1　ワークショップのテーマ

「防犯用LEDライト」を想定した【プロトタイプ】の製作

本日のゴール:【プロトタイプの作成】

WEBブラウザで動作するシミュレーター を使い、シミュレーターの中で実際に利用されている各種センサーを**マイコンボード**に接続し、プログラムの書き込みを行います。

IoT機器がどのように制御されているか？を作業を通じて理解を深めながら、最終的には、**「防犯用LEDライト」**を想定した**【プロトタイプ】**の完成を目指します。

・・・と、その前に、幾つか補足です！

冒頭の「マイコンボード」・・

・・・【マイコン】って、、、、何？

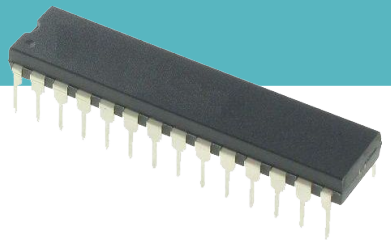
様々な定義が存在

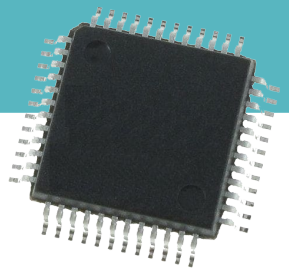
・マイクロコンピュータ（microcomputer）

・マイクロコントローラ（microcontroller）

・マイコンピュータ（mycomputer）・・・

電気製品の構成部材を制御する際に、
スイッチのON・OFFのみの回路で
制御が大変なものを、制御しやすくするための
【半導体チップ】





マイクロコントローラーユニット（Microcontroller Unit）
※略して、「MCU」と呼称したりする物もあります

様々な定義が存在

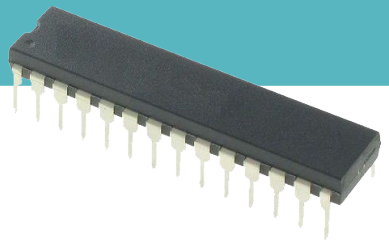
・温度を一定に保つ：エアコン・冷蔵庫

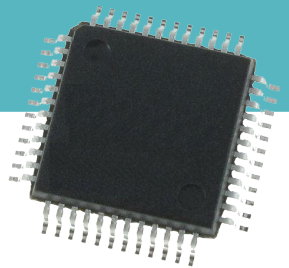
・タイマー機能：炊飯器・テレビ

・家電を操作する：リモコン

マイコンが無くてもつくることは可能、、、、
必要な部品の量が増えて、筐体が大きくなってしまう！

マイコンを利用することで、電気製品を
コンパクト化・小型化
より多機能なものにする！





この「マイコン」を使用して、ガジェットをつくります！

マイコンの開発基板【マイコンボード】について・・

【マイコンボード】実装部材を管理・コントロールする機器

機能検証を、迅速に行う上で欠かせない【中央処理】を行うための機器

プロトタイプ製作を行う上で、**作ろうとしているものが実現可能であるか？**

これを**迅速で且つ確実**な**検証**が求められるケースがあります。

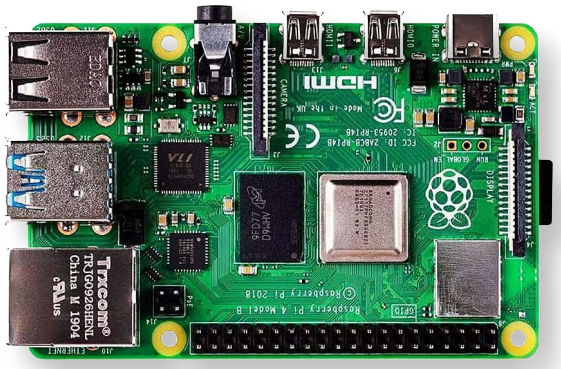
従来であれば、仕様に応じた部材を元に必要最小構成の回路設計を行い、それをベースに概念実証（PoC）を行うところですが、、、

【マイコンボード】実装部材を管理・コントロールする機器

機能検証を、迅速に行う上で欠かせない【中央処理】を行うための機器

近年においては、その工程を速やかに行える**プラットフォーム**が数多く存在しています。その代表的なものの中に**【Arduino】と【Raspberry Pi】**と言う名称の機器があります。

Arduino



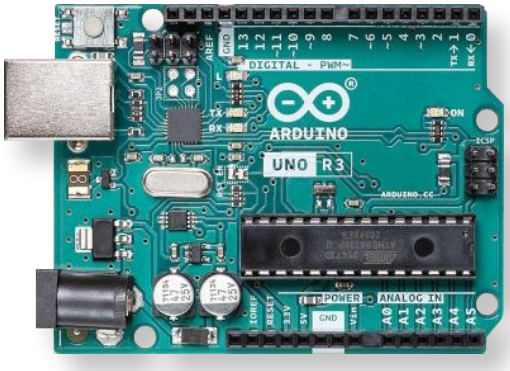
Raspberry PI

Arduino ? Raspberry Pi ?

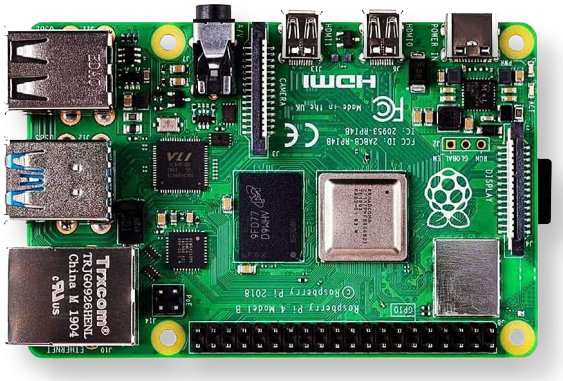
Arduino?　Raspberry Pi?

【Arduino】と【Raspberry Pi】それぞれの特徴

- IoTに携わるとよく耳にする呼称、【 Arduino】と【Raspberry Pi】
  一見同じような形状・大きさなのですが、、、それぞれに大きな特徴があります。



Arduino



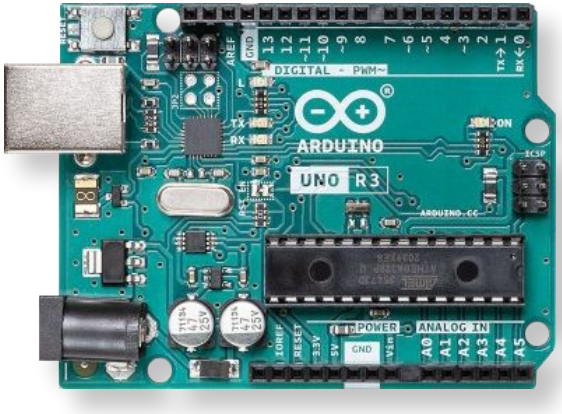
Raspberry PI

Arduino

Arduino

最も手軽でポピュラーな【ワンボードマイコン】

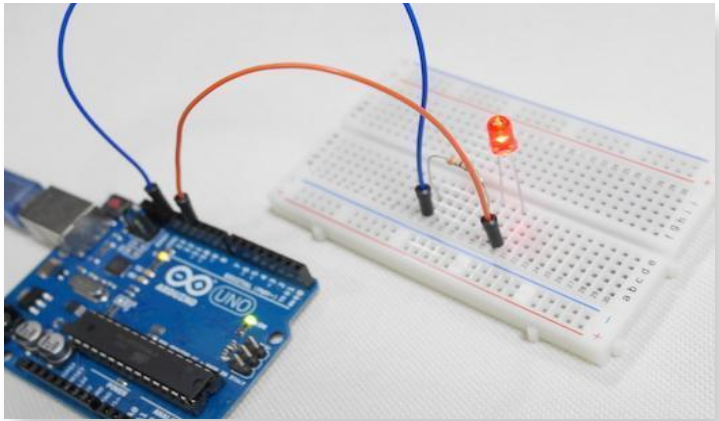
- 元来この**【Arduino】**は、**『学生でも容易に入手が可能』**になることを目的に開発された経緯もあり、**安価で小型且つ汎用性の高い設計**になっています。

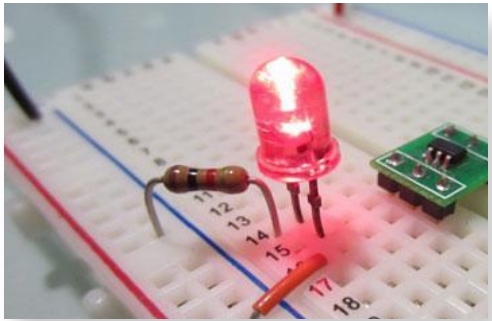


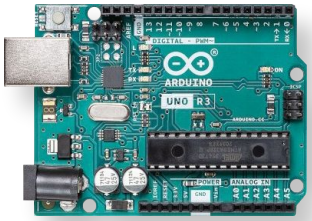
Arduino

「安価」で「シンプル」な　**機能特化型の**制御基盤

- センサーや表示器等の様々な部材の制御を行う上で、必要不可欠な **CPU**（中央処理装置）**・メモリー・ストレージ**(フラッシュROM)等が、**必要最小構成で1つの基盤に実装**されている**最もミニマムな制御基盤**の一つです。

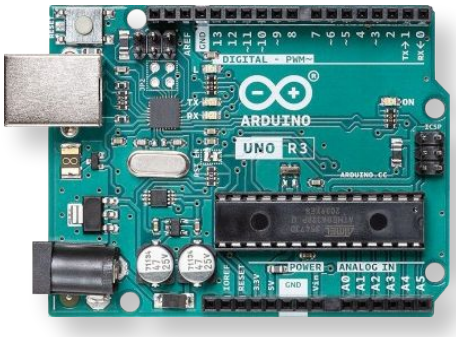




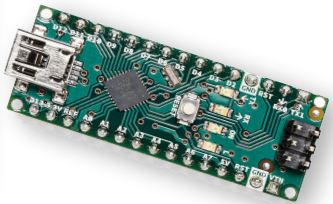


Arduino

- 処理速度や搭載されている機能は、後述の**【Raspberry Pi】には劣る**ものの、**必要に応じた機能**を比較的容易に**【追加・実装】が可能**となっており、**必要な要件課題**の**迅速な具現化**に伴なった**低コスト化や小型化**等、開発時にその実力を発揮する制御基盤です。

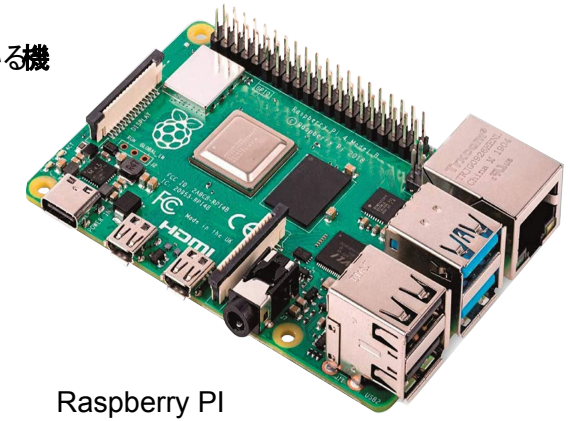


Arduino UNO



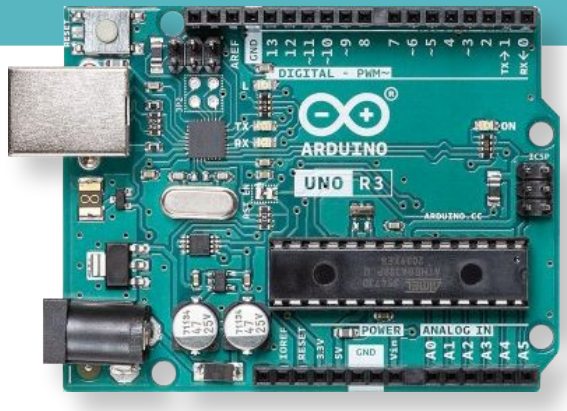
Arduino の小型版
Arduino NANO

処理速度や搭載されている**機能面**では、
【Raspberry PI】に**劣る**

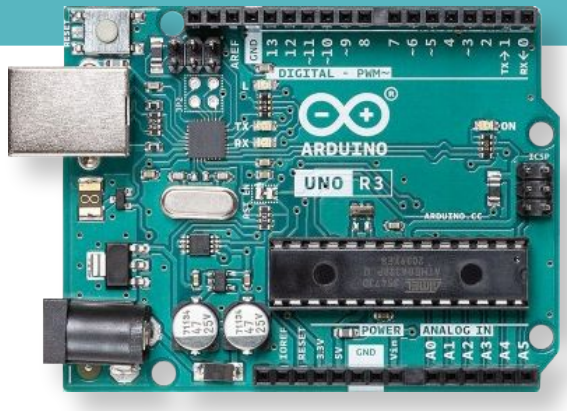


Raspberry PI

Arduinoの大きな特徴#01



オープンソースのハードウェアであること



Arduino

オープンソースのハードウェア

- **マイコンボード（ハードウェア）**と**統合開発環境（ソフトウェア）**で構成される**オープンソース（設計図が公開されていて、だれでも使える）**のハードウェア

- 本体の設計情報が**全てオープンソース**になっており、**個人で自作・改造**することも可能

  オープンソースならではの様々な**独自設計の派生品**が**用途やニーズ**に沿って**数多く存在**します。

Arduino

オープンソース故に・・・

- **利用者数が多い**
  インターネット上や書籍として多くの情報があるため問題解決がしやすい

- **比較的簡単に電子回路が作れる**
  基板などを作成する必要が少ない分コストを抑えることが可能

- **拡張性が高い**
  Arduinoに対応した拡張ユニット（シールド）も多い

※開発時に使用部材の選択の幅を広げやすい

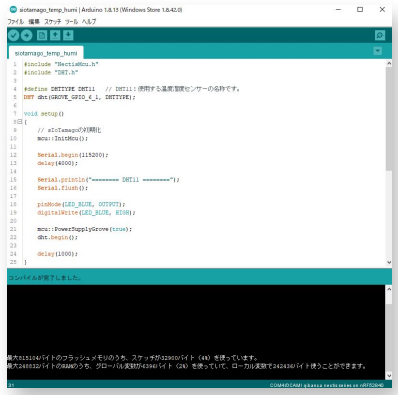
オープンソースのメリット #01：ソフトウェア

Arduino

オープンソースのメリット

➢ ソフトウェアがオープンソースであることによるメリット

統合開発環境（**Arduino IDE**）があり、汎用部材（センサー等）の**制御用サンプルプログラムが多数存在**し、**比較的容易にプログラムを作成**することができる



オープンソースのメリット #02：ハードウェア

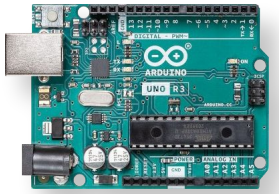
Arduino

オープンソースのメリット

➢ ハードウェアがオープンソースであることによるメリット

豊富な開発用のドキュメントや便利な制御用の拡張ライブラリ（≒プログラム）が数多存在するので、開発環境用のPCがあれば手軽に動作テストを行うことが可能になっている

これと併せArduinoは、公式ボードだけでなく**互換ボードも多数開発**されており、公式ボードより**安価なボード**や、公式ボードにはない**機能が追加されたボード**（高パフォーマンス）まで**多種多様な互換ボード**が存在する



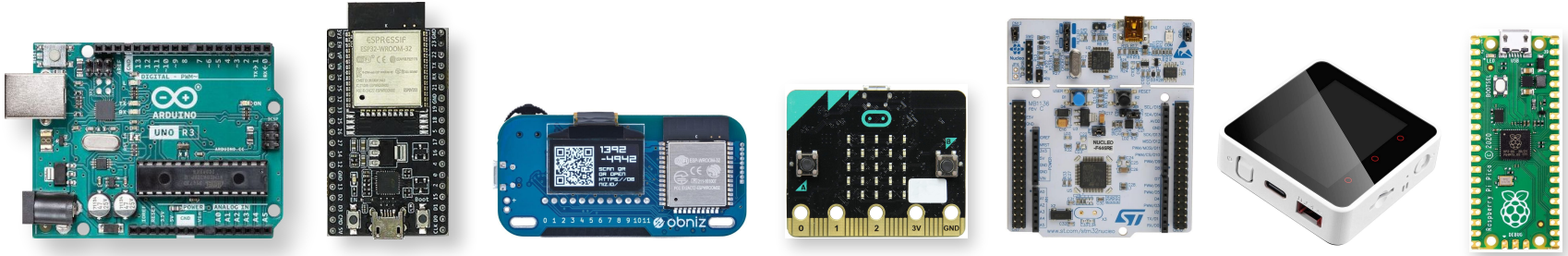
Arduinoの互換・派生ボード

Arduino

主な【ワンボードマイコン】Arduino互換・派生ボード等

Arduinoとはじめとする【ワンボードマイコン】は利用目的に応じた様々な種類の物が存在している
これらもArduino同様、**WINDOWS・MAC・LinuxOS**をはじめとした汎用OS（マウス操作が可能な一般的なOS）からの操作で動作

必要に応じて、**汎用・組み込み用途**の他、**コンピュータ学習・エデュケーション**向けに開発された、小型ディスプレイを搭載したタイプの物も市販されている



閑話休題:統合開発環境？・・・

Arduinoの制御‥制御用PCから書き込む

Arduino

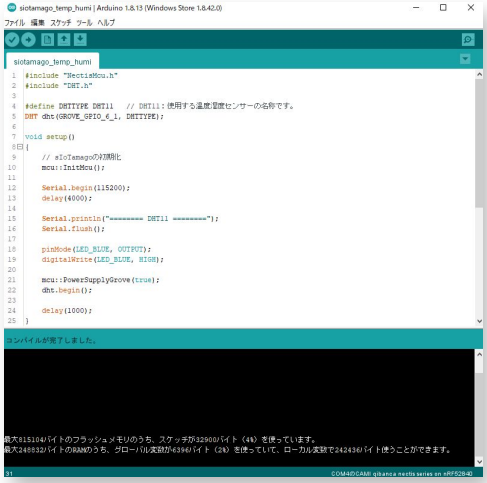
Arduinoの制御方法

➢ コード（プログラム）を【コンパイル】（機械語変換）して、外部端末から直接書き込む

Arduinoをはじめとしたいわゆる**【ワンボードマイコン】**の大きな特徴として、制御用プログラムを **外部の端末**から**書き込む**ことで動作させる仕様

先述した**統合開発環境（Arduino IDE等）**を、開発用端末（一般的なPC等）にインストールし、その中で制御用プログラムを作成を行い、Arduinoへ「書き込む」ことで始めて動作が実現

これは、**一つの処理をひとつずつ** 確実に行うワンボードマイコン特有の仕様と関連するところであり、機器制御プログラムの他にも、**複数のプログラムを同時に処理** を行うことが可能な**【Raspberry Pi】**との大きな違いでもあります。



※開発用端末側の汎用 OSは、WINDOWS・MAC・LINUXであれば開発環境の構築が可能です。

つまり、Arduinoは・・・

Arduino

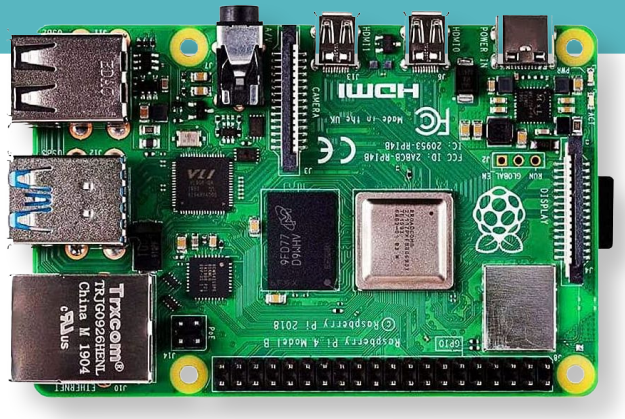
機械制御を行う上での「最前線」で、作業を行う【現場管理者】的なハード

先述の通り、プログラムを**【外部端末】**から書き込みをしなければならなかったり、**処理速度や搭載されている機能**は**【Raspberry Pi】に劣る**ものの、必要最小構成であるが故の**迅速且つフレキシブルな適応性**こそが【Arduino】をはじめとした【ワンボードマイコン】の最大の特徴

言うなれば、機械制御を行う上での現場**「最前線」**で、作業を行う**【現場監督者】**的なハードウェアとなります。

続いて・・・

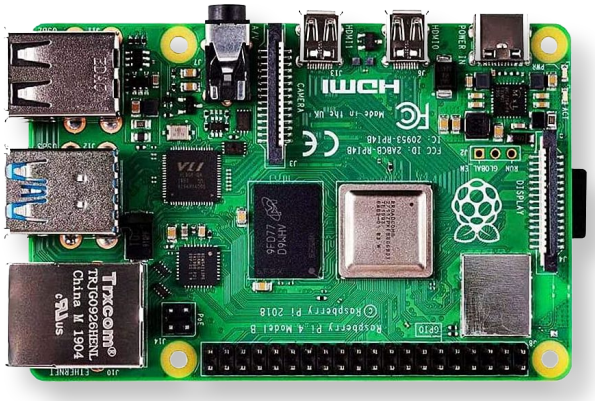
Raspberry Pi



Raspberry Pi

本体にOSがインストール可能な【シングルボードコンピュータ】

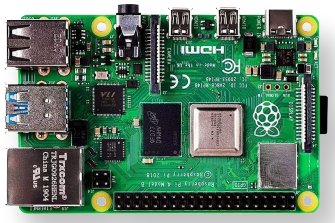
- 基盤サイズこそ、標準サイズのArduino（UNO）とほぼ同じですが、基板上に搭載されたSoc（CPUをはじめとした動作に必要な機能があらかじめ組み込まれた集積回路チップ）には、高性能なCPUやRAM(メインメモリ)が搭載された制御基盤で、その処理性能はArduinoよりも高速で複数の処理を同時に行う「マルチタスク処理」が可能となっています



Raspberry Pi

モニター表示に必要な外部インターフェイスを標準装備

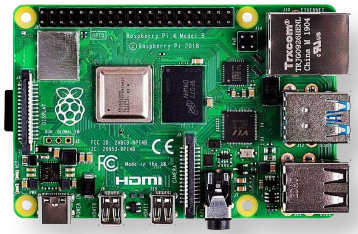
- さらに本体には、**USB**やモニター接続用の**HDMI端子**と言った**外部接続用**の各種インターフェイスがあらかじめ搭載しており、**有線LAN、Wi-Fi、Bluetooth**等の**通信モジュールも実装**されている
- **microsdカード**を記録媒体とし、**Linuxをはじめとした様々な汎用OS**を**本体に直接インストール**して、接続したセンサーやその他の機器・部材を、制御用のアプリケーションを用いて、マウス操作を伴なった**直接制御**も可能になっているのが最大の特徴です。

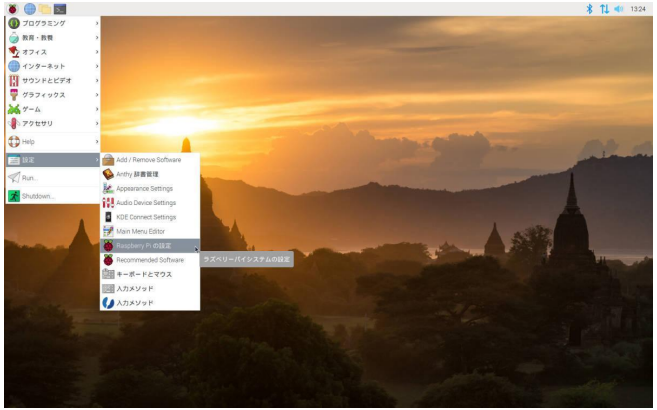


Raspberry Pi

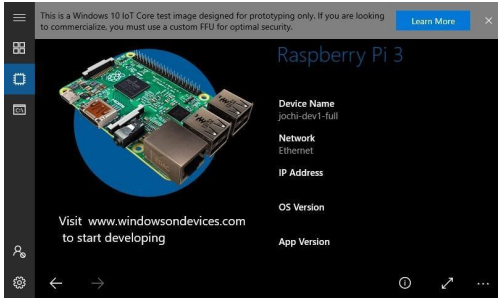
本体に取り付けたセンサー等を**リアルタイムに直接制御**が可能

- **【シングルボードコンピュータ】**と言う俗称で呼ばれたりもするこの**【Raspberry Pi】**ですが、先述の特徴を活かし、**WINDOWS**や**MAC**などの**一般的なOS**同様、各種アプリケーションを**Raspberry Pi用のOS**にインストールし、Raspberry Pi本体に取り付けたセンサー等を**リアルタイムに直接制御**や、集計した**データの可視化を単独で行う**ことが可能になっております。





Linuxベースで動作している Raspberry Pi OS



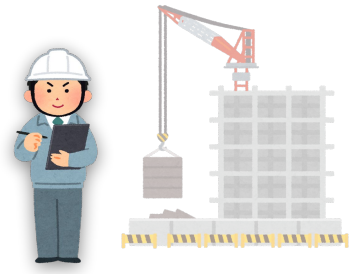
IoT用WindowsのWindows 10 IoT Core

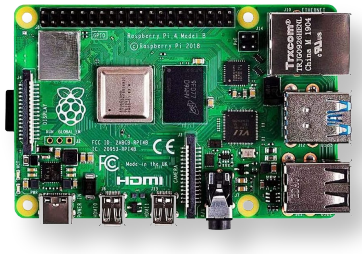
つまり、Raspberry Pi は・・・

Raspberry Pi

センサーや電子部材を直接制御が可能な端末

- 先述の特徴から、**【シングルボードコンピュータ】≒超小型PC**と言っても過言では無く、高速なマルチタスク処理を得意とする特性を生かし、様々なセンシングデバイスの直接監視、或いはデバイス管理サーバー用途や、集めたデータの集計・演算処理・可視化用途などで利用されています
  言うなれば、**「最前線」**で、作業を行う**【現場管理者】**も行え且つ、**【現場管理者を統括】**する**【司令塔】**の役割も果たせる制御基盤であると言えます





【シングルボードコンピュータ】の補足

Raspberry Pi

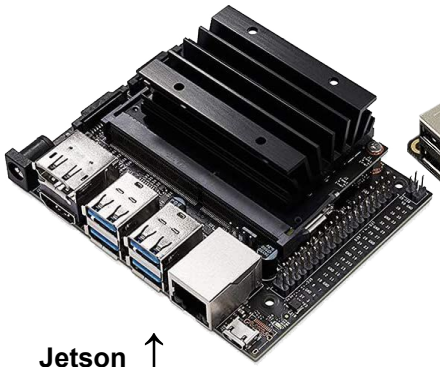
OSが搭載可能な【シングルボードコンピュータ】についての補足

【ワンボードマイコン】同様、**【シングルボードコンピュータ】**も利用目的に応じた様々な種類の物があります。

中には、機械学習等の演算処理用途に特化したものなども存在します。

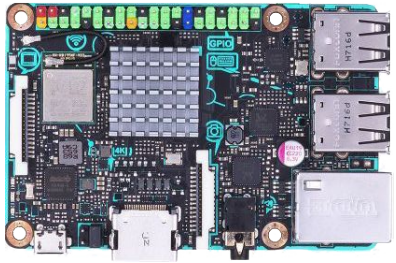
シングルボードコンピュータに該当する主な製品

- Raspberry Pi（汎用）
- Jetson（演算系・機械学習用途）
- LattePanda
- Tinker Board
- BeagleBone　他・・



