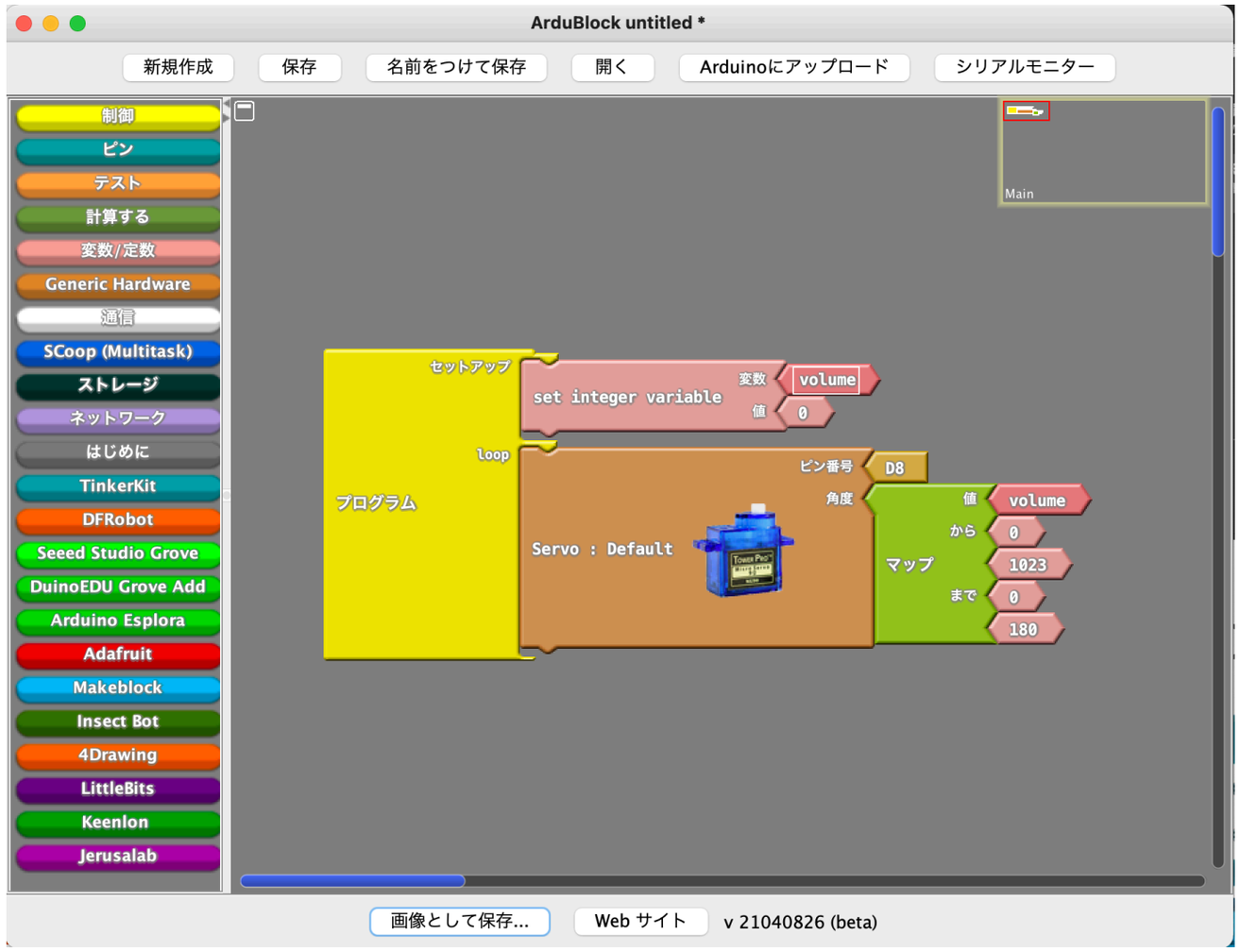
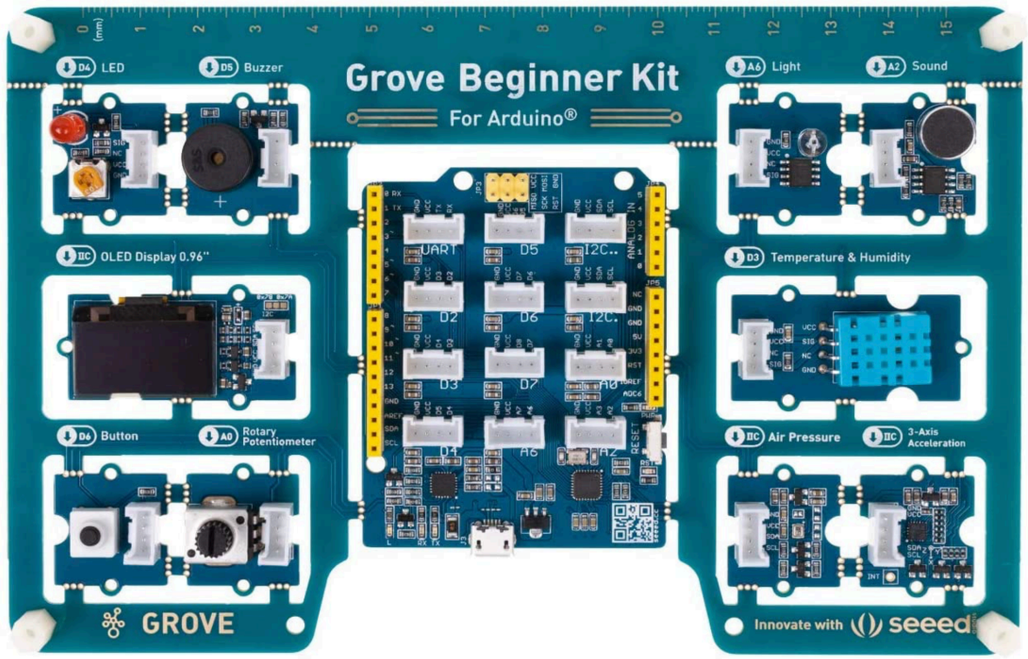
ArduinoとArduBlockを使った学習





Takerusoftware 2021.12

・一番簡単なArduinoの学習方法の１つで学んでみよう

Arduinoの学習というと、ArduinoIDEでサンプルプログラムを読み出して１つづつ実行させて確認！っていう方法を行うのが一般的だけど、キーボード操作に慣れてないとかC言語の文法がわからない場合には「難しい！」って感じちゃう人が多いとおもう。

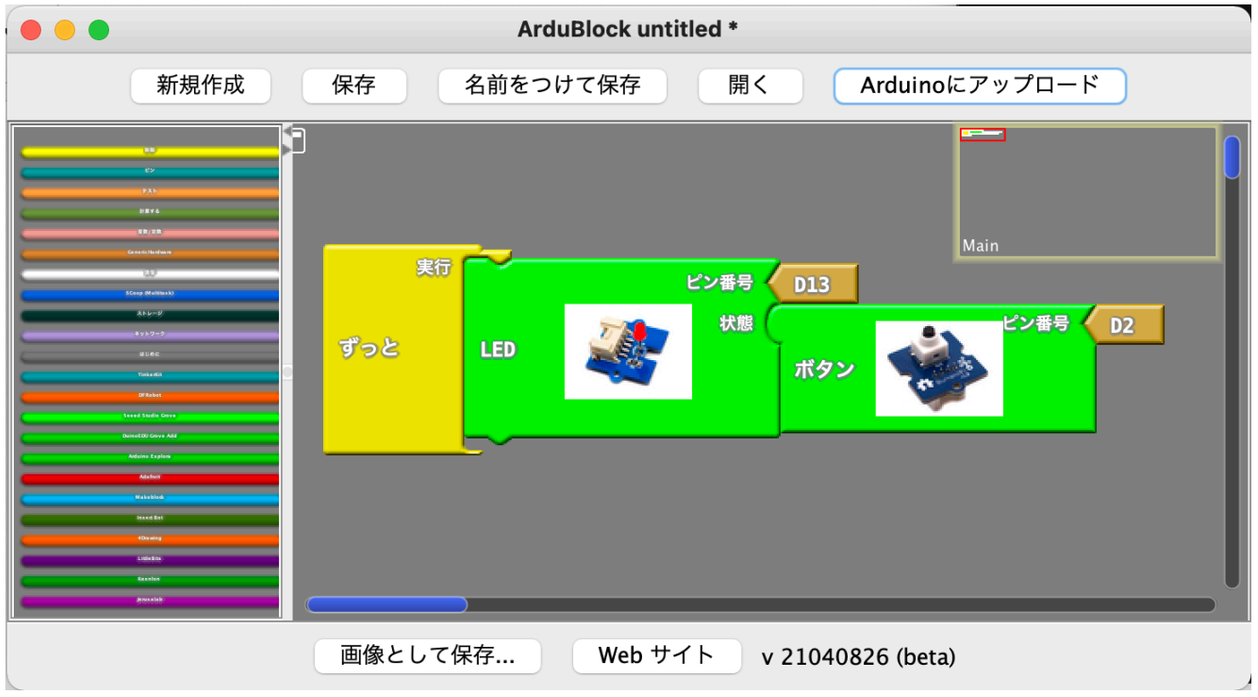
ちなみにArduinoIDEってこんな画面ね。

でも、キーボードをほとんど触らなくて良くて、C言語の文法がわからなくてもプログラミングが出来たら楽だと思わないかい？

そんなことができるのが**ArduBlock**という開発環境なんだ。

ArduBlockは使いたい機能パーツを画面に並べるだけでArduinoを動かすことができる。

例えば、スイッチが押されたらLEDをつけるプログラムはこんな感じ。



これは、D2に繋がったボタンの状態をそのままD13に繋がったLEDに反映する。

D2がONならD13もON。D2がOFFならD13もOFF。

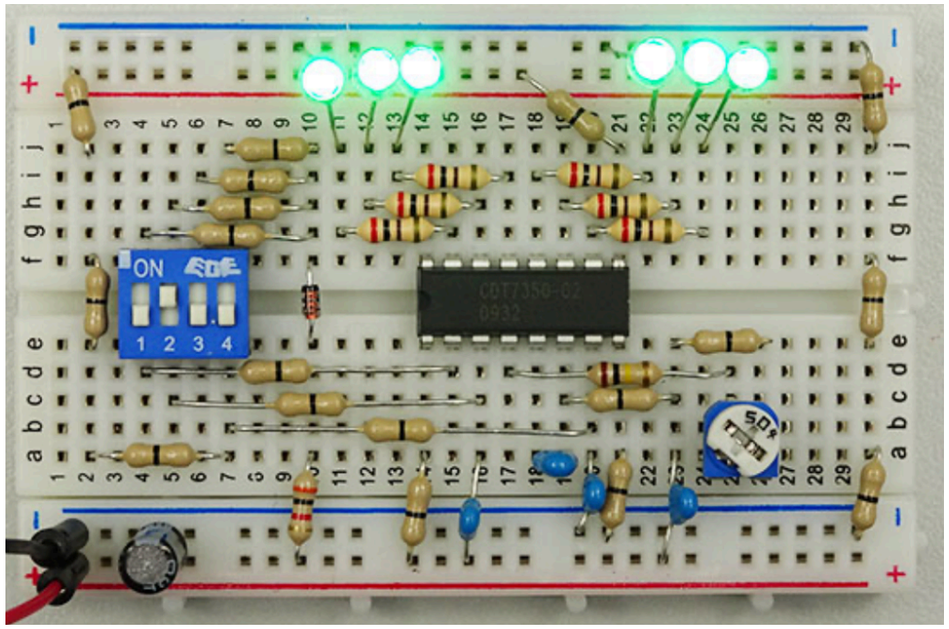
つまり、ボタンとLEDが連動して動くんだね。

そしてなんと、左右のプログラムは全く同じものなんだ。

ArduinoIDEで数分かかるプログラムが数１０秒で出来ちゃうのが**ArduBlock**という開発言語なんだよ。

・電子回路の問題をどうしょう？

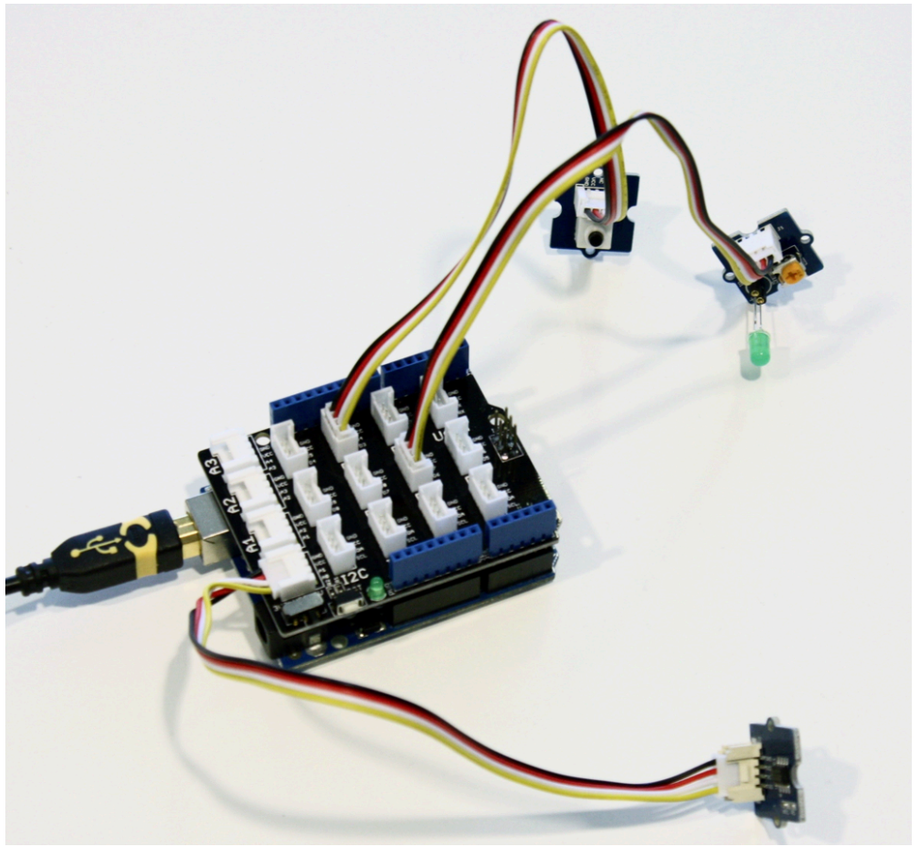
　Ardublockを使うと簡単にプログラムが作れる。
でも電子回路はどうしよう？
　電子回路の学習というと一般的なのはブレッドボードに電子部品を挿して回路を作るタイプ。
　こんな感じだね。



でもこれってすごく手間がかかる。
部品を挿す場所を１列でも間違えるとちゃんと動かないし、接触不良でも動かない。
回路を作り直す時も一旦バラバラにしないとダメ。
こんなめんどくさいものを使ってたら学習意欲も落ちちゃうね。
　でも大丈夫。Arduinoには簡単に電子回路を作れるアイテムがある！

・これがGroveシステムだ！

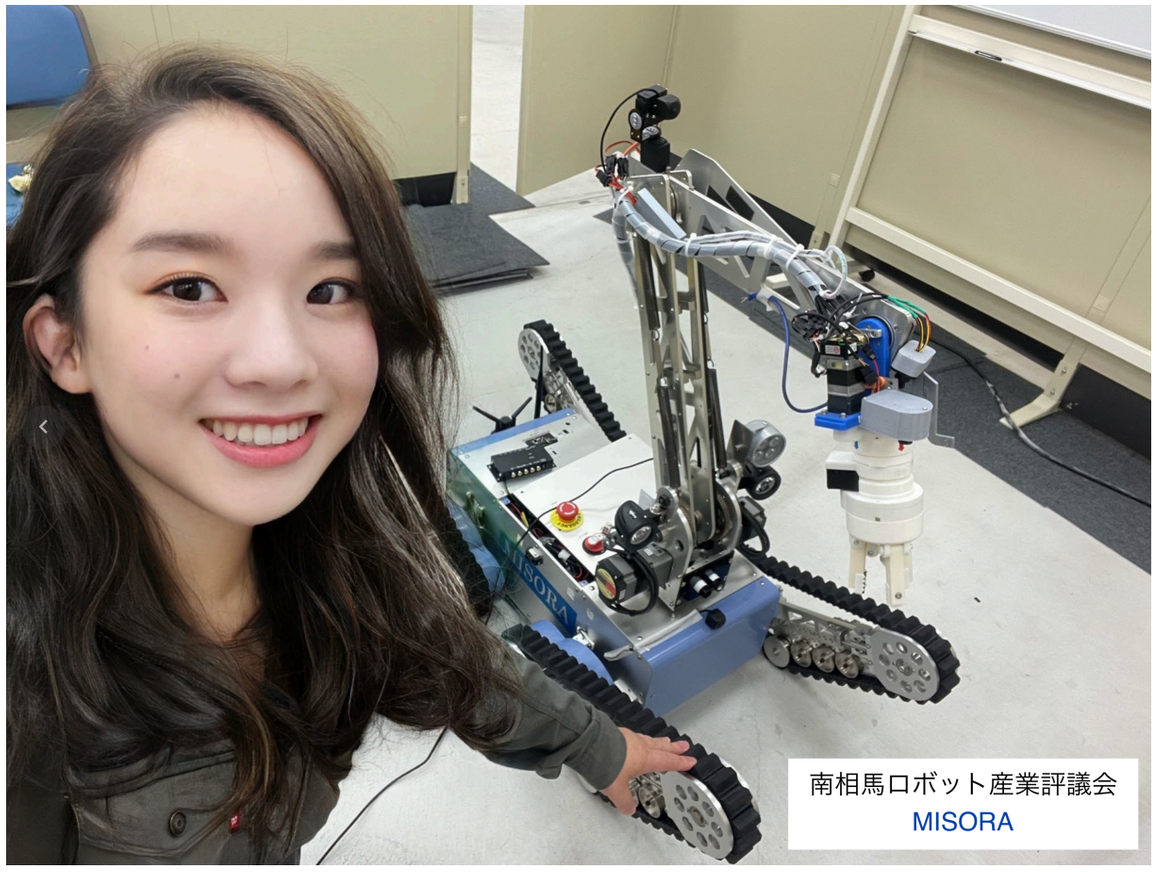
　SeedStudioという会社で売っているGroveシステムはたった１本の配線を電子部品とArduinoに挿すだけで複雑な電子回路を作ることができるんだ。



配線には４本の電線が繋がってて、それぞれ電源(5V)、GND(0V)、信号線1、信号線2になる。
その配線をArduinoに差したGroveベースシールドのコネクタに挿すことでスイッチやLED、サーボモータやいろんなセンサを持った電子回路が出来上がるんだ。

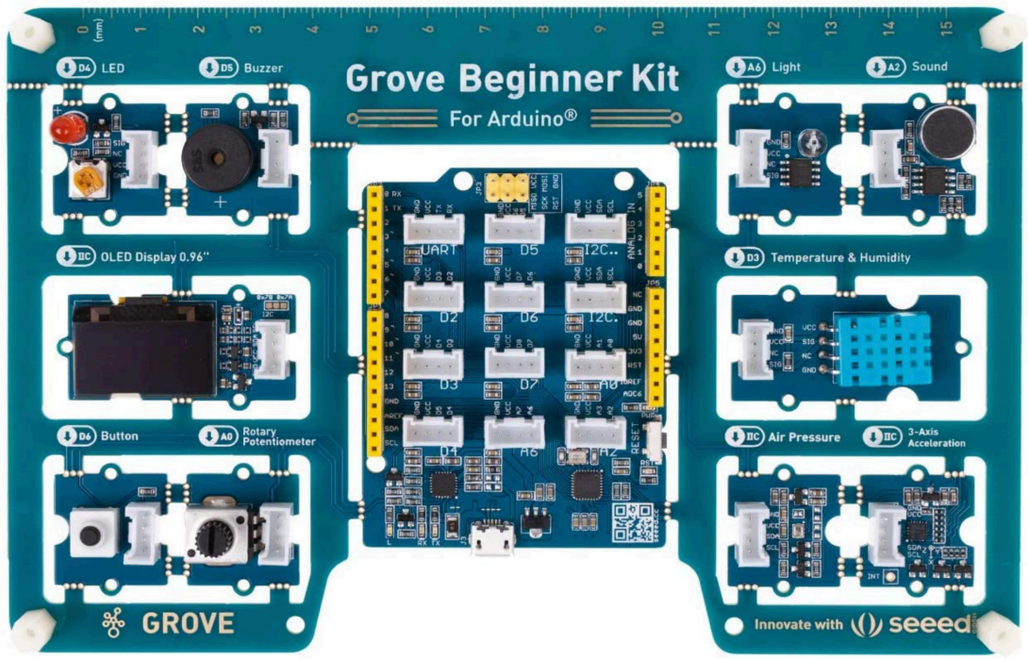
南相馬ロボット産業評議会で作成したクローラ型ロボット「**MISORA**」もこのArduinoとGroveシステムを使って動いている。

