第5章：計測データ処理(IoT)の応用

KYUSHU
UNIVERSITY

全体構成

Chapter 1
- 第1章：導入

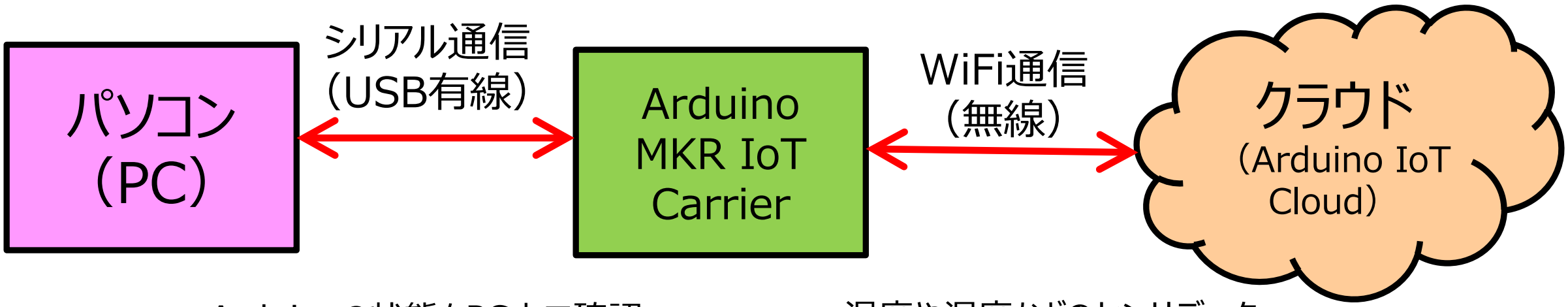
Chapter 2
- 第2章：IoT基礎（センサー）

Chapter 3
- 第3章：マイクロコントローラ（Arduino）の基礎

Chapter 4
- 第4章：計測データ解析の基礎

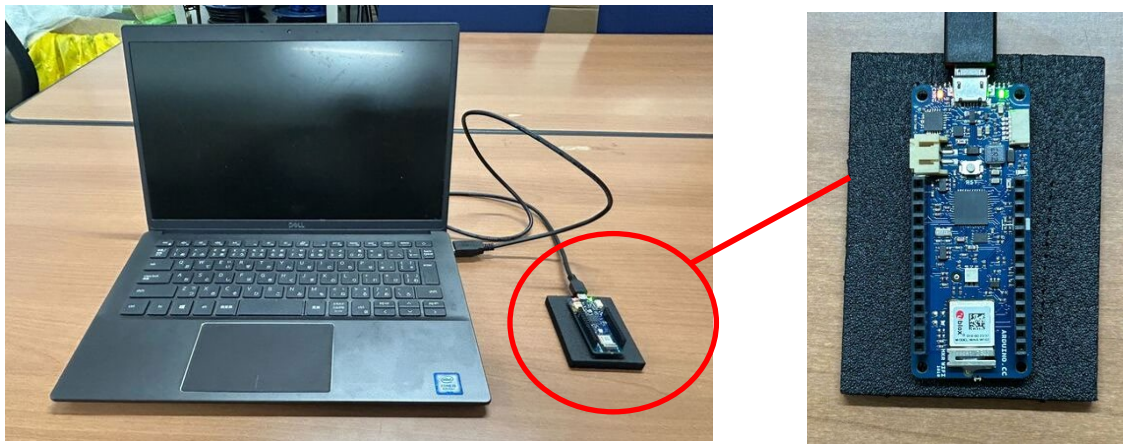
Chapter 5
- 第5章：計測データ処理(IoT)の応用

本演習で用いるシステム構成（1/4）

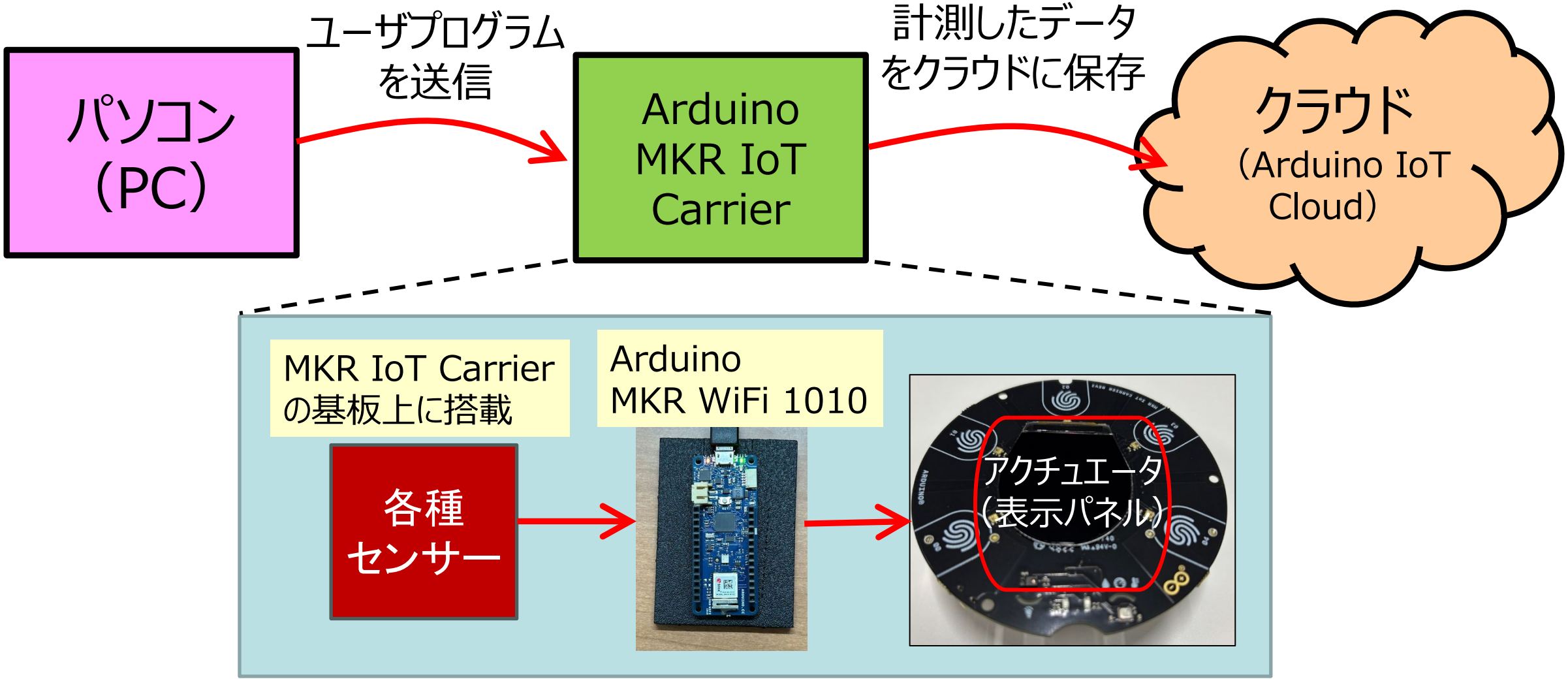


Arduinoの状態をPC上で確認
PCからArduinoにコマンド送信

温度や湿度などのセンサデータ
をリアルタイムでクラウドに送信

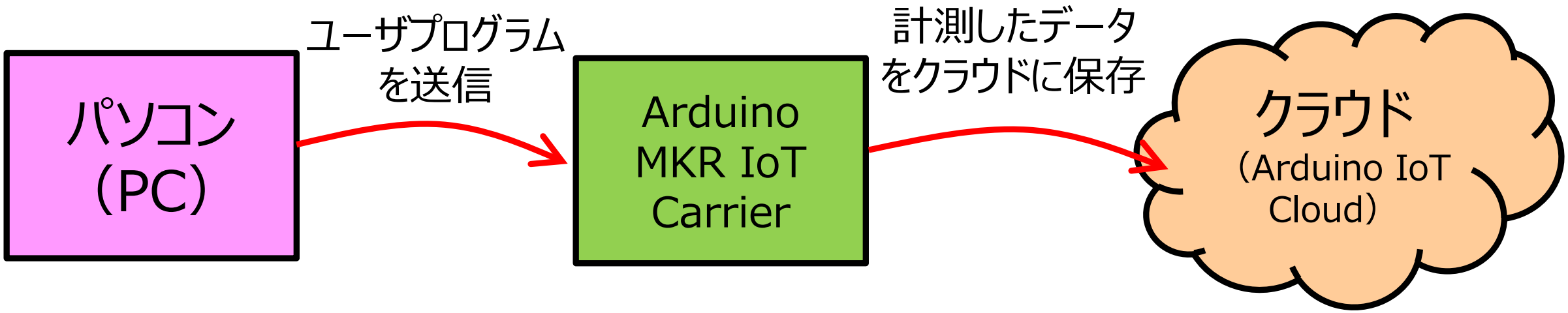


本演習で用いるシステム構成（2/4）



MKR IoT Carrier等のセットアップ手順や使い方の詳細は、付録に記載。

本演習で用いるシステム構成（3/4）



ここでは、以下のプログラムを作成する。

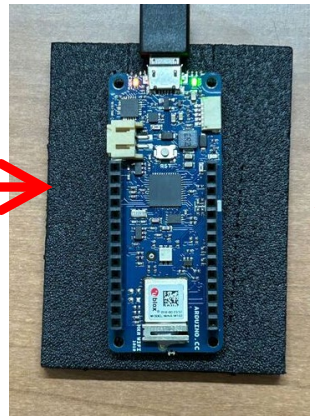
- 計測データは Arduino 内で「変数」として扱われる。

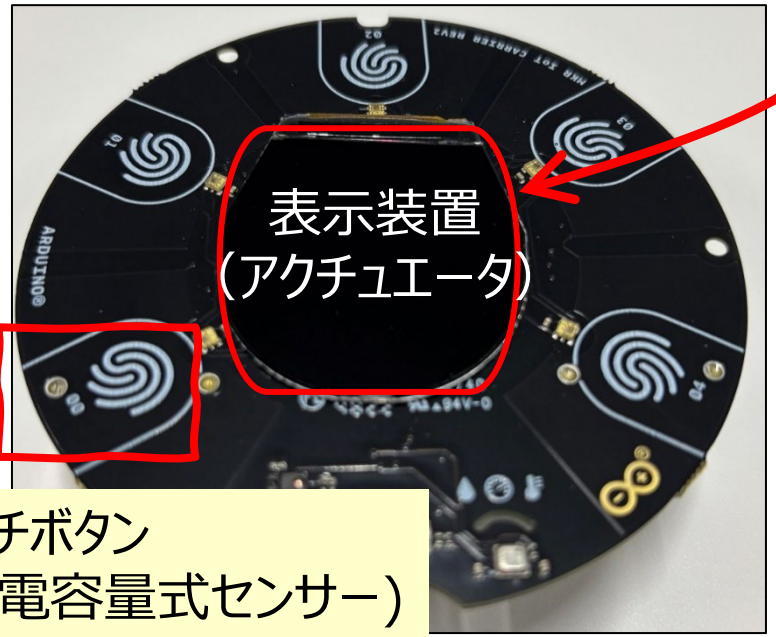
→センサーでの計測値を「変数」に入れてクラウドと同期するとともに、
その値を用いた計算・判定結果でアクチュエータを制御するプログラム

