IoT
演習資料

はじめに・・・Arduino と類似製品

本家 Arduino と Arduino Nano

本講座でも使用する Arduino は、ハードウェア仕様が公開されているため、格安の互換品が多く出回っています。また、スロット互換の製品も多く登場しています。より小型の製品として、Arduino Nano がありますが、物理形状がまったく異なるため、ハードウェアの互換性はありません（ソフトウェア互換有り）。

ESP32-WROOM-32, ESP32-DevKitC

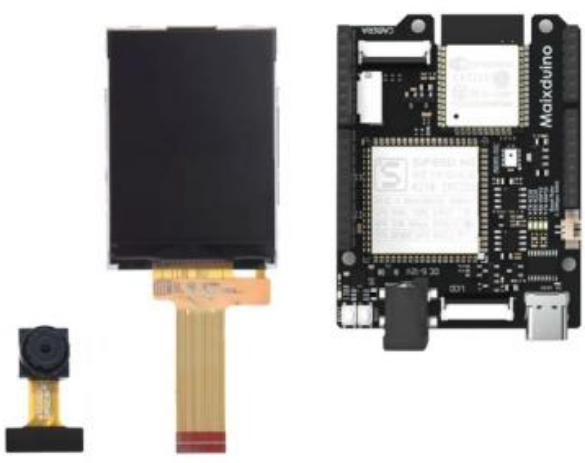
Arduino は通信機能を持たないため、Arduino に Wi-Fi 機能をプラスするために用いられるのが ESP32-WROOM32 です。ただし、ESP32-WROOM-32 は半田付けが必要で、Arduino とも多数の結線が必要になります。そのため、もっと簡単に使用できるようにピンヘッド等を接続したものが ESP32-DevKitC（技適取得済み）です。

ESP32-DevKitC は、これ単体で Wi-Fi や Bluetooth による通信機能を持ったワンチップマイコンとして機能します。Arduino とソフトウェア互換があるため、センサを接続し、ソフトウェア資産をそのまま流用できます。

M5Stack シリーズ

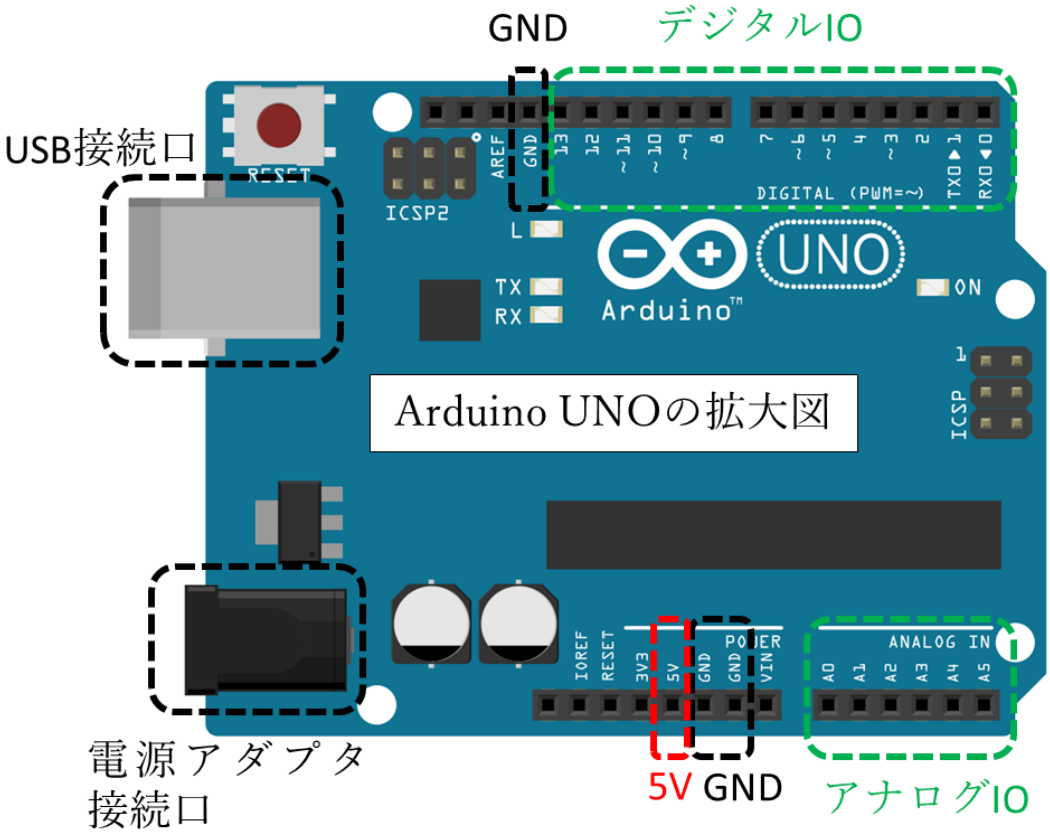
ESP32 に LCD や microSD スロット、スピーカを内蔵し、より便利な 1 台に仕上げたものが M5Stack シリーズです。9 軸センサなどを搭載した上位シリーズもあります。

M5StackV, Sipeed Maixduino



同じ M5Stack シリーズでも、M5StackV は RISC V チップに LCD、 K210(AI チップ)から構成されます。ただし、Wi-Fi は未搭載です。Maixduno は、RISC V、K210、LCD、Wi-Fi、カメラから構成されます。両者は従来の Arduino GUI も使用できますが、基本的に MaixPy IDE という専用統合環境を用います。使用するプログラミング言語は Python です。
K210 は単体で Deep Learning 学習を行うには非力ですが、PC 等で学習した重みデータを使用し、MobileNet や Tiny-YOLO といった推測エンジンを使用できます。

演習１　Arduino を使った電気回路の設計



～のマークあるデジタルピンは PWM　(Plus Width Modulation：パルス幅変調…疑似アナログ：256 段階）が使えるピンを表します。通常、3、5、6、9、10、11 で PWM 出力が行えます。

アナログピンでは 0～1023 の 1024 段階でのアナログ入力ができます。

センサなどの接続の＋側を 5V に、－側を GND に接続していきます。

電気の動きを理解することはなかなか一筋縄ではいきませんが、高いところ（5V）から低いところ（GND）に、（水と全く同じではないですが）水のように移動しているものとイメージすると想像しやすいかもしれません。

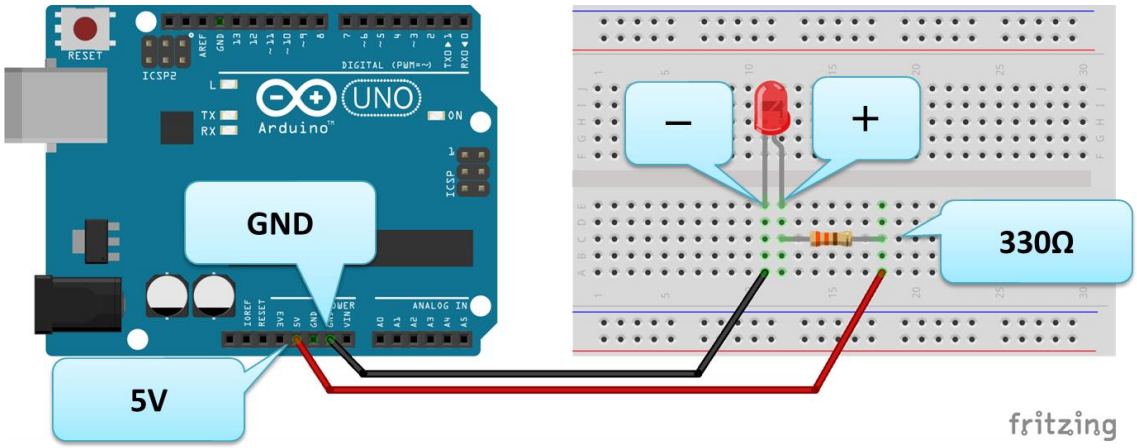
Arduino（×１）とブレッドボード（×１）をベースにして、次の課題の回路を順番に作成していきましょう。

① LED が点灯する回路

以下の部品のうち 330Ω抵抗以外は OSOYOO のセットに含まれています。330Ω抵抗が手元にない場合は OSOYOO のセットに含まれている 200Ω抵抗で代替しましょう。200Ω抵抗の場合、330Ω抵抗のときよりも LED への電力供給が増えるので、200Ω抵抗のときの方が LED が明るく光ります。

・赤色 LDE×１
・330Ω抵抗×１　　・・・330Ωが無い場合 200Ω抵抗でも可
・ジャンパーワイヤー×２　　・・・ワイヤーの外皮の色は何色でも可

下図にしたがって配線をすると自動的に電力が供給されて LED が点灯します。
抵抗を接続しないと LED が焼き切れることがあります。GND 側を先に接続するなど，接続する順番などにも注意しましょう。

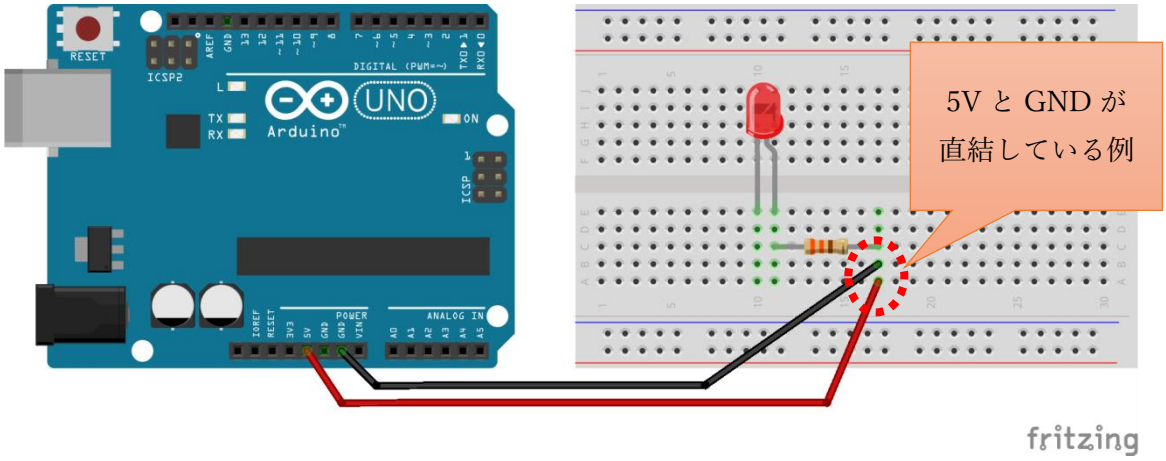


LED を物理的に点灯させる回路

電流が正しく流れない電気回路について

・ショートした回路

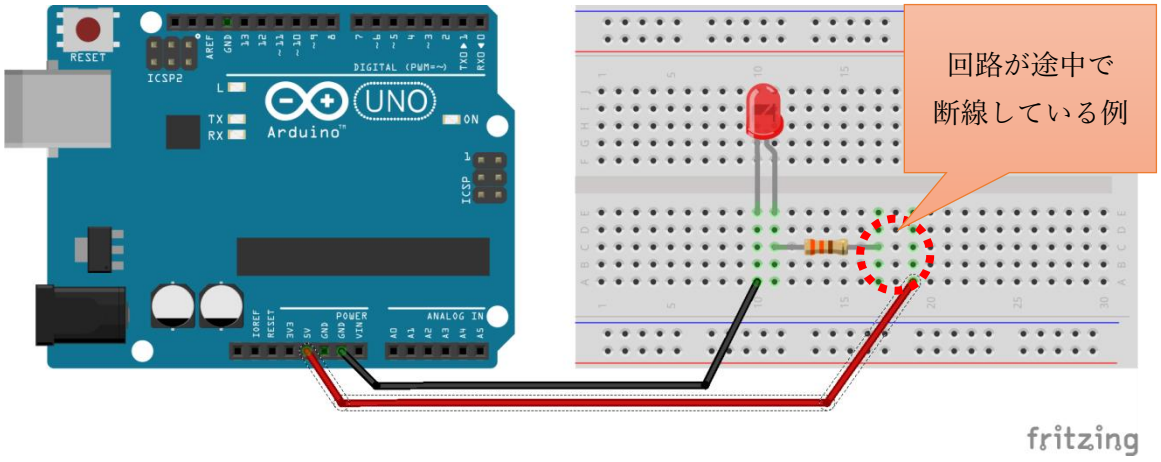
電流が正しい経路を通らずに近道してしまうものをショート（短絡）した回路と呼びます。ショートした回路では電流が流れすぎてしまい、発熱し火事につながることもあるので、このような回路は避けましょう。ショートしたときに電流が流れすぎるのを防ぐために、電流が流れすぎると溶断するヒューズやブレーカーなどをつけたりします。



ショートした回路の例

・オープンした回路

配線がうまくいっておらず回路が断線してしまっているものを「オープンしている」、「開放している」、「断線している」などと呼びます。英語では Open circuit と呼びます。オープンした回路では意図したように電流が正しく流れません。



オープンした回路の例

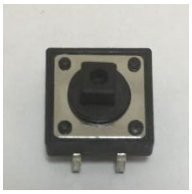
② スイッチで LED を ON・OFF する回路

赤色 LDE × 1

330Ω抵抗 × 1

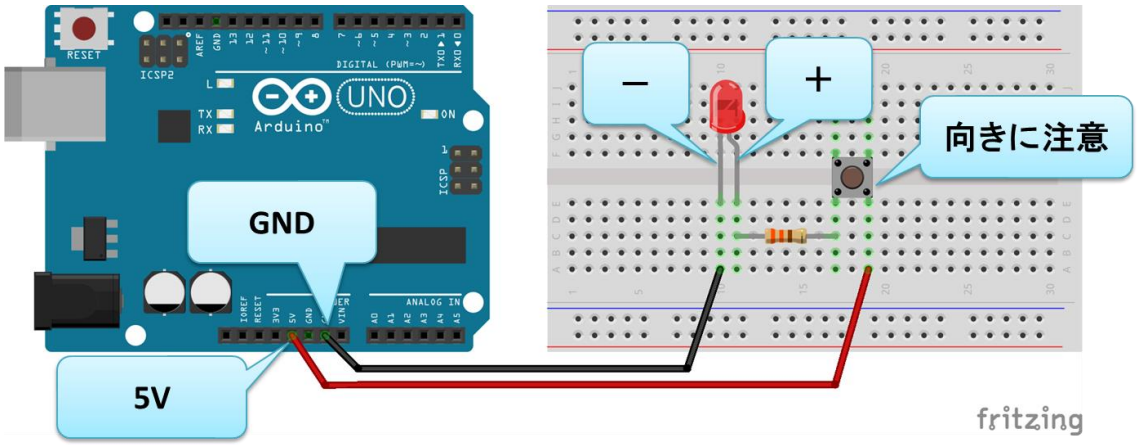
ジャンパーワイヤー × 2

タクトスイッチ × 1

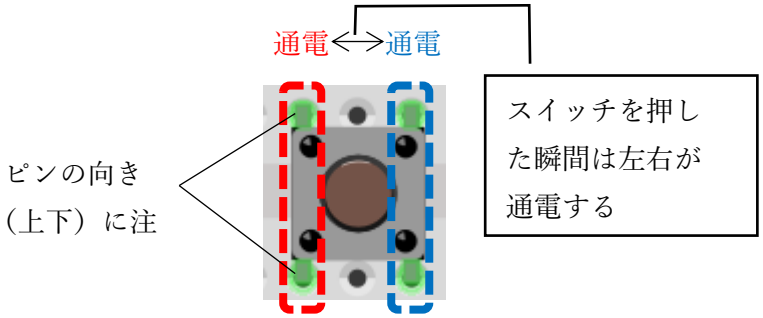


タクトスイッチ

下図のように配線すると、スイッチを押すと物理的に配線が接続されて電源が供給され LED が点灯します。



スイッチで LED を物理的に ON/OFF させる回路



タクトスイッチの通電

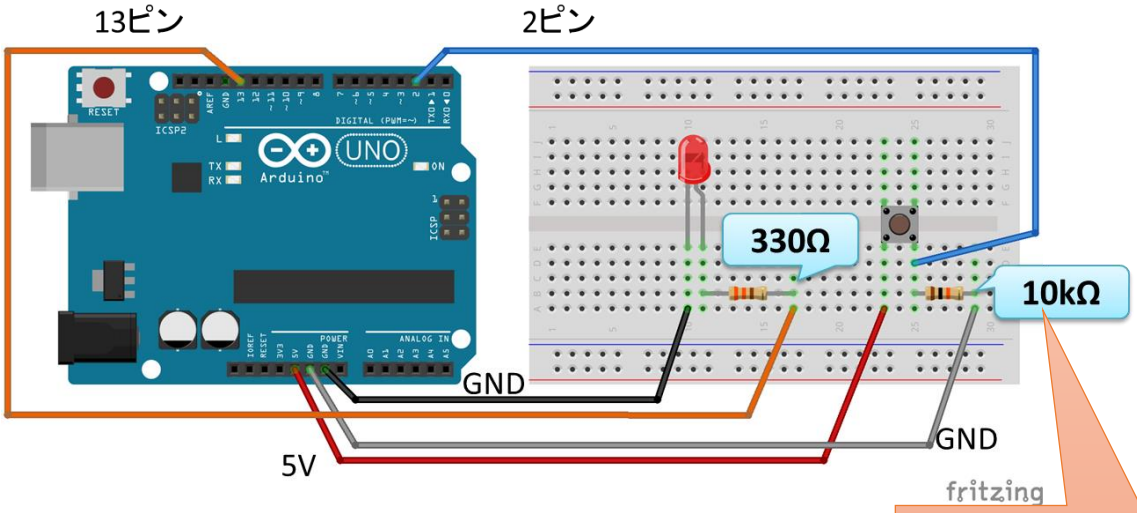
③　スイッチを押したときに LED を ON する回路とプログラム

赤色 LDE × １　　10kΩ抵抗 × １　　330Ω抵抗 × １

ジャンパーワイヤー × ５　　タクトスイッチ × １

Arduino　IDE から Arduino に書き込むプログラムは、Arduino　IDE のメニュー［ファイル］→［スケッチ例］→［Digital］→［Button］から読み込むことができます。プログラムの説明は次のサンプルプログラムのコメントを参考にしてください。

開いたファイルは別名（任意）のファイルに保存しましょう。



スイッチを押したときに LED を ON する回路

スイッチ OFF 時の挙動を安定化させるためのスイッチ～抵抗～GND への配線

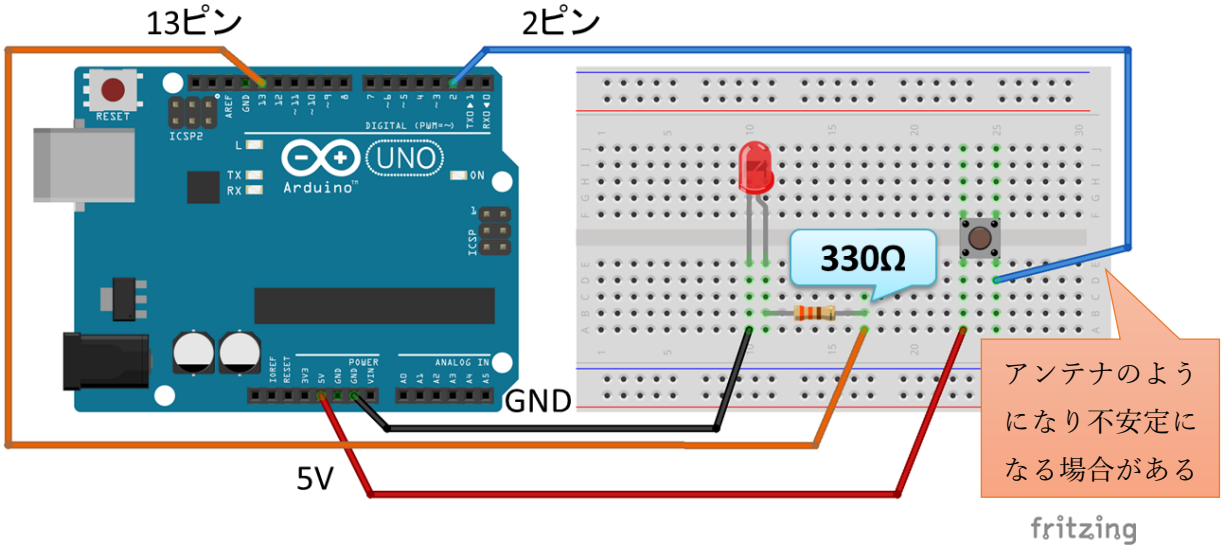
スイッチを押したときに LED を ON するプログラム

```
const int buttonPin = 2;     // ボタンを接続するピン
const int ledPin =  13;      // LED を接続するピン

int buttonState = 0;         // ボタンの状態変数を初期化

void setup() {
  pinMode(ledPin, OUTPUT);  // 13 ピンをアウトプットモードに設定
  pinMode(buttonPin, INPUT);  // 2 ピンをインプットモードに設定
}



void loop() {
  buttonState = digitalRead(buttonPin);  // 2 ピンからデジタルで読み取り
```

```
  if (buttonState == HIGH) {  // 2 ピンからの読み取りが HIGH の場合
    digitalWrite(ledPin, HIGH);  // 13 ピンの LED を点灯
  } else {
    digitalWrite(ledPin, LOW);  // その他の場合、13 ピンの LED を消灯
  }
}
```

