プログラミング演習 Ⅱ
フィジカルコンピューティング

第3回 Processing連携，サーボモータ，LEDマトリクス

中村，小松，小林，鹿喰

（監修：橋本直）

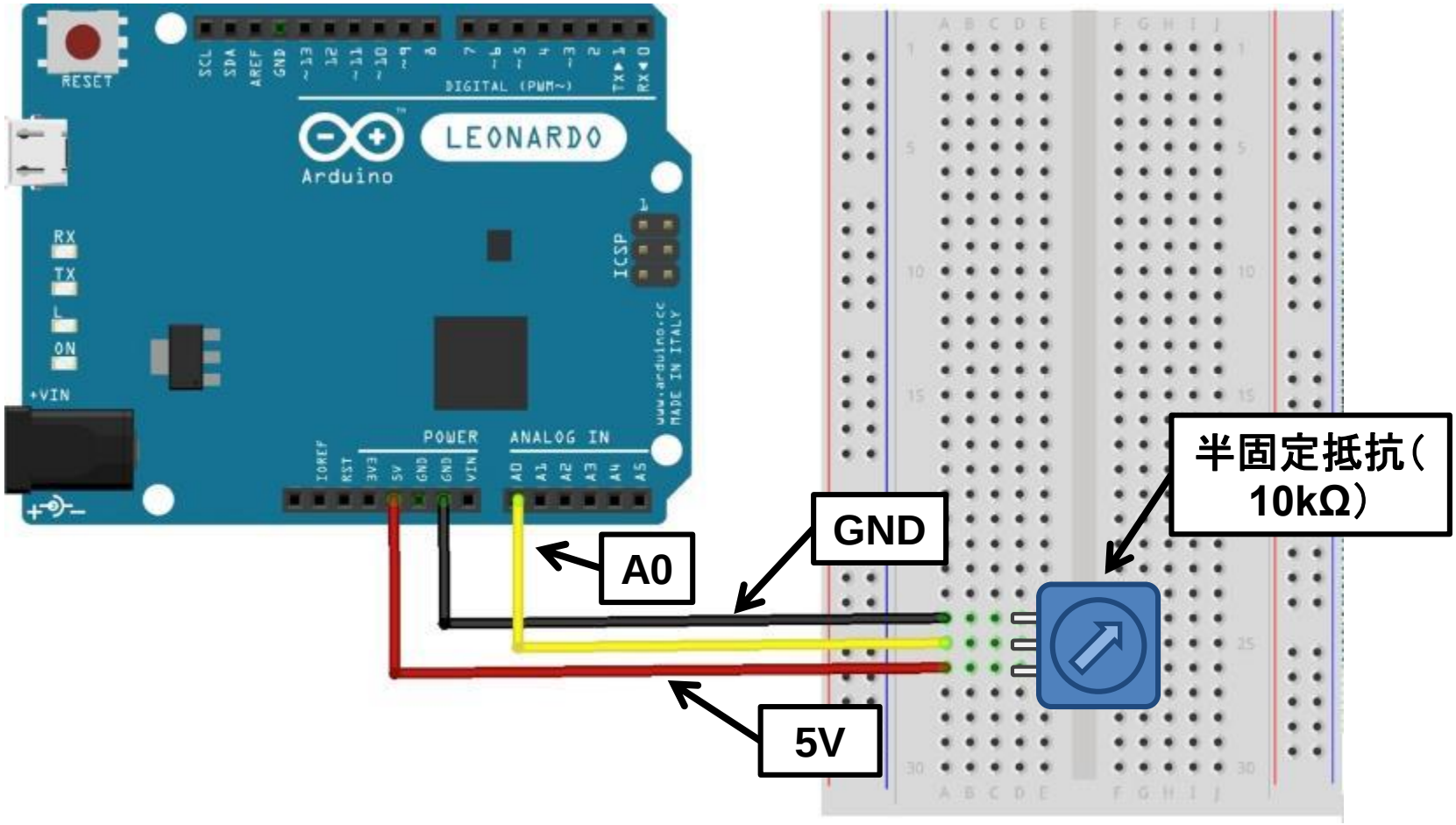
※マトリクスLEDのプログラムをLeonardo対応に修正した版

今日の内容

- **ProcessingとArduinoの連携**
  - データの受信（Arduino→Processing）
    - センサからの入力値をProcessingで可視化する
  - データの送信（Processing→Arduino）
    - Processingからの指令でLEDの点灯を制御する
- **サーボモータの使い方**
  - モータの回転角度を制御する
- **圧電スピーカの使い方**
  - 音を鳴らす＆振動を検知する
- **フォトリフレクタの使い方**
- **マトリクスLEDの使い方**

※Processingは最新版（演習で使ったバージョン）を使用する

ボリューム（半固定抵抗）の変化を読み取る回路



※ボリュームの真ん中の端子を入力（A0）に、
一方の端をGND、もう片方の端を5Vにつなぐ。

Arduinoのプログラム
（センサで読み取った情報をPCに送信）

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

void loop() {
    int value = analogRead(0);
    Serial.write(value/4);
}
```

シリアル通信を開始する

アナログ入力（A0）を読み取る

バイナリデータを送信
（値の範囲を0～255にするために4で割っている）

命令の意味

- **Serial.write(データ)**
  - シリアル通信でデータをバイナリ形式で送信する
  - 数値、文字、文字列を送れる

```
Serial.write(10);
Serial.write('a');
Serial.write("hashimoto");
```

Processingのプログラム（受信したデータを可視化）

```
import  processing.serial.*;

Serial  serial;
int  input;

void setup() {
    size(300,200);
    println( Serial.list() );
    serial = new Serial( this, Serial.list()[0], 9600 );
}