本演習で用いるシステム構成（4/4）

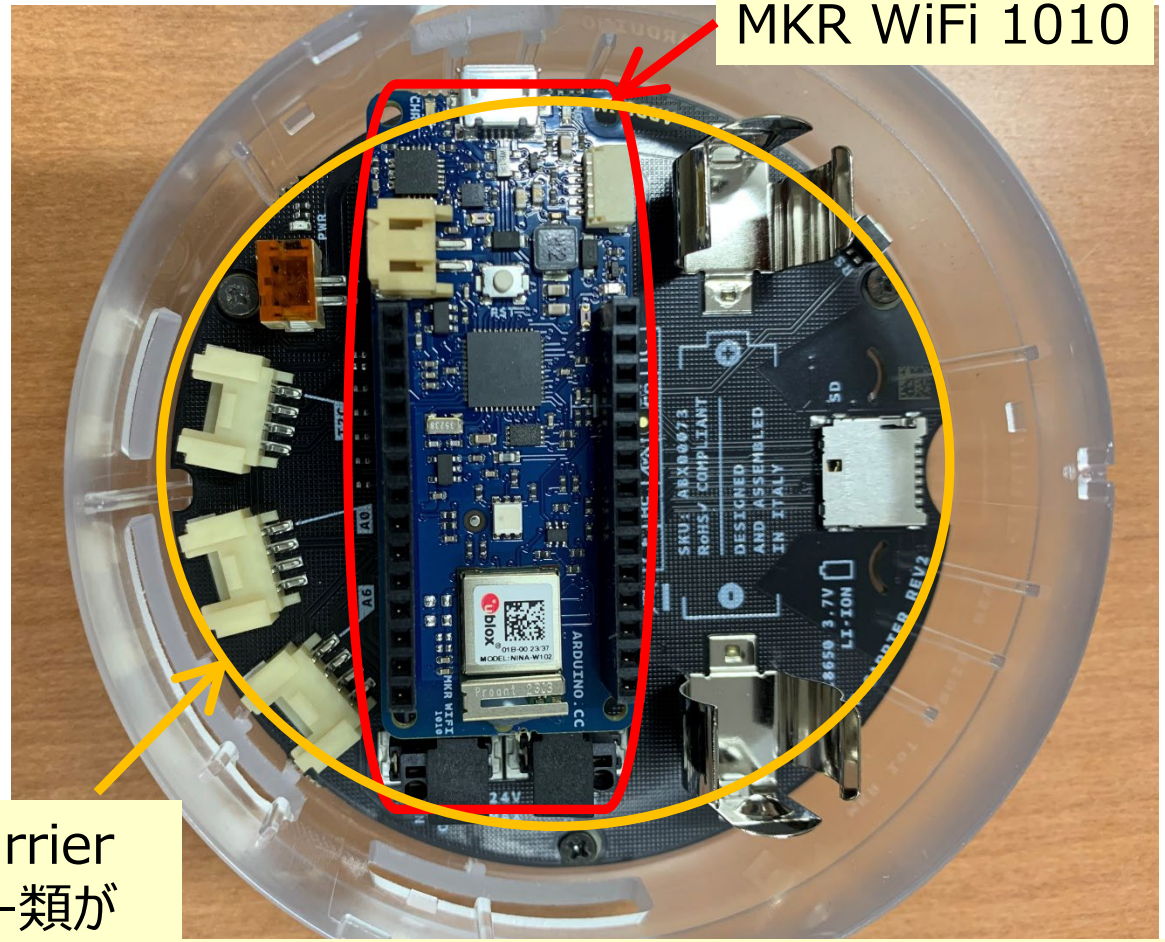
プロセッサ

各種
センサー





タッチボタン
(静電容量式センサー)



MKR IoT Carrier
以下のセンサー類が
基板上に搭載

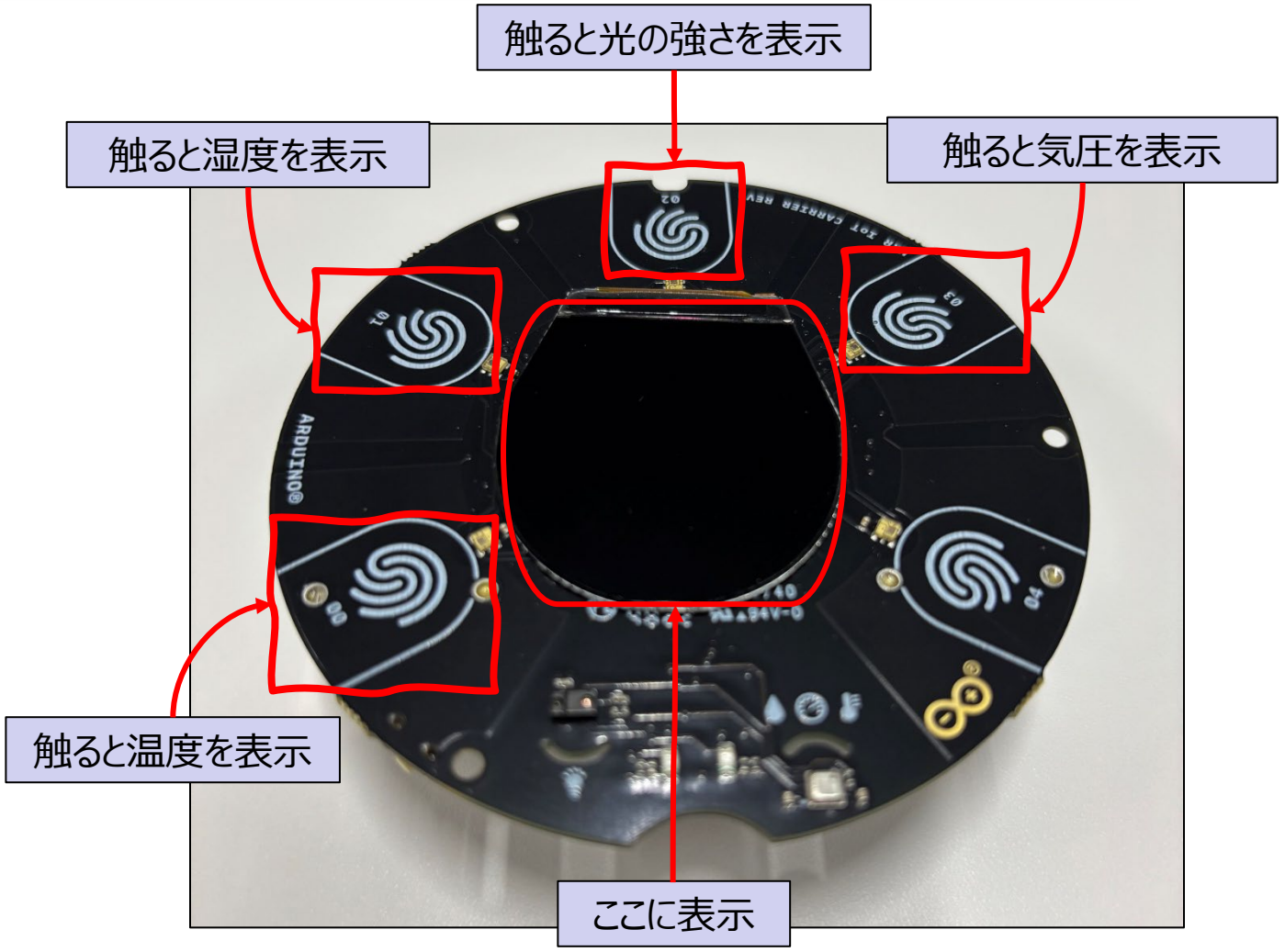
- 温度・湿度・気圧センサー（BME280センサー）
- 照度・光センサー（Ambient Light Sensor）
- カラーセンサー（APDS9960系）

実装例：気象モニタリングステーション

以下の機能を本演習では実装したい。
そのためのスケッチをここでは作成する

ボタン操作でデバイスに以下の計測値（データ）を表示

- ボタン０：温度
- ボタン１：湿度
- ボタン２：光量
- ボタン３：気圧

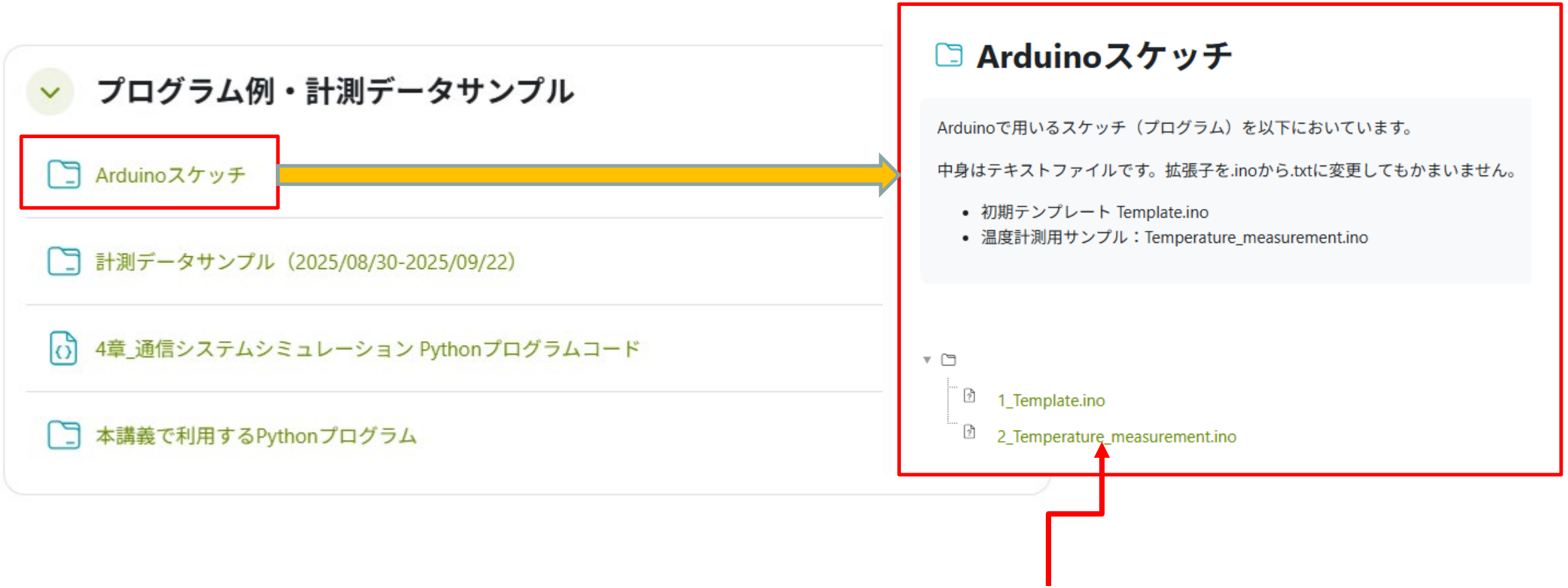


本演習の計測処理の流れ

1. 計測器の設定プログラムの導入
2. 計測器の動作確認
3. 計測器によりデータを取得する
4. 計測データを解析する

1. 計測器の設定プログラムの導入

まず、付録に記載の環境設定を行う。



2_Temperature_measurement.ino
をダウンロードする。

1. 計測器の設定プログラムの導入

2_Temperature_measurement.ino は温度値を取得し、それをクラウドに送信するプログラムです。これを以下の内容に拡張します。

1. センサ計測データ（湿度値、温度値、気圧値、光量値）をクラウドから取得する

2. 押下されたボタンに応じて、デバイスの画面を更新する

3. センサ計測データの更新をクラウドに送信する

1．計測器の設定プログラムの導入

スケッチの全体像

1．環境設定

2．setup()関数

3．loop()関数

```
#include "thingProperties.h“
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;

void setup()  {
    // Initialize serial and wait for port to open:
    Serial.begin(9600);


   …

}

void loop() {
     ArduinoCloud.update();
     // Your code here


   …

}
```

1. 計測器の設定プログラムの導入


```
#include "thingProperties.h“
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;

void setup()  {
    // Initialize serial and wait for port to open:
    Serial.begin(9600);


   ...


}

void loop() {
     ArduinoCloud.update();
     // Your code here


   ...


}
```

“thingProperties.h”を読み込む

Arduino IoT CloudのWiFi通信設定を含むヘッダファイル。Thing（クラウド上のデバイス）との接続情報などが含まれる。

“Arduino_MKRIoTCarrier.h”を読み込む

MKR IoT Carrier※1用ライブラリを読み込む。温度・湿度・気圧センサなどの機能を使用するために必要なプログラム群が含まれる。

“MKRIoTCarrier carrier;”

上記で読み込んだライブラリを用いてセンサや出力デバイスを操作したい。そのための窓口となる操作対象を「carrier」という名前で用意する。

例えば、光センサを利用する場合は carrier.Light.read()、温度センサを用いる場合は carrier.Env.readTemperature() のように用いる。

※1：各種センサ等の周辺機能を搭載(Carry)した基盤(拡張ボード)をCarrierと呼んでいる。

1. 計測器の設定プログラムの導入


```
#include "thingProperties.h“
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;

void setup()  {
    // Initialize serial and wait for port to open:
    Serial.begin(9600);

   …

}

void loop() {
     ArduinoCloud.update();
     // Your code here

   …

}
```

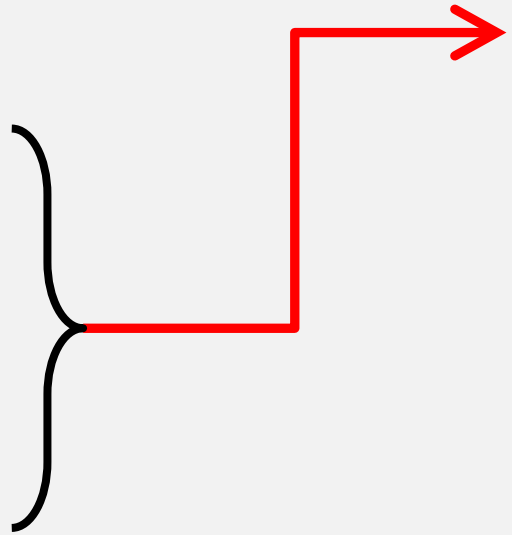
→ setup() 関数：初期設定を行う

センサーを用いる準備やクラウド接続の設定を行う。プログラムの最初に1回だけ実行する。具体的には以下の処理を行う。

1. シリアル通信を初期化し、ログを出力可能にする。
2. Cloud連携用の設定（Thing名、変数等の設定）
3. Arduino Cloudとの接続を開始する。

1. 計測器の設定プログラムの導入


```
#include "thingProperties.h“
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;

void setup()  {
    // Initialize serial and wait for port to open:
    Serial.begin(9600);


   …

}

void loop() {
     ArduinoCloud.update();
     // Your code here


   …

}
```



