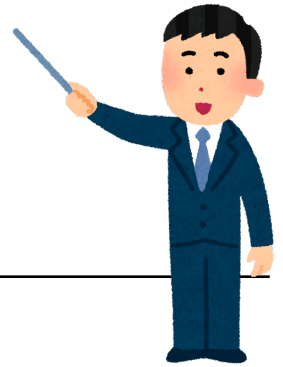


Arduinoテキスト

初級編



目次

1.Arduino IDE（ソフトウェア）の使い方

Arduino IDEとは 使い方 プログラム実行までの流れ

2.スケッチの構成

setup() loop()

3.基本的な文法

;(セミコロン) {}(波カッコ)

4.LEDを光らせてみよう

pinMode(pin, mode)とdigitalWrite delay

5.ボタンを使ってみよう

digitalRead Serial.print()

6.アナログ入出力関数

analogRead analogWrite PWMとは

7.算術演算子

+ - * / %(余剰) =(代入)

8.制御文と演算子

if (&&(論理積) ||(論理和) !(否定) == != <> <= >=) if else
for ++(加算) --(減算)
while break continue

9.Tone関数

tone(pin, frequency) tone(pin, frequency, duration) noTone(pin)

10.データ型

boolean(bool) byte int unsigned int (符号なし整数型)
word long (long整数型) unsigned long void

11.Stringクラス

String()

12.変数の応用

変数のスコープという概念

13.関数

戻り値・引数・return

おまけ

複合演算子

+= -= *= /=

1.Arduino IDE(ソフトウェア)の使い方

Arduino IDEとは

まず初めにArduino IDEについて軽く説明します。

Arduino IDEとはArduinoの開発環境でArduinoソフトウェアとも呼ばれるフリーウェアで、コンパイル・ダウンロード・実行などの一連の作業がこれ一つでできる優れものです。(wiki調べ)

Arduino IDE の使い方

1.Arduino IDEのインストール

1-1.Windowsの場合

①.以下の手順でArduino IDEをダウンロードします

- 1) Arduinoサイト(<https://www.arduino.cc/en/Main/Software>)にアクセスします。
- 2) 「Windows ZIP file for non admin install」 をクリックしてダウンロードします。

②.ダウンロードしたzipファイルを展開します。

③.展開したフォルダの内容を任意の場所に移動します。

1-2.Macの場合

①.以下の手順でArduino IDEをダウンロードします

- 1) Arduinoサイト(<https://www.arduino.cc/en/Main/Software>)にアクセスします。
- 2) 「Mac OS X 10.7 Lion or newer」 をクリックしてダウンロードします。

②.ダウンロードしたzipファイルを展開します。

③.展開されてできた、Arduinoアイコンをアプリケーションフォルダに移動します。

以上で、Arduino IDEのインストールは完了です。

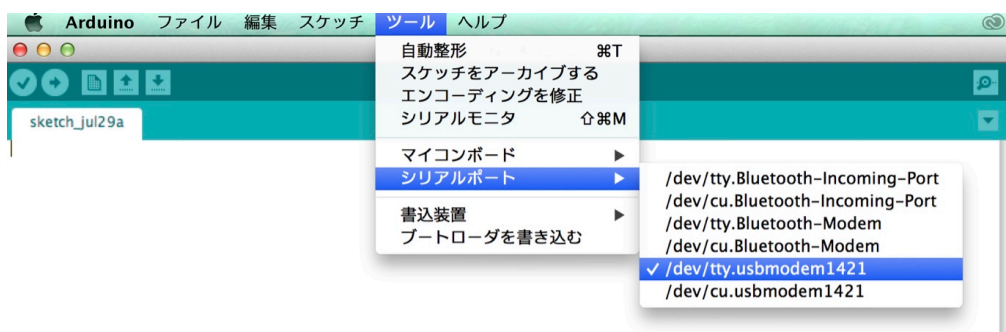
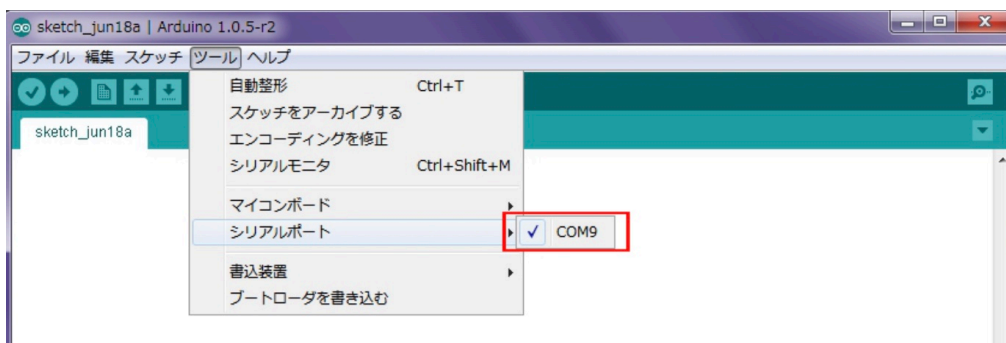
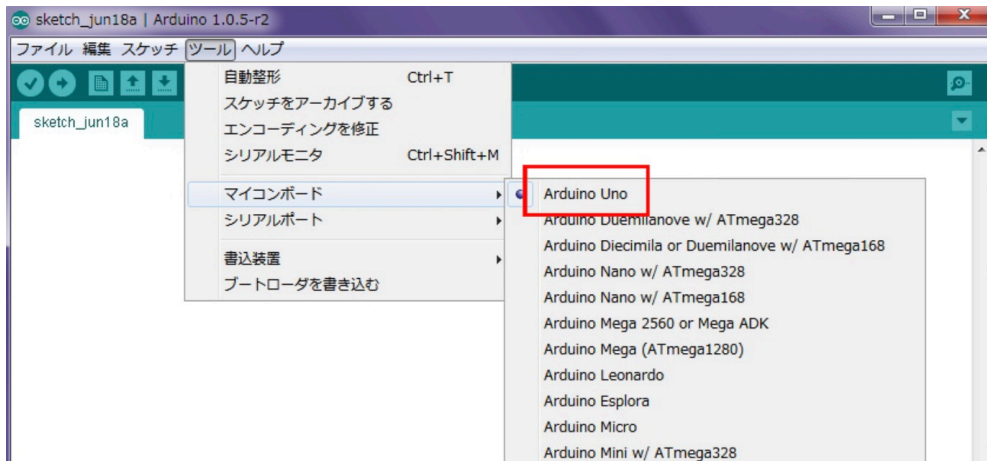
2.PCとArduinoを接続

2-1.インストールが完了したら、ArduinoとPCをUSBケーブルで繋がします。

2-2.Arduinoを起動してマイコンボード、シリアルポートの設定を行います。

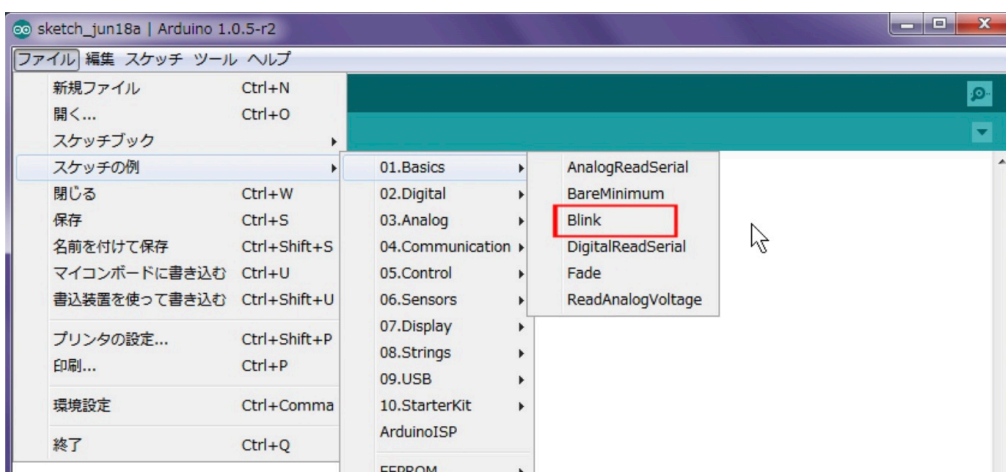
1)マイコンボードの設定メニューの「ツール」→「マイコンボード」より、使用するマイコンの種類(Arduino UNOなど)を選択します。

