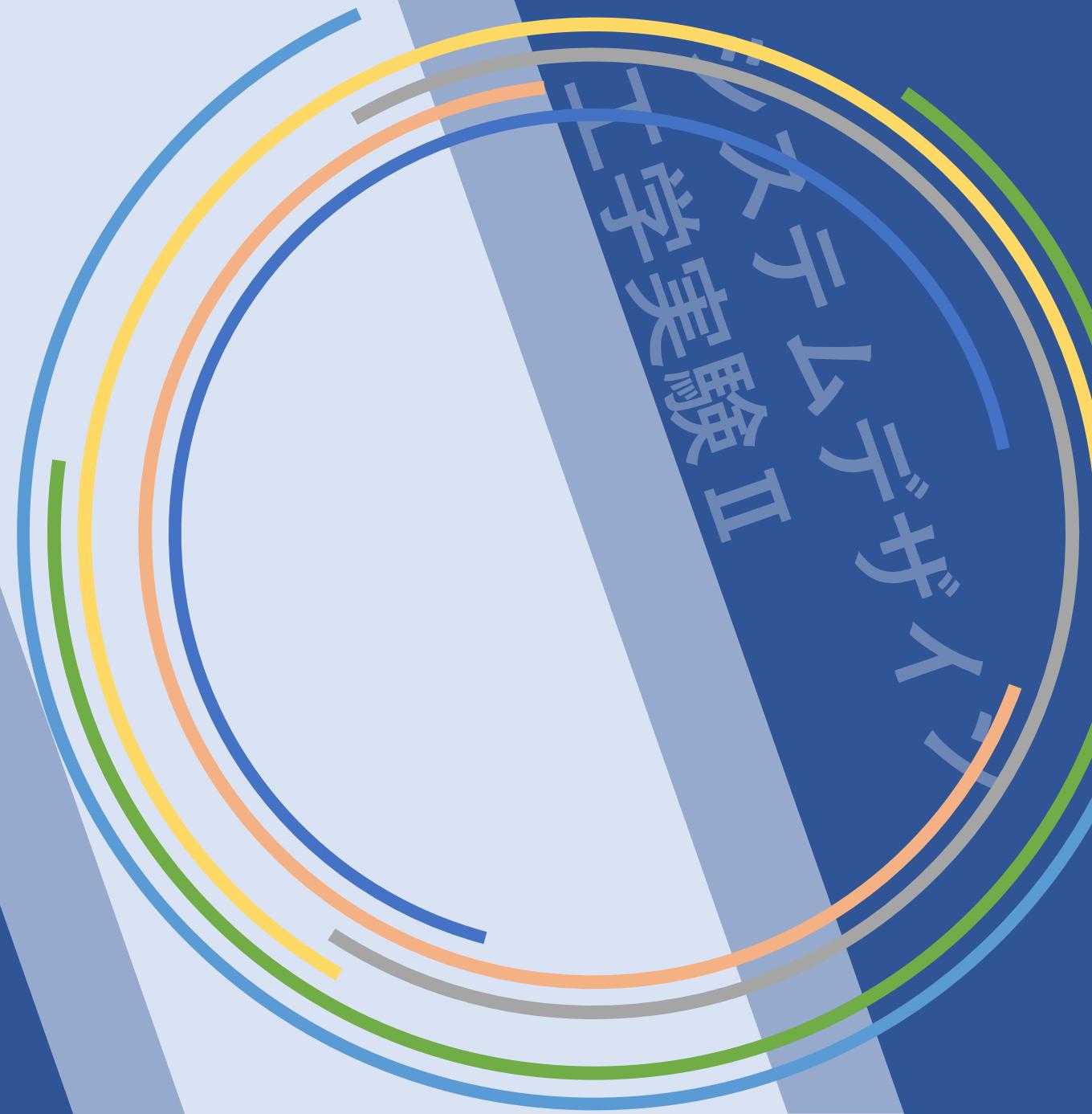




# Internet of Things

---

担当： 西 宏章



この授業に関する情報はLSMで取り扱います  
質問やレポートに関する質問などはSlackへ

<https://keio-sdexp-iot.slack.com>



# Slackについて

- 次の2つの専用チャネルと一般話題のチャネル(#general, #random)があります

- 質問箱

- 質問し貢献して下さった方、対応した方を敬います。救われる学生は必ずいます
- 「匿名じゃないから質問できない」というのは違います
  - 通常の質問は、対面で行い相手がわかります。西は匿名にはなりません。フェアではありません。
  - 質問する側にも、例えば正しく伝えるといったある程度の**責任と準備**が必要です。

- サポート

- 連絡やヒント、参考事項など、様々な追加情報が掲載されます
- 特に重要な内容はLSMでも掲載するようにしますが、基本的にはこちらで情報を公開します
- 質問など利用形態は成績に一切影響ありません。初歩的な質問・疑問でも構いません。

- ヒント共有を含め、Slack内での学生間やり取りは評価に影響せん

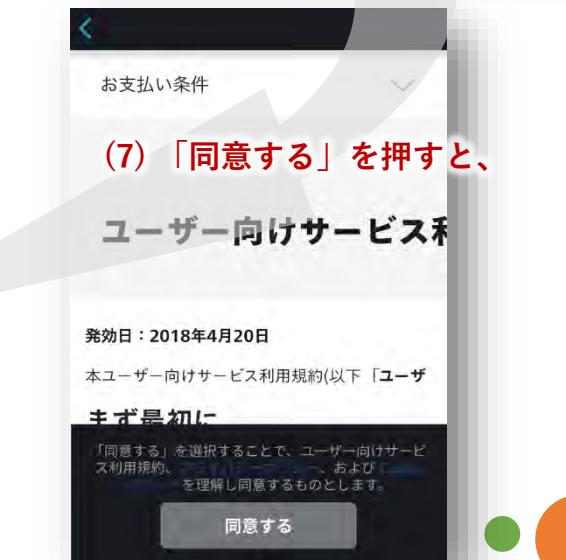
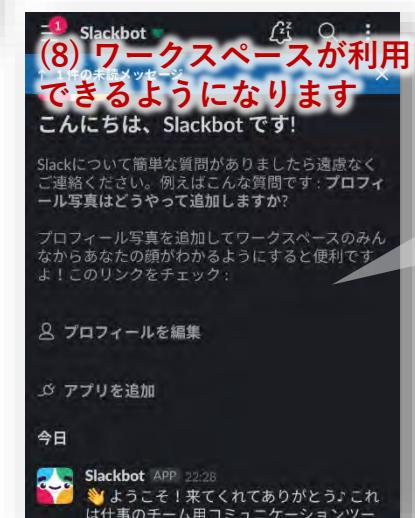
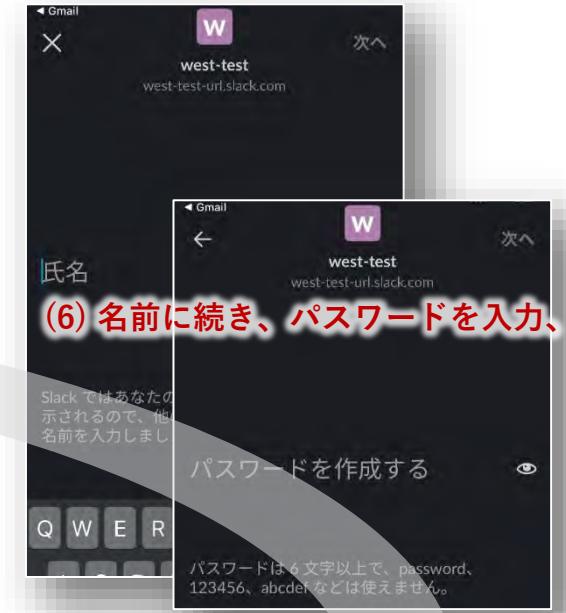
- 問題があった場合は、内容を変えるか、記述を説明の上削除すれば済む話です
  - Slackでの書き込みは基本的に剽窃ではないと考え、見守り、問題となった時のみ上記の対応を行います

- 是非活用してください

- Slackが使えない情報系技術者に仕事はない
  - Slackが使えないITベンチャーは存在しない。調べてみると良い。



# Slackアカウントを作成しよう



## keio.jpアカウントを用いたSlackによる授業ワークスペースへの参加の仕方

- keio.jpアカウントが必要です
- 西担当の授業はすべてSlackを利用します



実験内容を頭に入れておこう  
**実験内容と手順**

# IoTを学び作る — 実験の流れ

6

- Arduinoハンズオン
  - 多くの学生はすでに高校で扱っていることが多い
  - Raspberry PIを使って実習するのが筋ですが（検討中）
- 実際にみなさんがやること
  - 講義を聞く→実際に回路を作る→プログラムを書く→動作させる！
- IoTとは何か
  - IoTは技術もサービスも含む
    - Hiroaki Nishi, 2016
  - IoTとはシステムを考えることである
    - Masanori Takeshita, Fumi Koda, First IoT project textbook, 2016
  - “IT does not matter”
    - Nicholas G. Carr, Harvard Business Review, 2003
  - IoT技術には気づかなくなるぐらい身近になる



The Internet of Everything (IoE) – Aamer Azeemi, Cisco, 2013





# 古くからの理想、技術進歩で実現へ

- 技術的進歩 (“IoT” by Kevin Ashton in 1999)

- 電子回路、電気回路の製造技術進歩
- 無線通信の進歩 4G/LTE/5G, Wi-Fi, BLE
- バッテリ性能の向上 LIBなどの普及・LIBの管理もIoT
- 取り扱いが簡単に Linux搭載・IoTプラットフォーム
- 小型化 回路を直接触って作ることが難しくなっている
- 低価格化・高性能化・信頼性向上
- 様々な利用例・応用例
- ケーブルレス（無線も電源も）

- 様々な用途

- 趣味、研究、サービス
- 実世界に近い場所で情報を扱う = 新鮮





# センシング

- 2022年にセンサ1兆個
  - - Trillion sensor summit 2013
- 温度
- 湿度
- 照度
- エネルギ消費
- ガス(CO<sub>2</sub>, SO<sub>x</sub>, NO<sub>x</sub>)
- 匂い
- 圧力
- 赤外線(行動検出など)
- 風向、風速など



IoTとは?(英訳)  
– MONO WIRELESS, [http://mono-wireless.com/jp/tech/Internet\\_of\\_Things.html](http://mono-wireless.com/jp/tech/Internet_of_Things.html)

物理量  
現象  
行動

アナログ

デジタル

電圧  
データ





# 分析（データシステムの知能化とデザイン）

- データの処理には3段階ある - Hideyuki Tokuda, 2016

- 1 : 視覚化
- 2 : 最適化
- 3 : 予測

- 統計

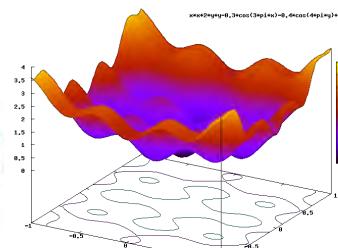
- 機械学習

Visualization



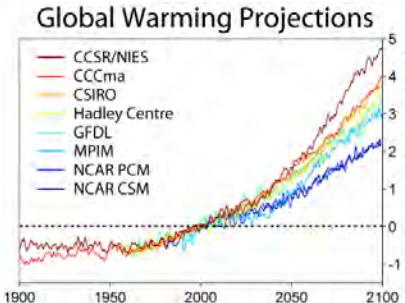
[https://upload.wikimedia.org/wikipedia/commons/9/9b/Social\\_Network\\_Analysis\\_Visualization.png](https://upload.wikimedia.org/wikipedia/commons/9/9b/Social_Network_Analysis_Visualization.png)

Optimization

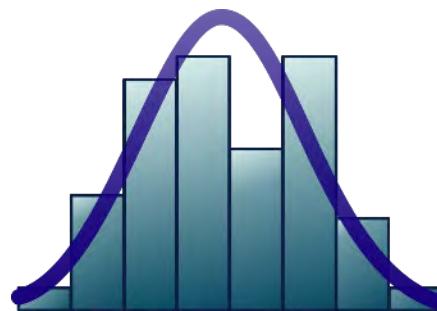


[https://upload.wikimedia.org/wikipedia/commons/7/7a/B2\\_optimization\\_function.png](https://upload.wikimedia.org/wikipedia/commons/7/7a/B2_optimization_function.png)

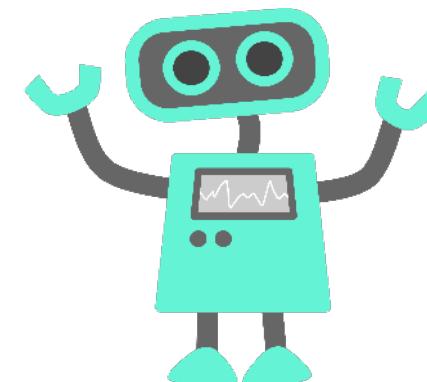
Prediction



[https://upload.wikimedia.org/wikipedia/commons/a/aa/Global\\_Warming\\_Projections.png](https://upload.wikimedia.org/wikipedia/commons/a/aa/Global_Warming_Projections.png)



[https://en.wikipedia.org/wiki/Wikipedia:Meeup/DC/Statistics\\_Edit-a-thon#/media/File:Fisher\\_iris\\_versicolor\\_sepalwidth.svg](https://en.wikipedia.org/wiki/Wikipedia:Meeup/DC/Statistics_Edit-a-thon#/media/File:Fisher_iris_versicolor_sepalwidth.svg)



<https://www.goodfreephotos.com/albums/vector-images/blue-robot-vector-art.png>



<https://pixabay.com/en/learn-note-sign-directory-897410/>





# IoT的なサービスの例

- なんでもかんでも IoT 化

- 環境制御 – スマートシティ / タウン / ビルディング / ハウス
- エネルギー制御 – HEMS / BEMS • HVAC 制御 • 系統電力制御
- 交通 – GPS • 自動運転 • 交通制御 • 環境改善
- 犯罪防止, セキュリティ – 警報 • ライト • カメラ
- 災害 – モニタリング • 避難指示
- 医療健康 – 遠隔診察
- 農業 – モニタリング • リコメンデーション • 農薬散布
- 設計 • 住居構造 – 最適化, 生命化建築

極めて SD 的



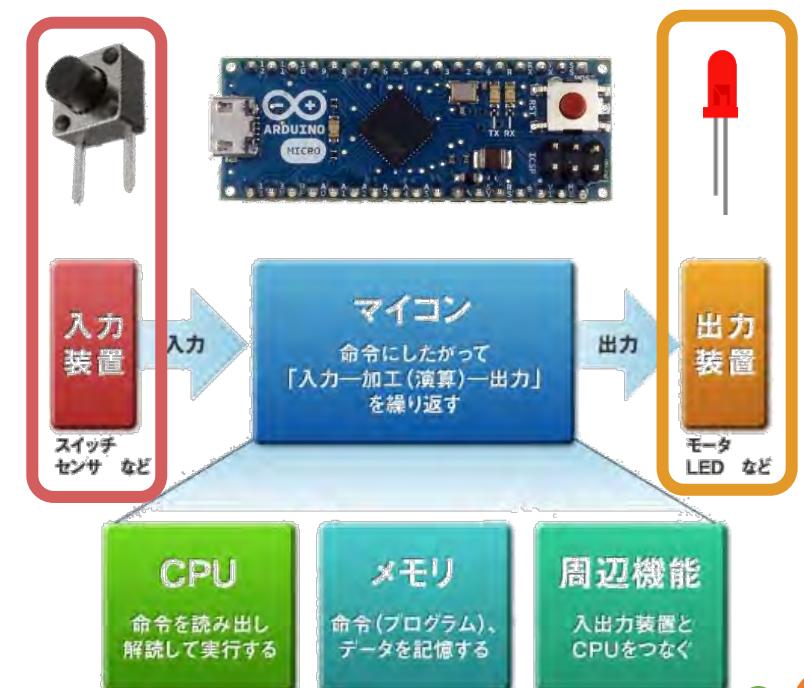
Images: The Internet of Everything, Aamer Azeemi, Cisco,  
2013



# マイクロコントローラArduino

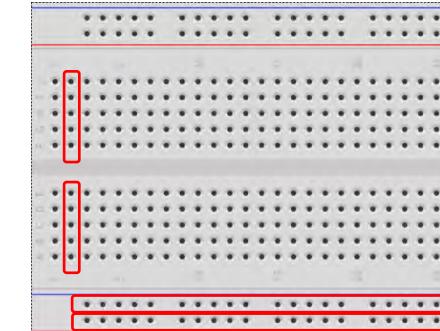
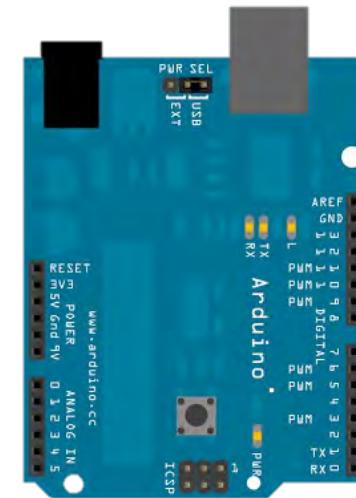
11

- マイコンの一種
  - マイコン=マイクロ・コンピュータ マイクロ・コントローラ
- マイコンチップは500円程度で購入可能
  - ATMegaシリーズは家電やロボットなどあらゆる電気機器で大量に利用されている
  - 周辺デバイスを合わせて安いもので1,500程度から
  - 20年前のPCよりも高性能
- 本来はマシン語（数値だけの言語）で設計
  - 高級言語による簡単設計ができるように工夫されている
- デバイス本来は駆動能力が弱く直接素子を扱えない
  - 使いやすいようにI/O周辺が強化されている
  - 動作周波数はトレードオフで遅いが、IoT用途では充分な性能
- 低消費電力化が可能
  - スリープモードなどはきちんと搭載
  - 様々な電圧に対応できるように工夫・そして壊れにくく



# 回路を作るために

- Arduino上のピン:各ピンに役割があるため適切なピンへ配線
- ブレッドボード:ワイヤの抜き差しをするだけで回路を作成
- LEDなどの素子には極性がある
- 電源の配線はプラスが赤、マイナスが黒
  - プラスマイナスを逆にすると素子などを壊しかねないので注意
- 丁寧に扱うように
- 前準備（すべてシミュレーターで確認できる）
  - Arduinoの基礎と設計方法を学ぶ
  - LEDをチカチカさせる
  - スイッチやボタンを押すと点灯・点滅する回路
  - 可変抵抗でぼんやり光る回路
  - サーボモータを動かす回路



内部はすべて結線されている

# Tinkercad.com Arduino Simulator

- Autocadの公式サイト (<https://www.tinkercad.com/>) にアクセス
- 右上の[今すぐ参加]をクリック
  - すでにアカウントを持っている場合は[サインイン]よりログイン
- [パーソナルアカウントで作成]をクリック
- Googleでサインインを選択
- Keio.jpアカウントでログイン
- 回路を選択
- 新しい回路を選択
- スターターからArduinoを選択
- 一番上のブレッドボードを選択して配置

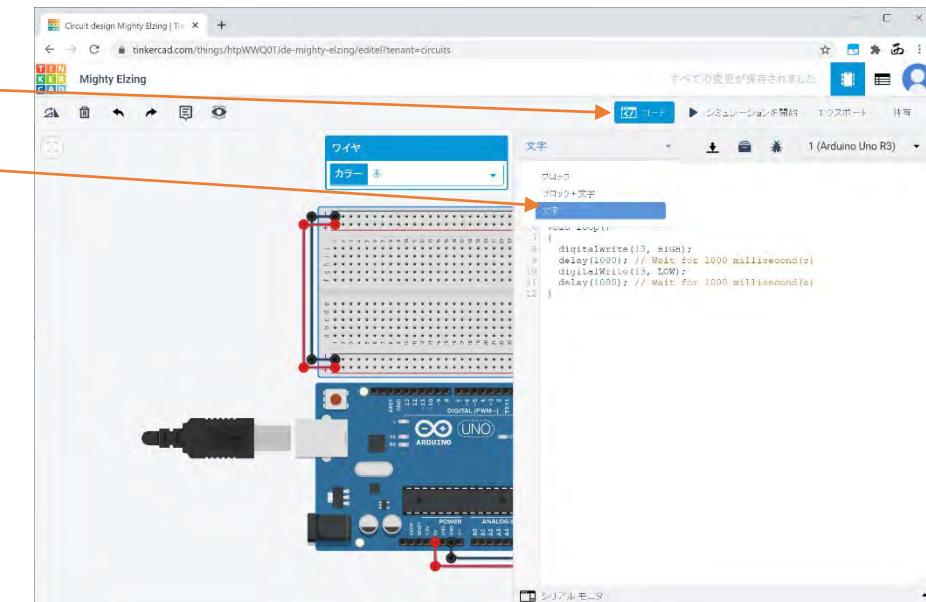




# Arduino Simulatorでコードを書く

14

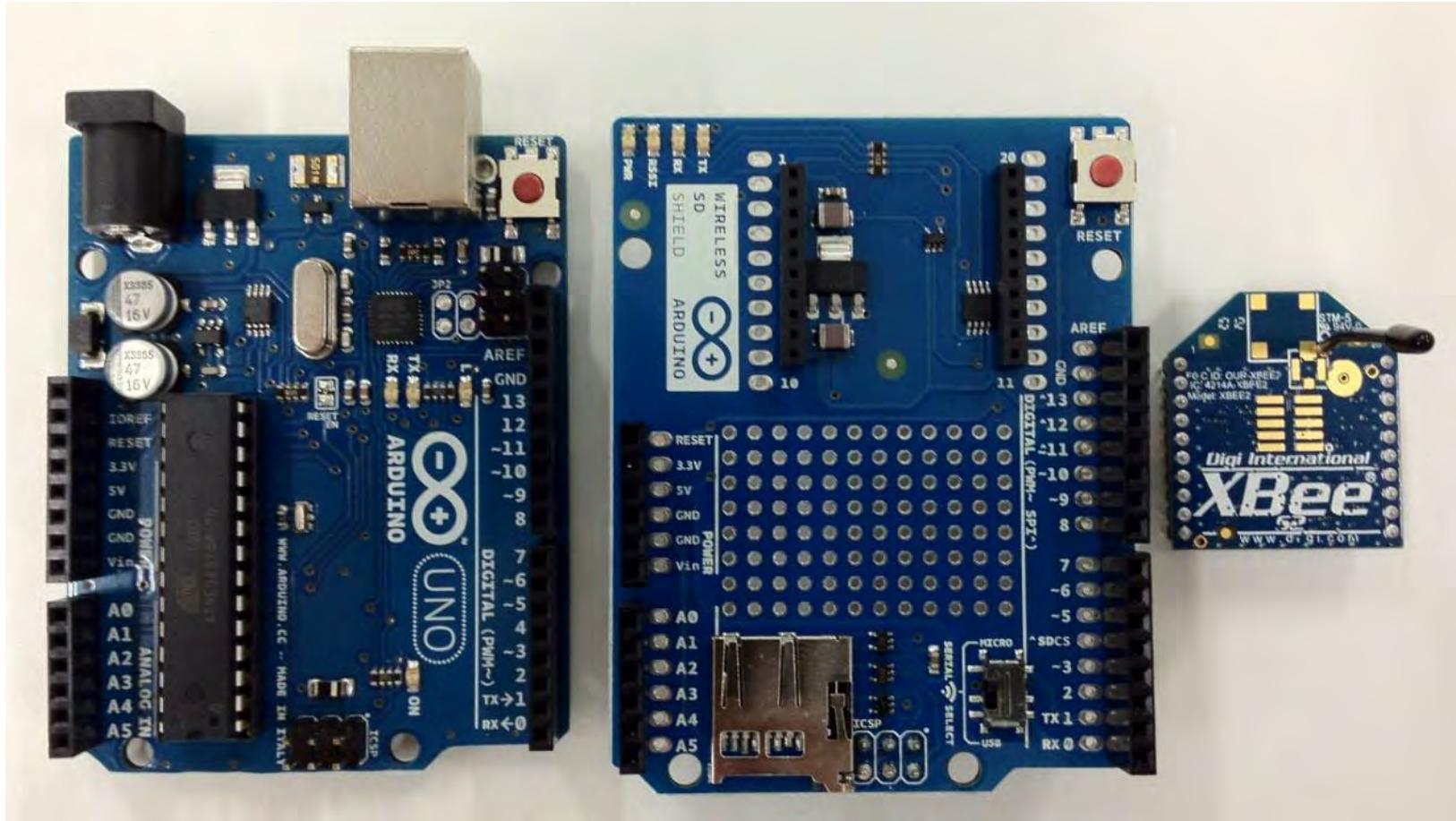
- コードを選択
  - 文字を選択するとプログラムが記述できる
  - 実際には実機を複数個用いて実験する
    - 事前学習ではシミュレータが扱えれば充分
  - プログラミング言語はC++由来ですが独自です
    - マルチメディアデザインで用いるProcessingも同様です  
文法体系とインターフェースが同じです
    - 実はArduino IDEはProcessingの設計環境を基に作られた
    - ifやforなど基本はプログラミング演習を習得していれば問題なく理解できるはず
      - C言語系であるため、行の最後にセミコロン;が必要
  - 今回はアルゴリズムを学ぶのではなく、手順を学ぶので、複雑な構文は用いない
    - 情報系授業では、「プログラミング演習」と「データシステムの知能化とデザイン」でPythonを、「IoT」と「マルチメディアデザイン」でProcessing由来のJavaライクな言語を扱う