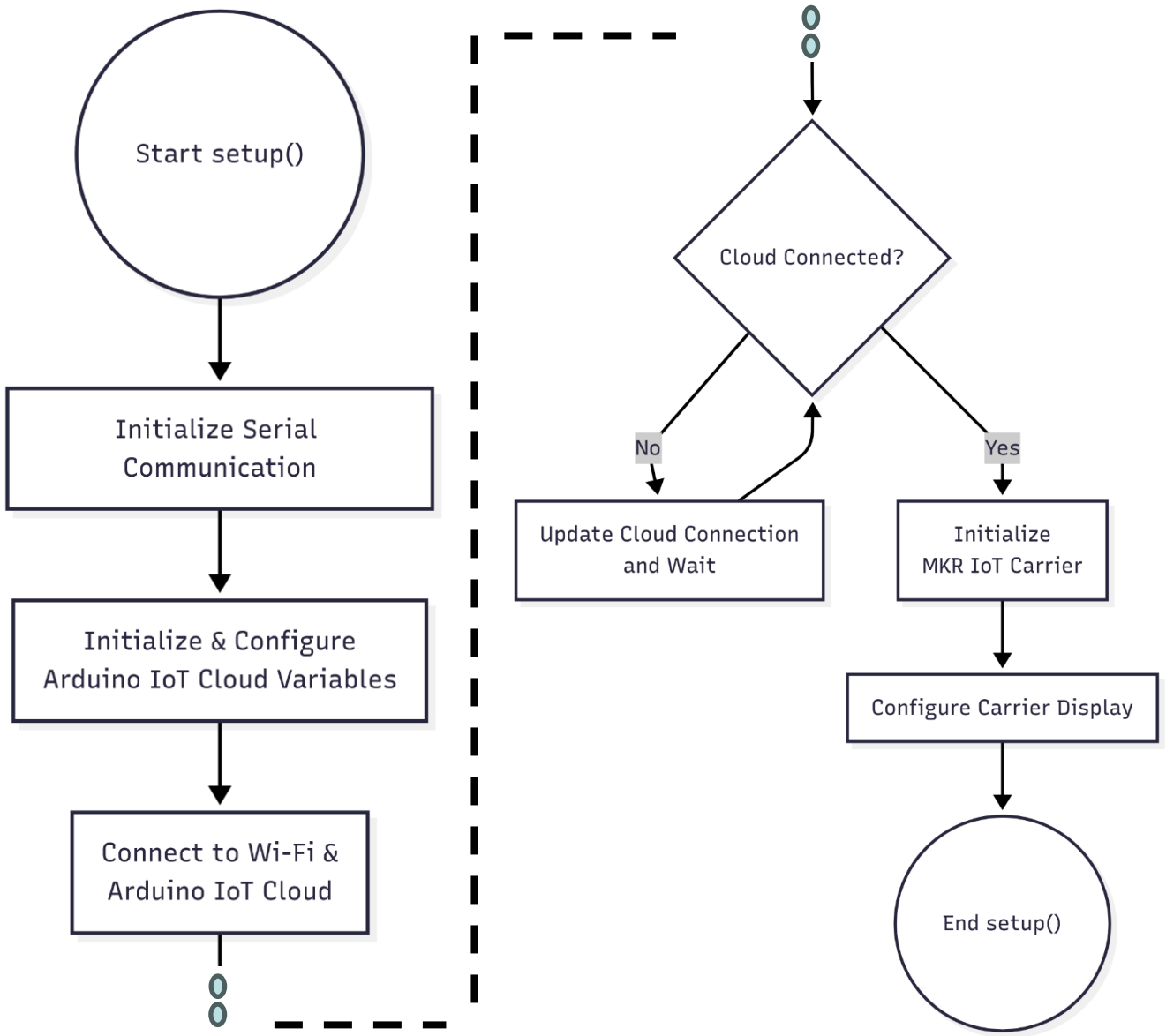
loop() 関数：繰り返し処理を行う

- センサーの値を読み取って表示したり、クラウドに送信する処理をここに記載する。この関数に記載した処理が繰り返し実行される。
- 本演習で用いるセンサーの読み取り処理などもこの関数の中に追加する。

1. 計測器の設定プログラムの導入

setup()関数（1/5）

1. シリアル通信の初期化：ArduinoとPC間の通信を可能する
2. Arduino IoT Cloud変数の初期化：クラウドとの同期に必要な内部変数の準備
3. WiFi接続：事前に設定したSSIDとパスワードでWiFiに接続
4. クラウド接続の確認：接続成功なら次に、失敗なら再試行
5. MKR IoT Carrierの初期化：各種センサ等を利用するためにハードウェアを起動し、ディスプレイ等の設定を行う



1. 計測器の設定プログラムの導入

setup()関数（2/5）

setup()関数には、Arduinoがセンサーデータを扱えるようにするための初期化処理（シリアル通信とクラウド接続の準備）が記載されている。

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  //Get Cloud Info/errors , 0 (only errors) up to 4
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  //Wait to get cloud connection to init the carrier
  while (ArduinoCloud.connected() != 1) {
    ArduinoCloud.update();
    delay(500);
  }
  delay(500);
  CARRIER_CASE = false;
  carrier.begin();
  carrier.display.setRotation(0); delay(1500);
}
```

Serial.begin(9600);
PC-Arduino間のシリアル通信の初期化
（9600bpsの速度で通信開始）

delay(1500);
シリアルモニタを開くまでの時間を確保するため、1.5秒間（1500ミリ秒）待機する。
※ Arduino IDEではArduinoにスケッチを書き込んだ後、手動でSerial monitorを起動する。

initProperties();
クラウド接続に必要な情報（WiFi・デバイスID等）を読み込む。

1. 計測器の設定プログラムの導入

setup()関数（3/5）

setup()関数には、Arduinoがセンサーデータを扱えるようにするための初期化処理が記載されている。

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  //Get Cloud Info/errors , 0 (only errors) up to 4
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  //Wait to get cloud connection to init the carrier
  while (ArduinoCloud.connected() != 1) {
    ArduinoCloud.update();
    delay(500);
  }
  delay(500);
  CARRIER_CASE = false;
  carrier.begin();
  carrier.display.setRotation(0); delay(1500);
}
```

ArduinoCloud.begin()
事前に設定されたWi-Fi経由で、Arduino IoT Cloudに接続を開始する

setDebugMessageLevel(2);
ArduinoCloud.printDebugInfo();

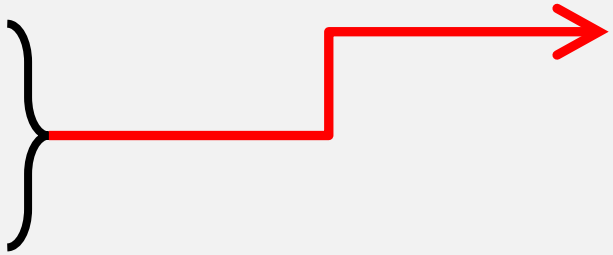
デバッグ情報の設定と表示
（レベル２はエラーの表示のみ）

1. 計測器の設定プログラムの導入

setup()関数（4/5）

setup()関数には、Arduinoがセンサーデータを扱えるようにするための初期化処理が記載されている。

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  //Get Cloud Info/errors , 0 (only errors) up to 4
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  //Wait to get cloud connection to init the carrier
  while (ArduinoCloud.connected() != 1) {
    ArduinoCloud.update();
    delay(500);
  }
  delay(500);
  CARRIER_CASE = false;
  carrier.begin();
  carrier.display.setRotation(0); delay(1500);
}
```



```
while (ArduinoCloud.connected() != 1) {
    ArduinoCloud.update();
    delay(500);
  }
```

- ArduinoCloud.connected()

→ この関数の返値が1の場合はクラウドに接続済み、0の場合は未接続を表す。

- ArduinoCloud.update();

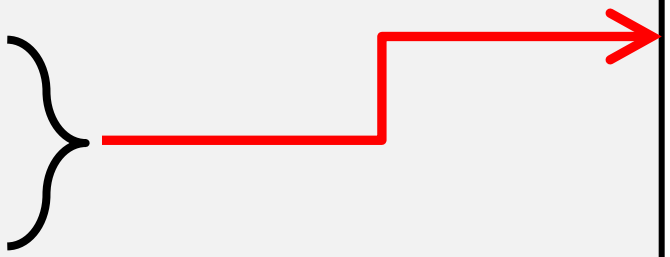
→ ArduinoとCloudとの間で通信処理を行い、クラウドとローカルの状態を同期するための関数

クラウドに接続が成功するまで、確認を繰り返す処理を行っている

setup()関数（5/5）

setup()関数には、Arduinoがセンサーデータを扱えるようにするための初期化処理が記載されている。

```
void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  //Get Cloud Info/errors , 0 (only errors) up to 4
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  //Wait to get cloud connection to init the carrier
  while (ArduinoCloud.connected() != 1) {
    ArduinoCloud.update();
    delay(500);
  }
  delay(500);
  CARRIER_CASE = false;
  carrier.begin();
  carrier.display.setRotation(0); delay(1500);
}
```



delay(500);
0.5秒の待機時間を設ける。

CARRIER_CASE = false;
Carrierボードがケースに入っていないことを指定
（ケースにいれている場合、falseのままでは、ボタンを押しても反応しない）

ケースありでもボタン操作を可能にするには、この変数を true に変更するとよい。
（試してみよう！）

carrier.begin();
Carrierボードの初期化

carrier.display.setRotation(0);
Carrierボードにおけるディスプレイの表示向きを「0度」に設定（0～3の値で回転方向（0°, 90°, 180°,270°）を指定）

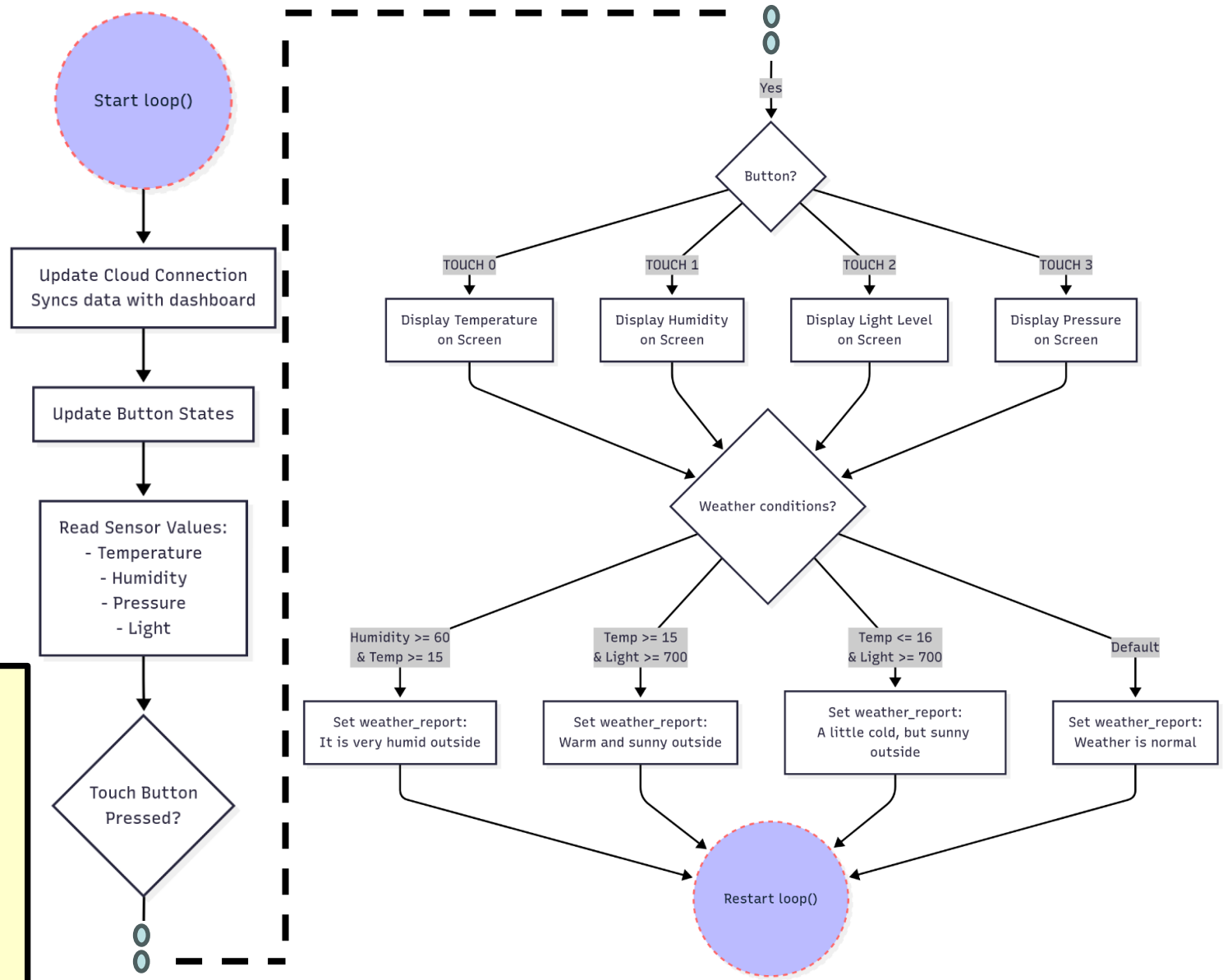
delay(1500);
1.5秒の待機時間

1. 計測器の設定プログラムの導入

loop()関数（1/8）

1. クラウドと通信し状態を更新する。センサーから温度・湿度・気圧・照度の値を取得
2. タッチボタンの入力を確認し、押されたボタン（番号）に応じてセンサーの計測値を画面に表示
3. 現在の天候条件を判定し、クラウドに以下の状態メッセージ(英文)を送信

- 湿度 ≥ 60 かつ 温度 ≥ 15℃
  → とても湿っている(It is very humid outside)
- 温度 ≥ 15 かつ 明るさ ≥ 700
  → 温暖で晴れている(Warm and sunny outside)
- 温度 ≤ 16 かつ 明るさ ≥ 700
  → 少し寒いが晴れている(A little cold, but sunny outside)
- それ以外（デフォルト）
  → 天候は通常（Weather is normal）



1. 計測器の設定プログラムの導入

loop()関数（2/8）

