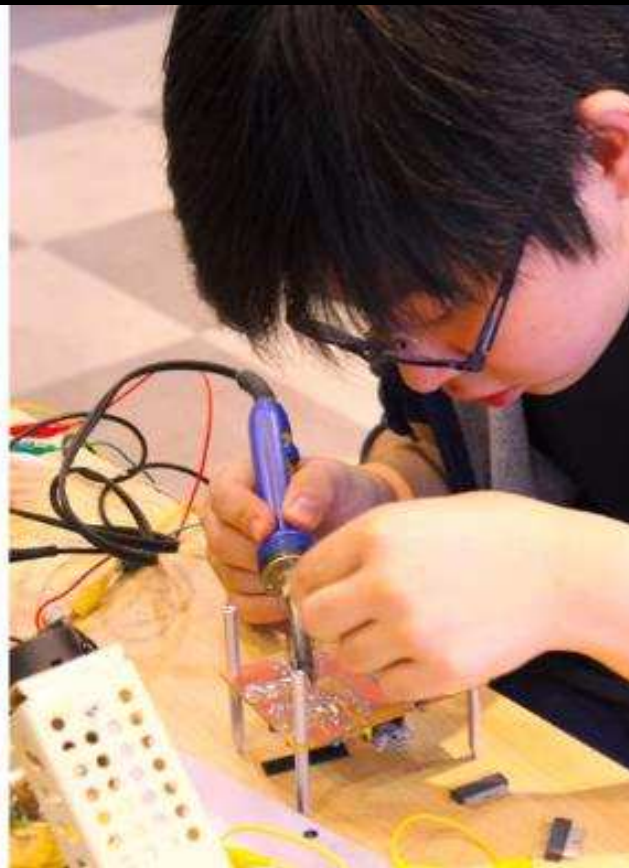




# Web×IoTメイカースチャレンジ ハンズオン講習テキスト



CHIRIMEN Open Hardware

<https://chirimen.org/>

運営事務局：YRP研究開発推進協会／株式会社ブール・ジャパン



# 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  
  うまくいかなかったら？  
  非同期処理について／GPIO とは  
  GPIOPort の処理まとめ

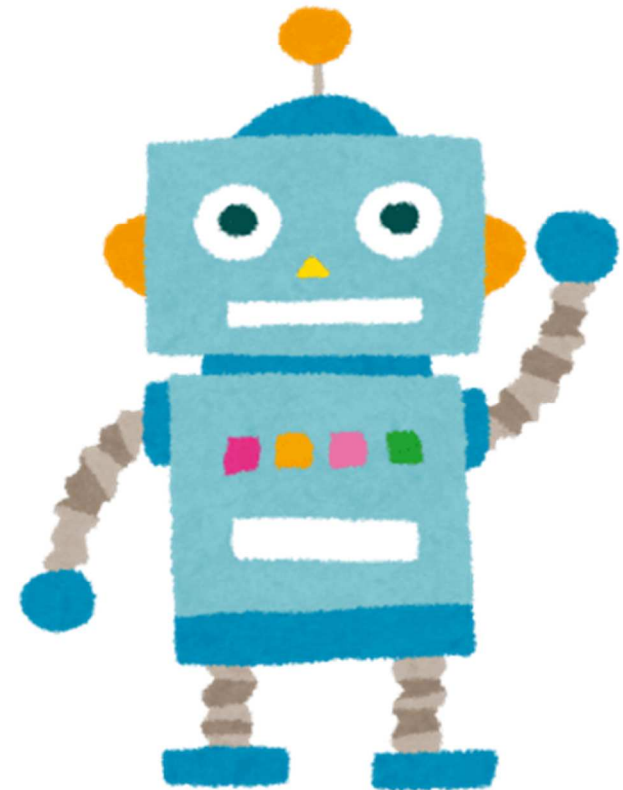
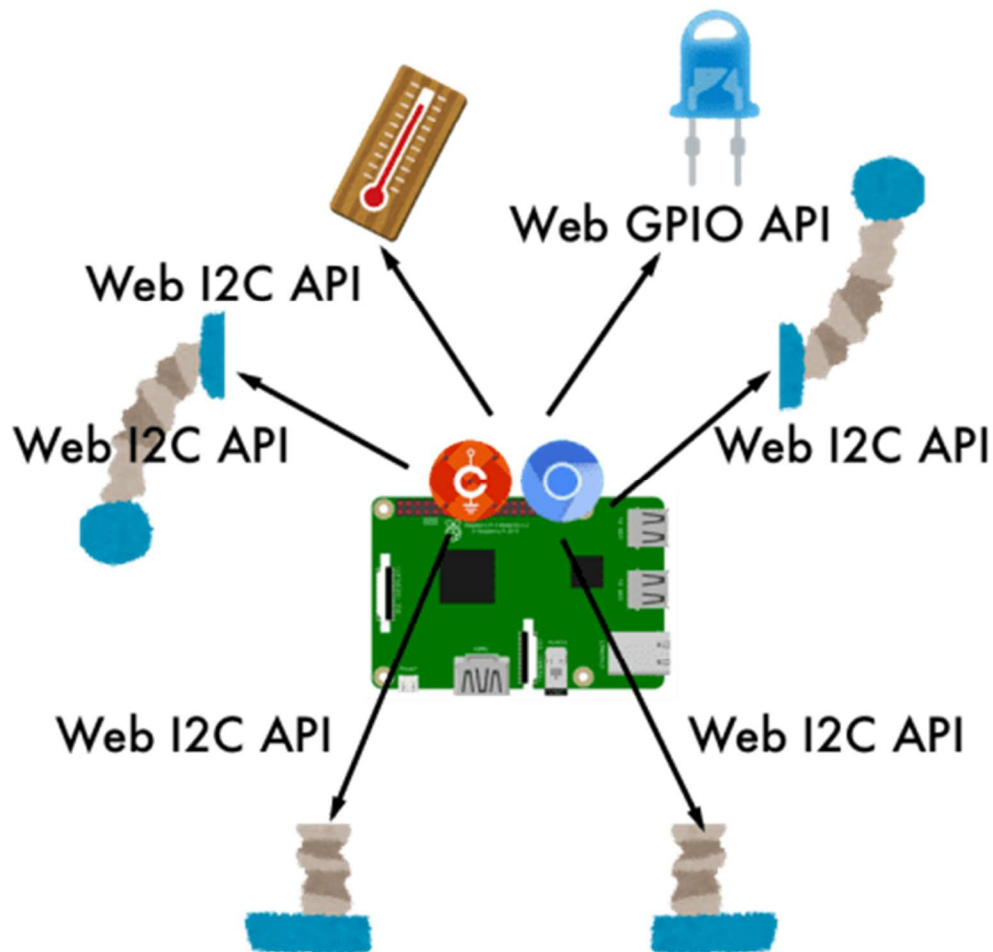
## 6. ここまでのまとめ

## 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. ここまでのまとめ

# 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。



ブラウザで制御されたロボットなど



## 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。

## 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




⑩ ジャンパーワイヤー <オス-メス> ×5




⑪ ジャンパーワイヤー <メス-メス> ×4




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





⑫ ジャンパーワイヤー <オス-オス> ×4

★ チュートリアル情報

このスター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/>





⑬ micro SDカード ×1

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>① ブレッドボード × 1</li> <li>② ちびギアモーター × 1</li> <li>③ LED × 1</li> <li>④ MOSFET × 1</li> <li>⑤ タクトスイッチ × 1</li> <li>⑥ 温度センサー&lt;ADT7410&gt;</li> </ul> | <ul style="list-style-type: none"> <li>⑦ 10kΩ抵抗&lt;茶黒橙金&gt; × 1</li> <li>⑧ 1kΩ抵抗&lt;茶黒赤金&gt; × 1</li> <li>⑨ 150Ω抵抗&lt;茶緑茶金&gt; × 1</li> <li>⑩ ジャンパーワイヤー (オス-メス) × 5</li> <li>⑪ ジャンパーワイヤー (メス-メス) × 4</li> <li>⑫ ジャンパーワイヤー (オス-オス) × 4</li> <li>⑬ Micro SDカード × 1</li> </ul> |
|--|---|

