やまなしキャリアアップ・
ユニバーシティ

2024 年度
やまなしキャリアアップ・ユニバーシティ
ものづくり講座（応用）

「生産性向上のための IoT 実践技術講座」

山梨県立産業技術短期大学校

電子技術科

技術士（電気電子部門）永田　靖貴

生産性向上のための IoT 実践技術講座

IoT システムを自社開発：自分たちで実現する生産性向上

やまなしキャリアアップ・ユニバーシティ「ものづくり講座（応用）」では、IoT システムの構築を通じて IoT システムの自社開発に必要な技術を習得します。本講座では、小型かつ低コストが特徴の「Raspberry Pi」を使用して、生産性向上を目的とした IoT システムを構築します。また、現在稼働中の生産設備を有効活用し、生産設備や PLC と連携した IoT システムを構築します。そのため、生産設備の IoT 化を「低コスト」かつ「短期間」で実現できます。

本講座が提供する IoT 技術は、令和 2 年度に山梨県産業技術センターで実施した研究テーマ「PLC と安価な組み込みコンピュータを用いた生産性向上 IoT システムの開発」に基づくものです。この研究成果に基づく「IoT 技術」は、これまで20社以上の県内企業に導入され、生産現場におけるさまざまな問題解決に繋がっています。

本講座は、当初の研究成果に加えて、県内企業の成功事例から得られた技術を講座内容に反映しています。そのため、短期間で実践的な技術を習得できます。また、本講座は、「データ取得」、「データ蓄積・活用」、「IoT システム構築」の 3 つのパートに分かれており、IoT システムの構築に必要な技術を段階的に学習できます。

概　要

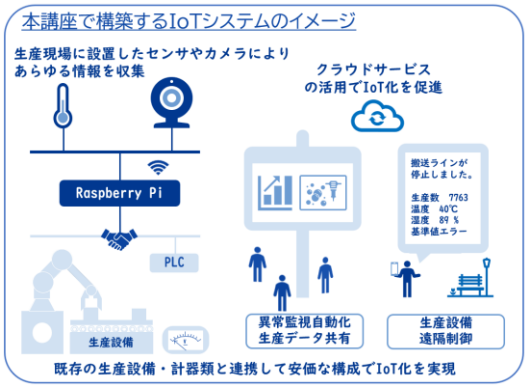
第 1 章　Raspberry Pi 開発環境構築
第 2 章　GPIO 制御
第 3 章　データ取得～センサ～
第 4 章　データ取得～生産設備～
第 5 章　データ取得～カメラ～
第 6 章　データ収集と活用
第 7 章　データベース構築・可視化
第 8 章　クラウドサービスの活用～オンラインストレージ～
第 9 章　クラウドサービスの活用～通知機能・AI～
第 10 章　クラウドサービスの活用～対話型システム～
第 11 章　システム拡張・機能強化

習得する IoT 技術

データ取得
センサ
生産設備（PLC）
カメラ・画像処理

データ蓄積・活用
データ分析，データベース
ダッシュボード構築

IoT システム構築
クラウドサービスとの連携
LINE, Google, OpenAI
無線通信（Wi-Fi, LoRa）



Copyright © 2024 YITJC All Rights Reserved.

Copyright © 2024 YITJC All Rights Reserved.

目次

第 1 章 Raspberry Pi 開発環境構築 1
1.1 ハードウェア 1
1.2 ソフトウェア 2
手順①～ソフトウェアダウンロード～ 2
手順②～ソフトウェアインストール～ 2
手順③～ショートカット作成～ 2
1.3 ネットワーク環境 3
1.4 OS インストール 3
手順①～MicroSD Card 挿入～ 3
手順②～MicroSD Card のフォーマット～ 3
手順③～Raspberry Pi Imager 起動～ 4
手順④～Raspberry Pi OS インストール設定～ 5
手順⑤～Raspberry Pi OS カスタマイズ設定～ 5
補足～OS カスタマイズ設定について～ 5
補足～ユーザー名・パスワードについて～ 6
手順⑥～MicroSD Card 装着～ 8
参考～OS ファイルのダウンロード～ 8
1.5 Raspberry Pi 操作方法（GUI） 9
手順①～HAT Board 装着～ 9
手順②～マウス・キーボード・ディスプレイ接続～ 9
手順③～AC アダプタ接続～ 10
手順④～起動画面確認～ 10
手順⑤～ユーザー名・ユーザー権限確認～ 11
手順⑥～ホームディレクトリ確認～ 11
1.6 Raspberry Pi 初期設定 12
手順①～LXTerminal 起動 12
手順②～Local 設定 13
補足～Raspi-config 画面の操作～ 13
手順③～Time Zone 設定～ 15
手順④～Keyboard 設定～ 16
手順⑤～Wi-Fi 使用国設定～ 18
手順⑥～再起動～ 20
1.7 無線 LAN 接続 21
手順①～Wi-Fi への接続～ 21
補足～IP Address の確認～ 22
手順②～ネットワーク設定ファイル～ 22
補足～Network サービスについて～ 22

Copyright © 2024 YITJC All Rights Reserved.

手順③～IP Address 固定～ 23
補足～sudo コマンド～ 23
補足～編集内容の説明～ 24
補足～Raspberry Pi 内部のファイル編集手順～ 25
手順④～設定内容の反映～ 26
補足～nmcli コマンド～ 26
1.8 Raspberry Pi 操作方法（CUI） 28
手順①～Tera Term～ 28
補足～cat コマンド～ 29
手順②～WinSCP～ 30
1.9 ディレクトリの作成 33
1.10 APT ライブラリの利用 34
手順①～apt-get update～ 34
手順②～apt-get upgrade～ 34
1.11 Python バージョン 35
参考～Python2 系の廃止～ 35
1.12 Raspberry Pi シャットダウン 36
手順①～シャットダウンコマンド実行～ 36
手順②～シャットダウン確認～ 36
1.13 バックアップファイル作成 37
手順①～シャットダウン～ 37
手順②～Windows PC に MicroSD Card 装着～ 37
手順③～Win32DiskImager を起動～ 37
手順④～img ファイル作成～ 38
手順⑤～img ファイル Read～ 38
1.14 バックアップファイル書き込み 39
1.15 システム開発環境 41
手順①～Raspberry Pi 起動～ 41
手順②～Tera Term 起動～ 41
手順③～WinSCP 起動～ 41
手順④～シャットダウン～ 42
参考～エディタの使い方～ 43
手順①～VS Code 日本語化～ 43
手順②～改行コードの可視化～ 44
手順③～改行コードの置換～ 45
補足～VS Code 画面表示の拡大・縮小～ 46
手順④～Python 拡張機能のインストール～ 46
補足～制限モードの解除～ 46
参考～Raspberry Pi の IP Address 調査～ 48

Copyright © 2024 YITJC All Rights Reserved.

参考～Raspberry Pi の時計事情～ 50
手順①～RTC モジュール装着～ 50
手順②～I2C アドレス確認～ 50
手順③～現在時刻確認～ 50
手順④～RTC 有効化～ 51
手順⑤～再起動～ 51
手順⑥～I2C アドレス再確認～ 51
手順⑦～時刻確認～ 51
手順⑧～RTC 時刻設定～ 52
手順⑨～サービスファイル作成～ 52
手順⑩～サービスファイル起動～ 52
手順⑪～再起動～ 52
手順⑫～システム時刻確認～ 53
補足～RTC 時刻について～ 53
第 2 章 GPIO 制御 54
2.1 HAT Board 54
2.2 GPIO 出力制御 55
手順①～プログラム配置～ 55
手順②～プログラム実行権限追加～ 55
手順③～プログラム実行～ 57
～プログラム解説～ 58
注意～GPIO.cleanup()～ 61
演習 1～GPIO ポート出力制御～ 62
2.3 GPIO 入出力制御 63
手順①～プログラム配置～ 63
手順②～プログラム実行権限追加～ 63
手順③～プログラム実行～ 63
～プログラム解説～ 64
演習 2～GPIO ポート入出力制御～ 66
2.4 シャットダウン機能の追加 67
手順①～GPIO ポート番号確認～ 67
手順②～プログラム配置～ 67
手順③～プログラム実行権限追加～ 67
手順④～プログラム実行～ 68
参考～自動起動の必要性～ 68
～プログラム解説～ 69
参考～Python の例外処理～ 72
2.5 自動起動 73
参考～Systemd について～ 73

Copyright © 2024 YITJC All Rights Reserved.

