# プログラミング実習(2)

Arduino互換マイコンを使ってプログラミングの基礎を学ぶ

② Arduino プログラミングの基礎

シリアル通信・デジタルIO(2)-入力・条件分岐

# シリアル通信

プログラムの出力を調べる

https://www.arduino.cc/reference/en/language/functions/communication/serial/

# シリアル通信を使う①

次のような関数を使うことで、Arduinoの出力をPCで表示することができます。

setup()関数の定義内で次のように初期化関数を呼び出す。

```
Serial.begin(9600);    // 通信速度を 9600bps にする。
```

通信速度がPC側と会わない場合には正しく表示されません。
調整方法は後で説明します。

必要に応じて、

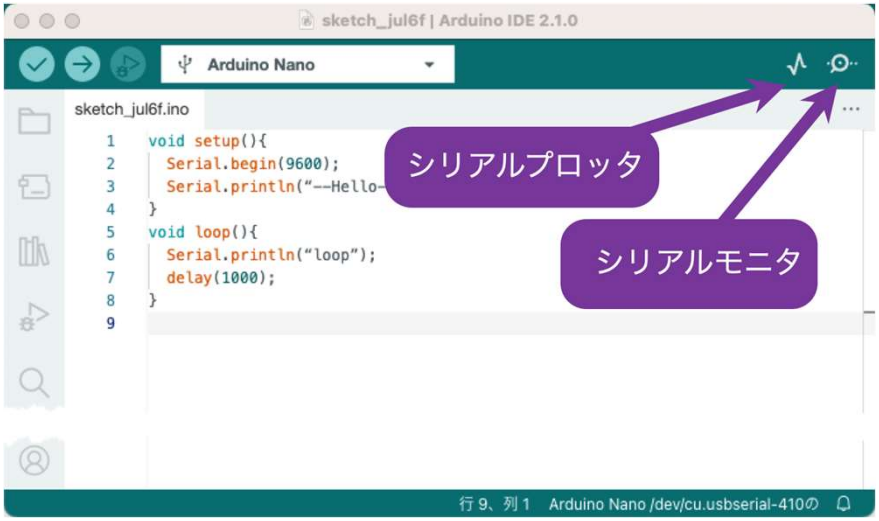
```
Serial.print(値);
Serial.println(値);  // 値を出力した後、改行する。
```