## 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)



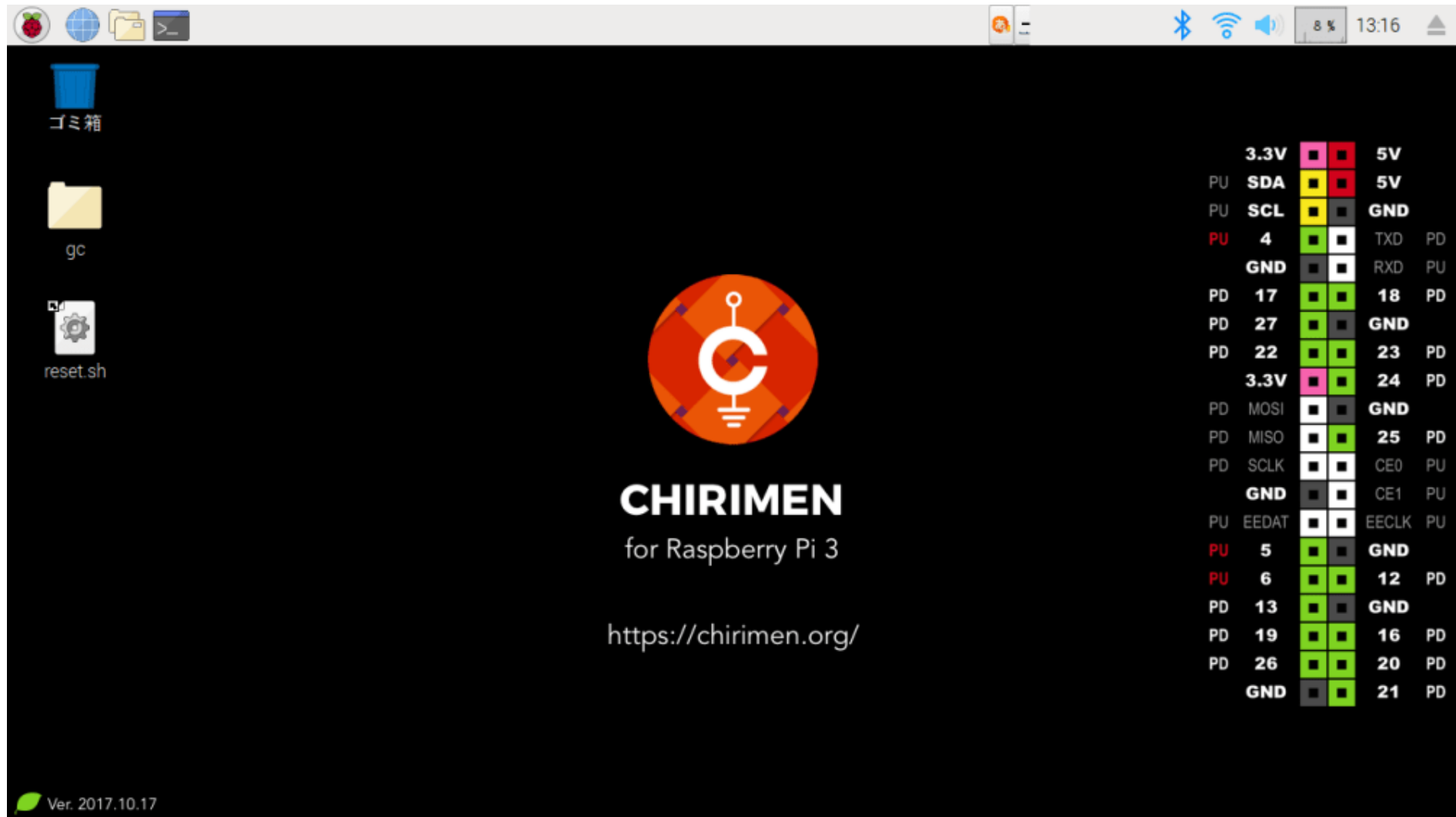
1. モニタ電源をコンセントに接続
2. RaspiにmicroSDカードを挿入
3. HDMIケーブルでRaspiとモニタを接続
4. マウスをUSBポートへ接続
5. キーボードをUSBポートへ接続
6. microUSB電源ケーブルをコンセントへ
7. microUSB電源ケーブルをRaspiに接続