参考～Unit ファイルの保存先～ 73
手順①～Unit ファイル作成～ 73
～Unit ファイル解説～ 75
手順②～サービス再読み込み～ 77
手順③～サービス自動起動有効化～ 77
手順④～サービス開始～ 77
手順⑤～サービス動作状況確認～ 77
手順⑥～動作確認～ 77
参考～systemctl コマンド～ 78
参考～改行コードの確認方法～ 79
第 3 章 データ取得～センサ～ 80
3.1 インターフェース 80
参考～Grove シリーズ～ 80
I2C～Inter Integrated Circuit～ 81
SPI～Serial Peripheral Interface～ 82
UART～Universal Asynchronous Receiver Transmitter～ 83
1-Wire～One Wire Interface～ 84
3.2 Python ライブラリ 85
手順①～プログラム配置～ 85
手順②～プログラム実行権限追加～ 85
手順③～プログラム実行～ 85
参考～pip ツールの使用禁止～ 86
参考～ライブラリインストール方法～ 87
手順④～仮想環境構築～ 88
手順⑤～ライブラリインストール～ 89
参考～pip と pip3～ 89
手順⑥～プログラム実行～ 89
手順⑦～.bashrc ファイル確認～ 90
手順⑧～再起動～ 91
手順⑨～環境変数 PATH 確認～ 91
手順⑩～ライブラリー覧確認～ 91
～プログラム解説～ 92
参考～pipx コマンドの利用～ 94
手順①～pipx インストール～ 94
手順②～仮想環境構築～ 94
手順③～.bashrc ファイル確認～ 95
手順④～pyfiglet ライブラリインストール～ 96
手順⑤～ライブラリー覧確認～ 97
手順⑥～プログラム配置～ 97

Copyright © 2024 YITJC All Rights Reserved.

手順⑦～プログラム実行権限追加～ 97
手順⑧～プログラム実行～ 97
～プログラム解説～ 98
3.3 I2C～BME280 センサ～ 99
手順①～I2C アドレスの確認～ 99
手順②～I2C 有効設定～ 99
手順③～I2C 接続確認～ 101
手順④～I2C レジスタ値確認～ 101
手順⑤～smbus2 インストール～ 101
手順⑥～プログラム配置～ 102
手順⑦～プログラム実行権限追加～ 102
手順⑧～プログラム実行～ 102
～プログラム解説～ 103
3.4 クラス（Class） 109
手順①～プログラム実行権限追加～ 109
手順②～プログラム実行～ 109
～プログラム解説～ 110
手順③～モジュールのインポート（import）～ 115
～プログラム解説～ 115
参考～if _name_ == '_main_':～ 117
参考～プログラムとスクリプトの違い～ 117
3.5 SPI～アナログセンサ LM35～ 118
A/D コンバータ～MCP3002～ 118
アナログセンサ～LM35～ 118
参考～A/D 変換における最小分解能～ 119
注意～測定分解能～ 120
参考～アナログ温度センサモジュールの増幅回路～ 121
手順①～シャットダウン・電源ケーブル外す～ 121
注意～活線挿抜について～ 121
手順②～温度センサモジュールの取り付け～ 122
手順③～SPI 有効設定～ 122
手順④～SPI 設定確認～ 124
補足～カーネル（Kernel）～ 124
補足～lsmod, grep コマンド～ 124
手順⑤～spidev インストール～ 125
手順⑥～プログラム配置～ 125
手順⑦～プログラム実行権限追加～ 125
手順⑧～プログラム実行～ 125
～プログラム解説～ 126

Copyright © 2024 YITJC All Rights Reserved.

参考～SPI クラス～ 131
第 4 章 データ取得～生産設備～ 132
4.1 生産設備との連携 132
4.2 Modbus 通信 133
Modbus RTU 通信概要 133
Modbus TCP 通信概要 134
Function Code 135
Function Code 0x01～Read Coils～ 136
Function Code 0x05～Write Single Coil～ 137
Function Code 0x03～Read Holding Registers～ 138
Function Code 0x10～Write Multiple Holding Registers～ 139
Function Code 0x02～Read Discrete Input～ 140
4.3 有線ネットワークの設定 141
参考～PLC が同一ネットワークに接続している場合～ 141
手順①～ネットワーク管理リスト表示～ 142
手順②～有線ネットワークの設定確認～ 143
手順③～有線ネットワークの IP アドレス設定～ 143
手順④～PLC との接続確認～ 144
注意～有線 LAN 接続時の注意～ 145
手順⑤～ネットワーク接続の優先順位～ 146
手順⑥～Metric 設定値の変更～ 146
手順⑦～ネットワーク設定変更の反映～ 147
4.4 Modbus TCP 通信 148
手順①～pymodbus インストール～ 148
手順②～PLC と接続～ 148
手順③～プログラム配置～ 148
手順④～プログラム実行権限追加～ 148
手順⑤～プログラム実行～ 148
～プログラム解説 Modbus TCP 通信～ 150
4.5 Modbus RTU 通信 153
～プログラム解説 Modbus RTU 通信～ 153
4.6 Isolator Board の活用 157
回路構成と機能 157
入力回路 157
入力機能 158
出力回路 158
出力機能 159
Raspberry Pi との接続 159
参考～I2C 通信距離延長モジュール～ 160

Copyright © 2024 YITJC All Rights Reserved.

4.7 生産設備との接続 161
参考～エアーコンプレッサとの接続～ 162
4.8 Isolator Board の制御 164
手順①～Isolator Board, VPE Board の接続～ 164
手順②～Isolator Board 動作確認～ 165
手順③～プログラム配置～ 165
手順④～プログラム実行権限追加～ 165
手順⑤～プログラム実行～ 165
～プログラム解説～ 166
手順⑥～モジュールのインポート（import）～ 170
～プログラム解説～ 170
第 5 章 データ取得～カメラ・画像処理～ 172
5.1 画像データ保存先 172
5.2 静止画像取得～fswebcam～ 174
手順①～fswebcam インストール～ 174
手順②～Web カメラ接続～ 174
手順③～静止画像取得～ 175
参考～fswebcam コマンド～ 175
手順④～静止画像の確認～ 176
手順⑤～プログラム配置～ 176
手順⑥～プログラム実行権限追加～ 176
手順⑦～プログラム実行～ 177
～プログラム解説～ 177
5.3 静止画像取得～OpenCV～ 179
手順①～OpenCV インストール～ 179
手順②～プログラム配置～ 179
手順③～プログラム実行権限追加～ 179
手順④～プログラム実行～ 179
～プログラム解説～ 180
5.4 動画取得～OpenCV～ 182
手順①～プログラム配置～ 182
手順②～プログラム実行権限追加～ 182
手順③～プログラム実行～ 182
～プログラム解説～ 182
参考～Python の繰り返し（for 文）処理～ 185
参考～動画プログラムの sleep 時間～ 186
5.5 ドライブレコーダー 187
手順①～RPi.GPIO インストール～ 188
手順②～プログラム配置～ 188

Copyright © 2024 YITJC All Rights Reserved.

手順③～プログラム実行権限追加～ 188
手順④～プログラム実行～ 188
～プログラム解説～ 189
5.6 OCR（光学的文字認識） 195
5.7 Tesseract 197
手順①～tesseract インストール～ 197
手順②～pyocr インストール～ 197
手順③～OCR 作業ディレクトリ作成～ 197
手順④～テキスト変換用画像の保存～ 198
手順⑤～プログラム配置～ 198
手順⑥～プログラム実行権限追加～ 199
手順⑦～プログラム実行～ 199
～プログラム解説～ 199
手順⑧～OCR プログラム配置～ 202
手順⑨～OCR プログラム実行権限追加～ 202
手順⑩～OCR プログラム実行～ 202
～プログラム解説～ 202
5.8 7 セグメント LED 207
手順①～7 セグメント LED 画像の保存～ 207
手順②～プログラム配置～ 207
手順③～プログラム実行権限追加～ 207
手順④～プログラム実行～ 208
参考～Tesseract の学習データ～ 208
手順⑤～7 セグメント LED 専用学習データ保存～ 209
手順⑥～プログラム配置～ 209
手順⑦～プログラム実行権限追加～ 210
手順⑧～プログラム実行～ 210
5.9 OpenCV 画像処理 211
手順①～プログラム配置～ 211
手順②～プログラム実行権限追加～ 211
手順③～プログラム実行～ 211
参考～画像処理解説～ 212
参考～テキスト変換用プログラム～ 216
手順④～画像処理結果の確認～ 217
～プログラム解説～ 218
5.10 Google Cloud Vision 222
手順①～テスト用画像データ確認～ 222
手順②～テストサイトアクセス～ 222
手順③～テスト結果の確認～ 223

Copyright © 2024 YITJC All Rights Reserved.