※先ほどの回路ではスイッチに抵抗が接続されています。これは回路を安定化させるプルダウンという接続方法になります。

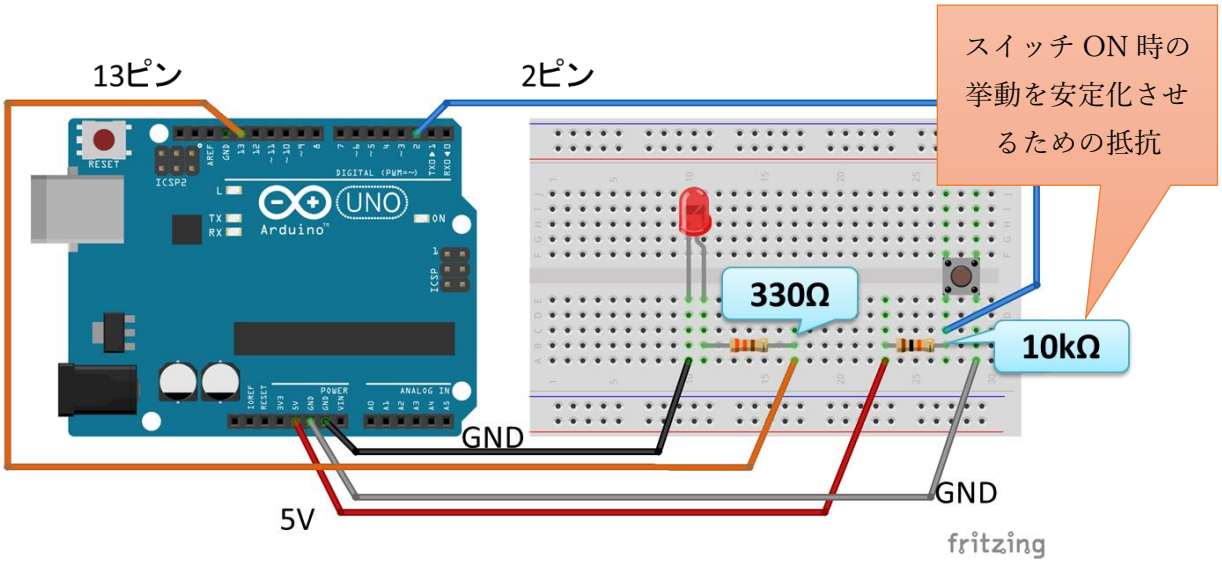
③の図で、もしスイッチに接続されている抵抗～GND の配線が下図のようになかった場合、プログラムが不安定になる可能性があります。③の回路でもし抵抗がない場合、インプットである 2 ピンからジャンパーピンによる配線が始まり、この配線がスイッチまで接続されて終了することになります。この状態では、スイッチが押されていない場合、スイッチから先はどこにも配線されておらず途中で切れている状態です。これは 2 ピンからスイッチまでのジャンパーピンがいわゆるアンテナのような役割をすることがあり、周りの環境によって 2 ピンにノイズが混入しプログラムが誤動作を引き起こす場合があります。



不安定になる回路の例

このような誤動作を防ぐために、③の回路ではスイッチから抵抗を経て GND に接続しています。この方式をプルダウンと呼びます。

プルダウンの方式の他に、プルアップという方式があります。
次の図はプルアップの回路です。この回路に先ほどの③のプログラム（p.9）を書き込んだ場合、ボタンを押したときに LED は消灯し、ボタンを離したときに LED は点灯します。



プルアップ方式の回路の例

プルダウンやプルアップは回路を安定させるためのテクニックです。回路設計のテクニックには、プルダウンとプルアップの他にも、ピンから外部へ電流を流すソース方式（吐き出し方式）や、ピンに向けて電流を流すシンク方式（吸い込み方式）といったように色々な回路設計の手法があります。**ここで、スイッチに接続された GND を外すと回路が不安定になり、場合よっては LED が点灯したままになります。**

Arduino では内部の抵抗（20k～50kΩ）と回路でプルアップ方式を実現できます。p.6 の回路とプログラムで、関数 setup()内の 2 ピンのモード設定の行を以下に変更するとボタンを押したときに OFF になり、ボタンを離したときが ON になるプルアップ方式が実現できます。次の演習課題で実際にやってみましょう。

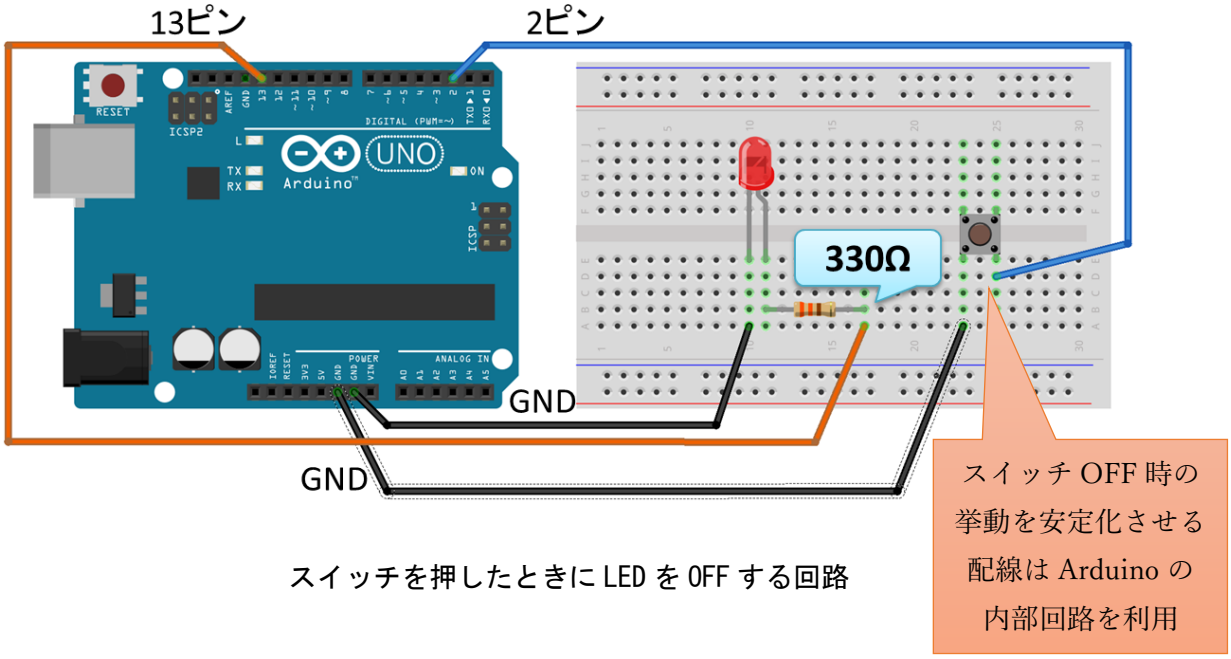
```
pinMode(buttonPin, INPUT_PULLUP);  //ピンモードをプルアップ方式に設定する
```

④　スイッチを押したときに LED を OFF する回路とプログラム

赤色 LDE × 1　　330Ω抵抗 × 1　　ジャンパーワイヤー × 4

タクトスイッチ × 1

Arduino IDE から Arduino に書き込むプログラムは、Arduino IDE のメニュー［ファイル］→［スケッチ例］→［Digital］→［Button］から読み込むことができます。プログラムの説明は次のサンプルプログラムのコメントを参考にしてください。



スイッチを押したときに LED を OFF する回路

スイッチを押したときに LED を OFF するプログラム

```
const int buttonPin = 2;     // ボタンを接続するピン
const int ledPin =  13;      // LED を接続するピン

int buttonState = 0;         // ボタンの状態変数を初期化

void setup() {
  pinMode(ledPin, OUTPUT);  // 13 ピンをアウトプットに設定
  pinMode(buttonPin, INPUT_PULLUP);  //2 ピンをプルアップ方式に設定する
}
```

③のプログラムで
この行のみを修正

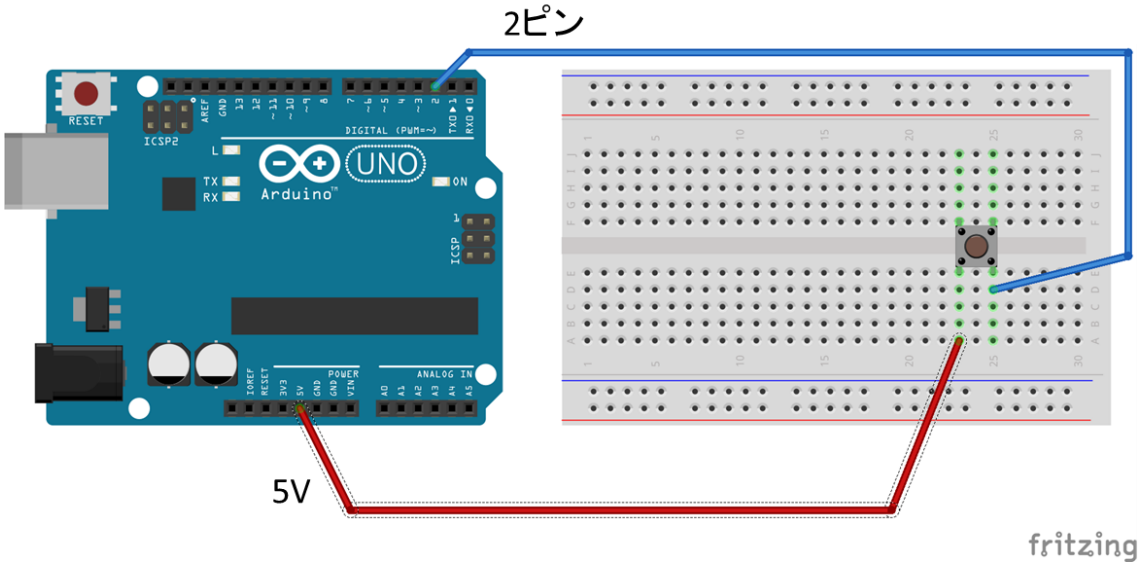
```
void loop() {
  buttonState = digitalRead(buttonPin);  // 2 ピンからデジタルで読み取り

  if (buttonState == HIGH) {  // 2 ピンからの読み取りが HIGH の場合
    digitalWrite(ledPin, HIGH);  // 13 ピンの LED を点灯
  } else {
    digitalWrite(ledPin, LOW);  // その他の場合、13 ピンの LED を消灯
  }
}
```

プルアップとプルダウンの基本的な考え方

１）ダメな回路１（オープンした回路）

「2 ピンをインプット用のピンに設定して High（5V）か Low（0V）を感知したい・・・」と考えて、下図のような回路をつくるのは結論から言うと「ダメ」です。スイッチが ON になっているときは OK ですが、OFF になっているとき、p.5 で紹介したダメな回路であるオープンな回路になってしまっています（特に 2 ピンが不安定になってしまいます）。

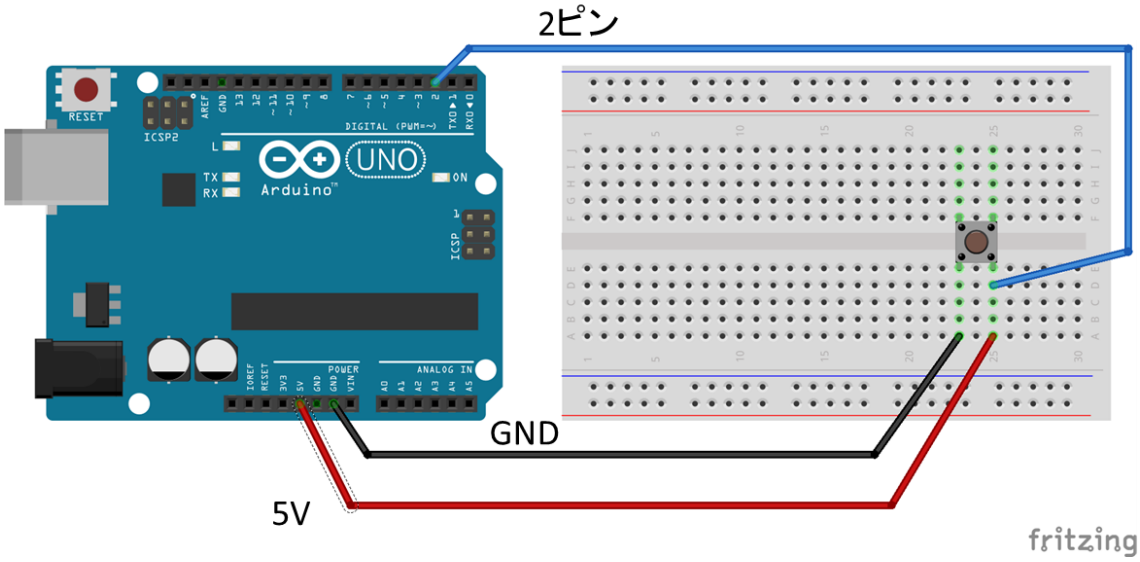
→解決版は３）を参照



オープンした回路

２）ダメな回路２（ショートした回路）

『上の１）でスイッチが OFF のときにオープンな回路になってしまっていた・・・。それではスイッチが OFF のときは GND につなげよう！』と考えて、次の図のような回路をつくるのも結論から言うと「ダメ」です。



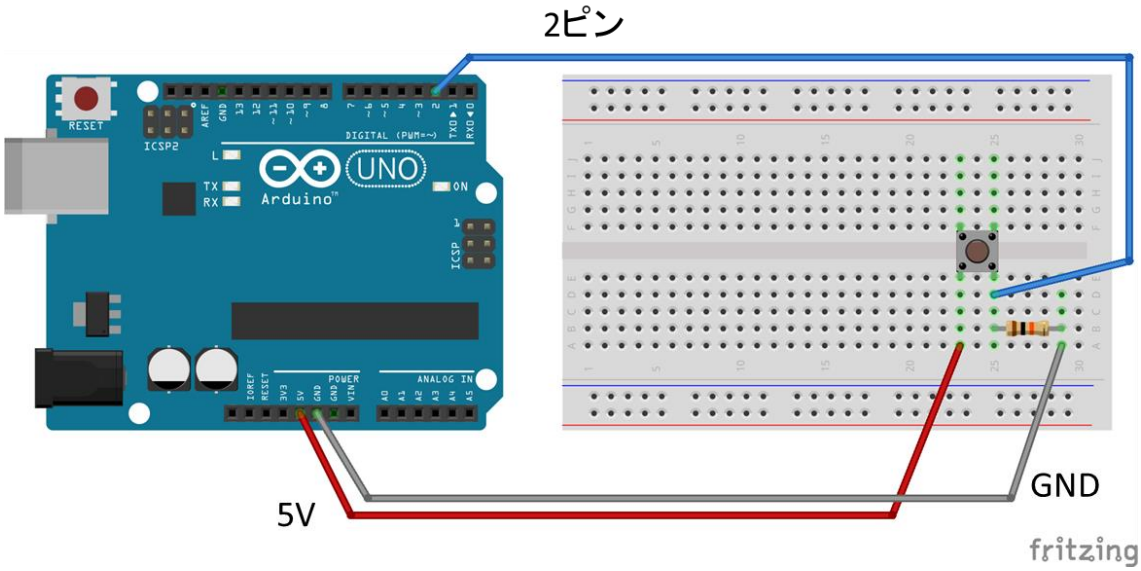
ショートした回路

この回路では、スイッチが ON のときスイッチ側にも 5V の電圧がかかることになります。スイッチは通常、ほとんど抵抗値をもっていません。したがって、オームの法則（V=R×I）を使ってスイッチに流れる電流を計算しようとすると､その計算式は I＝V／R になります。スイッチの抵抗は前述したようにほとんどありません（R≒0）。そうすると前述の計算式の分母の R が 0 に近づくとすると I は無限大に近づいてしまいます。無限大の電流がながれることは実際にはありませんが、それでも非常に大きな電流がスイッチに流れてしまい発熱などの危険があることが分かります。これがショートしたダメな回路になります。

→解決版は４）を参照

３）OK な回路１（プルダウン回路）

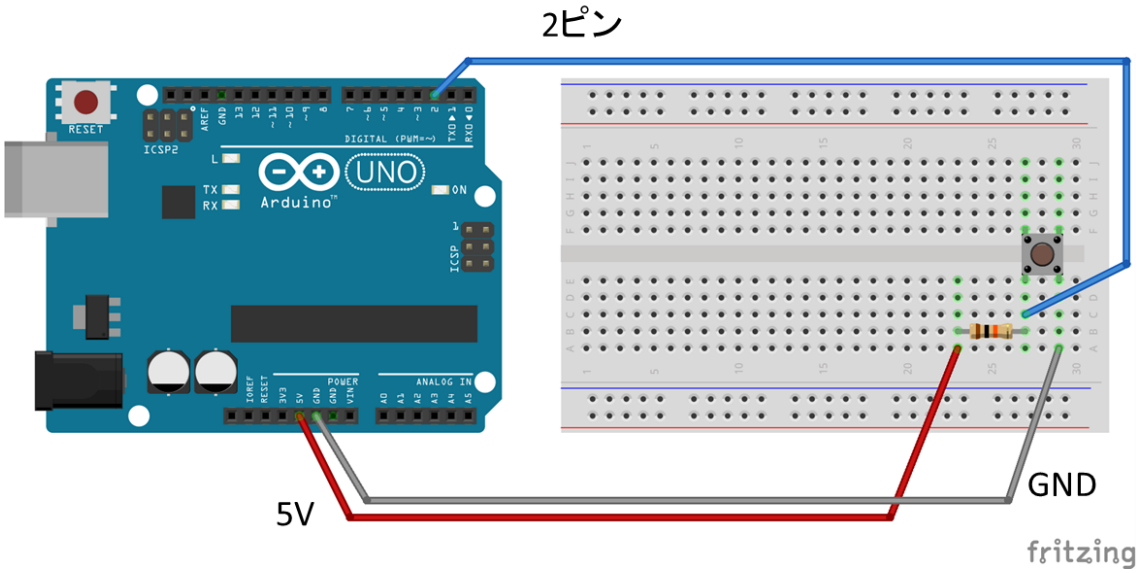
前述の１）ではスイッチが OFF のときオープンな回路になってしまっていました。それでは抵抗もつかってスイッチが OFF のときに GND につなげてやりましょう。その回路が次の図のようになります。この回路では抵抗のおかげでショートすることなく、スイッチが ON のとき 2 ピンには 5V の電圧がかかり High を検出します。逆に、スイッチが OFF のとき 2 ピンは GND につながり Low を検出します。



プルダウン回路の例

４）OK な回路２（プルアップ回路）

前述の２）ではスイッチが ON のとき、スイッチの抵抗値が 0 だったためにショートした回路になってしまっていました。それでは抵抗も使って回路をつくってやりましょう。その回路が以下になります。この回路では抵抗のおかげでショートすることなく、スイッチが ON のとき 2 ピンは GND につながり Low を検出します。逆に、スイッチが OFF のとき 2 ピンには 5V の電圧がかかり High を検出します。



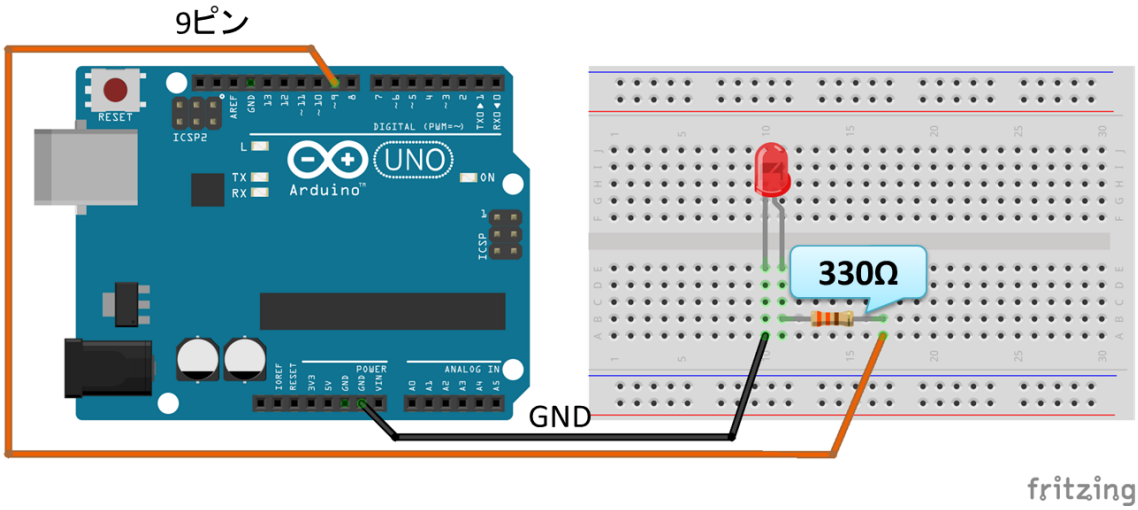
プルアップ回路の例

⑤　PWM を使ったアナログ出力による LED 点灯

赤色 LDE×１　　330Ω抵抗×１　　ジャンパーワイヤー×２

※プログラムは［ファイル］→［スケッチ例］→［Basics］→［Fade］から読み込む

※9 ピンは PWM マーク（～）があるので（疑似的な）アナログな操作が可能なピン



PWM を使ったアナログ出力による LED を自動で点灯させる回路

PWM を使ったアナログ出力による LED を自動で点灯させるプログラム

```
int led = 9;           // LED を接続するピン
int brightness = 0;    // LED の明るさ
int fadeAmount = 5;    // 変化量

void setup() {
  pinMode(led, OUTPUT);  // 9 ピンをアウトプットに設定
}

void loop() {
  analogWrite(led, brightness);  // 9 ピンに LED の明るさの値を設定

  brightness = brightness + fadeAmount;  // LED の明るさを変化量分変える

  if (brightness <= 0 || brightness >= 255) {//LED の明るさの範囲を 0～255 に設
定
```

