

# ラズパイとDashで環境ダッシュボードを作ろう

PyCon JP 2021

2021/10/15 ~ 10/16

# お前誰よ

- 佐野浩士 (Hiroshi Sano) [@hrs\\_sano645](#)
  - : 静岡県の富士市 
- Job 
  - 株式会社佐野設計事務所
  - 米農家 
- Community 
  - Python駿河, PyCon mini Shizuokaスタッフ
  - Code for ふじのくに

## 目次

- 環境ダッシュボードを作った話
- PythonとIoT
  - CircuitPythonとラズパイを使う選択
- Plotly Dashでダッシュボードアプリを作る
  - Dashの特徴を紹介
  - センサー情報を可視化する
- まとめ

本日のテーマ

**ラズパイとDashで環境ダッシュボードを作ろう**

- ラズパイ  
=> **Raspberry Pi**
- Dash  
=> **Plotly Dash**
- 環境ダッシュボード  
=> 環境センサー（今回は温度湿度気圧）を記録とグラフを表示する  
**ダッシュボードアプリ**

## このトークの趣旨、モチベーション

私的プロジェクトを紹介しつつ、Pythonを使ったIoTとデータ可視化を  
デモを交えてお伝えします

- PythonとIoT
- Plotly Dashでダッシュボードを作る

本日お伝えしたいこと



- 世の中にはないけど、ほしいなら自分で作ろう
- 世の中に存在しないデータを集めて見てみよう
  - 身近だけど見えないデータ
- 積みボードを活用していこう



## homeenvdashプロジェクトの紹介

環境センサーの情報が見れるダッシュボードアプリを作りました



このプロジェクトのモチベーションを紹介します。



## 家環境ダッシュボード

場所:

気温: 34.2°C

湿度: 42.4%

気圧: 1023.1hPa

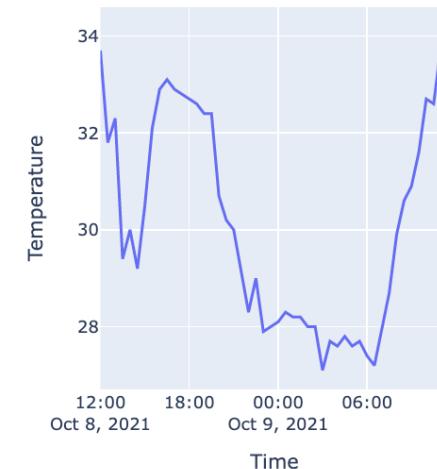
更新時間 :2021-10-09 11:30:04

### 環境グラフ

1日

温度 湿度 気圧

温度



- ラズパイ+環境センサーを接続してセンサー  
ノード
- ダッシュボードアプリはDashを利用する
- さまざま/複数の環境センサーと接続して  
1つのダッシュボードでモニターできる
  - 複数の部屋や種類が異なるセンサーを扱  
える

現在は、温度湿度気圧のBME280に対応。今後はCO2や非接触温度センサなどに対応したい

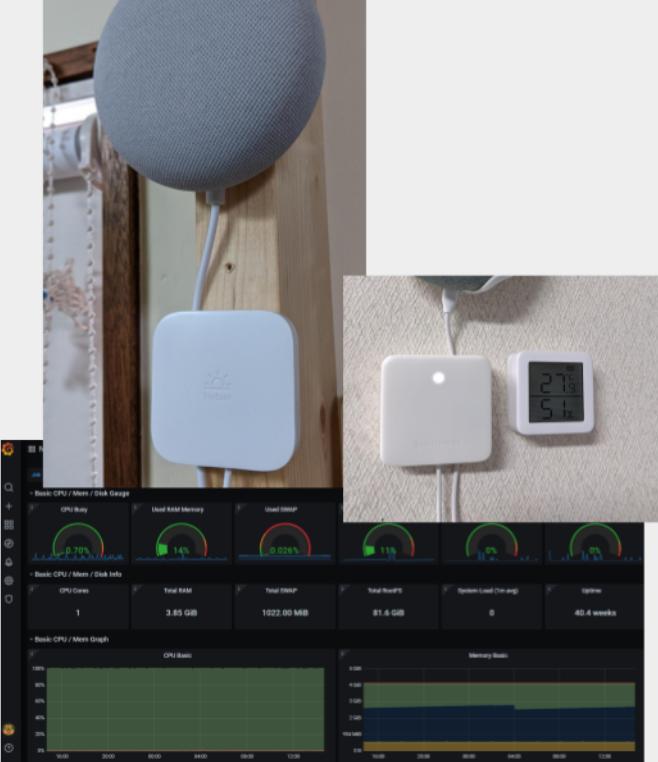


## なぜ作ったのか？

理由は3つほどあります



そもそも、センサーやダッシュボードはすでに製品サービスが多数ある



- 環境センサーは市販にも販売されている: 高価な製品は精度がいい
- ダッシュボードもさまざまなプロジェクトがある
  - OSSなサーバーアプリ
  - サービスとして提供されているもの:  
[Anbient](#), [Machinist](#), [MotionBoard](#)

なんで作る必要があるか?

車輪の再開発になるけどどうなのか?