参考～Google アカウント取得～ 224
第 6 章 データ収集と活用 228
6.1 データ収集 228
手順①～ディレクトリ作成～ 228
手順②～csv ファイル保存～ 228
手順③～プログラム配置～ 229
手順④～プログラム実行権限追加～ 229
手順⑤～プログラム実行～ 229
～プログラム解説～ 230
6.2 データ収集の自動化 234
手順①～Service ファイル作成～ 234
～Unit（サービス）ファイル解説～ 235
手順②～サービス再読み込み～ 235
手順③～サービス自動起動有効化～ 236
手順④～サービス開始～ 236
手順⑤～Timer ファイル作成～ 236
～Unit（タイマー）ファイル解説～ 237
手順⑥～タイマー再読み込み～ 238
手順⑦～タイマー自動起動有効化～ 238
手順⑧～タイマー開始～ 238
手順⑨～タイマー動作状況確認～ 238
参考～システムログの確認～ 239
6.3 データ可視化 240
手順①～matplotlib インストール～ 240
手順②～プログラム配置～ 240
手順③～プログラム実行権限追加～ 240
手順④～プログラム実行～ 240
～プログラム解説～ 242
6.4 データ分析 248
手順①～pandas インストール～ 248
手順②～プログラム配置～ 248
手順③～プログラム実行権限追加～ 248
手順④～プログラム実行～ 249
～プログラム解説～ 249
6.5 ヒストグラム作成 253
手順①～プログラム配置～ 253
手順②～プログラム実行権限追加～ 253
手順③～プログラム実行～ 253
～プログラム解説～ 254

Copyright © 2024 YITJC All Rights Reserved.

6.6 データ抽出・連結 258
手順①～プログラム配置～ 258
手順②～プログラム実行権限追加～ 258
手順③～プログラム実行～ 258
～プログラム解説～ 259
参考～Python のデータの取り出し～ 262
参考～データ活用例～ 264
第 7 章 データベース構築・可視化 265
7.1 システム構成 265
7.2 InfluxDB インストール 268
手順①～InfluxDB インストール～ 268
手順②～InfluxDB サービス自動化～ 268
手順③～データベース構築～ 269
7.3 Grafana インストール 270
手順①～Grafana インストール～ 270
参考～apt リポジトリ～ 271
手順②～Grafana サービス自動化～ 271
手順③～Grafana アクセス確認～ 272
手順④～Grafana パスワード設定～ 273
7.4 データ蓄積 275
手順①～influxdb インストール～ 275
手順②～プログラム配置～ 275
手順③～プログラム実行権限追加～ 275
手順④～プログラム実行～ 275
手順⑤～データ確認～ 276
参考～Query（クエリ）～ 276
参考～InfluxDB の構造～ 277
参考～Unix 時間～ 277
参考～InfluxQL（命令文）～ 278
～プログラム解説～ 279
7.5 データ蓄積の自動化 282
手順①～Service ファイル作成～ 282
～Unit（サービス）ファイル解説～ 283
手順②～サービス再読み込み～ 283
手順③～サービス自動起動有効化～ 283
手順④～サービス開始～ 284
手順⑤～Timer ファイル作成～ 284
～Unit（タイマー）ファイル解説～ 284
手順⑥～タイマー再読み込み～ 285

Copyright © 2024 YITJC All Rights Reserved.

手順⑦～タイマー自動起動有効化～ 285
手順⑧～タイマー開始～ 285
手順⑨～タイマー動作状況確認～ 285
7.6 ダッシュボードを活用した IoT システム（ver1） 287
手順①～Grafana アクセス～ 287
手順②～データベース設定～ 288
手順③～ダッシュボード作成～ 290
手順④～Query 設定～ 292
手順⑤～Panel 表示設定～ 293
手順⑥～ダッシュボード保存～ 296
参考～Dashboard メニュー画面～ 297
手順⑦～Panel 追加～ 297
手順⑧～Query 設定～ 298
手順⑨～グラフ選択～ 299
手順⑩～Panel 表示設定～ 300
手順⑪～Panel 複製～ 302
手順⑫～ダッシュボード完成～ 303
参考～生産設備稼働状況の監視～ 304
参考～Panel 作成例①～ 305
手順①～作成する Panel のイメージ～ 305
手順②～Panel の種類～ 305
手順③～凡例の設定～ 305
手順④～Panel の設定～ 306
手順⑤～データごとの書式設定～ 308
参考～Panel 作成例②～ 310
手順①～作成する Panel のイメージ～ 310
手順②～Panel の種類～ 310
手順③～凡例の設定～ 310
手順④～Panel の設定～ 311
手順⑤～データごとの書式設定～ 313
参考～Panel 作成例③～ 317
手順①～作成する Panel のイメージ～ 317
手順②～Panel の種類～ 317
手順③～凡例の設定～ 317
手順④～Panel の設定～ 318
参考～Panel 作成例④～ 320
手順①～作成する Panel のイメージ～ 320
手順②～Panel の種類～ 320
手順③～凡例の設定～ 320

Copyright © 2024 YITJC All Rights Reserved.

手順④～Panel の設定～ 321
第 8 章 クラウドサービスの活用～オンラインストレージ～ 323
8.1 ファイルアップロード 324
事前準備～アカウント取得～ 324
参考～Google Drive API の役割～ 324
手順①～Google アカウント設定～ 324
参考～認証方法の選択～ 331
手順②～サービスアカウント確認～ 332
手順③～Google Drive フォルダ作成～ 332
手順④～フォルダ ID 確認～ 335
手順⑤～サービスアカウントキーファイル保存～ 335
手順⑥～google-api-client インストール～ 335
手順⑦～プログラム配置～ 336
手順⑧～プログラム実行権限追加～ 336
手順⑨～プログラム実行～ 336
～プログラム解説～ 337
参考～Oauth2.0（オーオース 2.0）～ 340
8.2 フォルダ作成 343
手順①～プログラム配置～ 343
手順②～プログラム実行権限追加～ 343
手順③～プログラム実行～ 343
～プログラム解説～ 344
参考～MIME タイプ～ 345
8.3 指定フォルダへのアップロード 347
手順①～プログラム配置～ 347
手順②～プログラム実行権限追加～ 347
手順③～プログラム実行～ 347
～プログラム解説～ 348
8.4 ファイル上書き保存 351
手順①～プログラム配置～ 351
手順②～プログラム実行権限追加～ 351
手順③～プログラム実行～ 351
～プログラム解説～ 352
8.5 フォルダ内のデーター括アップロード 355
手順①～プログラム配置～ 355
手順②～プログラム実行権限追加～ 355
手順③～プログラム実行～ 355
～プログラム解説～ 357
参考～Google Drive API クラス～ 360

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説（GDrive_Class.py）～ 361
～プログラム解説（36_Pydrive.py）～ 369
8.6 オンラインストレージを活用した IoT システム（ver2） 372
～シェルスクリプト～ 372
～シェルスクリプト構成～ 372
手順①～サービス・タイマー停止～ 373
手順②～プログラム配置～ 374
手順③～プログラム実行権限追加～ 374
手順④～プログラム実行～ 374
～プログラム解説～ 375
手順⑤～Service ファイル編集～ 377
参考～WorkingDirectory の指定～ 377
手順⑥～Timer ファイル編集～ 378
手順⑦～サービス再読み込み～ 378
手順⑧～サービス自動起動有効化～ 378
手順⑨～サービス開始～ 378
手順⑩～サービス動作状況確認～ 379
手順⑪～Google Drive へのアップロード確認～ 379
参考～システムログの確認～ 380
参考～フンガ・トンガ噴火の観測～ 381
第 9 章 クラウドサービスの活用～通知機能・AI～ 382
9.1 通知機能～Gmail～ 383
手順①～アプリパスワードの発行～ 383
手順②～環境変数設定～ 385
参考～環境変数～ 385
手順③～環境変数の反映～ 386
手順④～プログラム配置～ 387
手順⑤～プログラム実行権限追加～ 387
手順⑥～プログラム実行～ 387
～プログラム解説～ 388
9.2 通知機能～X（旧 Twitter）～ 392
事前準備～アカウント取得～ 392
手順①～API 利用申請～ 392
手順②～tweepy インストール～ 401
手順③～環境変数設定～ 401
手順④～環境変数の反映～ 402
手順⑤～プログラム配置～ 402
手順⑥～プログラム実行権限追加～ 402
手順⑦～プログラム実行～ 402

Copyright © 2024 YITJC All Rights Reserved.