サーボモータが14個、センサーが16個、液晶表示器に、カメラが６台ついているけどこのくらいのシステムなら充分作ることができるんだね。



今回の学習ではこの配線さえも要らない（厳密には基板上で既に配線がされている）Grove Beginner Kit を使うよ。

このキットには、LED、ブザー、液晶表示器、ボタンSW、ポテンショナ（ボリューム）、光センサ、音センサ、温度・湿度センサ、気圧センサ、ジャイロセンサーなどがすぐに使える



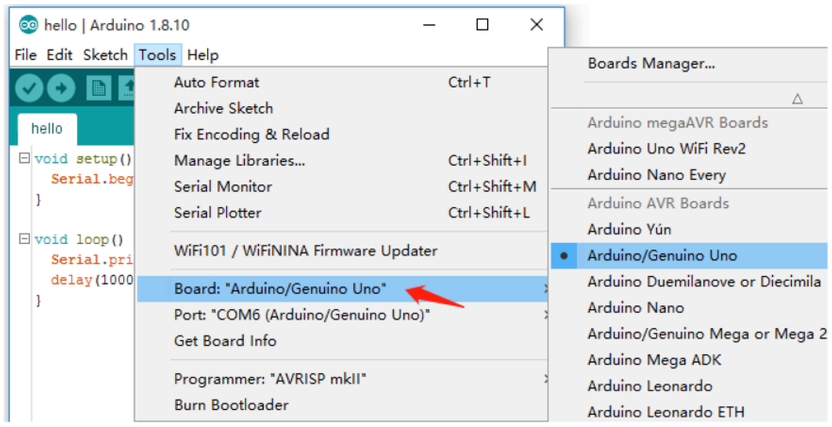
状態で揃っているんだ。

なので誰でも配線間違いなどなく簡単にArduinoを使った電子制御の実験を行うことができるよ。

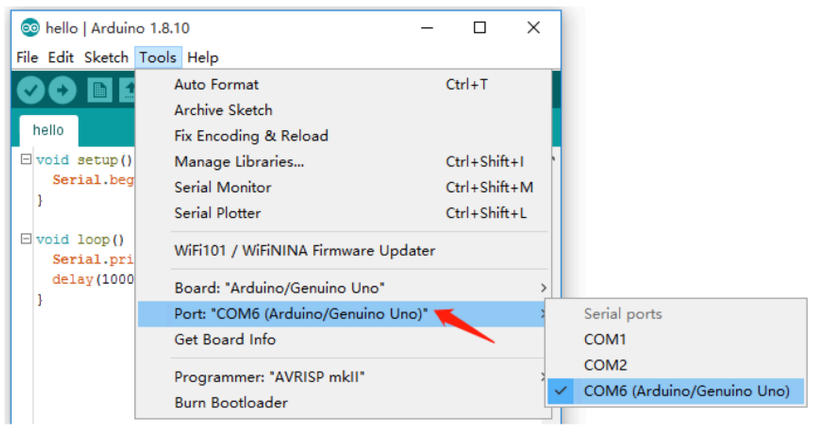
・ArduBlockを起動してみよう！

　既にみんなのパソコンにはArduBlockがイントールされていると思う。
　起動の仕方をMacの画面で説明するけどWindowsPCでも同じだよ。

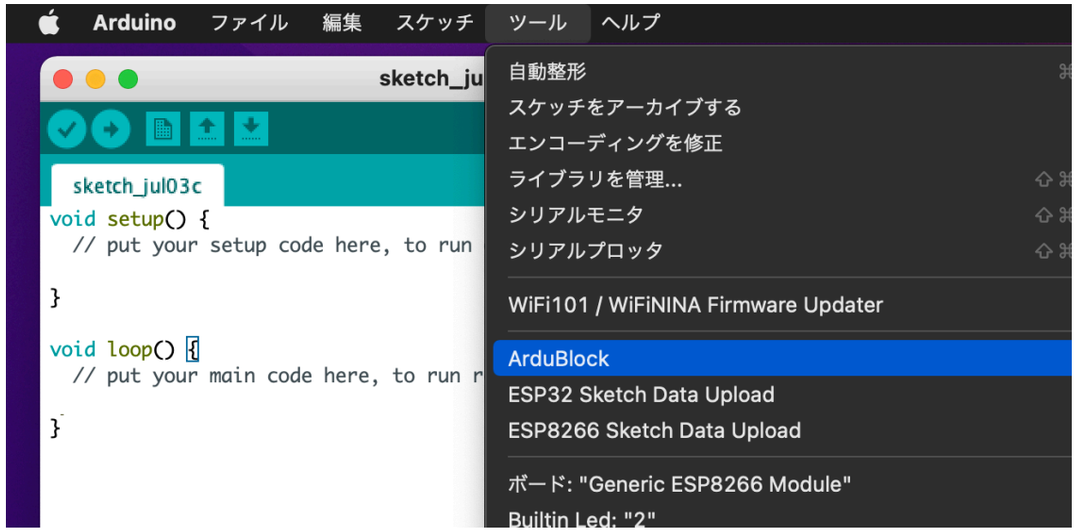
1）ArduinoIDEを起動する。
2）ツールの中にあるボードがArduno/Genuino Unoになっていることを確認。



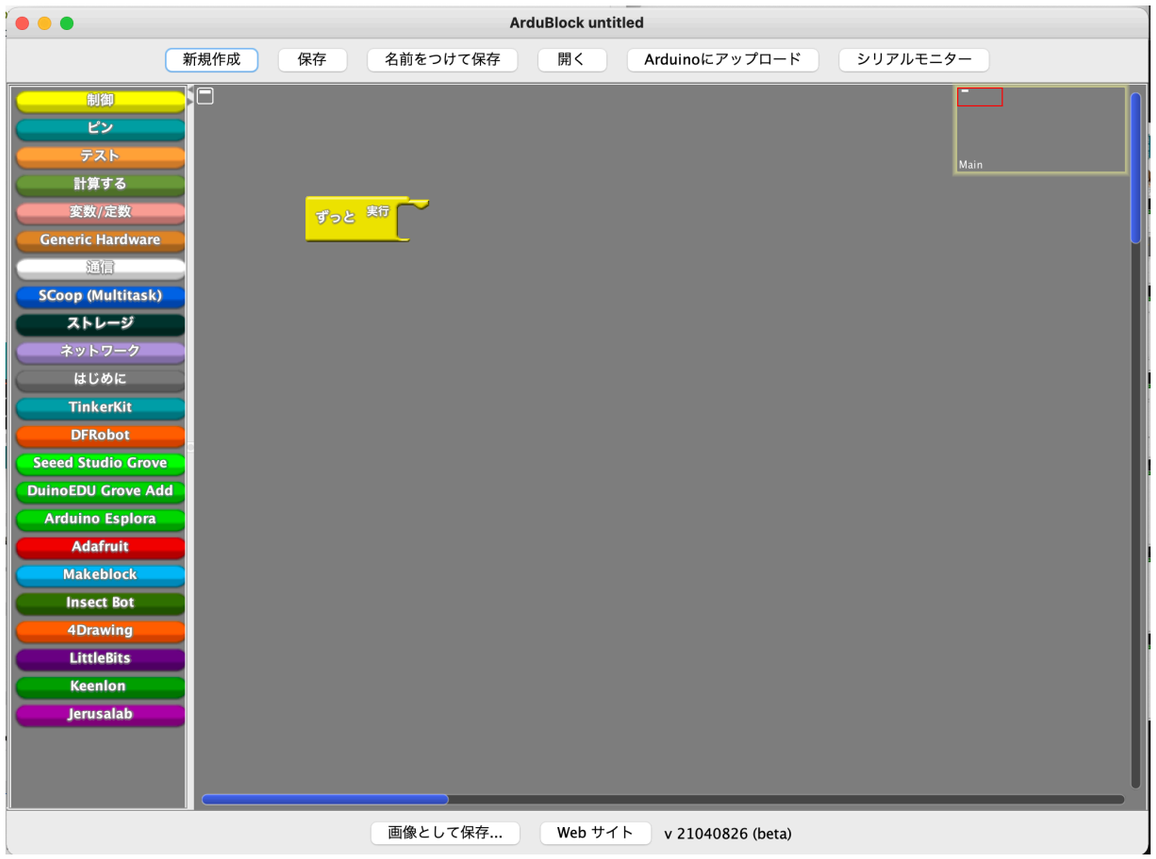
3）通信ポートがArduinoの文字があるポートになっていることを確認。



4)ツールからArduBlockをクリックする。

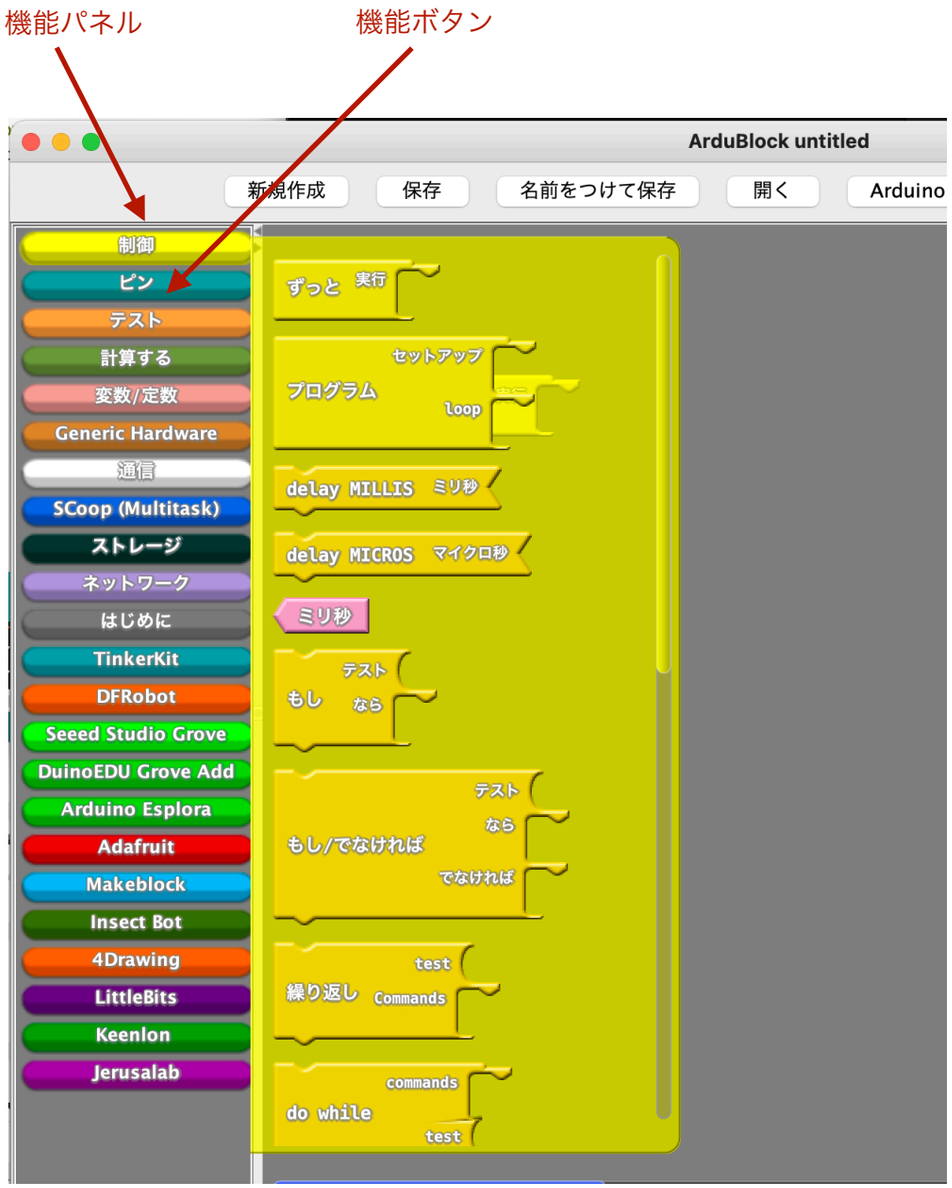


5）ArduBlockの画面が表示される。



・ArduBlockの基本操作をマスターしよう

　使い方は簡単。
まずは、機能パネルを開くためには、使いたい機能のボタンをクリックするだけ。
もう一度クリックすると閉じます。



開いてから使いたい部品をワークエリアにドラッグ&ドロップ。

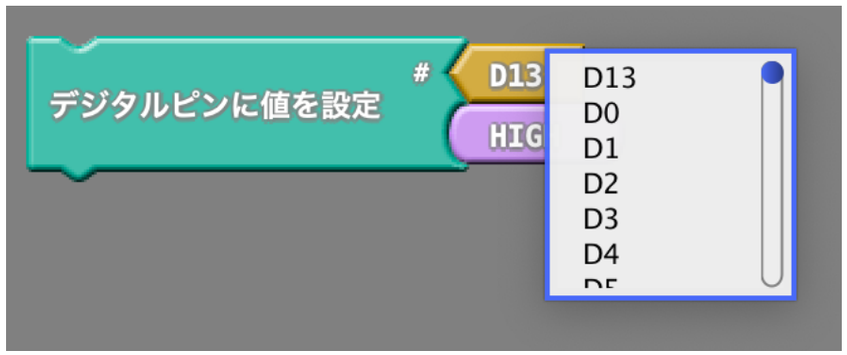
ワークエリアに部品が置かれた！

部品の削除は、消したい部品を機能パネル側にドラッグ＆ドロップする。

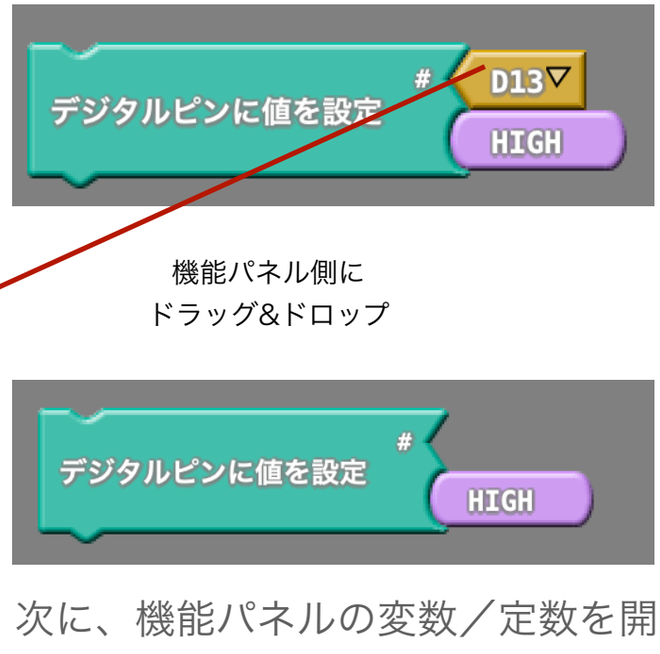
次に、機能パネルのピンから「デジタルピンに値を設定」をワークパネルに配置する。

D13の矢印のところにマウスカーソルを当てると下▲ボタンが表示される。その▲をクリックすると・・・

こんな感じで選択候補（この場合はピン番号）が出てくるので適当なものをクリックし選択する。



ピン番号や信号なども「部品」の１つなので機能パネル側にドラッグ&ドロップすると削除することができる。



機能パネル側に
ドラッグ&ドロップ

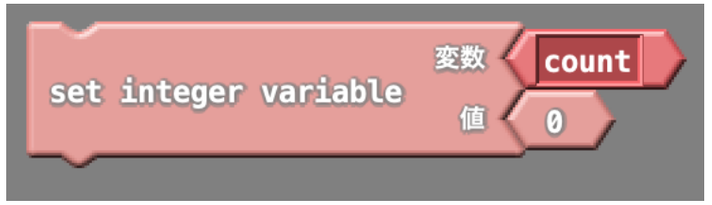
D13が消えた!

次に、機能パネルの変数／定数を開き、set integer variableをワーク画面に配置する。

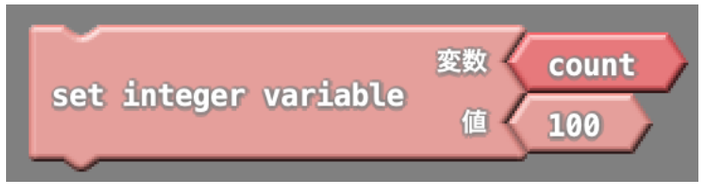
クリックしても下▲が出ないものは、直接キーボードで入力ができる。
この場合、Integerという名前をcountと変更するには、

Integerをクリックし、選択状態にする。

そのまま、countとキーボードから入力し、最後にEnterキーを押す。



同じように値も変更できる。値を100に変えてみよう！



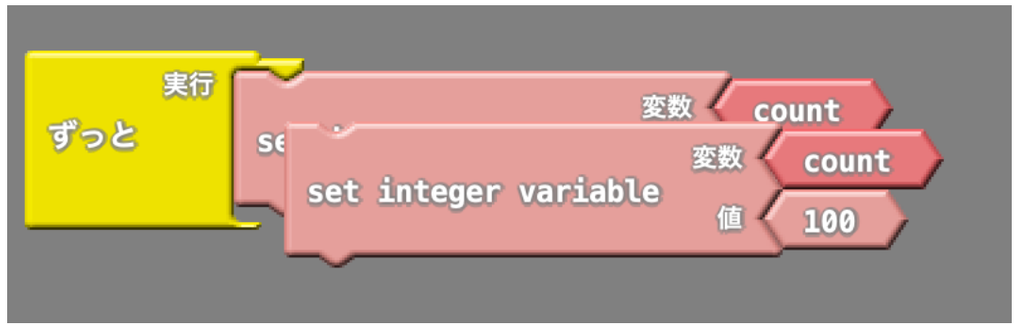
部品同士を嵌めてみよう。
先程のset integer variableをずっとに嵌めてみよう。
やり方は、set integer variableをずっとにドラッグ&ドロップする。

「カチッ」という音とともに嵌まる。

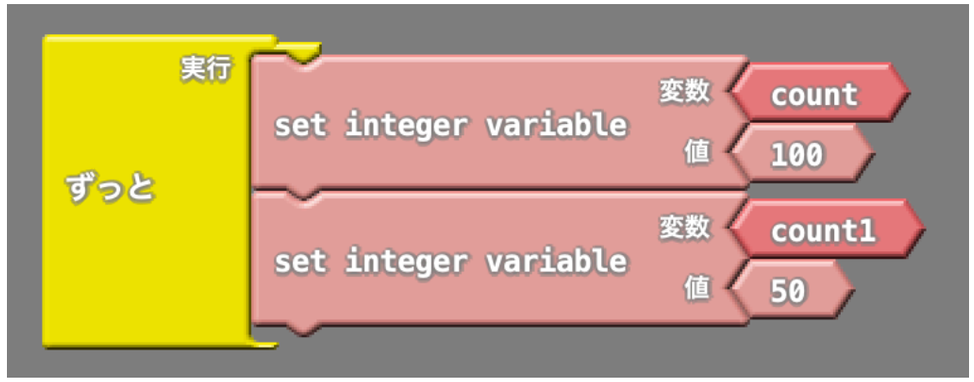
逆の動作をすると外すこともできる。やってみよう！

部品を右クリックすると複製を作ることができる。

複製をクリックすると、こんな感じに複製された部品が出てくる。



これを下の図のように直してみよう！



ちなみに上と下のset integer variable をそれぞれ複製すると違った結果になるよ。
確認してみよう！

・Arduinoからパソコンに文字を送ってみよう

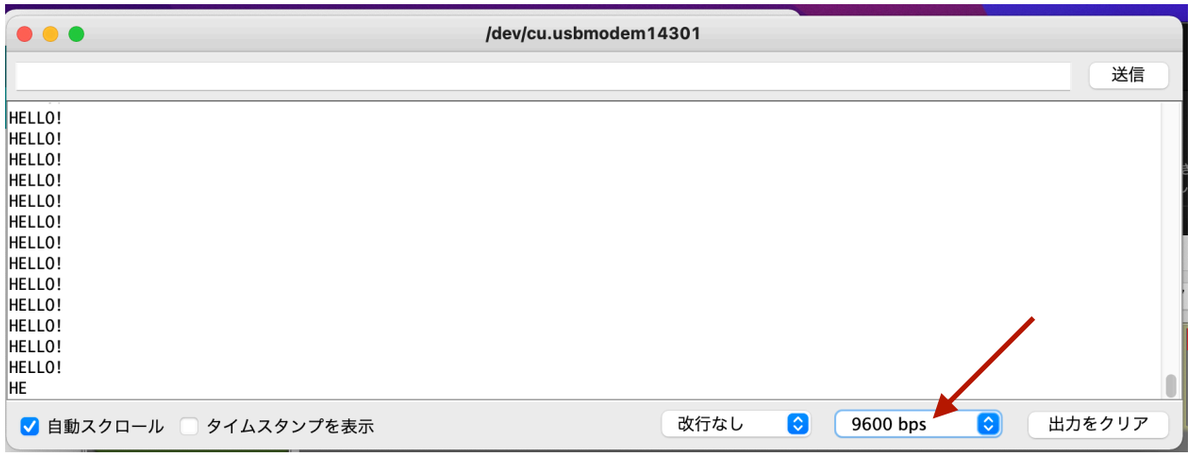
　まず、制御からプログラムをワークエリアに配置、次に通信からシリアル出力を配置してmessageの文字をHELLO!に変えてこんな感じにしてみよう！