Jetson

LattePanda



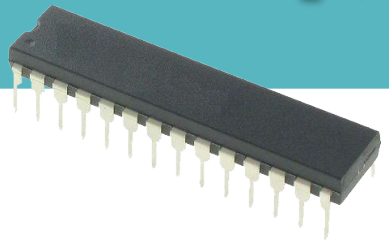
Tinker Board

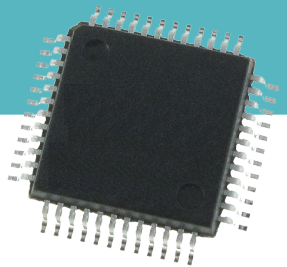
PC端末を購入するほどでもない用途・・・

※価格とランニングコストがPCよりも安い！

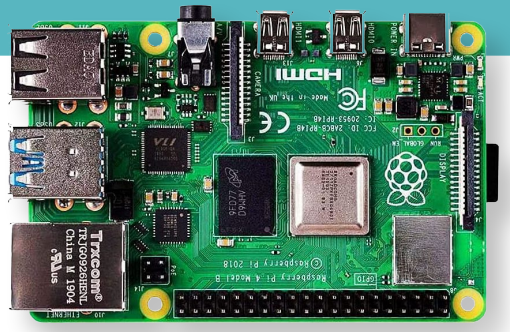
ここまでのまとめ

電気製品の構成部材を効率よく制御する
【マイコン】

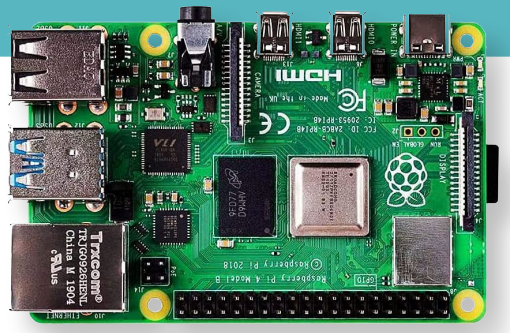




実装部材を管理・コントロールする
【マイコンボード】と【シングルボードコンピュータ】



外部端末（PC等）から制御するマイコンボード【Arduino】
OSを搭載、直接制御が可能なコンピュータ【Raspberry Pi】

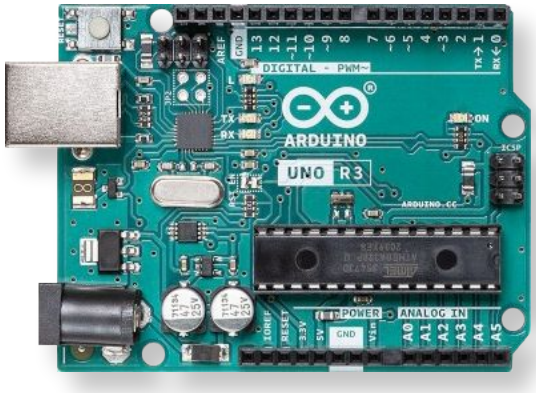


Arduino

【Arduino】を使用する判断規準として、、、

- **1～2つの単純な処理を行う・・・例えば、、**
  - 1種類のセンサーを繋げて、対象物を計測し続ける
  - センサー計測値の条件に応じて、接続しているLEDを点灯・点滅させる

・・・等が挙げられます

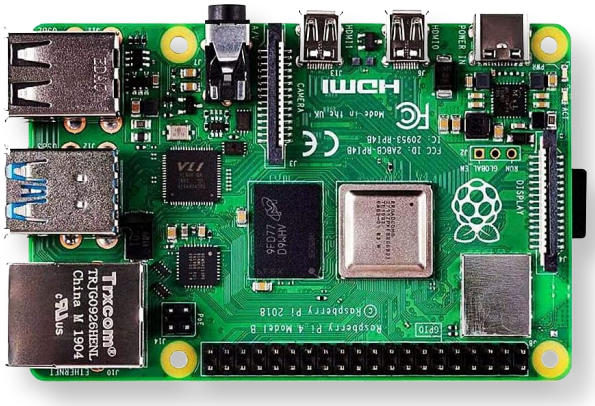


Raspberry Pi

【Raspberry Pi】を使用する判断規準として、、、

- **複数のタスク処理を行うケース・・・例えば、、**
  - 計測ログをデータベースに記録しながら、接続した複数のセンサーの計測
  - 複数のマイコンボード（Arduino等）で計測された数値を集計し、リアルタイムで計測値をモニターに表示

・・・等が挙げられます

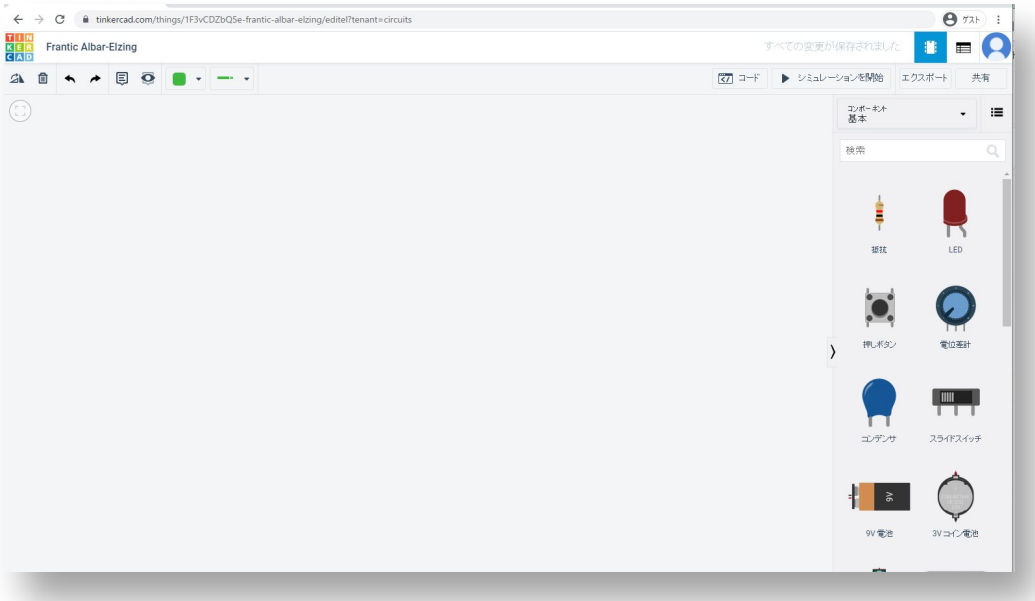


マイコンに関する前説はここまでです！

ワークショップの開発環境について

今回のワークショップでは、マイコン基盤や電子部品の実物を用いての作業はございません。

その代替として、WEBアプリケーション**「TinkerCad」**を使用して、作業を行います。



←WEBアプリケーション

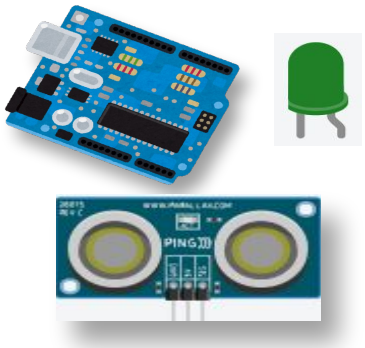
「TinkerCad」の作業画面

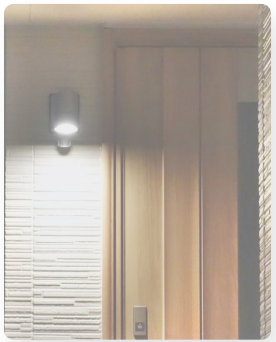
ワークショップの開発環境について

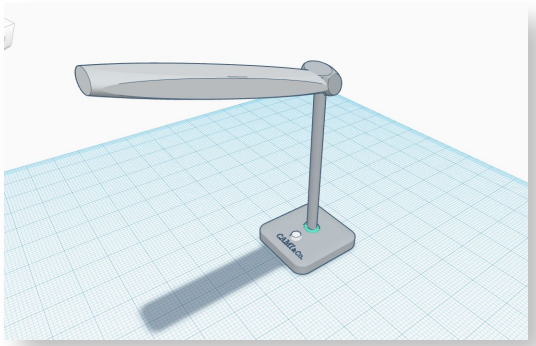
当該アプリケーションでも、実際のプロトタイプ製作で利用されている各部材がございますので、それらを用いて、作業を進めてまいります。

マイコン基盤を実際に動かしてみることから始まり、センサーをはじめとした各種部材の特徴や実際の動作を、検証作業を通して段階的に体験していただきます。

最終的には、本日の目標である**「防犯用LEDライト」**を想定した**プロトタイプ製作（≒原理試作）**完成を目指します。







- 元々は、3Dモデリングツールなので、**プロトタイプ**の外形デザインも可能です！

プロトタイプ・・

アイディアやひらめきが実現可能であるか？
実証し、広く証明するための手段 ≒ 原理試作

例えば今回は、

【防犯用LEDライト】です

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・

LED

何らかの状態（環境）変化を **知らせる**ための部材

人感（PIR）センサー

何らかの状態（環境）変化を **感知する**ための部材

これらの構成を前提に、、、、

「**どのような動きをさせたら良いのか？** 」「**想定した通りに動くのか？** 」 を【**実験**】【**実証**】する

転じて、

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・

光

何らかの状態（環境）変化を **知らせる**ための部材

知らせる手段

体温（対象物の温度）

何らかの状態（環境）変化を **感知する**ための部材

感知する手段

対象エリアの【**温度**】が変わったら、【**光**】を使って知らせることで【**防犯**】としての機能が成り立つと言うことになります。

もう少し細かく見てみると、、、

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・

光

何らかの状態（環境）変化を 知らせるための部材

知らせる手段

体温（対象物の温度）

何らかの状態（環境）変化を 感知するための部材

感知する手段

「どのような動きをさせたら良いのか？ 」・・・

「どこ」の【温度】を「どのように検知」させて、、、

「どんな（色・種類）【光】」を、「どのように」光らせ知らせるか？・・・

それは【防犯用】として機能するのか？を【 実験】【実証】によって確かめる。。。

パーツ（部材）構成を考えていたのに・・

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・

★人感（PIR）センサーを「玄関口」のエリアを検知出来るようにして、

「人」或いは「動物」が検知アリアに「入って」きたら・・・

★「3色」LEDを、「赤色」で「3秒間点滅させる」・・・

それは【防犯用】として機能するのか？を【実験】【実証】によって確かめる。。。

あるいは、

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・

音

何らかの状態（環境）変化を **知らせる**ための部材

知らせる手段

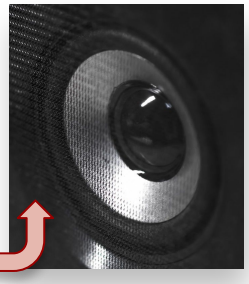
音（対象物の存在）

何らかの状態（環境）変化を **感知する**ための部材

感知する手段

対象エリアの【**音の状態**】が変わったら、【**音**】を使って知らせることで【**防犯**】としての機能が成り立つと考えた場合・・・

【防犯用】LEDライトとして機能するための必要なパーツ（部材）構成を考えた時・・・



スピーカー

何らかの状態（環境）変化を **知らせる**ための部材

超音波距離センサー

何らかの状態（環境）変化を **感知する**ための部材

プロトタイプの段階でも、完成メージが大きく変わる

それでは、いよいよ作業開始です！

今回の実践ではArduinoを使用します！

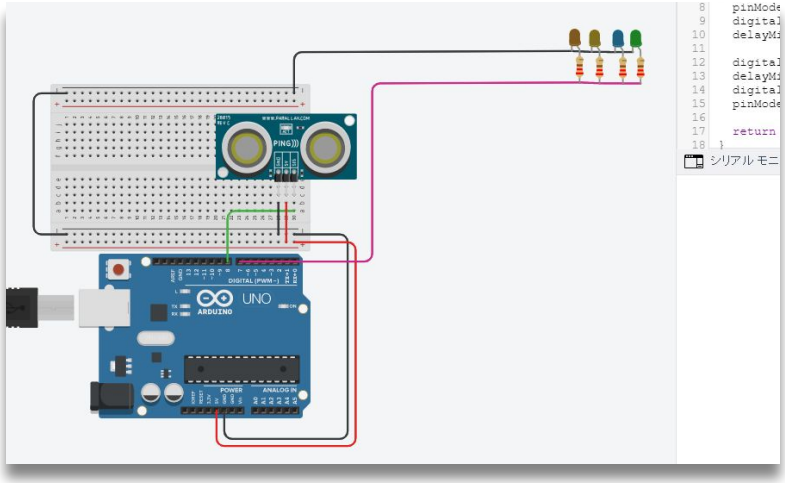
2　実践の流れと使用ツールについて

作業の流れと、使用するWEBアプリの補足

実践の流れについて

ワークショップの流れについて

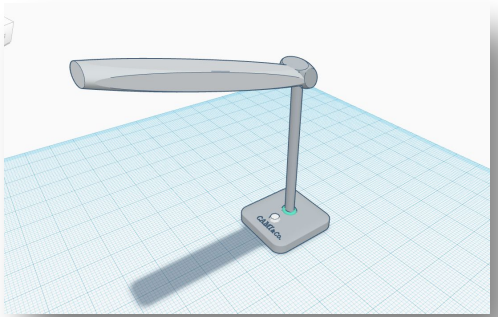
- **WEBアプリについて**
  - → WEBアプリ「**tinker cad**」についての紹介　https://www.tinkercad.com/
- **使用する部材について**
  - → 使用する部材の種類、マイコン・センサー・その他部材についてを紹介
- **IoT実践 ：各種部材の制御**
  - → 各種センサーやパーツ毎で解説を踏まえてプログラムの作成を行う
- **プロトタイプ実践 ：プロトタイプの制作**
  - → IoT実践で行った作業を踏まえ、**防犯用LEDライト**を想定した試作を行う

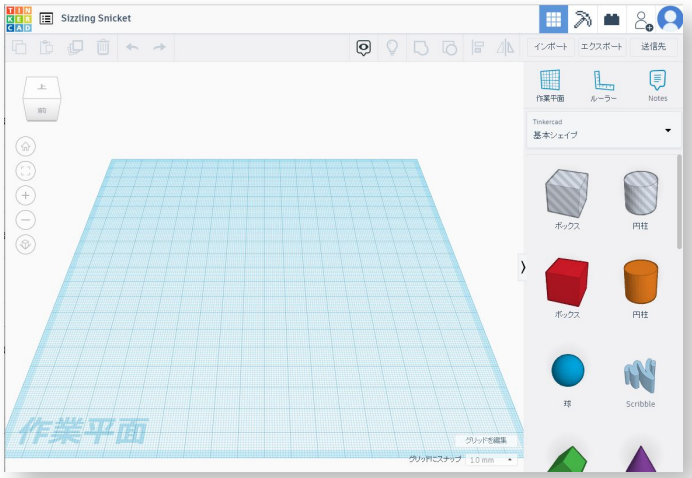


WEBアプリについて　～　実際に作る物　～

WEBアプリ「tinker cad」についての紹介　https://www.tinkercad.com/

Tinkercadは、**オートデスク社**よりリリースされている**ウェブベースの3DCADアプリケーション**で、**メールアドレスを登録**することで誰でも**無料で利用**することが出来ます。





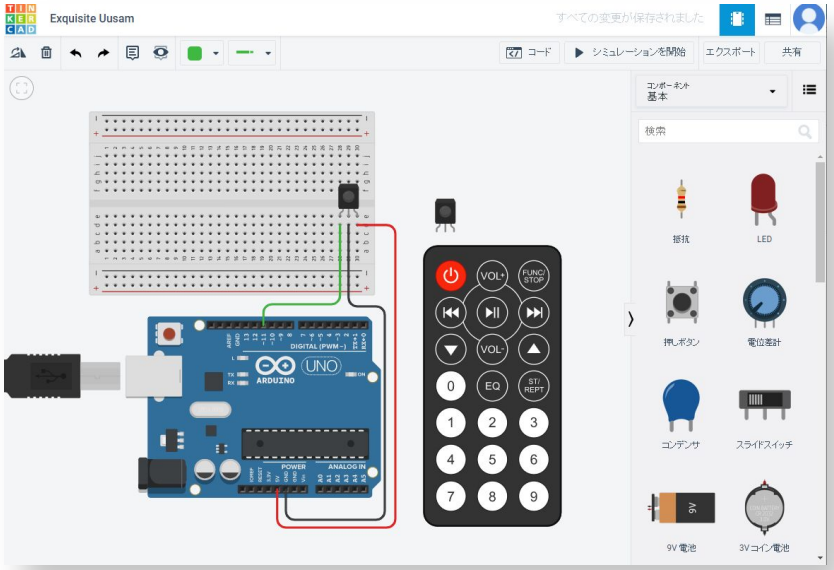
※オートデスク社は工業向け機械設計用CADツール「Fusion360」等を提供しているアプリ開発ベンダーです。
https://www.autodesk.co.jp/

WEBアプリについて　～　実際に作る物　～

WEBアプリ「tinker cad」についての紹介　https://www.tinkercad.com/

本来は**3Dモデルを作成**するための**WEBアプリケーション**なのですが、実装されている機能に**電子回路の試作**を行うことが出来る**エディター**があります。

今回の実践では、この電子回路エディターを利用し、各種部材の動作を確認しながらプロトタイプの作成を行います。



※オートデスク社は工業向け機械設計用CADツール「Fusion360」等を提供しているアプリ開発ベンダーです。
https://www.autodesk.co.jp/

準備が整ったので･･･

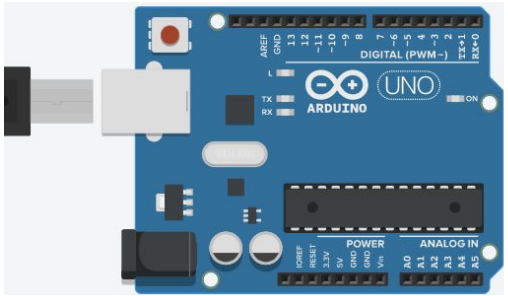
こんどこそ、作業開始です！

コーディング：「Hello World」の表示

tinker cadの「シリアルモニタ」上に「Hello World」を表示させる！

コーディング：「Hello World」の表示

tinker cad上で【Arduino】を動かしてみましょう：「 Hello world」

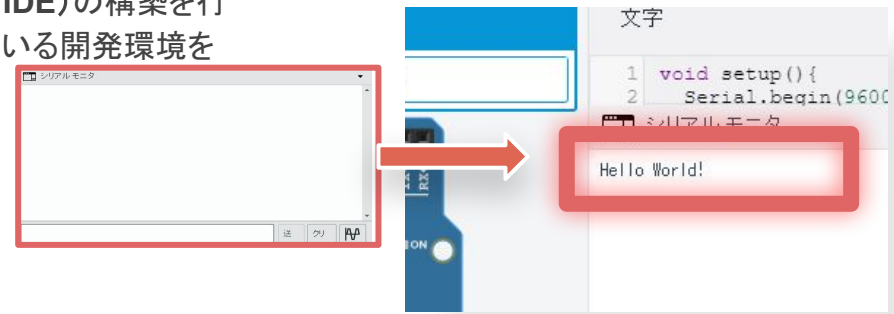


まずはIoT制御の第一歩として、Arduino本体が**【指示通りに動作】**を行っているか？を確認するためにArduino本体から・・・

【Hello world】の文字列を指定した場所へ表示させるプログラムを作成し、どのように動作しているかを、実際の動作を通して確認したいと思います。

本来であれば、Arduino開発用のPC端末へ**開発環境（Arduino IDE）**の構築を行わなければならないのですが、ここでは**TinkerCad**に実装されている開発環境を利用して作業を進めます。

この項目のゴールは、ズバリ。。。



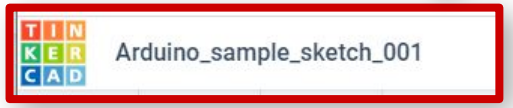
仮想シリアルモニタ上に【 Hello world】を表示させる
・・・です！

それでは、開発環境であるTinkercadの画面を開きます

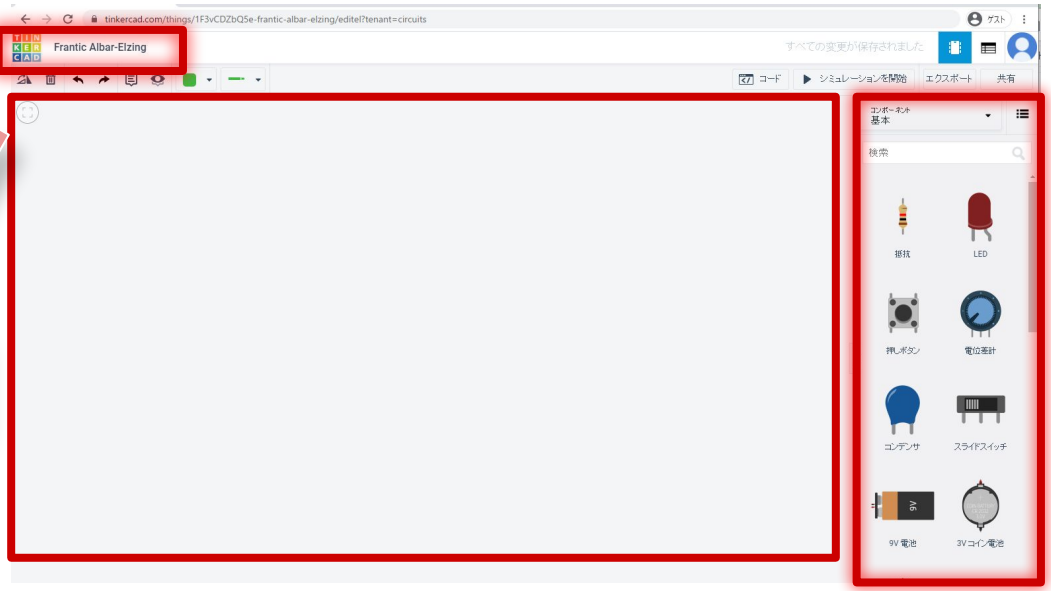
コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 右図が回路と回路を制御するコード作成用のメイン画面です。
- 画面左側の**フィールド**に画面右側の赤枠で囲まれている**コンポーネント（部材）リスト**から、必要なパーツを選択し、配置しながら回路を作成する流れになっています。



作成する回路の名前は、**画面左上の赤枠で囲まれたタイトル文字列をクリック**することで、**任意に変更することが出来ます**ので、まずは回路名を決めて記述しましょう。

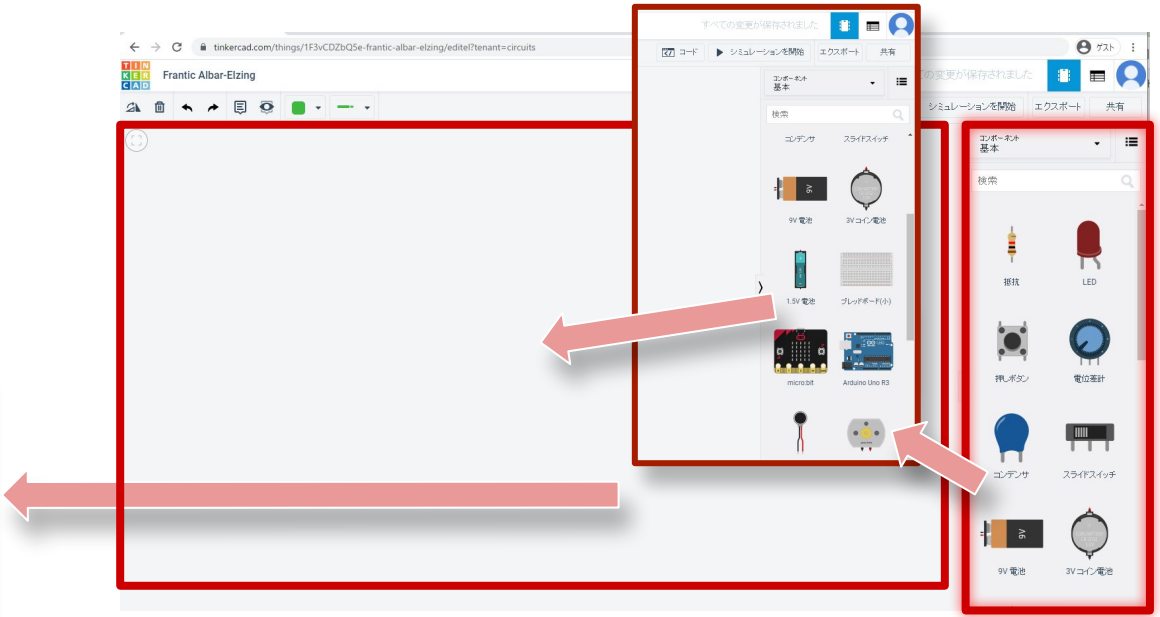


コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

それではいよいよコーディングに入りたいと思います！

1. **画面右側**にある**コンポーネント一覧** から下へスクロールして、Arduino本体を選択してみましょう。
2. 選択後、任意の場所へは**ドラッグ &ドロップ**で移動が可能です。



コーディング①:「Hello World」の表示

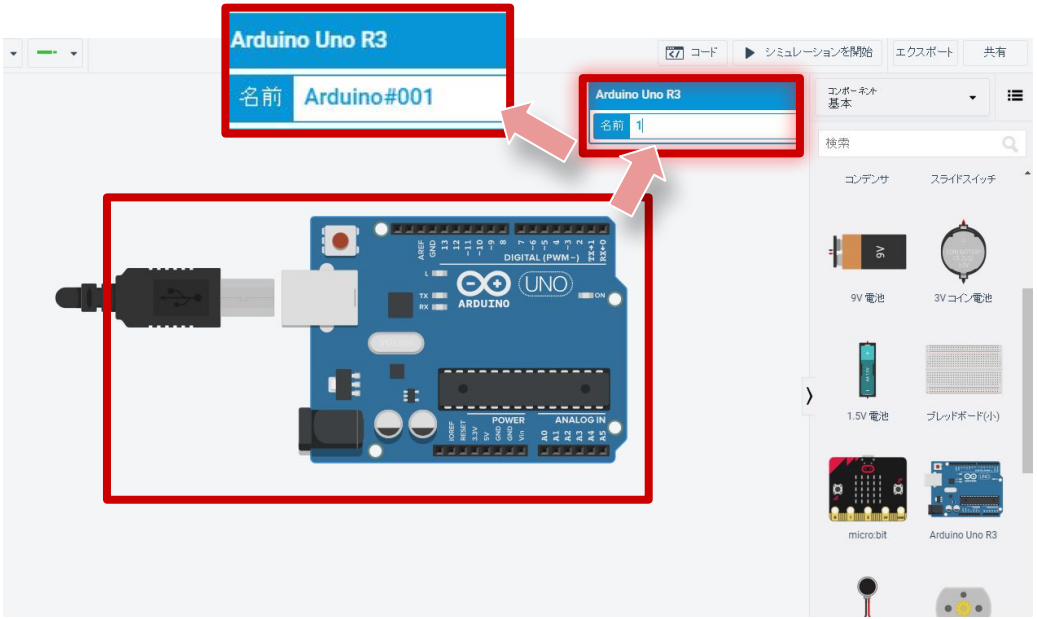
～ プロトタイプ制作　実践:前半 ～

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 回路画面に追加したArduinoを選択すると、選択した部材に関する情報が表示されます。

