0~5V の入力値が 0~1023 に変換されているのであれば、同様に、0~1023 を 0~5 に変換してやればよいことになります。

（ヒント 1）変換後の変数として val1 を用意する（double 型）。

変数宣言は、double val1; とすればよい。

（ヒント 2）変換する数式を作成し、変換後の変数である val1 に代入すればよい。

例：0~3V の入力値が 1024 分割されているときの変換

```
val1=3 / 1024.0 * analog0;
```

（ヒント 3）変換した値を Serial.println に入れてやればよい。

■ map 関数

変換するために Arduino では map 関数と呼ばれるものが用意されています。但し、map 関数で指定できるのは整数のみであり、変換される値も整数になります。

変数 = map(value,fromLow,fromHigh,toLow,toHigh);

変数:	対応後の値 (int)
value:	対応させる値 (int)
fromLow:	対応前の値の下限 (int)
fromHigh:	対応前の値の上限 (int)
toLow:	対応後の値の下限 (int)
toHigh:	対応後の値の上限 (int)

例えば、入力範囲が 0~3V のケースで考えると、変換後の変数を val とし、変換前の変数を analog0 とすると、以下のようになる。

```
val = map(analog0,0,1023,0,3);
```

但し、この変換をすると、0~1023 までの 1024 分割だった値が、0~3 までの 4 分割になってしまう。そのために、どうすればよいか少し考えてみてください。

（ヒント）例えば、0~3.3V までの範囲に変換したい場合、そのままでは変換できない。どうにかして範囲を int 型に直せないだろうか？

(今回までの注意 Point)

今までの説明では入力電圧が Arduino の入力電圧の上限である 5V までであると仮定していますが、実際に入力値が 5V を超える場合には、そのままでは Arduino が壊れてしまいます（ちなみに、入力上限を超えている場合は、変換前の生データでは、最大値の 1023 が出力されます。そのため、1023 という数値が頻繁に出る場合には 5V を超えた電圧が入っている可能性があります）。そこでそういった場合には、抵抗を入れることで分圧し、アナログ入力を複数に分けることで対応することができます。

4. データの保存方法について（データロガー）

データを保存する方法として一番簡単な方法は、シリアルモニタに出力された数値をコピーしてメモ帳などにペーストする方法です。但し、この方法では、長期間に渡るデータの保存などには向いていません。また、Arduino のメモリなどにデータを保存することも可能ですが、今回は、シリアル通信を用いて Processing 上で Arduino から送られてきたデータを保存する方法を考えます。

4.1 データをファイルに保存する方法（Arduino 下準備編）

まず、Processing 上でデータを保存する前に Arduino のプログラムを作る必要があります。以前、アナログ入力を出力するプログラムを作りましたが、それに少し付け加えるだけで作成が可能です。

(1) シリアル通信を行う上で大事なこと 1（Arduino）

以前のアナログ入力のプログラムでは、データを出力する際に `Serial.print` または、`Serial.println` を用いましたが、データを他のツールなどとやり取りする際にはこの方法は向いていません。何故かというと、`Serial.print`、または、`Serial.println` を利用すると出力形式はすべて ASCII 文字列形式となります。例えば、`Serial.print(123)` と入力すると、これは "123" という 3 バイトの情報として送られます（実際に 123 というのは数値で扱えば 1 バイト）。そのままバイトデータとして送る場合には、以下の命令を使用することで実現できます（但し、そのままでは、シリアルモニタで結果を確認できなくなります）。

```
Serial.write(変数);
```

(2) シリアル通信を行う上で大事なこと 2（Arduino）

シリアル通信を行う上で大事なことはまだあります。実際にデータを通信する際には、通信する相手とデータのやりとりを行うために準備が必要です。例えば、通信する相手が準備完了していないときにデータを送っても意味がありません。そういった準備を行う方法として以下のような手法があります。

```
if(Serial.available(>0){  
  処理内容  
}
```