## 低気圧の体調不良に対応したかった

- 低気圧に弱いので不調の前兆を調べたい
- 予報サービスはあるけど現在状況を見たい
- 世の中にはIoTの製品はあるものの  
**「気圧」を見られるものが手に入りにくい**
  - 高価な製品が多い

オムロンの環境センサーは高価ですが扱いやすそうです: <https://www.omron.co.jp/ecb/product-detail?partNumber=2JCIE-BL>



## 見守りに使う

- 実家の祖母の部屋の状況を見れるよう  
気をつけるために利用していました
- 高齢者は気温に対して間違えやすい
  - 体感より数値化された状態を見た方が対応しやすい
- センシティブな環境なので外部の  
ダッシュボードサービスを使いたくない





## 欲しいものを自分で作る

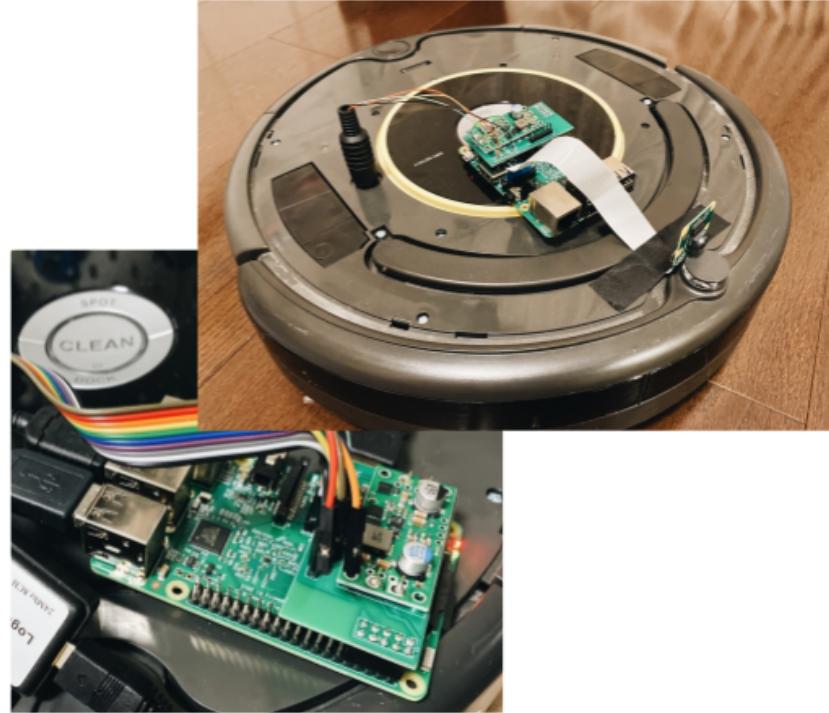
- Makerという文化
- ものづくりが好きである
- 世の中に存在していなければ  
自分で作る精神！