参考～非公開設定～ 403
～プログラム解説～ 404
9.3 定期的な通知機能をもつ IoT システム（ver3） 407
～ver3 システム構成検討～ 407
～シェルスクリプト構成～ 408
手順①～csv ファイル保存～ 409
手順②～CSV プログラム作成～ 409
手順③～プログラム実行権限追加～ 410
手順④～プログラム実行～ 410
手順⑤～Post プログラム作成～ 410
手順⑥～プログラム実行権限追加～ 410
手順⑦～プログラム実行～ 410
～Post プログラム解説～ 412
手順⑧～シェルスクリプト作成～ 418
手順⑨～プログラム配置～ 418
手順⑩～プログラム実行権限追加～ 418
手順⑪～プログラム実行～ 418
～プログラム解説～ 420
手順⑫～サービス・タイマー停止～ 422
手順⑬～Service ファイル編集～ 422
手順⑭～サービス再読み込み～ 423
手順⑮～サービス自動起動有効化～ 423
手順⑯～サービス開始～ 423
手順⑰～サービス動作状況確認～ 424
手順⑱～システムログ確認～ 424
9.4 生成 AI「ChatGPT」の活用 426
手順①～API Key 作成～ 426
参考～OpenAI API の料金体系～ 429
参考～トークン数の確認～ 429
手順②～openai インストール～ 431
手順③～環境変数設定～ 431
手順④～環境変数の反映～ 432
手順⑤～プログラム配置～ 432
手順⑥～プログラム実行権限追加～ 432
手順⑦～プログラム実行～ 432
～プログラム解説～ 433
参考～生成 AI を活用した Post～ 437
9.5 アラート通知機能をもつ IoT システム（ver4） 438
～ver4 システム構成検討～ 438

Copyright © 2024 YITJC All Rights Reserved.

～プログラム構成～ 439
手順①～プログラム配置～ 440
手順②～プログラム実行権限追加～ 440
手順③～プログラム実行～ 441
～プログラム解説～ 443
手順④～Unit ファイル作成～ 452
～Unit ファイル解説～ 452
手順⑤～サービス再読み込み～ 454
手順⑥～サービス自動起動有効化～ 454
手順⑦～サービス開始～ 454
手順⑧～サービス動作状況確認～ 455
手順⑨～IoT システム ver4 動作確認～ 455
参考～IoT システム稼働状況～ 456
参考～IoT システム適用例～ 457
参考～生産設備の保全方法～ 458
9.6 Google Cloud Vision～利用環境整備～ 459
手順①～支払い情報登録～ 459
手順②～google-cloud-vision インストール～ 462
手順③～プログラム配置～ 462
手順④～ディレクトリ作成・画像保存～ 462
手順⑤～プログラム実行権限追加～ 463
手順⑥～プログラム実行～ 463
9.7 Google Cloud Vision～OCR 機能活用～ 464
手順①～画像保存～ 464
手順②～プログラム配置～ 464
手順③～プログラム実行権限追加～ 464
手順④～プログラム実行～ 465
～プログラム解説～ 465
手順⑤～7 セグメント LED 画像保存～ 468
手順⑥～プログラム配置～ 468
手順⑦～プログラム実行権限追加～ 468
手順⑧～プログラム実行～ 468
手順⑨～日本語画像保存～ 471
手順⑩～プログラム配編集～ 471
手順⑪～プログラム実行～ 472
第 10 章 クラウドサービスの活用～対話型システム～ 473
10.1 AWS アカウント取得 474
10.2 対話型システム構成～クラウドサービス～ 478
10.2 対話の流れ 480

Copyright © 2024 YITJC All Rights Reserved.

10.3 自動応答の仕組み 482
10.4 Pub/Sub メッセージングモデル 483
参考～HTTPS 通信と MQTT 通信～ 484
参考～対話型システムの通信プロトコル～ 485
参考～メッセージの送受信の流れ～ 486
10.5 Subscribe 実装 487
手順①～IoT Core～ 488
手順②～リージョンの変更～ 488
手順③～モノのタイプ作成～ 489
手順④～ポリシーの作成～ 490
手順⑤～モノの登録～ 492
手順⑥～ディレクトリ作成～ 495
手順⑦～証明書配置～ 495
手順⑧～エンドポイント確認～ 496
手順⑨～AWSIoTPythonSDK インストール～ 497
手順⑩～環境変数設定～ 497
手順⑪～環境変数の反映～ 498
手順⑫～プログラム配置～ 498
手順⑬～プログラム実行権限追加～ 498
手順⑭～プログラム実行～ 499
～プログラム解説～ 501
重要～Subscriber 動作～ 505
10.6 LINE Bot～LINE Messaging API 設定～ 507
手順①～LINE Developers ログイン～ 508
手順②～LINE Business ID 作成～ 509
手順③～新規プロバイダー作成～ 510
手順④～チャネル作成～ 511
手順⑤～チャネルシークレット・ユーザーID 取得～ 513
手順⑥～LINE Bot 登録～ 514
手順⑦～チャネルアクセストークン取得～ 515
手順⑧～LINE Manager 応答設定～ 516
10.7 LINE Bot～Python 環境構築（Windows）～ 518
手順①～Python インストール～ 518
手順②～環境変数設定～ 520
手順③～Python インストール確認～ 522
10.8 LINE Bot～Bot プログラム作成～ 523
手順①～LINE Messaging API SDK インストール～ 523
手順②～LINE Bot プログラム作成～ 523
手順③～zip ファイル作成～ 524

Copyright © 2024 YITJC All Rights Reserved.

10.9 LINE Bot～Bot プログラム保存～ 525
手順①～AWS S3 バケット作成～ 525
手順②～ファイルアップロード～ 527
手順③～オブジェクト URL 取得～ 528
10.10 LINE Bot～Bot プログラムデプロイ～ 530
手順①～Lambda 関数作成～ 530
手順②～IAM Role の作成～ 532
手順③～zip ファイルアップロード～ 533
重要～Lambda 関数の編集～ 534
10.11 LINE Bot～API Gateway～ 535
手順①～REST API 構築～ 535
手順②～API 作成～ 535
手順③～リソース作成～ 536
手順④～メソッド作成～ 536
補足～LINE Messaging API 署名検証～ 538
注意～x-line-signature～ 538
手順⑤～API デプロイ～ 539
手順⑥～Webhook URL 取得～ 540
手順⑦～Webhook URL 設定～ 540
手順⑧～動作確認～ 541
～プログラム解説～ 542
10.12 LINE Bot～ボタンテンプレート～ 547
手順①～Lambda プログラム作成～ 548
手順②～Lambda 関数ファイルアップロード～ 548
手順③～Lambda 関数デプロイ～ 550
手順④～動作確認～ 551
～プログラム解説～ 551
10.13 LINE Bot～Publish 機能実装～ 557
手順①～Lambda プログラム作成～ 558
手順②～Lambda 関数ファイルアップロード～ 559
手順③～Lambda 関数デプロイ～ 560
手順④～Publish 動作確認～ 561
手順⑤～Subscribe プログラム実行～ 564
手順⑥～Raspberry Pi 動作確認～ 564
手順⑦～Unit ファイル作成～ 565
～Unit ファイル解説～ 566
手順⑧～サービス再読み込み～ 568
手順⑨～サービス自動起動有効化～ 568
手順⑩～サービス開始～ 568

Copyright © 2024 YITJC All Rights Reserved.

手順⑪～サービス動作状況確認～ 568
手順⑫～動作確認～ 569
手順⑬～システムログ確認～ 569
～プログラム解説～ 570
10.14 直接応答～LINE ユーザーID～ 577
参考～開発した LINE Bot の共有～ 577
取得方法①～LINE Developers（復習）～ 578
取得方法②～AWS CloudWatch～ 578
手順①～CloudWatch～ 579
手順②～通信ログ確認～ 579
手順③～LINE ユーザーID 取得～ 580
手順④～line_bot_sdk インストール～ 581
手順⑤～環境変数設定～ 581
手順⑥～環境変数の反映～ 582
手順⑦～プログラム配置～ 582
手順⑧～プログラム実行権限追加～ 582
手順⑨～プログラム実行～ 582
～プログラム解説～ 583
参考～画像ファイル URL 取得～ 586
手順①～ファイル共有設定～ 586
手順②～共有設定変更～ 586
手順③～共有リンク（URL）取得～ 586
手順④～画像ファイルのリンク先（URL）作成～ 587
10.15 対話機能をもつ IoT システム（ver5） 588
～対話型システム～ 588
手順①～Lambda プログラム作成～ 589
手順②～Lambda 関数ファイルアップロード～ 590
手順③～Lambda 関数デプロイ～ 591
手順④～対話プログラム編集～ 592
手順⑤～サービス・タイマー停止～ 593
手順⑥～Service ファイル編集～ 593
手順⑦～サービス再読み込み～ 594
手順⑧～サービス自動起動有効化～ 594
手順⑨～サービス開始～ 594
手順⑩～サービス動作状況確認～ 595
手順⑪～システムログ確認～ 595
～Publish プログラム解説～ 596
～対話プログラム(TEMP.py)解説～ 604
参考～IoT システム稼働状況～ 608

Copyright © 2024 YITJC All Rights Reserved.