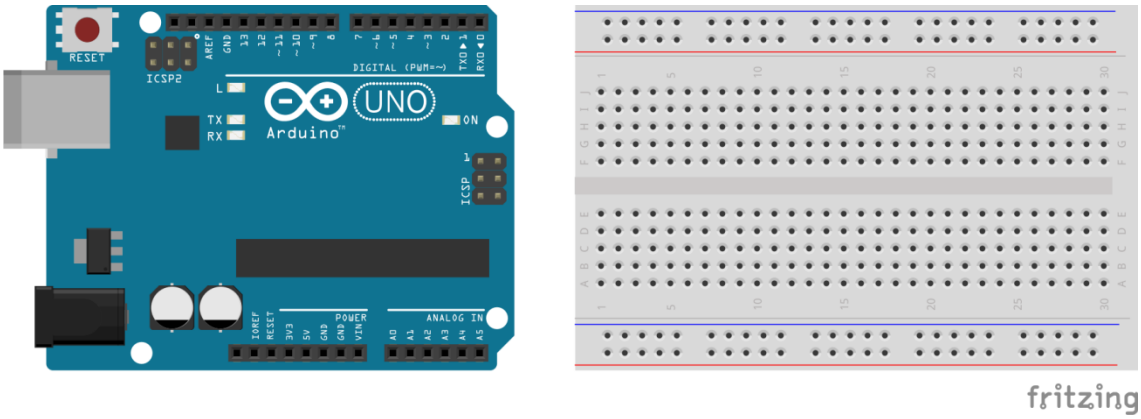
```
    fadeAmount = -fadeAmount;
  }
  delay(30);  // 30 ミリ秒待機
}
```

※delay 関数の引数は msec ですが、unsigned long 型です。32767 より大きい整数を指定するときは、値の後ろに UL をつけます。　例：delay(1000UL * 60 * 5) // 5 分待機

⑥　応用編：LED の種類や個数を変更
　　※作成した回路の回路図を描き抵抗値を書き込む

これまでやってきた①～③をもとに LED の種類を変えたり個数を変えたりして回路を作ってみましょう。以下の中から課題を選んで回路を作成し、回路図を描いた後に、それぞれの抵抗の値を計算して書き込んでみましょう。

A）LED の色を変えて適切な抵抗を選びましょう。
B）LED の個数を直列接続もしくは並列接続で増やして適切な抵抗を選びましょう。
C）LED の個数を直列接続と並列接続の 2 種類で増やして適切な抵抗を選びましょう。



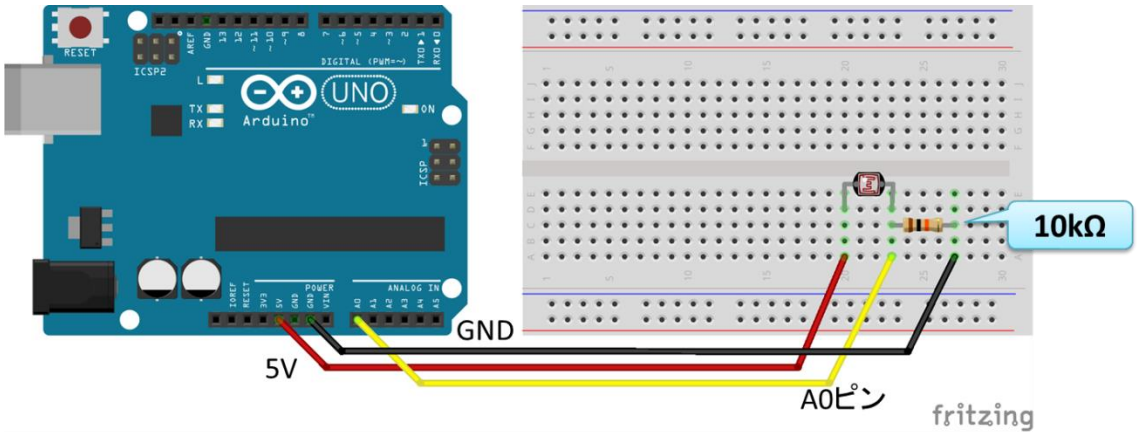
演習２　Arduino とセンサを使った回路設計

光センサ（アナログ

以下の回路を順番に作成していきましょう。

環境センサ
①光センサの利用
光センサ×１　　・・・±の接続はどちらでも OK
10Ω抵抗×１
ジャンパーワイヤー×３

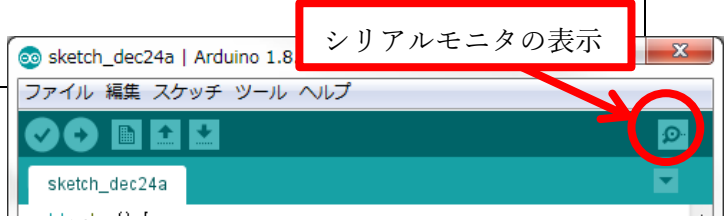
※プログラムは［ファイル］→［スケッチ例］→［Basics］→［AnalogReadSerial］から読み込む



光センサの値を読み込む回路

光センサの値を読み込むプログラム（シリアルモニタに値を表示）

```
void setup() {
  Serial.begin(9600);  // シリアルモニタに出力するためのシリアル通信の設定
}

void loop() {
  int sensorValue = analogRead(A0);  // A0 ピンから値を受け取る
  Serial.println(sensorValue);  //受け取った値をシリアルモニタに出力する
  delay(1);  // 1 ミリ秒待機
}
```

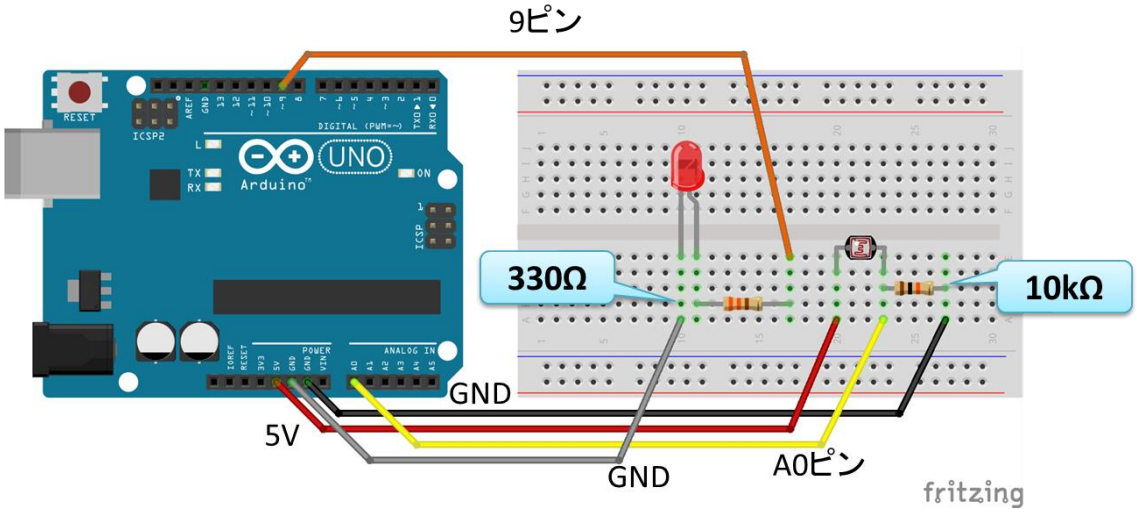


②光センサによる LED 点灯

光センサ×１　　赤色 LED×１　　10Ω抵抗×１

330Ω抵抗×１　　ジャンパーワイヤー×５

※プログラムは［ファイル］→［スケッチ例］→［Analog］→［AnalogInOutSerial］から読み込む



光センサの値に対応させて LED を点灯させる回路

※値を強調するなら、抵抗 10kΩを 200Ωなどの小さいものに入れ替える

光センサの値に対応させて LED を点灯させるプログラム

```
const int analogInPin = A0;  // 光センサを接続するピン
const int analogOutPin = 9;  // LED を接続するピン

int sensorValue = 0;         // 光センサの値
int outputValue = 0;         // LED にアウトプットする値

void setup() {
  Serial.begin(9600);   // シリアルモニタに出力するためのシリアル通信の設定
}

void loop() {
  sensorValue = analogRead(analogInPin);  // 光センサの値を受け取る
  outputValue = map(sensorValue, 0, 1023, 0, 255); // 光センサの値を 0-255 に変換
  analogWrite(analogOutPin, outputValue);  // LED にアウトプットの値を設定
```

値変換の調整をしていないのでセンサの感度と光度の対応に偏りがある可能性有り

```
  // シリアルモニタに値を出力する
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("¥t output = ");
  Serial.println(outputValue);
  delay(2);  // 2 ミリ秒待機
}
```

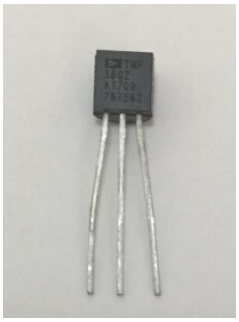
ここで、

outputValue = map(sensorValue, 0, 1023, 0, 255);

↓

outputValue = **255** - map(sensorValue, 0, 1023, 0, 255);

とするとどうなるでしょうか？



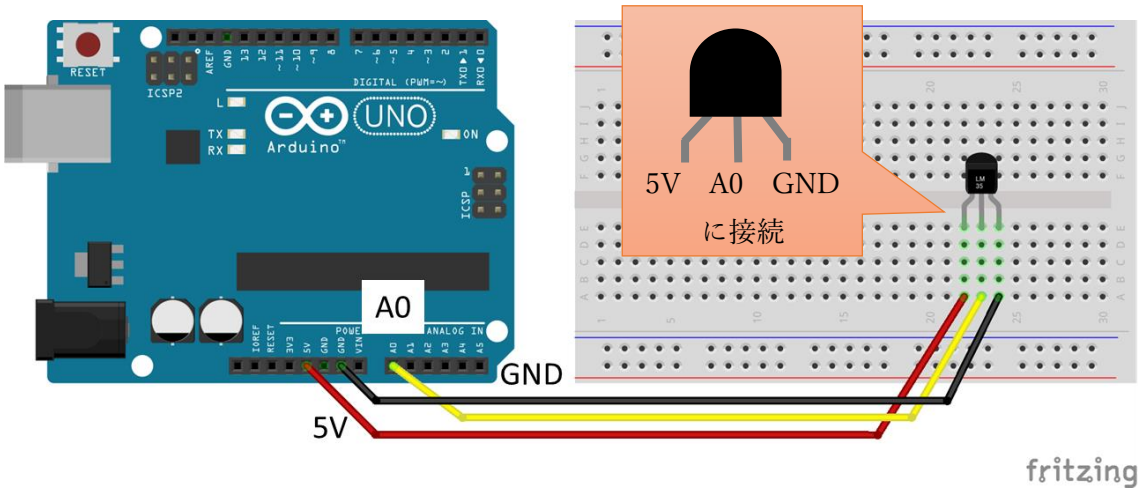
温度センサ（アナログ

③温度センサ TMP36 によるセンシング

温度センサ（TMP36/2N3904）×１

ジャンパーワイヤー×３

※プログラムは以下のサンプルプログラムを新規ファイルに書き込む（ファイル名は任意）



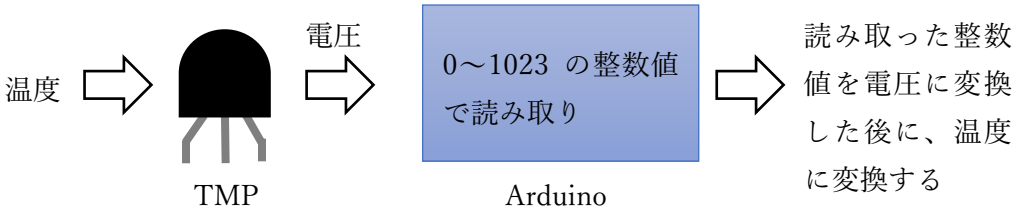
温度センサ TMP36 の値を読み取る回路

温度センサ TMP36 の値を読み取るプログラム（シリアルモニタに値を表示）

```
const int analogInPin = A0;  // 温度センサの出力に接続するピン

int sensorValue = 0;         // 温度センサの値

void setup() {
  Serial.begin(9600);   // シリアルモニタに出力するためのシリアル通信の設定
```

```
}

void loop() {
  float sensorValue = (float) analogRead(analogInPin); //温度センサの値を受け取る
  float ReadVolt = sensorValue * 5.0 / 1023.0; // 受け取った値を電圧に変換
  float Temperature = ReadVolt * 100.0 - 50.0; // 電圧を温度に変換

  // シリアルモニタに値を出力する
  Serial.print("Temperature = ");
  Serial.println(Temperature);

  delay(5000);  // 5秒待機
}
```

TMP36は温度を電圧に変換して出力しています。ArduinoはTMP36の出力している電圧を0～1023の整数値で読み取っているので、温度として認識するには整数値を変換し直す必要があります。



TMP36を経由した温度データの読み取りの流れ図

TMP36の仕様（の一部）はデータシートに以下のように記載してあります。

表4. TMP3xの出力特性

Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage @ 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500

アナログ・デバイセズ社データシートより転載

https://www.analog.com/media/jp/technical-documentation/data-sheets/TMP35_36_37_jp.pdf

この表より、TMP36 は、オフセット電圧（入力がゼロの場合に流れてしまう電圧）が 0.5[V]、出力電圧は 1℃毎に 10[mV]（=0.01V）で変化し、出力電圧は 25℃で 750[mV]（=0.75V)となることが分かります。したがって、TMP36 のセンサの値は出力電圧を Vout[V]、温度を T[℃]とすると、

$$Vout[V] = 0.01[V/℃] * T[℃] + 0.5[V]$$

と表せることになります。さらにこの式を変形すると、

$$T = 100 * Vout - 50$$

となり、温度 T を求める計算式が導出できます。
また、Arduino のアナログピン A0～A5 は、10 ビット（＝1024）の AD（Analog to Digital）コンバータを持っています。したがって、5V の電圧の Arduino のボードの場合、入力電圧の範囲は 0V から 5V となり、関数 analogRead()は、0V から 5V の入力電圧を 0 から 1023 の 1024 個の整数値に変換して読み取っていることになります。したがって、analogRead()で読み取った値を N、出力電圧を Vout とすると、

$$Vout[V] = N * 5.0 / 1023.0$$

と表せることになります（読み取りの最大値が 1023 なので分母は 1023.0 です)。
したがって、先のプログラムでは、

```
float ReadVolt = sensorValue * 5.0 / 1023.0; // 受け取った値を電圧に変換
```

が

$$Vout[V] = N * 5.0 / 1023.0$$

に相当しています。また、

```
float Temperature = ReadVolt * 100.0 - 50.0; // 電圧を温度に変換
```

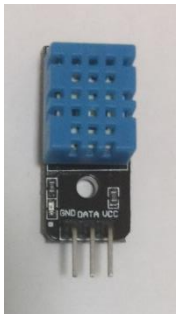
が

$$T = 100 * Vout - 50$$

に相当しています。

④　温湿度センサ DHT11 によるセンシング
温湿度センサ（DHT11）× 1
ジャンパーワイヤー× 3

※プログラムは公開されているライブラリと
サンプルプログラムを入手して利用する。

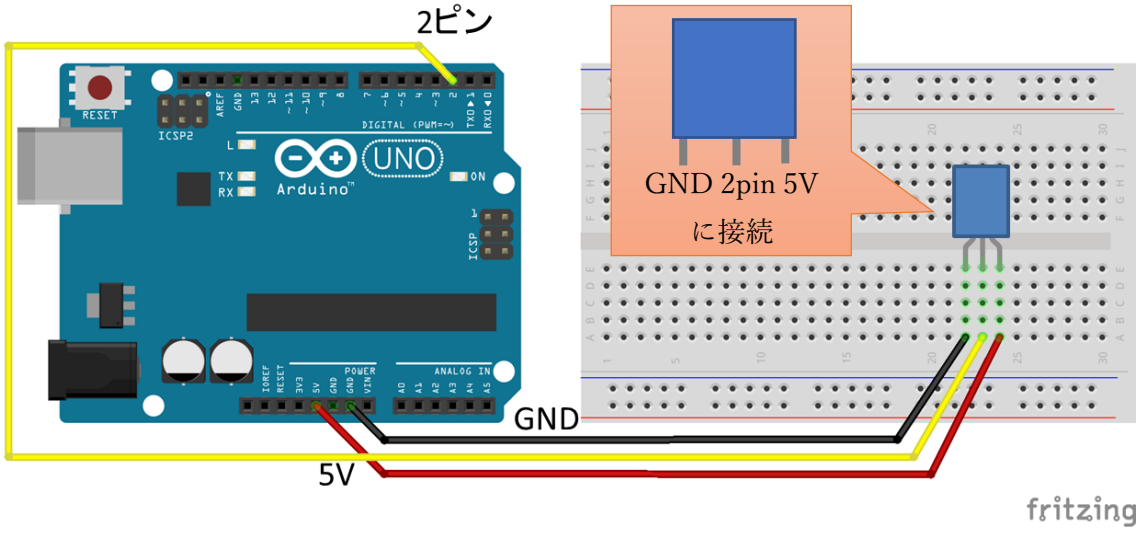


温湿度センサ（OSOYOO セット

１）GitHub に公開されているライブラリ「DHT-sensor-library-master.zip」を入手
https://github.com/adafruit/DHT-sensor-library
※上記の DHT 用のライブラリを利用するには別のライブラリ「Adafruit_Sensor-master.zip」も入手してインストールしておく必要あり（上記のサイトからも飛べます）
https://github.com/adafruit/Adafruit_Sensor

２）Arduino IDE のメニューから「スケッチ」→「ライブラリをインクルード」→「.ZIP 形式のライブラリをインストール…」を選択し、先ほどダウンロードしてきた zip ファイルを選択する。

３）プログラムは［ファイル］→［スケッチ例］→［DHT sensor library］→［DHTtester］から読み込む。



温湿度センサ DHT11 の値を読み取る回路

温湿度センサ DHT11 の値を読み取るプログラム（シリアルモニタに値を表示）

```
#include "DHT.h"  // 温湿度センサ DHT 用ライブラリの読み込み

#define DHTPIN 2     // 2 ピンに DHT 温湿度センサの出力ピンを接続

#define DHTTYPE DHT11   // DHT11 用の設定を有効にする ここのコメントアウトを外す
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321 用の設定をコメントアウト
//#define DHTTYPE DHT21   // DHT 21 (AM2301)用の設定をコメントアウト

DHT dht(DHTPIN, DHTTYPE);  // DHT 用のクラス DHT を定義

void setup() {
  Serial.begin(9600);  // シリアルモニタに出力するためのシリアル通信の設定
  Serial.println(F("DHTxx test!"));  // 説明文の出力

  dht.begin();  // dht の初期設定
}

void loop() {
  delay(2000);  // 2 秒待機

  float h = dht.readHumidity();  // 湿度データの読み込み
  float t = dht.readTemperature();  // 温度データの読み込み
  float f = dht.readTemperature(true); // 温度データ（華氏）の読み込み

  if (isnan(h) || isnan(t) || isnan(f)) {  // エラーの場合に出力
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  float hif = dht.computeHeatIndex(f, h);  // 華氏での体感温度の計算
  float hic = dht.computeHeatIndex(t, h, false);  // 摂氏での体感温度の計算

  // 湿度、温度、体感温度の値をシリアルモニタに表示
  Serial.print(F("Humidity: "));  // 湿度の表示
  Serial.print(h);
  Serial.print(F("%  Temperature: "));  // 温度の表示（摂氏と華氏）
  Serial.print(t);
  Serial.print(F("° C "));
  Serial.print(f);
  Serial.print(F("° F  Heat index: "));  //体感温度の表示（摂氏と華氏）
  Serial.print(hic);
  Serial.print(F("° C "));
  Serial.print(hif);
  Serial.println(F("° F"));
```

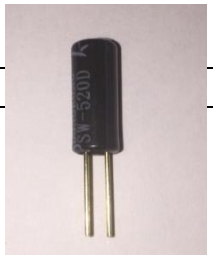
```
}
```

入力モジュール

⑤　傾斜スイッチの利用

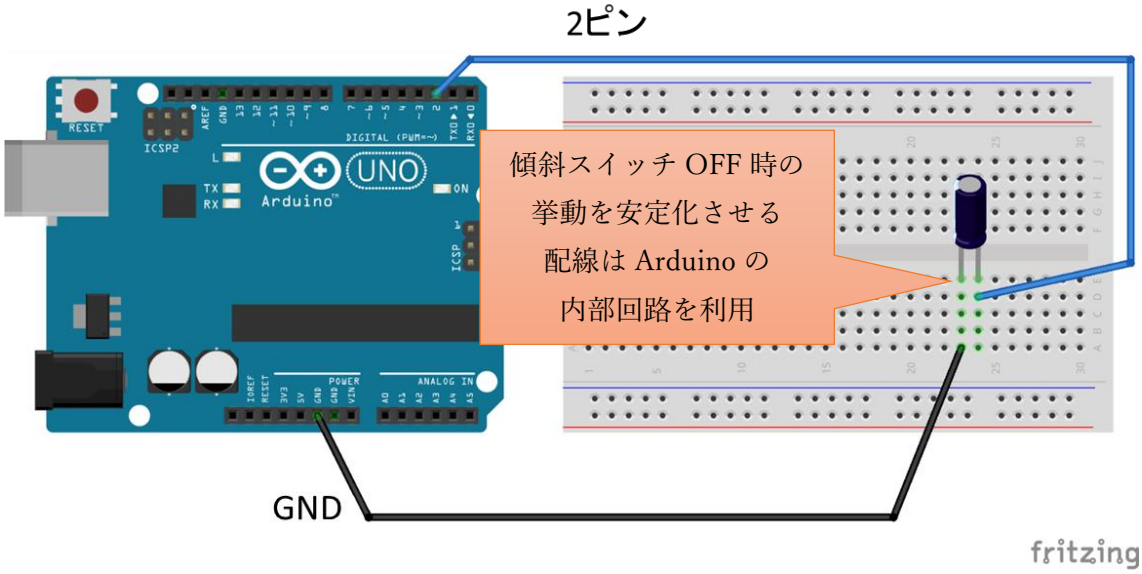
傾斜スイッチ×１

ジャンパーワイヤー×２



傾斜スイッチ（デジタル

※プログラムは［ファイル］→［スケッチ例］→［Digital］→［DigitalInputPullup］から読み込む・・・プルアップ方式の点に注意！



傾斜スイッチを傾けると内蔵 LED が消灯する回路

傾斜スイッチを傾けると内蔵 LED が消灯するプログラム

```
void setup() {
  Serial.begin(9600);   // シリアルモニタに出力するためのシリアル通信の設定
  pinMode(2, INPUT_PULLUP);  // 2 ピンからプルアップ方式で値を受け取る
  pinMode(13, OUTPUT);  //  基盤に内蔵されている LED にアウトプットする

}

void loop() {
  int sensorVal = digitalRead(2);  // 2 ピンから値を受け取る
  Serial.println(sensorVal);  //  受け取った値をシリアルモニタに出力する

  if (sensorVal == HIGH) {  // 傾斜時（＝傾斜スイッチ OFF 時）
```

```
    digitalWrite(13, LOW);  //  内蔵 LED を消灯
  } else {  // 通常時（＝傾斜スイッチ ON 時）
    digitalWrite(13, HIGH);  //  内蔵 LED を点灯
  }
}
```