ここで、ifという命令がでていますが、一般的には以下のように記載されます。()内に記載された条件を満たした場合、処理1を行い、条件を満たさない場合は、処理2を行います。elseは省略することもできますし、if文の処理1の部分に更にif文を追加することも可能です(ifのネストと呼ばれます)。なお、条件は、演算子や論理演算子を用いて1つまたは複数の式から構成されています。

```
if(条件){  
    処理1  
}  
else{  
    処理2  
}
```

■ 代表的な演算子

$x == y$ (x と y は等しい)
 $x != y$ (x と y は等しくない)
 $x < y$ (x は y より小さい)
 $x > y$ (x は y より大きい)
 $x <= y$ (x は y 以下)
 $x >= y$ (x は y 以上)

■ 代表的な論理演算子

$X \&\& Y$ (AND: X と Y 両方が成り立つ場合)
 $X \|\| Y$ (OR: X か Y どちらかが成り立つ場合)
 $!X$ (NOT: X が成り立たない場合)

(X と Y はそれぞれ条件式を表している)

次に、Serial.available()>0 ですが、これは、何かデータが受信されていることを意味します。すなわち、先ほどのプログラムの意味は、何かデータが受信されたら、処理内容を行うといったことになります。

4.2 サンプルプログラム

今までのことをまとめると以下のとおりになります。

(1)文字列通信は効率が悪いのでバイト通信を行う。

(2)通信をする際には下準備が必要（相手が準備できたかどうかの確認）

以上に加えて、次のようにします。

(3)通信するデータを小さくするため、データ範囲を0~255(1バイト)に変換する（3.5参照）。

なお、サンプルプログラムでは一部分が抜けていますので補足してプログラムを完成させてください。

```
int sensor1; // 変数の定義 (sensor1 はアナログ0番の入力)
int senout; // 変数の定義 (senout は sensor1 を変換したもの)

void setup(){
  Serial.begin(9600);
}

void loop(){
  sensor1= ? ; // アナログ0番ピンの入力を sensor1 に代入
  senout = ? ; // データを1バイト(0~255)に圧縮
  if( ? ){ // データを受信したら次の行の内容を実行
    ? ; // データをシリアル通信で送信(バイトデータ)
  }
  delay(100);
}
```

太字で記載された?マークの部分が抜け落ちています。正しい命令を記載してください。

(シリアル通信のPoint)

文字列データ通信は、シリアルモニタなどで確認できるといった利点はあるのですが、こういった情報量が多くなってしまうことが欠点です。特に Arduino 自体の処理時間は1ms~2ms くらいの周期でリアルタイムで処理できるのですが、通信速度を57600bps（2番目に高速な通信速度）まで上げたとしても57.6bit/ms \div 7byte/msしか通信できないこととなり、文字列通信を行うことで通信で何msも消費してはリアルタイム性が失われてしまいます。そのために、バイト通信できるものはバイト通信を行うほうが良いとされています。

4.3 シリアル通信を行うために (Processing)

4.3.1 シリアル通信のための準備1 (宣言)

さて、いよいよ Processing を用いてシリアル通信を行います。Processing では、シリアル通信を利用したい際には以下の宣言をする必要があります。

```
import processing.serial.*;
```

この命令はシリアル通信をするためのライブラリを組み込むことを表しています（このライブラリにはシリアル通信を行うために必要な設定やプログラムが組み込まれています。私たちは、このライブラリを利用することで、シリアル通信をするためのプログラムを最初から書く必要がなくなります）。

シリアル通信を行う場合、そのポートの名前をつけてやる必要があります。その命令は以下のとおりです。

```
Serial port_name;
```

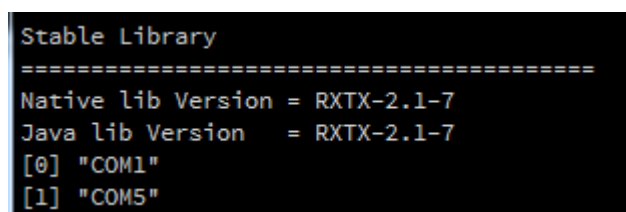
ここで、port_name の部分はどのような名前でもかまいません。また、ポートについては、Arduino がどのポートを使用しているか理解しておかないとプログラムを組むことができません。しかし、Arduino やプログラム初心者にとって、ポート番号など普通分かるはずがありません。次にポート番号の確認方法を説明していきます。