次に、ArduBlockのArduinoにアップロードボタンを押して、プログラムをArduinoに書き込んでみよう。

うまく書けたかな？書き込めないよーって人はUSBケーブルを抜き差ししたり、４ページの通信ポートが正しいか確認しよう。
１回目だけ書いたプログラムを保存するかどうか聞いてくるので保存してね。

さあ、書き込めた！という人は隣にあるシリアルモニターボタンをクリックしてみよう。

Arduinoが正常に動いているとこんな感じでHELLO!という文字がスクロールしながら流れていく。
なんか変だなー？？？表示されないなー？？って人は、通信速度が9600になっているか確認してね。

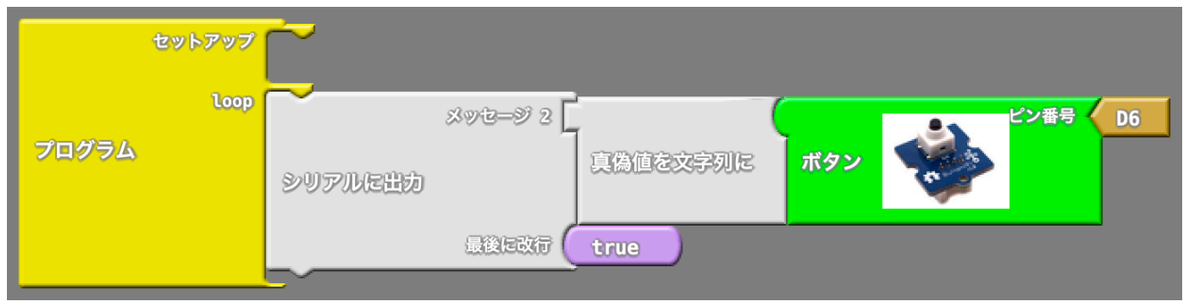


シリアルに出力という部品だけど、変更できるのは２ヶ所なんだ。

画面に出したい文字をHELLO!から自分の名前をローマ字にしたものに書き換えて結果が変わるか見てみよう。
そして、最後の改行のtrueを偽(false)に変えて書き込んだら表示がどんなふうになるか確認してみよう。

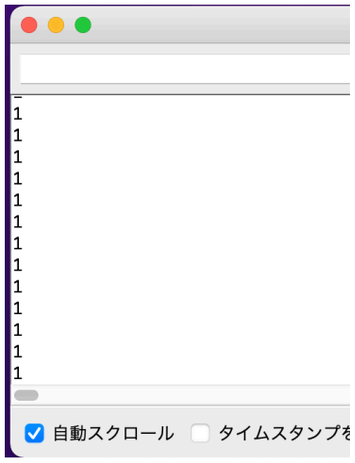
Arduinoからパソコンに文字が送れると何が便利なんだろう？
例えばこんなのはどうだろうか？
ワークエリアにこんな感じで貼って、messageを削除する。

次に通信の「真偽値を文字列に（後端が<ではなく丸い方）」を貼って、その後にSeed Studio Groveのボタンを貼り、ピン番号をD6にする。



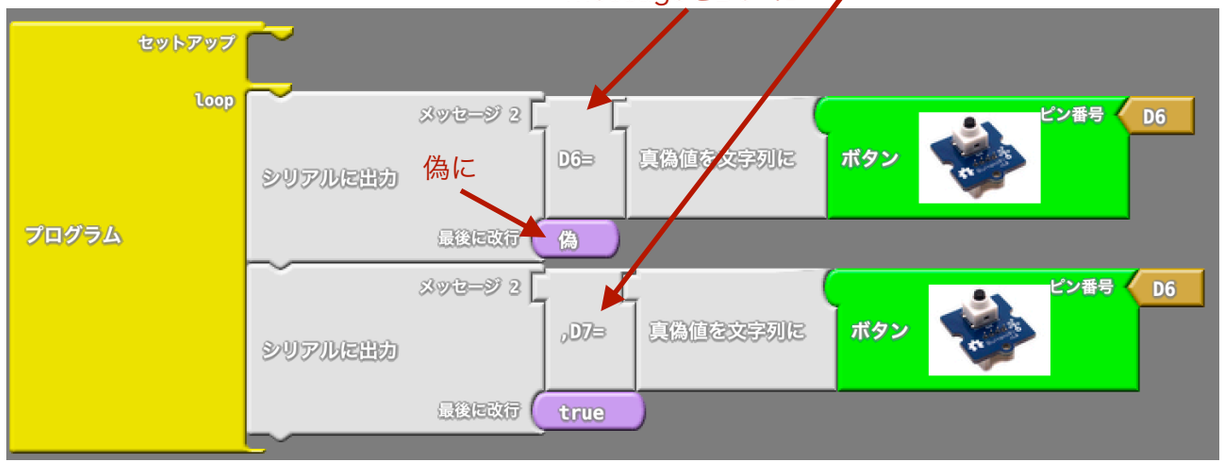
これを書き込んで、シリアルモニターでボタンを押したり離したりしながら見てみよう。
　どうだい？ボタンの押されている状態が目で見えるようなったかい？
　電子回路には接触不良が付き物。外からみてちゃんと動いているようでも「あれ？動きがおかしいなぁ」って時は、こんな感じでシリアルモニタにスイッチなどの状態を表示することで確認がしやすくなるよ。
でも、ボタンが１０個あって、その状態を画面に出すときに０とか１だけではどのボタンの入力なのかわからないよね？

（ボタンが複数個ある場合にシリアルモニタの表示が０と１だけではどのボタンかわからない）



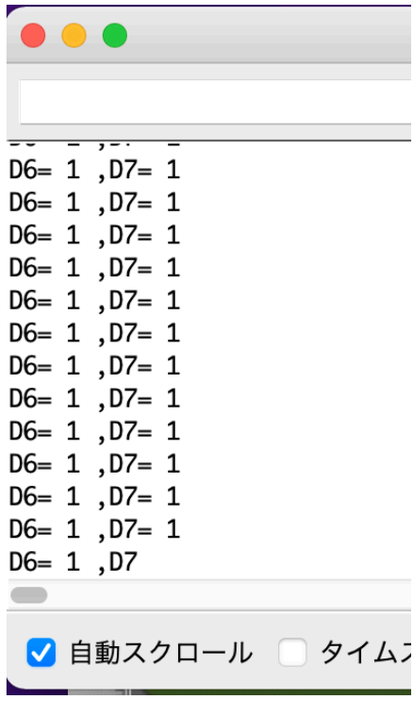
←この１という表示は、D6？D7??
　どっちの入力？？

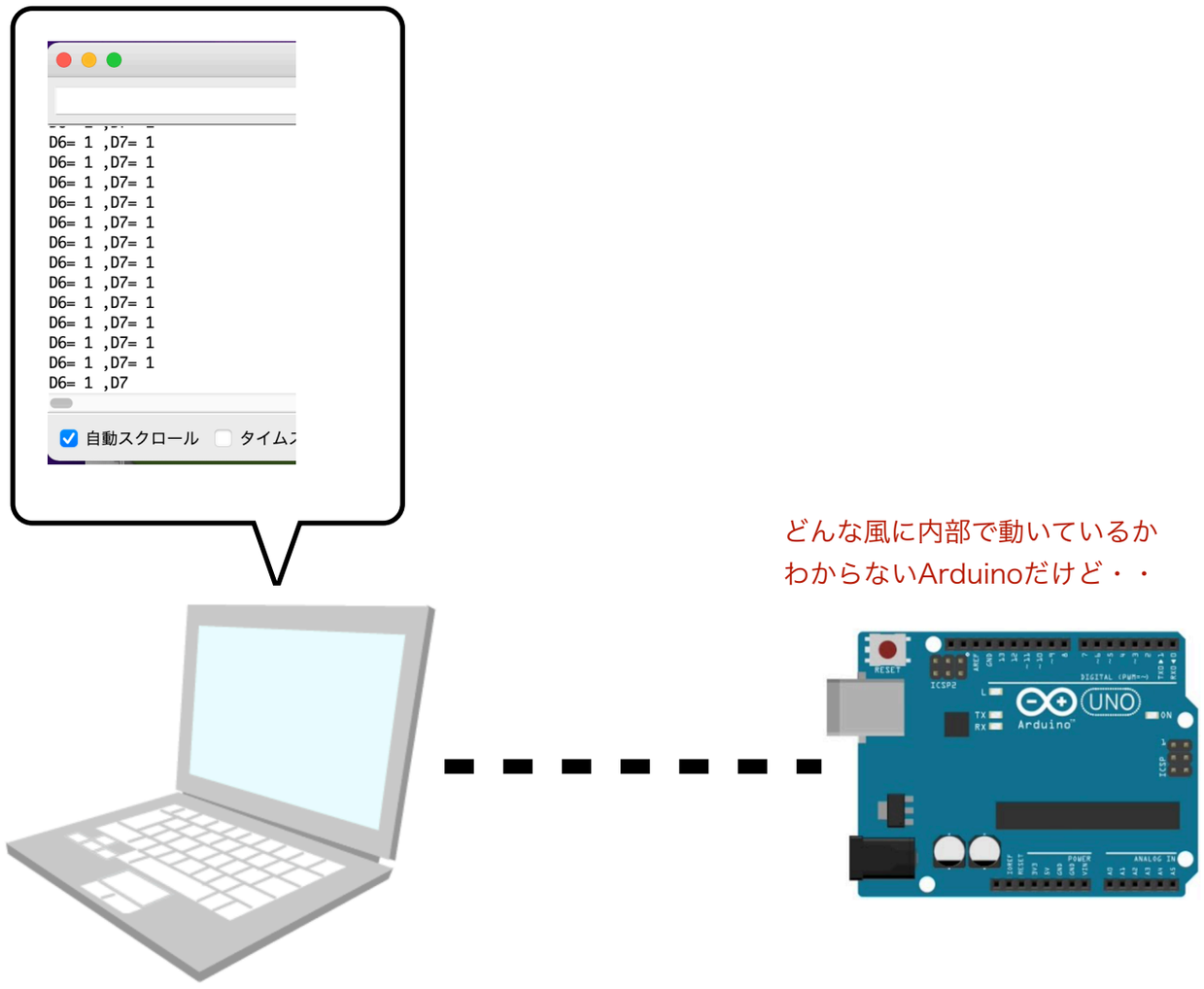
こんなふうにしてみようか。



こんな感じに入力がどっちのものなのか分かりやすくなったね。

　シリアルに出力を使うとプログラムの動きが思ったものと違うときにプログラムの修正を行うけどその手がかりを見つけるのにとても役立つよ。

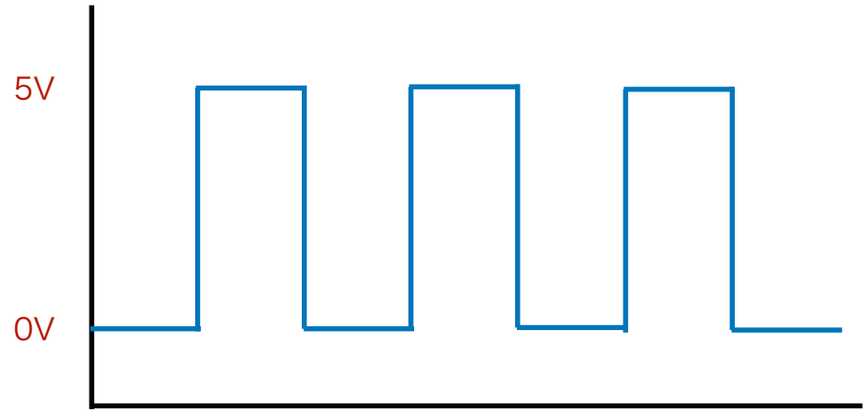




・デジタル信号の入出力を実験してみよう！

デジタル信号ってなんだ？？

今回扱うデジタル信号は0Vか5Vのどちらかの状態しかない電気信号になります。

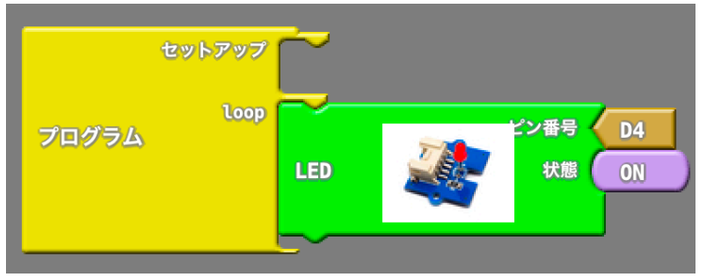


信号が5Vになっている状態を**HIGH**、0Vになっている状態を**LOW**とも言います。

ボタンが押された状態や、LEDが点灯しているか消灯しているかの状態もONかOFFかの２値しかないのでデジタル信号として処理することができます。

デジタル出力の実験をしてみよう

Seed Studio GroveのLEDを配置しピン番号をD4にしてArduinoに書き込んでみよう。
どうだい？ちゃんとLEDが点灯したかな？

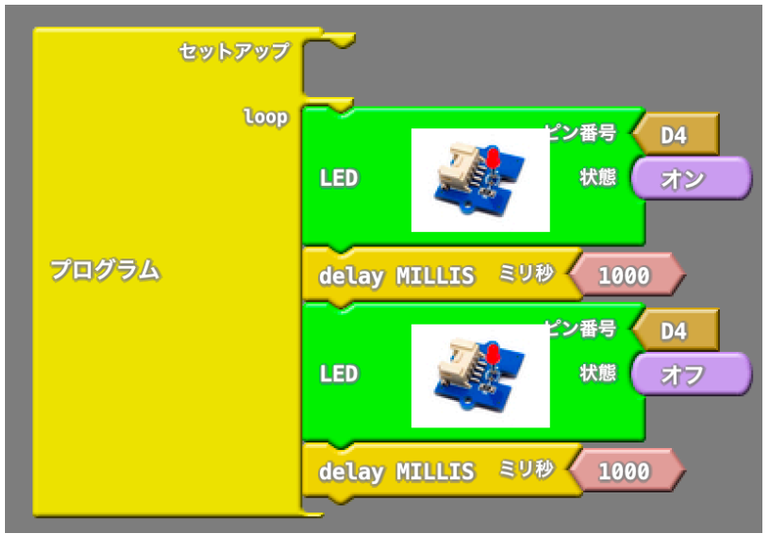


次に、ONをオフに変えて書き込んでみよう。
今度は消えたはずだ。

これらのプログラムは、状態がONの時はD4端子に5Vが出力されて、LEDが点灯する。
そしてオフの時は、D4端子が0VになってLEDが消灯するんだ。

まさにデジタル信号でコントロールしているね。

それじゃあ、プログラムをこんな感じに変えてみようか。
delay MILLIS ミリ秒は指定した時間（この場合は1000ミリ＝１秒）だけ止める部品なんだ。



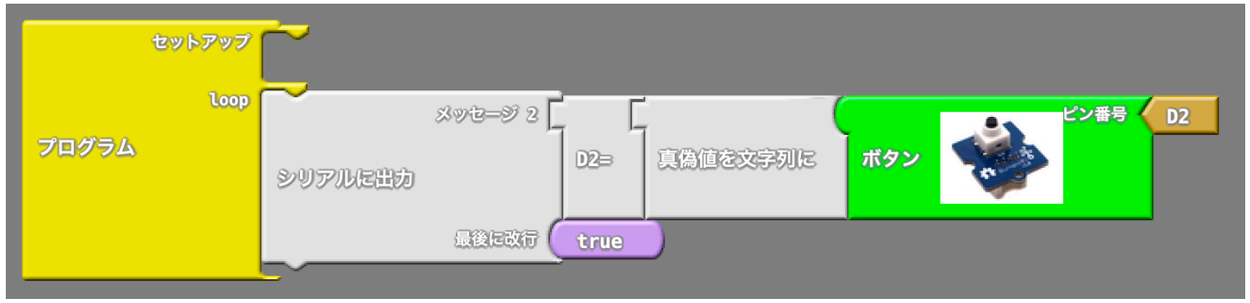
このプログラムを見ると、D4がオンになりその後1秒停止、次にD4がオフになりその後１秒停止をずっとループして実行するようになる。

LEDがどんな風になるか想像してみよう。
そしてその想像どおりの動きになるか書きこんで確認してみよう。

書きこんだ後は、1000ミリ秒の待ち時間を500とか100とか5000とか10000とかに変更してどんな動きになるか確認してみよう。
もちろん、片方の数値を大きくしたり小さくしても大丈夫だよ。

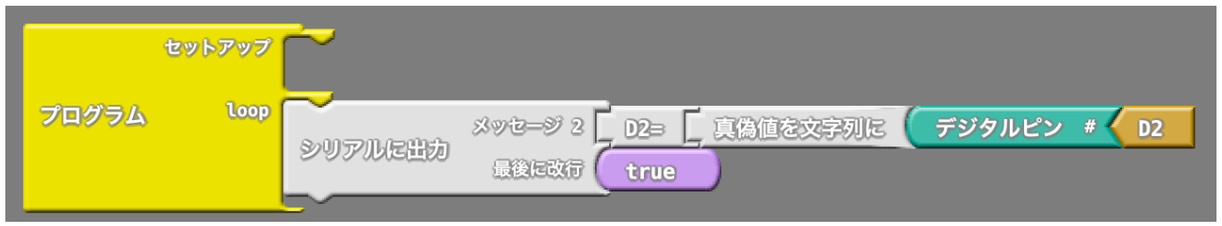
デジタル入力の実験をやってみよう

端子を5Vにしたり0Vにしたりするのがデジタル出力だけど、その逆で端子に5Vをかけるのか0Vにするのかを操作するのがデジタル入力になる。
このプログラムを書いてみよう。
（注意！ピン番号はD2にして、付属のケーブルでArduinoのD2端子とスイッチを配線してから実行してみよう）

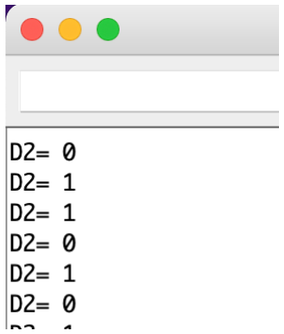


問題なくスイッチの入力を確認できたと思う。
ところで、今までボタンの入力はSeed Studio Groveのボタンを使ってきた。

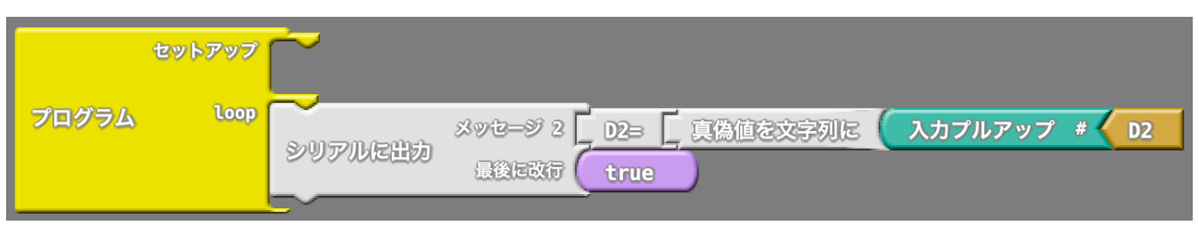
でも本当の？デジタル入力部品はピンのデジタルピンと入力プルアップの２つになる。
下の感じに変えてみようか。そして実行して動きを確認してみよう。



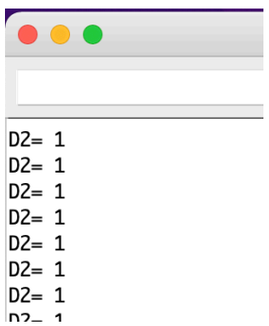
問題なくちゃんと動いているね。
じゃあ、D6に差しているケーブルを抜いてみようか。