  Arduinoの場合は、【名前】の項目が表示されますので、任意の名称に変更することが可能です。

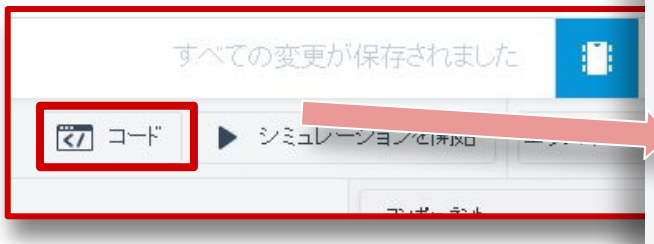
- 一台のArduinoを管理するだけなら、名称変更を行わなくても問題なく作業を行えますが、複数のArduinoを利用する場合に、任意の名称を設定することでデバイスの管理を円滑に行うことが出来ます。



コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- Arduinoの配置も出来たところで、いよいよコーディング作業に移りたいと思います。
- 画面右上にある【コード】ボタンを押下すると、右隅からコードブロック画面がスライド表示されます。

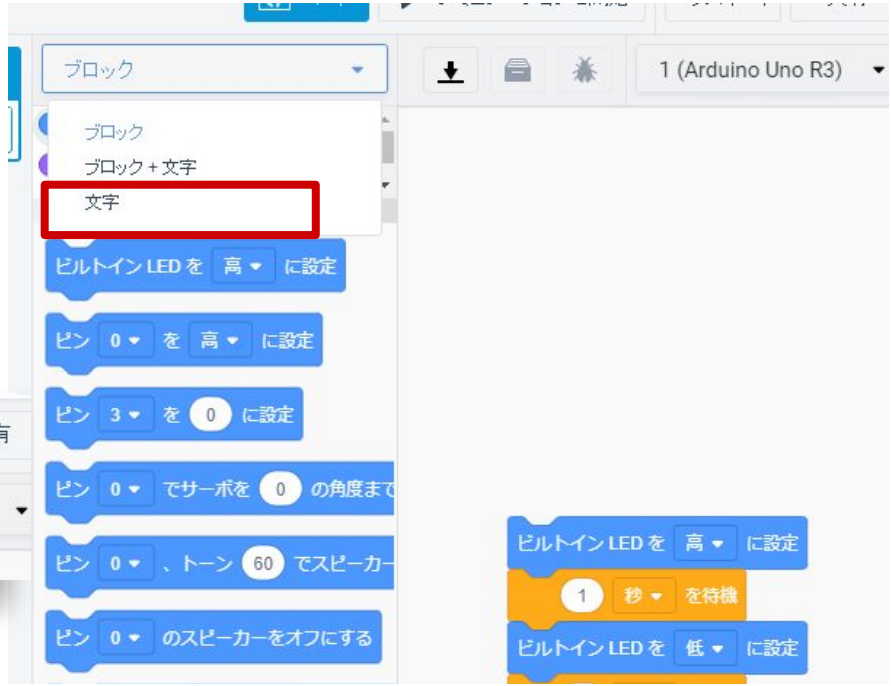


- 右図の赤枠（太枠）で囲まれた箇所が、実際にプログラムを記述するエリアになっています。

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- コードブロックは、米国の MITで教育用途に開発された、プログラム経験がなくても視覚的にプログラムが作成出来る、大変優れた開発ツールになっていますが、ここではテキストベースでのプログラム作成を行います。
- コード編集スペースの左上にある【ブロック】ボタンを押下し、【文字】を選択します。



※【文字】選択時に警告画面が出る場合でも、そのまま【続行】を押下して画面を切り替えてください。

コーディング①:「Hello World」の表示

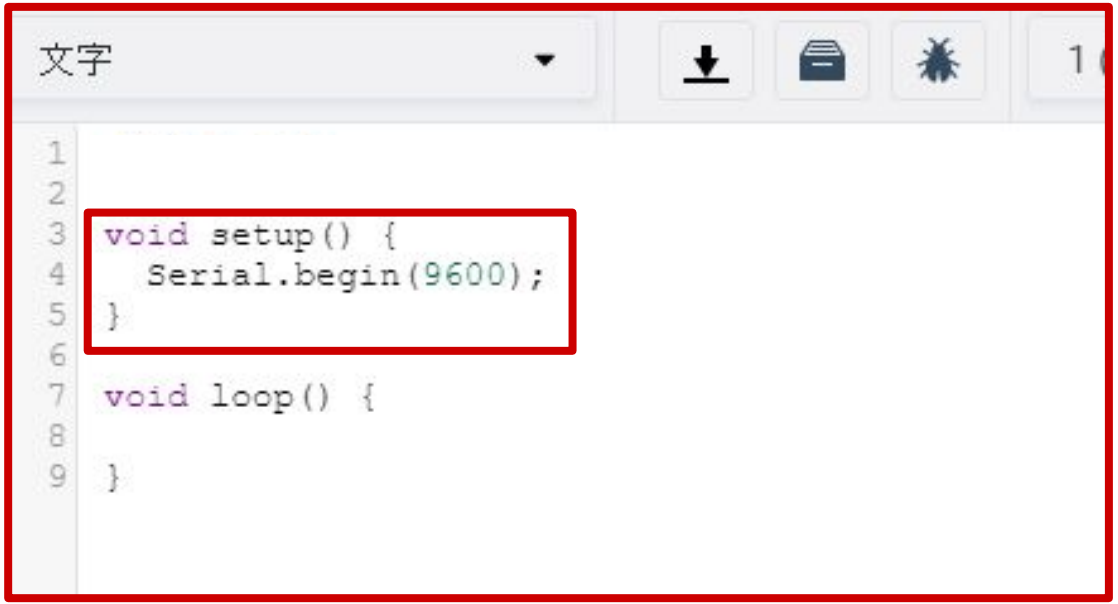
仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 赤枠で囲まれているエリア

  void setup() {　　}

- のカッコ内【{　}】に記述のテキストの補足です。
- **Serial.begin(9600)**
- これはArduinoの状態を検証するための【シリアル通信】を開始させる「初期設定」になります。

※シリアル通信をArduinoに開始させることによって、【シリアル画面】に文字列が表示されるようになります。



コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 次に赤枠で記述されている箇所

  void loop() { }

- のカッコ内【{ }】に記述のテキストの補足です。
- **Serial.println("Hello World！”);**
- 薄々お気づきの方もいるかもしれませんが、これは先程初期設定として記述した【シリアル通信】で画面に「**Hello World！**」の文字列を表示させる構文となります。

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 続いて先程記述した行の次の行に（赤枠の部分です。）記述された構文です。
- **delay(2000);**
- これは、文字列表示後に 2秒待機する構文になります。
- 単位はミリ秒　上記の場合は 2000ミリ秒＝2秒となります。

※最後にプログラム全体を通しての補足です。

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- まずは冒頭の**「setup()」**関数です。
- この**【setup()】**関数は、スケッチ（Arduinoではコードのことを**「スケッチ」**と呼称しています）の**実行開始時**、、、つまり、Arduino本体の**電源を入れた直後**、或いは**リセットした直後**に**一度だけ**読み込みます。
- スケッチで使用する**変数の初期化**や、接続する**デバイスの配線**（ピンモードと呼んだりします。）**の設定**、利用するライブラリ（便利な補助プログラム）の設定（読み込み）などを行うのに利用します。

```
1
2  //
3  void setup() {
4    Serial.begin(9600);
5    //シリアル通信の有効化と通信レート（≒速度）を9600に設定
6  }
7
8  void loop() {
9    Serial.println("Hello World!");
10   //シリアルコンソール（画面）に文字列「Hello World！」を出力
11
12   delay(2000);
13   // 2秒待機（単位はミリ秒　上記の場合は2000ミリ秒＝2秒となります。）
14 }
```

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 次の「**loop()**」関数ですがsetup()関数で呼び出された後に、**実際に動かす命令**（メインルーチン）を繰り返し呼び出すために使用します。
- Arduino上では、先程のsetup()関数と併せ必ず記述が必要になります。
- Arduinoはこの【**loop()**】関数内の処理を繰り返し続けます。

今回のコードでは、以下のルーチンが動作します。

1) **コンソールに【Hello World！】を改行付きで表示**

2) **2秒待機**

```
4    Serial.begin(9600);
5    //シリアル通信の有効化と通信レート（≒速度）を9600に設定
6  }
7
8  void loop() {
9    Serial.println("Hello World!");
10   //シリアルコンソール（画面）に文字列「Hello World！」を出力
11
12   delay(2000);
13   // 2秒待機（単位はミリ秒　上記の場合は2000ミリ秒＝2秒となります。）
14 }
```

以上でコーディング①:「 Hello World」の表示は完了です！
次は、部材を使用した動作検証を行います

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う
コピペ用サンプルコード （補足付き）

```
void setup() {
  Serial.begin(9600);
  //シリアル通信の有効化と通信レート（≒速度）を 9600に設定
}

void loop() {
  Serial.println("Hello World!");
  //シリアルコンソール（画面）に文字列「 Hello World ！ 」を出力

  delay(2000);
  // 2秒待機（単位はミリ秒　上記の場合は 2000ミリ秒＝2秒となります。）
}
```

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う
コピペ用サンプルコード（文字化け対策用）

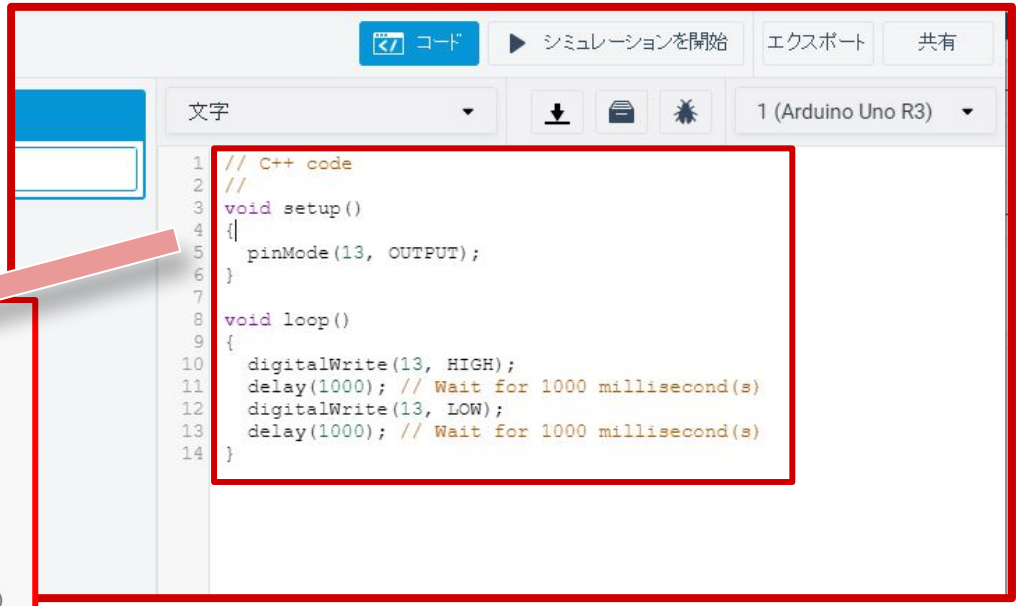
```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println("Hello World!");
  delay(2000);
}
```

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- テキストエディター画面に切り替わると、先程表示されていたブロックエディタのコードがテキストコードとして表示されます。

- 図の赤枠で囲まれている箇所を全て削除して、前ページのテキストをコピー＆ペーストしてください。



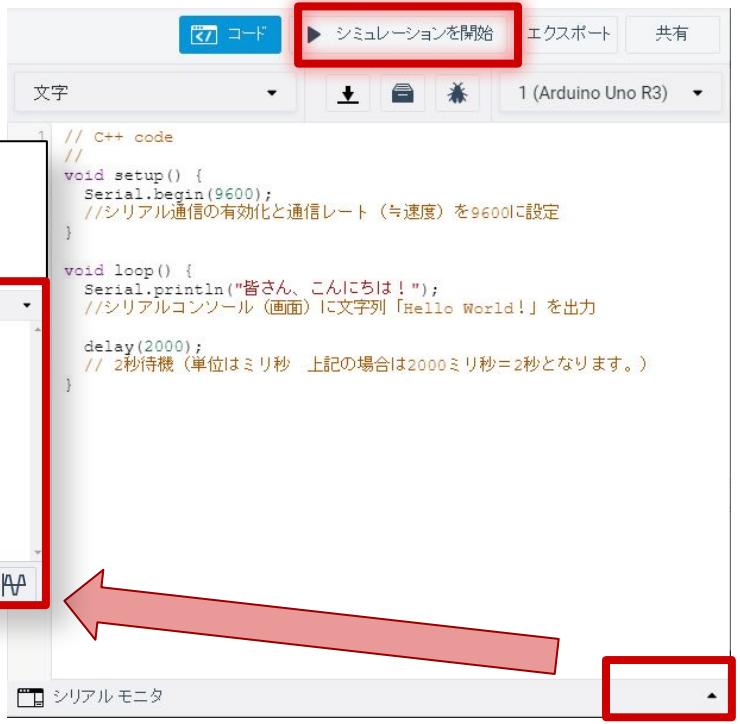

```
void setup(){
  Serial.begin(9600);
  //シリアル通信の有効化と通信レート（≒速度）を9600に設定
}
void loop(){
  Serial.println("Hello World!");
  //シリアルコンソール（画面）に文字列「Hello World！」を出力

  delay(2000);
  // 2秒待機（単位はミリ秒　上記の場合は2000ミリ秒＝2秒となります。）
}
```

コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

まずは、記述したプログラムを起動してみたいと思います。

1. エディター最下部（右下の赤枠部分）【▲】を押下すると、右図のシリアルモニタが下からスライドアップして出てきます。
2. 次にエディター上部（赤枠部分）にある【シミュレーションを開始】を押下すると、作成したコードが起動します。



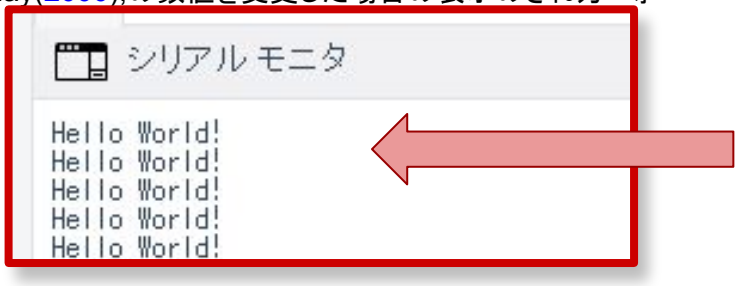
先程表示したシリアルモニタにご注目ください！

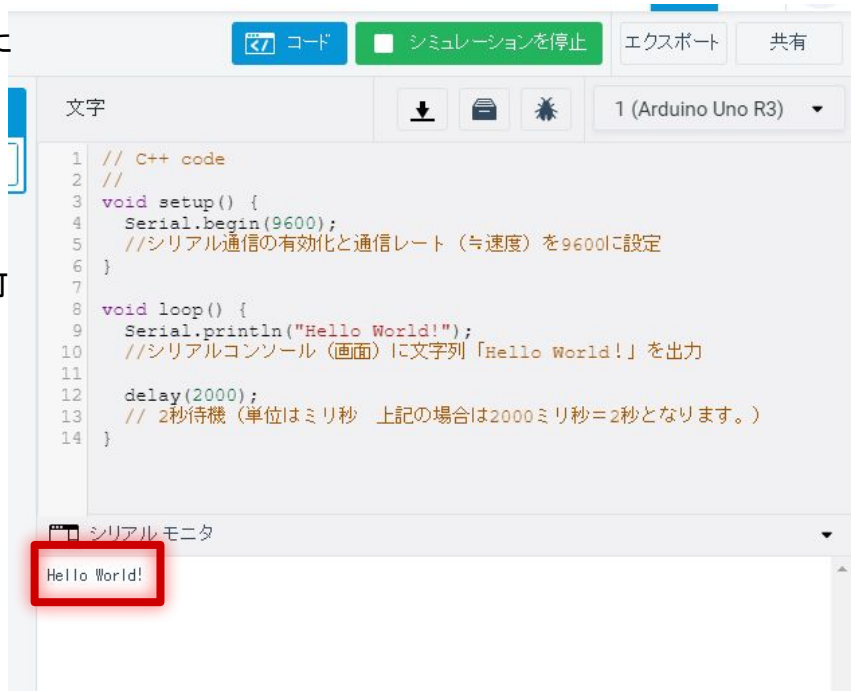
コーディング①:「Hello World」の表示

仮想のシリアルコンソール（制御画面）に『Hello World」を表示させるプログラムの作成を行う

- 画面の通り、シリアルモニタ上に【**Hello World！**】の表示が2秒毎に表示されれば成功です！
- 早めに終わった方は、記述したコードの記述を変更してみて動きを検証してみてください。

1. 例：表示内容を「**Hello World！**」から別の表記・日本語表記は可能か？　等・・・
2. delay(**2000**);の数値を変更した場合の表示のされ方　等・・・





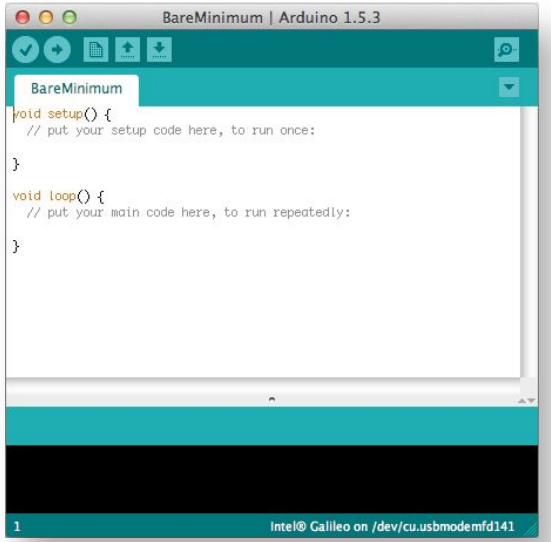
※次ページ以降では、プログラムの各記述内容の解説を行っていますので、動作検証と併せ確認してみてください。

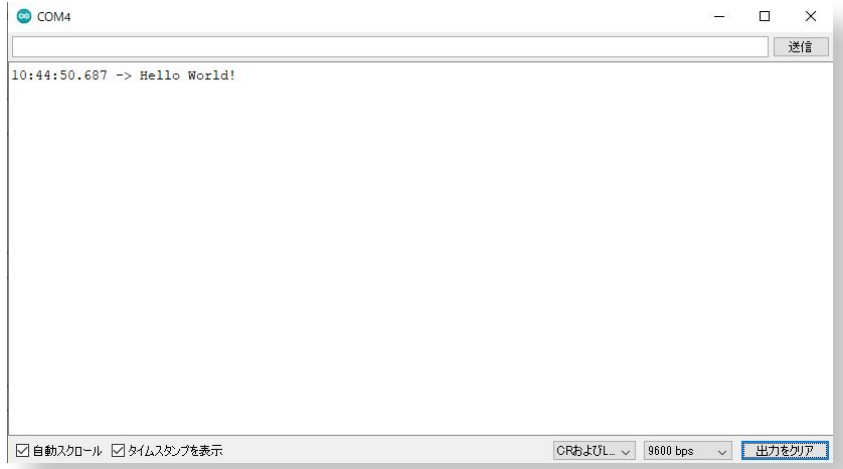
コーディング：「Hello World」の表示

実際の開発ツールとシリアルコンソール（制御画面）のサンプル

- コーディング：「Hello World」の表示に関する作業は終了です！
- 下図は、実際のArduinoの開発環境「Arduino IDE」（図左）と「シリアルコンソール」（図右）の画面になります。

※実際に使用されているArduinoの開発環境【Arduino IDE】の画面（図左）と「シリアルコンソール」（図右）の画面。





・・・ちなみに、、

- Arduinoスケッチの作成に限らず、プログラムを作成する際には、**ワードプロセッサアプリ（Wordなど）**ではなく、**テキストエディタアプリ（メモ帳など）**を使用します。

- ワードプロセッサアプリを使用した場合、プログラムそのものだけでなく **レイアウトなどのデータも含まれてしまい**、エラーの発生を招いてしまいます。

Arduino言語の概要

Arduino IDEでスケッチを作成するには、C言語風の「Arduino言語」を使用します。

- Arduino言語の土台になっているのは、**C言語とC++言語**
- **C言語のすべての構造と、C++言語のいくつかの機能**に対応

★Arduino言語とC言語・C++言語の違い

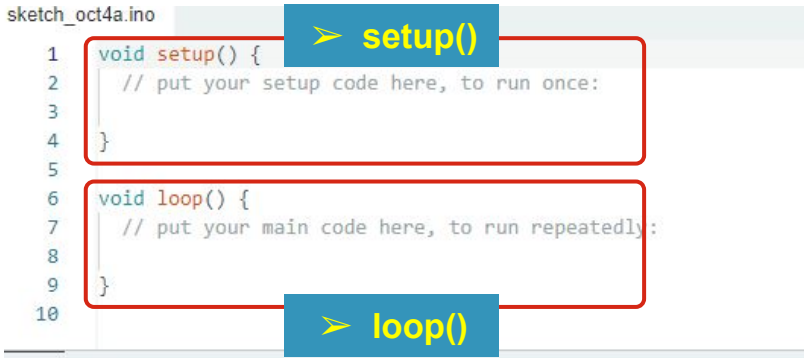
Arduino言語には「setup()」と「loop()」という処理が必須

➢ **setup()　※最初の1回だけの処理**

ボードの電源を入れた際、リセットした際に、一度だけ実行。
初期化するためのプログラムを入力。
変数の初期化、ピンの設定など

➢ **loop()　※繰り返し行う処理**

setup()が実行された後、繰り返し実行。
ボードの動きをコントロールするためのプログラムを入力。
センサーの読み取り、オン・オフなど



Arduinoスケッチを読み書きするための原則

Arduinoスケッチの記述ルール

1. プログラムは指示が無い限り、基本的に**上から下へ**実行される
2. 「**//**」で記述された文字は**コメント扱い**となり、動作しない
3. 文字の表示用途以外は、**全て半角文字で入力**する
4. 一つの命令分毎で、**文末はすべてセミコロン「;」**

★その他、頻度の高い要素・・・

- 括弧内で括弧を使うときはわかりやすくなるように**行頭の階層を下げる（インデント）**
- 大文字・小文字は**区別する**

```
void setup() {
  Program.hello();
}
```

```
void loop() {
  if(world < 10){
    xyz = world * 10;
  }
}
```

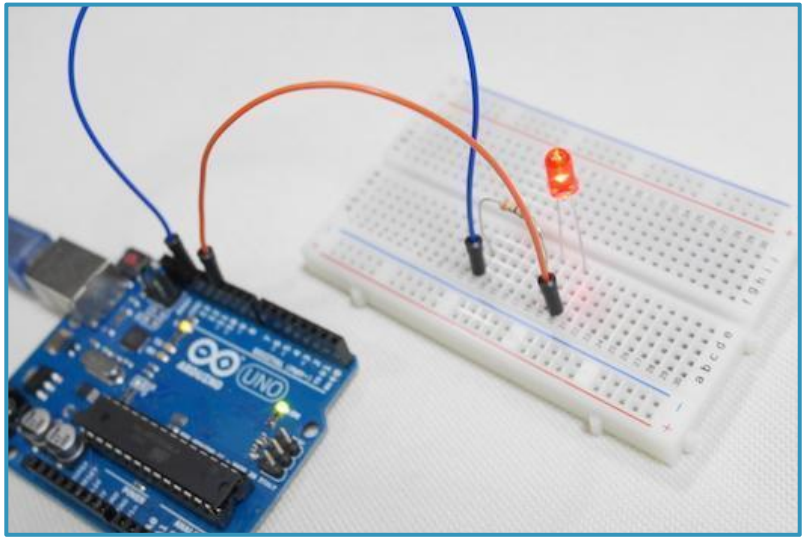
コーディング：LEDを点灯させる！

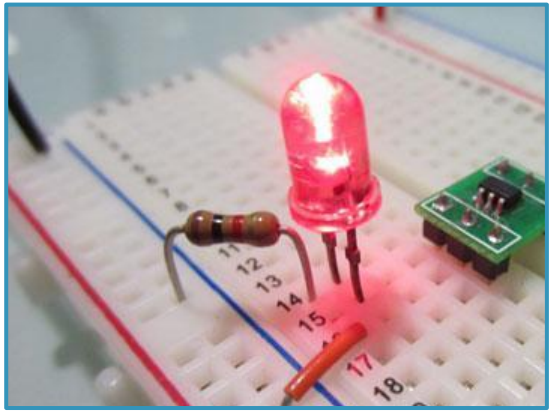
「ブレッドボード」上のLEDを点灯させる！

コーディング：LEDを点灯させよう！

「ブレッドボード」上で「Lチカ」を行う

- 先程行いました**「Hello World」**の表示では、マイコンボードの動作プロセスを体験いただきましたが、これからは実際の部材をマイコンボードに繋げて、プログラムを通じて制御します。
  ここでは、動作検証の**【定番】**ともいえる、**【LED】**の動作検証、、通称**【Lチカ】**のコードを作成しましょう！



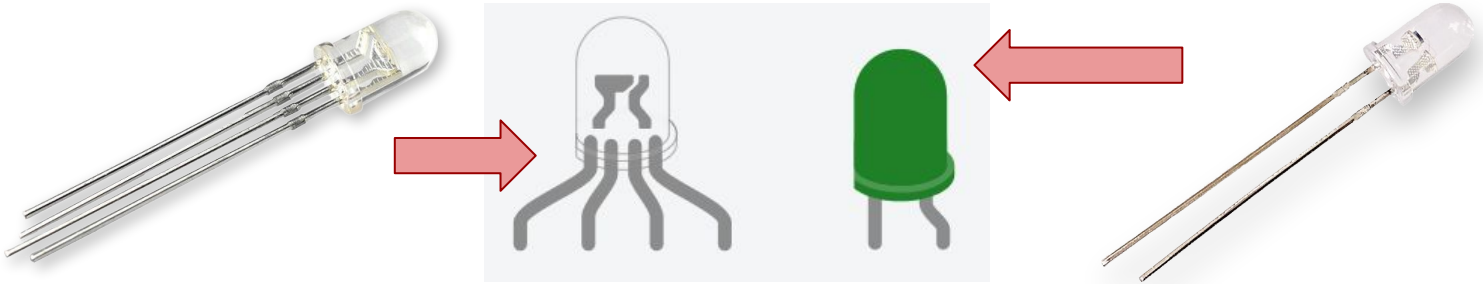


※写真は実際に動作させているイメージです。

使用する部材について

使用するLEDの補足：単色LED・多色LED

- ここで動作させるLEDは、電流を流すことで自らを発光させる部材です。
- 実際には、大きさや形状で様々な種類があり、仕様・用途に応じて選定を行います。
- 大きな特徴として、単色で発光するタイプのものと、多色（ 2色・3色フルカラー）で発光するものがあります。