```
void loop() {
    ArduinoCloud.update();
    // Your code here
    carrier.Buttons.update();

    while(!carrier.Light.colorAvailable()) {
        delay(5);
    }
    int none;
    carrier.Light.readColor(none, none, none, light);
    temperature = carrier.Env.readTemperature();
    humidity = carrier.Env.readHumidity();
    pressure = carrier.Pressure.readPressure();

    if (carrier.Buttons.onTouchDown(TOUCH0))  {
      carrier.display.fillScreen(ST77XX_WHITE); carrier.display.setTextColor(ST77XX_RED);
      carrier.display.setTextSize(2);          carrier.display.setCursor(30,110);  carrier.display.print("Temp:");
      carrier.display.print(temperature);      carrier.display.print(" C");
    }

   if (carrier.Buttons.onTouchDown(TOUCH1))  {
     carrier.display.fillScreen(ST77XX_WHITE); carrier.display.setTextColor(ST77XX_RED);
     carrier.display.setTextSize(2);          carrier.display.setCursor(30,110);  carrier.display.print("Humi:");
     carrier.display.print(humidity);         carrier.display.print(" %");
   }

   if (carrier.Buttons.onTouchDown(TOUCH2)) {
     carrier.display.fillScreen(ST77XX_WHITE); carrier.display.setTextColor(ST77XX_RED);
     carrier.display.setTextSize(2);          carrier.display.setCursor(30, 110);  carrier.display.print("Light:");
     carrier.display.print(light);
   }

   if (carrier.Buttons.onTouchDown(TOUCH3))  {
     carrier.display.fillScreen(ST77XX_WHITE);     carrier.display.setTextColor(ST77XX_RED);
     carrier.display.setTextSize(2);              carrier.display.setCursor(30,110);  carrier.display.print("Pressure:");
     carrier.display.print(pressure);
   }

   if (humidity >= 60 && temperature >= 15)  {
     weather_report = "It is very humid outside";

   } else if (temperature >= 15 && light >= 700)  {
       weather_report = "Warm and sunny outside";

   } else if (temperature <= 16 && light >= 700)  {
      weather_report = "A little cold, but sunny outside";

   } else {
       weather_report = "Weather is normal";
   }
}
```

① センサーから温度・湿度・気圧・明るさを取得

② タッチボタン別の表示処理

③ 天気の条件判定（クラウドに送信する weather report の設定）

1. 計測器の設定プログラムの導入

loop()関数（3/8）

① センサーから温度・湿度・気圧・明るさを取得

```
ArduinoCloud.update();
// Your code here
carrier.Buttons.update();

while(!carrier.Light.colorAvailable()) {
    delay(5);
}
int none;
carrier.Light.readColor(none, none, none, light);

temperature = carrier.Env.readTemperature();
humidity = carrier.Env.readHumidity();
pressure = carrier.Pressure.readPressure();
```

ArduinoCloud.update()
クラウドとの通信状態や変数の同期

carrier.Buttons.update()
タッチボタンの押下状態を更新

光センサーの準備が完了するまで待機する

colorAvailable()
光センサーの準備完了を確認する関数（返値：true / false）
※ループが速くなりすぎないように5ミリ秒の遅延を加えている。

int none;
carrier.Light.readColor(none, none, none, light);
光センサーからRGBの値を取得する関数。ここでは、明るさ（light）のみを抽出し、他は捨てている、none はダミー変数。

MKR IoT Carrierに搭載された温湿度・気圧センサーの測定値データを取得

1. 計測器の設定プログラムの導入

loop()関数（4/8）

② タッチボタン別の表示処理：温度を表示（ボタン0）

```
if (carrier.Buttons.onTouchDown(TOUCH0))  {
    carrier.display.fillScreen(ST77XX_WHITE);
    carrier.display.setTextColor(ST77XX_RED);
    carrier.display.setTextSize(2);

    carrier.display.setCursor(30,110);
    carrier.display.print("Temp:");
    carrier.display.print(temperature);
    carrier.display.print(" C");
  }
```

タッチボタン0（左下）が押されたことを検出

ディスプレイ全体の背景色を白に設定

テキストの色を赤に設定

テキストのフォントサイズを2倍に設定

表示位置を (x=30, y=110) に設定
「Temp:」という文字列を表示
センサから取得した温度を表示
「 C」を数値の後に表示（単位：摂氏）

1. 計測器の設定プログラムの導入

loop()関数（5/8）

② タッチボタン別の表示処理：湿度を表示（ボタン1）

湿度表示のプログラム
を記載しよう
（練習課題）

1. 計測器の設定プログラムの導入

loop()関数（6/8）

② タッチボタン別の表示処理：光量を表示（ボタン2）

光量表示のプログラム
を記載しよう
（練習課題）

1. 計測器の設定プログラムの導入

loop()関数（7/8）

② タッチボタン別の表示処理：気圧を表示（ボタン3）

気圧表示のプログラム
を記載しよう
（練習課題）

loop()関数（8/8）

③ 天気の条件判定（クラウドに送信する weather report の設定）

```
if (humidity >= 60 && temperature >= 15)  {
      weather_report = "It is very humid outside";

    } else if (temperature >= 15 && light >= 700)  {
        weather_report = "Warm and sunny outside";

    } else if (temperature <= 16 && light >= 700)  {
       weather_report = "A little cold, but sunny outside";