あれれ？おかしいね。スイッチを押してないのに1とか0になる。
じゃあ、デジタルピンから入力プルアップに変えてみようか。

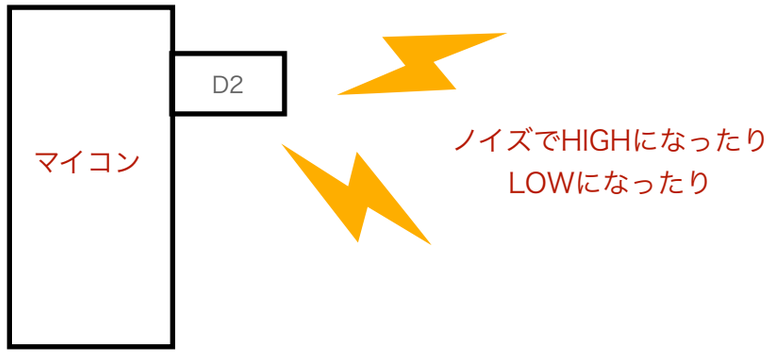


ケーブルを元に戻して確認してみよう。
スイッチを操作して正常に動いているのを確認してからまたケーブルを抜いてみようか。

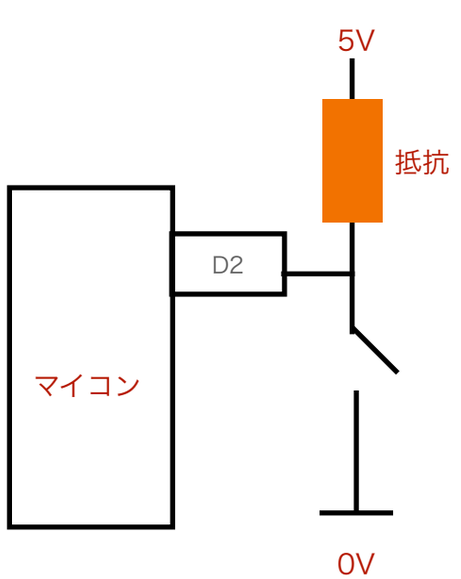


こんどは１で安定しているね。勝手に０になることもない。
ん？？でもスイッチが繋がってないのになんで１？なんで5Vになっているの？？

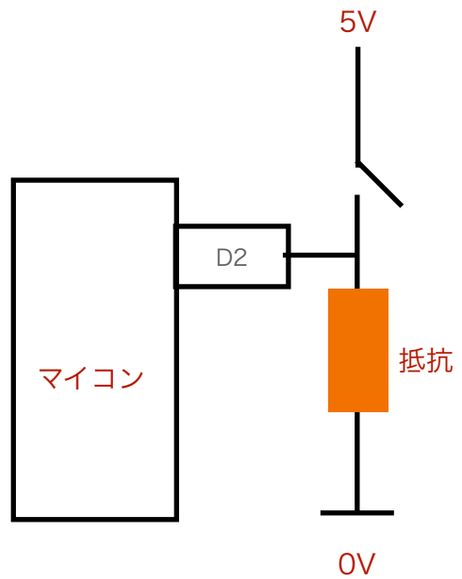
実は、マイコン（Arduinoも含む）のデジタル入力はその端子の先に何も繋がってない状態だと環境のノイズなどを拾って不安定な動きになってしまうんだ。
なので絶対そんな状態にしちゃいけないんだよ。



そこで、入力端子を常に5VをかけてHIGHにする「プルアップ」もしくは0Vに落としてLOWにする「プルダウン」という方法で安定させるんだよ。



プルアップ回路

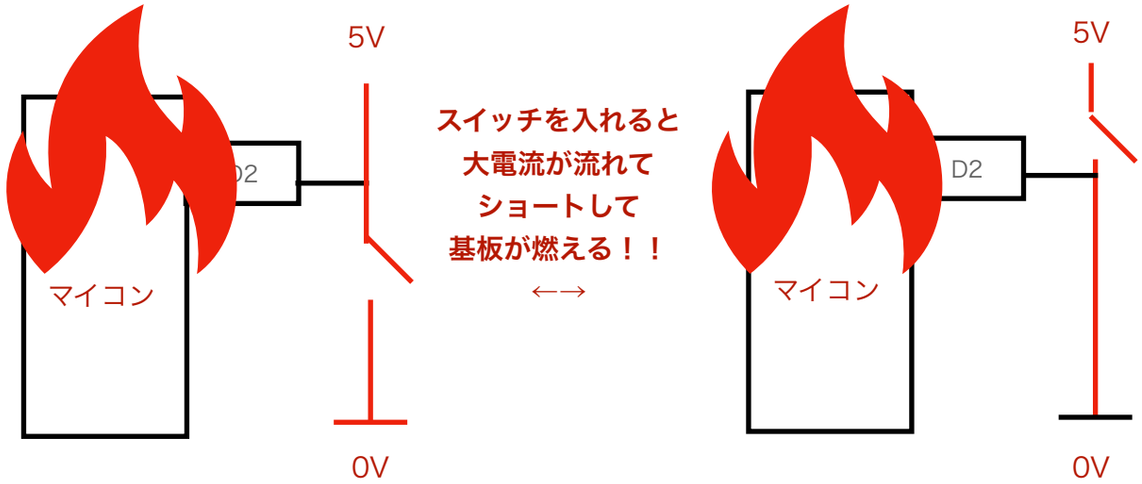


プルダウン回路

デジタル入力では必ずプルアップかプルダウン状態にしないと不安定なることを覚えておこう。

忘れちゃうとプログラムが意図しない動きをしたりするよ。

ちなみにGroveのスイッチ類はちゃんとプルダウン回路が入っているので何も考えなくてもいいけど、自分でスイッチ用意するときはちゃんとプルアップ回路にしておこう。
その時は抵抗を必ず入れること。
入れないとどうなるか・・・？



プルアップ・ダウンの抵抗はオームの法則で求めることができる。(抵抗＝電圧/電流)

マイコンの仕様書を見ると入力のときにどのくらいの電流にすればいいか書いてあるので確認しよう。

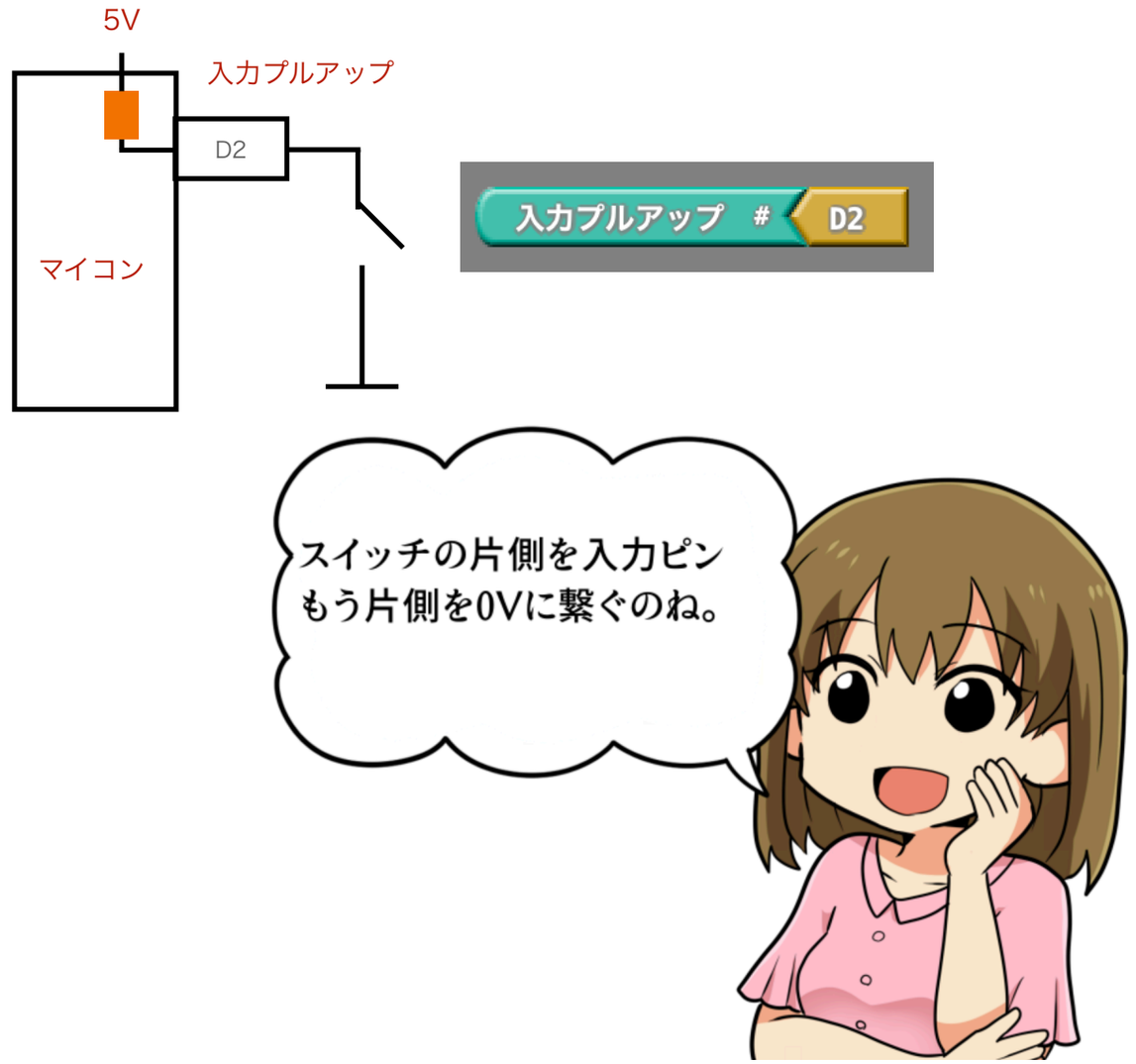
ちなみにArduino Unoの場合最大で40mA以下。でも入力の時は5mAくらいで充分なので5V/0.005A=1000Ω=1KΩの抵抗が必要になるね。

でも、スイッチをつけるたびにいちいち抵抗を準備してスイッチと配線して・・・ってめんどくさいよね。

そこで思い出してほしいのが14ページで「**入力プルアップ**」という部品を使ったときに入力が１のままで安定してた事。

実は、Arduinoの内部にはあらかじめこのプルアップ抵抗が入っているんだ。

だから、デジタルピンで入力をした時と違って安定して入力が出来てたんだね。



これまでの学習ではスイッチやLEDを単体で動作させてたけど、今度は組み合わせて動作させてみよう！

・ボタンを押したらLEDが点くプログラム

・大きい音を検出するとLEDが点くプログラム

ちなみにボタンを押したらLEDが点くプログラムはこんな感じにかくこともできる。

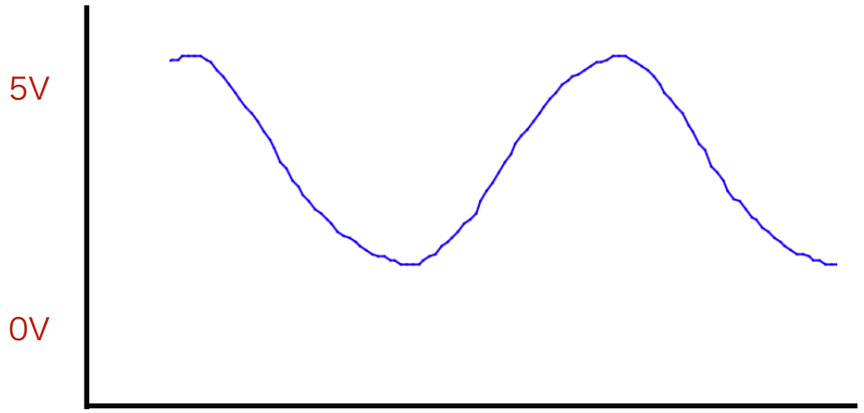
もう、みんなも気づいていると思うけど部品の写真にあまり意味はなく（意味があるものもあるけど）、重要なのはピン番号なんだ。

それと同じプログラムでもピン番号を変えるだけで全く別の動作に簡単に変えられる（スイッチ入力から音の入力へ）のもGroveの便利なところなんだよ。

・アナログ信号の入出力を実験してみよう！

アナログ信号ってなんだ？？

今回扱うアナログ信号は0V～5V間で変化する電気信号になります。



デジタル信号のようにHIGHやLOWといった感じで入力値を分けることはなく、5Vまでの電圧を0～1023までの数値(1024段階）で読み込むことができる。

例えば2.5Vの時は1024の半分の512近辺の数値になるよ。読み取れる電圧の最小ステップは5V/1024なので0.0049V単位の細かい変化を読み取れることになるね。

アナログ出力はちょっと注意が必要で、0V～5Vまでの出力が出来るのだけど、その指定が0～255までの256段階なんだ。

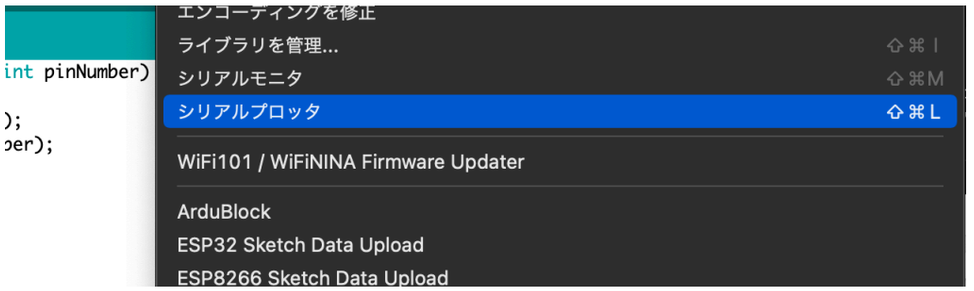
つまり出力電圧の最小ステップは5V/256＝0.019V単位になる。

アナログ入力の実験をしてみよう

下のプログラムのように、アナログピンA0の内容をシリアルに出力するようにして、Arduinoに書き込み、ポテンショナーをぐりぐり回してみよう。

シリアルモニターで見ると数値が0～1023で変わるはずだ。
それが0～5Vまでの電圧を1024段階で読み取った結果なんだよ。

次にシリアルモニタを閉じて、Arduinoのツールメニューからシリアルプロッターを開いてみよう。



今度は読み取った数値が左図のアナログ信号のような波形のグラフで表示されたと思う。
こんな感じで滑らかに変化する信号がアナログ信号なんだ。

いろんなアナログ信号を見てみよう

このキットでは、アナログ信号としてポテンショナーの他に、音と光を検出することができる。
アナログピンの番号を変えるだけで簡単に実験できるから、シリアルプロッターの画面を開いたままでピン番号を変えてプログラムを書きこんで動作を確認してみよう。

・光のアナログ信号（A6に変更）

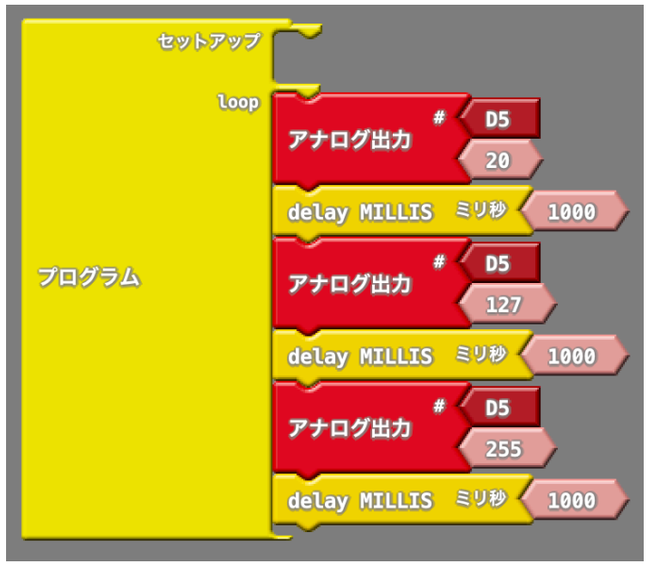
光センサーを手で覆ったりして変化を見てみよう。

・音のアナログ信号（A2に変更）

マイクセンサーを指で突いてみたりして変化を見てみよう。

アナログ出力の実験をやってみよう

　アナログ出力の実験ができるのは、LEDになります。
プログラムをこんな感じに書いて、D5とD6をケーブルで繋いで実行してみよう。



アナログ入出力を組み合わせて実験してみよう

ポテンショナーや光センサー、音センサーからの**入力に応じた値**をブザーに電圧として出力してあげれば、アナログ入力でアナログ出力をコントロールできる。

しかし、ここで１つ問題が。

そう、入力は0〜1023で取り込むけど、出力は0〜255で行われる。

つまり、入力値をそのままこんな感じで出力しちゃうと

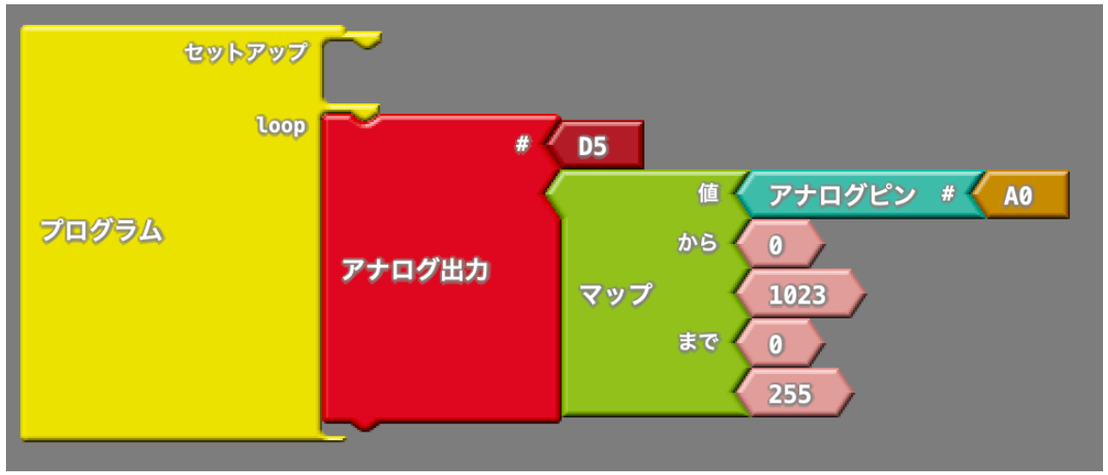
255を超える数値を渡してしまうことになる。
なので、入力の1023段階を出力の255段階に変換してあげないといけない。

１つの方法は、入力値を4(1024/256=4)で割ってしまうこと。
でも1023/4＝255.75となんか微妙。0.75の行方が気になる。
でも大丈夫。ArduBlockにはそれ用の部品がある。

「計算する」を開くと「マップ」という部品がある。

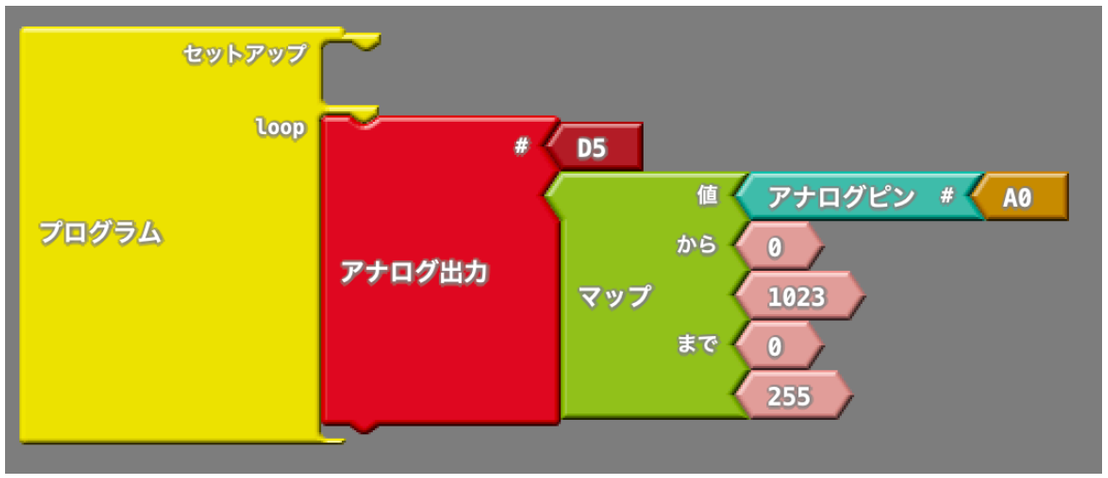
これをワークエリアに貼るとこんな感じになる。

ああ、これ欲しかったやつじゃんね！これをこうすれば一挙解決！！！

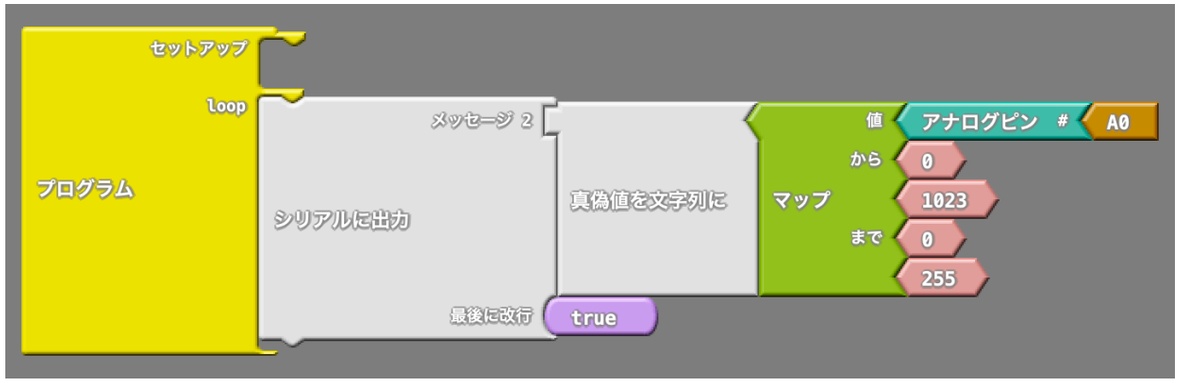


マップは上の段の「から」の組み合わせの範囲の数値を下の段の「まで」の範囲に丸めてくれる部品なんだ。

つまりこの図ではアナログピンA0の内容が０～1023段階で変化しても、それを0～255の数値として変換してくれる。

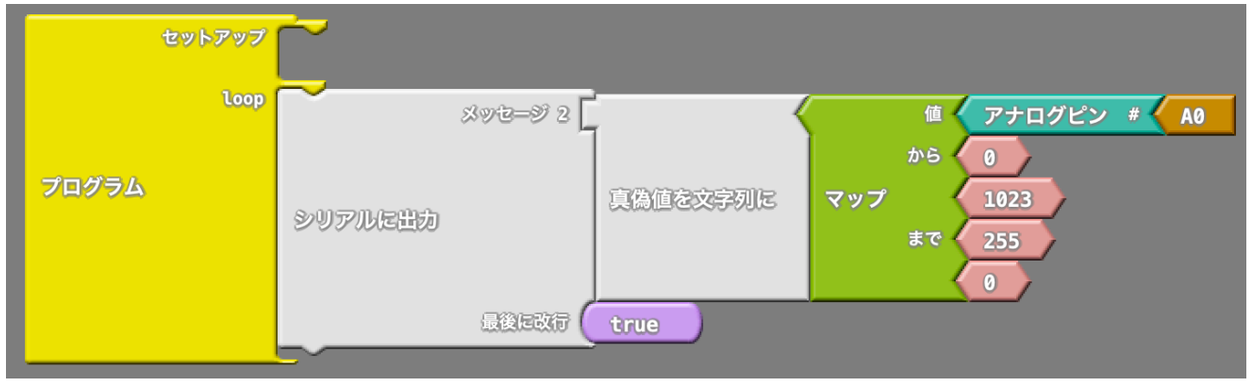


本当にそうなのか、下のプログラムを書きこんでシリアルモニタかプロッタで見てみよう。



どうだい？ちゃんと0～255の間の数値の変化に収まったかい？

ちなみにこのマップ、数値をこんな感じに入れ替えることもできる。



この場合はポテンショナーを回す方向と数値の変化がさっきとは逆になったはずだ。
こんな感じで変化の方向を入れ替えたり、数値を丸めたりするのに便利なのがマップなんだよ。