※LED を 8pin に接続し、ソースコード中の 13 ピンの箇所を 8 ピンに変更してみましょう。（3 箇所あります）

⑥　ロータリーエンコーダによる LED 点灯

ロータリーエンコーダ×１　・・・回転スイッチ

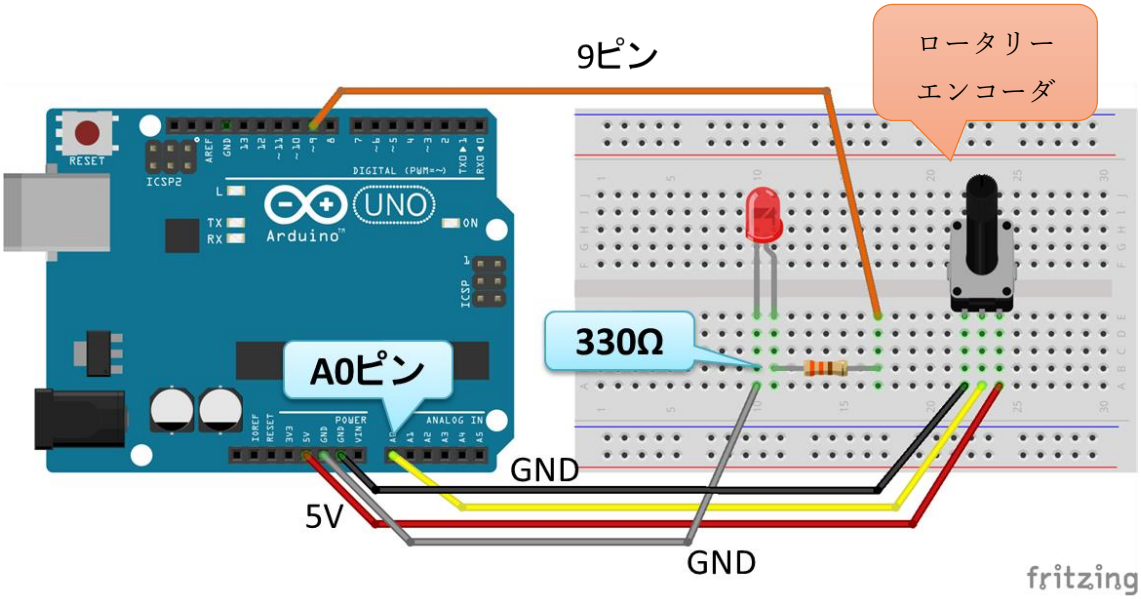
赤色 LED×１

330Ω抵抗×１

ジャンパーワイヤー×５

ロータリーエンコーダ（アナログ

※プログラムは［ファイル］→［スケッチ例］→［Analog］→［AnalogInOutSerial］から読み込む



回転スイッチで LED の光度を調整する回路

回転スイッチで LED の光度を調整するプログラム

```
const int analogInPin = A0;  // ロータリーエンコーダを接続するピン
const int analogOutPin = 9;  // LED を接続するピン
```

```
int sensorValue = 0;        // ロータリーエンコーダの値
int outputValue = 0;        // LED にアウトプットする値

void setup() {
  Serial.begin(9600);  // シリアルモニタに出力するためのシリアル通信の設定
}

void loop() {
  sensorValue = analogRead(analogInPin);  // ロータリーエンコーダの値を受け取る
  outputValue = map(sensorValue, 0, 1023, 0, 255); // ロータリーエンコーダの値
を
                                                    // 0-255 に変換する関数
  analogWrite(analogOutPin, outputValue);  // LED にアウトプットの値を設定

  // シリアルモニタに値を出力する
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("¥t output = ");
  Serial.println(outputValue);
  delay(2);  // 2 ミリ秒待機
}
```

⑦　赤外線コントローラから赤外線レシーバへの受信

赤外線リモコン×１　　赤外線レシーバ×１

赤色 LED×１　　330Ω抵抗×１

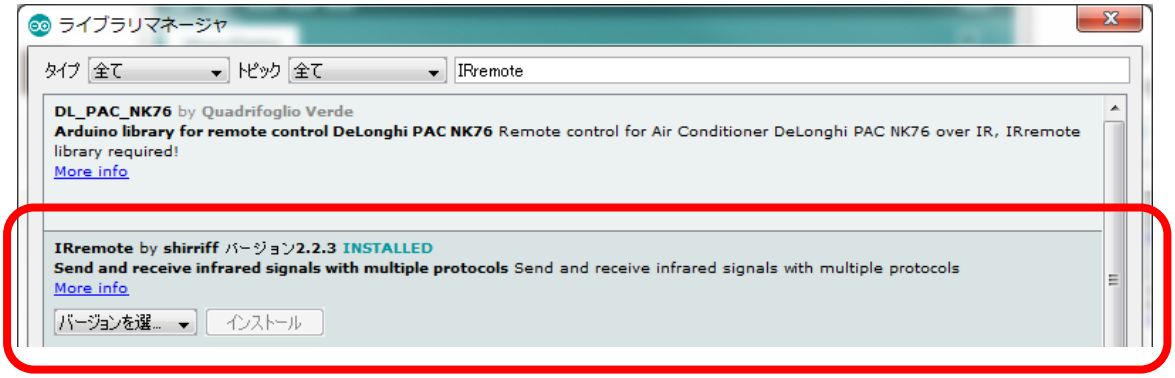
ジャンパーワイヤー×５

※プログラムは［ライブラリを管理...］から入手できるライブラリとサンプルプログラムを利用する。

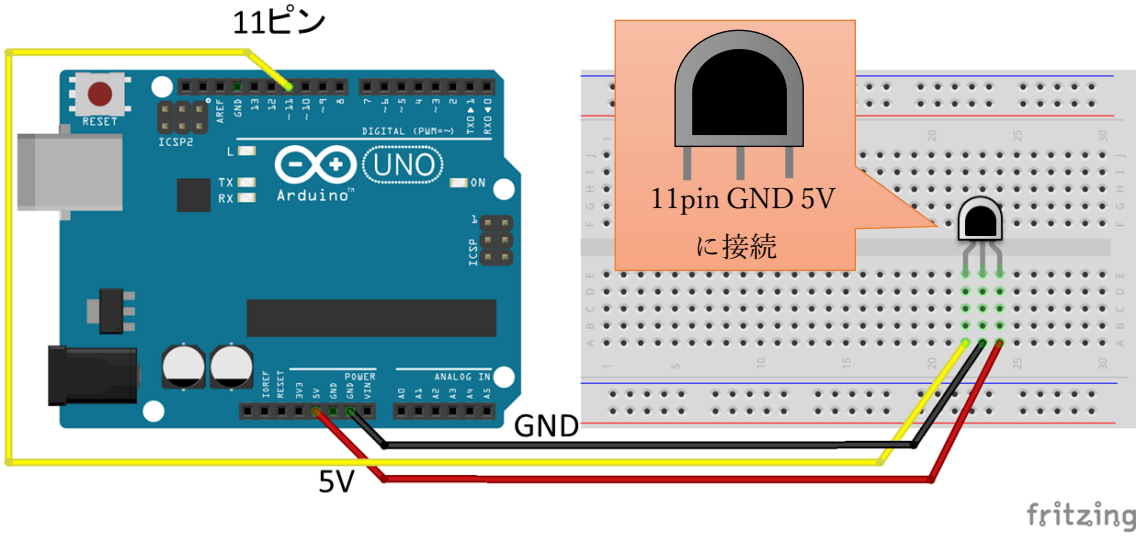
赤外線リモコンと赤外線レシーバ

１）Arduino IDE のメニューから「ツール」→「ライブラリを管理...」を選択し、ライブラリマネージャを開く。

２）ライブラリマネージャの検索機能を利用して、IRremote をキーワードにしてライブラリを検索し、IRremote by shirriff を選択しインストールする。



３）プログラムは［ファイル］→［スケッチ例］→［IRremote］→［IRrecvDemo］から読み込む。



赤外線コントローラから赤外線レシーバへ受信する回路

赤外線コントローラから赤外線レシーバへ受信するプログラム（受信値をシリアルモニタに表示）

```
#include <IRremote.h>  // 赤外線デバイス用ライブラリの読み込み

int RECV_PIN = 11;  // 赤外線レシーバの出力を 11 ピンに接続

IRrecv irrecv(RECV_PIN);  // 赤外線用のクラスを生成

decode_results results;  // 受信結果を格納する変数

void setup()
{
  Serial.begin(9600);  // シリアルモニタに出力するためのシリアル通信の設定
  Serial.println("Enabling IRin");  // 受信開始前
  irrecv.enableIRIn(); // 赤外線の受信開始
```

```
  Serial.println("Enabled IRin");  // 受信開始後
}

void loop() {
  if (irrecv.decode(&results)) {  //受信結果を 16 進数でシリアルモニタに表示
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
  delay(100);
}
```

受信した信号（16 進数）の先頭に 0x を付けて IF 関数すればボタン作が可能にな

DC5V ブザー

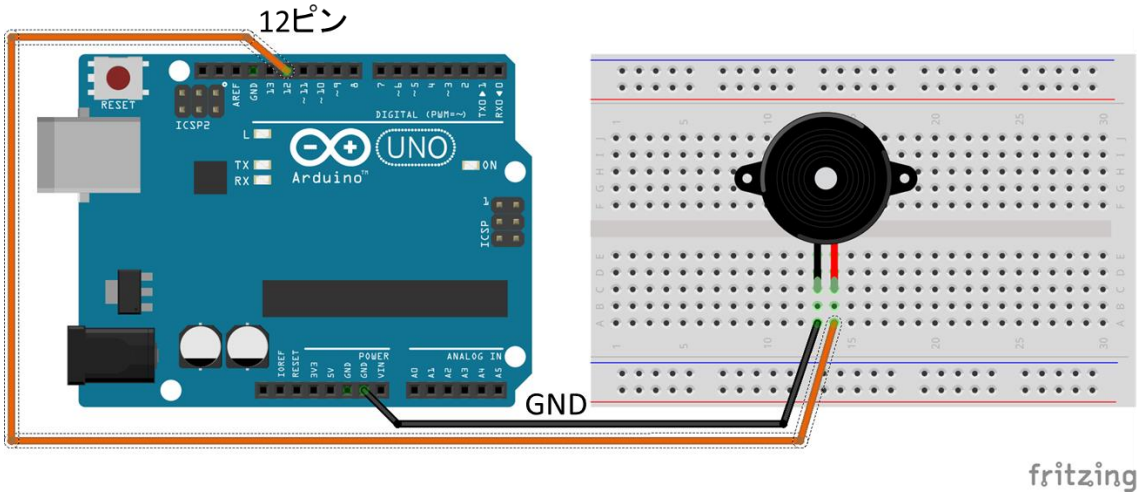
出力モジュール

⑧　ブザーによる音出力

DC5V ブザー×１

ジャンパーワイヤー×２

※プログラムは以下のサンプルプログラムを新規ファイルに書き込む（ファイル名は任意）



ブザーで音を出力する回路

ブザーで音を出力するプログラム

```
int pin = 12;      // ブザーの＋を 12 ピンに接続

void setup() {
}

void loop() {
     tone(pin,262,1000) ;  // ド
     delay(1500) ;         // 1500-1000 ほど待機
```

```
      tone(pin,294,1000) ;  // レ
      delay(1500) ;
      tone(pin,330,1000) ;  // ミ
      delay(1500) ;
      delay(1000) ;         // 1秒待機
}
```

⑨　LCDによるテキスト出力

LCD×１

ジャンパーワイヤー（凸～凹）×４

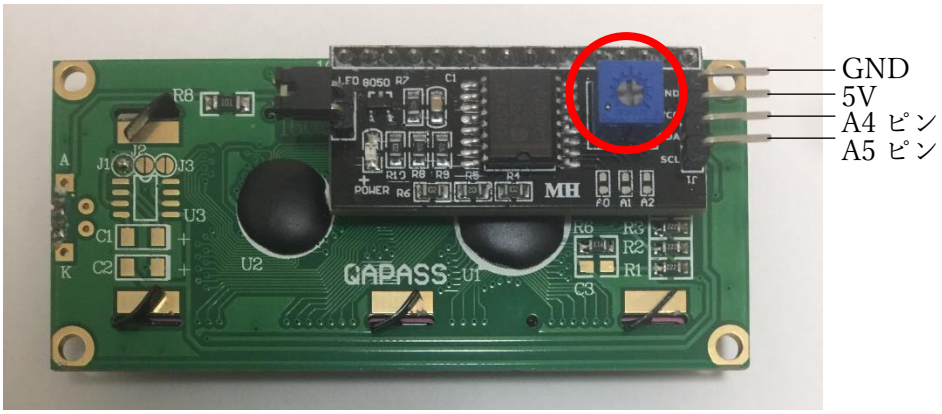
LCD（Liquid Crystal Display）

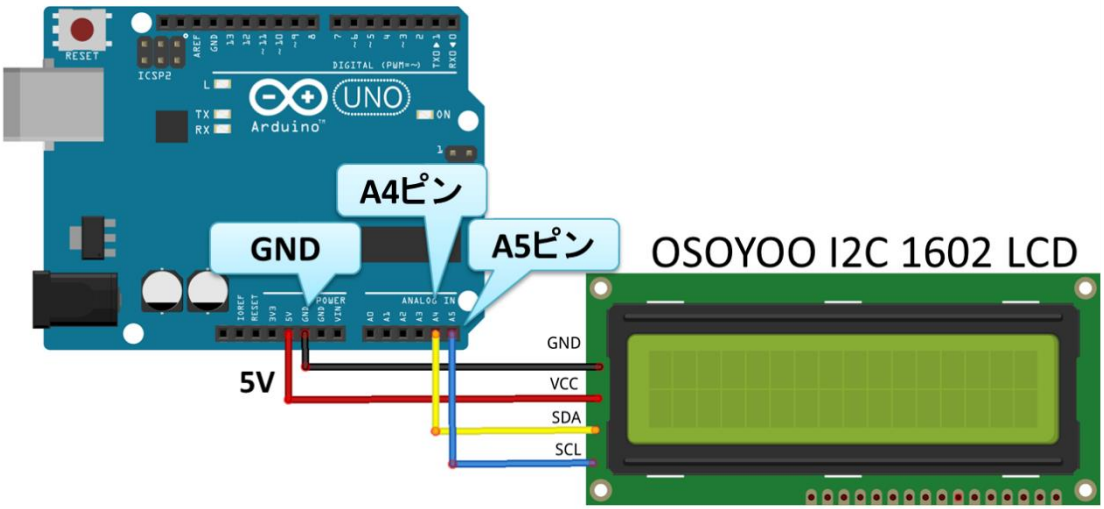
１）以下のOSOYOOのサポートサイトからライブラリ(LiquidCrystal_I2C.zip)をダウンロードする。 ※短縮URL: http://bit.ly/osoyooLCD
http://osoyoo.com/ja/2014/12/07/16x2-i2c-liquidcrystal-displaylcd/

２）Arduino IDEのメニューから「スケッチ」→「ライブラリをインクルード」→「.ZIP形式のライブラリをインストール…」を選択し、先ほどダウンロードしてきたzipファイルを選択する。

３）プログラムは以下のサンプルプログラムを新規ファイルに書き込む（ファイル名は任意）。
※もしくは、Arduino IDEのメニューから[ファイル]→[スケッチ例]→[LiquidCrystal_I2C]→[HelloWorld]から読み込んでもOKです(下のサンプルプログラムとは若干異なります)。

※LCDの裏面にコントラストを調整するつまみがあります。開封時等は、マイナスドライバーでつまみを左右に回して文字が表示されるように調整してください。コントラストが適切に設定されていないと、テキストが正しく出力されていても目視することができません。





LCD にテキストを表示する回路

LCD にテキストを表示するプログラム

```
#include <Wire.h>  // I2C デバイスとの通信用ライブラリ
#include <LiquidCrystal_I2C.h> // LCD 用ライブラリ

LiquidCrystal_I2C lcd(0x27,16,2);  // LCD の設定(設定用アドレス, 16 文字, 2 行)

void setup()
{
  lcd.init();  // LCD の初期化

  lcd.setCursor(0,0);  // 0 列 0 行に表示位置を移動、左上左端が(0, 0)
                       // 2 行目は(0, 1)
  lcd.backlight();  //LCD のバックライトを点灯
  lcd.print("Hello, world!");  // LCD への出力
}
```

注） 0x27, 0x3F など機器によって異なる

```
void loop()
{

}
```

lcd.clear();で表示を消去できる
表示を変えるときに利用

注）
OSOYOO のサポートページから ic2_scanner というファイルをダウンロードして Arduino でロードしましょう（または、ic2_scanner のプログラムを Arduino にコピペします）。
プログラムを Arduino に書き込んだあと、Arduino の操作画面でシリアルモニタ（「Tools」-「serial monitor」）を開くと I2C アドレスが表示されます。0x27、0x3F など機種によって異なります。詳細は次の OSOYOO LCD サポートページを参照してください。
OSOYOO LCD サポートページの URL
http://osoyoo.com/ja/2014/12/07/16x2-i2c-liquidcrystal-displaylcd/

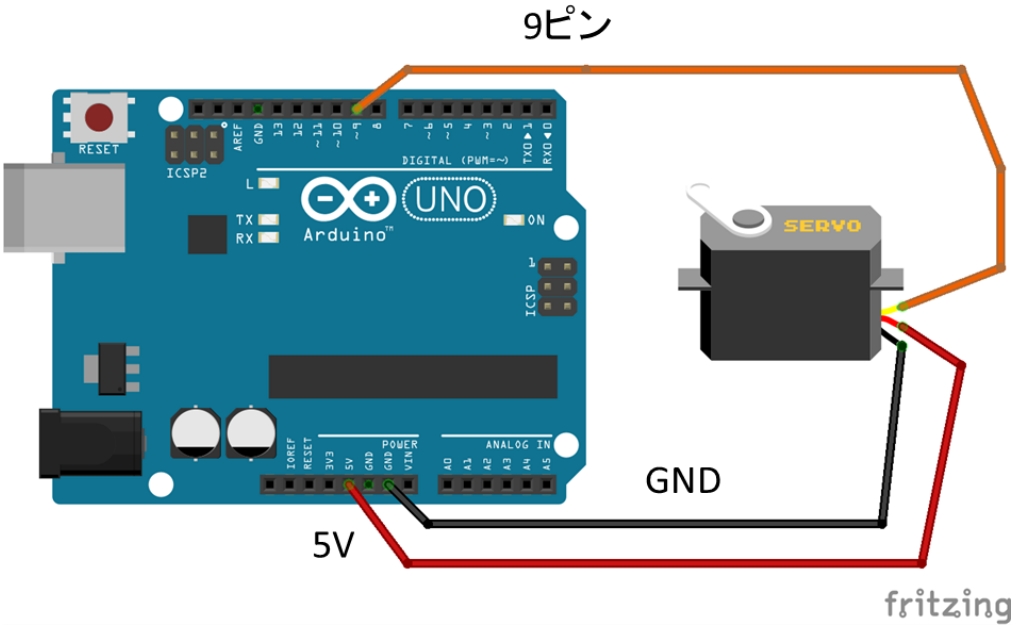
ic2_scanner ファイルの URL
http://osoyoo.com/wp-content/uploads/samplecode/ic2_scanner.txt

