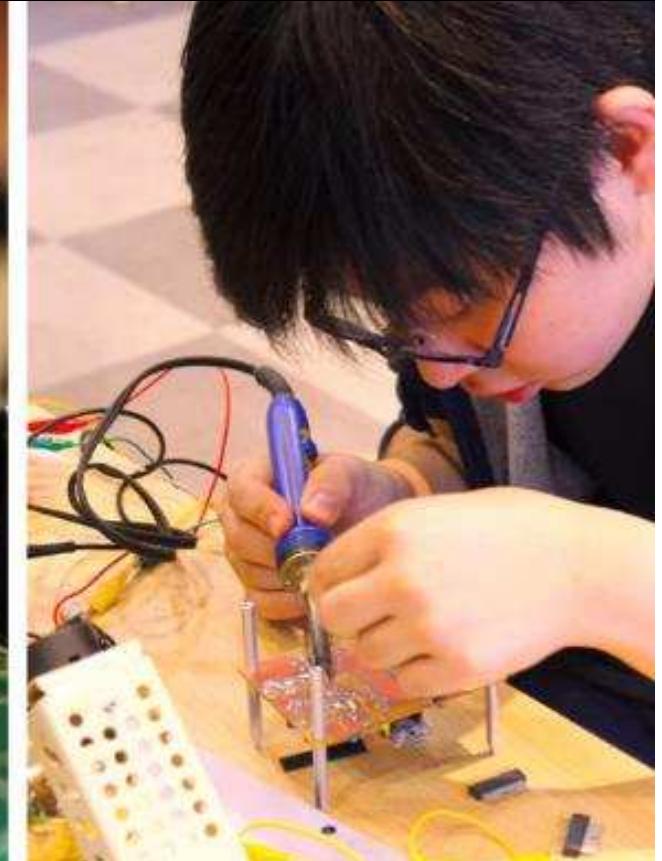




# Web×IoTメイカーズチャレンジ ハンズオン講習テキスト



**CHIRIMEN Open Hardware**  
<https://chirimen.org/>



# 目 次

## ① Lチカしてみよう (初めての GPIO)

1. はじめに
2. 事前準備 (機材確認)～基本ハードウェア
3. 事前準備 (機材確認)～電子パーツ
4. CHIRIMEN for Raspberry Pi 3 を起動してみよう  
接続方法  
起動確認  
残念ながら上記画面が表示されなかった！？  
WiFi の設定
5. 「Lチカ」をやってみよう  
そもそも「Lチカ」って何?  
ブレッドボードの使い方／LEDの使い方  
配線してみよう  
example を実行してみる  
コードを眺めてみよう  
JSBIN の example を起動  
JSBIN の 画面構成  
「Lチカ」のコード HTML  
「Lチカ」のコード JavaScript  
処理の解説  
うまくいかなかつたら?  
非同期処理について／GPIO とは  
GPIOPort の処理まとめ
6. ここまでまとめ

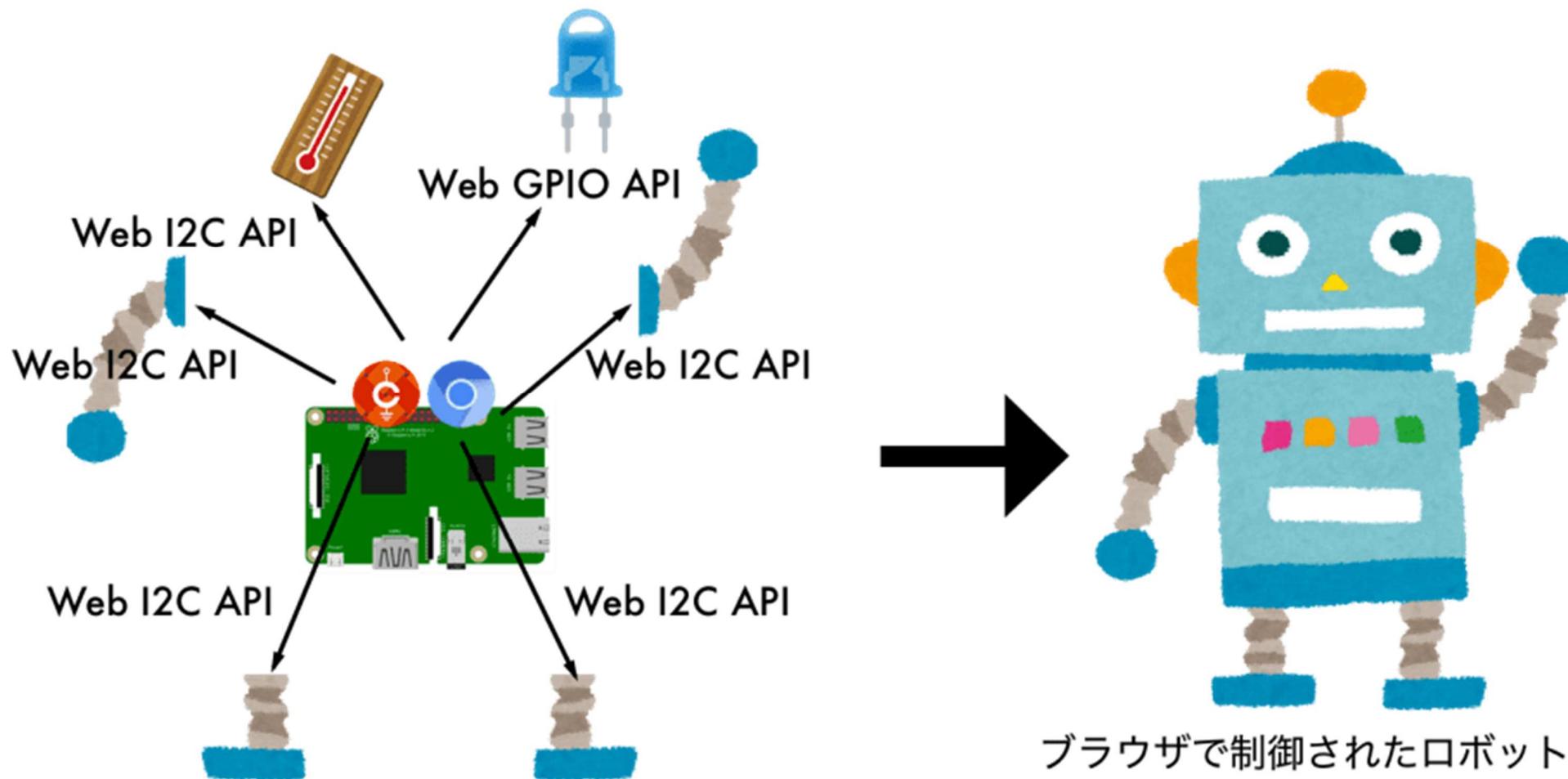
## ② GPIO の使い方

7. GPIOの使い方  
用意するもの  
CHIRIMEN for Raspberry Pi 3 の起動と Lチカの確認  
マウスクリックで LED の ON/OFF を制御してみる  
JSFiddle の 画面構成  
HTML/CSS を記載する  
ボタンに反応する画面を作る  
ボタンに LED を反応させる  
画面のボタンをモーメンタリ動作に変えておく  
マウスクリックのかわりにタクトスイッチを使ってみる  
部品と配線について  
スイッチは「プルアップ」回路で接続  
スイッチに反応するようにする (port.onchange())
8. LED のかわりに ギアモータ（ちびギアモータ）を動かしてみる  
部品と配線について  
コードは... 書き換えなくて良い  
MOSFET とは
9. ここまでまとめ
10. 付録  
Node.js + NODE-RED + GrovePiを利用した開発について  
トラブルシューティング  
用語集



# はじめに

CHIRIMEN for Raspberry Pi 3 は、Raspberry Pi 3 (以下 Raspi) で動作する IoT プログラミング環境です。[Web GPIO API](#) や [Web I2C API](#) といった JavaScript でハードを制御する API を活用したプログラミングにより、Web アプリ上で Raspi に接続した電子パーツを直接制御できます。

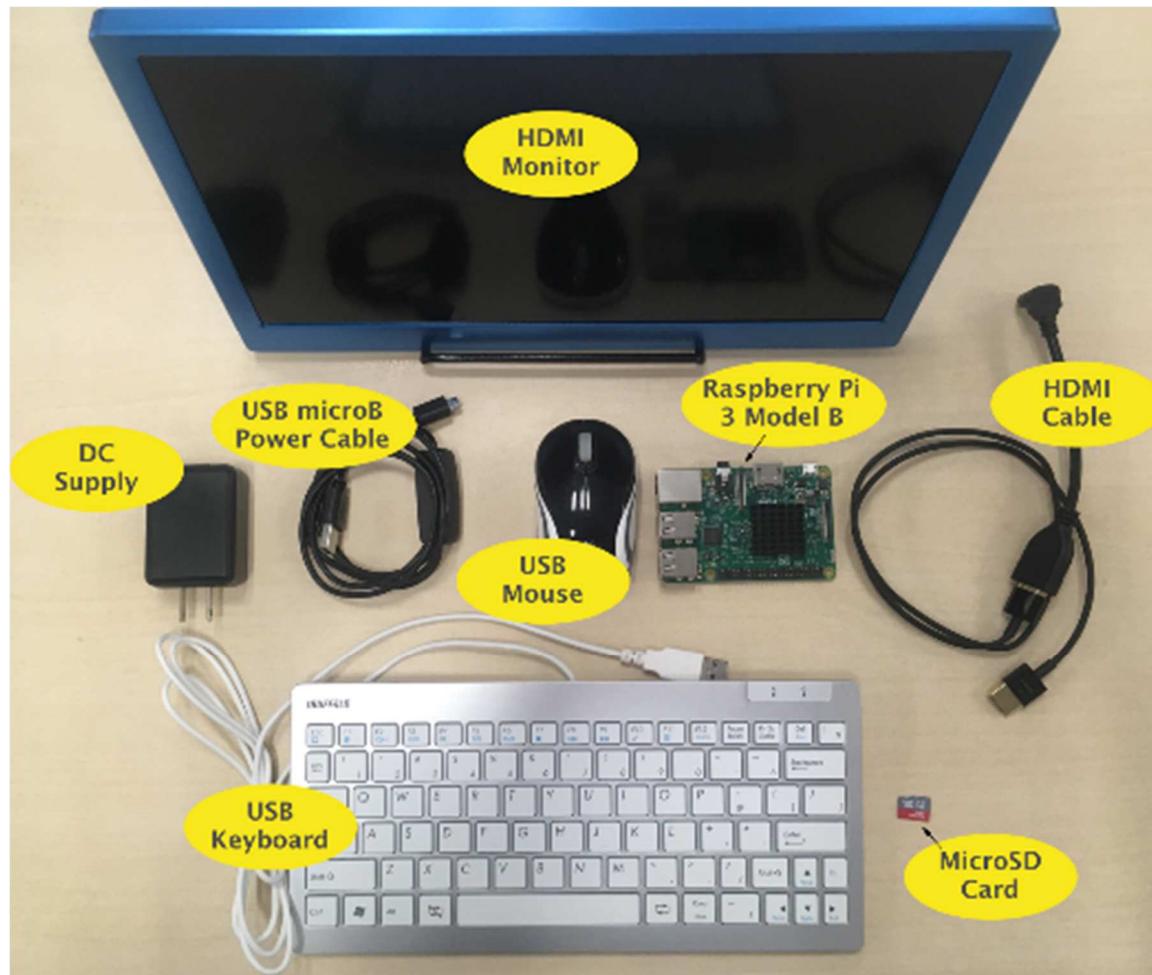




# 事前準備（機材確認）～基本ハードウェア

## 基本ハードウェア

下記が CHIRIMEN for Raspberry Pi 3 の起動に最低限必要となる**基本ハードウェア**です。



- [Raspberry Pi 3 Model B+](#) × 1
- AC アダプタ + micro B USB ケーブル × 1  
例: Raspberry Pi 用電源セット(5V 3.0A)
- HDMI 入力のモニタ (720P解像度対応) × 1
- HDMI ケーブル× 1
- USB マウス × 1
- USB キーボード (日本語配列) × 1
- [CHIRIMEN起動イメージ](#)入りの micro SD カード (8GB 以上必須、Class 10 以上で高速なもの を推奨) × 1

※上記の機器一式はハンズオン講習中は全員にレンタルします。



# 事前準備（機材確認）～電子パーツ

## ハンズオン講習で使用する電子パーツ

**CHIRIMEN for Raspberry Pi 3  
スター・キット**

① ブレッドボード × 1      ③ LED × 1      ④ MOSFET × 1      ⑤ タクトスイッチ × 1      ⑥ 温度センサー <ADT7410> × 1

② ちびギアモーター × 1      ⑦ 10kΩ 抵抗 <茶黒橙金> × 1      ⑧ 1kΩ 抵抗 <茶黒赤金> × 1      ⑨ 150Ω 抵抗 <茶緑茶金> × 1

⑩ ジャンパーウイヤー <オス-メス> × 5      ⑪ ジャンパーウイヤー <メス-メス> × 4      ⑫ ジャンパーウイヤー <オス-オス> × 4

⑬ micro SDカード × 1

**★ チュートリアル情報**  
このスター・キットのパーツで Hello World. チュートリアル1と2まで進められます。  
<https://tutorial.chirimen.org/>

- ① ブレッドボード × 1
- ② ちびギアモーター × 1
- ③ LED × 1
- ④ MOSFET × 1
- ⑤ タクトスイッチ × 1
- ⑥ 温度センサー <ADT7410>
- ⑦ 10kΩ 抵抗 <茶黒橙金> × 1
- ⑧ 1kΩ 抵抗 <茶黒赤金> × 1
- ⑨ 150Ω 抵抗 <茶緑茶金> × 1
- ⑩ ジャンパーウイヤー (オス-メス) × 5
- ⑪ ジャンパーウイヤー (メス-メス) × 4
- ⑫ ジャンパーウイヤー (オス-オス) × 4
- ⑬ Micro SDカード × 1

# CHIRIMEN for Raspberry Pi 3 を起動してみよう

## 接続方法

機材が揃ったら、いよいよ Raspi を接続して起動してみましょう。基本ハードウェアを下図のように接続してください。(Raspi への電源ケーブルの接続は最後にしましょう)



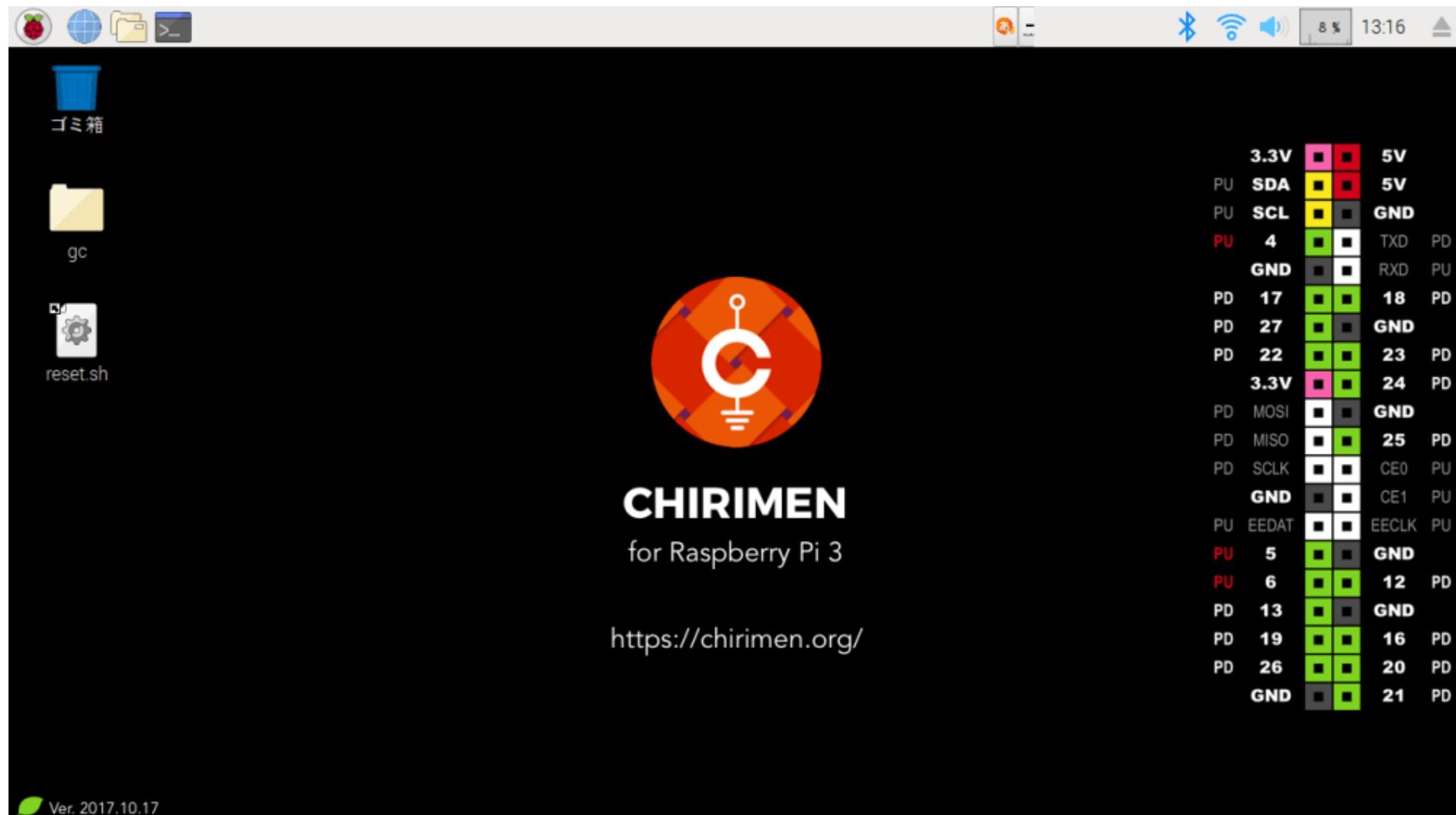
1. モニタ電源をコンセントに接続
2. RaspiにmicroSDカードを挿入
3. HDMIケーブルでRaspiとモニタを接続
4. マウスをUSBポートへ接続
5. キーボードをUSBポートへ接続
6. microUSB電源ケーブルをコンセントへ
7. microUSB電源ケーブルをRaspiに接続



# CHIRIMEN for Raspberry Pi 3 を起動してみよう

## 起動確認

電源を入れると Raspi の microSD コネクタ横の赤い LED が点灯し、OS の起動後、下記のようなデスクトップ画面が表示されたら CHIRIMEN Raspi3 の起動に成功しています。おめでとうございます！





# CHIRIMEN for Raspberry Pi 3 を起動してみよう

残念ながら上記画面が表示されなかった！？

電源を入れても何も画面に映らないような場合には、配線が誤っている可能性があります。配線を再度確認してみましょう。LED が点灯していない場合、AC アダプタまで電気が来ていないかも知れません。[トラブルシューティングページ](#) も参考にしてください。

## WiFi の設定

デスクトップ画面が表示されたら、さっそく WiFi を設定して、インターネットに繋げてみましょう。CHIRIMEN Raspi3 では、ネットワークに繋がなくてもローカルファイルを使ったプログラミングが一応可能ですが、[JS Bin](#) や [JSFiddle](#) などの Web 上のエディタを活用することで、より便利にプログラミングが進められるようになります。また、CHIRIMEN Raspi3 に関する情報も今後インターネット上で充実していく予定です。