さあ、数値が安全に変化することを確認できたら左上のアナログ出力をするプログラムに書き換えてブザーからどんな音が出るかポテンショナーを回してみよう。

確認できたらポテンショナーから光センサ（A6）に変えて見て手で光センサーを覆ったりしてブザー音が変化するか確認してみよう。

・ArduBlockの機能パネルのいろんな部品をいじってみよう！（最低限必要なものを解説）

【制御パネル】プログラム

　セットアップに書いた部分はArduino電源投入時に１回だけ実行される。その後はloopの中身を繰り返し動作する。

【制御パネル】もし

　「テスト」の内容が成立すると「なら」の部分が実行される。(==はテストから、HIGHは変数/定数から配置)

【制御パネル】もし/でなければ

　「テスト」の内容が成立すると「なら」を実行、成立しない場合は「でなければ」を実行する。

【制御パネル】繰り返し

　「test」の条件が成立するとcommandsの中身を実行し、繰り返し以降の処理を行わない。

【機能パネル】回数分繰り返し

「回」で指定した回数だけ「commands」を実行する。

【機能パネル】break

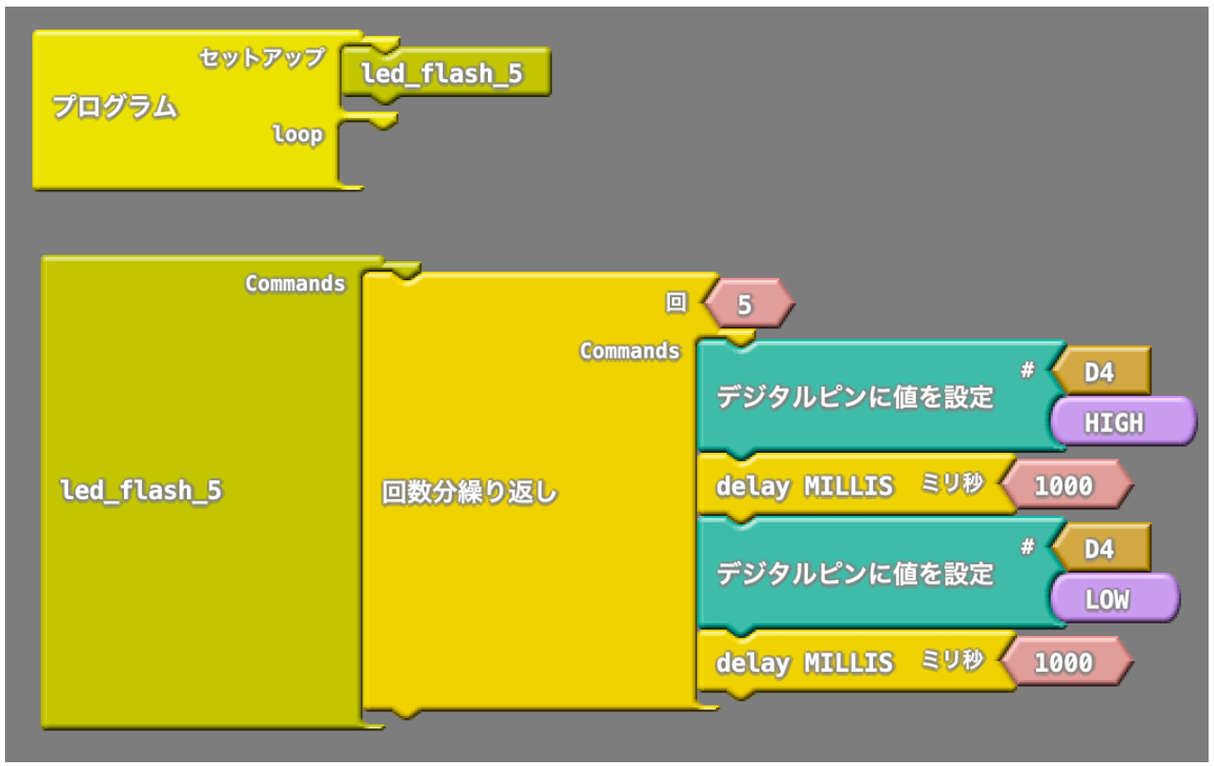
繰り返し動作から強制的に抜ける。

【機能パネル】サブルーチン

よく使う処理をまとめて呼び出すことができる。
部品の「サブルーチン」と書かれているところを編集し名前を付ける。

この場合はled_flash_5としたよ。
処理はcommandsがある方のサブルーチンに書く。

下の図の場合、セットアップの所にled_flash_5を２個おくと合計10回の点滅をするプログラムにすることができる。

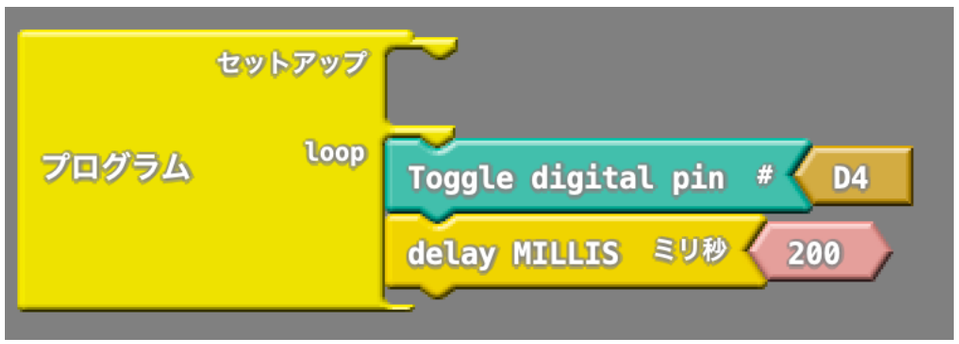


【ピンパネル】デジタルピンに値を設定

指定したピンの値(HIGH/LOW)を指定する。

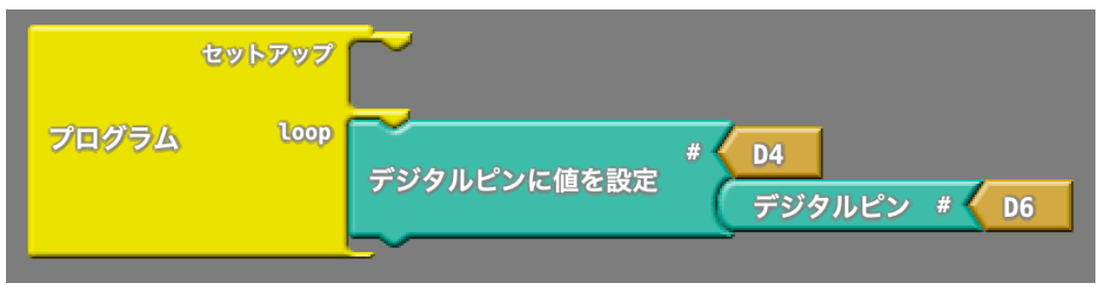
【ピンパネル】Toggle digital pin

指定したピンの信号をHIGHならLOWに、LOWならHIGHにする。



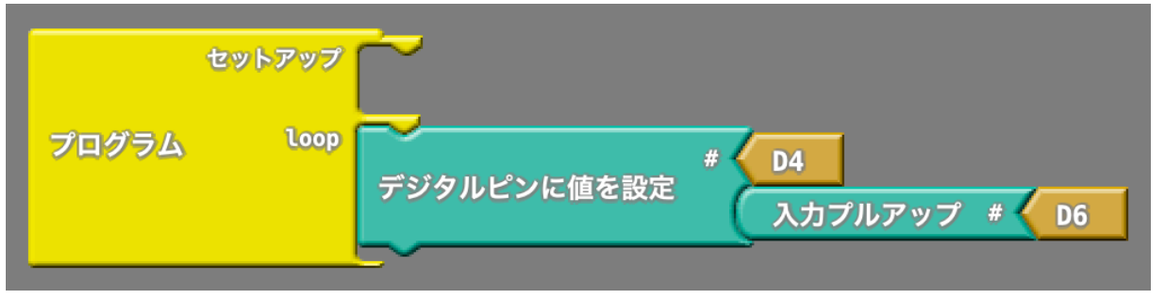
【ピンパネル】デジタルピン

指定したピンのデジタル信号を読み取る



【ピンパネル】入力プルアップ

指定したピンをプルアップ状態で入力する。



【ピンパネル】アナログピン

指定したピンをアナログ信号を読み取る

【ピンパネル】アナログ出力

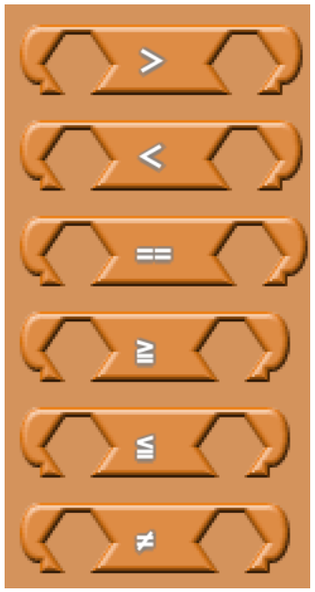
指定したピンにアナログ出力をする（0~5V)。

【ピンパネル】トーン

指定したピン番号に指定した周波数を出力する。
周波数の単位はHz。

【テストパネル】

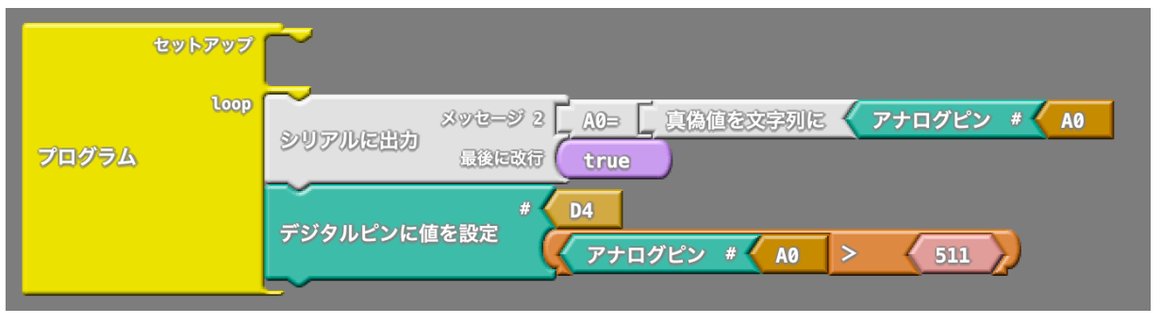
部品を入れる部分が＜＞の形になっているのは、アナログなどの数値を比較する部品になるよ。



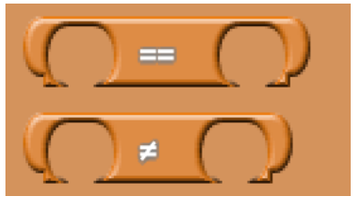
上から例えば比較対象がAとBの場合
A>Bのときに成立
A<Bのときに成立
A=Bのとき（同じとき）に成立
A>=Bのときに成立
A=<Bのときに成立
A<>Bのとき（同じではないとき）に成立するよ。
成立する時は１という数字が帰り、しない時は０の数字が返ってくる約束になっている。

なので、下のプログラムの場合は、D4(LED）の値としてアナログピンA0を読み取った値が511より大きかったら１、小さかったら０が値として戻ってくる。

なので、シリアルモニタを開いてポテンショナーの数値を見ながら回していって511を超えたらLEDが点灯するよ。



部品を入れる部分が、(　)の形になっているのは、デジタルの値を比較する部品になるよ。



上から、例えば比較対象がAとBの場合
A=Bのときに（同じとき）に成立
A<>Bのとき（違うとき）に成立するよ。
デジタルデータだから比較が２種類しかないんだね。

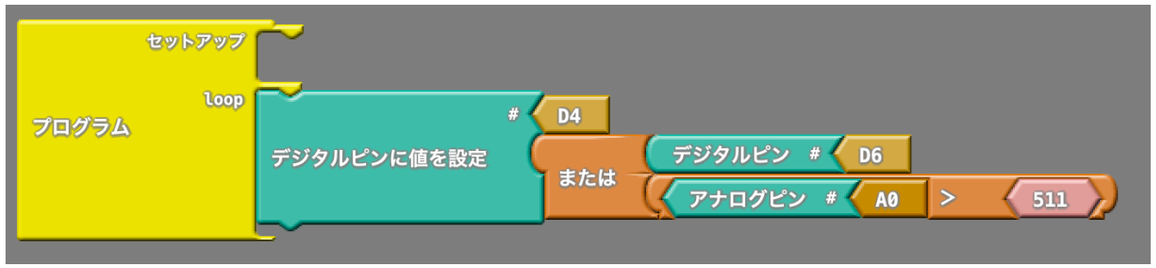
【テストパネル】かつ

この部品は２つの条件がどちらも成立したときに１が返り、どちらか一方もしくはどちらも成立しない場合には０が返ってくるよ。

例えば、上のプログラムは、D6(スイッチ）が押された状態で）、A0（ポテンショナー）が左に回されて511を超えていないとD4(LED)が点灯しないんだ。

【テストパネル】または

　この部品は２つの条件のうちどちらか一方が成立すると１を返し、どちらも成立していないと０を返すんだ。
このプログラムの場合は、D6(スイッチ）を押すかA0（ポテンショナー）を左に回して511を超えているかのどちらかが成立していると１を返す。
　どちらも成立していないと０を返すんだ。



【テストパネル】ではない

　この部品は、条件が成立していないときに１を返す。
成立していると０を返すんだ。
　下のプログラムの場合、スイッチを押してない場合はデジタルピンD6は０のままだけど、押されていないのでテスト結果として１が変える。
　それがそのままD4(LED)の出力になるので、スイッチを押さないとLEDが点灯し、押すと消灯する。

　テストパネルの部品は、プログラムを構成する「比較判断」を司る部品になる。
　とても重要な部品なんだよ。
これらの部品は、制御パネルの「もし」「もし／でなければ」「繰り返し」などとセットで使われるのが一般的。

　例えば左のプログラムはこんな感じに書き直すことができる。
　こっちの方がわかりやすいかな？？

　ここまでで機能パネルの制御、ピン、テストまでの説明をしました。
　ここからは、Arduinoをもっとコンピューターらしく使う部品の説明をします。

　いままでの制御プログラムは「スイッチが押されたらLEDが点灯する」とか「ポテンショナーの角度がある程度回ったらLEDを点灯する」という感じの

「何かアクションがあったらすぐに動く」

というプログラムを書いてた。

　なので「スイッチを10回押したら何かアクションを行う」みたいなプログラムは作れなかったんだ。

　それと、もうLEDをチカチカっと点灯させるのは飽きてきたよね？

　ということでここからは、「何か動くもの」を実際に作ってみようか。

　みんなの手元には、学習キットの他にサーボモータが１つあると思う。
　このサーボモータ使って何かないかな？と思って考えた結果、右のページのアイテムを思いついた。

・輪ゴムガトリングガン零式を作ろう！

　このサーボモータは
モータの軸が180度くらい
旋回動作をする。

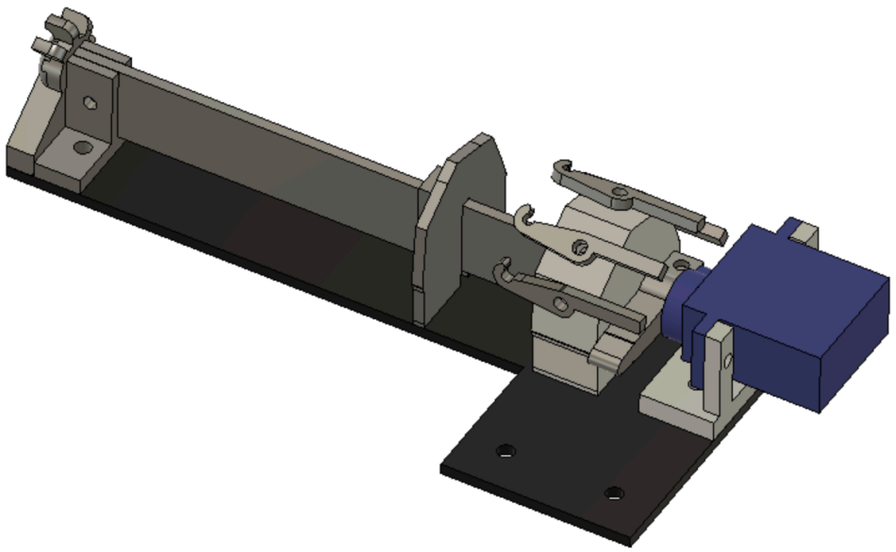
　そこで、４本の輪ゴムを
１つづつ発射できる
ガトリングガン
「零（ゼロ）式」を設計
してみたよ。

　セットされた輪ゴムは
トリガーに引っ掛けられて
いて、そのトリガーをサーボ
モーターについたレバーが
押し出すことで輪ゴムが
外れて発射される。

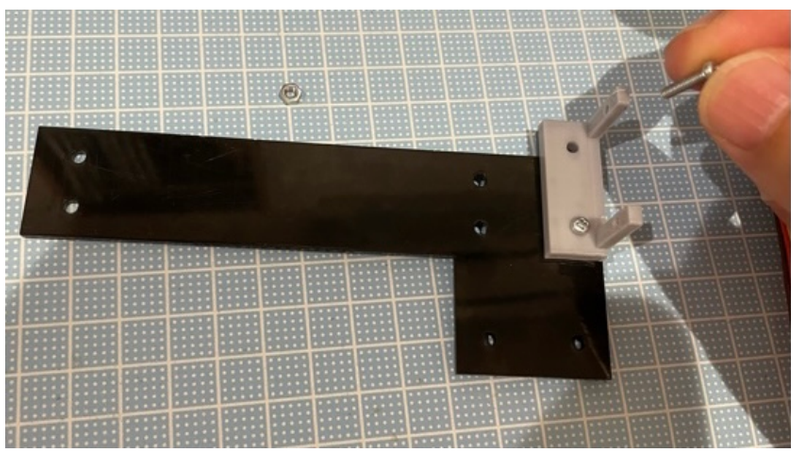
　ここからの学習では、
部品の組み立て、サーボ
モータの動かし方、ボタンを押したときに１つづつ輪ゴムを発射できるような動きの実現などを学んでいってみよう！

・ガトリングガンの組み立て

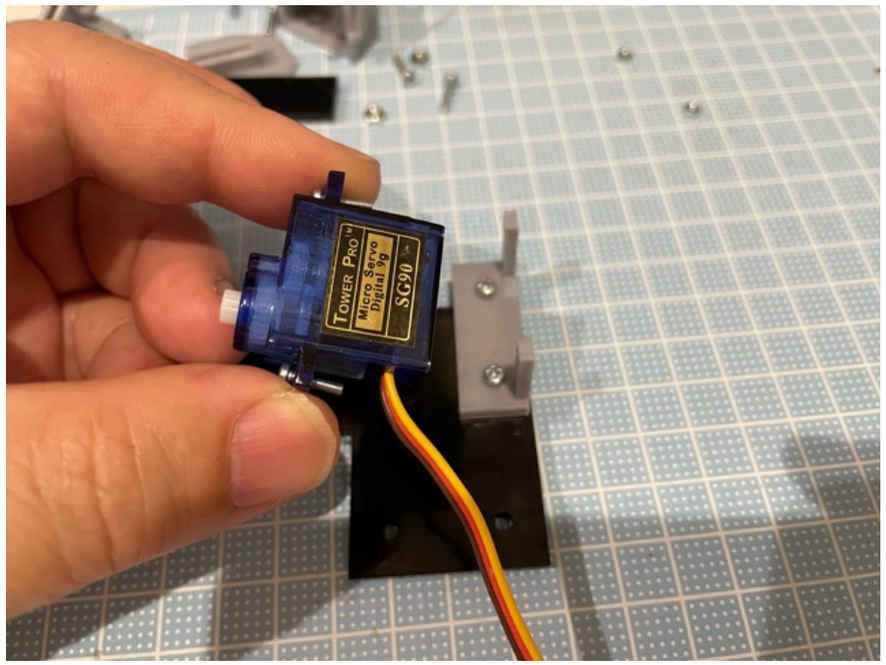
　これが組み立てるガトリングガンの完成図になるよ。
部品点数は１１点と少ないけど、小さいネジを使った組み立てになるのと樹脂部品なので力一杯ネジを締めると壊れちゃうかもしれないから、注意して組み立てを行おう。
組み立てる前に全ての手順を読んでおこう。