void draw() {
    background(0);
    rect( 10, 50, input, 50 );
}

void serialEvent(Serial port) {
    input = port.read();
}
```

「シリアル通信用のライブラリを使用します」という意味

シリアル通信用の変数

シリアルポートの一覧を表示

シリアル通信を開始

四角形の描画（inputで幅が変化）

データを受信したときのイベント

データの読み込み

命令の意味

- **serial = new Serial(this, ポート番号, 通信速度);**
  - 使用するシリアルポートの番号（PCによって異なる）
    - Windowsの例：“COM1” とか “COM2”
    - Macの例: “/dev/tty.usbmodem1411” など
    - この文字列はSerial.list()で取得できる
  - Serial.list()[0]は0番目のシリアルポートの番号
    うまくつながらない時は、0を他の数字（1, 2,3…）に変更
  - 通信速度はArduinoと同じにする

命令の意味

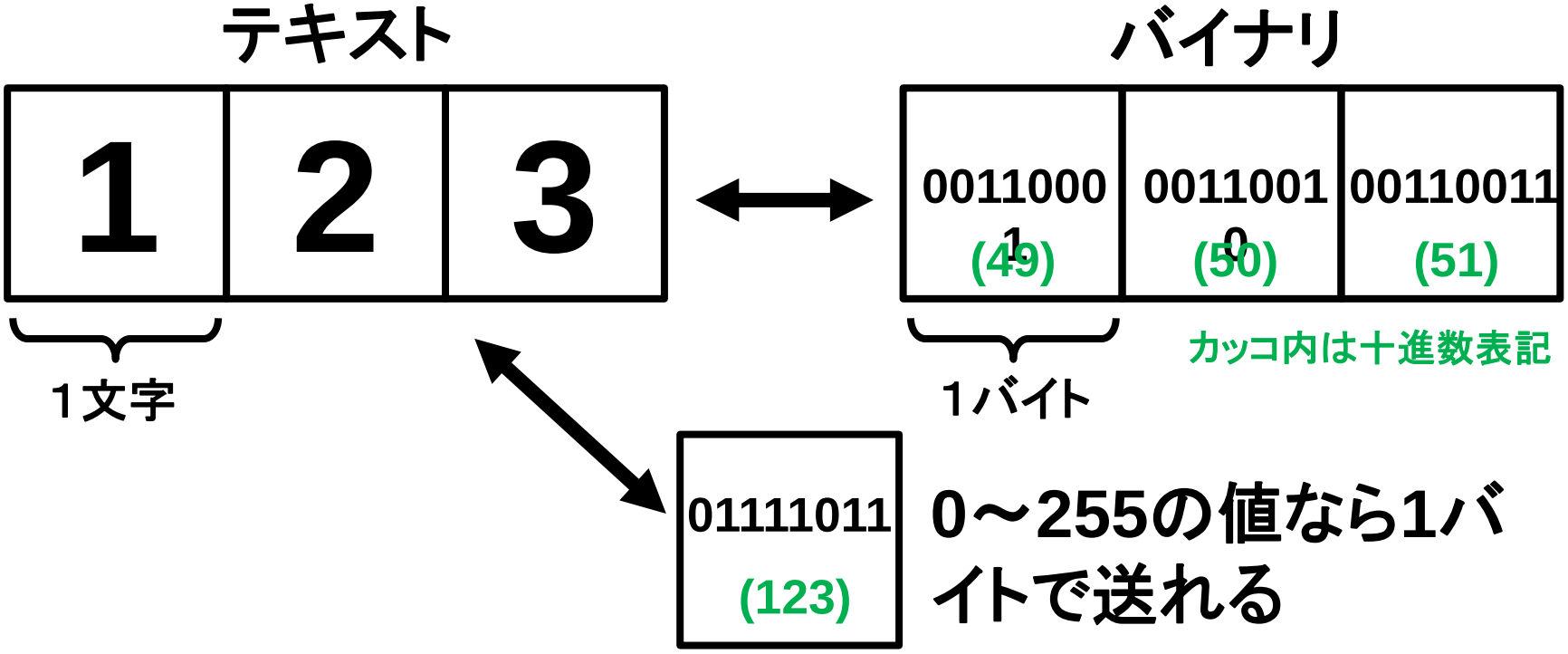
```
void serialEvent( Serial port ) {
   input = port.read();
}
```

- serialEvent()はシリアルポートにデータがきたときに自動的に呼び出される関数。引数（port）はデータがきていたシリアルポート。

- .read() は受信データの先頭から1バイト（0～255）読み取る命令。

テキストとバイナリ

- テキストデータ
  - 文字によるデータ表現
- バイナリデータ
  - 数値列（バイト列）によるデータ表現。
  - コンピュータ上での内部表現。バイナリ＝binary（二進法）

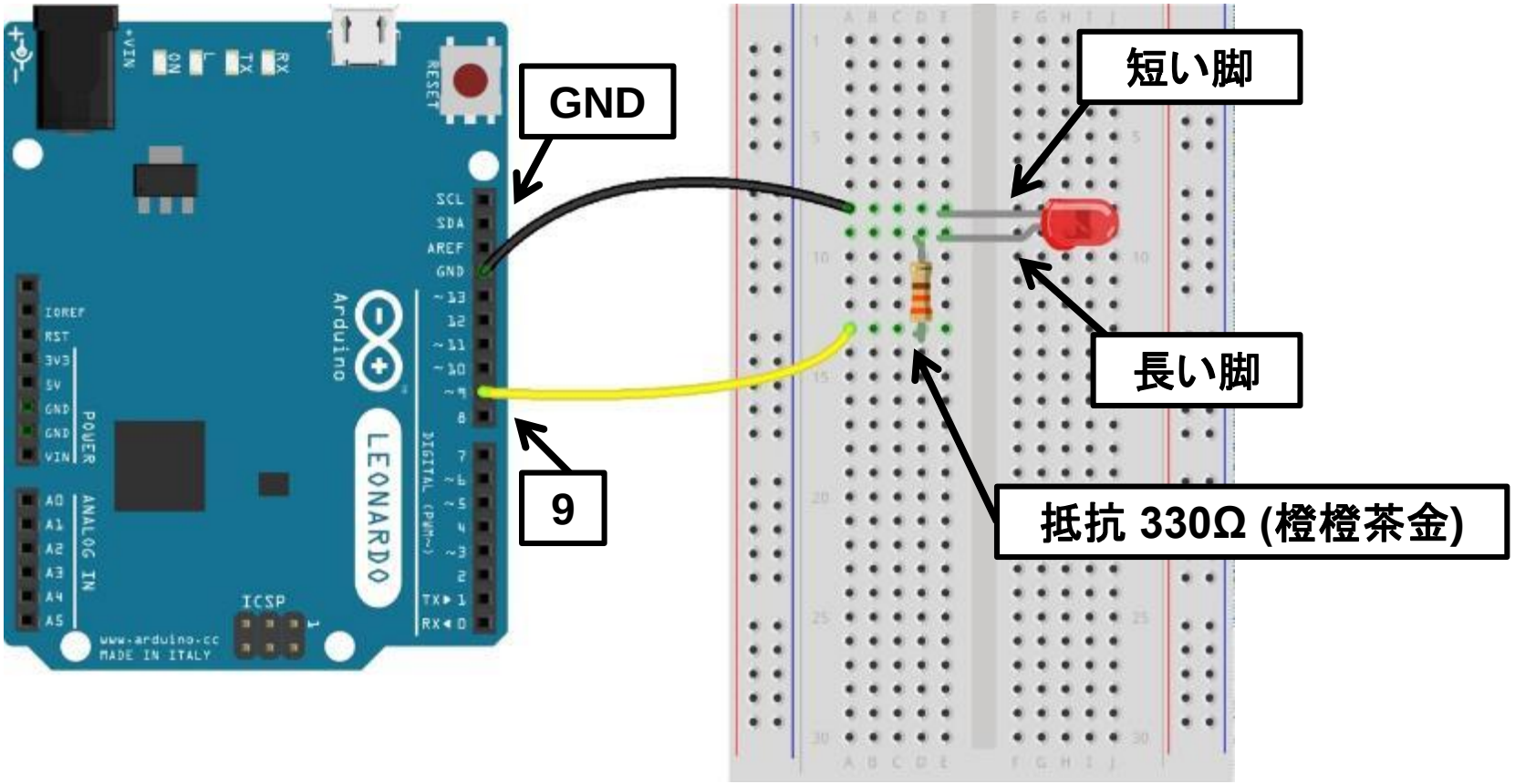


１バイトずつのやりとり

- １バイト＝８ビット
  - ２進数で、00000000～11111111
  - １０進数で、0～255

- 「123」というデータを送りたいとき
  - テキストで送ると、「1」「2」「3」の３バイトかかってしまう。
  - バイナリで送れば、１バイトで済む。
    （ただし、１バイトで表現できる情報は0～255）

LEDを点灯させる回路



Arduinoのプログラム（コマンドでLED制御）