4.3.2 シリアル通信のための準備 2（初期設定[void setup()]）

Processing では、void setup()内に、以下のように記載することで使用しているポート番号がわかるようになります。

```
println(Serial.list());
```

この命令を記載しておくで、図1のように使用できるポート番号とポート名がでてきます。Arduino で設定したポート名を確認してそのポートに対応した番号を覚えておきましょう。



```
Stable Library
=====
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
[0] "COM1"
[1] "COM5"
```

図1. Serial.list の例（COM1 に対応するのが0、COM 5に対応するのが1となっています）

ポート番号が分かったら以下の命令を void setup()内に記載する必要があります。

```
port_name= new Serial(this, Serial.list()[ポート番号], 伝送速度);
```

ここで、port_name は、宣言のときに設定したポート名、ポート番号のところには、対応するポート番号を、伝送速度には Arduino で設定した伝送速度を入力します。例えば、port_name が myPort、ポートは COM5（図1 から COM5 はポート番号1）、伝送速度は 9600bps とすると、以下のように記載します。

```
myPort=new Serial(this,Serial.list()[1],9600;
```

(注意点) 使用するポートなどは、使用する環境に応じて変化するので先述したプログラム例を利用する場合には、そのままではエラーが出る場合があります。必ず、自分の環境を確認し、ポート番号などを事前に調べておいてください。

4.3.3 シリアル通信のための準備3 (データの取得)

次に、Arduino から送られてきたデータを受け取る方法について記載します。今回は1バイトのデータを送ることになっていますので1バイト受け取ることができる命令を利用します。

```
変数=ポート名.read();
```

例えば、ポート名：myPort、変数名：val の場合は、

```
val = myPort.read();
```

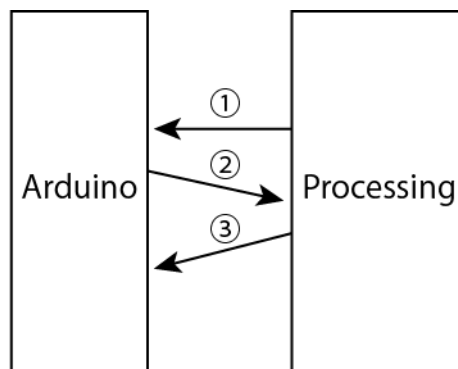
と成ります。

4.3.4 サンプルプログラム2

さて、以下にシリアル通信を行ってデータを受け取るサンプルプログラムを記載します。4.2 のサンプルプログラムに対応させるために、以下の点に注意します。

1) Arduino からデータをもろうためには、何かのデータを送る必要がある

後、データを受け取る回数を設定しておかないと無制限にうけとってしまうので、今回は 100 回うけとればプログラムを終了させるようにします。イメージは図2のようになります。



- ① データを送る (データは何でも良い)
- ② データを受信できたので Arduino 側で取得したデータを送る
- ③ Arduino から取得したデータを出力し、ポート内のデータをクリアし、再度受け取る準備を整え①に戻る

図2. プログラムの流れ

```

import processing.serial.*;           // シリアルライブラリの導入
Serial myPort;                       // myPort という名前のインスタンス作成 (名前は自由)

int senout;                          // Arduino から送られてくるデータを代入する変数
int count=0;                         // カウント用変数

void setup () {                      // 初期条件設定
  size(400, 400);                   // ウィンドウサイズの設定 (このケースでは 400×400 ピクセル)
  background(255);                 // ウィンドウ内の背景色 (このケースでは白色)
  println(Serial.list());           // 使えるシリアルポートの表示 (ポートの確認が楽)

  // シリアルポートリストの 1 番目のポートを使用。必ず 1 番目ではないので、
  // 必ずシリアルポートリストを確認して適切なポート番号を入れること。
  myPort = new Serial(this, Serial.list()[1], 9600);
}

void draw () {                    // Arduino でいうところの void loop(){}
  if (count>100) {                 // もし count が 100 を超えたら以下の命令を行う
    exit();                        // プログラムから抜ける
  }

  if(count==0){                   // count が 0 のとき以下の命令を行う (一番最初のみ)
    myPort.write(255);             // ポートに 255 を送る(Arduino に準備が整ったことを知らせる)
  }

  if(myPort.available()>0){
    senout=myPort.read();         // データを読み取る。この myPort.read()は 1 バイトのデータに対応。
    background(255);             // 背景を白で塗りつぶす (重ね書き防止用)
    fill(0);                     // 図形内部の色を黒にする (今回のケースでは次の行の文字)
    text(senout,30,40);         // 変数 senout の値を、(x,y)=(30,40)に文字として表示
    myPort.clear();             // ポート内のデータをクリアする
    myPort.write(255);         // ポートに 255 を送る(Arduino に準備が整ったことを知らせる)
    count++;                      // count を 1 増やす count=count+1;と同じ意味です。
    // (こういう手法をインクリメントという)
  }
}