2)マイコンボードの設定メニューの「ツール」→「シリアルポート」より使用するシリアルポートの種類を選択します。

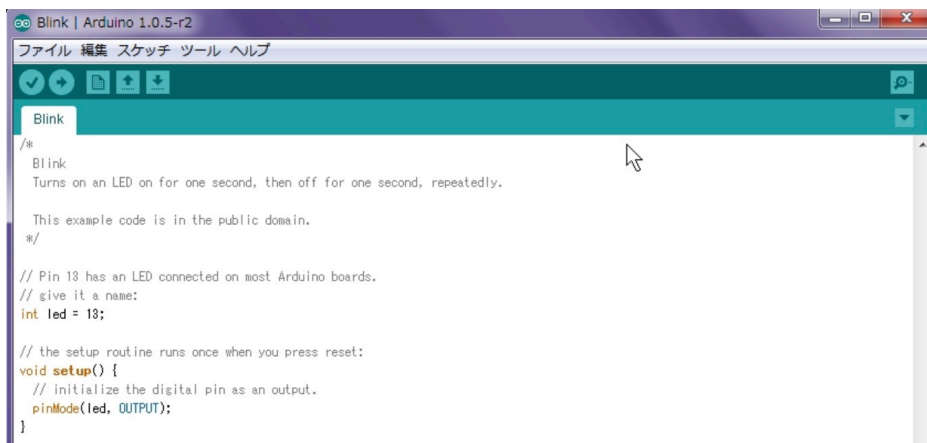


3. サンプルコード読み込み

3-1. マイコンボードの設定メニューの「スケッチ例」→「01.Basics」→「Blink」を選択します。

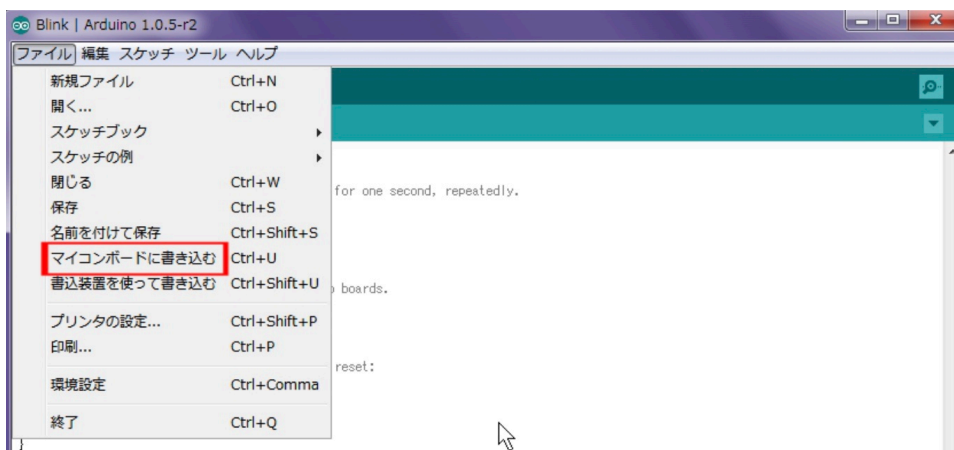


3-2. 「Blink」を選択するとLED点滅用のサンプルコードが表示されます。
Arduinoではこのコードをスケッチと呼びます。



4.実機転送

4-1.スケッチをマイコンボードに書き込みます。マイコンボードの設定メニューの「マイコンボードに書き込む」を選択します。



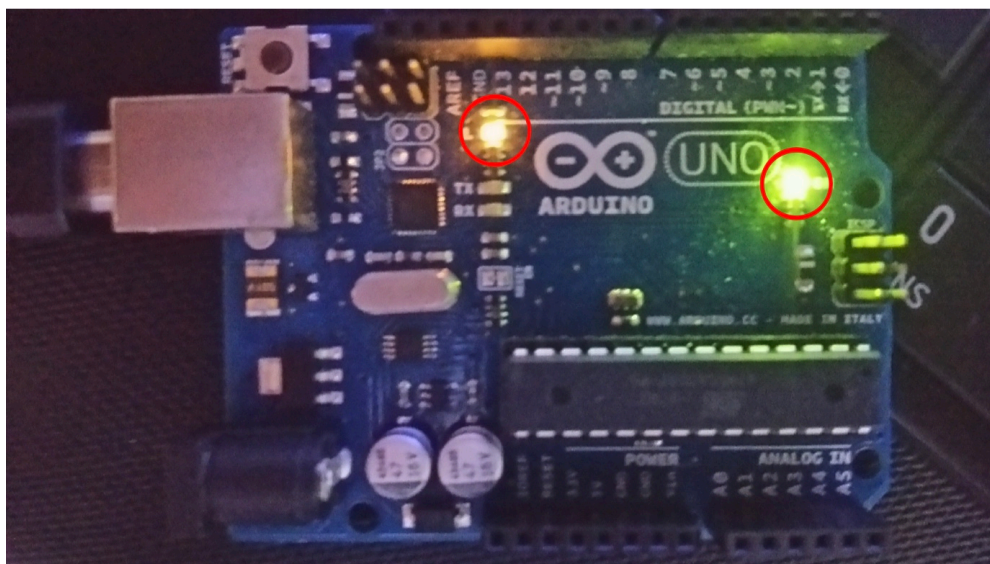
4-2.書き込みに成功すると、「マイコンボードへの書き込みが完了しました。」のメッセージが表示されます。



ここでエラーになる場合は、マイコンボード設定、シリアルポート設定に誤りがある可能性がありますので一度確認してください。それでもエラーが出る場合は配線ミスや基板の不具合の可能性があります。

5. 結果確認

書き込みが正しく完了するとArduinoについているランプが点灯します。緑が常時点灯、オレンジが点滅です。



プログラム実行までの流れ

① スケッチを書く

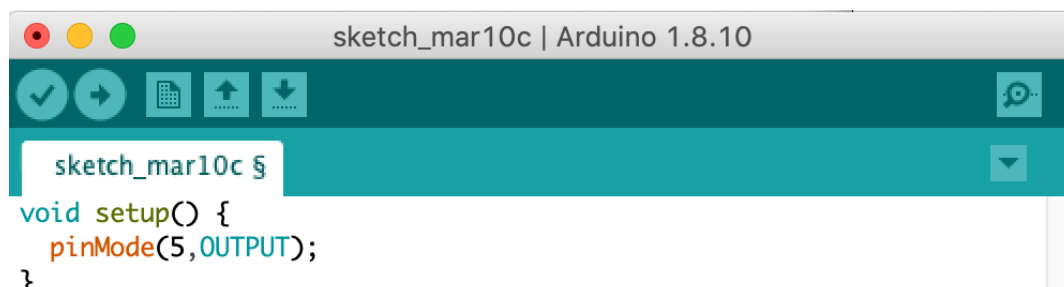
スケッチに、Arduinoに実行させるプログラムを書きます。

```
sketch_mar10c | Arduino 1.8.10
void setup() {
  pinMode(5, OUTPUT);
}

void loop() {
  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);
}
```

