Arduino ビギナーのための

センサー活用入門

株式会社　イーケイジャパン

はじめに

電子回路を使って何かを作ろうと思ったとき、マイコンと電子部品を使うととても便利になります。今までは多くの電子部品を並べてつないで、それぞれの動作を理解して回路を設計し、希望の機能を作る必要がありました。さらに機能を変更したいときには、大変な手間をかけて回路を修正しなければなりませんでした。しかし、マイコンの登場で高度な動きをわずか数点の電子部品で実現できるようになり、動作の修正も簡単になりました。だだし、マイコンを使いこなすには専門的な知識や言語の理解、専用の機器を用意する必要があり、ハードルが高かったのですが、最近では例えば Arduino のようなマイコンボードが登場し、比較的簡単な言語で、希望の動作を得られるようになりました。

マイコンで電子部品を制御できるようになると、思いついたことをどんどん試すことができます。しかし、「電子部品」を使うためには、どうしてもその特徴や使い方を知っておく必要があります。電子部品を知るための近道は、やはり、実際に触って動かしてみることです。さらに、電子部品の特徴やポイントを知っておくと、余計なことに悩まされることなく、自分の作りたいモノに集中できるようになります。

本製品は、実際の部品の動作を確認しながら、電子部品の特徴や解説を読むことで効率的に学習できるセットです。マイコンには Arduino ボードを利用します。説明を読みながら、実際に動作させ、結果を確認していくことで、プログラムで電子部品を動かすときのポイントがわかるようになっています。本製品があなたのアイデアを創造する手助けになればと思います。

重要

本機(SU-1203：センサーシールド)を動作させるためには、Arduino 基板が必要です。
Arduino 基板には多くの種類が発売されていますが、本書の説明には Arduino-UNO(R3)を使っています。
また、Arduino 基板の接続にはご使用の環境に合わせた USB ケーブルを準備してください。

●おことわり

本書では Arduino 基板の説明や、プログラム開発環境の入手～セットアップ方法、プログラム用命令の説明については詳しく記載していません。本格的に Arduino のプログラムを学びたい場合は、Arduino のサイト http://www.arduino.cc/や、インターネット、書籍などの説明や解説を参考にして学習してください。

もくじ

Arduino 基板と Arduino-IDE

準備

●用意するもの

（１）　Arduino-UNO　(R3)　センサーシールド(SU-1203)を動作させるために必要です。

（２）　センサーシールド　　SU-1203

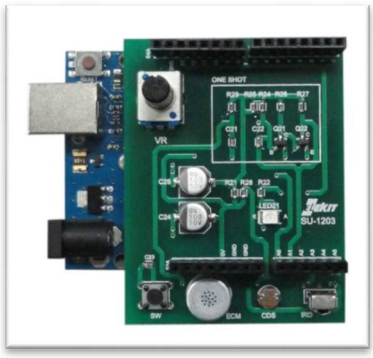
（３）　USB ケーブル　　Arduino 基板とパソコンを接続します。

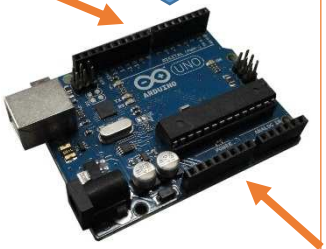
（４）　パソコン　　Arduino プログラム開発環境がインストールされているパソコン

（※プログラム開発環境の入手～インストールがまだの場合は、巻末をご参照の上インストールしてください。）

●Arduino とセンサーシールドの接続

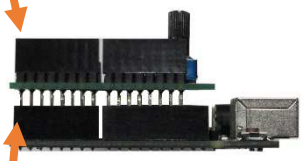
Arduino のコネクタに表示シールドのピンを差し込みます。
向きを確認し、ピンの位置を合わせて、ピンが曲がらないように注意して差し込んでください。





Arduino 基板の DIGITAL-0 番ピンとシールド基板の S2A-0 番ピンを合わせます。

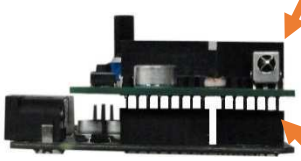
S2A-0 番ピン



DIGITAL-0 番ピン

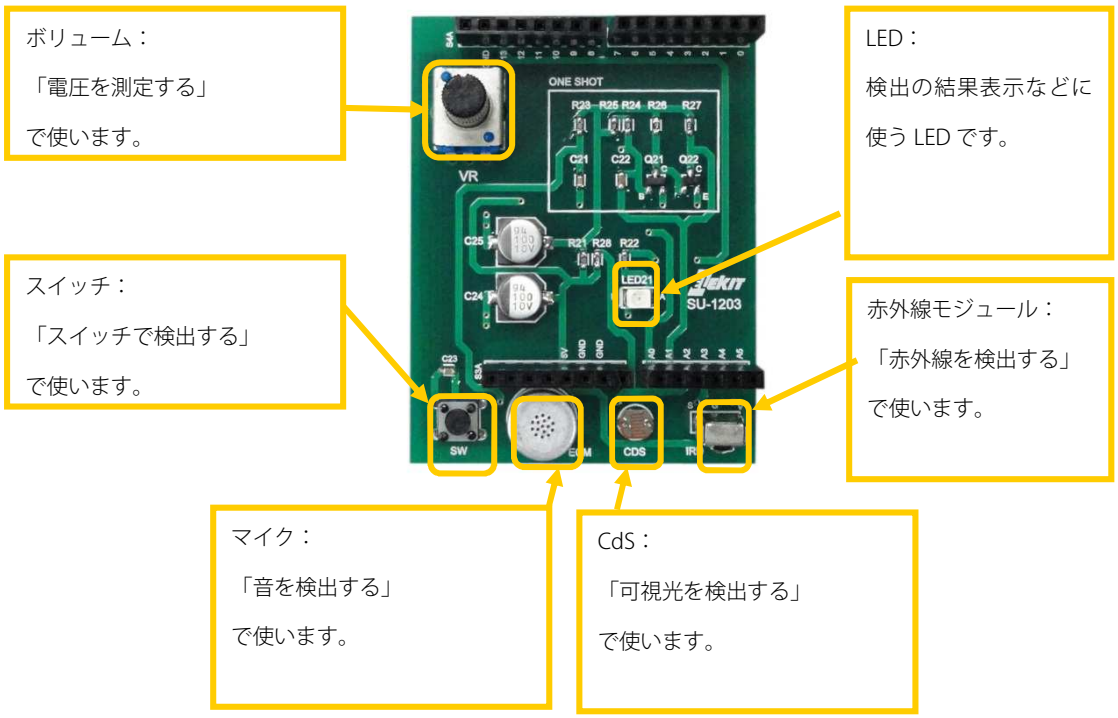
Arduino 基板の ANALOG-A5 番ピンとシールド基板の S1A-A5 番ピンを合わせます。

S1A-A5 番ピン



A5 番ピン

●センサーシールドについて

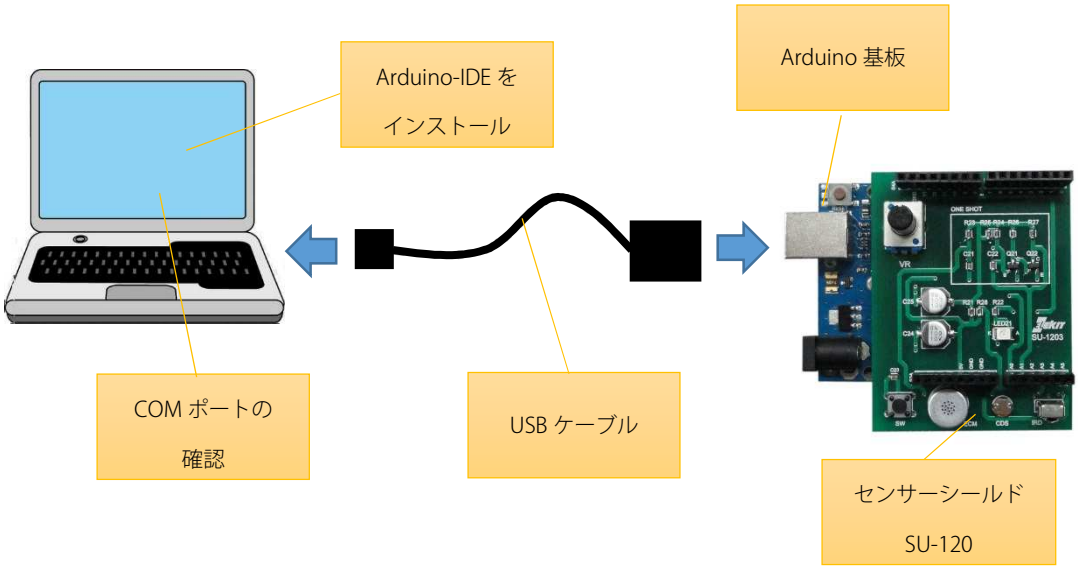


※電源には Arduino 基板の 5V を利用しています。

●パソコンとの接続

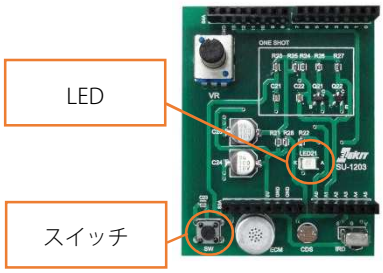
本書の「Arduino 基板と Arduino-IDE」を参考に、パソコンに Arduino の開発環境 Arduino-IDE をインストールし、USB ケーブルでパソコンと Arduino 基板を接続します。

COM ポートの設定・確認も忘れないようにします。



1．スイッチで検出する

最初の一歩は、身の回りによくある「スイッチ」を使ってみます。スイッチはあらゆる場面で使われる基本的な機械です。多種多様なものがあり、それぞれ使い方に特徴があります。今回使ってみるのは、「押しスイッチ」です。



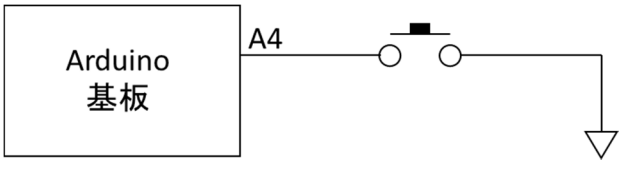
●プログラム

次のプログラムを作成して、マイコンにアップロードします。

スイッチを押すと LED が点灯するプログラム

```
int valSW;                          //変数の定義
void setup(){
   pinMode(13, OUTPUT);             //13 番ピンを出力にする
   pinMode(A4, INPUT_PULLUP);       //A4 番ピンをプルアップ機能を使った入力にする
}
void loop(){
   valSW = digitalRead(A4);         //A4 番ピンの状態を変数 valSW に記憶
   if(valSW == 0){                  //valSW の値で条件分岐する
      digitalWrite(13, HIGH);       //valSW が 0 のときは LED を点灯　------(処理 A)
   }else{
      digitalWrite(13,LOW);         //valSW が 0 以外のときは LED を消灯　---(処理 B)
   }
}
```

●回路



Arduino のアナログ 4 番ピンにスイッチの片方の端子がつながっています。

スイッチのもう片方の端子は GND につながっています。

●解説

まず、**setup** の中で出力ポートを宣言します。これは Arduino が電源を入れたとき全てのポートは入力の状態になっているからです。次に、**pinMode(A4, INPUT_PULLUP)**と書きます。これは、指定したピンをプルアップする命令です。プルアップについて後述する「プルアップ」で説明しています。

loop の中では、まずスイッチがつながっているピンの状態をチェックし、**if** 文を使って、そのピンの状態が H であれば、LED を点灯、L であれば LED を消灯します。
プログラムに **valSW = digitalRead**（**A4**）と書くと、マイコンは、**digitalRead** 命令を処理して A4 番のピンの状態が H であるか、L であるかをチェックしてその状態を送り返します。
チェックして送り返された値を、変数 **valSW** に記憶します。具体的には、H であれば、1 が、L であれば 0 が代入（＝記憶）されることになります。ここで、変数"**valSW**"を使います。**valSW** はあなたが新しく作る変数ですので、プログラムの先頭で、「新しい変数を作りました。」と宣言する必要があります。そのための命令が、**int=valSW** です。

次に、**if**（**valSW == 0**）の命令を処理します。この **if** 文の構造は、

```
if（条件）{
     条件に一致したときに実行する処理；          ----------(処理 A)
} else {
     条件に一致しなかったときに実行する処理；    ----------(処理 B)
}
```

というようになっています。
あえて日本語で表現すると、「もしも **valSW** が **0** ならば（A）を処理して、そうでなければ（B）を処理する。」という意味になります。つまり「**valSW** が **0** なら、すぐ後のカッコで囲まれた命令を実行し、**valSW** が **0** でないならば、**else** に続くカッコで囲まれた命令を実行する。」ということです。

if 文の処理が終わったら、他の命令はなく、プログラムの最後なので、**loop** の先頭の命令に戻り、プログラムを繰り返します。

●使い方、ポイント

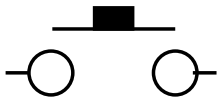
・タクトスイッチとは

スイッチは、人が触れることが多い部品で、大電流から小電流まで多種多様な場面で使われますので、数多くの種類があります。ここでは本機で使っているタクトスイッチについて説明します。
タクトスイッチは、押しスイッチとかプッシュスイッチに分類されるもので、その中でも指で軽く触って操作するタイプのスイッチをタクトスイッチと呼ぶことが多いようです。

タクトスイッチは ALPS 電気の商標で、英語では tactile switch(タクティール スイッチ)と言います。

タクトスイッチの例

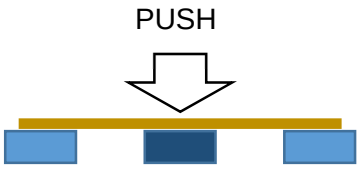
回路記号



・タクトスイッチの原理

タクトスイッチの中には、薄く湾曲した金属板が入っており、タクトスイッチのノブを押すと、金属板が変形して端子に接触します。

押されていないとき



押されているとき

タクトスイッチを分解したところ

・モーメンタリータイプ／オルタネートタイプ

プッシュスイッチの動作には大きく 2 つのタイプがあります。1 つは、押している間だけ ON するモーメンタリータイプ。もう 1 つは、スイッチを押すごとに ON・OFF が切り替わるオルタネートタイプです。タクトスイッチは一般的にモーメンタリータイプとなります。