```
void setup() {
    Serial.begin( 9600 );
    pinMode( 9, OUTPUT );
}

void loop() {
    if ( Serial.available() > 0 ) {
      int data = Serial.read();
      if ( data == 'a' ) {
        digitalWrite( 9, HIGH );
      }
      else if ( data == 'b' ) {
        digitalWrite( 9, LOW );
      }
    }
}
```

シリアル通信を開始する

9番ピンを出力に設定

データがきているかチェック

データを1バイト読み込む

データが「a」だったら
LEDをONにする

データが「b」だったら
LEDをOFFにする

Processingのプログラム（マウスクリックでLEDをON/OFF）

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

void setup() {
  size(400,400);
  println( Serial.list() );
  serial = new Serial( this, Serial.list()[0], 9600 );
}

void draw() {
  background(0);

  if ( mousePressed ) {
    serial.write('a');
  } else {
    serial.write('b');
  }
}
```

「シリアル通信用のライブラリを使用します」という宣言

シリアル通信用の変数

シリアルポートの一覧を表示（無くてもOK）

シリアル通信を開始

もしマウスボタンが押されていたら

「a」という文字を送信

ボタンが押されてないときは「b」という文字を送信

命令の意味

- **Serial.write(データ)**

  シリアル通信でデータをバイナリ形式で送信する

  数値、文字、文字列を送れる

  serial.wrirte(10);

  serial.wrirte('a');

  serial.write("hashimoto");

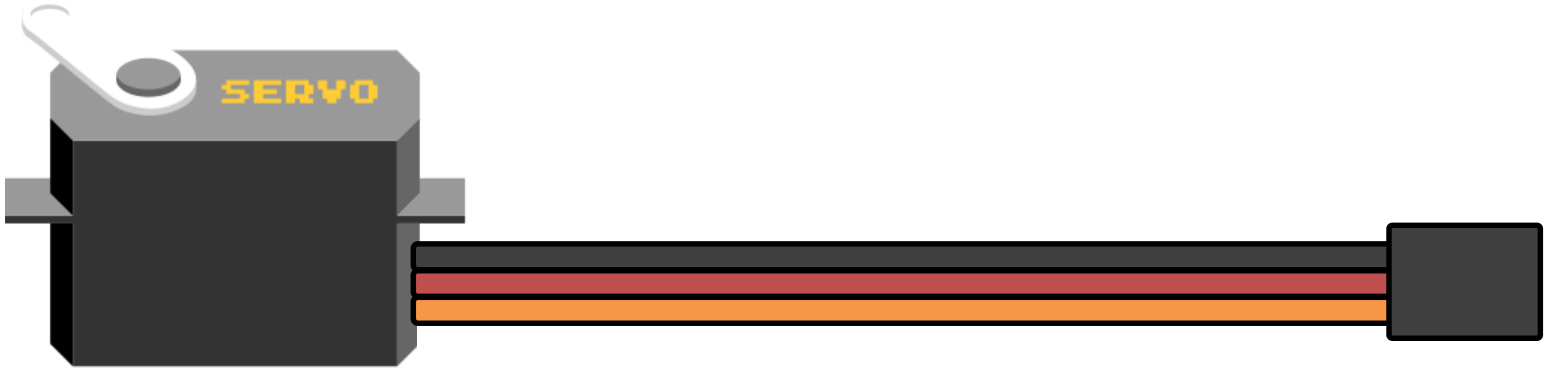
サーボモータ

- 与えられた目標値に追従するような制御を自動で行うモータ
- サーボ(Servo) の語源はラテン語の"servus"(英語のslave・servantの意）

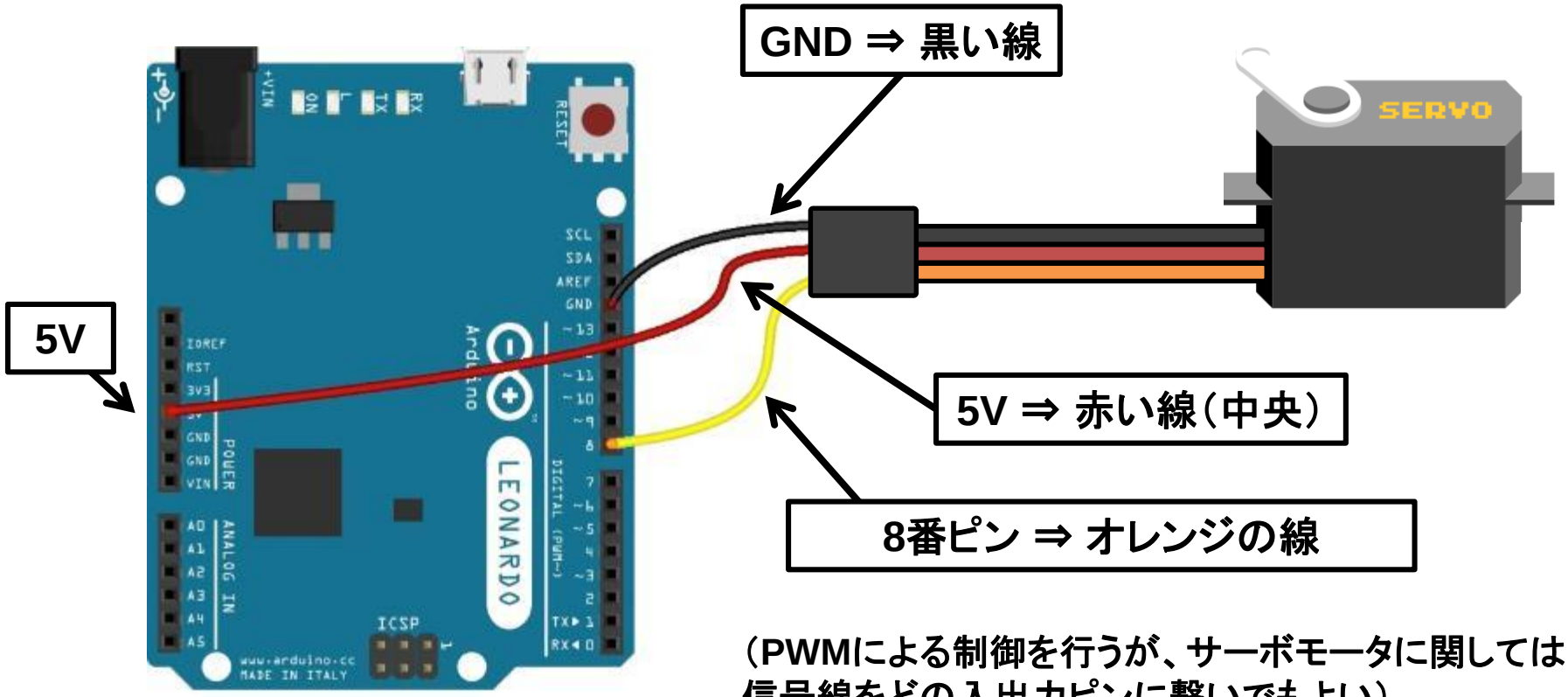
- キットに入っているサーボモータ
  - 5V駆動
  - 回転角度：0～180度