# キットのパート 1/3

15

- Arduino UNO, Arduino shield, XBee

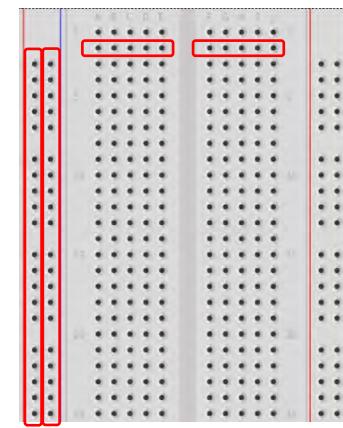
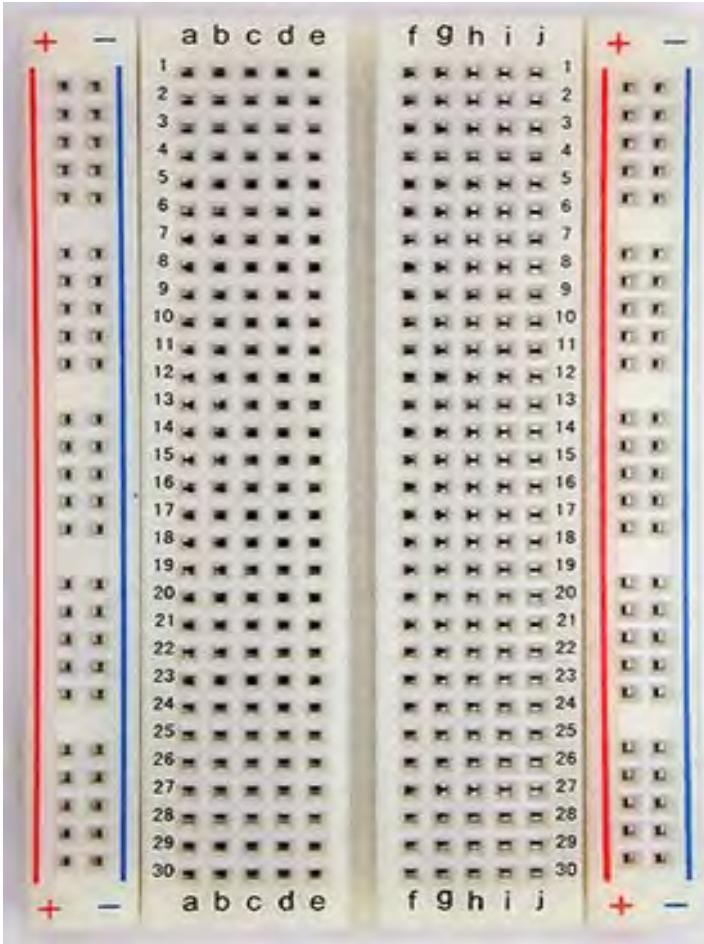




# キットのパート 2/3

16

- ブレッドボード



内部はすべて結線されている

- ジャンパケーブル



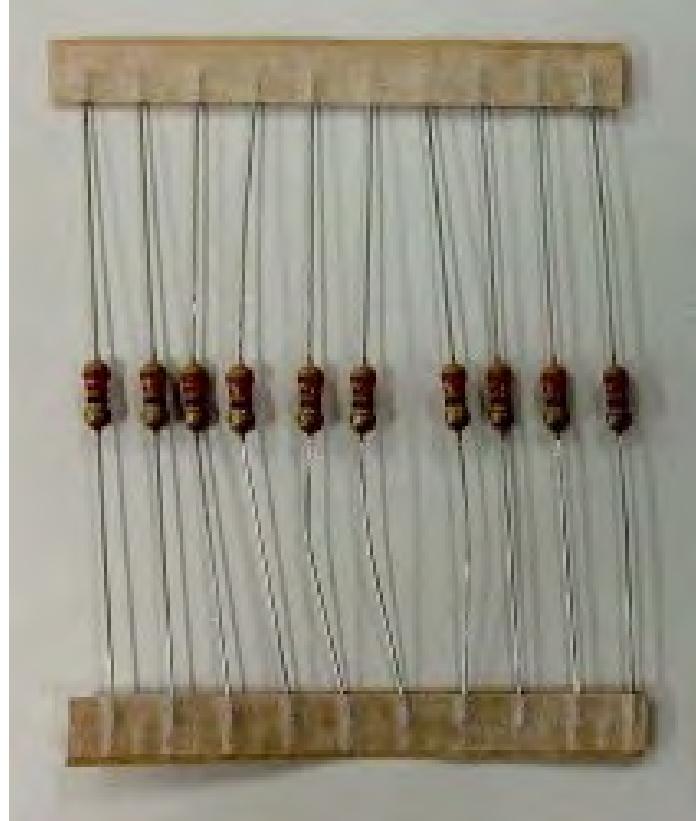


# キットのパーツ 3/3

17

- 抵抗

- 今回は $10\text{k}\Omega$ (茶黒橙金)

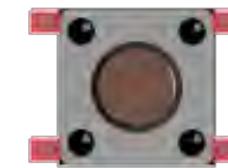


- フルカラーLED

- 最長は黄色ケーブルに、その他は抵抗に、つなぐ



- ボタン



- その他にもサーボや

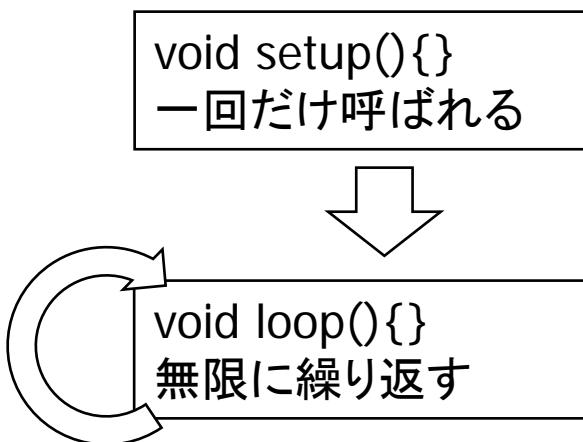




# (1) LEDチカチカ(Lチカ)の設計 digitalWrite

18

- 右図のように配線、抵抗は330Ωに設定、抵抗値によって明るさが変わる  
カラーコードは（橙橙茶）
- Arduinoのコードも入力する
- シミュレーションを開始すると  
毎秒点滅する
  - エラーの場合はプログラムを確認
  - 回路のエラーは教えてくれない



TINKER CAD LED-Blink すべての変更が保存されました

コード シミュレーションを開始 エクスポート 共有

LED Blink

文字 1 (Arduino Uno R3)

```
int led = 2;
void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

```
int led = 2;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```





# Lチカの流れ

19

int led = 2; 初期設定

ピンledをOUTPUT  
(出力) にする

↓  
pinMode(led, OUTPUT);

digitalWrite(led, HIGH);

ピンledで  
HIGH(5V)出力

1000ms待つ

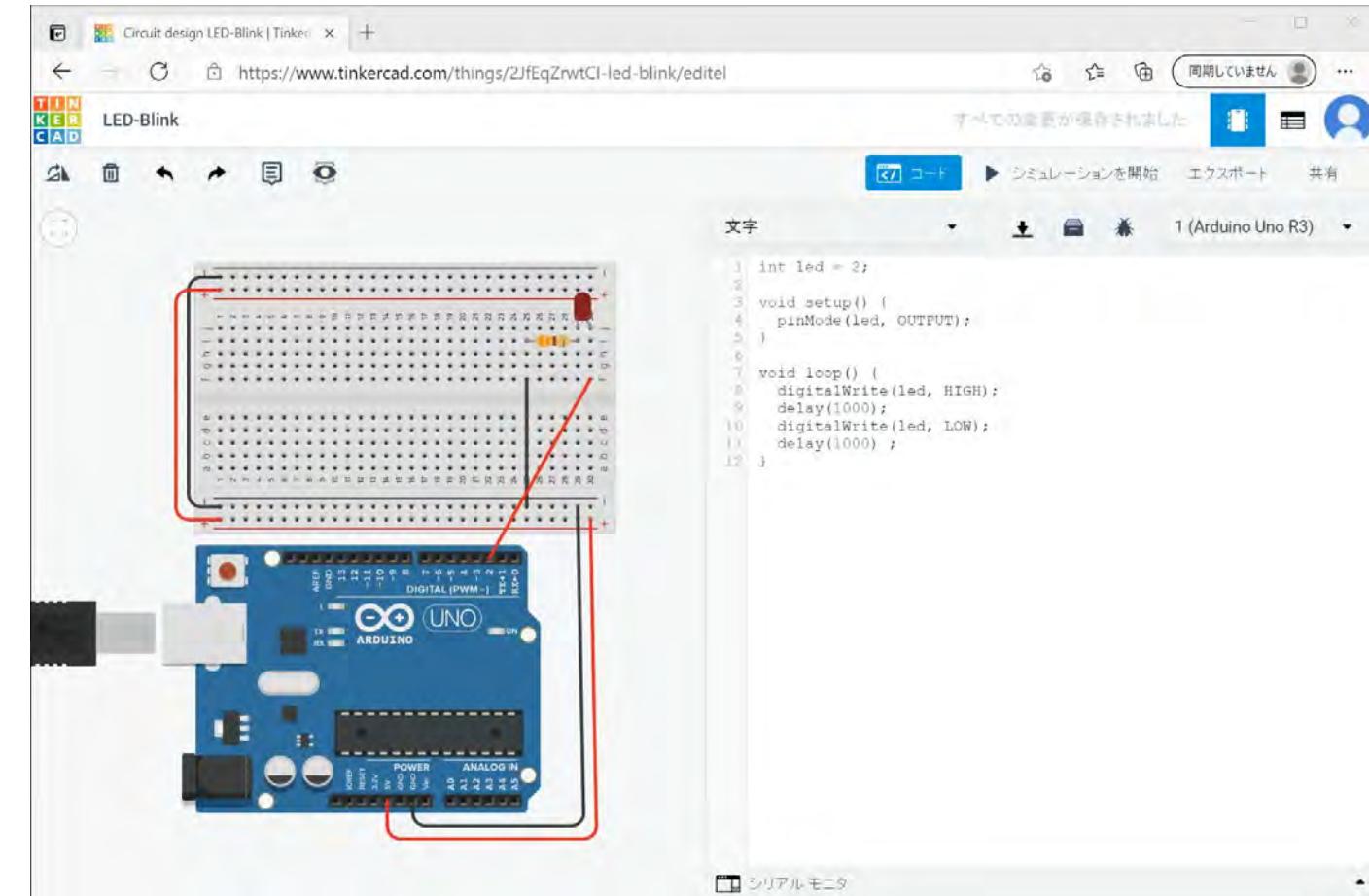
delay(1000);

ピンledで  
LOW(0V)出力

1000ms待つ

delay(1000);

digitalWrite(led, HIGH);





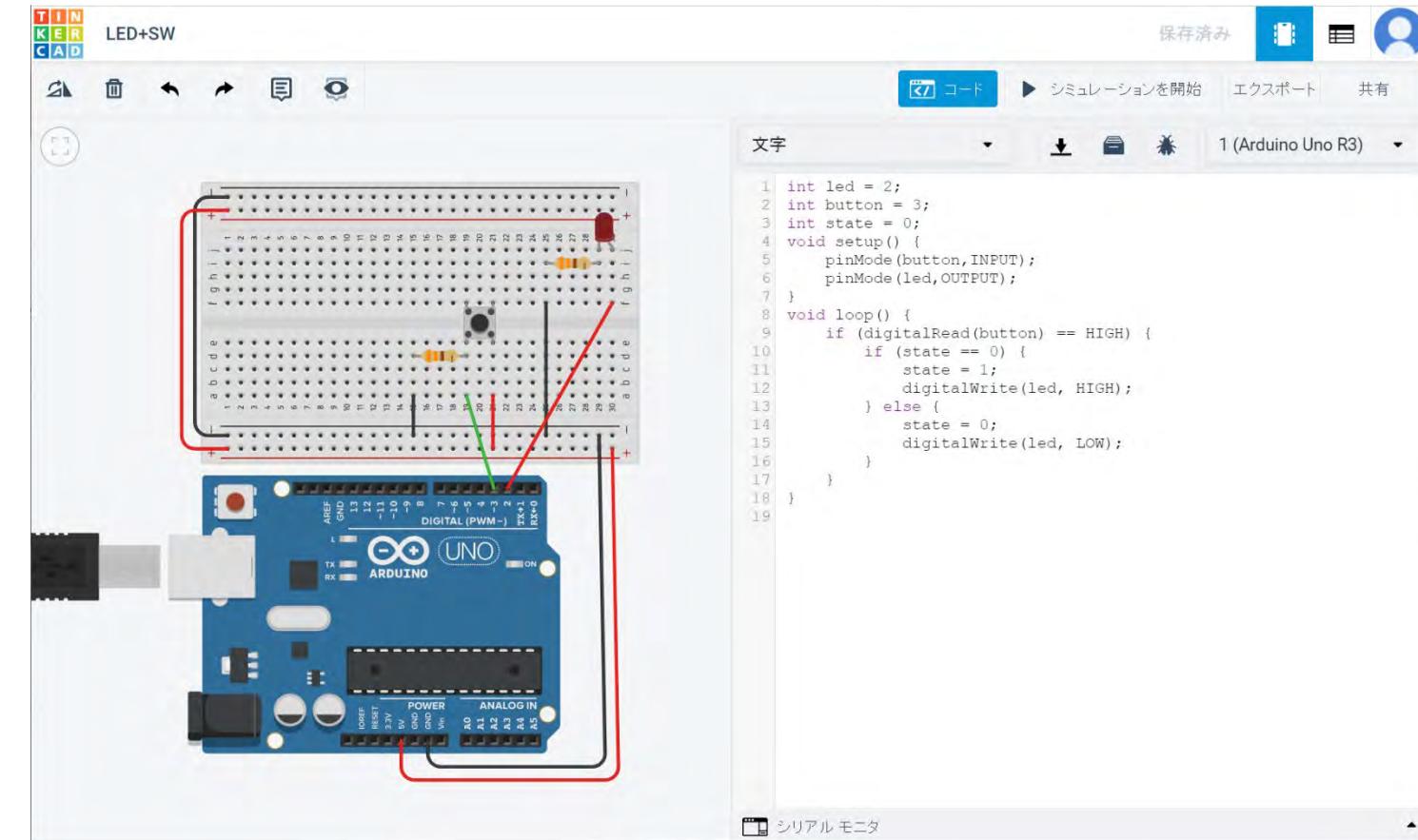
## (2) ボタンで明滅を制御 digitalRead

20

- 右図のように配線
  - 抵抗はすべて330Ω
- Arduinoのコードは後述
- シミュレーションを開始
- 仕様上はボタンを押す毎に、点灯と消灯を繰り返すはず
  - だが、実際にはうまく動作しない
  - 実機でもうまく動作しない
  - このうまく動作しないということをきちんとシミュレートする



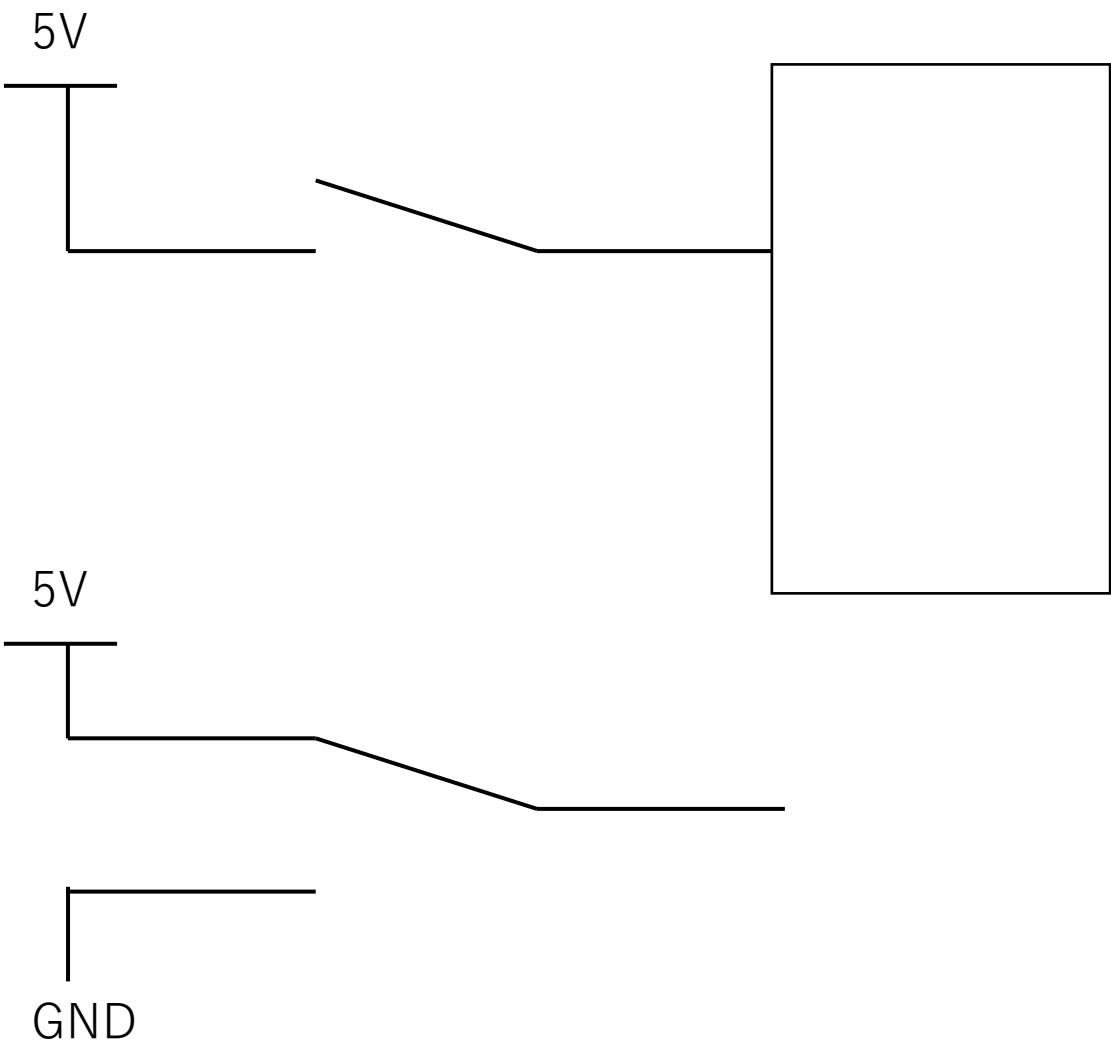
TINKER CAD LED+SW



```
1 int led = 2;
2 int button = 3;
3 int state = 0;
4 void setup() {
5     pinMode(button, INPUT);
6     pinMode(led, OUTPUT);
7 }
8 void loop() {
9     if (digitalRead(button) == HIGH) {
10         if (state == 0) {
11             state = 1;
12             digitalWrite(led, HIGH);
13         } else {
14             state = 0;
15             digitalWrite(led, LOW);
16         }
17     }
18 }
```

シリアル モニタ





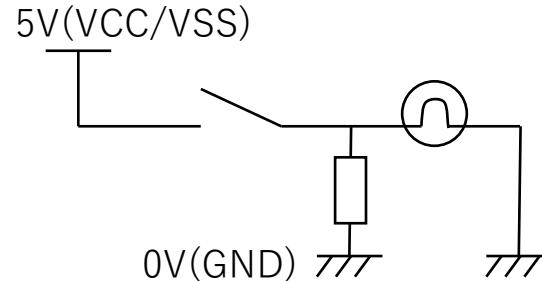
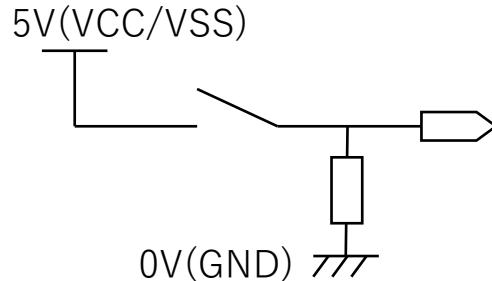
WestLab SD/Keio



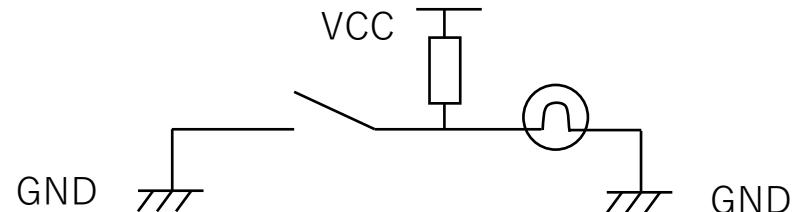
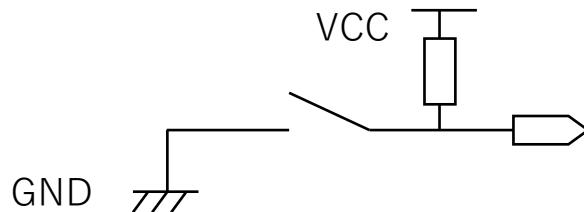
# Active High(正論理)とActive Low(負論理)

22

- ボタンを押すと5Vになる、5Vにするとランプがつくなどは正論理(Active High)



- 逆は負論理(Active Low)で、押すと0Vになり、0Vにするとランプがつく



- この実験では正論理を扱う

- どちらでもよいが、ボタンは押している時間<<離している時間なので、消費電力を考慮して不論理にするのが一般的であるが、正論理のほうが理解しやすい
- 抵抗は1Kから100K程度を使う
- 抵抗値があまりにも大きいとノイズに弱くなり、抵抗値が小さいと電力消費が増える
- つまり、トレードオフの関係にあるが、実験では1KΩを利用する



# コードと動作

23

```
int led = 2;
int button = 3;
int state = 0;
void setup() {
    pinMode(button,INPUT);
    pinMode(led,OUTPUT);
}
void loop() {
    if (digitalRead(button) == HIGH) {
        if (state == 0) {
            state = 1;
            digitalWrite(led, HIGH);
        } else {
            state = 0;
            digitalWrite(led, LOW);
        }
    }
}
```

The screenshot shows the Tinkercad interface with a project titled "Circuit design LED+SW". The breadboard circuit is connected to an Arduino Uno. The breadboard has a red power rail at the top and a black ground rail at the bottom. A digital button is connected between pin 3 and ground. A digital LED is connected between pin 2 and 5V. The Arduino Uno is connected to a computer via USB. The code editor on the right shows the following Arduino sketch:

```
1 int led = 2;
2 int button = 3;
3 int state = 0;
4 void setup() {
5     pinMode(button,INPUT);
6     pinMode(led,OUTPUT);
7 }
8 void loop() {
9     if (digitalRead(button) == HIGH) {
10         if (state == 0) {
11             state = 1;
12             digitalWrite(led, HIGH);
13         } else {
14             state = 0;
15             digitalWrite(led, LOW);
16         }
17     }
18 }
```

The "シリアル モニタ" (Serial Monitor) tab is visible at the bottom.