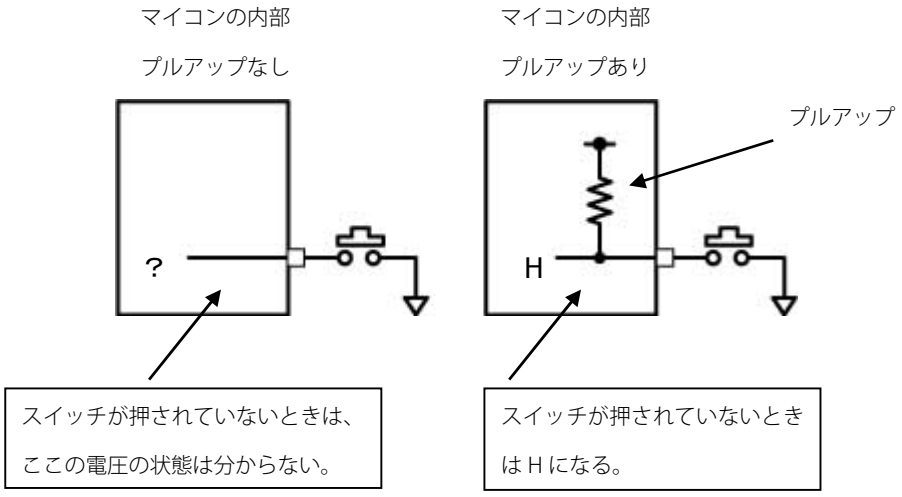
・最大定格

タクトスイッチに限らずスイッチには、最大定格といって、これ以上の電圧を加えてはいけないという最

大電圧と、これ以上の電流が流れてはいけないという最大電流が決められています。最大電流や最大電圧を超えてしまうと、接点が焼き付いてくっついたままになったり、接点が溶けたり変形したりして、電気が流れなくなる場合があります。他の部品の定格と同じように最大定格の 50%程度以内で使うようにします。

・プルアップ

`pinMode(A4,INPUT_PULLUP)`と書くと、A4 番ピンにプルアップ機能を使うことになります。
プルアップはタクトスイッチを使うときによく使われます。プルアップが必要な理由ですが、
まず、プルアップ機能を使用しないときのことを考えます。下の図を見ながら動作をイメージしてください。
SW は Arduino 基板の A4 番ピンがつながっていて、SW を押すと、A4 番ピンが LOW になります。それでは、SW が押されていないときは、A4 番ピンはどうなるのでしょうか？SW が押されていないときは、A4 番ピンは電気的にどこにもつながっていません。つまり、どんな状態になるのか分からないのです。もちろん Arduino（マイコン）にも分かりません。だから `digitalRead` 命令で A4 番ピンの状態を確認しても、H なのか L なのかはそのとき次第ということになってしまいます。
そこで、これを解決するための機能が「プルアップ」です。マイコンの内部で、A4 番ピンを数十 kΩの抵抗を経由して電源につなぐ機能です。こうすることで、普段は（SW が押されていないときは）A4 番ピンは H になり、SW が押されると L になります。

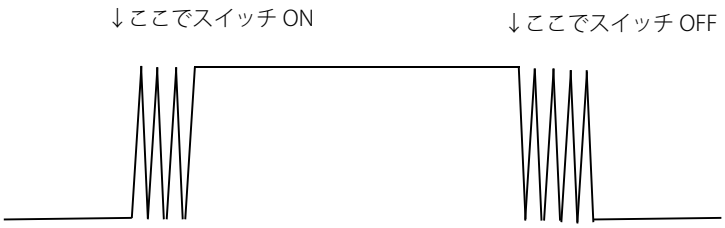


・チャタリング

チャタリングとは、金属接点が動いて接触状態になるときに、非常に速い機械的な振動を起こす現象のことです。日本語ではチャタリングと言われることが多いですが、英語では contact bounce と言われます。例えば、タクトスイッチは原理の図で説明しているように、接点に金属の板を押しつけることで電気の流れを ON/OFF します。スイッチが ON 時または OFF 時になるとき、つまり板が接触状態になるときに、衝

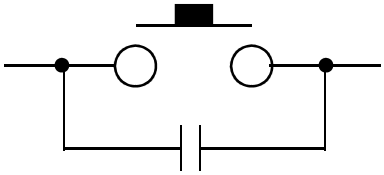
撃や振動が発生してしまい、その振動が電気の流れに影響します。人間の感覚ではとても短い時間で発生しているのですが、電気回路では誤動作の原因になるので、誤動作しないような対策が必要です。

チャタリングのイメージ



誤動作対策として簡単に行えるものとしては、
図 1 のようにコンデンサを並列に取り付ける、図 2 のようにソフト（プログラム）で判断する、などがあります。

図 1：コンデンサを付けた例



コンデンサの値は実際の使用環境で実験して決めるが、0.1uF 程度のセラミックコンデンサが使われることが多い。
小さすぎると効果がなく。
大きすぎると反応が遅くなったり、反応しなくなったりする。

図 2：ソフトで対策する例

```
valSW = digitalRead(A4);
if(valSW){
        delay(1);          //1m 秒待つ
        valSW = digitalRead(A4);
        if(valSW){
                処理 A
        }else{
                処理 B
        }
```

スイッチが押されたら、チャタリングが収まる程度の時間を待って、念のためにもう一度スイッチの状態を確認し、次の処理に進むようにする。

● コラム　1

いろいろなスイッチ

スイッチには様々な種類があり全ては紹介しきれませんが、代表的なタイプのスイッチをいくつか紹介します。

トグルスイッチ

金属のノブを操作して ON/OFF します。小電力～大電力用まで多種多様です。自作機器、産業機器を問わず幅広く使われています。

ロッカスイッチ

主にパネルなどに取り付けて使います。見た目も綺麗です。
信号切り替えよりも、電源の ON/OFF に使われます。

スライドスイッチ

ノブをスライドさせて ON/OFF します。小型・小電力のものが多く、大電力用のものはあまりありません。
自作機器、産業機器を問わず幅広く使われています。

マイクロスイッチ（リミットスイッチ）

軽いタッチで ON/OFF 動作するようになっています。
機器の移動終端の検出に使われます。

DIP スイッチ

小さなスイッチが多数並んでセットになっているスイッチです。
小型なので機器の内部でマイコンなどの初期値固定用に使われます。

ロータリースイッチ

つながる先を次々に切り替えることができるスイッチです。
信号を切り替えて動作や表示を変える回路などに使われます。

パソコン画面に表示する　Arduino-IDE 利用編

実験しているときは、スイッチやセンサーの状態をパソコンで確認できると、とても便利です。
Arduino は PC と通信するしくみを持っており、このしくみを使うことで、比較的簡単に PC 画面に文字を表示させることができます。もちろん、何を表示させるかは、自分で作成するプログラムで書いておく必要があります。ここでは、PC 画面への表示方法を説明します。

【Step1】

Arduino のプログラムにシリアル通信の設定をする。

（例）　スイッチの状態が H であるか L であるかを PC 画面に表示させるプログラム

```
int valSW;
void setup(){
        Serial.begin(9600);              <--------(1)
        pinMode(A4, INPUT_PULLUP);
}
void loop(){
        valSW = digitalRead(A4);
        Serial.println(valSW);           <--------(2)
        delay(100);                      <--------(3)
}
```

Arduino 基板とパソコンの間でデータをやりとりするためには、「シリアル通信」を使います。Arduino にはシリアル通信を簡単に行えるような命令が準備されています。

（1）　まず、Arduino 基板がシリアル通信を始める準備をするための命令 `Serial.begin(9600)`を書きます。この命令は、実際にデータを送受信する前に書く必要があることに注意してください。カッコの中の数字は通信のスピードを設定する数字です。特に必要がなければ `9600` から変更する必要はないでしょう。

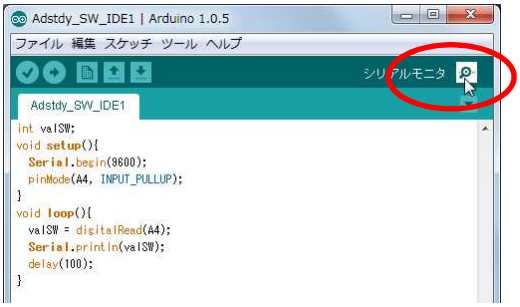
（2）　`Serial.println(valSW)`と書くと、`valSW` の値がパソコンに送られます。

（3）　パソコンはデータ（`valSW` の値）を受け取って表示するなどの処理時間が必要で、その間にデータが送られてきても大丈夫なようにデータを貯める場所があります。しかし貯めることができる量を超えてしまうと、通信が遅くなったり、停止したりしてしまいます。そこで `delay` 命令を入れて、あらかじめある程度の間隔でデータを送り、データが溢れないようにしています。
`delay` の時間は 1 ミリ秒程度で OK です。例えば `delay(1)`とすると、1 秒間に約 1000 回ピンの状態をチェックしてデータを送ります。`delay(10)`にすると、1 秒間に約 100 回ピンの状態をチェックしてデータを送ることになります。実際のセンサー動作を確認して決めるとよいでしょう。

【Step2】

Arduino 基板からパソコンに送られてきたデータを表示するには、Arduino-IDE の機能を使います。

Step1 で作成したプログラムを Arduino 基板にアップロードしたあと、Arduino-IDE の画面上のメニューから、ツール　→　シリアルモニタ　を選択します。または、画面上の右上の虫眼鏡の画像を押しても同じ動作になります。



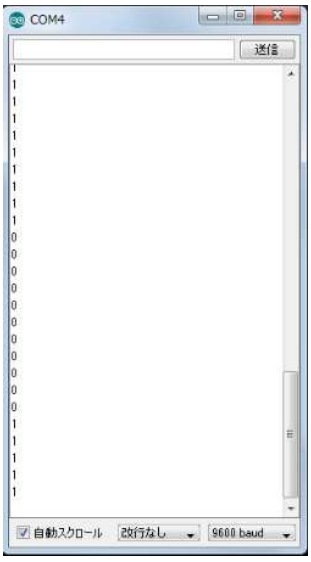
シリアルモニタを選ぶと、別のウィンドウが開きます。

その画面に、さきほどのプログラムでパソコンに送るように命令した、`valSW` の値が表示され始めます。

プログラムの中で `delay(100)`としましたので、1 秒間に約 10 回スイッチの状態が表示されます。

スイッチを押していないときは、画面に 1 が表示され、

スイッチを押している間は、画面に 0 が表示されます。



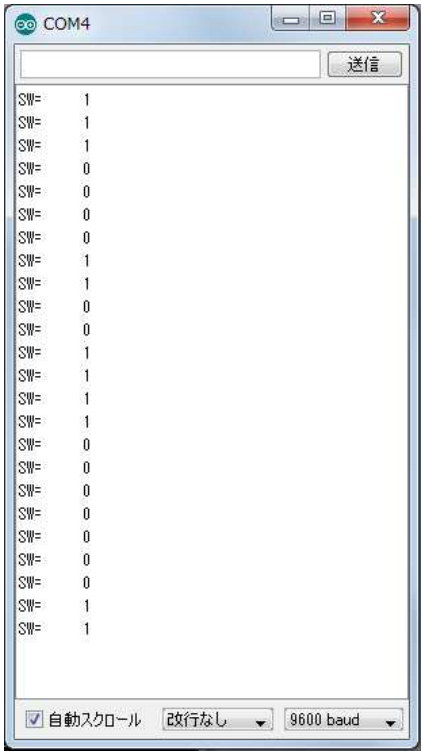
【Step3】

数字だけが表示されて見づらい場合には、文字や、タブなどを表示させて見やすくすることもできます。

```
int valSW;
void setup(){
        Serial.begin(9600);              //シリアルポートを使う
        pinMode(A4,INPUT_PULLUP);        //SW(A4 に接続)をプルアップする
}
void loop(){
        valSW = digitalRead(A4);         //SW の値をリードする
        Serial.print("SW=");             //シリアルポートに「SW=」を出力
        Serial.print("¥t");              //タブを出力
        Serial.println(valSW);           // valSW の値をシリアルポートに出力
        delay(100);                      //0.1 秒待って次に進む
}
```

上のプログラムの中にある
Serial.println と Serial.print の違いは、
データを表示後に改行するか、しないかの違いです。

カッコの中のダブルクオテーションで囲んだ文字は、
そのままの文字が画面に表示されます。
またダブルクオテーションの中に　￥ｔ　と書くと
タブとなりますので、表示の位置を揃えたいときに
使うと便利です。



2．電圧を測定する

電子回路は、スイッチのようにオン、オフだけでなく、アナログ量も扱わなければなりません。例えば、「明るさ」には暗い、薄暗い、明るい、などの状態があり、それぞれの状態を検出できると便利です。このようなときには、アナログ電圧を出力するセンサー、つまり暗いときは０V、薄暗いときは２V、明るいときは４V というように変化する出力電圧をマイコンが判定できればよいことになります。
マイコンで電圧を測定する方法について説明します。

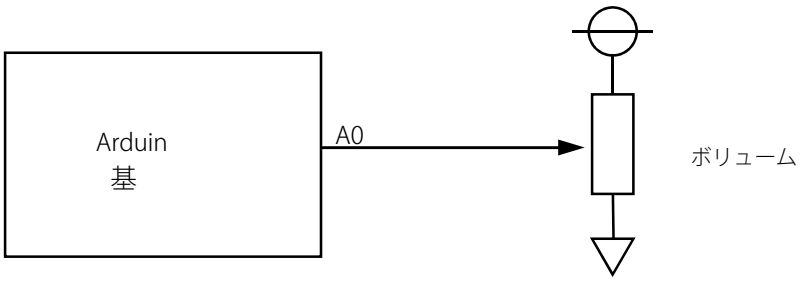
●プログラム

ボリュームを回して、ある程度の角度（約半分）以上になると、LED が点灯します。

```
int valVR;
void setup(){
        pinMode(13,OUTPUT);
}
void loop(){
        valVR = analogRead(A0);
        if(valVR > 512){
                digitalWrite(13,HIGH);
        }else{
                digitalWrite(13,LOW);
}
```