# 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！



## 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

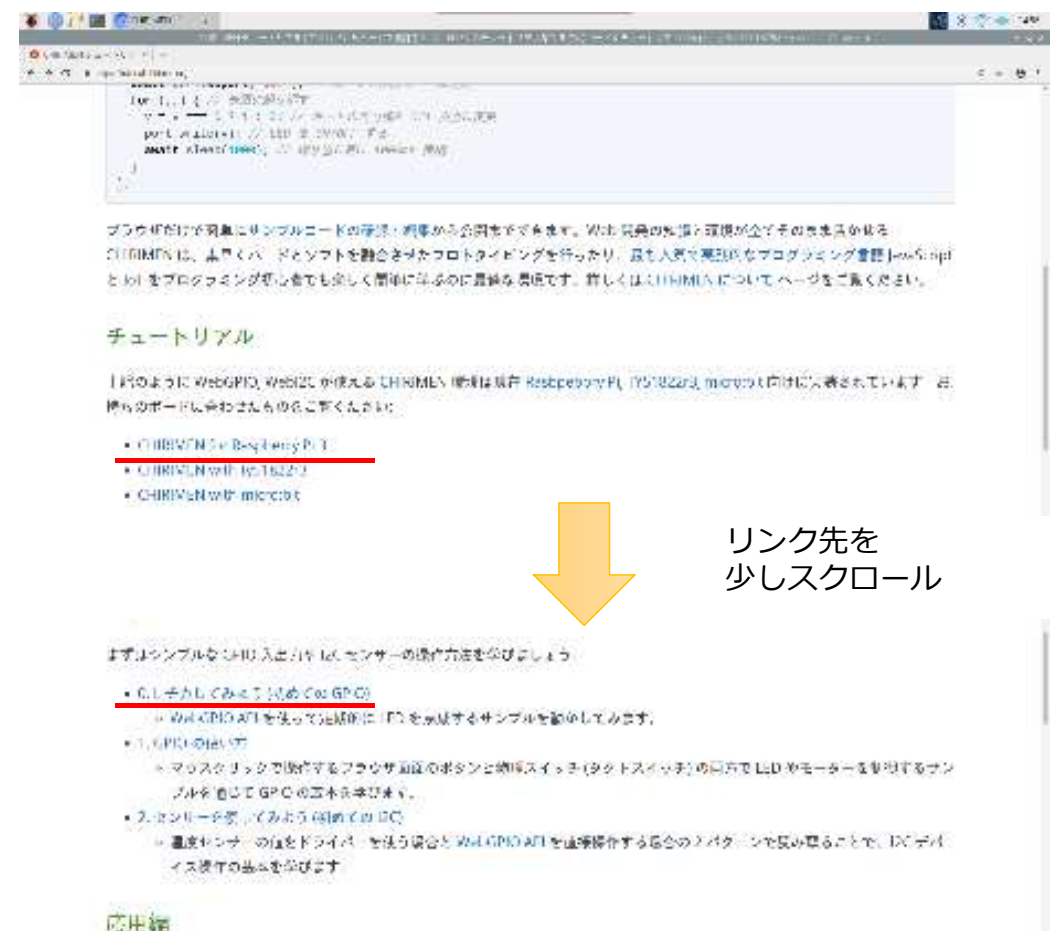
## そもそも「L チカ」って何？

「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。



画面に出ている「ちゅーとりある！」から  
リンクをたどっていきましょう。



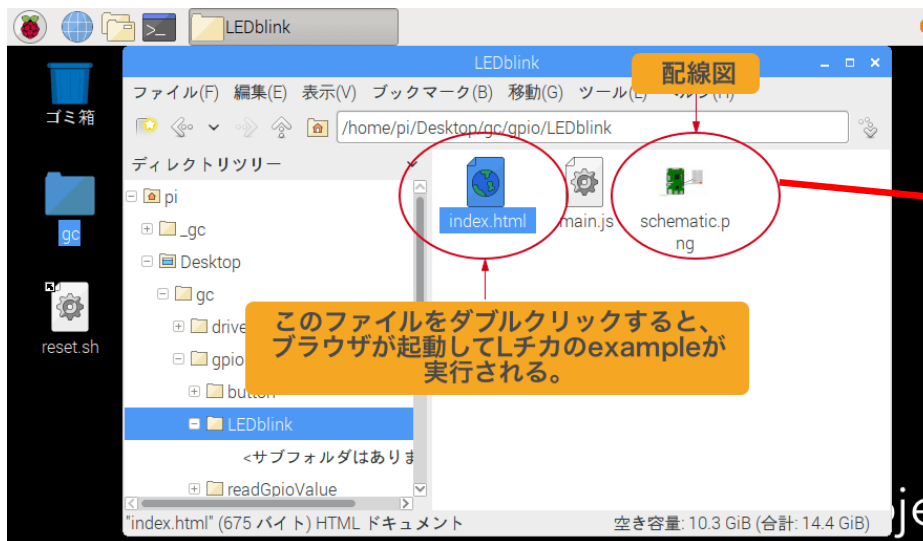
# 「L チカ」をやってみよう

## 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
```

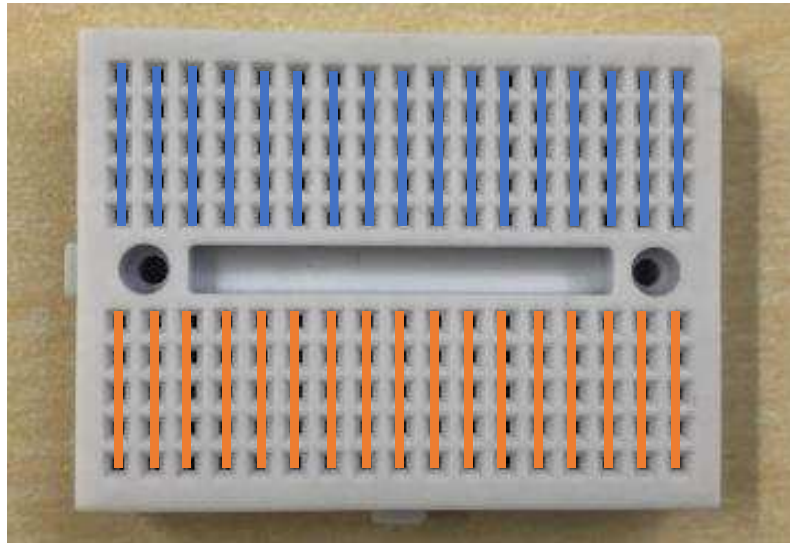
LED のリード線の方角に注意しながら、この図の通りにジャンパーワイヤやブレッドボードを使って配線してみましょう。



# 「Lチカ」をやってみよう

## ■ ブレッドボードの使い方

表面

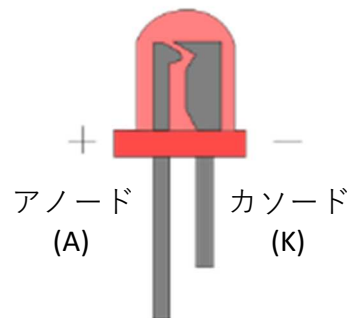


裏面

