Vol.1 動作チェックを行いながら、プログラム操作に慣れよう。

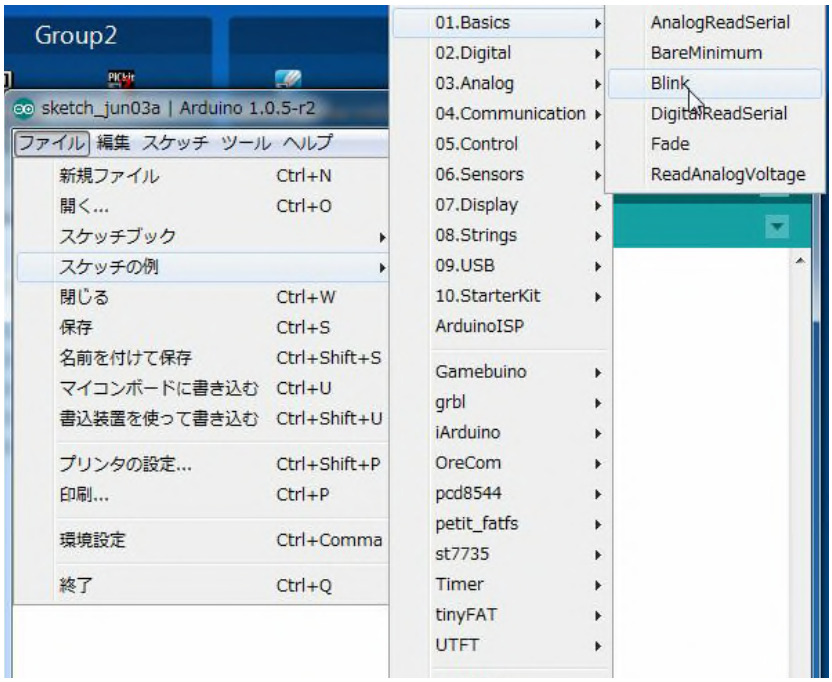
ver1．2．0以降　対応

プログラミングのスタート

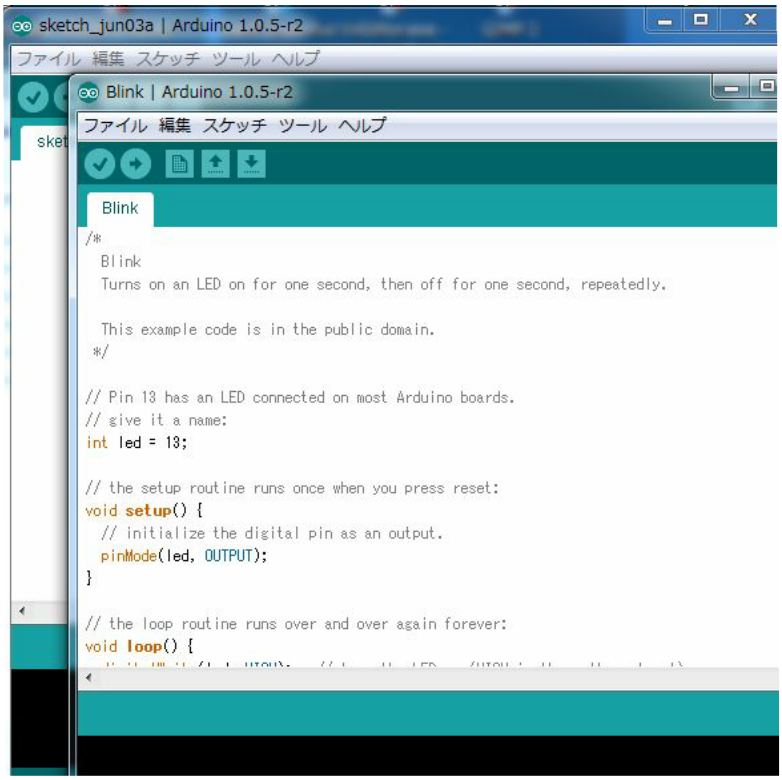
プログラムに慣れるために、サンプルの一部を変更しながらボードの機能を確認していきます。

サンプルスケッチの読み込み

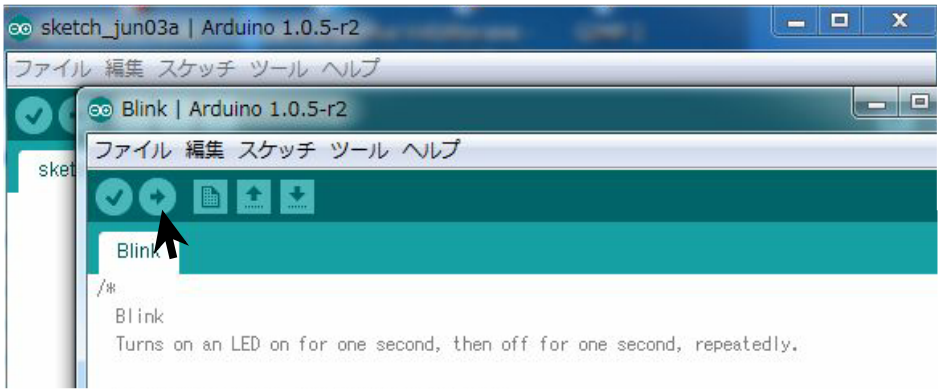
ファイル>>スケッチの例>>01.Basics>>Blink　を選びます。



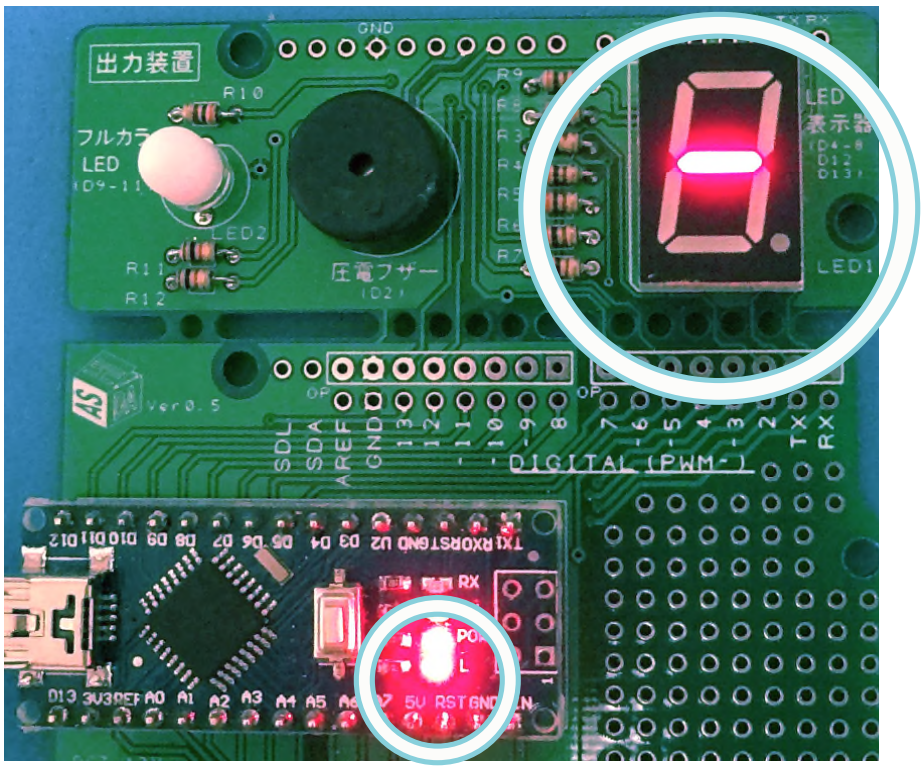
スケッチが読み込まれ、新しいウインドウにコードが表示されます。



そのまま、右矢印の書かれた丸いボタンをクリックしてマイコンボードに書き込みます。



書き込みが完了すると、マイコン上にあるＬＥＤのひとつと、ＬＥＤ表示器の真ん中の横棒が点滅します。



これからプログラムに慣れるために、少しプログラムを変更してみます。
もしおかしくなったら、そのウィンドウを消して、最初の「スケッチの例」を開くところからやり直せば大丈夫です。入力はかならず半角文字で行います。

点滅時間を変える。

プログラムの最後の方の　delay(1000);　のカッコ内の数字を変えてみます。
1が1/1000秒をあらわすので今が1秒です。これを500にしてみます。
delay(1000);　は2ヶ所ありますが、最初が点灯している時間、後の方が消灯している時間を指定しています。

変更したら、「マイコンボードに書き込む」ボタンを押して、書き込みます。
点滅が早くなったと思います。色々値を変えることで点滅の早さや点灯と消灯の比率が変わることを試してみましょう。