# ピンのプルアップ、プルダウン

24

- INPUT\_PULLUPを使うとプルアップ抵抗を省略出来る
- デジタル入出力ピンで押しボタンスイッチなどを繋ぐ場合、押されていない時の入力電圧を作るためプルダウンあるいはプルアップ抵抗が必要です
  - なぜ必要かは動画のActive LowやActive Highの議論が理解できている必要があります
  - プルアップ抵抗はマイコンチップ内部にすでに実装されており、これをピンモードの設定でINPUT\_PULLUPキーワードを使用すると利用できるようになります
  - つまり、外部のプルアップ抵抗を省略できます
- 使用例
  - pinMode(2,INPUT\_PULLUP);
  - 2番ピンがマイコン内部の抵抗を介して電源レベルに繋がるため、2番ピンに直接ボタンを繋ぎ、ボタンのもう片方をGNDにつなぐだけでよい
  - digitalRead(2)とすることで、ボタンが押されていない時にHIGH、押された時にLOWになる
    - つまり、この場合はActive Lowとなる、Pull Upだから当然ですよね

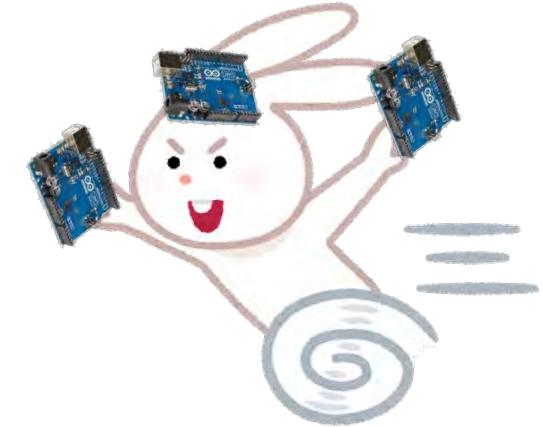




# なぜうまく動作しないのか？

25

- Arduinoマイコンの動作速度が速すぎるから
  - 搭載されているAVRマイコンATMegaの動作周波数は16MHz
    - 1600万回計算/秒、これでも非常に遅いほうで、PCは3GHzを簡単に超える
  - すると、チャタリング現象が発生する
    - ボタンを押す
    - StateとLEDの明滅が変わる
    - 人間は1600万回/秒で操作できないため、まだボタンが押されている
    - しばらく2と3を繰り返す
- ありきたりな解決法として、ボタンを押してから少し待つとよい（遅延させる方法）
- 待つ命令はdelayで実行すると指定時間何もしない
  - delay(N); → N ms待つという意味。 (1000ms=1秒)
  - Nのとるべき値について考え、さらにdelay(N);を正しい場所に追加してチャタリングを防止する
  - 各自演習課題として試しておくこと（次のページ）
- より優れた方法がある
  - エッジ検出する方法もあるが、結局追加でメカニカルなチャタリングを防ぐため遅延は必要





# 演習問題

- チャタリングを防止する
  - 先のソースコードに例えばdelay(500); を適切な位置に入れてチャタリングを防ぐ
  - どこに入れるとよいだろうか？
    - ボタンが押されたら待つ、と考えれば？
- ボタンでモードを切り替える
  - ボタンを押すたびに、点滅を行うモードと、消灯モードを切り替える
  - こちらはチャタリングが発生するだろうか？ 実行する前にコードを見て考えよう
- 作って試してみること

```

int led = 13;
int button = 3;
int state = 0;
void setup() {
  pinMode(button,INPUT);
  pinMode(led,OUTPUT);
}
void loop() {
  if (digitalRead(button) == HIGH) {
    state = ~state;
  }
  if (state == 0) {
    digitalWrite(led, LOW);
    delay(500) ;
  } else {
    digitalWrite(led, HIGH);
    delay(250) ;
    digitalWrite(led, LOW);
    delay(250) ;
  }
}

```





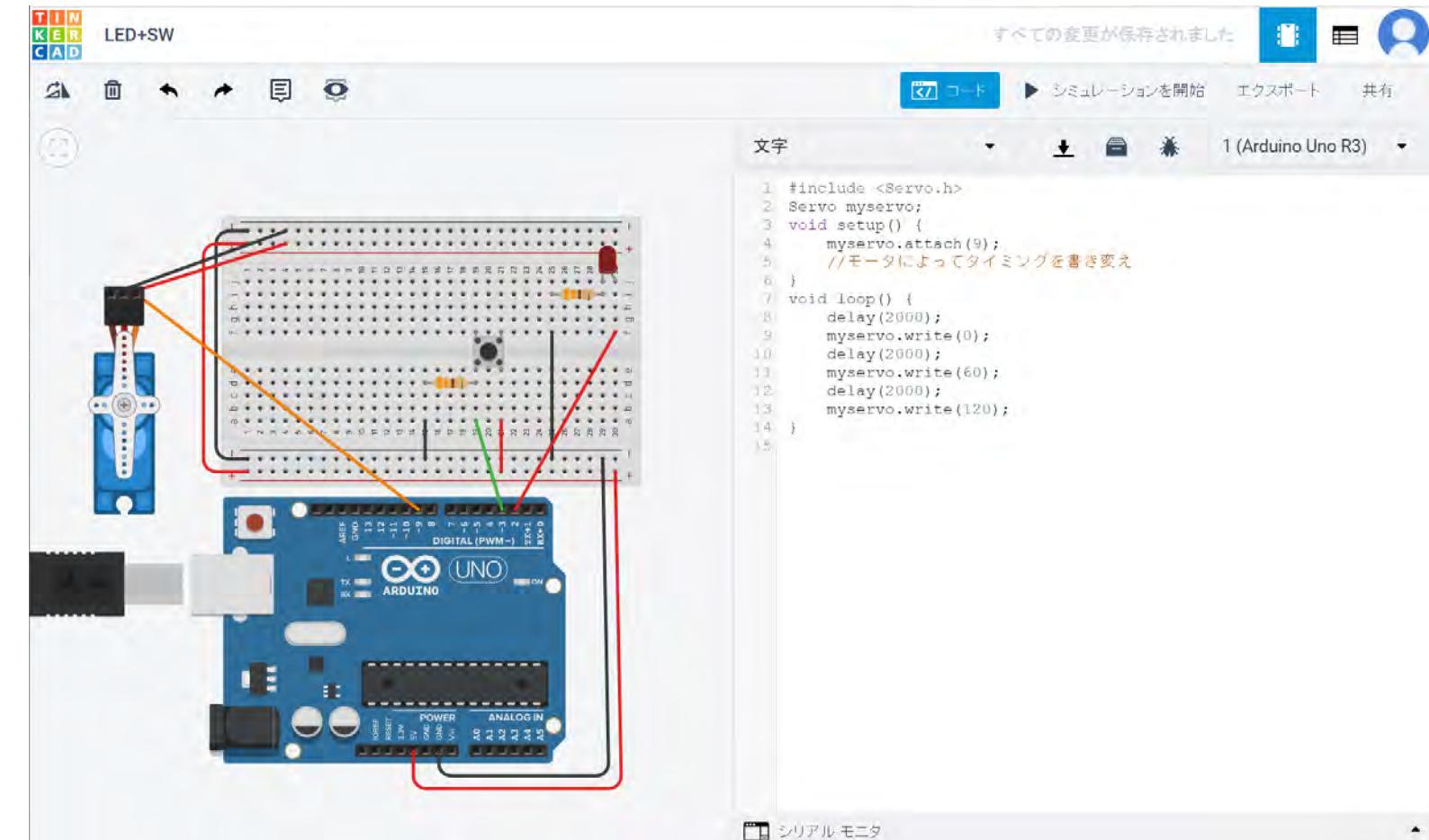
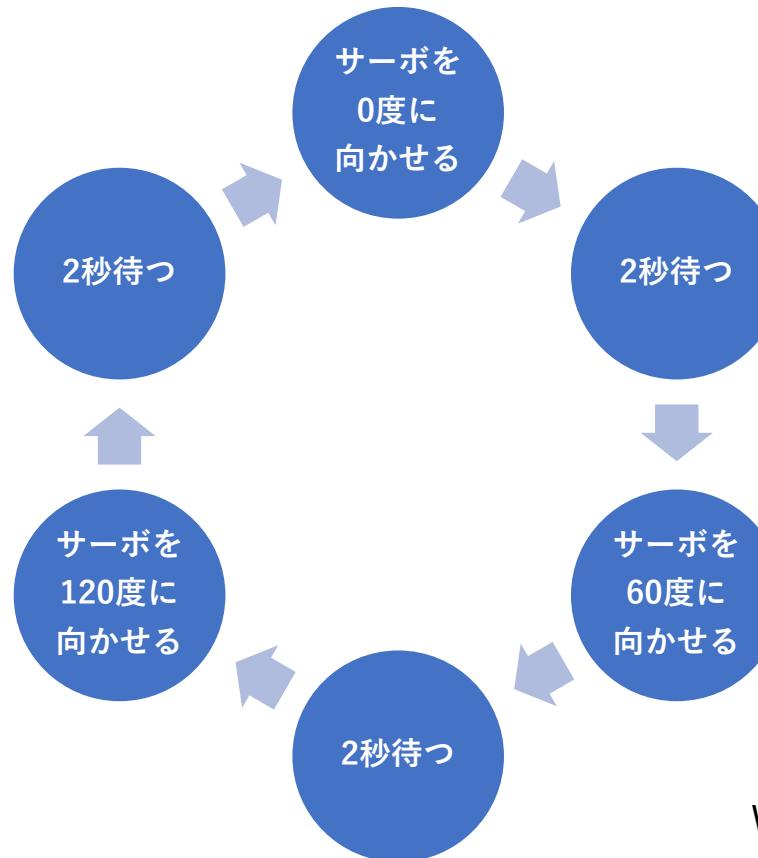
### (3) サーボモータを動かす

- サーボモータ
  - 本来は制御できるモータの意味、狭義ではラジコンなどで搭載されているPWM位置制御のモータ
  - 2種類ある
    - 回転サーボモータ：指定した速度で回り続ける
    - 位置サーボモータ：指定した位置（角度）まで移動して停止（魔改造で回転サーボになる）
- 実際のサーボと配線
  - 実際のサーボはギア機構がよく壊れる  
手では回してはいけない  
電源とGNDを間違えて接続しない  
無茶な角度指令値を入れない  
(ムギュと音を立てて回らない位置まで無理やり動こうとして壊れる)
- 3つの配線
  - プラス電源：赤色
  - マイナス電源：黒色や茶色など
  - 制御信号線：それ以外の黄色やオレンジなど



# 回路設計

- 右図のように配線
- 過去の回路は削除せず残しておくと後で再利用できる
- Arduinoのコードは後述
- シミュレーションを開始
- 先ほどの説明通りの動作を行う





# コードと動作

```
#include <Servo.h>
Servo myservo;
void setup() {
    myservo.attach(9);
    //モータによってタイミングを調整
    //例えば myservo.attach(9, 1500, 1900);
}
void loop() {
    delay(2000);
    myservo.write(0);
    delay(2000);
    myservo.write(60);
    delay(2000);
    myservo.write(120);
}
```

The screenshot shows the Tinkercad interface for a 'Circuit design LED+SW' project. On the left is a breadboard diagram with a servo motor connected to digital pin 9 of an Arduino Uno R3. A pushbutton is also connected to the breadboard. On the right is the code editor window displaying the following C++ code:

```
1 #include <Servo.h>
2 Servo myservo;
3 void setup() {
4     myservo.attach(9);
        //モータによってタイミングを書き換え
5 }
6 void loop() {
7     delay(2000);
8     myservo.write(0);
9     delay(2000);
10    myservo.write(60);
11    delay(2000);
12    myservo.write(120);
13 }
14 }
15 }
```

The code uses the Servo library to control a servo motor attached to pin 9. It sets up the servo and then enters a loop where it alternates between positions 0, 60, and 120 degrees, with 2-second delays between each position change.

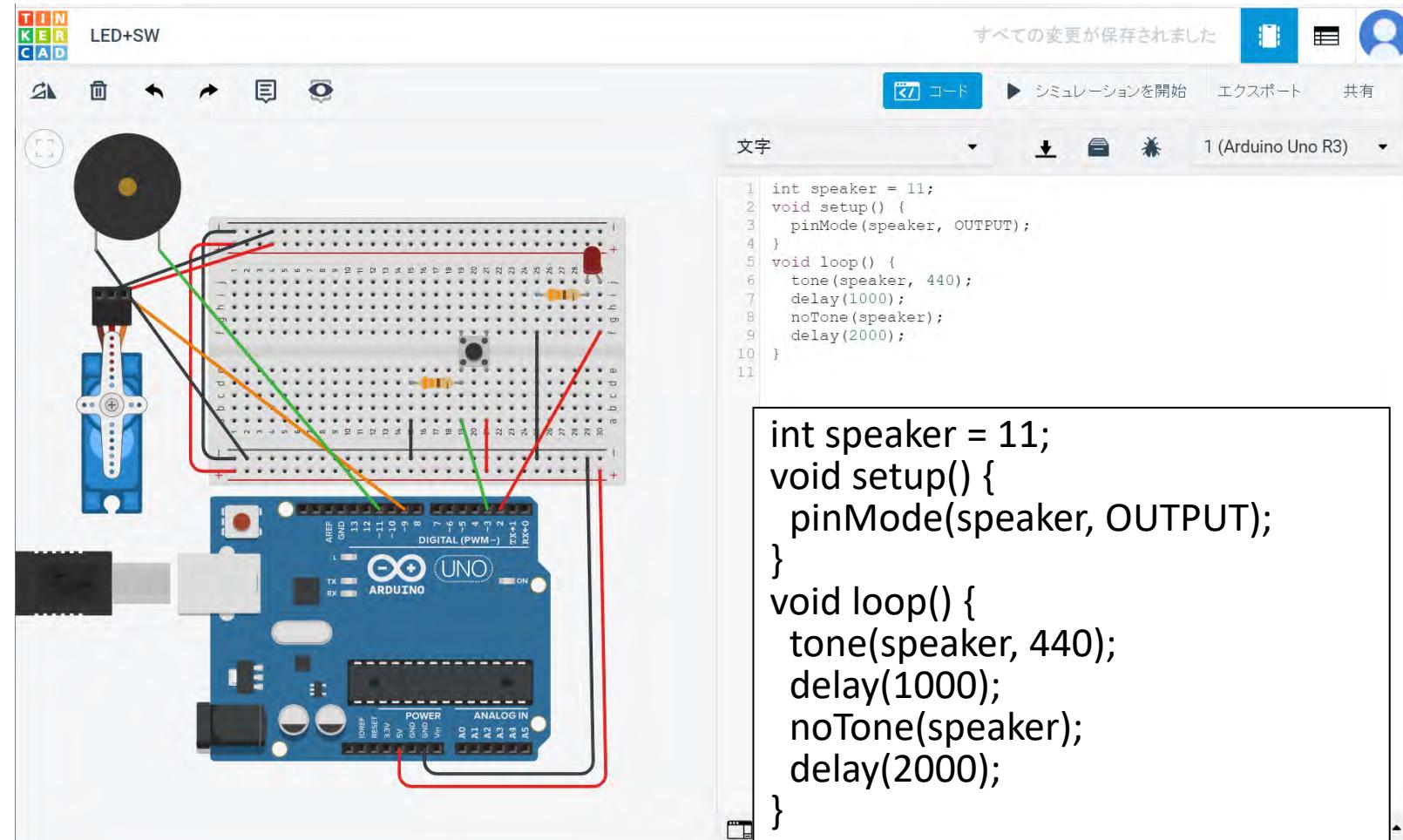




# (4) 音を鳴らす

30

- 音を鳴らす関数
  - `tone(pin, frequency);`
- 音を止める関数
  - `noTone();`
- 周波数と音階の関係は自分で調べること
  - 例えばラは440
  - ちゃんと音もなります
- 回路は消さずに追加している
- 例えば
  - 2つ並べて和音を作る
  - 曲を奏でさせる





# Toneの一例

```
#define NOTE_C3 131          #define NOTE_C4 262          #define NOTE_C5 523          #define NOTE_C6 1047  
#define NOTE_CS3 139         #define NOTE_CS4 277         #define NOTE_CS5 554         #define NOTE_CS6 1109  
#define NOTE_D3 1477        #define NOTE_D4 294         #define NOTE_D5 587         #define NOTE_D6 1175  
#define NOTE_DS3 156        #define NOTE_DS4 311         #define NOTE_DS5 622         #define NOTE_DS6 1245  
#define NOTE_E3 165          #define NOTE_E4 330          #define NOTE_E5 659          #define NOTE_E6 1319  
#define NOTE_F3 175          #define NOTE_FA 349          #define NOTE_F5 698          #define NOTE_F6 1397  
#define NOTE_FS3 185         #define NOTE_F4 349          #define NOTE_FS5 740          #define NOTE_FS6 1480  
#define NOTE_G3 1961        #define NOTE_FS4 370          #define NOTE_G5 784          #define NOTE_G6 1568  
#define NOTE_GS3 208         #define NOTE_G4 392          #define NOTE_GS5 831          #define NOTE_GS6 1661  
#define NOTE_A3 220          #define NOTE_GS4 415          #define NOTE_A5 880          #define NOTE_A6 1760  
#define NOTE_AS3 233         #define NOTE_A4 440          #define NOTE_AS5 932          #define NOTE_AS6 1865  
#define NOTE_B3 247          #define NOTE_AS4 466          #define NOTE_B5 988          #define NOTE_B6 1976  
                           #define NOTE_B4 494
```



# 動作

32

Circuit design LED+SW | Tinkercad <https://www.tinkercad.com/things/3NpFR3a9vn9-ledsw/editel>

LED+SW

すべての変更が保存されました

コード シミュレーションを開始 エクスポート 共有

文字

1 int speaker = 11;  
2 void setup() {  
3 pinMode(speaker, OUTPUT);  
4 }  
5 void loop() {  
6 tone(speaker, 440);  
7 delay(1000);  
8 noTone(speaker);  
9 delay(2000);  
10 }  
11

The circuit diagram shows an Arduino Uno connected to a breadboard. A speaker is connected to digital pin 11 through a 220 ohm resistor. A push button is connected between digital pin 10 and ground. The Arduino's 5V and GND pins are connected to the breadboard power and ground rails respectively. The breadboard has a grid of 4 columns (a-d) and 20 rows (1-20). The Arduino Uno is shown at the bottom, with its pins labeled: AREF, GND, 1.3, 1.2, -1.1, 8, 7, 6, 5, 4, 3, 2, TX, ARDUINO, RX, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, ANALOG IN, A0, A1, A2, A3, A4, A5.

WestLab SD/Keio



# 可変抵抗

- 可変抵抗を読み取る

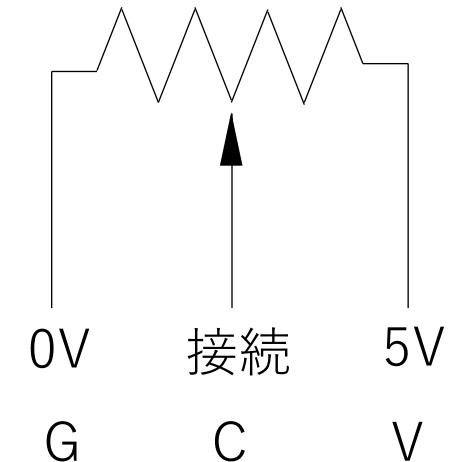
- 可変抵抗を使って、ピンに与える「電圧」を変化させ、その電圧を Arduino内蔵のA/Dコンバータで読み取る
- ArduinoにはA/Dコンバータが入っており、Analogと書かれた部分で読み取ることができる

```
int val; // 変数を定義  
val = analogRead(ピン番号); // 抵抗の電圧を読み取り
```

- DigitalとAnalogの同じ番号のピンは「同じ番号では」同時に利用できないので注意

- Arduino Unoの場合、アナログ入力ピンの0～5番をデジタル入出力ピンの14～19として利用できる
- さらにマクロ定義のA0～A5番を利用すると基板のシルク記載の通りに指定でき、次のような指定が可能である

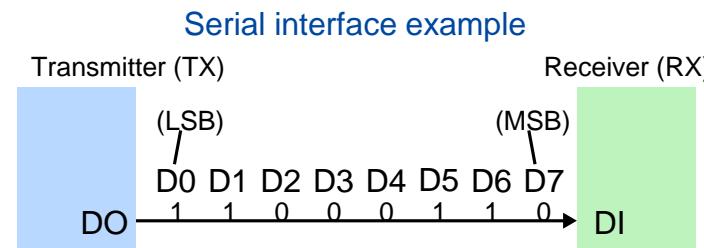
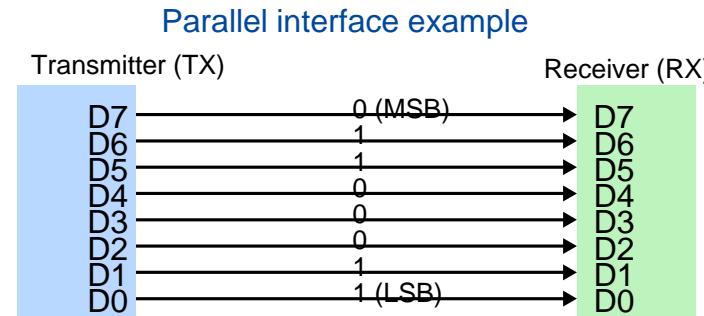
```
pinMode(A0,OUTPUT);  
digitalWrite(A0,HIGH);
```





## (5) シリアル通信（実機のみで動作）

- 配線一本で情報を送る通信方式
    - 電圧がHIGHつまり5Vならば1、LOWつまり0Vならば0
  - 実際にはいろいろケアするべき点がある
    - 電圧は相対的なため、別途グラウンドを伝える必要がある
    - 信号を相手に伝える出力と、相手からもらう入力が一般的には必要
    - 1と0をバタバタしただけでは、正しく値が取得できないので、ボーレート（転送速度）をあらかじめ決めておく
      - 例えば、0000111100001111と0101の区別はつくだろうか？
    - 0から始まるデータの始まりがわからないので、スタートビットを設ける
    - ノイズに弱く、エラーが発生しやすいため、パリティーをつける
    - 処理できないうちに次々にデータが送られてこないように、バックプレッシャーを設ける
    - などなど
  - USBも元々はシリアル(ユニバーサルシリアルバス)





# (6) Hello,Worldの設計

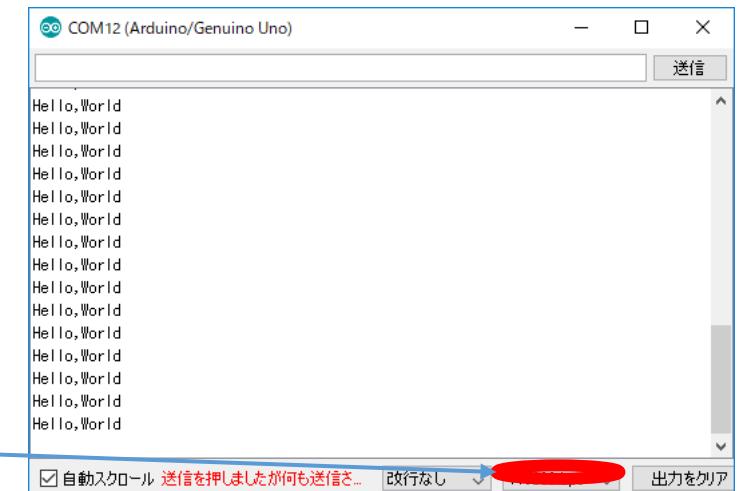
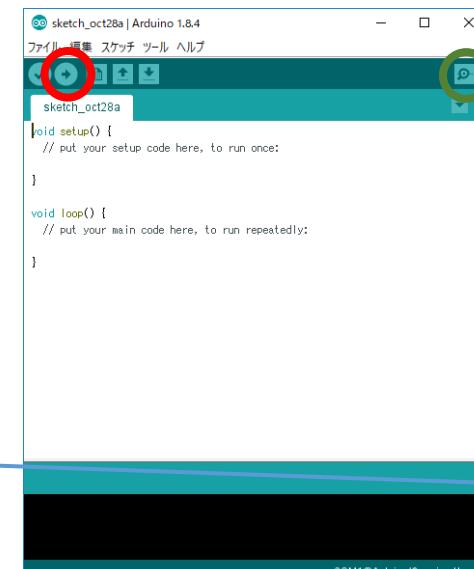
- ArduinoとPC間で通信する
- 手順
  - ArduinoとPCを繋げる
  - IDEを立ち上げる
  - ツールをクリック
    - ボードで「Arduino nano」を選択
    - ポートを選択
      - ()にArduinoの記載がある項目を選択
  - プログラムを記入
  - を押す
    - プログラムをArduinoに書き込む
  - ツールでポートを設定
    - Arduinoが接続されているポートを指定
  - をクリック
  - 通信速度を9600bpsに変更