ぜひ、最初にインターネット接続環境を整えておきましょう。

WiFi の設定は、タスクバーの右上の WiFi アイコンから行えます。

無線通信の設定

SSID :

PASS :

※忘れないようにメモしましょう。

ここからWIFIを設定する





# 「Lチカ」をやってみよう

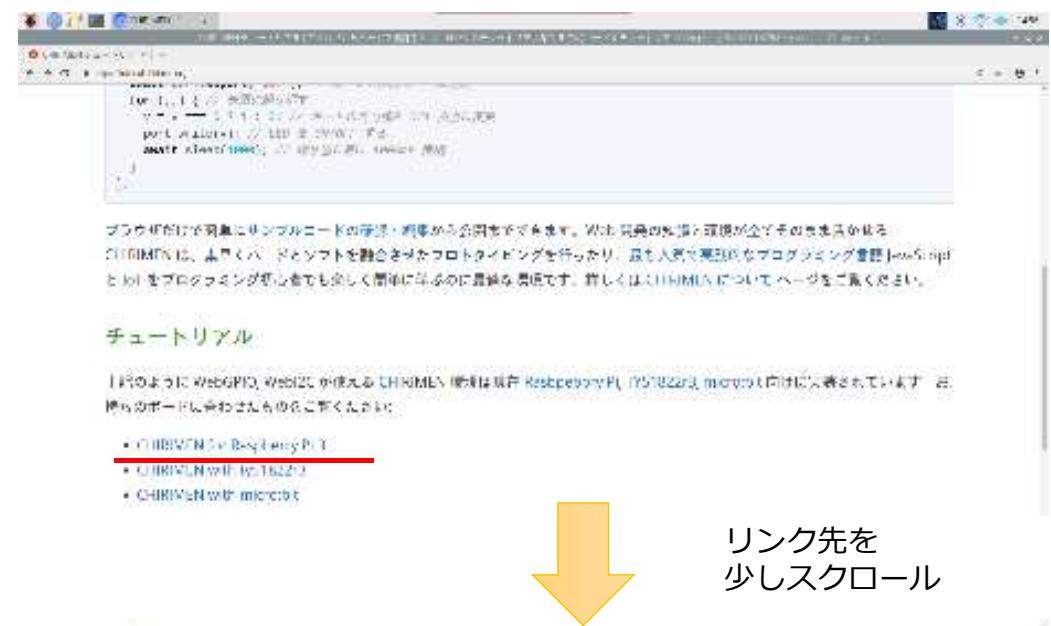
## そもそも「Lチカ」って何？

「Lチカ」とは、LEDを点けたり消したり、チカチカ点滅させることです。  
今回は「LEDを点ける」「LEDを消す」をプログラムで繰り返し実行することで実現します。

チュートリアルページを参考にして、回路を接続し、サンプルプログラムを動かしてみましょう。



画面に出てる「ちゅーとりある！」から  
リンクをたどっていきましょう。



リンク先を  
少しスクロール

- まずはシンプルなGPIO入出力(1)センサーの操作力を学びましょう
- 1. Lチカしてみよう(初めてGPIO)
    - 基本GPIO APIを使って定期的にGPIOを操作するサンプルを動かしてみます
  - 2. GPIOの使い方
    - マルチタスクで動作するフックサブルのボタンと喇叭スイッチ(タクトスイッチ)の両方でLEDやモーターを操作するサンプルをはじめてみます
    - 2. カンリーライブ(読み込み用GPIO)
    - 溫度センサーの値をドライブAPIを使う場合をGPIO APIを直接操作する場合の2パターンで読み取ることで、GPIOデバイス操作の基本を学びます

応用編



# 「Lチカ」をやってみよう

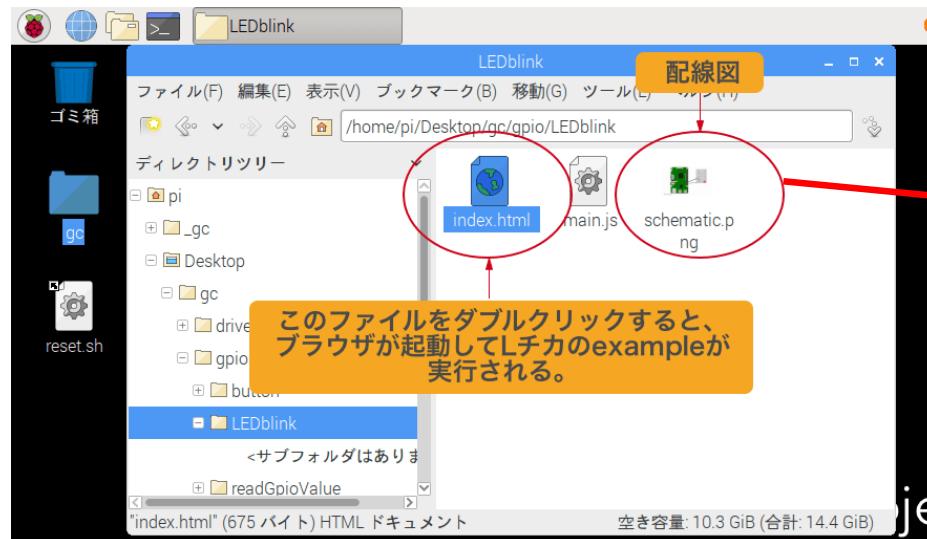
## そもそも「Lチカ」って何？

LED を点けたり消したりするためには、Raspi と正しく配線する必要があります。

LED は 2 本のリード線が出ていますが、長い方がアノード (+側) 、短い側がカソード (-側) です。Lチカのための配線図は、基本的な作例集 (examples) と一緒に、下記にプリインストールされています。

/home/pi/Desktop/gc/gpio/LEDblink/schematic.png

LED のリード線の方向に注意しながら、この図の通りにジャンパーウイヤやブレッドボードを使って配線してみましょう。

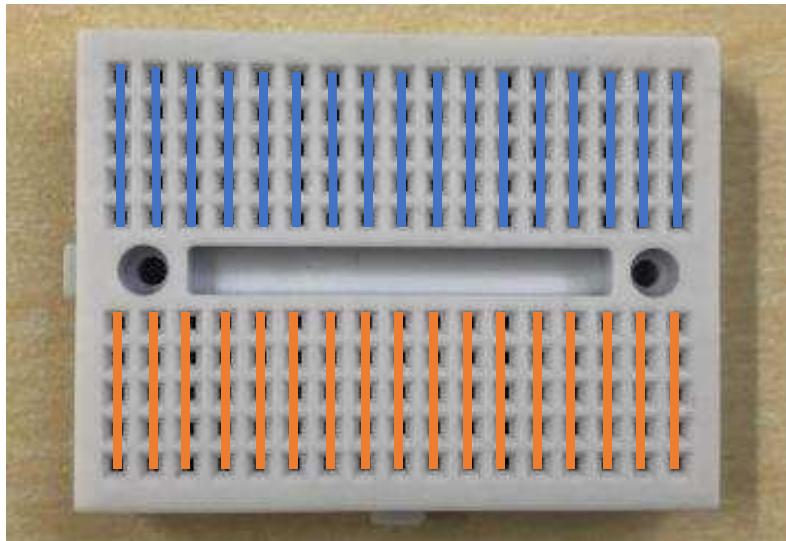




# 「L チカ」をやってみよう

## ■ ブレッドボードの使い方

表面

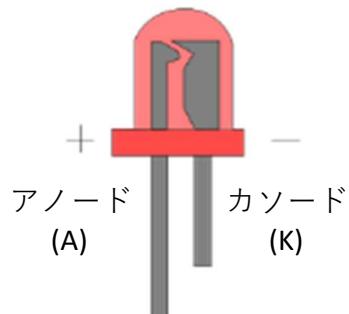


裏面

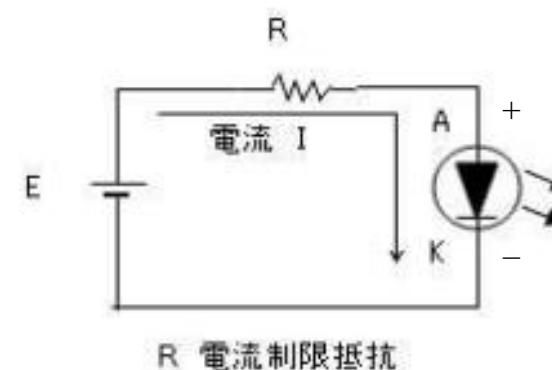
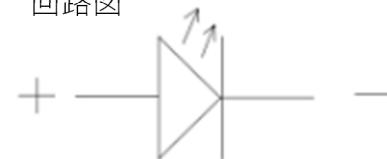


写真の様に1列5pinでつながっています。電子パーツを差し込んで回路を作ります。  
※少し硬い場合もあります、しっかり中まで差し込みましょう。

## ■ LED（発光ダイオード）の使い方



回路図

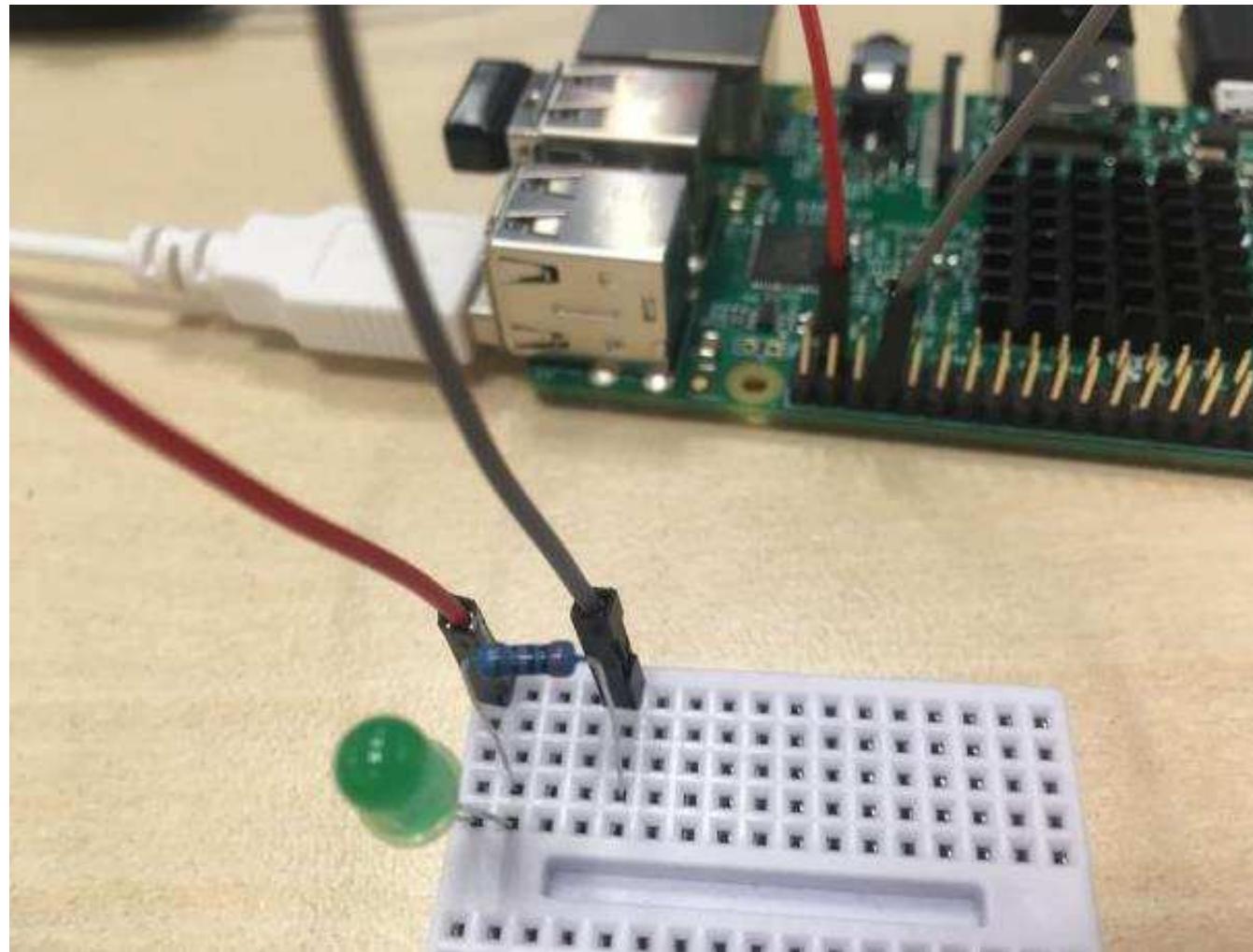


基本回路)

LEDも電流が流れる極性があり、  
基本回路のようにアノードに電源  
のプラス側を接続すれば電流が流  
れて点灯します。

# 「L チカ」をやってみよう

実際に配線してみると、こんな感じになりました。



配線図では LED の下側のリード線が折れ曲がり長い方がアノード (+側) です。図 (schematic.png) で使用されている抵抗は"150Ω"です。使用する抵抗は、LED にあったものを使用してください。



# 「L チカ」をやってみよう

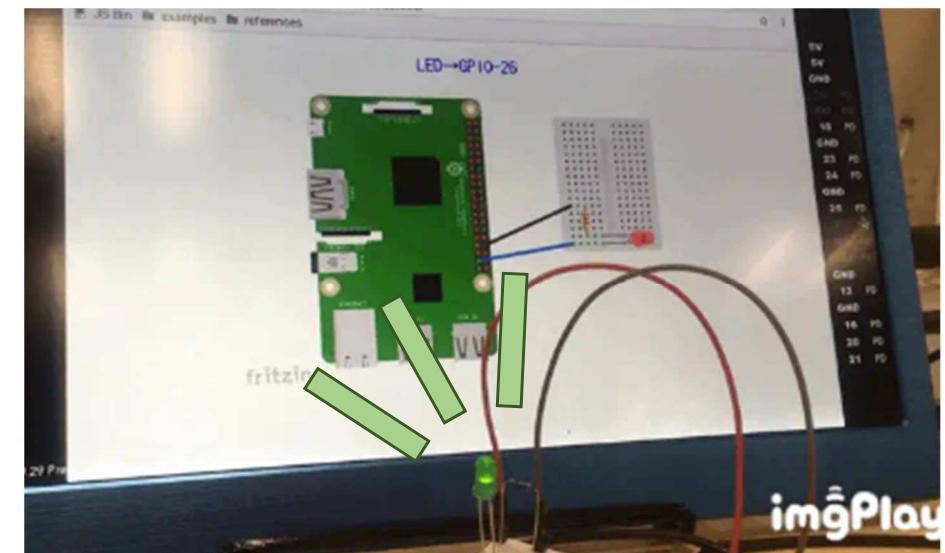
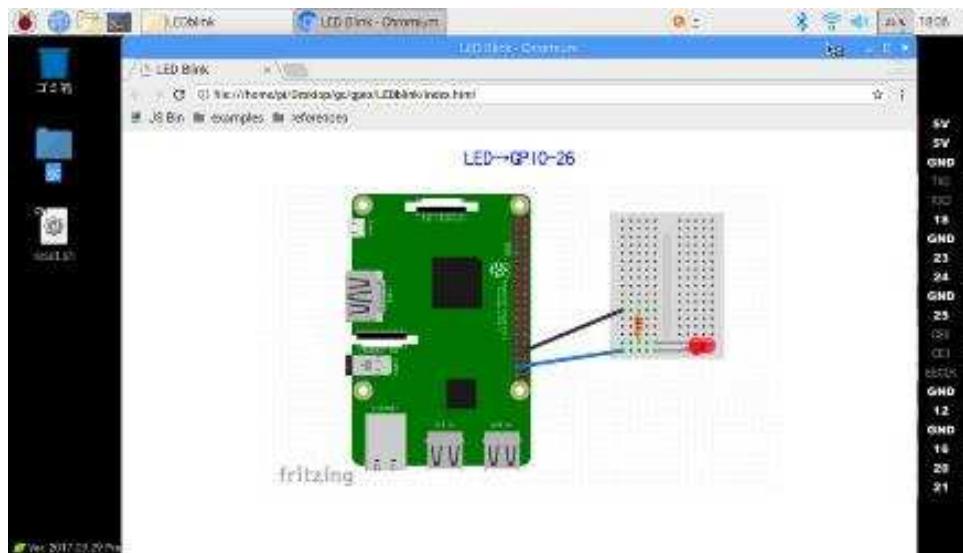
## example を実行してみる

配線がうまくできたら、さっそく動かしてみましょう。 L チカのためのサンプルコードは先ほどの配線図と同じフォルダに格納されています。

```
/home/pi/Desktop/gc/gpio/LEDblink/index.html
```

index.html をダブルクリックすると、ブラウザが起動し、先ほど配線した LED が点滅します！

## ブラウザ画面



※チュートリアルサイトで動画で確認ができます。

L チカに成功しましたか？！



# 「L チカ」をやってみよう

## コードを眺めてみよう

CHIRIMEN Raspi3 にはもうひとつ、「オンラインの example」が用意されており、オンライン版ではコードを書き換えながら学習を進められます。

今度はオンラインの example からさきほどと同じ L チカを実行してコードを眺めてみましょう。その前に一つ注意です。

**オンラインの example を起動する前に、必ず先ほど開いたブラウザウィンドウ（タブ）は閉じてください。先に既存ウィンドウを閉じておかないと、サンプルが正常に動作しない※なります。**



※CHIRIMEN Raspi3 での GPIO などの操作には排他制御があり、同一の GPIO ピンを複数のプログラムから同時操作はできません。

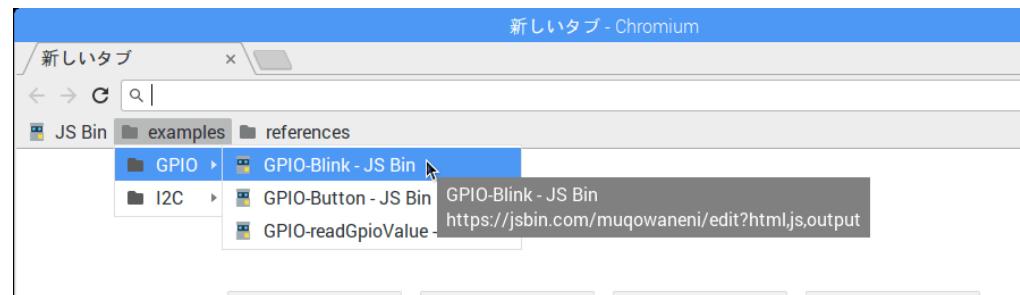


# 「Lチカ」をやってみよう

JS Bin の example を起動 ※ 配線は、さきほどのままで OK です。

ブックマークバーから、以下を選んでアクセスしてください。

examples > GPIO-JSBIN > GPIO-Blink - JS Bin



そのまま起動すると右図のような画面になります。  
(スクリーンショットはアクセス直後の画面から  
JS Bin のタイトルバー部の「Output」タブ非表  
示にしています)

それでは、コードを眺めてみましょう。

The screenshot shows the JS Bin Collaborative interface. The title bar reads "JS Bin - Collaborative...". The main window displays the "GPIO-Blink - JS Bin" example. The left pane shows the HTML and JavaScript code for the blink script. The right pane shows a detailed pinout diagram for a Raspberry Pi, mapping pins 1 through 28 to various functions like 3.3V, 5V, GND, SDA, SCL, etc.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>GPIO-Blink</title>
</head>
<body>
<script src="https://mz4u.net/libs/gc2/polyfill.js"></script>
</body>
</html>
```