「Serial」が大文字から始まることに注意してください。

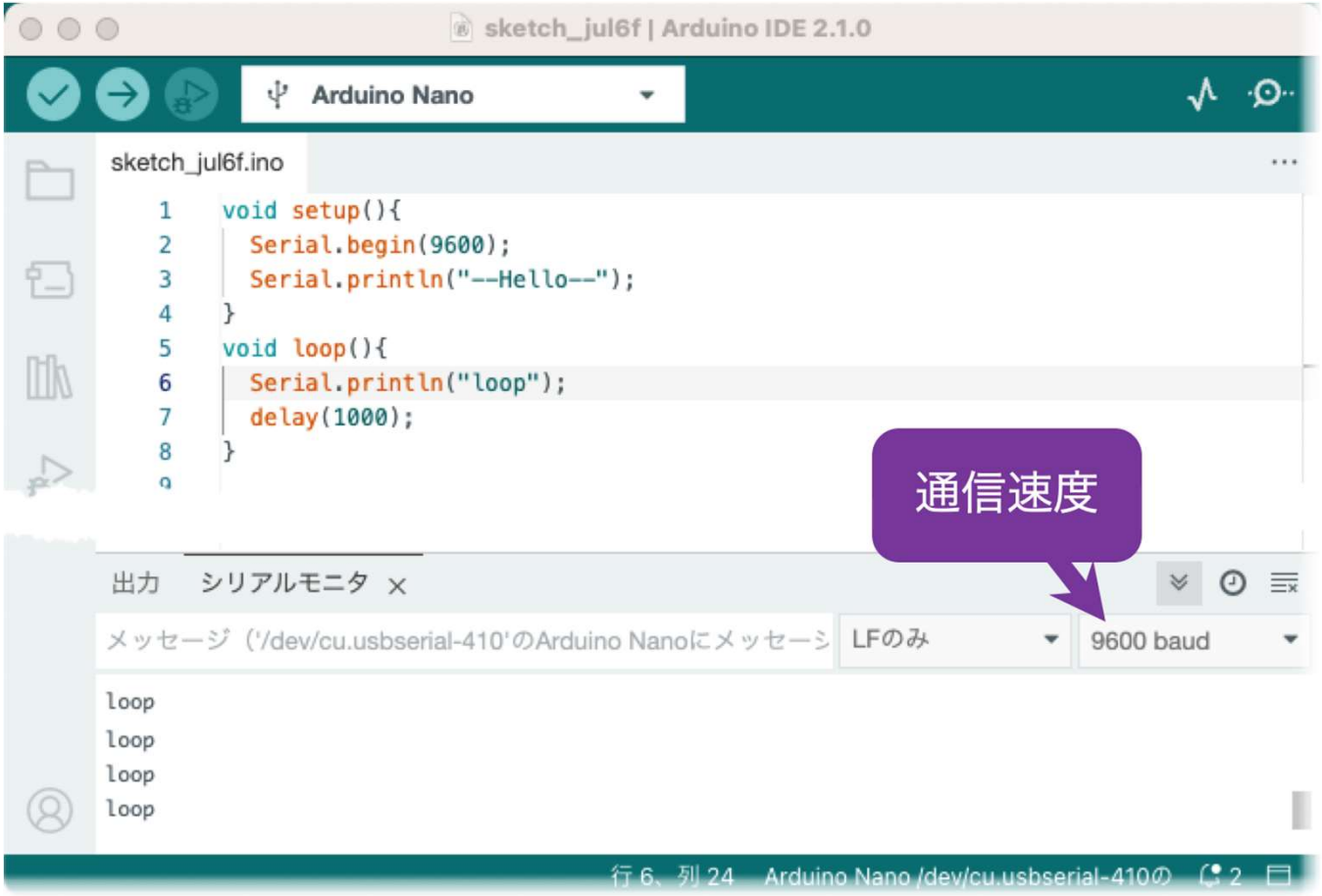
# シリアル通信② シリアル通信のプログラム例

```
void setup(){
      Serial.begin(9600);
      Serial.println("--Hello--");
}
void loop(){
      Serial.println("loop");
      delay(1000);
}
```

プログラムをマイコンに書き込んだら、右のシリアルモニタボタンをクリックしてください。



# シリアル通信を使う③




うまく表示されない場合には
通信速度をプログラムに合わせて
変更してください。

“—Hello—”の出力から確認する
には、マイコンのRESETボタン
を押してください。

# デジタルIO

# デジタル出力の使用例

```
void setup(){
     pinMode(13, OUTPUT);
}

void loop(){
     digitalWrite(13, HIGH);
     delay(500);
     digitalWrite(13, LOW);
     delay(500);
}
```