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    Serial.print("Hello");  
    Serial.println(", world");  
    delay(1000);  
}
```

シリアル通信を9600bpsで開始  
その他の設定はデフォルトのまま

シリアルで"Hello"と出力

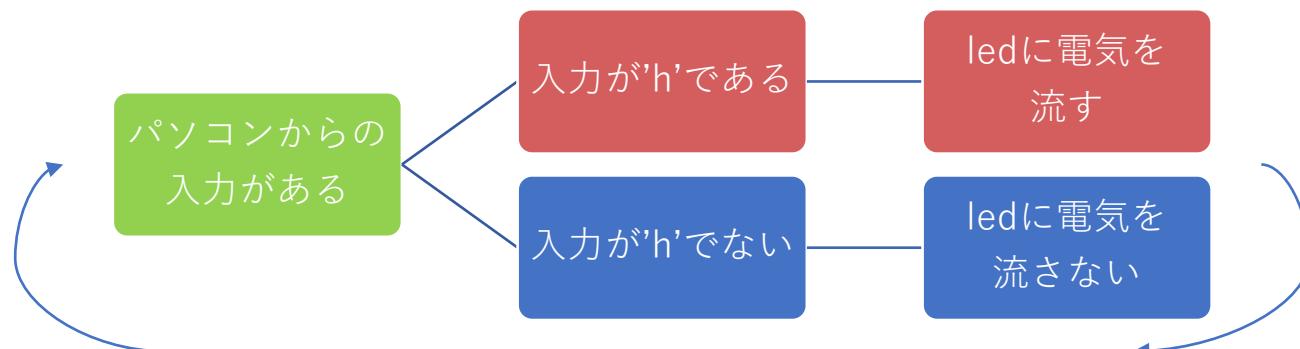
シリアルで", world"と出力 + 改行



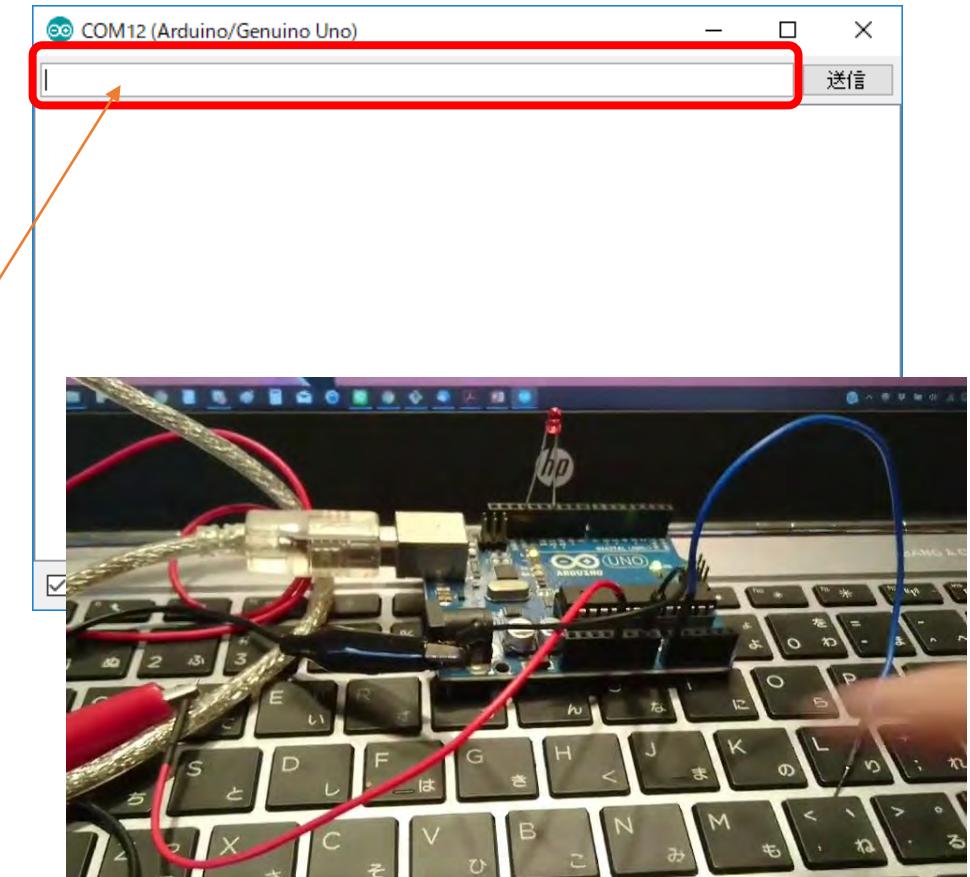
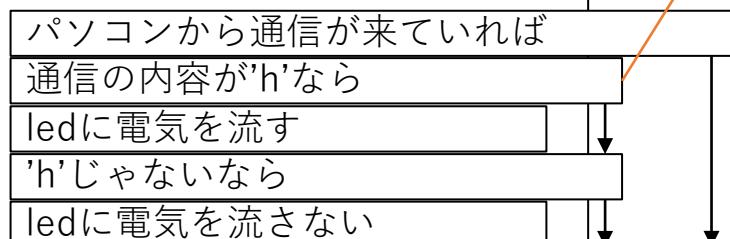


# PCの指令でLEDを点灯・消灯 (ON/OFF制御)

36



```
int led = 2;  
void setup() {  
    pinMode(led, OUTPUT);  
    Serial.begin(115200);  
}  
void loop() {  
    if (Serial.available() > 0) {  
        if(Serial.read() == 'h'){  
            digitalWrite(led, HIGH);  
        }else{  
            digitalWrite(led, LOW);  
        }  
    }  
}
```



# 設計と動作内容

37

The screenshot shows the Arduino IDE interface. At the top, there's a message "すべての変更が保存されました" (All changes saved). Below the menu bar, there are tabs for "コード" (Code), "シミュレーションを開始" (Start Simulation), "エクスポート" (Export), and "共有" (Share). The "Code" tab is selected. The main area displays the following Arduino sketch:

```
1 int led = 2;
2 void setup() {
3     pinMode(led, OUTPUT);
4     Serial.begin(115200);
5 }
6 void loop() {
7     if (Serial.available() > 0) {
8         if(Serial.read() == 'h'){
9             digitalWrite(led, HIGH);
10        }else{
11            digitalWrite(led, LOW);
12        }
13    }
14 }
```

To the right of the code editor is a large, empty rectangular area with a play button icon at the bottom center, likely a placeholder for a simulation or preview window. At the very bottom of the interface, there's a tab labeled "シリアル モニタ" (Serial Monitor).





# 言語について簡単な知識を得る

- Arudino IDEをインストールする
  - <https://www.arduino.cc/en/software>
  - Arduino IDE 2.0のリリース待ち
- ファイル→スケッチ例に様々な例がある
- シミュレータでの動作には制限があるので注意する
- 文法そのものは難しくない
  - Pythonを知っていればセミコロンや波括弧でのブロック表記さえ慣れれば問題ないはず
  - 多くの人がArduinoを利用しているため、Google先生も大変役に立つ

The screenshot shows the Arduino IDE interface with the title bar "mote | Arduino 1.8.13 (Windows Store 1.8.42.0)". The menu bar includes "ファイル" (File), "編集" (Edit), "スケッチ" (Sketch), "ツール" (Tools), and "ヘルプ" (Help). The "スケッチ" (Sketch) menu is expanded, showing options like "新規ファイル" (New File), "開く..." (Open...), "最近使った項目を開く" (Open Recent), "スケッチブック" (Sketchbook), and "スケッチ例" (Sketch Examples). The "スケッチ例" (Sketch Examples) submenu is also expanded, showing categories like "内蔵のスケッチ例" (Built-in Sketch Examples) and "あらゆるボード用のスケッチ例" (Sketch Examples for All Boards). Under "内蔵のスケッチ例" (Built-in Sketch Examples), the "Blink" example is highlighted. The code for the "Blink" sketch is visible in the main editor area:

```
#define BUTTON_RED_PIN 10
#define BUTTON_GREEN_PIN 11
#define BUTTON_BLUE_PIN 12

int oldR = 0, oldG = 0, oldB;
#define CNT_MAX 2
int cntR = 0, cntG = 0, cntB;
int colorR = 0, colorG = 0, colorB = 0;

#include "source.h"
MyXBee myxbee;
unsigned long pastMillis = millis();
#define SEND_INTERVAL 1000
#define LED_HEADER 'L'
#define ID_PACKET_OFFSET '0'

int clickDetection(int _pin, int buttonClicked = 0);
int newButtonState = digitalRead(BUTTON_RED_PIN);
if (*_oldstate == HIGH && *_oldstate != newButtonState)
    newButtonState = buttonClicked;
return buttonClicked;
}

void setup() {
    Serial.begin(9600);
    myxbee.init(Serial);
}

pinMode(LED_RED_PIN, OUTPUT);
pinMode(LED_GREEN_PIN, OUTPUT);
pinMode(LED_BLUE_PIN, OUTPUT);

pinMode(BUTTON_RED_PIN, INPUT);
pinMode(BUTTON_GREEN_PIN, INPUT);
pinMode(BUTTON_BLUE_PIN, INPUT);

void loop() {
    if (clickDetection(BUTTON_RED_PIN))
        cntR = cntR + 1;
    if (cntR >= CNT_MAX) cntR = 0;
    colorR = map(cntR, 0, CNT_MAX, 0, 255);
    colorG = map(cntG, 0, CNT_MAX, 0, 255);
    colorB = map(cntB, 0, CNT_MAX, 0, 255);
    analogWrite(LED_RED_PIN, colorR);
    analogWrite(LED_GREEN_PIN, colorG);
    analogWrite(LED_BLUE_PIN, colorB);
}
```

The status bar at the bottom right shows "COM1のArduino Uno". The bottom left corner of the slide has the number "12".



- Arduino言語 = avr-gcc + Wiring用ライブラリ + Arduino用ライブラリ
  - C++に準拠(SDではすでにC++は教えておらず、難解言語の一つ)

```
if(条件文) {
  // 条件に該当時実行
}
```

```
if(a < 500) {
  // 動作A
} else if(a >= 1000) {
  // 動作B
} else {
  // 動作C
}
```

```
x == y (xとyは等しい)
x != y (xとyは異なる)
x < y (xはyより小さい)
x > y (xはyより大きい)
x <= y (xはy以下)
x >= y (xはy以上)
```

```
&& : and, || : or, ! : not
```

```
switch (var) {
  case 1:
    // varが1のとき実行
    break;
  case 2:
    // varが2のとき実行
    break;
  default: // (省略可能)
    // 不一致のとき実行
}
```

**break** と **continue**  
**break** : ブロック離脱  
**continue** : 再実行

**return** と **goto**  
**return** : 関数から戻る  
 戻り値の追加も可  
**goto** : ラベル(文字列:) ヘジャンプ

```
for(初期化; 条件式; 加算) {
  // 実行される文;
}
```

```
while(条件式){
  // 実行される文
}
```

```
do {
  // 実行される文
} while(条件式);
```

**boolean** : 二値, **char** : 1文字  
**整数** (**unsigned**で符号なし)  
**Byte** : 8bit, **int** : 16bit  
**word** : 32bit, **long** : 64bit  
**小数**  
**float** : 32bit, **double** : 64bit  
**形無し** : **void**  
**高機能文字列クラス** : **String**

**pinMode(pin, mode)**  
**digitalWrite(pin, value)**  
**digitalRead(pin)**

**analogRead(pin)**  
**analogWrite(pin, value)**  
**analogReference(type)**  
**analogReadResolution(bits)**  
**analogWriteResolution(bits)**

**pulseIn(pin, value, timeout)**  
**tone(pin, frequency)**  
**noTone(pin)**  
**delay(ms)**

**Serial.begin(speed)**  
**Serial.available()**  
**Serial.read()**  
**Serial.flush()**  
**Serial.print(data, format)**  
**Serial.println(data, format)**  
**Serial.write(val)**





# IoTを作る

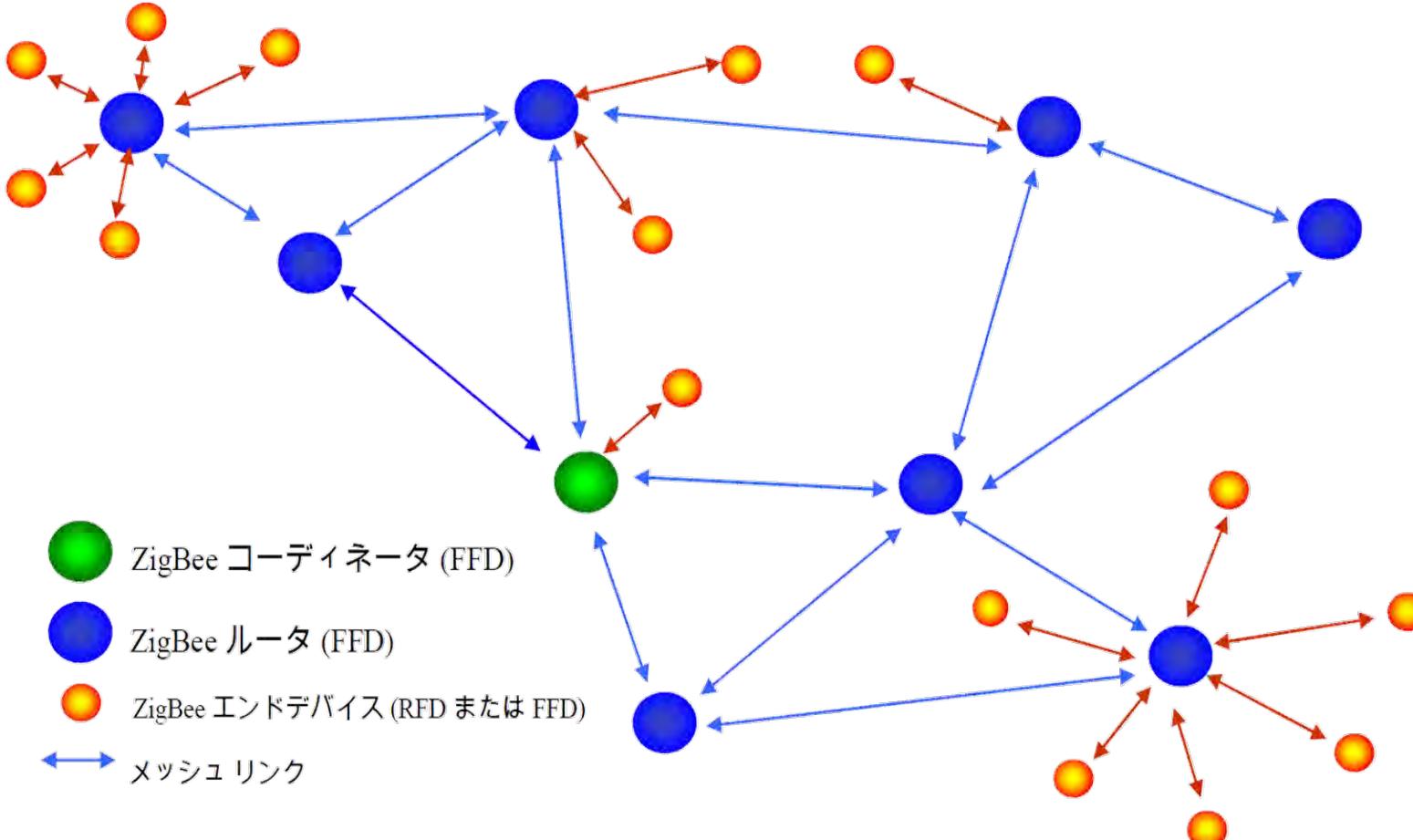
40

- ZigBeeと呼ばれる無線センサネットワークデバイスを利用してセンサネットワークを構築し、IoTシステムを実装する
- 実験では、ここから先の内容を実際に作成、実験内で仕様拡張を行う
- ここから先はシミュレータでは動作しないので注意すること



# ZigBeeとは

- ネットワーク通信モデル

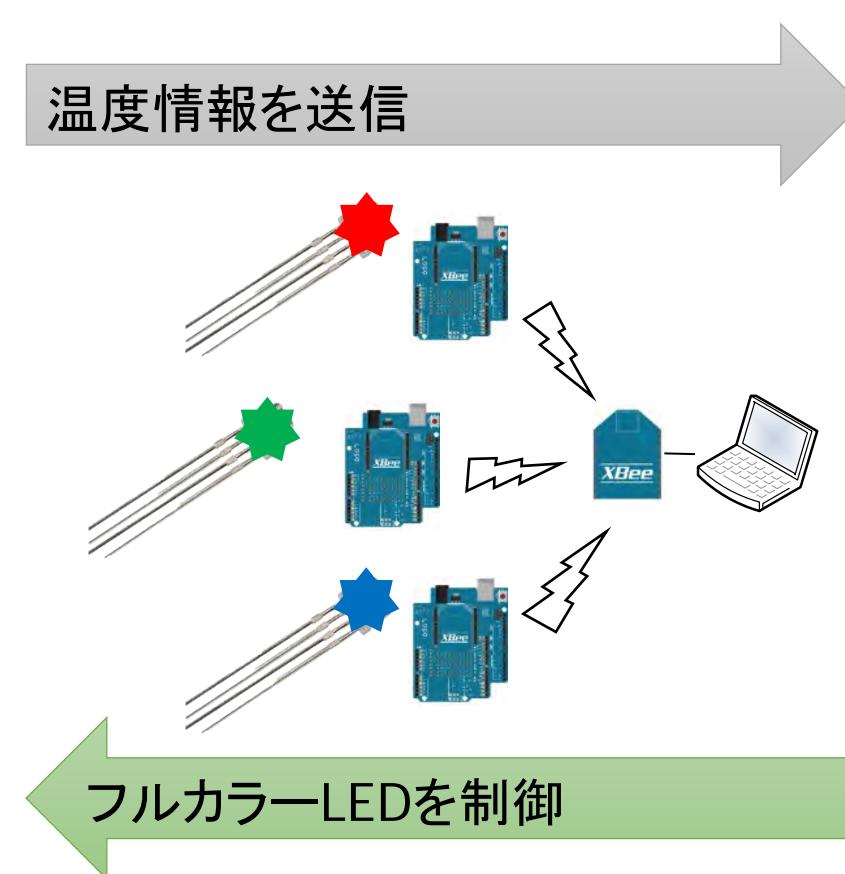
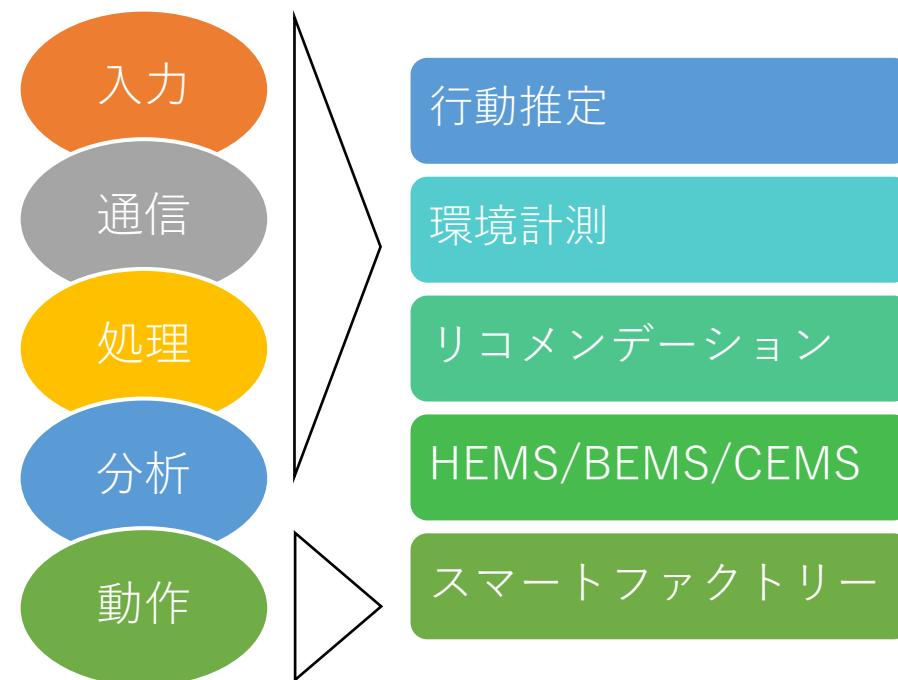


# 環境計測・制御システムの概要

42

- データを無線で集め（センサを実際に設計）、視覚化し（著名フリーアプリで設計）、ノードを制御

- 応用例





# IDEの使い方と事前準備

## ・事前準備としてデスクトップの「SDrefresh.bat」を実行する

- D:\SD\日付 というディレクトリが生成されている。これが、今日使う作業場所



## • Arduino IDEを起動する

## • アイコンと機能



- コンパイルコードを確認（確認するだけ）



- スケッチをコンパイルしアップロードする



- 新しいエディタウィンドウを開く



- ファイルを開きます。



- スケッチを保存します。



- シリアルモニターを開く（主にデバッグに利用）



- タブを管理するオプションを表示（ほぼ使わない）

## • Arduino UNOにプログラムを書き込み、通信させる

- USBが接続されており、ハブのスイッチがONであることを確認する

- ツールの、ボードとシリアルポート、両方にArduino UNOと表示させる

- ZigBeeシールドのスイッチが下(PCと接続)にする