```
navigator.requestGPIOAccess().then(gpioAccess=>{
  var port = gpioAccess.ports.get(26);
  var v = 0;
  return port.export("out").then(()=>{
    setInterval(function(){
      v *= 1;
      port.write(v);
    },1000);
  });
});
```



# 「Lチカ」をやってみよう

## JS Bin の 画面構成

JS Binとは、ブラウザ上でJavaScriptを動作させる事が出来るサービスです。いくつかのフレームワークにも対応しています。その他、類似のサービスが複数（JSFiddleなど）あります。

JS Bin の画面構成について解説します。

- <ライブラリ読み込みボタン>**  
JS.Binで用意されたライブラリを読み込みます。  
※本チュートリアルでは使わない
- <画面切り替えボタン>**  
対応した機能のフレームを表示します。
- <コードを記述するフレーム> (HTMLペイン、CSSペイン、JavaScriptペイン)**  
フレーム内に対応した言語でコーディングします。
- <実行結果を確認するフレーム> (Console、Output)**  
実行結果を見るフレームを表示します。

LED→GPIO-26

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta content="width=device-width, user-scalable=no" name="viewport" />
    <title>LED Blink</title>
    <script src="https://r.chirimen.o...</script>
    <style>
      p {
        color: blue;
        text-align: center;
        font-size: 24px;
      }
      img {
        display: block;
        margin-left: auto;
        margin-right: auto;
      }
    </style>
  </head>
  <body>
    <p>LED→GPIO-26</p>
  </body>
</html>
```

```
main();
```

```
async function main() {
  var gpioAccess = await navigator.requestGPIOAccess();
  var port = gpioAccess.ports.get(26);
  var v = 0;

  await port.export("out"); // ポートを出力モードに
  for (;;) {
    v = v === 0 ? 1 : 0; // ポートの出力を反転
    port.write(v); // LED を ON/OFF する
    await sleep(1000); // 繰り返し毎に
  }
}
```



# 「L チカ」をやってみよう

## HTML

HTMLのコーディングを、以下の様に記述します。ここでは必要なライブラリを読み込みます。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta content="width=device-width" name="viewport">
    <title>GPIO-Blink</title>
  </head>
  <body>
    <script src="https://r.chirimen.org/polyfill.js"></script>
    <script src="s0.js"></script>
  </body>
</html>
```

オレンジでマークしたラインで、**polyfill.js** ※という JavaScript ライブラリを読み込んでいます。これを最初に読み込むと、それ以降のコードで GPIO や I2C を操作する JavaScript API が使えるようになります。

※チュートリアルページにコードサンプルが載っています。

<https://tutorial.chirimen.org/raspi3/section0#html>

※polyfill.js : これは Web GPIO API と、Web I2C API という W3C でドラフト提案中の 2 つの API への Polyfill (新しい API を未実装のブラウザでも同じコードが書けるようにするためのライブラリ) になります。

※ローカル環境にあるexample では、インターネット未接続時にも動作するよう Polyfill を含めたコード一式をローカルにコピーしてあります。node\_modules/@chirimen-raspi/polyfill/polyfill.js で読み込みます。

オンラインにホストされている最新版を読み込む場合には https://r.chirimen.org/polyfill.js を指定します。



# 「Lチカ」をやってみよう

## 「Lチカ」のコード JavaScript

JavaScript のコーディングを以下の様に記述します。

```
async function mainFunction() {
    // プログラムの本体となる関数。非同期処理を await で扱えるよう全体を async 関数で包みます
    ① var gpioAccess = await navigator.requestGPIOAccess(); // 非同期関数は await を付けて呼び出す
    var port = gpioAccess.ports.get(26);
    ② var v = 0; // 0(Off)と1(On)を切り替えるための変数を定義

    await port.export("out");
    for (;;) {
        // 無限ループ
        ③ await sleep(1000); // 無限ループの繰り返し毎に 1000ms 待機する
        v = v === 0 ? 1 : 0; // vの値を0,1に入れ替える。1で点灯、0で消灯するので、1秒間隔でLEDがON OFFする
        port.write(v);
    }
}

// await sleep(ms) と呼べば指定 ms 待機する非同期関数
// 同じものが polyfill.js でも定義されているため省略可能
function sleep(ms) {
    return new Promise(resolve => {
        setTimeout(resolve, ms);
    });
}

mainFunction(); // 定義したasync関数を実行します（このプログラムのエントリーポイント）
```

※チュートリアルページにコードサンプルが載っています。

<https://tutorial.chirimen.org/raspi3/section0#javascript>



# 「L チカ」をやってみよう

## 処理の解説

- ① 最初に呼び出されるコードは `await navigator.requestGPIOAccess()` です。  
ここで先ほど出て来た [Web GPIO API](#) を使い、`gpioAccess` という `GPIO※` にアクセスするためのインターフェースを取得しています。
- ② `var port = gpioAccess.ports.get(26);`  
で、**GPIO** の **26 番ポート**にアクセスするためのオブジェクト を取得しています。続いて、  
`await port.export("out")` で `GPIO` の 26 番を「出力設定」にしています。  
これにより LED への電圧の切り替えが可能になっています。  
ポート方向は「出力設定 ("out")」と「入力設定("in")」を持っています。
- ③ 最後に `await sleep(1000)` で 1000ms = 1 秒 待機させて無限ループをつくることで、  
1 秒毎に `port.write(1)` と `port.write(0)` を交互に呼び出し、`GPIO 26 番`に対する電圧を  
`3.3V → 0V → 3.3V → 0V` (ループ) と繰り返し設定しています。
- ④ LED は一定以上の電圧 (赤色 LED だと概ね 1.8V 程度、青色 LED だと 3.1V 程度) 以上になると  
点灯する性質を持っているので、`3.3V` になったときに点灯、`0V` になったときに消灯を繰り返す  
ことになります。Raspi のポート出力電圧は`3.3V`になります。



# 「L チカ」をやってみよう

## 非同期処理について

物理デバイス制御やネットワーク通信などを行う際には、応答待ち中にブラウザが停止しないよう非同期処理を使う必要があります。本チュートリアルではこれを **async** 関数で記述しています。

※必要に応じて「[非同期処理 \(async await 版\)](https://tutorial.chirimen.org/raspi3/appendix0)」も参照してください。<https://tutorial.chirimen.org/raspi3/appendix0>

本チュートリアルでは次のルールでコードを書けば大丈夫です：

### 1. 非同期関数の呼び出し時には前に `await` を付けて呼び出す

- ・非同期関数呼び出し前に `await` を付けるとその処理が終わってから次のコードが実行されます
- ・GPIO や I2C の初期化、ポートの設定などは非同期関数なので `await` キーワードを付けて呼び出します

### 2. 非同期処理を含む関数は前に `async` を付けて非同期関数として定義する

- ・`async function` 関数名() { ... } のように頭に `async` を付けると非同期関数となります

## GPIO とは

GPIOは、「General-purpose input/output」の略で汎用的な入出力インターフェースのことです。

Raspi3 に実装されている 40 本のピンヘッダから GPIO を利用することができます。  
(40 本全てのピンが GPIO として利用できるわけではありません)

CHIRIMEN Raspi3 では Raspi3 が提供する 40 本のピンヘッダのうち、下記緑色のピン(合計 17 本)を Web アプリから利用可能な GPIO として設定しています。

参考情報 (<https://tool-lab.com/make/raspberrypi-startup-22/>)

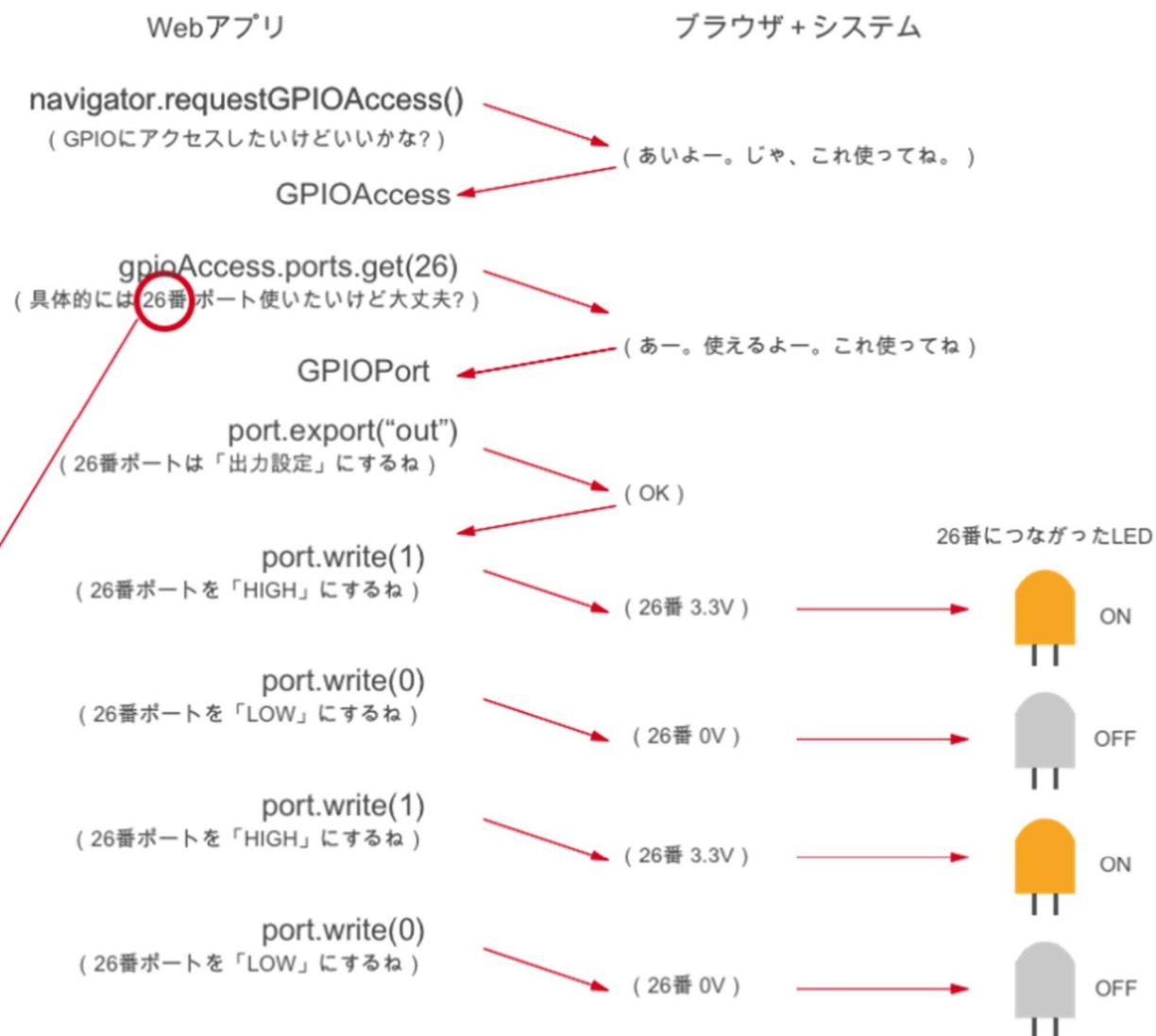
3.3V	[Pink]	[Black]	5V	[Red]	[Black]	GND	[Black]
SDA	[Yellow]	[Black]					
SCL	[Yellow]	[Black]					
4	[Green]	[Black]					
GND	[Black]	[Black]					
17	[Black]	[Black]					
27	[Black]	[Black]					
22	[Black]	[Black]					
3.3V	[Black]	[Black]					
MOSI	[Black]	[Black]					
MISO	[Black]	[Black]					
SCLK	[Black]	[Black]					
GND	[Black]	[Black]					
EEDAT	[Black]	[Black]					
5	[Green]	[Black]					
6	[Green]	[Black]					
13	[Green]	[Black]					
19	[Green]	[Black]					
26	[Green]	[Black]					
GND	[Black]	[Black]					
12	[Green]	[Black]					
GND	[Black]	[Black]					
16	[Green]	[Black]					
20	[Green]	[Black]					
21	[Black]	[Black]					



# 「Lチカ」をやってみよう

## GPIOPort の処理まとめ

3.3V	[Pink]	[Red]
SDA	[Yellow]	[Black]
SCL	[Yellow]	[Black]
4	[Green]	[White]
GND	[Black]	[Black]
17	[Green]	[Black]
27	[Green]	[Black]
22	[Green]	[Black]
3.3V	[Pink]	[Green]
MOSI	[White]	[Black]
MISO	[White]	[Green]
SCLK	[Black]	[Black]
GND	[Black]	[Black]
EEDAT	[White]	[Black]
5	[Green]	[Black]
6	[Black]	[Black]
13	[Black]	[Black]
19	[Green]	[Black]
26	[Green]	[Black]
GND	[Black]	[Green]
20	[Black]	[Black]
21	[Black]	[Green]



JS Bin の JavaScript のペイン (コードが表示されているところ) をクリックするとカーソルが表示され  
コード修正も可能です。 試しにいろいろ変えてみましょう。



# 「L チカ」をやってみよう

## ここまでのまとめ

このチュートリアルでは、下記を実践してみました。

- CHIRIMEN for Raspberry Pi 3 の起動
- L チカのサンプルを動かしてみた
- JS Bin で L チカのコードを変更してみた

このチュートリアルで書いたコードは以下のページで参照できます:

- [GitHub リポジトリで参照](#)  
(<https://github.com/chirimen-oh/tutorials/tree/master/raspi3/examples/section0>)



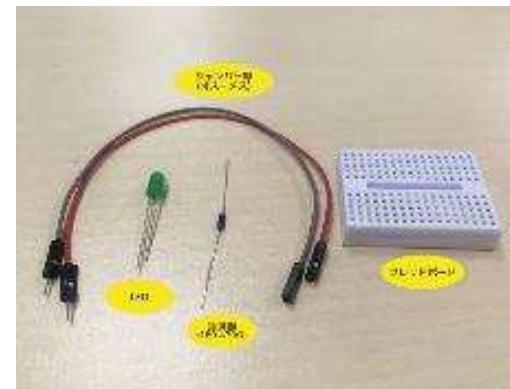
# GPIO の使い方

## GPIO の使い方

CHIRIMEN for Raspberry Pi 3 (以下 CHIRIMEN Raspi3) を使ったプログラミングを通じて、Web GPIO API の使い方を学びます。

## 用意するもの

このチュートリアル全体で必要になるハードウェア・部品は下記の通りです。



「基本ハードウェア」と「L チカに必要となるパーツ」



- タクトスイッチ (2pin) × 1
- ジャンパーケーブル (オス-メス) × 5
- Nch MOSFET (2SK4017)
- リード抵抗 (1KΩ) × 1
- リード抵抗 (10KΩ) × 1
- ちびギアモータ × 1



# GPIO の使い方

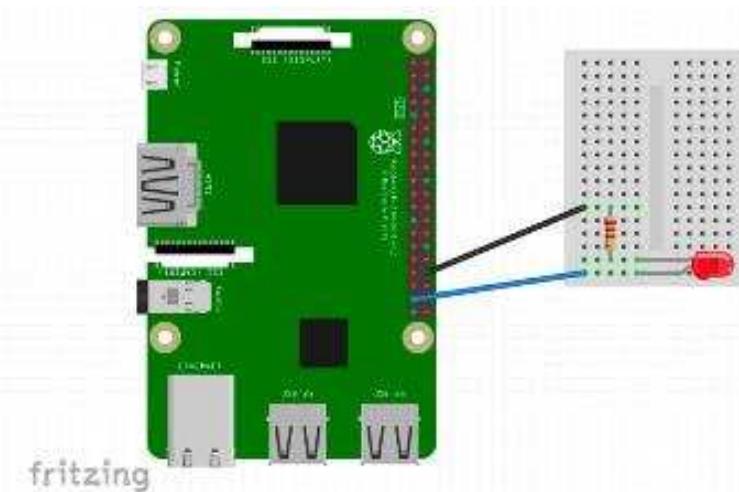
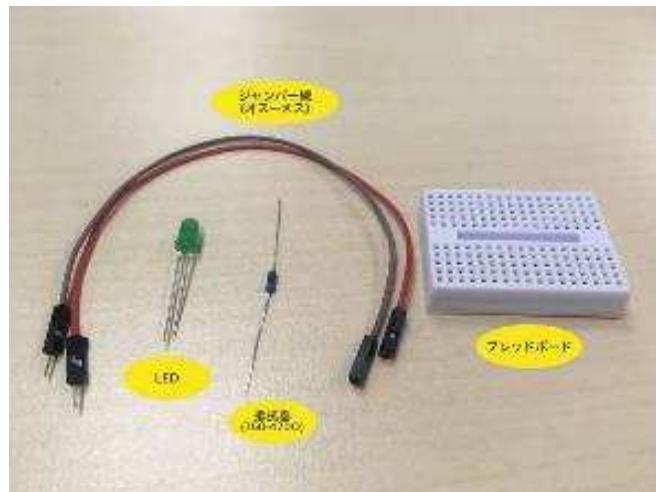
## マウスクリックで LED の ON/OFF を制御してみる

Lチカしてみようでは、**JS Bin** を使ってLチカのexampleコードを少し触つてみるだけでしたが、今度は最初から書いてみます。  
今回は他のオンラインエディタ [JSFiddle](#) を使ってみましょう。

Web 上のオンラインサービスは便利ですが、メンテナンスや障害、サービス停止などで利用できなくなることがあります。ローカルでの編集も含め、いくつかのサービスを使いこなせるようにしておくと安心です。

各サービスにはそれぞれ一長一短がありますので、利用シーンに応じて使い分けると良いかもしれません。

このパートでは「Lチカしてみよう」で実施したLチカの配線をそのまま利用します。必要な部品も同じです。  
LEDは、**26番ポート**に接続しておいてください。





# GPIO の使い方

## JSFiddle の 画面構成

JSFiddleとは、ブラウザ上でJavaScriptを動作させる事が出来るサービスです。いくつかのフレームワークにも対応しています。前に使用した JS BIN も類似のサービスになります。

The screenshot shows the JSFiddle interface with the following components:

- Fiddle meta**: Left sidebar with options like "Add file to GitHub", "Resources", "HTML", "CSS", "JavaScript", and "Other files".
- HTML ペイン**: Top-left panel for HTML code.
- CSS ペイン**: Top-right panel for CSS code.
- JavaScript ペイン**: Bottom-left panel for JavaScript code, highlighted with a red box.
- 実行**: Bottom-right panel where the code is run.
- Right sidebar**: Contains settings for "LANGUAGE" (JavaScript), "FRAMEWORKS & EXTENSIONS" (No-Library (pure JS)), "LOAD TYPE" (No wrap - bottom of <head>, highlighted with a red box), and "FRAMEWORK <SCRIPT> ATTRIBUTE".

注意: JSFiddle 利用時にはいずれかの対応をしてください (ローカルファイル編集時や JS Bin では不要):

- LOAD TYPE の設定変更 (推奨)  
JavaScript + No-Library (pure JS) と書かれているところをクリックし LOAD TYPE の設定を **On Load** 以外 (**No wrap - bottom of <head>** など) に変更する
- mainFunction を自分で呼び出す  
onload に関数を登録せず mainFunction を定義後に自分で呼び出す。(最初の window.onload = を削除して最後に mainFunction(); を追加する)



# GPIO の使い方

## HTML/CSS を記載する

さて、今回はボタンと LED の状態インジケータを画面上に作ってみましょう。  
HTML に <button> と <div> 要素を 1 つづつ作ります。

JSFiddle にアクセスすると、初期状態でコード編集を始めることができます。  
この画面の **HTML ペイン**に下記コードを挿入します。

ledView のすぐ下に下記 <script> タグを記載し、 Web GPIO API を利用可能にするPolyfill を読み込ませましょう。

