1. LED の点灯プログラム

```
// 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
}
```

2. LED の点滅プログラム

```
void setup() {
 pinMode(13, OUTPUT);   //13 番ピンを出力として使用
}
void loop() {
 digitalWrite(13, HIGH);  // 13 番ピンに HIGH（5V）を出力する
delay(1000);              //  1000ms 待つ（1 秒待つ）
digitalWrite(13, LOW);  // 13 番ピンに LOW（0V）を出力する
delay(1000);              //  1000ms 待つ（1 秒待つ）
}
```

このプログラムは 1 秒点灯した後、1 秒消灯するといったことを繰り返すプログラムです。すなわち、1 秒間隔で点滅します。delay()の()内の数字を変更すれば、間隔を変更することができます。

2.1 LED の点滅プログラム（修正版）

```
// 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
// 整数の変数 Waittime に 1000 を代入
int LEDpin =  13;
int Waittime = 1000;

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(5 V)を出力
delay(Waittime);                  //  Waittime 分待つ。このケースでは 1000ms
digitalWrite(LEDpin, LOW);      // LEDpin に LOW(0 V)を出力
delay(Waittime);                  //  Waittime 分待つ。このケースでは 1000ms
}
```

このケースでは、変数を使うことで回路の構成が変更されて接続されるピンが変わったとしても変数の部分の数字を変更することで対応することができます。いろいろな参考書や資料を読みましたが、大体こういった書き方が推奨されています。

3. 直流安定化電源からアナログ入力を読み取るプログラム（その 1）

```
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
}
```

サンプルプログラムと同じです。変数名を分かりやすいように変更してもよいかと思います。この

ケースでは、analn0 の値は 0～1023 になるので、実際の値を知りたいときには、変換してやる必要があります。但し、Processing などとやり取りする場合には、実際の値に変換せずに、最後に Processing 側でデータの整形を行うほうが効率がよいので注意してください。

3.1 実際の値に変換して出力するプログラム(map 関数不使用）

```
int analn0;                 // 変数の宣言（アナログ入力の値を入れるための変数）
double val;                 // 変数の宣言（変換した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
val = 5/1024.0*analn0;      // 変換（ここですべて int 型にしておくと不具合あり）
Serial.println(val);        // パソコンに val の値を転送（改行あり）
delay(100);                 // 100ms 待つ
}
```

注意点としては、変換するところで、1024.0 としているのは、こうすることで double 型の状態にしています。ちなみに、val の型は float 型でも double 型でもかまいません。また、5 の部分を例えば,一番最初の行に int analn0, v=5;という風に記載し、変換する行を val=v/1024.0*analn0;としても問題ありません。

後、デフォルトでは小数を出力する際には、小数点第 2 桁までしか出なくなっています。任意に設定したいときは、

Serial.print(val,桁数);

とすれば、自分で設定した桁数まで出すことができます。例えば、Serial.println(val,4);とすると小数点第 4 桁まで出てきます。

3.2 実際の値に変換して出力するプログラム(map 関数使用）

```
int analn0,val0;                    // 変数の宣言（アナログ入力の値を入れるための変数）
double val1;                        // 変数の宣言（変換した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
val 0 = map(analn0,0,1023,0,5000);  // 0～1023 の値を 0～5000 に変換（mV に変換）
val1 = val0/1000.0;                 // V にするために 1000 で割る
Serial.println(val1);               // パソコンに val1 の値を転送（改行あり）
delay(100);                         // 100ms 待つ
}
```

今回のケースでは map 関数と呼ばれるものを使用しています。これは、入力値の範囲を指定の範囲に変換できる関数です。但し、注意点として変換された値は整数であるという点です。0～5V にする場合、変換範囲も 0 と 5 を入れたくなりますが、整数値で出てくるので 1024 分割からただの 6 分割に変換されてしまいます。そこで、今回は 0～5000 を変換する範囲に指定してあります。こうすることでそのまま map 関数で変換した値を出力すれば mV として使えます。今回のケースでは、V に変換するので 1000.0 で割ってあります。

3.3 複数の入力を入れる場合（配列未使用&2 入力）

```
int analn0, analog1;                // 変数の宣言（アナログ入力の値を入れるための変数 2 つ）

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
analn1=analogRead(1);               // アナログ 1 番ピンの入力値を analn1 に代入
Serial.print(analn0);               // パソコンに analn0 の値を転送（改行なし）
Serial.print(",");          // パソコンに「,」転送（改行なし）
Serial.print(analn1);               // パソコンに analn1 の値を転送（改行なし）
Serial.println();                   // パソコンに改行を転送
delay(100);                         // 100ms 待つ
}
```