```

このサンプルプログラムの中に、今までに説明していなかった命令がいくつか入っています。以下に簡単に説明していきます。

■ size() ウィンドウサイズの設定

表示に使用するウィンドウのサイズをピクセルという単位で設定します。今回は、表示させるだけなのでサンプルでは400×400ピクセルとしていますが、もっと小さくてもかまいません。

size(width, height);

width : 描画ウィンドウの幅、height : 描画ウィンドウの高さ

■ background() 背景色の設定

ウィンドウの背景の色を設定します。デフォルトは灰色となっています。Processingでの描画は基本的に上に上書きしていく方式なので、先に書かれたものを消しておかないと重ね書きになってしまいます。そのため、この命令を **draw()** の先頭に置くことでウィンドウをクリアさせたりします。今回のサンプルでも重ね書きを防ぐ目的で他の場所に使用しています。

background(val);

valの部分は、いろいろな与え方がある。例えば、デフォルトではグレースケールというものになっているため、0を入れると黒、255を入れると白になります。きちんとした色で表したい場合には、以下のように入れるとよい。

background(valR, valG, valB);

この方式はRGB方式になっており、

valR : 赤または色相 (0~255)

valG : 緑または彩度 (0~255)

valB : 青または明度 (0~255)

と記述することができる。他にも、WEBページを作成する際に利用するカラーコードも指定できる。その場合は以下のように記載する。

background(#FFFFFF); この場合は、白色になる。

■ draw()

これは、Arduinoで使用したloop()と同じものだと考えてかまいません。draw(){ }の{ }内の命令が繰り返し実行されます。

■ exit() プログラムからの脱出

この命令を実行するとプログラムから脱出（すなわち、プログラムの全ての処理を終了）することができます。よく似た命令に **break** がありますが、これは **switch**、**for**、**while** などの繰り返し命令のブロックから処理を打ち切って脱出し、そのブロックの次の行にジャンプすることができます。

■ ポート名.write(数値) データの送信

今回のケースでは、ポート名がmyPortだったので、myPort.write(255);と記述してあります。255という数字ですが、別段意味はありません。ただ、これ以上になると2バイトになるので、1バイトで送ることのできる最大の数値をいれてあるだけです。これは4.2のサンプルプログラムでArduinoが何ら

かのデータを受信したらシリアル通信を始めるようにプログラムしているのでそのきっかけのデータを渡すためだけのものです。このデータが送られると同時に Arduino と Processing の間でシリアル通信が始まります。

■ fill() 塗り色の指定

図形の内部に塗られる色を指定できます。色の指定については background()と同じです。

■ text() 文字の表示

文字または文字列を指定した場所に表示する命令です。文字の色は先述した fill 関数で指定できます。

```
text(data,x,y);
```

data : 表示する文字 (String, char, int, float 型対応)

x : 表示位置の x 座標

y : 表示位置の y 座標

■ ポート名.clear() ポート内のデータ消去

この命令は、ポート名で指定されたポート内のデータを全て消去する命令です。今回のケースでは、myPort.clear();と記載しています。なぜこの命令を使用するかというと、ポート内のデータを消去せずに何かデータがある状態にしておくとうデータを受信した状態と Arduino が勘違いしてしまうので、Processing 側で Arduino からデータを受け取った後は、ポート内のデータをクリアしています。サンプルでは、クリアして何も無い状態にした後、255 を送信し、Arduino が通信するようにしています。