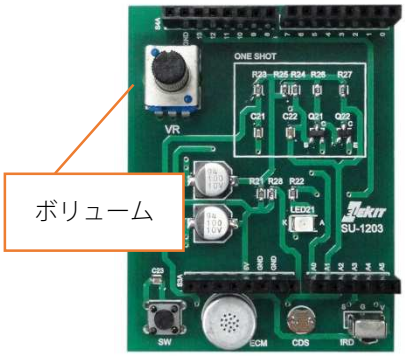
●回路

ボリュームとマイコンは以下のように接続されています。



●解説

本機のボリュームは、抵抗の両端は電源と GND につながれており、抵抗の途中につながれている端子が、マイコンにつながっています。



プログラムに `valVR　＝　analogRead（A0）` と書くと、マイコンの A/D 変換が働き `A0` のピンの電圧を測定し、その結果が `valVR` に代入（記憶）されます。

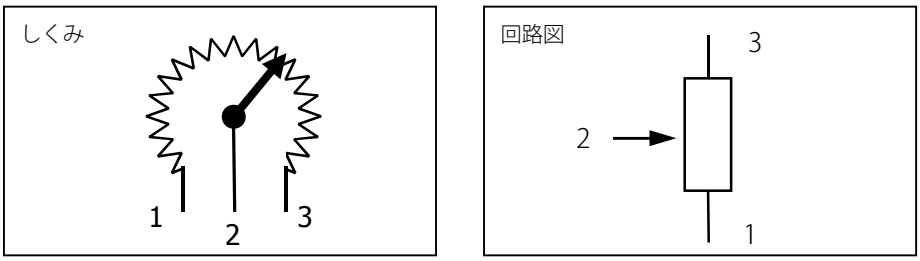
`if` 文の中の条件式で、A/D 変換結果と設定した数値を比較して、それぞれの条件にあった動作をさせています。

ここで大きなポイントとなるのは「A/D 変換」です。センサーの出力などをマイコンで扱う場合には覚えておきたい機能です。A/D 変換については次の「A/D 変換とは」で説明します。

●使い方、ポイント

・ボリュームとは

ボリュームは、つまみを回すことで抵抗値を変えることができる部品です。ボリュームには一般的に下図のように、３つの端子があります。



しくみの図にある 1 番と 3 番の端子は抵抗でつながっています。2 番の端子がツマミにつながっており、ツマミを回すことで抵抗に接触する場所を変化させて、抵抗の値を変化させています。2 番端子につながって動く部分のことを摺動子といいます。また、1 番と 3 番の間の抵抗値のことを全抵抗値といいます。用途に応じて、いろいろな抵抗値が用意されています。

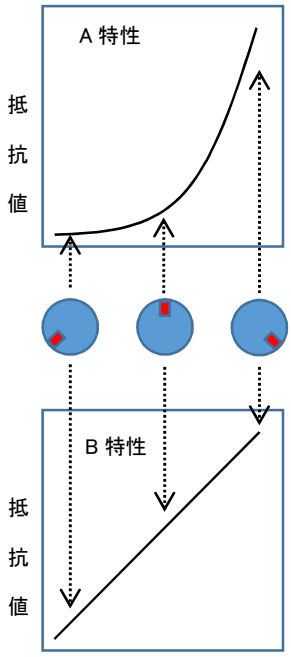
・最大定格

ボリュームには最大定格が決められています。最大定格を超えてしまうと、接点が焼き付いてくっついたままになったり、発熱して煙が出たり、火を噴いたりします。他の部品の定格と同じように、最大定格の 50%程度を目安に使うようにします。ボリュームの定格はほとんどの場合「W（ワット）」で決められています。

・抵抗カーブ

ボリュームツマミを回したときに変化する抵抗の割合のことを抵抗カーブといいます。使用目的に応じてA カーブや B カーブ特性のものがあります。

B カーブはいつも同じ割合で抵抗が変化していくものです。
A カーブは、2 次曲線的に抵抗値が変化していくものです。A カーブ特性のボリュームはオーディオなどの音量コントロールに使われます。B カーブでも使えないことはありませんが、ボリュームを少し回しただけで急に音が大きくなったように聞こえてしまいます。
余談ですが、最近のオーディオ IC などでは電子ボリューム機能が内蔵されていて、外部に B カーブ特性のボリュームを接続して、内部で自動的に A カーブ特性に変化させるものもあります。オーディオ機器の自作やボリューム交換では、目的にあった特性のものを選ぶようにしましょう。



本機には B カーブ特性のボリュームを使っています。

● コラム　2

ボリュームいろいろ

半固定抵抗

頻繁に回す必要がない場所に使われます。例えばラジオの電波の送受信回路を一台一台微調整するときに使われたりします。

可変抵抗

音量調整のように頻繁に調整する場所に使われます。

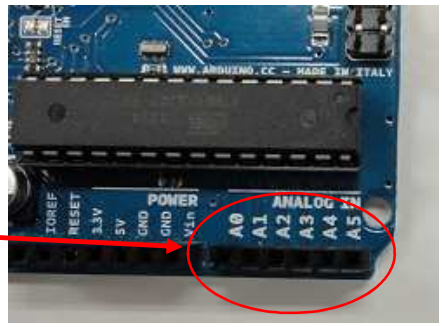
2 連ボリューム

可変抵抗が 2 つ一緒になったような構造なのでこう呼ばれます。左右信号を持つオーディオ機器のボリューム調整用によく使われます。2 連だけでなく 3 連や 4 連といったものもあります。これに対して抵抗が 1 つだけのものを単連ボリュームと呼ぶことがあります。

●A/D 変換とは

アナログの値をマイコンなどで扱えるようなデジタルの値に変換する機能のことを
Analog　Digital　Convertor といい、A/D 変換、または、ADC と言われます。
Arduino の場合、A/D 変換機能を持ったピンは
ANALOG　0～5 の 6 本となっています。

Arduino のアナログピン



・分解能

A/D 変換の性能を表すものとして「分解能」があります。測定できる最小の電圧が分かります。
例えば、Arduino の場合、分解能は 10 ビットです。
分解能とは、基準となる電圧、別の言い方をすれば測定できる最大の電圧（特別に指定しない限りマイコンの電源電圧と同じであることが多い）を何分割して比較しているのかという性能を示しています。

10 ビットは 1024 分割することができるという意味になり、8 ビットだと 256 分割することができるという意味になります。ビット数と分割できる数は以下の計算で求めることができます。

10 ビット	2^{10}	=1024	（2 の 10 乗）
8 ビット	2^{8}	=256	（2 の 8 乗）

分かりやすくするためにちょっと極端な例を示します。
以下の図は分解能が 2 ビットの場合です。2 ビットの場合は 4 つの状態を表すことができるので、5V を 4 つに分割すると、1 つ分は 1.25V になり、これが測定できる最小の電圧になります。

ピンに加わっている電圧	測定結果	ビットで表示	
		2 ビット目	1 ビット目
0　～1.25 未満	0	0	0
1.25 ～2.5 未満	1.25	0	1
2.5　～3.75 未満	2.5	1	0
3.75 ～5	3.75	1	1

Arduino の場合は、10 ビットの分解能つまり 1024 分割できるということですので、マイコンの電源が 5V の場合は、測定できる最小の電圧は、5V÷1024＝約 5mV になります。

・閾値

今回のプログラムの動作では、ボリュームを回して変化した電圧が「設定した値」を越えたときに、LED を点灯させています。このように、動作が切り替わる境になる「ある決められた値」のことを「閾値」といいます。

閾値は正しくは「いきち」と読みますが、「しきいち」と読むことが多いようです。

● コラム　3

2 進数　ビット数

マイコン内部の電子回路は、電圧がある(=1)、電圧がない(=0)状態を使って動作しています。1 と 0 の 2 つの状態でデータの処理を行うマイコンと、やはり 1 と 0 の 2 つの数字を使って表す 2 進数はとても都合がよいので、マイコンの説明には 2 進数が多く使われます。

2 進数の桁のことを「ビット」といい、桁が 2 桁あれば 2 ビット、8 桁あれば 8 ビット、64 桁あれば 64 ビットです。例えば 2 桁つまり 2 ビットあれば、00,01,10,11 の 4 つの状態があり、4 ビットあれば、0000～1111 までの 16 コの状態があることになります。

10 進数	4 ビットで表示
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	←4 ビットでは表示できない。

4 ビットで 0～15 まで表示することができますが、「16 以上」は表示できません。
ビットの数によって、一度に取り扱える数字の範囲は決まってしまいます。

3．可視光を検出する

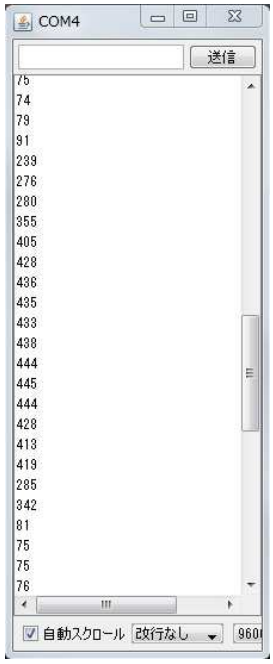
夕方薄暗くなったらライトを点けたいという場合や、ロボット競技などで、暗い場所と明るい場所を検出したいというようなときは光センサーを使います。「光」には、目に見える光＝可視光や、目には見えない光＝赤外線などがあります。今回は目に見える光を検出するのによく使われる、CdS を使ってみます。

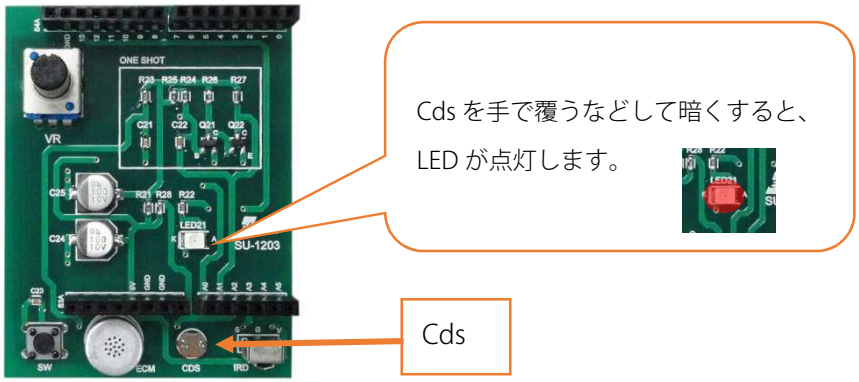
● プログラム

光センサーが光を検出したとき、つまり周囲が明るいときは LED が消灯。光センサーが光を検出していない、つまり周囲が暗いときは LED が点灯します。同時に、パソコンの画面に CdS の値を表示します。

```
int valCDS;
void setup(){
        Serial.begin(9600);          //シリアル通信の準備
        pinMode(13, OUTPUT);         //LED 用の 13 番ピンを出力
}
void loop(){
        valCDS = analogRead(A2);    //Cds の電圧をリードする
        if(valCDS >= 200){           //Cds の電圧によって分岐する
                digitalWrite(13, HIGH);
        }else{
                digitalWrite(13, LOW);
        }
        Serial.println(valCDS);     //Cds の電圧をシリアルに出力
        delay(100);
}
```

シリアル出力画面





●　解説

シリアル通信を利用して、パソコンの画面に光センサーの値を表示するので、`Serial.begin` 命令を使います。13 番ピンにつながっている LED を ON/OFF させるので、`pinMode` で `OUTPUT` に設定します。

`analogRead` 命令で、`A2` 番ピンを A/D 変換して、その結果を `valCDS` という変数に代入（記憶）させます。A/D 変換結果は、明るさによって 0～1023 の範囲の値になります。

`if` 文を使って、`valCDS` の値によって LED の ON/OFF を制御します。

`Serial.println` 命令で、パソコンに `valCDS` の値を送ります。

●　使い方、ポイント

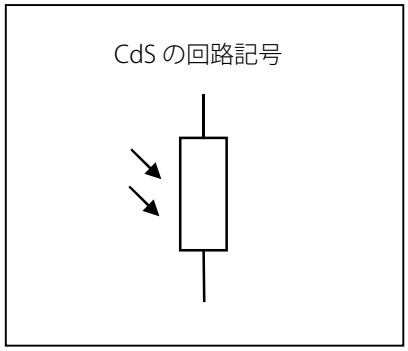
・光センサーとは

光センサーには検出したい光によって、様々な種類のセンサーが用意されています。本機では CdS という光センサーを使っています。CdS は可視光～赤外線の幅広い光の検出に向いたセンサーです。

CdS は光の当たる量によって自動的に抵抗の値が変わるボリュームと考えることができます。光の当たる量が多いと抵抗値が小さくなり、光の当たる量が少ないと抵抗値が大きくなります。

反応速度が遅いので最近の電子機器ではあまり使われませんが、しくみや動作が単純で使いやすいので、スピードが必要ない用途、例えば暗くなると点灯する街灯などにはよく使われています。

光に反応する物質に、硫化カドミウム（化学式：CdS）を使っています。カドミウムは欧州連合の RoHS 指令の対象となっていることもあり、最近では入手が難しくなっています。