前の資料の最後のスケッチ例

出力に digitalWrite()を使いました。

入力はどのように書くのかを見ていきます。

# デジタル入力 digitalRead() の使用例

指定したデジタルピンに、HIGHかLOWを出力する。

文法

**digitalRead(pin)**

パラメータ

**pin** ： 設定を行うArduinoのピン番号

戻り値

HIGH または LOW

https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

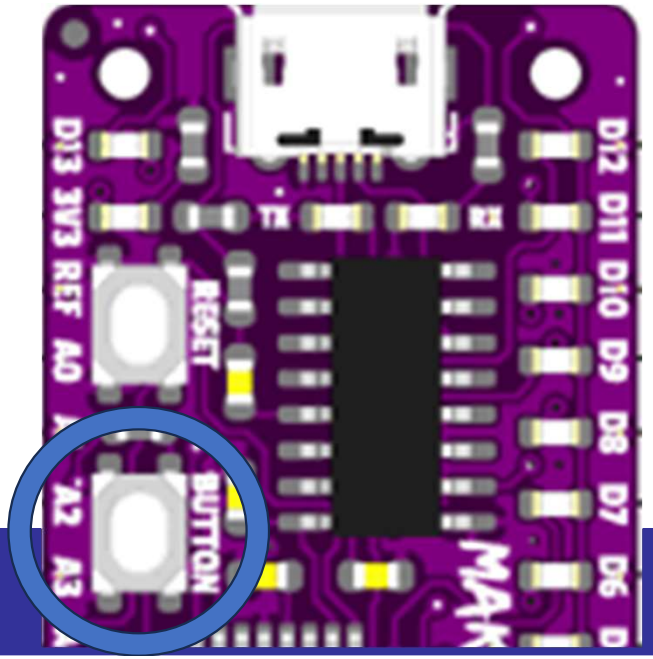
# digitalReadで読み取った値をシリアルポート出力する

```
void setup(){
    Serial.begin(9600);
    pinMode(2, INPUT_PULLUP);
}
void loop(){
    Serial.println(digitalRead(2));
}
```

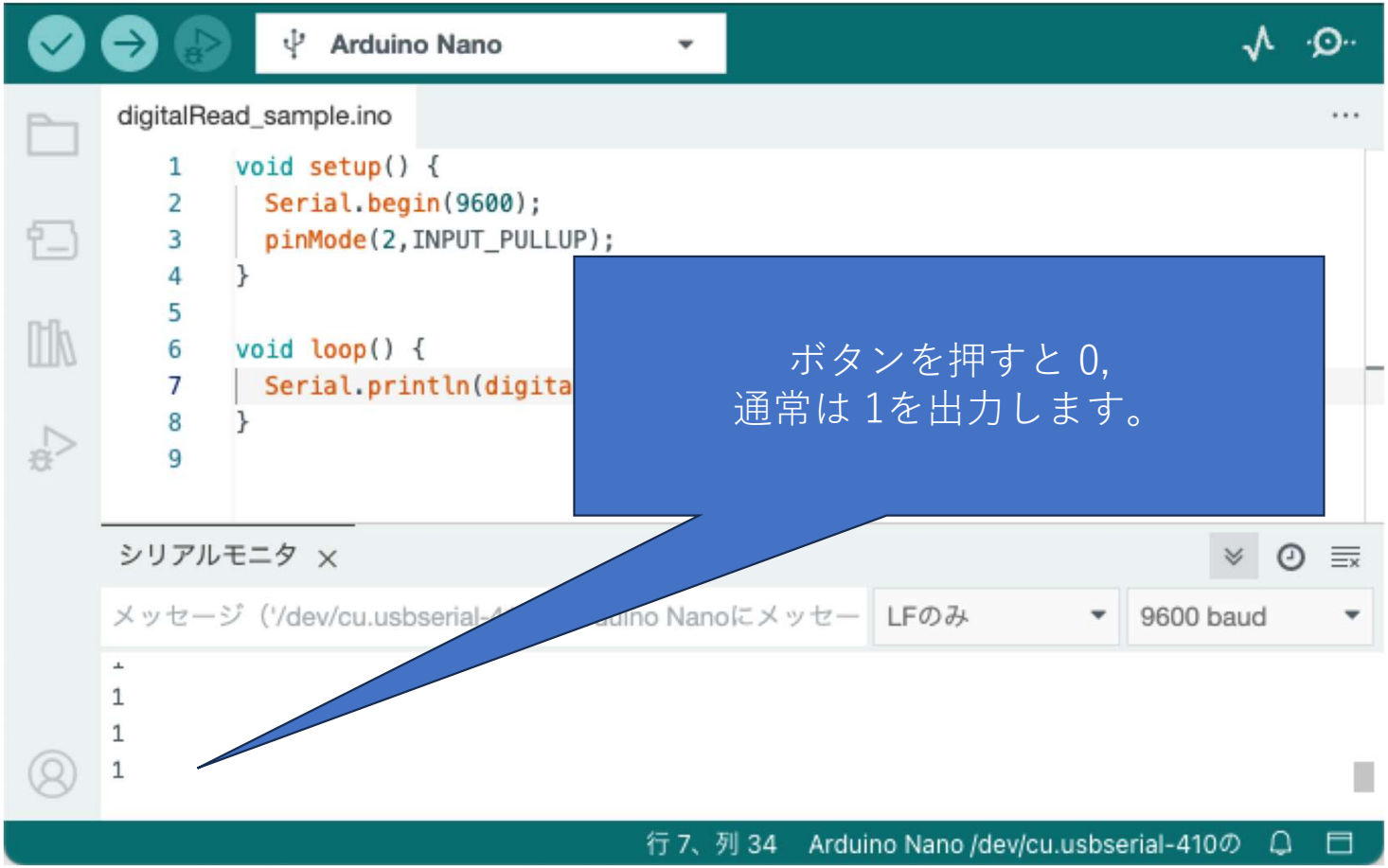
マイコンにプログラムを書き込んだら、シリアルモニタを開いて、どんな出力が得られるかみてください。