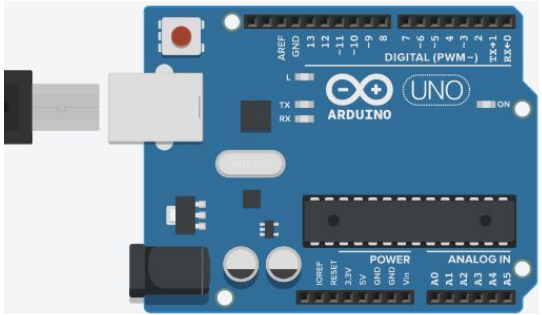
↑3色（多色）タイプのLED

↑単色タイプのLED

使用する部材について

使用する部材（LED）とプロトタイプとの関連性

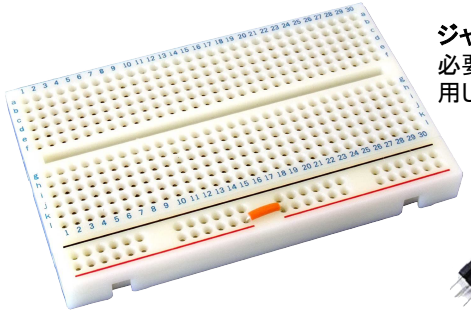
- 言わずもがな、LEDライトの【ライト】の部分に該当しますが、今回の実践を通して、この　2種類のタイプのLEDをそれぞれの特徴を踏まえながら、配線とプログラム作業を行い動作検証を行います。
- まずは、【単色】LEDの動作検証から行います



使用する部材について

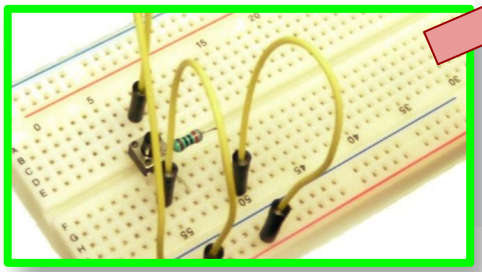
使用するその他の部材、ブレッドボード、ジャンパーワイヤー等の補足

- マイコンボードと併せ、プロトタイプ開発にあたって利用頻度の高い部材の紹介です。
- ブレッドボード：マイコンボードと他の部材を簡易的に接続するための配線固定用部材です
- ジャンパーワイヤー：ブレッドボード上で回路を形成するための配線用ワイヤーです。

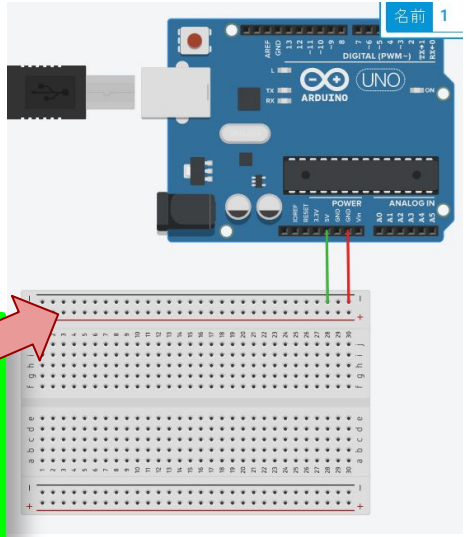


↑ブレッドボード

ジャンパワイヤー　↓
必要に応じて切り離して使用します。



※写真のようにブレッドボードにジャンパワイヤーを「刺して」配線を行います



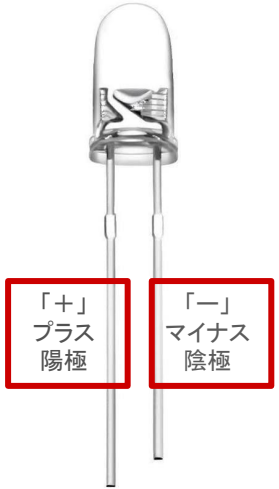
まずは、【配線】（≒回路設計）です！

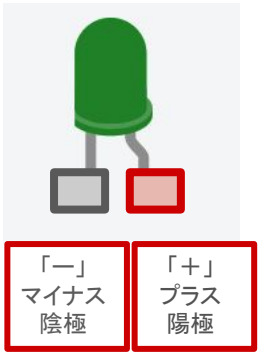
使用部材【LED】について

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う
LEDの極性について

- LEDには、通常の白熱球と同様、単色・多色問わず **極性**があります。
  一般的に線の長いほうが**「＋」**、短い方が**「ー」**になります。



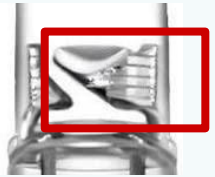


極性に関する呼称の補足

一般的な呼称として、、

【＋】極を**【アノード】**
【ー】極を**【カソード】**

それぞれ呼称します。



電極の幅が広い方が**「カソード」**

線の長さのほかに、LED内部の電極でも判別が可能ですが、種類によっては判別が難しい物もありますので、注意が必要です！

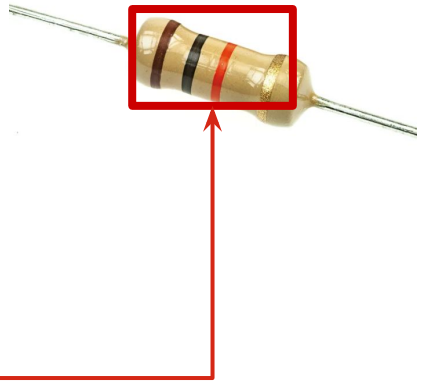
使用部材【抵抗器】について

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う
抵抗の特徴について

今回使用する抵抗器の特徴です。

- カーボン抵抗（炭素皮膜抵抗器）
- 回路に流す電流・**電圧の大きさ**や、**流れの方向を調整**する際に使用します。
- **極性はありません**。
  どの方向でも接続が可能です。
- **【抵抗値】**と呼ばれる値を読み、作成する回路に適合するか検討します。

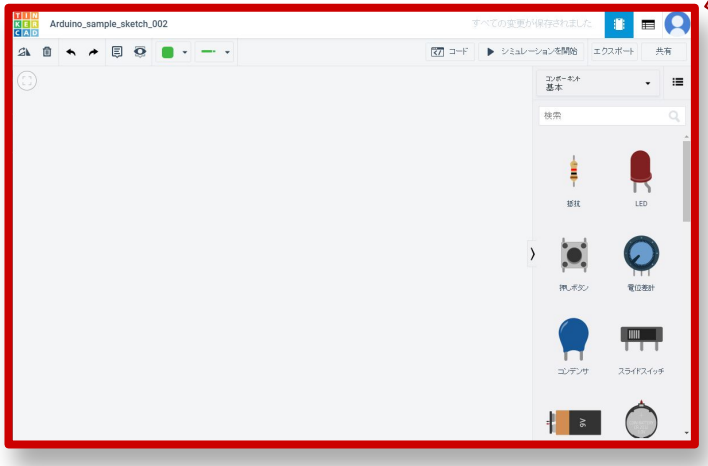


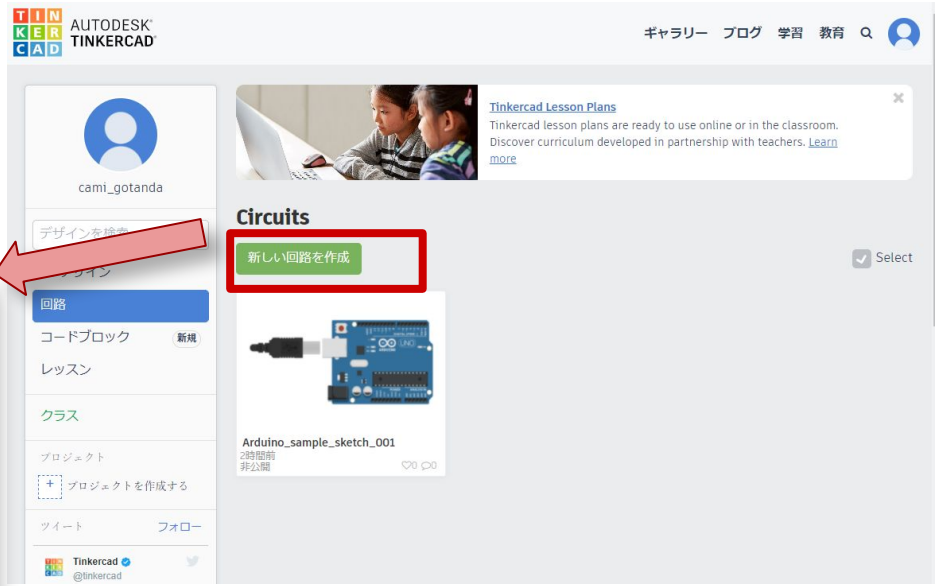
- **抵抗値**は、本体に印字されている**オビの色**で判断します。

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 前回の「Hello World！」作成時と同様に、【回路】制作のダッシュボードから新規作成を行います。





コーディング：単色LED点灯・点滅

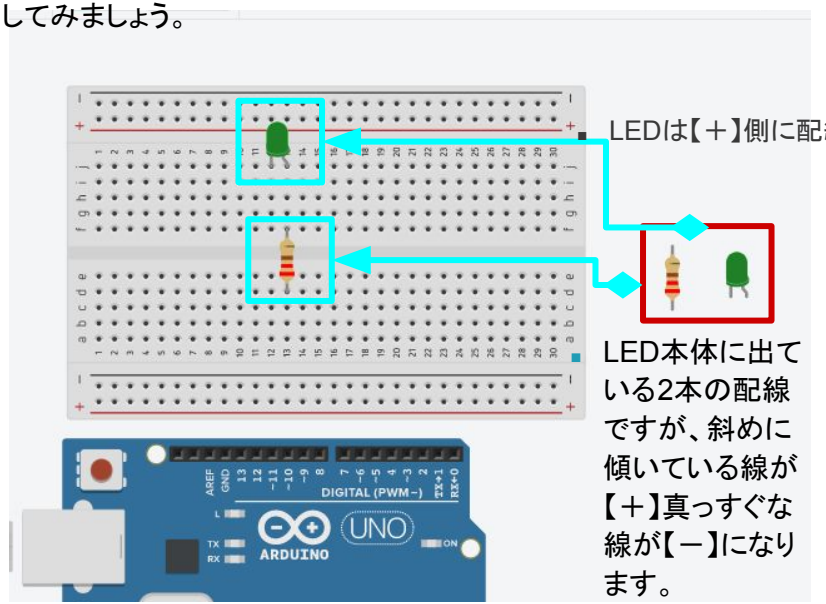
単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 制作画面が表示されたら、【コンポーネント一覧】から、【 Arduino Uno R3】【ブレッドボード（小）】【LED】【抵抗】を回路画面にドラッグ &ドロップして配置します

コーディング：単色LED点灯・点滅

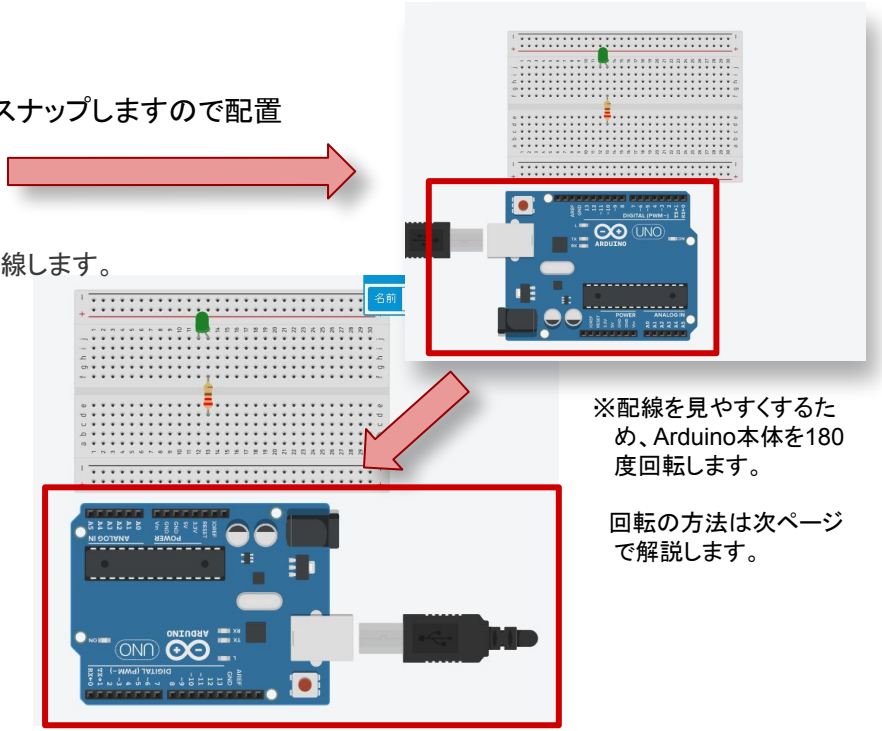
単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 配置した部材をブレッドボードに図のように接地します。
- 部材の先端をブレッドボードの穴（黒い点）に近づけると、自動的にスナップしますので配置してみましょう。



LEDは【＋】側に配線します。

LED本体に出ている2本の配線ですが、斜めに傾いている線が【＋】真っすぐな線が【－】になります。



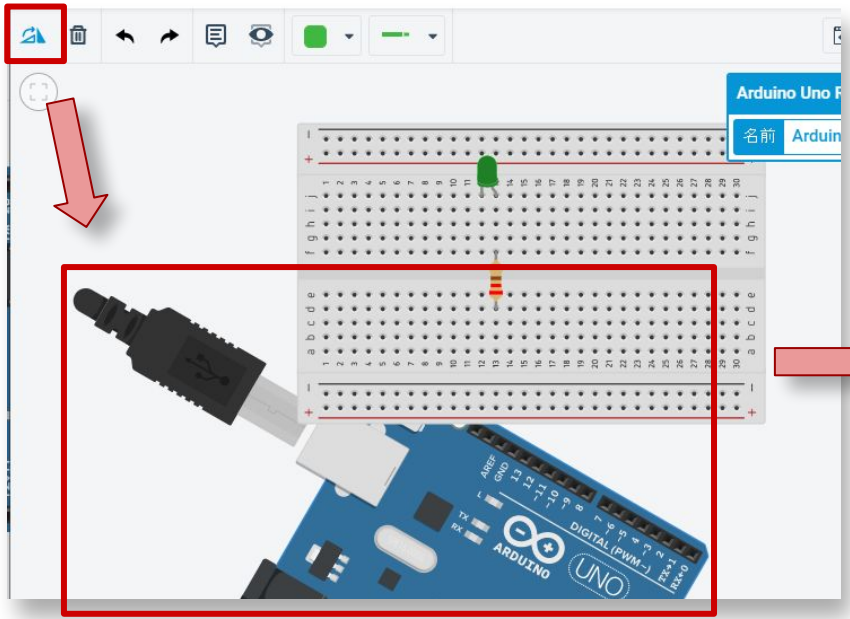
※配線を見やすくするため、Arduino本体を180度回転します。

回転の方法は次ページで解説します。

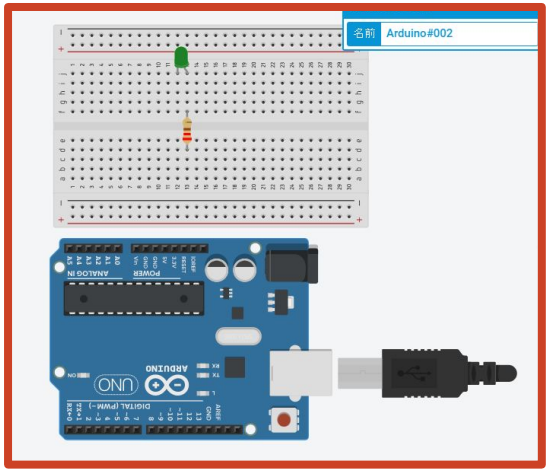
コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 部材の回転方法です。
- 画面左上にある【回転】アイコン（下図赤枠の箇所）を回転させたい部材を選択した状態でクリックすると回転します。



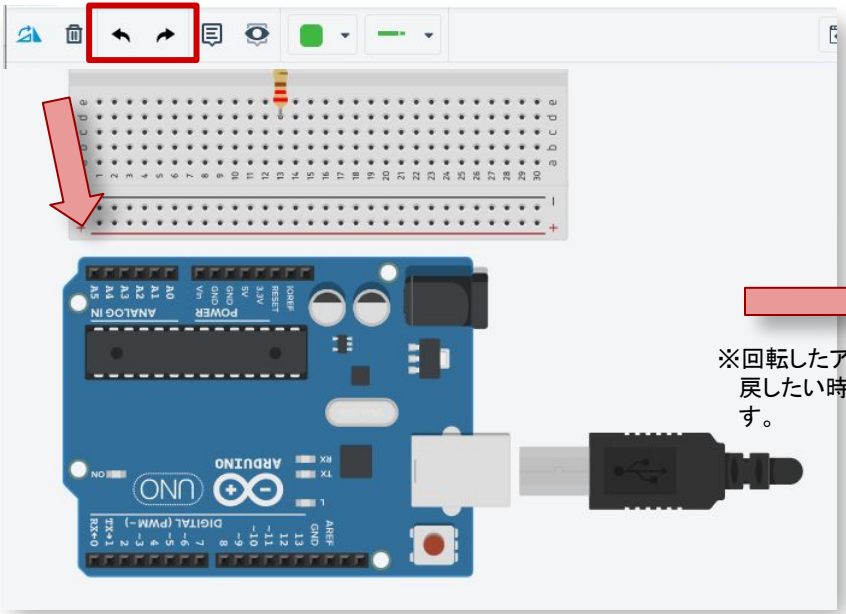
※数回押して右図のように
配置します



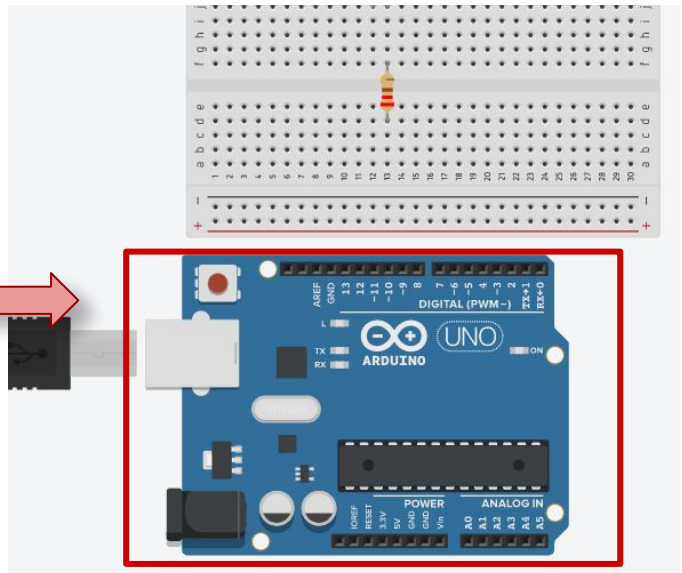
コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 【元に戻す】と【やり直し】の方法です。
- 画面左上にある【矢印】アイコン（下図赤枠の箇所）を押下すると、直前に行った作業に戻る【やり直し】と、もう一度同じ作業を行う【元に戻す】。



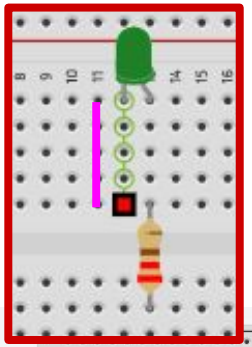
※回転したアイテムを元に戻したい時などに便利です。



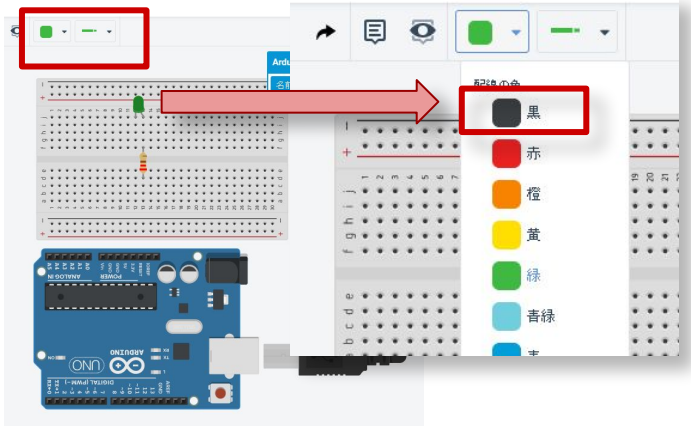
コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

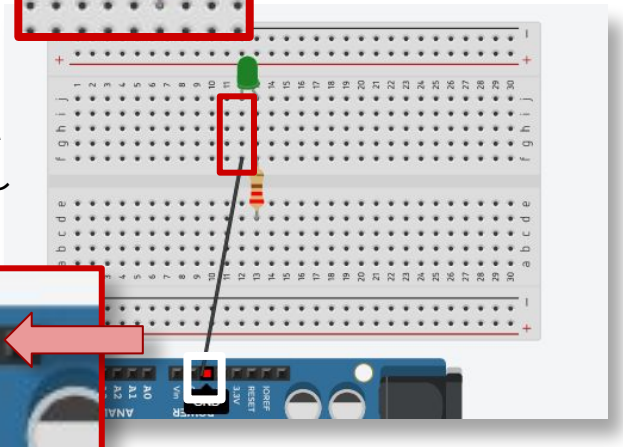
- LEDとArduinoを繋げます。
- まずは【－】極（カソード）です。
  右図のようにLED左側の線をArduino本体の**【GND】（グランド：マイナス極）**に繋げます。
- LED側の接点をクリックし、続けて Arduino側の【GND】PIN上でクリックすると接続が出来ます。



- ブレッドボードの配線は、図のピンク色の線に沿って繋がっています。
- LEDの接続している箇所では緑色の部分の穴でしたらどこにさしても配線が可能です。



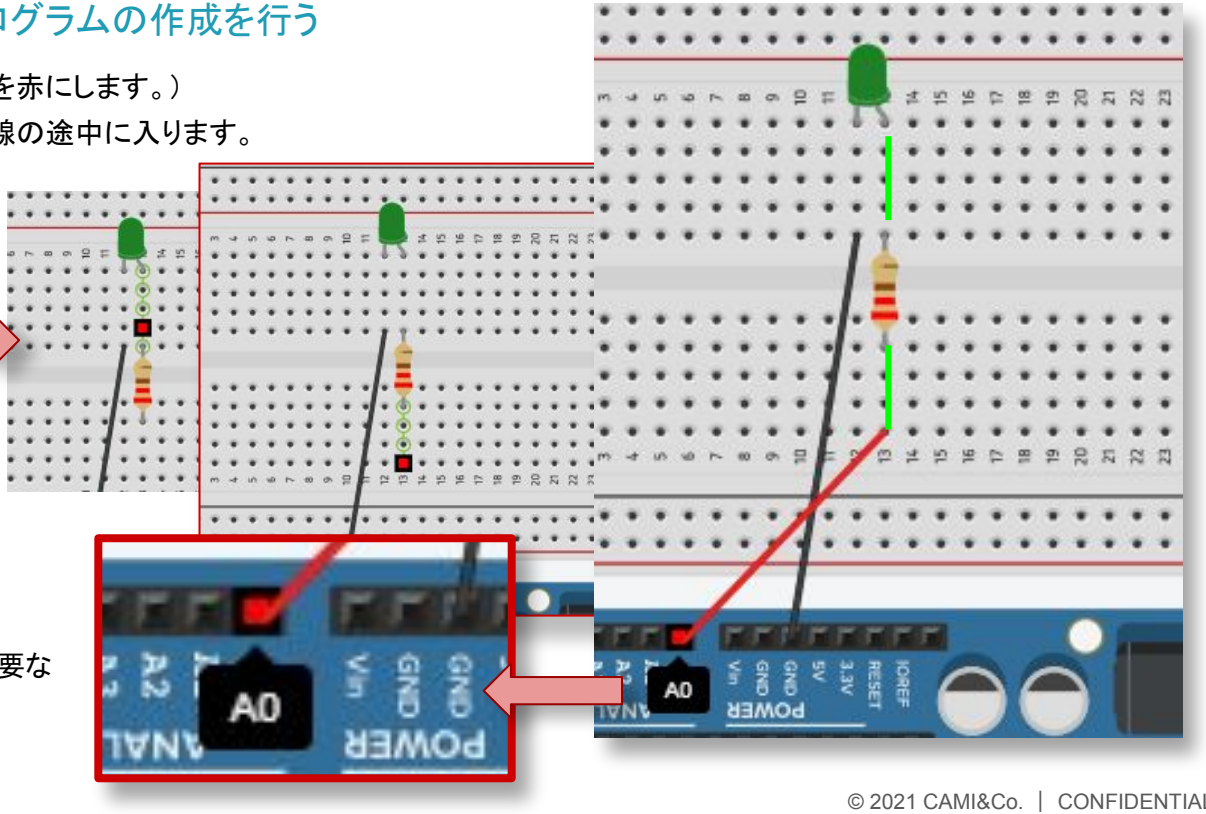
- 配線の際、選択中の線の色を変更することが可能です。
- ここでは、一般的な【－】極を意味する【黒】を選択し配線します



コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 続いてLED【＋】極の配線です。（線の色を赤にします。）
- ここで、冒頭で用意した部材【抵抗】が配線の途中に入ります。