```
<button id="onoff">LED ON/OFF</button>
<div id="ledView"></div> <script src="https://r.chirimen.org/polyfill.js"></script>

<script src="https://r.chirimen.org/polyfill.js"></script>
```

※JSFiddle の HTML ペインには HTML タグの全てを書く必要はなく、<body> タグ内のみを書けばあとは補完してくれます。

ledView 要素には下記のようなスタイルを付けて黒い丸として表示させましょう。  
こちらは **CSS ペイン**に記載します。

```
#ledView {
  width: 60px;
  height: 60px;
  border-radius: 30px;
  background-color: black;
}
```



# GPIO の使い方

## ボタンに反応する画面を作る

GPIO を実際に使う前に、まずは「**ボタンを押したら LED の ON/OFF 状態を表示する画面を切り替える**」部分を作つてみます。

```
window.onload = function mainFunction() {
  var onoff = document.getElementById("onoff");
  var ledView = document.getElementById("ledView");
  var v = 0;
  onoff.onclick = function controlLed() {
    v = v === 0 ? 1 : 0;
    ledView.style.backgroundColor = v === 1 ? "red" : "black";
  };
};
```

このコードでは **onoff** 要素と **ledView** 要素を取得し、**onoff** ボタンのクリックイベント発生時に **ledView** の色を書き換えるイベントハンドラを登録しています。また、その処理は HTML 要素の読み込み後に実行するよう **window.onload** に設定する関数内に処理を書いています（HTML の読み込み前に処理すると **getElementById()** で要素が取得できません）。

実行タイミングを考えてコードを書くことは重要ですが、HTML の読み込み後に処理させることが多いので、実は JSFiddle では JavaScript は **onload** 後に実行する初期設定となっています。しかしこのままでは「読み込み完了時の処理を読み込み完了後に登録することになってしまい、折角書いたコードが実行されません。

ここまでできたら JSFiddle の JavaScript の ▶ **Run** をクリックして実行してみましょう。  
**LED ON/OFF** ボタンが表示されたら、ボタンをクリックしてみてください。ディスプレイの丸が、  
**赤 → 黒 → 赤 → 黒 → 赤 → 黒 →** とクリックする都度切り替えるようになったら成功です。



# GPIO の使い方

## ボタンに LED を反応させる

画面ができたので、いよいよ Web GPIO を使った LED 制御コードを書きます。  
まずは書き換えてみましょう。

```
window.onload = async function mainFunction() {
  var onoff = document.getElementById("onoff");
  var ledView = document.getElementById("ledView");
  var v = 0;
  var gpioAccess = await navigator.requestGPIOAccess();
  var port = gpioAccess.ports.get(26);
  await port.export("out");
  onoff.onclick = function controlLed() {
    v = v === 0 ? 1 : 0;
    port.write(v);
    ledView.style.backgroundColor = v ? "red" : "black";
  };
};
```

```
HTML
<button id="onoff">LED ON/OFF</button>
<div id="ledview"></div>
<script src="https://mz4u.net/libs/gc2/polyfill.js"></script>
```

```
JAVASCRIPT
(async ()=>{
  var onoff = document.getElementById("onoff");
  var ledview = document.getElementById("ledview");
  var v = 0;
  var gpioAccess = await navigator.requestGPIOAccess();
  var port = gpioAccess.ports.get(26);
  await port.export("out");
  onoff.onclick = ()=>{
    v ^= 1;
    port.write(v);
    ledview.style.backgroundColor = (v)? "red" : "black";
  };
})()
```

```
#ledview{
  width:60px;
  height:60px;
  border-radius:30px;
  background-color:black;
}
```

LED ON/OFF

注意: JSFiddle 利用時には LOAD TYPE を変更するか、mainFunction を自分で呼び出すようにするのを忘れずに。

これで、画面のボタンクリックに反応して LED の ON/OFF ができたら成功です。



# GPIO の使い方

## 画面のボタンをモーメンタリ動作に変えておく

下記のように、現在は onclick イベントで切り替えていきます。クリックイベントは、「マウスのボタンを押して離す」ことで発生します。

```
:  
onoff.onclick = function controlLed() {  
  v = v === 0 ? 1 : 0;  
  port.write(v);  
  ledView.style.backgroundColor = v ? "red" : "black";  
};  
:  
:
```

これを、マウスボタンを押した時と離した時にそれぞれオンオフさせるように変えましょう。  
押している間だけONになる「モーメンタリ」動作です。

- マウスのボタンを押す → LED を ON
- マウスのボタンを離す → LED を OFF

```
:  
onoff.onmousedown = function onLed() {  
  port.write(1);  
  ledView.style.backgroundColor = "red";  
};  
onoff.onmouseup = function offLed() {  
  port.write(0);  
  ledView.style.backgroundColor = "black";  
};  
:  
:
```

スイッチの動作：オルタネートとモーメンタリ

**オルタネート**：状態をトグル（切り替え）します。一度ボタンを押すと ON になりボタンから手を離しても OFF に変わりません。次にボタンを押すと OFF になります。ボタンから手を離しても ON に変わることはできません。

**モーメンタリ**：押している間だけ ON になります。スイッチから手を離すと OFF に戻ります（タクトスイッチはこちら）。

これでマウスもタクトスイッチも同じ挙動で揃いました。



# GPIO の使い方

## マウスクリックのかわりにタクトスイッチを使ってみる

タクトスイッチから操作する時も同じ処理を呼ぶことになるので、ここで LED の ON/OFF と ledView のスタイル切り替えをまとめて関数化しておきましょう。すると次のようなコードになります：

```
var port;

function ledOnOff(v) {
  var ledView = document.getElementById("ledView");
  if (v === 0) {
    port.write(0);
    ledView.style.backgroundColor = "black";
  } else {
    port.write(1);
    ledView.style.backgroundColor = "red";
  }
}

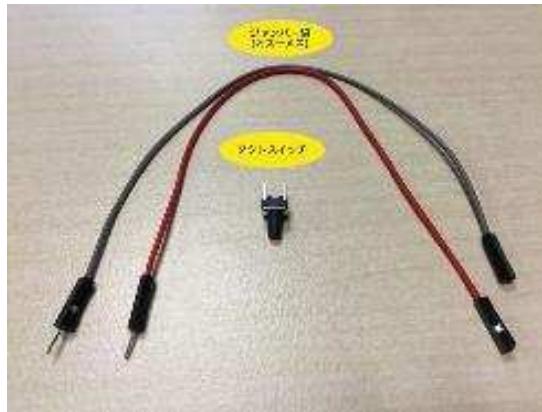
window.onload = async function mainFunction() {
  var onoff = document.getElementById("onoff");
  var gpioAccess = await navigator.requestGPIOAccess();
  port = gpioAccess.ports.get(26);
  await port.export("out");
  onoff.onmousedown = function onLed() {
    ledOnOff(1); ←
  };
  onoff.onmouseup = function offLed() {
    ledOnOff(0); ←
  };
};
```

ledOnOffを関数化

Main処理で呼び出し

# GPIO の使い方

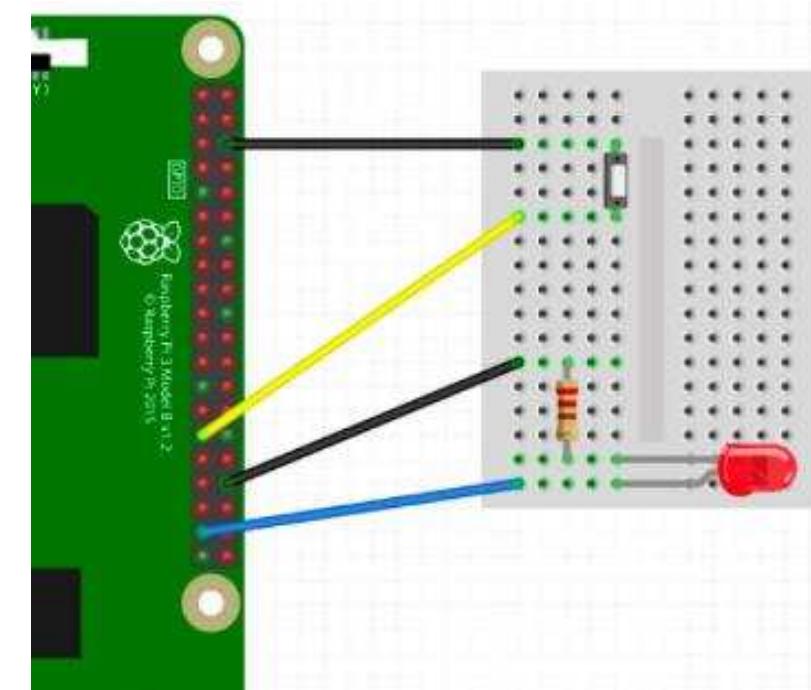
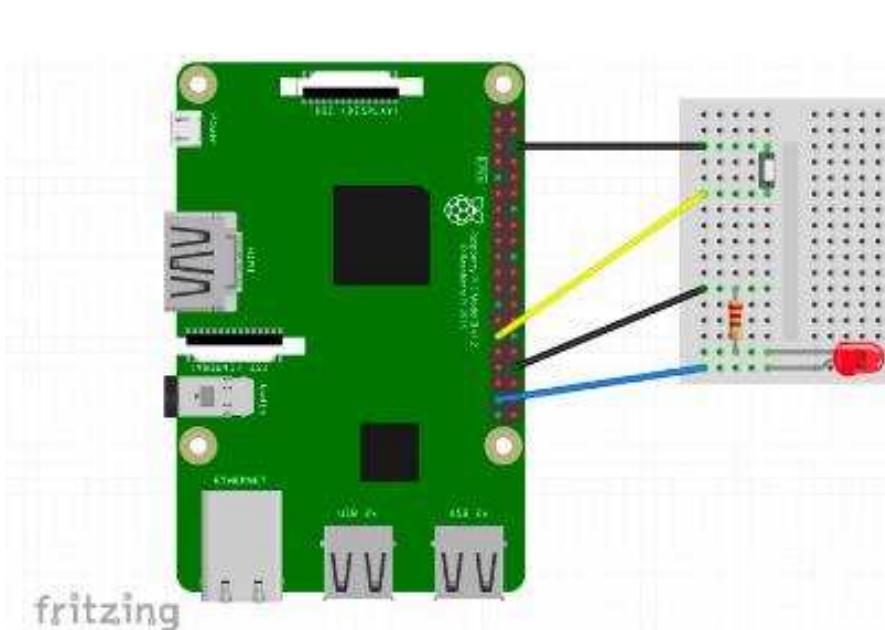
## 部品と配線について



今回追加するのは下記部品です。

- ・前述のタクトスイッチ × 1
- ・ジャンパーウイヤー（オス一メス）× 2

下図のように、さきほどの LED の配線にタクトスイッチを追加しましょう。





# GPIO の使い方

今回のスイッチは「プルアップ」回路で接続

上記回路ではスイッチが下記のように接続されています。

- Port 5 にスイッチを接続
- GND にスイッチの反対側を接続

これでどのようになるかというと、下記のようになります。

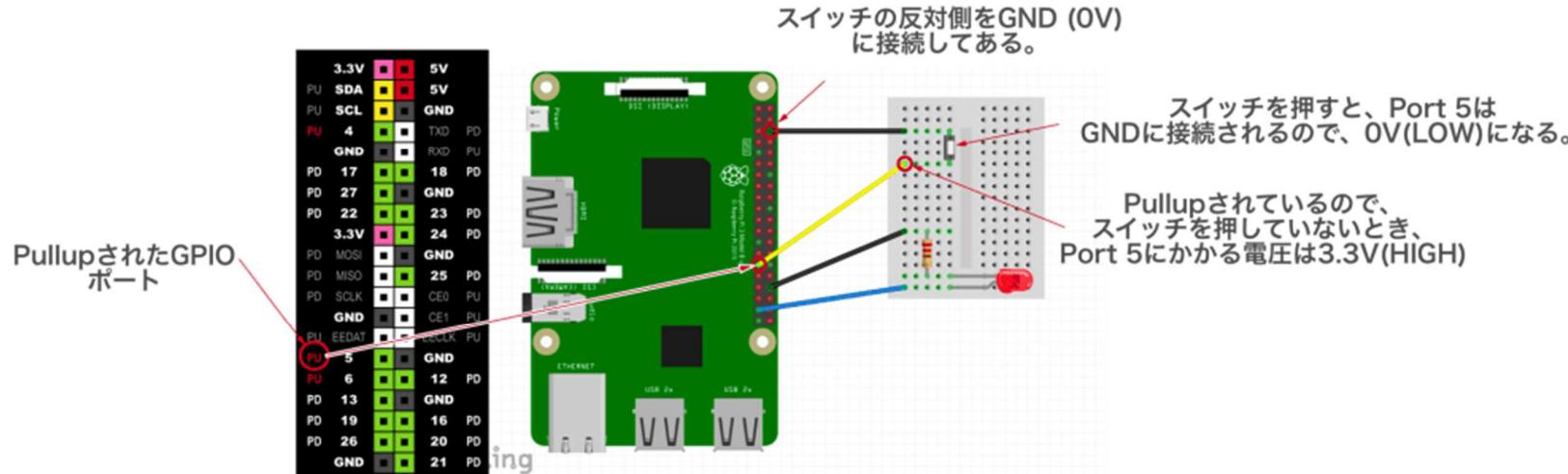
- スイッチを押す前は、Port 5 は HIGH (3.3V)
- スイッチを押している間、Port 5 は LOW (0V)

実は、Raspi3 の GPIO ポートのいくつかは、初期状態で「プルアップ」されています。

プルアップとは、回路を初期状態で「HIGH にしておく」ことですが、CHIRIMEN Raspi3 で利用可能な GPIO ポートのうち、図のポート番号がプルアップ状態となっています。

今回の回路では、このうち、**Port 5** を利用しています。さきほどの動作となるメカニズムは下図の通りです。

3.3V	PU	5V
SDA	5V	5V
SCL	GND	GND
4	PU	TXD
GND	PD	RXD
17	PD	18
27	PD	GND
22	PD	23
3.3V	3.3V	24
MOSI	PD	GND
MISO	PD	25
SCLK	CE0	CE1
EEDAT	PU	PU
5	PU	GND
6	PU	12
13	PD	GND
19	PD	16
26	PD	20
GND	PD	21





# GPIO の使い方

## スイッチに反応するようにする (port.onchange())

Web GPIO API には「入力モード」の GPIO ポートの状態に応じて処理する方法がもうひとつ用意されています。それが port.onchange() です。

さきほどのサンプルを port.onchange() を使ったコードに書き換えてみます。

```
var ledPort;
var switchPort; // LED とスイッチの付いているポート

function ledOnOff(v) {
  var ledView = document.getElementById("ledView");
  if (v === 0) {
    ledPort.write(0);
    ledView.style.backgroundColor = "black";
  } else {
    ledPort.write(1);
    ledView.style.backgroundColor = "red";
  }
}

window.onload = async function initialize() {
  var onoff = document.getElementById("onoff");
  var gpioAccess = await navigator.requestGPIOAccess();
  ledPort = gpioAccess.ports.get(26); // LED のポート番号
  await ledPort.export("out");
  switchPort = gpioAccess.ports.get(5); // タクスイッチのポート番号
  await switchPort.export("in");
  // Port 5 の状態が変わったタイミングで処理する
  switchPort.onchange = function toggleLed(val) {
    // スイッチは Pull-up なので OFF で 1、LED は OFF で 0 と反転させる
    ledOnOff(val === 0 ? 1 : 0);
  };
}
```