```
BareMinimum
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

1 Intel® Galileo on /dev/cu.usbmodemfd141

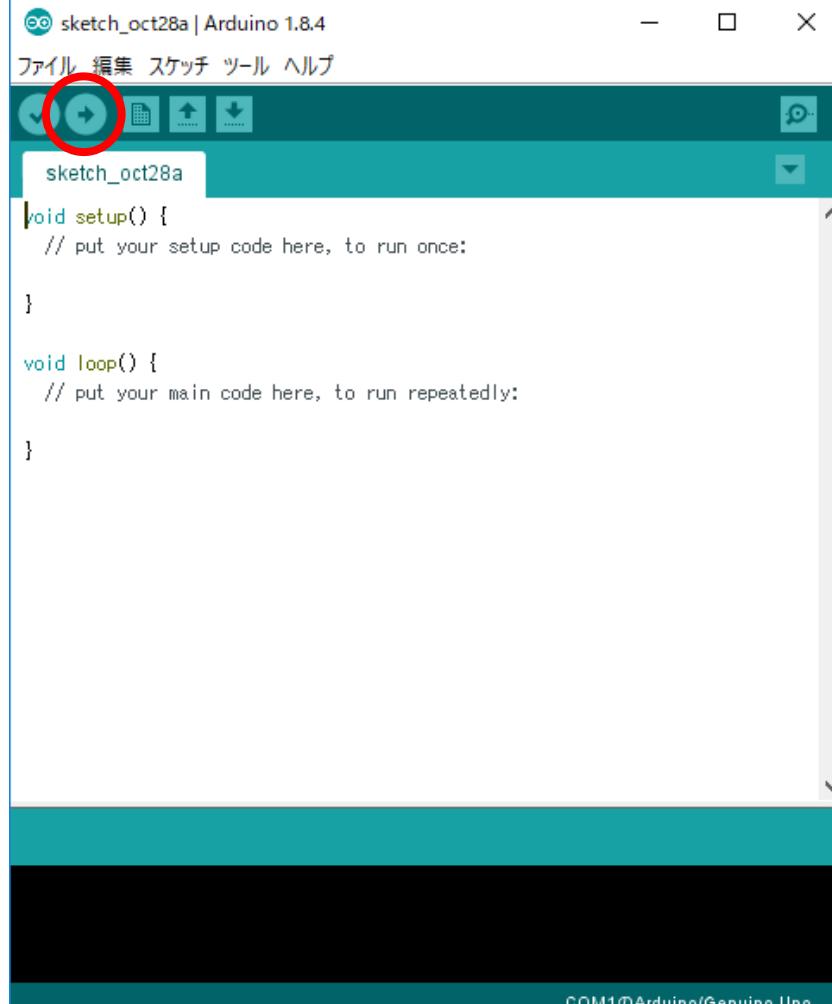




# PCとの接続と焼きこみ

44

- ArduinoとパソコンをUSBで繋げて、Arduino IDEを起動する
- ツールをクリック
  - ボードで「Arduino UNO」を選択
  - ポートを選択 (Arduino UNO)
- 白枠内にプログラムを書く
-  を押す
  - プログラムがArduinoに書き込まれる
  - エラーがでたらコードを見直す
  - 当然ながら回路のエラーは教えてくれない
- エラーの場合
  - エラーメッセージをよく読むこと
  - よくある間違い
    - セミコロン「;」が抜けている
    - String変数にダブルクオート「""」がついていない
  - インターネットで調べるか、得意な友達やTAに聞く



```
sketch_oct28a | Arduino 1.8.4
ファイル 編集 スケッチ ツール ヘルプ
upload (red circle)
sketch_oct28a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

COM1のArduino/Genuino Uno





# 実装上の注意

45

- とにかく優しく扱うこと
  - チカラは必要なく、きちんと向きや方向があっていれば簡単に入る
- わからなかつたら質問すること
- 小さいパーツに気を付けること！本当に刺さります
  - 地面に落として踏まないように
- 最後に元通りに片づけますので、最初のおさまりを覚えておくと
  - 写真を撮っておくとよい
- パーツはなくさないでください
  - なくしたり、壊したりしたら、黙秘せず、すぐに連絡してください
  - 最も困ることは次の実験ができなくなることです





# XbeeによるZigBee通信

- X-CTUでXBeeを初期設定する
  - 必要な場合は指示しますので、設定してください
  - 2台のXBeeを使用しArduinoとパソコンの双方から簡単な通信確認をとる
  - なお、すでに初期設定済みであるが、場合によっては修正が必要かもしれない
- XBeeの通信モード
  - ATモード（透過モード/Transparent mode）  
有線シリアル通信で制御するためArduinoで一般的なシリアル通信をそのまま無線化できる  
(シリアル通信は知っているかい?)
  - APIモード  
APIフレームと呼ばれるデータ構造で通信を行う方法で高機能
- 実験では基本的にAPIモードを利用する

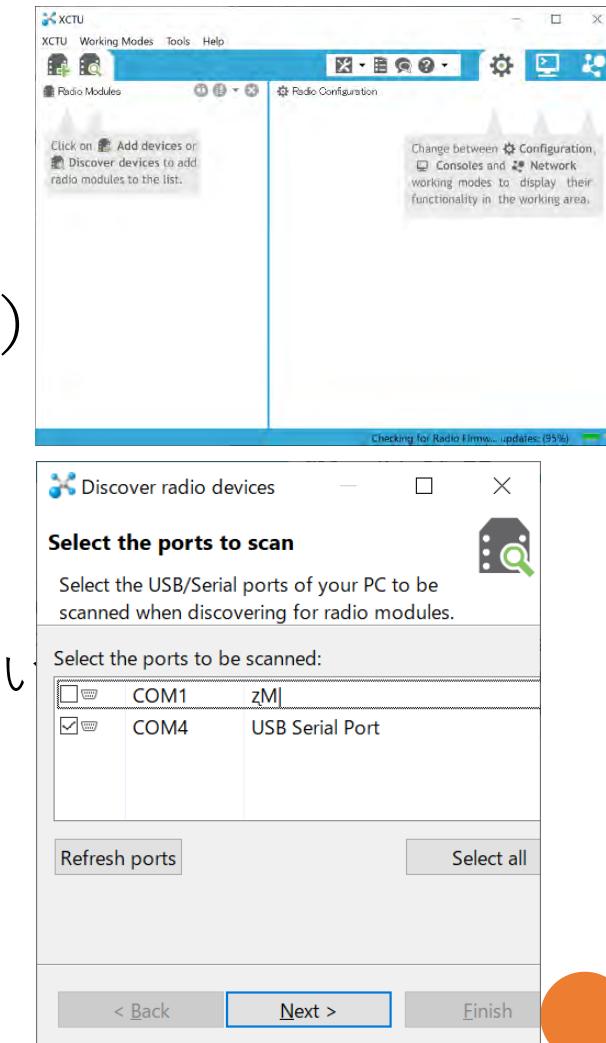




# USBシリアルXbeeボードによる更新

47

- Digiが提供しているX-CUTをインストール
  - <https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu#productsupport-utilities>
  - 起動したら左上の+の「Add devices」もしくは🔍の「Discover devices」でXbeeを認識させる
  - USB Serial Portなどと表示されるので「これだけ」選択、デフォルトの設定ままでFinishを選択すると、発見してくれる
- 必要に応じてファームウェアを更新する（指示があった場合のみ）
  - 少々時間がかかるので、一つのマシンでまとめてやった方が早い
  - S2B(Xbee PRO)は古くS2Cでなければ接続できない？#HelpからLegacyを入れると認識はする
  - FirmwareをUpdateボタンで更新すること(現在4061が最新)
  - ScanすればほかのXbeeも発見でき、リモートでファームを更新できる(遅い)
- IDは全員違う番号になっている
  - 番号ごとにコーディネータが1つ存在
  - 番号が重なると正しく動作しない！



# XBeeの設定における躓きどころ

48

- Arduinoに乗ったままでは何もできない
  - XBeeをシリアルモジュールに搭載しPCにつなげて書き換える
- PAN ID
  - 2401から2415まで設定 それぞれにラベルが振られている
- APIモードにすること
- 次の設定パラメータの通りになっていれば問題ないはず
- X-CTUで見えない！
  - 解法1：X-CTUのシリアルコンソールを開いてresetを押し、+++と連打、OKと帰ったらすかさず、AT FS FORMAT CONFIRMと入力してファイルシステムを初期化する
  - 解法2：ファームウェアを更新する





# 設定パラメータ（指示があった場合のみ設定）

49

## Networking

- ID: 242201-(順番)
- SC: 7FFF
- SD: 3
- ZS: 0
- NJ: FF
- NW: 0
- JV: 1
- JN: 0
- OP: 3320
- OI: A6D
- CH: 14
- NC: 14
- CE:
  - Router: 0
  - Coordinator: 1
- DO: 0
- DC: 0

## Addressing

- SH: 13A200
- SL: 41637A8A  
(P2P通信で必要)
- MY: C262
- MP: FFFE
- DH: 0
- DL: 0
- NI:
- NH: 1E
- BH: 0
- AR: FF
- DD: A0000
- NT: 3C
- NO: 0
- NP: FF
- CR: 3

## ZigBee Addressing

- SE: E8
- DE: E8
- CI: 11
- TO: 0

## RF Interfacing

- PL: 4
- PM: 1
- PP: 8

## Security

- EE: 0
- EO: 0
- KY: 00
- NK: 00

## Serial Interfacing

- BD: 3
- NB: 0
- SB: 0
- RO: 3
- D6: 0
- D7: 1
- AP: 2
  - Node-REDとCoordinatorをつなげる際の便宜を考え、0にする場合もあり
- AO: 0

## AT Command Options

- CT: 64
- GT: 3E8
- CC: 2B

## Sleep Modes

- SP: 20
- SN: 1
- SM: 0
- ST: 1388
- SO: 0
- WH: 0
- PO: 0

## I/O Setting

## I/O Sampling

- デフォルトから設定値が異なる場所について赤で示しており、X-CTUでは青色の右下三角マークが現れる
- 変更を施し、Writeする際に更新されるパラメータは、X-CTUにおいて緑色の右下三角マークが現れる



# プログラムの概要

// Program for Arduino version KEIO SD Westlab 2021.3

// Author: Tada Matz, Kanami Yuyama, Hiro West

// 定義

#define ARD\_LED 13

#define RECVINTERVAL 50

#define BTNO 10 // 10番ピン以降(12まで)を連続してボタン入力とする

#include "source.h" // 独自軽量化XBee関数

#include <MsTimer2.h> // タイマー割り込み

MyXBee myxbee;

// メッセージ受信用バッファ

String message;

int msglen;

// ボタン入力検出用関数

int cDflag[3];

int clickDetection(int \_pin) {

int buttonClicked = 0;

int button = \_pin - BTNO;

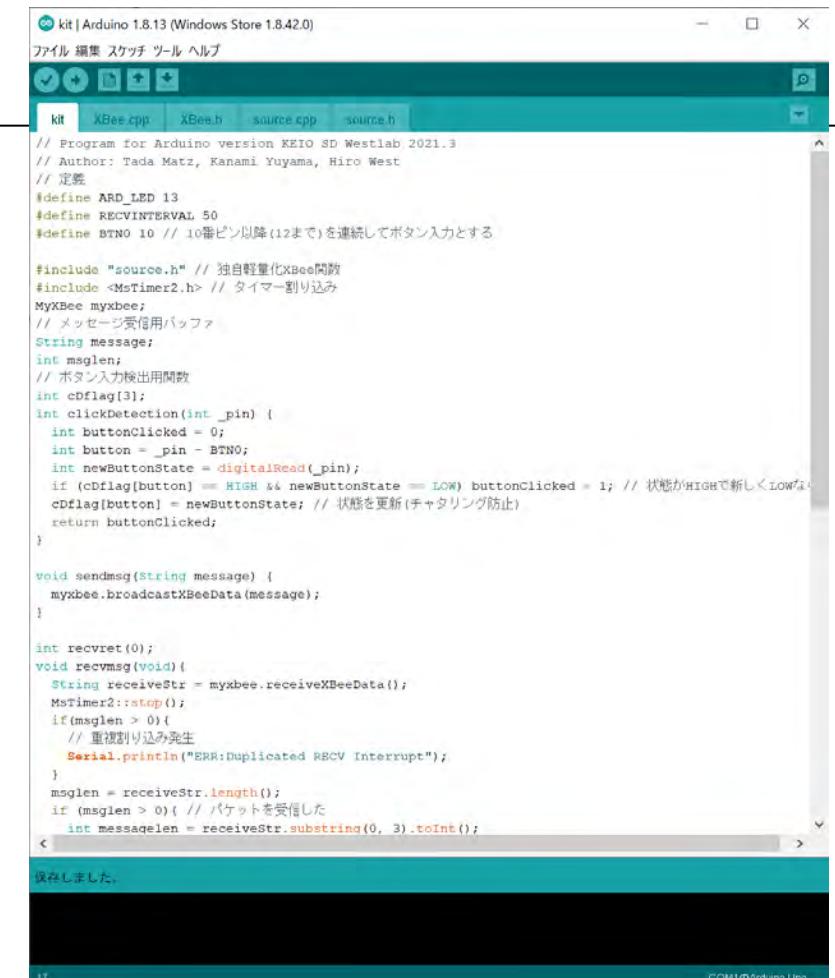
int newButtonState = digitalRead(\_pin);

if (cDflag[button] == HIGH && newButtonState == LOW) buttonClicked = 1; // 状態がHIGHで新しくLOWなら

cDflag[button] = newButtonState; // 状態を更新(チャタリング防止)

return buttonClicked;

}



```
kit | Arduino 1.8.13 (Windows Store 1.8.42.0)
ファイル 編集 スケッチ ツール ヘルプ
kit XBees.cpp XBees.h source.cpp source.h
// Program for Arduino version KEIO SD Westlab 2021.3
// Author: Tada Matz, Kanami Yuyama, Hiro West
// 定義
#define ARD_LED 13
#define RECVINTERVAL 50
#define BTNO 10 // 10番ピン以降(12まで)を連続してボタン入力とする

#include "source.h" // 独自軽量化XBee関数
#include <MsTimer2.h> // タイマー割り込み
MyXBee myxbee;
// メッセージ受信用バッファ
String message;
int msglen;
// ボタン入力検出用関数
int cDflag[3];
int clickDetection(int _pin) {
    int buttonClicked = 0;
    int button = _pin - BTNO;
    int newButtonState = digitalRead(_pin);
    if (cDflag[button] == HIGH && newButtonState == LOW) buttonClicked = 1; // 状態がHIGHで新しくLOWなら
    cDflag[button] = newButtonState; // 状態を更新(チャタリング防止)
    return buttonClicked;
}

void sendmsg(String message) {
    myxbee.broadcastXBeeData(message);
}

int recvret();
void recvmsg(void){
    String receiveStr = myxbee.receiveXBeeData();
    MsTimer2::stop();
    if(msglen > 0){
        // 重複割り込み発生
        Serial.println("ERR:Duplicated RCV Interrupt");
    }
    msglen = receiveStr.length();
    if (msglen > 0){ // パケットを受信した
        int messagelen = receiveStr.substring(0, 3).toInt();
    }
}

```

保存しました。

COM1のArduino Uno



# 通信部分

```
void sendmsg(String message) {  
    myxbee.broadcastXBeeData(message);  
}  
  
int recvret();  
void recvmsg(void){  
    String receiveStr = myxbee.receiveXBeeData();  
    MsTimer2::stop();  
    if(msglen > 0){  
        // 重複割り込み発生  
        Serial.println("ERR:Duplicated RCV Interrupt");  
    }  
    msglen = receiveStr.length();  
    if (msglen > 0){ // パケットを受信した  
        int messagelen = receiveStr.substring(0, 3).toInt();  
        int loc = 3;  
        message = receiveStr.substring(loc, loc+messagelen);  
        // [即時処理]変数代入など簡単な処理を記述、時間のかかる処理はアウト  
    }  
    MsTimer2::start();  
    // [通常処理]比較的時間のかかる処理はここで記述することができる。  
    if(msglen > 0){  
        // message;  
    }  
}
```

- 送信はいたってシンプル
- 相手を指定する場合は、sendXBeeData(ノードID, Message);とする
- タイマー割り込みを用いて受信データを読み込む
  - 本来はシリアル通信による割り込みを用いるがPCとのシリアル通信に利用されている
- 50msに一度確認している
- パケットを受信するとメッセージ長が0よりも大きくなることを利用
  - 現在の中身はあくまでも例





# プログラムの本体

- loop()の中で主に設計する
- Node-REDの方で工夫して「ゴミ」を取るため、コンマで囲って送りたいデータを送ることにしている
  - スマートな方法はいろいろあるが、javascriptをなるべく必要としない方法としている