アクチュエータ
⑨ サーボの制御
マイクロサーボ×１
ジャンパーワイヤー×３

中央が 5V
茶色が GND
オレンジが 9 ピン

マイクロサーボ

※プログラムは［ファイル］→［スケッチ例］→［Servo］→［Sweep］から読み込む。



サーボを自動操作する回路

サーボを自動操作するプログラム

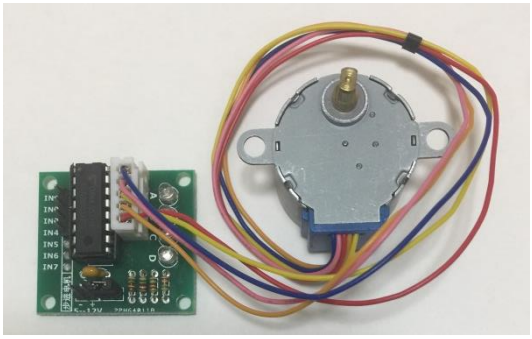
```
#include <Servo.h>

Servo myservo;  // サーボを動作させるクラスを定義

int pos = 0;    // サーボの角度を初期化

void setup() {
  myservo.attach(9);  // サーボの制御ピンを 9 ピンに接続
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // 0 度から 180 度に 1 度ずつ移動
    myservo.write(pos);              // 指定した角度に移動
    delay(15);                       // 0.015 秒待機
  }
  for (pos = 180; pos >= 0; pos -= 1) { // 180 度から 0 度に-1 度ずつ移動
    myservo.write(pos);              // 指定した角度に移動
    delay(15);                       // 0.015 秒待機
```

```
    }
}
```

⑩ モータの制御

ステッピングモータ×１

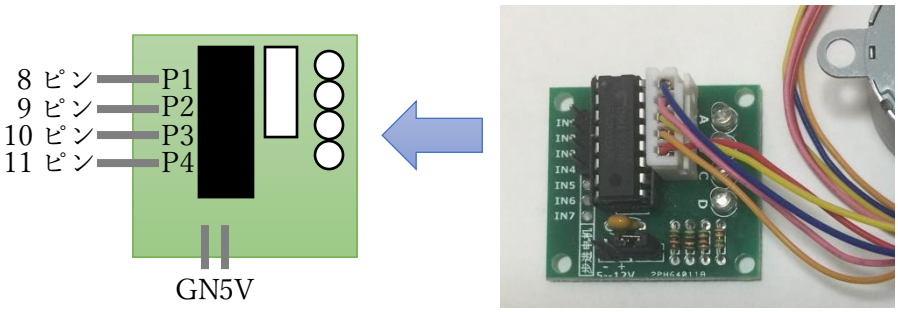
ジャンパーワイヤー×６



ステッピングモータ

※プログラムは、以下の OSOYOO のサポートサイトのサンプルプログラムを Arduino の新規ファイルにコピペする

http://osoyoo.com/wp-content/uploads/samplecode/stepper.txt



モータのドライバと Arduino の接続

⑪　応用編：各自で色々なセンサやモジュールを組合せて利用

※作成した回路の回路図を描く

これまでやってきた①～⑤をもとに様々なセンサやモジュールを組合せて回路を作ってみましょう。

A）センサ値を LCD に出力

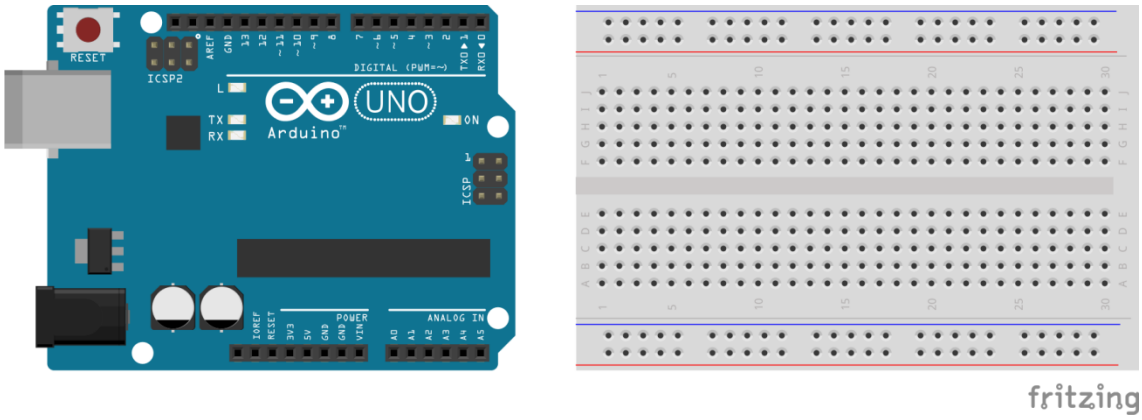
B）赤外線コントローラで LED を光らせる or ブザーで音を鳴らす

音階と周波数

	ド	レ	ミ	ファ	ソ	ラ	シ
1	131	147	165	175	196	220	247
2	262	294	330	349	392	440	494
3	523	587	659	698	784	880	988

C）ロータリーエンコーダでサーボやモータを動かす

D）その他



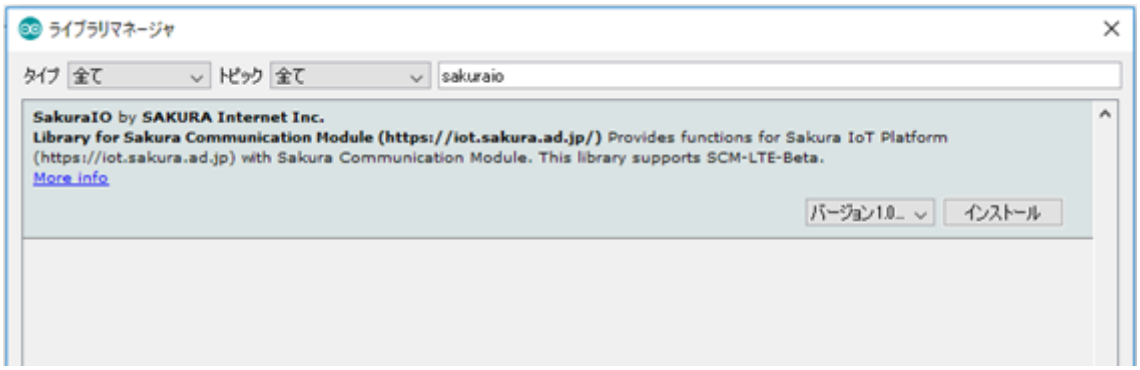
「OSOYOO 公式チュートリアル」Web サイト http://osoyoo.com/ja/2014/12/06/arduino-starter-kit/

も参考にしてみましょう。

演習３　さくら LTE モジュールの回路設計と利活用

sakura.io リファレンス
https://sakura.io/docs/

① 通信モジュールの接続とデータ送信
sakura.io リファレンスを参考に、LTE ボードの取り付けとモジュール登録を行います。Arduino を起動し、[スケッチ]＞[ライブラリをインクルード]＞[ライブラリを管理…]から検索機能を使用し、SakuraIO ライブラリをインストールします。



ライブラリをインストール後、連携サービスを登録します。sakura.io コントロールパネル（https://secure.sakura.ad.jp/iot/login）からプロジェクト内の［詳細］に進み、詳細の中の［連携サービス］のタブから［連携サービス追加］を選らんで、WebSocket を選択し適当な名前を付けて登録します。


#3438
テスト接続００１
データストアプラン
ライト
簡易位置情報提供機能
Off
ファイル配信
編集
削除
連携サービス
連携サービス追加
種類
名前
AWS test001
sakuratest2
Incomingtest
testservice
test
5 件中 1 - 5 件を表示
表示件数
50
①連携サービス
②連携サービス追加


ホーム › プロジェクト詳細 › 連携サービスカタログ
外部サービスとsakura.ioを連携し、データのやり取りを行います。
詳しくはドキュメントをご覧ください。 sakura.ioドキュメント - 連携サービス仕様
WebSocket
Outgoing Webhook
Incoming Webhook
MQTT Client
Datastore API
AWS IoT
Azure IoT Hub(α)：正式版提供に伴い廃止予定
Google Cloud Pub/Sub Publisher
Azure Event Hubs
Azure IoT Hub


連携サービス追加 - WebSocket
リアルタイムの双方向通信を行う連携サービスです。
詳しくはドキュメントをご覧ください。 sakura.ioドキュメント - WebSocket
名前
sakuratest1
①名前
②追加
追加


ホーム › プロジェクト詳細 › 連携サービス詳細
リアルタイムの双方向通信を行う連携サービスです。
詳しくはドキュメントをご覧ください。 sakura.ioドキュメント - WebSocket
#14986
WebSocket
名前
sakuratest1
URL 情報
URL
wss://api.sakura.io/ws/v1/d070
Token 情
Token
d070
編集
削除
最終到着データ(50件)
チャンネル別到着データ
接続
時刻
モジュール
タイプ
ペイロード
データはありません


ホーム › プロジェクト詳細
#3438
テスト接続００１
データストアプラン
ライト
簡易位置情報提供機能
Off
ファイル配信
編集
削除
モジュール
連携サービス
イベントアラート
連携サービス
連携サービス追加
種類
名前
設定
websocket
sakuratest1
プロジェクト詳細画面から上の連携サービス詳細への移動

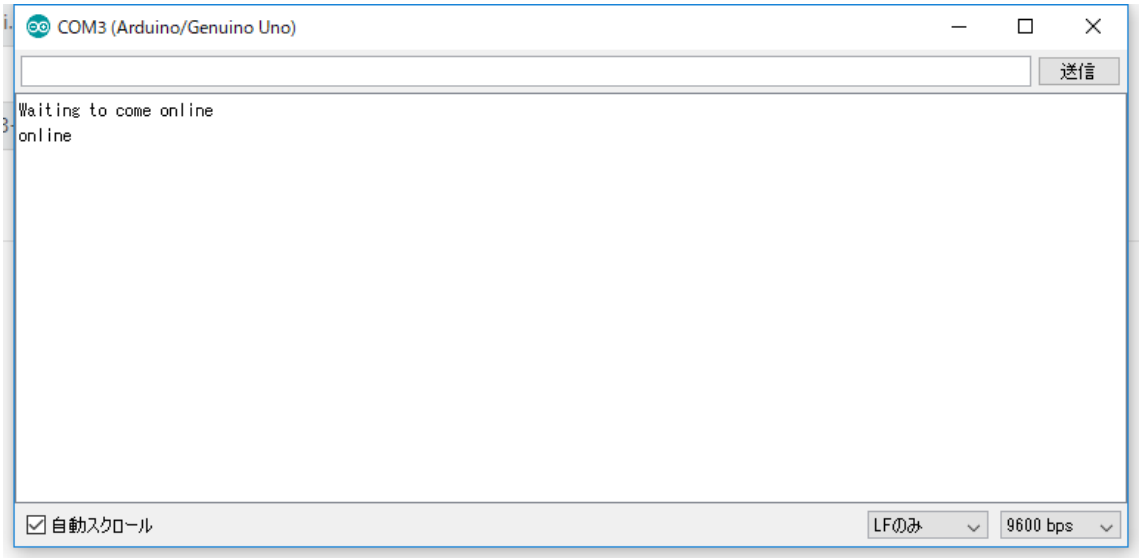
これで WebSocket の連携サービスが登録できました。次に、サンプルプログラムを入力し、実行して下さい。

Arduino 側のサンプルプログラム

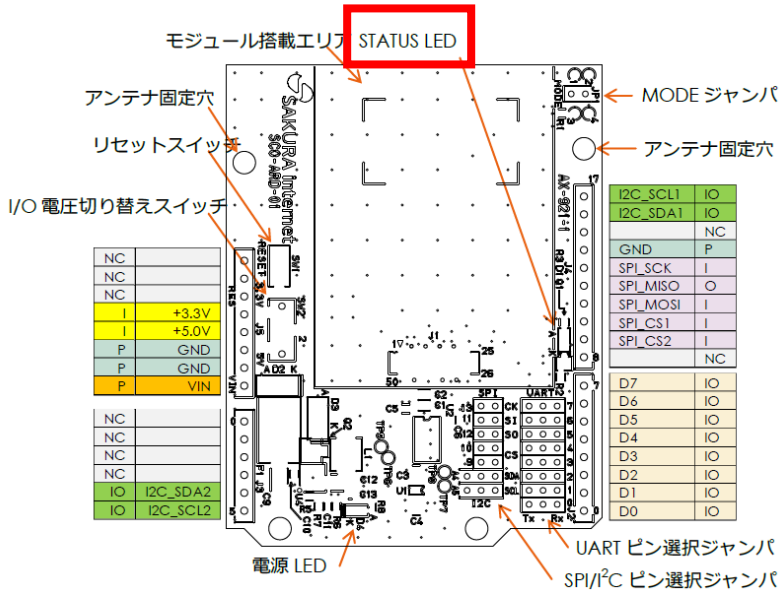
```
#include <SakuraIO.h>

SakuraIO_I2C sakuraio;
uint32_t cnt;

void setup() {
  Serial.begin(9600);
  Serial.print("Waiting to come online");
  for(;;){//接続
    if( (sakuraio.getConnectionStatus() & 0x80) == 0x80 ) break;
    Serial.print(".");
    delay(1000);
  }
  Serial.println("");
  Serial.println("online");
}

void loop() {
  cnt++;
  sakuraio.enqueueTx(0,cnt); //1 つ目のダミーデータの送信
  sakuraio.enqueueTx(1,cnt); //2 つ目のダミーデータの送信
  sakuraio.enqueueTx(2,cnt); //3 つ目のダミーデータの送信
  sakuraio.send();
  delay(10000);
}
```

ここでは意図にダミーデータを使っています.
ぜひ後で、各自でセンサを接続してセンサ値の送信にチャレンジし

シリアルモニタに、次のように online と表示されれば通信成功です。



なお、シリアルモニタに online が表示されたにも関わらずデータが上手く送信できていない場合，AC アダプなどで電源を供給すると上手く通信できることがあります。また、STATUS LED の状態も以下のように 1 回点滅しているか確認してください。



点滅回数	意味
3 回	起動中
2 回	オフライン
1 回	オンライン

このサンプルプログラムは、10 秒ごとに WebSocket で 3 つのチャンネルにカウンタデータを送信しています。setup 部では sakura.io の接続を行い、loop 部で送信しています。実際にデータが受信できているかをチェックするには、コントロールパネルから確認できます。連携サービスから先ほど作成した名前のサービスをクリックして下さい。次のように、ペイロードのデータがリアルタイムで表示されます。

・最終到着データの表示

時刻	モジュール	タイプ	ペイロード
2019-08-19 13:21:21		keepalive	

2019-08-19 13:21:15	ub4iRG9GadgC	channels	{
			"channels": [
			{
			"channel": 0,
			"type": "I",
			"value": 22,
			"datetime": "2019-08-19T04:21:14.982015863Z"
			},
			{
			"channel": 1,
			"type": "I",
			"value": 22,
			"datetime": "2019-08-19T04:21:15.004015863Z"
			},
			{
			"channel": 2,
			"type": "I",
			"value": 22,
			"datetime": "2019-08-19T04:21:15.025015863Z"
			}
			]
			}

2019-08-19 13:21:11		keepalive	

・チャンネル別到着データの表示

時刻	モジュール	チャンネル
2019-08-19 13:21:56	ub4iRG9GadgC	ch0: 26 ch1: 26 ch2: 26

② jQuery による WebSocket のデータ取得

次に、この WebSocket の情報をローカル PC から HTML で取得して Web ブラウザで表示させましょう。次ページの HTML ソースをテキストエディタで作成し、html ファイル（名前は任意）で保存しましょう。ここでは、jQuery を使用して簡単に WebSocket の情報を取得します。サンプルソース 6 行目は、jQuery のライブラリ URL を直接指定して読み込んでいます。次に 8 行目で、各モジュールに発行される wss://から始まる固有の URL を埋め込みます。インターネット上からは、この URL で各モジュールにアクセスできます。

Arduino 側では、3 つのデータを送信しています。これらは、次の JSON 形式で送られてきます。

{"module":"モジュールシリアル番号","type":"channels","datetime":"2019-08-19T04:48:16.645267212Z","payload":{"channels":[{"channel":0,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.58026841Z"},{"channel":1,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.60226841Z"},{"channel":2,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.62426841Z"}]}}

Arduino 側では、sakuraio.enqueueTx(0,cnt)が「ch0 に変数 cnt の内容を送信せよ」という内容です。すなわち、JSON の内容と対応づけると次のようになります。

```
sakuraio.enqueueTx(0,cnt);
 ↓
{"channel":0,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.58026841Z"}

sakuraio.enqueueTx(1,cnt);
 ↓
{"channel":1,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.60226841Z"}

sakuraio.enqueueTx(2,cnt);
 ↓
{"channel":2,"type":"I","value":183,"datetime":"2019-08-19T04:48:16.62426841Z"}
```

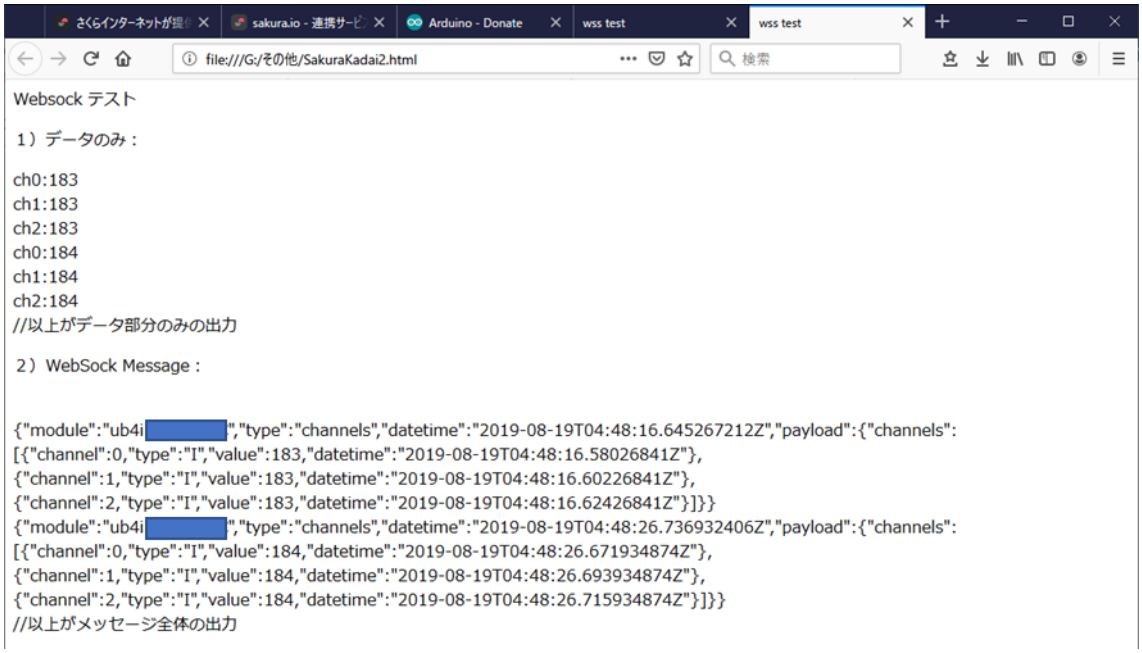
実際の値は value に格納されているため、data.payload.channels[0].value を読み出すことで、値の読み出しが可能です。

サーバ側のサンプルプログラム（今回はローカル環境で可）

```
<!DOCTYPE html>
<html lang="ja">
<head>
<title>wss test</title>
</head>
<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script type="text/javascript">
        var webSocket = new WebSocket('モジュールの URL');

        //エラー出力
        webSocket.onerror = function(event) {
                onError(event)
        };
        webSocket.onopen = function(event) {
                onOpen(event)
        };
        webSocket.onmessage = function(event) {
                onMessage(event)
        };

        function onMessage(event) {
                //websock を受け取る
                var data = $.parseJSON(event.data);  // JSON 形式からデータ部分
を取り出す
                var module = data.module; // モジュールシリアルの代入
                if(module == "モジュールシリアル"){  // 任意のモジュールのみ抽
出
                    document.getElementById("data").innerHTML += "ch0:"+ data.
payload.channels[0].value +"<br />";
                    document.getElementById("data").innerHTML += "ch1:"+ data.
payload.channels[1].value +"<br />";
                    document.getElementById("data").innerHTML += "ch2:"+ data.
payload.channels[2].value +"<br />";
                    document.getElementById("dump").innerHTML += "<br />" + ev
ent.data;
                }
        }
        function onError(event) {
                alert(event.data);
        }
</script>
</head>
<body>
        Websock テスト
        <p>
        １）データのみ：
        <div id="data"></div> //以上がデータ部分のみの出力
        <p>
        ２）WebSock Message：
```

```
        <div id="dump"></div> //以上がメッセージ全体の出力
        <p>
</body>
</html>
```

実行例：



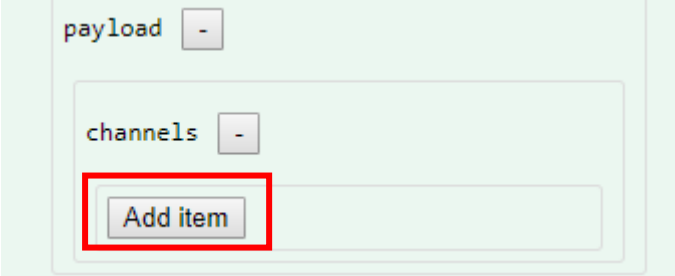
なお、この HTML はローカル PC がインターネットに接続していればローカル環境から実行できます。