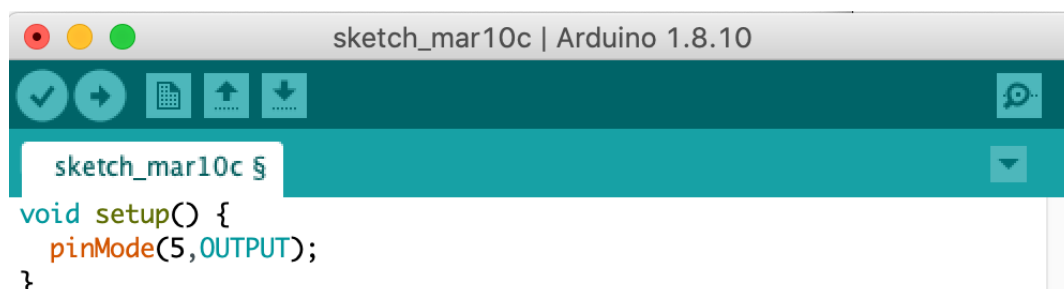
②コンパイルを行う

画面左上のチェックマーク✓をクリックしてコンパイルを行い、プログラムにエラーがないかをチェックします。コンパイルが完了すれば③書き込みを行い、エラーがあったなら修正します。コンパイルはプログラムを保存しなくても実行できます。



③書き込み

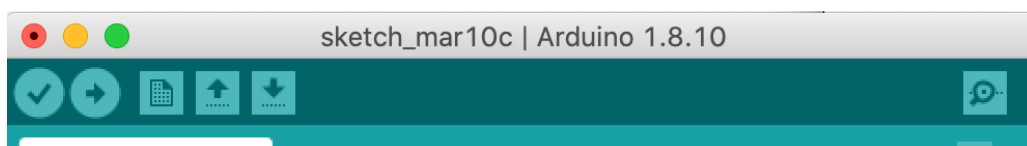
コンパイルの横の矢印→をクリックしてArduinoボードにプログラムを書き込みます。完了すると、Arduinoボードがプログラムを自動的に実行されます。書き込みを行うには、プログラムを保存する必要があります。



その他

シリアルモニター

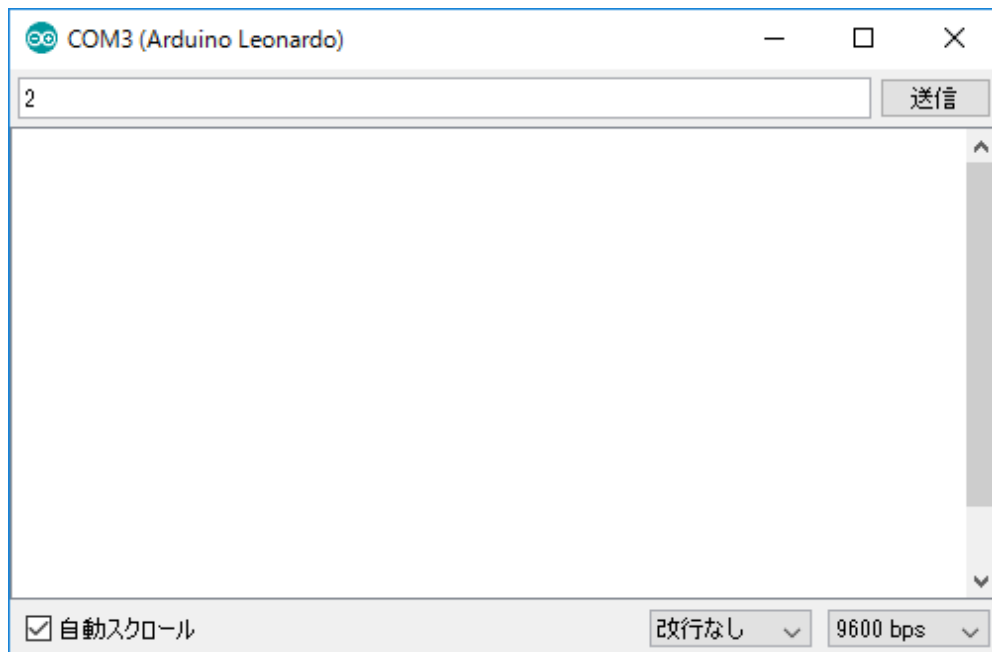
PCとArduinoなどのデバイスのシリアル通信で使われます。「ツール」>「シリアルモニター」または画面右上の虫眼鏡マークをクリックして起動します。



Arduinoで読み取ったデータや出力されたテキストなどを表示させたりする事ができます。

(Serial.print()など)

ここでシリアルモニターを使ったプログラムをいくつか紹介します。



2.スケッチの構成

setup()

Arduinoに電源が入った時やリセットボタンが押された時に1度だけ{}内が実行される。変数やピンモードの初期化(注1)や、ライブラリの準備等に使えます。尚、ここでの「初期化」は、メインのプログラムを実行する前に状態を整える事を指し、変数の初期化においては、変数の宣言や定義を行います。ただし、変数の初期化は関数の外のグローバルで行うのが一般的。setup()内で break;やcontinue;を使う事は出来ません。

<プログラム例>

```
int buttonPin = 3;    // 変数の初期化
void setup() {
  Serial.begin(9600); // シリアル通信の通信レートの設定とシリアル通信の開始
  pinMode(buttonPin, INPUT); // ピンの初期化
}
```

(注1) ここでの「初期化」は、メインのプログラムを実行する前に状態を整える事。変数についての初期化は、宣言(注2)や定義(注3)です。

(注2) 「宣言」は、変数の型と名前を決めて生成する事。

(注3) 「定義」は、変数に値を代入する事。

パソコンでコンパイル前にプログラムの内容で指定した文字列を定数や式で置き換えるマクロ(`#define`)は、Arduinoが行う処理ではないので、`setup()`内に入れるとエラーになります。

(`Serial.begin(9600);`と`beginSerial(9600);`は同じ役割を持つ)

loop()

setup()の処理が終わると、loop()の{}内が電源が切れるまで繰り返し実行されます。

<プログラム例>

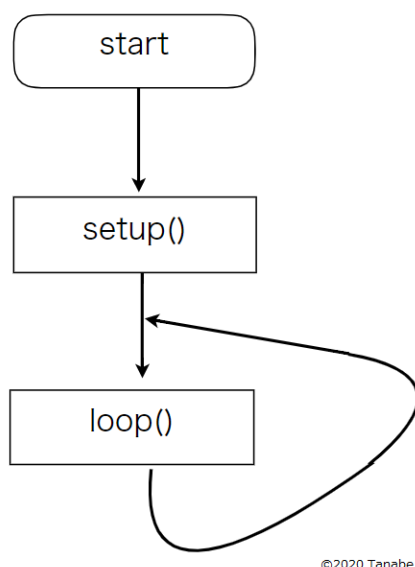
```
int buttonPin = 3;  
// ピンとシリアル通信の初期化
```

```
void setup() {  
  Serial.begin(9600);  
  pinMode(buttonPin, INPUT);  
}  
// buttonPinを繰り返しチェックして、  
// その状態をシリアルで送信する
```

```
void loop() {  
  if (digitalRead(buttonPin) == HIGH) {  
    Serial.write('H');  
  } else {  
    Serial.write('L');  
  }  
  delay(100);  
}
```

上のプログラムでは、ボタンのオンオフを約0.1秒ごとに調べ、押すとHが、そうでない場合はLが表示されます。高速で繰り返し状態を調べることでリアルタイムにボタンの状態が分かります。
(Serial.write()とserialWrite()は同じ役割を持つ)

setup()、loop()のフローチャート



注意

setup(){}やloop(){}を省略するとエラーになります。
loop()は繰り返しの処理ですが、break; や continue; を使うことは出来ません。

問題2

以下のプログラムの間違いを指摘してください

```
void Setup(){
  #define ledPin 3
  pinMode(ledPin,OUTPUT);
}

void loop{
  digitalWrite(ledPin,HIGH);
  delay(1000);
  break;
}
```

3.基本的な文法

;(セミicolon)