サーボモータの使い方

- 3本の線
  - 黒・茶色: GND（電源－）
  - 赤: 5V（電源＋）
  - オレンジ: 信号線
    - PWM（パルス幅変調）の信号によって角度制御できる
    - パルス幅と角度が対応している（製品によって異なる）
      例）パルス幅が1.8msのとき90度、3.3msのとき180度



サーボモータを動かす回路



（PWMによる制御を行うが、サーボモータに関しては、信号線をどの入出力ピンに繋いでもよい）

※モータを駆動するときは大きな電流が流れるため、普通は外部電源から電力供給を行う。今回は小型のモータを使用しているため、マイコンボードから電力供給している。

Arduinoのプログラム（サーボモータ制御）

```
#include <Servo.h>

Servo servo;

void setup() {
    servo.attach(8, 400, 3300);
}

void loop() {
    servo.write(0);
    delay(1000);
    servo.write(90);
    delay(1000);
    servo.write(180);
    delay(1000);
}
```

「サーボモータ用のライブラリを使います」という宣言

サーボモータ用の変数

サーボモータを使う準備

サーボモータの角度を 0度 にする

サーボモータの角度を 90度 にする

サーボモータの角度を 180度にする

※回転に時間がかかるので各1秒待ち時間を入れている

命令の意味

- **#include <ライブラリ名>**
  - Arduino（C言語）において外部プログラム（ライブラリ）を使用するときの宣言文。ProcessingやJavaにおけるimportに相当。

- **Servo**

  .attach(ピン番号, 最小パルス幅[μs], 最大パルス幅[μs])

  サーボモータの設定を行う

  .write(角度)

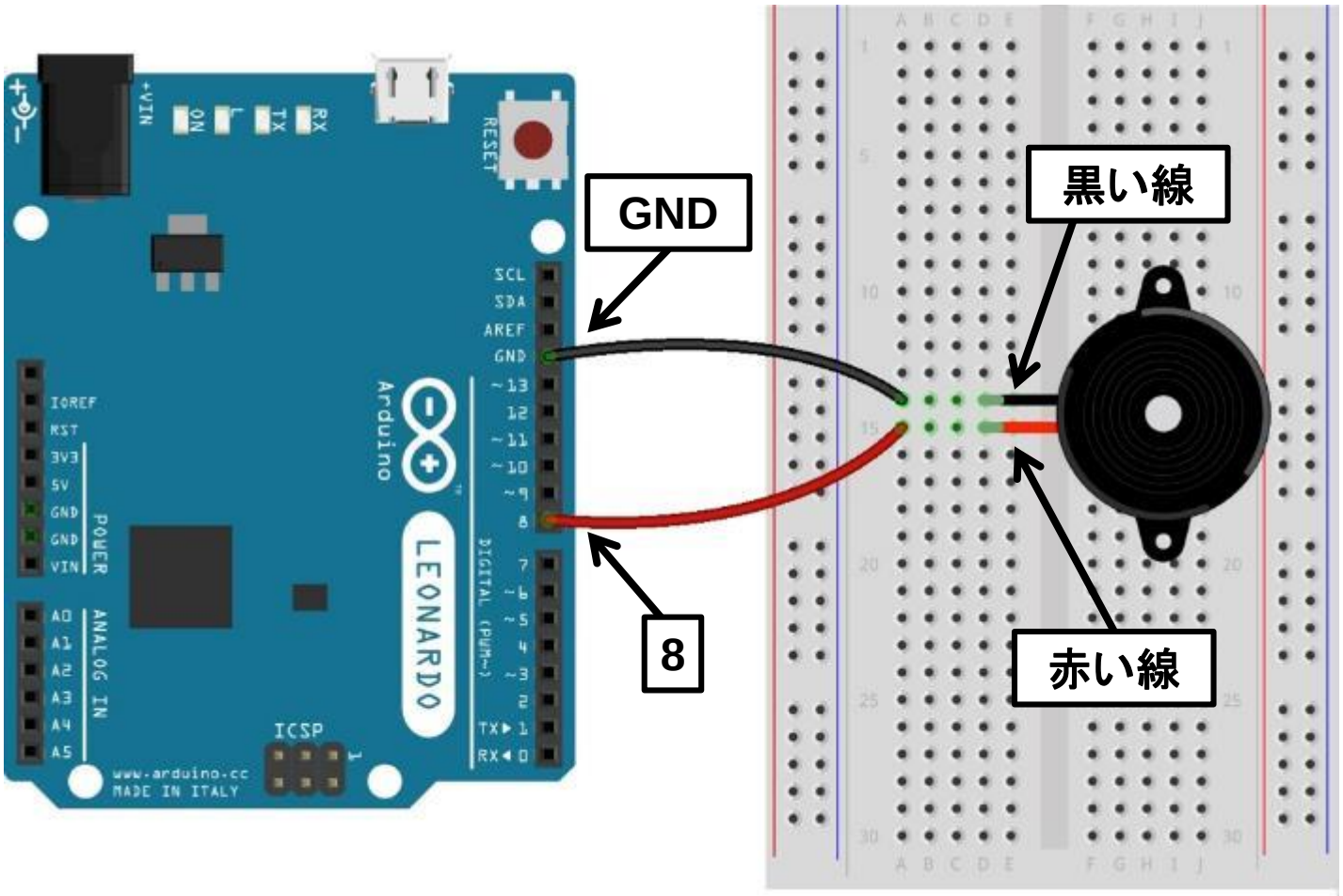
  指定した角度に回転させる

圧電スピーカ

- 圧電素子を用いたスピーカ
- 音を出したり、振動を検知したりできる
  - 音を出すには、数kHzのパルス信号を与える

- **圧電素子**
  - 力を加えると、圧力に比例した電圧が発生する素子
  - 逆に電圧をかけると、力が発生する

圧電スピーカで音を出す回路



圧電スピーカを直接Arduinoに挿してもよいが、線が細くて接触がよくないのでブレッドボードを介して接続する。

Arduinoのプログラム（圧電スピーカで音を出す）

```
int  len = 300;
int  pin = 8;

void setup() {
   pinMode(pin, OUTPUT);
}

void loop() {
   tone(pin, 262, len) ;
   delay(len) ;
   tone(pin, 294, len) ;
   delay(len) ;
   tone(pin, 330, len) ;
   delay(len) ;
   delay(3000);
}
```