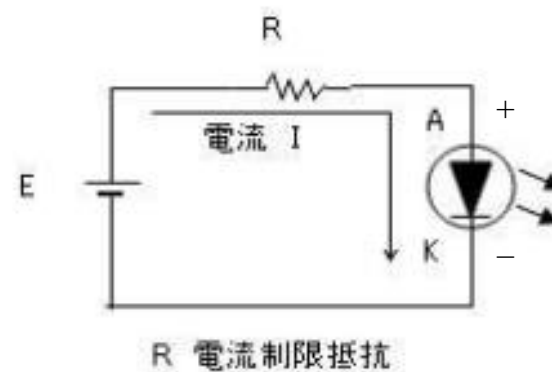


写真の様に1列5ピンでつながっています。電子パーツを差し込んで回路を作ります。  
※少し硬い場合もあります、しっかり中まで差し込みましょう。

## ■ LED（発光ダイオード）の使い方



回路図



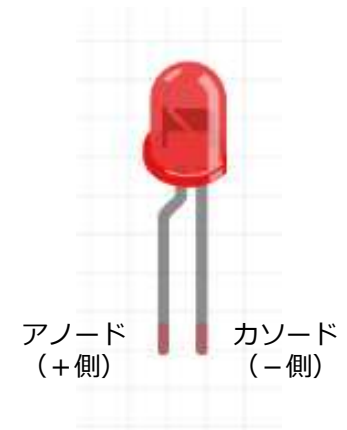
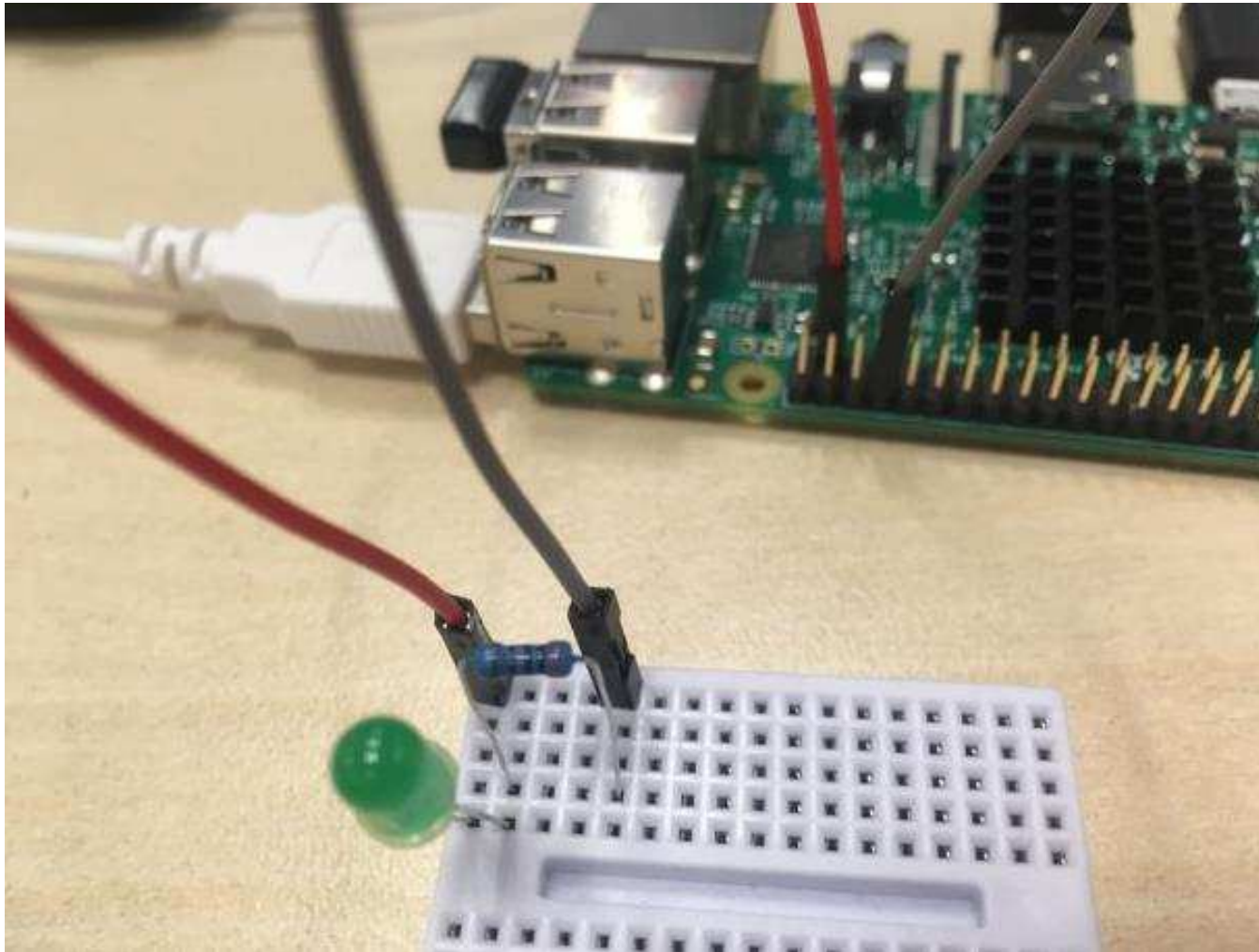
基本回路)

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

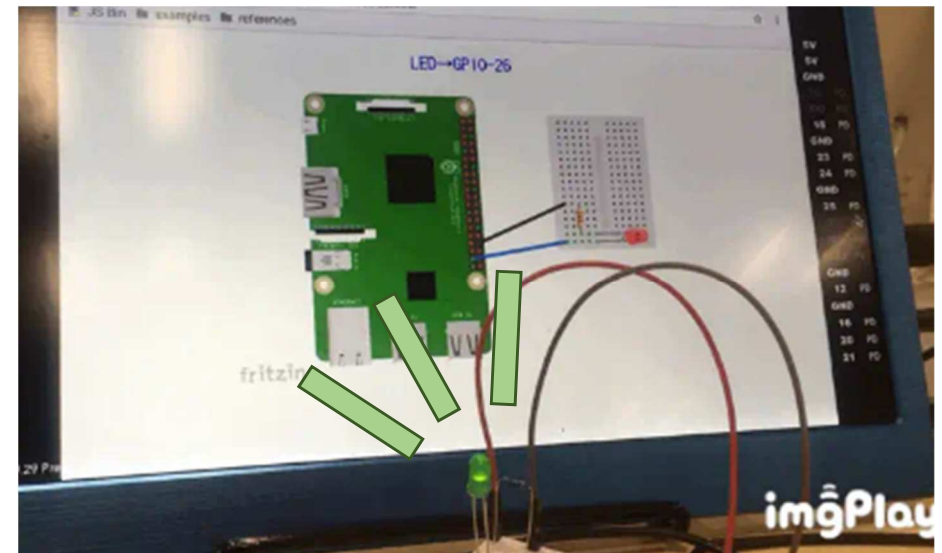
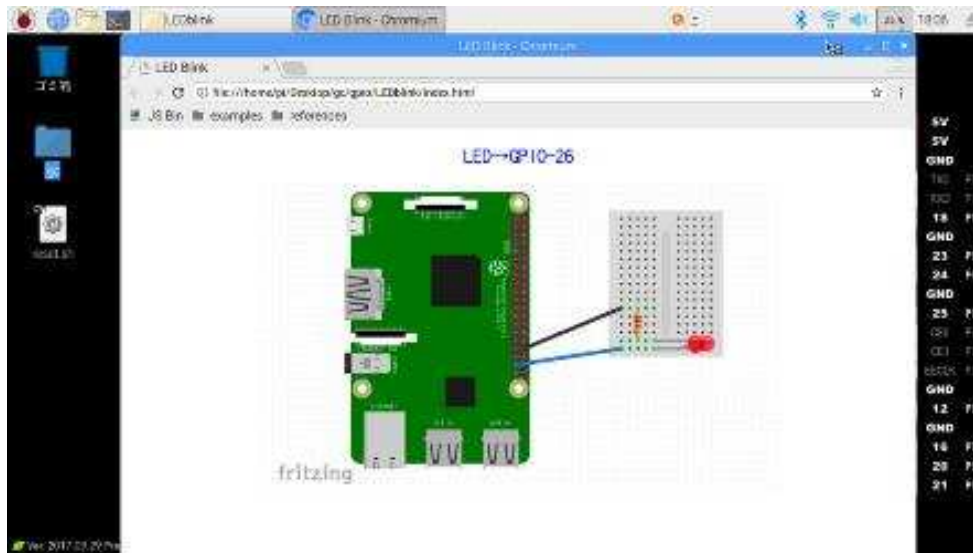
## example を実行してみる

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？！

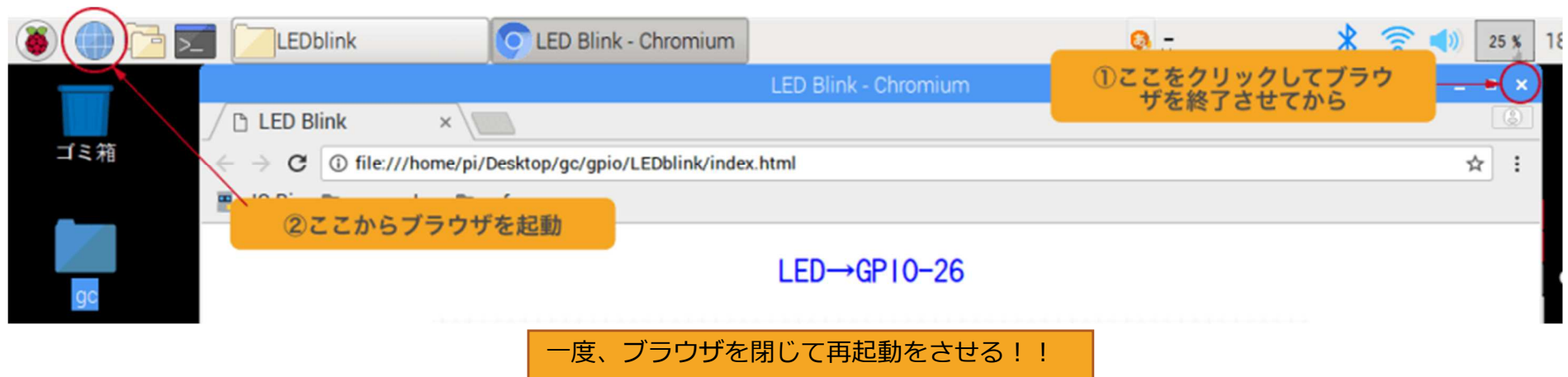
# 「L チカ」をやってみよう

## 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く※なります。**



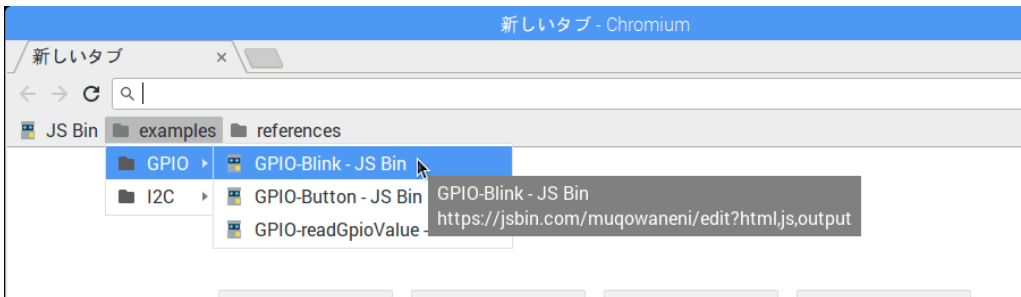