```
void setup() {  
    Serial.begin(9600);  
    myxbee.init(Serial);  
    pinMode(ARD_LED, OUTPUT);  
    MsTimer2::set(RECVINTERVAL, recvmsg); // タイマー割り込みをかける  
    MsTimer2::start();  
}  
void loop() {  
    sendmsg(",送りたい値,");  
    delay(1000); // データは頻繁に送信しないように  
}
```

- プログラムを修正する
  - main.h, source.h, source.cpp, XBee.h, XBee.cppはkit.inoと同じ階層に入れること
  - オリジナルは保存しておいたほうがよい

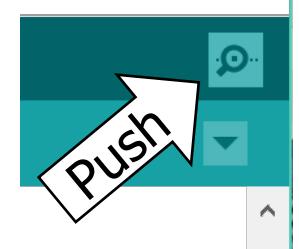




# 動作チェック

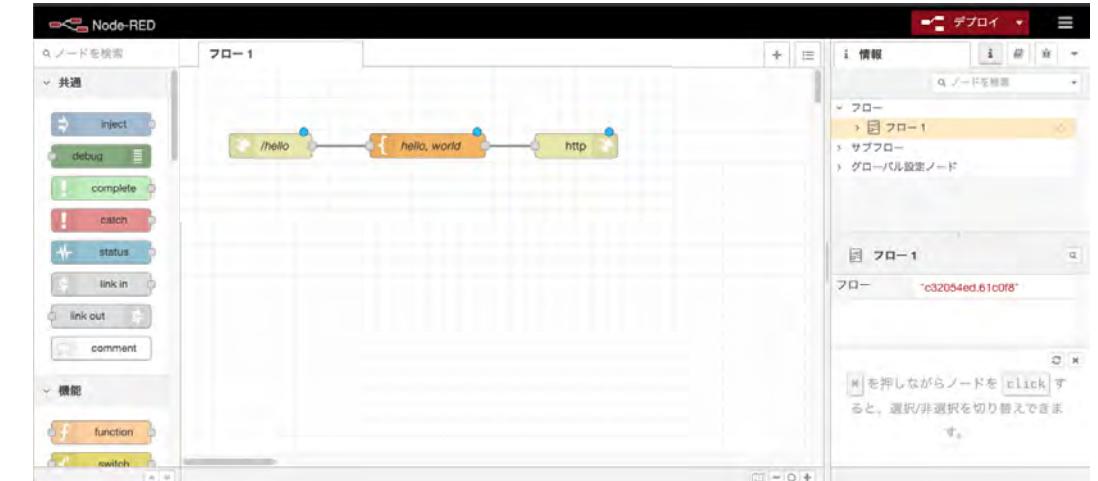
54

- シリアルモニターで通信内容を見る
  - PCとArduinoはシリアル通信しており、Xbeeとの通信も見ることができる
- よく見てみる
  - 送っているタイミングでスクロールするか確認
  - 送っている内容が含まれているか確認
  - 送っている内容が正しいか確認

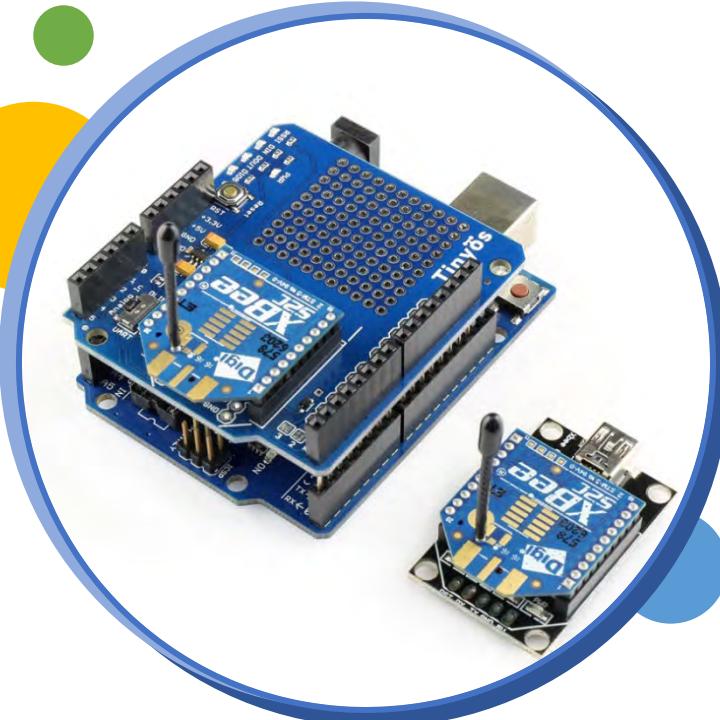




- 「データの流れ」をフローで描画しプログラムからある程度解放された形として記述
- データの流れを記述することで、データの変換や可視化などあらゆる処理を実現
  - 3,000を超えるプラグインが存在、スマートスピーカ、データベース、メール、skype、LINE、Twitter、HomeKitなどIoT、AIもとにかく何でも「簡単なことならすぐにできる」ようになる
  - ブラウザ上で動作し、インターネットにつなげておけば世界のどこからでも操作できる
  - 中身はNode.jsにより動作し、javascriptで記述するが、意識する必要はない
- 使い方
  - <https://www.youtube.com/watch?v=EiG1nNG6WIM>
- クラウド版を試してみよう（必須ではない）
  - インストールせずに利用できるサービス
  - <https://users.sensetecnic.com/register>
  - FRED shortより無料で利用できる  
(FREDのサーバを間借りする)
  - 制限はあるが充分利用できる
  - できるだけ自分のPCにインストールして利用しよう



# 午後の実験

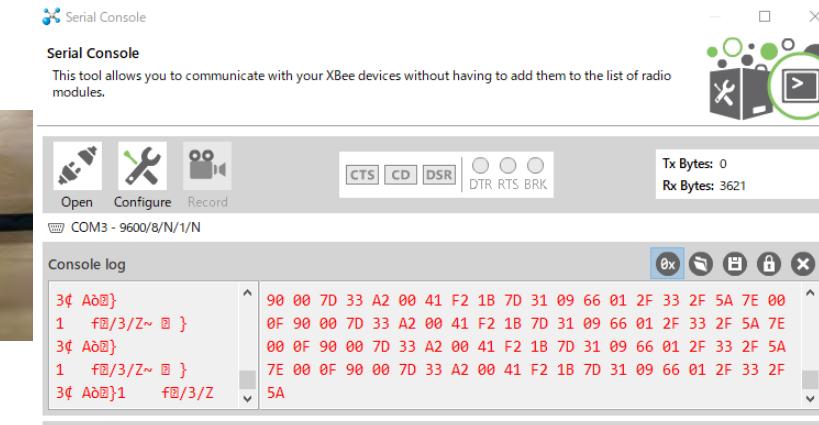
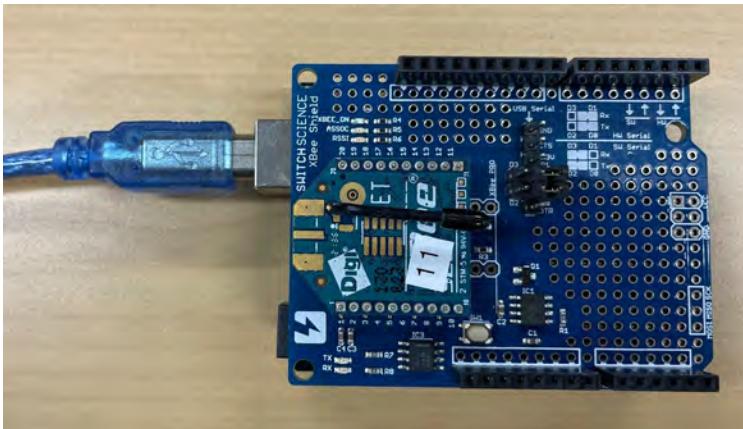




# 午前の最後にやったこと

57

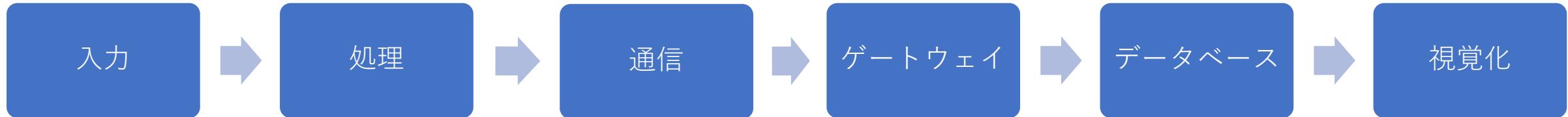
- 可変抵抗の値をArduinoで読む
- 読んだ値を無線通信(ZigBee)でコーディネータへ送信
- 受け取った内容をXCTUのserial consoleで確認



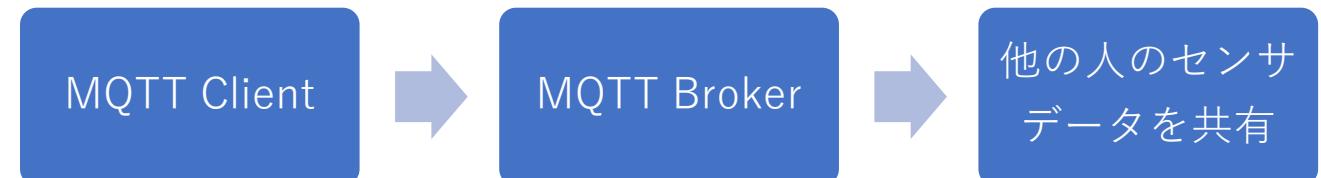


# これから設計するシステムの流れ

58



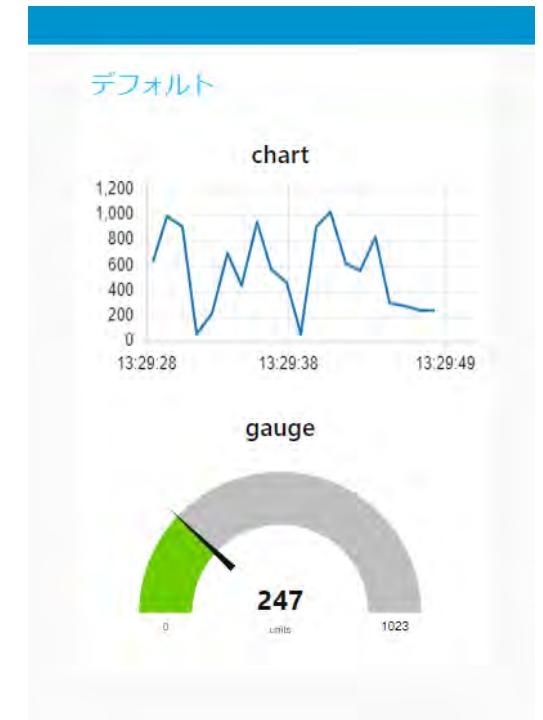
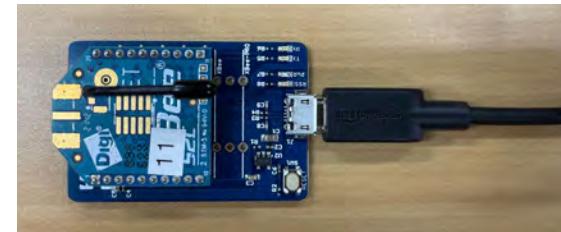
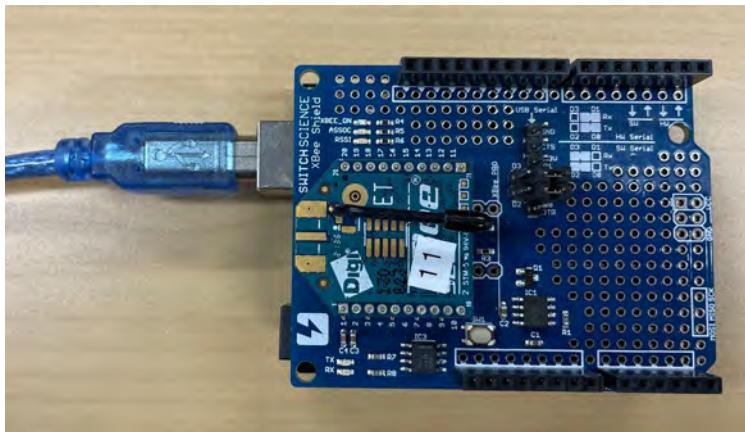
可能であれば他の人のセンサの値も一緒に表示してみよう





# 今からやりたいこと

- コーディネータで受け取った可変抵抗の値をNode-Red上で処理/可視化する
- 解説動画ではInjectで入れた任意の値をグラフ化したが、ここでは実際に無線で受け取った値をグラフ化する
- 処理内容は動画とほぼ同じ





# 準備

60

- XCTUを閉じる
- 3台のArduinoの電源スイッチを切る
- コーディネータ(黒ケーブル)のみ電源が入る状態にする

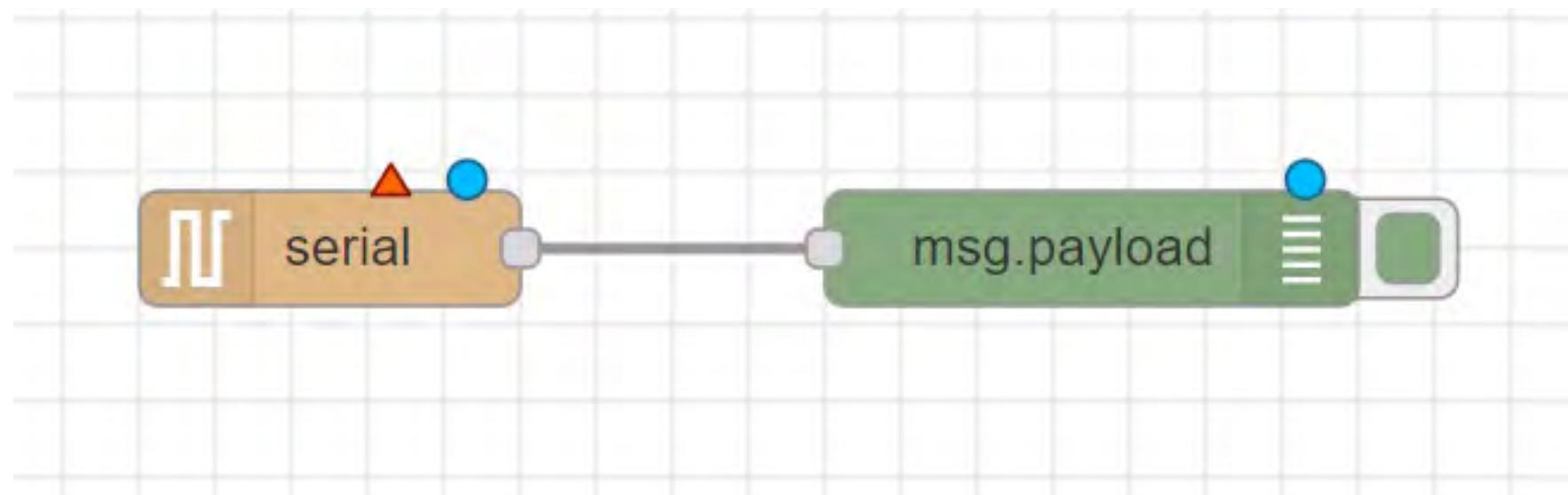
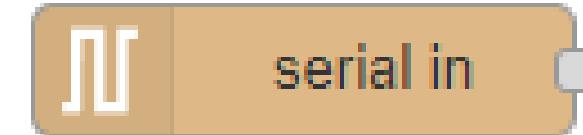




# Node-Redで通信内容が見えるか確認

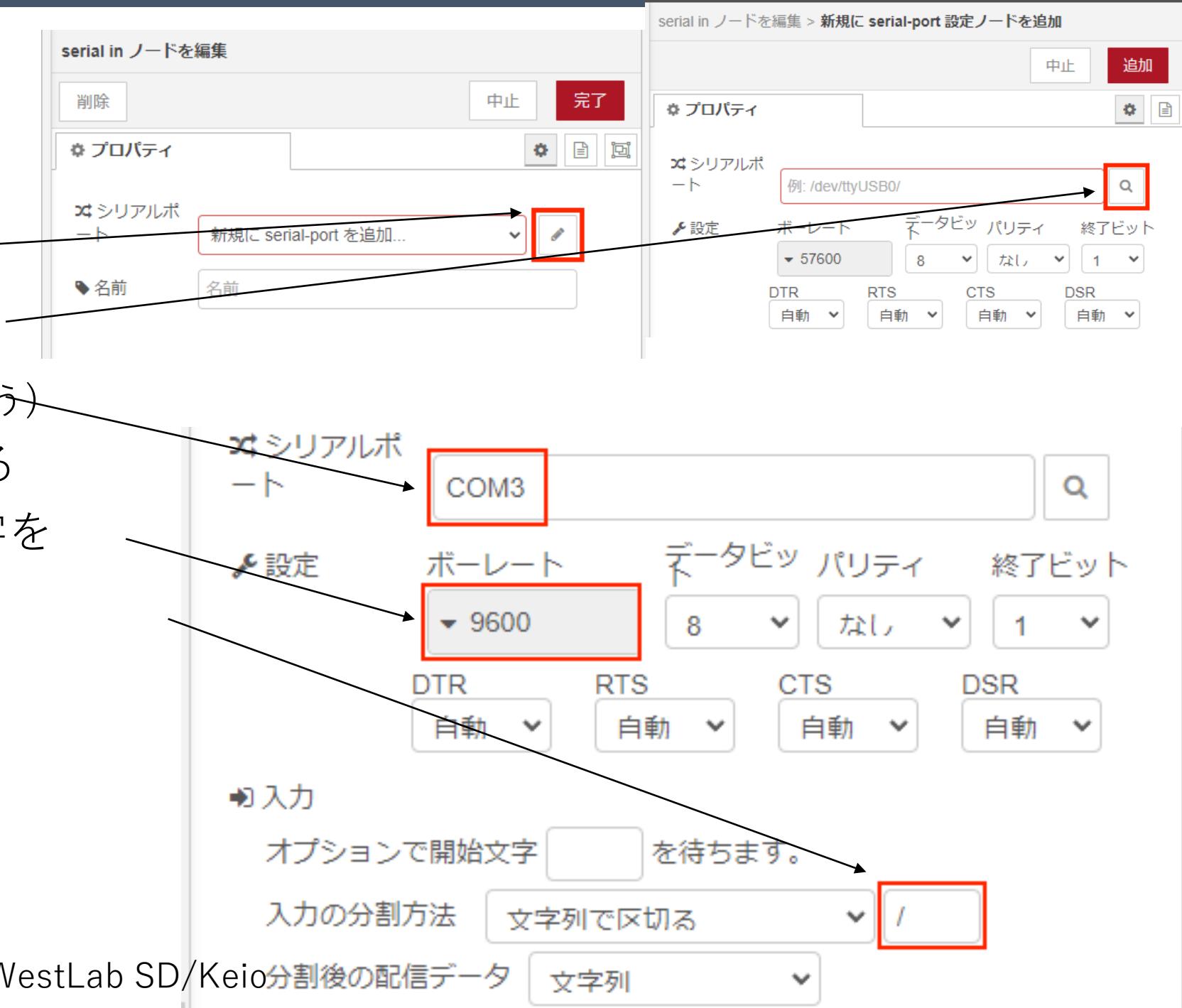
61

- serial inを左から探して配置する
- serial inとdebugを繋げる



# serial in の設定

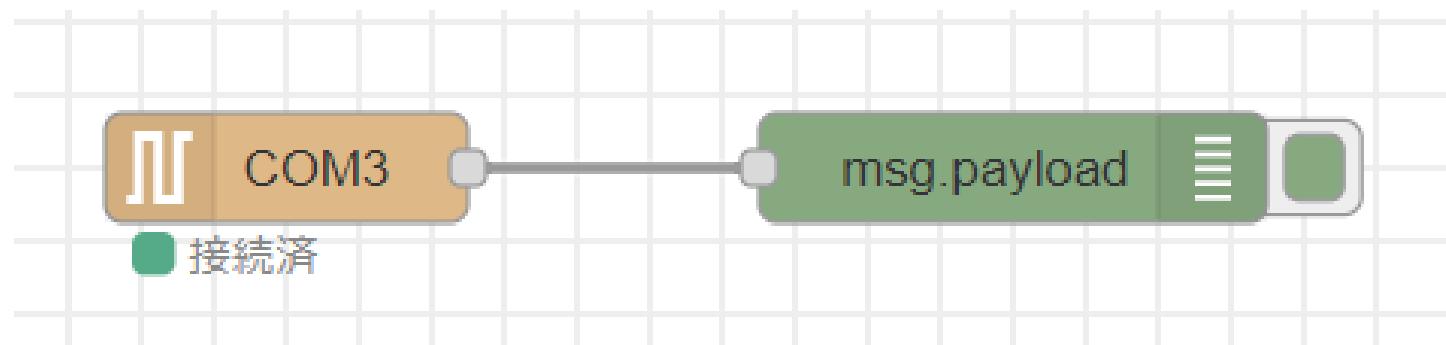
- serial in をダブルクリック
- 鉛筆マークをクリック
- 虫眼鏡マークをクリック
- COM1じゃない方を選択
  - ここではCOM3(人によって違う)
- ポーレートを「9600」にする
- 入力の分割方法の区切り文字を「/」にする
- 追加を押す
- 完了を押す





# デバッグで確認

- デプロイ
- デバッグコンソールを見る
- 午前に設計したArduinoの電源を入れる
- XCTUで見たのと同じようなパケットが
- 確認できるはず



WestLab SD/Keio

A screenshot of a serial monitor window. The title bar says 'デバッグ' (Debug). The window contains several identical log entries:

```
▶ "♦~}1♦}3♦A♦日}1→f日/"  
2022/5/24 13:05:38 node: a714ea41.2e4ff8  
msg.payload : string[4]  
"654/"  
2022/5/24 13:05:39 node: a714ea41.2e4ff8  
msg.payload : string[20]  
▶ "♦~}1♦}3♦A♦日}1→f日/"  
2022/5/24 13:05:39 node: a714ea41.2e4ff8  
msg.payload : string[4]  
"631/"  
2022/5/24 13:05:40 node: a714ea41.2e4ff8  
msg.payload : string[20]  
▶ "♦~}1♦}3♦A♦日}1→f日/"  
2022/5/24 13:05:40 node: a714ea41.2e4ff8  
msg.payload : string[4]  
"638/"
```



# 可視化

64

- ここまでで無線通信で受け取った内容をNode-Redの入力にすることができた
- あとは動画内で作成したフローとほとんど同じ内容
- 次のスライドに一連のフローと設定が掲載してあるのでそれを参考に可視化を行う
  - コピー又はそのまま繋げて構わない
  - 印の付いている項目は必ずチェックすること
  - 可変抵抗の値は0-1023
  - chartのx軸の時間はお好みで



# Node-REDによる視覚化

65

デプロイすると保存して実行

Node-RED

ノードを検索

http response  
http request  
websocket in  
websocket out  
tcp in  
tcp out  
tcp request

フロー 1

chart

msg.payload

COM9  
接続済

split

switch

chart

gauge

デプロイ

プロパティ

図サイズ: 6 x 6  
ラベル: chart  
種類: 折れ線グラフ  
X軸: 直近 20 秒 又は 1000 ポイン  
X軸ラベル: HH:mm:ss  
Y軸: 最小 0 最大 1023  
凡例: 非表示 補完 直線  
有効

プロパティ

Group: [ホーム] デフォルト  
Size: 6 x 6  
Type: Gauge  
Label: gauge  
Value format: {{value}}  
Units: units  
Range: min 0 max 1023  
有効

プロパティ

名前: 名前  
プロパティ: msg. payload  
分割: >= 0  
is not empty  
最初に合致した条件で終了  
メッセージ列の補正  
有効

9600  
9000  
DTR: 自動  
RTS: 自動  
CTS: 自動  
DSR: 自動

入力  
オプションで開始文字 を待ちます。  
入力の分割方法 文字列で区切る /  
分割後の配信データ 文字列

出力  
出力メッセージに分割文字を追加する

リクエスト  
デフォルトの応答タイムアウト 10000 ミリ秒

有効 1 個のノードが、この設定を使用しています

文字列の最後を意味するNULLが入り込むとswitchノードが反応してしまうので、これを避けるため、ストリームつまりすべてくっつけて一つの流れにする。

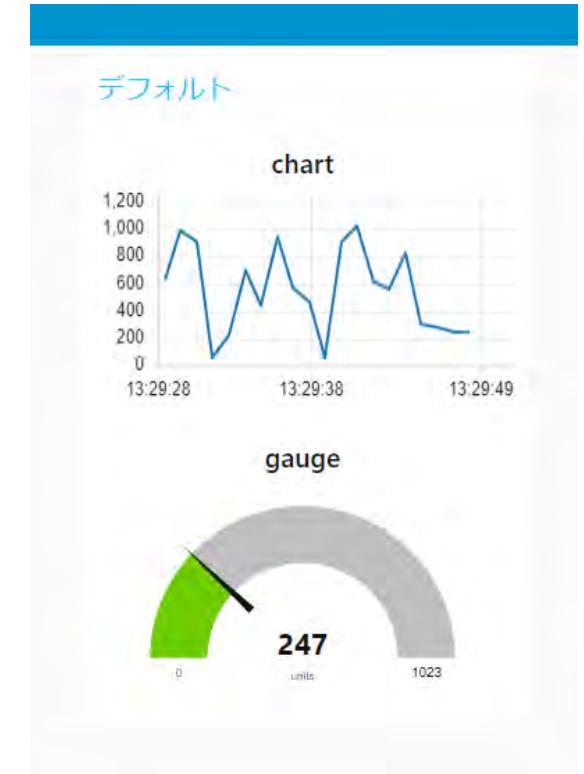
WestLab SD/Keio



# ダッシュボード

66

- デプロイしたらダッシュボードを開く
  - 可変抵抗を回すとグラフの値が変化することを確認する





# 複数センサの値を表示

## チャレンジ（1）

- 複数のArduinoを準備し、プログラムと設計とする
- 複数のArduinoのセンサ値が混ざってNode-REDで表示されることを確認する

## チャレンジ（2）

- 複数のArduinoのセンサ値を異なるメータで表示させる  
(ヒント) メッセージに識別子を付与し、Node-RED内でこれを認識、仕訳する

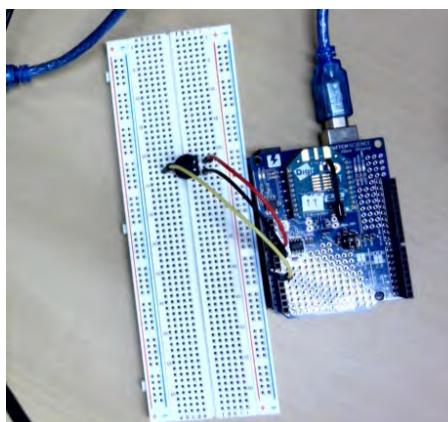




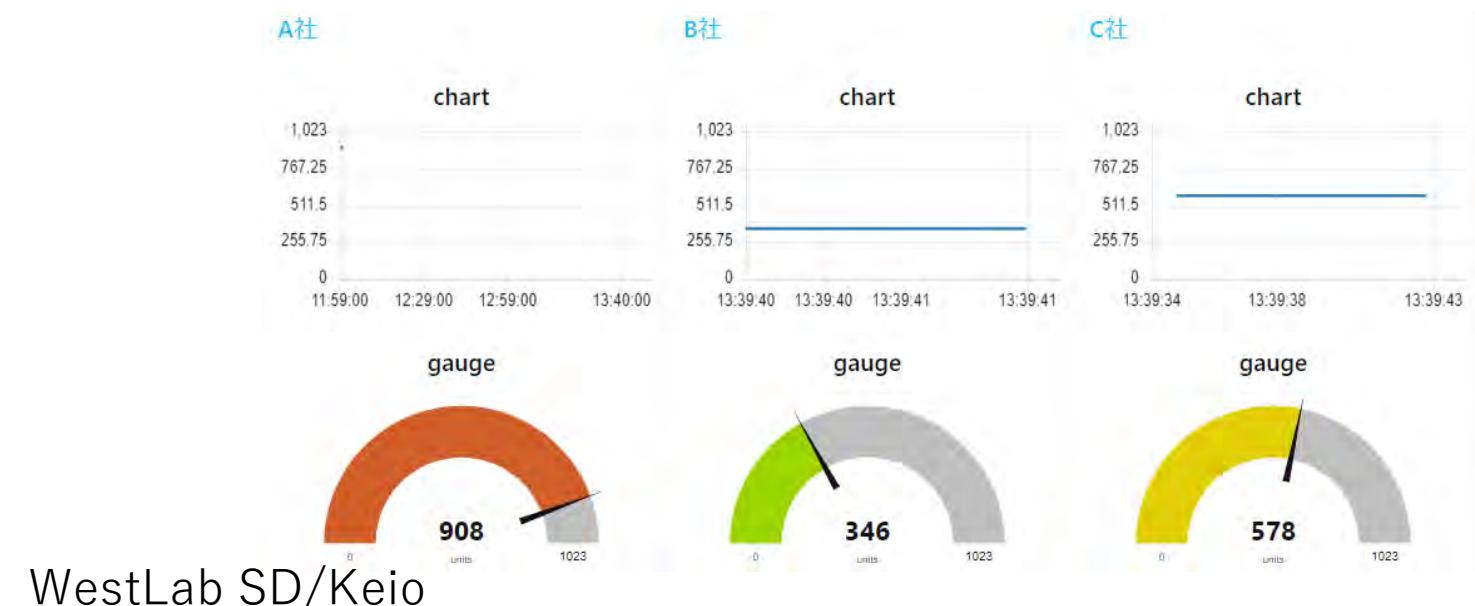
# 動画を見た後のチャレンジ

68

- 可変抵抗を付けたArduinoを3セット用意する
- どのArduinoから来たデータかを識別するためにデータの頭に識別子をつける
  - /データ/ → /識別子,データ/ (例) /1,データ/
- それぞれのArduinoにコードを書き込む
  - ツール >シリアルポートで書き込み先の変更
  - もちろん書き込み先ごとに識別子は変える
- データの識別子をみて、node-redでArduinoごとにグラフ化
  - スライド67,68,69参照
- 例えば



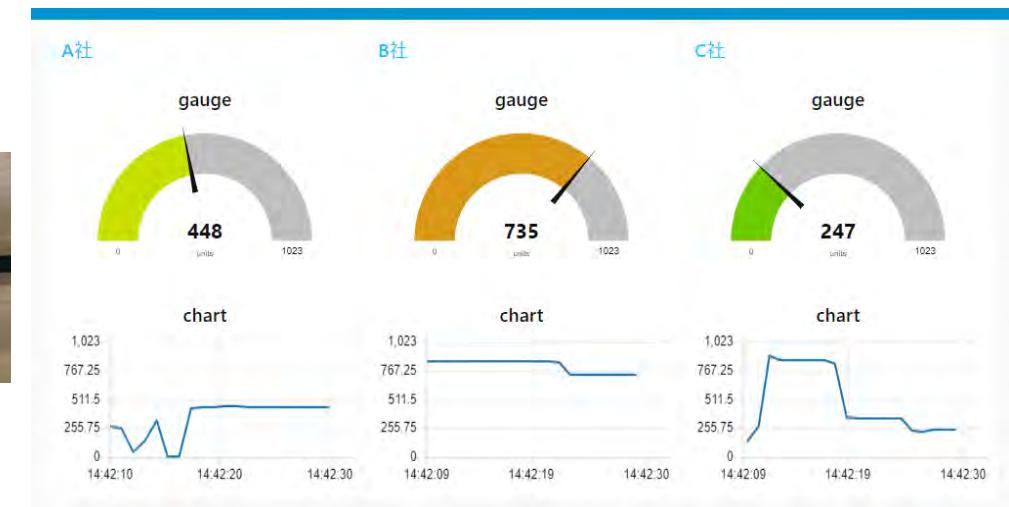
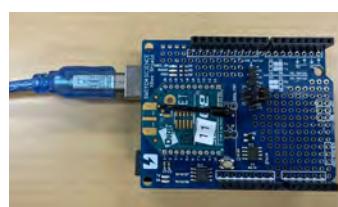
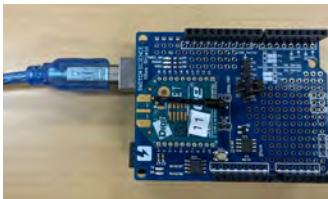
× 3



# 複数のセンサの値を集める

69

- 1台のArduinoから送られてきたデータをNode-Red上に表示することができた
- 1台のセンサだけではIoTとは言えない
- ここからは3台のArduinoからデータを送信してNode-Red上で表示させる

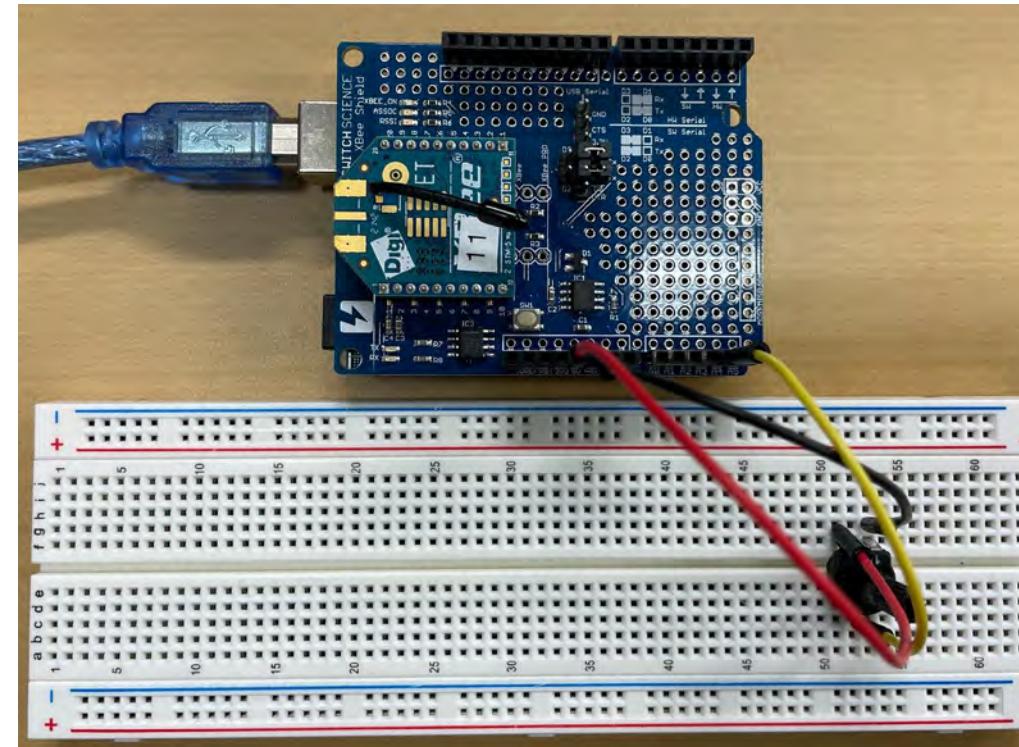




# 残り2台のArduino

70

- 残り2台のArduinoに可変抵抗を付けたものを用意する
- 可変抵抗しか使わないため写真のように電源(5V)やGNDと直接配線してよい

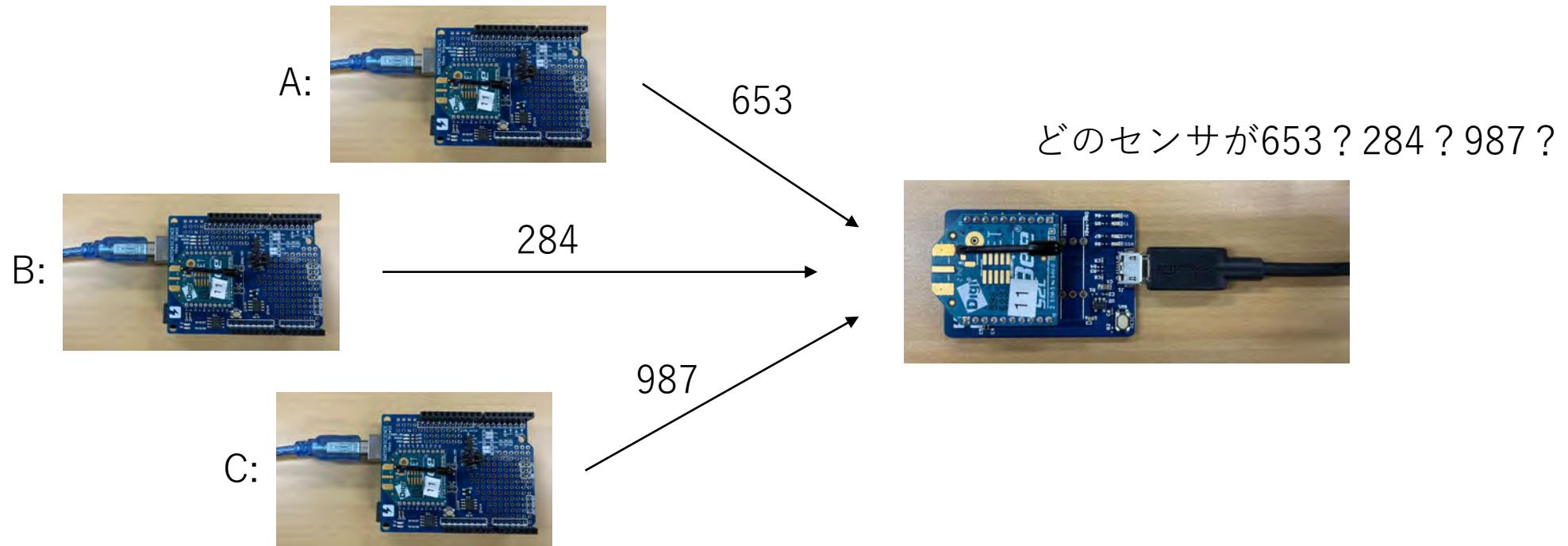




# それぞれのデータを識別するためには？

71

- 午前と同じコードを3台全てに書き込むとArduinoは可変抵抗の値のみをコーディネータに送信する
- どのセンサから来た値か分からぬ

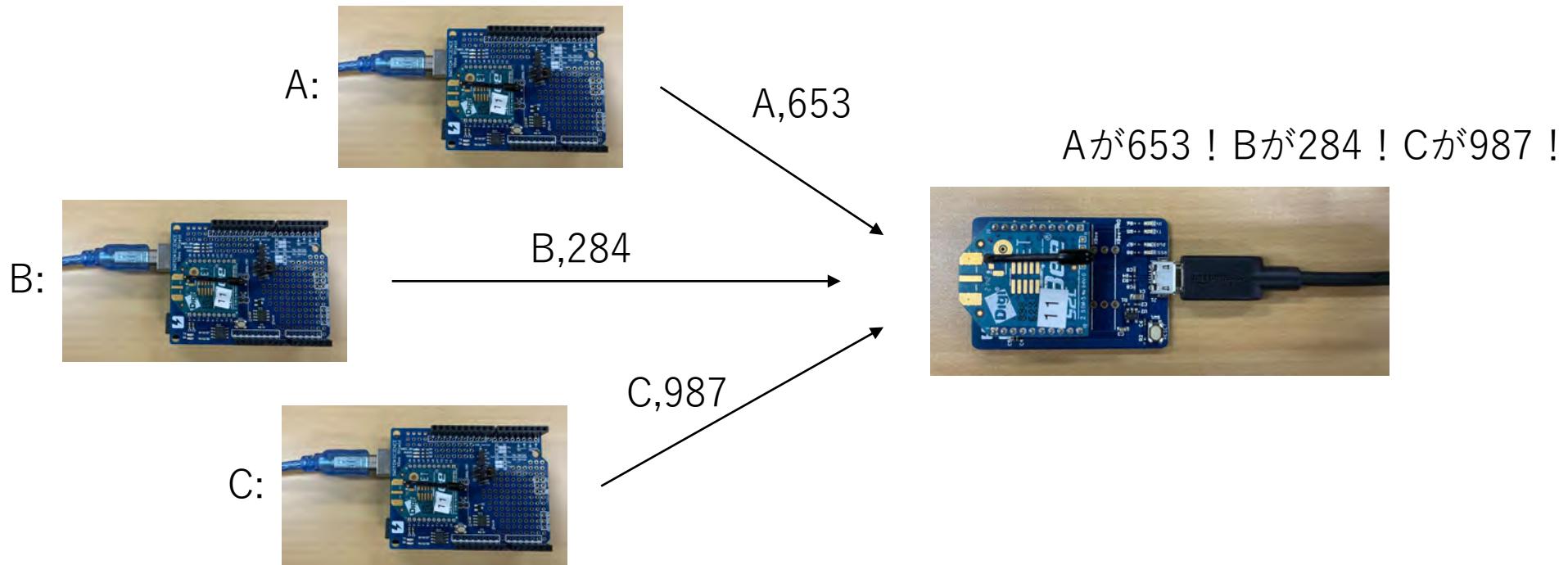




# それぞれのデータを識別するためには？

72

- データと一緒にどのセンサから送信したのかを示す識別子を送信する





# それぞれのデータを識別するための工夫

73

- 今回利用するのはXBeeのAPIモードのため、通信上の**すべての文字が見えててしまう**
  - APIモードのみ無線接続の確認や無線経由の設定が行える
  - 本来データのみを転送するモードを選ぶべきであるがあえてAPIモードにしている
- APIエスケープモードを利用しているため、通常のデータ以外はエスケープされる
  - エスケープとは、ここでは「**文字コードを邪魔にならない大きな値にすること**」
  - つまり、コマンドを意識せず、文字を見ることができる
    - 例えば10という数値データを受け取るとすると  
?(エスケープによりなんだかわからない文字列)?10?(なんだかわからない文字列)  
といった具合に、10だけきちんと読めるようになる
    - 10の部分だけ切り出せるようにすればよい
- ここでは、何かしらの文字を用いて数値部分を切り出して用いる
  - 例えば'/'スラッシュを用いて/10/として送信すれば、/より前と後はゴミと判断できる
  - Arduinoでは、例えばsが文字配列、aが値とするとString型の文字列変数に対して、  
`String s = "/" + String(a) + "/";`などとする

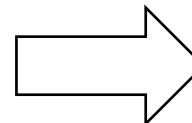




# データに識別子を付けて送信

- コードを少し変更する
- データの前に何か識別子を付ける
  - A,B,Cでも1,2,3,でも#,,\$,%でもなんでもよい
- 書き込み方は次のスライド参照