音の長さ（300ms

圧電スピーカが接続されている端子

8番ピンを出力に設定

「ド」の音を出す

「レ」の音を出す

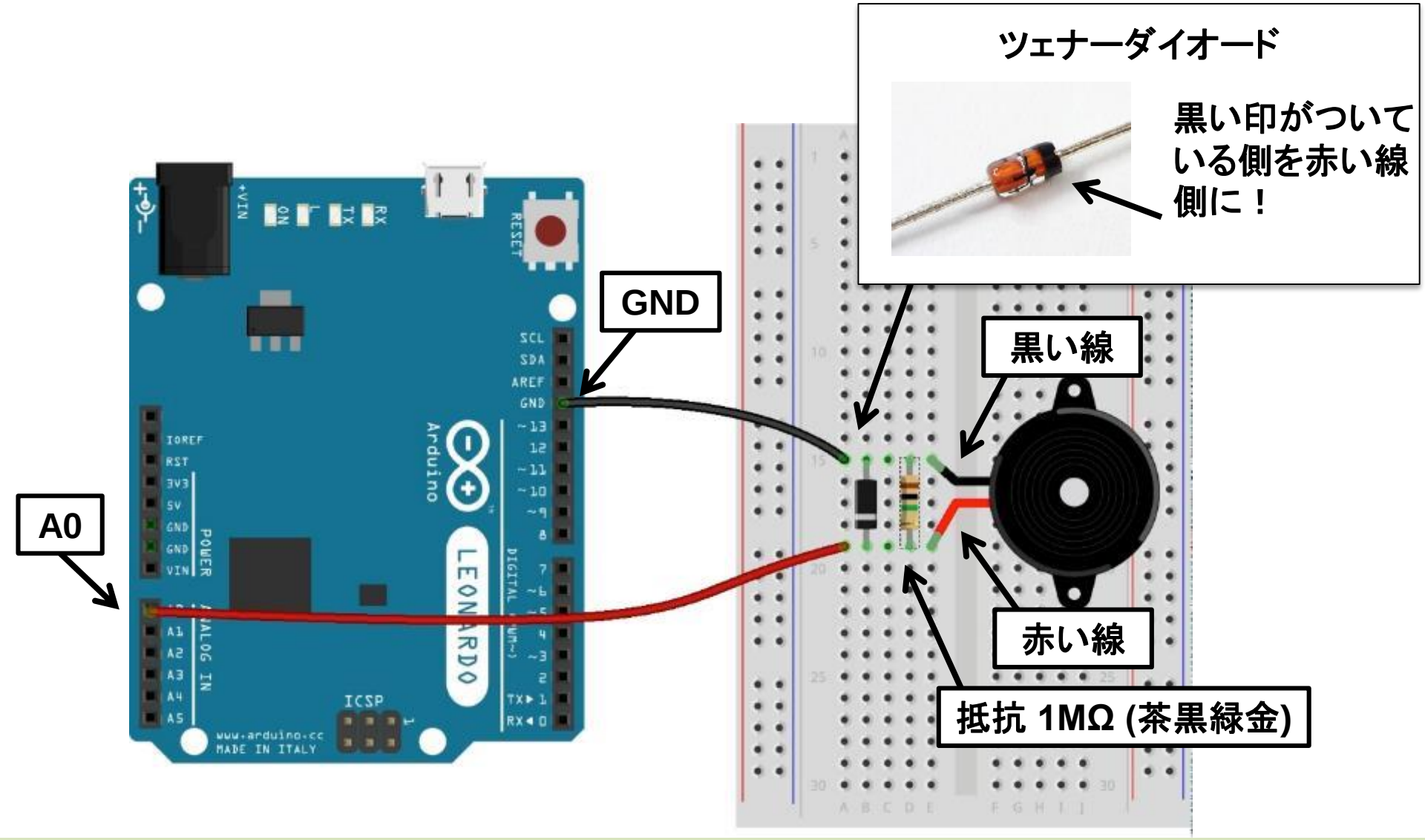
「ミ」の音を出す

命令の意味

- tone(ピン番号, 周波数[Hz])
- tone(ピン番号, 周波数[Hz], 時間[ms])
  - 指定した周波数のパルス（デューティ比50%）を出力する
  - 時間を指定しなかった場合、noTone()を実行するまで動作し続ける