※CHIRIMEN Raspi3 での GPIO などの操作には排他制御があり、同一の GPIO ピンを複数のプログラムから同時操作はできません。

# 「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。

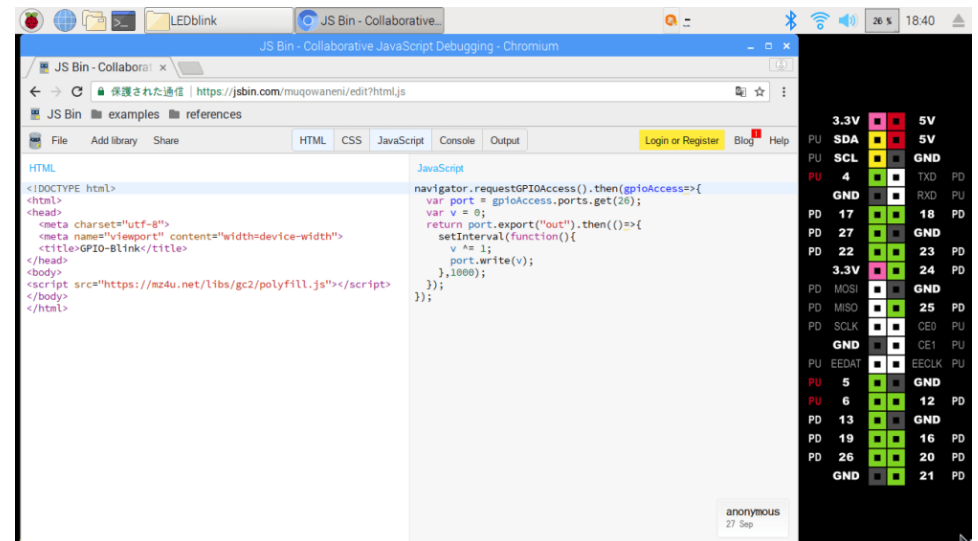
ブックマークバーから、以下を選んでアクセスしてください。

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。



# 「Lチカ」をやってみよう

## JS Bin の 画面構成

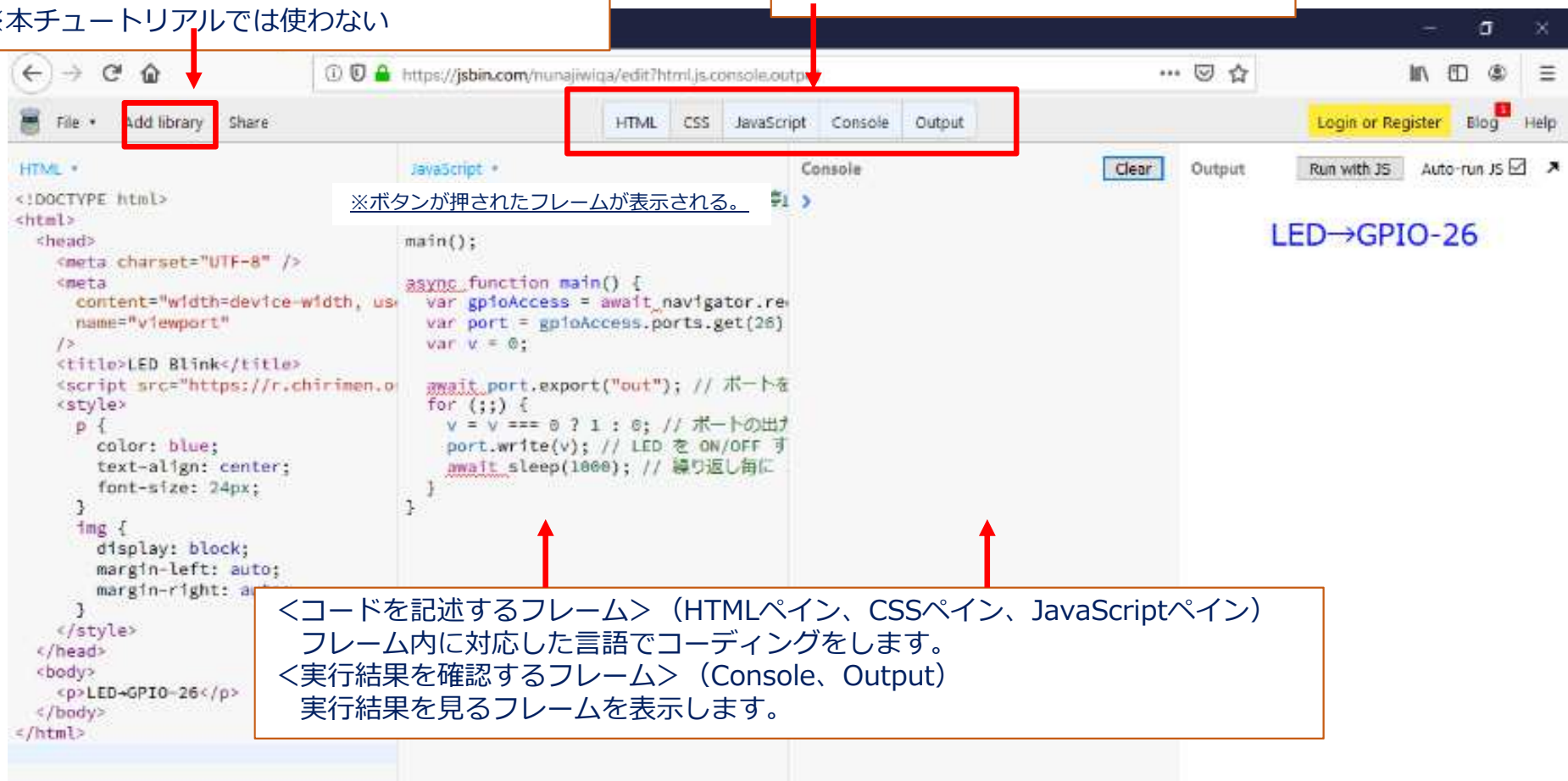
JS Binとは、ブラウザ上でJavaScriptを動作させる事が出来るサービスです。いくつかのフレームワークにも対応しています。その他、類似のサービスが複数（JSFiddleなど）あります。

＜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。







# 「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ませます。