車輪の再開発上等だ！という文化です

世の中に存在していなければ自分で作る！

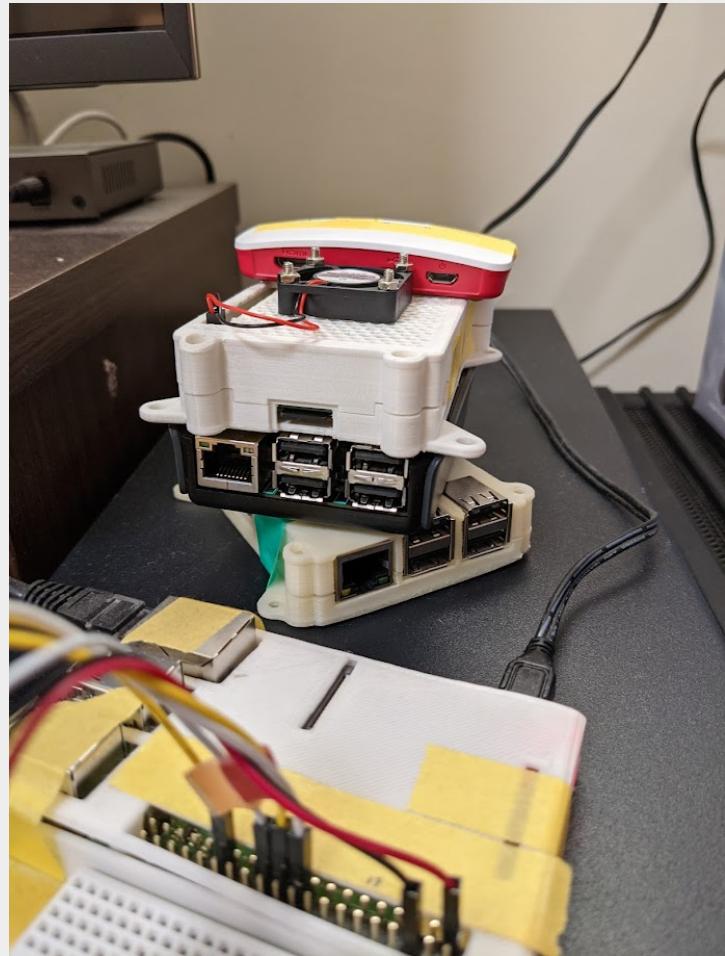


小池さん: きゅうり仕分け機



からあげさん: AIルンバ

個人的に解決したい問題として



ラズパイが大量に転がっているので有効活用することが目的

実際のところみなさんもありますよね？積みボード



からあげ  
@karaage0703

マイコンはとりあえず全部買って積みましょう

午後1:00 · 2021年7月9日 · Twitter for iPhone

---

20 件のリツイート 2 件の引用ツイート 107 件のいいね

<https://twitter.com/karaage0703/status/1413347181705105410?s=20>

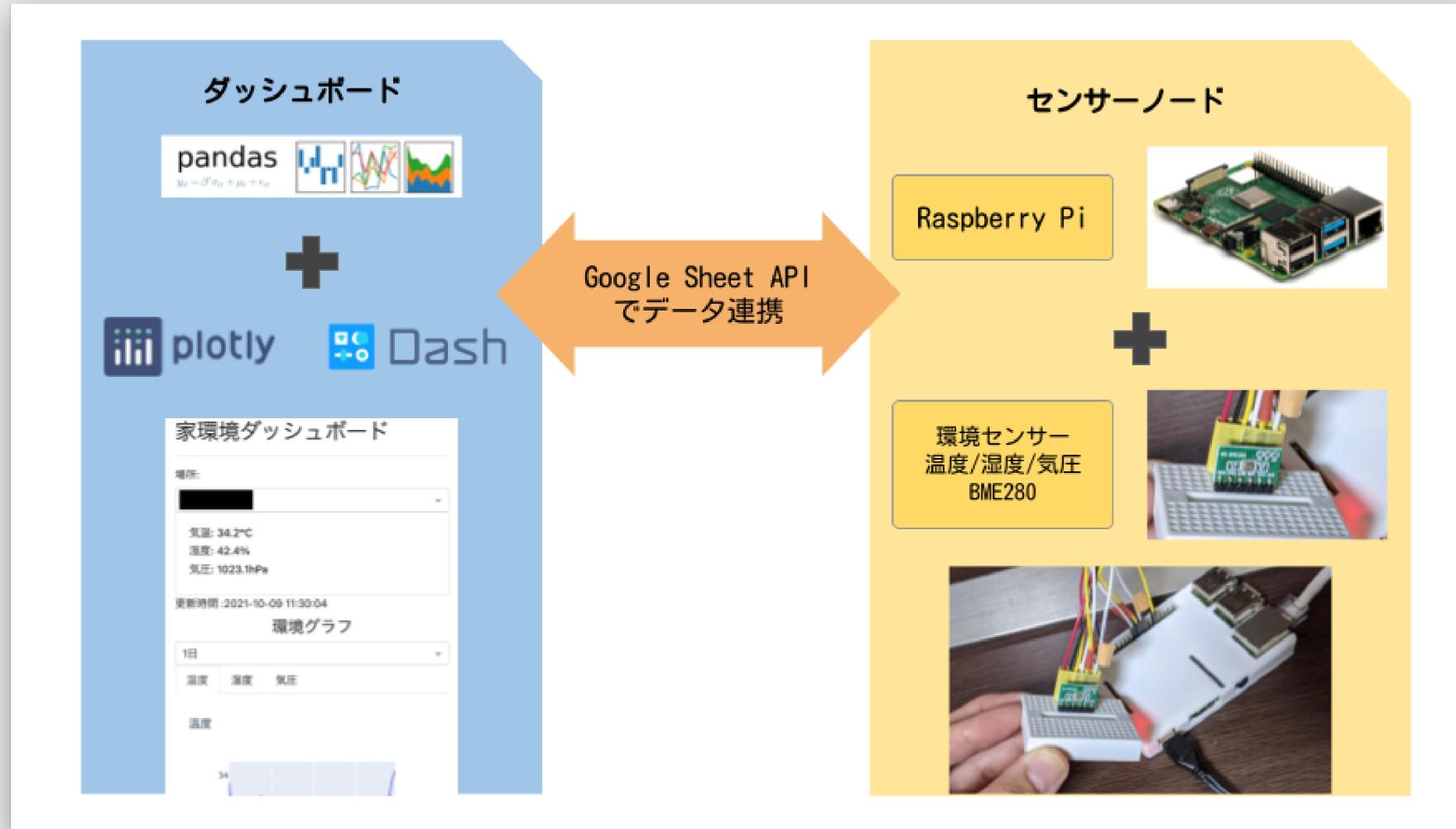
やっぱり闇のエンジニアはちげーわ！

積みボードがある方は贅沢に使って快適な日常を手に入れる！



ラズパイは一応2台ぐらい有効活用しています。踏み台サーバーとか実験用とか...

## homeenvdashの全体構成



- センサーノードはラズパイ + センサーを接続
- センサーノードは複数対応
- ダッシュボードはDash + Plotly
- センサーで取得した情報はGoogleスプレッドシートで保存
  - 今後はローカルなDBに保持して、エクスポートする形が望ましい

今日は `homeenvdash-mini` というデモ用のアプリを使って解説します。

`honeenvdash-mini` はこちら ->

<https://github.com/hrsono645/homeenvdash-mini>

- ラズパイ1つで動作しています
- センサーの値を取得して現在情報とグラフ表示はほぼ同じ
- センサーの値はCSVファイルへ保存される
- (デモの都合上) ダッシュボード起動時にしかセンサー値の記録はされません

# PythonとIoT

より手軽にPythonとIoTを行う方法を紹介します

## PythonでIoTを行う選択肢

- Raspberry Pi + CPython
- MicroPython / CircuitPython
- Raspberry Pi + CircuitPython

上記は私が知っている限りで一例です。ほかにも選択したがある場合は教えていただけると嬉しいです

- Raspberry Pi + CPython
- MicroPython / CircuitPython
- Raspberry Pi + CircuitPython