(プログラムの Point)

4.3.4 のサンプルプログラム 2 でテキストを作成したウィンドウ上に表示するプログラム部分は、以下の 3 行ですが、

```
background(255);
```

```
fill(0);
```

```
text(senout,30,40);
```

確認作業を Processing の画面の下側にある黒い場所で行うのであれば、次のように記載すると、黒い場所にデータの値が改行有りて出力されます。

```
println(senout);
```

状況に応じて使い分けてください。

4.4 データをファイルに保存する方法 (Processing)

今までの Processing のプログラミングで、Arduino と Processing 間でのデータ通信を実現することができました。次のステップとして、ここでは、データをファイルに保存する方法について簡単なサンプルプログラムを使って説明します。

```

PrintWriter output;                // PrintWriter 型のオブジェクトを宣言。
                                     // output の部分は自由に変更可。

int data=50;

void setup(){
  size(100,100);
  output = createWriter("test.txt"); // ファイル名 test.txt でファイルを開く
}

void draw(){
  output.print("kobe");              // ファイルに出力する。このケースでは kobe(改行無し、文字)
  output.print("kosen");           // ファイルに出力する。このケースでは kosen(同上、文字)
  output.print(data);              // ファイルに出力する。このケースでは 50 (同上、数値)
  output.println();               // ファイルに出力する。このケースでは改行のみ

  output.flush();                  // ポート内に残っているデータを全て書き込む
  output.close();                  // ファイルを閉じる

  exit();
}

```

このプログラムで行っているデータ保存の処理を以下の4つのパートに分けて説明します。

- 1) 宣言
- 2) 初期化
- 3) 書き込み
- 4) 終了処理

■ 宣言部分

```
PrintWriter output;
```

ここで、PrintWriter の後ろにある output という名前は、自由に変更しても良い。例えば、PrintWriter test:でもかまいません。但し、test のように自分で決めた場合は、この後の説明の部分で output としている部分を自分で決めた名前に書き換える必要があります。

■ 初期化

```
output = createWriter("test.txt");
```

createWriter()は、()内にデータを保存したいファイル名を記載します。ファイル名については自由です(今回のケースでは test.txt としています)。もし、該当するファイルがない場合には、指定したファ

イル名で新しく作り直してくれます。同じファイル名がある場合には、上書きされます（前のデータはなくなります）。

■ 書き込み

```
output.print(data);
```

```
output.println(data);
```

ファイルに書き込むときには、上記命令を使用します。dataの部分は数値でも、文字でもかまいません。もちろん、データを代入している変数でもかまいません（その場合は、代入してある内容がファイルに書き込まれます）。ファイルに出力するという点以外は、今までに扱ってきたprint()やprintln()と変わりません。

■ 終了処理

```
output.flush();
```

```
output.close();
```

ファイルへデータを出力し終わった後は、上記の命令を入れて終了処理を行います。output.flush()の部分では、データがもし残っている場合には全て書き出す処理になります。最後のoutput.close()の部分は、初期化のところでファイルを開いた状態にしているため、開いていたファイルを閉じる作業をしています（C言語などの他の言語でもこのようにファイルを利用（open）したら、最後にファイルを閉じる（close）することが多いです）。この2行の命令については決まり事だと思ってください。

4.5 サンプルプログラム（データ保存）

さて、今までにシリアル通信とデータ保存の方法について説明してきましたが、それらを使えば、ArduinoからProcessingに送られたデータをファイルに保存することができます。以下のサンプルプログラムは、4.3.4 サンプルプログラミングに4.4で説明したデータ保存の部分を付け加えるものになります。自分で（ ）内の空いている部分を補完して完成させてみてください。但し、実際のプログラムからは、補完する際に（ ）は消すようにしてください。

```
import processing.serial.*;
Serial myPort;

int senout;
int count=0;
(   ここに宣言を入れる   )

void setup () {
  size(400, 400);
  background(255);
(   ここに初期化を入れる   )
  println(Serial.list());
```

```
myPort = new Serial(this, Serial.list()[1], 9600);
}

void draw () {
  if (count>100) {
    (   ここに終了処理を入れる   )
    exit();
  }

  if(count==0){
    myPort.write(255);
  }

  if(myPort.available()>0){
    senout=myPort.read();
    (   ここに書き込みを入れる   )
    myPort.clear();
    myPort.write(255);
    count++;
  }
}
```