```
<!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

## JavaScript

JavaScript のコーディングを以下の様に記述します。

```
async function mainFunction() {  
  ① // 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  
  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ますので、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。

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。

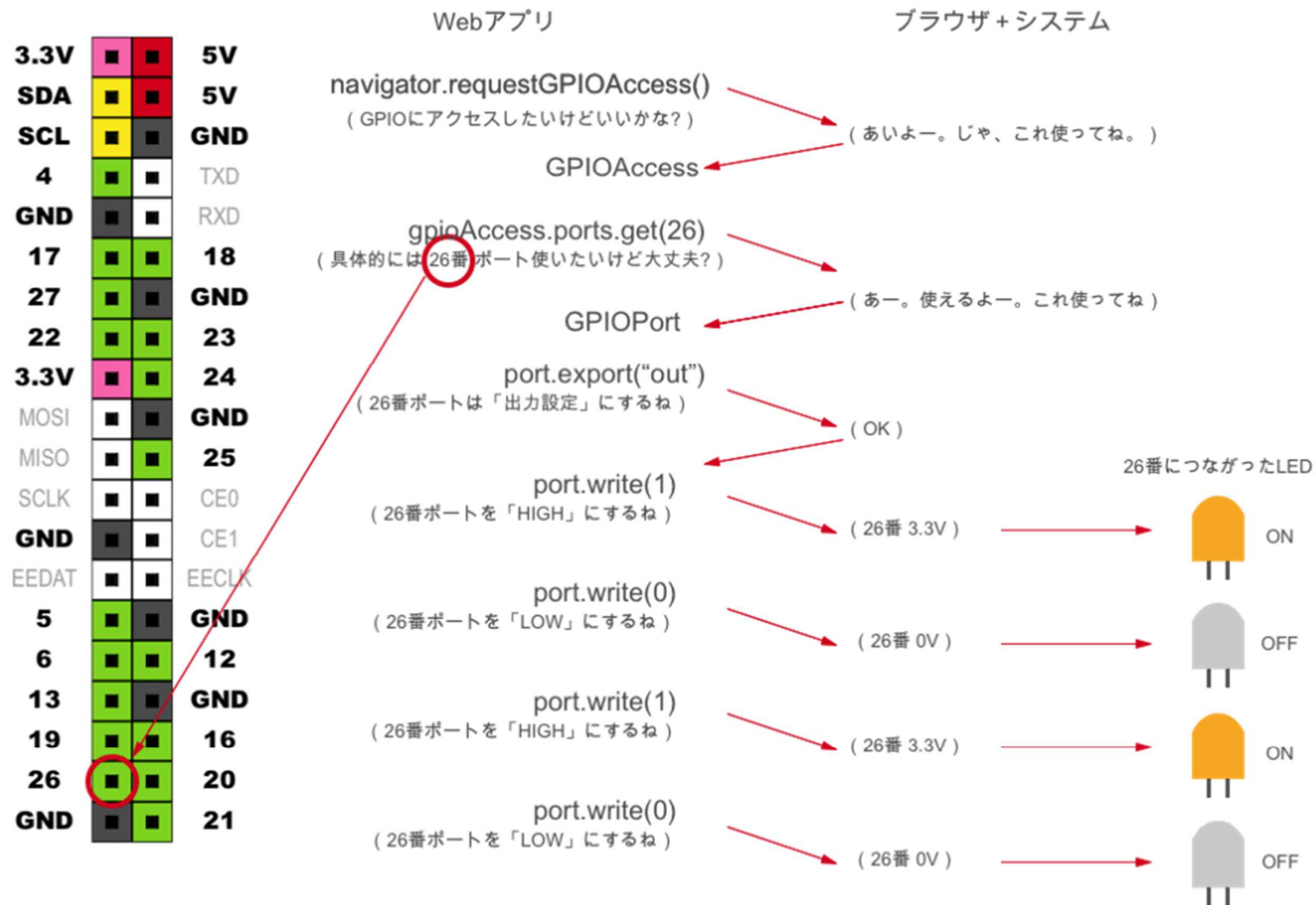
3.3V			5V
SDA			5V
SCL			GND
4			TXD
GND			RXD
17			18
27			GND
22			23
3.3V			24
MOSI			GND
MISO			25
SCLK			CE0
GND			CE1
EEDAT			EECLK
5			GND
6			12
13			GND
19			16
26			20
GND			21

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



# 「Lチカ」をやってみよう

## GPIOPort の処理まとめ



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：

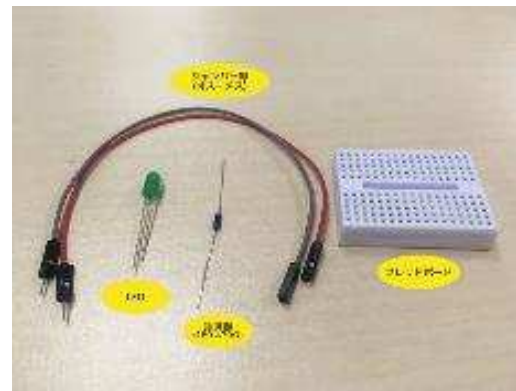
- [GitHub リポジトリで参照](https://github.com/chirimen-oh/tutorials/tree/master/raspi3/examples/section0)  
(<https://github.com/chirimen-oh/tutorials/tree/master/raspi3/examples/section0>)

## 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チカに必要なパーツ」



- タクトスイッチ (2pin) x 1
- ジャンパーワイヤー (オス-メス) x 5
- Nch MOSFET (2SK4017)
- リード抵抗 (1K $\Omega$ ) x 1
- リード抵抗 (10K $\Omega$ ) x 1
- ちびギアモータ x 1

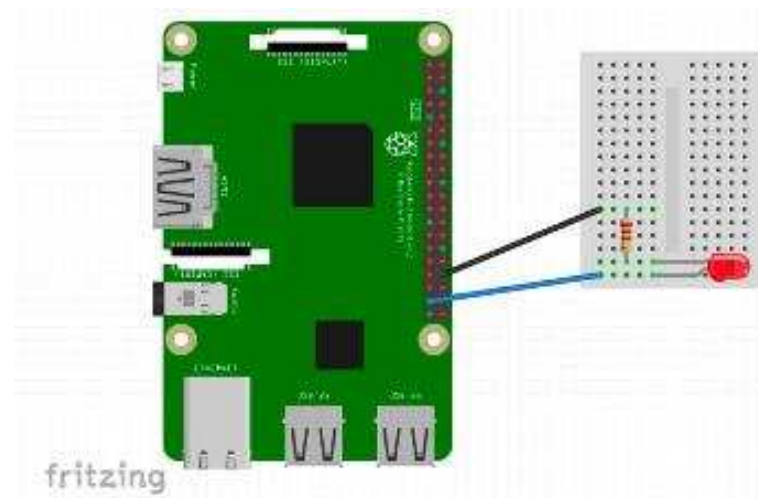
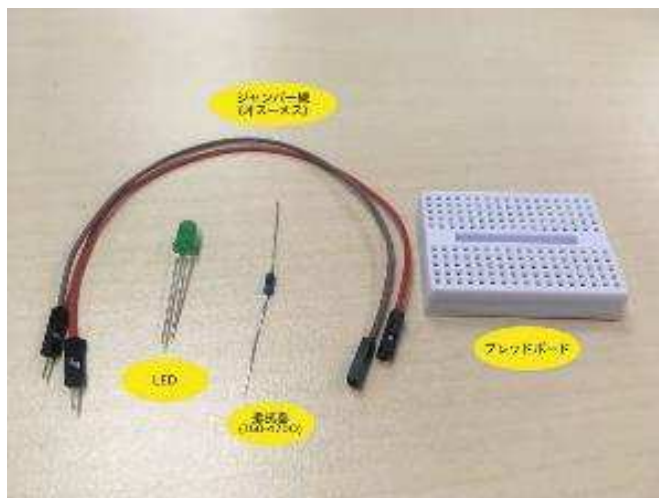
## マウスクリックで LED の ON/OFF を制御してみる

L チカしてみよう では、**JS Bin** を使って L チカの example コードを少し触っ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う安心です。

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。

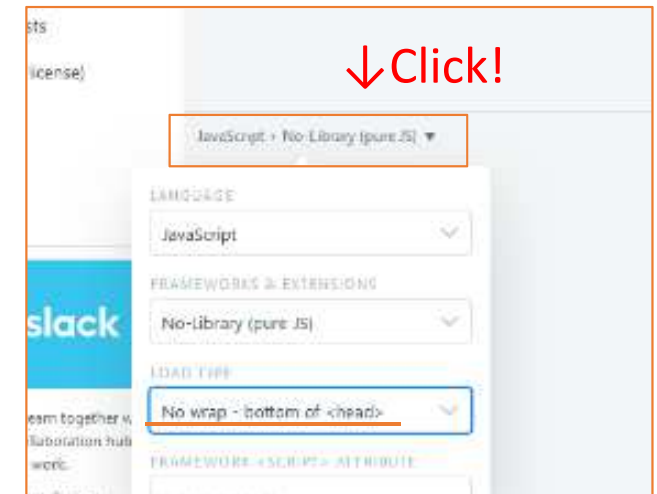
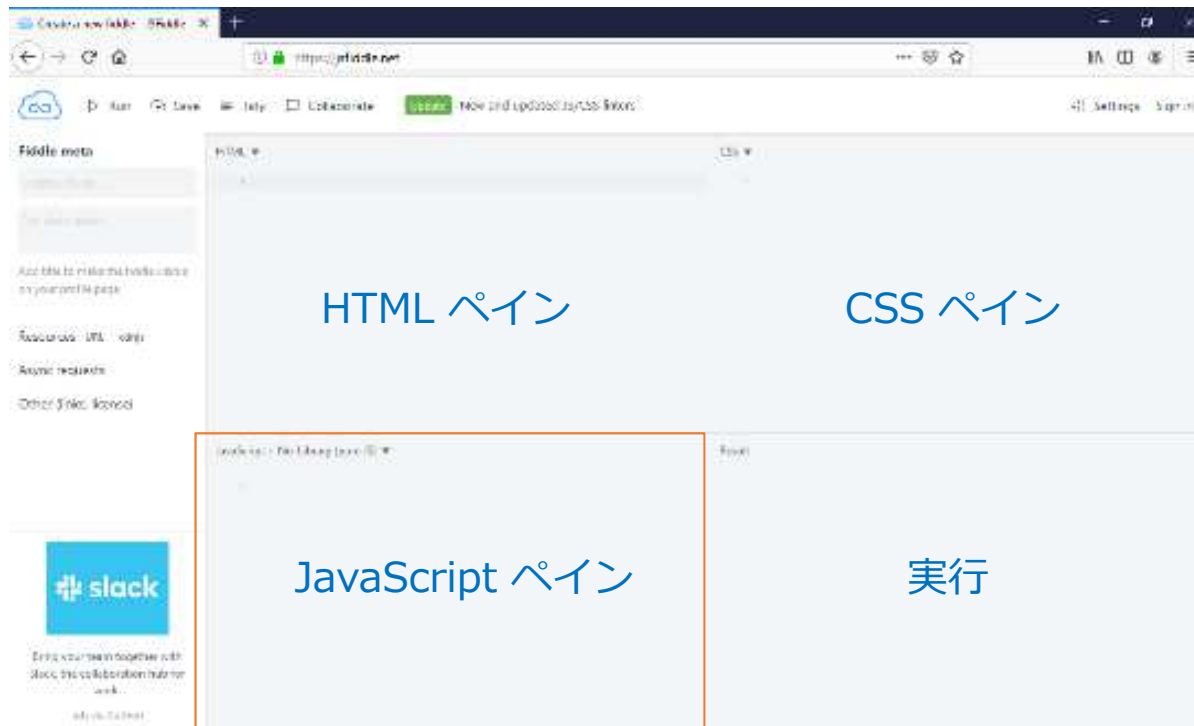
このパートでは「L チカしてみよう」で実施した L チカの配線をそのまま利用します。必要な部品も同じです。  
LED は、**26 番ポート**に接続しておいてください。





## 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。



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 つずつ作ります。

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**」部分を作ってみます。

```
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できるようになったら成功です。

## 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";
  };
};

```




```

1 <button id="onoff">LED ON/OFF</button>
2 <div id="ledview"></div>
3 <script src="https://mz4u.net/libs/gc2/polyfill.js"></script>
4
5 (async ()=>{
6   var onoff = document.getElementById("onoff");
7   var ledview = document.getElementById("ledview");
8   var v = 0;
9   var gpioAccess = await navigator.requestGPIOAccess();
10  var port = gpioAccess.ports.get(26);
11  await port.export("out");
12  onoff.onclick = ()=>{
13    v = v === 0 ? 1 : 0;
14    port.write(v);
15    ledview.style.backgroundColor = (v) ? "red" : "black";
16  };
17 })();

```

**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

ここでもういちど GPIO を使う流れをおさらいします。

## `await navigator.requestGPIOAccess()`

まずは **Web GPIO を利用する GPIOAccess インタフェースを取得** します。

`requestGPIOAccess()` は非同期処理でインタフェース初期化を行う非同期メソッドですので `await` で完了を待ってから次の処理を行います。

## `gpioAccess.ports.get()`

`GPIOAccess.ports` は利用可能なポートオブジェクトの一覧 ([Map](#)) です。

`gpioAccess.ports.get(26)` のようにすることで利用可能なポートオブジェクトの一覧から、

**GPIO ポート番号 26 を指定して port オブジェクトを取得** しています。

## `await port.export()`

`port.export("out")` により取得した GPIO ポートを「**出力モード**」で**初期化** しています。この初期化処理も非同期処理となっているため、`await` を付けて完了を待ってから次の処理に進めます。

GPIO ポートにかける電圧を Web アプリで変化させたい時には「出力モード」を指定する必要があります。一方、GPIO ポートはもうひとつ「入力モード」があり、これは GPIO ポートの状態 (電圧の High/Low 状態) を読み取りたい時に利用します。入力モードについてはスイッチを使う例の中で説明します。

## `port.write()`

`port.write()` は、出力モードに指定した **GPIO ポートの電圧を切り替える** API です。

`port.write(1)` で、指定したポートから HIGH (Raspi3 では 3.3V) の電圧がかかり、

`port.write(0)` で LOW(0V) になります。





# 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

準備：画面のボタンをモーメンタリ動作に変えておく

これまでに作成したプログラムは「ブラウザ画面のボタンをクリックしたら LED の HIGH/LOW を切り替える」というものでした。

クリック後は変更後の状態が維持されます。これは「オルタネート」のスイッチと同じ動きです。一方で、今回用意したタクトスイッチは「モーメンタリ」のものです。

### 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 に変わることはありません。

**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)。

画面のマウス操作がオルタネートでタクトスイッチがモーメンタリと、2 つの動作が混在すると整合性がとれないので、ブラウザ画面のボタンを「モーメンタリ」に合わせて変更しましょう。

## 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

下記のように、現在は onclick イベントで切り替えています。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**押した時と離れた時にそれぞれオンオフ**させるように変えましょう。  
押している間だけオンになる「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";  
};  
:
```