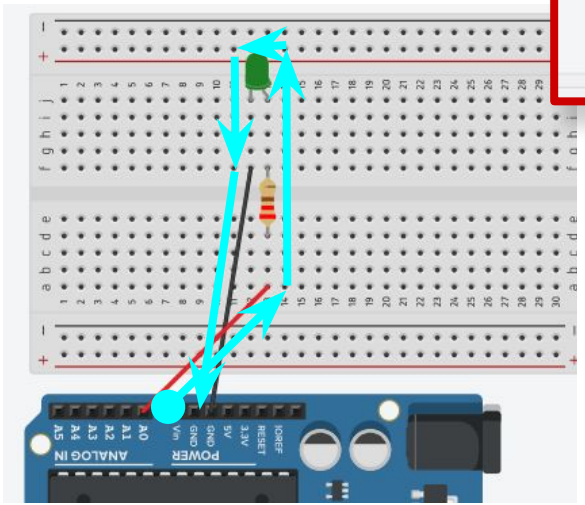
これはLEDを過度な電流から守るために必要な部材となります。

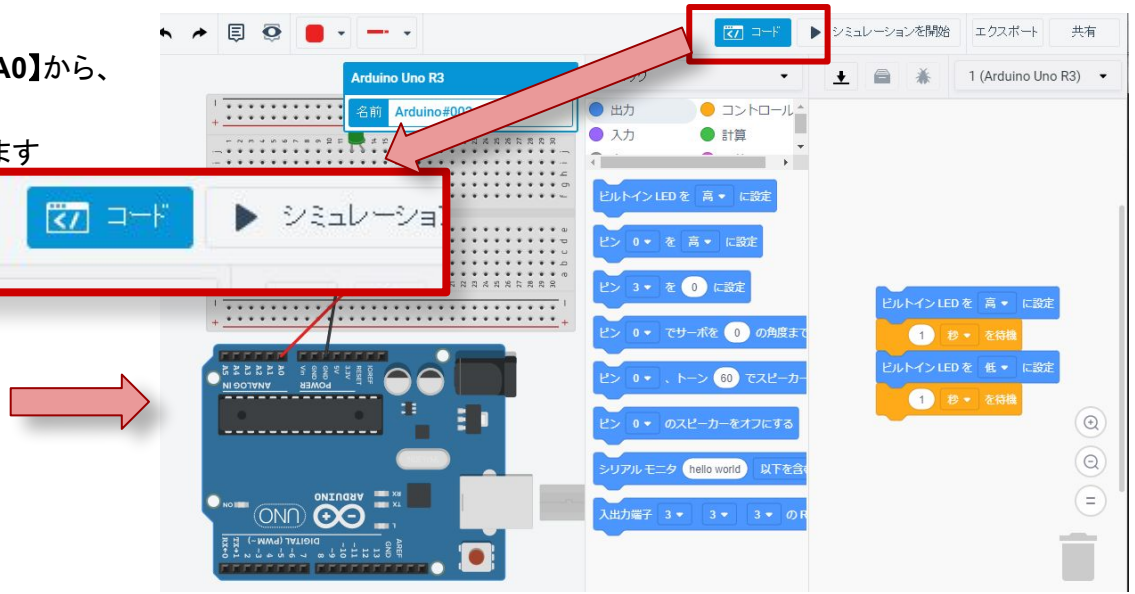
Arduino側の【A0】ピンへ接続します。

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 回路完成イメージです。
- 電流は水色の矢印線の通り、Arduino本体【**A0**】から、**LED**を通り、【**GND**】の方向に流れます
- 配線が終わりましたら「コードエディタ」を開きます



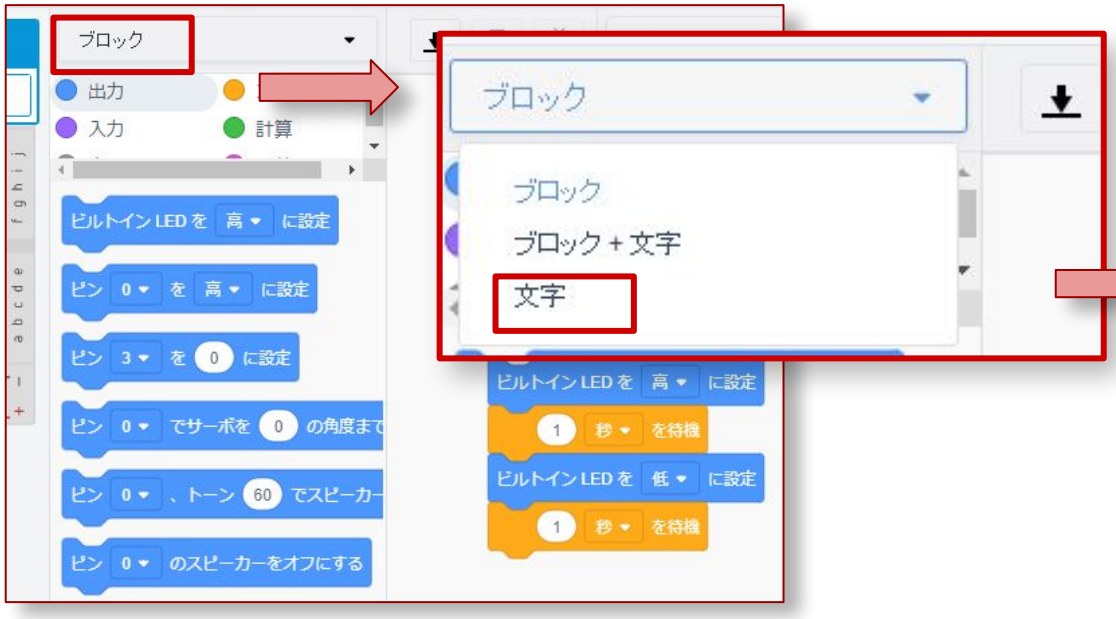


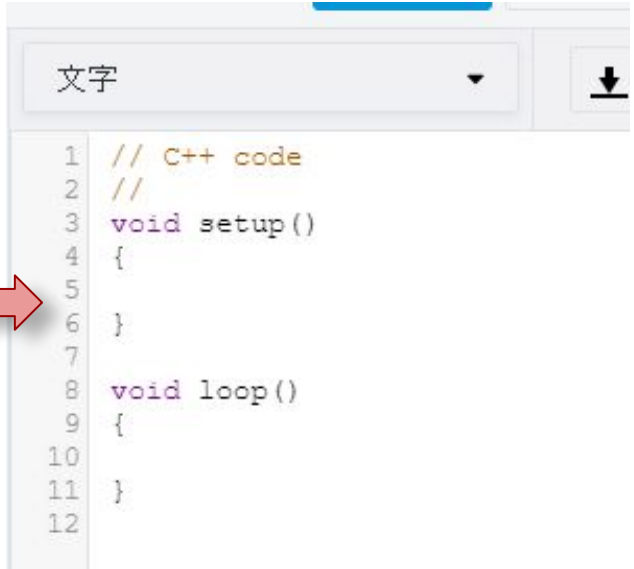
プログラム作成に移ります！

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 前回同様に【ブロック】エディタから【文字】エディタに切り替えます。





この状態にします

コーディング：単色LED点灯

単色LEDを点灯をさせるプログラムの作成を行う

- **今回から始めて記述した内容の補足です。**

```
1   #define LED_PIN = 5
2   int dltime = 1000;
3
4   void setup() {
5     Serial.begin(9600);
6     pinMode(LED_PIN, OUTPUT);
7   }
```

初期設定

1：変数のスコープ：「 dltime」と言う任意の文字列に「 int型」（整数型）と言うデータ形式の代入を可能にする設定です。
文字列【dltime】（Delay Timeを略しました。）を新たに設定（定義）し、整数【1,000】を代入します。
※関数の外で設定された変数を「グローバル (大域的)変数」と言い、どの関数内でも利用することが出来ます。

6：アナログPIN「5」番を有効化（通電可能状態）します。

```
9   void loop() {
10    digitalWrite(LED_PIN, HIGH);
11    Serial.println("LED : ON!");
12    delay(dltime);
13    digitalWrite(LED_PIN, LOW);
14    Serial.println("LED : OFF!");
15    delay(dltime);
16  }
```

メイン処理：ループ関数

10：PIN（5番）をON（HIGH）にします。

12：変数【dltime】に設定した秒数（ 1,000ミリ秒＝1秒）待ちます。

13：PIN（5番）をOFF（LOW）にします。

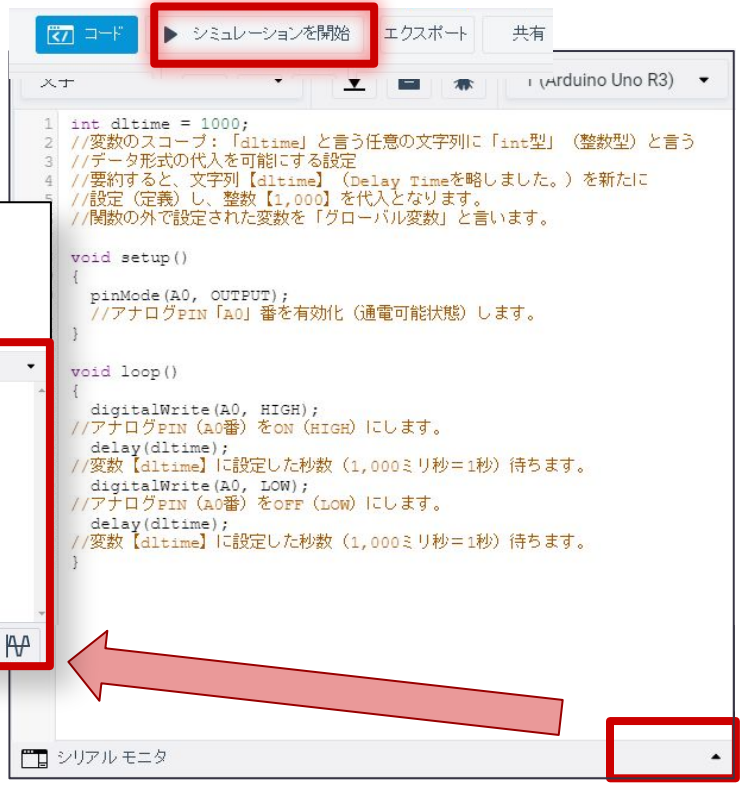
コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

前節同様、記述したプログラムを起動してみたいと思います。

1. エディター最下部（右下の赤枠部分）【▲】を押下し、シリアルモニタが下からスライドアップして出てきたことを確認します。

2. 【シミュレーションを開始】を押下すると、作成したコードが起動します。

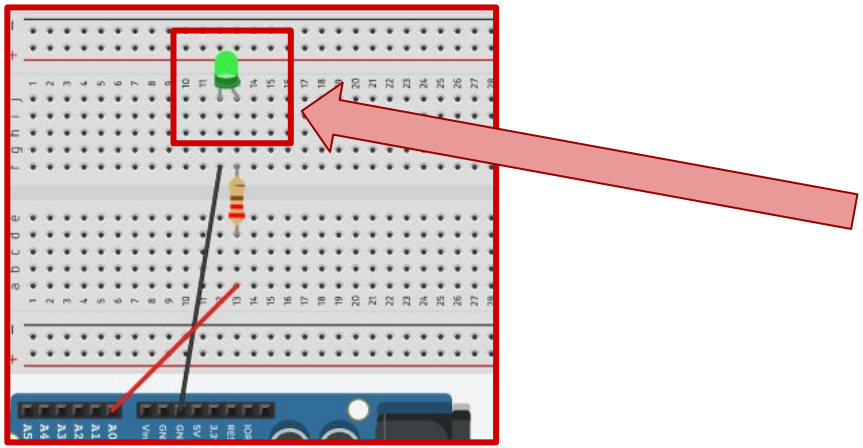


表示したシリアルモニタと併せ、フィールドに配置したブレッドボード上のLEDにご注目ください！

コーディング：単色LED点灯・点滅

単色LEDを点灯・点滅をさせるプログラムの作成を行う

- 画面の通り、シリアルモニタ上に【**LED : ON!**】と【**LED : ON!**】の表示が1秒毎に表示され、フィールド上の LEDが点滅していれば成功です！
- 早めに終わった方は、記述したコードの記述を変更してみて動きを検証してみてください。



```
10    pinMode(A0, OUTPUT);
11    //アナログPIN「A0」番を有効化（通電可
12
13    Serial.begin(9600);
14
15  }
16
17  void loop()
```

シリアル モニタ

```
LED : ON!
LED : OFF!
LED : ON!
LED : OFF!
LED : ON!
```

※次ページ以降では、プログラムの各記述内容の解説を行っていますので、動作検証と併せ確認してみてください。

コーディング：単色LED点灯

単色LEDを点灯をさせるプログラムの作成を行う：コピペ用サンプルコード（補足付きバージョン）

```
int dltime = 1000;
//変数のスコープ：「dltime」と言う任意の文字列に「int型」（整数型）と言う/データ形式の代入を可能にする設定
//要約すると、文字列【dltime】（Delay Timeを略しました。）を新たに/設定（定義）し、整数【1,000】を代入となります。
//関数の外で設定された変数を「グローバル変数」と言います。

void setup()
{
  pinMode(5, OUTPUT);
  //「5」番を有効化（通電可能状態）します。

  Serial.begin(9600);

}

void loop()
{
  digitalWrite(5, HIGH);
//アナログPIN（5番）をON（HIGH）にします。

  Serial.println("LED : ON!");

  delay(dltime);
//変数【dltime】に設定した秒数（1,000ミリ秒＝1秒）待ちます。
  digitalWrite(5, LOW);
//アナログPIN（5番）をOFF（LOW）にします。

  Serial.println("LED : OFF!");

  delay(dltime);
//変数【dltime】に設定した秒数（1,000ミリ秒＝1秒）待ちます。
}
```

コーディング：単色LED点灯

単色LEDを点灯をさせるプログラムの作成を行う：コピペ用サンプルコード
※前ページのシンプル・LEDピン変数化 版

```
int LED_PIN = 5;
int dltime = 1000;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  Serial.println("LED : ON!");
  delay(dltime);
  digitalWrite(LED_PIN, LOW);
  Serial.println("LED : OFF!");
  delay(dltime);
}
```

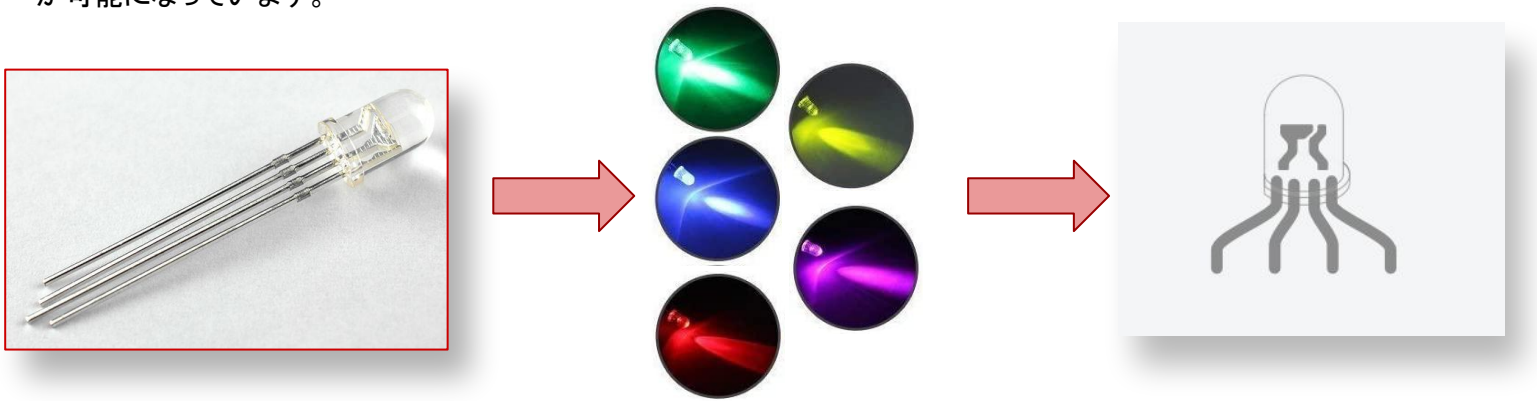
コーディング：LEDの点灯　3色LED編

「ブレッドボード」上のLEDを、複数色同時に点灯させる！

コーディング②:3色LED点灯・点滅

3色LEDを点灯・点滅をさせるプログラムの作成を行う

- 単色LEDに続いて、3色LEDを動作させるプログラムの作成を行います。
- 単色LEDとの大きな違いとして、各色に対応した極性があります。
- 今回使用するLEDの場合、赤（R）緑（G）青（B）の3色になり、各色の光の加減をコントロールすることによって、様々な色を再現することが可能になっています。

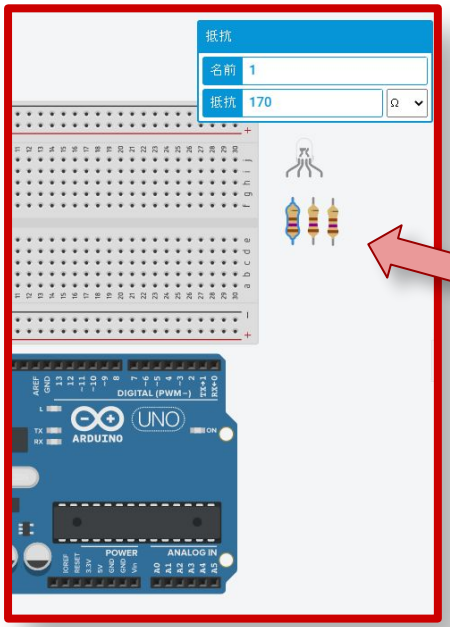


ここではこの3色LEDをArduinoに繋げて、前節の単色LEDプログラムを応用し、動作原理の違いを確認いたします。

コーディング②:3色LED点灯・点滅

コーディング②:3色 LED点灯・点滅

- 前節同様、【コンポーネント一覧】から、【 Arduino Uno R3】【ブレッドボード(小)】【 3色LED】【抵抗】を回路画面にドラッグ &ドロップして配置します

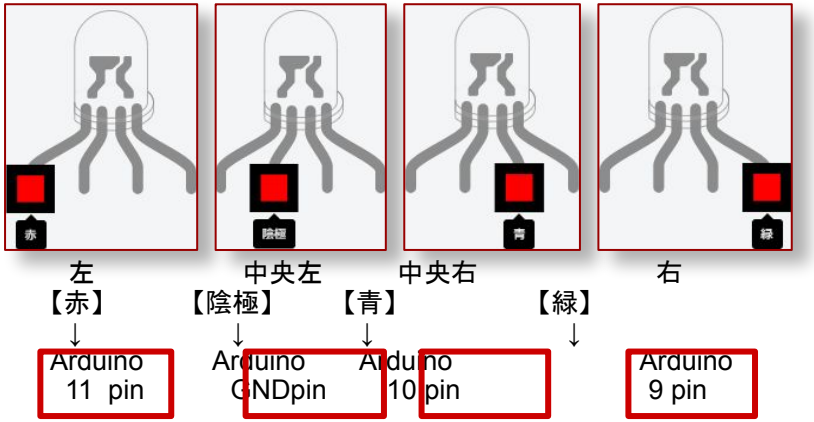


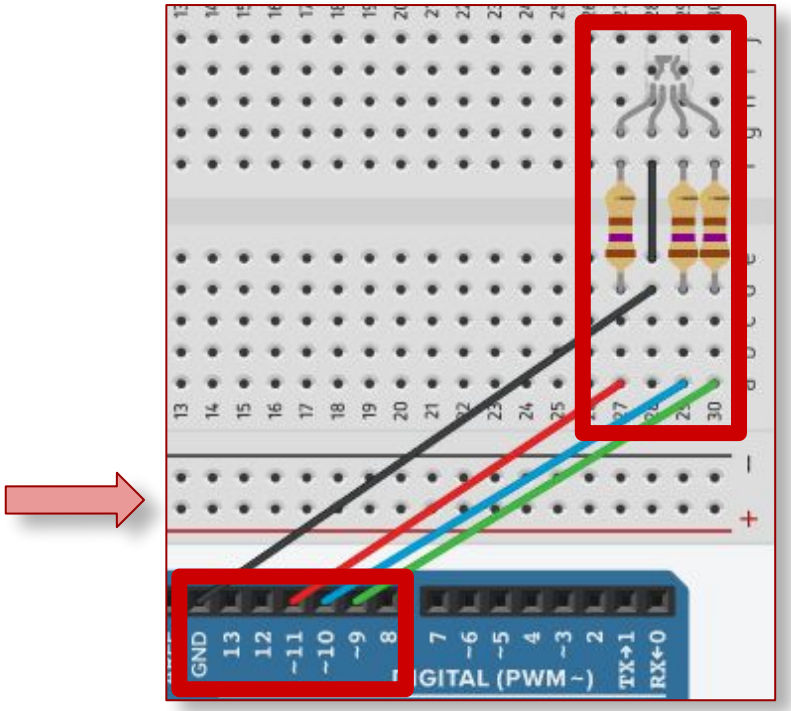
- 【抵抗】は色数に併せ 3本使用しますので、フィールドには3本配置してください。

コーディング②:3色LED点灯・点滅

コーディング②:3色 LED点灯・点滅

- 続いて配線です。
- 画面左から**2番目が【GND】(グランド)**になります。
- 【＋】側は下記の通りに配線します。
  - **【赤】をArduino側の【11番】ピン**
  - **【緑】をArduino側の【9番】ピン**
  - **【青】をArduino側の【10番】ピン**

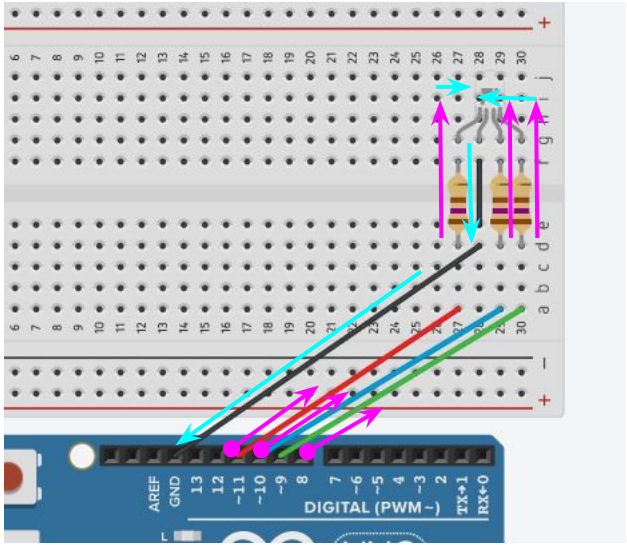




コーディング②:3色LED点灯・点滅

コーディング②:3色 LED点灯・点滅

- 回路完成イメージです。
- 電流は矢印線の通り、Arduino本体**【9番】【10番】【11番】**から、**LED**を通り、**【GND】**の方向に流れます



- 一見複雑そうな多色 LEDですが、基本的には単色 LEDと同様に、各色の導線からGNDに配線し、プログラムによって各色の光の加減を調整します。
- 一つのGNDを各色で共有し、**【アノード】**(＋) から**【カソード】**(ー) 方向に電流が流れます。
- **配線が終わりましたら「コードエディタ」を開きます**

コードディング用：テキスト

プログラムの作成を行うコピペ用コピペ用サンプルコード

- 複数段組になっていますが、前ページのテキストを順にコピー＆ペーストしてください。

```
int Led_red = 11;
int Led_blue = 10;
int Led_green = 9;

void LED_color (
  unsigned char color_RED,
  unsigned char color_Green,
  unsigned char color_Blue
)
{
  analogWrite(Led_red, color_RED);
  analogWrite(Led_green, color_Green);
  analogWrite(Led_blue, color_Blue);
}
```

```
void setup() {
  Serial.begin(9600);
  pinMode(Led_red, OUTPUT);
  pinMode(Led_green, OUTPUT);
  pinMode(Led_blue, OUTPUT);
}

void loop() {
  for (int i=0; i < 200; i++) {
    LED_color(i, 50, 0);
    delay(1);
  }
  for (int i=200; i > 0; i--) {
    LED_color(i, 0, 100);
    delay(10);
  }
}
```

コーディング②:3色LED点灯・点滅

3色LEDを点灯・点滅をさせるプログラムの作成を行う

- **記述した内容の補足です。**

```
2  int Led_red = 11;
3  int Led_blue = 10;
4  int Led_green = 9;
5
```

初期設定
2:変数のスコープ: 3色LEDアノード側のピン番号をそれぞれ「 Led_red」「Led_blue」「Led_green」に対し、Arduinoの11番、10番、9番に設定

```
6  void LED_color (
7    unsigned char color_RED,
8    unsigned char color_Green,
9    unsigned char color_Blue
10 )
11 {
12 analogWrite(Led_red, color_RED);
13 analogWrite(Led_green, color_Green);
14 analogWrite(Led_blue, color_Blue);
15 }
```

6:LEDの色情報を同梱して発光処理を関数化します。
関数名【LED_color()】
引数として【color_RED】【color_Green】【color_Blue】を定義

13:各色に対応した引数にピン番号の色情報を紐づけ、適時変更しやすいように一つの関数にまとめます。
※引数の設定例: analogWrite(Led_red, color_RED);
電流 ON (11番ピン, 発光の大きさ);

※設定した関数の設定内容: LED_color (color_RED, color_Green, color_Blue);
関数処理 (赤の光量, 緑の光量, 青の光量);

コーディング②:3色LED点灯・点滅

3色LEDを点灯・点滅をさせるプログラムの作成を行う

```
18  void setup()
19  {
20    Serial.begin(9600);
21    pinMode(Led_red, OUTPUT);
22    pinMode(Led_green, OUTPUT);
23    pinMode(Led_blue, OUTPUT);
24  }
```

```
25  void loop()
26  {
27    for (int i=0; i < 200; i++) {
28      LED_color(i, 50, 0);
29      delay(1);
30    }
31
32    for (int i=200; i > 0; i--) {
33      LED_color(i, 0, 100);
34      delay(10);
35    }
36  }
```

初期設定

21:Arduino側指定ピンの有効化
「Led_red(11番)」「Led_blue(10番)」「Led_green(9番)」をそれぞれ出力用で設定

メイン処理:ループ関数

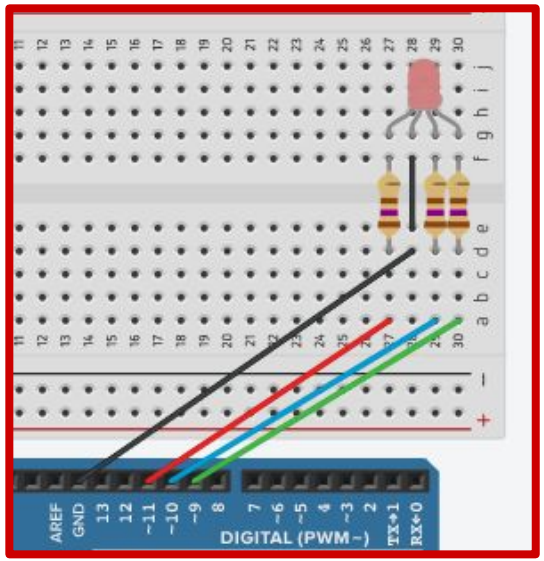
27:変数「i」を設定し、0(i=0)から200(i<200)までを1ずつ増加(i++)させるループ処理を設定
for文内のループ処理 ------
発光処理:11番ピンのみ変数「i」に準じて加算(フェードイン発光)
他の色(緑・青)は無発光(0で指定)
1ミリ秒待機
for文内のループ処理 ------ここまで

32:変数「i」を設定し、200(i=200)から0(i>0)までを1ずつ減算(i++)させるループ処理を設定
for文内のループ処理 ------
発光処理:11番ピンのみ変数「i」に準じて増加(フェーアウト発光)
他の色(緑・青)はそれぞれ「緑:0」と「青:100」で指定
10ミリ秒待機
for文内のループ処理 ------ここまで

コーディング②:3色LED点灯・点滅

3色LEDを点灯・点滅をさせるプログラムの作成を行う

- プログラムを実行すると、画面に配置した 3色LEDの赤系がゆっくり点滅していれば成功です
- 時間に余裕があれば、緑・青の色情報の値も変更して、 LEDの色の変化を確認してみてください。



- 3色LEDが赤くフェードイン・フェードアウトを繰り返しながら発光します。

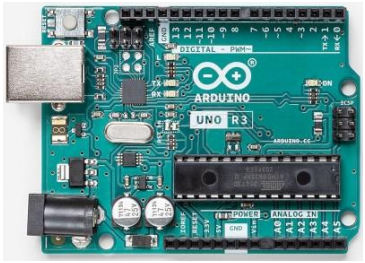
※次ページ以降では、プログラムの各記述内容の解説を行っていますので、動作検証と併せ確認してみてください。

コーディング②:3色LED点灯・点滅

使用する部材とプロトタイプとの関連性

- LEDの点灯・点滅を行う制御ですが、いかがでしたでしょうか。
- 同じLEDでも、単色・多色の違いでそれぞれ制御の方法が違うことが実践を通してご理解いただけたかと思います。
- 次の作業では、いよいいよセンサーの制御方法について作業を通して体験いただきたいと思います。

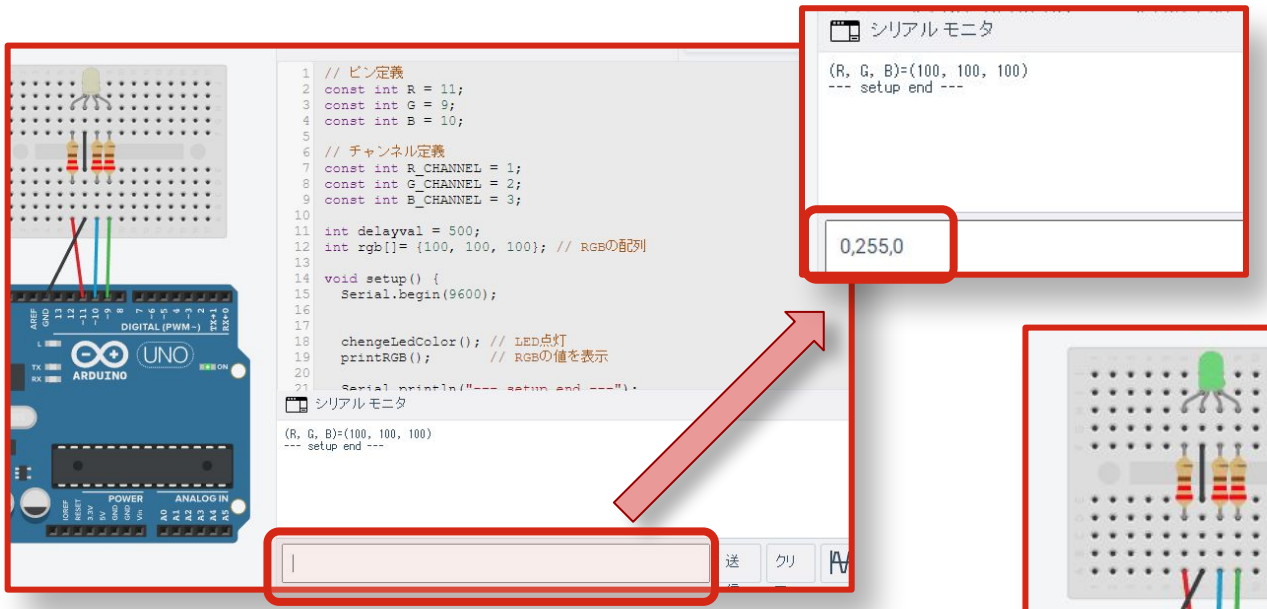
Switch ON !