プログラムの内容を調べてみる

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {   ← 繰り返しのはじまり
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
  }   ← 繰り返しのおわり
}
```

プログラムのdelay()はプログラムを一時停止しなさいという命令です。カッコの中は時間を1/1000秒単位で入れます。
その前のdigitalWrite()はledを点灯したり、消灯したりする命令です。HIGHはledを点けて、LOWはledを消します。つまりHIGHのときは電気ＯＮ、LOWのときは電気ＯＦＦです。

このプログラムは、digitalWrite(led,HIGH); でledを点け、そのまま delay(1000); で1秒待った後、 digitalWrite(led,LOW); でledを消し、delay(1000); で1秒待ちます。各命令の最後には行末を示す ; (セミコロン) を必ず付けます。
そして void Loop()のあとの { } で囲まれた内容は繰り返される約束なので、プログラムは始まりにもどって、ledを点灯させます。

出力先を変えて、色々な機器を試す。

その他のＬＥＤ表示器やブザーも動かしてみます。これらのマイコンによって動かされる機器を出力機器と言います。動かす機器を変えることを出力先を変えると言います。

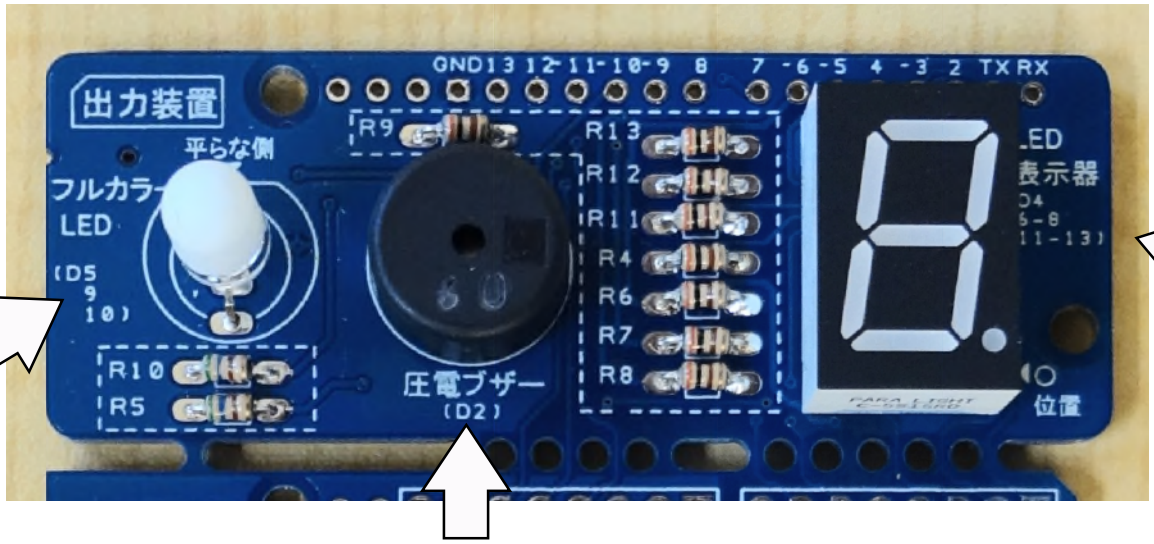
プログラムの最初の方に、

int led = 13;

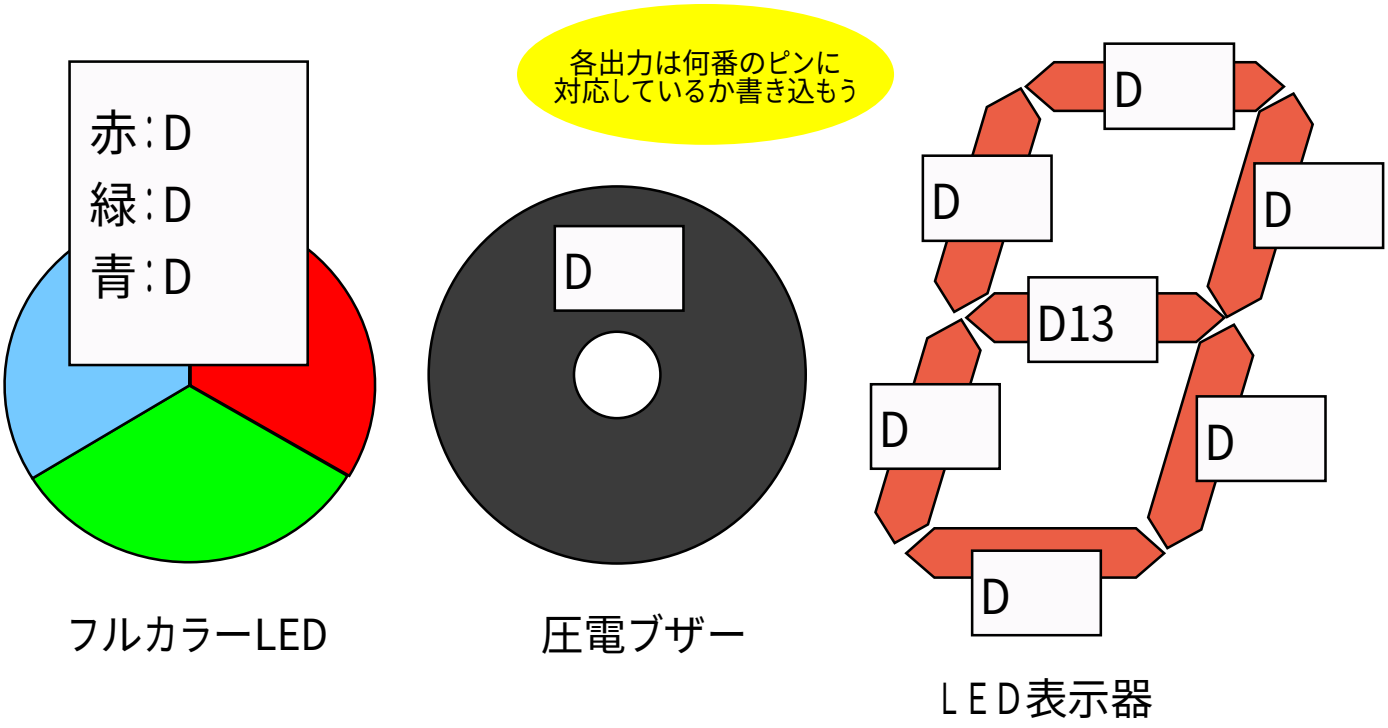
という部分があります。これはこのプログラムの中で led というのは13のことだと宣言しています。そしてこの13が出力先の番号です。このボードでは13番が現在点滅している2つのＬＥＤに割り付けられています。
ためしに13を12に書き換えて再度ボードに書き込んでみてください。
別の場所のＬＥＤが光り、出力先が変わったことがわかります。

出力を確認する

このボードにどのような出力があるのかを調べます。
ボードには出力先の番号が、D**で表示されています。



Dを除いた**の部分の数字に書き換えてみて、どの部分が動くか確認して
下の表に書き込んで整理しましょう。



圧電ブザーはＬＥＤではないのですが、プログラム中での名前の宣言だけなので、ここで
はそのまま使用します。ブザーはチッチッという音しかしないと思いますが、それで
正常です。
7セグ表示器の番号が飛んでいる理由は、D5、9、10の出力に特別な機能があって、そ
れをフルカラーＬＥＤに使いたいからです。これは後で試します。

もしまったく反応しない出力がある時は、プログラムに間違いがないか、ボードの出力機器、マイコン、抵抗のはんだ付けがちゃんとされているか、抵抗の値などに間違いが無いか確認してください。

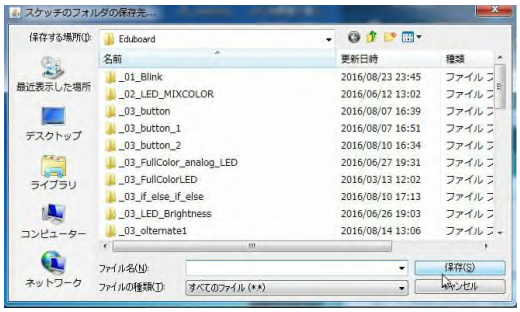
スケッチの保存

サンプルスケッチは書き込み禁止になっているので、保存する場合は別の名前をつけます。

ファイル>>名前をつけて保存　を選びます。

フォルダは自動的に作成されるのでそのままファイル名をつけます。

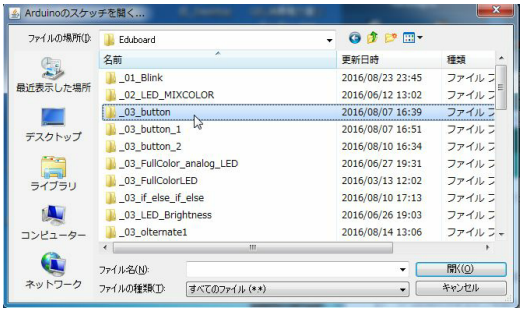
使えるのは半角英数字です。 1文字目は数字以外でないといけません。 長さは最大64文字以内にします。



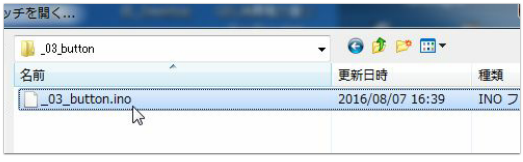
保存したスケッチの読み込み

ファイル>>開く　を選びます。

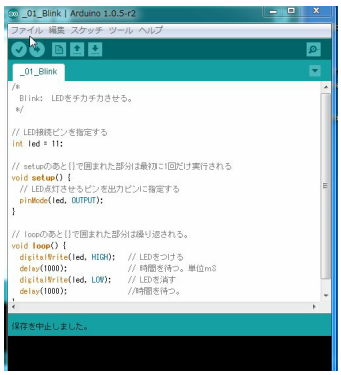
フォルダー一覧が開きます。



目的のフォルダを開き、その中の***.ino ファイルを指定して開きます。



スケッチ読み込み完了



Vol.1　動作チェックを行いながら、プログラム操作に慣れよう。

今回のまとめ

1）　サンプルスケッチを開いて、マイコンボードに書き込むことをしました。

2）　**delay()**という命令を使いました。

3）　フルカラーＬＥＤ、圧電ブザー、ＬＥＤ表示器がどのピンに対応しているか確かめました。

4）　スケッチの保存と読み込む方法を試しました。

新しく出てきた命令

delay(時間);

プログラムの動きを指定された時間、停止します。
() の中には時間を1/1000秒単位で入力します。
(例)　0.1秒　→　delay(100)
　　0.5秒　→　delay(500)
　　1秒　→　delay(1000)

digitalWrite(出力先, HIGH);　digitalWrite(出力先, LOW);

出力をオンオフする命令です。
出力先のところには、ピン番号や変数を指定します。

HIGHで出力オン、LOWで出力をオフします。

Vol.2　ＬＥＤイルミネーションのプログラミング

プログラミングのスタート

前回作ったプログラムを利用します。進めながらプログラムの細かい部分も説明していきます。
最初に、これが前回使ったプログラムです。

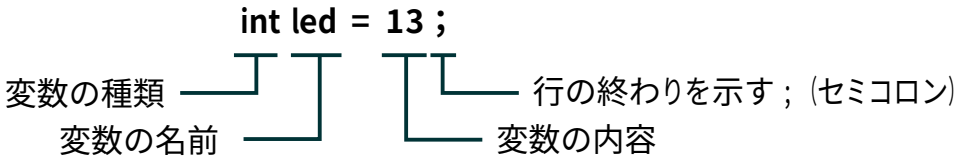
①

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

ＬＥＤの追加 ・ 変数の宣言部分

今まではひとつずつしかＬＥＤを点灯させていませんでしたが、複数のＬＥＤを点けられるようにします。①の部分を見てください。

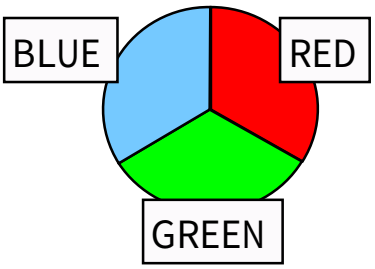
led の部分は変数と言われ、プログラムの中で使用される値や文字の入れ物です。最初に書かれた int の部分は、変数の種類をあらわします。 int は整数用の変数であること示します。



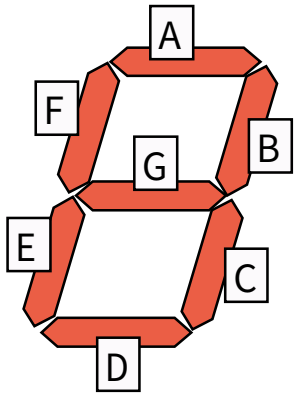
led は整数をあらわす変数で、その値は13」 となります。
変数を使うことでプログラムの内容を見やすくしたり、内容を変えるときの手間を減らすことができます。

これから変数を追加していきます。

最初に変数の名前を決めておきます。
変数の名前は自由に決めて良いのですが、漢字や平仮名などの全角文字は使えません。わかりやすいように、今回は下のようにします。



フルカラーLED



ＬＥＤ表示器

変数の名前

それではプログラミング開始です。
ためしにＬＥＤを３つ点滅させてみます。使用するのはLED表示器のA,G,Dの横棒３ヶ所です。
前回調べた出力先と変数をむすびつけていきます。

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int A = 4;
int G = 13;
int D = 7;
```

セットアップ部分

続いてその下の　void　setup(){　}　の部分を変更します。
　setupにはプログラム開始時に1回だけ行うことを記述します。
今回は指定する３つをデジタル出力先に指定するという命令を書きます。

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int A = 4;
int G = 13;
int D = 7;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

変更前

②

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int A = 4;
int G = 13;
int D = 7;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(A, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(D, OUTPUT);
}
```

変更後

各行の終わりには命令の区切り記号 ; (セミコロン) と、
setup(){ の終わりを示す } (かっこ) を忘れないようにしてください。

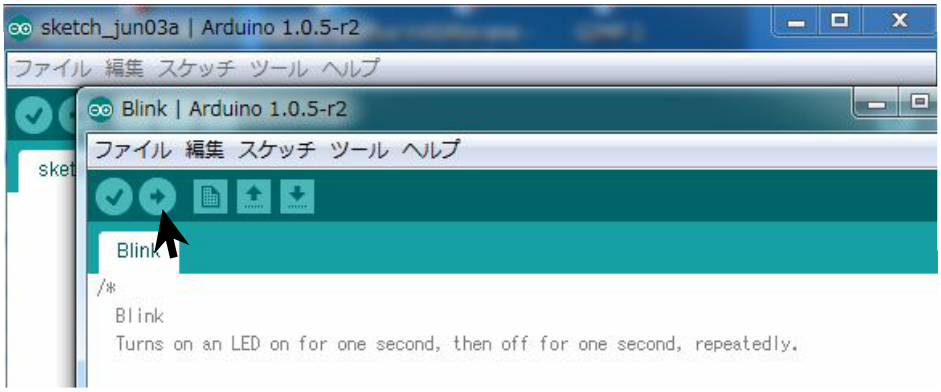
プログラム本体の部分

void loop(){ } で囲まれた部分は、繰り返し実行される処理です。ここにプログラム本体部分を書いていきます。今回は、各ＬＥＤを順次点滅させるプログラムを書きます。

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int A = 4;
int G = 13;
int D = 7;
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(A, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(D, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(A, HIGH);
  delay(200);
  digitalWrite(A, LOW);
  digitalWrite(G, HIGH);
  delay(200);
  digitalWrite(G, LOW);
  digitalWrite(D, HIGH);
  delay(200);
  digitalWrite(D, LOW);
}
```

追加部分

プログラムが書けたら、マイコンボードに書き込み動作を確認します。

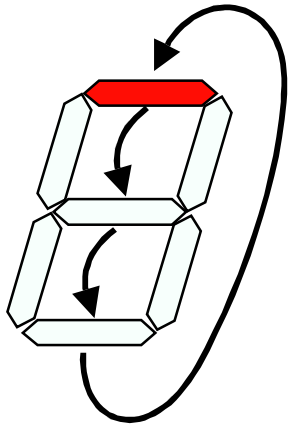


エラーが出た場合は、変数の指定や綴りにまちがいが無いか、
かっこやセミコロンの位置や抜けていたり多かったりしないかチェックします。

コンパイルが終了したあと書き込み時にエラーが出る場合はケーブルの接続や、
ツール>>シリアルポートで接続先の確認をします。

ＬＥＤ表示器のＬＥＤが上下に流れるように点滅したらプログラムは成功です。

時間があれば、時間を変えてみたり、
外側の6個のＬＥＤを使って
光がぐるぐる回るプログラミングをしてみましょう。



おまけですが、プログラムを見やすくする方法の紹介です。
行の先頭に　//　がつくと、その行はプログラムとしては無視されます。こうやって
プログラムの説明などを書いておくと、後になって思い出すときなどに便利です。