③ Curl コマンドによる LED 点灯

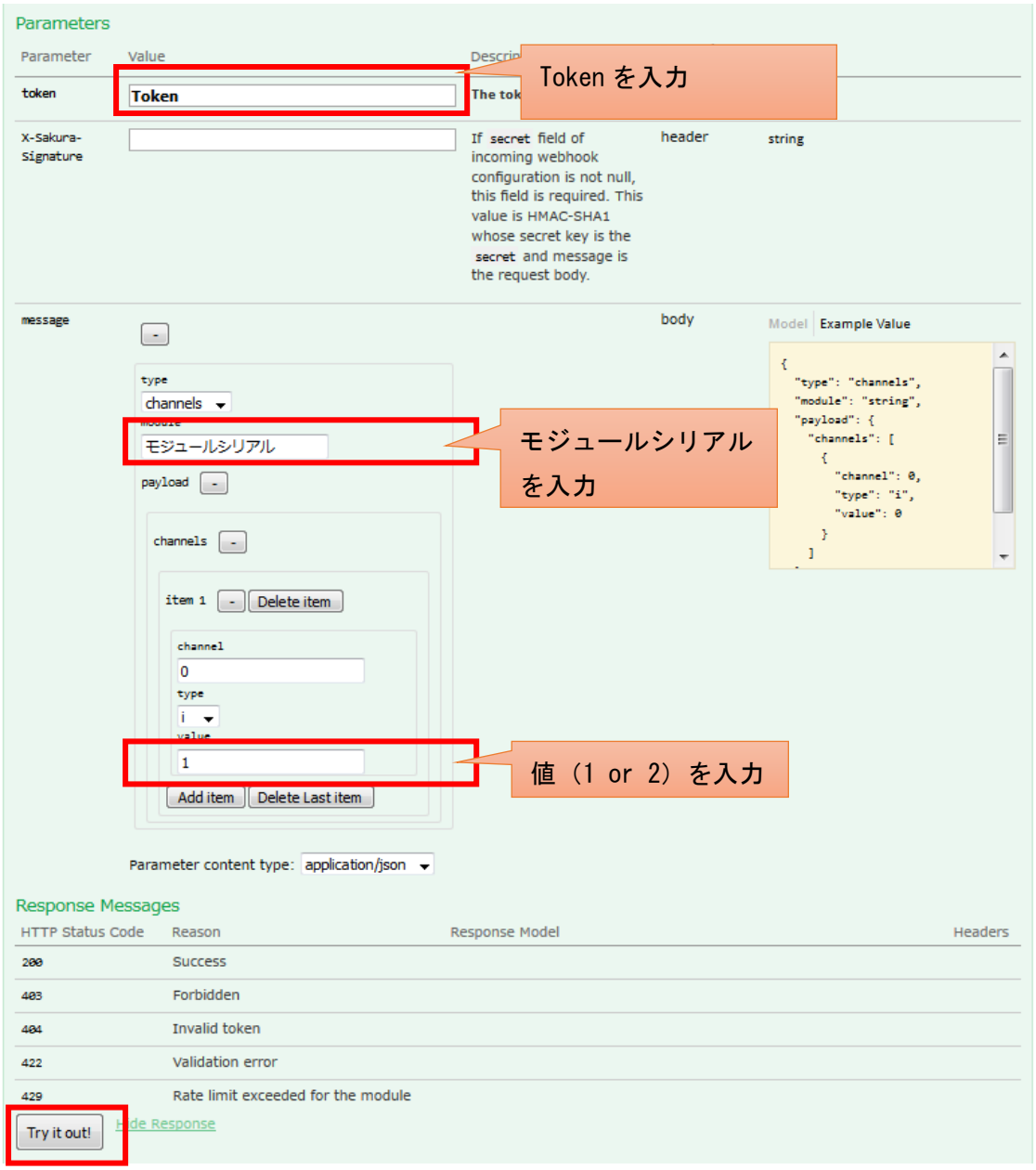
Incoming Webhook を使用することで、インターネットを経由して PC からボードを制御することができます。データ形式は JSON です。

まず WebSocket 同様に Incoming Webhook のサービス連携を登録します。コントロールパネルから適当な名前を入力して登録しますが、この時に Token と URL を控えておきます。

次に、データを送信する JSON を組み立てますが、さくらインターネットが JSON 生成用フォームを用意しているのでこれを使用しましょう。sakura.io Incoming Webhook（https://api.sakura.io/incoming/v1/docs/）にアクセスし、ページ中央やや下の[Add item]をクリックします。



ここでは、例として int 型データ 1 を送信する Incoming Webhook を組み立てることとします。次の例を参考に、Token と Secret、モジュールシリアルを入力し、type を i(int 型)、value に値の 1 を入力して画面下の[Try it out!!]をクリックします。



これでヘッダー情報や JSON、URL などが生成されました。実際の JSON は、Curl の枠に出力された次の内容のうち、以下の太字の部分が JSON に相当します。なお、以下は Linux ベースの PC から Curl コマンドを使って JSON を送信するコマンドになります（Windows 版は後述）。このコマンドは JSON 部分をメッセージとして、ヘッダー情報をつけ POST 送信するコマンドになります。

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '**{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}**' '**URL**'

JSON に相当する部分を抜き出すと以下になります（上記太字部分)。

{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}

では、次に Arduino 側にこの通信を受けるプログラムを Arduino IDE から書き込みます。ここでは、4 番のデジタルピンに白の LED を、7 番のデジタルピンに赤の LED を接続し、Incoming Webhook の内容でそれぞれを点灯させるようにします。

Arduino 側の Incoming Webhook の受信プログラム

```
#include <SakuraIO.h>
SakuraIO_I2C sakuraio;

void setup() {
  pinMode(7,OUTPUT);
  pinMode(4,OUTPUT);
  // シリアル通信を開始
  Serial.begin(9600);

  // オンラインになるまで待つ
  while ((sakuraio.getConnectionStatus() & 0x80) != 0x80) {
    Serial.print(".");
    delay(1000);
  }

  Serial.println("");
  Serial.println("Online");
}

void loop() {
  // 受信キューの状態を取得
  uint8_t rxAvailable;
  uint8_t rxQueued;
  sakuraio.getRxQueueLength(&rxAvailable, &rxQueued);

  digitalWrite(7,LOW);
  digitalWrite(4,LOW);
  // 受信キューにたまっているメッセージの数だけ繰り返す
  for (uint8_t i = 0; i < rxQueued; i++) {
    uint8_t channel;
    uint8_t type;
```

```
    uint8_t values[8];
    int64_t offset;

    // キューからのメッセージを取り出しに成功したら以下を実行
    uint8_t ret = sakuraio.dequeueRx(&channel, &type, values,
                                     &offset);
    if (ret == CMD_ERROR_NONE) {
      Serial.print("channel: ");
      Serial.println(channel);
      Serial.print("type: ");
      Serial.println((char)type);
      Serial.print("value: ");

      if (type == 'i') {
        // 32 ビット整数型の場合
        int32_t value = 0;
        memcpy(&value, &values, sizeof(int32_t));
        Serial.println(value);

        if (value == 1) { // value が 1 だったら 7 番ピンの LED を光らせる
          digitalWrite(7,HIGH);
          sakuraio.enqueueTx(0,7UL);
          sakuraio.send();
        }
        if (value == 2) { // value が 2 だったら 7 番ピンの LED を光らせる
          digitalWrite(4,HIGH);
          sakuraio.enqueueTx(0,4UL);
          sakuraio.send();
        }

      }
    } else {
      Serial.println("ERROR");
    }
  }

  delay(3000);
}
```

この例では、インターネット上から int 型整数が送信されてきて、その値が 1 なら 7 ピン（赤色 LED）を、値が 2 なら 4 ピン（白色 LED）のデジタルポートを HIGH に、すなわち LED を点灯させます。ここで、Curl コマンドが使用できる PC から先ほどの生成された Curl コマンドを含んだ JSON を Curl コマンドとして発行すると、i = 1 なので赤色 LED が点灯します。

PC のコマンドプロンプトで

curl -V

と入力して Curl コマンドが動作するかどうか確認しましょう。Curl が動作する場合は先ほどの Curl コマンドをコマンドプロンプトから実行してみましょう。ただし、Windows から Curl コマンドを実行するには、'（シングルコーテーション）を"（ダブルコーテーション）に変換し、""内の"の前に¥（バックスラッシュ）を付ける必要があります。その修正をしたコマンドが以下になります。

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{¥"type¥":¥"channels¥",¥"module¥":¥"モジュールシリアル¥",¥"payload¥":{¥"channels¥":[{¥"channel¥":0,¥"type¥":¥"i¥",¥"value¥":1}]}}" "URL"

Curl が動作しない場合は、公式ページ（https://curl.haxx.se/）から exe ファイルをダウンロードしてきて、exe ファイルがあるフォルダで上記のコマンドを実行しましょう。
※Curl コマンドは、色々なプロトコルを使ってデータ転送をするコマンドで、cURL（Clinent for URL）とも表記されます。オープンソースソフトウェアです。

この Curl コマンドをそのまま流用して、Curl を扱える PHP などで Web プログラミングを作成すれば（例えば、HTML のフォームを POST メソッドで送り、受け取った内容を JSON 形式に組み立てて Curl を発行するなど）、Web ページ上のボタンなどからインターネット経由で LED を点灯させることができます。後述の演習課題で類似課題を実施しましょう。

なお、secret キーを設定した場合は、JSON 部分をメッセージとし、Secret を Key とする HMAC-SHA1 を計算します。そして計算して算出した HMAC-SHA1 を X-Sakura-Signature ヘッダーに入れて、POST リクエストを送ることになります。詳細は以下の sakura.io ドキュメントを参照してください。

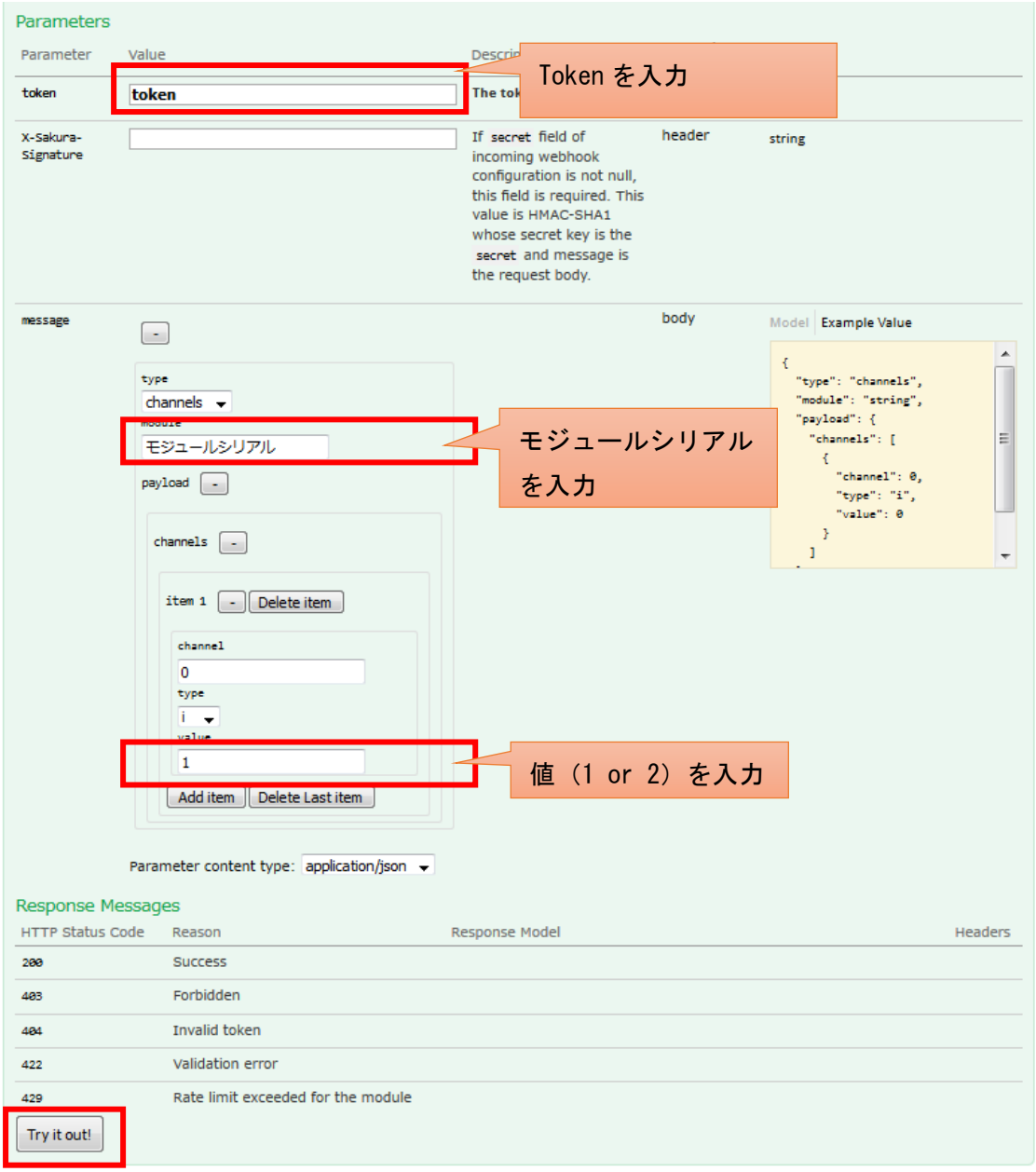
Docs 》API 利用ガイド 》連携サービス API
https://sakura.io/docs/pages/guide/api-guide/integrated-service-api.html

③’ Secret を入力した Curl コマンドによる LED 点灯

それでは Secret を指定して、暗号ハッシュ関数 HTMAC-SHA1 を利用したメッセージ認証による通信を行ってみましょう。ここでは先ほどの③の 1 と 2 を送る通信に認証機能を追加させます。まず以下のように Secret を入力した新しい Incoming Webhook の連携サービスを追加しましょう。

次にリクエストボディをメッセージとし、Secret を Key とする HMAC-SHA1 を計算します。リクエストボディ（JSON 部分）を確認するために、③と同様に、**さくらインターネットが用意している JSON 生成用フォームを使用しましょう。**sakura.io Incoming Webhook

（https://api.sakura.io/incoming/v1/docs/）にアクセスし、token、モジュールシリアル、value を設定しましょう（ここではまだ Secret は未入力です）。

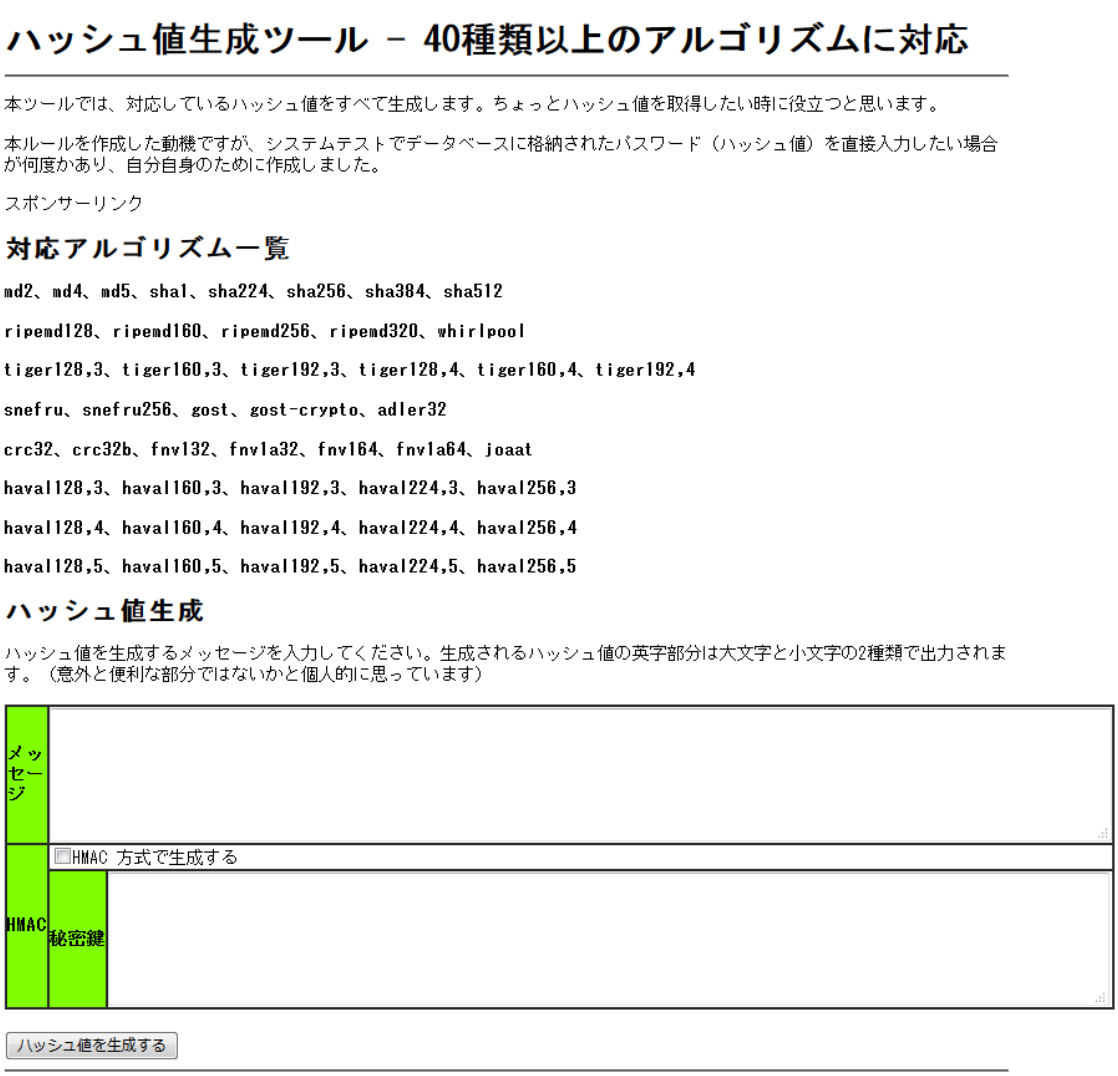


［Try it out!］を押すと以下のような JSON 生成してくれます。この背景色が黄色の部分が暗号化のためのメッセージ部分になります。

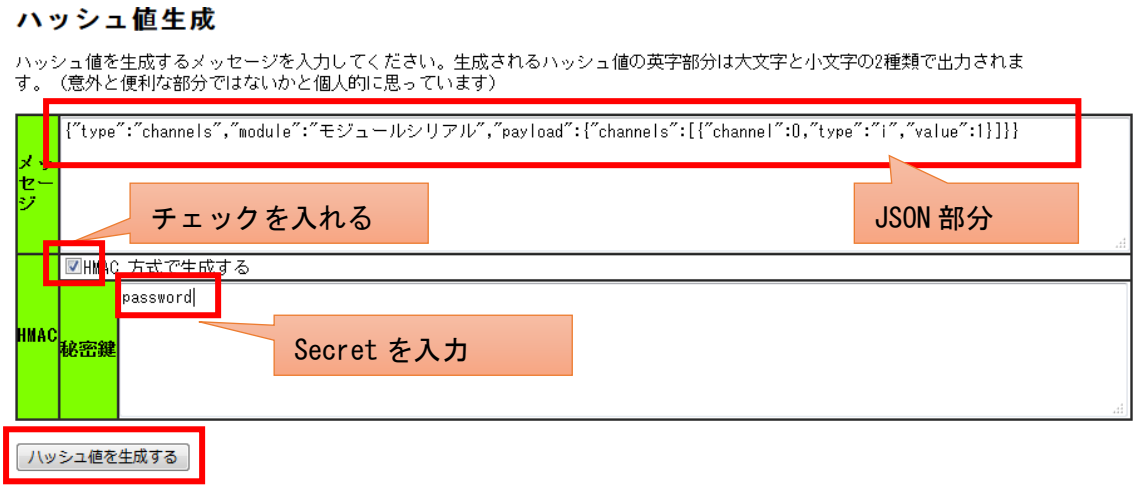
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}' 'URL'

次に HMAC-SHA1 アルゴリズムに準拠して先ほどのメッセージと Secret からハッシュ値を作成します。ここでは Web ページ上でハッシュ値生成ができる以下のサイトを利用します。

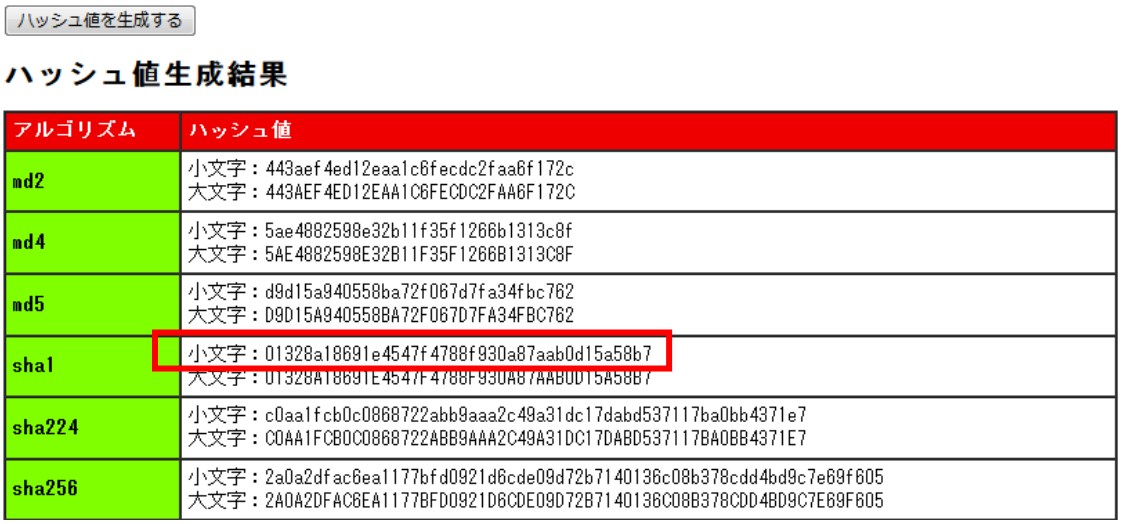
http://hensa40.cutegirl.jp/software/create_hash/