## Raspberry Piとは

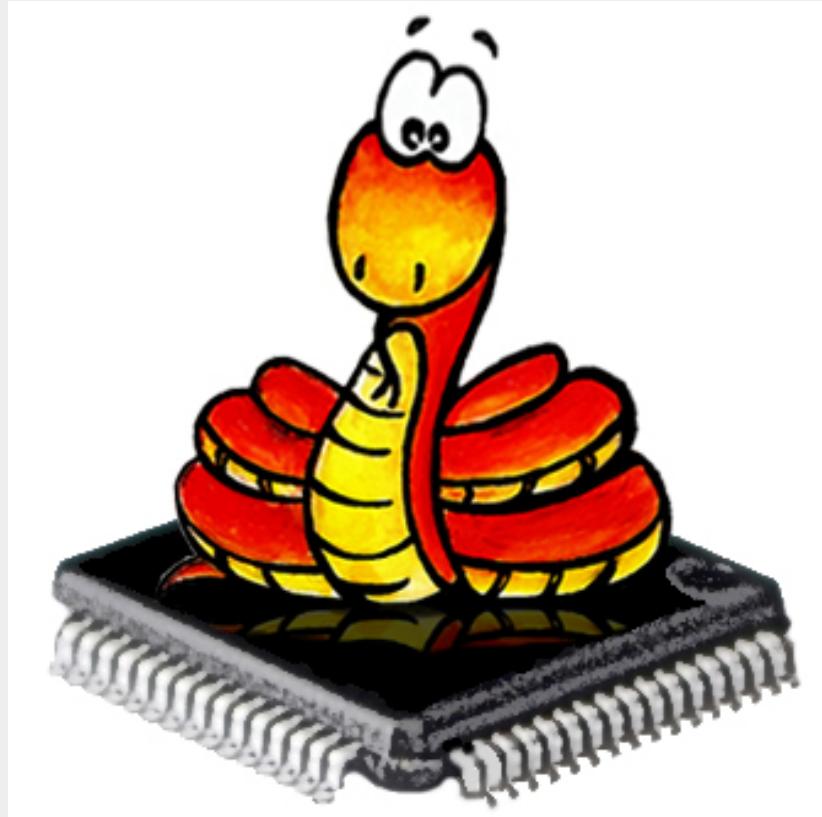
- もともとは教育目的のLinuxが動作するシングルボードコンピューター (SBC)
  - 工場自動化やサイネージ、センサーノードとして業務利用も
  - クラスター構成を作ってクラウドっぽく（おうちクラウドと呼ばれている）
- インターフェイスが豊富
  - WLAN, Ethernet, USB, Bluetooth, HDMI出力
- GUI/CUIで利用可能
  - 最新版は高性能なのでデスクトップ端末としても
  - ヘッドレスなサーバーとしても扱える

# Raspberry PiとIoT

特徴は

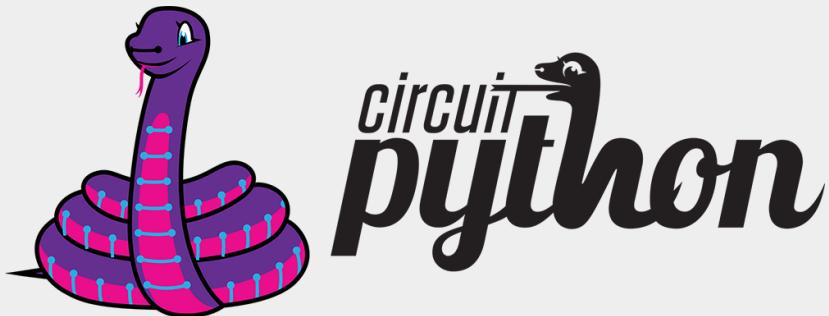
- Linuxが動きCPythonを扱える
  - GPIO（デジタル）でセンサーと接続可能
  - シリアル通信規格対応: SPI, I2C
  - ディスプレイを繋ぐとサイネージ的なデバイスも作れる
- 安価ながら高機能なIoT端末として扱える
- ▲ ACアダプタなど給電環境が必要 電源がない環境では扱いづらい

- Raspberry Pi + CPython
- MicroPython / CircuitPython
- Raspberry Pi + CircuitPython



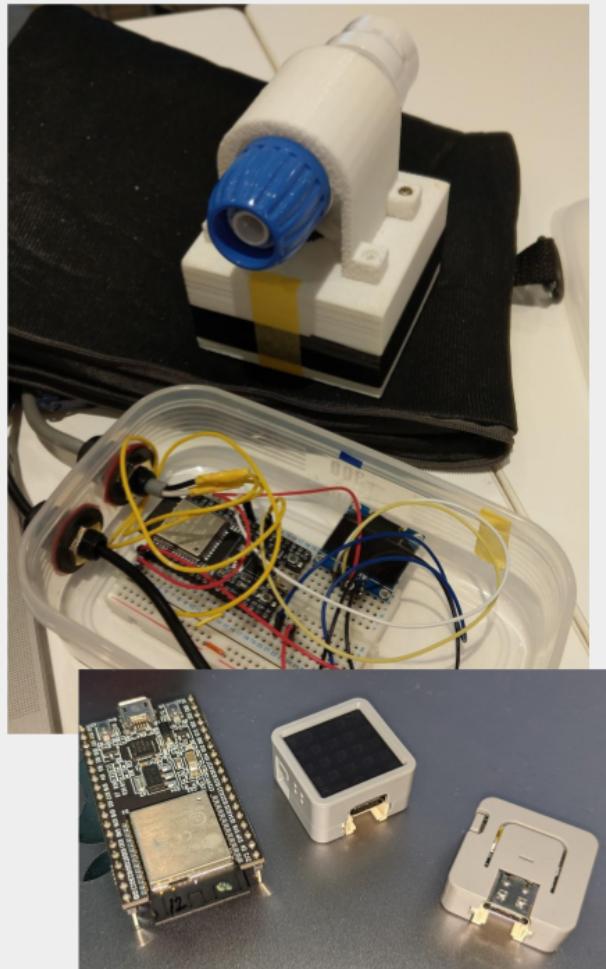
## MicroPythonとは

- MicroPythonはマイコンボード向けの処理系
  - マイコン=マイクロコントローラ
  - CPUより用途が限られている
  - 特定の機器制御に最適化された集積回路
- クラウドファンディングで生まれたpyboardの開発環境として作られた



## CircuitPythonとは

- MicroPythonの派生版
  - adafruitというSTEAM系に取り組んでる電子部品の販売や教育分野のメーカーが作成
- adafruitのボードに対応したり、専用のライブラリを用意
  - メーカーのセンサーデバイスと接続しやすい