セミicolonは、プログラム内で区切りをつけるのに使い、文末などにつけます。

セミicolonをつけ忘れてコンパイルすると、セミicolonがない箇所以外がエラーとして示されることがあります。

<プログラム例>

```
Serial.begin(9600);  
Serial.println("test");
```

<プログラム例>

```
for (int i=0; i <= 255; i++){ } // 必ずしも文末とは限らない
```

{}(波カッコ)

必ず開きカッコ『{』と閉じカッコ『}』があり、基本的に文をその中に書きます。

波カッコは様々な使い方があります。

関数

```
void myfunction(=引数) {  
  文  
}
```

ループ

```
while (式) {  
  文  
}  
do {  
  文  
} while (式);  
for (初期化; 式; 加算) {  
  文  
}
```

条件分岐

```
if (式) {  
  文  
}  
else if (式) {  
  文1  
} else {  
  文2
```

波括弧の使用際、

『{』を入れたら、すぐに『}』も打つことを習慣づけるとよいでしょう。



後に「カッコを打ち忘れて動かない!!」案件が起こらなくなります。

4.LEDを光らせてみよう(実験)

下のプログラムを書いてみましょう。

```
void setup(){
  pinMode(3,OUTPUT);    //3ピンを出力に設定
}

void loop(){
  digitalWrite(3,HIGH);  //3ピンから出力
}
```

Arduinoボードに書き込みましょう。すると、LEDが光ります。

`pinMode(3,OUTPUT);`は、Arduinoボードの3ピンを出力に設定します。

出力とは、電流を流すということです。

`digitalWrite(3,HIGH);`は、3ピンから出力を行います。

つまり、3ピンから5Vを出力するということです。

3ピンから電流が出力されることで、LEDに電気が通り、結果的に光るのです。

`pinMode(pin, mode)`

Arduinoで使いたいピンを指定し、出力か入力を設定します。

基本的にsetup()内で行われ、「`pinMode([設定するピンの番号], [入出力設定])`」と記述します。

※pinModeのMはpinとmodeが別の単語であることを示すためにあえて大文字にしています

`mode = [入出力設定]`

使うピンを出力か入力に設定します。

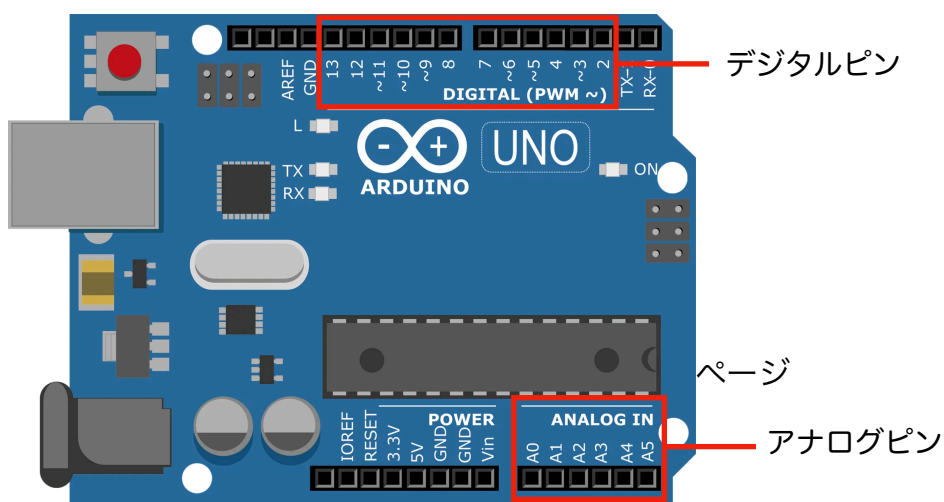
出力の場合は「OUTPUT」、入力の場合は「INPUT」と書きます。

`pin = [設定するピンの番号]`

赤い刺客で囲った部分が設定できるピンです。

ピンで行う動作によって使える番号が変わります。

デジタルピン (2~13) アナログピン (A0~A5)



digitalWrite(pin, value)

指定したデジタルピンにHIGHかLOWを出力します。
pinMode()でOUTPUTに設定されたピンで使します。



pin = デジタルピンの番号（2～13ピンが使用できます）

value = HIGHかLOWを指定します。HIGHにすると5V、LOWにすると0V(GND)が出力されます。

試しにdigitalWriteを使ってLEDを光らすプログラムを書いてみましょう。

valueをHIGHにするとLEDが光り、LOWだとLEDは光りません。

<プログラム例>

```
void setup(){
  pinMode(7, OUTPUT);    //7ピンをOUTPUT(出力)に設定
  pinMode(8, OUTPUT);    //8ピンをOUTPUT(出力)に設定
}

void loop(){
  digitalWrite(7, HIGH);  //LEDが光る（7ピンから5Vの電圧が流れる）
  digitalWrite(8, LOW);   //LEDは光らない（8ピンから0Vの電圧が流れる）
}
```

上のプログラムは、7ピンのLEDが光り、8ピンのLEDは光らないという動作を実行します。

問題4-1

LEDを光らせてみよう（できれば回路も作って確かめてみよう）
片方を光らせて片方を消えたままにするプログラムを書いてみましょう
使うピンは2ピン3ピンとします

delay(ms)

プログラムの処理を一時的に止めたい時(遅延させたい時)にはdelay()という関数を使います。この関数はプログラムを指定した時間だけ止めることができます。()内には止めたい時間を入れることができ、単位はミリ秒(1/1000秒)となっています。

例えば1秒止めたい時にはdelay(1000)と入れます。

このdelayを使った次のプログラムを書いてみましょう。

<プログラム例>

```
void setup(){
    pinMode(7, OUTPUT);    //7ピンをOUTPUT(出力)に設定
}

void loop(){
    digitalWrite(7, HIGH);  //LEDが光る (7ピンから5Vの電圧が流れる)
    delay(1000);            //1秒間停止(LEDが光ったまま)
    digitalWrite(7, LOW);   //LEDが消える (7ピンから0Vの電圧が流れる)
    delay(1000);            //1秒間停止(LEDが消えたまま)
}
```

上のプログラムは、LEDが1秒間光り、1秒間消えるという動作を繰り返します。LEDが光ったり消えたりするこれをチカチカと言います(LEDチカチカ)。

問題4-2

LEDを二つ交互にチカチカ光らせてみよう (片方が光っている時、もう片方は光っていない)

- ・ 使うピンは2ピンと3ピン
- ・ delay使用

5. ボタンを使ってみよう（+α）

digitalRead(pin)

指定したデジタルピンからの値を読み取り、その値を返します。デジタルピンからの値とは、1と0の事です。

pinMode()でINPUTに設定されたピンで使します。

pin = デジタルピンを指定します。

基本的に、スイッチのON/OFFを読み取るのに使します。

digitalReadを使って、スイッチの入力状況を取得するプログラムを書きましょう。

押された場合はHIGH（1）が、押されていない場合はLOW（0）が返ってきます。

<プログラム例>

```
void setup(){
    Serial.begin(9600);
    pinMode(7, INPUT); //7ピン(スイッチ)をINPUT(入力)に設定
}

void loop(){
    int value;    //変数「value」を定義
    value = digitalRead(7); //digitalReadで読み取った7ピンからの値を「value」に格納
    Serial.println(value); //シリアルモニターに「value」の値を表示
}
```

上のプログラムを実行してシリアルモニターを開き、スイッチを押してないと「0」、押すと「1」が表示されます。

<プログラムの説明>

Serial.begin(9600);の説明は、難しいので省きます。

pinMode(7, INPUT);で、7ピンを入力に設定します。7ピンは、スイッチに繋がっています。

int value;では、intが使われています。「int (型の名前)」と書くと、値を格納できる変数を宣言します。今回は、valueという変数を作ります。

value = digitalRead(7);は、7ピンから読み取った値を、valueという変数に代入します。

Serial.println(value);で、valueの値をシリアルモニターに表示させます。()の中に入れた物が表示されます。

Serial.print()

シリアルモニターに文字(数値)を表示させるには、これを使用します。PCとArduinoボードをUSBケーブルで繋いでるときのみ、表示できます。

()の中には、整数型とSrting型（文字）を書き込みます。つまり、数字と文字を入れます。それにより、センサーが読み取った値を表示させたり出来ます。

文字や数字を表示する際は、`”`の間に文字を書きます。

```
Serial.print("Hello!");
```

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

→Serial.printにlnを付け加える。これにより、シリアルモニターで表示させる時、改行が行える。こっちの方がよく使うので、覚えておきましょう。

printとprintlnで比較してみましょう。

Inを付けない場合

```
Serial.print("abc");
```

↓

abcabcbcabcbcabcbcabcbcabcbcabcbcabcbcabcbcabcbcabcb

このように文字が改行されずに連続して表示されます。とても見辛いですね。

Inを付けた場合

```
Serial.println("abc");
```

↓

abc

abc

abc

abc

abc

abc

...