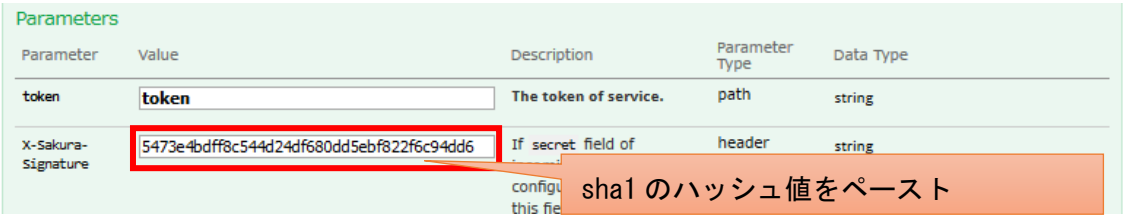
ここで、メッセージは先ほど生成した JSON 部分「{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}」とし、「HMAC 方式で生成する」のチェックボックスにチェックを入れ、秘密鍵に先ほど Incoming Webhook の連携サービスを作成したときに入力した Secret（ここでは password）を入力して［ハッシュ値を生成する］を押します。



ボタンを押すとハッシュ値が生成されるので、sha1 の小文字のハッシュ値をコピーします。



次に、先ほど利用したさくらインターネットが用意しているJSON生成用フォームsakura.io Incoming Webhook（https://api.sakura.io/incoming/v1/docs/）の［X-Sakura-Signature］にハッシュ値をペーストして［Try it out!］を押すとCurlの欄に完成したコマンドが表示されます。



・生成された Curl コマンド

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'X-Sakura-Signature: 5473e4bdff8c544d24df680dd5ebf822f6c94dd6' -d '{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}' 'URL'

・Windows 用に加工した Curl コマンド

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" --header "X-Sakura-Signature: 5473e4bdff8c544d24df680dd5ebf822f6c94dd6" -d "{¥"type¥":¥"channels¥",¥"module¥":¥"モジュールシリアル¥",¥"payload¥":{¥"channels¥":[{¥"channel¥":0,¥"type¥":¥"i¥",¥"value¥":1}]}}" "URL"

③と同様に Windows 用に加工した Curl コマンドをコマンドプロンプト上で実行すると、Arduino の対応する LED が点灯します。

同様のやり方でもう片方の LED を作成させる Curl コマンドを作成してみましょう。具体的には、先の Curl コマンドの中で value が 1 から 2 に変更する必要があります。つまりハッシュ値を生成する基となるメッセージが変わってしまうので、またハッシュ値を作りなおす必要があります。

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'X-Sakura-Signature: ~~5473e4bdff8c544d24df680dd5ebf822f6c94dd6~~' -d '{"type":"channels","module":"モジュールシリアル ~~yload~~":{"channels":[{"channel":0,"type":"i","value":2}]}}' 'URL'

ハッシュ値を作り直して書き換える必要がある

1 から 2 に変わる

④ センサデータの送信と取得

A) ①の課題を参考にして任意のセンサデータを Arduino に接続して、計測したセンサデータを sakura.io に送信してみましょう。

B) ②の課題を参考にして sakura.io に送信した任意のセンサデータをローカル環境から HTML を利用して取得してみましょう。

⑤ Node-RED を使ったデータ通信

A) データの受信とファイルへの書き出し

Node-RED は IBM を中心として開発された開発環境です。Flow エディタを使って、プラグイン/モジュールであるノードを視覚的に接続しながら、IoT デバイスとオンラインサービスをつなぐことができる開発ツールになります。

Windows では Node.js のサイト **https://nodejs.org/** からまず Node.js (Ver 10.x) をインストールし、次に管理者権限のコマンドプロンプトか PowerShell 上で次のコマンドで Node-RED をインストールします。

C:¥> **npm install -g --unsafe-perm node-red**

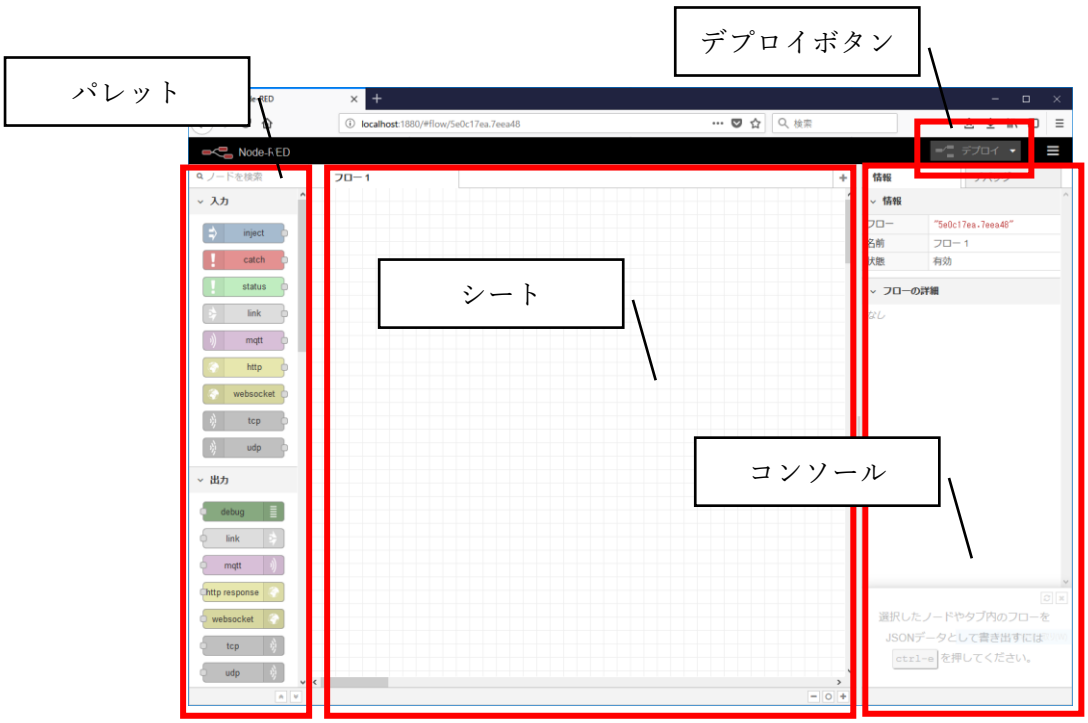
Node-RED の起動は次のコマンドで行います。

C:¥> **node-red**

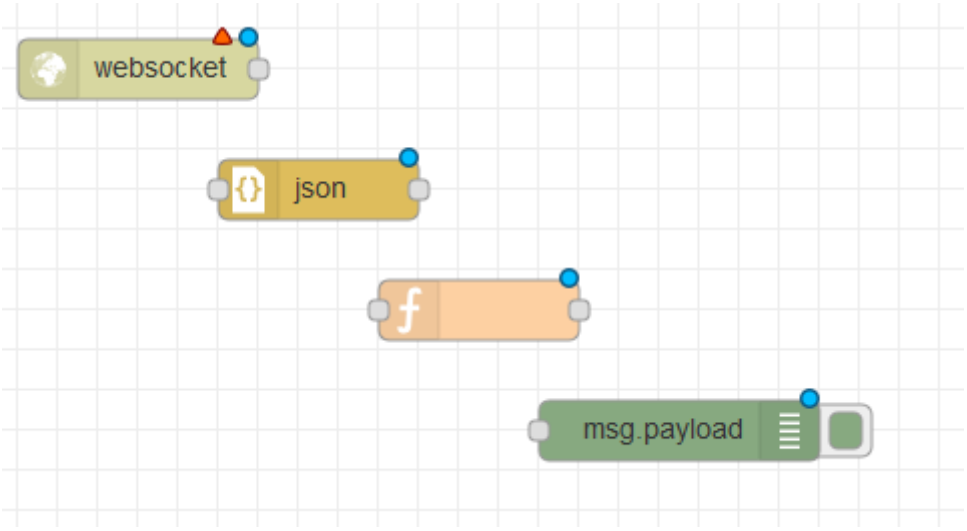
起動後に Web ブラウザで http://127.0.0.1:1880 または http://localhost:1880 にアクセスすると Node-RED が開きます。もし動作しない場合は次のコマンドを試して下さい。

C:¥> **node C:¥Users¥<ユーザ名>¥AppData¥Roaming¥npm¥node_modules¥node-red¥red.js**

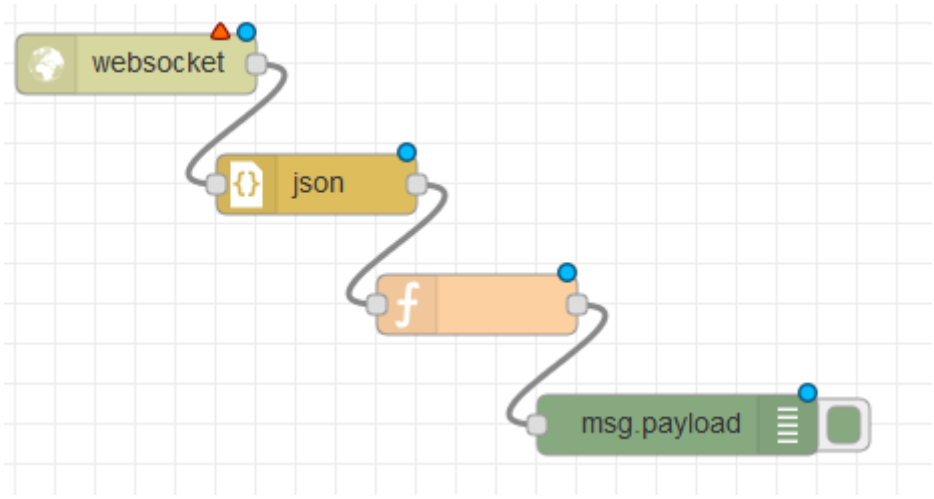
設定ファイルは、**C:¥Users¥<ユーザ名>¥.node-red¥settings.js** になります。



まず、左のノードのリストがあるパレットから Websocket、json、function、debug の 4 つをドラッグ＆ドロップで中央のシートに配置します。



それぞれのノードはコネクタで接続することができます。



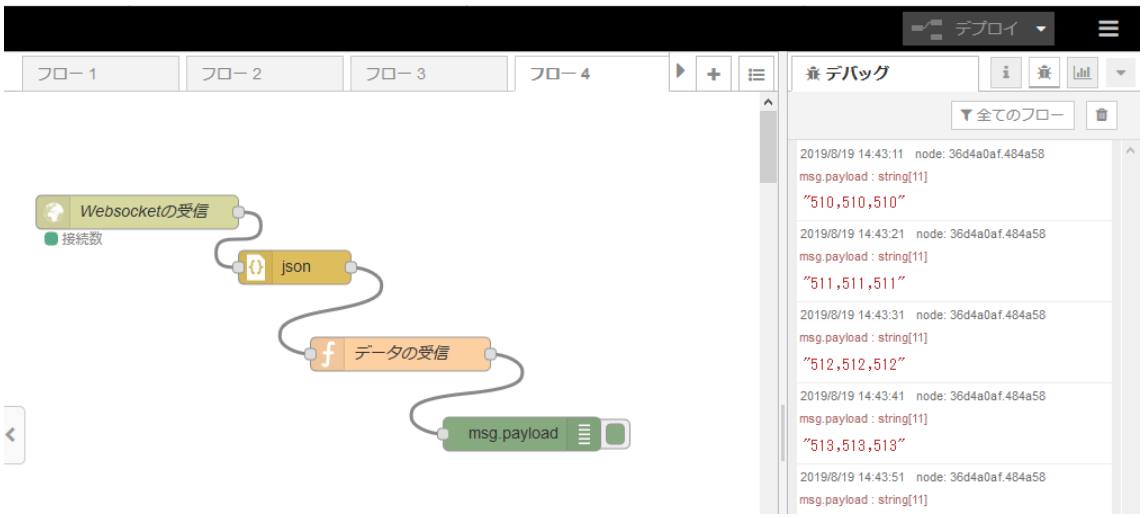
ノードをダブルクリックするとそれぞれ設定できます。

まず最初に、Websocket のノードをダブルクリックして、モジュールの URL と名前(Websocket の受信）を設定します。

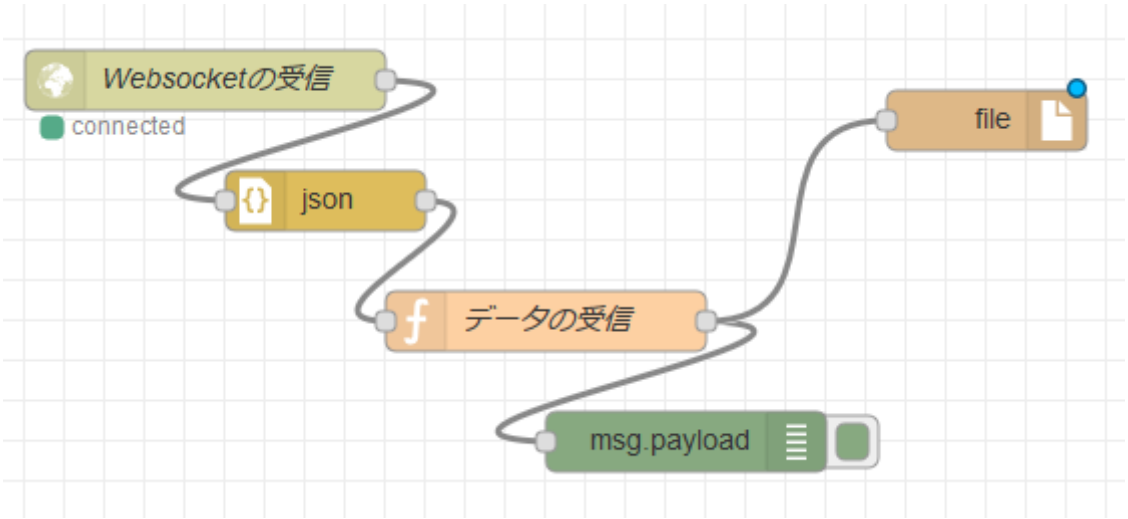
次に function のノードをダブルクリックして、名前とデータ受信用のプログラムを入力します。

```
if(msg.payload.module == "xxxxxxxxxx"){
    if(msg.payload.type =="channels"){
        var d1 = msg.payload.payload.channels[0].value;
        var d2 = msg.payload.payload.channels[1].value;
        var d3 = msg.payload.payload.channels[2].value;
        msg.payload = d1 + "," + d2 +"," + d3;
        return msg;
    }
}
```

設定したあと右上の[デプロイ]ボタンを押すとデータ受信を開始し、debug(msg.payload)にて受信したデータを確認することができます。



次にデータをファイルに書き出すために、file のノードを設置してデータの受信(function)と接続します。



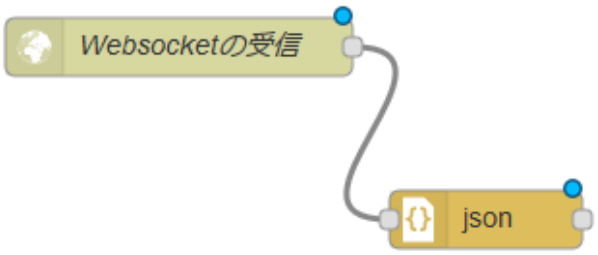
次に file のノードをダブルクリックして、名前と書き出しようのファイル名をフルパスで入力します。

最後にデプロイボタンを押すと、データを受信して書き出すプログラムが動作し始めます。

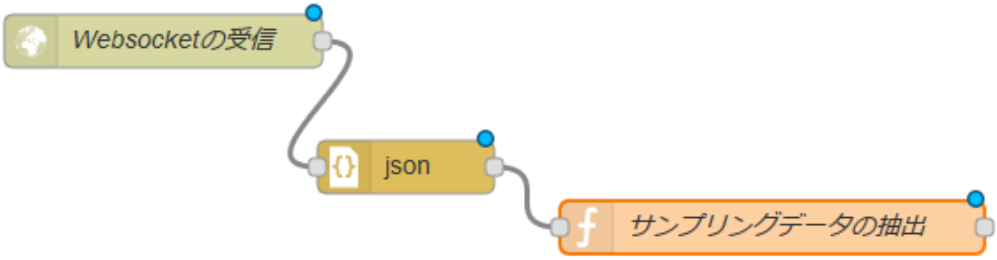
B) データの受信とリアルタイムのグラフ化

ここでは IoT デバイスから受信したデータをリアルタイムに可視化していきます。

まず、先ほどの演習 A)と同様の Websocket、json を準備します。ただし、タブでフロー画面を切り替えて作成する場合（つまり、先の演習課題を動作させながら残したまま、この演習課題を実施する場合）、Websocket のプロパティからパスを設定する時に「新規に websoket-lisener を追加」から Websocket の URL を再度登録してください。例えば、フロー１とフロー２で同じ websoket-lisener を利用していた場合、両方で同一のものが使われるため場合によってはエラーが発生してしまいます。同じ URL を登録する場合でも別の websoket-lisener として新規登録して動作させることでフロー１とフロー２で独立して動作できるようになります。

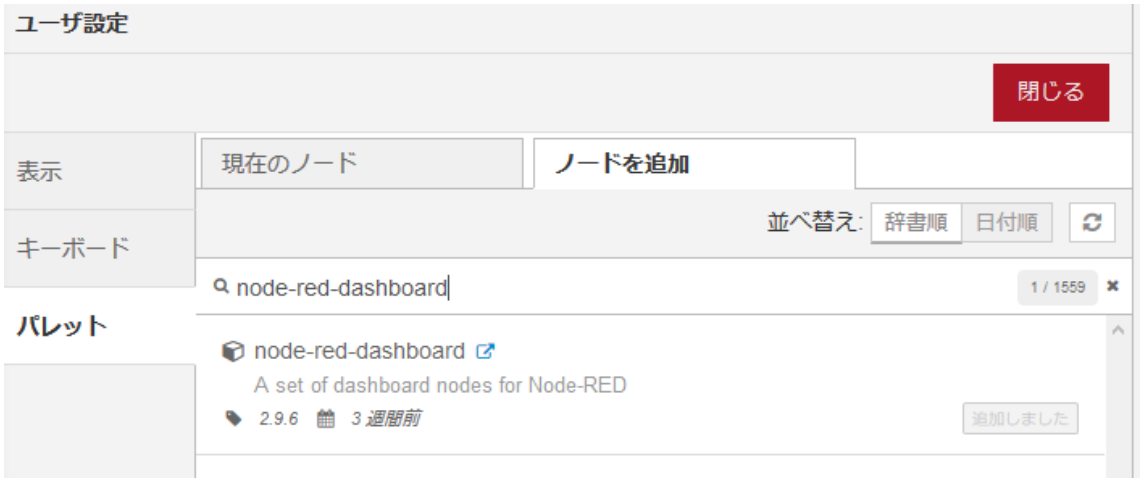


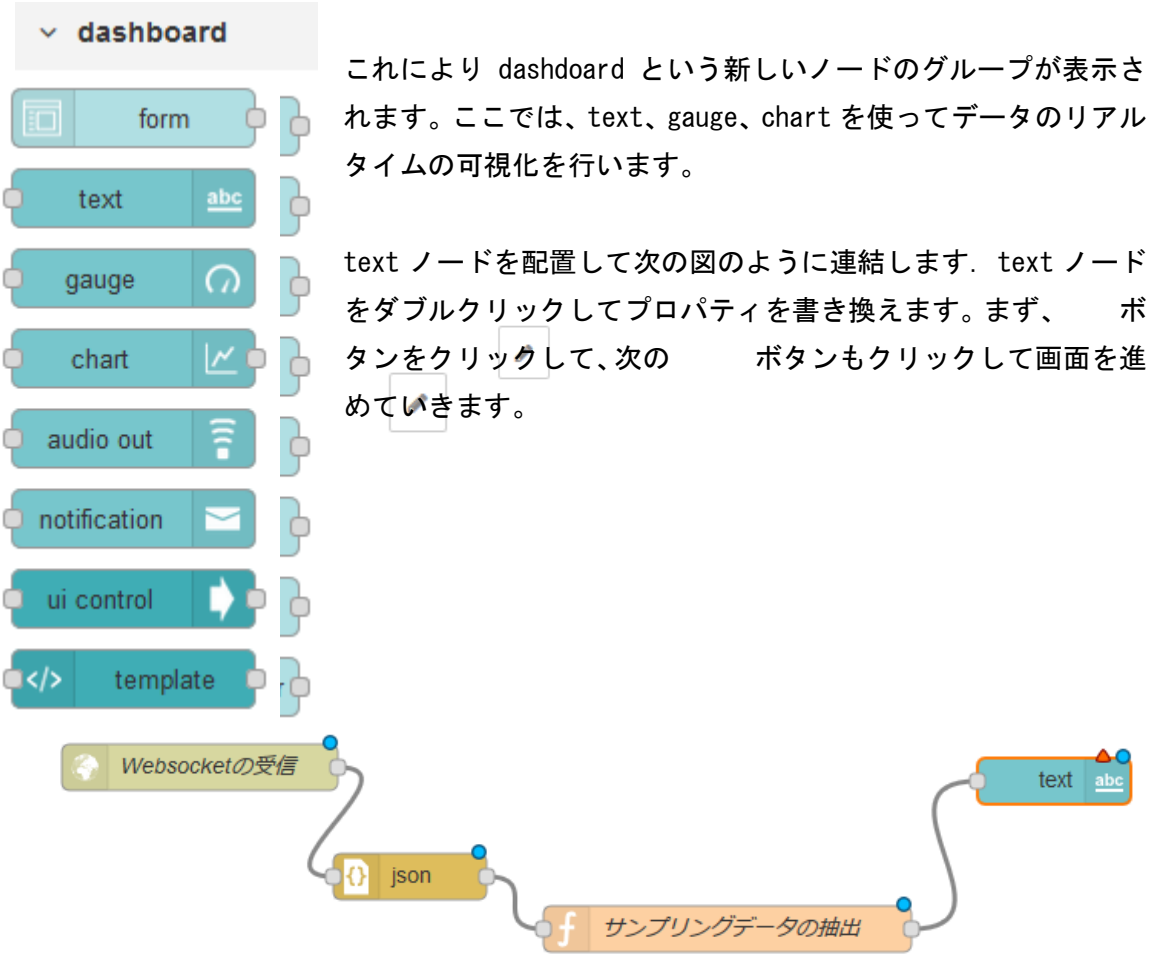
次に、function でリアルタイムに可視化する任意の１つのデータを指定します。




```
if(msg.payoad.module == "xxxxxxxxxx"){
    if(msg.payload.type =="channels"){
        var d1 = msg.payload.payload.channels[0].value;
        msg.payload = d1;
        return msg;
    }
}
```