第 11 章 システム拡張・機能強化 609
11.1 Sensor Board ver1 機能 609
11.2 LoRa Board ver1 機能 611
参考～ES920LR2（EASEL 社）～ 611
参考～ISM バンド～ 611
参考～LoRa 通信テスト結果～ 612
11.3 ESP32 開発環境構築 613
手順①～PlatformIO IDE インストール～ 613
手順②～PlatformIO Home 起動～ 614
手順③～プロジェクト作成～ 614
手順④～プログラム作成～ 617
手順⑤～ビルド実行～ 617
手順⑥～ESP32 接続～ 618
手順⑦～プログラム書き込み～ 618
手順⑧～動作確認～ 619
～プログラム解説～ 620
参考～Sensor Board ピン割付～ 621
11.4 BME280 センサデータ取得 622
手順①～プロジェクト配置～ 622
手順②～プロジェクトオープン～ 622
手順③～シリアルモニタ設定～ 623
手順④～ビルド実行～ 624
手順⑤～ESP32 接続～ 624
手順⑥～プログラム書き込み～ 624
手順⑦～動作確認～ 624
～プログラム解説～ 625
11.5 MQTT 通信～データ送信～ 629
復習～MQTT 通信～ 629
手順①～プロジェクト配置～ 630
手順②～プロジェクトオープン～ 630
手順③～シリアルモニタ設定～ 631
手順④～JSON ライブラリ追加～ 632
手順⑤～PubSubClient ライブラリ追加～ 634
手順⑥～プログラム編集～ 636
手順⑦～ビルド実行～ 636
手順⑧～ESP32 接続～ 636
手順⑨～プログラム書き込み～ 637
～プログラム解説～ 637
注意～MQTT 通信のクライアント ID～ 641

Copyright © 2024 YITJC All Rights Reserved.

参考～ESP32 の Sleep モード～ 642
11.6 MQTT 通信～データ受信～ 644
手順①～mosquitto インストール～ 644
手順②～Broker ファイル編集～ 644
手順③～サービス再読み込み～ 645
手順④～サービス再起動～ 645
手順⑤～paho インストール～ 645
手順⑥～プログラム配置～ 646
手順⑦～プログラム実行権限追加～ 646
手順⑧～プログラム実行～ 646
手順⑨～ESP32 接続～ 646
手順⑩～動作確認～ 646
～プログラム解説～ 647
参考～Raspberry Pi レス IoT システム～ 650
11.7 LoRa 通信～データ送信～ 651
（復習）～活線挿抜～ 651
手順①～プロジェクト配置～ 651
手順②～プロジェクトオープン～ 652
手順③～シリアルモニタ設定～ 653
手順④～ビルド実行～ 653
手順⑤～ESP32 接続～ 653
手順⑥～プログラム書き込み～ 654
～プログラム解説～ 654
11.8 LoRa 通信～データ受信～ 666
手順①～Lora Board 接続～ 666
手順②～UART 設定～ 666
補足～Raspi-config 画面の操作～ 666
手順③～pyserial インストール～ 669
手順④～プログラム配置～ 669
手順⑤～シャットダウンプログラムの更新～ 669
手順⑥～Service ファイル編集～ 670
手順⑦～サービス再読み込み～ 670
手順⑧～サービス再起動～ 670
手順⑨～プログラム実行権限追加～ 670
手順⑩～プログラム実行～ 670
手順⑪～電池 Box 接続～ 671
手順⑫～ESP32 接続～ 671
手順⑬～動作確認～ 672
参考～レスポンスコード～ 672

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～ 673
注意～データ取りこぼし対策～ 678
11.9 LoRa 通信パラメータが与える影響 681
注意～送信時間制限（4 秒）～ 681
参考～通信パラメータの設定～ 682
参考～LoRa 通信コマンド～ 683
参考～LoRa 通信テスト（屋内）～ 684
手順①～子機準備～ 684
手順②～親機準備～ 684
手順③～親機プログラム実行～ 684
手順④～子機データ送信～ 685
参考～LoRa 通信テスト（屋外）～ 686
手順①～子機準備～ 686
手順②～ライブラリ追加～ 686
手順③～Ambient 環境設定～ 687
手順④～Ambient インストール～ 688
手順⑤～親機準備～ 688
手順⑥～親機プログラム実行～ 689
手順⑦～子機データ送信～ 689
手順⑧～通信範囲確認～ 690
参考～屋外通信テスト（見通しの良い環境）～ 693
参考～屋外通信テスト（見通しの悪い環境）～ 694
11.10 Isolator Board 制御 695
手順①～Isolator Board, VPE Board の接続～ 696
手順②～Isolator Board 動作確認～ 697
手順③～プロジェクト配置～ 697
手順④～プロジェクトオープン～ 698
手順⑤～シリアルモニタ設定～ 699
手順⑥～MCP23017 ライブラリ追加～ 699
手順⑦～ビルド実行～ 701
手順⑧～ESP32 接続～ 702
手順⑨～プログラム書き込み～ 702
手順⑩～動作確認～ 702
～プログラム解説～ 703
やまなしキャリアアップ・ユニバーシティ「ものづくり講座」 707

Copyright © 2024 YITJC All Rights Reserved.

Copyright © 2024 YITJC All Rights Reserved.

第 1 章 Raspberry Pi 開発環境構築

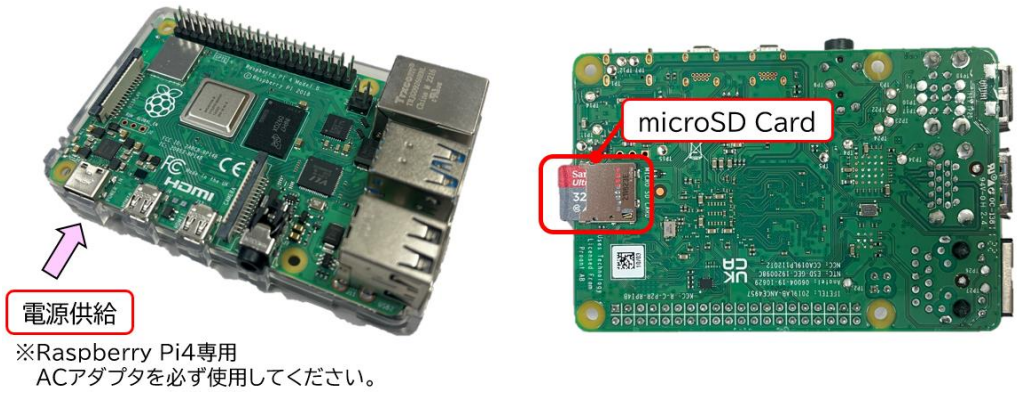
本講座は Raspberry Pi を使用して生産現場の生産性向上に貢献するシステム（以降、IoT システム）を構築しながら、IoT システムを自社で開発するために必要な技術を習得します。ここでは、IoT システムを開発するために必要な開発環境について説明します。

1.1 ハードウェア

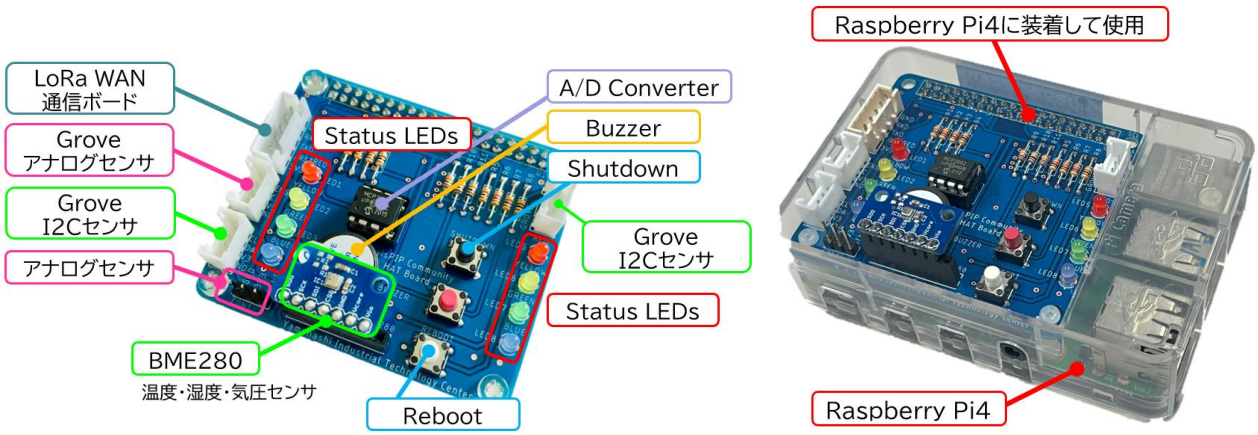
IoT システムを開発するために必要なハードウェアは、次のとおりです。

1. Raspberry Pi 4 （AC アダプタ含む）
2. microSD card: 32GB
3. HAT Board （本講座専用基板： Raspberry Pi に装着して使用）
4. PLC (Mitsubishi: FX5U Series)
5. シーケンス制御盤（生産設備を想定した教材）

※4 および 5 については、山梨県立産業技術短期大学校の設備を使用します。



Raspberry Pi 4



HAT Board

Copyright © 2024 YITJC All Rights Reserved.