    } else {
        weather_report = "Weather is normal";
    }
```

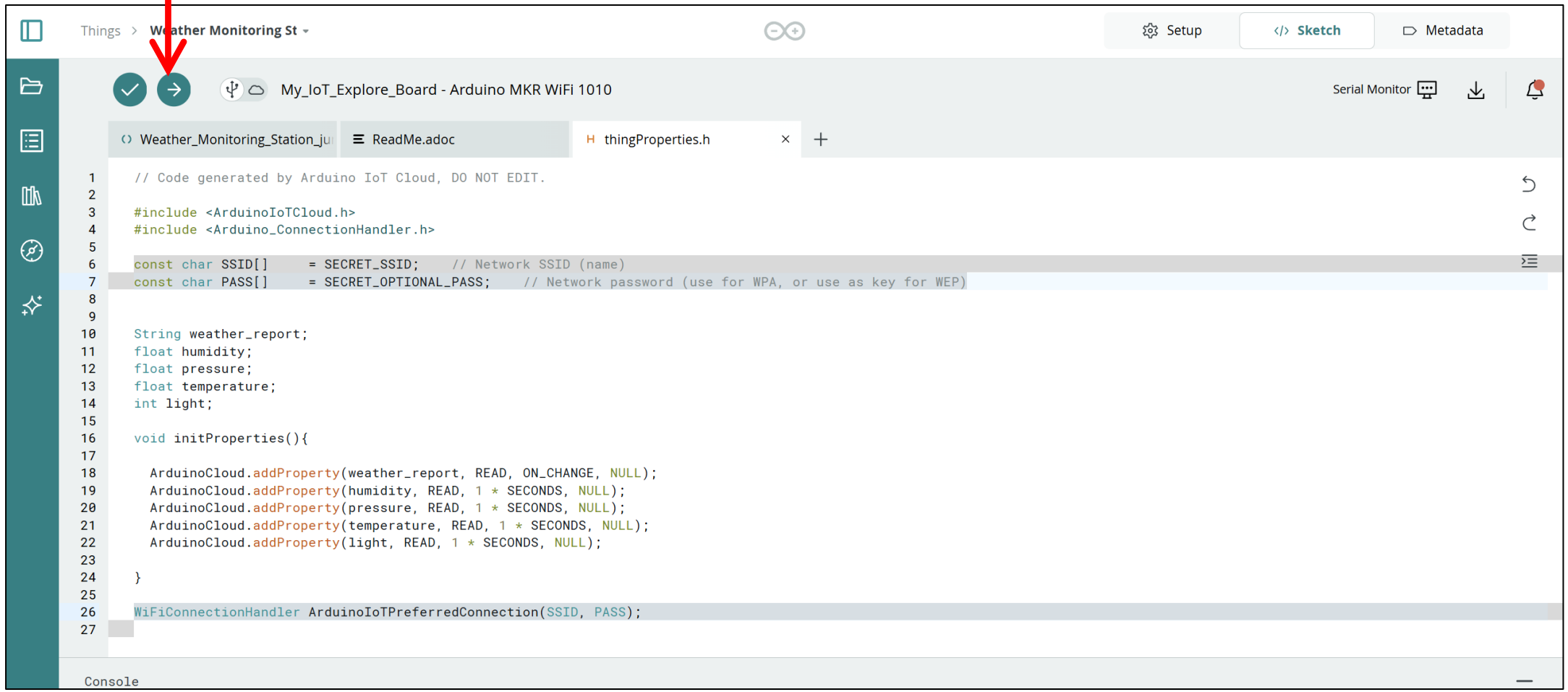
装置に表示するメッセージや
条件を変更しよう
（練習課題）

- 湿度 ≥ 60 かつ 温度 ≥ 15℃
  → とても湿っている(It is very humid outside)
- 温度 ≥ 15 かつ 明るさ ≥ 700
  → 温暖で晴れている(Warm and sunny outside)
- 温度 ≤ 16 かつ 明るさ ≥ 700
  → 少し寒いが晴れている(A little cold, but sunny outside)
- それ以外（デフォルト）
  → 天候は通常（Weather is normal）

本演習の計測処理の流れ

1. 計測器の設定プログラムの導入
2. 計測器の動作確認
3. 計測器によりデータを取得する
4. 計測データを解析する

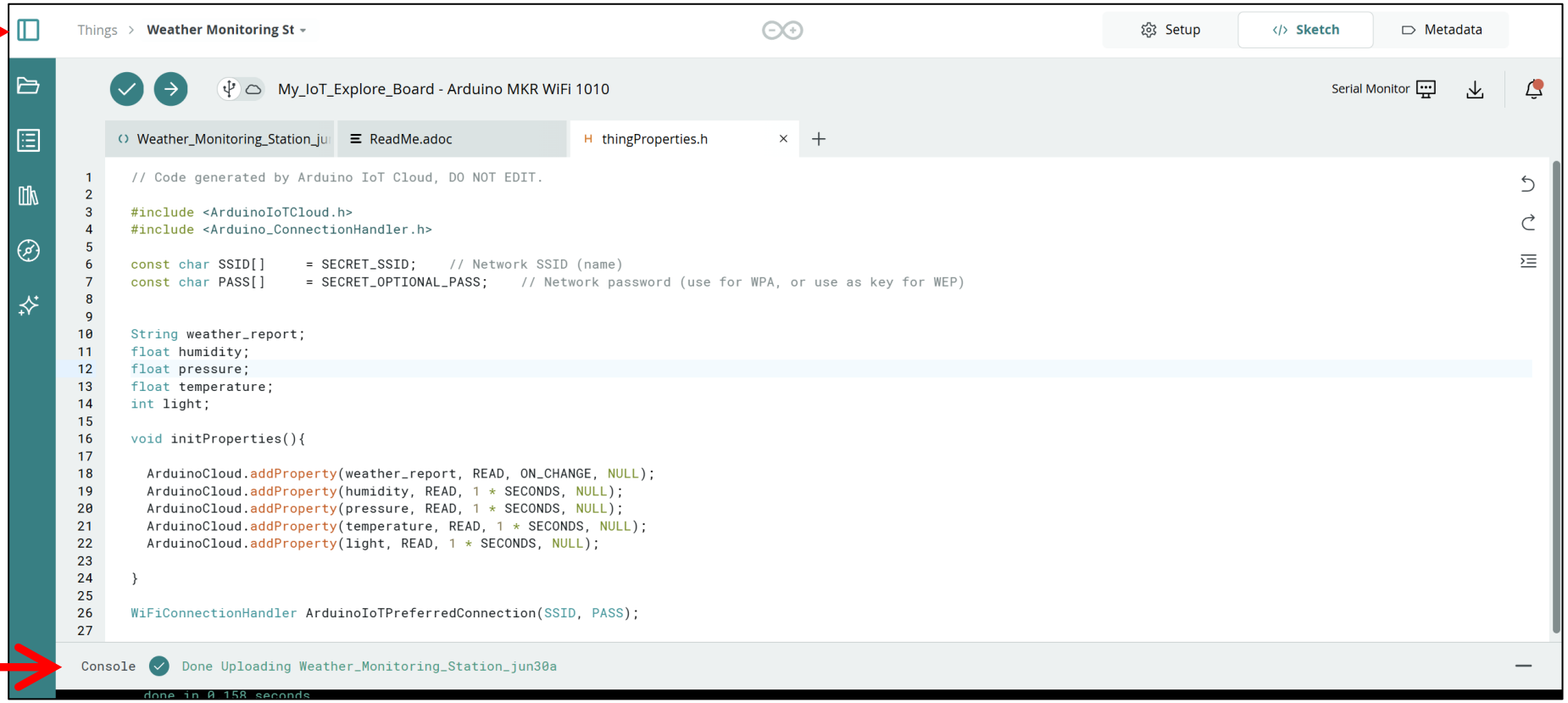
2. 計測器の動作確認



スケッチをアップロードする

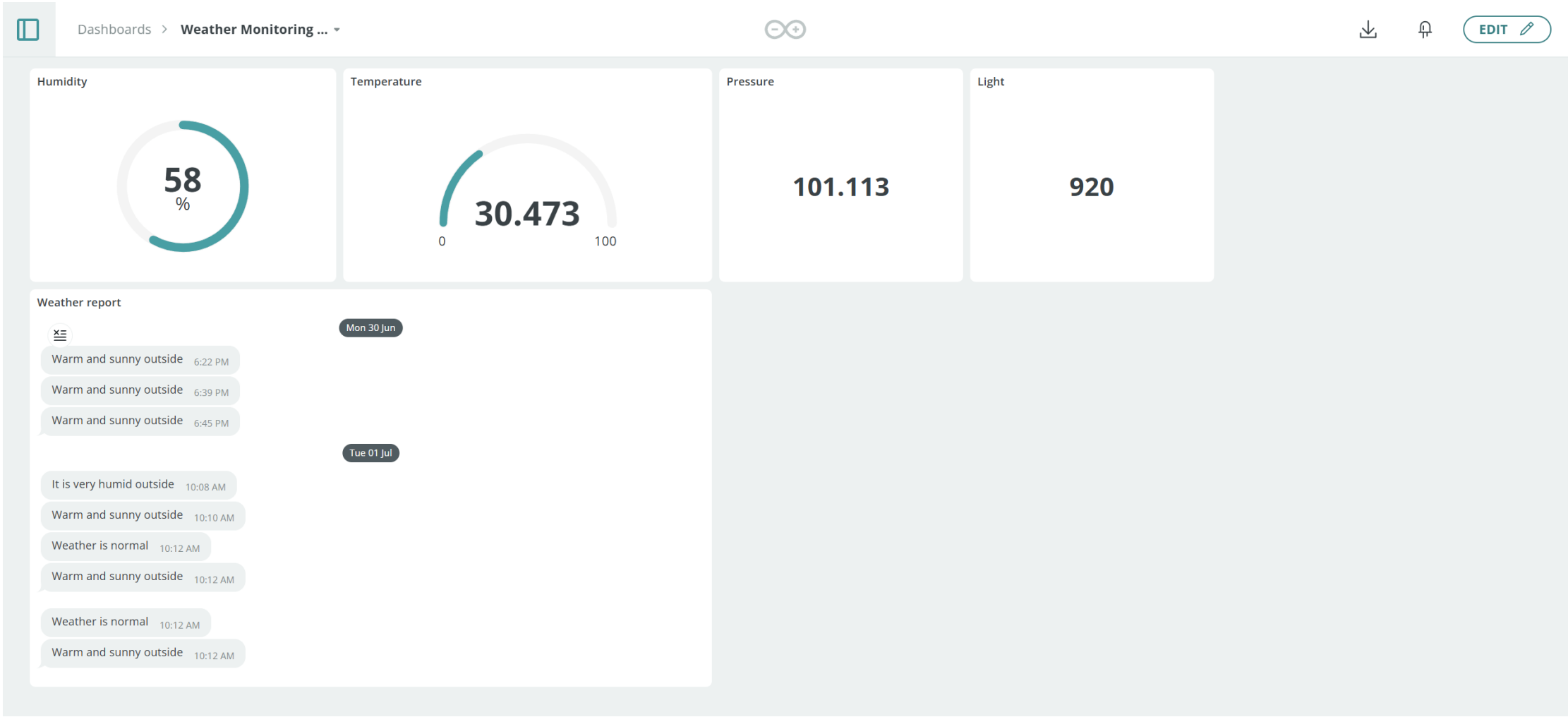
2. 計測器の動作確認

ここをクリックして
ダッシュボードへ移動

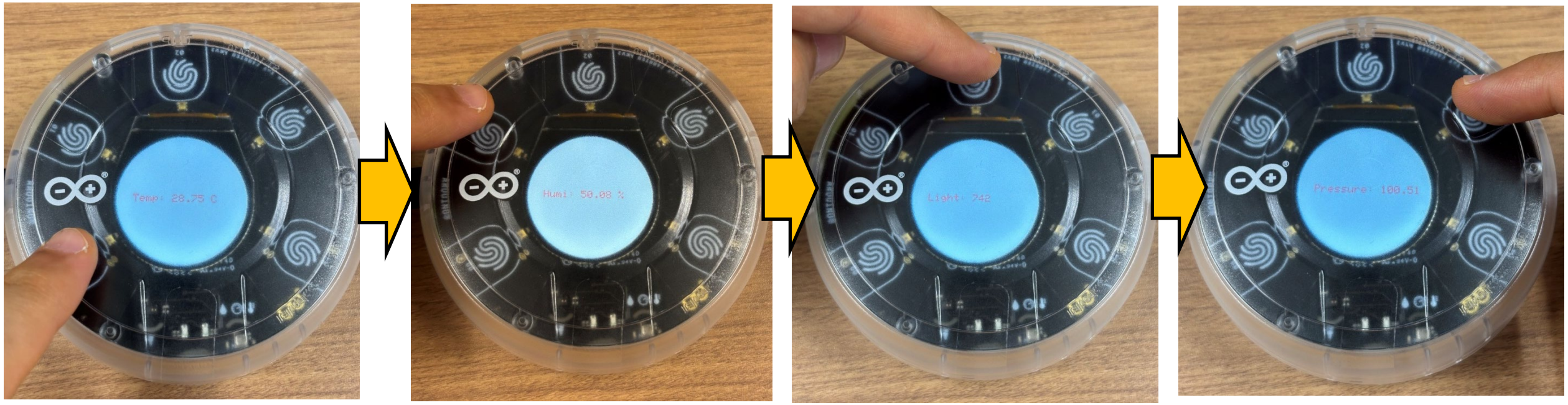


スケッチがアップロード
されたことを確認する

2. 計測器の動作確認


Dashboards > Weather Monitoring ...
EDIT
Humidity
58
%
Temperature
30.473
0
100
Pressure
101.113
Light
920
Weather report
Mon 30 Jun
Warm and sunny outside 6:22 PM
Warm and sunny outside 6:39 PM
Warm and sunny outside 6:45 PM
Tue 01 Jul
It is very humid outside 10:08 AM
Warm and sunny outside 10:10 AM
Weather is normal 10:12 AM
Warm and sunny outside 10:12 AM
Weather is normal 10:12 AM
Warm and sunny outside 10:12 AM

2. 計測器の動作確認



setup()関数において、

CARRIER_CASE = false; → CARRIER_CASE = true;

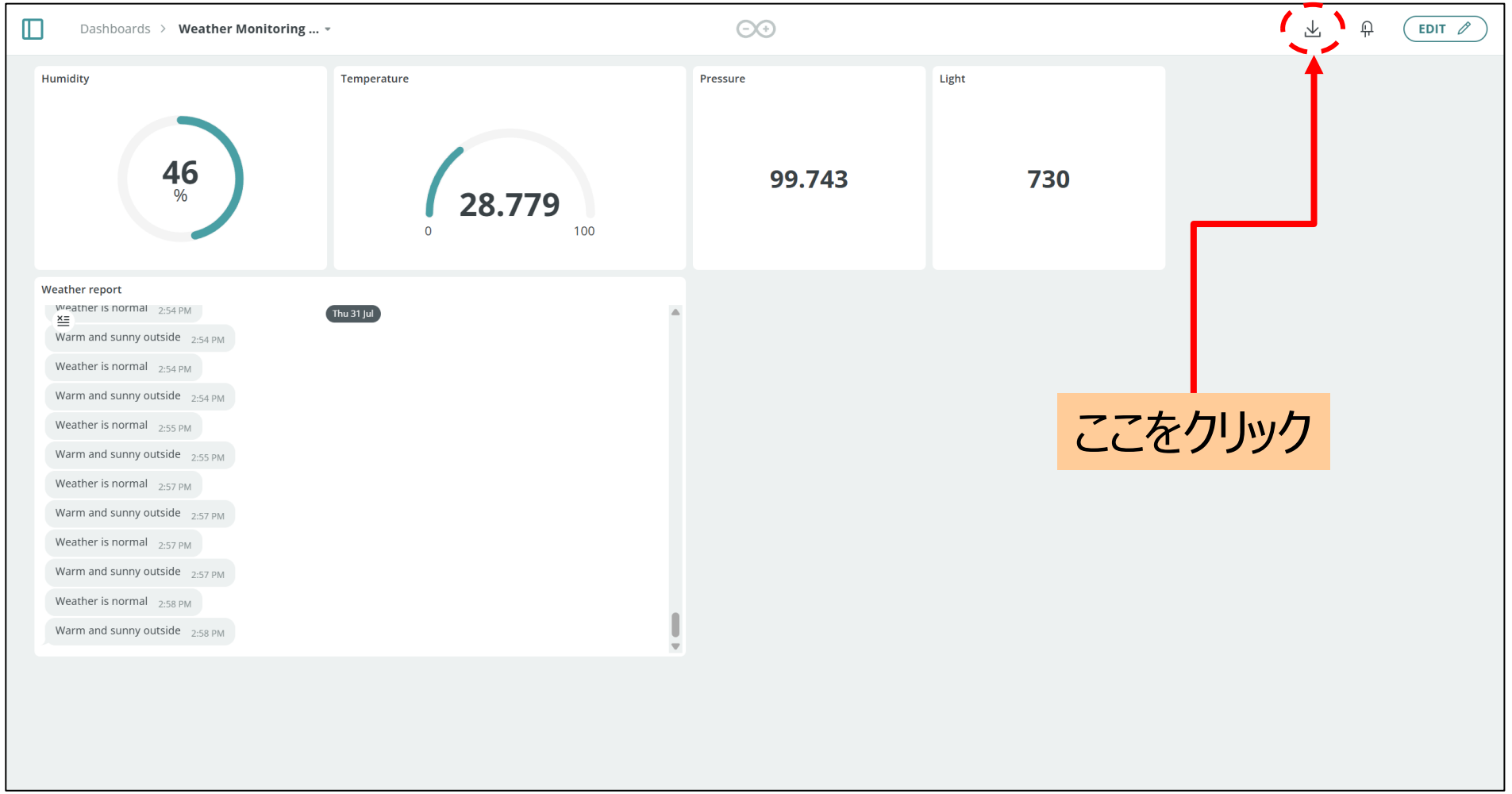
のように修正を加えることで、ボタン操作が可能になる。

本演習の計測処理の流れ

1. 計測器の設定プログラムの導入
2. 計測器の動作確認
3. 計測器によりデータを取得する
4. 計測データを解析する