```
void loop() {  
    int a;  
    String s;  
    a = analogRead(4);  
    s = "/" + String(a) + "/";  
    Serial.println(s);  
    sendmsg(s);  
    delay(1000);  
}
```



```
void loop() {  
    int a;  
    String s;  
    a = analogRead(4);  
    s = "/A," + String(a) + "/";  
    Serial.println(s);  
    sendmsg(s);  
    delay(1000);  
}
```

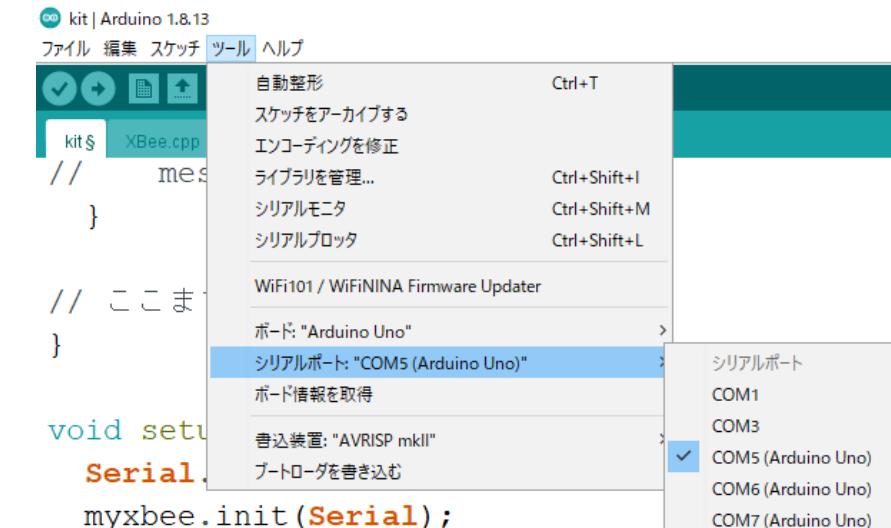




# 3台のArduinoへの書き込み方

75

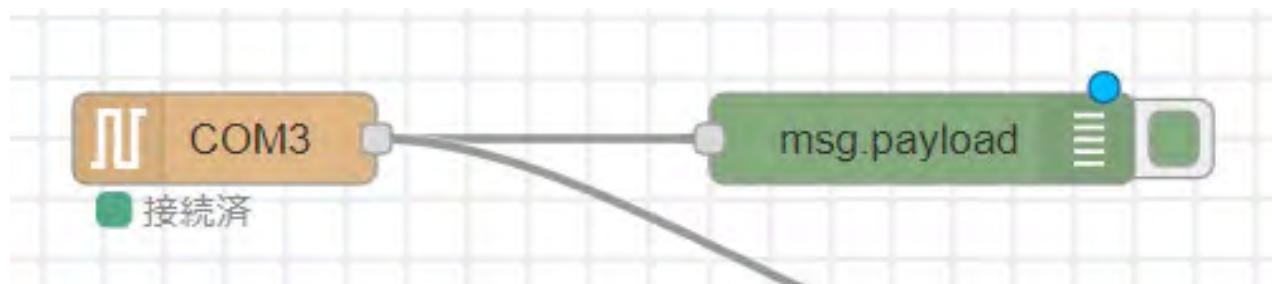
- 3台のArduinoの電源を入れる
- Arduinoのアプリを開いて左上のツールからシリアルポートを選ぶ
  - 3台のArduinoが見えているはず
- ここで選択しているArduinoに対してプログラムが書き込まれる
- 一度書き込まれたArduinoはそのプログラムをずっと覚えている
  - プログラムを編集しても上書きしない限り前に書き込まれたプログラムを覚えている
- 編集/書き込みの流れは以下の通り  
編集(識別子A)→選択/書き込み(COM5)→編集(識別子B)→選択/書き込み(COM6)  
→編集(識別子C)→選択/書き込み(COM7)





# 識別子の確認

- Node-Redでserial inとdebugを繋ぎ
- 受信したデータに識別子が付いている
- ことを確認する



デバッグ

▼ 全てのプロー

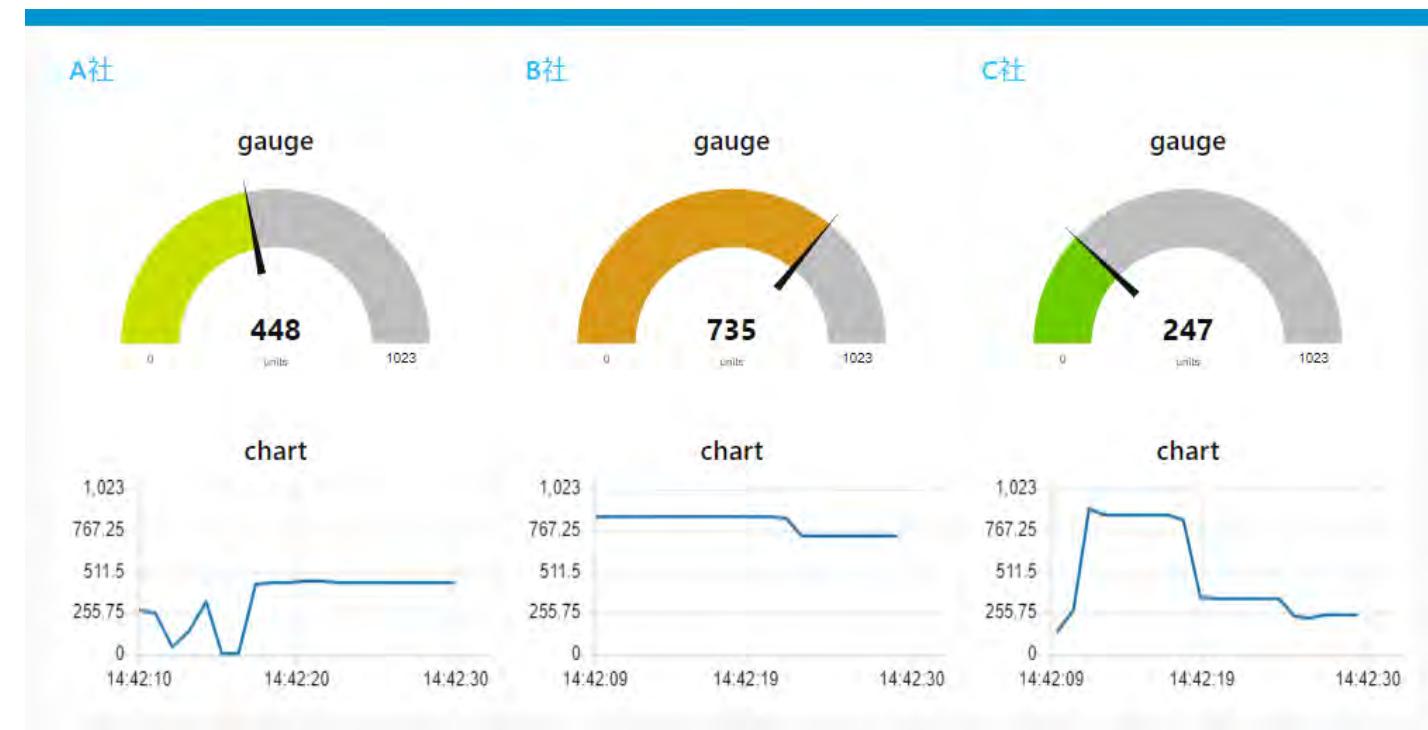
"♦~}3♦}3♦A♦日\*0♦日/"  
2022/5/24 14:29:10 node: a714ea41.2e4ff8  
msg.payload : string[6]  
"A, 267 /"  
2022/5/24 14:29:10 node: a714ea41.2e4ff8  
msg.payload : string[19]  
"♦~}3♦}3♦A♦日日日:日/"  
2022/5/24 14:29:10 node: a714ea41.2e4ff8  
msg.payload : string[6]  
"B, 698 /"  
2022/5/24 14:29:10 node: a714ea41.2e4ff8  
msg.payload : string[20]  
"♦~}3♦}3♦A♦日}1→f日/"  
2022/5/24 14:29:10 node: a714ea41.2e4ff8  
msg.payload : string[6]  
"C, 242 /"  
2022/5/24 14:29:11 node: a714ea41.2e4ff8



# 3台のセンサの値を分けて表示

77

- ここまで出来れば後は動画で習った内容を用いて3台のセンサの値を分けて表示することができる
- 以下のようにそれぞれのセンサの値を表示させてみよう





# ヒント

- switchで条件を複数作成して上手く識別子ごとに分岐させる
- splitでもよいがchangeの置換機能の方が使い易い
- シリアルのコンフリクトに注意すること
  - シリアルは複数を相手にできず、同じシリアルを複数で使うとコンフリクトします
  - ただし、この実験では基本的にはシリアルはコンフリクトしません
    - PCはSerial AでArduinoと通信
    - Node-REDはSerial BでXbeeボードと通信
- **XBeeの設定を変える場合などでコンフリクトする場合は、Node-REDを止める必要があります**
  - Node-REDを動かしているコマンドラインで**Ctrl-C**として止めると良いです
- エラーメッセージはよく読むこと
  - 書き込めない場合
    - **旧式ZigBee Arduino Shield**の場合は切り替えスイッチをまず見る
    - USBシリアルのポートや、Arduinoボードのタイプをよく見る
- **Node-RED解説動画（37分）をよく見よう！**





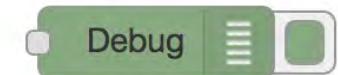
# Node-REDのノード（1）

- Injectノードは、エディタ内でクリックすることで手動でフローを始動させることができ、また、一定間隔で自動的にフローを始動させることもできる



- 送出メッセージに、payloadおよびtopicプロパティを設定できる
  - 通常はpayloadで、payloadには様々な型を設定することができます。：
    - フローまたはグローバルコンテキストのプロパティ値
    - 文字列型、数値型、Boolean型、バッファ型、オブジェクト型
    - 送信間隔（エポックミリ秒）

- Debugノードは、Debugサイドバーにメッセージを表示させる



- メッセージを受信した時刻とそのメッセージを送出したDebugノードの情報が含まれる
- ソースノードIDをクリックするとワークスペース内のどのノードなのかがわかる
- ノードのボタンで出力の有効化無効化を制御できる
- ランタイムログにすべてのメッセージを出力したり、Debugノードの下に短い（32文字）のステータステキストを表示させることができる

- Functionノードは、JavaScriptコードを実行する

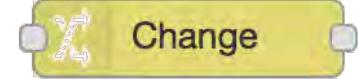
- 例えば、長さを返すならば、`var newMsg = {payload: msg.payload.length}; return newMsg;`





# Node-REDのノード（2）

- Changeノードは、メッセージプロパティ変更、コンテキストプロパティの設定が可能
  - 各ノードには、順番に適用される複数の操作を設定可能
    - 値の代入 - プロパティの代入、各種型を利用でき、既存のメッセージやコンテキストプロパティもOK
    - 値の置換 - メッセージプロパティの一部を検索、置換
    - 値の移動や削除 - プロパティの移動や名称変更、削除、プロパティ設定にJSONata形式を設定できる
- Switchノードは各メッセージを評価し、メッセージを異なるフローに振り分ける
  - 評価するプロパティ（メッセージプロパティまたはコンテキストプロパティ）を設定
  - ルールには以下の4種類がある
    - Value ruleは設定されたプロパティに対して評価する
    - Sequence ruleはSplitノードによって生成されるようなメッセージシーケンスを利用できる
    - JSONata式はメッセージ全体を評価し trueの値を返した場合に一致したとみなす
    - その他は前述のルールのどれにも一致しなかった場合に一致する
  - 全一致ルールにメッセージを送出するか、一致ルールがあった時点で評価をやめるか指定可能
- Templateノードは、メッセージプロパティからテキストを生成する(Mustache記法)
  - たとえば、This is the payload: {{payload}} !
- Splitノードは、フローを分割

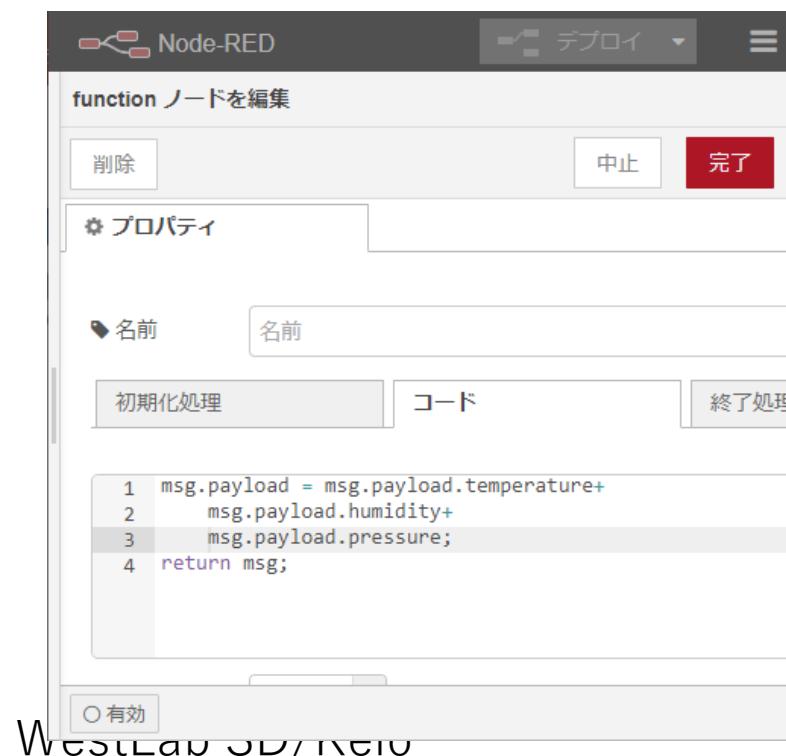




# Node-REDを用いた待ち合わせ計算について

81

- Node-REDは「複数のストリーム間の計算」がちょっと面倒
  - 当然ながら、異なるストリームなので同じタイミングに値はやってこない
- うまい方法を用いて、Node-REDは待ち合わせを行う
  - メッセージにtopicをつけてメッセージに名前を付ける
  - joinノードを用いて、右のように設定する
  - 「指定数のメッセージパートを受信後」で混ぜるストリームの数を指定
  - 最後にfunctionで足す
  - msg.payload.topic名となる



join ノードを編集

削除 中止 完了

プロパティ

動作: 手動

結合: msg. payload

出力: key/valueオブジェクト

使用する値: msg. topic をキーとして使用

メッセージ送信:

- 指定数のメッセージパートを受信後 3
- 最初のメッセージ受信からのタイムアウト後 秒
- msg.complete プロパティが設定されたメッセージ受信後

名前: 名前

有効

# 参考設計と動作

82

The screenshot shows the Node-RED interface with the following components:

- Left Sidebar:** Contains categories like "機能" (Function), "ネットワーク" (Network), and "MQTT" nodes.
- Flow Area:** Shows three parallel flows labeled "フロー 1", "フロー 2", and "フロー 3". Each flow starts with a "function" node (orange) with multiple output ports. These outputs converge at a "join" node (yellow). The "join" node then connects to another "function" node (orange), which finally outputs a "msg.payload" (green).
- Center Panel (Function Node Editor):**
  - Title:** function ノードを編集
  - Buttons:** 削除 (Delete), 中止 (Stop), 完了 (Finish)
  - Properties:** 明細 (Details) button
  - Name:** Name input field
  - Initialization:** 初期化処理 tab
  - Code:** msg.payload = msg.payload.d1+  
msg.payload.d2+  
msg.payload.d3;  
return msg;
  - Termination:** 終了処理 tab
  - Output:** 出力数: 1, 有効 (Enabled) checkbox
- Right Panel (Logs):** Displays log entries from the Node-RED runtime, showing messages like "2022/5/31 16:23:13 node: 2385016a.82c82e d3 : msg.payload : Object > { d1: 1, d2: 10, d3: 100 }" and "2022/5/31 16:23:17 node: 2385016a.82c82e d3 : msg.payload : Object > { d1: 2, d2: 20, d3: 200 }".





# Access to <https://fred.sensetecnic.com>

83

- フリーのNode-REDサーバを提供

The screenshot shows a browser window titled "Sense Tecnic Accounts" with the URL "users.sensetecnic.com/app/services". The main content area is titled "FRED" and contains the following text:  
Front End for Node-RED (FRED) manages instances of Node-RED for multiple users in the cloud. Node-RED is a visual tool for wiring the Internet of Things developed by IBM Emerging Technology and the open source community. With Node-RED you can wire up input, output and processing nodes to create flows to prototype IoT applications.  
[Click here to go to FRED >](#)

Below this, there is a "MQTT" section with the text: "The STS MQTT service allows you to send and receive data streams to your devices and FRED account easily using the standard MQTT protocol. Use the service to manage your client connections and access control for public and private topics." and a link "<https://mqtt.sensetecnic.com> >".

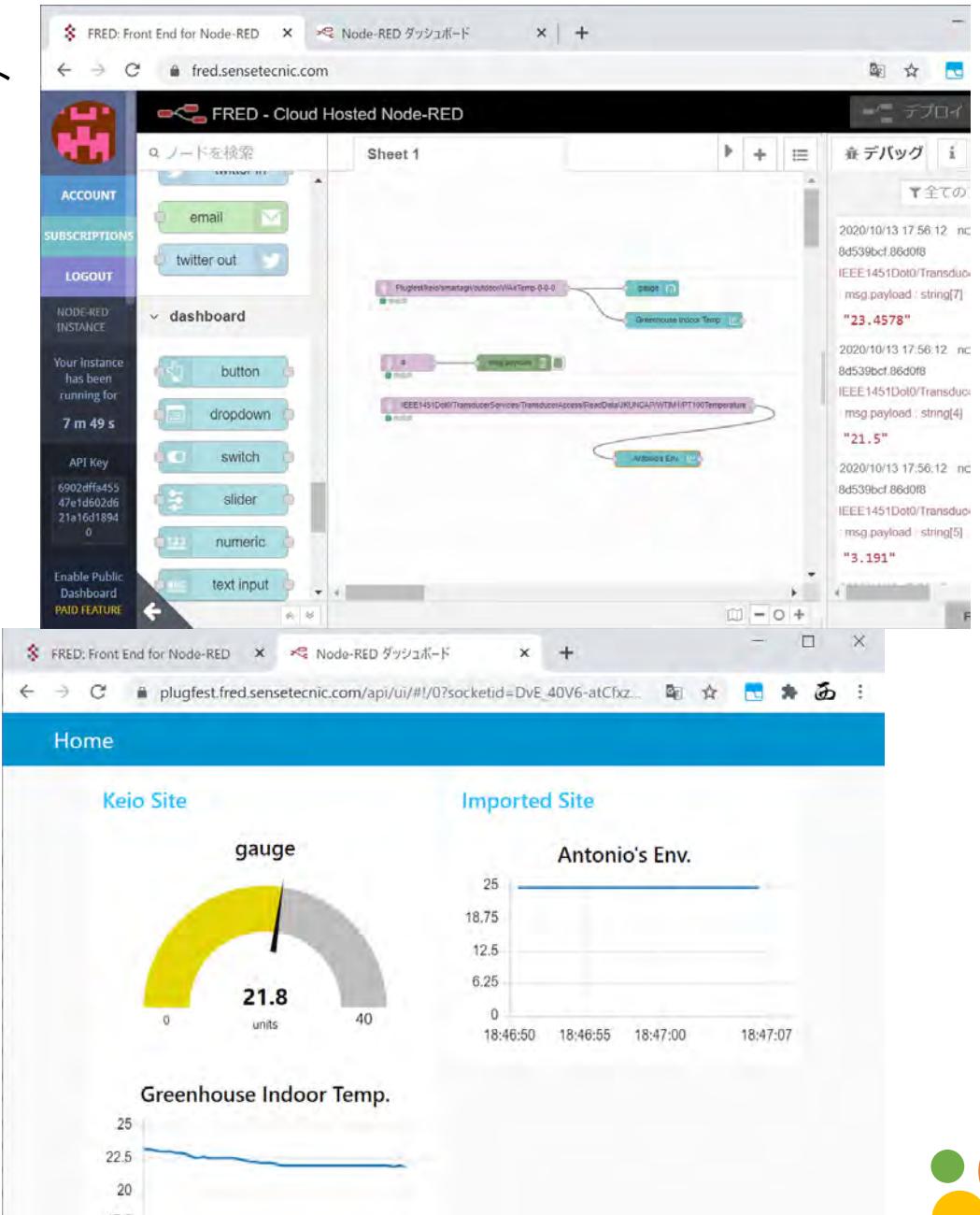
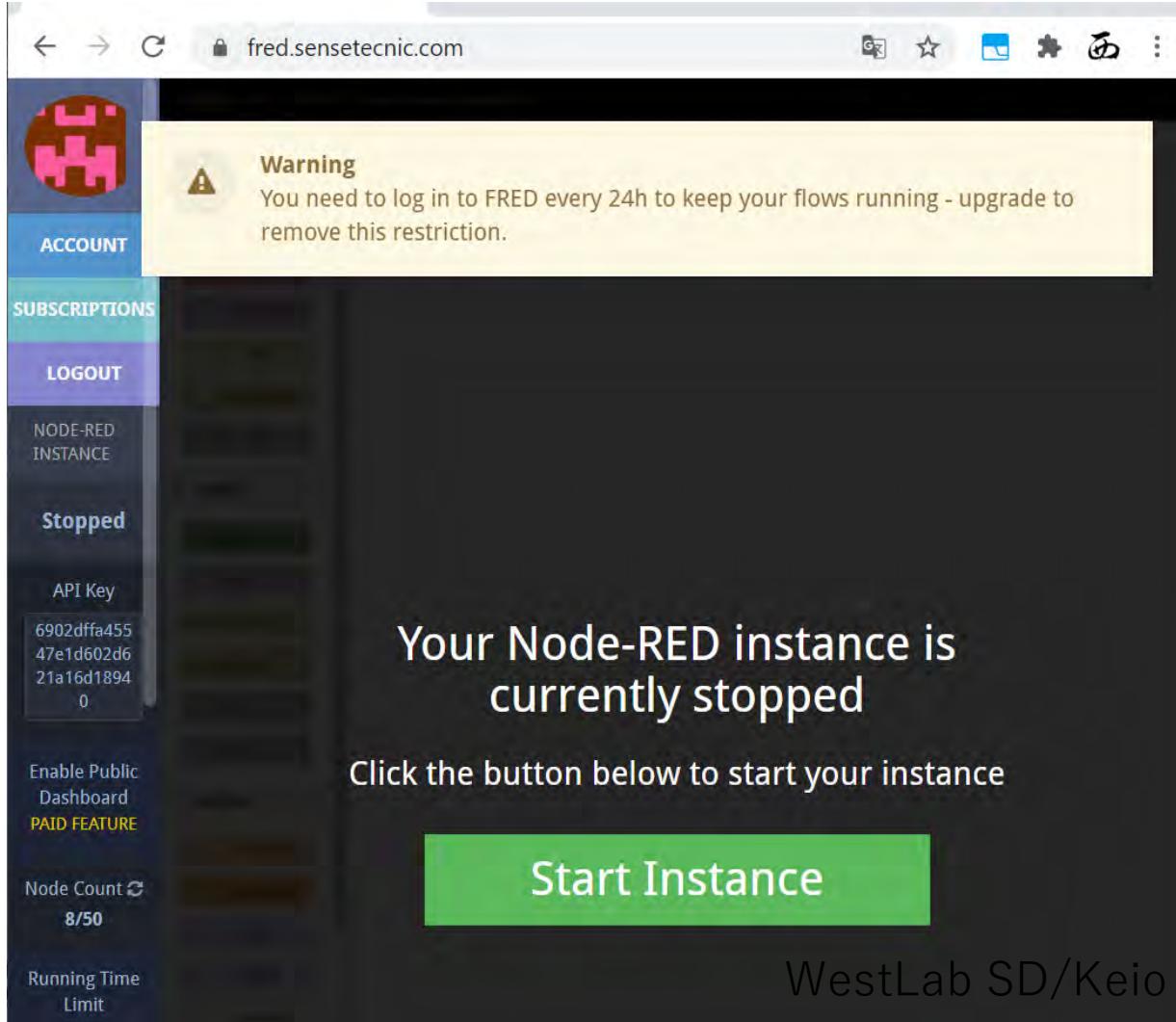
The screenshot shows a browser window titled "FRED: Front End for Node-RED" with the URL "fred.sensetecnic.com". The page has a dark background with large white text. It says "Welcome to FRED" and "We're hosting Node-RED so you don't have to". Below this, it lists "PAID PLANS" with four options:

- FRED Tall: \$9.99 /mo. Includes 150 node limit (limited memory) and 24x7 run time, always.
- FRED Grande: \$49.99 /mo. Recommended. Includes No node limit (up to 500mb memory) and 24x7 run time, always.
- FRED Venti: \$249 /mo. Includes No node limit (up to 1gb memory) and 24x7 run time, always.
- FRED Short: Free. Includes 50 node limit (limited memory) and 24 hour run time.



# Node-REDで設計する

- Start Instanceが現れたらこれを押すとスタート
- 他の人と設計を共有することもできる





# Node-REDを試してみよう

85



- MQTTとは
  - 普通の通信はPeer-to-Peerつまり、サーバクライアントモデル
  - サーバとクライアントの間で相互通信する
- 問題点
  - 例えば複数の相手に同時に投げたいし、相手も変えたいという場合に面倒で、負荷がかかる
  - 相手が変わったらサーバが変わる？設計が面倒
- MQTTはpublish-subscribeモデルである
  - トピックという名前でデータをブローカーに投げると、同じトピックを持っているクライアント全員にデータが届くという仕組みで、構造がシンプルなため組み込みでも簡単に実装できる
  - IoTでよく利用されるプロトコル
- Node-RED内の設計だけでデータ共有ができる！ペアを組んでやってみよう
  - MQTTブローカへのpublishノードで送り、subscribeノードで値を受け取って表示させればよい





# MQTT brokerの場所

- mosquittoを利用します
- 皆さんのPC
  - コマンドラインプロンプトを開く(ウィンドウズアイコン→cmdと入力)
  - cd c:\Program Files\mosquitto
  - mosquitto -v
  - 実行させたコマンドラインプロンプトは閉じないでください
- 実験部屋に専用のMQTT broker mosquittoを設置します
  - そのIPアドレス(相手先の名前)は別途お伝えします
  - ログイン・パスワードは必要ありません
- フリーのオーブンはMQTT brokerも利用できます
  - broker.hivemq.com
    - パスワードもログインも不要で、1883番ポートにアクセスすれば、すぐに利用できます。
  - test.mosquitto.org
    - 同様の設定で利用できます。その他、Googleで検索すると様々な同様のサービスがあることがわかります。





# 片付け

- 元通りに片づけること
  - 何も無くさないこと、何も壊さないこと
  - 力を入れて無理やりやらないこと
- ちゃんと蓋を閉じる
  - 閉じない場合は入れ方が間違えている
- 最後に確認しOKが出たら終了です
  - おつかれさまでした



# レポートと ディスカッション

- しっかりと学習内容をまとめよう





# レポートとディスカッション（1）

90

- レポートは手書き禁止
  - MS Wordなどを利用して書くこと（手書き文字は読みにくいのでスキャンも不可）
    - 回路図は、写メか、エミュレータで入力して全画面キャプチャしなさい（もしくは実行画面）
    - 設計した回路（もしくはコード）もすべてレポートのWordファイルに張り付けて、PDFで提出
- 実験目的、実験理論、実験手順、実験方法、実験結果などは絶対に記述するな
  - 実験レポートは写経ではなく、たとえテキストであっても写すのは剽窃と同じである
  - コピペレポートは厳禁と指導するのに、テキストを写させるのは意味不明である
  - 与えられた問題のみ回答すること
    - たくさん解いても加点されないが、ミスがあると減点されるため得をしない
    - シミュレータで設計可能であり、その設計のスクショと動作結果について記述、考察すること
- オリジナリティを最も重要視します（LMSによる自動剽窃チェックを利用します）
  - ググっても参考文献をつけること、参考文献の言及があれば剽窃ではない
    - 逆に手の内を明かしているので、覚悟を持ってこの剽窃チェックに捕まらないようにコピペしなさい
  - 残念ながら、偶然にも一致してしまった場合でも減点の対象となります
    - これは、オリジナリティが不足したことによる減点という扱いになります





# レポートとディスカッション（2）

- シミュレーションでIoTシステムを設計・動作
- ZigBeeは存在しないため**想定実装**すること。
- Arduino設計では、**シリアル出力に「本来通信される内容」を表示させること。**
  - シリアル出力すらないレポートは特に大きく減点されています。
  - 例えば、/1,100/などのように、ゴミと区別可能で、必要な情報が送られていることを示してください。
  - 想定実装であるが、実験同様ZigBeeが生成する**不要な文字列が取り除けるように工夫**すること。
  - Arduinoの設計はTinkercadを利用するとよい。（実機でもかまわない）
- Node-REDは**ノードの設計と動作内容をスクショすること。**
  - 設計に用いたノードの動作内容も記述すること。
  - **Arduinoの出力とNode-REDの入力のフォーマットが一致していること。**
  - Node-REDの設計はFREDを利用するとよい。（インストールしても構わない）
    - 理工学ITCでもNode-REDを準備して頂いたので利用可能です。
  - 想定実装であるが、**複数のInjectを混合させて仮想的にZigBeeからの通信を模擬すること。**
    - Inject毎にフローを作らず、一度束ねること。Node-RED説明動画にならって実装するとよい。
    - Tinkercadで設計したArduinoのシリアル出力について、仕様を説明できる適切な「代表値」をInjectするとよい。





# ディスカッションと再提出について

92

- レポート再提出およびディスカッションは対象者のみK-LMS連絡が届きます。
  - K-LMSのレポートに対するコメントとして記述します。
- 遅延レポートは提出場所が期限切れで提出できない場合があります。
  - 実験準備室に問い合わせて、対応してください。
- レポート再提出は提出場所が異なります。
  - 再レポート専用の提出場所に提出してください。





# Tinkercadの設計例

93

- 次のように、設計、コード、シリアルモニタの出力を確認してください。

Bodacious Gaaris-Kieran

すべての変更が保存されました

コード シミュレーションを停止 送信先

1 (Arduino Uno R3)