ド	262Hz	ソ	392Hz
レ	294Hz	ラ	440Hz
ミ	330Hz	シ	494Hz
ファ	349Hz	ド	523Hz

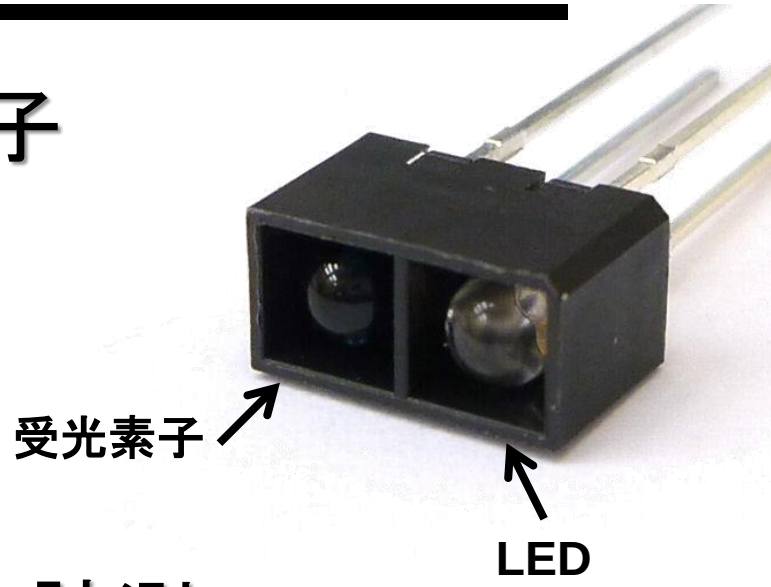
圧電スピーカで振動を検知する回路



センサ入力をProcessingで可視化するプログラムを使って動作を確認しよう！

フォトリフレクタ

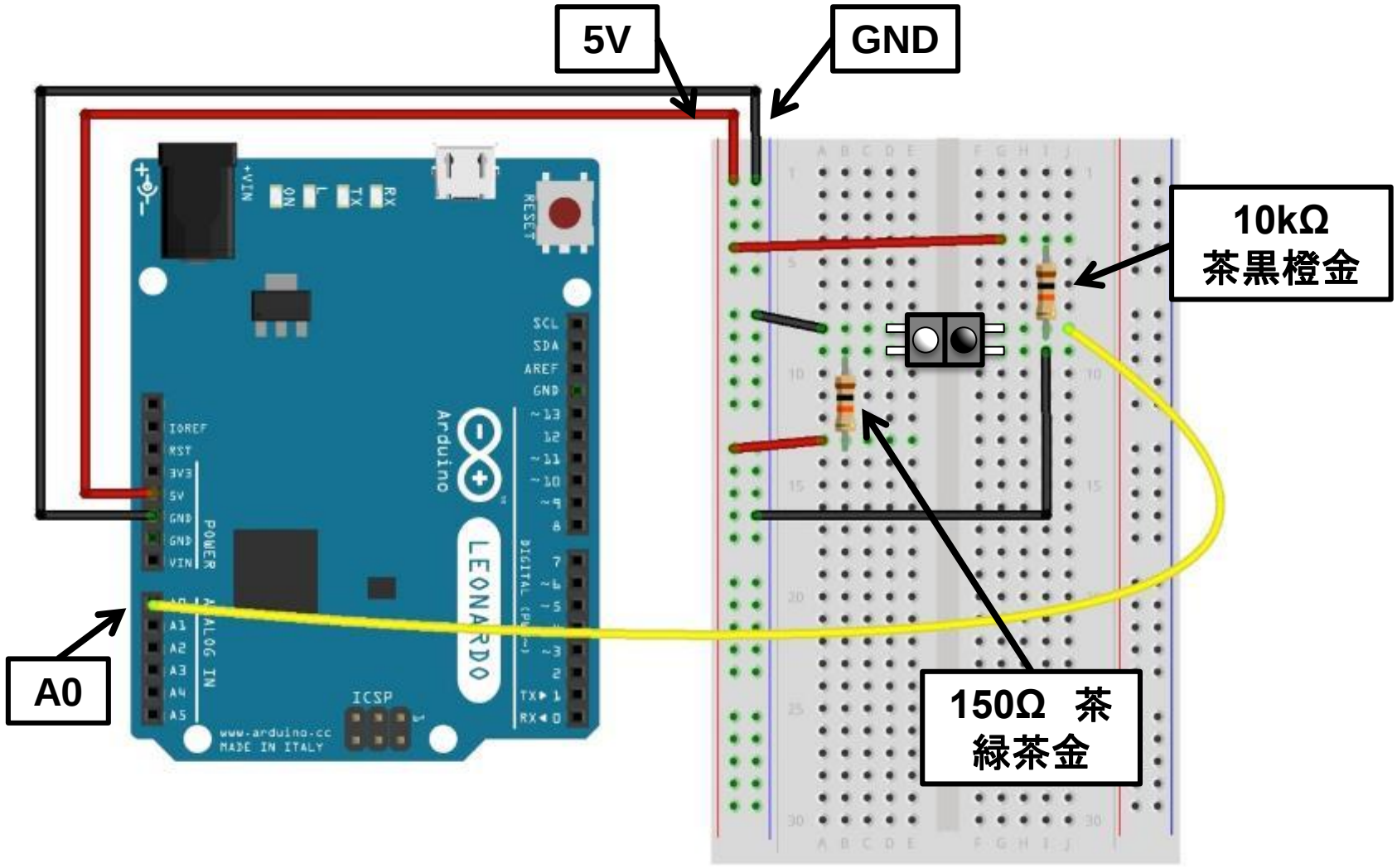
- 赤外線LEDと赤外線受光素子を合体したセンサ



- 赤外線の反射光の「強度」を計測
  - 紙の白黒を判別したり、かざした手との距離を測ったり、目の横に付けてまばたきを検出したり、クッションの中にいれて圧力を計測したり、指先の脈波を計測したり、etc…

- 参考ページ　http://kougaku-navi.net/sensact/

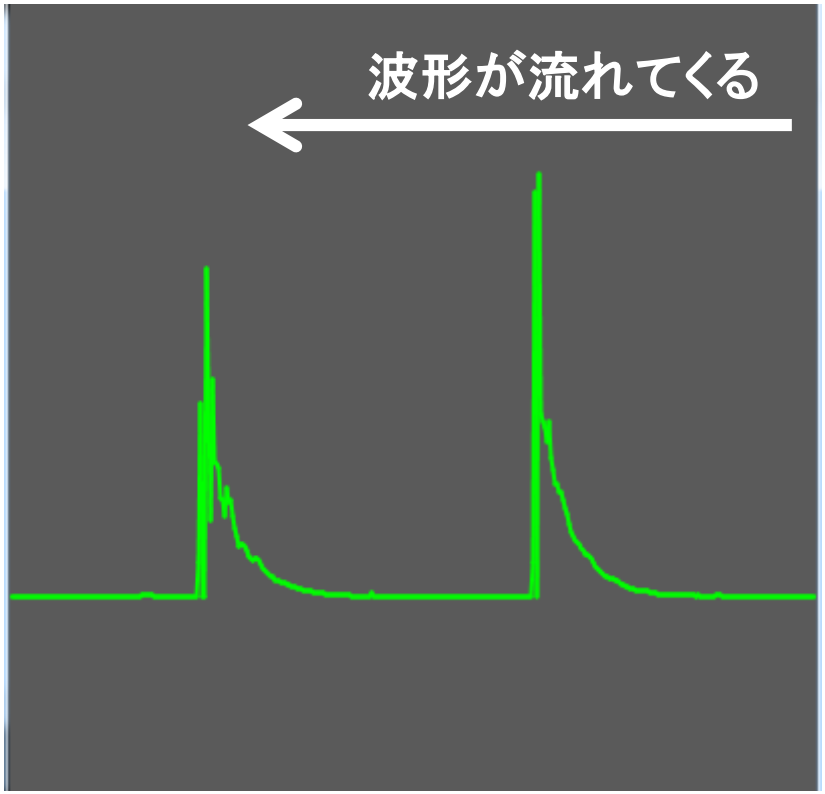
フォトリフレクタを使う回路



光センサや曲げセンサと同じように、アナログ入力を使って電圧の変化を読み取ろう

練習問題1

- センサ入力の時間的な変化を表す波形をリアルタイムに表示するプログラムを作ってみよう。

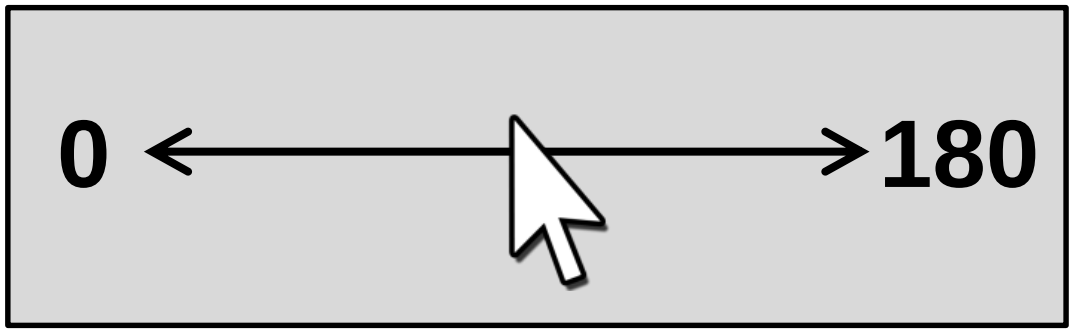


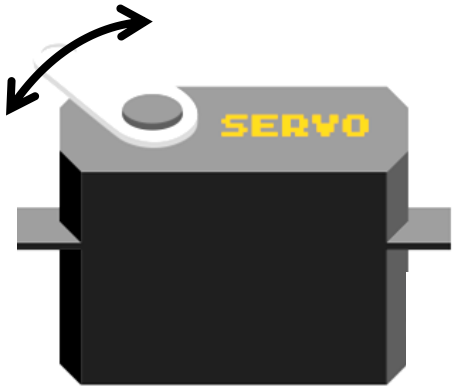
光センサ、ボリューム、曲げセンサ、圧電スピーカなど、好きなセンサで試してみよう。

練習問題2

- 画面上でマウスカーソルを動かすと、それに合わせてサーボモータの角度が変化するプログラムを作ってみよう。

例）左右にマウスカーソルを動かすと、0～180度の範囲で角度が変化する





応用問題

- Arduinoに曲を弾かせてみよう

  例）マリオ

  http://www.linuxcircle.com/2013/03/31/playing-mario-bros-tune-with-arduino-and-piezo-buzzer/