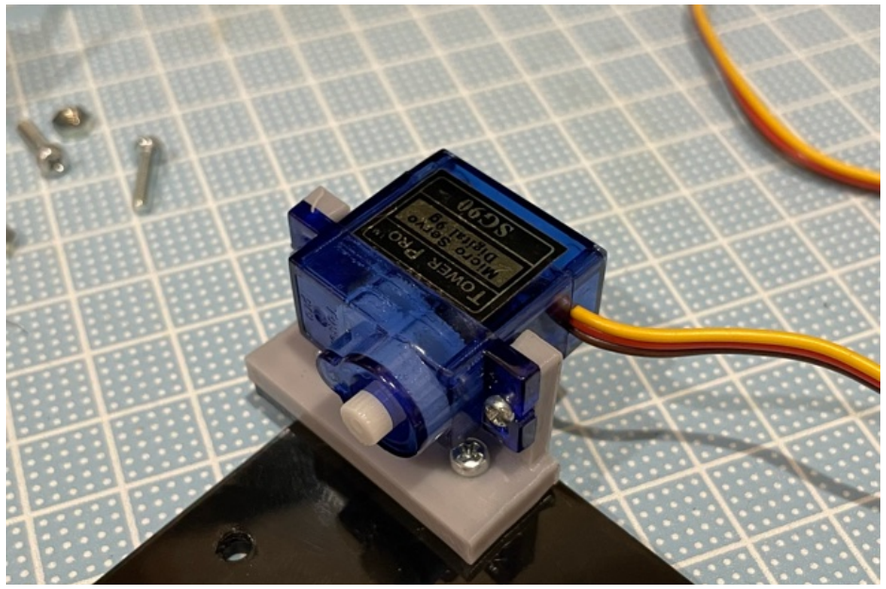


1）ベースにサーボステーをM2.5x10ネジ2本とM2.5ナットで固定する。**(M2.5x12ネジの場合あり)**

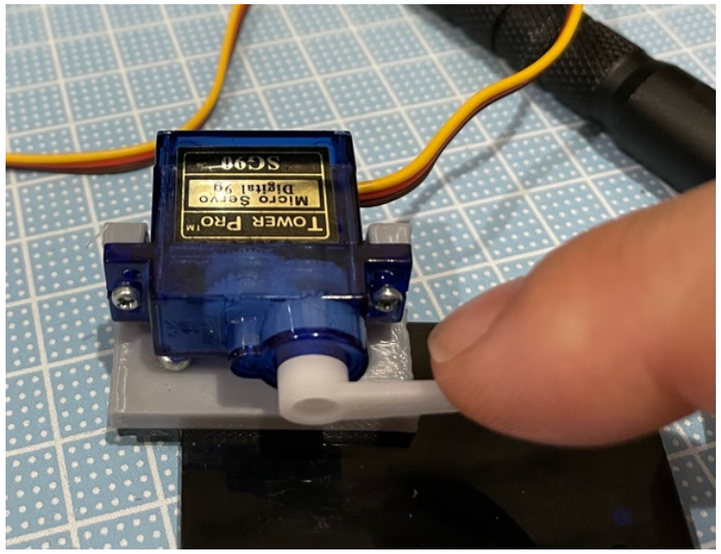


2)サーボをM2x8ネジ２本とM2ナットでサーボステーに固定する。

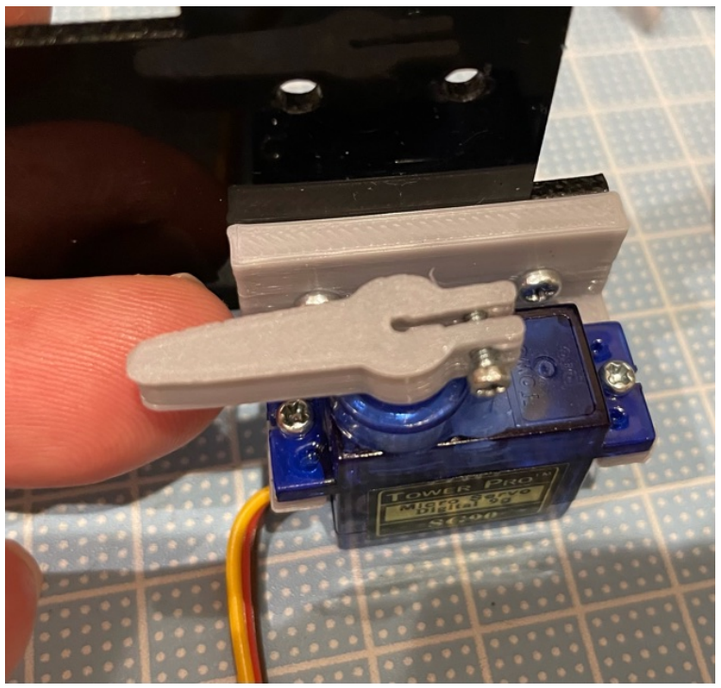




３）付属のサーボホーンを取り付け、写真のように時計回りに優しく回したときに止まる位置まで回転させておく。
（止まった場所がサーボの原点位置になります）



４）サーボホーンを先ほどの位置出しをしたサーボホーンと同じ位置に取り付ける。
M2x8ネジとM2ナットで固定するが、サーボホーンがきつくてサーボの軸に嵌まらないはずなので、マイナスドライバーを溝に差し込み少しこじってから押し込むと軸に入るようになる。



その後に付属ネジでサーボーンを固定する。

【作業のコツ】

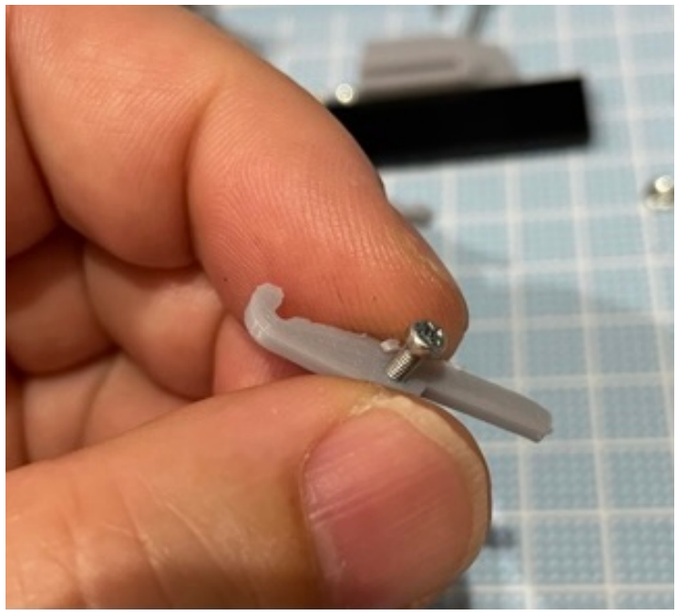
１）組み立ては磁力のあるドライバーを使うと便利

２）3Dプリンタで成型された樹脂部品は弱いので金属のネジを強く締め込むと変形したり割れたりするので注意。
締めすぎないように注意しよう。

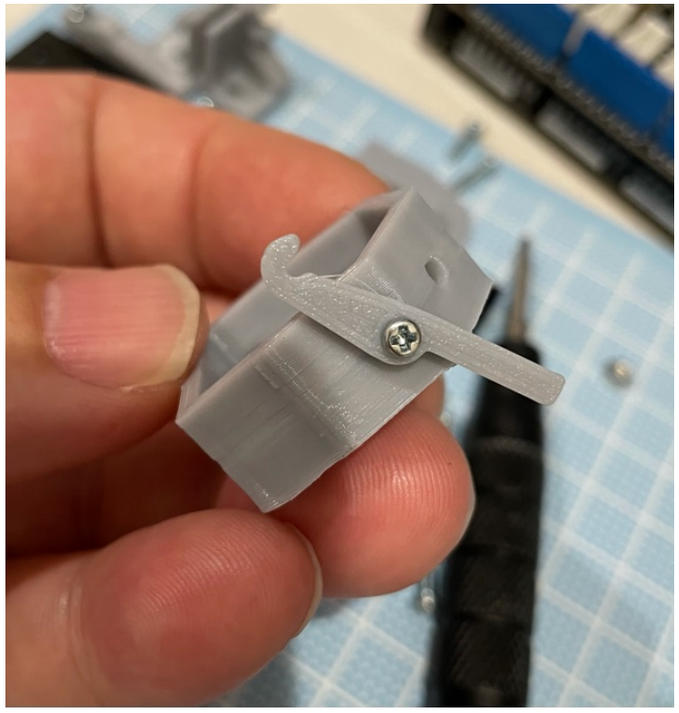
３）トリガーは、サーボモーターで跳ね上げて輪ゴムを発射するから、押されたときに旋回できないとダメ。
トリガーを指で軽く押して旋回することを確認しよう。

４）ベースに取り付ける部品（サーボステー、トリガーステー、ストッパーヘッド）は少し強めにネジを締めても壊れないのでしっかりと固定する。

4）トリガーを組み立てる。M2x8**(もしくはM2x6)**ネジをトリガーにこの方向で捩じ込む。４組作成する。

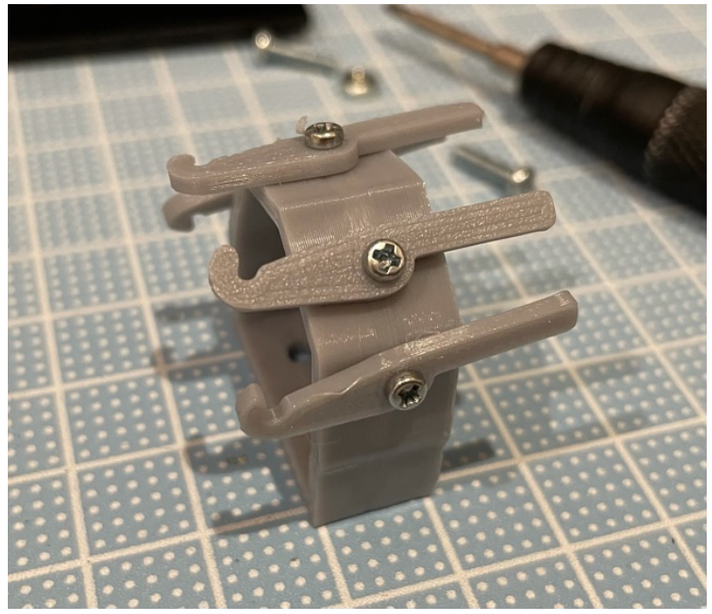


5）トリガーステーにこの方向で動くように固定する。

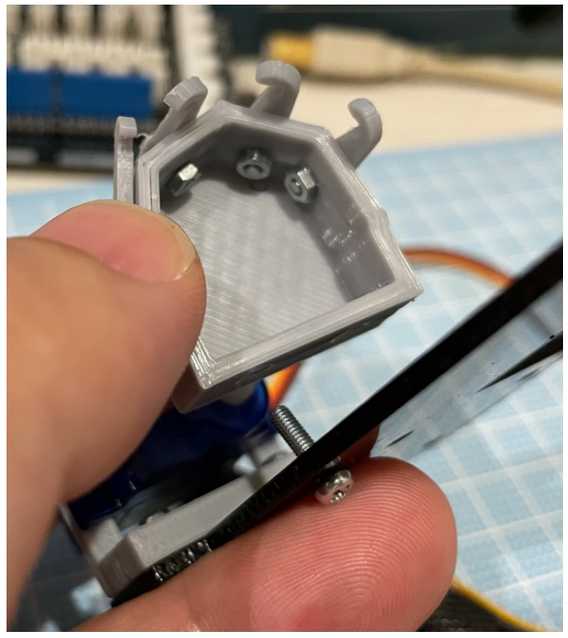


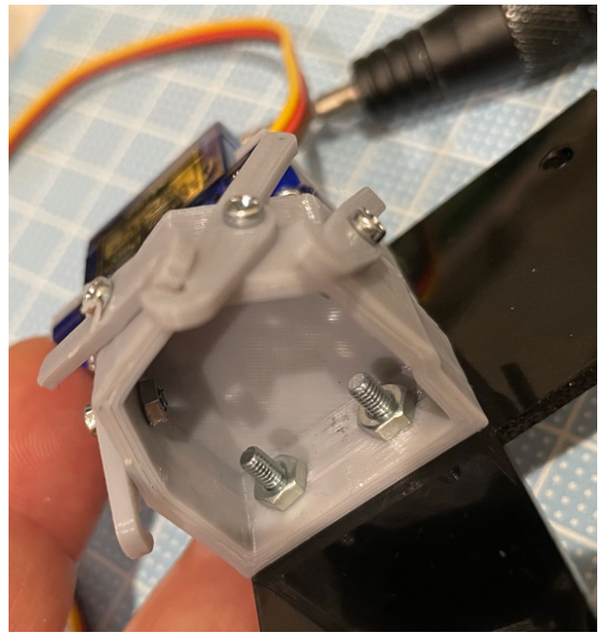
回転が止まるまで軽く締め込み、そこから1/4くらい戻すと丁度いいよ。

6）４組分、このように固定する。

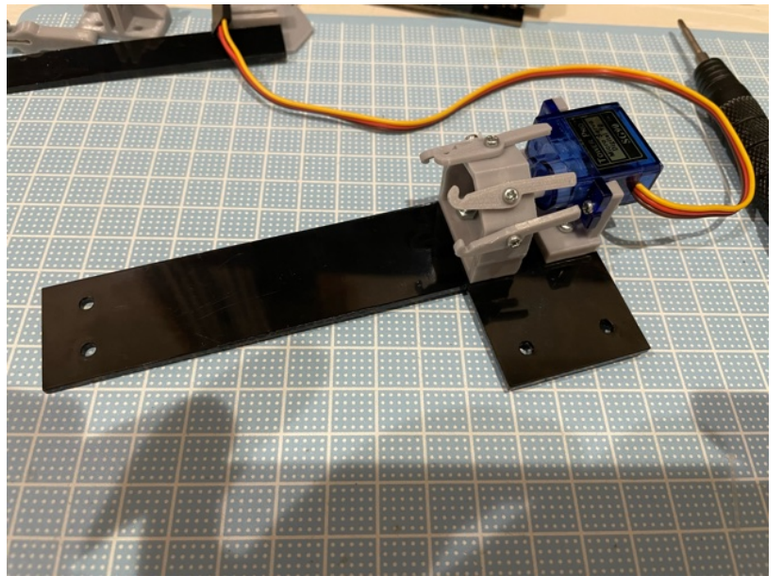


7）ベースにトリガーステーをM2.5x5ネジ**(もしくはM2.5x10)**２本で固定する。写真のナットは要りません。

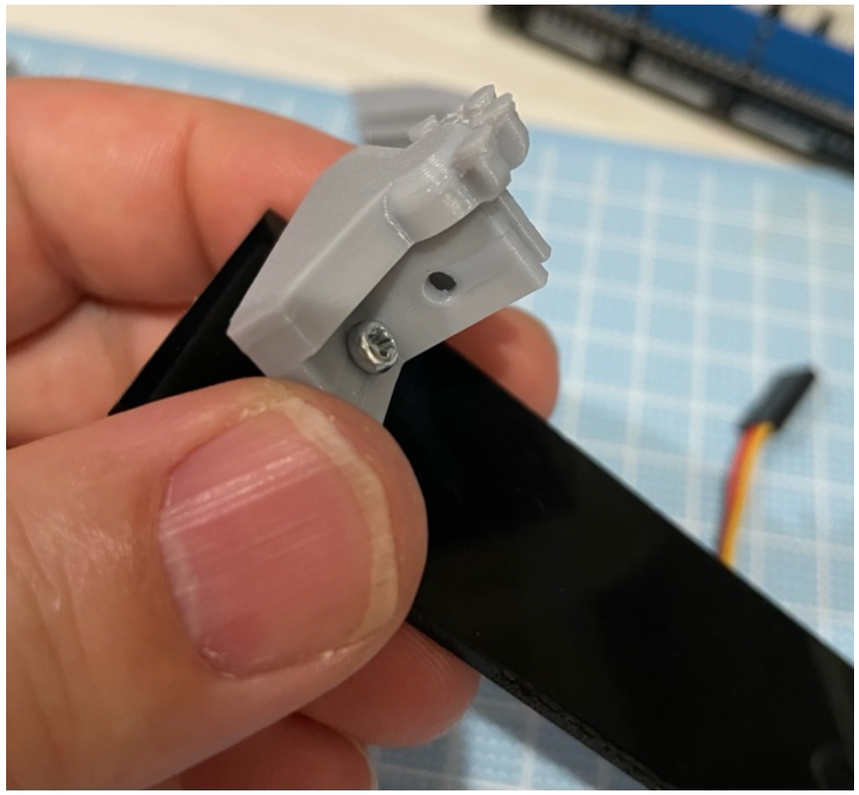




8）ここまで作業するとこの形になります。

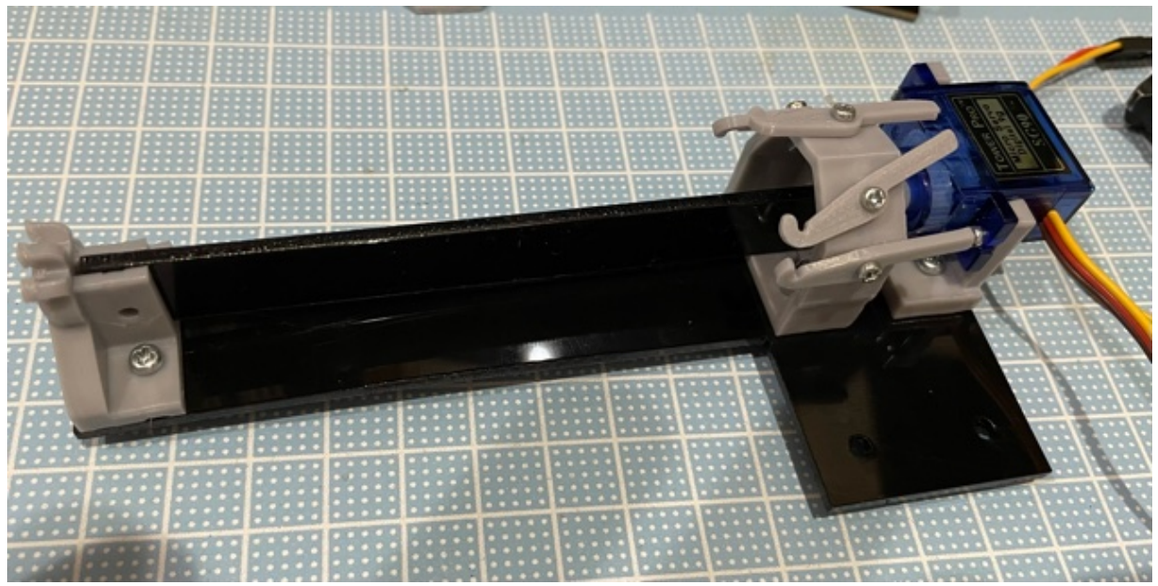


9）ストッパーヘッドをベースにM2.5x10ネジ**(もしくはM2.5x12)**２本とM2.5ナットで固定する。

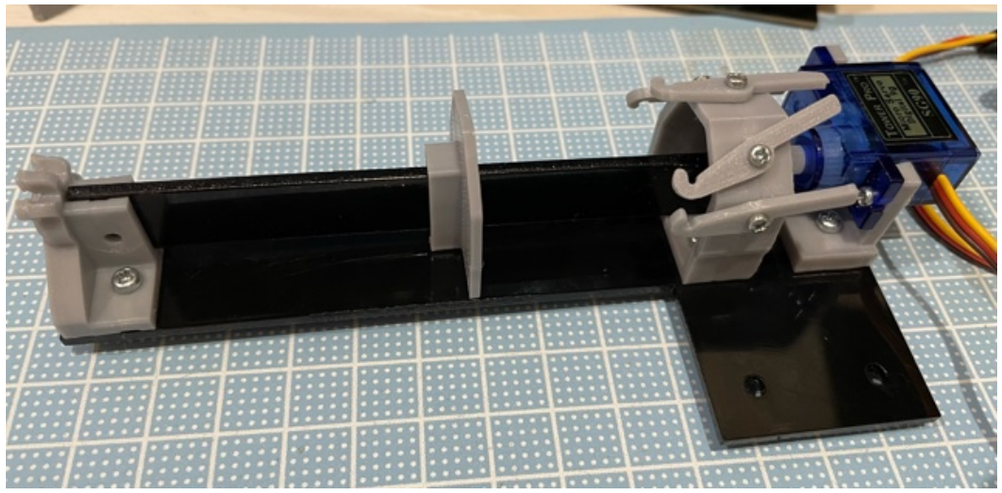


10）ビームを差し込みます。穴の開いている方がストッパーヘッド側になります。

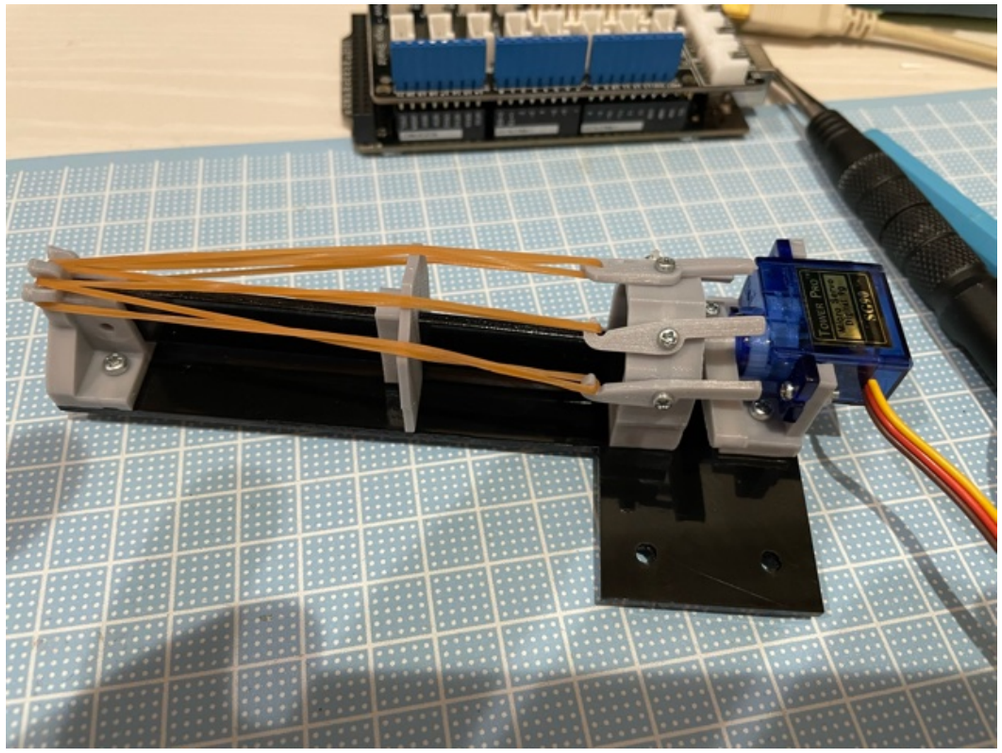
　反対側を最初にトリガーステーに差し込んでから穴の開いた方をストッパーヘッドの溝にはめてください。