・暗抵抗／明抵抗

CdS は光の量によって抵抗が変わる部品です。その抵抗がどのくらいであるかはデータシートに書いてありますので設計するときには確認する必要があります。

本機の CdS は、周りが暗いときの抵抗値＝暗抵抗が 1MΩ程度、周りが明るいときの抵抗値＝明抵抗が 10kΩ程度となっています。

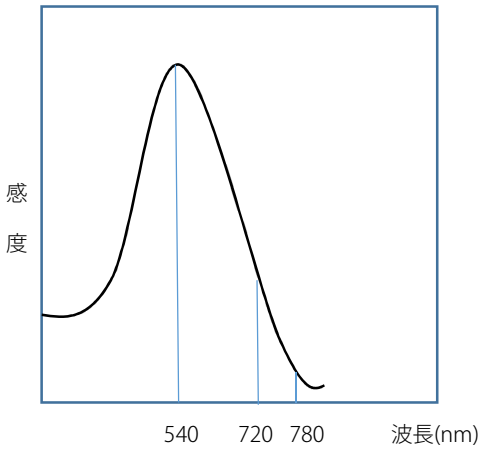
・ピーク波長

CdS などの光センサーには「ピーク波長」があります。これは、どのような光に対して一番感度良く動くかという目安になります。

本機で使っている CdS のピーク波長は 540nm です。人の目は 555nm の波長の光を一番明るく感じると言われていますので、人間の感じる明るさに近い動き方をすることが分かります。つまり人間が暗いと感じるときは CdS の抵抗値は大きくなり、人間が明るいと感じると抵抗値が小さくなります

ピーク波長や、波長に対する感度は、「スペクトラム特性図」を見ることでその目安が分かります。

スペクトラム特性図



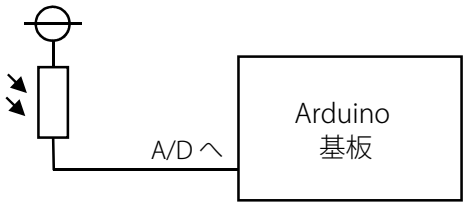
特性図を見ると、540nm の光にピークがあります。また、感度が半分程度になりますが、赤色の光（およそ 720nm）にも反応することがわかります。780nm 以上といわれている赤外線だと反応するけどちょっと厳しいかもしれないということがわかります。

・バイアス回路

CdS の出力を A/D 変換に入力するときに良く使われるのがバイアスを掛ける方法です。本機もこの回路を使っています。

まず、バイアスを加えない回路で、よくやってしまう間違いを説明します。

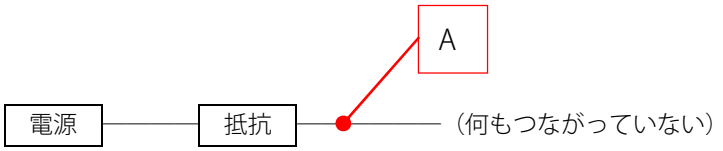
間違った回路の例



光の量によって抵抗値が変わるので、マイコンの A/D 変換器でいろいろな電圧が測定できそうな気がしますが、これは上手くいきません。

マイコンの A/D 変換器の入力回路は、ほとんど電流が流れない回路になっています。それは A/D 変換器の入力回路に電流が流れてしまうと、測定したい電圧が変わってしまうためです。電流はほとんど流れませんので、電気的に無視できる、つまり、つながっていないものと考えることができます。

そこで、先ほどの間違った回路を置きかえると、

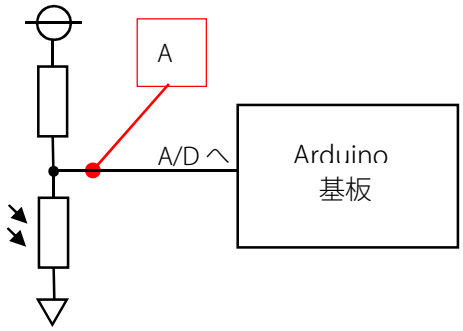


となっているだけですので、A の電圧を測定しても、電源電圧と同じ電圧になってしまいます。

（どこにもつながっていない＝電流が流れない　→　抵抗に電圧が発生しない＝電圧降下しない　）

抵抗の値がどんな値になっても結果は同じで電源と同じ電圧になります。

CdS の出力を A/D 変換器で変換するには、下図のような回路にします。

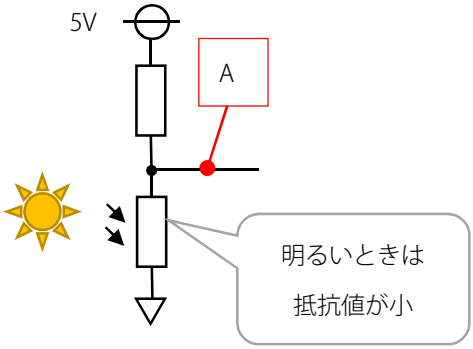


CdS の片側を GND に接続、反対側は抵抗を介して電源につなぎます。この抵抗のことをバイアス抵抗と言います。A/D 変換器には、図の A 点の電圧を入力します。

明るいときは、CdS の抵抗値が小さくなるので、A 点の電圧が下がります。

暗いときは CdS の抵抗値が大きくなるので A 点の電圧は上がります。

例）明るいときの A 点の電圧の計算方法



バイアス抵抗を 100kΩ、光を当てたときに
Cds の抵抗値が 10kΩ になったとすると、
A 点の電圧は・・・
(5V÷(100kΩ＋10kΩ))×10kΩ＝0.45V
になります。

バイアス抵抗の値は、前述した CdS の明抵抗よりも十分大きくします。バイアス抵抗の値が小さすぎると、A 点が直接電源につながっている状態と変わらなくなり、A 点の電圧が下がらず、正しい明るさの判定ができなくなります。

逆に、バイアス抵抗の値が大きすぎても問題が発生します。抵抗値が大きすぎると、A 点がどこにもつながっていない状態と同じになり、暗いときであっても A 点の電圧が上がらず、やはり正しい明るさの判定ができなくなります。

バイアス抵抗の値は実際に使う部品や接続される回路によって調整しなければなりません。

目安としては、CdS の明抵抗の、10 倍～1000 倍の感じになります。しかしマイコンの A/D 変換器に入力する場合は大きくても 100kΩ程度にしないと正常な A/D 変換ができませんので、あまり大きな値は使えません。この辺りの抵抗の値は、実際に実験しながら、反応スピードはどうか、A/D 変換は正しくされているかを確かめながら決めるのが一般的です。

４．赤外線を検出する

テレビや玩具などで使われているリモコンは赤外線を発射することで、チャンネルを変えたり、玩具本体を操作したりしています。赤外線の届く範囲でないと操作できないので、主に近距離の通信に使われます。赤外線は目には見えませんが光の一種なので、途中に障害物があれば光が届かず通信できません。また、赤外線は太陽光やランプの光の中にも含まれていますので、太陽の下や強い光の下では通信するのが難しくなります。しかし、その手軽さから赤外線通信は多くの場所で使われており、できるだけ通信が安定するようにいろいろな工夫がされています。今回はその赤外線通信で利用されている中の、リモコン用赤外線受光センサーを使ってみます。

● プログラム

なんでもよいので、お手持ちのリモコンを使います。

このプログラムでは、リモコンの信号を受け取ったら、LED が点灯します。

```
int valIR;
void setup(){
  pinMode(13, OUTPUT);
}
void loop(){
  valIR = digitalRead(A3);
  if(valIR == 0){
     digitalWrite(13, HIGH)
   }else{
  digitalWrite(13, LOW);
  }
}
```

● 解説

setup の中で、LED の ON/OFF を制御するための **13** 番ピンを **OUTPUT** にします。

本機の赤外線受光センサーは **A3** 番ピンに接続されています。リモコンの赤外線信号を受け取ると、L を出力し、それ以外のときは H を出力しています。

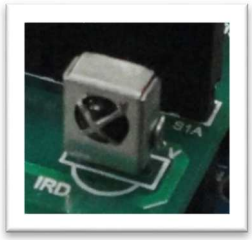
digitalRead 命令で **A3** 番ピンの状態をチェックして、**valIR** 変数にピンの状態を記憶させます。

次の **if** 文で、**valIR** 変数の値が、**0**（つまり Low）であったら、LED を ON し、そうでないときは LED を OFF します。

リモコンのボタンを押し続けると、リモコンの赤外線信号の特徴のため、LED が素早く点滅しているような感じで ON します。ボタンから手を離すと LED は OFF します。

●　使い方、ポイント

・赤外線受光センサーとは



　ここで使われている、赤外線受光センサーは、主にテレビやエアコンなどのリモコン信号の受信用に使われていることが多いセンサーです。赤外線のことを英語で infrared というので、IR センサーとか、IR レシーバーといわれています。

　（1）主に赤外線だけをキャッチしやすいようになっている。

　（2）赤外線がある一定の周波数で送られてきたときに、センサーの出力が変化する。

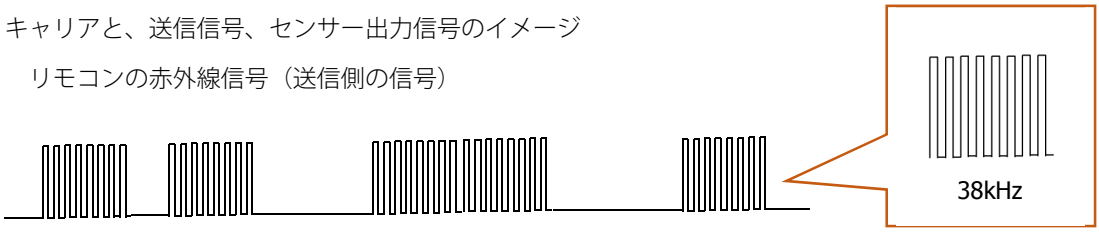
といった特徴があり、これは、太陽光にも含まれている赤外線によって誤動作しないよう、ある一定の周波数のものしか正しい信号と認識しないような回路が内部にあるためです。この周波数のことをキャリア周波数といいます。

・種類

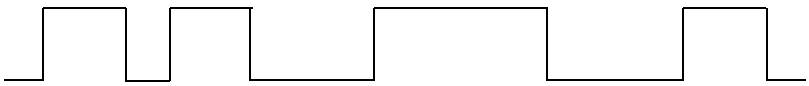
一般的に入手しやすい IR センサーは、そのキャリア周波数が 38kHz のものが多いようです。ちなみに 38kHz でないと絶対に受信できないというわけでもありません。もっとも効率よく受信できるのが 38kHz で、多少違っても、信頼性や安定性は下がるかもしれませんが、受信できるということを知っておくと、何かを自作するときに役に立ちます。

キャリアと、送信信号、センサー出力信号のイメージ

　リモコンの赤外線信号（送信側の信号）



　IR センサーの出力信号



・リモコン信号

リモコンは、いろいろなメーカーで作られて、多くの機器で使われていますが、それぞれメーカーが勝手に信号を作ってしまうとトラブルになります。例えば TV のチャンネルを変えようとしてリモコンを操作すると、近くにあったファンヒーターが ON したりすると大変です。そうならないようにリモコン信号には、そのデータの送り方に約束事が決められています。その約束事をリモコンフォーマットといっています。大きく、家電協会フォーマット、NEC フォーマット、SONY フォーマットというものがあります。このフォーマットを必ず守らなければいけないという訳ではありませんが、リモコンを自作するときには、不要なトラブルを避ける配慮が必要です。

●コラム　4

赤外線を見たい

赤外線は人間の目で見ることができない光です。でも、リモコンが不調なときなどは「このリモコンからちゃんと赤外線がでているのか？見ることできないかなあ？」と思うことがありませんか？赤外線は人間の目では見ることができませんが、実は多くのデジタルカメラは赤外線を見ることができます。

デジタルカメラを撮影モードにして、リモコンの赤外線を発する部分に向け、リモコンのボタンを押してデジタルカメラのディスプレイを見ると、赤外線が光っているのが見えます。

ただし、最近は、赤外線を遮断する機能やフィルターの付いたカメラが増えてきていますので、赤外線が写らないこともあります。赤外線が見えない場合は、1 台だけでなく、デジタルカメラや携帯電話のカメラ、ビデオカメラなどいろいろな機器で試してみましょう。

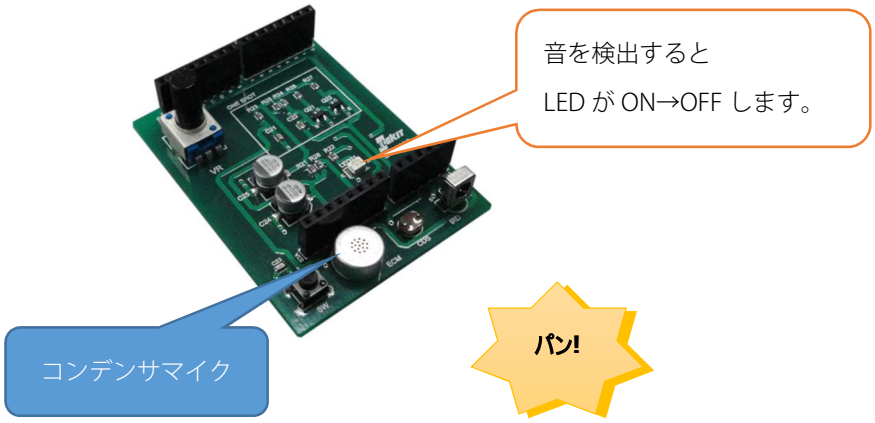
5．音を検出する

集音装置、つまりマイクを使って、ノックの音、大声、拍手などの音を検出する方法があります。「音」の検出には光やスイッチと違って、少し独特な知識が必要です。ここではその方法について説明します。

● プログラム

手ばたきなどの音を検出してランプを点灯、もう一度音を検出するとランプを消灯します。

```
int valMIC;
int valLED = 0;
void setup(){
  pinMode(13,OUTPUT);
  digitalWrite(13,valLED);
}
void loop(){
  valMIC = digitalRead(A1);
  if(valMIC == 0){
    valLED = ~valLED;
    digitalWrite(13,valLED);
    while(valMIC == 0){
      valMIC = digitalRead(A1);
    }
  }
}
```