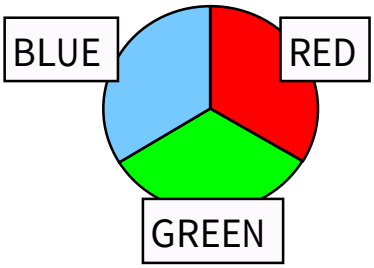
また各命令は；（セミコロン）で区切りになりますので、一行の中に複数の命令を
書くことで見やすくすることもできます。

```
// the loop routine runs over and over again forever:
void loop() {
//Aを0.2秒点けて消す
  digitalWrite(A, HIGH);delay(200);digitalWrite(A, LOW);
//Gを0.2秒点けて消す
  digitalWrite(G, HIGH);delay(200);digitalWrite(G, LOW);
//Dを0.2秒点けて消す
  digitalWrite(D, HIGH);delay(200);digitalWrite(D, LOW);
}
```

ひとつのＬＥＤの動作を
同じ行にまとめました。

フルカラーＬＥＤを使う

今度はフルカラーＬＥＤをつかってみます。
変数は図のように色の名前を変数にします。



変数の宣言をします。　　変数の名前
出力先と色の関係は、自分で調べた値を使ってください。

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int RED= 9;
int GREEN = 5;
int BLUE = 10;
```

void　setup(){　}　で出力先の指定をします。

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
}
```

点滅の動作を書き込みます。
今回は、
① 1色ずつ点灯
② 2色を同時に点灯
③ 3色全部を同時に点灯
というのをやってみます。

色を混ぜることで、多くの色をあらわしてみます。

フルカラーＬＥＤ点滅プログラム

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int RED= 9;
int GREEN = 5;
int BLUE =10;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
//赤を1秒点けて消す
  digitalWrite(RED, HIGH);delay(1000);digitalWrite(RED, LOW);
//緑を1秒点けて消す
  digitalWrite(GREEN, HIGH);delay(1000);digitalWrite(GREEN, LOW);
//青を1秒点けて消す
  digitalWrite(BLUE, HIGH);delay(1000);digitalWrite(BLUE, LOW);
//赤と青を1秒点けて消す
 digitalWrite(RED, HIGH);digitalWrite(BLUE, HIGH);
 delay(1000);
 digitalWrite(RED, LOW);digitalWrite(BLUE, LOW);
//赤と緑を1秒点けて消す
 digitalWrite(RED, HIGH);digitalWrite(GREEN, HIGH);
 delay(1000);
 digitalWrite(RED, LOW);digitalWrite(GREEN, LOW);
//緑と青を1秒点けて消す
 digitalWrite(GREEN, HIGH);digitalWrite(BLUE, HIGH);
 delay(1000);
 digitalWrite(GREEN, LOW);digitalWrite(BLUE, LOW);
//３色全部を1秒点けて消す
 digitalWrite(RED, HIGH);
 digitalWrite(GREEN, HIGH);digitalWrite(BLUE, HIGH);
 delay(1000);
 digitalWrite(RED, LOW);
 digitalWrite(GREEN, LOW);digitalWrite(BLUE, LOW);
   }
```

実際に動かしてみてどうだったでしょうか？

赤、緑、青の3色で下の図に示すような各色を出すことができたと思います。

フルカラーＬＥＤは、「光の混色の原理」を利用しています。
光は混ぜ合わされることで色が変わり、3原色を混ぜ合わすと白色になります。

http://www.biwako.shiga-u.ac.jp/sensei/mnaka/ut/sozai/color.html

もう少し高度なプログラミングで、各色の光量を調整すると、この図の中間の色も表現することが可能です。それも今後やっていきます。

今回やったような方法を組み合わせればオリジナルのＬＥＤイルミネーションを作ることができます。

Vol.2 ＬＥＤイルミネーションのプログラミング

今回のまとめ

1） 変数とはどういうものか学びました。

2） **setup()**は、開始時に実行される部分、**loop()は**繰り返し行われる部分というプログラムの構造を学びました。

3） ＬＥＤを組み合わせてイルミネーションにしました。

4） フルカラーＬＥＤの混色を試しました。

新しく出てきた命令

pinMode(指定先 **,OUTPUT);　pinMode(** 指定先 **,INPUT);**

デジタルピンは入力用にも出力用にもつかえます。
ピンの動作を入力または出力に切替えるための命令です。
指定先にはピン番号や変数を指定します。

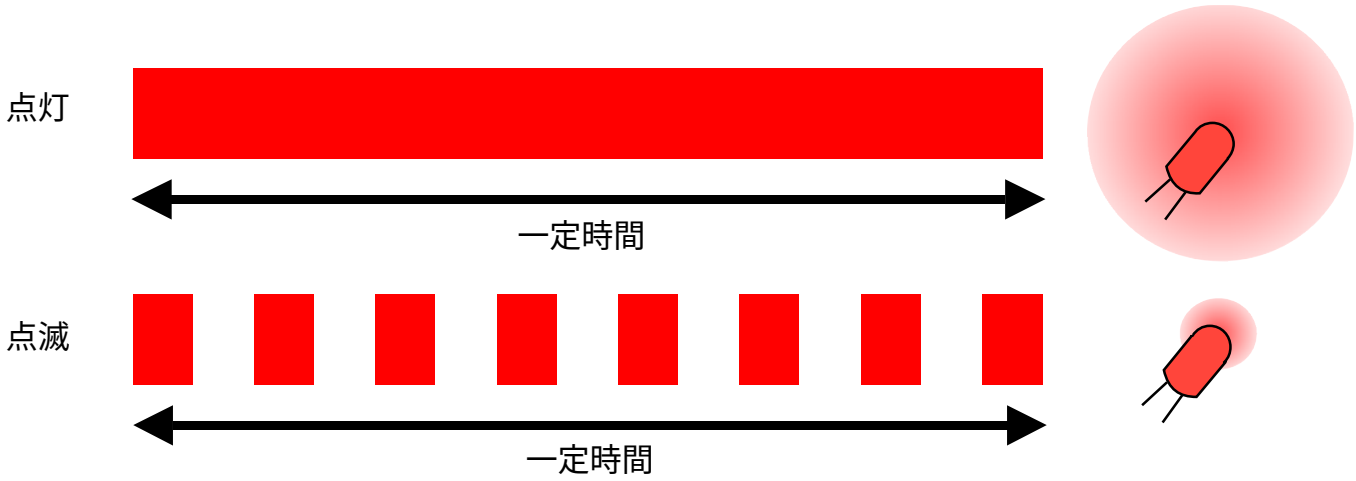
Vol.3 ＬＥＤの明るさを変える

明るさを変える機能

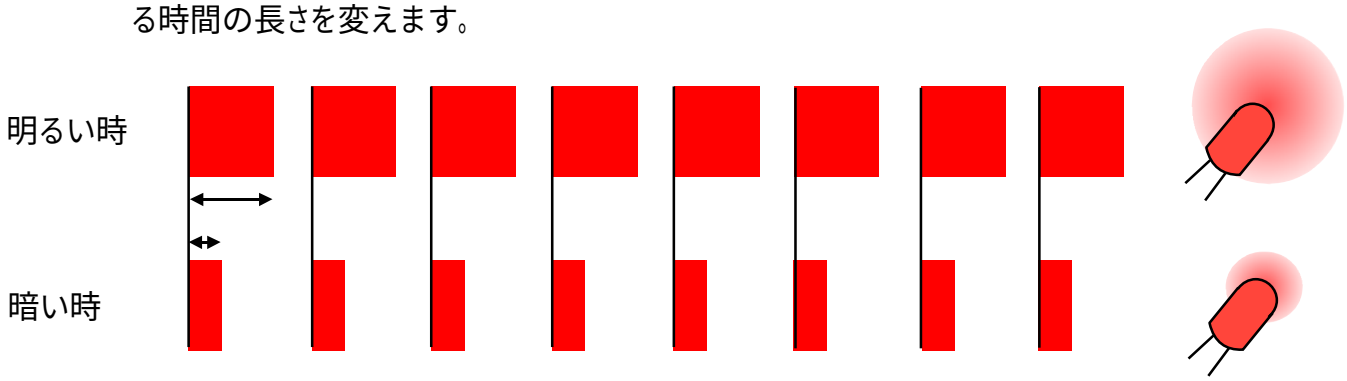
今まではＬＥＤを点けるか消すかしかできませんでしたが、今回はＬＥＤの明るさを変える機能を試します。

図を見てください。
上の場合はＬＥＤを点灯したままの様子です。
下の場合は、ＬＥＤを点けたり消したり、点滅させた様子です。消えている時間があることで、一定時間に発する光の量は点灯している時の半分になります。

この時に、点滅の周期が早ければ、人間の目には点滅しているのがわからず、ＬＥＤが暗くなったように見えます。このことを利用してＬＥＤの明るさを変化させます



実際のマイコンボードは点滅の周期は一定で、点灯している時間と消灯している時間の長さを変えます。



このようにオンオフを繰り返して出力を制御することを　PWM制御と言います。
引き続きプログラミングの方法を説明します。

プログラミングの方法

マイコンボードには、あらかじめPWM制御用の出力が用意されています。
番号で言うと、3、5、6、9、10、11の6つです。ただし3番は予備の出力
なので現在は何もつながれていません。

PWM出力を使うには

analogWrite(出力先,値)　という命令を使います。

値の部分は0～255の数字になります。0が消えた状態、255が点灯した状態で
、その間は値に比例して明るさが変わります。

今までに作ったプログラムを改造して明るさを変化させてみます。

```
int led = 10;

// the setup routine runs once when you press reset:
void setup() {
  // ledピンを出力に設定する
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  analogWrite(led, 50);   // LEDをつける。明るさ50
  delay(500);             // 0.5秒待つ
  analogWrite(led, 125);    // 明るさを125に変える
  delay(500);             // 0.5秒待つ
  analogWrite(led, 255);    // 明るさを255に変える
  delay(500);             // 0.5秒待つ
}
```

LEDの明るさが3段階に変わったと思います。

またフルカラーLEDの各色の明るさの比率を変えることで、今までの中間
の色を表現することもできます。

フルカラー表示

```
int RED = 9;
int GREEN = 10;
int BLUE = 5;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  analogWrite(RED, 125);
  analogWrite(GREEN, 50);
  analogWrite(BLUE, 125);
  delay(500);               // 0.5秒待つ
  analogWrite(RED, 50);
  analogWrite(GREEN, 125);
  analogWrite(BLUE, 50);
  delay(500);               // 0.5秒待つ
  analogWrite(RED, 0);
  analogWrite(GREEN, 0);
  analogWrite(BLUE, 0);
  delay(500);               // 0.5秒待つ
}
```

今まで表現できなかった中間色も表示できるようになりました。

Vol.3 ＬＥＤの明るさを変える

今回のまとめ

1） **analogWrite()** という　オンとオフの中間の出力をする命令を学びました。
そのしくみは、PWM出力という早い周期でオンオフさせるものでした。

2） **analogWrite()** を利用することで、ＬＥＤの中間色を出せるようになりました。

新しく出てきた命令

analogWrite(出力先 **,** 値 **);**

アナログ出力を行います。
出力先のところには、ピン番号や変数を指定します。
値のところには、0から255までの値を指定します。

analogWrite()を使用できるのは、3,5,6,9,10,11のピンだけです。

Vol.4 入力を使う

入力について

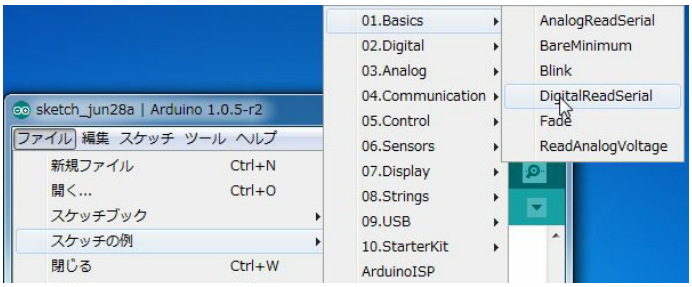
多くの機器は入力の内容によって出力の様子を変えるということを行っています。例えばエアコンは、室温が上がれば冷風を送り、温度が下がれば冷風を弱めます。

このマイコンボードにもいくつかの入力がついているので、それらを使ったプログラミングを試していきます。

ボードの下のほうに、ボタン、明るさセンサー、ボリュームの3つの入力がついています。

デジタル入力

ボタンはデジタル入力です。デジタルのときは、オンかオフの2つの値しか示しません。
最初にサンプルスケッチを開きます。
ファイル>>スケッチの例>>01. Basics>>DigitalReadSerial を選びます。



サンプルが読み込まれたら、 int pushButton = 2; の2の部分を

このマイコンボード用の16に直し、書き込みます。

プログラム全体と変更する部分を次のページに示します。

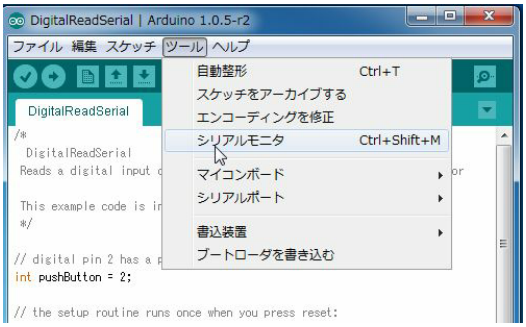
```
/*
  DigitalReadSerial
 Reads a digital input on pin 2, prints the result to the serial monitor