複数の入力がある場合には、Serial.print を使って並べるのが基本です。ただ、並べるだけでは繋がってしまうので、例では「,(コンマ)」を途中に入れることで分かりやすくしています。すべての出力が終了したときには、Serial.println に何の数値も入れないで記載することで、改行のみを行ってくれます。この結果として、以下のようにシリアルモニタには表示されます。

[実行結果例(analn0 と analn1 にはそれぞれその時に読み込んだ入力値が出力されます）]
analn0,analn1
analn0,analn1
・
・
・

4. データロガー用プログラム

4.1 Arduino 用プログラム（指導書 4.2 サンプルプログラムの解答)

```
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= analogRead(0);                         // アナログ 0 番ピンの入力を sensor1 に代入
senout=map(sensor1,0,1023,0,255);               // データを 1 バイト(0~255)に圧縮
if(Serial.available()>0){                       // データを受信したら次の行の内容を実行
        Serial.write(senout);                   // データをシリアル通信で送信(バイトデータ)
 }
delay(100);
}
```

このプログラムでは、何をしているかというとアナログ 0 番ポートから読み込んだ入力を map 関数を用いて 1 バイトまで圧縮しています。そのデータを、相手先（今回は Processing）に通信するプログラムです。但し、通信するタイミングは、相手先から通信してほしいという伺いが来てからになります。

4.2 Processing 用プログラム（カウントバージョン)

```
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
double senout2;                                // senout を 0~5V に変換してデータを代入する変数
int count=0;                                   // カウント用変数

PrintWriter output;                            // PrintWriter 型のオブジェクトを宣言

void setup () {                                // 初期条件設定
  size(400, 400);                              // 描画エリアの設定（このケースでは 400×400 ピクセル）
  background(255);                             // 描画エリアの背景色（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);

output = createWriter("log.txt");              // ファイル名 log.txt でファイルを開く
}



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
    output.flush();                            // 出力バッファに残っているデータを全て書き出す
    output.close();                            // ファイルを閉じる
    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
    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。
    senout2= 5/256.0*senout;                   // 読み取ったデータを実際の値に変換
    output.println(senout2);                   // 変換したデータをファイルに書き込む
    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（こういうことをインクリメントという）

}
}
```

カウント回数分ログを保存するプログラムです。太字の部分が Processing→Arduino への通信を開始する最初のきっかけになります。if(count>100){・・・}の 100 の部分の数字を任意の数字に変更すれば、好きな回数分ログを保存できるようになります。

4.3 Processing 用プログラム（マウスクリックバージョン)

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

float senout,senout2;
int count=0;
int flag=0;

PrintWriter output;

void setup () {
 size(400, 400);
 background(255);

 println(Serial.list());
 myPort = new Serial(this, Serial.list()[1], 9600);
 output = createWriter("log.txt");
}

void draw () {
}

void serialEvent (Serial myPort) {        // Processing が myPort でデータを受信した際に作動する関数
 if(myPort.available()>0 && flag==1){     // myPort 内にデータがあって且つ、変数 flag が 1 のとき
  senout=myPort.read();                   // 変数 senout に myPort のデータを代入
  senout2= 5/256.0*senout;                // 変数 senout2 に実際の値に変換したデータを代入
  output.println(senout2);                // ファイルに senout2 の内容を改行有りで書き込む
  myPort.clear();                         // myPort 内のデータをクリアする
  myPort.write(255);                      // myPort に 255 を送る（すなわち次のデータ受信する準備が
                                          // できたことを示している
```

```
 }
}

void mousePressed(){          // マウスが押されたときに作動する
 if(mouseButton == LEFT){     // マウスの左ボタンが押された場合
  println("Mouse Click");     // マウスが押されたことをあらわすためにメッセージを送信
  if(flag==0){               // 変数 flag が 0 の場合
   flag=1;                   // flag を 1 にする
   myPort.clear();           // myPort 内のデータをクリアする
   myPort.write(255);        // myPort に 255 を送信する
  }