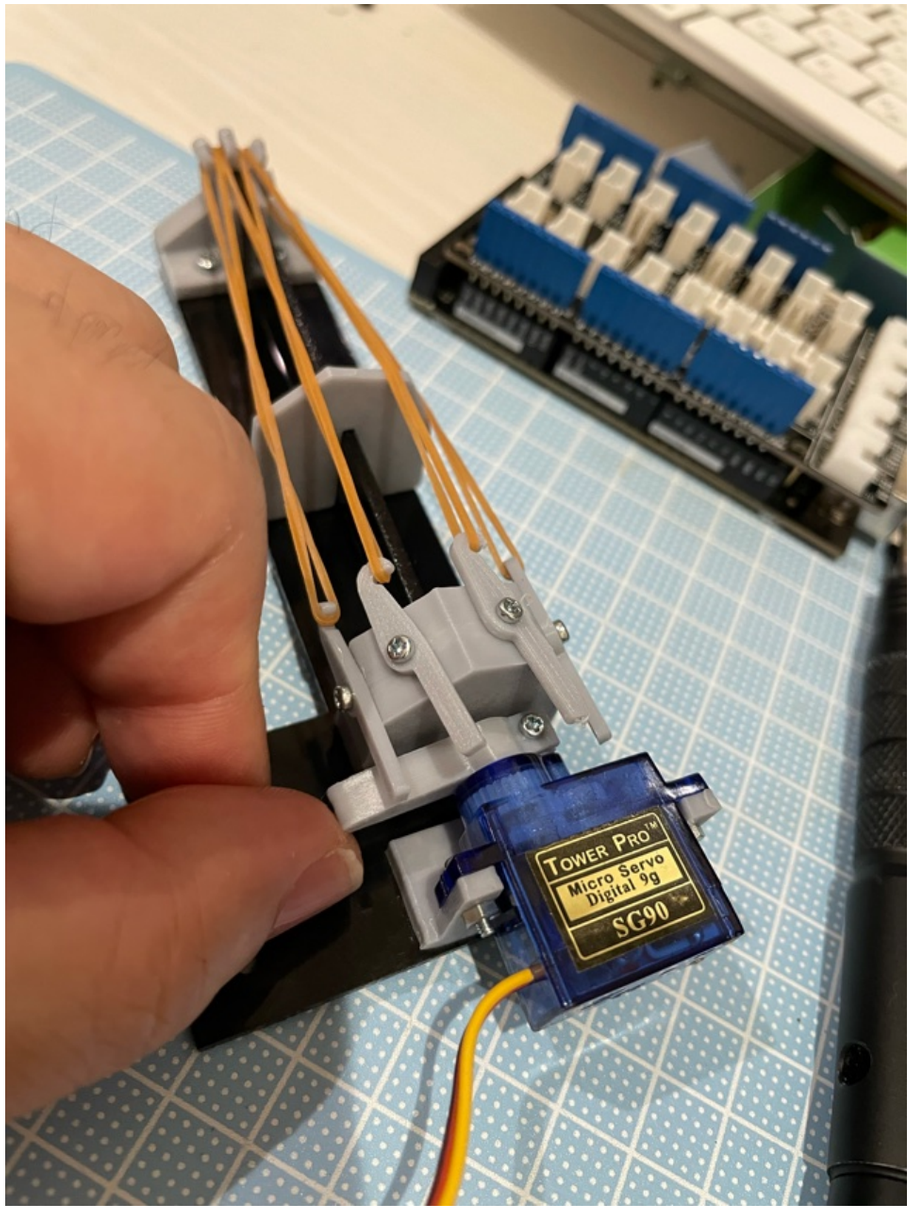
１１）テンショナープレートを適当な場所に差し込みます。
位置は写真を参考にしてください。



１２）輪ゴム奥側から順番にかけて完成。



１３）指でサーボホーンを時計回りに回してみて、輪ゴムが飛ぶことを確認する。
うまく飛ばない時は、テンショナープレートを前後に移動し、よく飛ぶ場所を探してみる（一番奥の輪ゴムが飛びにくい）。



・ガトリンググガンを制御してみよう

　サーボモータを動かす部品は、Generic Hardwareパネルの Servoを使用する。

　ガトリンググガンのサーボのコネクタをD2に繋いで次のプログラムを実行してみよう。

　どうだい？ちゃんとサーボが奥側まで回ってトリガーを全て回して行ったかい？
　角度０が原点でそこから角度１８０まで一気に回す動きをさせたよ。うまく動いたら、サーボのコネクタを一旦抜いて原点までサーボホーンを戻しておこう。

次にボタンを押したらサーボが動くようにしてみよう。

うまく動いたら輪ゴムをセットして危なくない方向にガトリンググガンを向けてからボタンを押して発射してみよう。
うまく飛んだかな？

　このプログラムは一気に０から１８０まで動くので連射する感じで輪ゴムを発射する。
　これを１つ１つ発射するにはどうしたらいいだろう？

・これまで学んだ方法でのアプローチ

　トリガーを１～４まで１つづつ回して輪ゴムを発射するには、サーボモータをそれぞれを押し回した場所で止めないといけない。
　まずは、これまで学んだ方法で考えてみようか。
例えば、ポテンショナーとサーボモータを組み合わせるとサーボモータを好きな位置で止めることができる。

ポテンショナーを時計回りにいっぱい回したあとに、上のプログラムを書きこんでポテンショナーをゆっくり回してみよう。

　うまく動いたら、輪ゴムをセットして１つづつ発射してみよう。
（決してポテンショナーを逆に回さないこと。トリガーが壊れる事があります）

・新しいアプローチ

　これから作りたいのは「ボタンを押すたびにトリガー１、２、３、４を押し回して輪ゴムを発射する動き」になる。

　この動きを実現するためには、
１）それぞれのトリガーがサーボのどの位置で輪ゴムを発射するか調べる。
２）ボタンを押すたびにトリガー１、２、３、４の輪ゴムが発射される位置へ順にサーボモータを回す。

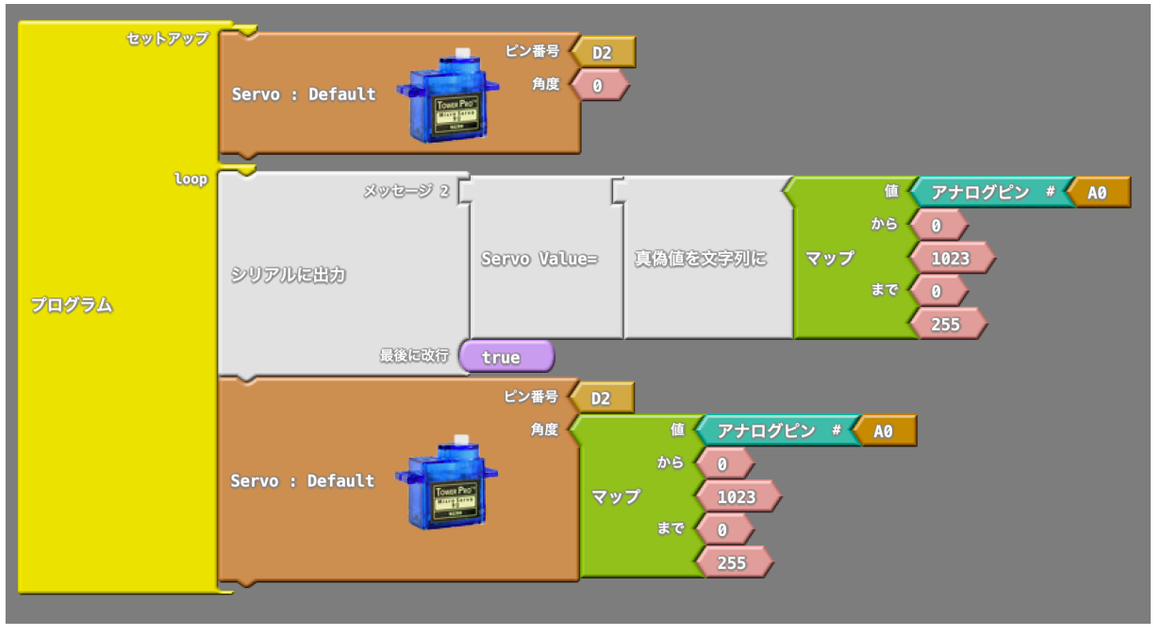
　この２つを解決しないといけない。
じゃあ、まずは１）からやってみようか。

１）トリガーの発射位置を調査する

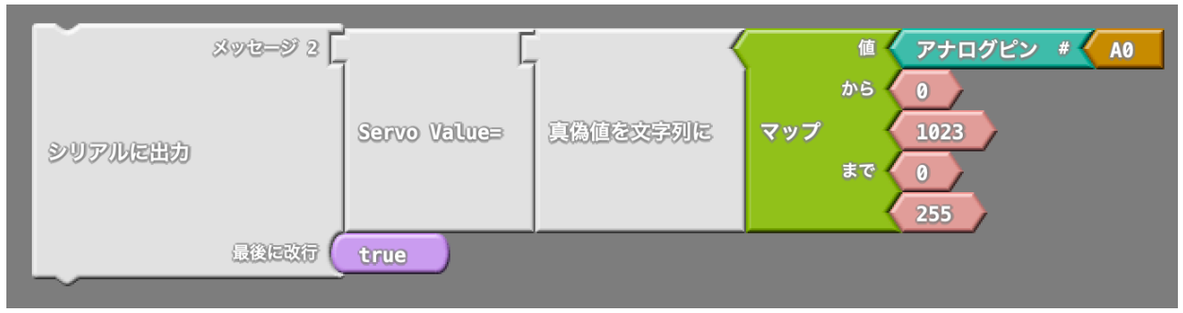
　どの位置で輪ゴムが発射されるかは、サーモボータの位置をシリアルモニタで確認してその値をメモすればいいね。
　次のページでそのプログラムを書いてみたよ。

サーボモータの停止位置を表示するプログラム

さっきのプログラムにシリアル出力を付け足すよ。

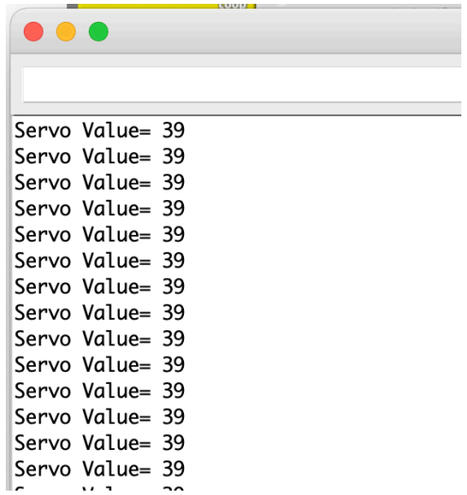


付け足すのはこの部分だね。



まずはトリガー１に輪ゴムをかけて、発射されるまでポテンショナーを回してみよう。

こんな感じで、39のときにトリガー１の輪ゴムが発射された。



こんな感じに２、３、４も調べてみたよ。

トリガー２：64
トリガー３：111
トリガー４：160

　これらの値をボタンを押すたびにサーボモータに書き込んだら１つづつ輪ゴムを発射するプログラムができるね！

・変数を使おう

　ボタンを押すたびにサーボモータを指定した位置へ移動させるのには、一番目ならここ、二番目ならここ・・・みたいな感じにプログラムを書けばいいね。

　その一番目、二番目という判断をするために変数を使うんだよ。
　変数というのはデータを保管する箱みたいなものなんだ。

この変数にボタンを押すたびに１→２→３→４と１つづ増えていくデータを保管して、その次に制御パネルの「もし」部品で「もし変数が1ならサーボモータの位置を39に２なら64に、3なら111に、４なら160に」とプログラムを書けば目的は達成するね！

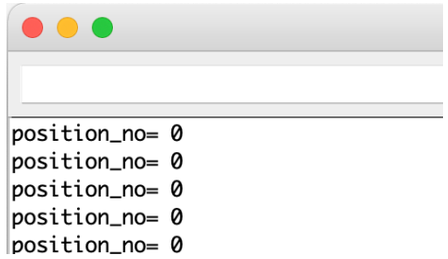
じゃあ、次からその変数の使い方を説明するよ。

変数／定数パネルで使うのは、set integer variableになるよ。

これは、整数を覚える変数なんだ。
変数のIntegerのところは名前になり、値の０がこの変数「Integer」が覚えている数値になるよ。

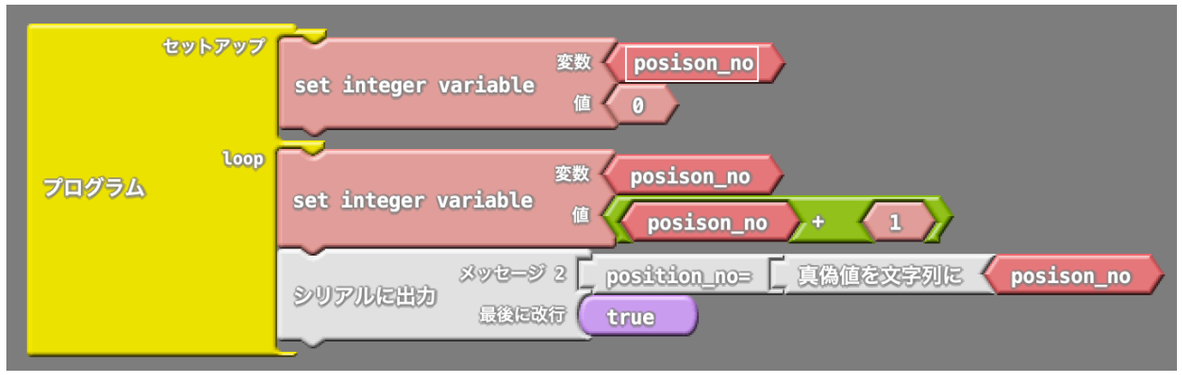
まずは、この変数をシリアルモニタで表示するプログラムを書いてみようか。

変数の名前はposition_noにするよ。名前は入力が面倒でも後から見てわかりやすいものにするのが良いんだ。
これを実行してシリアルモニタでみると



こんな感じでずっと０が書かれているね。
じゃあこれを１づつ増やしてみようか。

これが変数position_noを１づつ増やすプログラムになるよ。



流れを見てみると、
セットアップでposition_noに０を入れる（代入というよ）。

次にぐるぐる回っているloopのところで、
変数position_noに、position_noに１を足したものを代入する。
そしてposition_noの値をシリアルモニタに表示。
これを繰り返すとposition_noはloopで回るたびに1づつ増えていく。

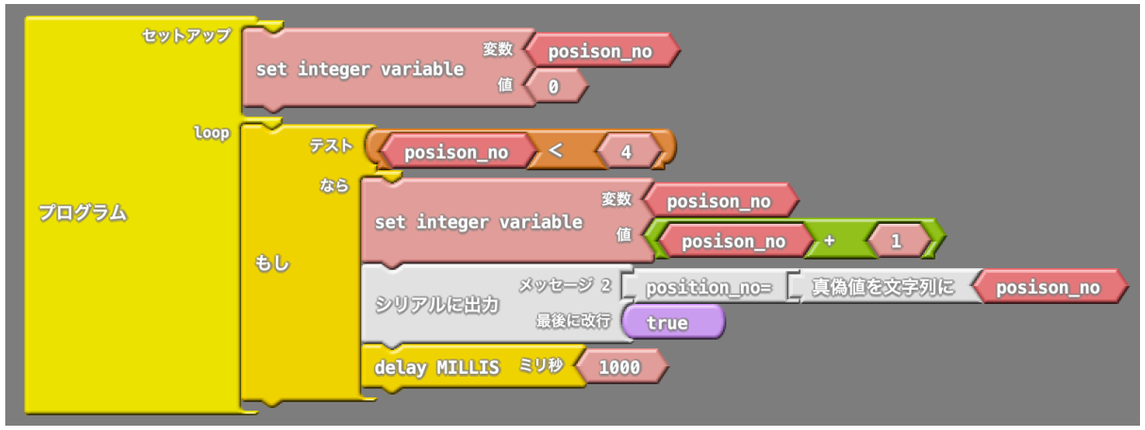
この足し算をするところは「計算するパネル」のところに部品があるんだ。
ここには足し算以外にもいろんな計算する部品がある。

【計算するパネル】

よく使うのは四則演算（+ - × ÷）と今までよく使ってたマップ、絶対値などになるよ。

＋：足し算をする
ー：引き算をする
×：掛け算をする
÷：割り算をする
割り算の余りを求める
絶対値：数値の符号(+-)を外す
マップ：数値を変換する
制限：数値の範囲を指定する

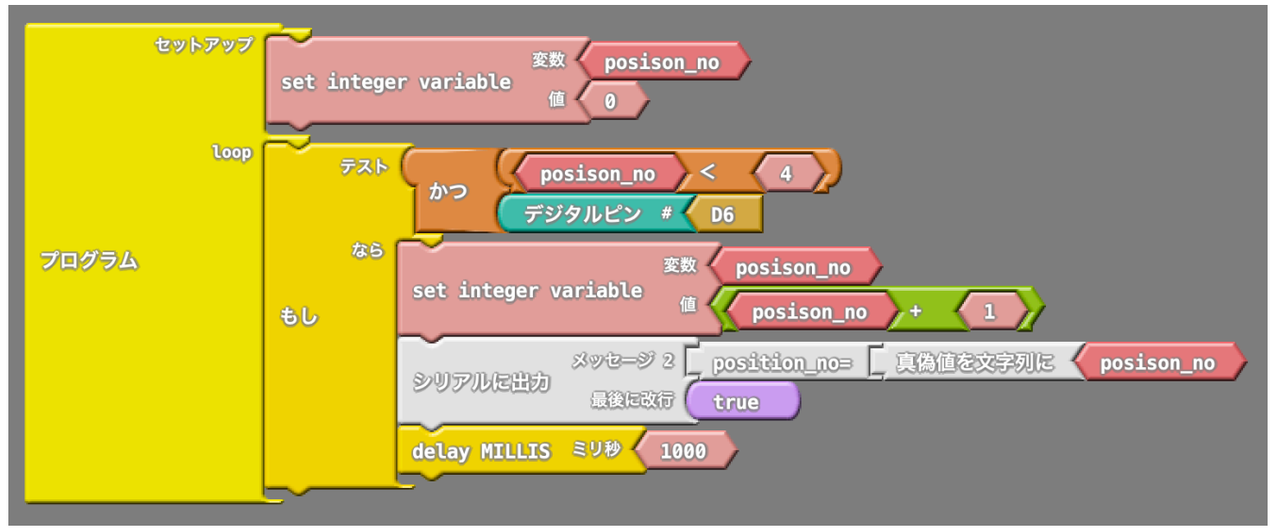
さっきのプログラムは延々と１づつ足してしまう。
実際は４まで足すプログラムでいいので、「もし、position_noが４以下なら１づつ足す」という感じでプログラムを書けば、目的通りの動きになりそうだね。
　じゃあ書いてみよう。