1.2 ソフトウェア

「生産性向上 IoT システム」を開発するために必要なソフトウェアは、以下のとおりです。

共有サイト「3_Software」に保存してあります。

1. Raspberry Pi Imager（imager_1.8.5）
   - 用途: Raspberry Pi OS を MicroSD に書き込む

2. VS Code（VSCodeUserSetup-x64-1.87.2）
   - 用途: Python プログラム作成・編集

3. Tera Term（teraterm-5.2）
   - 用途: Windows PC から Raspberry Pi をリモート操作（SSH 接続）

4. WinSCP（WinSCP-6.3.2-Setup）
   - 用途: Windows PC から Raspberry Pi をリモート操作（SSH 接続）

5. SDCardFormatter（SD Card Formatter 5.0.2 Setup EN）
   - 用途: MicroSD Card をフォーマット

6. Win32DiskImager（Win32DiskImager-1.0.0-binary フォルダ内の Win32DiskImager）
   - 用途：Raspberry Pi で構築したシステムをバックアップ（img ファイルとして読み出し）
   - 用途: img 形式のイメージファイルを MicroSD Card へ書き込み

手順①～ソフトウェアダウンロード～

共有サイト「3_Software」に保存してあるファイルをすべてダウンロードしてください。

手順②～ソフトウェアインストール～

1～5 のソフトウェアをインストールしてください。

各ソフトウェアをインストールする際は、インストーラの指示にしたがって（インストーラデフォルトの設定にしたがって）、インストールしてください。

手順③～ショートカット作成～

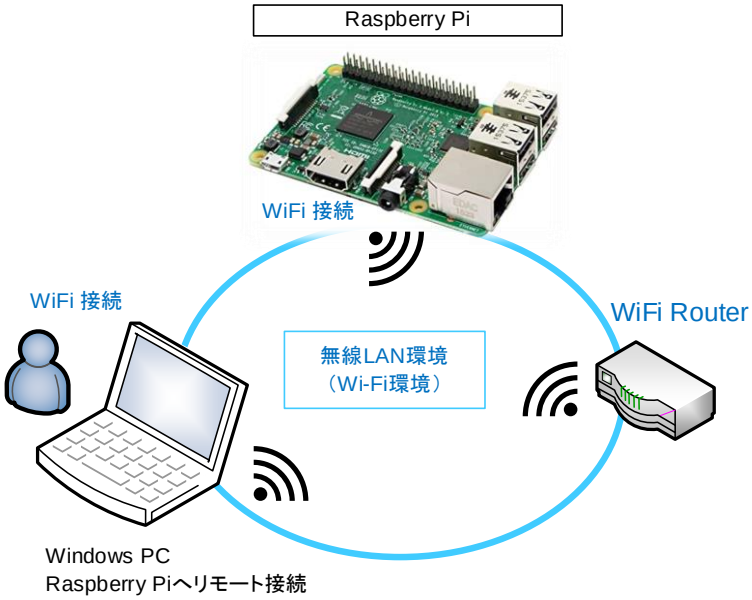
「6 Win32DiskImager」については、インストールする必要はありません。ただし、「Win32DiskImager-1.0.0-binary」フォルダ内にある Win32DiskImager（アプリケーションファイル）を使用するため、ショートカットをデスクトップに作成してください。

開発に使用する Windows PC のデスクトップに次のショートカットを作成してください。

Copyright © 2024 YITJC All Rights Reserved.

1.3 ネットワーク環境

ネットワーク接続環境を示します。Windows PC と Raspberry Pi を同じ無線 LAN（Wi-Fi）に接続します。Raspberry Pi の有線 LAN ポートは PLC との通信で使用するため、この段階ではまだ LAN ケーブルを接続しないでください。



1.4 OS インストール

Raspberry Pi 用の OS（オペレーティングシステム）をインストールします。ここでは、Raspberry Pi 財団にサポートされた公式 OS である「Raspbian（ラズビアン）」のインストール手順について説明します。Raspbian は、Linux の Debian という OS がベースの無料オペレーティングシステムです。

余談ですが、Debian のコードネームには「トイ・ストーリー」のキャラクター名が採用されています。本講座で使用する最新版は、2023 年 10 月 11 日にリリースされた「Raspbian Bookworm」です。

手順①～MicroSD Card 挿入～

MicroSD Card（32GB）を Windows PC に挿入してください。MicroSD Card スロットがない PC の場合は、MicroSD Card リーダを使用して Windows PC に挿入してください。

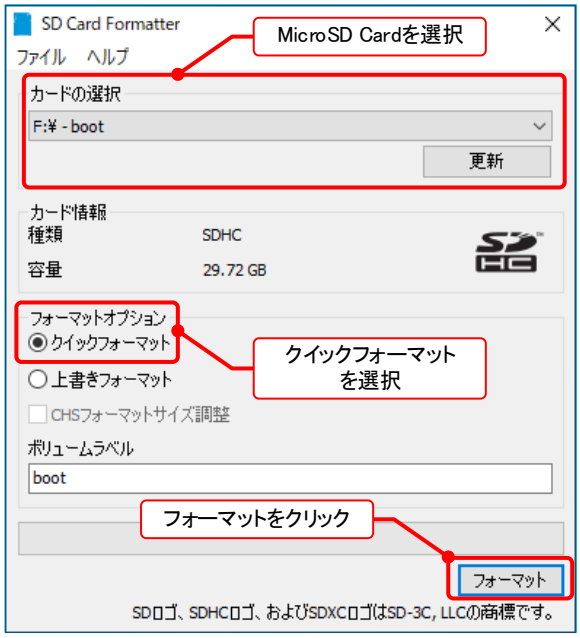
手順②～MicroSD Card のフォーマット～

ソフトウェア「SDCardFormatter」を使用して MicroSD Card をフォーマットします。

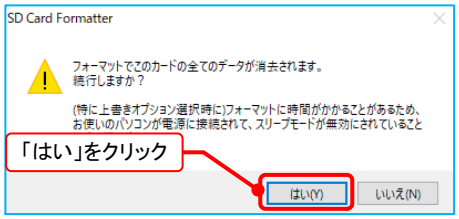
まず、「SDCardFormatter」をダブルクリックして起動してください。

Copyright © 2024 YITJC All Rights Reserved.

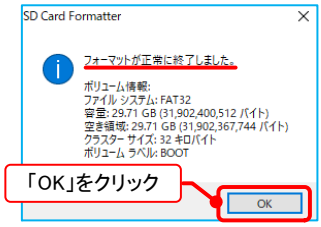
続いて、次の画面にしたがってフォーマットを実行してください。



確認メッセージが表示されるので、「はい」をクリックしてください。

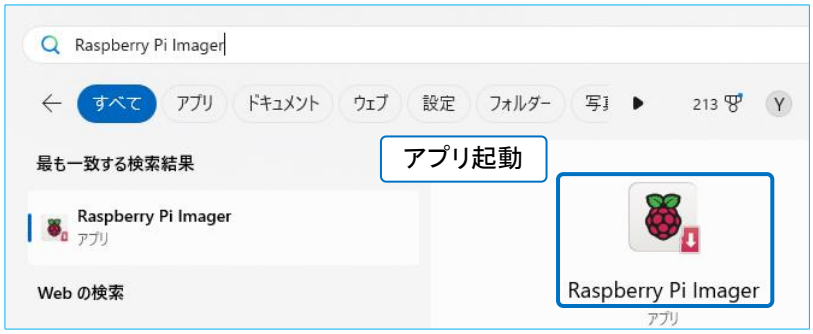


次のメッセージが表示されたら、フォーマット終了です。



手順③～Raspberry Pi Imager 起動～

Raspberry Pi OS をインストールするため、Raspberry Pi Imager を起動します。Windows アプリ検索から Raspberry Pi Imager を検索して起動してください。

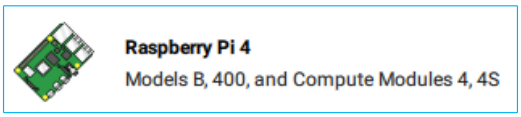


Copyright © 2024 YITJC All Rights Reserved.