## MicroPython/CircuitPythonの特徴

- CPythonの3系（3.4, 3.5の一部）の文法をベースにした独自の実装系
  - マイクロコントローラー向けカスタムした標準ライブラリやサードパーティライブラリがある
  - シリアルコンソールからREPLが動く
  - 他のボードにもポートされてインストール可能
- Raspberry Piより安価。電源はバッテリーも  
▲ CPythonライクだが完全互換ではない

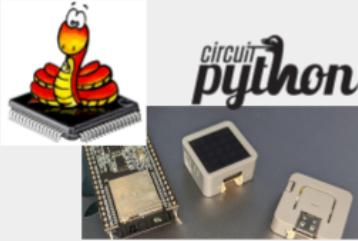
写真はESP32+MicroPythonで水栓を開け閉めするものを作ってました

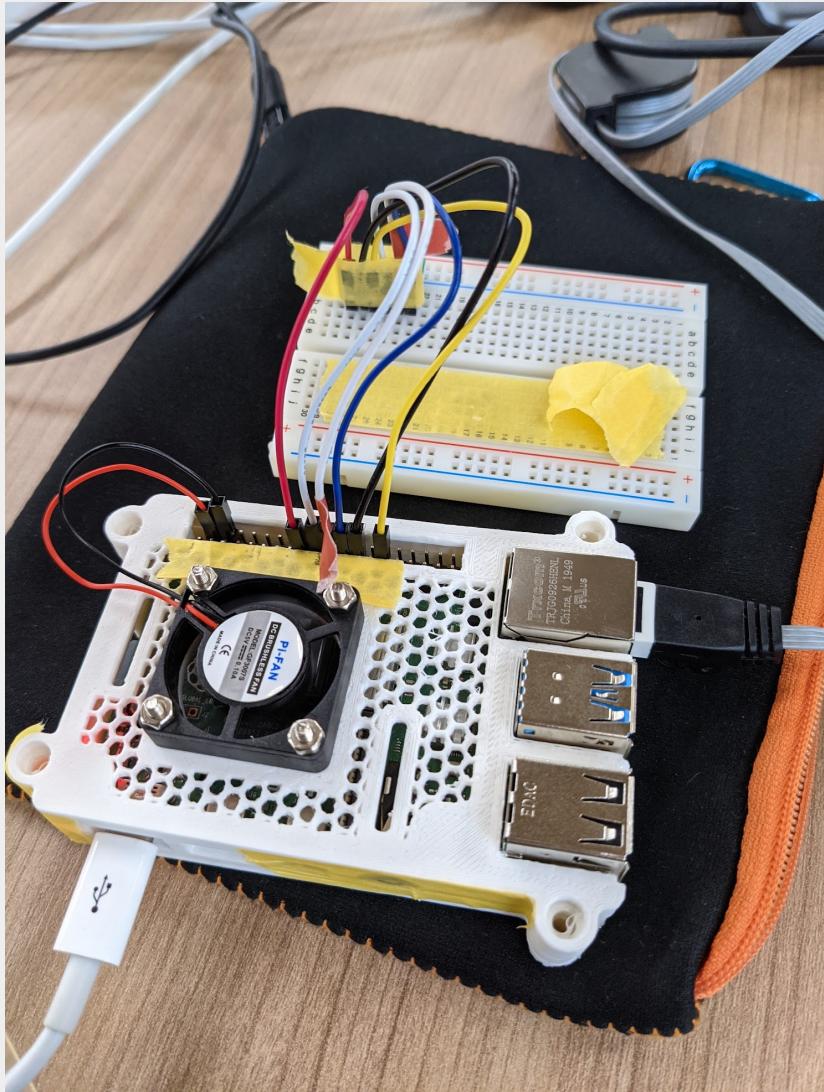
- Raspberry Pi + CPython
- MicroPython / CircuitPython
- *\*Raspberry Pi + CircuitPython)*

## CircuitPythonのライブラリをRaspberry Piで扱う

- blinkaというライブラリを使う
  - [CircuitPython on Linux and Raspberry Pi](#)
  - CircuitPythonで使うマイコンボードの機能をRaspberry Pi向けに変換するレイヤー
- CircuitPython向けのライブラリをRaspberry Piで扱うことができる（すべてではない）
  - Raspberry PiとCircityPythonのコードの相互利用もやりやすい（らしいです）

- CircityPythonのライブラリを使うことで接続センサーの扱いが楽
- ▲ どちらかの環境依存のコードを書く場合は扱いが難しい

	Raspberry Pi	MicroPython CircuitPython	Raspberry Pi Blinkaライブラリ
イメージ			
Python処理系	CPython	マイコン向け	CPython Blinkaライブラリ
扱いやすさ	通常のPythonとして	専用のライブラリ	CPython + CircuitPython 両者いいとこどり
電源	ACアダプタ	バッテリー利用も	ACアダプタ



## センサー情報取得の実例: BME280という温度湿度センサーから情報を取得する

Raspberry Pi + Blinkaライブラリを使って、  
デモしながら様子を見せていきます

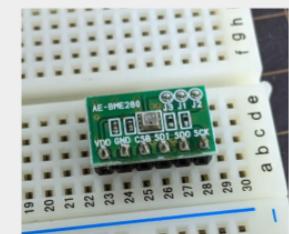
- 実際にラズパイ4BとBME280を接続します
- BME280はこちらものを使ってます
  - SPI接続です (I2Cの場合接続方法とセンサーのコードが少し違います)



ブレッドボード



ジャンプワイヤー



センサー: BME280  
(AE-BME280)