● 解説

本機の音を検出する回路には「ワンショット回路」を使っています。ここで使ったワンショット回路は音が無いときは H が入力されており、音を検出するとしばらくの間 **A1** 番ピンに L が入力される回路です。

setup の中で、LED 用にを 13 番ピンを出力にして、LED の初期状態を書いています。

loop の中では、まず、マイクがつながっている **A1** 番ピンの状態をチェックします。音を検出していれば L(=0)が、音を検出していなければ H(=1)が **valMIC** に記憶されます。

次に if 文で条件分岐します。**valMIC** が **1** ならば、**if** 文の中は関係ありませんので、何もせずに **if** 文が終わり、**loop** の先頭に戻ります。**valMIC** が **0** ならば、**if** 文の中の命令を実行します。

if 文の中では、**valLED = ~valLED** とありますので、**valLED** が記憶している値が反転します。つまり今 **0** ならば、**1** になり、今 **1** ならば **0** になります。

続いて 13 番ピンにつながっている LED を制御します。13 番ピンから **valLED** を出力するので、**valLED** が **1** ならば **LED** が点灯、**0** ならば消灯します。

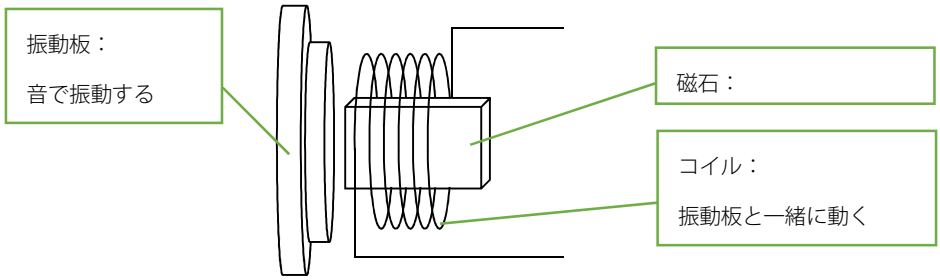
最後に、1 回のワンショット回路の動作の間に、何度も A1 番ピンの状態をチェックしてしまわないように、ワンショット回路の動作が 1 旦終わるまで待つために **while** 文を使っています。

● 使い方、ポイント

・ 音センサーとは

音センサーには大きく 2 種類があります。1 つは、ダイナミックマイク、もう 1 つはコンデンサーマイクです。ダイナミックマイクは、カラオケなどでよく見かけるマイクをイメージすれば良いと思います。スピーカのような形をした振動板と、それに取り付けられたコイル、コイルの近くに永久磁石を配置した構造であり、振動板が揺れることで、コイルが同時に揺れます。磁石の磁界の中でコイルが揺れると、揺れに合わせた電気信号が発生する仕組みを使っています。

ダイナミックマイクの構造



構造が簡単で、電源が要らないことから、多くの場所で使われています。

一方、コンデンサーマイクはとても小型・軽量にできることから、最近では薄型、軽量が求められるモバイル機器などで多く使われています。本機で使っているものもコンデンサーマイクです。

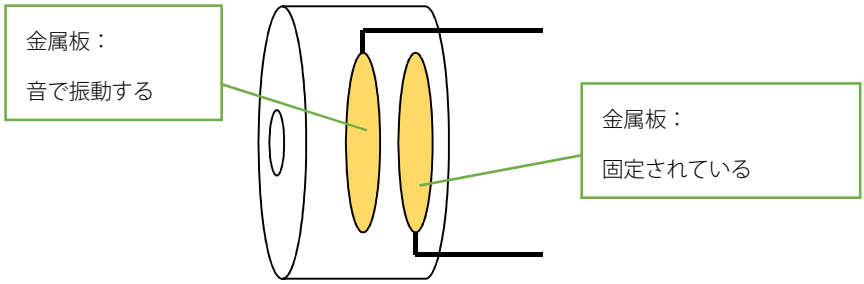
・　コンデンサーマイクのしくみ

エレクトレットコンデンサーマイク、略してコンデンサーマイクまたは ECM といわれることがあります。コデンサーマイクは、2 枚の金属板をほんのわずかに隙間を空けて向かい合わせにしたような構造です。

音を拾うと金属板が変形します、すると金属板の隙間の間隔が変化し、2 枚の金属板の間に溜まっていた静電容量の値が変化します。この容量の変化を電気信号として取り出す仕組みです。

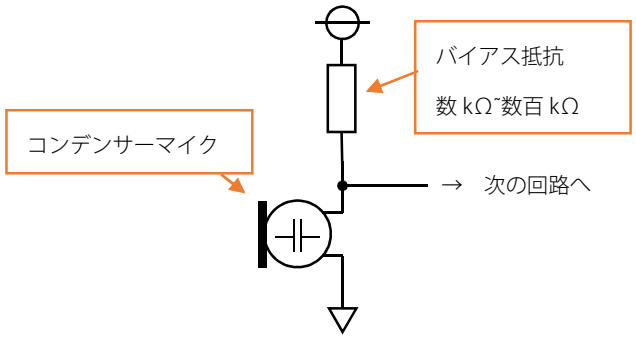
本機ではこのコンデンサーマイクを使用しています。

コンデンサーマイクの構造



金属板の間に電気を溜める必要があるので、コンデンサーマイクには必ず電源が必要になります。電源は下の図のように抵抗を介して接続して、マイクからの信号を取り出すのが一般的です。また、このような使い方をする抵抗のことをバイアス抵抗といいます。

コンデンサーマイクのバイアス回路例



・音声信号と他センサーの違い

音声信号と一般的なセンサーの信号は何が違うのでしょうか。信号波形で見ると分かりやすいと思います。例えば、手ばたきを一回した場合の音センサーから出てくる信号の状態です。

音声の信号の例

これが一般的なセンサーの出力信号、例えば光センサーに一瞬光があたるような場合は下図のような綺麗にＨとＬが分かるような信号になります。

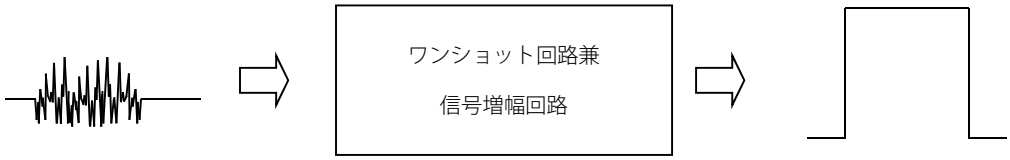
多くのセンサー信号の例

音声信号は波形で見ると分かるように、とても多くの波でできています。本来ならば、音が入力されたときに一度だけ処理を実行したいだけなのに、実際にはマイコンに何度も H/L が入力されてしまい、マイコンが意図しない動作をしてしまいます。音声信号の波形が、センサーの波形と同じようになると、マイコンで処理しやすくなります。そこで使われるのがワンショット回路です。

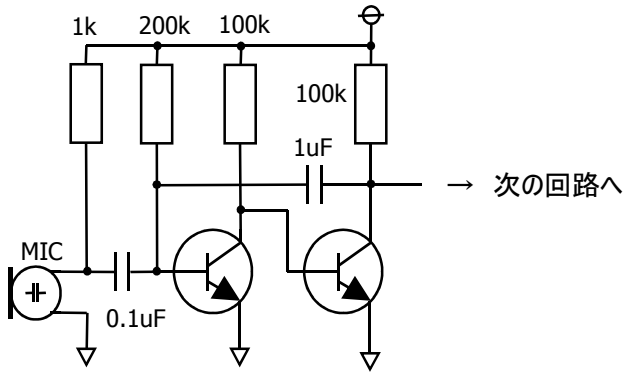
・ワンショット回路

ワンショット回路は、とにかく最初に一度入力信号に変化があると、一定時間信号を出力します。その一定の時間は、以下の回路の 1uF のコンデンサと 200kΩの抵抗の値で計算された時間が目安になります。
目安としての計算方法は　1(uF)　×　200(kΩ)　＝　0.2(秒)　となりますが、
実際には、使用する電源電圧や周りの部品の影響がありますので、回路を作った後は実際に測定する必要があります。本機の場合、実際に測定すると約 0.25 秒間ワンショット信号を出力しています。

ワンショット回路のイメージ



以下の回路図はワンショット回路の一例です。



・小信号増幅

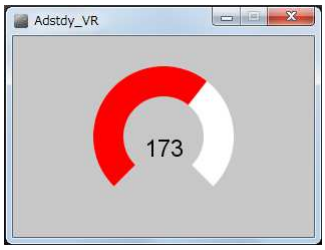
コンデンサーマイクから出力される信号は、その電圧レベルが小さく、そのままの電圧では、次につながる回路で上手く取り扱うことができません。このような場合は、信号を増幅します。本機ではワンショット回路の初段のトランジスタが信号を増幅する役目をしています。

パソコンに表示する　Processing 利用編

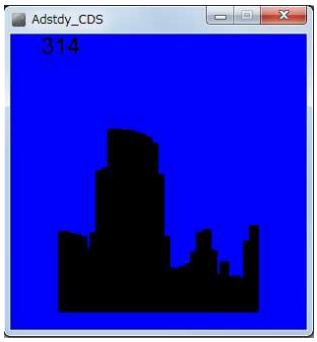
Arduino-IDE を使って Arduino のピンの状態を知ることはできますが、もっと直感的に見やすくしたい、また人に見せるために、もう少し格好良く表示したい場合もあると思います。そんな時に使えるのが、「Processing」ソフトです。Arduino-IDE と似た操作で、パソコン上に絵やグラフを描くものです。

Processing で作った表示例

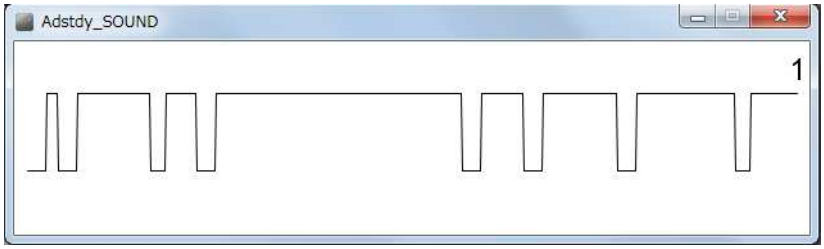
本機のボリュームの位置を絵で表示



CdS の光の変化をグラフで表示



音センサーの反応状態を波形で表示



ここでは実際に動かしてみることを体験してもらうために、Arduino、processing のサンプルプログラムをいくつか用意しています。

Processing の本格的な使い方は、インターネットや書籍などの説明や解説を参考にして学習してください。

Processing の入手方法

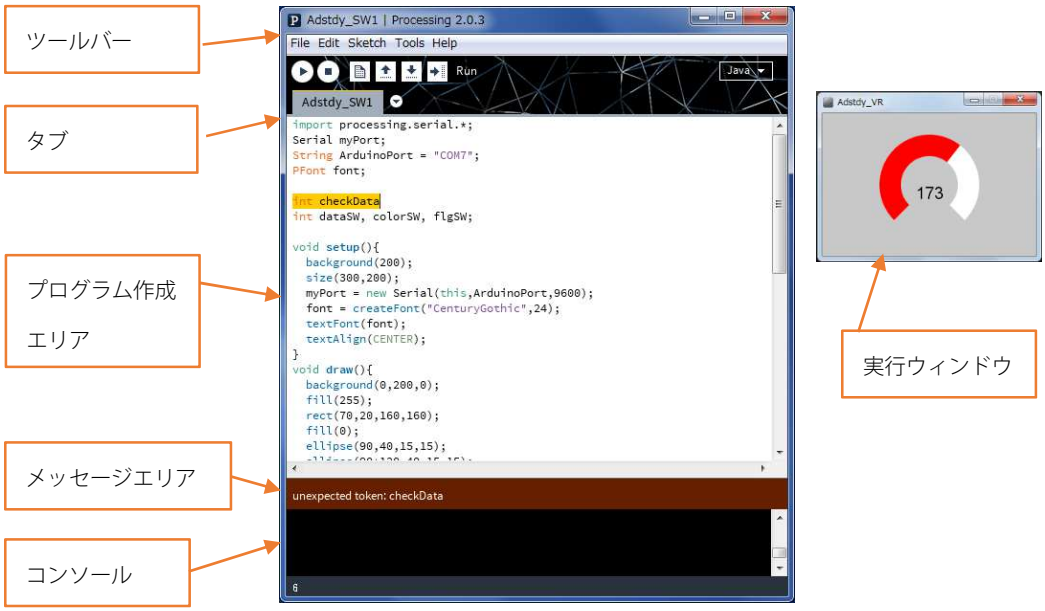
http://processing.org/

から入手できます。download の項目を選んで操作を進め、お使いの OS に合った Processing をダウンロードしてください。

ダウンロードが終わると、圧縮ファイル（.zip ファイル）が手に入ります。その圧縮ファイルをダブルクリックすると processing の名前の付いたフォルダが作成されます。そのフォルダをパソコン上の好きな場所、ディスクトップ上や、Program Files の中などに移動します。

フォルダの中の processing アイコンをダブルクリックすると、processing が起動します。

起動画面の説明



サンプルプログラムを動かす準備

【COM ポートの確認】

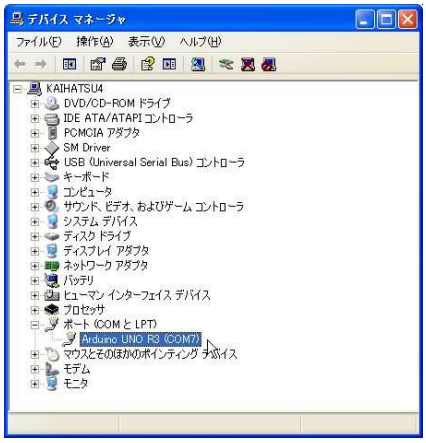
Processing はパソコンの COM ポートと呼ばれる場所を使って Arduino 基板と連携して動きます。そこでまず、Arduino が使っている COM ポートの番号を知り、その後、その COM ポートの番号に Processing のプログラムを合わせる必要があります。