ピン2(D2)は、BUTTONに繋がっています。

このプログラムはボタンの状態をシリアルポートに出力します。

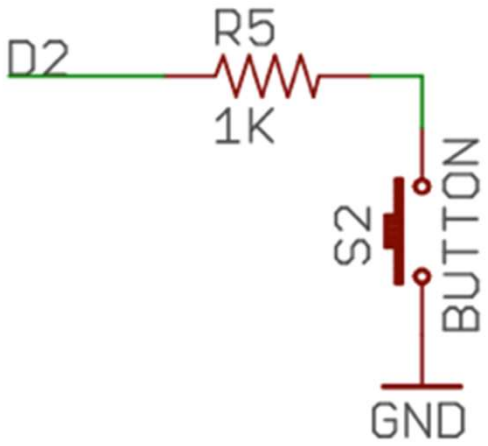


# digitalReadで読み取った値をシリアルポート出力する



# (参考)digitalReadで読み取った値をシリアルポート出力する

MakerNanoのD2に繋がっているボタンは、以下のような回路になっています。




ボタンを押すと GND(0V)に繋がります。

pinMode(2, INPUT_PULLUP)

とすることで、マイコンの内部にある回路で
入力がないときの状態を5Vに引き上げます。

pinMode(2, INPUT)
だと、ボタンおさないと入力がない状態になるので
ボタンの状態にかかわらず 0 となってしまうようです。

# 変数を使って、読み込んだ値を保存(?)する

digitalRead()の戻り値は、0か1の整数です。

前のプログラムでは、Serial.println()の引き数として使い、戻り値を直接シリアルポートに出力しました。値をとっておかなかったので、そのときのボタンの状態を、プログラムの別の場所で使うことができません。

値をとっておくには「変数」を使います。

「変数」は、いろいろな値を撮っておくための「入れ物」です。

「変数」には「データ型」があり、プログラムで変数を使う際に、どんな値を入れられるかをあわせて指定します。

# 整数型 int

- 整数値を扱う最も一般的なデータ型は int です。
- int の名前の由来は英語 "integer"(=整数)です。

- Arduinoでは、 - 32768 ～ 32767 の範囲の整数値を扱えます。

- 基本的な使い方

```
int a;      // int変数 aを使えるようにする(変数の宣言)
a = 1;      // aに 1を入れる。
```