- はんだごて
- はんだ

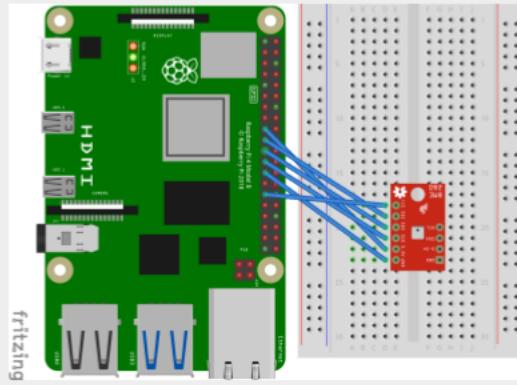


はんだこて台

## デモを試す時に必要なもの

- 利用する部品: ブレッドボード、ワイヤー、BME280 (利用するセンサー)
- 道具: はんだごて、はんだ、はんだこて台 (センサーにピンが実装されていない場合は必要です)
- 購入先:  
Amazonとかでも集まる。  
[秋月電子通商](#)、[スイッチサイエンス](#)、[aitendo](#)、[マルツオンライン](#)、[せんごくネット通販](#) がおすすめ

配線の様子を見せます



ラズパイGPIO	ラズパイ ピン番号(例)	SPI	AE-BME280のピン 名
GND	25	GND	VDD
3V3	17	3.3V	GND
GPIO 5	29	CS	CSB
GPIO 10 (SPI0 MOSI)	19	MOSI	SDI
GPIO 9 (SPI0 MISO)	21	MISO	SDO
GPIO 11 (SPI0 SCLK)	23	SCLK	SCK

## 配線方法まとめ

- 配線は一例です。
- 画像で利用しているBME280は AE-BME280ではないので、表を元に配線してください

環境作成してデモを見せながら披露

## 動作させたテストコードの例

```
import time
import board
from adafruit_bme280 import basic as adafruit_bme280
import digitalio

# SPI接続
spi = board.SPI()
bme_cs = digitalio.DigitalInOut(board.D5)
bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, bme_cs)

# 海の気圧を描くこと。大体1013hPaとのこと
bme280.sea_level_pressure = 1013.25

while True:
    print(f"温度: {bme280.temperature:.1f} C")
    print(f"湿度: {bme280.relative_humidity:.1f} %%")
    print(f"気圧: {bme280.pressure:.1f} hPa")
    print(f"高度: {bme280.altitude:.2f} meters")
    time.sleep(2)
```

## Tips: Raspberry Pi上で開発をしやすくする

VS Codeのリモート開発が便利です -> Remote-SSH

- ssh経由で扱うといい。ただpi3あたりからでないと、リモート開発できない (vscodeのリモートサーバーが対応するCPUアーキテクチャの問題)
- <https://www.raspberrypi.org/blog/coding-on-raspberry-pi-remotely-with-visual-studio-code/>

Remote SSH needs a Raspberry Pi 3 or 4. It is not supported on older Raspberry Pis, or on Raspberry Pi Zero.

# Dashでセンサー情報を可視化する

Dashライブラリを使ってセンサー情報を表示する  
ダッシュボードを作ります

## Dashの紹介

- Plotlyが作成しているWebアプリフレームワーク
  - Plotlyはグラフライブラリの名称でもある: `Plotly.js`、`Plotly.py`
- Plotlyとセットで使うと、データ分析向けのプロトタイピングがしやすい
- サンプル（有料機能を使ったものもあるので注意）<https://dash.gallery/Portal/>
  - ライブラリのサンプルには自動運転時の状況の可視化とかもある。  
かなりおもしろい

## Dashの特徴

- Dashはflask + reactで作られている。

Built on top of Plotly.js, React and Flask, Dash ties modern UI elements like dropdowns, sliders, and graphs directly to your analytical Python code.

<https://github.com/plotly/dash>
- (ほぼ) PythonのみでWebサイト構成が作れる
- コールバック機能を使ってインタラクティブ操作が可能
- データセットやDBを扱いたい場合は自前で用意
  - pandasを使ってplotlyのグラフを作れるので、pandas経由で何かしらをするときに便利



## (ほぼ) PythonのみでWebサイト構成が作れる

- HTMLを書く必要がない
- htmlのフォームや構造をラッピングしたコンポーネントを呼び出して構成を用意する
- Plotlyと連携して豊富なグラフを扱うことができる
- htmlな操作は知らないと扱いづらい

```
from dash import Dash, callback, html, dcc, Input, Output

# dashアプリの初期化
app = Dash(
    __name__,
    meta_tags=[{"name": "viewport", "content": "width=device-width, initial-scale=1"}],
)
app.title = "Hello Dash App"

def _layout():
    """アプリの全体のレイアウト"""
    return html.Div(
        [
            html.H2(app.title),
            html.Label("PythonのみでWEBアプリを作ります")
        ],
    )

if __name__ == "__main__":
    app.layout = _layout
    app.run_server(debug=True, host="0.0.0.0")
```

```
from dash import Dash, callback, html, dcc, Input, Output

# dashアプリの初期化
app = Dash(
    __name__,
    meta_tags=[{"name": "viewport", "content": "width=device-width, initial-scale=1"}],
)
app.title = "Hello Dash App"

def _layout():
    """アプリの全体のレイアウト"""
    return html.Div(
        [
            html.H2(app.title),
            html.Label("PythonのみでWEBアプリを作ります")
        ],
    )

if __name__ == "__main__":
    app.layout = _layout
    app.run_server(debug=True, host="0.0.0.0")
```