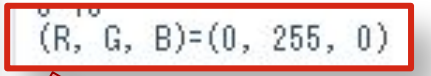
コーディング：おまけプログラム3色LED点灯・点滅

3色LEDを任意の色に点灯

- シリアルモニタの最下段にある【入力欄】に LEDのRGB輝度を入力すると、指定の輝度で点灯します。



- ←ここでは、R,G,Bの値をそれぞれ、0,255,0と入力



LED]は、緑色に点灯します。

コードディング用：テキスト

プログラムの作成を行うコピペ用コピペ用サンプルコード

- 複数段組になっていますが、テキストを順にコピー＆ペーストしてくださ い。

```
// ピン定義
const int R = 11;
const int G = 9;
const int B = 10;

// チャンネル定義
const int R_CHANNEL = 1;
const int G_CHANNEL = 2;
const int B_CHANNEL = 3;

int delayval = 500;
int rgb[]= {100, 100, 100}; // RGBの配列

void setup() {
  Serial.begin(9600);


  chengeLedColor(); // LED点灯
  printRGB();       // RGBの値を表示

  Serial.println("--- setup end ---");
}
```

```
void loop() {
  if (Serial.available() > 0) {
    setRGB();         // シリアルがあれば RGBの値を設定
    chengeLedColor(); // LED点灯
    printRGB();       // RGBの値を表示
  }
  delay(delayval);
}
//  LEDの色を変更する関数

void chengeLedColor() {
  analogWrite(R,map(rgb[0],0,1023,0,255)) ;
  analogWrite(G,map(rgb[1],0,1023,0,255)) ;
  analogWrite(B,map(rgb[2],0,1023,0,255)) ;
}
// シリアル通信で送られたコマンドを文字列にする関数
String getSerialCommand() {
  String s = "";
  while (Serial.available() > 0) {
    char c = Serial.read();
    s = s + String(c);
  }
  return s;
}
```

```
// 現在のRGBの値を表示させる関数
void printRGB() {
  Serial.print("(R, G, B)=(");
  Serial.print(rgb[0]);
  Serial.print(", ");
  Serial.print(rgb[1]);
  Serial.print(", ");
  Serial.print(rgb[2]);
  Serial.println(")");
}
void setRGB() {
  String s = getSerialCommand();
  for (int i = 0; i <= 2; i++) {
    int mark = s.indexOf(",");
    Serial.print(mark);
    if (mark > 0) {
      rgb[i] = s.substring(0, mark).toInt();
      Serial.println(s);
      s = s.substring(mark + 1);
      Serial.print(s);
    } else {
      rgb[i] = s.toInt();
      Serial.println(s);
    }
  }
}
```

コードディング用：テキスト

プログラムの作成を行うコピペ用コピペ用サンプルコード（文字化け対策用）

- 複数段組になっていますが、テキストを順にコピー＆ペーストしてくださ い。

```
const int R = 11;
const int G = 9;
const int B = 10;

const int R_CHANNEL = 1;
const int G_CHANNEL = 2;
const int B_CHANNEL = 3;

int delayval = 500;
int rgb[]= {100, 100, 100};

void setup() {
  Serial.begin(9600);
  chengeLedColor();
  printRGB();
  Serial.println("--- setup end ---");
}
void loop() {
  if (Serial.available() > 0) {
    setRGB();
    chengeLedColor();
    printRGB();
  }
  delay(delayval);
}
```

```
void chengeLedColor() {
  analogWrite(R,map(rgb[0],0,1023,0,255)) ;
  analogWrite(G,map(rgb[1],0,1023,0,255)) ;
  analogWrite(B,map(rgb[2],0,1023,0,255)) ;
}

String getSerialCommand() {
  String s = "";
  while (Serial.available() > 0) {
    char c = Serial.read();
    s = s + String(c);
  }
  return s;
}

void printRGB() {
  Serial.print("(R, G, B)=(");
  Serial.print(rgb[0]);
  Serial.print(", ");
  Serial.print(rgb[1]);
  Serial.print(", ");
  Serial.print(rgb[2]);
  Serial.println(")");
}
```

```
void setRGB() {
  String s = getSerialCommand();
  for (int i = 0; i <= 2; i++) {
    int mark = s.indexOf(",");
    Serial.print(mark);
    if (mark > 0) {
      rgb[i] = s.substring(0, mark).toInt();
      Serial.println(s);
      s = s.substring(mark + 1);
      Serial.print(s);
    } else {
      rgb[i] = s.toInt();
      Serial.println(s);
    }
  }
}
```

コーディング：センサーを使ってみよう！

「tinker cad」上で各種センサーの動作検証を行う

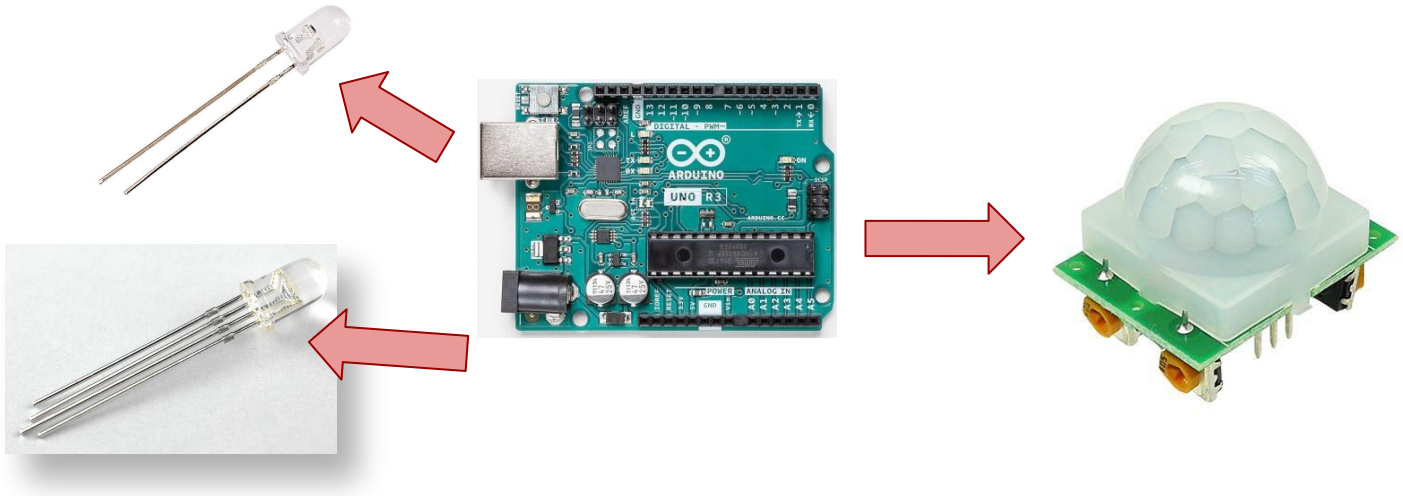
動作検証 PIR（人感）センサー編

焦電型赤外線センサーの動作検証を行う

コーディング：PIRセンサー（人感）

使用する部材PIRセンサーについて

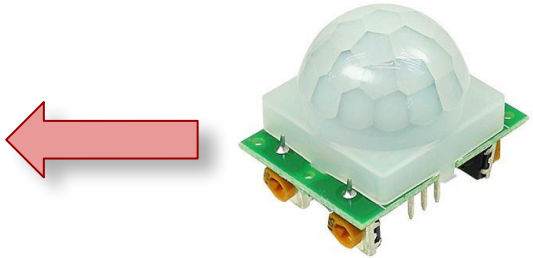
- 前節では、光を発する側の部材を用いて Arduinoの制御をおこないました。
- ここでの作業は、計測する側の部材、センサーを用いた動作検証を通してその制御方法を体験いただきたいと思います。



コーディング：PIRセンサー（人感）

赤外線検知型の人感センサー(PIR)の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

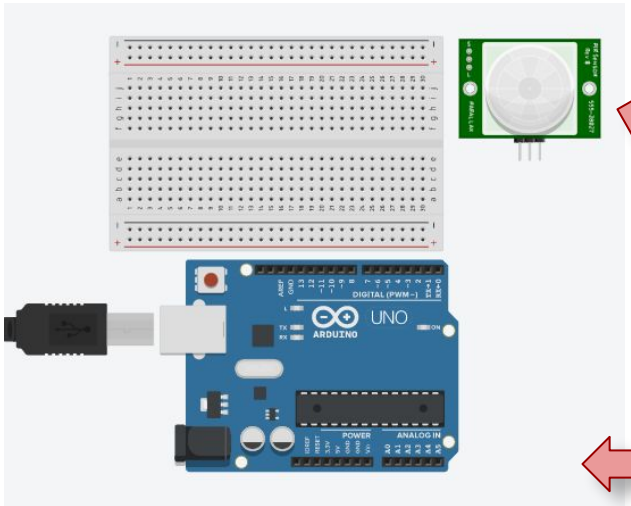
- 人感センサーとは、人や動物などから発せられる熱放射という赤外線などの電磁波を感知し、周囲温度との比較により検知エリア内の状態を判定します。
- 外気温を基準にして、何らかの熱源が検知エリア内に入った時や、エリア内で動きがあった時に状態変化を検知し、スイッチのオン・オフなどをコントロールします。

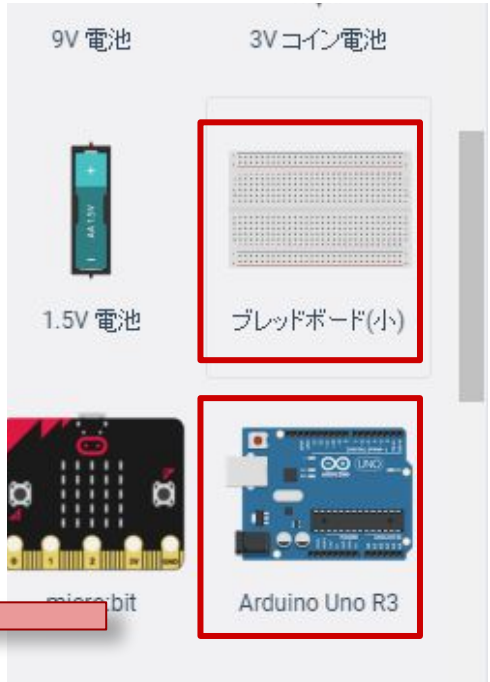


コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- 【コンポーネント一覧】から、【Arduino Uno R3】【ブレッドボード（小）】【PiRセンサー】を回路画面にドラッグ &ドロップして配置します。
- 今回は【抵抗】は使用しませんが、実機でこれを行う場合は、使用するセンサーに適合した抵抗も用意しておきます。

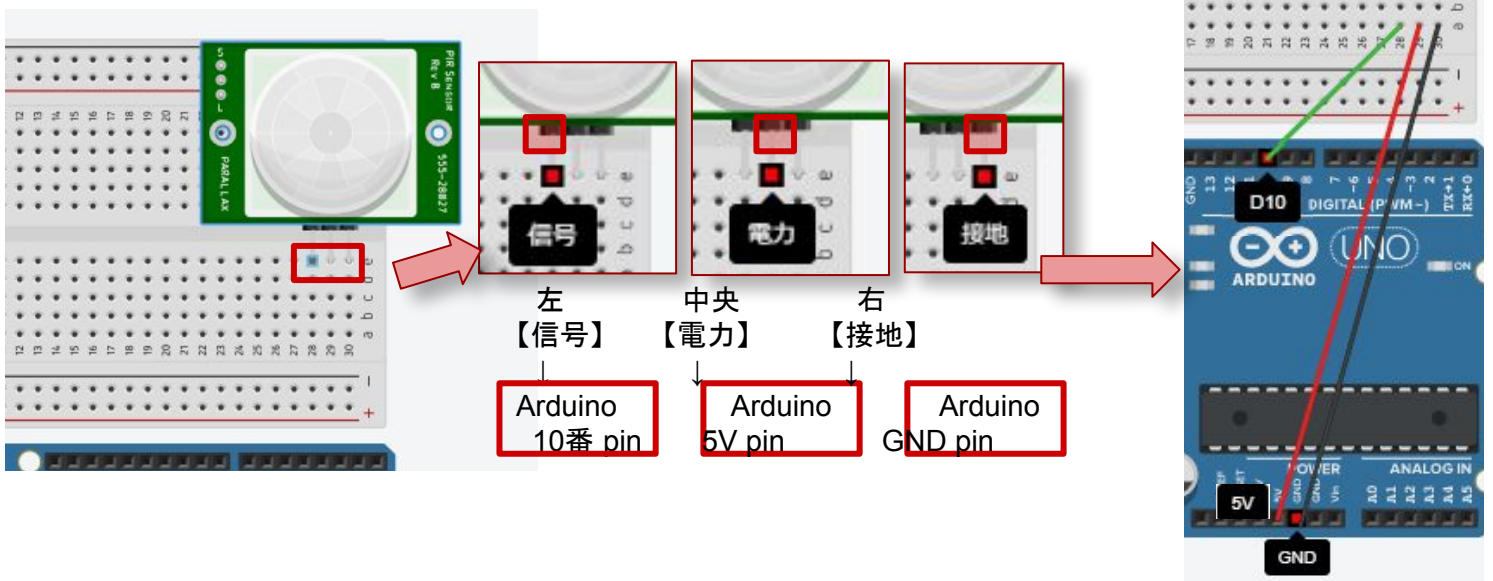




コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

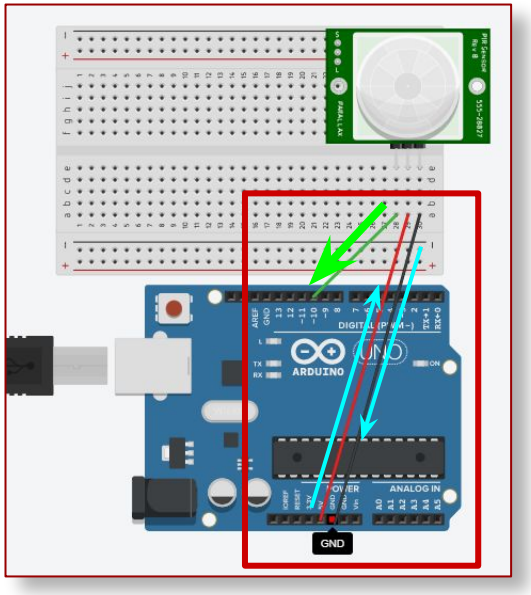
- 続いて配線です。
  画面左から【信号】（Arduino側10番pin）【電源】（5V）【GND】（グランド）になります。

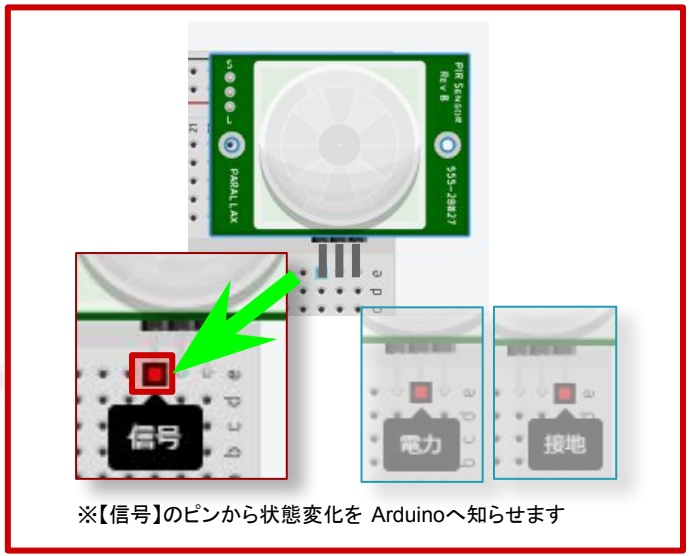


コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- 回路完成イメージです。
  電流は**水色の矢印線**の通りに流れるのと併せ、センサーは **状態変化の検知** を**緑色の矢印線**から信号を発信します。





※【信号】のピンから状態変化を Arduinoへ知らせます

- 配線の確認が終わり次第、コーディングに進みましょう。

コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う：コピペ用サンプルコード

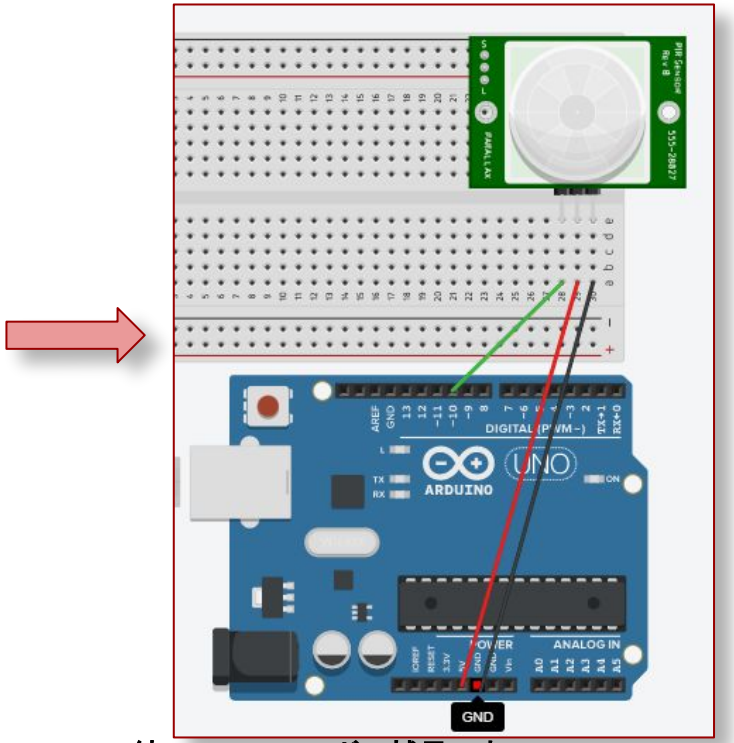
```
int myPIR = 0;
#define PIRPIN 10

void setup()
{
  pinMode(PIRPIN, INPUT);
  Serial.begin(9600);
}

void loop()
{
  myPIR = digitalRead(PIRPIN);
  Serial.print("PIR status: ");
  Serial.println(myPIR);
}
```

コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- 前ページのテキストをコピー＆ペーストします。

```
int myPIR = 0;
#define PIRPIN 10

void setup()
{
  pinMode(PIRPIN, INPUT);
  Serial.begin(9600);
}

void loop()
{
  myPIR = digitalRead(PIRPIN);
  Serial.print("PIR status: ");
  Serial.println(myPIR);
}
```

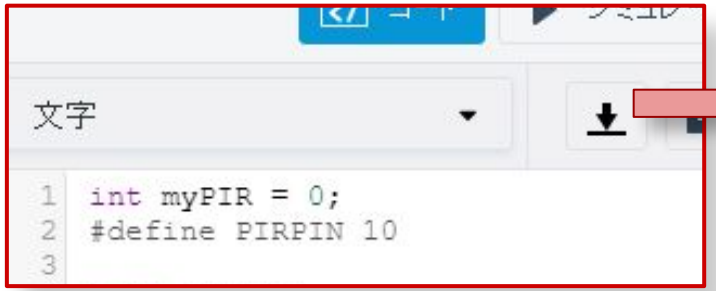


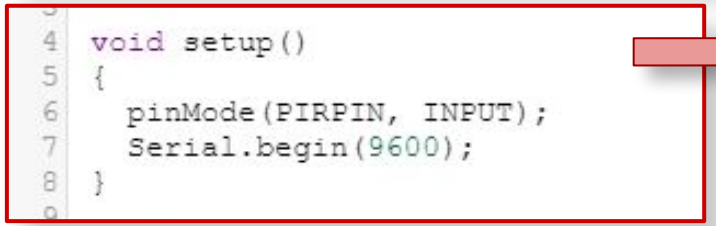
続いて、このコードの補足です

コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- **記述した内容の補足です。**





初期設定
1：人感センサー（PIR）の状態検知（On Offの判定）用の変数
【myPIR】を設定＆初期化　※ここでは、「 0」（ ＝ OFF状態）を設定
※変数名は任意に決めても OKです。

2：人感センサーの信号を読み取るための配線（緑）用ピン番号の設定
※ここでは、デジタルピンの 10番を指定

4：Arduinoの初期化：電源投入時に1度だけ行う処理

6：【PIRPIN】に代入されている番号（＝ピン番号） 10番を入力待ちに設定

7：シリアルモニタを有効化

コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- **記述した内容（メイン処理）の補足です。**

```
10  void loop()
11  {
12    myPIR = digitalRead(PIRPIN);
13    Serial.print("PIR status: ");
14    Serial.println(myPIR);
15  }
16
```

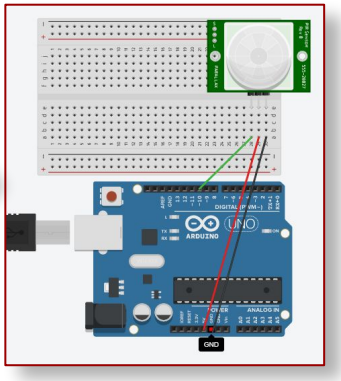
メイン処理

12：人感センサーの状態を読み込み → 変数【myPIR】に状態を代入

13：状態をシリアルモニタに【改行無し】で表示【 PIR status: 】

14：人感センサーの状態を【改行付き】で表示

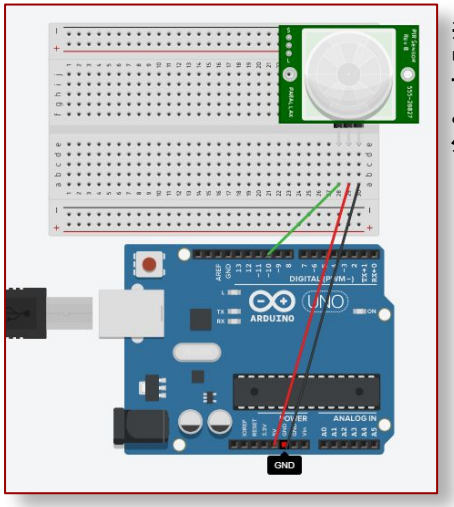
※熱源を検知していない場合は、【 0 】を、検知した場合は【 1 】をそれぞれ表示します。



コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- 記述の間違いが無いか確認を行い、プログラムを実行します。
- シリアルモニタを開き、**センサーの状態**【PIR status: 0】が表示されていたら成功です。