・初期値を指定する。

```
int b = 0;  // 宣言のときに最初の値も指定する
```

# ボタンの状態を読み取るプログラムで変数を使う

```
void setup(){
      Serial.begin(9600);
      pinMode(2, INPUT_PULLUP);
}
void loop(){
     int v;
     v = analogRead(2);
     Serial.println(v);
}
```

# ボタンの状態を読み取るプログラムで変数を使う

変数は、プログラムの冒頭でも宣言することができます。

```
int b = 2;
void setup(){
      Serial.begin(9600);
      pinMode(b, INPUT_PULLUP);
}
void loop(){
    int v;
    v = analogRead(b);
    Serial.println(v);
}
```

# ボタンの状態を読み取るプログラムで変数を使う

変数は、プログラムの冒頭でも宣言することができます。

```
int b = 2;
void setup(){
      Serial.begin(9600);
      pinMode(b, INPUT_PULLUP);
}
void loop(){
    int v;
    v = analogRead(b);
    Serial.println(v);
}
```

変数の名前は、大文字小文字の区別があります。

変数の名前は、英字から始まる英数字列で'_'などの一部の記号を含めることができます。

左のプログラムでは b, v など1文字なので入力は楽ですがあとで、プログラムを見たときに意味が分からなくなることがあります。

b を button_pin
v を button_state

などと、もっと意味がはっきりするように書くとあとでプログラムを見直すときに便利です。

# ボタンの状態を読み取るプログラムで変数を使う

長い名前で意味をはっきりさせた例です。

```
int b = 2;
void setup(){
      Serial.begin(9600);
      pinMode(b, INPUT_PULLUP);
}
void loop(){
    int v;
    v = analogRead(b);
    Serial.println(v);
}
```

```
int button_pin = 2;
void setup(){
      Serial.begin(9600);
      pinMode(button_pin, INPUT_PULLUP);
}
void loop(){
    int button_state;
    button_state = analogRead(button_pin);
    Serial.println(button_state);
}
```

# ボタンの状態を読み取るプログラムで変数を使う

次のように、button_stateの宣言と、初期化を1行で行うこともできます。

```
int button_pin = 2;
void setup(){
      Serial.begin(9600);
      pinMode(button_pin, INPUT_PULLUP);
}
void loop(){
    int button_state = analogRead(button_pin);
    Serial.println(button_state);
}
```

# データ型

| データ型 | 値 | |
|---|---|---|
| boolean | true または false | 論理型 |
| char | -128 ～ 127 | 英文字1文字分を表すのにも用いる |
| int | -327,68 ～ 32,767 | 16ビット 整数 |
| float | -3.4028235E+38<br>～ 3.4028235E + 38 | 浮動小数点数 |

他にも、 符号なしの整数などの型がありますが、この教材では使いませんので省略します。

# データ型

| データ型 | 値 | |
|---|---|---|
| boolean | true または false | |
| char | -128 ～ 127 | 英文字1文字分を表すのにも用いる |
| unsigned char | 0 ～ 255 | 符号なしの char |
| byte | 0 ～ 255 | |
| int | -327,68 ～ 32,767 | 16ビット 整数 |
| unsigned int | 0 ～ 65,535 | 符号無し16ビット整数 |
| word | 0 ～ 65,535 | |
| long | -2,147,483,648 ～ 2,147,483,647 | 32ビット整数 |
| unsigned long | 0 ～ 4,294,967,295 | 符号無し 32ビット整数 |
| float | -3.4028235E+38 ～ 3.4028235E + 38 | 浮動小数点数 |
| double | -3.4028235E+38 ～ 3.4028235E + 38 | 通常倍精度だけど、 floatと同じ? |

# 変数への値の代入

変数に値をセットするには 次のように、＝ を使います。
今回のプログラムでは、以下のように使われました。

```
button_state = analogRead(button_pin);
```