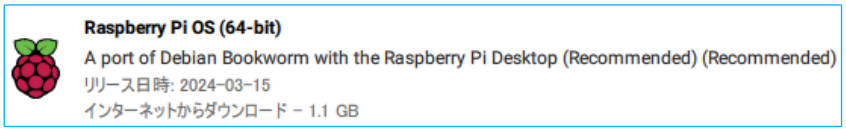
手順④～Raspberry Pi OS インストール設定～

次の画面にしたがってインストールの設定を進めます。

①デバイスの選択：Raspberry Pi4 を選択してください



②OS の選択：Raspberry Pi OS（64-bit）を選択してください

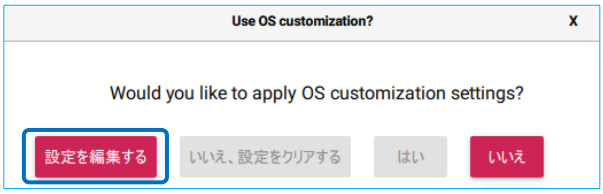


③ストレージの選択：ノート PC にマウントした MicroSD Card を選択してください

①～③を設定した後、「次へ」をクリックしてください。

手順⑤～Raspberry Pi OS カスタマイズ設定～

OS のカスタマイズに関する確認メッセージが表示されるので、「設定を編集する」をクリックしてください。



補足～OS カスタマイズ設定について～

Raspberry Pi Imager を使用して OS を MicroSD Card に書き込む際、Raspberry Pi が接続する Wi-Fi（ssid, パスワード）や Time Zone などの設定を書き込むことができます。カスタマイズで設定する項目については、Raspberry Pi OS を書き込んだ後でも設定できます。そのため、ここでは Raspberry Pi を起動するために必要最低限の項目だけ設定します。

Copyright © 2024 YITJC All Rights Reserved.

次の画面にしたがってカスタマイズの設定を進めます。

「一般」タブをクリックし、ユーザー名とパスワードを入力してください。

ユーザー名：cuu

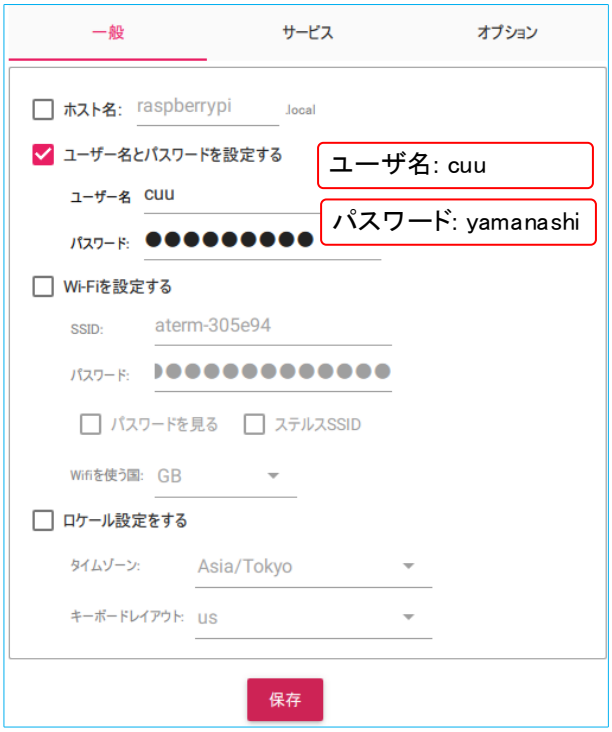
パスワード：yamanashi

補足～ユーザー名・パスワードについて～

ユーザー名（cuu）およびパスワード（yamanashi）は、後から変更することが可能です。

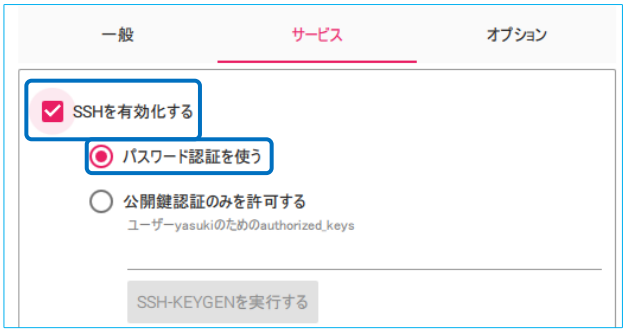
本講座は、「cuu」をユーザー名、「yamanashi」をパスワードとして使用します。

なお、Raspberry Pi のデフォルトユーザー名およびパスワードとして、「pi」および「raspberry」が長らく使用されてきました。最新の OS はセキュリティ強化のため、ユーザー名およびパスワードには独自の設定が求められています。



「サービス」タブをクリックし、「SSH を有効化する」にチェックを入れてください。

また、「パスワード認証を行う」が選択されていることを確認してください。

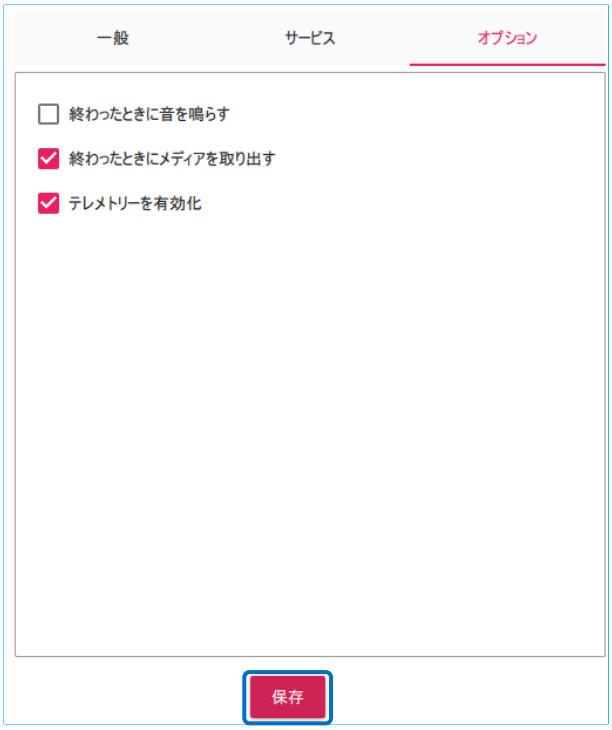


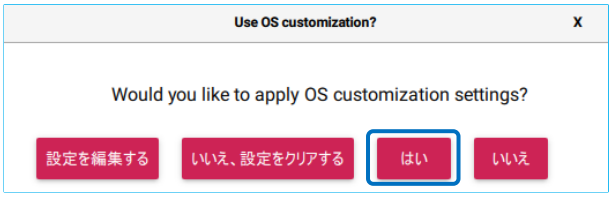
SSH とは「Secure Shell」の略称で、ネットワークに接続された機器を遠隔操作し、管理するための手段です。

Windows PC から同じ無線 LAN（Wi-Fi）に接続した Raspberry Pi を遠隔操作する際に使用する手段です。

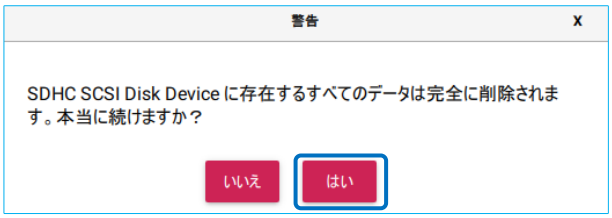
Copyright © 2024 YITJC All Rights Reserved.

最後に、「保存」をクリックしたあと、「はい」をクリックしてください。





MicroSD Card への書き込み前に警告（書き込みの確認）メッセージが表示されますが、「はい」をクリックして書き込みを開始してください。



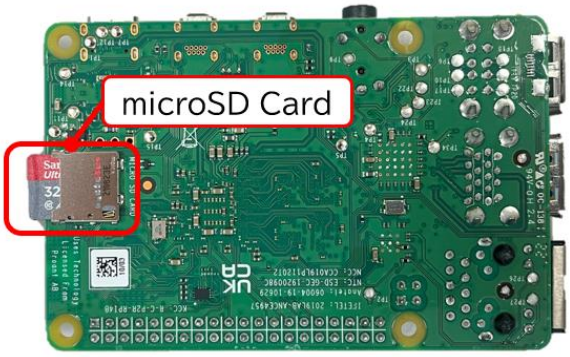
書き込みが正常に開始されると、次の画面が表示されます。

Copyright © 2024 YITJC All Rights Reserved.