 This example code is in the public domain.
 */

// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

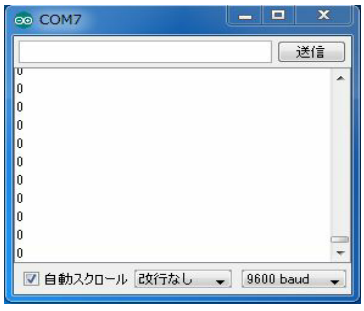
⇦ 16に変更する部分

⇦ (参考) シリアル通信開始命令

⇦ (参考) 変数の内容を送る命令

書き込みが完了したら　ツール>>シリアルモニタ　を選びます。



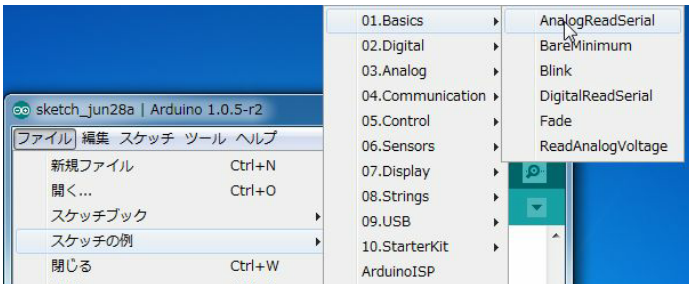


別ウィンドウが開き、数字の0が連続して表示されるかと思います。
もし数字が表示されないようなら、右下のボックスが「9600baud」になっている
か確認してください。

今はボタンの状態をパソコンでモニターしています。**ボタンを押すと数字が1に変わります。**ボタンからの入力は、**押されたとき1、離したとき0** を示します。
プログラミングコードでは、1を**HIGH**、0を**LOW**としています。

アナログ入力

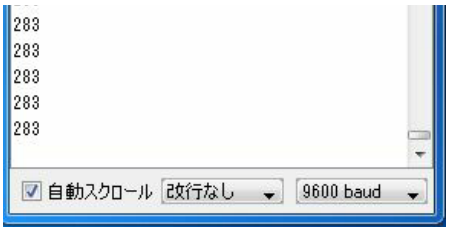
続いてアナログ入力を試します。
アナログ入力モニタのスケッチを開きます。
　ファイル>>スケッチの例>>01. Basics>>AnalogReadSerial　を選びます。

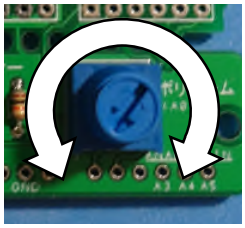



```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins
to +5V and ground.

 This example code is in the public domain.
 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

サンプルが読み込まれたら、 そのままマイコンボードに書き込みます。
書き込みが完了したら　デジタル入力と同様に
　ツール>>シリアルモニタ
を選びます。 さきほどと同様にシリアルモニタウィンドウが開きます。

数字が表示されたら、ボリュームのつまみを回してみてください。つまみの位置に合わせて数字が変化する様子がわかります。
数字は一番左に回したとき0、右に回したとき1023となります。
このマイコンボードは、ボリュームの位置を0から1023までの1024段階で表していることになります。





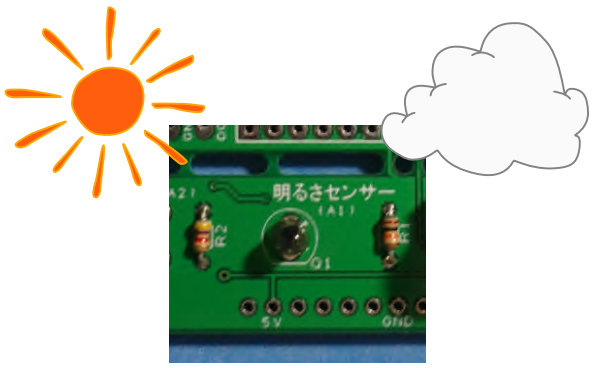
となりにある明るさセンサーも試してみます。
int sensorValue = analogRead(A0);　の　**A0**　を**A1**　に直してマイコンボードに書き込みます。

```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A1);
```

変数　sensorValueの指定を
明るさセンサー　A1　に変更

これで、アナログ入力0につながれているボリュームからアナログ入力1につながれたセンサーに変更できました。

マイコンボードに書き込んで、シリアルモニターを開始したら、明るさセンサーを窓に向けたり、手をかざすなどして、センサーに入る光量を変えてみてください。
明るさに応じてセンサーの値が変わります。



デジタル入力とアナログ入力の使い分け

同じ入力機器でも、ボタンのようなスイッチ類は電圧の掛かっているオンの状態と電圧の掛かっていないオフの状態の2つしかありませんので、デジタル入力で十分です。

ボリュームや明るさセンサーはその位置や明るさを電圧の大小で示しますので、その値をアナログ入力で段階的に読み取ることで、より精密な制御を行うことができるようになります。

Vol.4 入力を使う

今回のまとめ

1） デジタル入力（押しボタン）を試す。
デジタル入力は0または1の値になることを確認する。

2） アナログ入力（ボリューム、光センサー）を試す。
アナログ入力はボリュームやセンサーの状態を0から1023の間の値で示すことを確認する。

3） 入力の状態をパソコンのシリアルモニタで確認する。

新しく出てきた命令

digitalRead(入力元);

指定された入力元の値を読取ります。
ピンに5V掛かっていれば**HIGH**を、0Vであれば**LOW**を返します

analogRead(入力元);

指定された入力元に掛かる電圧を読取ります。
ピンに掛かった0～5Vの電圧を、0～1023の値に変換します。

analogRead()を使用できるのは、A0からA7までのピンです。

以下は、シリアル通信という方法で外部の機器と通信する場合に使用する命令です。
参考に載せておきます。

Serial.begin(通信速度);

シリアル通信を始めるための命令です。通信速度は相手側の機器と同じ値に設定しておく必要があります。

Serial.println(メッセージ);

メッセージを送る命令(改行付) です。送信可能なメッセージは半角英数字です。文字列の場合は、" "で囲みます。

例) Serial.println(1); //数字
Serial.println(BUTTON); //変数を指定
Serial.println("NO.1 ok!"); //文字列

Vol.5 入力によって出力を操作する　1

今までは、個別に入力と出力を確かめてきました。
これから、入力によって出力を操作するというプログラミングを始めましょう。
最初は、ボタンを押している間だけＬＥＤを点灯させるプログラムを作ってみます。
まず、その時に使う命令をひとつ覚えましょう。

If文　(イフぶん)

「if文」はもっとも多く使われる命令です。
英語の「if」は、「もし何々なら」という意味ですが、プログラミングのときも
同じように使われます。
実際にプログラムの中で使うときは、下のように書きます。

```
if(digitalRead(BUTTON) == HIGH) {
  digitalWrite(LED,HIGH);
}
```

Ifのあとの () の中には条件を書きます。そのあとの { } の中には
() 内の条件に合致したときに行うことを書きます。

例文は、ボタンの値を読み、その値がHIGHだったら、ＬＥＤをつけるという
プログラムです。
このときに注意しなければならないことをひとつ、
条件を判定する () 内の部分で、条件に合致したらばという場合は「=」を2つ
重ねて、「==」と記述します。
プログラムでは　a =3;　というように、「＝」1つの場合、変数aに3を代入しなさいと言う意味で使われるので、それと区別するためです。
実際に使われる条件式をメモしておきます。

xと**y**が同じ場合、　x == y

xと**y**が異なる場合、**x != Y**

大小の比較を行う

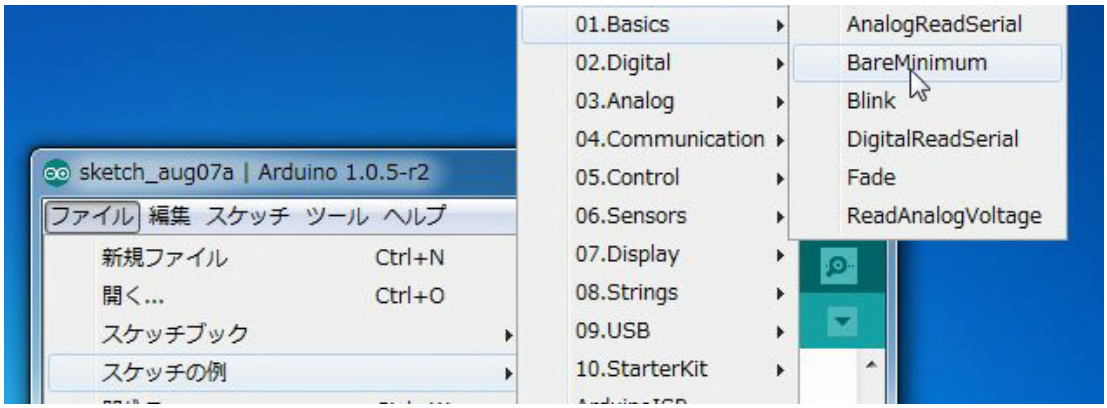
xが**y**より大きい場合、　**x > y**

Xが**y**より小さい場合、　**x ＜ y**

xが**y**と同じか大きい場合、　**x ＞= y**

xが**y**と同じか小さい場合、　**x ＜= y**

プログラム作りを開始します。
ファイル>>スケッチの例>>01.Basics>>BareMinimum　を開きます。



BareMinimumはプログラムの雛形です。**setup()とloop()**が用意されているので、これに書き加えていきます。

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

次のページに、最初に**if**文を試すプログラムを載せています。

コメントはプログラムの動作に影響しないので無くても構わないのですが、後でプログラムを見直す時などに役立ちますので、できるだけ入れるようにしましょう。

loop()や**if**文の{　}は必ず始まりと終わりに必要です。かっこが多すぎても少なすぎてもうまく動きません。かっこの数をまちがわないように注意します。

```
/*
  D9   :赤LED出力
  D16  :ボタン入力
 */

int  BUTTON = 16;
int LED = 9;