## フォームなどの操作から動的な変更:コールバック機能

- dashは動的な操作を可能にするためのコールバックという機能がある
- たとえばグラフの種類を変更することができる
- homeenvdashでは
  - ドロップダウンリストで部屋単位のセンサーグラフの切り替え
  - 定期的な表示の更新を行う (dcc.Interval)

```
from dash import Dash, callback, html, dcc, Input, Output

# dashアプリの初期化
app = Dash(
    __name__,
    meta_tags=[{"name": "viewport", "content": "width=device-width, initial-scale=1"}],
)
app.title = "Hello Dash Callback"

def _layout():
    return html.Div(
        [
            html.H1(app.title),
            html.Hr(),
            html.P("文字を入力すると、出力の部分が更新されます"),
            html.Div(
                [
                    html.Span("入力: "),
                    dcc.Input(id="input-form", value="Callbackを試しています", type="text"),
                ]
            ),
            html.P(id="output-p"),
        ]
    )
# 次へ続く
```

```
# 続き
@app.callback(Output("output-p", "children"), Input("input-form", "value"))
def update_output_text(input_value):
    # 引数がInputのvalueの値を取得
    # return側に更新したいコンポーネントを指定する。childrenは指定コンポーネントの子要素の事
    return "出力: {}".format(input_value)

if __name__ == "__main__":
    app.layout = _layout
    app.run_server(debug=True, host="0.0.0.0")
```

## Hello Dash Callback

文字を入力すると、出力の部分が更新されます

入力:

出力: Callbackを試しています



Inputフォームに文字を入れると  
「出力:」の先が更新される

## Hello Dash Callback

文字を入力すると、出力の部分が更新されます

入力:

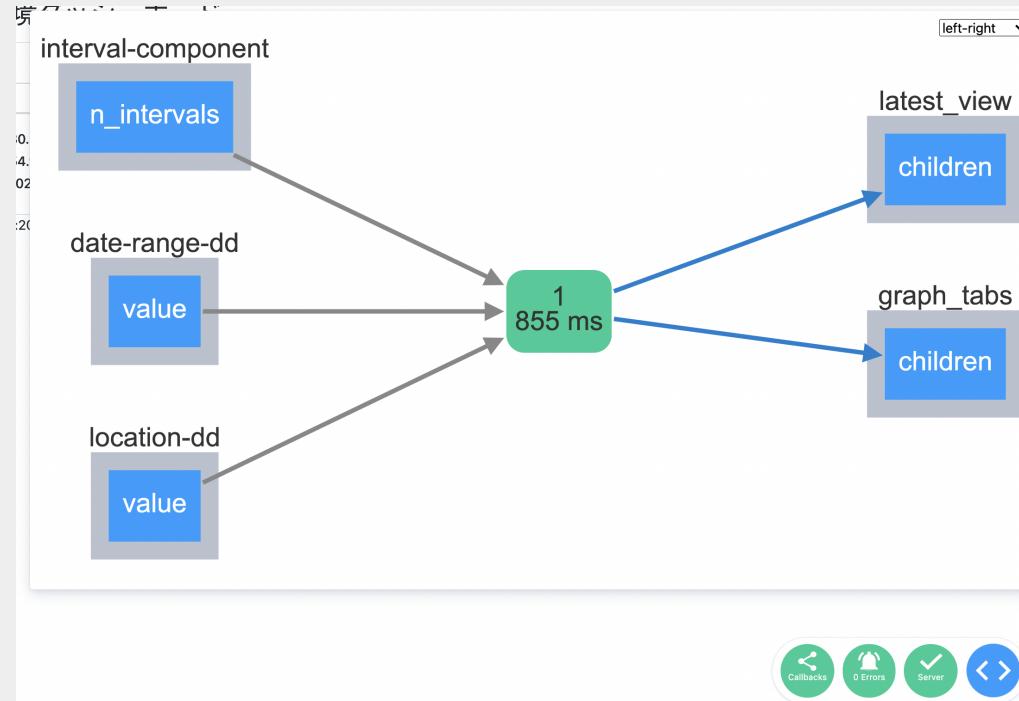
出力: 変更します😊

homeenvdashの1日のグラフ



ドロップダウンリストを切り替えることで、  
グラフの範囲を切り替える  
(実際はグラフを再描写している)

homeenvdashの1週間後のグラフ



## Tips: ホットリロードによる開発

- 自動的にリロードしてくれる。デバッグモードを有効にしておくと開発しやすい

```
if __name__ == "__main__":
    app.layout = _layout
    app.run_server(debug=True, host="0.0.0.0")
```

## デモ: センサー情報を可視化する

センサー情報の取得方法は、直接センサーの値を取りに行く

- 時系列グラフを作るなら、データの記録は必須になる。
  - 実際のところはファイルかDB, 外部のデータソースへ保存して扱う方がわかりやすい
  - 今回はCSVファイルに1分ごと、30回分の測定結果を保存

## センサーの最新の値を見る

```
# センサー取得用関数
def get_sensor_values():
    """センサーの値を取得する。記録は文字列にする"""
    temperature = f"{bme280.temperature:.1f}"
    relative_humidity = f"{bme280.relative_humidity:.1f}"
    pressure = f"{bme280.pressure:.1f}"
    return (temperature, relative_humidity, pressure)

# callback関数部分
def update_sensor_values(n):

    now_dt = datetime.datetime.now().astimezone()
    sensor_values = get_sensor_values()
    #中略
    latest_values = latest_sensor_values(sensor_values, now_dt)
    return latest_values
```

```
# レイアウト
def latest_sensor_values(sensor_values: tuple, now_datetime: datetime.datetime):
    """現在のセンサー値を描写する。"""
    latest_datetime = now_datetime.strftime("%Y-%m-%d %H:%M:%S")
    latest_temperature = sensor_values[0]
    latest_pressure = sensor_values[1]
    latest_humidity = sensor_values[2]

    return html.Div(
        [
            html.Label(f"更新時間 :{latest_datetime}"),
            html.Div(
                [
                    html.P(f"気温: {latest_temperature}°"),
                    html.P(f"湿度: {latest_pressure}%"),
                    html.P(f"気圧: {latest_humidity}hPa"),
                ],
            ),
        ],
        id="latest_values",
    )
```