```
onoff.onmousedown = function onLed() {
  ledOnOff(1);
};
onoff.onmouseup = function offLed() {
  ledOnOff(0);
};
```

port.onchange は 入力モードの GPIO ポートの「**状態変化時に呼び出される関数を設定する**」機能です。

port.read() を使ったコードと異なりポーリング処理が不要となり、ポーリングによる LED 制御処理を行なっていないので、ブラウザ画面のボタンも正しく反応できるようになります。

# GPIO の使い方

LED のかわりに ギアモータ（ちびギアモータ）を動かしてみる

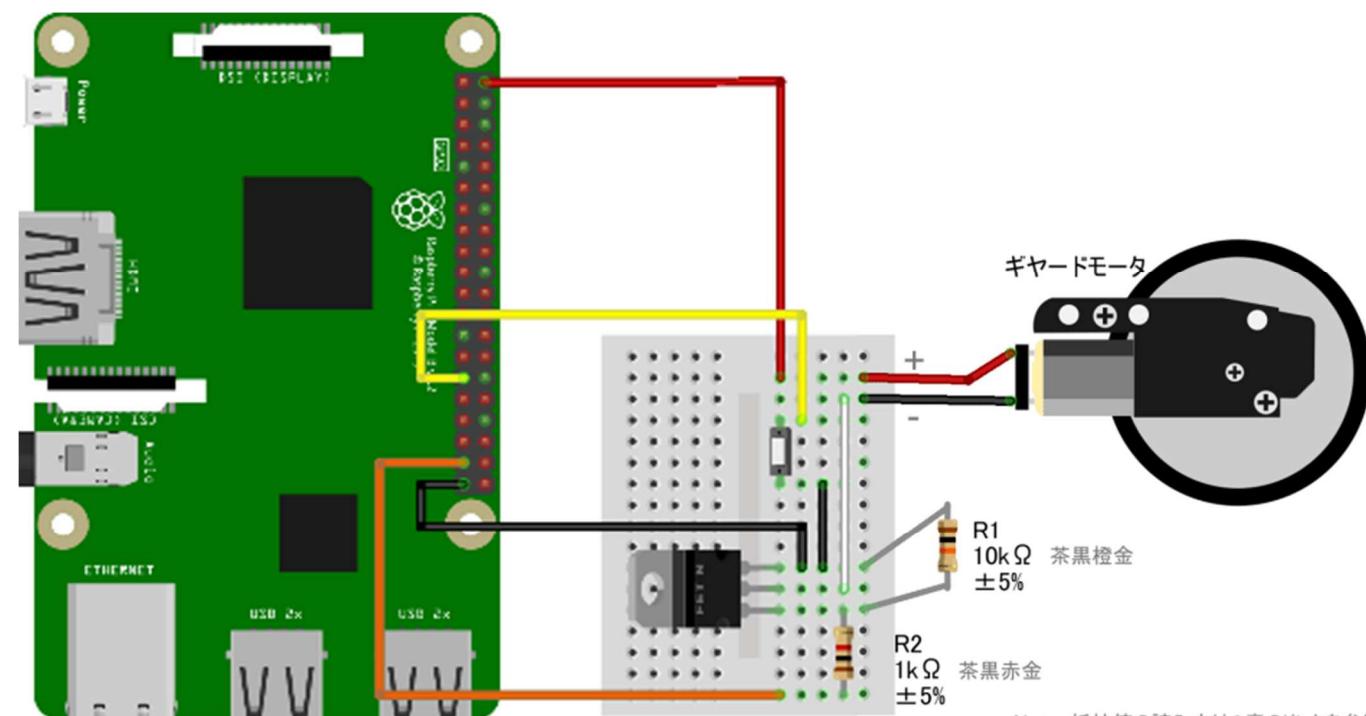
## 部品と配線について

Web GPIO API の機能が一通り確認できましたので、次は違う部品、**MOSFET** を使ってちびギアモータの単純な ON/OFF を制御してみましょう。

ちびギアモータ本体に加え、以下のものを用意し、先ほどの「タクトスイッチを押したら LED をつけたり消したり」する回路から、LED と LED 用の抵抗を一旦外して、MOSFET と抵抗、ちびギアモータを次のように配置します。



- Nch MOSFET (2SK4017)
  - リード抵抗 ( $1\text{K}\Omega$ ) × 1
  - リード抵抗 ( $10\text{K}\Omega$ ) × 1
  - ちびギアモータ × 1



Note: 抵抗値の読み方は0章のリンクを参照

黄色のジャンパー・ピンと黒のジャンパー・ピンの間をスイッチで「オン：オフ」できるように配線するのと同じです。



# GPIO の使い方

## コードは... 書き換えなくて良い

この回路は先ほどまでのコード 「スイッチに反応するようにする (port.onchange())」 と同じコードで動きます。 LED が点灯する替わりにちびギアモータが動くようになりました。

電圧を加える対象のデバイスが変わっただけで、プログラムで制御する、スイッチのオンオフに連動して電圧を変える処理は同じだからです。

## しかし... (オチ w)

スイッチを押してギアモータが回るだけなら、

5V → タクトスイッチ → ちびギアモータ → GND

と繋げば プログラムを書かなくても出来る!!!!  
..... スイッチではなく何かセンサーの値に連動するようにしましょう。



# GPIO の使い方

## まとめ

このチュートリアルでは、実際にコードを書きながら Web GPIO API の基本的な利用方法を学びました。

- Web GPIO API を使った GPIO 出力ポートの設定と出力処理までの流れ (`navigator.requestGPIOAccess()～port.write()`)
- Web GPIO API を使った GPIO 入力ポートの設定と変化検知受信の流れ (`navigator.requestGPIOAccess()～port.onchange()`)

このチュートリアルで書いたコードは以下のページで参照できます。

- GitHub リポジトリで参照 (<https://github.com/chirimen-oh/tutorials/tree/master/raspi3/examples/section1>)
- ブラウザで開くページ (各ステップ)
  - 画面のボタンで画面の要素の色を変える ([https://tutorial.chirimen.org/raspi3/examples/section1/s1\\_1](https://tutorial.chirimen.org/raspi3/examples/section1/s1_1))
  - 他面のボタンで LED が光り画面の要素の色も変わる ([https://tutorial.chirimen.org/raspi3/examples/section1/s1\\_2](https://tutorial.chirimen.org/raspi3/examples/section1/s1_2))
  - マウスで画面のボタンを押している間だけ LED が光る ([https://tutorial.chirimen.org/raspi3/examples/section1/s1\\_3](https://tutorial.chirimen.org/raspi3/examples/section1/s1_3))
  - タクトスイッチを押している間だけ LED が光る ([https://tutorial.chirimen.org/raspi3/examples/section1/s1\\_4](https://tutorial.chirimen.org/raspi3/examples/section1/s1_4))
  - 画面のボタンまたはタクトスイッチを押している間だけ LED が光る ([https://tutorial.chirimen.org/raspi3/examples/section1/s1\\_5](https://tutorial.chirimen.org/raspi3/examples/section1/s1_5))

最新情報は オンライン情報 (<https://tutorial.chirimen.org/>) をご覧ください。



# 付録

## その他の開発方法のサンプル

Grove システムの活用、Node.js や Node-REDを利用しての開発サンプルです。  
※以下の内容は講座は行いません。独習用の資料として参考にしてください。

# 使用機材 | 概要

## 主催者側で用意

GrovePi+ ボード & センサー



・GrovePi+ ボード



・赤色LED



・超音波センサー



・4ピンケーブル

Raspberry Pi & アダプタ



・Raspberry Pi



・ACアダプタ



・microSDカード

## 参加者が各自持参

無線LAN内蔵ノートパソコン



Wi-Fi



Windows

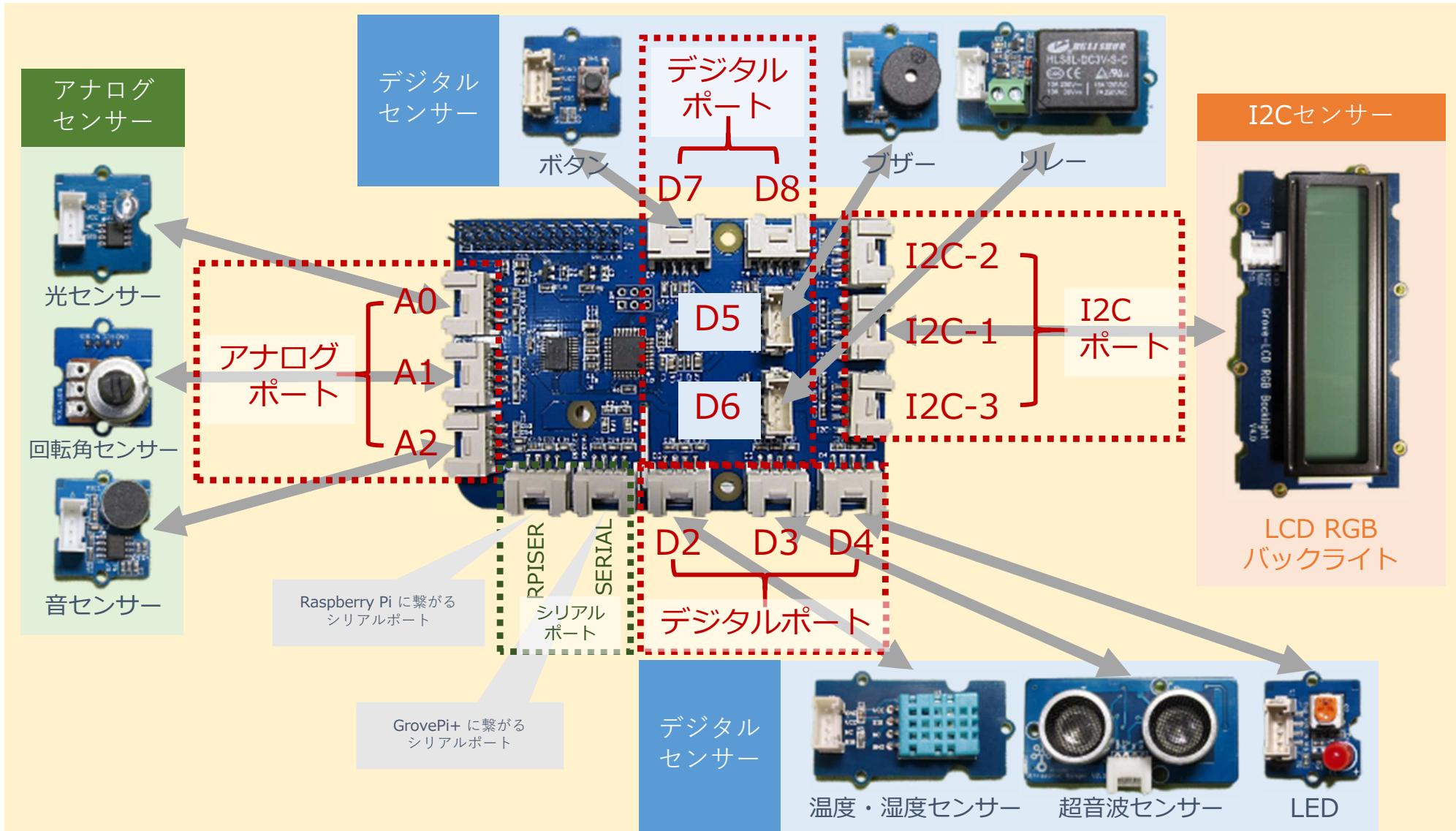
または

MacBook

・環境構築、アプリ作成に必要

# 使用機材 | GrovePi+ ボード

GrovePi+は、Raspberry Pi の上に取り付けてセンサー類を使えるようにする拡張ボード。デジタルポート、アナログポート、I2Cポートを装備。



# 使用機材 | センサー類

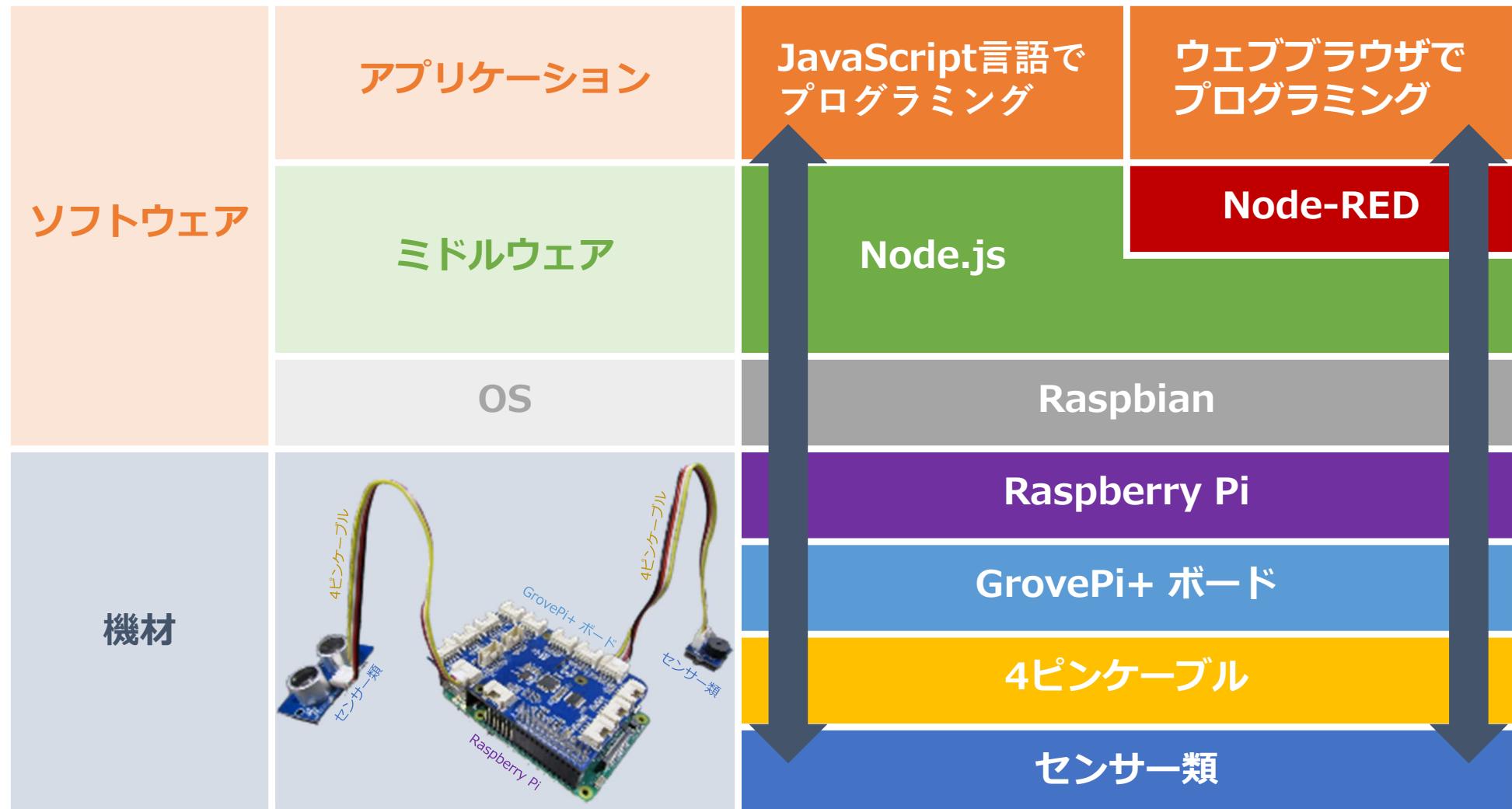
センサー類は用途によって種類も様々。はんだ付け不要で簡単にボードへ接続可。

入力系	アナログセンサー		光センサー	光を感じるセンサー
			音センサー	音を感じるセンサー
			回転角センサー	つまみを回すと抵抗値が変化するセンサー
実習で使用	デジタルセンサー		ボタン	押す動作でON/OFFを切り替える
			温度・湿度センサー	温度、湿度、体感温度を測定するセンサー
出力系	デジタルセンサー		超音波センサー	反射波を受信して対象物までの距離を測定するセンサー
			LED(赤、緑、青)	電圧をかけると発光する半導体素子
			ブザー	電圧を加えると音が鳴る
I2Cセンサー			リレー	スイッチ等から電気信号を受けてモータ等の出力部に伝える中継係
			LCD RGBバックライト	RGBバックライト・黒文字タイプを搭載した液晶ディスプレイ

# システム構成

使用する機材・ソフトウェアのシステム構成は以下の通り。

JavaScript言語で書かれたアプリケーションプログラムを実行して、Raspberry Pi、GrovePi+ ボードに接続されたセンサー類を制御する。



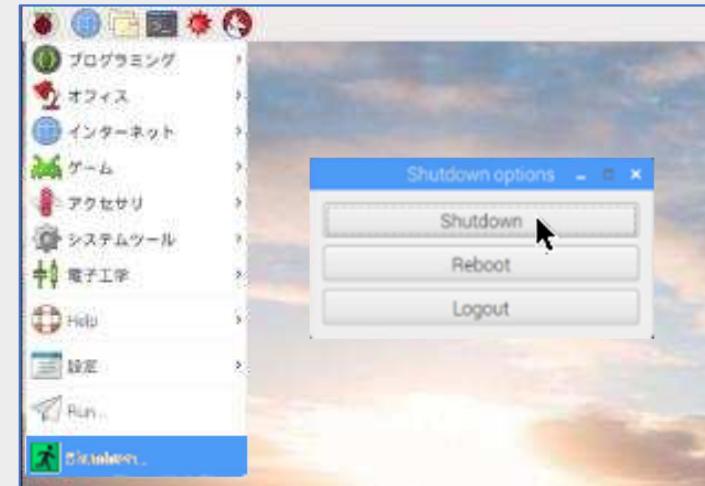
# GrovePi+ ボードの取り付け

Raspberry Pi の上に GrovePi+ ボードを取り付ける。

1. GrovePi+ ボードの取り付け、取り外しは、  
Raspberry Pi の電源を切ってから行うこと

Raspberry Pi の電源の切り方

- (1)メニューからShutdown...を選択しシャットダウン
- (2)Raspberry Pi の緑ランプ消灯を確認し電源を落とす



GrovePi+ ボード



Raspberry Pi

2. Raspberry Pi の上、右端のピンに合わせて、  
GrovePi+ ボードを取り付け

3. Raspberry Pi の電源を入れる

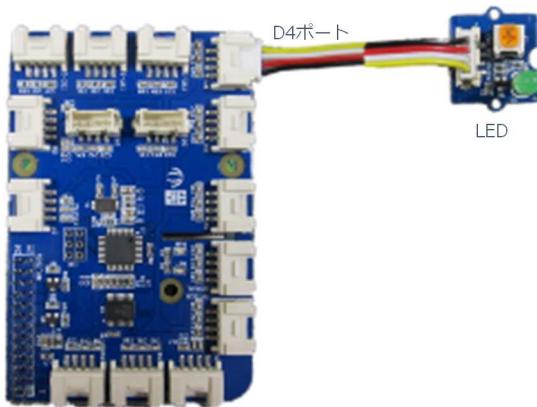


さあ、それでは実習を始めましょう！

# 実習① | LEDを光らせてみる

Raspberry Pi をJavaScriptで制御してLEDを光らせてみよう。

## 1. GrovePi+ボードのD4にLEDを挿す



## 2. ターミナル(コマンドプロンプト)から以下コマンドを実行

node led.js

【コマンドの意味】

Node.jsで「led.js」というJavaScript言語のプログラムを実行。「led.js」の中身は右 ➡



1秒間隔でLEDが点滅したら 成功！

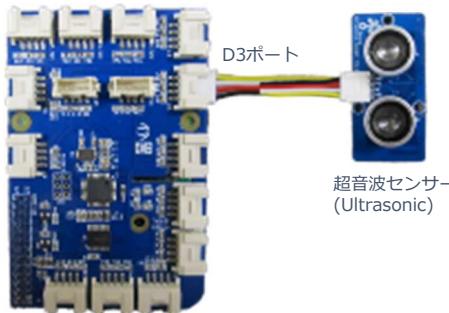
```
// LED点滅
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var LedDigital = GrovePi.sensors.base.Digital;
var blink = false;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var ledDigital = new LedDigital(4); // D4ポートを指定
      setInterval(function(){
        ledDigital.write(blink); // LEDを点滅
        if(blink){
          blink = false;
        }else{
          blink = true;
        }
      },1000); // 1000ミリ秒(=1秒)のインターバル
    }
  });
  board.init();
```

# 実習② | 超音波センサーを使ってみる

超音波センサーを使って距離を測定してみよう。

Node-RED を使うと、GUIで直感的・簡単にプログラミングできる。

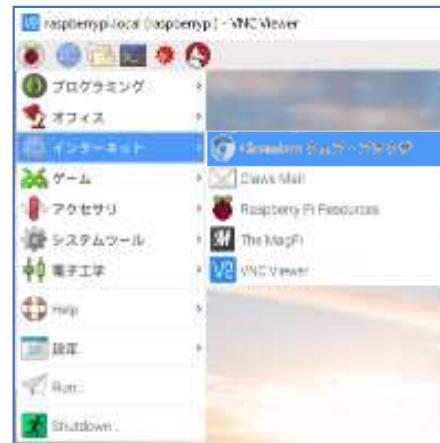
1. GrovePi+ボードのD3に超音波センサーを挿す



2. Node-REDを起動



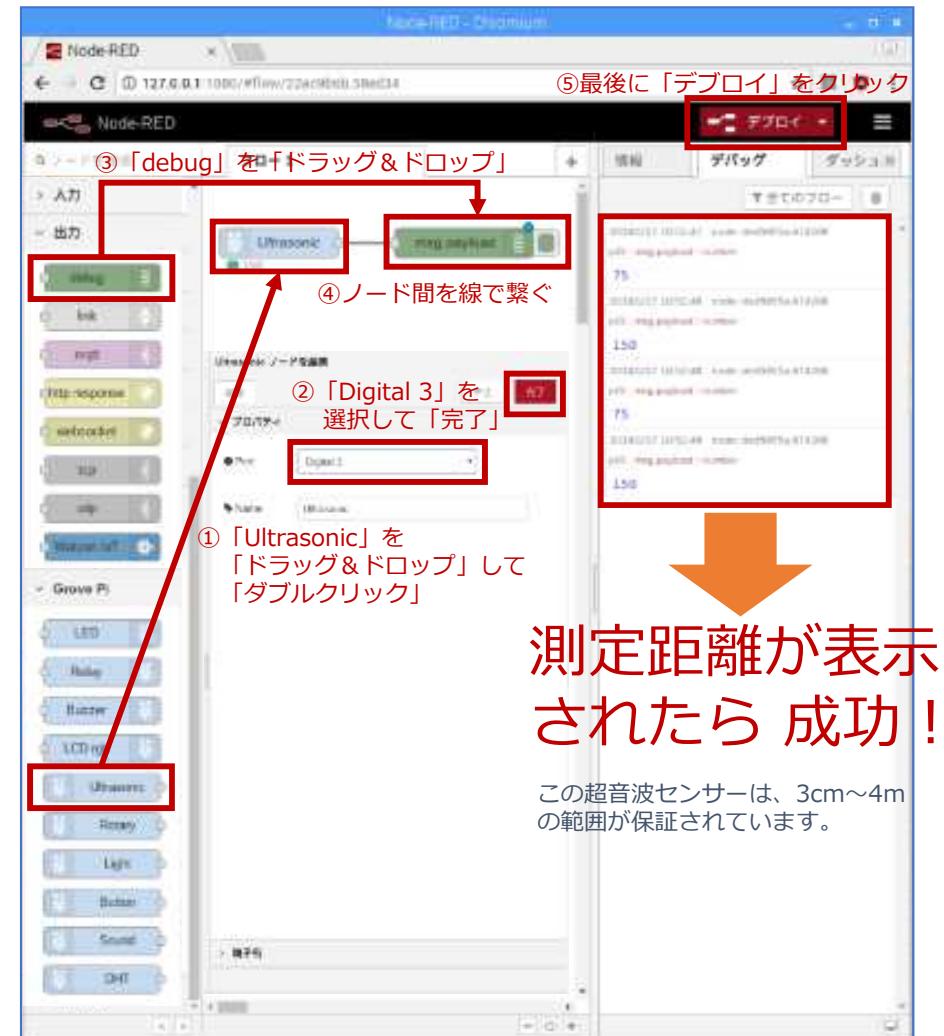
3. ウェブブラウザを起動



URLは、<http://127.0.0.1:1880/>

機能毎に用意されたノードをマウスで繋ぐ操作で、簡単にアプリケーションを作成。

Server now running at <http://127.0.0.1:1880/>  
と出力されたら Node-REDは起動完了。



# 実習③ | 測定結果を表示してみる

超音波センサーを使って測定した距離を Web画面に表示してみよう。

Node-RED のダッシュボードを使うと、グラフィカルにリアルタイム描画できる。

The image shows two screenshots of the Node-RED interface. On the left, the flow editor displays a flow with an 'Ultrasonic' node connected to a 'Gauge' node. Annotations explain the steps: ① 'gauge'を「ドラッグ&ドロップ」して「ダブルクリック」 (Drag & Drop 'gauge' and double-click), ② Groupを作成して「完了」 (Create a group and finish), ③ ノード間を線で繋ぐ (Connect nodes with lines), and ④ 「デプロイ」をクリック (Click 'Deploy'). A red arrow points from the flow editor to the right screenshot. The right screenshot shows the deployed dashboard with a gauge element labeled 'Gauge'. Annotations explain: ⑤ 「ダッシュボード」タブの右上をクリックすると… (Click the top-right of the 'Dashboard' tab...), followed by two URL examples: http://127.0.0.1:1880/ui/ (with 'ui' appended) and http://(ホスト名):1880/ui/ (host name). A large orange arrow at the bottom points to the text 'ゲージが表示されたら 成功！' (Success when the gauge is displayed!).

① 「gauge」を「ドラッグ&ドロップ」して「ダブルクリック」

② Groupを作成して「完了」

③ ノード間を線で繋ぐ

④ 「デプロイ」をクリック

⑤ 「ダッシュボード」タブの右上をクリックすると…  
http://127.0.0.1:1880/ui/  
(URLの後ろに「ui」を付ける)  
でも表示できる

http://(ホスト名):1880/ui/  
のようにホスト名でアクセスすれば、PC・スマートフォンから  
でも参照できる

配置やテーマが変更可

0      149      200  
units

ゲージが表示されたら 成功！

# 付録 | センサーのサンプルコード①

## 1. 光センサー(light.js)