void setup() {
  // 使用するピンの指定
  pinMode(LED, OUTPUT);
  pinMode(BUTTON,INPUT);
}


void loop() {
 if(digitalRead(BUTTON)==HIGH){
 digitalWrite(LED,HIGH);
 }
}
```

/*から　*/で囲まれた部分は
コメント

//で始まる行もコメント。

loop()の始まりの {

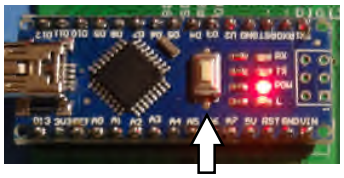
条件がそろったとき行うことの始まりの {

条件がそろったとき行うことの終わりの }

loop()の終わりの }

プログラムができたら、スケッチをアップロードします。

動作を確認してみてください。
どうでしょうか？ボタンを押したら赤ＬＥＤは点きますが、ボタンを離しても
点いたままです。



リセットボタン

最初からもう一度試す場合はＵＳＢケーブルを一旦抜いて電源を
切るか、マイコンボード上のリセットボタンを押します。

このプログラムは、ＬＥＤを点ける条件は書かれていますが、消す条件は書か
れていないのが問題です。
次に説明する　if～else～文を使って消す条件を付け加えましょう。

If～　else～ 文　(イフ～　エルス～　ぶん)

if～ else～ 文はifの条件に合えば、そこに示された動作を行い、ifの条件に合わなければ、
elseのあとに示される動作をしなさい。という命令です。

今回のプログラムなら、

if　　もしボタンが押されていれば　　→　ＬＥＤを点ける

else　そうでなければ（＝ボタンを押していなければ）　→　ＬＥＤを消しなさい　となります。

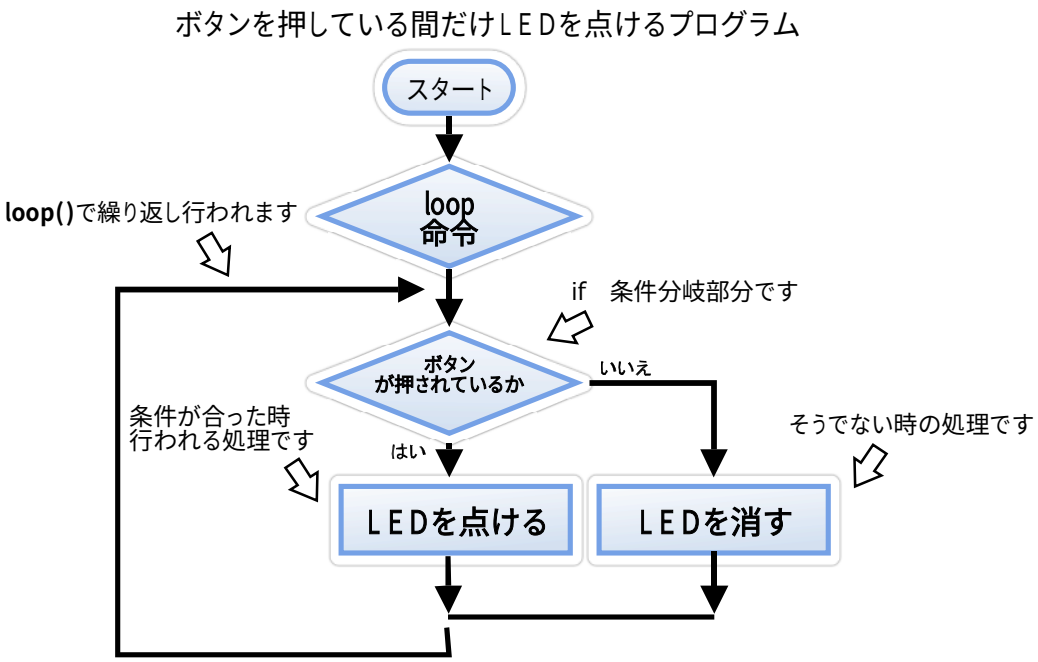
else～　を追加したプログラムです。

```
/*
  D9   :赤LED出力
  D16  :ボタン入力
 */

int  BUTTON = 16;
int LED = 9;

void setup() {
  // 使用するピンの指定
  pinMode(LED, OUTPUT);
  pinMode(BUTTON,INPUT);
}


void loop() {
  if(digitalRead(BUTTON)==HIGH){
  digitalWrite(LED,HIGH);
  }
else {
   digitalWrite(LED,LOW);
  }
}
```

⇦ else の処理を追加します。

スケッチをアップロードして、動作を確認してみましょう。
今度は押している間だけＬＥＤが点灯したと思います。

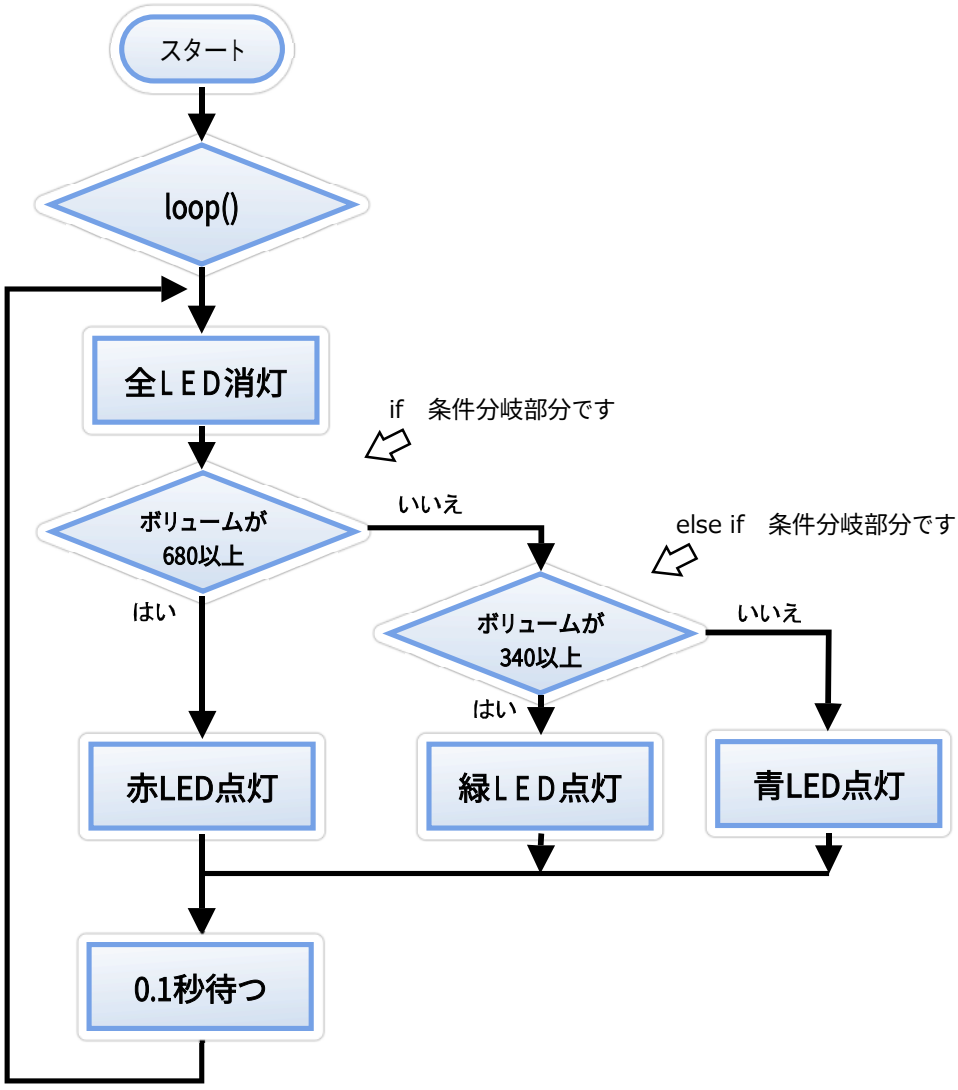
プログラムの構成を考えるときに、フローチャートを作りながら考えると
わかりやすくなります。
先ほどのプログラムのフローチャートは次のようになります。



if～ else～ 文は **if** の条件とそうでないときの2パターンの条件への分岐処理でしたが、もっと多くの分岐を行うために **if** 条件を追加した　if～ else if～ else～文というものもあります。

If～　else if～ else～文　(イフ～　エルスイフ～　エルス～　ぶん)

if～ else if～ else～　を使ったプログラム例のフローチャート



ボリュームの値によってＬＥＤの色を変えるプログラムです。

ボリュームの値によってＬＥＤの色を変える

```
//変数の準備
int LED_R = 9;
int LED_G = 10;
int LED_B = 5;
int value = 0;  //ボリュームの値を入れるところ

void setup() {
//ピン設定
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);
pinMode(11,OUTPUT);

}

void loop() {
  //全LED消灯
  digitalWrite(LED_R,LOW);
  digitalWrite(LED_G,LOW);
  digitalWrite(LED_B,LOW);

  //ボリュームの値を読み込む
  value = analogRead(A0);

  //条件分岐
  if(value > 680){
    digitalWrite(LED_R,HIGH);
  }else if(value > 340){
    digitalWrite(LED_G,HIGH);
  }else{
    digitalWrite(LED_B,HIGH);
  }

  //0.1秒待機
  delay(100);

}
```

参　考

if～ の条件は、2つの組み合わせで判定することもできます。

「aが0で、bが0の両方の条件が成り立つ場合」というのは

if(a==0 && b==0)　　と書きます。

「aが0、bが0のどちらかの条件が成り立つ場合」というのは

if(a==0 || b==0)　　と書きます。

Vol.5 入力によって出力を操作する　1

今回のまとめ

1)　ボタン（入力）の状態を　if～　else～　文　で　判定、処理する。
ボタンの状態に合わせて、ＬＥＤ（出力）を点けたり、消したりした。

2)　if～　else　if～　else～　文を使って、判定を増やす。
ボリュームの値によって３色LEDのうちの1個を点灯させた。

新しく出てきた命令

if(条件){　条件が成立した時に行うこと　**}**

（ ）内の条件が成立したときに、あとの{ }内の事を行う。
条件が成立しなかったら、何もしない。

if(条件 **){**　条件が成立した時に行うこと
}else {　条件が成立しなかった時に行うこと**}**

（ ）内の条件が成立したときに、あとの{ }内の事を行う。
条件が成立しなかったら、**else** のあとの{ } 内のことを行う。

if(条件1 **){**　条件１が成立した時に行うこと
}else if(条件２**){**条件1は成立しないが条件2が成立した時に行うこと
}else {　どの条件も成立しなかった時に行うこと　**}**

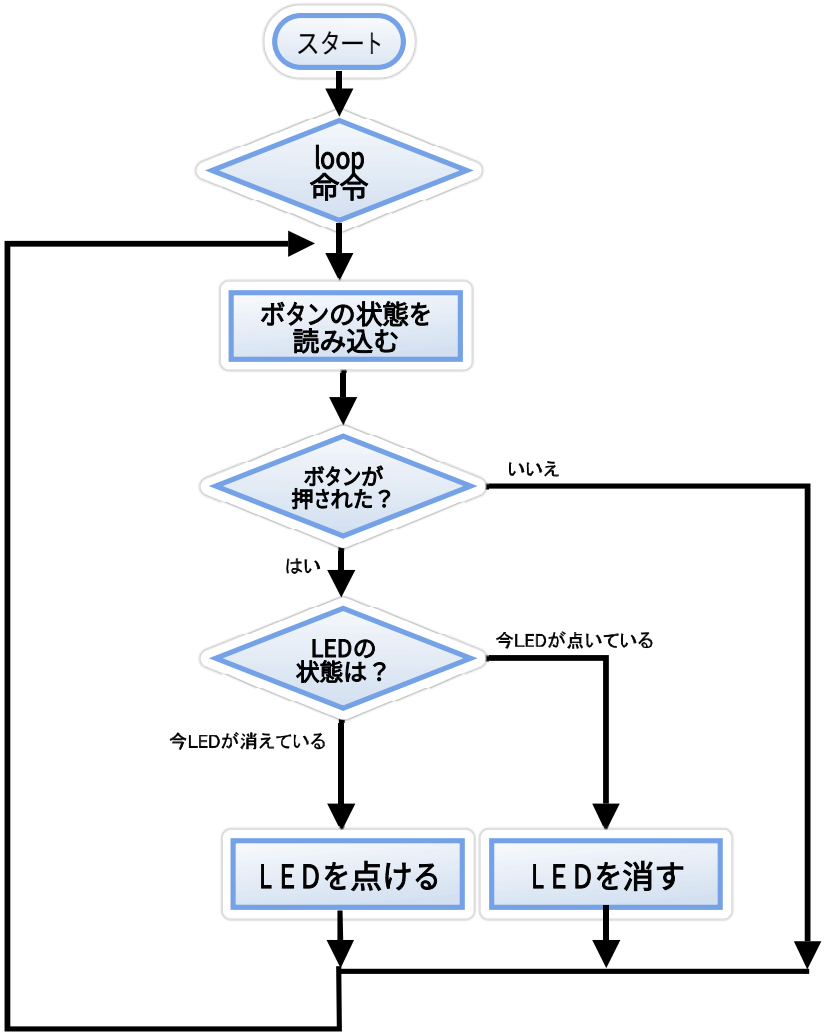
順番に条件を判定して、成立したところの { } 内の事を行う。
条件が成立しなかったら、**else** のあとの{ } 内のことを行う。

Vol.6 入力によって出力を操作する　2

前回のプログラムはボタンを押している間だけＬＥＤを点灯させるものでしたが、今回はボタンを押すとＬＥＤが点き、もう一度ボタンを押すとＬＥＤが消えるというプログラムを作ります。

プログラムの構成を考えます。

ボタンを押すと点灯、もう一度押すと消灯するプログラム



作成したフローチャートを元に、プログラムを作成します。
最初に作ってみたプログラムを、次のページに示します。

今回のプログラミングでは、「今、ＬＥＤが点いていたら消す。」「今、ＬＥＤが消えていたら点ける」というように、現在のＬＥＤの状態を記憶しておくことが必要になります。
例文では、boolean　nowLED　という変数を用意してここに現在のLEDの状態を入れています。boolean　は　変数の型で、真を表す　**true**か　偽を表す　**false**が入ります。

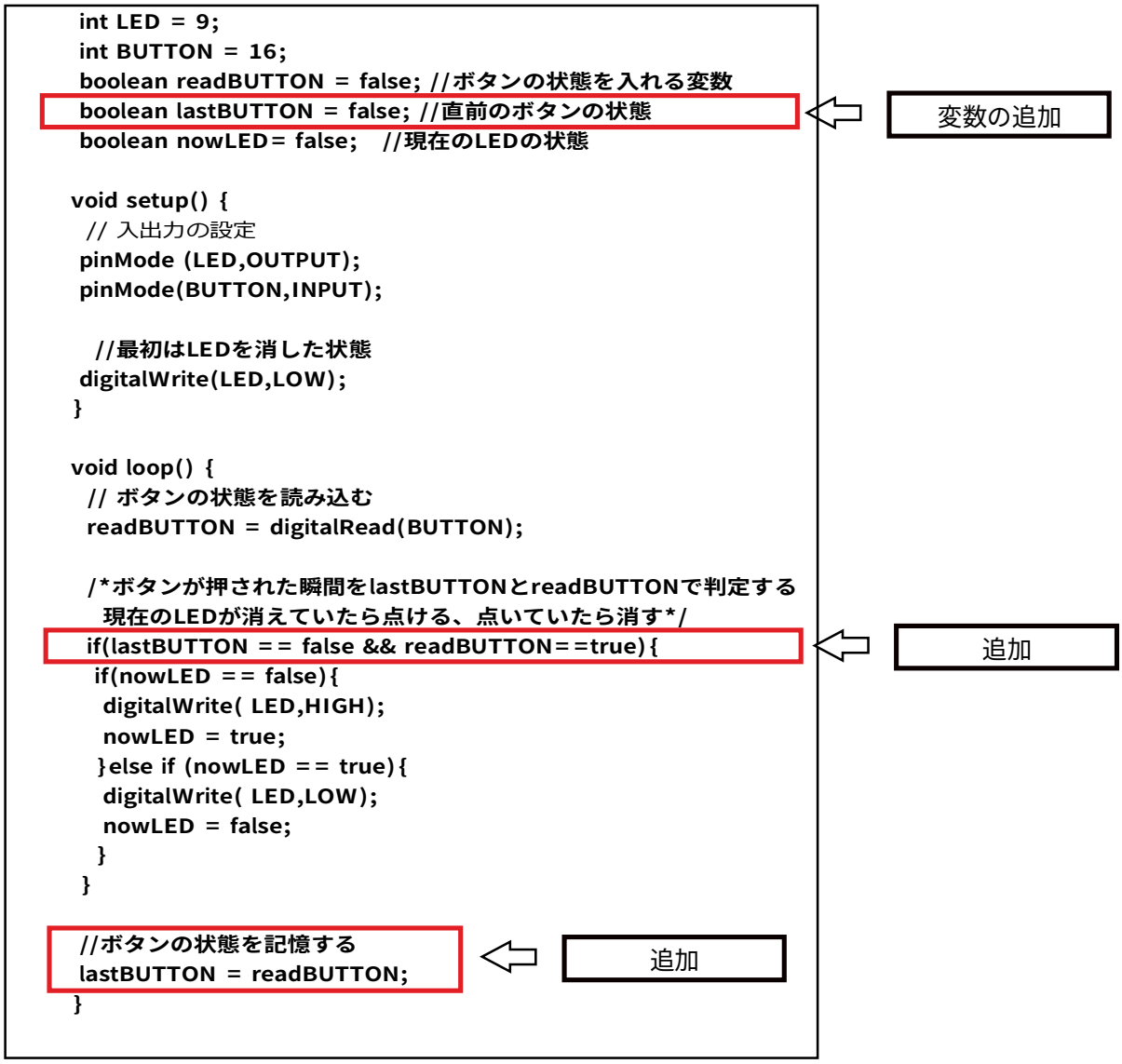
```
int LED = 9;
int BUTTON = 16;
boolean readBUTTON = false;  //ボタンの状態を入れる変数
boolean nowLED= false;       //現在のLEDの状態