このサンプルプログラムを作成して保存されるデータは、0~255の数値になってしまいます。実際の値を保存するためにはどうしたらよいでしょうか？今までに数値の変換はやってきましたので、過去のサンプルなどを参考にして実際の値を保存するプログラムに変更してみてください。

4.6 イベント処理

ここでは、Processing に用意されているイベントと呼ばれるものについて説明します。難しそうと思った人は飛ばしてもらっても構いません。ただ、イベント処理を使いこなせるようになると非常に便利ですので目を通すだけ通してもらったらいと思います。

まずは、イベントについて簡単に説明します。Processing で使用できる代表的なイベントとしては、マウスイベントとキーボードイベントと呼ばれるものがあります。これは、マウスやキーボードに何らかの変化があった時（こういったものをイベントといいます）、その変化によって処理を行うことができるようになっています。また、変化の種類を条件に入れて条件分岐をつくることでいろいろなイベントに対応して処理を分けることもできます。

4.6.1 マウスイベント

次にマウスイベントの一例を記載します。

```
void mousePressed(){
```

```
    // マウスボタンが押されたときに{ }内の命令が実行されます。
```

```
}
```

```
void mouseReleased(){
```

```
    // マウスボタンが離されたときに{ }内の命令が実行されます。
```

```
}
```

```
void mouseMoved(){
```

```
    // マウスボタンを押さずにマウスを動かしたときに{ }内の命令が実行されます。
```

```
}
```

```
void mouseDragged() {
```

```
    // マウスがドラッグされたときに{ }内の命令が実行されます。
```

```
}
```

(マウスイベントの Point)

描画ウィンドウ内のマウスポインタの座標や、押したボタンの種類などが以下の変数を用いると簡単に取得することができます。これらの変数と先ほどのマウスイベントを if 文などの条件分岐と組み合わせることで複雑な処理を行うことができます。

- mouseX : X 座標
- mouseY : Y 座標
- mouseButton : マウスボタンの押された位置を表します。

LEFT, CENTER, RIGHT の 3 種類の値があります。

4.6.2 マウスイベントを用いた例

では、実際のマウスイベントを用いた例を考えてみます。押されたマウスボタンを読み取って画面に表示するプログラムを作ってみます。

```
void setup(){
    size(100,100);
    background(255);
}

void draw(){
}

void mousePressed(){ // マウスのボタンが押されたとき、{ }内が実行されます
    background(255); // 重ね書き解消のため、最初に初期設定の背景色で塗ります
    if(mouseButton==LEFT){ // 押されたボタンが左側 (LEFT) なら{ }内を実行します
        fill(255,0,0); // 図形を赤色 (255,0,0) で塗ります
        text("LEFT",30,30); // LEFT という文字列を(x,y)=(30, 30)に表示します
    }

    if(mouseButton==CENTER){ // 押されたボタンが真ん中 (CENTER) なら{ }内を実行します
        fill(0,255,0); // 図形を緑色 (0,255,0) で塗ります
        text("CENTER",30,30); // CENTER という文字列を(x,y)=(30, 30)に表示します
    }

    if(mouseButton==RIGHT){ // 押されたボタンが右側 (RIGHT) なら{ }内を実行します
        fill(0,0,255); // 図形を青色 (0,0,255) で塗ります
        text("RIGHT",30,30); // RIGHT という文字列を(x,y)=(30, 30)に表示します
    }
}
```

if文を使用することで、このプログラムでは3つのケースに処理が分けることができます。例えば、もう少し応用を考えれば、マウスの右ボタンを押せば通信が始まり、左ボタンを押せば通信が終了するなどといったことにも利用することが可能になります。