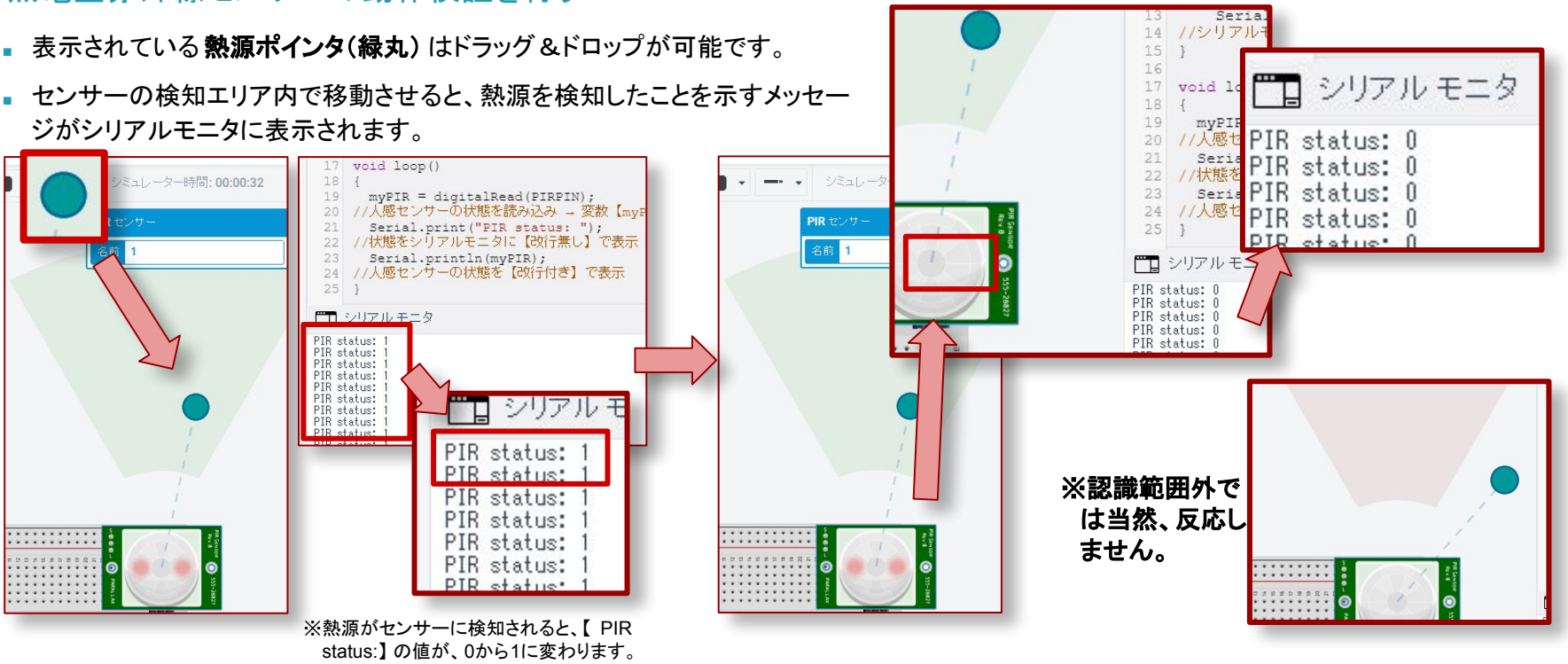


※ 動作中に、センサーをクリックすると、センサーの **検知可能エリア（薄い緑のエリア）**と、移動が可能な **熱源ポインタ（緑丸）**が表示されます。

コーディング：PIRセンサー（人感）

焦電型赤外線センサーの動作検証を行う

- 表示されている**熱源ポインタ（緑丸）**はドラッグ＆ドロップが可能です。
- センサーの検知エリア内で移動させると、熱源を検知したことを示すメッセージがシリアルモニタに表示されます。



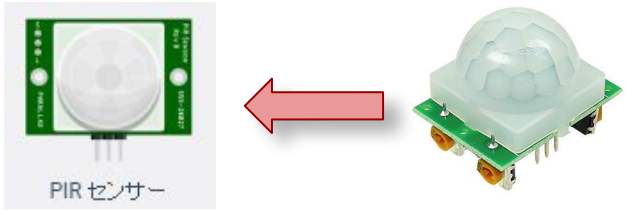
※熱源がセンサーに検知されると、【 PIR status:】の値が、0から1に変わります。

コーディング：PIRセンサー（人感）

赤外線検知型の人感センサー（ PIR）の用途について

- 住宅設備では、省エネタイプの外灯・防犯用ライトや足元灯などの照明器具に使用されたり、自動洗浄トイレの開閉蓋などにも使用されています。

- ウィークポイント
- 熱線をキャッチするため、通行する車に反応するなどの誤作動もあります。
- 外気温の高い夏には感知しにくいこともあり、特に体温に近い猛暑日などは検知しづらい場合があります。

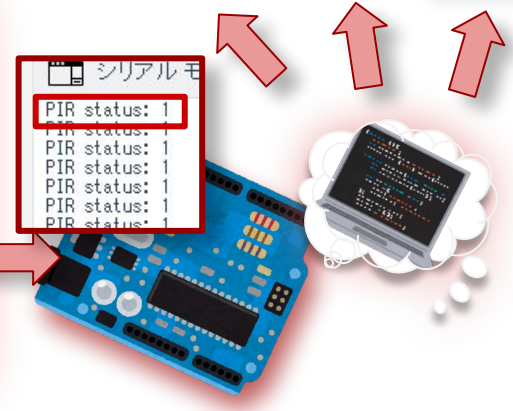


コーディング：PIRセンサー（人感）

赤外線検知型の人感センサー（PIR）の用途について

- 人感センサーの特徴に基づいた動作イメージです。
- 実際の動作イメージの一例として、図のような利用シーンが考えられます。

- 検知後の処理はどのようなことを行うか？
- 設置する場所や利用シーンによって、使用する部材が変わってくることがわかります。



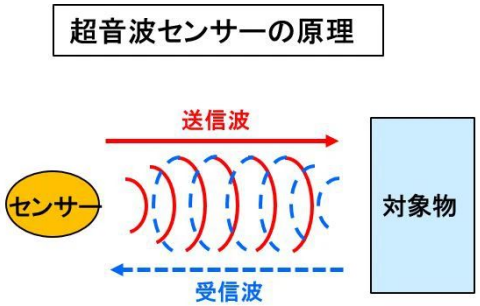
動作検証　超音波距離センサー編

超音波距離センサーの動作検証を行う

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

- 超音波距離センサーは、2つのスピーカーユニットを使い、一つのスピーカーで超音波を発信し、もう一つのスピーカーでをの音を拾い、超音波の送受信にかかる時間から距離を計測します。



コーディング：距離センサー補足（種類について）

距離センサーの種類を解説　閑話休題

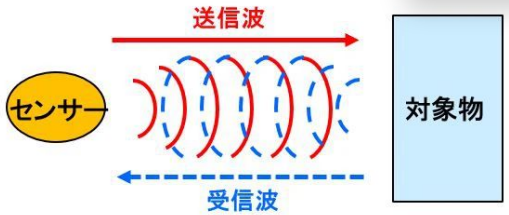
- 光学タイプ
  高性能ドローン：超音波距離センサーと併用してレーザー距離センサーを使用し、計測値を相互補完しながらより正確・精密な距離を計測します。

- 超音波タイプ
  自動車用途：バンパーなどに埋め込み、駐車時に障害物の距離が一定以下になると警告音で知らせるパーキングソナー等に利用されています。

- 基本的な動作原理は、音や光を発信し、対象物から反射してきた状態を受信し計測を行います。

  超音波式：発信から受信までの「時間」を計測することで対象物までの距離を測定します。

  光学式：発信したレーザー光が対象物表面で反射し、その反射光を受けるまでの光路を【三角測量】を用いて距離を測定します。

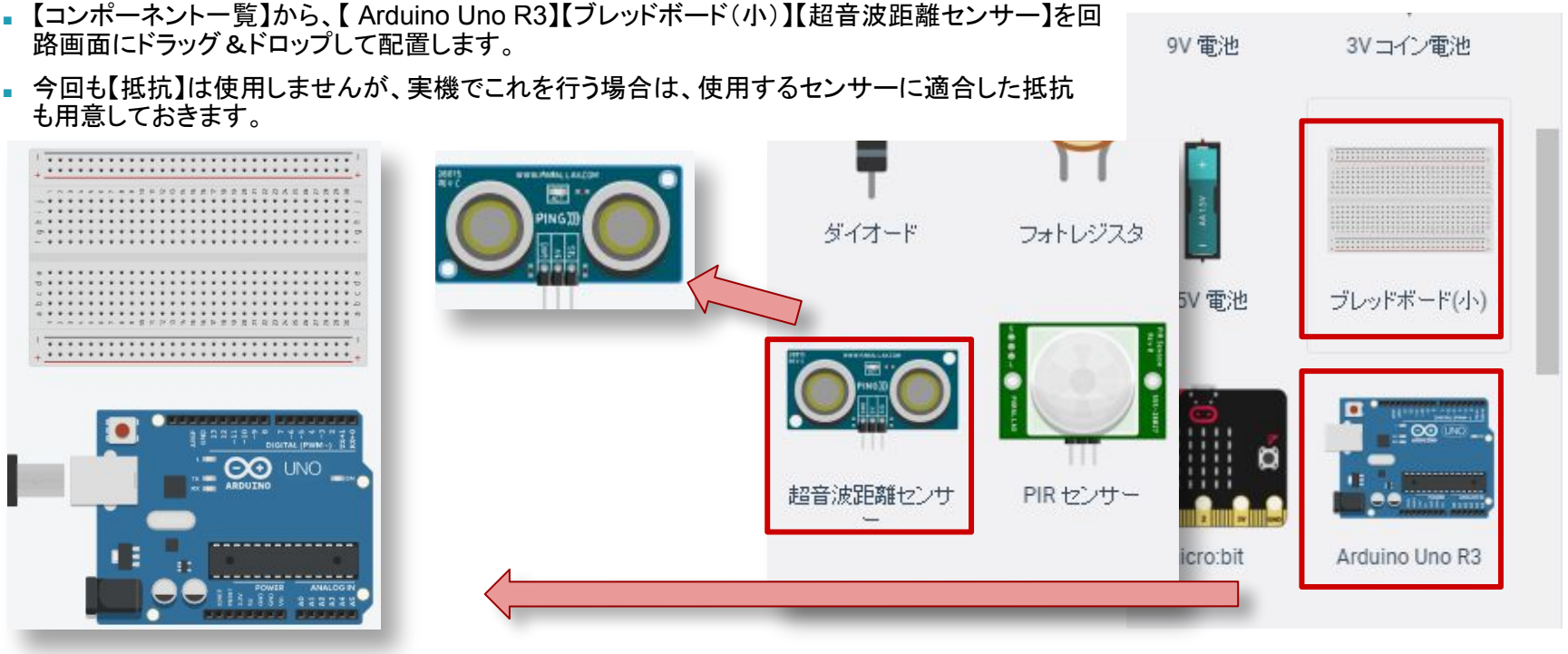


※動作原理

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

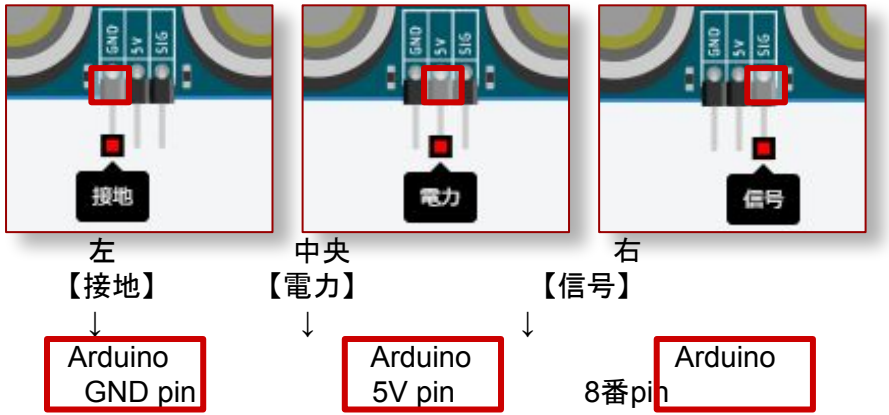
- 【コンポーネント一覧】から、【 Arduino Uno R3】【ブレッドボード（小）】【超音波距離センサー】を回路画面にドラッグ &ドロップして配置します。
- 今回も【抵抗】は使用しませんが、実機でこれを行う場合は、使用するセンサーに適合した抵抗も用意しておきます。

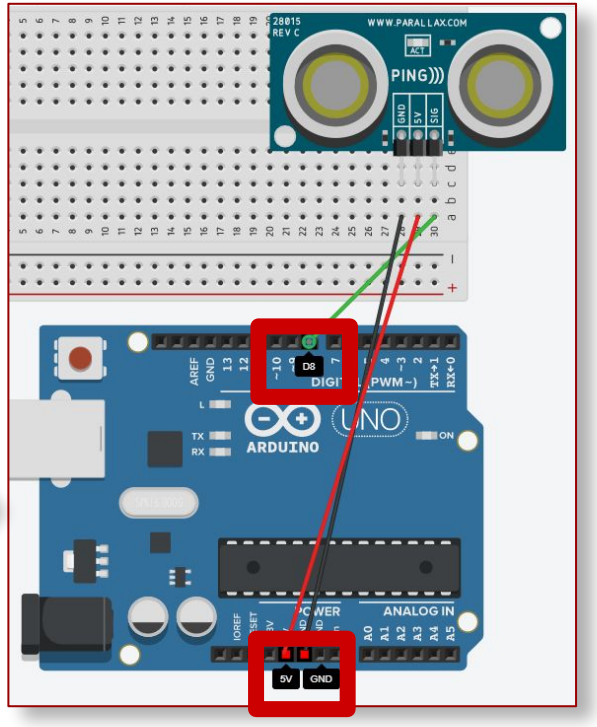


コーディング：距離センサー（超音波）

距離センサー（超音波）の動作検証を行う

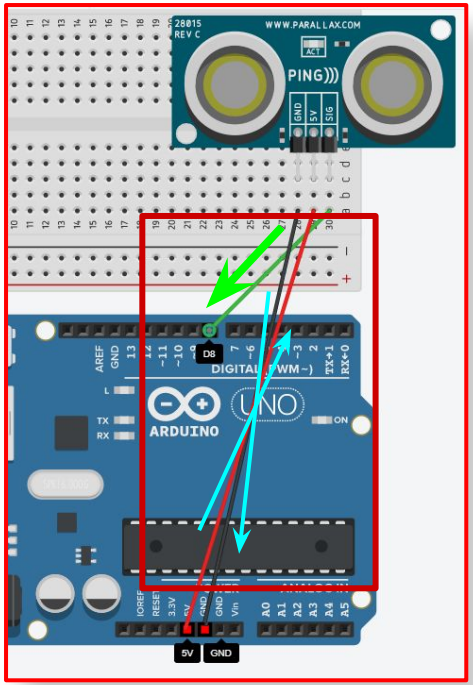
- 続いて配線です。
  前節のPIRセンサーとは違い、画面左から【 GND】（グランド）【電力】（ 5V）【信号】
  （Arduino側8番pin）になります。
- それぞれのpinをArduinoに配線しましょう。





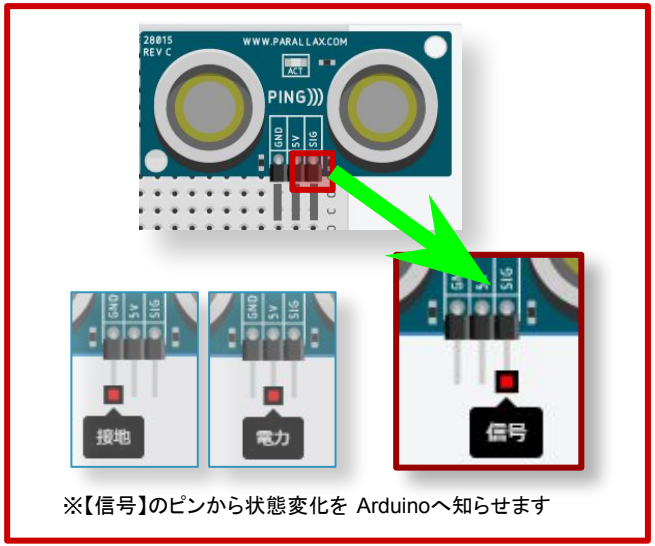
コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証



- 回路完成イメージです。
  前節と同様に、**電流**は**水色の矢印線** の通りに流れるのと併せ、センサーは **状態変化の検知** を**緑色の線**から信号を発信します。

- センサーのpin配列が人感センサーとは異なるため、【電力 +】【GND-】【信号】それぞれの配線が前節とは異なり交差します。
- 実際の回路作成もこのようなケースがありますので、配線の際は結線時のチェックは怠らないようにしましょう。



※【信号】のピンから状態変化を Arduinoへ知らせます

※配線の確認が終わり次第、コーディングに進みましょう。

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

```
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou)
{
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);

  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);

  return pulseIn(sncpin, HIGH);
}

void setup()
{
  pinMode(sncpin, INPUT);
  Serial.begin(9600);
}

void loop()
{
  cm = 0.01723 * oresonic(sncpin);

  Serial.print(cm);
  Serial.println("cm");
  delay(100);
}
```

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

```
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou)
{
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);

  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);

  return pulseIn(sncpin, HIGH);
}
```

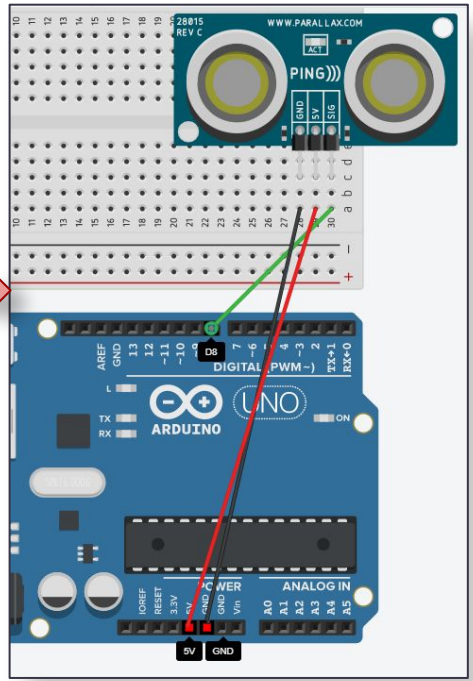
```
void setup()
{
  pinMode(sncpin, INPUT);
  Serial.begin(9600);
}

void loop()
{
  cm = 0.01723 * oresonic(sncpin);

  Serial.print(cm);
  Serial.println("cm");
  delay(100);
}
```

コーディング：距離センサー（超音波）

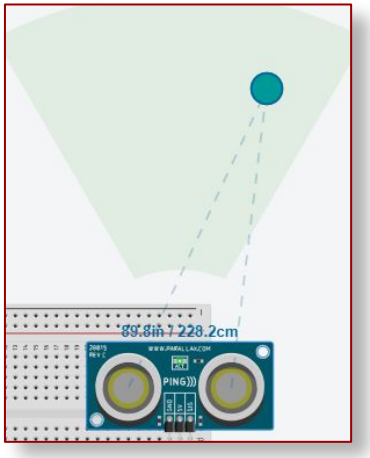
距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

- 前ページのテキストを順にコピー＆ペーストします。

```
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou)
{
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);

  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);

  return pulseIn(sncpin, HIGH);
}

void setup()
{
  pinMode(sncpin, INPUT);
  Serial.begin(9600);
}
```



- このプログラムは、センサーの検知エリアにいる物体の位置を、超音波で検知し距離を計測します。



コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

- **記述した内容の補足です。**

```
1  int cm = 0;
2  int sncpin = 8;
```

```
4  long oresonic(int Pin_Bangou)
5  {
6    pinMode(sncpin, OUTPUT);
7    digitalWrite(sncpin, LOW);
8    delayMicroseconds(2);
9
10   digitalWrite(sncpin, HIGH);
11   delayMicroseconds(10);
12   digitalWrite(sncpin, LOW);
13   pinMode(sncpin, INPUT);
14
15   return pulseIn(sncpin, HIGH);
16 }
```

初期設定

1：超音波センサーの距離計測用の変数設定
【cm】を設定＆初期化　※今回は、**単位をセンチ「cm」**を想定
※**変数名・計測単位** は任意に決めてもOKです。（**ミリ：mm** 等）

2：超音波センサーの信号を読み取るための配線（緑）用ピン番号の設定
※ここでは、変数**【sncpin】**デジタルピンの**8番**を指定

4：超音波センサーの制御用関数の設定　**※次ページで補足**

6：**【sncpin】**に代入されている番号（＝ピン番号）**8番**を**出力**待ちに設定
7：【sncpin】の初動をOFF
8：2マイクロ秒待機
※delayMicroseconds()関数でマイクロ秒（100万分の1秒）単位の制御が可能

10：超音波を発信・・・・・【 sncpin】をON
11：10マイクロ秒待機
12：超音波の発信を停止・・【 sncpin】をOFF
13：【sncpin】を入力待ちに変更

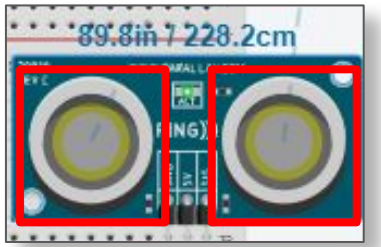
15：10マイクロ秒発信した音波（パルス）を【 sncpin】で検出

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

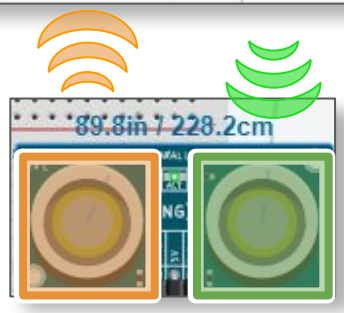
- **超音波センサー制御用関数の補足です。**

この形式の超音波センサーは、2つのスピーカー（兼マイク）を駆使して状態を検知しています。



一つは**発信用**でもう一つは**受信用**です。

発信した音波を受信した時間で距離を割り出し数値化します。
それを踏まえ、先述のコード内容は右図のような動作となります。




```
6    pinMode(sncpin, OUTPUT);
7    digitalWrite(sncpin, LOW);
8    delayMicroseconds(2);
9
10   digitalWrite(sncpin, HIGH);
11   delayMicroseconds(10);
12   digitalWrite(sncpin, LOW);
13   pinMode(sncpin, INPUT);
14
15   return pulseIn(sncpin, HIGH);
```

6：【sncpin】を**出力待ち**に設定
7：【sncpin】の初動をOFF
8：2マイクロ秒待機

10：超音波を発信・・・【 sncpin】をON
11：10マイクロ秒待機
12：超音波の発信を停止【 sncpin】をOFF
13：【sncpin】を入力待ちに変更

15：パルスを【 sncpin】で検出（受信）

- **上記の処理を12マイクロ秒の周期で行っています。**

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証

```
18  void setup()
19  {
20    pinMode(sncpin, INPUT);
21    Serial.begin(9600);
22  }
```

初期設定＃2

20：**【sncpin】を入力待ち**に設定

21：：シリアルモニタを有効化

```
24  void loop()
25  {
26    cm = 0.01723 * oresonic(sncpin);
27
28    Serial.print(cm);
29    Serial.println("cm");
30    delay(100);
31  }
```

メイン処理

26：計測距離を単位センチ（ cm）に調整して出力用の変数**【cm】**に代入

28：シリアルモニタに計測距離【 cm】を表示

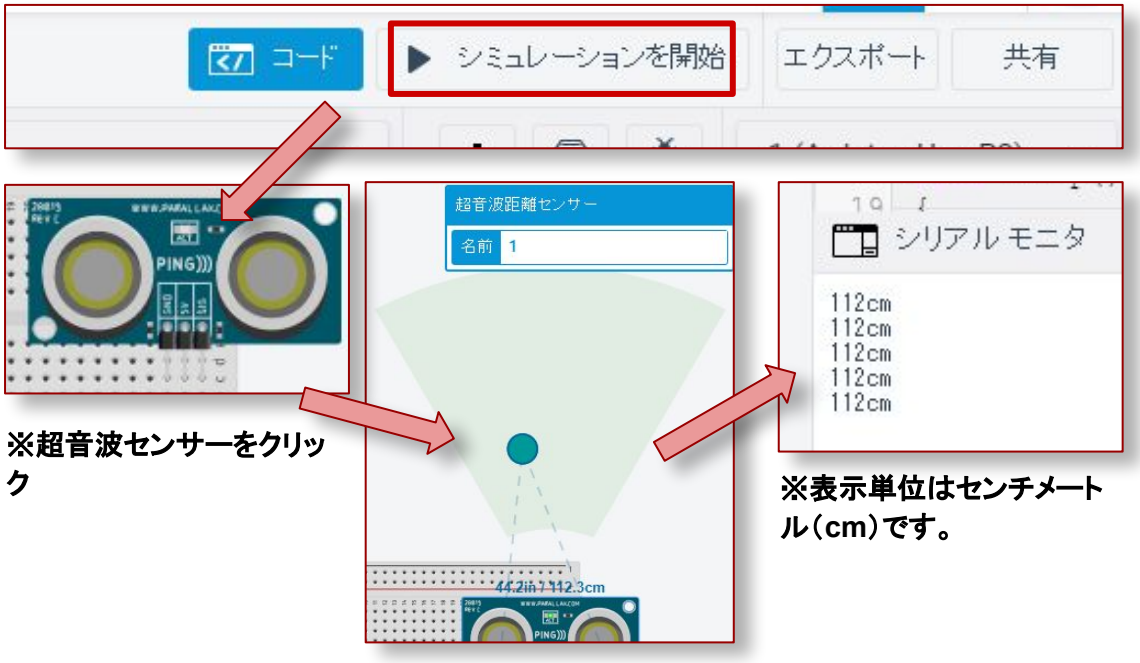
29：シリアルモニタに文字列「 cm」（単位）を表示

30：100ミリ秒待機

※②赤枠で囲まれている所を押下

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証



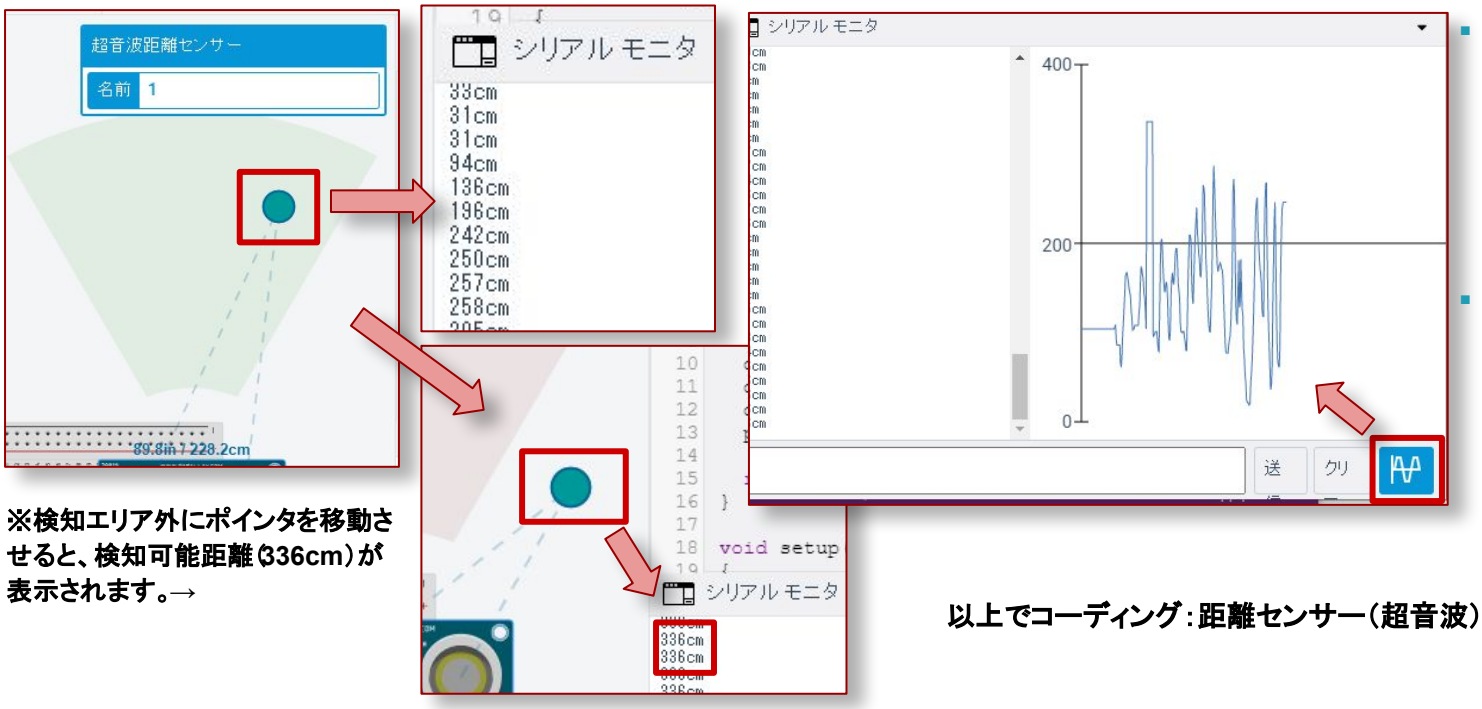
※超音波センサーをクリック

※表示単位はセンチメートル（cm）です。

- 【シミュレーションを開始】を行い、超音波センサーをクリックすると、**物体ポインタ（緑丸）**とセンサーの**検知可能エリア（薄緑）**が表示されます。
- 人感センサーの時と同様、表示されている**物体ポインタ（緑丸）**はドラッグ＆ドロップが可能です。
- センサーの検知エリア内でポインタを移動させると、センサーと物体間の距離がシリアルモニタに表示されます。