これでマウスもタクトスイッチも同じ拳動で揃いました。



# 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

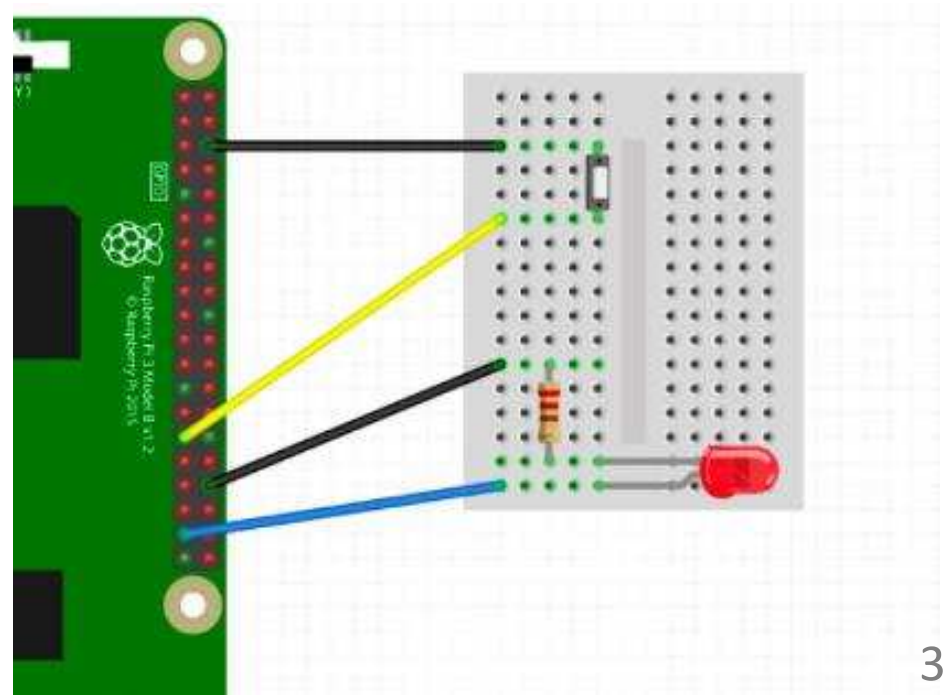
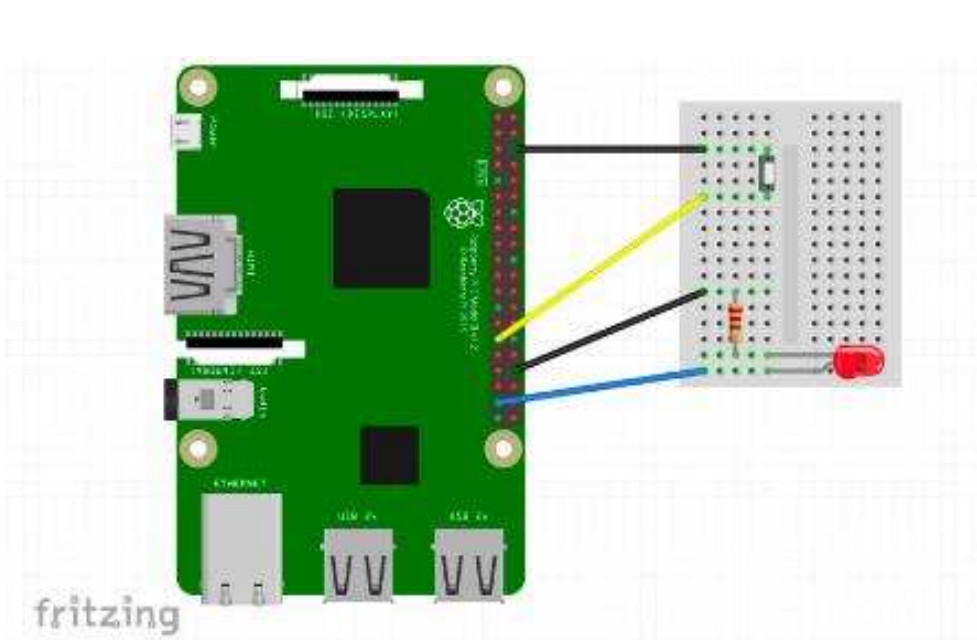
## 部品と配線について



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
- ジャンパーワイヤー（オスメス） × 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。



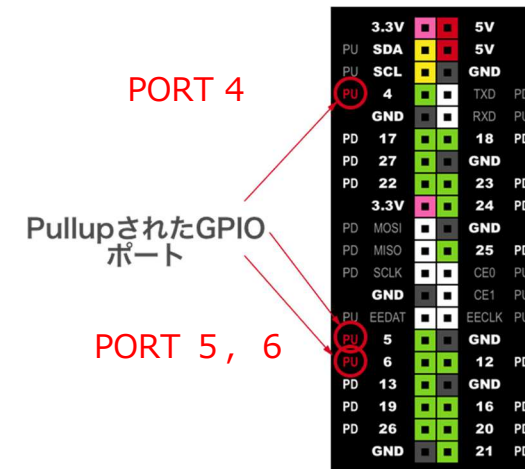
## 今回のスイッチは「プルアップ」回路で接続

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。

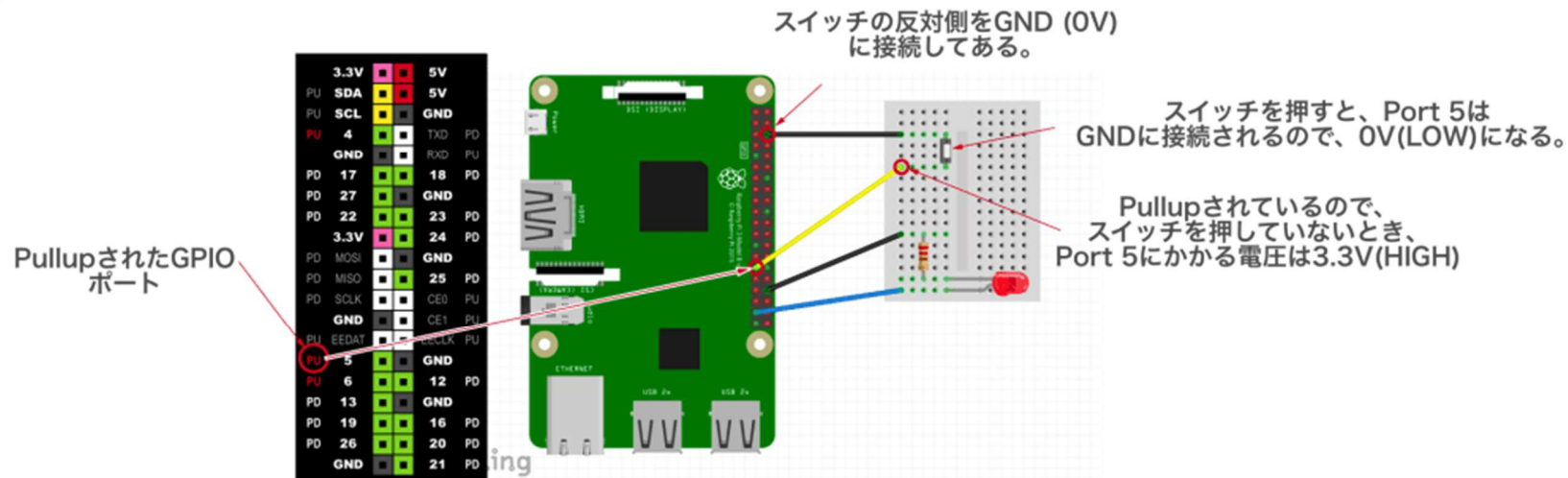
- Port 5 にスイッチを接続
- GND にスイッチの反対側を接続

これでどのようなになるかというと、下記ようになります。

- スwitchを押す前は、Port 5 は HIGH (3.3V)
- スwitch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。