このように文字が改行されるためとても見やすいです。こっちを使いましょう。

基本の使い方

Arduinoを使って、シリアルモニターに文字(数字)を表示させましょう。

```
void setup(){
  Serial.begin(9600);
  Serial.println("Hello! World");    // 「Hello World!」 を表示させる
}
```

これを実行してシリアルモニターを確認すると、「Hello World!」と表示されます。

整数型を使って数字を表示させましょう。

```
int x = 30;
void setup(){
  Serial.begin(9600);
  Serial.println(x);    //x(=30)を表示する
}
```

今回、xは値の型で、30という整数を格納しているので、結果的にシリアルモニターには30が表示されます。

6.アナログ入出力関数

analogRead(pin)

指定したアナログピンからの値を読み取り、0～1023までの整数値を返します。

0～1023の整数値は、0～5Vの入力電圧を変換したものです。

アナログの値を読むパーツ(光センサー、可変抵抗器など)を読み取るのによく使います。

pin = 読み取るアナログピン番号。

Arduino UnoではA0～A5ピンが有効、

Arduino Mini、NanoはA0～A7ピンが有効、

Arduino MegaではA0～A15が有効です。

ボードの種類によって使用できるピンは異なりますが、どのボードでもA0～A5は必ず使えます。

今回は、可変抵抗器を使用します。ダイヤルを回すと抵抗値が変わり、電圧も変化するため、digitalRead()で読み取る値も変わります。

<プログラム例>

```
int analogPin = 3;          //可変抵抗器を3ピンに指定
void setup(){
    Serial.begin(9600);
}
void loop(){
    int val;                 //値を格納する整数型「val」を宣言
    val = analogRead(3);     //3ピンから読み取った値を「val」に代入
    Serial.println(val);     //シリアルモニターで値を表示
}
```

このプログラムを実行してシリアルモニターを開くと、読み取った電圧が表示されます（値はリアルタイムで常に更新されます）。可変抵抗器のダイヤルを回して、値が正しく変化していれば成功です。

豆知識

analogRead、analogWriteで使うピンは、pinMode()でピン設定を行う必要がありません。

```
int (適当な名前) = (ピン番号)
int analogPin = 3;    //3ピンを設定
```

これだけでピン設定が完了します。便利ですね。

可変抵抗器

抵抗の値を変更できるもの。ダイヤルを回して値を変化させます。例えば100kΩの可変抵抗なら、0～100kΩまでの抵抗値を自在に調整できます。

プログラムでは`analogRead()`を使用します。

一般的なものはピンが3つ付いており、（+）、（-）、アナログ入力ピンの役割を担っています。どれも同じピンなので、どれに何を接続しても問題ないです。

`analogWrite(pin, value)`

指定したピンからPWMによるアナログ電圧を出力します。細かい出力が可能です。一度`analogWrite()`を使用すると、次に`analogWrite()`、`digitalWrite()`、`digitalRead()`が同じピンに対して使われるまで、安定したPWM波が出力されます。

pin = [出力するピン番号]



Arduino Unoでは、デジタルピンの「3, 5, 6, 9, 10, 11」で使用できます。ピン番号の隣に「～」が表記されてるのが目印です。

`value` = 0～255の数値を設定してデューティ比を指定。この値を調整する事で、出力する電圧が設定できます。（デューティ比はこの場合、出力電圧の割合と考えましょう）

`value`が0の時、デューティ比は0で、0Vの電圧を出力、

`value`が255の時、デューティ比は1で、5Vの電圧を出力、

`value`が128の時、デューティ比は0.5で、2.5Vの電圧を出力します。

試しに`analogWrite()`を使用してLEDを光らせてみましょう。

<プログラム例>

```
int ledpin = 3;    //出力するピン（3ピン）を設定
void setup(){
  Serial.begin(9600);
}
void loop(){
  analogWrite(3, 255); //LEDを光らす。（3ピンから5Vを出力）
}
```

このプログラムを実行すると、LEDが明るく光ります。`analogWrite()`の`value`の値を変えれば、明るさも変化します。

PWMとは

Pulse Width Modulationの略で、電圧を制御する方法の一つです。

analogWrite()で使用されています。

Arduinoから出力される電圧は、0Vか5Vのどちらかしか出力できません。しかし、PWMを利用すれば、0Vや5V以外の細かい電圧を疑似的に出力することができます。

つまり例えば、5Vを出力しているが、PWMで出力する割合を調節し、50%分の2.5Vを出力できるという感じです。

7.算術演算子

プログラムを書く上で、読み取った値を算術演算子を使って値を変えることができます。

+ - * /

整数の加算、減算、乗算、除算の結果を返します。

+-は加算と減算、つまり足し算と引き算を表す。記号は数学でもお馴染みの+-で書きます。

例：**m=1+2** **//mは3である**
 m=2-1 **//mは1である**

***** は乗算、つまり掛け算を表す。記号は数学で使う×ではなく*と書きます。

例：**m=2*3** **//mは6である**

/は除算、つまり割り算を表す。記号は数学で使う÷ではなく/と書きます。

例：**m=6/3** **//mは2である**

% (剰余)

整数の割り算を行った際の**余り**を返します。割った答えでは無く**余り**です。（間違えやすいので気を付けましょう）

この%（剰余）は、floatの値には機能しない、つまり小数点を含む数字には使えないので注意です。

例：**m = 10 % 3** **//mは1である** （ $10 \div 3 = 3 \cdots 「1」$ ）
 m = 5 % 5 **//mは0である**

例：**m = 4 % 5** **//mは4である** （小数点を含む数字は出せないなので、割られる数である4を返す）

= (代入)

=の右側の値を、左の変数に代入します。

例

a=3; **//変数aに3を代入** （数学のように左辺と右辺が等しいことは表していません）

a=a+1; **//変数aにa+1を代入** （違和感があると思いますが、aにa+1した値を代入します）

8.制御文と演算子

if (+&&(論理積),||(論理和), ! (否定),== != <> <= >=
 ヨーダ記法)

ifは下のように入ります。

```
if(条件式){  
    処理  
}
```

if文は、「**条件式が満たされたなら処理を行う**」という物です。条件が満たさなければ処理をスキップします。

() カッコの中に条件を書き、{}波カッコの中に条件が満たされた場合の処理（動作）を書きます。

条件式が満たされるのを「true」、満たされないのを「false」と表します。

※{}波カッコは複数の処理をまとめるためがあるので、処理が1つしかない場合は{}を省略してif()条件式の隣に処理を書いても大丈夫です。

例：if(digitalRead(3) == HIGH) digitalWrite(4,HIGH);

実際のプログラムを書いてみましょう。

「スイッチを押されたら、LEDが光る」プログラムです。

```
void setup(){  
    pinMode(3, INPUT);    //3ピンを入力に設定（スイッチ）  
    pinMode(9, OUTPUT);   //9ピンを出力に設定（LED）  
}  
  
void loop(){  
    if(digitalRead(3) == HIGH){ //3ピンがHIGHならば  
        digitalWrite(9, HIGH); //9ピンから出力する  
    }  
}
```