➡COM ポートの確認方法

使用する Arduino 基板は PC に接続しておきます。

WindowsVista/7 では、「コントロールパネル」→「システムとセキュリティ」→「デバイスマネージャー」と進みます。

デバイスの一覧が表示されますので、「ポート（COM と LPT)」という項目の下を確認します。



Arduino 基板が接続されたポートが表示されています。図では Arduino 基板が「COM7」に接続されていることが分かります。この COM の番号を覚えておきます。

【COM ポートの番号を反映する】

Processing のプログラムの中の COM ポートの番号を書き換えます。

使用するサンプルプログラムを開いて、下図で示した場所の数字を書き換えます。

例

確認した COM ポートが　COM7 のとき

```
string　ArduinoPort　=“COM7”　　//確認した COM ナンバーに書き換えます
```

【サンプルプログラムの実行】

Processing のサンプルプログラムは、Arduino から COM ポートを通じて渡されたデータと連携して動作しますので、Arduino には動作目的にあったデータを出力するプログラムを書き込んでおく必要があります。

つまり手順としては、

Step1：Arduino 側のプログラムを作成して、Arduino に書き込む。

Step2：Arduino 基板が使っている COM ポート番号を確認する。

Step3：processing 側のプログラムを作成して、COM ポート番号を確認し、実行する。

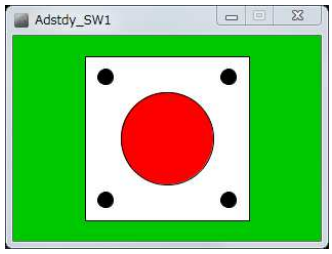
という流れになります。

実際にやってみると雰囲気がつかめると思います。本書では各センサーの動きに連動して processing で作成したサンプルを用意してみました。是非やってみてください。

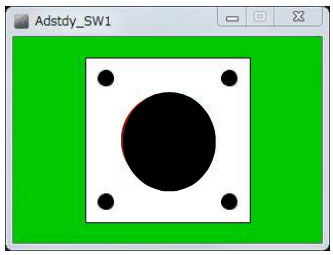
サンプル 1　「SW を押すと、画面上のスイッチの色が変わる。」

●Arduino 側に書き込むプログラム

```
#define SW A4

boolean valSW;

void setup(){
  Serial.begin(9600);
  pinMode(SW, INPUT_PULLUP);
}
void loop(){
  valSW = digitalRead(SW);
  Serial.write(77);
  Serial.write(valSW);
  delay(100);
}
```



SW を押すと色が変わります



●processing で作成するプログラム

```
import processing.serial.*;
Serial myPort;
String ArduinoPort = "COM4";
PFont font;
int checkData;
int dataSW, colorSW, flgSW;

void setup(){
  background(200);
  size(300,200);
  myPort = new Serial(this,ArduinoPort,9600);
  font = createFont("CenturyGothic",24);
  textFont(font);
  textAlign(CENTER);
}
void draw(){
  background(0,200,0);
  fill(255);
  rect(70,20,160,160);
  fill(0);
  ellipse(90,40,15,15);
  ellipse(90+120,40,15,15);
  ellipse(90,40+120,15,15);
  ellipse(90+120,40+120,15,15);
  if(dataSW == 0){
    if(flgSW == 1)  colorSW = ~colorSW;
    flgSW = 0;
  }else{
    flgSW = 1;
  }
  if(colorSW == 0){
    fill(0);
  }else{
    fill(255,0,0);
  }
  ellipse(150,100,90,90);
}
void serialEvent(Serial p){
  if(myPort.available() > 2){
    checkData = myPort.read();
    if(checkData == 77){
      dataSW = myPort.read();
    }
  }
}
```

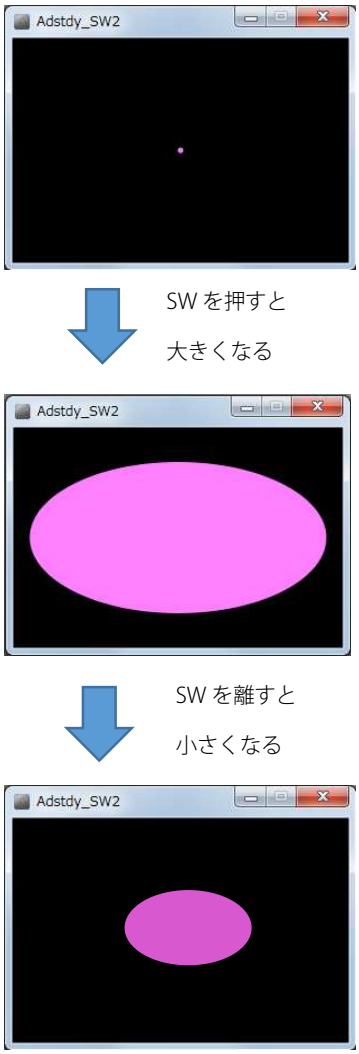
サンプル 2　「SW を押し続けると円が大きくなり、離すと小さくなる。」

●Arduino 側に書き込むプログラム

Arduino に書き込むプログラムは、サンプル 1 のものと同じものを利用します。

●processing で作成するプログラム

```
import processing.serial.*;
Serial myPort;
String ArduinoPort = "COM4";
PFont font;
int checkData, dataSW =1;
float maxSW;
int dx = 5, dy = 5;

void setup(){
  background(200);
  size(300,200);
  myPort = new Serial(this,ArduinoPort,9600);
  font = createFont("CenturyGothic",24);
  textFont(font);
  textAlign(CENTER);
}
void draw(){
  background(0);
  noStroke();
  if(dataSW == 0){
    if(dx <=400){
      dx = dx + 2;
      dy = dy + 1;
    }
  }else{
    if(dx > 5){
      dx = dx - 2;
      dy = dy - 1;
    }
  }
  fill(255,128,255);
  ellipse(150,100,dx,dy);
}
void serialEvent(Serial p){
  if(myPort.available() > 2){
    checkData = myPort.read();
    if(checkData == 77){
      dataSW = myPort.read();
    }
  }
}
```



サンプル 3 「VR の回し具合を表示する。」

●Arduino 側に書き込むプログラム

```
#define VOL A0
int valVR;

void setup(){
  Serial.begin(9600);
}
void loop(){
  valVR = analogRead(VOL);
  valVR = map(valVR, 0, 1023, 0, 255);
  Serial.write(77);
  Serial.write(valVR);
  delay(100);
}
```

本機のボリュームの角度と連動して、赤色の部分が動きます

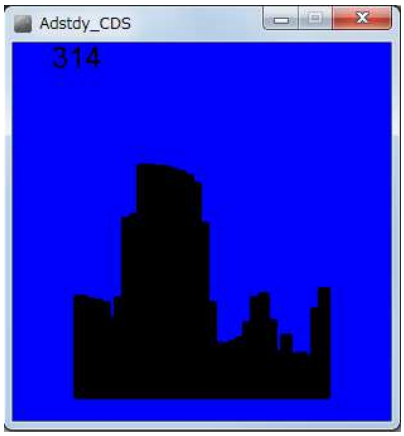
●processing で作成するプログラム

```
import processing.serial.*;
Serial myPort;
String COMNUM = "COM4";
PFont font;
int checkData;
int dataVR;
float valueVR;

void setup(){
  background(200);
  size(300,200);
  myPort = new Serial(this,COMNUM,9600);
  font = createFont("CenturyGothic",24);
  textFont(font);
  textAlign(CENTER);
}
void draw(){
  background(200);
  fill(255);
  arc(150,100,140,140,HALF_PI + QUARTER_PI, TWO_PI + QUARTER_PI);
  fill(255,0,0);
  arc(150,100,140,140,HALF_PI + QUARTER_PI, HALF_PI + QUARTER_PI + radians(dataVR));
  fill(200);
  noStroke();
  ellipse(150,100,80,80);
  fill(0);
  text(dataVR,150,120);
}
void serialEvent(Serial p){
  if(myPort.available() > 2){
    checkData = myPort.read();
    if(checkData == 77){
      valueVR = myPort.read();
      valueVR = map(valueVR, 0, 255, 0, 270);
      dataVR = round(valueVR);
    }
  }
}
```

サンプル４　「光センサーの受光状態を表示する。」

●Arduino 側に書き込むプログラム

```
#define CDS A2
int valCDS;
byte valCDS_high, valCDS_low;

void setup(){
  Serial.begin(9600);
}
void loop(){
  valCDS = analogRead(CDS);
  valCDS_high = valCDS >>8;
  valCDS_low = valCDS;
  Serial.write(77);
  Serial.write(valCDS_high);
  Serial.write(valCDS_low);
  delay(100);
}
```



CdS の周囲が明るければ黒の棒グラフが低く、暗ければ黒の棒グラフが高くなります。

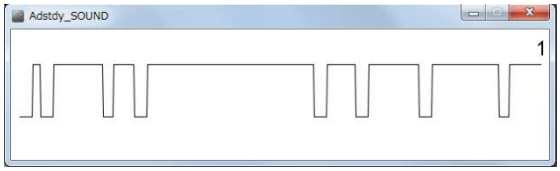
●processing で作成するプログラム

```
import processing.serial.*;
Serial myPort;
String ArduinoPort = "COM4";
PFont font;

int checkData,dataCDS_dsp;
int dataCDS, dataCDS_high, dataCDS_low;
int x;
int[] y = new int[201];

void setup(){
  background(0,0,255);
  size(300,300);
  myPort = new Serial(this,ArduinoPort,9600);
  font = createFont("CenturyGothic",24);
  textFont(font);
  textAlign(CENTER);
  stroke(0);
  strokeWeight(5);
  strokeCap(ROUND);
}
void draw(){
  background(0,0,255);
  for(int j=200; j>0; j--){
    y[j] = y[j-1];
  }
  y[0] = dataCDS_dsp;
  for(x=50; x<250; x++){
    line(x,280,x,(280-y[x-50]));
  }
  fill(0);
  text(dataCDS,50,20);
}
void serialEvent(Serial p){
  if(myPort.available() > 3){
    checkData = myPort.read();
    if(checkData == 77){
      dataCDS_high = myPort.read();
      dataCDS_low = myPort.read();
      dataCDS = dataCDS_high * 256 + dataCDS_low;
      dataCDS_dsp = dataCDS >> 2;
    }
  }
}
```

サンプル 5　「音センサーの入力状態を波形で表示する。」

●Arduino 側に書き込むプログラム

```
#define SND A1

int valSND;

void setup(){
  Serial.begin(9600);
}
void loop(){
  valSND = digitalRead(SND);
  Serial.write(77);
  Serial.write(valSND);
  delay(50);
}
```



音センサーが音を検出すると L を、何も検出していないときは H となる波形を描きます。

●processing で作成するプログラム

```
import processing.serial.*;
Serial myPort;
String ArduinoPort = "COM4";
PFont font;
int checkData;
int dataSND = 1;
int x;
int[] y = new int[601];

void setup(){
  background(255);
  size(620,150);
  myPort = new Serial(this,ArduinoPort,9600);
  font = createFont("CenturyGothic",24);
  textFont(font);
  textAlign(CENTER);
  for(int j=600; j>=0; j--){
    y[j] = 1;
  }
}
void draw(){
  background(255);
  for(int j=0; j<600; j++){
    y[j] = y[j+1];
  }
  y[600] = dataSND;
  for(x=10; x<610; x++){
    line(x+1,(100-y[x-9]*60),x,(100-y[x-10]*60));
  }
  fill(0);
  text(dataSND,610,30);
}
void serialEvent(Serial p){
  if(myPort.available() > 3){
    checkData = myPort.read();
    if(checkData == 77){
      dataSND = myPort.read();
    }
  }
}
```

Arduino 基板と Arduino-IDE

（1）　Arduino 基板

●いろいろな Arduino

市販されている Arduino には多くの種類があります。Arduino の最も標準的な Arduino-UNO に加えて、ブレッドボード上でのテストに便利な Nano、衣服に縫い付けることを考えられた LilyPad、PC との通信部分を取り外して安価にした Pro、入出力が強化された Mega などがあります。

これら全ての Arduino は、プログラムを作る仕組みが同じで、同じ Arduino-IDE といわれるプログラム開発ソフト（総合開発環境といいます）で作成することができます。

プログラムの書き換えに特別な機器は必要としないので、プログラムの書き込みが手軽ですぐに使い始められます。

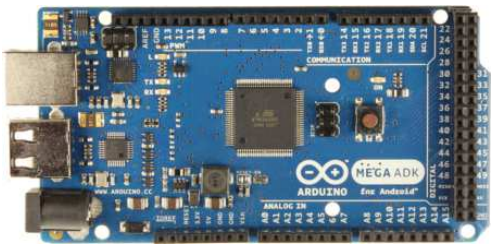
Arduino の例

Arduino-UNO

Arduino-Nano

LilyPad

Arduino-Pro



Arduino-Mega

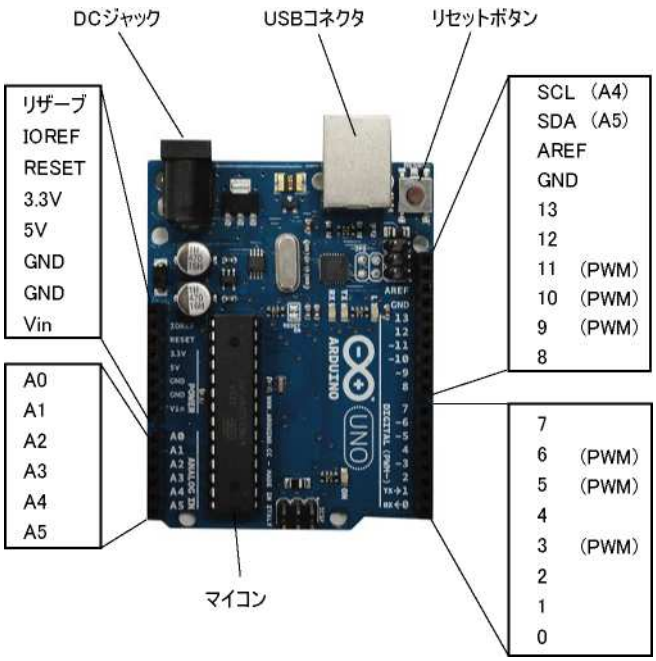
本書で説明に使用しているシールドは、**Arduino-UNO(R3)**と接続します。