3. 計測器によりデータを取得する

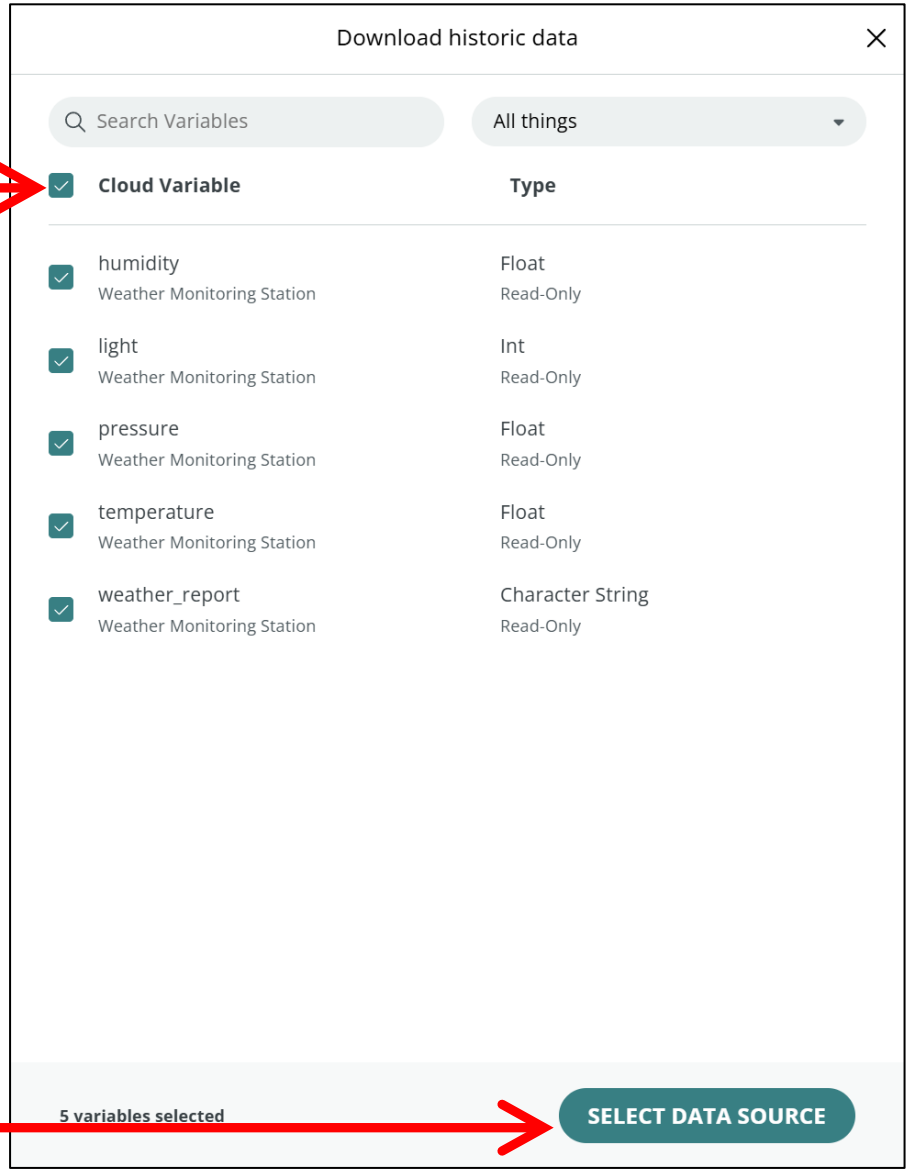
センサーによる計測データはクラウドに保存されている。データを取得してみよう。



3. 計測器によりデータを取得する

すべての変数を選択

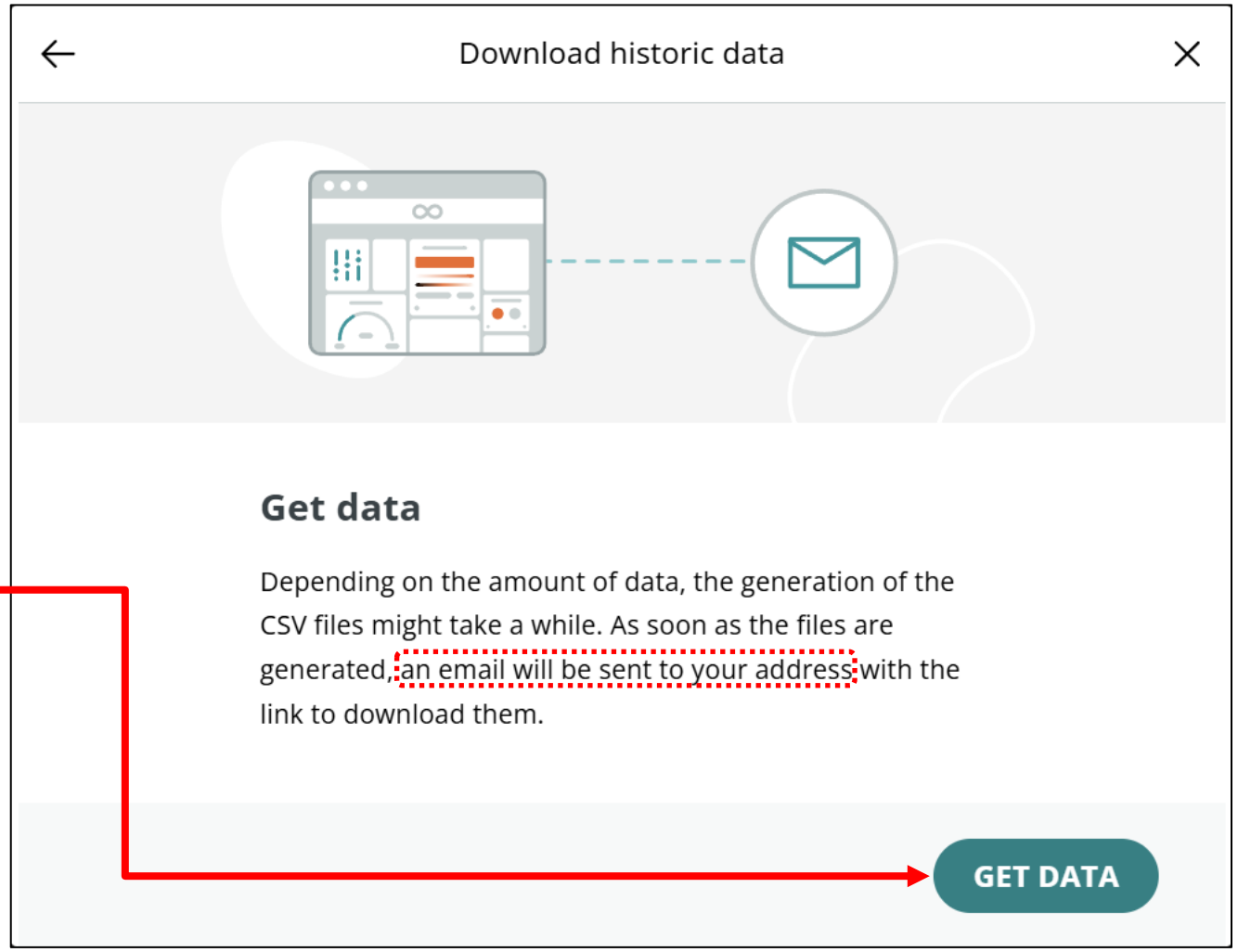
ここをクリック



3. 計測器によりデータを取得する

ここをクリック

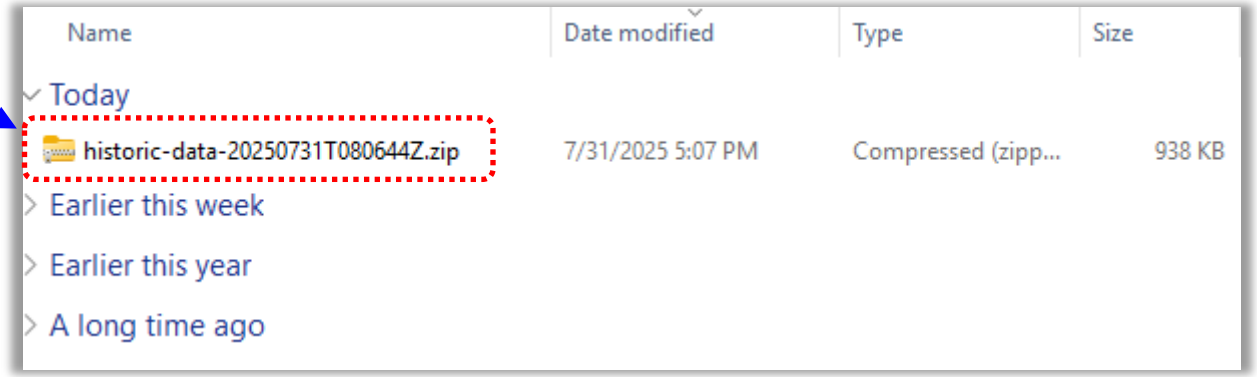
データダウンロード用のリンクがアカウントとして設定したメールアドレス宛に送付されるので確認ください。



3. 計測器によりデータを取得する

メールアドレス宛に届いたリンクをクリックすると、
右の画面が表示される

センサーの取得データがCSVファイル(Zip形式での圧縮ファイル)として保存されている。
→ このファイルがダウンロードされる。



ここをクリック

補足：Arduino IoT Cloudに保存されるデータの時刻（タイムスタンプ）はUTC（協定世界時）です。通常は英国（ロンドン）の標準時と一致しますが、夏時間では1時間進む点に注意してください。日本時間とは最大9時間の差があります。

本演習の計測処理の流れ

1. 計測器の設定プログラムの導入
2. 計測器の動作確認
3. 計測器によりデータを取得する
4. 計測データを解析する

4. 計測データを解析する

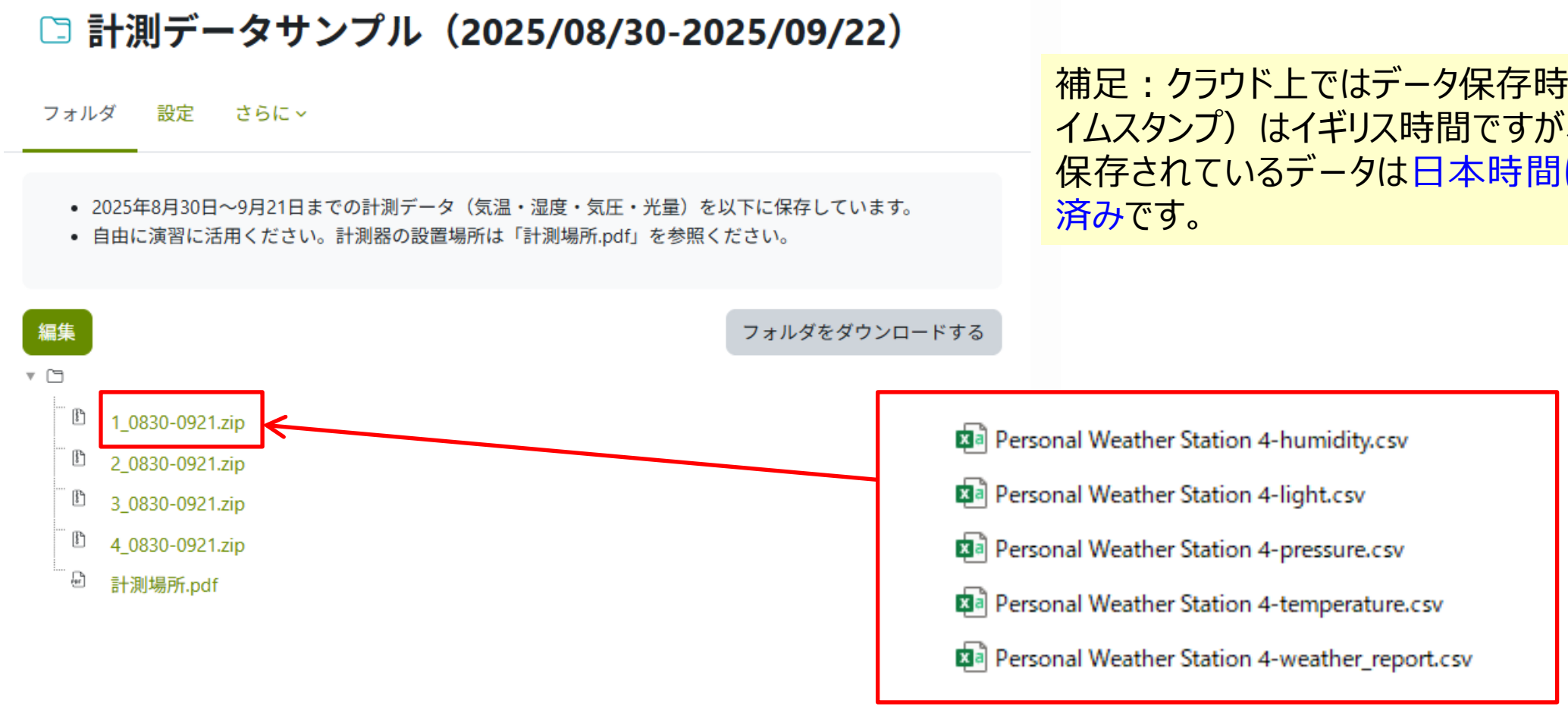
2025年8月29日18時から9月22日8時まで取得した計測データ（気温・湿度・気圧・光量）を以下に保存しています。ZIPファイルを展開すると、各項目ごとのCSVファイルを利用できます。

CSV形式のファイルがこちらに保存されています。

自由に活用ください。本節では、これらの計測結果に基礎的なデータ処理を適用することで、その傾向等を分析します。

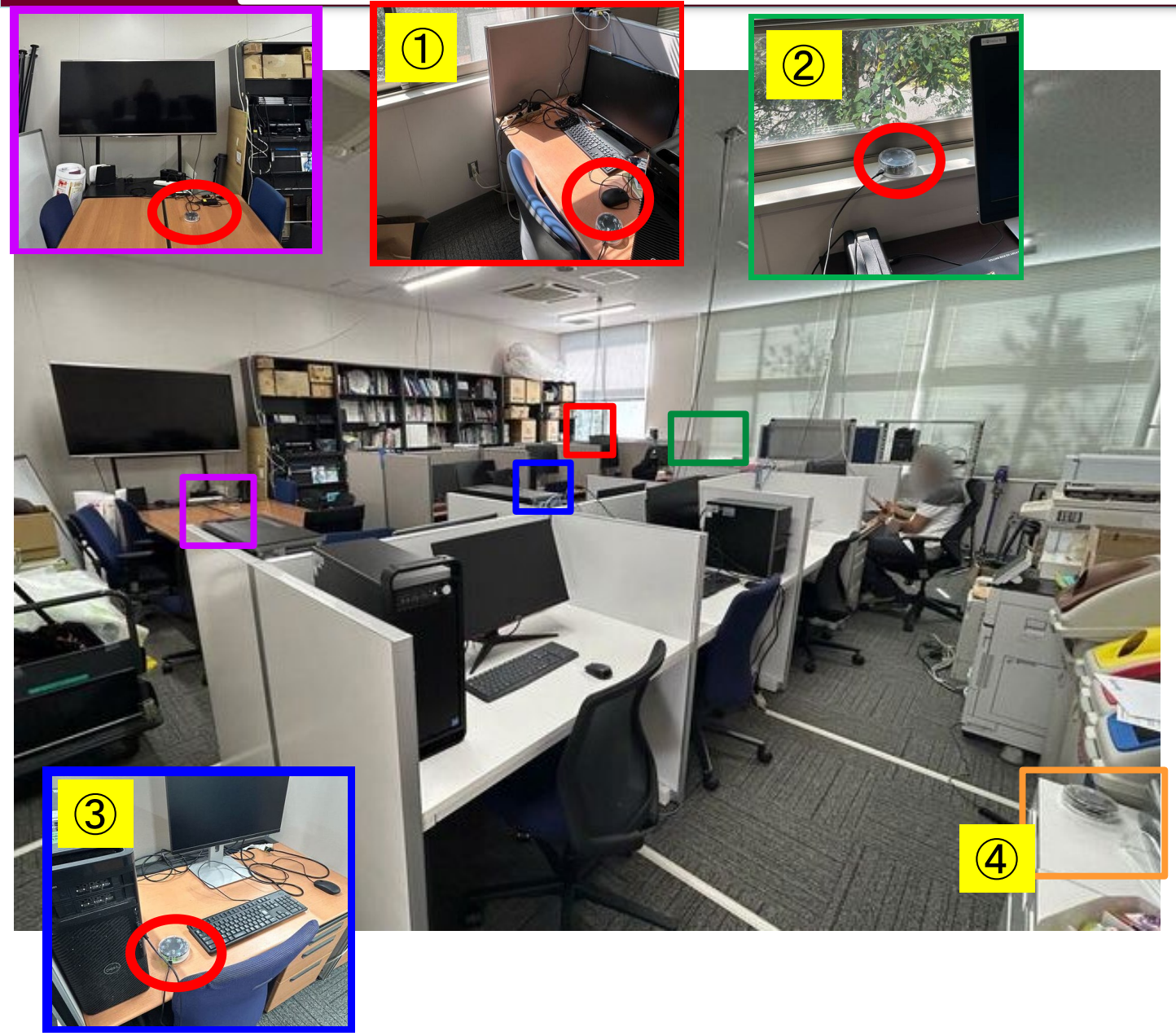
4. 計測データを解析する

2025年8月29日18時から9月22日8時まで取得した計測データ（気温・湿度・気圧・光量）を以下に保存しています。ZIPファイルを展開すると、各項目ごとのCSVファイルを利用できます。

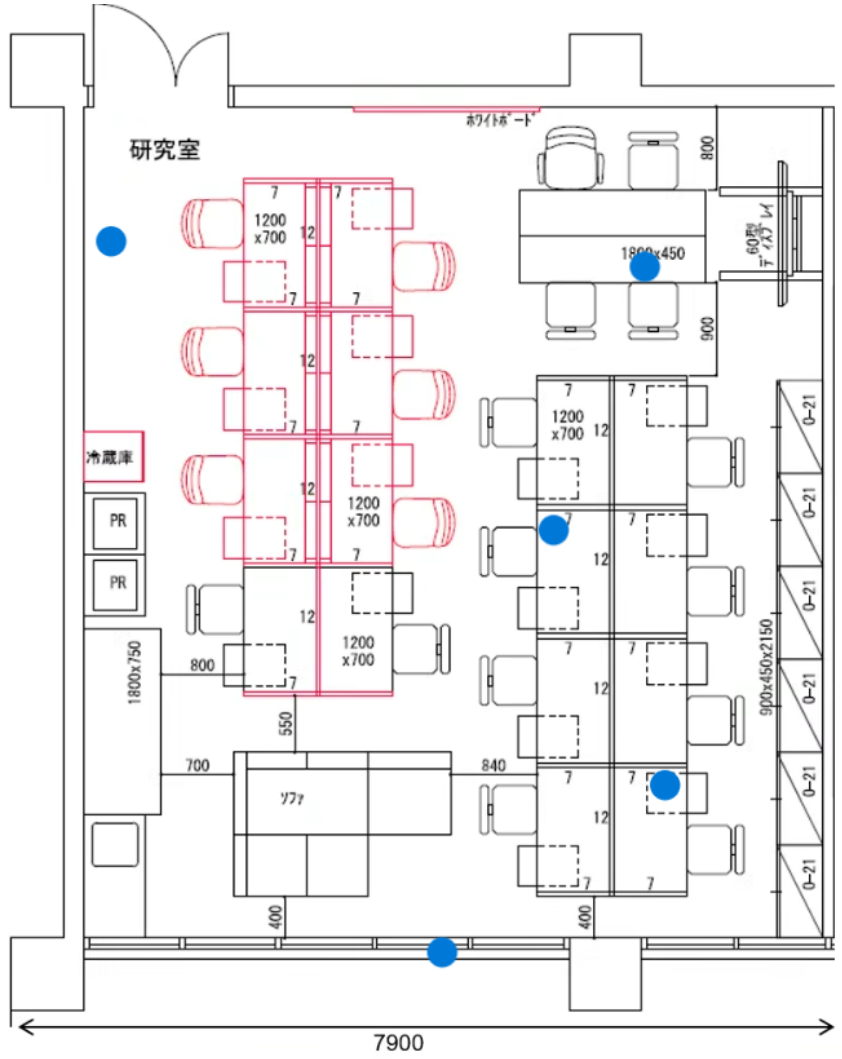


補足：クラウド上ではデータ保存時刻（タイムスタンプ）はイギリス時間ですが、ここに保存されているデータは日本時間に変換済みです。

4. 計測データを解析する

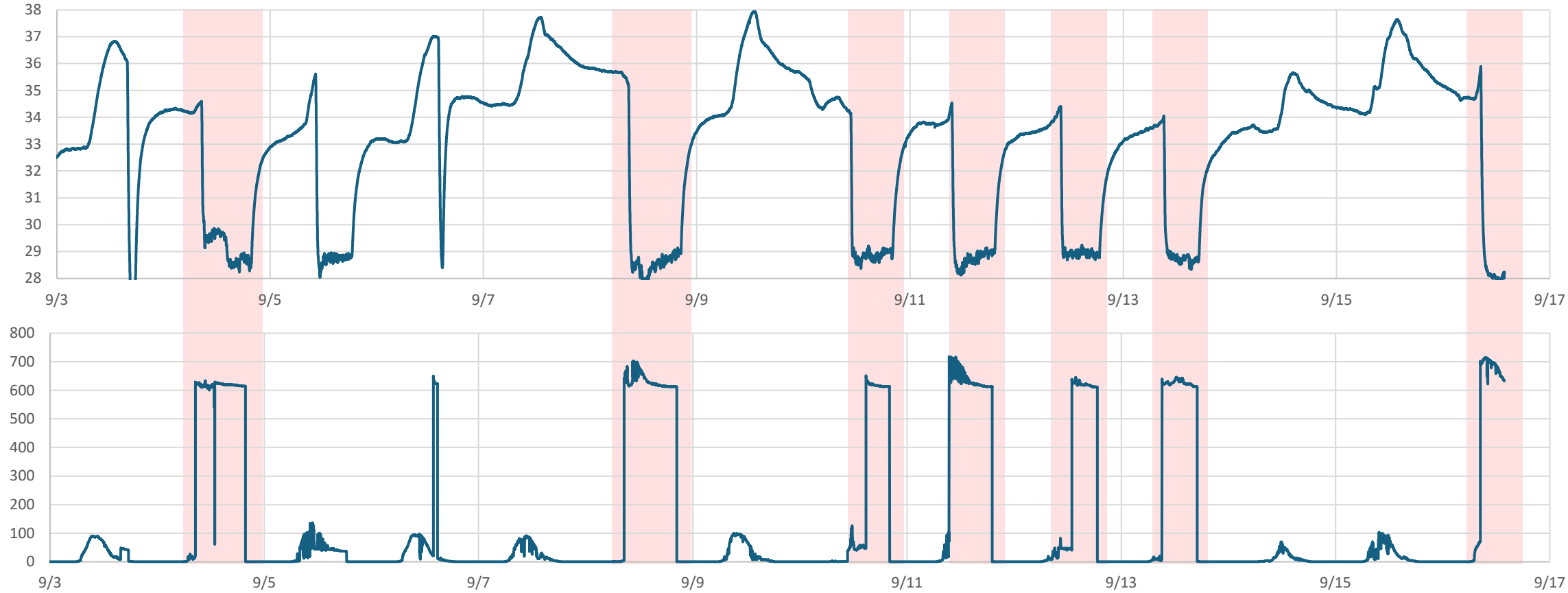


屋内の複数の箇所にセンサーを設置



4. 計測データを解析する

〇温度と光度の関係を見てみよう

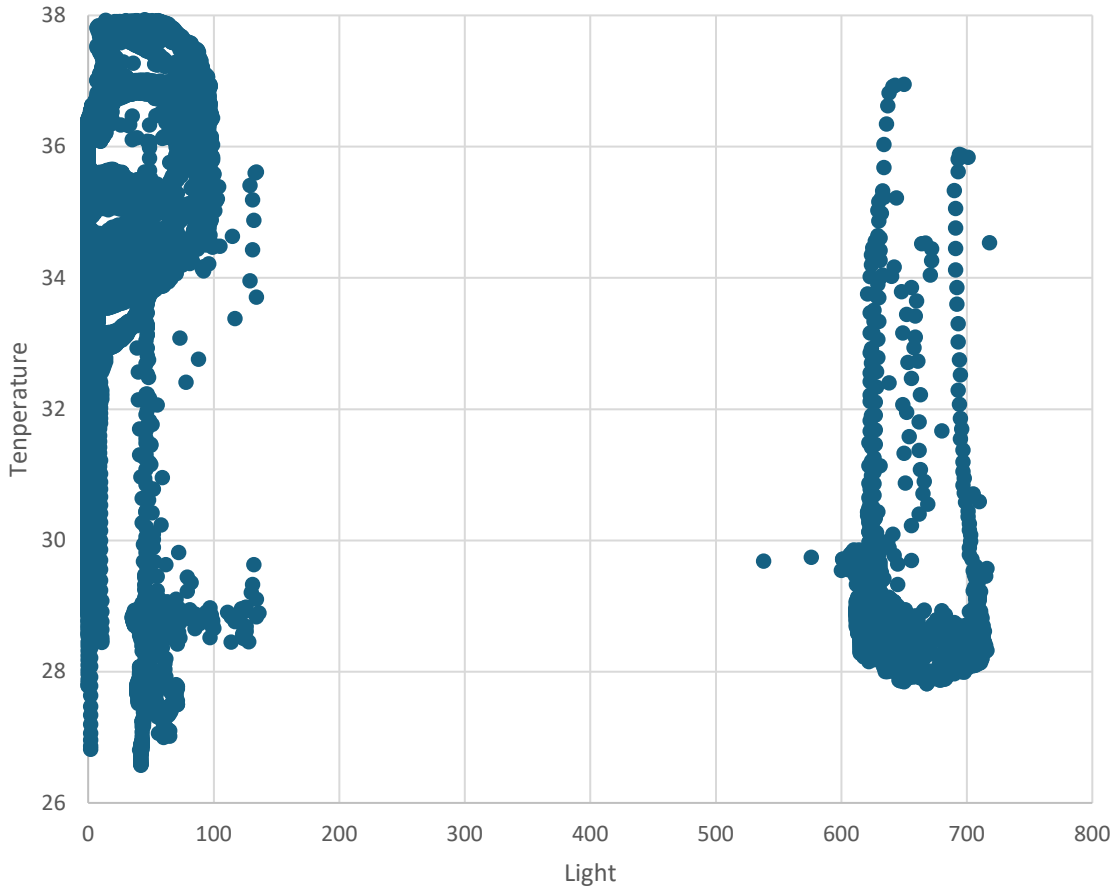


明るい時に気温が低い？⇒分析してみよう！

4. 計測データを解析する

〇温度と光度の関係を見てみよう

1. 散布図を確認