次に右上メニューの設定を選び、開いたユーザ設定画面からパレットタブに切り替えます。
さらにノードの追加タブから node-red-dashboard を検索して追加します。





これにより dashdoard という新しいノードのグループが表示されます。ここでは、text、gauge、chart を使ってデータのリアルタイムの可視化を行います。

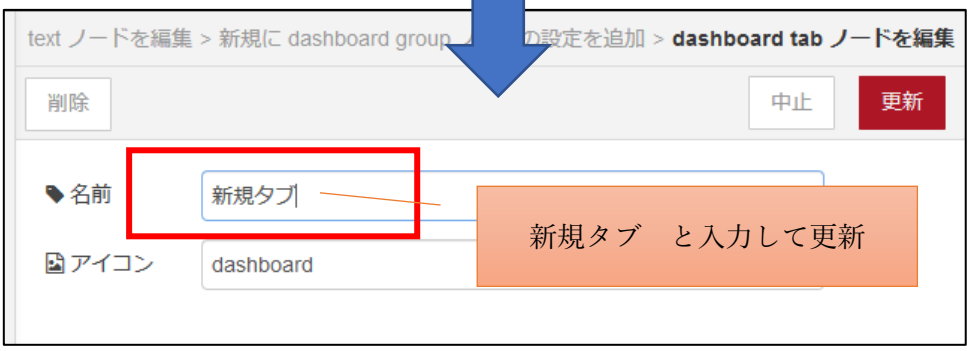
text ノードを配置して次の図のように連結します．text ノードをダブルクリックしてプロパティを書き換えます。まず、　　ボタンをクリックして、次の　　ボタンもクリックして画面を進めていきます。


text ノードを編集
削除
中止
完了
プロパティ
選択
Group
新規に ui_group を追加...
Size
自動
Label
text
Value format
{{msg.payload}}
Layout
label value
labelvalue
labelvalue
label value
label value
Name


text ノードを編集 > 新規に dashboard group ノードの設定を追加
中止
追加
名前
デフォルト
選択
タブ
新規に ui_tab を追加...
幅
6
グループ名を表示する
Allow group to be collapsed

（次のウィンドウが開く）

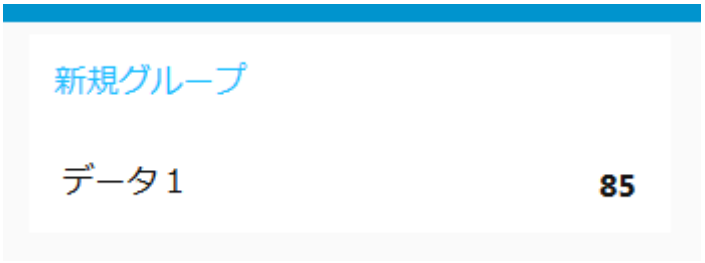

text ノードを編集 > 新規に dashboard group ノードの設定を追加 > 新規に dashboard tab ノードの設定を追加
中止
追加
名前
ホーム
アイコン
dashboard


text ノードを編集 > 新規に dashboard group
の設定を追加 > dashboard tab ノードを編集
削除
中止
更新
名前
新規タブ
アイコン
dashboard
新規タブ　と入力して更新

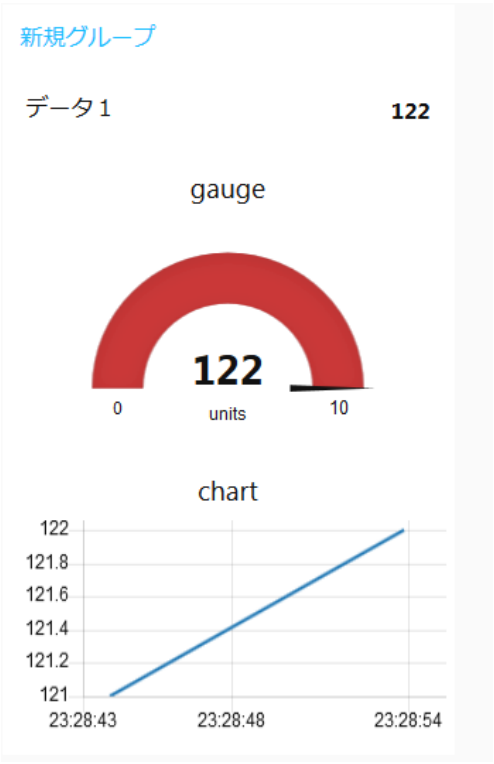

text ノードを編集 > 新規に dashboard group ノードの設定を追加
中止
追加
名前
新規グループ
タブ
新規タブ
幅
6
グループ名を表示する
Allow group to be collapsed
新規グループ　と入力して追加

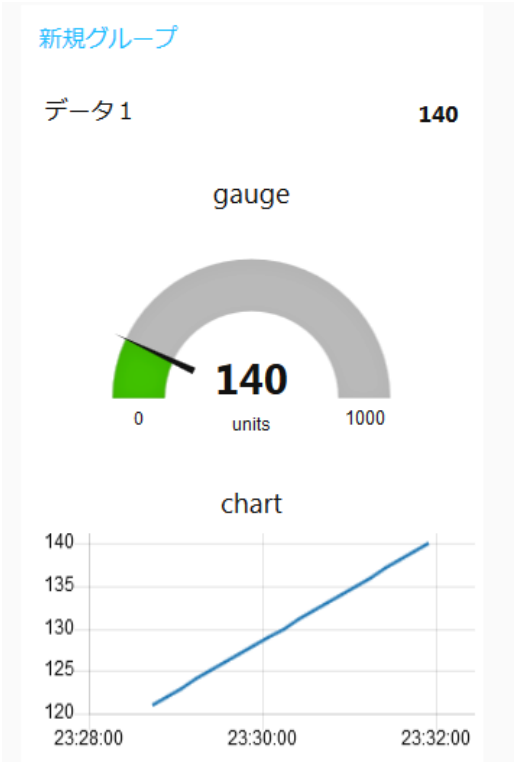
そうすると、右端のコンソールのダッシュボードタブに「新規タブ」と「新規グループ」、新しいデータの名前「データ１」が階層構造で表示されます。

以上で設定が終了したので、デプロイした後にダッシュボードタブのサイトボタンを押すと http://localhost:1880/ui が Web ブラウザが開かれて次のようにテキスト情報でデータをリアルタイムに可視化することができます。



同様に gauge と chart のノードを配置した後に、それぞれ先ほど設定した「新規グループ[新規タブ]」に設定して完了するとプロパティの設定が終了しまう。なお、プロパティを開くとあらかじめ「新規グループ[新規タブ]」が選ばれているように見えますが、完了ボタンを押して設定を終了してはじめて設定が完了となります。そうするとサイトボタンを押して開かれるダッシュボードに、左下図のようにテキストとゲージ、折れ線グラフの 3 つが表示されます。guage の range を設定すると右下図のように range で設定した範囲のゲージ内に値が収まります。なお、折れ線グラフは時間経過と共にグラフの横軸が拡張されていきます。





データの可視化に関しては、データを他のサービスと連携することで可視化を実現することもできます。例えば、クラウドサービスの Thinger.io（https://thinger.io/）では、最大デバイス数：2 や最大ダッシュボード数：4 などの条件付きの無料サービスがあります。

C) Node-RED からの LED 点灯

③で実施した LED の点灯を Node-RED 上で実施してみましょう。Node-RED では、WebSocket の双方向通信を使って実施します。前述の演習 3 の①で既に WebSocket の連携サービスを作成している人はそれをここでも使いましょう。作っていない人は演習 3 の①を参照して連携サービスを作成しましょう。

また、送信用の JSON 形式は演習 3 の③で作成したものを参考にしましょう（以下参照）。

{"type":"channels","module":"モジュールシリアル","payload":{"channels":[{"channel":0,"type":"i","value":1}]}}

では、次に Arduino 側にこの通信を受けるプログラムを Arduino IDE から書き込みます。ここでは、4 番のデジタルピンに白の LED を、7 番のデジタルピンに赤の LED を接続し、Incoming Webhook の内容でそれぞれを点灯させるようにします。

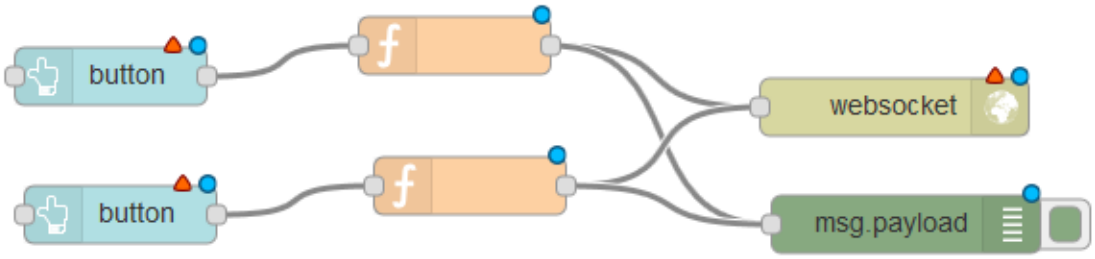
Arduino 側の Incoming Webhook の受信プログラム（再掲載）

```
#include <SakuraIO.h>
SakuraIO_I2C sakuraio;

void setup() {
  pinMode(7,OUTPUT);
  pinMode(4,OUTPUT);
  // シリアル通信を開始
  Serial.begin(9600);

  // オンラインになるまで待つ
  while ((sakuraio.getConnectionStatus() & 0x80) != 0x80) {
    Serial.print(".");
    delay(1000);
  }

  Serial.println("");
  Serial.println("Online");
}

void loop() {
  // 受信キューの状態を取得
  uint8_t rxAvailable;
  uint8_t rxQueued;
  sakuraio.getRxQueueLength(&rxAvailable, &rxQueued);

  digitalWrite(7,LOW);
  digitalWrite(4,LOW);
```

```
  // 受信キューにたまっているメッセージの数だけ繰り返す
  for (uint8_t i = 0; i < rxQueued; i++) {
    uint8_t channel;
    uint8_t type;
    uint8_t values[8];
    int64_t offset;

    // キューからのメッセージを取り出しに成功したら以下を実行
    uint8_t ret = sakuraio.dequeueRx(&channel, &type, values,
                                     &offset);
    if (ret == CMD_ERROR_NONE) {
      Serial.print("channel: ");
      Serial.println(channel);
      Serial.print("type: ");
      Serial.println((char)type);
      Serial.print("value: ");

      if (type == 'i') {
        // 32 ビット整数型の場合
        int32_t value = 0;
        memcpy(&value, &values, sizeof(int32_t));
        Serial.println(value);

        if (value == 1) { // value が 1 だったら 7 番ピンの LED を光らせる
          digitalWrite(7,HIGH);
          sakuraio.enqueueTx(0,7UL);
          sakuraio.send();
        }
        if (value == 2) { // value が 2 だったら 7 番ピンの LED を光らせる
          digitalWrite(4,HIGH);
          sakuraio.enqueueTx(0,4UL);
          sakuraio.send();
        }

      }
    } else {
      Serial.println("ERROR");
    }
  }

  delay(3000);
}
```

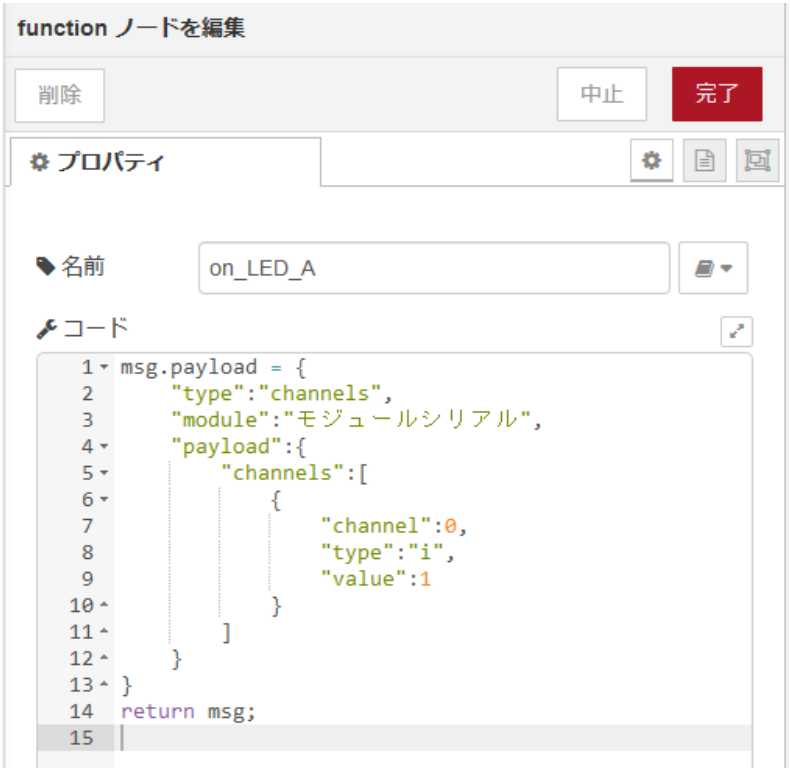
この例では、インターネット上から int 型整数が送信されてきて、その値が 1 なら 7 ピン（赤色 LED）を、値が 2 なら 4 ピン（白色 LED）のデジタルポートを HIGH に、すなわち LED を点灯させます。ここで、Curl コマンドが使用できる PC から先ほどの生成された Curl コマンドを含んだ JSON を Curl コマンドとして発行すると、i = 1 なので赤色 LED が点灯します。

次に、Node-RED のノード（button、function、websocket 出力、debug）を以下のように配置し、それぞれのノードを設定していきます。この例は value の 1 と 2 を送信するサンプルになります。



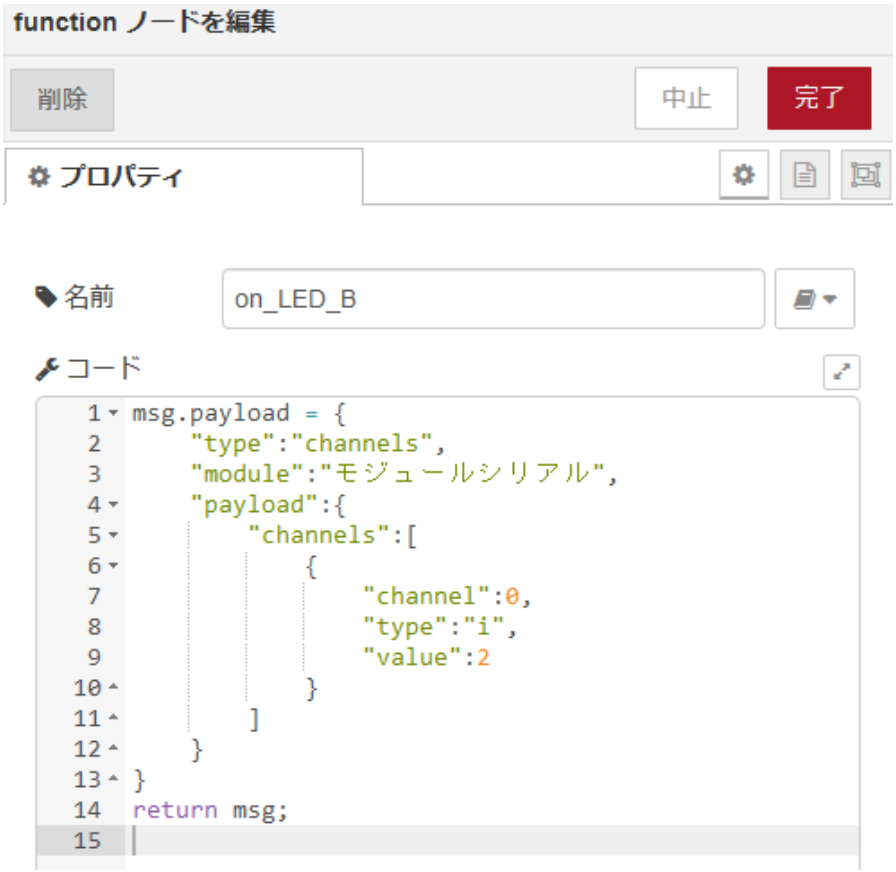
上側の button ノードと function ノードの設定を次のようにします。ここでは、グループを前の B)で作成したグループに登録しています。


button ノードを編集
削除
中止
完了
プロパティ
Group
新規グループ [新規タブ]
Size
自動
Icon
optional icon
Label
on_button_A
Colour
optional text/icon color
Background
optional background color
When clicked, send:
Payload
Topic
If msg arrives on input, pass through to output:
Name
グループ
名前


function ノードを編集
削除
中止
完了
プロパティ
名前
on_LED_A
コード
msg.payload = {
"type":"channels",
"module":"モジュールシリアル",
"payload":{
"channels":[
{
"channel":0,
"type":"i",
"value":1
}
]
}
}
return msg;

```
msg.payload = {
    "type":"channels",
    "module":"モジュールシリアル",
    "payload":{
        "channels":[
            {
                "channel":0,
                "type":"i",
                "value":1
            }
        ]
    }
}
return msg;
```

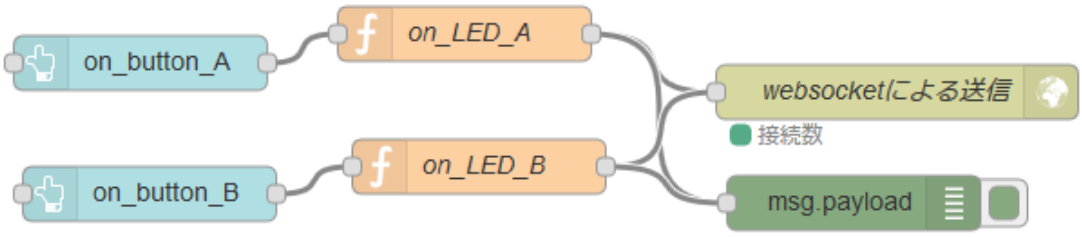
以上を参考に下側の button ノードと function ノードも設定していきます。



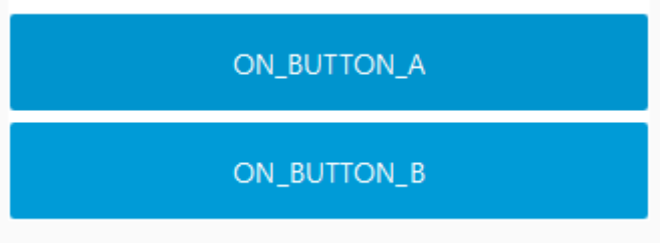

```
msg.payload = {
    "type":"channels",
    "module":"モジュールシリアル",
    "payload":{
        "channels":[
            {
                "channel":0,
                "type":"i",
                "value":2
            }
        ]
    }
}
return msg;
```

次に websocket 出力のノードの設定を以下のように行います。種類を［接続］、名前を任意に入力し、URL に websocket の URL を追記します。

設定が終わってデプロイするとノードの表示が以下のように変わります。



先の B)と同じようにダッシュボードのサイト画面を開くと以下のようなボタンが新しく追加されています。ボタンを押すとそれぞれに対応した Arduino に連結された LED が点灯します。



演習４　総合演習

これまで学んだものに基づいて各自の IoT システムを構築してみましょう。

【必須】
・IoT デバイスに任意のセンサを利用しましょう
・取得したセンサの値を sakura.io にアップロードしてみましょう

【任意】
A. センサを複数にする
B. sakura.io に集めたデータを可視化する
C. IoT デバイスへのフィードバック機能を任意につける
D. IoT デバイスにアクチュエータをつける
E. その他
・IoT を利用したビジネスモデルについて考えてみる。
・sakura.io 以外で自社利用に適した IoT プラットフォームを探してみる。
・復習として演習課題の最初から最後まで一人で作成してみる。
・まだ使っていないセンサやアクチュエータを試用してみる。
・複数のセンサでデータを集めてそれらの値を Node-RED のダッシュボードに表示すると共に、何らかの統計値も一緒に表示させるシステムを作成してみる。
・Node-RED でファイルに書き出したセンサのデータを Chart.js などの JavaScript ライブラリを使ってグラフ化してみる。
・Node-RED の Tweet ノードを利用してノードレットから Tweet してみる。
※ツイッターのアカウントは各自で準備してください。
・「OSOYOO 公式チュートリアル」Web サイトを参考にして、Node-RED でダッシュボードの数字が書かれたボタンを押されると、Arduino に接続した 1 桁 LED デジタル表示管にボタンの数字が表示されるシステムを作成してみる。
・XAMPP（https://www.apachefriends.org/jp/index.html）などの PHP 開発環境を利用して Web サーバなどを立ち上げ、Arduino と連携した任意の Web システムを作成してみる。

「OSOYOO 公式チュートリアル」Web サイト
http://osoyoo.com/ja/2014/12/06/arduino-starter-kit/