```
// 光センサー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var LightAnalogSensor = GrovePi.sensors.LightAnalog;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var lightSensor = new LightAnalogSensor(0); // A0ポートを指定
      lightSensor.stream(1000, // 1000ミリ秒(=1秒)のインターバル
        function(res){
          console.log(res) // 値を出力
        });
    }
  });
board.init();
```

## 2. 音センサー(sound.js)

```
// 音センサー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var SoundAnalogSensor = GrovePi.sensors.LoudnessAnalog;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var soundSensor = new SoundAnalogSensor(2); // A2ポートを指定
      soundSensor.stream(1000, // 1000ミリ秒(=1秒)のインターバル
        function(res){
          console.log(res) // 値を出力
        });
    }
  });
board.init();
```

# 付録 | センサーのサンプルコード②

## 3. 回転角センサー(rotary.js)

```
// 回転角センサー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var RotaryAnalogSensor = GrovePi.sensors.RotaryAnalog;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var rotarySensor = new RotaryAnalogSensor(1); // A1ポートを指定
      rotarySensor.start();
      rotarySensor.on('data', function(res){
        console.log(res) // 値を出力
      });
    }
  }
});
board.init();
```

## 4. ボタン(button.js)

```
// ボタン
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var DigitalButtonSensor = GrovePi.sensors.DigitalButton;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var buttonSensor = new DigitalButtonSensor(7); // D7ポートを指定
      buttonSensor.on('down', function(res){ // ボタンが押された時
        console.log(res) // 値を出力
      });
      buttonSensor.watch();
    }
  }
});
board.init();
```

# 付録 | センサーのサンプルコード③

## 5. 温度・湿度センサー(dht.js)

```
// 温度・湿度センサー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var DHTDigitalSensor = GrovePi.sensors.DHTDigital;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var dhtSensor = new DHTDigitalSensor(2); // D2ポートを指定
      dhtSensor.on('change', function(res){ // 値が変化した時
        console.log(res) // 値を出力
      });
      dhtSensor.watch(1000); // 1000ミリ秒(=1秒)のインターバル
    }
  });
board.init();
```

## 6. 超音波センサー(ultrasonic.js)

```
// 超音波センサー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var UltrasonicDigitalSensor = GrovePi.sensors.UltrasonicDigital;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var ultrasonicSensor = new UltrasonicDigitalSensor(3); // D3ポートを指定
      ultrasonicSensor.stream(1000, // 1000ミリ秒(=1秒)のインターバル
        function(res){
          console.log(res) // 値を出力
        });
    }
  });
board.init();
```

# 付録 | センサーのサンプルコード④

## 7. ブザー(buzzer.js)

```
// ブザー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var BuzzerDigital = GrovePi.sensors.base.Digital;
var buzzer = false;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var buzzerDigital = new BuzzerDigital(5); // D5ポートを指定
      setInterval(function(){
        buzzerDigital.write(buzzer); // ブザーを鳴らす・止める
        if(buzzer){
          buzzer = false;
        }else{
          buzzer = true;
        }
      },1000); // 1000ミリ秒(=1秒)のインターバル
    }
  });
board.init();
```

## 8. リレー(relay.js)

```
// リレー
var GrovePi = require('node-grovepi').GrovePi;
var Board = GrovePi.board;
var RelayDigital = GrovePi.sensors.base.Digital;
var relay = false;
var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var relayDigital = new RelayDigital(6); // D6ポートを指定
      setInterval(function(){
        relayDigital.write(relay); // リレースイッチをオン・オフ
        if(relay){
          relay = false;
        }else{
          relay = true;
        }
      },1000); // 1000ミリ秒(=1秒)のインターバル
    }
  });
board.init();
```

# 付録 | センサーのサンプルコード⑤

## 9. LCD RGBバックライト(lcdrgb.js)

```
// LCD RGBバックライト
var GrovePi = require('node-grovepi').GrovePi;
var i2c = require('i2c-bus');
var sleep = require('sleep/');

var Commands = GrovePi.commands;
var Board = GrovePi.board;

var DISPLAY_RGB_ADDR = 0x62;
var DISPLAY_TEXT_ADDR = 0x3e;

function setRGB(i2c1, r, g, b) {
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,0,0)
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,1,0)
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,0x08,0xaa)
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,4,r)
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,3,g)
  i2c1.writeByteSync(DISPLAY_RGB_ADDR,2,b)
}

function textCommand(i2c1, cmd) {
  i2c1.writeByteSync(DISPLAY_TEXT_ADDR, 0x80, cmd);
}

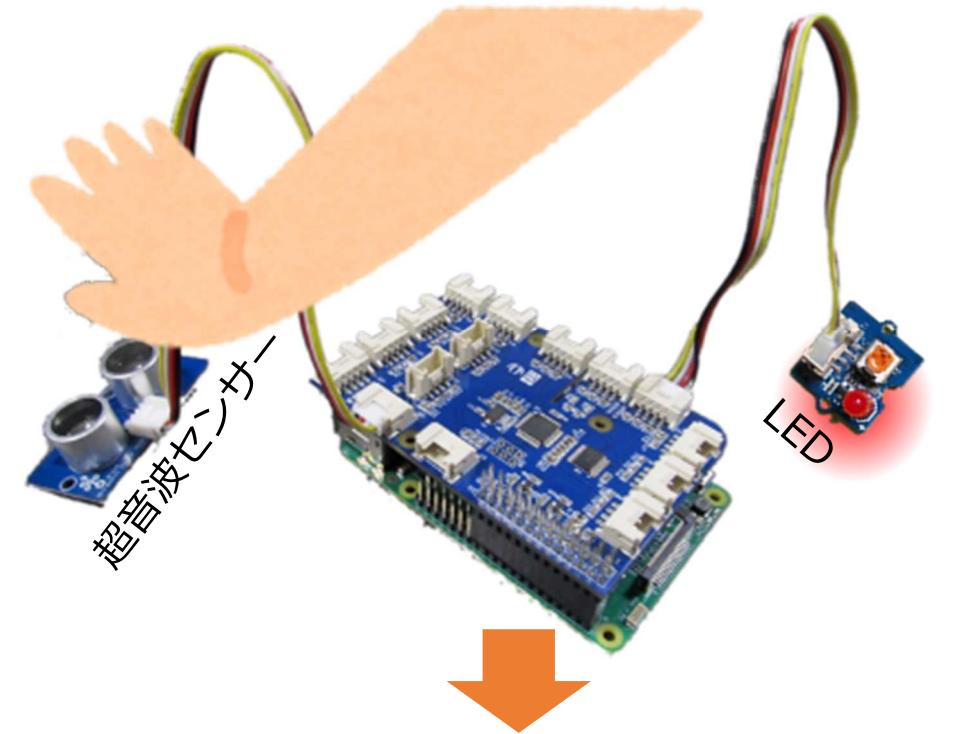
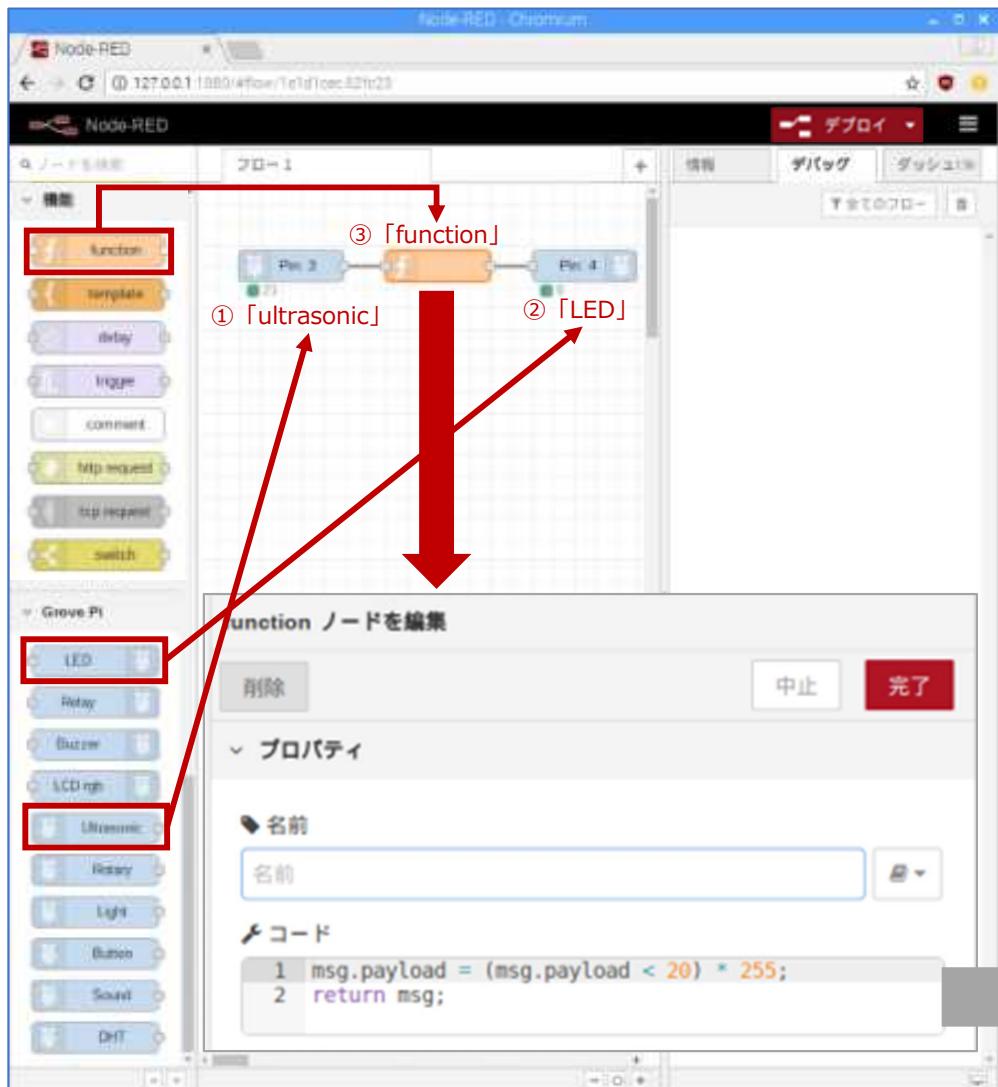
function setText(i2c1, text) {
  textCommand(i2c1, 0x01) // clear display
  sleep.usleep(50000);
  textCommand(i2c1, 0x08 | 0x04) // display on, no cursor
  textCommand(i2c1, 0x28) // 2 lines
  sleep.usleep(50000);
  var count = 0;
  var row = 0;
```

```
for(var i = 0, len = text.length; i < len; i++) {
  if(text[i] === '\n' || count === 16) {
    count = 0;
    row++;
    if(row === 2)
      break;
    textCommand(i2c1, 0xc0)
    if(text[i] === '\n')
      continue;
  }
  count++;
  i2c1.writeByteSync(DISPLAY_TEXT_ADDR, 0x40,
text[i].charCodeAt(0));
}
}

var board = new Board({
  debug: true,
  onError: function(err){
    console.log('ERROR');
    console.log(err)
  },
  onInit: function(res){
    if(res){
      var i2c1 = i2c.openSync(1); // I2Cポートを指定
      setRGB(i2c1, 255, 255, 255); // バックライトのRGB値を指定
      setText(i2c1, "Hello\nWorld!"); // 表示テキストを指定
      i2c1.closeSync();
    }
  }
});
board.init();
```

# 付録 | Node-REDのサンプルコード①

超音波センサーとLEDを組み合わせて、手をかざすとLEDが光るプログラムを作る。Functionノードを使うと、JavaScript のプログラムを追加・実行できる。



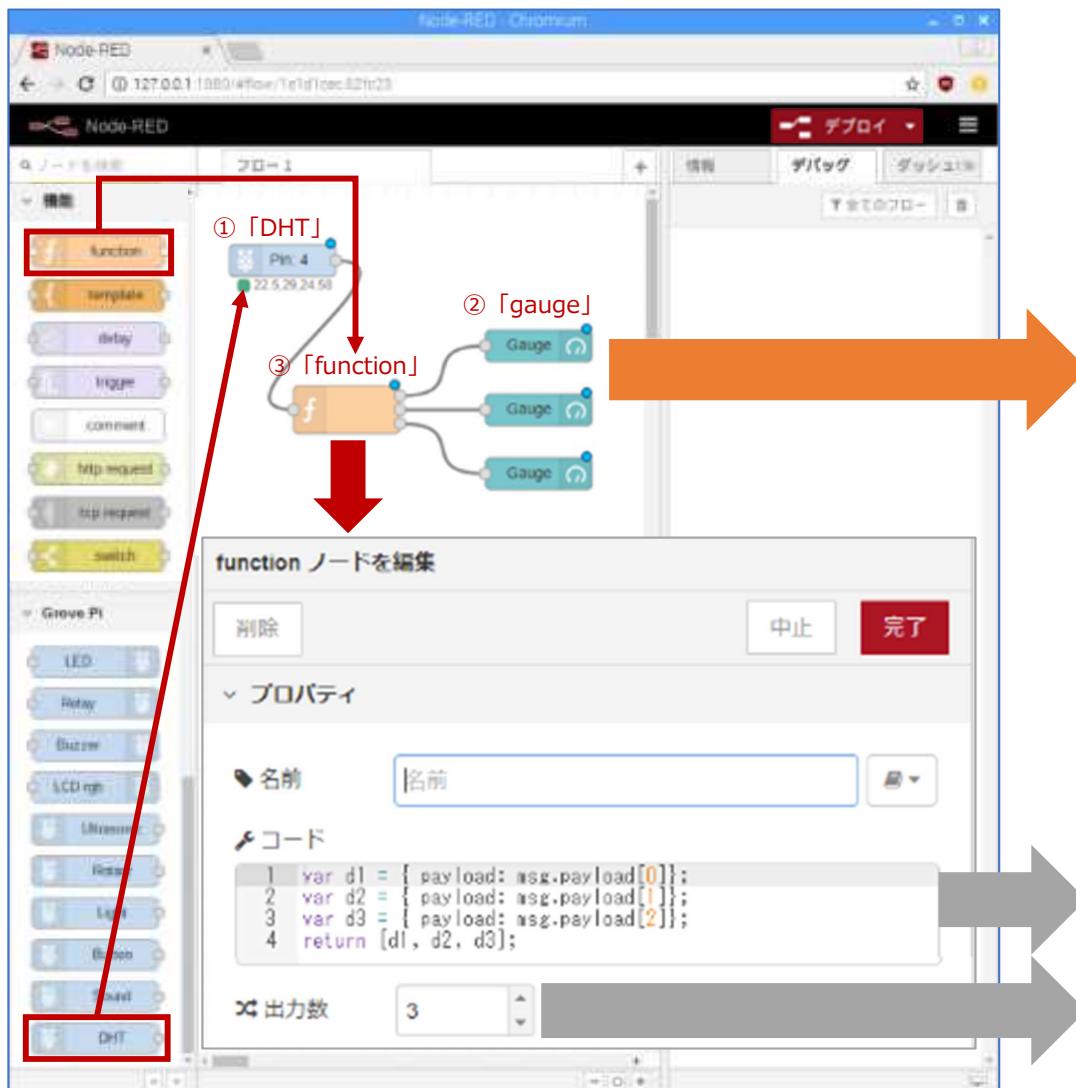
手をかざすとLEDが光り、  
手を離すとLEDが消えれば、成功！

## ④コードを記述