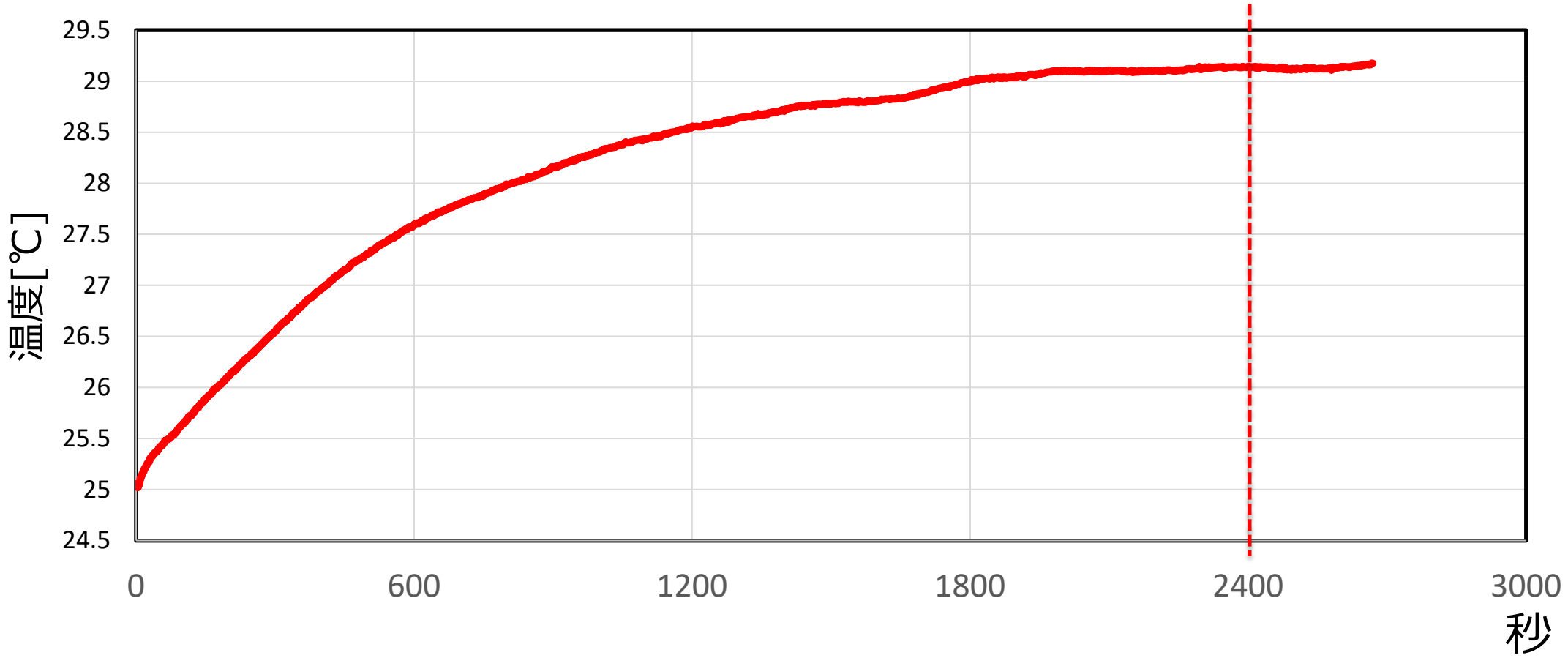
2. 光度300以上と光度300以下の平均気温を確認

明るさ >= 300　　の平均気温: 28.93
明るさ <　300　　の平均気温: 33.67

4. 計測データを解析する

エアコンの設定温度は26℃
データは29℃前後を記録

一定時間経過後、29℃付近で安定



練習課題

気温、湿度、気圧、光量の約３週間分のデータが利用可能です。

例えば、以下のような関係をみてみましょう。

- 曜日と光量/温度の関係
- 気圧、湿度について、移動平均、移動分散、ＰＭＦ，ＣＤＦ等を求めてみよう。
- 朝・昼・夜による気象データの違い

Arduino MKR IoT Carrier用の関数一覧(1/3)

本演習で使用する Arduino MKR WiFi 1010 と MKR IoT Carrier を組み合わせた環境で利用可能な専用ライブラリ（関数一覧）を以下に示す。これらを用いることで光量や気圧なども測定可能となる。

ライブラリー覧

機能	関数例	説明
温度	carrier.Env.readTemperature()	温度（℃）を取得
湿度	carrier.Env.readHumidity()	湿度（%）を取得
気圧	carrier.Pressure.readPressure()	気圧（kPa）を取得
照度	carrier.Light.readIlluminance()	明るさを取得
カラー	carrier.Light.readColor(r,g,b,c)	RGB＋クリア値を取得
カラー更新確認	carrier.Light.colorAvailable()	新しいカラー値があるか確認
ボタン更新	carrier.Buttons.update()	タッチボタンの状態を更新
タッチ判定	carrier.Buttons.onTouchDown(TOUCH0)	タッチ0が押されたかを判定
画面塗りつぶし	carrier.display.fillScreen(color)	画面を指定色で塗りつぶす
文字色設定	carrier.display.setTextColor(color)	表示する文字の色を設定
文字サイズ	carrier.display.setTextSize(size)	文字の大きさを設定
カーソル位置	carrier.display.setCursor(x, y)	テキスト表示の開始位置を指定
文字表示	carrier.display.print("text")	文字を画面に表示
画面回転	carrier.display.setRotation(0〜3)	ディスプレイの表示方向を変更
初期化	carrier.begin()	Carrierボード全体を初期化
ケース設定	CARRIER_CASE = true/false;	ケース付き/なしを指定

Arduino MKR IoT Carrier用の関数一覧(2/3)

LEDの利用方法

LEDを光らせたいとき

- 光らせるLEDと色を設定する（まだ光らない）