```
// C++ code
//
void setup()
{
    pinMode(0, INPUT);
    Serial.begin(9600);
}

void loop()
{
    int a;
    String s;
    a = analogRead(0);
    s = "/A,"+String(a)+"/";
    Serial.println(s);
    delay(300);
}
```

シリアル モニタ

```
/A,184/
/A,368/
/A,552/
/A,737/
/A,1003/
/A,634/
/A,368/
/A,327/
/A,205/
/A,266/
/A,471/
/A,491/
/A,491/
/A,491/
```

Wind のラクーンフリーウィンドウ  
設定を適用して行う場合はこのボタンをクリック

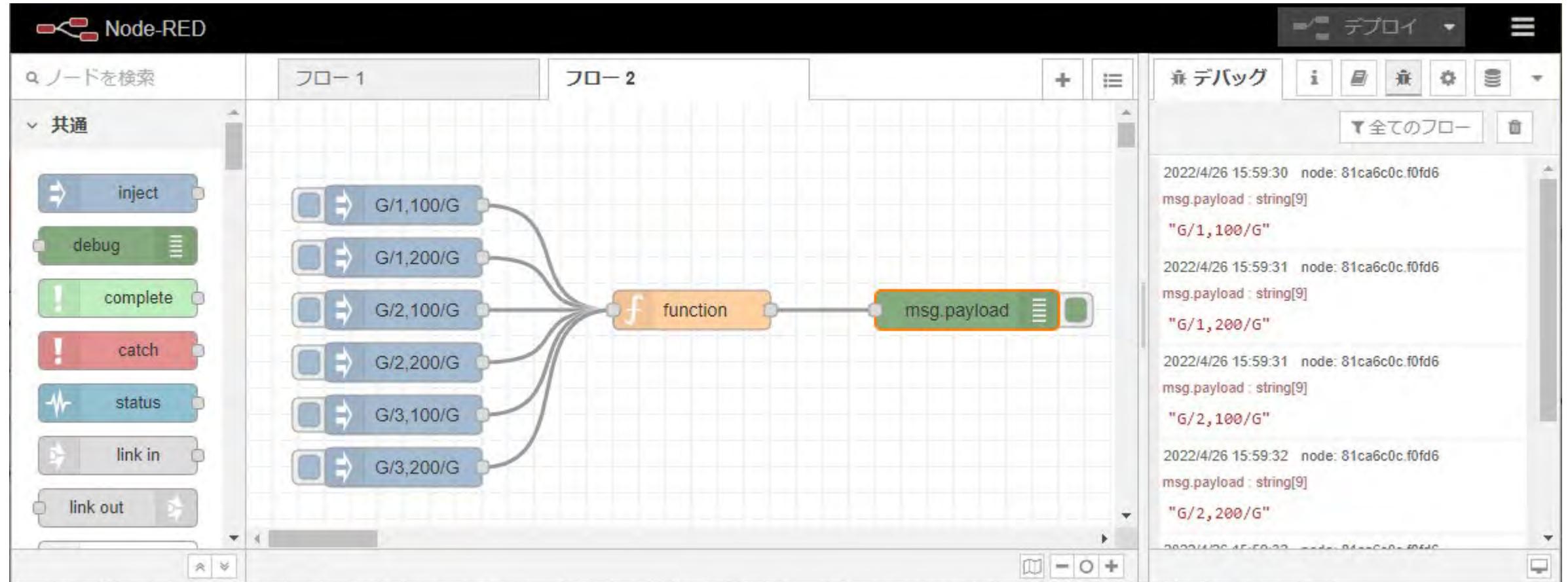




# Node-REDでの複数Arduinoからの入力

94

- 次のように、複数のInjectを結合して実現すること。





# レポートの点数をきちんと取るコツ

95

- Arduinoのコードはきちんと「テキスト」として貼り付けること。
- Arduinoのシリアル出力はあるか？通信に含まれるゴミを分けるセパレータはあるか？
- Arduinoは複数の端末を全て実装する必要はない。違いを書けばよい。ただし、Arduudinoのシリアル出力は複数の端末を見分けることができなければならない。
- Node-REDの全体設計、各ノードの設計は記述されているか？
- Node-REDの入力(Inject)の文字のフォーマットと、Arduinoのシリアル出力のフォーマットは一致しているか？
- Node-REDは複数のArduinoの情報が全て集まるため、これを分離できる必要がある。きちんと分離しているか？端末の個数分の処理が記述されているか？
- ここまでできて、規定の点数となる。加点はオリジナリティのある工夫になる。何がオリジナリティかきちんと記述して主張すると得点になりやすい。





# 演習問題集（1）

96

1. ボタンを押すとLEDが1秒点灯し、同様に離すと別のLEDが1秒点灯するようにせよ
2. 可変抵抗とサーボモータを組み合わせて、可変抵抗でサーボを制御せよ
  - 要するに、リモートロボットを作る
  - <https://www.arduino.cc/en/Tutorial/Sweep>
3. タイマー関数とCallbackを用いて、ボタンを押したら2秒間点灯するようにせよ
4. 毎秒LEDが200ms点灯するようにせよ
  - 正し、正確に毎秒であること
  - sleepなどを用いると、sleep以外の処理時間が加算されていく
5. 足し算電卓を作成せよ
  - ボタンは0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, =があればよい
  - 答えはシリアル表示とする
6. シリアル足し算電卓を作成せよ
  - シリアルから、式を入力してもらい、シリアルで答えを返す
  - 受け付けるのは上記と同じ





# 演習問題集（2）

## 7. ビンゴ抽選機を作れ

- 抽選ボタンを押すとビンゴ抽選を行い、ある数を「重複なく」シリアルで出す
- 全ての球が数字ができったら、「終了」LEDを点灯させる
- リセットボタンを押すと、初期状態に戻り、再び抽選できる
- 抽選の数字は1から30としなさい

## 8. 2個のArduinoを1本で接続し、LEDを制御せよ

- 片側のArduinoのボタンを押すと、もう片方のArduinoのLEDが1秒点灯し、同様に離すと別のLEDが1秒点灯する

## 9. 2個のArduinoをパラレル接続し、サーボを制御せよ

- 片側のArduinoのつまみの角度に応じて、もう片方のArduinoのサーボが動作する

## 10. 2個のArduinoをシリアル接続し、送信したa-zの文字をシリアルで出すようにせよ

- 配線電圧のHIGH、LOWを制御して伝えるようにすること

## 11. 2個のArduinoをシリアル接続し、サーボを制御せよ

- シリアル通信で片側のArduinoのつまみの角度に応じて、もう片方のArduinoのサーボが動作する





# 演習問題集（3）

12. 3個のArduinoをカスケードシリアル接続しLEDを制御せよ

- プライマリとなるArduinoのボタンを押すと、残り全てのArduinoのLEDが1秒点灯する
- （ヒント）カスケードなので、プライマリ以外のArduinoは親からの情報の受信と子への送信を同時に使う

13. 3個のArduinoをカスケードシリアル接続しサーボを制御せよ

- プライマリArduinoのつまみの角度に応じて、残り全てのArduinoのサーボが動作する

14. 3個のArduinoをカスケードシリアル接続し送信したa-zの文字をシリアルで出すようにせよ

- 配線電圧のHIGH、LOWを制御して伝えるようにすること
- プライマリとなるArduinoから送信すると残り全てに伝わるようにする

15. 3個のArduinoをカスケードシリアル接続し送信したa-zの文字をシリアルで出すようにせよ

- マスターを規定せず、カスケード接続をループで構成して、指令を出したArduino以外のArduinoで文字が出るようにせよ
- 配線電圧のHIGH、LOWを制御して伝えるようにすること
- すべてのArduinoが送信・受信両方共できるようにする





# 演習問題集（4）

16. 3個のArduinoをバスシリアル接続しLEDを制御せよ

- どれか一つのArduinoのボタンを押すと、残りのArduinoのLEDが1秒点灯する、同様に離すと、別のLEDが1秒点灯する
- (ヒント)バスなので、一つが送信で後の1つは同時に受ける

17. 3個のArduinoをバスシリアル接続しサーボを制御せよ

- どれか一つのArduinoのつまみの角度に応じて、残りのArduinoのサーボが動作する
- 制御するArduinoを選択するためのボタンかスイッチも設けなさい
- アクティブHighに接続した配線に対してLowに繋ぐと、全体としてLowになるなどする

18. 3個のArduinoをバスシリアル接続し送信したa-zの文字をシリアルで出すようにせよ

- シリアル通信は、それぞれHIGH、LOWを制御して伝えるようにすること
- どれか一つのArduinoから送信すると残り全てに伝わるようにする
- アクティブHighに接続した配線に対してLowに繋ぐと、全体としてLowになるなどする





# 演習問題集（5）

10  
0

19. 2個のArduinoをボーレートを合わす必要がないようにシリアル接続せよ

- 2本配線してよい

20. 2個のArduinoを次の仕様でシリアル接続せよ

- ボーレートを合わす必要がないようにすること(2本配線してよい)
- 通信中に配線を外しても、エラーとなって何も出ないようにする
- 配線外れを検出するわけではなく、通信中にエラーが発生しても問題ないようによること

21. 2個のArduinoをボーレートを合わす必要がないようにシリアル接続せよ

- 1本のみ配線するとし、100bps（動作限界は環境に依存する）までならばどのようなボーレートでもよいようにすること

22. 2個のArduinoを次の仕様でシリアル接続せよ

- ボーレートを合わす必要がないようにすること（1本のみ配線するとし、100bps（動作限界は環境に依存する）までならばどのようなボーレートでもよいようにすること）
- 通信中に配線を外しても、エラーとなって何も出ないようにする
- 配線外れを検出するわけではなく、通信中にエラーが発生しても問題ないようによること