```
msg.payload = (msg.payload < 20) * 255;
return msg;
```

## 付録 | Node-REDのサンプルコード②

温度・湿度センサーを使って、温度・湿度・体感温度を表示する。  
Functionノードで、コードを記述、出力数を3つにすることで値が取り出せる。



3つが表示されれば、成功！

センサーからの出力は  
「msg.payload」  
に格納される。

温度・湿度センサーでは、  
温度は「msg.payload[0]」  
のように配列で、  
温度・湿度・体感温度の順  
に格納されている。

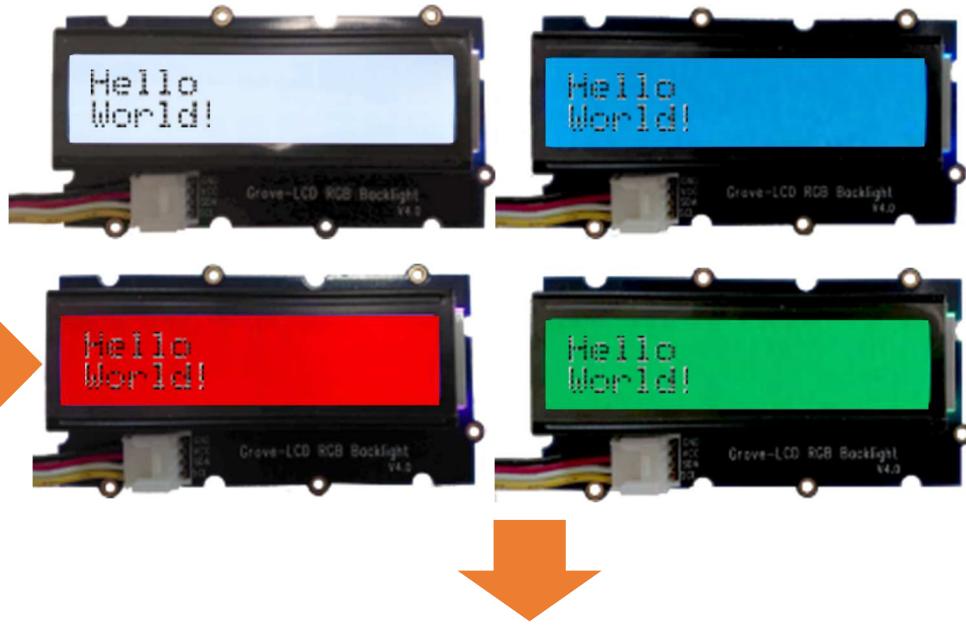
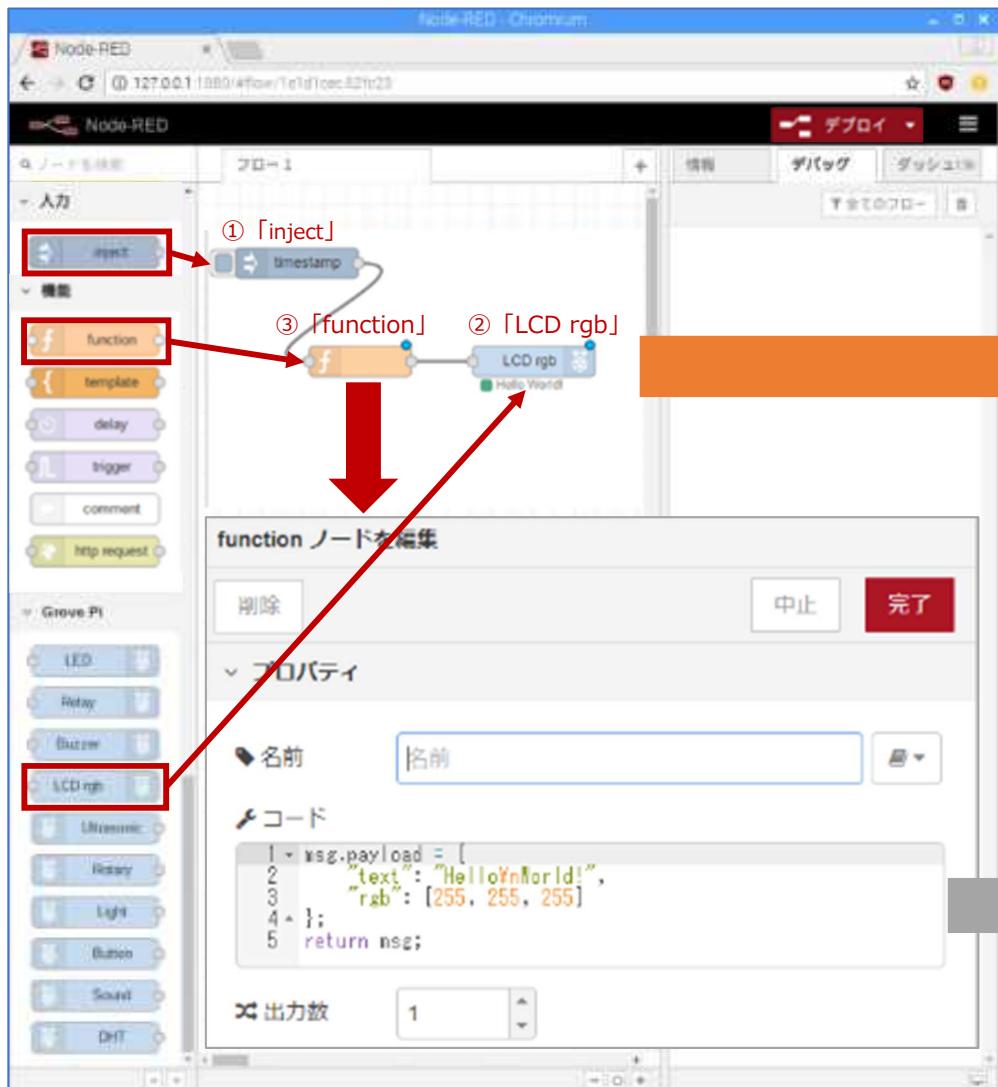
### ④コードを記述

```
var d1 = {payload: msg.payload[0]};  
var d2 = {payload: msg.payload[1]};  
var d3 = {payload: msg.payload[2]};  
return [d1, d2, d3];
```

### ⑤出力数を3に設定

## 付録 | Node-REDのサンプルコード③

LCD RGBバックライトを使用して、テキストを表示、背景の色を設定する。  
Functionノードのコードをいろいろと変えてみよう。



テキスト、背景色が表示されれば、成功！

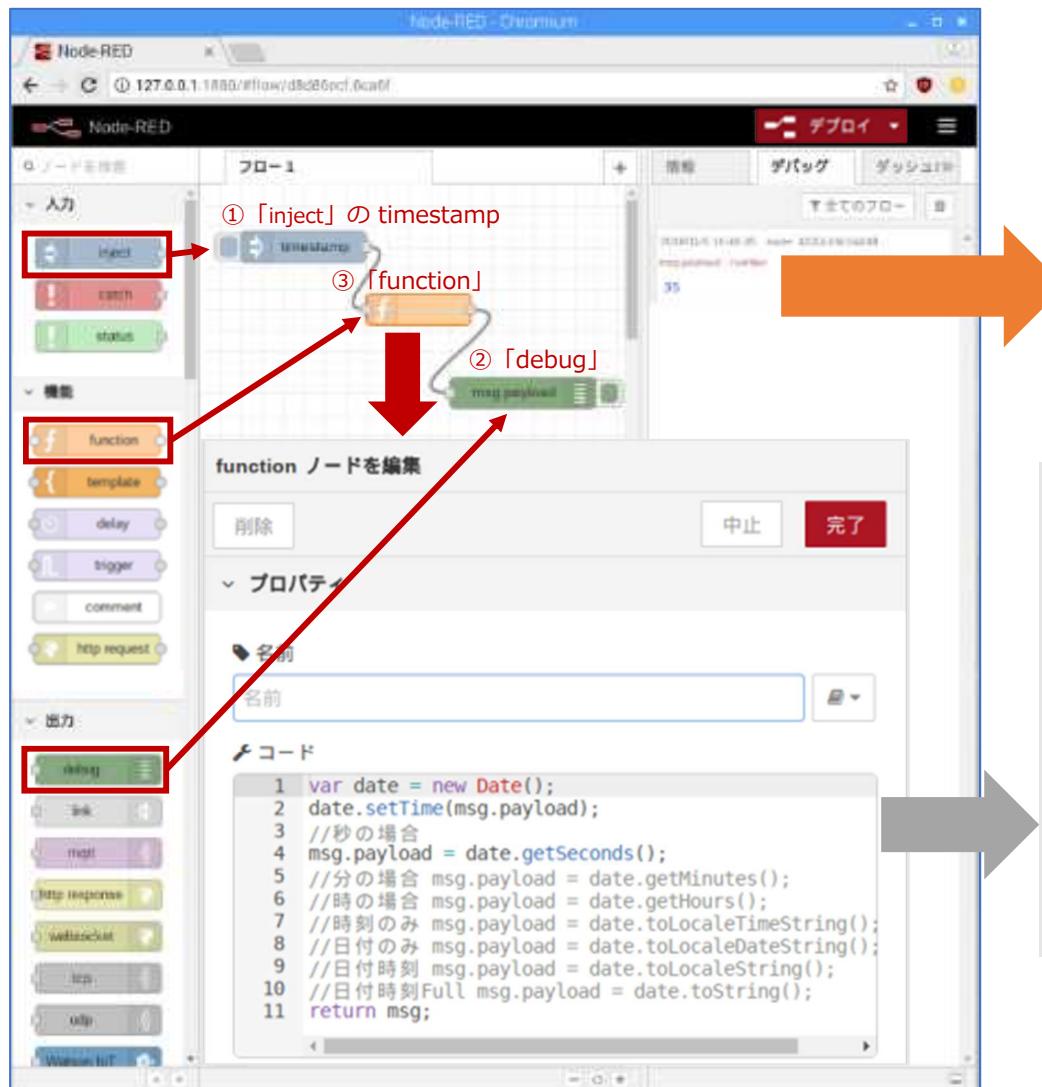
### ④コードを記述

```
msg.payload = {
  "text": "Hello\nWorld!", // 表示するテキスト
  "rgb": [255, 255, 255] // Red,Green,Blue値
};
return msg;
```

“text” “rgb” の値を変更するとテキスト 背景色が変わります。  
( “text” 内の ¥n は改行を表す制御コードです。)

# 付録 | Node-REDのサンプルコード④

Node-REDで現在時刻の情報を使いたい場合、Injectノードの timestamp と、Functionノードで「日付時刻の関数」を使うと 時刻が取得できる。



この例 (getSeconds関数) では  
秒のみが取得できれば、成功！

日付時刻の関数は その他にも  
いろいろあるので試してみよう。

## ④コードを記述

```
var date = new Date();
date.setTime(msg.payload);
//秒の場合
msg.payload = date.getSeconds();
//分の場合 msg.payload = date.getMinutes();
//時の場合 msg.payload = date.getHours();
//時刻のみ msg.payload = date.toLocaleTimeString();
//日付のみ msg.payload = date.toLocaleDateString();
//日付時刻 msg.payload = date.toLocaleString();
//日付時刻Full msg.payload = date.toString();
return msg;
```

関数を使わないので取得される数値は、  
「1970年1月1日からの経過時間（単位：ミリ秒）」です！

# トラブルシューティング① | ホスト名で接続できない

## ■ホスト名で接続できない

**対策** ホスト名の代わりに、IPアドレスで接続してみる。

Raspberry Pi のIPアドレスは、コマンドプロンプトに  
以下のpingコマンドを入力することで、調べることができます  
(コマンドプロンプトは「プログラムとファイルの検索」で検索すると出てきます)

入力するコマンド : ping 「Raspberry Pi のホスト名」  
(ホスト名を変更していない場合はデフォルト : raspberrypi )

1. ホスト名に対応した  
IPアドレス [■. ■. ■. ■] を調べる！  
注意) IPアドレスは固定ではありません



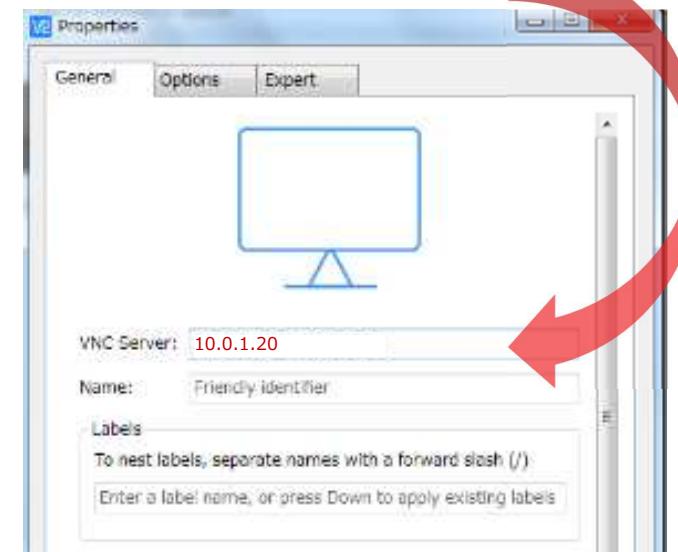
```
C:\> コマンド プロンプト
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\$Users\$admin>ping raspberrypi

raspberrypi [10.0.1.20] に ping を送信しています 32 バイトのデータ:
10.0.1.20 からの応答: バイト数 = 32 時間 <4ms TTL=64
10.0.1.20 からの応答: バイト数 = 32 時間 <5ms TTL=64
10.0.1.20 からの応答: バイト数 = 32 時間 <3ms TTL=64
10.0.1.20 からの応答: バイト数 = 32 時間 <2ms TTL=64

10.0.1.20 の ping 統計:
パケット数: 送信 = 4、受信 = 4、損失 = 0 (0% の損失),
ラウンド トリップの概算時間 (ミリ秒):
最小 = 2ms、最大 = 5ms、平均 = 3ms
```

2. 取得したIPアドレスを  
「VNC Server」の項目に入力



# トラブルシューティング② | ファイルサーバへのアクセス

## ■ Windows からファイルサーバへアクセスできない

**原因** Windows の SMB 1.0 が無効になっている可能性があります。

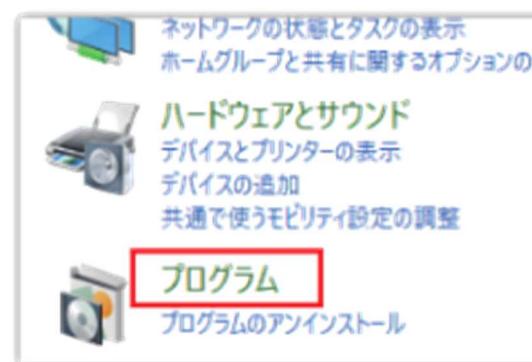
**対策** 以下の操作で SMB 1.0 を有効にできます。

1 コントロールパネルを開く

### コントロールパネルの開き方

- ① スタートメニューを開く
- ② [Windows システム ツール] → [コントロールパネル] を開く

2 [プログラム] を開く



3 [Windows の機能の有効化または無効化] を開く

「プログラムと機能」内にあります。



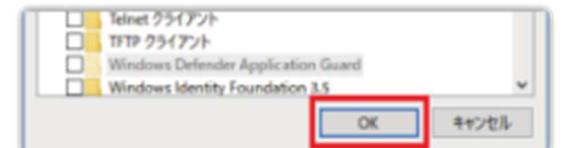
4 [SMB 1.0/CIFS クライアント] をチェックする

「SMB 1.0/CIFS ファイル共有のサポート」内にあります。



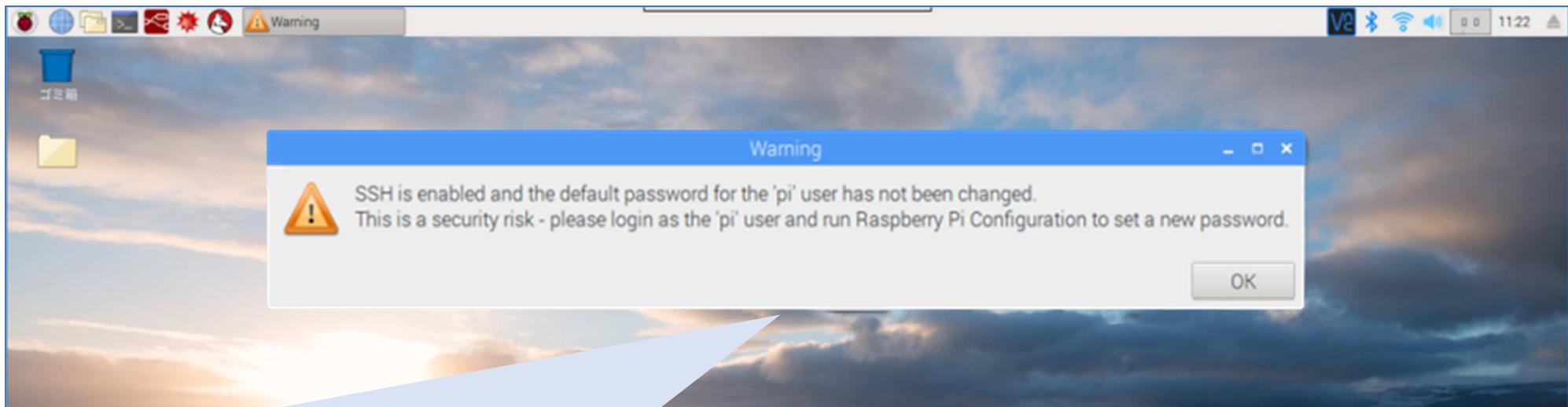
SMB 1.0/CIFS クライアント  
にチェックを入れる！

5 [OK] をクリックする



# トラブルシューティング③ | Raspberry Pi 起動直後の警告

■Raspberry Pi を起動直後に、次の警告メッセージが表示される。



【警告メッセージの日本語訳】

SSHが有効で、「pi」ユーザーのデフォルトパスワードは変更されていません。

これはセキュリティ上のリスクです。新しいパスワードを設定するには、「pi」ユーザーでログインし、「passwd」と入力してください。

原因

「**パスワードが初期値のままなので、変更してください**」という警告メッセージ。

対策

今回の講習では、初期パスワードのまま使用しますので、**変更せずに「OK」をクリックします。**

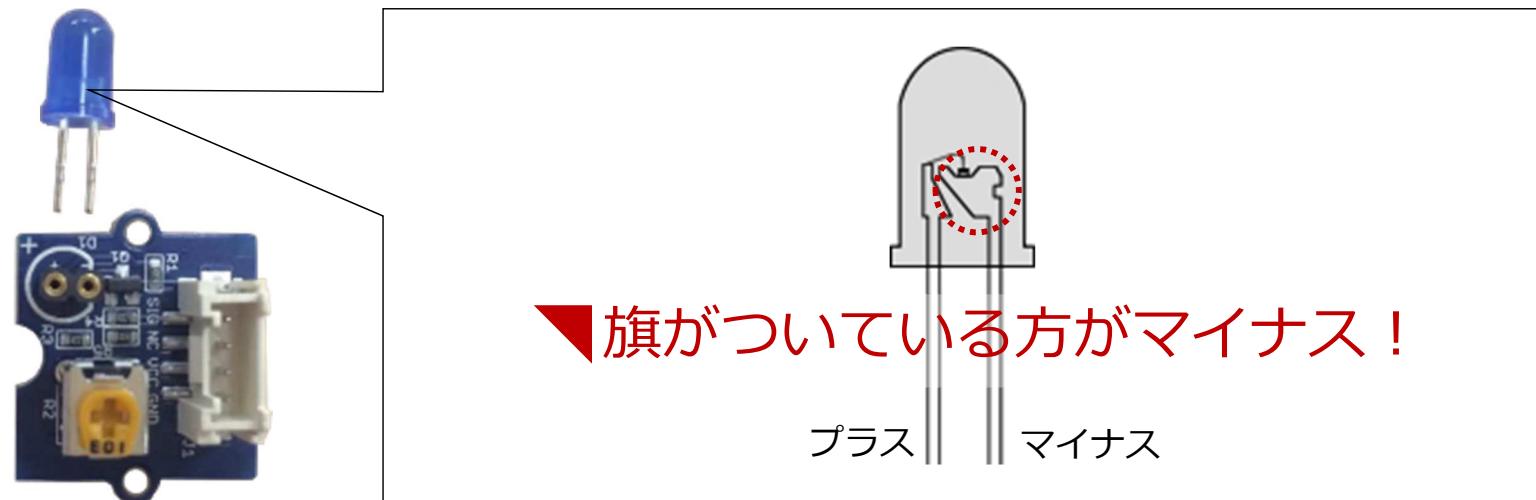
自分で使用する場合は、セキュリティ上問題となりますので、必ずパスワードを変更しましょう。

# トラブルシューティング④ | サンプルコードの実行

■サンプルコードを実行したが、センサーが反応しない。

**原因①** LEDが点灯しない原因として、プラスとマイナスが逆になっていますか？

**対策①** LEDはプラスとマイナスがあり、差し込む基盤側にプラスとマイナスが書かれています。正しい位置に差し込まれているか確認しましょう。



**原因②** センサーを接続したGrovePi+ボードのポートが間違っていますか？

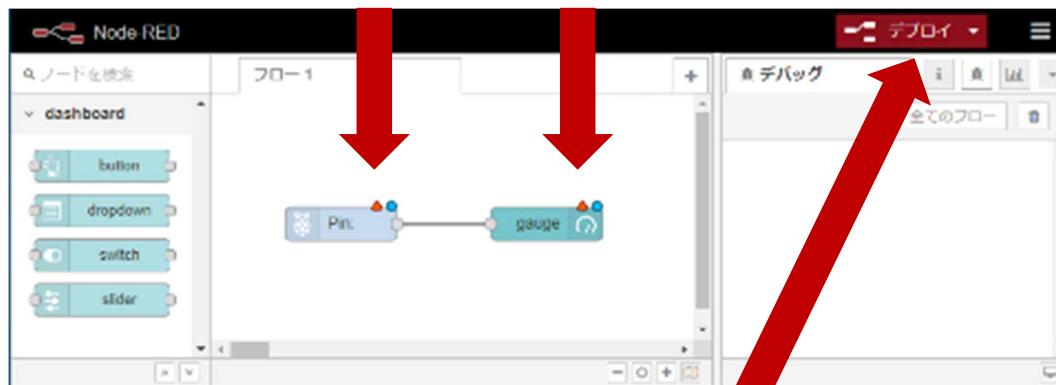
**対策②** サンプルコードには、接続するポートを固定で指定しています。  
サンプルコードに指定してあるポートを確認しましょう。

# トラブルシューティング⑤ | Node-RED のデプロイ

■ ドラッグ＆ドロップしたNode-REDのノードが動作しない。

**原因①** ノードの上に赤色の三角マーク▲が出ていませんか？

**対策①** 三角マーク▲はエラーがあることを示しています。  
ノードをダブルクリックして、プロパティを正しく設定します。



**原因②** 「デプロイ」のクリックを忘れていませんか？

**対策②** ノードを配置＆ノード間を線で繋ぐだけでは、動作しません。  
最後は必ず「デプロイ」をクリックします。

**原因③** 「デプロイ」でエラーが出ていませんか？

**対策③** 「デプロイ」を中止し、  
該当ノードをダブルクリックして、  
プロパティの設定を見直します。

以下のノードは、正しくプロパティが設定されていません:

- [フロー 1] Pin: (grovepi-ultrasonic)
- [フロー 1] gauge (ui\_gauge)

このままデプロイしても良いですか？

中止

デプロイの確認

# トラブルシューティング⑥ | 過去のQ&A①

質問

Node-REDで条件分岐したい

switchノードを使います。  
プロパティを開くと一行目に

== 空白 → 1

となっています。

意味は

「空白の文字や数値が等しい場合、入力された値を1から出力する」  
です。

このプロパティの下の方に[追加]ボタンがありますので、押すと条件を増やすことができます。

条件を増やした場合、さらに下の方にある動作設定を  
「最初に合致した条件で終了」  
にしてください。

過去にあった質問と回答を集めてみた  
聞く前に要チェック！

回答

質問

Node-RED上でGroveボタンを使った時に、複数回の入力が検出される（チャタリングが起きる）

Node-REDの「デプロイ」の設定を「変更したフロー」に変えて、もう一度デプロイしてみてください。  
この時、再デプロイの前に、