```
carrier.leds.setPixelColor(index, red, green, blue);
```

index: LED番号（0-4）、red, green, blue: RGB形式（0-255）
（例）LED0を赤く光らせる：carrier.leds.setPixelColor(0, 255, 0, 0);

・明るさを設定する（まだ光らない）

```
carrier.leds.setBrightness(255);    // 0–255
```

・設定を反映（ここで点灯）

```
carrier.leds.show();
```

・設定をクリアする（消灯設定にする。LEDはついたまま）※反映には show() が必要

```
carrier.leds.clear();
```

LED1~4を緑に光らせた場合

LEDは設定した後show()で設定をLEDに反映させる必要があることに注意

例：１秒毎に点灯と消灯を繰り返す

```
void loop() {
  delay(1000);  carrier.leds.clear(); carrier.leds.show();
  delay(1000); carrier.leds.setPixelColor(0, 255, 0, 0);
  carrier.leds.show(); //show()を呼ばないと反映されない
}
```

Arduino MKR IoT Carrier用の関数一覧(3/3)

ブザーの利用方法

ブザーを鳴らしたいとき

・指定した周波数の音を鳴らす

```
carrier.Buzzer.sound(freq);
```

・音を止める

```
carrier.Buzzer.noSound();
```

・ブザーとして短く鳴らす

```
carrier.Buzzer.beep();
```

・周波数と時間を指定して鳴らす

```
carrier.Buzzer.beep(800, 20);
//周波数：800Hz、時間：20ミリ秒
```

自由課題

自由課題の例を示します。

- デバイスのボタンを押すと、表示パネルに文字（"Hello" など）が表示される。
- ボタンごとに表示される文字を切り替える。
- 表示パネルに文字を表示し、ダッシュボード上のボタン操作で表示をクリアできるようにする。
- ダッシュボード上のボタン操作やデバイスのボタン操作によって、表示内容を変化させる。
- ボタン操作でLEDを点灯・消灯、あるいはブザーを鳴らす

※本演習のデバイス（MKR IoT Carrier）に搭載されているLEDや近接センサーの詳細は、以下を参照ください：
https://docs.arduino.cc/tutorials/mkr-iot-carrier-rev2/cheat-sheet/

LEDとブザーを用いた自由課題例

- タッチセンサー入力に応じてLEDやブザーを制御する自由課題の例です
- サンプルコードがMoodleに保存されていますので実際に試してみよう

  ファイル名: LED_and_Buzzer.ino

  自分なりに動作をアレンジしてみましょう。

- LEDとブザーの使い方は本資料末尾の関数一覧に記載

```
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;
void setup() {
    // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  delay(500);
  carrier.withCase();
  carrier.begin();
  carrier.display.setRotation(0);
  delay(1500);
}
```

```
void loop() {
  carrier.Buttons.update();
  carrier.leds.setBrightness(10);

  if(carrier.Buttons.onTouchDown(TOUCH0)){
    carrier.Buzzer.sound(262);    delay(250);
    carrier.Buzzer.sound(294);    delay(250);
    carrier.Buzzer.sound(330);    delay(250);
    carrier.Buzzer.sound(349);    delay(250);
    carrier.Buzzer.sound(392);    delay(250);
    carrier.Buzzer.sound(440);    delay(250);
    carrier.Buzzer.sound(494);    delay(250);
    carrier.Buzzer.sound(523);    delay(250);
    carrier.Buzzer.noSound();
  }
  if(carrier.Buttons.onTouchDown(TOUCH1)){
    carrier.Buzzer.beep();
  }
  if(carrier.Buttons.onTouchDown(TOUCH2)){
    carrier.Buzzer.beep(440,200);
  }
  if(carrier.Buttons.onTouchDown(TOUCH3)){
    carrier.leds.setPixelColor(0,255,255,255);carrier.leds.show();    delay(250);
    carrier.leds.setPixelColor(1,255,255,255);carrier.leds.show();    delay(250);
    carrier.leds.setPixelColor(2,255,255,255);carrier.leds.show();    delay(250);
    carrier.leds.setPixelColor(3,255,255,255);carrier.leds.show();    delay(250);
    carrier.leds.setPixelColor(4,255,255,255);carrier.leds.show();    delay(250);
     carrier.leds.clear();carrier.leds.show();
  }
}
```

まとめ

本演習では、気象情報の収集システムを構築する例として、計測デバイス（Arduino MKR WiFi 1010）とArduino Cloudを活用したプログラミング方法について学習した。

- センサーデータ（温度、湿度、気圧、明るさ）を取得し、クラウドに送信・可視化する一連の処理の実装に演習形式で取り組んだ。
- 収集したデータに対して、移動平均・移動分散・PMF（確率質量関数）、CDF（累積分布関数）といった基本的な統計処理を適用し、気象データの傾向や特性の基礎的な分析に取り組んだ。

Arduino　共通関数一覧

基本関数

機能	関数例	説明
初期化	setup()	最初に1回だけ実行される処理を記述
ループ	loop()	繰り返し実行される処理を記述
遅延	delay(ms)	指定ミリ秒だけ処理を止める
時間計測	millis()	起動からの経過時間（ms）を返す
時間計測	micros()	起動からの経過時間（μs）を返す

入出力関数

機能	関数例	説明
ピン設定	pinMode(pin, mode)	デジタルピンを入力/出力に設定
デジタル出力	digitalWrite(pin, value)	HIGH/LOW を出力
デジタル入力	digitalRead(pin)	ピンの状態を読み取る
アナログ出力	analogWrite(pin, value)	PWM信号を出力
アナログ入力	analogRead(pin)	アナログ値を読み取る

Arduino　共通関数一覧

数学関数

機能	関数例	説明
最小値	min(x, y)	小さい方を返す
最大値	max(x, y)	大きい方を返す
絶対値	abs(x)	絶対値を返す
制限	constrain(x,a,b)	範囲 [a,b] に収める
写像	map(x,a,b,c,d)	区間[a,b]を[c,d]に写像
累乗	pow(x,y)	x^y を返す
平方根	sqrt(x)	平方根を返す
三角関数	sin(x), cos(x), tan(x)	角度の三角関数

乱数・ビット演算

機能	関数例	説明
乱数シード	randomSeed(val)	乱数生成の初期化
乱数生成	random(min,max)	指定範囲の乱数を返す
ビット読み	bitRead(x,n)	n番目のビットを読む
ビット書き	bitWrite(x,n,b)	n番目のビットをbにする
ビット操作	bitSet(x,n), bitClear(x,n)	ビットを1/0に設定
ビット生成	bit(n)	2^n を返す

Arduino　共通関数一覧

シリアル通信

機能	関数例	説明
通信開始	Serial.begin(9600)	シリアル通信を開始する
文字送信	Serial.print()	データを送信
文字送信	Serial.println()	改行付きで送信
文字受信	Serial.read()	受信データを1文字取得
受信確認	Serial.available()	受信データがあるか確認

割り込み・その他

機能	関数例	説明
割り込み登録	attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)	指定ピンに割り込みを登録
割り込み解除	detachInterrupt(pin)	割り込みを解除
シフト入力	shiftIn(dataPin, clockPin, bitOrder)	シリアルデータを読み込み
シフト出力	shiftOut(dataPin, clockPin, bitOrder, value)	シリアルデータを出力
パルス測定	pulseIn(pin, value)	パルスの長さを測定