<プログラムの説明>

setup()内では、3ピンを入力（スイッチ）に、9ピンを出力（LED）に設定します。

if(digitalRead(3) == HIGH)を見てみましょう。この条件式は「読み取った3ピンがHIGHである」という意味です。これが満たされれば、digitalWrite(9, HIGH); (LEDが光る) が行われます。

比較演算子

ifの条件式でよく使われます。

`if(digitalRead(3) == HIGH)`で使われる「==」が比較演算子です。

例：

`X == Y` (XとYが等しい) (= (代入) とは異なるので注意です)

`X < Y` (XはYより小さい)

`X > Y` (XはYより大きい)

`X <= Y` (XはY以下)

`X >= Y` (XはY以上) (>=は大なりイコールと呼びます。=>イコール大なりとは書きません)

&& (論理積)

if文にて、条件が複数ある場合に使われます。条件と条件の間に書きます。

条件AとBがある場合、「AがtrueかつBがtrueならば、処理が行われる」となります。

つまり、複数の条件を全て満たすと実行されるということです。

例：XとYどちらもtrueであれば処理が行われます。

```
if(X == 1 && Y == 1){  
  処理  
}
```

|| (論理和)

if文にて、条件が複数ある場合に使われます。条件と条件の間に書きます。

条件AとBがある場合、「AがtrueまたはBがtrueならば、処理が行われる」となります。

つまり、複数の条件のうち、いずれかを満たすと実行されるということです。

(条件を全て満たしても実行されます)

例：XまたはYのどちらかがtrueであれば処理が行われます。

```
if(X == 1 || Y == 1){  
  処理  
}
```

ヨーダ記法

```
int a=4;  
if(a=3){  
    Serial.print("OK");  
}
```

”何故か”OKと表示されてしまいました。
何がいけないでしょう？

a=3のところですね。
aに3を代入してしまっています。
a==3と書きたかったのです。
しかし、こういった場合にエラーが出ません。
なので、エラーは出ないのに想像していた動きをしないといったことが起きてしまいます。
長いプログラムを書いていて、これがミスだとガックリきます。

※これを避けるために**ヨーダ記法**というものがあります。
条件式の左辺に値を書くスタイルのことを指します。

```
int a=4;  
if(3==a){  
    Serial.print("OK");  
}
```

といった書き方があります。見つらくなる代わりに

```
int a=4;  
if(3=a){  
    Serial.print("OK");  
}
```

と書いたときに3==aは認められるが、3=aはわけわからないので、
エラーを吐きます。(3にaを代入しているから)

ちなみにヨーダ記法の由来はヨーダの
「我々で、シスを倒さねばならない。」
みたいな倒置法のしゃべり方から来てるらしいです。
フォースと共にあらんことを…



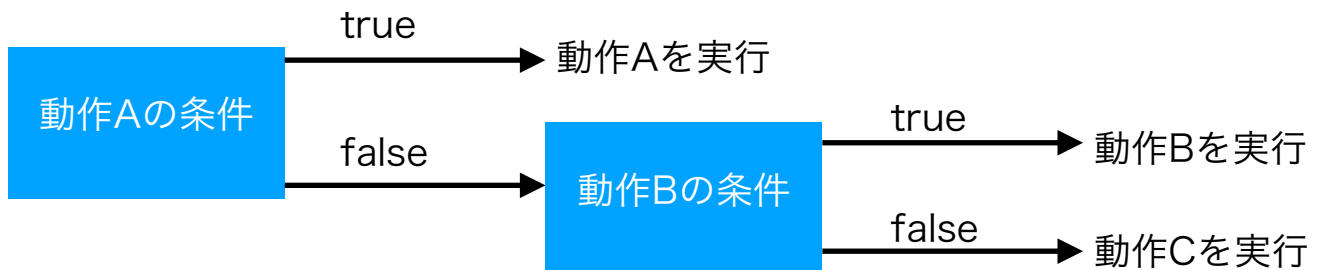
else if

単なるifよりも条件分岐をさせたい時に使います。

通常のifは、falseでプログラムがスキップされます。しかし、if elseならfalseの先に新たな条件式を設けることができます。

構文

```
if (pinFiveInput < 500) {  
  // 動作A  
} else if (pinFiveInput >= 1000) {  
  // 動作B 動作Aの条件がfalseで動作Bの条件がtrueなら実行  
} else {  
  // 動作C 該当する条件が無かった時  
}
```



※elseはなくても問題なく、trueになるものが無ければ何も処理が行われずにスルーされます。

問題8-1

1.下記の条件式の意味を読みとって下さい

- ① `if(X == 10)` ② `if(Y >= 10)` ③ `if(Z < 10)`

2.if文を使って「digitalReadで読み取った2つの値がどちらもLOWの時のみにLEDが光る」というプログラムを書いてみましょう

今回は入力ピンを2ピンと3ピン、出力ピンを4ピンとして使う事とします

(ヒント: 論理積を使おう)

for (+ ++(加算), --(減算))

for文は、指定した回数分処理を繰り返すことができます。
一見難しそうですが、そんなことはありません。

for文は下のように書きます。

```
for( 初期化 ; 条件式 ; 計算式 ){  
    処理 ;  
}
```

処理をして、その後に計算式が実行され、条件式がtrueの間、繰り返します。
といっても分からないので、試しに書いてみます。
下のプログラムは、「OK」を3回表示します。

```
for(int i=0; i<3; i++){  
    Serial.print("OK");  
}
```

for(int i=0; i<3; i++)

int i=0; → i=0を宣言します。つまり、i=0からカウントが始まります。

i<3; → i<3がtrueである間 (i=2になるまで)、処理を行います。

i++ → 処理が終わるごとに、i+1が行われます。

(流れ)

i=0 OKと表示

i=1 OKと表示

i=2 OKと表示

i=3 i < 3がfalseとなりループから抜け出す。

3回OKと表示されます。

LEDが段々明るくなって、段々暗くなる、を繰り返すプログラムを書いてみましょう。

```

int ledpin = 5;    //出力するピン（5ピン）を設定
void setup(){
    Serial.begin(9600);
}

void loop(){
    int i;
    for(int i=0; i<255; i++){    //iの値を0～255に上げていく
        analogWrite(ledpin, i);    //iの値でLEDを光らす
        delay(20);                //20ms毎に更新(明るさが増す)
    }

    for(int i=255; i>=0; i--){    //iの値を255～0に下げていく
        analogWrite(ledpin, i);    //iの値でLEDを光らす
        delay(20);                //20ms毎に更新(暗くなっていく)
    }
}

```

<プログラムの説明>

1つ目のfor文では、LEDが段々明るくなる動作を実行している。

for文でiの値を0から255まで上げていく事で、それに比例してanalogWriteの出力電圧が上がっていき、LEDが明るくなっていく。。

2つ目のfor文は、LEDが段々暗くなるよう実行している。

for文でiの値を255から0まで下げていき、それに比例してanalogWriteの出力電圧も下がっていき、LEDが暗くなっていく。

この2つのfor文が繰り返されることによって、結果的にLEDが明るくなったり暗くなったりするのです。

i や a など普通の変数として使えます。

つまり...

```

for(int i=0; i < 3; i++){
    Serial.print(i);
}

```

0,1,2と表示されます。

普通の変数なので、他の変数と同じようにその中でしか使えません。なので、

```

for(int i=0; i < 3; i++){
    Serial.print(i);
}

```

```
Serial.print(i);
```

などと書くと、エラーが出ます。(詳しくは変数のスコープの欄参照)

```
for(int i=0;i<3;i++)Serial.print("OK");
```