```
$node-red-stop  
$node-red-start
```

と順に実行し、Node-REDサービスをリセットすると改善される場合があります。

それでも改善しない場合は、Raspberry Pi本体を再起動した後に、上記の操作をすると解決する場合があります。

ちなみに、Groveボタンのノードは状態が変化したらmsgを送ります。つまり、1プッシュにつき押した時と離した時で2回のmsgが生じます。

Node-REDでその他のセンサー値に異常がある場合（ある程度の頻度で異常値が生じる場合）、上記の方法で解決する場合があります。

回答

# トラブルシューティング⑥ | 過去のQ&A②

質問

Grove超音波センサーをGrovePi本体から2m~10m離れた場所に置きたい

回答

ハブを使わず、比較的簡単にケーブルを延長する方法を2つ示します。

1. 4ピンケーブルをGrovePi側と超音波センサー側にそれぞれ接続し、4ピンケーブルのコネクタの穴に延長するケーブルをつなげる方法
  - ・ ケーブルの種類には単線とより線があり、単線であれば適切な太さのケーブルを用意すればはんだ付けが不要です。
  - ・ より線は柔らかいため、コネクタの穴にさす部分をはんだ付けするか、ピンヘッダに一度はんだ付けしてからピンヘッダをコネクタに接続ことになります。
2. 4ピンケーブルを中間で切断し、切断した部分に延長するケーブルをはんだ付けする方法
  - ・ はんだ付けした部分は被覆がないため、ショートしないように絶縁テープなどで保護する必要があります。

なお、ケーブルを長くする場合に発生する問題がありますので注意してください。

- ・ 長くすればするほどケーブルの抵抗値が高くなり信号が弱くなる（5Vで接続しても相手側は5Vにならない）ことがあります。
- ・ 周りの電波の影響を強く受けるため、ノイズの影響で正しく信号を送れないことがあります。

上記のことから、ケーブルはできるだけ短くすることをお勧めします。

技術  
情報

遠隔地のセンサー値を取得するには、UDP接続するのが比較的簡単だと思います。

■送信側（センサー側）の構成

- [ハードウェア] センサー本体(A)、センサー駆動およびWi-Fiモジュールとの通信が可能なマイコン等(B)、Wi-Fiモジュール(C)
- [ソフトウェア] センサーを駆動し、取得値をパケットとして、指定したアドレスにUDPプロトコルで送ります。

■受信側（アプリケーション側）の構成

- [ハードウェア] Wi-Fi接続が可能な機器(D)
- [ソフトウェア] UDPサーバを動作させ、UDPプロトコルのパケットとしてセンサー値を受け取ります。

この方法では、同じWi-Fiルータにつながる範囲であれば、無線で接続できます。

Raspberry Piを2台使えるのであれば、センサー側の(B)と(C)を兼ねたものとして1台、受信側としてもう1台のRaspberry Piを充てれば、上記の構成が実現できます。

また、各ソフトウェアはNode.jsやNode-REDで実現できます。

Raspberry Piを2台使うのではなく、センサー側の(B)と(C)としてプログラム可能なWi-Fiモジュールや、マイコン+Wi-Fiモジュールの組み合わせを用いる方法もあります。こちらの方が少し安価ですが、マイコン等のプログラミングが必要になります。  
プログラム可能なWi-Fiモジュールの例としてArduino IDEを使う開発があります。

# トラブルシューティング⑥ | 過去のQ&A③

質問

他のノードとデータを共有したい

グローバル変数を使いましょう。

JavaScriptの場合は、関数の外に変数を宣言します。

Node-REDの場合は、functionノードで以下のような使い方ができます。

```
// グローバル変数sumが無ければ0を入れて用意、ある場合はsum変数の値を取得し、ローカル変数sに代入
var s = global.get("sum")||0;
//sにこのノードに入力された値を入れる
s += msg.payload;
//sumにsの値を代入
global.set("sum",s);
msg.sum = s;
return msg;
```

回答

後は、他のfunctionでglobal.getやsetで参照できます。

また、上記を使えばカウンタも作れます。

```
var count = global.get("counter")||0;
count += 1;
global.set("counter",count);
msg.counter = count;
return msg;
```

上記のようにfunctionを作った後、debugノードなどで、msg.counterを参照します。

# トラブルシューティング⑥ | 過去のQ&A④

質問

Grove GPSモジュールの使い方がわからない

回答

Grove GPSモジュールの商品説明ページは以下にあります。

<https://www.switch-science.com/catalog/1179/>

Raspberry PiのPythonで、GPSモジュールを使う設定方法が以下に書いてあります。

<https://qiita.com/AmbientData/items/fff54c8ac8ec95aeeee6>

手順によっては起動時に固まって進まなくなるので、その時は、GrovePiをRaspberry Piから抜いて、続きの設定を行いましょう。

質問

Node-REDのfunctionノード（JavaScript）で、カンマ区切りのデータ文字列を配列に格納したい

回答

split(",")メソッドで実現できます。

以下のページに「カンマ区切りの入力文字列を配列に分割して格納する方法」があります。

<https://www.nishishi.com/javascript-tips/split-comma.html>

ユーザが入力した「カンマ区切り」の文字列を、JavaScriptを使ってカンマ記号で分割して配列に格納するには、splitメソッドを使うと楽です。ユーザに入力してもらう文字列の個数が不定の場合には、入力欄を事前にたくさん用意しておくよりもカンマ区切りで入力してもらう方が入力フォームをシンプルにできるでしょう。区切り文字にはもちろんカンマ記号以外の文字も使えます。

# トラブルシューティング⑥ | 過去のQ&A⑤

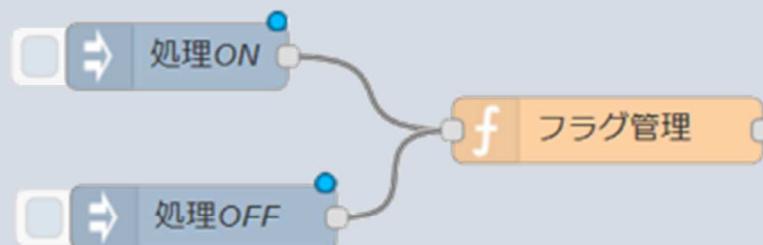
質問

Node-REDで特定の処理時間をカウントしたい

グローバル変数のフラグを使うと特定の処理だけカウントしやすいです。  
Node-REDで処理がONのときにカウントする処理を図で示します。

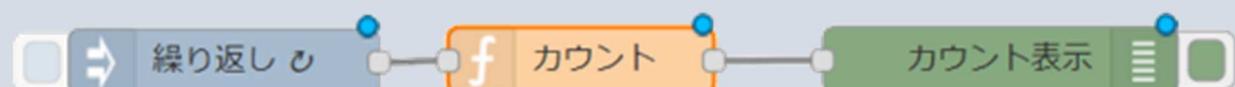
■function「フラグ管理」の内容

```
if(msg.payload === true){  
    global.set("flag",true);  
}else{  
    global.set("flag",false);  
}
```



■function「カウント」の内容

```
var cntflag = global.get("flag")||0;  
var cnt = global.get("count")||0;  
if(cntflag === true){  
    msg.payload=cnt;  
    cnt++;  
    global.set("count",cnt);  
    return msg;  
}else{  
    cnt = 0;  
    global.set("count",cnt);  
}
```



※trueとfalseは1と0に置き換えて大丈夫です。

delayを使ったループによるカウントは、複数回実行すると処理が重複して正しく動作しないことがあります。  
こちらの方法は処理が重複しないため、その問題はありません。

回答

# トラブルシューティング⑥ | 過去のQ&A⑥

質問

LCD RGBバックライトで、日本語の表示はできますか

日本語は、半角カタカナと一部の漢字（「千」、「万」、「円」）が表示できます。

JavaScript(Node.js、Node-RED) 上で、`String.fromCharCode()` などにより文字コードを指定しましょう。

以下のページは他の液晶を用いた例ですが、同様に文字コードが指定可能です。

<https://qiita.com/autani/items/1b82d1fc5822dcfddb7d>

文字コード表  
(下位4bit 横軸表記)

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
00															
10															
20	!	*	#	\$	%	&	'	( )	*	+	-	,	.	/	
30	0	1	2	3	4	5	6	7	8	9	:	;	=	>	?
40	Ⓐ	Ⓑ	Ⓒ	Ⓓ	Ⓔ	Ⓕ	Ⓖ	Ⓗ	Ⓘ	Ⓛ	Ⓚ	Ⓛ	Ⓜ	Ⓝ	Ⓞ
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	]	^	-
60	‘	a	b	c	d	e	f	g	h	i	j	k	¥	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{	}	-	-
80															
90															
A0	.	「」	、	・	ヲ	ア	イ	ウ	エ	オ	ヤ	ユ	ヨ	ツ	
B0	—	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ
C0	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ
D0	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	・
E0	α	α	β	ε	μ	ο	ρ	φ	χ	π	×	¢	ℳ	₪	₪
F0	p	q	ø	∞	Ω	u	Σ	π	x	y	干	万	円	÷	■

回答

# 付録 | 初期セットアップの詳細

※2019/11/24現在

Raspberry Pi にOS (Raspbian) 導入後の手順です。

## 1. OS(Raspbian) のアップデート

```
sudo apt-get update  
sudo apt-get upgrade
```

## 2. Node-RED 最新版インストール

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-deb-package/master/resources/update-nodejs-and-nodered)
```

※上記コマンドは1行で入力実行してください。スクリプトは日本Node-REDの会、公式より最新情報を確認してください。  
(<https://nodered.jp/docs/getting-started/raspberrypi>)

## 3. Node.js のダウングレード ※Grove Nodeは旧Ver仕様の為

```
n list //ダウンロード可能なバージョン一覧を表示  
n 9.11.2 //バージョンを指定してインストール
```

## 4. GrovePi+のソフトウェアのインストール

```
sudo curl -kL dexterindustries.com/update_grovepi | bash  
sudo reboot  
cd Dextor/Grovepi/Script/  
sudo chmod +x install.sh  
sudo ./install.sh
```

## 5. GrovePi+のNode.jsに対応したライブラリのインストール

```
sudo npm install node-grovepi
```

## 6. Node-REDのGrovePi+ノードのインストール

```
sudo npm install -g node-red-grovepi-nodes
```

おまけ. RSTライト点灯時 (GrovePi+が動作しない) のリセットコマンド

```
avrdude -c gpio -p m328p
```

# 用語集①

用語	意味
ハンズオン	体験学習を意味する教育用語で、実際に手で触れるなどの体験を通じて、より理解を深めることを目的とする。
GPIO (ジーピーアイオー)	General Purpose Input/Outputの略称で「汎用入出力」の意。本講習では、Raspberry Pi と GrovePi+ ボードの接続に使用する。
I2C (アイツーシー) (アイスクエアシー)	Inter-Integrated Circuitの略称で、主に同じ基板上の回路やチップの間で通信する用途などを想定しており、組み込み機器や携帯電話などで利用されている。
Raspbian (ラズビアン)	Raspberry Piに対応したオペレーティングシステムであり、小規模な開発者向けの Linux (debianベース) のコンピュータOS。
Node.js (ノードジェイエス)	サーバ側でプログラムを動作させることができる、スケーラブルなネットワークアプリケーションを構築するために設計された非同期型のイベント駆動のJavaScript環境。
Node-RED (ノードレッド)	APIやオンラインサービス、デバイスなどの機能を繋げてアプリケーションを作成していくプラットフォームであり、処理フローをブラウザの操作によって作成することができる。
コマンド プロンプト	利用者にキーボードからコマンド入力を促し、システムが命令入を受け付けられる状態を示すために表示されるもの。

## 用語集②

用語	意味
GUI (ジーユーアイ)	Graphical User Interfaceの略称でUIの類型の一つ、コンピュータへ出す命令や指示を、ユーザが画面上で視覚的に捉えて行動を指定できるもの。
127.0.0.1	ローカル・ループバック・アドレスと呼ばれ、自分自身を指す特別なIPアドレスであり、自分自身の上で動作しているサービスへ接続する場合は、このIPアドレスを利用できる。
SSH (エスエスエイチ)	Secure Shellの略称で、主にUNIXコンピュータで利用される、ネットワークを介して別のコンピュータにログインして操作するためのソフトウェアの一つ。
VNC (ヴィエヌシー)	Virtual Network Computingの略称で、 ネットワークを通じて別のコンピュータに接続し、 他コンピュータの画面を遠隔操作できるリモートデスクトップソフト。 Raspbian OSにはVNCサーバが組み込まれているため、 設定を有効化すればVNCクライアントから接続できるようになる。 ※VNCクライアントのVNC Viewerは、下記URLからダウンロード可能。 <a href="https://www.realvnc.com/en/connect/download/viewer/">https://www.realvnc.com/en/connect/download/viewer/</a>
npm (エヌピーーエム)	Node Package Managerの略称で、JavaScript開発者がコードを共有したり、 シェアしているコードをアップデートすることを容易にする JavaScript系のパッケージを管理するツール。
日本語入力メソッド fcitx-mozc	Raspberry Pi などは日本語入力ができないので、 日本語入力環境をインストールすることによって日本語入力が可能になる。

## License

### The MIT License (MIT)

GrovePi for the Raspberry Pi: an open source platform for connecting Grove Sensors to the Raspberry Pi. Copyright (C) 2018 Dexter Industries

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



### ライセンス

本テキストは、クリエイティブ・コモンズ・ライセンス（表示 4.0 国際）「CC-BY（表示）」の下に提供されています。

本テキストに記載されている内容は、クリエイティブ・コモンズ・ライセンス（表示4.0国際）に従うかぎり、商用利用を含め、自由に複製または改変してご利用いただくことが可能です。ご利用いただく際は、著作者の名前を、作品タイトルおよびクリエイティブ・コモンズ・ライセンス（表示 4.0 国際）のURL表記、またはハイパーリンクと共に、ご利用いただく媒体の閲覧者が見える場所に記載していただく必要があります。

- ・著作者 : Web×IoT メイカーズチャレンジ 実行委員会（篠田）  
CHIRIMEN Open Hardware コミュニティ (<https://chirimen.org/>)
- ・作品タイトル : 「Web×IoT メイカーズチャレンジ ハンズオン講習テキスト」
- ・ライセンス : クリエイティブ・コモンズ・ライセンス（表示 4.0 国際）
- ・URL : <https://creativecommons.org/licenses/by/4.0/>



END

---

ハンズオン講習会テキスト