  例）クリスマスソング

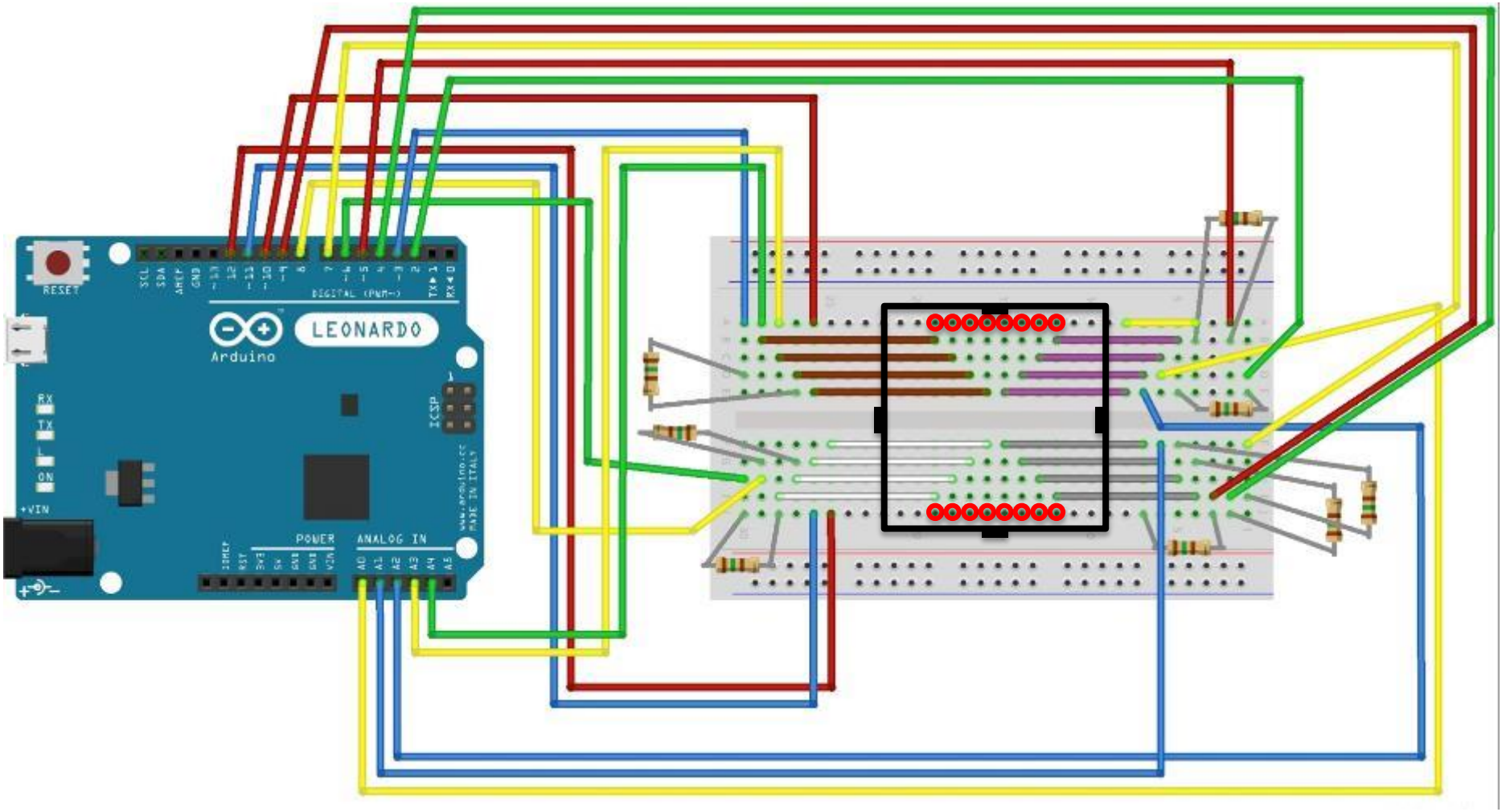
  http://meatfinish.wordpress.com/2010/12/12/arduino-christmas-tunes-player/

- センサとスピーカを組み合わせてオリジナルの電子楽器を作ってみよう
- フルカラーLEDや、マトリクスLEDと音を連動させてみよう

マトリクスLED

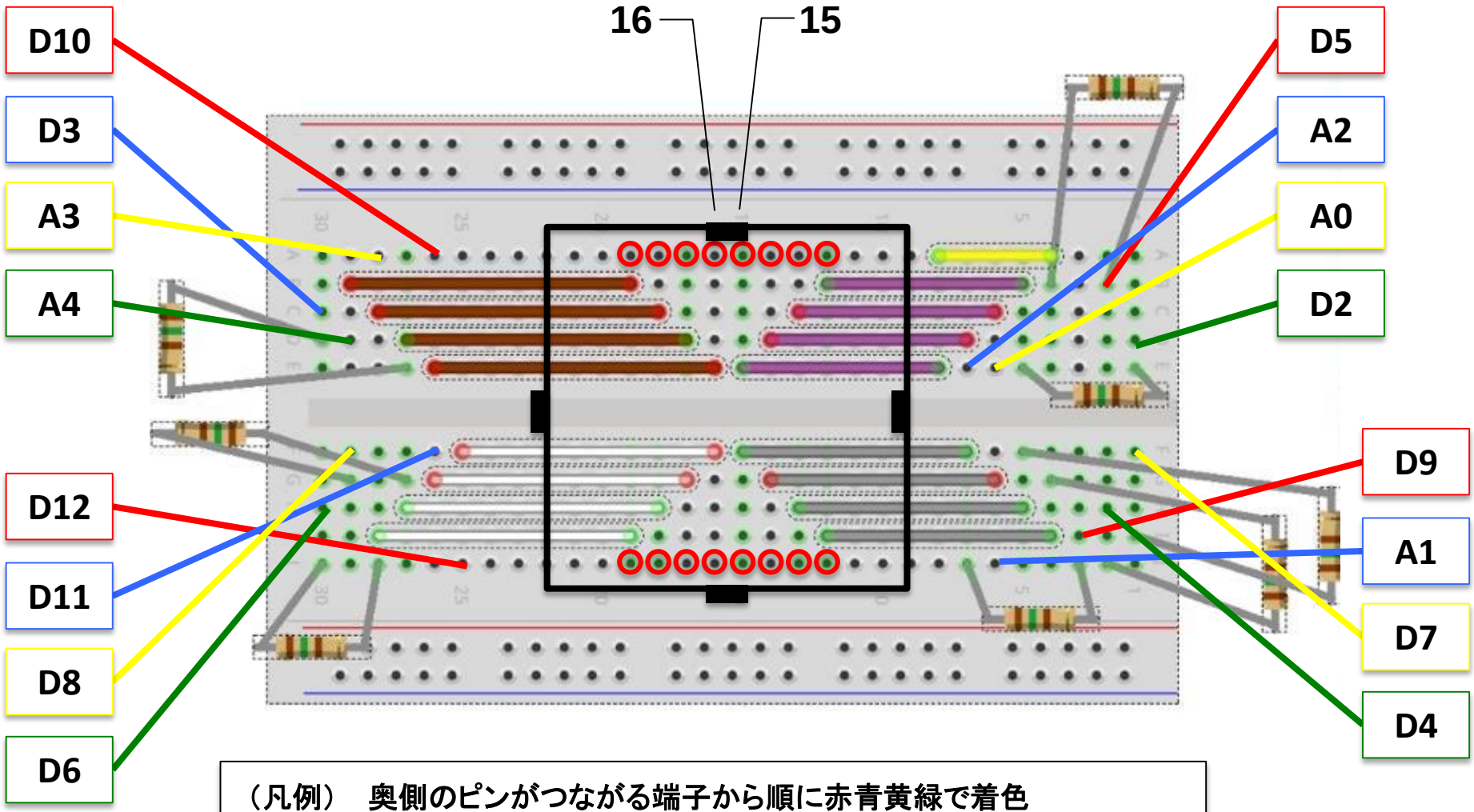
- LEDを8×8個=64個まとめた部品
- 文字やマークを表示できる
- 行と列の電圧を制御して点灯；個別にON/OFFを制御できない
  - 列の電圧が高くて、行の電圧が低い交点にあるLEDが点灯する