のように一文なら{ }がいりません。見やすさ的に付けるか付けないかは決めるといいかもです。

問題8-2

1 下記のプログラムで「ドーン！！」は何回表示されるでしょう？

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  int val;
  for(val=0; val<=54; val++){
    Serial.println("ドーン！！")
  }
}
```

2 次の条件を考慮しプログラムを書きなさい

条件：forを用いて1から100までの値を書き出せ

while

while文は括弧内の条件式がfalseになるか、break;が関数内で実行されるまで{}を繰り返し実行します。

構文

```
while(条件式){
  // 実行される文
}
```

<プログラム例>

```
var = 0;
while(var < 200){
  // この部分が200回繰り返される
  var++;
}
```

whileを使い、プログラムを途中でストップさせることもできる。

<プログラム例>

```
Serial.print("ready");
```

```
while(digitalRead(btnPin)) {}; // ボタンを押すとfalseになり、次のコードへ進む
```

```
Serial.print("go!");
```

※btnPinはタクトスイッチが繋がれていて、プルアップに設定されている。

問題8-3

1. 次のプログラムの動きを説明して下さい。

```
void setup(){
    pinMode(3, INPUT);    //スイッチのピン設定
    pinMode(6, OUTPUT);   //LEDのピン設定
}

void loop(){
    while(digitalRead(3) == LOW){
        digitalWrite(6, HIGH);
    }
}
```

2. 「analogReadで読み取る値が500以下になるまでLEDが光る」というプログラムを書いて下さい。

※analogReadは3ピンを、digitalWriteは7ピンを使う事とします。

break

ループから脱出します。(loop()では使えない)

```
for(int j = 0; j < 3; j++){ //0と1だけ表示されます。
```

```
    Serial.println(j);
```

```
    if(j==1){
```

```
        break;
```

```
    }}
```

```
for(int i = 0; i < 3; i++){
```

```
    for(int j=0; j < 3; j++){
```

```
        Serial.println(i);
```

```
        Serial.println(j);
```

```
        break;
```

```
    }}
```

このbreakはjの方のループをから脱出するだけでiの方のループは動き続けます。

ので、0・0 1・0 2・0 と出力されます。

問題8-4

選択肢のうち、正しい答えを選んで答えて下さい。

1. void loop()の繰り返しは、breakで脱出できるか。

- ①できる ②できない

2. if内でbreakは使用できるか。

- ①できる ②できない

continue

ループの途中で、処理を一個分飛ばします。（loop()では使えない）

```
for(int i=0;i<3;i++){  
  for(int j=0;j<3;j++){  
    if(0==j){  
      continue;  
    }  
    Serial.println( i );  
    Serial.println( j );  
  }  
}
```

jが0の時だけ i j が表示されません。

問題8-5

選択肢のうち、正しい答えを選んで答えて下さい。

1. void loop()の繰り返しは、continueで飛ばすことができるか。

- ①できる ②できない

2. for文内でcontinueは使用できるか。

- ①できる ②できない

9.Tone関数

tone(pin, frequency)

pin = toneを出力するピン。Tone関数でのみ使用するピンであれば、pinModeで指定する必要はありません。

frequency = 周波数 (Hz) (音の高さ) のことを指す。31Hz以下の周波数は生成できないので注意しましょう。

下の二つの例は、void setupで実行されているため

例:11ピンに接続し、500Hzの音を1秒間だけ鳴らす(void loopは省略)

```
void setup(){
    tone(11,500);    //11ピン、500Hz
    delay(1000);     //1秒間
    noTone(11);      //11ピン
}
```

例:1秒毎に音程が上がっていきます(void loopは省略)

↓ 同じピンのまま並べれば、音が途切れることなく音程が変化していきます。

```
#define SPEAKER 10    //スピーカーは10ピンに接続
```

```
void setup(){
    tone(SPEAKER,500);    //500Hz
    delay(1000);
    tone(SPEAKER,1000);   //1000Hz
    delay(1000);
    tone(SPEAKER,1500);   //1500Hz
    delay(1000);
    noTone(SPEAKER);
}
```

※tone(pin, frequency, duration)

duration = 音を鳴らしている時間を指定する。使い方はtone(pin, frequency)とほぼ同じ。なのであまり使いません。

noTone(pin)

上の例は、void setupの中で実行されているため、一度しか鳴りません。仮にvoid loopで上のプログラムを実行すると、止まることなくずっと鳴り続けます。

しかし、noToneを使えば、loop内でもtone関数を止める事が出来ます。スピーカーのピンを指定して、そのピンから音の出力を止める事が出来ます。また、delayを使って止める時間を設定する(一時停止)事も可能です。

pin = 止めるスピーカーのピンを指定。

例: 1 秒間隔で音が鳴るプログラム

```
void loop(){  
    tone(11,1000);    //11ピンで1000Hzを鳴らす  
    delay(1000);      //1000ms=1秒間鳴らす  
    noTone(11);        //11ピンを止める  
    delay(1000);      //1 秒間待機(何もしない)  
}
```

問題9

- 1.tone関数で周波数の出力を指定した時間だけ止める方法を2通り答えなさい
- 2.tone関数を使って800Hzの周波数を5秒間出力するプログラムを書いてみましょう
今回は4ピンを使うことにします

10.データ型

boolean(bool)

bool型は0か1 falseかtrueを取る。

しかし、0かそれ以外で判別しているので、boolに100とか投げても1つまりtrueをとる。

```
int a=3;
int b=3;
bool button=a==b;
if(button){
    Serial.print("OK");
}
```

とするとOKが表示されます。

変数buttonはa==bよりtrue(1)なので、OKを表示します。

```
if(a==b){
    Serial.print("OK");
}
```

とやってることは変わりませんね。

byte

byte型は0から255までの8bitの数値を格納します。符号無し of データ型で、これは負の数値は扱えないという意味です。この関数の中にintとfloatがあります。

今回この『初級編』ではintの説明をします。

int

2バイト(=8ビット)を使い、-32768~32767の整数を記憶します。

変数がその型の最大値(int型: 32767)を超えると、一回りし、その型の最小値になり、これを「オーバーフロー」といいます。

<オーバーフローが発生するプログラム例>

```
int x;
x = -32768;
x = x - 1; // このときxは32767となる。
x = 32767;
x = x + 1; // このときのxは-32768。反対方向にひとまわり
```

詳しくは「ビット演算子」を読んで頂きたいが、int型は最上位のビット、つまり符号ビットが0であれば正の数、1であれば負の数とする、2の補数という方法を使っています。よって、ビットシフト演算の際、符号拡張に関する問題等が発生する。注意が必要です。

unsigned int (符号なし整数型)

2バイトを使い、0~65535の整数を記憶する。符号ビットは無い。unsigned intと機能や使い方は同じです。

long (long整数型)

unsigned long (符号なしlong整数型)

32ビット (4バイト) の値を格納する変数。つまり、(2の32乗 -1)、0~4294967296の値を格納できるという事です。

主には、millis()の値を格納するのに使用する。使用例は、millis()の項目を参照すると良いです。

void

void型は関数の定義にのみ使い、変数の宣言には使えない。

関数の定義の際戻り値の型を指定するが、戻り値がない場合は指定する必要がないのでその時に使います。