この ＝ は、数学の等号とは違います。使い方に向きがあり、

左辺は必ず「変数」、すなわち値の入れ物が置かれます。
右側の式を計算・処理し、その結果を「変数」に入れます。

左右を逆にすることはできません。

https://www.arduino.cc/reference/en/language/structure/arithmetic-operators/assignment/

# ボタンでLEDを点灯させる

```
int button_pin = 2;
int led_pin = 13;
void setup(){
       Serial.begin(9600);
       pinMode(button_pin, INPUT_PULLUP);
       pinMode(led_pin, OUTPUT);
}
void loop(){
    int button_state = analogRead(button_pin);
    Serial.println(button_state);
    if(button_state == LOW) {
         digitalWrite(led_pin, HIGH);
    }else{
        digitalWrite(led_pin, LOW);
    }
}
```

D2に繋がっているボタンは
通常の状態は HIGH
押したときに LOW
になります。

ボタンがおされていたら
D13 を HIGHに
推されていなかったら
D13 を LOWに
します。

# 変数の使える範囲

プログラムの冒頭で宣言された変数は、プログラムのどこからでも使えます。

それに対して、関数などの中で宣言された変数は、その関数の中でしか使うことができません。

# ボタンでLEDを点灯させる（誤りを含むプログラム）

```
void setup(){
    int button_pin = 2;
    int led_pin = 13;
    Serial.begin(9600);
    pinMode(button_pin, INPUT_PULLUP);
    pinMode(led_pin, OUTPUT);
}
void loop(){
    int button_state = analogRead(button_pin);
    Serial.println(button_state);
    if(button_state == LOW) {
        digitalWrite(led_pin, HIGH);
    }else{
        digitalWrite(led_pin, LOW);
    }
}
```

変数の宣言を setup()の内側で
おこないました。
このため これらの変数はsetup()内でしか
使えません。

setup()内ではエラーになりませんが、

loop()内では button_pinも
led_pinも変数が定義されていないという
エラーになります。

# 条件分岐 (if – else)

コンピュータは、プログラムに書かれた順番に実行しますが、なにかの計算結果や、外部からの入力によって、動作を変えたい場合があります。このときに使うのが、if文で、次のように使います。

```
if (条件式) {
    条件式が真のときに実行する部分
}
```

```
if (条件式) {
    条件式が真のときに実行する部分
}else{
    条件式が偽のときに実行する部分
}
```

# 条件分岐 (if – else)

コンピュータは、プログラムに書かれた順番に実行しますが、なにかの計算結果や、外部からの入力によって、動作を変えたい場合があります。このときに使うのが、if文で、次のように使います。

```
if (条件式) {
    条件式が真のときに実行する部分
}
```

# 条件分岐 (if – else)

コンピュータは、プログラムに書かれた順番に実行しますが、なにかの計算結果や、外部からの入力によって、動作を変えたい場合があります。このときに使うのが、if文で、次のように使います。

```
if (条件式) {
    条件式が真のときに実行する部分
}else{
    条件式が偽のときに実行する部分
}
```

# 条件分岐 (if – else if - else)

コンピュータは、プログラムに書かれた順番に実行しますが、なにかの計算結果や、外部からの入力によって、動作を変えたい場合があります。このときに使うのが、if文で、次のように使います。

```
if (条件式) {
    条件式が真のときに実行する部分
}else if(条件式2){
    条件式2が真のときに実行する部分
}else{
    条件式,条件式2のどちらも偽のときに実行する部分
}
```

# 条件式

| 記述方法 | 意味 | 注意事項 |
| --- | --- | --- |
| x == y | 等しい | ==を2つ並べる　= だけだと別の意味になる |
| x != y | 等しくない | ≠ をあわらすのに！記号を使っている |
| x > y | xがyより大きい | |
| x >= y | xはy以上 | >と= の順番に注意 more or equal 英語の語順と同じ |
| x < y | xがyより小さい | |
| x <=y | xはy以下 | <と= の順番に注意 less or equal 英語の語順と同じ |