// 上記 if 文は、最初マウスの左ボタンを押すと if 文の最後まで実行し、関数 void mousePressed()を
// 抜ける。その後、再度マウスの左ボタンが押されるまでは void draw()内の命令を繰り返し行う。
// 再度マウスの左ボタンが押されたとき、flag は 1 のままなので、次の行の else if が実行される。

  else if(flag==1){           // flag が 0 でなく、flag が 1 の場合
   flag=0;                    // flag を 0 にする
   output.flush();            // 残っているデータを全てファイルに書き出す
   output.close();            // ファイルを閉じる
   println("Finish button");  // 再度マウスの左ボタンが押されたことを出力。
  }
 }
}
```

カウントバージョンでは、決められた回数分しか保存できませんでしたが、このバージョンではマウスの左ボタンを押すことでログの保存が開始され、再度左ボタンを押すとログの保存を終了するようになっています。カウントバージョンでは記録開始のきっかけになる部分を太字にしていますが、このプログラムについても太字部分が記録開始のきっかけになる部分になっています。

5. グラフィック

次にグラフィックを用いたプログラムの例を記載します。

5.1 Arduino 用プログラム

基本的に 4.1 で記載したプログラムを用いれば今回のような 1 入力のケースは問題ありません。

```
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= analogRead(0);                         // アナログ 0 番ピンの入力を sensor1 に代入
senout=map(sensor1,0,1023,0,255);               // データを 1 バイト(0~255)に圧縮
if(Serial.available()>0){                       // データを受信したら次の行の内容を実行
        Serial.write(senout);                   // データをシリアル通信で送信(バイトデータ)
 }
delay(100);
}
```

5.2 入力の値によって背景色を変化させる

以下にサンプルプログラムを記載します。データロガーのカウントバージョンで作成したものとほとんど変わりはありません。データを保存する部分を背景色を変化させるように変更したのとカウントの部分を flag という変数で置き換え，一番最初だけ通信のために使用しています。

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

int senout;
int flag=0;                 //一番最初の通信を行うためのフラグ用変数（あくまでも安全用）

void setup () {
  size(400, 400);
  background(255);
  println(Serial.list());
  myPort = new Serial(this, Serial.list()[1], 9600);
}
```

```
void draw () {
 if(flag==0){                    //flag=0 のとき、すなわち一番最初の通信が行われていないとき
  myPort.write(255);             //データを送って Arduino に通信準備ができたことを知らせます。
 }

 if(myPort.available()>0){
  senout=myPort.read();
  background(senout);            //Arduino から送られてくるデータは 0～255 の数値になっているので
                                 //そのまま background の中に入れて使用することが可能です。
  myPort.clear();
  myPort.write(255);
  flag=1;                        //flag=1 とすることで、上記の flag=0 の if 文を回避します

 }
}
```

5.3 入力によって円の大きさや色を変化させる

円についても背景のときと同じように作成できます。Arduino から受け取った値を背景用の関数の background に入れるか、円を描画する関数 ellipse と円の内部の色を塗る fill に入れるかの違いだけです。

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

int senout;
int flag=0;

void setup () {
 size(400, 400);
 background(255);
 println(Serial.list());
 myPort = new Serial(this, Serial.list()[1], 9600);
}

void draw () {
 if(flag==0){
  myPort.write(255);
 }
```

```
 if(myPort.available()>0){
  background(255);                    //重ね書きを防止するために背景(白)でまず塗りつぶしています
  senout=myPort.read();
  fill(senout);                       //図形内部を senout の数値で塗りつぶします
  ellipse(200,200,senout,senout);     //円の中心を(200,200)とし、縦横の直径を senout としています
  myPort.clear();
  myPort.write(255);
  flag=1;
 }
}
```

5.4 直線を使ってグラフを作成する

　ここでは、直線を描く関数を使ってグラフを作成します。配列などを使い、今までとは少し違ったものになります。

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

static int myGraphWidth =800;
static int myGraphHeight=600;

int [] graphValue = new int[myGraphWidth];
int senout;
int flag=0;
int count=0;

void setup () {
 size(myGraphWidth,myGraphHeight);
 background(255);
 println(Serial.list());
 myPort = new Serial(this, Serial.list()[1], 9600);
}

void draw () {
 if(flag==0){
  myPort.write(255);
 }
```

```
  if(myPort.available()>0){
    senout=myPort.read();
    graphValue[count]=senout;
    myPort.clear();
    myPort.write(255);
    count++;
    flag=1;

    if(count==myGraphWidth-201){
      count=0;
      background(255);
    }
    else if(count>1){
      line(count+100,height-graphValue[count-2],count+101,height-graphValue[count-1]);
    }
  }
}
```