```
void setup() {  
}
```

→戻り値はありません。

一般的なArduinoプログラミングにおいて、voidキーワードは関数の定義にだけ使われます。

問題10

1.byte型の変数Xの値を255とした上でX+1の計算をしてみよう

2.選択肢のうち、正しいものを選んでください

ア.boolは0か1以外の値を取る イ.byteは0から255までの値を格納する

ウ.void型は関数の定義や変数の宣言に使える

11.Stringクラス

String()

Stringクラスのインスタンスを作成します。いきなりクラスとかインスタンスとかって言われてもどうゆうことだよっと思うかもしれません。簡単に説明するとクラスは"設計図"でインスタンスは"設計図をもとに作った物"ということです。詳しくはテキスト上級編にて解説します。

初級編では簡単な使い方のみ説明します。

構文：`String(val);`
`String(val,base);`

val: 文字列に変換される値。従来型の文字列以外にchar,byte,int,long…などに対応

base(オプション): 基数 (例えばHEXと入れた場合、valの数字が16進数の文字列になる)

例：`String a = String(13);` //13という文字列がインスタンスに与えられます
`String b = String(13,HEX);` //16進数を指定。"D"という文字列が与えられます
`String c = String(13,BIN);` //2進数を指定。"1011"という文字列が与えられます

(補足説明)

Stringの()の中に文字や数字を入れてセミコロンで囲むと文字列を表示することができます。

例：`String a = String("Hello");`

問題11

1.次のプログラムは何を表しているか、答えて下さい。

`String a = String(3, BIN);`

2.「110」を表すStringクラスを書いて下さい。変数名はなんでも構いません。

12.変数の応用

変数のスコープという概念

グローバル(大域的)とローカル(局所的)という2つの領域にわけられる考え方の事である。グローバルというのは、プログラムにおける、`setup(){}`や`loop(){}` などの関数の外のことである。

右の画像で黄色い部分がグローバル、水色の部分がローカルである。

変数の利用できる場所

- ・グローバルで宣言された変数：プログラム内のどこでも使えます
- ・関数内やfor文の中で宣言された変数：それぞれの関数、ループ内でのみ使えます
- ・関数内やfor文の中の、さらに関数内やfor文の中で宣言された変数：それぞれの関数、ループ内でのみ使える。

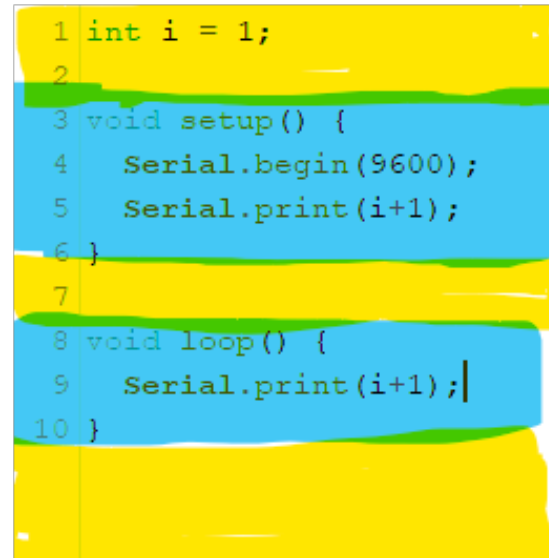
例

`int` gPWMval; // すべての関数から見える変数

```
void setup(){  
  // ...  
}
```

```
void loop(){  
  int i;  // loop関数の中でだけ見える  
  float f; // loop関数の中でだけ見える
```

```
  for (int j = 0; j <100; j++){  
    // 変数jはこの波カッコ内でだけアクセス可能  
  }  
}
```



```
1 int i = 1;  
2  
3 void setup() {  
4   Serial.begin(9600);  
5   Serial.print(i+1);  
6 }  
7  
8 void loop() {  
9   Serial.print(i+1);  
10 }
```

問題12

1.次のプログラムで、変数iが利用できるのはどこか、選択肢から正しい答えを選んで下さい。

- ①全体 ②loop()の中のみ ③setup()の中のみ

```
int i;  
void loop(){  
    //...  
}  
  
void loop(){  
    //...  
}
```

2.次のプログラムで、変数jが利用できるのはどこか、選択肢から正しい答えを選んで下さい。

- ①全体 ②loop()の中 ③for文の処理部分のみ

```
int i;  
void setup(){  
    //...  
}  
  
void loop(){  
    for(int j=0; j<10; j++){  
        //処理  
    }  
}
```

13.関数

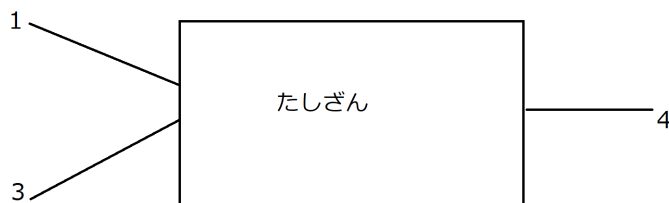
戻り値・引数

関数

<関数 3分クッキング>

今日は関数を作っていこうと思います。

What's 関数？



こんな感じで「たしざん」っていう関数に1と3を投げ入れると4が戻ってきます。

これを実装してみます。
1と3を引いて、4を戻してます。

```
int add(int a,int b){  
    return a+b;  
}
```

`int add` ←int型の結果を戻す
`add(int a,int b)` ←int型の2つを受け取る
`return a+b` ←a+bを返す

関数は何かを受け取って、処理をして、結果を返します。

return

returnを実行すると、プログラムの実行はそこで打ち切れ、呼び出し元の関数に戻ります。

<プログラム例>

```
loop() {  
  bool checkButton() {  
    if (digitalRead(buttonPin) == 0) {  
      return 1;  
    } else {  
      return 0;  
    }  
  }  
}
```

上のプログラム例では、センサーの値を読み取ったらreturn 0;かreturn 1;が実行されるため、checkButton()という関数を抜け出し、loop()の処理に戻ります。「return」の後ろの数字は「戻り値」というもので、呼び出し元に値を返すのですが、これは関数に関する項目を参照。returnの性質を利用してコメントアウトの代わりにできます。

<プログラム例>

```
void loop(){  
  // 完成したプログラムはここに  
  return;  
  // 未完成のプログラムはここ(実行されない)  
}
```

また、setup()内でreturn;を実行すると、setup()の処理が最初から行われる。これは、setup()も見えないさらに深層の関数に呼び出されているためです。

問題13

以下の4つの動作をするためのLED制御関数を作ってください。今回は3ピンを使う事とします。

- ・ オン (点灯)
- ・ オフ (消灯)
- ・ フェードイン (だんだん明るくなる)
- ・ フェードアウト (だんだん暗くなる)

(余裕がある人は以下の2つを追加で作ってください)

ヒント：既に作った上の関数を使えば簡単に作れます

- ・ ウェーブ (だんだん明るくなったり暗くなったりを繰り返す)
- ・ 点滅 (明かりが点いたり消えたりを繰り返す)

おまけ

複合演算子

++ (加算) -- (減算)

変数に対して1を加算(++）・減算(--)します。

例： `a++;` //変数aに1を足す
`a--;` //変数aから1を引く

※

`a++`、`a+=1`、`a=a+1`
これらはすべて同じ意味である

`a--`、`a-=1`、`a=a-1`
これらも全て同じ意味である

ただし、
`a++` と `++a`、`a--` と `--a`は違う。

```
int a=2;  
y=a++; //yにaを代入した後にaを++  
よって、aは3 yは2
```

```
int a=2;  
y=++a; //yにaを代入する前にaを++してる。  
よって、aは3 yは3
```

`+=` `-=` `*=` `/=`

演算(+*/)と代入(=)をまとめたものです

例：
`a+=2;` //`a=a+2`と同じ式
`a-=2;` //`a=a-2`と同じ式
`a*=2;` //`a=a*2`と同じ式
`a/=2;` //`a=a/2`と同じ式
`a%=2;` //`a=a%2`と同じ式

&= (AND)

変数の特定のビットを0にしたい時によく使う演算子です。

構文：`a&=b` `//a=a&bと同じ式`

例：`&=`を使ったマスキングの例

```
byte a=B10110101;  
a &=B11111100; //このB11111100がマスクパターン  
Serial.println(a,BIN); //結果はB10110100
```

|= (OR)

変数の特定のビットを「セット」(1にすること)したい時によく使います。

構文：`x |= y;` `//x|y;と同じ`

x: 変数

y: 整数型定数または変数

例：`byte myByte = B10101010;`
`myByte |= B0000011;` `//結果はB10101011`