4.6.3 キーボードイベント

Processing では、キーボードに関してもマウスと同じようにイベントとして処理することができます。但し、マウスに比べてキーの種類が増えるので条件として用いる場合には非常にたくさんの条件を設定することが可能です。以下に、代表的なキーボードイベントを記載します。

`void keyPressed(){`

```
// キーが押されたときに{ }内の命令が実行されます。  
}
```

`void keyReleased(){`

```
// キーが離されたときに{ }内の命令が実行されます。  
}
```

(キーボードイベントの Point)

キーボードイベントの中では以下の変数でキーの種類を獲得することができます

- `key`: 基本この変数を使います。押されたキーの文字列を得ることができます。
- `keyCode`: 方向キーなどは `key` では取得できない。そのため `keyCode` を用いる必要があります。

以下のようなキーの種類を取得することができます。

- `ALT, CONTROL, SHIFT, BACKSPACE, ENTER, RETURN, TAB, ESC, DELETE`

キーボードイベントでもマウスイベントのようにキーの種類に応じて設定を変更することができます。自分で簡単なプログラムを作ってみましょう。

5. グラフィック

今まで Processing を使用していくつかのグラフィックに関する関数を利用してきましたが、Processing には他にもたくさんのグラフィック用関数があります。これらと計測データを使用することでより視覚的な情報を得ることが可能になります。

・今までに利用した主な関数(4.3.4 サンプルプログラム 2 参照)

```
background( )  
fill( )  
text( )
```

5.1 受け取った値によって背景の色を変更する

さて、今まで利用した関数を使って簡単なグラフィカルなツールを作ってみましょう。ここでは、目標と、ヒントだけ記載しますので自分で考えてプログラムを作成してみてください。

目標：Arduino から値を受け取り、受け取った値に応じて背景の色が変化するツールの作成

(ヒント 1) Arduino のプログラムは 4.2 サンプルプログラムを流用して OK

(ヒント 2) Processing のプログラムは 4.3.4 サンプルプログラムを利用すると作りやすい

(ヒント 3) background()関数には 0~255 の数値を入れることで背景色が変わります

(ヒント 4) 終了条件(100 回動作するとプログラムを抜けるなど) を指定しなければプログラムを止めない限り、動き続ける。

5.2 いろいろな関数

5.2.1 矩形（長方形、正方形）

矩形を描くためには以下の関数を使用します。

rect(X, Y, WIDTH, HEIGHT);

X：画面左上から右方向へ指定した距離だけ始点を移動する

Y：画面左上から下方向へ指定した距離だけ始点を移動する

WIDTH：矩形の横幅

HEIGHT：矩形の高さ

例えば、rect(100,100,200,200);とすると以下の図 3 のようになります。ここで気を付けてほしいのは、X と Y は、画面左上からの指定となっており、矩形の左上角がその指定場所に来るようになっています。また、rect(250,250,100,100);を続けて書いた場合、図 4 のように上書きされます。

fill()関数を用いれば、内部の色を変更することもできます。fill()関数を使用していない場合は、デフォルトでは白色になります。

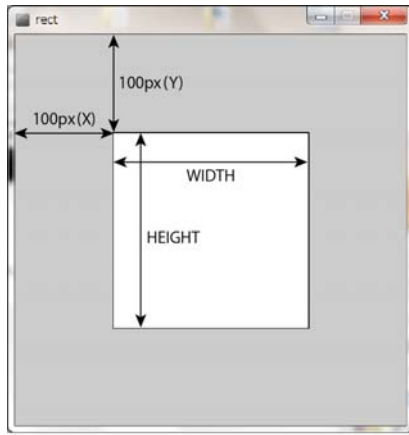


図3. 矩形を描く(画面サイズ 400×400)

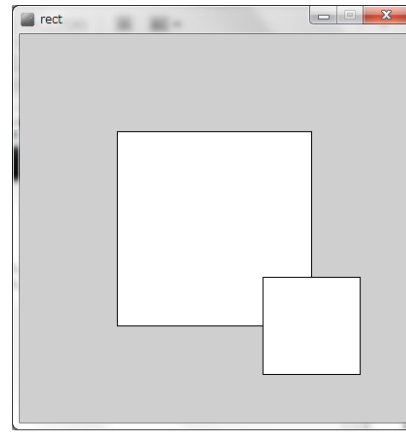


図4. 矩形を2つ描いたケース

5.2.2 円形(楕円、正円)