マトリクスLEDを使う回路（実体配線図）



※マトリクスLEDに表示するプログラムは別ファイルに掲載

マトリクスLEDを使う回路(接続先の整理)



（凡例）　奥側のピンがつながる端子から順に赤青黄緑で着色

D5　A2　A0　D2

Dはデジタル入出力ピン
Aはアナログ入力ピン

マトリクスLEDを使うための準備

(1) Flextimer2をダウンロード

- http://www.pjrc.com/teensy/td_libs_MsTimer2.html から FlexiTimer2.zip をダウンロードしてzipファイルを解凍
- 解凍してできたフォルダをそのまま、Arduinoのフォルダ（自分の作ったプログラムが保存される場所）の中の、librariesにコピー
- Arduino IDEを再起動

(2) プログラムの中の以下の点を修正

- MSTimer2 → FlexiTimer2 に修正（3ヶ所）
- `int ANODEPIN[] = {` の次の行を `10,11,12,18,19,20,21,22};` に修正 ※Uno とLeonardoのピン番号の指定方法の差異による修正です

この資料と一緒に掲載されているプログラムは修正済です。

マトリクスLEDに表示するArduinoプログラム

```
//
// LEDマトリクスにパターンを表示するプログラム
//

// タイマー割込みを使用する
#include <FlexiTImer2.h>
//  LEDマトリクスのピン割当て:ROW側がカソード、COLOMN側がアノードの8x8マトリックスLED
int ANODEPIN[]   = {
  10, 11, 12, 18, 19, 20, 21, 22 };  // Leonardo の場合
//  10, 11, 12, 14, 15, 16, 17, 18 };  // Uno の場合
int CATHODEPIN[] ={
  2,  3,  4,  5,  6,  7,  8,  9    };

int testPatternF[8][8] = {
  0,1,1,1, 1,1,1,0,
  0,1,1,1, 1,1,1,0,
  0,1,1,0, 0,0,0,0,
  0,1,1,0, 0,0,0,0,
  0,1,1,1, 1,1,0,0,
  0,1,1,1, 1,1,0,0,
  0,1,1,0, 0,0,0,0,
  0,1,1,0, 0,0,0,0};

int testPatternM[8][8] = {
  1,1,0,0, 0,0,1,1,
  1,1,1,0, 0,1,1,1,
  1,1,1,1, 1,1,1,1,
  1,1,1,1, 1,1,1,1,
  1,1,0,1, 1,0,1,1,
  1,1,0,0, 0,0,1,1,
  1,1,0,0, 0,0,1,1,
  1,1,0,0, 0,0,1,1,};

int testPatternS[8][8] = {
  0,1,1,1, 1,1,1,0,
  1,1,1,1, 1,1,1,1,
  0,1,1,0, 0,0,1,1,
  0,0,1,1, 1,0,0,0,
  0,0,0,1, 1,1,0,0,
  1,1,0,0, 0,1,1,0,
  1,1,1,1, 1,1,1,1,
  0,1,1,1, 1,1,1,0};

int matrix[8][8];   // LEDマトリクスの各点の状態を保持する配列(1:ON、0:OFF)
int displayRow;    // 現在表示中のrow
int pat = 0;       // メインプログラムで指定するパターン番号

void setup() {
  initMatrix();
}
void loop() {
  if(pat == 0) {
    setPattern(testPatternF);
  }
  else if(pat == 1) {
    setPattern(testPatternM);
  }
  else {
    setPattern(testPatternS);
  }
  pat++;
  pat = pat%3;
  delay(1000);
}



///----------------------------------------------------------
// LEDマトリクスを制御する関数
//----------------------------------------------------------

//
// LEDマトリクスを初期化する
// (1) Arduino の出力ピンの初期化
// (2) 各LEDの状態を保持する配列の初期化
// (3) LEDを表示するタイマ割込みの設定
//
void initMatrix() {
  //  出力を初期化、アノードHIGH、カソードLOWで点灯、その逆に設定する
  // アノードを、LOWで初期化(coulmn方向)
  for( int ano = 0; ano < 8; ano++ )
  {
    pinMode( ANODEPIN[ano], OUTPUT );
    digitalWrite( ANODEPIN[ano], LOW );
  }
  // カソードを、HIGHで初期化(rowに対応)
  for( int cat = 0; cat < 8; cat++ )
  {
    pinMode( CATHODEPIN[cat], OUTPUT );
    digitalWrite( CATHODEPIN[cat], HIGH );
  }

  clearMatrix();  // 状態を保持する配列をOFFで初期化
  startMatrix();  // LEDマトリクスの表示を開始(表示行を0に設定して、タイマ割込みを開始)
}

//
// LEDマトリクスの表示を更新(表示行 displayRow の1行分を表示)
//
void updateMatrix() {
  if(displayRow >= 0) {
    digitalWrite( CATHODEPIN[displayRow], HIGH );  // 1つ前のROWをHIGHに変更(消す)
  }
  displayRow++;
  displayRow = displayRow%8;
  //  コラム方向(アノード方向)の出力を順にセット
  for(int column = 0; column<8; column++) {
    if(matrix[displayRow][column] == 1) {
      digitalWrite( ANODEPIN[column], HIGH );// HIGHに変更
    }
    else {
      digitalWrite( ANODEPIN[column], LOW );// LOWに戻す
    }
  }
  // LOWに戻す(点灯;次のタイマー割込みまで、その状態で表示)
  digitalWrite( CATHODEPIN[displayRow], LOW );
}

//
// LEDマトリクスの表示を開始(表示行を0に設定して、タイマ割込みを開始)
//
void startMatrix(){
  displayRow = -1;
  FlexiTImer2::set(1, updateMatrix); // 500ms period
  FlexiTImer2::start();
}
/
// LEDマトリクスの状態を保持する配列を変更する
//    void clearMatrix(): 全て0に設定
//    void setPoint(int x, int y): (x,y)を1(ON)に設定
//    void resetPoint(int x, int y): (x,y)を0(OFF)に設定
//    void setPattern(int _pattern[8][8]): _patternに設定されたパターンを設定

void clearMatrix() {
  for(int row = 0; row<8; row++) {
    for(int column = 0; column<8; column++) {
      matrix[row][column] = 0;
    }
  }
}

void setPoint(int _row, int _column) {
  matrix[_row][_column] = 1;
}

void resetPoint(int _row, int _column) {
  matrix[_row][_column] = 0;
}

void setPattern(int _pattern[8][8]) {
  for(int row = 0; row<8; row++) {
    for(int column = 0; column<8; column++) {
      matrix[row][column] = _pattern[row][column];
    }
  }
}
```