void setup() {
 // 入出力の設定
pinMode (LED,OUTPUT);
pinMode(BUTTON,INPUT);

//最初はLEDを消した状態
digitalWrite(LED,LOW);
}

void loop() {
 // ボタンの状態を読み込む
 readBUTTON = digitalRead(BUTTON);

 //現在のLEDが消えていたら点ける、点いていたら消す
 if(readBUTTON==true){
  if(nowLED == false){
   digitalWrite( LED,HIGH);
   nowLED = true;
  }else if (nowLED == true){
   digitalWrite( LED,LOW);
   nowLED = false;
  }
 }
}
```

このプログラムを動作させてみたらどうだったでしょうか。
残念ながら、ボタンを押しても思ったように動かなかったと思います。

どうして動かなかったかについて、loop()の部分は繰り返し行われると前に述べましたが、実際に1秒間に何十万回も繰り返されます。そのためボタンを押して指を離すまでの間、ＬＥＤは点いたり消えたりして、最後に離した瞬間の状態で終わるため、思ったような動きにならないわけです。

そこで、ボタンを押した瞬間だけ動作させて、押し続けている間は動作させないプログラムに改造します。

ボタンが押された時をどうやって知るかですが、ボタンが押された時というのは、「一つ前に見たときがオフで、今見たときがオン」ということです。
ですので、一つ前のボタンの状態を記憶して、それと現在のボタンの状態を比べるようにします。
改造したプログラムを次のページに示します。



追加した部分は枠で囲んであります。
直前のボタンの状態を入れる変数を用意します。
ボタンが押された判定部分を変更します。
最後の部分で、現在のボタンの状態を記録します。

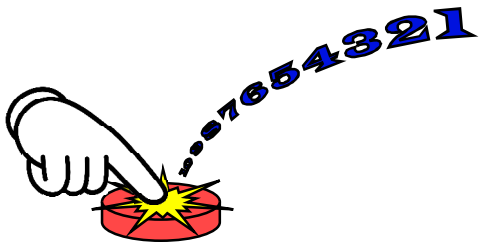
改造後の動作はどうでしょう？
おおむね考えた通りに動くようになったかと思います。

しかし、ボタンをたたくように押したときなど、うまく切り替わらない場合があるかもしれません。
これはチャタリングとかバウンシングと呼ばれる現象が起こって、ボタンの接点が細かくオンオフするため正しく切り替わっていないのです。
このような時の対策はいくつかあるのですが、今回はスイッチの動きが落ち着くまでの安定時間を加えることにします。
安定時間は10mS～100mS程度のことが多いですが、実際に試しながら決定することが必要です。

安定時間を加えました。
短いと効果が無く、長すぎると早押ししたときのボタンの反応が悪くなります。

```
int LED = 9;
int BUTTON = 16;
boolean readBUTTON = false; //ボタンの状態を入れる変数
boolean lastBUTTON = false; //直前のボタンの状態
boolean nowLED= false;   //現在のLEDの状態

void setup() {
 // 入出力の設定
pinMode (LED,OUTPUT);
pinMode(BUTTON,INPUT);

  //最初はLEDを消した状態
digitalWrite(LED,LOW);
}

void loop() {
 // ボタンの状態を読み込む
 readBUTTON = digitalRead(BUTTON);

 /*ボタンが押された瞬間をlastBUTTONとreadBUTTONで判定する
   現在のLEDが消えていたら点ける、点いていたら消す*/
 if(lastBUTTON == false && readBUTTON==true){
  delay(100);
  if(nowLED == false){
   digitalWrite( LED,HIGH);
   nowLED = true;
  }else if (nowLED == true){
   digitalWrite( LED,LOW);
   nowLED = false;
  }
 }

 //ボタンの状態を記憶する
 lastBUTTON = readBUTTON;
}
```

追加

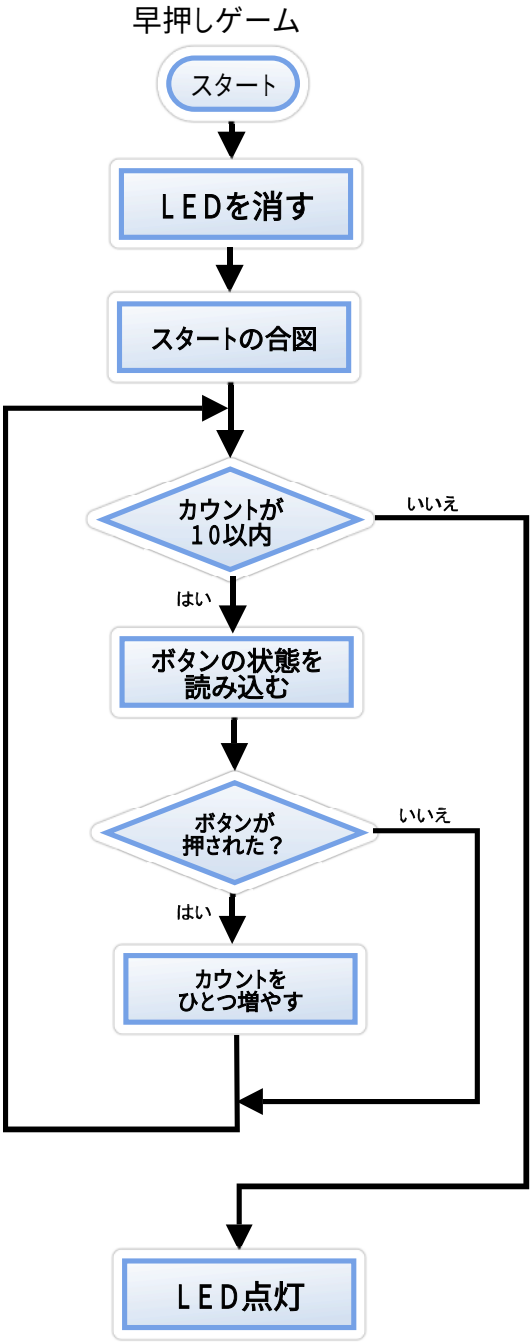


早押しゲームの製作

スイッチ入力を色々試したところで、早押しゲームを製作してみましょう。
ゲームの内容は、「ボタンを10回押すとＬＥＤが点灯する」というものです。

いつものようにフローチャートを作って考えます。

右が、早押しゲームのフローチャートです。



カウントが10以内か判断する部分と、カウントアップする部分に新しい命令を使います。

最初に「カウント１０以内」の部分ですが
while()　という命令を使います。

whileは日本語に訳すと　「～までずっと」

という意味になりますので、「カウントが１０以内の間はボタンの状態を見てカウントアップしなさい。」というプログラミングをします。

カウント数を増やす部分ですが、プログラミングの場合の　＝　は代入を表すので、

　カウント数　＝　カウント数　＋　1;

という書き方をします。右側のカウント数に1を加えて、左側の　カウント数に代入しなさいということで、結果カウント数をひとつ増やすことになります。
また上の式を簡単に書く場合、

　カウント数＋＋;

と、＋をふたつ続けて書くこともできます。

同様にひとつ減らす場合は、

　カウント数　＝　カウント数　－　1;

または

　カウント数--;

と書きます。

また今回はゲームが終了したらプログラムを止めるので、いままでのように**loop()**には書かず、setup()　の部分にプログラムを書くことにします。

そうして作ったプログラムを次のページに示します。

回数達成のときのLED、スタートの合図のＬＥＤは好きなものに変更しても
構いません。

```
int LED = 9;                //回数達成表示
int sLED = 13;              //スタートの合図
int BUTTON = 16;
boolean readBUTTON = false; //ボタンの状態を入れる変数
boolean lastBUTTON = false; //直前のボタンの状態
boolean nowLED= false;   //現在のLEDの状態

int count = 0;          //ボタンを押した回数

void setup() {
 // 入出力の設定
pinMode (LED,OUTPUT);
pinMode (sLED,OUTPUT);
pinMode(BUTTON,INPUT);

  //最初はLEDを消した状態
digitalWrite(LED,LOW);
//スタートの合図
digitalWrite(sLED,HIGH);
while(count <10)
{
   // ボタンの状態を読み込む
 readBUTTON = digitalRead(BUTTON);
 delay(20);      //安定時間
 /*ボタンが押された瞬間をlastBUTTONとreadBUTTONで判定する
   ボタンが押されていたらカウントを増やす*/
 if(lastBUTTON == false && readBUTTON==true){

   count++;        //カウント追加
 }

 lastBUTTON = readBUTTON;   //ボタンの状態を記憶する
}

//ここからカウント10の時の処理
digitalWrite(LED,HIGH);
}

void loop() {
}
```

リセットボタンを押してマイコンをリセットしたら準備完了、スタートＬＥＤの点灯
でゲーム開始です。
10回ボタンが押せたら、ＬＥＤが点灯します。

10回に到達したときの表示をもう少し派手に演出してみましょう。
新しい命令　for　を使って、LEDを点滅させてみます。

プログラムの、枠で囲んだ部分を変更します。