●機能説明

【ピン配置図】

Arduino-UNO(R3)
の場合

※ Arduino-UNO(R3) 以外ではピン数、機能が異なる場合があります。



【ピンの機能】

機能	説明
リザーブ	将来の拡張用に予約されているピンで使うことはできません。
IOREF	ArdunoUNOではVccに接続されています。
RESET	リセットボタンに接続されています。リセットボタンの追加に使用できます。
3.3V	3.3Vの電源になります。
5V	5Vの電源になります。シールド基板の電源はここから供給しています。
GND	グラウンドです。シールド基板の電源のマイナスはここにつながっています。
GND	
Vin	DCジャックから入力された電源につながっています。 （内部でダイオードを経由しますので、電圧が少し低くなります。）

機能1	機能2	説明
A0		アナログ入力ポートです。 センサーなどの接続に向いています。 デジタル入出力ピンとして使うこともできます。
A1		
A2		
A3		
A4	SCL	I2C(Wire)通信を使うときには、SCL/SDAとしてここを使います。
A5	SDA	

機能1	機能2	説明
SCL	(A4)	A4,A5はここにもつながっています。
SDA	(A5)	
AREF		AD変換器の基準電圧のためのピンです。使用するためには特別な命令や配線が必要です。
GND		グラウンドです。
13		13～0はデジタル入出力ピンとして使用します。 11,10,9,6,5,3はPWM制御できるピンとして使用することもできます。
12		
11	PWM	
10	PWM	
9	PWM	
8		
7		
6	PWM	
5	PWM	
4		
3	PWM	
2		
1	TX	1,0はUSB経由でPCと通信するときに使用します。
0	RX	

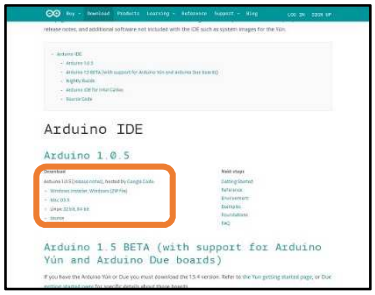
（2）　Arduino-IDE の準備

Arduino を制御するためには、プログラムを作成し、Arduino 基板へ書き込みを行う必要があります。このプログラムを作成するエディタとプログラム書き込み機能を持った開発環境のことを、Arduino-IDE といいます。

Arduino-IDE の入手と起動

1. まず Arduino のホームページにアクセスします。
   http://www.arduino.cc/ (英語のサイトです)

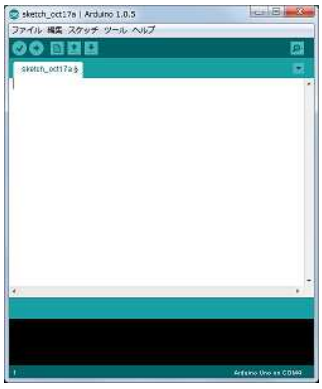
2. ダウンロード画面から、お使いの OS に対応したソフトを選択し、適当なフォルダーに保存します。



画面は変わる可能性があります
画面は 2013 年 10 月現在のものです。

3. ダウンロード後、保存したフォルダー内に、arduino-X.X.X-XXX が作成されます。このファイルは圧縮されているので解凍(展開)ソフトなどで解凍してください。
   ※XXX は使用する OS やソフトのバージョンによって異なります。
4. ファイルを解凍すると、Arduino-1.0.1 のように、バージョン番号が付いたフォルダーが作成されます。
5. 先ほど解凍したフォルダー内にある、arduino のアイコンをダブルクリックすると、arduino-IDE が起動し画面が表示されます。

↓フォルダーの中の arduino をクリック　　IDE 起動画面



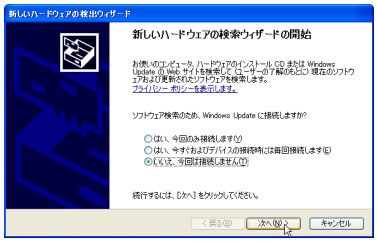
ドライバーのインストールとシリアルポート(COM ポート)の確認

Windows の場合

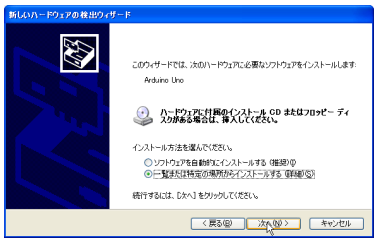
次に、Arduino 基板とパソコンが通信するために必要なドライバーをインストールします。

1. Arduino 基板とパソコンを USB ケーブルで接続します。
2. WindowsXP の場合：

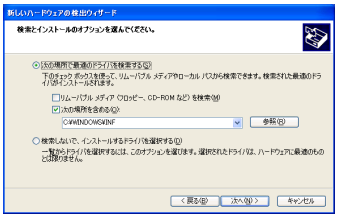
   接続後に「新しいハードウェアの検出」画面が表示されますので、「いいえ。今回は接続しません。」を選択して次へ進みます。

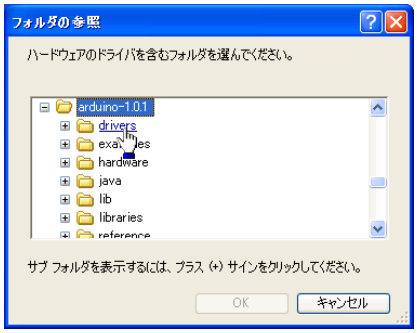


   「一覧または特定の場所からインストールする。」を選択して次へ進みます。



   「次の場所を含める。」で、参照のボタンをクリックして、先ほど解凍したフォルダーの中にある「Drivers」を選択して、次へ進みます。





   ドライバーのインストールが始まり、自動的に完了します。

   完了後に、もう一度ドライバーのインストールを求められる場合は、1 回目と同じように進めて完了させます。

   WindowsVista/7 の場合：

   「デバイスドライバーソフトウェアをインストールしています。」と表示され、しばらく待つと、パソコンが自動でドライバーを検索しインストールが完了します。もしも自動でインストールができない場合は手動でインストールする必要があります。

   ＊トラブルシューティング「ドライバーを手動でインストールする。」をお読みください。

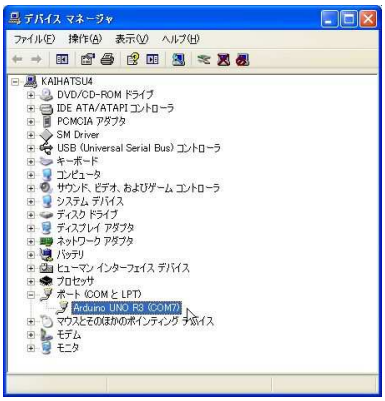
3. 次に Arduino 基板が、どのシリアルポート（COM ポート）に接続されたか確認します。

   WindowsXP では、マイコンピュータのアイコンを右クリックして「プロパティ」を選択し、「ハードウェア」のタブの中にある「デバイスマネージャー」をクリックします。

   WindowsVista/7 では、「コントロールパネル」→「システムとセキュリティ」→「デバイスマネージ

ャー」と進みます。

デバイスの一覧が表示されますので、「ポート（COM と LPT)」という項目の下を確認します。



Arduino 基板が接続されたポートが表示されます。

図では Arduino 基板が「COM7」に接続されていることが分かります。この COM の番号を覚えておきます。

4. Arduino-IDE 画面の「ツール」メニューから、「マイコンボード」を選択し、パソコンに接続した Arduino 基板を選択します。
5. 次に Arduino-IDE 画面の「ツール」メニューから、「シリアルポート」を選択し、先ほど確認した COM ポートの番号と同じものを選択します。

これで Arduino を使用する準備が整います。

ドライバーのインストールとシリアルポート（COM ポート）の確認

Mac OS X の場合

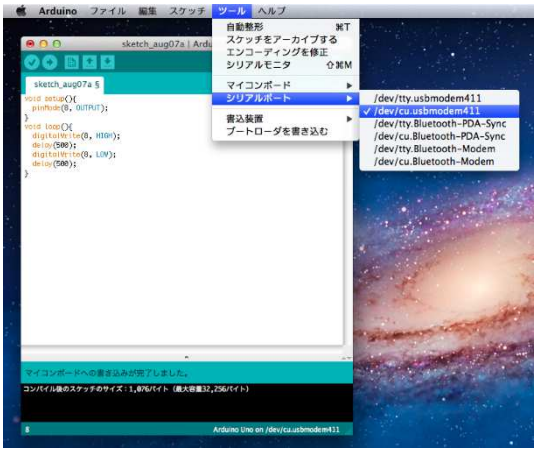
Mac OS X の場合、ダウンロードが完了すると自動的にドライバーがインストールされますので、表示されるメッセージに従い必要に応じて管理者パスワードの入力や再起動を行ってください。

Mac OS X でシリアルポートを選択する。

1.Arduino 基板をパソコンと接続します。

2.Arduino-IDE を起動し、画面の「ツール」メニューから、「マイコンボード」を選択し、パソコンに接続した Arduino 基板を選択します。

3.次に Arduino-IDE 画面の「ツール」メニューから、「シリアルポート」を選択し、「/dev/cu.usbmodem-」または「/dev/tty.usbmodem-」ではじまる項目を選んでください。



これで Arduino を使用する準備が整います。

起動画面

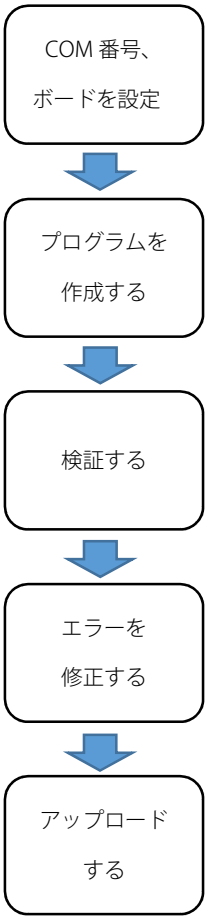
新規ファイル　：　新しいプログラム作成エリアを開きます。
開く　：　保存しているプログラムを開きます。
保存　：　作成したプログラムを保存します。
検証(コンパイル)　：　正しい文法でプログラムが作成されているかチェックします。
シリアルモニタ　：　シリアルデータを表示します。
（データに何を表示させるかあらかじめプログラムに書く必要があります。）
マイコンボード　：　プログラムを PC に接続している Arduino 基板に書き込みます。
に書き込む　アップロードと表示されている場合もあります。
プログラム作成エリア　：　プログラムを編集するエリアです。
メッセージエリア　：　操作に応じてメッセージやエラーが表示されます。
コンソール　：　メッセージやエラーの詳細が表示されます。(※)
ボード・COM 番号　：　Arduino-IDE で設定している、Arduino 基板の種類と COM ポート番号が表示されます。

※　ほとんどの場合、非常に分かりづらい意味不明の文字が表示されます。そんなときは、表示された文字や単語を使って検索サイトで調べることになります。

（3）　Arduino-IDE の使い方

●プログラムの作成からアップロードまでの流れ

プログラムを作成し、Arduino 基板に書き込むまでのおおよその流れは以下のようなイメージになります。



Arduino-IDE 起動後に一度だけ設定すれば OK です。
特に COM ポートの設定は忘れることが多いので注意しましょう。
Arduino 基板と PC は USB ケーブルでつないでおく必要があります。

プログラム作成エリアに、プログラムを作成します。

Arduino 基板に書き込む前に検証します。
エラーが表示された場合はエラーを修正します。
エラー表示は、どこが間違っているか的確に表示してくれることもありますが、ほとんどの場合、読んでもよく分からない表示になっています。それでも、エラーの発生している近くを指摘していることが多いので、エラー表示された部分の周辺をチェックすることから始めましょう。

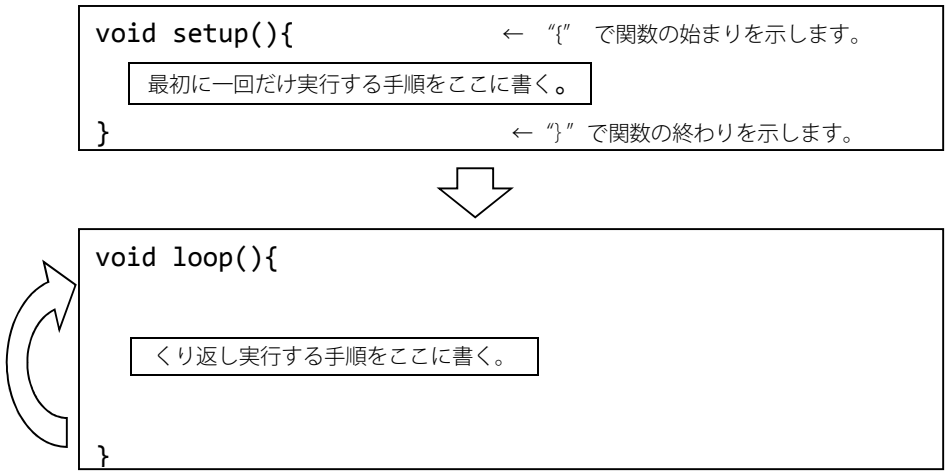
作成したプログラムを Arduino 基板に書き込みます。書き込み完了後はメッセージエリアに「書き込みが完了しました」と表示されます。

●Arduino のプログラムの基本

Arduino-IDE で作成したプログラムをアップロードすると、Arduino 基板のマイコンの中に書き込まれます。書き込まれたプログラムは、基板の電源を ON/OFF しても消えません。書き込まれたプログラムは、基板の電源を ON 後、またはリセット後に、自動で実行されます。