もしのテストに「position_noが4より小さい」と書いているので、実行すると４より小さい間だけ１を足していくよ。

これをボタンが押したときに１、２、３、４と足していけばいいので、ボタンを押したときという条件も追加しようか。
２つの条件が合致したときなのでテストパネルの「かつ」を使えばいいね。

これがボタンを押したらposition_noを1づつ足していくプログラムになる。
実行して試してみよう。



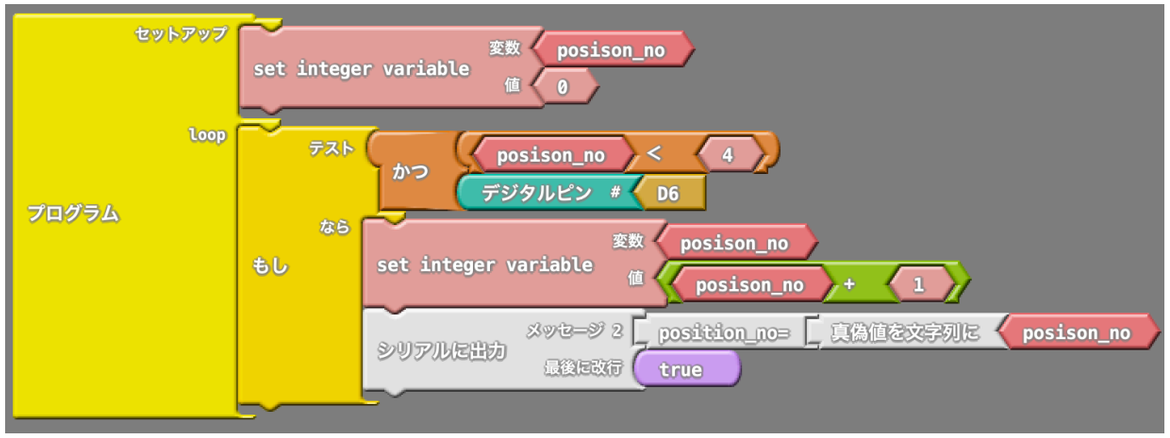
確認できたら、arduinoのリセットボタンを押して初期化し、ボタンを押しっぱなしにしてみよう。

あれ？これ・・・押しっぱなしにしていると勝手に４まで足されてしまうね。
連射したい時はボタンを長押しで！って話ならそれでもいいかもしれないけど、そうじゃないし、逆にボタンを連打してもdelay MILLSで処理が止まっているので連射ができない。
実際に連射して確認してみよう。

・実はボタンが押された時ではなく、ボタンの状態が変化した時で処理しなければいけなかった！

もしのテストの「ボタンが押された時」というのはデジタル入力がHIGHになっている間なので、押しっぱなししているとずっとHIGHの条件が入ってしまい、position_noが４以下ならばすぐに１が足されてしまう。

確認するためにdelay MILLISを外して実行してみようか。



arduinoの実行速度は、人が指でボタンを押した時もあっと今に処理をするから一気に４まで足してしまうんだ。

そのため、ボタンが押されたらの条件をHIGHからLOWに変わった瞬間（ボタンが離された瞬間）かLOWからHIGHに変わった瞬間（ボタンが押された瞬間）に変えないとダメなんだよ。

　この変化した瞬間ってどうやって捉えればいいんだろう？

・瞬間を捕まえろ！

　デジタル信号の読み出しは、デジタルピンの部品でおこなっている。

　loopのプログラムはぐるぐる回っているので、デジタルピンで読み出したときに、前のデジタルピンの状態を覚えておいてそれと違っていたら「瞬間」と認識すればいいんじゃないかな？

　こんな感じで

loop

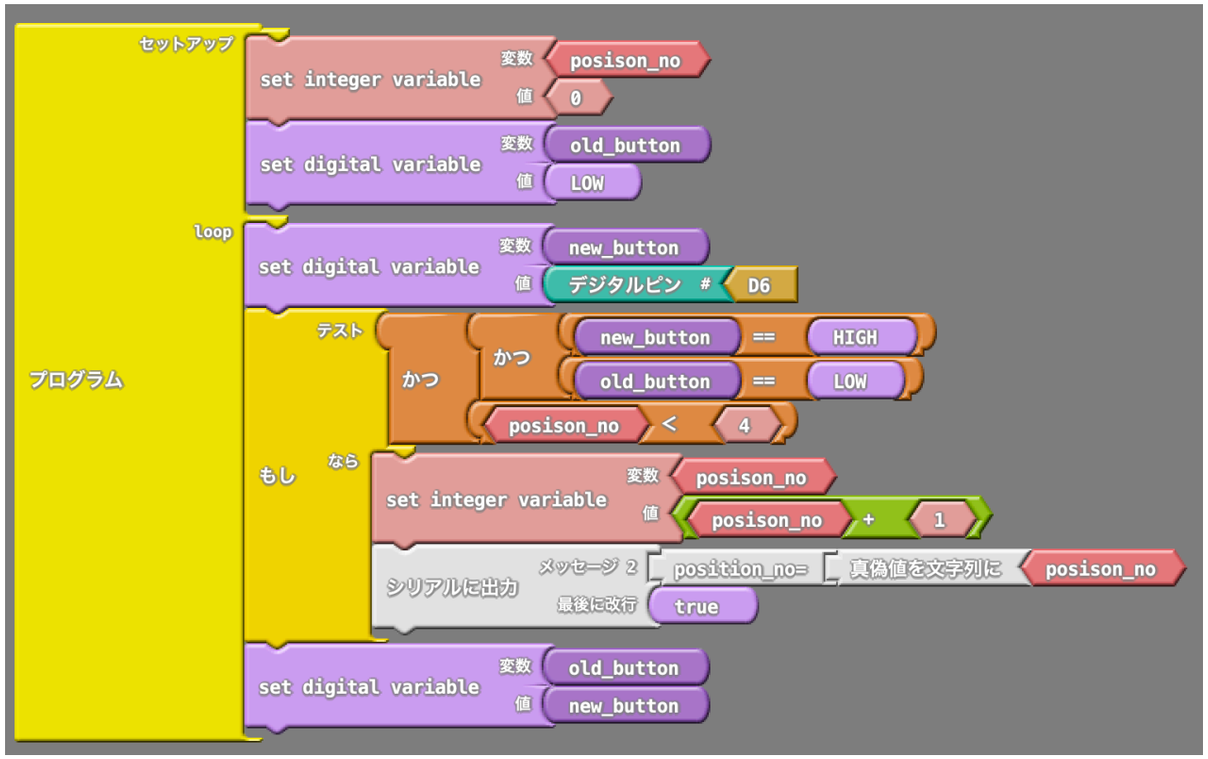
もしデジタルピンの内容がHIGHかつ
　前のデジタルピンの内容がLOW
　　かつposition_no＜４なら

　　＋１処理

前のデジタルピンの内容として今のデジタルピンの内容を変数に記憶

プログラムを書けば動きそうだ。
じゃあ書いてみよう。

ちょっと長いけど１つづつ部品を読んでみよう

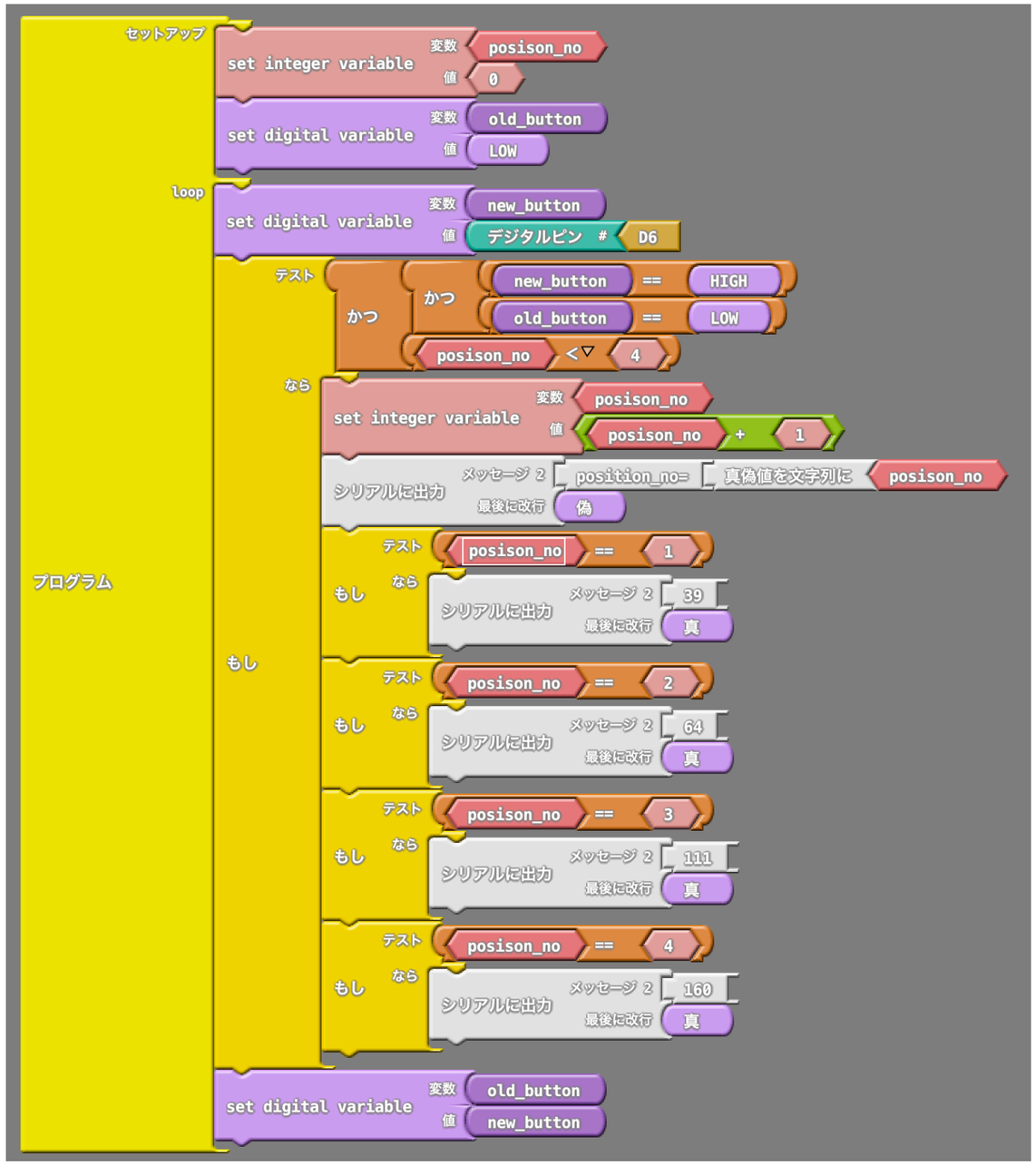


さっきはset integer variableを使ったけど、今度はボタンのデジタル値を変数に保存するのでset digital variableを使うよ。
old_buttonにD6の古い値を代入し、new_buttonにD6の新しい値を代入している。
loop先頭で新しい値を取得しているからそれを古い値と比較して瞬間をチェックしているよ。

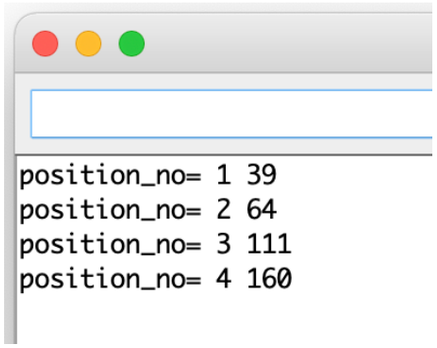
このプログラムを書きこんで、ボタンを長押ししたり連打したりして動きを確かめてみよう。
操作をやり直す時は、arduinoのリセットボタンを押すといいよ。

・１、２、３、４の順にサーボの値を書きこんでみよう

さあ、あとはposition_noの変化に合わせてサーボを動かすだけになった。
でもいきなりサーボモータを動かすのではなく、ちゃんとシリアルモニタでサーボモータ書き込む値を確認してみよう。

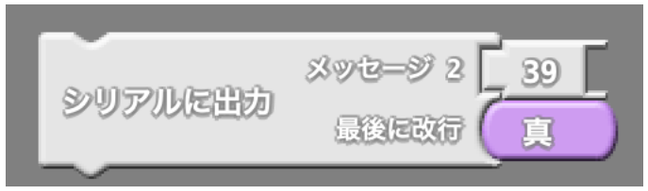


実行結果はこんな感じでちゃんとサーボモータに書かないといけない数値がでてきたね！

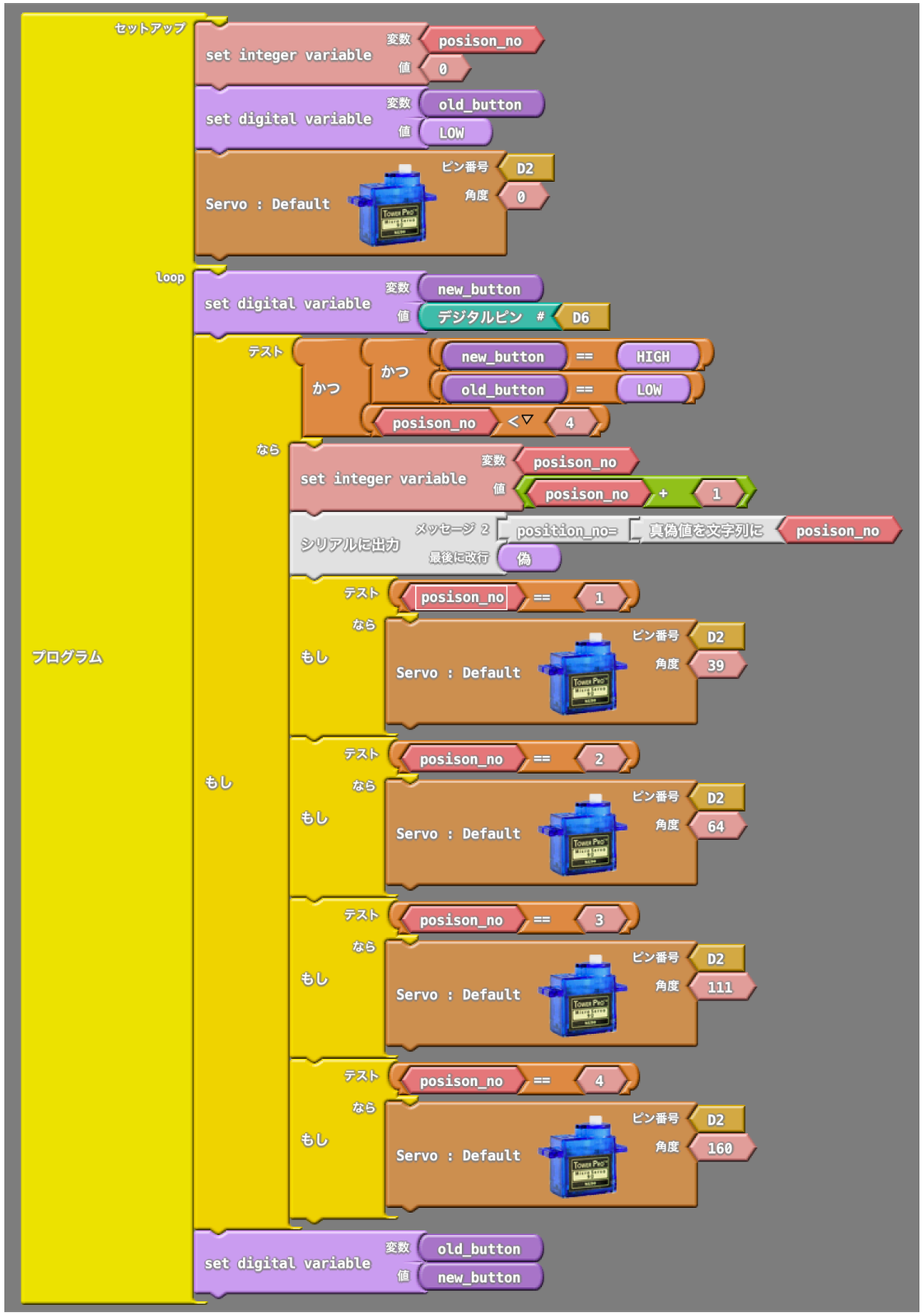


```
position_no= 1 39
position_no= 2 64
position_no= 3 111
position_no= 4 160
```

あとは、シリアルに出力の部品をサーボに置き換えればプログラムは完成だ。

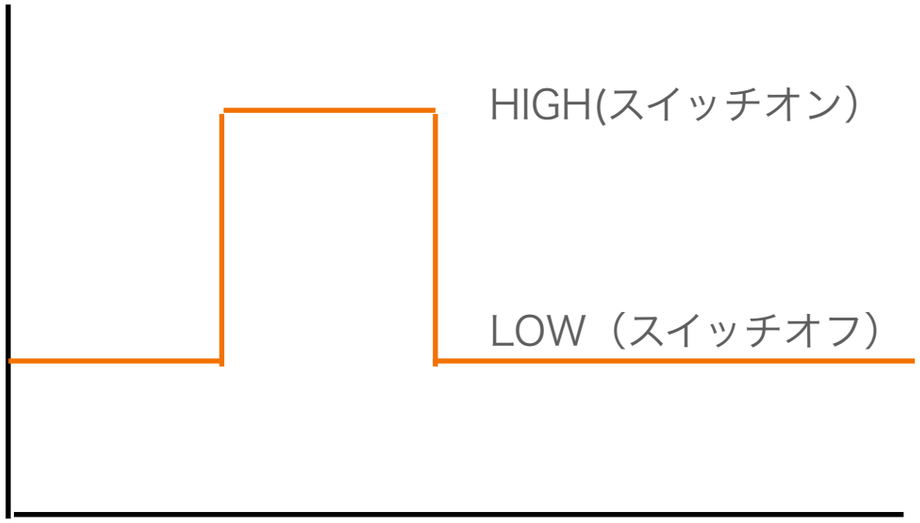


完成した輪ゴムガトリングガンのプログラム

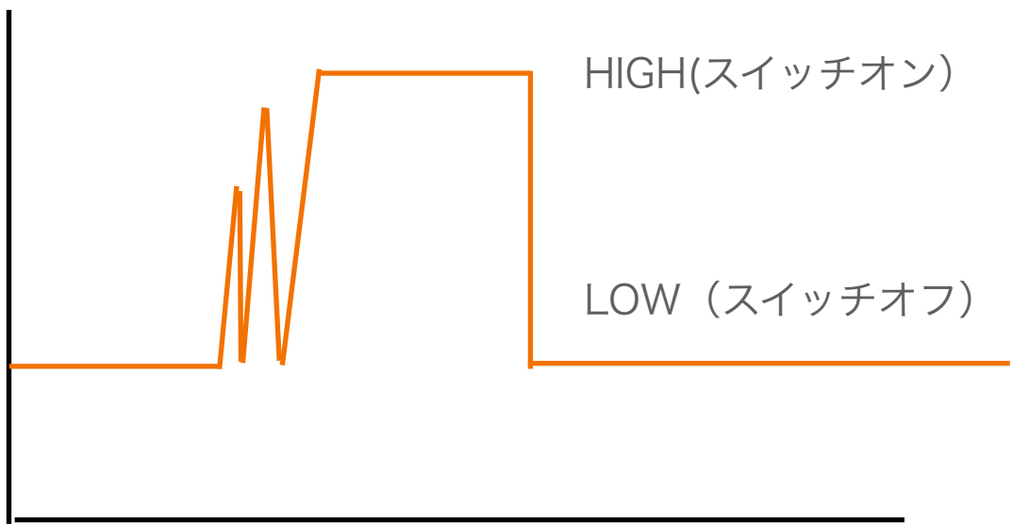


　先ほどのプログラムは、スイッチの「チャタリング」という現象に対応できてないので、「もし部品」の「なら」の一番最後に0.2秒くらいの待ち時間を入れると安定して動く。

チャタリングというのは、スイッチの電気接点が接触する瞬間に入るノイズで、電気信号の実際というのは下のイメージではなく、



　こんな感じで、瞬間的にHIGHとLOWを繰り返したあとにHIGHになるためなんだ。



その最初のバタバタした部分を避けるために待ち時間を少し入れると良いんだよ。

　さあ、ここまでの内容で、最低限ArduinoをArdublockで動かすための学習は終わった。
　次からはもうちょっとロボット制御らしいことを学んでいくよ。

巻末資料

・ガトリングガンのプログラム