```
int LED = 9;                //回数達成表示
int sLED = 13;              //スタートの合図
int BUTTON = 16;
boolean readBUTTON = false; //ボタンの状態を入れる変数
boolean lastBUTTON = false; //直前のボタンの状態
boolean nowLED= false;   //現在のLEDの状態

int count = 0;          //ボタンを押した回数

void setup() {
 // 入出力の設定
pinMode (LED,OUTPUT);
pinMode (sLED,OUTPUT);
pinMode(BUTTON,INPUT);

  //最初はLEDを消した状態
digitalWrite(LED,LOW);

//スタートの合図
digitalWrite(sLED,HIGH);
while(count <10)
{
   // ボタンの状態を読み込む
 readBUTTON = digitalRead(BUTTON);
 delay(20);      //安定時間
 /*ボタンが押された瞬間をlastBUTTONとreadBUTTONで判定する
   ボタンが押されていたらカウントを増やす*/
 if(lastBUTTON == false && readBUTTON==true){

   count++;        //カウント追加
 }

 lastBUTTON = readBUTTON;   //ボタンの状態を記憶する
}

//ここからカウント10の時の処理。点滅を8回繰り返す
 for(int i=1;i<=8;i++)
 {
 digitalWrite(LED,HIGH);   //LEDを点ける
 delay(50);                //0.05秒待つ
 digitalWrite(LED,LOW);    //LEDを消す
 delay(50);                //0.05秒待つ
 }
}

void loop() {
}
```

for は、指定された回数だけくり返しなさいという命令です。

forのすぐ後のかっこ内の変数 i が回数を表します。この場合 **i** は1から始まって { } 内のプログラムを実行するごとにひとつづつ増えます。条件式は、**i**が8またはそれ以下の時は { } 内のこと行い、**i**が9になると繰り返しをやめて次の命令に進みます。

for 文を使うことで、LEDを点けて消すという命令を何回も書かなくて良くなります。

Vol.6 入力によって出力を操作する　2

今回のまとめ

1）　ボタンを押した瞬間を判定することで、ボタンを押すたびにＬＥＤを点けたり、消したりするプログラムに改造しました。

2）　チャタリングやバウンシングと呼ばれる、スイッチが細かくオンオフしても誤動作しないようにしました。

3）　条件を満たすまで繰り返す　while　命令、指定された回数繰り返す　for　命令を試しました。

新しく出てきた命令

boolean　変数名

変数の型のひとつです。
値は　**true** (真)　　または　**false** (偽)　が入ります。

&&

2つの条件式がどちらも成立する場合、真になるという記号です。
どちらかが成立したら　真になる場合は ¦¦ を使用します。

＋＋

変数の値を1づつ加算します。　例）aの値を1増やす　　**a＋＋;**

減算の場合は -- を使います。　例）　aの値を1減らす　　**a--;**

while(条件式){ 処理 }

条件が成立するまで　{　} 内の処理を繰り返し行う。

for(初期化 ; 条件式 ; 加算または減算　){ 処理 }

指定した回数、処理を行う命令です。

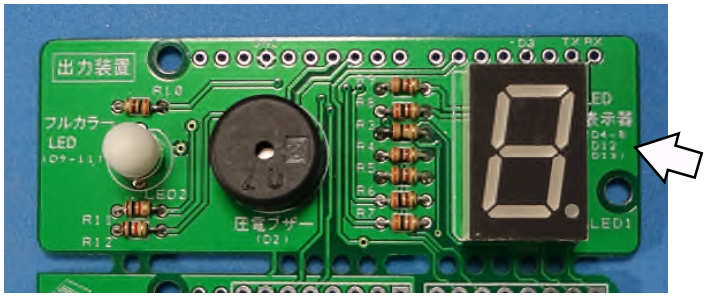
初期化の部分には、変数を初期化して、最初の値を入れます。 例） **int i = 1;**

条件式には繰り返しの条件を入れます。 例）　i <= 10;

最後は　i　の加算をいれます。 例）　i++

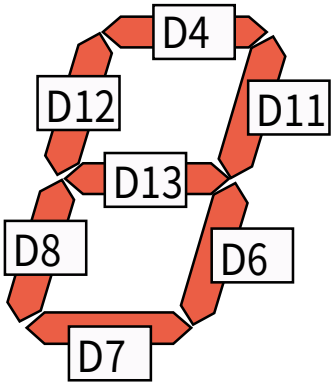
最初の値を大きくして、減算していくこともできます。

Vol.7 ＬＥＤ表示器を使う



ＬＥＤ表示器を使ってみましょう。
この表示器は7個のＬＥＤを使って表現します。

０１２３４５６７８９



ＬＥＤ表示器

最初の頃、表示器の縦棒、横棒がマイコンのどの出力に対応しているのか調べたのを思い出してください。
思ったＬＥＤをつけるのは**degitalWrite()**を使うというのは同じです。
例えば、1を表す場合は、D11とD6をオン、それ以外をオフにすれば良いわけです。

各数字を表示する時にどのピンをオンオフするか整理してみましょう。
「1」を表示するときはD11､D6のみオンなので、その欄にHを、それ以外はＬを記入します。

表を完成させてから
プログラミングに進もう

		ピン						
		D4	D6	D7	D8	D11	D12	D13
表示	0							
	1	L	H	L	L	H	L	L
	2							
	3							
	4							
	5							
	6							
	7							
	8							
	9							

作った表を見ながらプログラミングしていくわけですが、ひとつの数字を表すのには、7行のプログラムが必要です。

```
//1を表示
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
```

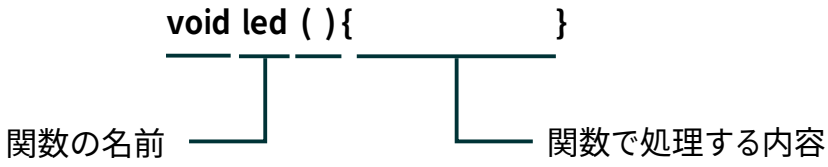
何回も数字の表示が変わるようなプログラムだと、毎回書くのも大変ですし、間違えるかもしれません。プログラムも長くなって見にくくなります。
そういう時のためにプログラムでは決まった処理をする部分だけをひとまとめにして、必要なときに呼び出せば実行してくれるというしくみがあります。
これを「**関数**」と言います。

関数について

実は、いままでプログラムに最初から含まれていた **setup()** も、**loop()** も関数ですが、それ以外にも自分で関数を作って付け加えることができるのです。

関数の作り方

関数は入れ物の形が決まっています。



関数の名前は自由につけることができます。参考ではled という名前にしてみました。関数で行うことは**{ }** かっこの中に書き込みます。

void　や関数の名前のあとの**()**は、もっと複雑はプログラミングで使うためのものです。今は説明しませんが、無くては動かないものなので、そのまま残しておきます。
ＬＥＤ表示器で1を表示する関数は次のようになります。

```
//1を表示する関数 名前をLED_1とする
void LED_1(){
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}
```

では実際に関数を使って、数字を表示するプログラムを作ってみます。
これは、1秒おきに数字をカウントするプログラムです。

```
/*
  7セグメントLED表示器で順に数字表示
 */

//関数の宣言
void LED_0();
void LED_1();
void LED_2();
void LED_3();
void LED_4();
void LED_5();
void LED_6();
void LED_7();
void LED_8();
void LED_9();

// setupのあと{}で囲まれた部分は最初に1回だけ実行される
void setup() {
  //ピンを出力に設定する
  pinMode(4, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}

// loopのあと{}で囲まれた部分は繰り返される。
void loop() {
  LED_0();
    delay(1000);
  LED_1();
    delay(1000);
  LED_2();
    delay(1000);
  LED_3();
    delay(1000);
  LED_4();
    delay(1000);
  LED_5();
    delay(1000);
  LED_6();
    delay(1000);
  LED_7();
    delay(1000);
  LED_8();
    delay(1000);
  LED_9();
    delay(1000);
}
//ここから関数をまとめる//
void LED_0() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, LOW);
}
```

関数の宣言

関数を使う時、
頭の **void** は
いらないが、
(　) は必要。
最後に **;** をつけます。

関数の内容はここから

次のページへ続く

前ページからの続き

```
void LED_1() {
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}


void LED_2() {
 digitalWrite(4, HIGH);
 digitalWrite(6, LOW);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, HIGH);
}

void LED_3() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, HIGH);
}

void LED_4() {
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_5() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
 digitalWrite(11, LOW);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_6() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, LOW);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}
```

次のページへ続く

前ページからの続き

```
void LED_7() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}

void LED_8() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_9() {
 digitalWrite(4, HIGH);
  digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}
```

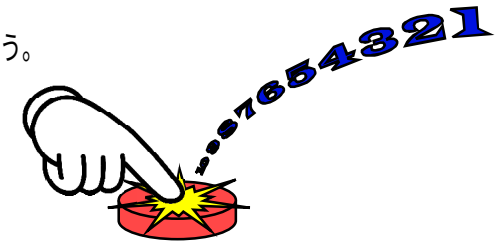
プログラム全体を見た時、最初のほうで「関数の宣言」ということを行っていたのに気が付きましたか？
Arduinoはマイコンボードに書き込む前にコンパイルという、プログラムの内容に間違いがないか確認しながらマイコン用の言語に書き換えています。
その時、関数が **loop()**よりも後に書かれているとエラーになってしまうため
最初に、使っている関数を宣言しているのです。

今までで一番長いプログラムなので、大変かもしれません。
まずコンパイルしてマイコンボードに書込めるところまで進めましょう。
大文字、小文字の間違いはないか、コメント部分以外で全角文字（特に空白）を使っていないか。始まりのかっこと終わりのかっこがちゃんとあるかどうかなどを確認していきましょう。

けれどコンパイルはプログラム上の矛盾を調べるだけなので、コンパイルがうまくいっても、考えていた通りに動かないこともあります。その時は、実際の動作とプログラムを良く見て、どの部分が違うか見つけていきましょう。

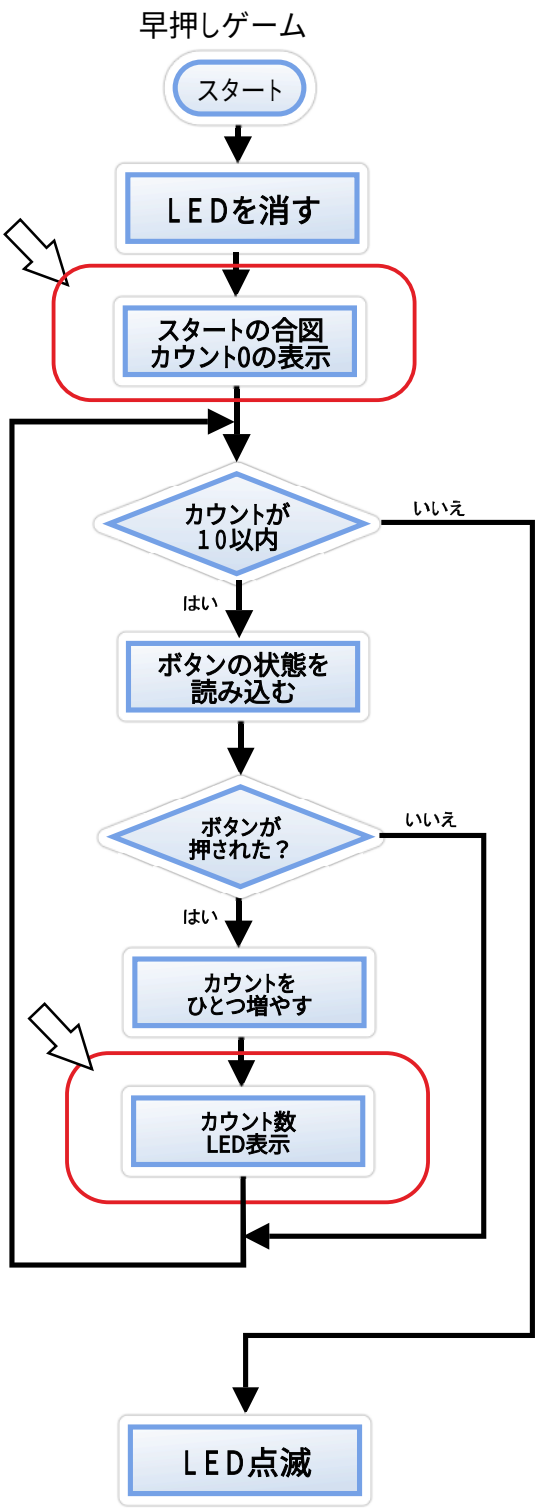
早押しゲームの改造　(ボタンを押した数の表示)

数字の表示ができるようになったので、せっかくなら
早押しゲームでボタンを押した数の表示をしてみましょう。



前回のフローチャートを元にゲームの構想をします。

フローチャートに枠で囲んだ部分を追加しました。



新しいゲームは、スタートの時に、
カウント数　"0"を表示します。

ボタンを押してカウントが増えたら、カウントの値をＬＥＤ表示します。

ＬＥＤは1桁しかないので、カウントが10になった時は全部のＬＥＤを点滅させるようにします。

こうして考えると、

ＬＥＤ表示を全て消す関数は作っていなかったので必要になってきました。

カウント数をＬＥＤ表示する部分は、繰り返し処理の中でもわかりやすくするために関数を作って処理することにします。
今までのif文でもできるのですが不便な

ところがあるので、switch　case　という

新しい命令も使ってみます。

まずswitch　case　命令の説明をします。

switch　case 命令

今まで、条件によって処理を変える命令は　if　や　if　～　else　if　などを使ってきました。条件の分岐が少ない時は良いのですが、今回のように0～9までの関数に分岐する場合は、if　関係の命令では複雑になってきます。

そういう時に使う命令が、 switch　case　命令です。

switch　case　を使って書くとこんな風になります。