円形を描く場合には、以下の関数を使用します。

ellipse(X, Y, WIDTH, HEIGHT);

X, Y : 円の中心の X 座標、Y 座標を表す

WIDTH : 横の直径

HEIGHT : 縦の直径

WIDTH と HEIGHT が同じなら正円を描くことが可能。

5.2.2 点

点を描くためには、以下の関数を使用します。

point(X, Y);

X : X 座標 (左上から右方向)

Y : Y 座標 (下方向)

この関数を利用すると、 $(x,y) = (X, Y)$ に点を打つことができます (画面左上が $(x,y) = (0,0)$ になる)。

5.2.3 直線

直線を描く場合には、以下の関数を使用します。

line(X1, Y1, X2, Y2);

始点 $(X1, Y1)$ から終点 $(X2, Y2)$ まで直線を引くことができます。

なお、直線の色を変える際には、`stroke()`を利用すればよい。色の指定方法は他の `background` や `fill` で

設定する場合と同じです。

5.3 配列

プログラムを記載する際に複数の同じような記述を行う必要が出てくる場合があります。こういったときに配列といった概念を覚えておくとうまく記述することができる場合があります。以下に簡単に説明します。

5.3.1 配列の定義方法

配列の定義方法には大きく分けて2つあります。

(a) 配列を宣言して初期値を代入する方法

この方法は、以下のように定義します。

```
型 [] 配列名 = {要素 1, 要素 2, ...};
```

例えば、String 型の配列として、aisatsu というものを作成するケースを考えてみると、

```
String [] aisatsu = {"おはよう","こんにちは","こんばんわ"};
```

というように記載することができます。それぞれ先頭から aisatsu[0],aisatsu[1],aisatsu[2] という箱の中にそれぞれの値が格納されるようになります。図5を見てもらえば大体の様子ができるかと思えます。

今まで取り扱ってきた変数では、基本的に箱が1個しかないので図6のように3つの変数を用意してやる必要があります。



図5. 配列の概念

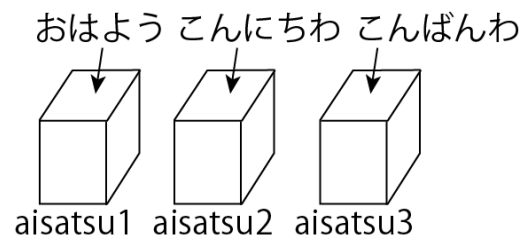


図6. 変数を用いた場合

(b) 配列を宣言して要素数を指定する方法

この方法では、以下のように定義します。

```
型 [] 配列名 = new 型 [要素数];
```

例えば、要素数10個のint型のxという配列を作成する場合は以下のように記載します。

```
int [] x = new int [10];
```

ここで注意しないといけないのは、10個用意した場合、使用できるのはx[0]~x[9]までの10個です。10個なのでx[1]~x[10]まで使用できると勘違いするケースが多いです。通常、プログラミングなどでは0から始まるのでx[0]~x[9]の10個になります。注意して要素数を設定してください。

5.3.2 配列の参照方法

配列の中身を参照する場合には、以下のように記載します。

配列名[番号];

先ほど 5.3.1(b)で説明しましたが、番号は0から始まることに注意してください。使用例としては、以下のように使います。

```
println(aisatsu[0]);  
println(aisatsu[1]);  
println(aisatsu[2]);
```

5.3.3 配列に値を代入する方法

配列に値を代入したい場合には、以下のように記載します。

配列名[番号] = 値;

但し、代入する値は配列の型にあったものを代入するようにしてください。ちがう型のものをいれてしまうとエラーの原因になります。

6. 最後に

今までに記載してきたことを利用すれば課題の作成を行えると思います。いろいろと考えて自分の満足のいくものを作り上げてください。