記録したセンサーの値をグラフ化する

```
import pandas
import plotly.express as px
# 中略
from dash import Dash, callback, html, dcc, Input, Output
# 中略
def sensor_graphs():
    # センサー値を記録しているCSVファイルをDataFrameにする
    sensor_values_df = pandas.read_csv(
        SENSOR_VALUES_FILE, names=("datetime", "temperature", "humidity", "pressure")
    )
    # DataFrameをグラフに展開
    fig1 = px.line(sensor_values_df, x="datetime", y="temperature", title="温度")
    fig2 = px.line(sensor_values_df, x="datetime", y="humidity", title="湿度")
    fig3 = px.line(sensor_values_df, x="datetime", y="pressure", title="気圧")
    # レイアウトのコンポーネントを返す
    return html.Div(
        [
            dcc.Graph(id="temperature", figure=fig1),
            dcc.Graph(id="humidity", figure=fig2),
            dcc.Graph(id="pressure", figure=fig3),
        ],
        id="graphs",
    )
# 後略...
```

## Tips: 複雑化したらまとめてコンポーネント化する

複雑なアプリやレイアウトを作ると、構造も複雑になりがち

- 複雑になることはしょうがない
- 複雑になるので関数などで部品（カプセル）化していく
- コールバックで更新したい部分をカプセル化すると、コールバック側の更新処理を作る時に呼び出しやすい

慣れてくるとwebアプリを書いている様な扱いになってくる

## 最初にレイアウトの一部を関数化しておく

```
# 中略、関数化しておくと呼び出しやすくなり管理もしやすい
def sensor_graphs():
    """過去に記録したセンサーの値をグラフにする"""

    # センサー値を記録しているCSVファイルをDataFrameにする
    sensor_values_df = pandas.read_csv(
        SENSOR_VALUES_FILE, names=("datetime", "temperature", "humidity", "pressure")
    )
    # DataFrameをグラフに展開
    fig1 = px.line(sensor_values_df, x="datetime", y="temperature", title="温度")
    fig2 = px.line(sensor_values_df, x="datetime", y="humidity", title="湿度")
    fig3 = px.line(sensor_values_df, x="datetime", y="pressure", title="気圧")
    # レイアウトのコンポーネントを返す
    return html.Div(
        [
            dcc.Graph(id="temperature", figure=fig1),
            dcc.Graph(id="humidity", figure=fig2),
            dcc.Graph(id="pressure", figure=fig3),
        ],
        id="graphs",
    )
```

## 最後にレイアウト上で呼び出す

```
def _layout():
    now_dt = datetime.datetime.now().astimezone()
    sensor_values = get_sensor_values()
    save_sensor_values(sensor_values, now_dt, 50)

    return html.Div(
        [
            html.H2(app.title),
            html.Hr(),
            # 現在の値を取得
            latest_sensor_values(sensor_values, now_dt),
            # 温度、湿度、気圧のグラフ

            sensor_graphs(), # <= 関数でレイアウトオブジェクトを生成して呼び出す
            dcc.Interval(
                id="interval-component",
                interval=UPDATE_MINITS * 60 * 1000, # in milliseconds
                n_intervals=0,
            ),
        ],
    )
```

## Tips: Bootstrapを使ってデザインを良くする

Dashの便利なライブラリ: dash-bootstrap-components (dbc) によるデザインの整え方

- Dashのデメリットは、CSSを扱ったデザインがしづらい
  - レスポンシブ対応とか
- CSSフレームワークのBootstrapを扱いやすくコンポーネント化されている
- 扱いやすいもののBootstrapを知っている必要あり

※Dashのv2バージョンアップに伴って、dash-bootstrap-componentsも追従したバージョンアップが行われます。

デモ中は最新バージョンのリリース候補版を利用してます。

## デモ: デザインを整えてみる

- 実際にコードを見せつつ解説します
- レスポンシブ対応を例にします
- (スライドにコードを収めるには長すぎるので、詳しくはこちらをご覧ください)
  - (url掲載する)

## まとめ

homeenvdashプロジェクトを紹介しつつ、Pythonを使ったIoTとデータ可視化をデモを交えてお伝えしました。

- PythonとIoT
- Plotly Dashでダッシュボードを作る

本日お伝えしたこと

- 世の中にはないけど、ほしいなら自分で作ろう
- 世の中に存在しないデータを集めて見てみよう
  - 身近だけど見えないデータ
- 積みボードを活用していこう

## 最後にお知らせ

## PyCon mini Shizuoka 2021 やります

開催します🎉是非来てください🙌

- 2021/11/20 土曜日
- 詳しくは公式サイトをチェック
  - <https://shizuoka.pycon.jp/2021>
  - Twitterアカウント: @PyconShizu
- LTと参加者募集をします
- 同時にイベントの詳しい内容は近日公開します！

## 質問対策

- Dashを使うメリット,デメリットは?
  - メリットはHTMLを書く必要なくWEBアプリ作成ができる
  - デメリットはDB接続や高度な動的処理（ログイン処理やセッション管理）、API連携はそれほど得意ではない
    - それを解決する有償のサービスが提供されている
- 気圧センサーいろいろあるよ!
  - 知らなかつたのでありがとうございます！
- 精度はどう?
  - 数字というより変化を見たかったので、精度については今回は扱っていない
  - 実際にキャリブレーションする必要はあると思うし、高度なセンサーもあるので、用途によると思います。