Arduino のプログラムには、「setup」と「loop」という 2 つの関数が必要です。
関数とは、Arduino 基板上のマイコンに、何をどんな順番で行わせるかを記述した手順書です。
電源 ON 後に、「setup」の中の手順を一回だけ実行し、続けて、「loop」の中の手順を、電源を切るまでくり返し実行するということを覚えてください。setup と loop は省略できません。

プログラムを実行するイメージ



void setup()、void loop() って何？と思うかもしれませんが、これは Arduino-IDE で、setup 関数と loop 関数を使うときの決められた形ですので、「この形で使うもの。」と覚えてください。

●よくやってしまうミス

しっかり注意しながらプログラムしたつもりでも、書き忘れやタイプミスをやってしまいます。とくに以下のようなミスはエラー表示でどこを示しているのか分かりづらいので十分注意しましょう。

；　←セミコロンを忘れる。またはセミコロンではなく、：　←（コロン）を使ってしまっている。
，　←(コロン)ではなく、　．　←（ピリオド）を使ってしまっている。
｛　と　｝　←（波カッコ）が対になっていない。　｛ ｝が多重になるときは要注意です。
全角スペースを使っている。プログラムの中では全角文字は使用できません。

●Arduino の言語

ここに書いたものが全てではありませんが、Arduino のプログラムでよく使う文法や関数をまとめています。Arduino-IDE のその他の関数や詳細については書籍などで学習を行ってください。

基本となる文法

形式　　;　（セミコロン）

説明　　文の終わりに使います。基本的に Arduino では文の終わりにセミコロンが必要です。

セミコロンによって文がどこで区切られているかを示します。日本語の句点に似ています。

使用例

```
int redLED 9 ;
pinMode(9, OUTPUT);
delay(100);
```

形式　　{　}　　（波カッコ）

説明　　複数の文をまとめるために使用します。

使用例

```
if(x==1){
  文 1;
  文 2;
}
```

デジタル入出力

形式　　`pinMode(pin, mode)`

説明　　ピンの動作を入力または出力または、プルアップ機能を利用した入力に設定します。

パラメータ　`pin`　　設定したいピンの番号

`mode`　　`INPUT`　または　`OUTPUT` または　`INPUT_PULLUP`

使用例

```
pinMode(9, OUTPUT);          9 番ピンが出力になります。
pinMode(13,INPUT);           13 番ピンが入力になります。
pinMode(A0, INPUT_PULLUP);   A0 番ピンがプルアップされた入力になります。
```

形式　`digitalWrite(pin, value)`

説明　指定したピンを、`HIGH` または `LOW` にします。`HIGH` は 5V、`LOW` は 0V(GND)になります。

パラメータ　`pin`　設定したいピンの番号

`value`　`HIGH` か `LOW`　（または、`1`　か　`0`）

使用例　`digitalWrite(9, HIGH);`　9 番ピンが `HIGH` になります。

`digitalWrite(2, 0);`　2 番ピンが `LOW` になります。

形式　`digitalRead(pin)`

説明　指定したピンの状態を調べます。結果は `HIGH` か `LOW` になります。

パラメータ　`pin`　調べたいピンの番号

使用例　`x = digitalRead(9);`　x に 9 番ピンの状態が `HIGH` か `LOW` で記憶されます。

例えば 9 番ピンが `GND` に接続されている場合は `LOW` になります。

アナログ入出力

形式　`analogRead(pin)`

説明　指定したアナログピンの状態を調べます。結果は 0 から 5V の電圧範囲を 0 から 1023 の範囲に変換した値になります。Arduino-UNO では A0～A5 がアナログピンとして利用できます。

パラメータ　`pin`　調べたいアナログピンの番号

使用例　`x = analogRead(A0);`　x に A0 番ピンの状態が 0 から 1023 の整数値で記憶されます。

形式　`analogWrite(pin, value)`

説明　指定したピンから PWM 波を出力（アナログ出力）します。

`analogWrite` の前に `pinMode` で出力にする必要はありません。

Arduino-UNO では３，５，６，９，１０，１１番ピンがアナログ出力ピンとして利用できます。

`analogWrite` を実行すると、次に同じピンで `analogWrite` を実行するまで PWM 波が出力されます。

パラメータ　`pin`　出力に設定するピン番号

`value`　PWM 出力のデューティ比（0 から 255 の範囲で設定します。）

使用例　`analogWrite(3, 64);`　3 番ピンからデューティ比が 64 の波形を出力します。

`analogWrite(3, 0);`　3 番ピンからはデューティ比が 0(=0V)が出力されます。

`analogWrite(3, 255);`　3 番ピンからはデューティ比が 255(=5V)が出力されます。

制御文

形式　　if (条件) { 条件に一致したときに実行する文 }

説明　　カッコ内の条件が true つまり条件が満たされている場合は、次に続く波カッコ内の文を実行します。
false つまり条件が満たされていない場合は、波カッコ内の文を実行せずに次に進みます。
波カッコの中の文が 1 つだけの場合は波カッコを省略できます。

使用例
```
if ( x == 1 ) {
  digitalWrite(9, HIGH);
  delay(100);
}

if ( y < 500 ) digitalWrite(4, LOW);
```

注意点　　比較に利用する　＝（等号）は　＝＝のように 2 つ書かなければいけません。
＝が 1 つの場合は代入となり、全く違った意味になり、結果としていつも true と判断されます。

形式　　if(条件){条件に一致したときに実行する} else {条件に一致しないときに実行する}

説明　　カッコ内の条件に一致したときに実行する文と、一致しないときに実行する文をそれぞれ分けて書くことができ、if よりも細かく制御できます。

使用例
```
if (x == 1) {
  digitalWrite(9, HIGH);
}else{
  digitalWrite(9, LOW);
}
```

形式　　for (初期化 ; 条件式 ; 加算) {実行する文}

説明　　条件を満たしている間、波カッコで囲まれた文を繰り返し実行します。
カッコの中は 3 つの部分から成り立っています。
動作は、まず初期化が一度だけ実行されます。次に、条件式がテストされ条件を満たしていれば、加算と波カッコ内の文が繰り返し実行されます。次に条件がテストされたときに条件を満たしていなければ、そこで繰り返しが終わります。

使用例
```
for ( i = 0; i <= 255; i++){
  analogWrite(9, i);
  delay(100);
}
```

形式　　while(条件) {条件に一致したときに実行する文}

説明　　カッコ内の条件を満たさなくなるまで（false になるまで）波カッコ内を永遠に繰り返します。

使用例

```
x = 1;
While( x < 50 ){
  x++;
}
```

時間

形式　　delay(ms)

説明　　カッコ内で指定された時間待って、次の文の処理へ進みます。単位はミリ秒です。

パラメータ　ms　　次の処理までの待ち時間。単位はミリ秒。（1 ミリ秒は 1000 分の 1 秒です。）

使用例

```
digitalWrite(9,HIGH);
delay(500);
digitalWrite(9,LOW);
```

便利な機能

形式　　#define　定数名　値

説明　　検証時（コンパイル時）に自動的に定数名を値に変換します。プログラムを見やすく、間違いを少なくするために有効です。#を付けることを忘れないようにします。

#define 文の最後には；（セミコロン）は必要ありません。

使用例

```
#define ledPin 4
digitalWirte(ledPin, HIGH);
```

ledPin は検証時に自動的に 4 に置き換えられます。

形式　　#include　<ライブラリ名>

説明　　外部に用意されているライブラリをプログラムに取り入れたいときに使います。

ライブラリは Arduino-IDE にあらかじめ用意されているものや、インターネットから入手できるものなどがあります。

#include 文の最後には；（セミコロン）は必要ありません。

使用例

```
#include  <LiquidCrystal.h>
```

LCD 表示器のライブラリを使うことができるようになります。

形式　　// または /* ～ */

説明　　プログラムの中にメモしておきたい内容、つまりコメントを書くときに使います。
コメントは検証（コンパイル）から無視され、プログラムに影響しません。
コメントは、プログラムの動作内容を理解したり、思い出したりするために使われます。

使用例　x = 1;　//ここがコメントになる。　ダブルスラッシュの後ろはすべてコメントです。

```
/*      ここから複数行のコメントになります。
  digitalWrite(3,LOW);
  x = analogRead(A0);
*/      ここまでがコメントです。
```

シリアル通信

形式　　Serial.begin(speed)

説明　　シリアル通信の通信速度を設定します。

パラメータ　speed　　通信スピードの値。一般的にパソコンと通信する場合は、
300,1200,2400,4800,9600,14400,28800,38400,57600,11520
から 1 つを選びます。

使用例　Serial.bigin(9600);

形式　　Serial.print(data, format)

説明　　シリアルポートに指定されたデータを出力します。

パラメータ　data　　すべての整数型と String 型(*)
(※アルファベット、数字、区切り記号などの文字が連続したもの。
ダブルクオテーションで囲むことで String 型になります。)

format　data を変換する方法を指定します。省略可能です。
省略すると、data は 10 進数の数値として取り扱われます。

使用例　Serial.print(58);　　58 を出力します。
Serial.print(58, HEX);　　3A を出力します。
Serial.print(“Hello”);　　Hello を出力します。

形式　　Serial.println(data, format)

説明　　データの最後に改行コードを付けて出力します。それ以外の動作は Serial.print と同じです。

使用例　Serial.println(58);

データ型

変数そのものや、変数に代入される値の範囲に合わせて、データ型を選ぶ必要があります。
取り扱える範囲を超えた計算を行うと、変数は期待している値と違ったものになるので注意が必要です。
データ型と、値の範囲は以下の表のようになります。

データ型	値の範囲
boolean	true か false のどちらかの値を取り扱います。
byte	0 から 255 までの値を取り扱います。負の値は扱えません。
int	-32,768 から 32.767 までの値を取り扱います。
long	-2,147,483,648 から 2,147,483,647 までの値を取り扱います。

※これ以外にもデータ型はあります。代表的なものだけ紹介しています。

使用例　`int x ;`　　x は int 型の範囲の数値を取り扱う変数になります。
　　`byte y = 128 ;`　　byte 型の変数 y に 128 を代入します。

演算子

形式　＝
説明　定数や計算結果を変数に記憶させるには、＝(等号) を使います。＝は代入演算子といわれます。
使用例　`a = 10;`　　a は 10 になります。
　　`b = a;`　　b は a と同じ値になります。

形式　`+ , - , * , /`
説明　加算、減算、乗算、除算を行います。これらは算術演算子といわれます。
　　データ型によって計算結果が期待しているものと違ってくることがありますので、
　　計算結果を格納するのに十分な大きさの型になっていることに注意が必要です。
　　※プログラムをスマートにする演算子は他にもあります。ここでは基本的なものだけです。
使用例　`a = 10 + 1;`　　a は 11 になります。
　　`a = 9 - 1;`　　a は 8 になります。
　　`a = 3 * 33;`　　a は 99 になります。
　　`a = 9 / 3;`　　a は 3 になります。
　　`a = 9 / 4;`　　a のデータ型によって変わります。a が int 型であれば、
　　a は 2 になります。
　　※少数が取り扱いたい場合は別のデータ型を選ぶ必要があります。

形式　　++　--

説明　　++をインクリメントといい、1 を足します。--をデクリメントといい、1 を引きます。

使用例　++a;　　a に 1 を足して a に記憶します。

　　--a;　　a から 1 を引いて a に記憶します。

比較演算子

形式　　== , != , < , > , <= , >=

説明　　2 つの変数や定数の関係を調べるには、==,!=,<,>,<=,>=を使います。

関係が正しい場合を真(true)といい、正しくない場合を偽(false)といいます。

演算子	使用例	結果
==	a==b	a と b が等しければ真
!=	a!=b	a と b が等しくなければ真
<	a<b	a が b より小さければ真
>	a>b	a が b より大きければ真
<=	a<=b	a が b 以下なら真
>=	a>=b	a が b 以上なら真

使用例　a = (1>2);　　a は false(=0)になる。

a = (1!=2);　　a は true(=1)になる。

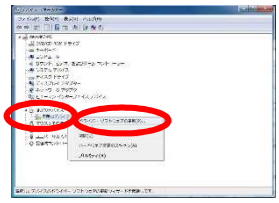
a = (1==2);　　a は false(=0)になる。

トラブルシューティング

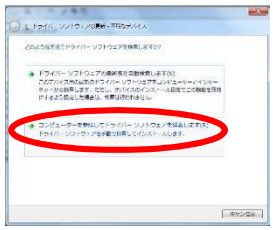
WindowsVista/7 に手動でドライバーをインストールする

（1）　コントロールパネル　→　　システムのセキュリティ　　→　デバイスマネージャー　と選んで、デバイスマネージャーの画面を開きます。

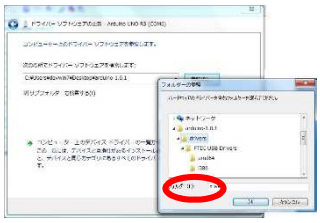
（2）　デバイスマネージャー画面の中に、「不明なデバイス」という表示がされていますので、その文字の上で右クリックして、「ドライバーソフトウェアの更新・・・」を選びます。



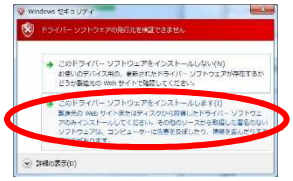
（3）　「ドライバーソフトウェアの更新」画面から、「コンピューターを参照してドライバーソフトウェアを検索します。」を選びます。



（4）　「次の場所でドライバーソフトウェアを検索します。」→「参照」を選び、最初に保存した Arduino のフォルダーの中から Drivers を選びます。



(5)　Windows セキュリティの画面が表れますので、「このドライバーソフトウェアをインストールする。」を選びます。



(6)　しばらく待つと、ドライバーのインストール完了画面が表示されます。

(7)　もう一度デバイスマネージャーを表示して、Arduino 基板が、ポート（COM と LPT）に登録されていることを確認してください。