# GPIO の使い方

## スイッチに反応するようにする (port.read())を使ってみる)

いよいよ、スイッチに対応させます。まずは、単純に「GPIO ポートの状態を読み込む」  
port.read() を使います。port.read() で GPIO を読み込むコードは次のように書けます。

```
var gpioAccess = await navigator.requestGPIOAccess(); // writeと一緒に。  
var port = gpioAccess.ports.get(5); // Port 5 を取得  
await port.export("in"); // Port 5 を「入力モード」に。  
var val = await port.read(); // Port 5の状態を読み込む
```

### await port.export()

port.export("in") により取得した GPIO ポートを「入力モード」で初期化 しています。このモードは GPIO ポートにかかる電圧を Web アプリ側から読み取りたい時に使います。初期化は非同期処理であり **await** で完了を待つ必要があることに注意してください。

### await port.read()

port.export("in") で入力モードに設定した GPIO ポートの現時点の状態を読み取ります。読み取りは非同期処理になるので **await** で完了を待つようにしてください。

上記コードで GPIO ポートの読み取りが **1度だけ**行えます。

今回は「スイッチが押され状態を監視する」必要がありますので、定期的に port.read() を繰り返して GPIO ポートの状態を監視する必要があります。



# GPIO の使い方

## スイッチに反応するようにする (port.read())を使ってみる)

LED の処理と組み合わせた全体のコードは次のようになります

```
var ledPort;
var switchPort;

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 mainFunction() {
  var onoff = document.getElementById("onoff");
  var gpioAccess = await navigator.requestGPIOAccess();
  var val;

  ledPort = gpioAccess.ports.get(26); // LED のポート番号
  await ledPort.export("out");

  switchPort = gpioAccess.ports.get(5); // タクトスイッチのポート番号
  await switchPort.export("in");

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

```
for (;;) {
  val = await switchPort.read(); // Port 5の状態を読み込む
  val = val === 0 ? 1 : 0;
  // スイッチは Pull-up なので OFF で 1、LED は OFF で 0 なので反転させる
  ledOnOff(val);
  await sleep(100);
}
};
```

さて、ここまで出来たらスイッチを押してみてください。  
LED が押してる間だけ点灯したら成功です。

ただ、このコードでは今度は**ブラウザ画面上の「LED ON/OFF」ボタンを押したときに正しく点灯しなくなってしまう**ています。スイッチの状態を読んで LED を切り替える処理がポーリング動作しているため、スイッチが押されていないとすぐに LED が消えてしまいます。



# 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); // スwitch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。

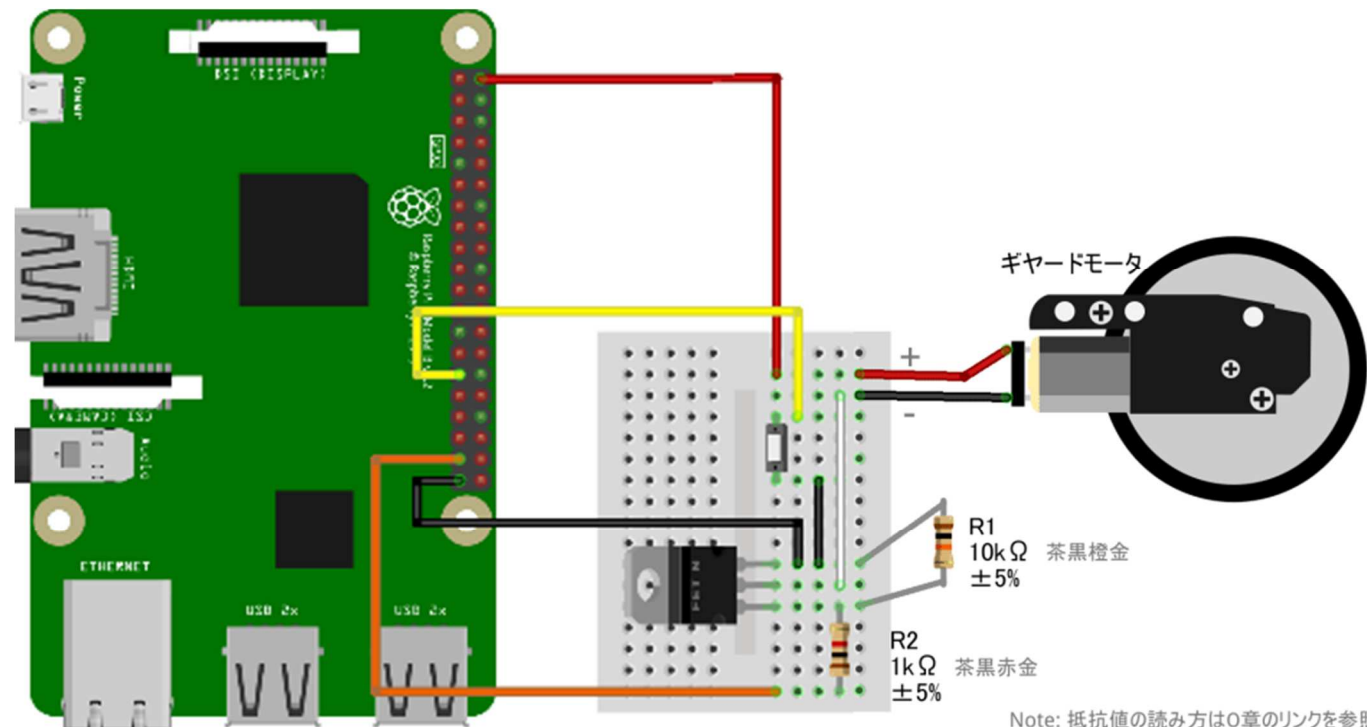
## LED のかわりに ギアモータ（ちびギアモータ）を動かしてみる

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 (1KΩ) x 1
- リード抵抗 (10KΩ) x 1
- ちびギアモータ x 1



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は同じです。



# 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

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

と繋がれば 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。

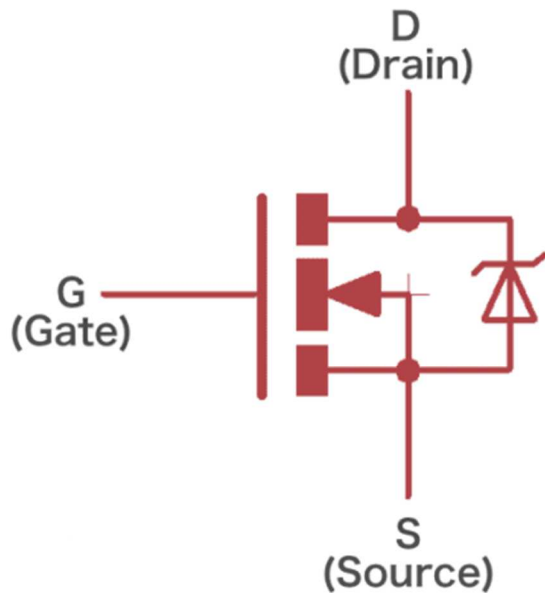


## MOSFET とは

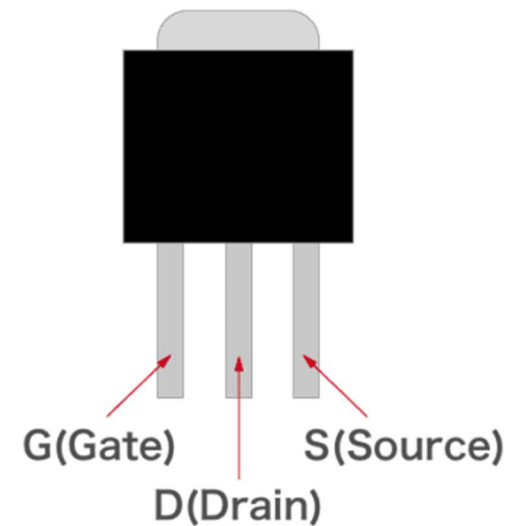
MOSFET とは電界効果トランジスタ (FET) の一種で、主にスイッチング素子として利用される (小さな電圧の変更で大きな電流・電圧のオンオフを切り替える) 部品です。

今回は Nch MOSFET 「**2SK4017**」を利用します。

Nch MOSFETの回路記号



2SK4017のピン配置





# 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読み出し処理の流れ (`navigator.requestGPIOAccess()`～`port.read()`)
- 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) (<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](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](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](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](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](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。



# メモ

---

---

---

---

---

---

---

---

---

---



## 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