コーディング：距離センサー（超音波）

距離センサー（超音波）の特徴を解説を交え、検知する際の動作をプログラム作成を行いながら検証



※検知エリア外にポインタを移動させると、検知可能距離（336cm）が表示されます。→

- 画面右下のグラフアイコンを押下すると、計測結果をグラフ表示することが出来ます。時系列で移動範囲が推移します。
- 実際のArduino開発環境（Arduino IDE）でも同様の機能が実装されています。（シリアルプロッタ）

以上でコーディング：距離センサー（超音波）は完了です！

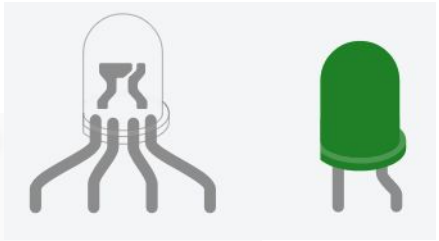
防犯用LEDライトを想定したプロトタイプ

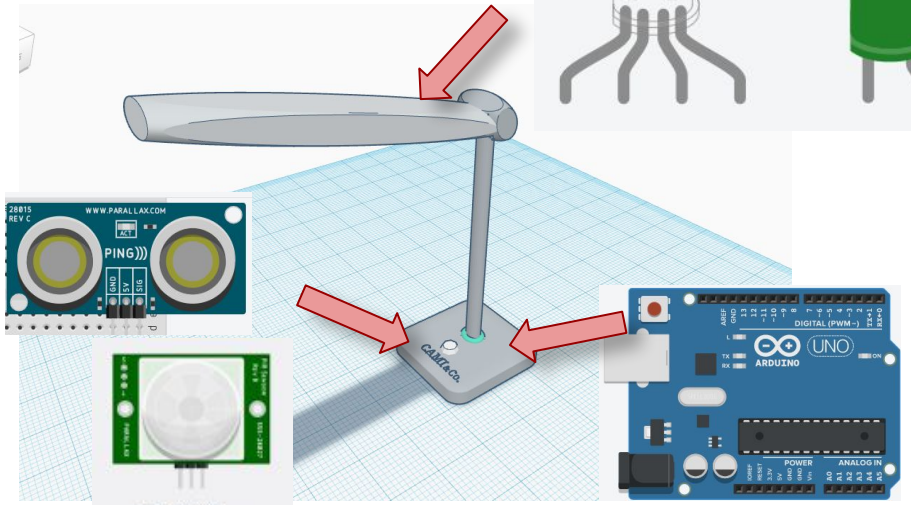
前段の各種部材の特性を踏まえ、防犯用LEDライトのプログラムを作成する

防犯用LEDライトの制作

各種部材の特性を踏まえ、LEDライトの動作を想定し、プログラムを実際に作成する

- 前節の作業を通して【LEDの点灯】（もしくは点滅）の動作が、各種センサーとの組み合わせ次第で、様々な要求を満たせることがお分かりいただけたかと思います。
- 本来であれば、その構想プロセスを踏襲しながらプロトタイプ製作を行うところなのですが、今回は距離センサー、もしくは人感センサーを用いた LEDスタンドの作成を行っていただきたいと思います。





プロトタイプ制作：作業の流れ

仕様の策定からプロトタイプの完成まで～

プロトタイプ制作 実践！： 防犯用LEDライトの制作

前段の各種部材の特性を踏まえ、防犯用 LEDライトのプログラムを作成する

- **防犯用LEDライトの制作**
  - **仕様の策定** → 作成する防犯用 LEDライトにどのような挙動をさせるか？を想定し、挙動に準じた部材の選定を行う
  - **フローチャートの作成** → 作成するLEDライトの挙動を想定し、使用部材の動きも踏まえたフローチャートを作成する
  - **コーディング：防犯用 LEDライト** → 作成したフローチャートを元にプログラム作成を行う
  - **動作検証・フィードバック** → 作成したプログラムが想定通りの挙動をしたか？の検証を行う

- **完成・作業完了！**

前段の各種部材の特性を踏まえ、防犯用 LEDライトの動作を想定し、プログラムを実際に作成する

プロトタイプ制作：仕様の策定を行う

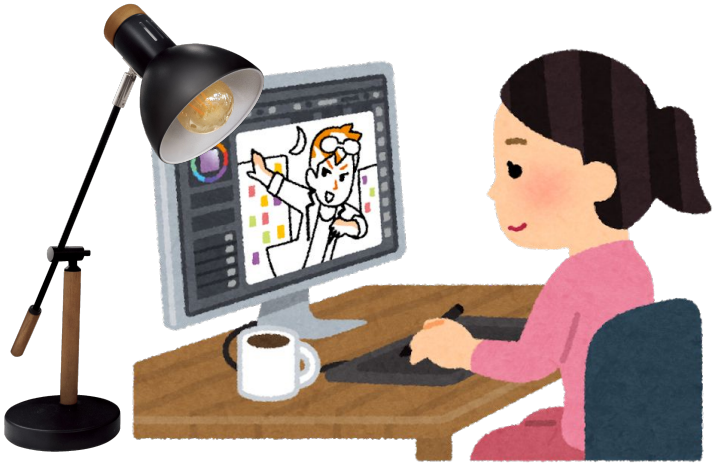
仕様に基づいた部材の選定

プロトタイプ制作 実践！： 仕様の策定

仕様の策定を行う：仕様に基づいた部材の選定

- **仕様の策定** → 作成するLEDライトにどのような動作をさせるか？

  要件に基づいた**場面を想定**し、動作に準じた部材の選定を行います

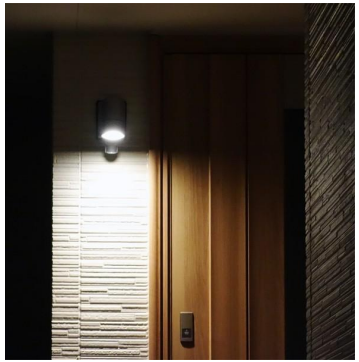


プロトタイプ制作 実践！： 仕様の策定

作成するLEDライトにどのような動作をさせるか？をイメージし、動作に則した部材の選定を行う

- 通常の場合、【要件】に基づいた【仕様】の策定を行います。
- これは、プロトタイプ製作における **最も重要なプロセス** であると言っても過言ではございません。

- 仕様の策定が重要なのは、それがそのまま製作物に反映されるため、ここを疎かにすると、最終的に完成するものが想定したものと大きく乖離してしまうことに繋がるからです。



- 例えば、**「誰かが近づいて来た時にライトが点灯する」**
  この一連の動作を実現させるためには通常、**【人感センサー】**を用いて作業を進めると思いますが、上記の条件に距離の概念が入ると。。。

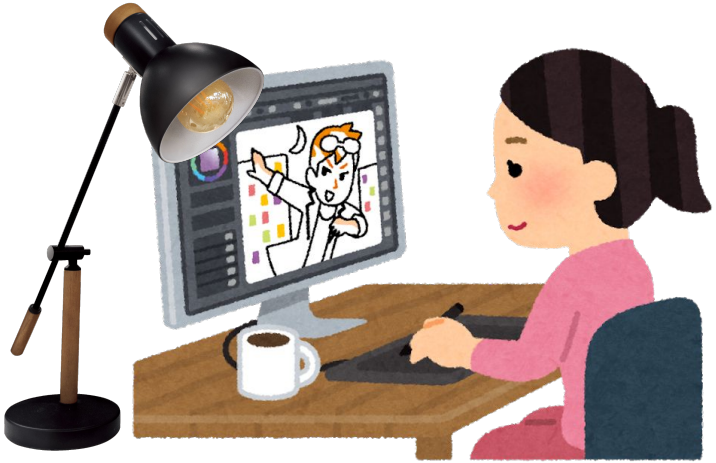
  「誰かが、5メートル以内に近づいて来た時にライトが点灯する」

となり、選定するセンサーも**【距離センサー】**のほうが適切と言うことになります。

プロトタイプ制作 実践！： 仕様の策定

仕様の策定を行う：仕様に基づいた部材の選定

- **仕様の策定** → **作成するLEDライトにどのような動作をさせるか？ → 最終的な目的は？**
  誰の希望？ クライアント？ クライアントの顧客？ 万人？



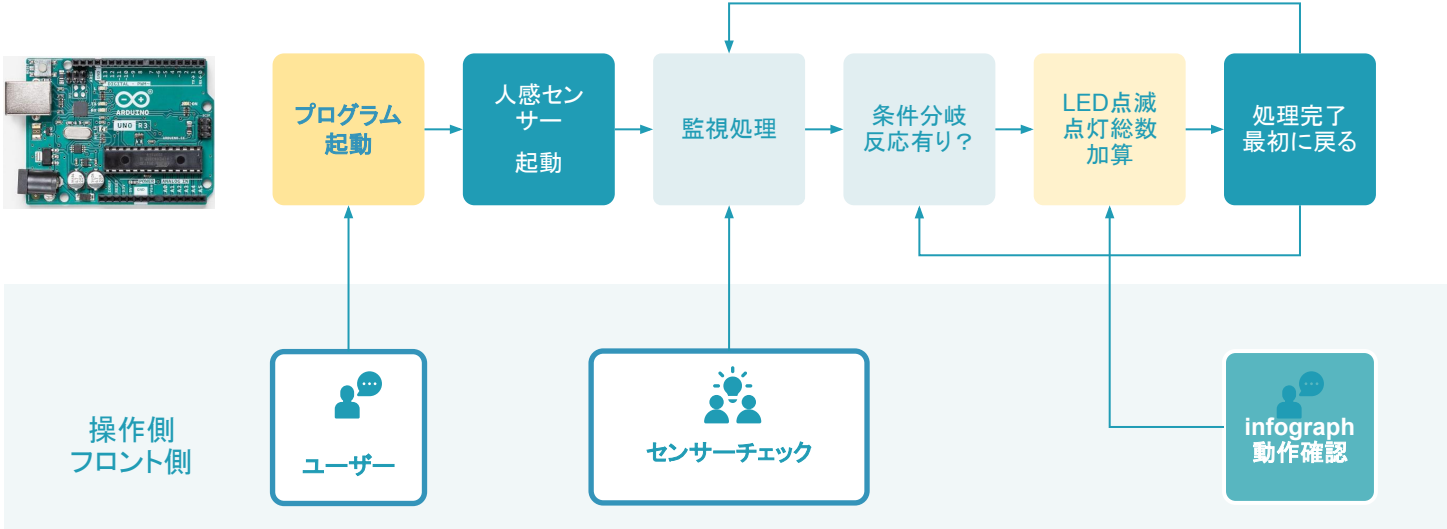
プロトタイプ制作：フローチャートの作成

防犯用LEDライトの動作環境を想定し、フローチャートを作成する

プロトタイプ制作 実践！：フローチャートの作成

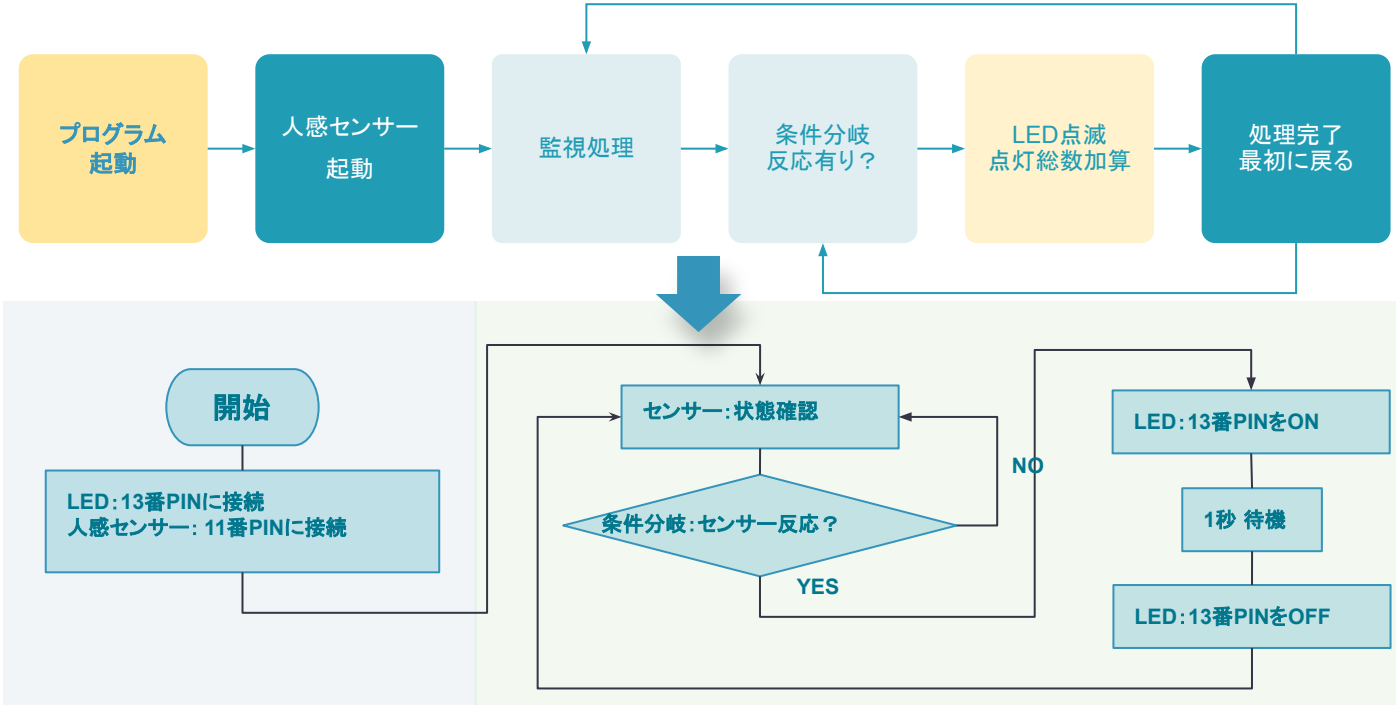
防犯用LEDライトの動作環境を想定し、フローチャートの作成を行う

- **フローチャートの作成**
  - → 作成する防犯用LEDライトの挙動を想定し、使用部材の動きも踏まえたフローチャートを作成します
  - → 利用者側の目線になった時に、どのよに動作するか？をイメージしながら動作環境を想定し、使用部材の動きも踏まえた各フローを策定しましょう



プロトタイプ制作 実践！：プログラムの作成

防犯用LEDライトの動作環境を想定し、フローチャートの作成を行う



- 想定される動作イメージが固まったら、よりプログラム形態に近い図【フローチャート】に落とし込みます。

※フローチャート作成用WEBアプリを利用すると、チャートをきれいにまとめることが出来ます。

【draw.io】

https://www.diagrams.net/

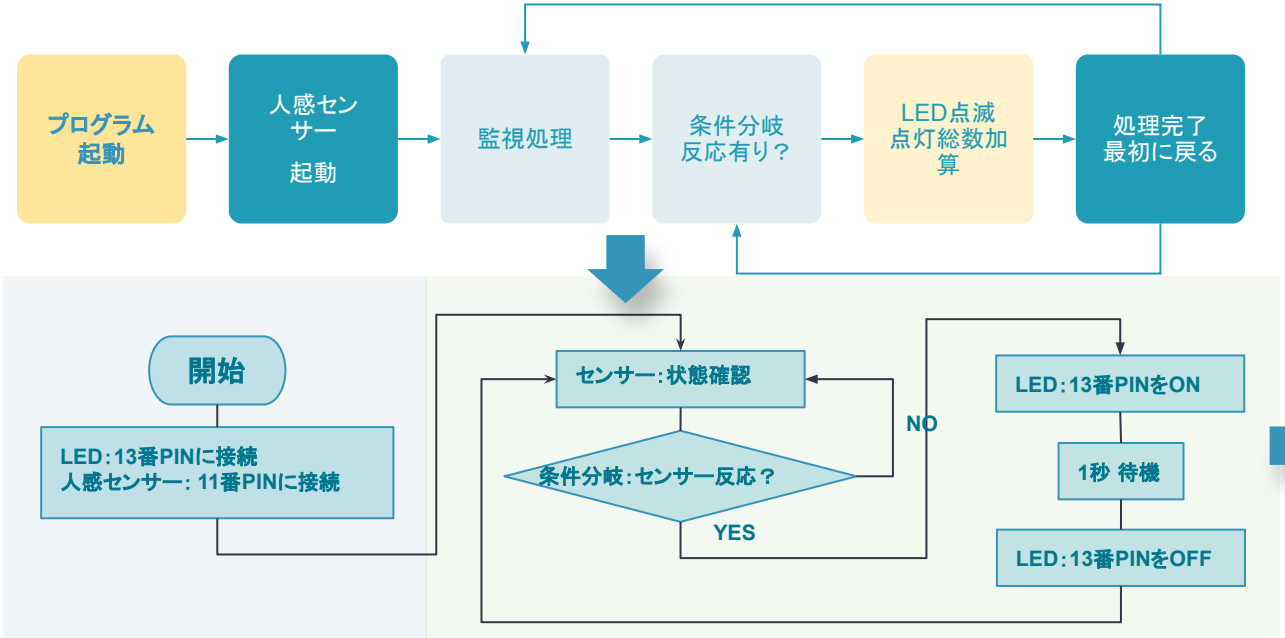
プロトタイプ制作：プログラムの作成

作成したフローチャートを元にプログラムを作成する

プロトタイプ制作 実践！：プログラムの作成

前段の各種部材の特性を踏まえ、防犯用 LEDライトのプログラムを作成する

- **コーディング：防犯用 LEDライト** → 作成したフローチャートを元にプログラム作成を行います。



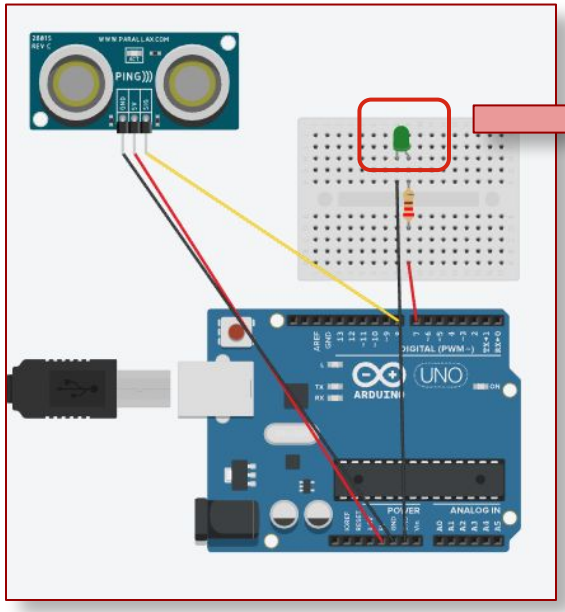
- → 各フロー毎にその処理を行うプログラムに落とし込みます。
- → 全てのフロー（処理）を落とし込んだら、一つまとめて完成となります。

```
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou)
{
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);

  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);

  return pulseIn(sncpin, HIGH);
}
```

コーディング：IoT 防犯用LEDライト

超音波距離センサーとLEDを配線したサンプル

- フローチャートに基づいて選定した部材（超音波距離センサー・単色　LED）を配置します。
- 配線も、あらかじめ策定を行った所定の箇所へ接続を行い、コードの書き込みを行いましょう。



→ 今回は、指定した距離（70cm～220cm）に近づいてきたらLEDを点灯させる設定。

```
int dltime = 1000;
int LED = 7;
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou)
{
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);

  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);

  return pulseIn(sncpin, HIGH);
}

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(sncpin, INPUT);
  Serial.begin(9600);
}

void loop()
{
  cm = 0.01723 * oresonic(sncpin);
  Serial.print(cm);
  Serial.println("cm");

  if (cm < 90){
    digitalWrite(LED, HIGH);
    Serial.println("LED : ON!");
  }
  else {
    digitalWrite(LED, LOW);
    Serial.println("LED : OFF!");
  }
}
```

コードディング用：テキスト

プログラムの作成を行うコピペ用コピペ用サンプルコード

```
int dltime = 1000;
int LED = 7;
int cm = 0;
int sncpin = 8;

long oresonic(int Pin_Bangou) {
  pinMode(sncpin, OUTPUT);
  digitalWrite(sncpin, LOW);
  delayMicroseconds(2);
  digitalWrite(sncpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sncpin, LOW);
  pinMode(sncpin, INPUT);
  return pulseIn(sncpin, HIGH);
}
```

```
void setup() {
  pinMode(LED, OUTPUT);
  pinMode(sncpin, INPUT);
  Serial.begin(9600);
}

void loop() {
  cm = 0.01723 * oresonic(sncpin);
  Serial.print(cm);
  Serial.println("cm");
  if (cm > 70 && cm < 220){
    digitalWrite(LED, HIGH);
    Serial.println("LED : ON!");
  } else {
    digitalWrite(LED, LOW);
    Serial.println("LED : OFF!");
  }
}
```

プロトタイプ制作：動作検証・フィードバック

作成したフローチャートを元にプログラムを作成する

プロトタイプ制作 実践！：動作検証・フィードバック

前作成したプログラムが想定通りの動作を行っているか？を検証する

❑ 動作検証

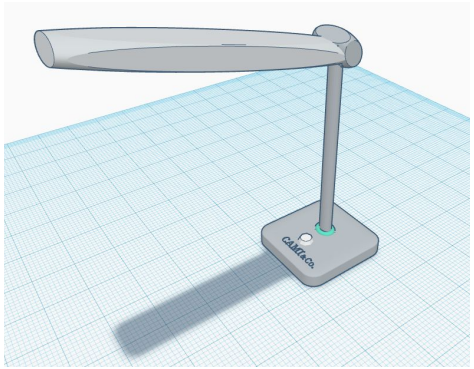
- 作成したプログラムが想定通りの動作をしたか？の検証を行います
- 動作の検証にあたっては、想定される動作に基づいたチェック表を作成すると作業を円滑に行うことが出来ます。
- 表を作成することによって、複数人でのチェックを行うことも容易となり、併せて様々な角度からの検証により思わぬ【気づき】も得られることから、フィードバックがより精度の高いものになります。

	A	B	C	D
1				
2		基板導通・ハード（回路）チェック項目	チェック	備考
3	1	各電源とGND間のショートがないか確認	☐	
4	1-1_	A0（Arduino）ー R1（220Ω 抵抗：1）間	☐	
5	1-2_	R1ーLED1（緑）間	☐	
6	1-3_	LED1－GND（Arduino）間	☐	
7				
8		ソフト・ファームウェア（スケッチ）動作チェック項目		
9	2	スケッチの動作確認 【初期設定】	☐	
10	2-1_	Arduino初期設定	☐	
11	2-2_	シリアルコンソール アクティブ化	☐	
12	2-3_	LED1 ★番pin アクティブ化	☐	A0を【出力用】
13	2-4_	LED1 A0番pin LOW	☐	
14				
15	3	スケッチの動作確認 【メイン】	☐	
16	3-1_	LED1 A0番pin High	☐	
17	3-2_	シリアルコンソールに文字列表示	☐	Hello world!
18	3-3_	2000ms（2秒間）処理停止	☐	
19	3-4_	LED1 A0番pin LOW	☐	
20	3-5_	2000ms（2秒間）処理停止	☐	
21				

プロトタイプ制作 実践！：動作検証・フィードバック

前作成したプログラムが想定通りの動作を行っているか？を検証する

- **フィードバック** → 動作検証時にを行います
  - → フィードバックに基づいた修正を行い動作検証を繰り返し行います。
  - → 無事に思い通りの動作が行われていればプロトタイプの完成です！



チェック	備考	修正・変更内容
☑		
☑		
☐		
☑		
☑		
☑		
☑		
☑	A0を【出力用】	
☑		
☐		
☑		
☐	Hello world!	文字列の変更
☑		
☑		
☐		停止時間を1000msに

補足資料:TinkerCad　アカウントの作成

WEBアプリ【Tinker Cad】アカウントの作成

WEBアプリ【Tinker Cad】アカウントの作成

今回のワークショップでは、マイコン基盤や電子部品の実物を用いての作業はございません。

その代替として、無料で利用可能な WEBアプリケーション**「TinkerCad」**を使用して、作業を行います。

登録用のメールアドレスをご用意いただき、下記 URLにアクセスしてアカウントの作成をお願いします。

登録用URL　https://www.tinkercad.com/

WEBアプリ【Tinker Cad】アカウントの作成

下記URLにアクセスしてアカウントの作成をお願いします。

登録用URL　https://www.tinkercad.com/

WEBアプリ【Tinker Cad】アカウントの作成

メールアドレス（gmail AppleIDとの連携でも可能です。）

WEBアプリ【Tinker Cad】アカウントの作成

下記画面（赤枠）が表示されたら準備完了です。