```
switch(count){
  case 1:   LED_1(); break;
  case 2:   LED_2(); break;
  case 3:   LED_3(); break;
  case 4:   LED_4(); break;
  case 5:   LED_5(); break;
  case 6:   LED_6(); break;
  case 7:   LED_7(); break;
  case 8:   LED_8(); break;
  case 9:   LED_9(); break;
  case 10: LED_8(); break;
  default: break;
}
```

変数　count　や　関数　**LED_1()**から**LED_9()**までは、今までの早押しゲームのものと同じ意味です。**case**の後は**:**（コロン）です。プログラム区切りの**;**（セミコロン）と間違えないようにしてください。

switch命令は　**switch**の後の　**()**　内の変数の値を読んで、その値がどの**case**に当てはまるかを探して、その**case**部分の処理を行います。処理を行った後は**switch**文を抜けるために　break;　も書いておきます。

最後の**default**は、どのケースにも当てはまらなかった場合の処理です。今回は**count**が0～10以外の値をとることが無いので、省略しても構いません。

case 10 の場合は、全部のＬＥＤ点灯なので、結局"8"と同じになりました。

実際のプログラム

実際に改造した「早押しゲーム」プログラムです。

関数　void LED_0()　～　void LED_9() の部分は前のプログラムからコピペしました。

そこに新たに、全LEDを消す **void LED_OFF()** という関数を追加しました。

また、上のswitch　case　の部分をvoid Display()　というひとつの関数にして、プログラム全体を見やすくしました。

早押しゲーム（ボタンを押した数を表示）

```
int LED = 9;                //回数達成表示
int BUTTON = 16;
boolean readBUTTON = false; //ボタンの状態を入れる変数
boolean lastBUTTON = false; //直前のボタンの状態
boolean nowLED= false;   //現在のLEDの状態
int count = 0;           //ボタンを押した回数

 //関数の宣言
void Display();
void LED_OFF();
void LED_0();
void LED_1();
void LED_2();
void LED_3();
void LED_4();
void LED_5();
void LED_6();
void LED_7();
void LED_8();
void LED_9();

void setup() {
 // 入出力の設定
pinMode (LED,OUTPUT);
pinMode(4, OUTPUT);
pinMode(6, OUTPUT);
pinMode(7, OUTPUT);
pinMode(8, OUTPUT);
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);

pinMode(BUTTON,INPUT);

  //最初はLEDを消した状態
digitalWrite(LED,LOW);
LED_OFF();

//スタートの合図
LED_0();          //カウント数　0　表示
while(count <10)
{
   // ボタンの状態を読み込む
 readBUTTON = digitalRead(BUTTON);
 delay(20);      //安定時間
 /*ボタンが押された瞬間をlastBUTTONとreadBUTTONで判定する
   ボタンが押されていたらカウントを増やす*/
 if(lastBUTTON == false && readBUTTON==true){

   count++;       //カウント追加
   Display();     //カウントをLED表示させる
 }

 lastBUTTON = readBUTTON;   //ボタンの状態を記憶する
}
```

変数　sLEDは使わないのでなくす。

関数の宣言を追加

LED表示器のピンを設定する

LED表示器を消す関数を実行

0を表示する関数を実行

カウント数を表示する関数を実行

次のページへ続く

前ページからの続き

```
 //ここからカウント10の時の処理。点滅を8回繰り返す
  for(int i=1;i<=8;i++)
  {
  digitalWrite(LED,HIGH);  //LEDを点ける
  LED_8();                 //LED表示全点灯
  delay(50);                 //0.05秒待つ
  digitalWrite(LED,LOW);   //LEDを消す
  LED_OFF();
  delay(50);                 //0.05秒待つ
  }
}


//void loop()は使用しない
void loop() {
}

//カウント数を見てLEDを表示させる関数
void Display(){
 switch(count){
  case 1:   LED_1(); break;
  case 2:   LED_2(); break;
  case 3:   LED_3(); break;
  case 4:   LED_4(); break;
  case 5:   LED_5(); break;
  case 6:   LED_6(); break;
  case 7:   LED_7(); break;
  case 8:   LED_8(); break;
  case 9:   LED_9(); break;
  case 10:  LED_8(); break;
  default: break;
 }
}

//以下、LED点灯に関する関数
void LED_OFF() {
 digitalWrite(4, LOW);
 digitalWrite(6, LOW);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, LOW);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}

void LED_0() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, LOW);
}
```

表示器を全部点灯させる関数を実行

表示器を全部消す関数を実行

switch case を使った
カウント数をＬＥＤ表示
する関数

ＬＥＤ表示を
消す関数を追加

次のページへ続く

前ページからの続き

```
void LED_1() {
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}

void LED_2() {
 digitalWrite(4, HIGH);
 digitalWrite(6, LOW);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, HIGH);
}

void LED_3() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, HIGH);
}

void LED_4() {
 digitalWrite(4, LOW);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_5() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
 digitalWrite(11, LOW);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_6() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, LOW);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}
```

右上へ続く

左下からの続き

```
void LED_7(){
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, LOW);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, LOW);
 digitalWrite(13, LOW);
}

void LED_8() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, HIGH);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}

void LED_9() {
 digitalWrite(4, HIGH);
 digitalWrite(6, HIGH);
 digitalWrite(7, HIGH);
 digitalWrite(8, LOW);
 digitalWrite(11, HIGH);
 digitalWrite(12, HIGH);
 digitalWrite(13, HIGH);
}
```

void LED_1() ～ void LED_9() の関数はそのまま使う

その他

ＬＥＤ表示器で0～9までの数字はよく見かけますが、アルファベットの字体もあるので、これらを使って文字を表示するのもおもしろいかもしれません。
少ないLEDで表現するためにずいぶん変形された文字ですが、マイクロコンピューターが発明された頃、ブラウン管や液晶のディスプレイが使われるようになるまでは、これが使われていました。
ｘＥＩＩｏ（XをHとして使用） や、ｇｏｏｄなどはわかりやすいので、マイコンのメッセージとして使えそうです。

7セグメント字体 (他にもパターンがあります)

０１２３４５６７８９

ａｂｃｄＥｆｇｈｉｊｋｌｍｎｏｐｑｒｓｔｕｖｗｘｙｚ
ａｂｃｄｅｆｇｈｉｊｋｌｍｎｏｐｑｒｓｔｕｖｗｘｙｚ

Vol.7 ＬＥＤ表示器を使う

今回のまとめ

1） ＬＥＤ表示器で数字を表示しました。

2） プログラムを簡潔なものにするため、関数を使用しました。

3） 多くの条件分岐に対応するための、switch　case　命令を試しました。

新しく出てきた命令

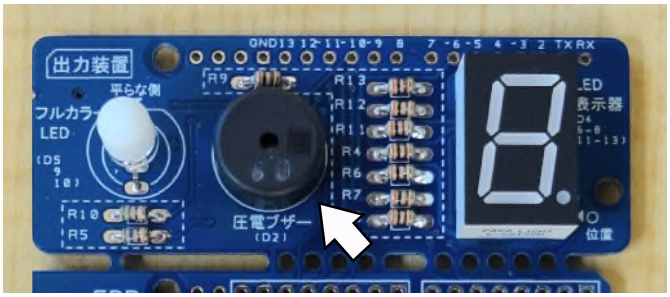
戻り値 関数名(引き数){実行内容}

関数。
勉強したのは、戻り値、引き数の無い、　void　関数名　( ){ 実行内容 }
という形のものでした。　例）　void LED_OFF() {　・・・・}
関数を使うことで、命令ひとつでいくつかの処理を行うことができる。

```
switch（変数）{
  case 1：変数が1の時実行する内容; break;
  case 2：変数が2の時実行する内容; break;
  default: どれにも当てはまらない時の内容;break;
}
```

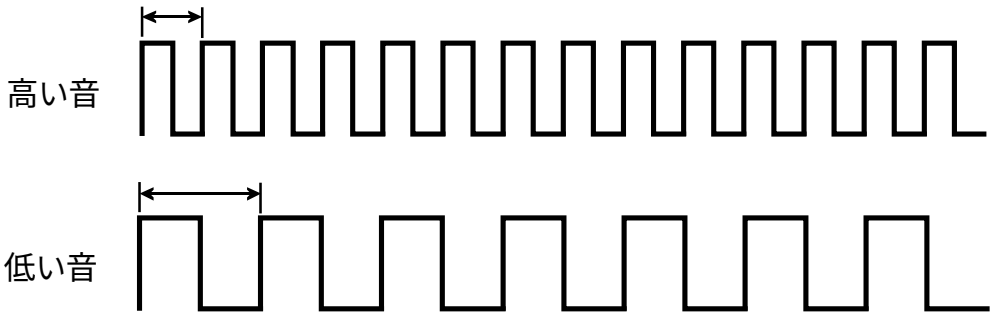
if　～　else　if　～　else　と同様の処理
プログラムを簡潔にすることができる。

Vol.8 音を鳴らす



圧電ブザーでメロディーを奏でます。
vol.1で動作チェックをした時には、圧電ブザー（D2ピン）をオンオフしても、
チッチッという音しかしなかったと思いますが、圧電ブザーを鳴らすには、すばやくオンオフを繰り返すことが必要です。
そしてこの繰り返しの周期を変えることで音の高さを変えることができます。

圧電ブザーを動かす信号



周期が短ければ高い音、長ければ低い音が出せる。

実際のプログラミング

このように、周期を変えて出力をオンオフさせるための　**tone**関数が用意されています。

tone(出力先**,**音の周波数**,**音の長さ**);**　がその命令ですが、

この関数を使う時は少し注意があります。連続して音を出すときには次の音までの間にdelay関数で作っておかないと音が重なって発声しません。
連続した音を出すときは、

tone(出力先**,**音の周波数**,**音の長さ**);**

delay(toneで指定した音の長さ×1.3倍**);**

としてください。ひとつの音だけの場合はdelay()は不要です。
1.3倍としたのは、1倍では音が連続して聞こえるためです。好みで調整してもらって構いません。

1オクターブの発声

1回鳴らすだけなので、setup()に書きましたが、繰り返す場合はloop()に書いてください。

```
/* 各音の周波数
ド:523
レ:587
ミ:659
ファ:698
ソ:784
ラ:880
シ:988
ド:1047

圧電ブザーはD2ピンに接続
*/

//1回だけの演奏なのでsetupに書く
void setup() {
  //ドの音を出す
  tone(2,523,500);
 /*すぐ次の音を出し始めないように
  * 先ほどの音の長さ×1.3ぐらいの待ち時間を入れる*/
  delay(650);
  //レの音
  tone(2,587,500);
  delay(650);
  //ミの音
  tone(2,659,500);
  delay(650);
  //ファの音
  tone(2,698,500);
  delay(650);
  //ソの音
  tone(2,784,500);
  delay(650);
  //ラの音
  tone(2,880,500);
  delay(650);
  //シの音
  tone(2,988,500);
  delay(650);
  //上のドの音
  tone(2,1047,500);
  delay(650);
  }

void loop() {
}
```

音の出し方がわかったら、以前作った「早押しゲーム」に音を出す機能を追加してみましょう。

半音や、1オクターブ上下の音を使いたいときは、インターネットで調べると音階と周波数の関係を記したものを見つけることができます。

Vol.8 音を鳴らす

今回のまとめ

1） 圧電ブザーで音を出しました。

新しく出てきた命令

tone(出力先,周波数,音の長さ);

圧電ブザーで音を出す命令です。
指定された出力先に、周波数、音の長さを指定して方形波を出力します。