書き込みが完了すると、次のメッセージが表示されます。
「続ける」をクリックし、MicroSD Card をノート PC から外してください。

手順⑥～MicroSD Card 装着～

MicroSD Card を Raspberry Pi に装着します。装着する際、MicroSD Card の向きに注意してください。



参考～OS ファイルのダウンロード～

Raspberry Pi Imager で書き込む OS ファイルは、次のサイトで公開されています。このサイトにある OS ファイルをダウンロードして、Win32DiskImager で書き込むこともできます。

ダウンロード先：https://www.raspberrypi.com/software/operating-systems/#raspberry-pi-os-64-bit

なお、本講座で使用する OS「2024-03-15-raspios-bookworm-arm64-full.img」は、共有サイト「2_OS」に保存してあります。

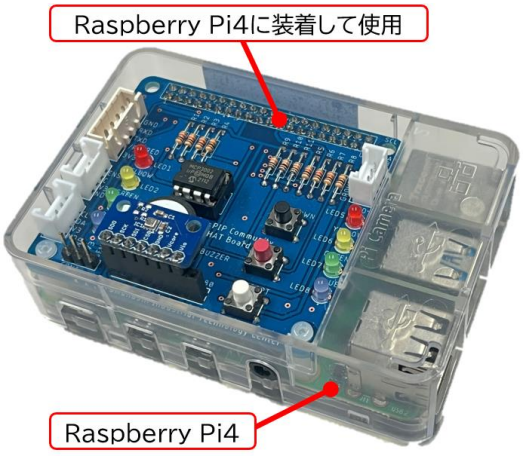
Copyright © 2024 YITJC All Rights Reserved.

1.5 Raspberry Pi 操作方法（GUI）

Raspberry Pi の操作方法「GUI」を紹介します。GUI（Graphical User Interface）は、Raspberry Pi を視覚的に操作する方法です。次の手順にしたがって、GUI 操作を学習します。

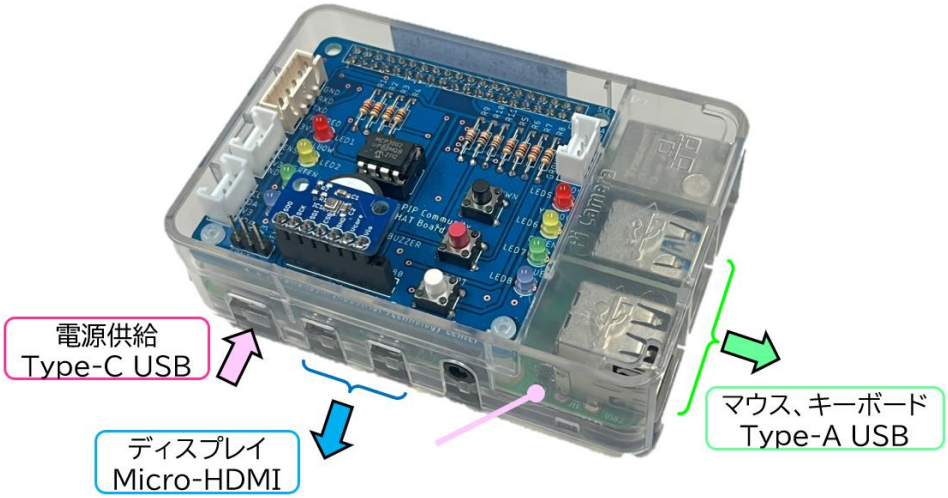
手順①～HAT Board 装着～

Raspberry Pi に HAT Board を装着し、専用のケースに収納してください。ケースに収納する際、ケースの一部（リブ）が HAT Board と干渉する場合があります。この場合は、干渉するリブをカットしてから収納してください。



手順②～マウス・キーボード・ディスプレイ接続～

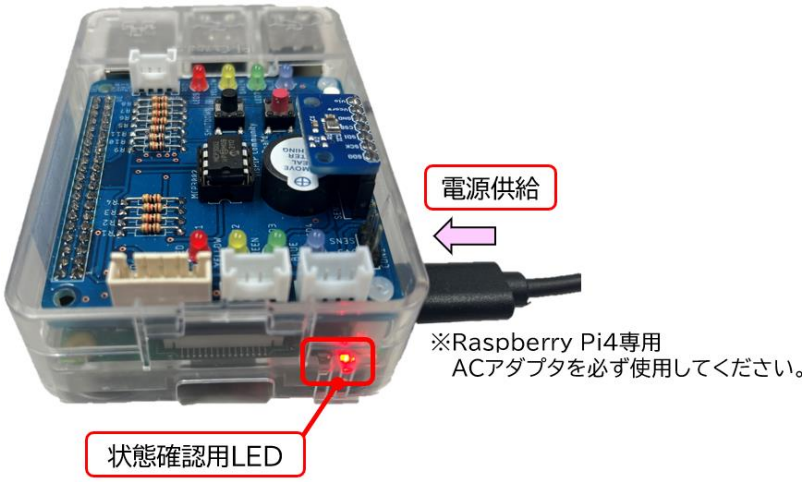
Raspberry Pi にマウス、キーボード、ディスプレイを接続してください。マウスおよびキーボードは、USB ポートを使用してください。ディスプレイとの接続は、2 つある映像出力端子（Micro-HDMI）のうち、いずれかを使用してください。このとき、AC アダプタはまだ接続しないでください。



Copyright © 2024 YITJC All Rights Reserved.

手順③～AC アダプタ接続～

Raspberry Pi に専用の AC アダプタを接続して電源を供給します。



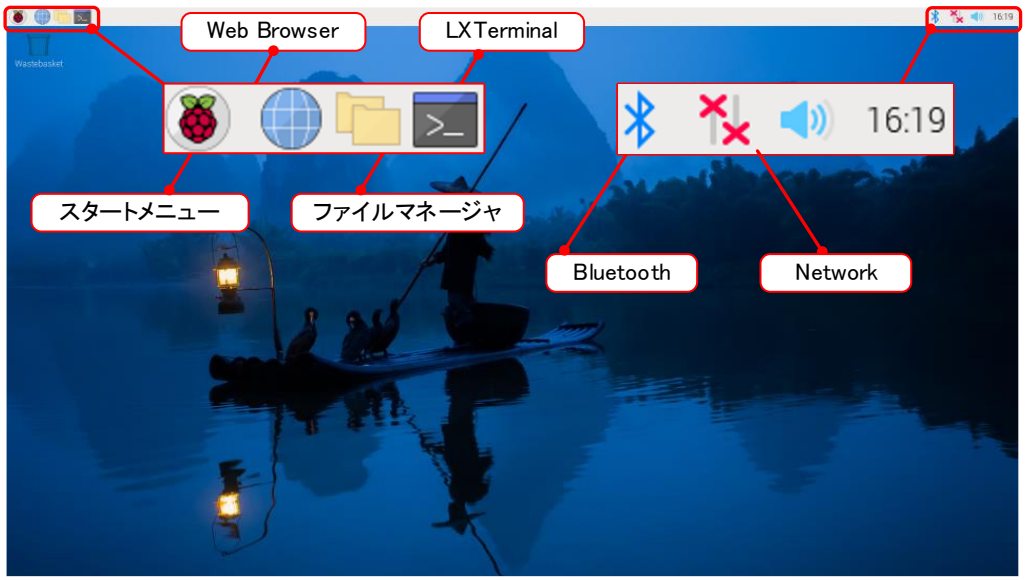
状態確認用の LED の点灯状態を確認します。

赤色 LED は、電源供給に問題がなければ点灯状態です。供給電圧が低下した場合は点滅します。赤色 LED が消灯または点滅している場合は、電源系統の不具合が考えられるため、供給電圧を確認してください。

緑色 LED は、MicroSD Card へのアクセス時に点滅します。アクセスが無ければ、消灯状態です。

手順④～起動画面確認～

Raspberry Pi が正常に起動すると、次の画面が表示されます。



Copyright © 2024 YITJC All Rights Reserved.

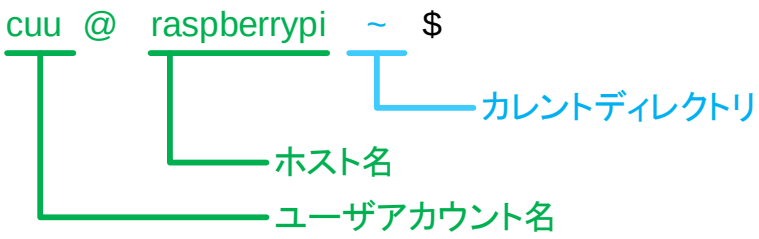
手順⑤～ユーザー名・ユーザー権限確認～

LXTerminal を起動（LXTerminal アイコンをクリック）すると、次の画面が表示されます。

「cuu」は、ユーザーカウント名

「raspberrypi」は、ホスト名

「~」は、カレントディレクトリを示しています。



whoami コマンドを実行してユーザー名を確認します。

cuu@raspberrypi: ~ $ whoami

```
cuu@raspberrypi:~ $ whoami
cuu
```

本テキストは、実行するコマンドを緑字で表現しています。コマンドを実行するときは、テキストからコピー＆ペーストしてください。

groups コマンドを実行して cuu ユーザーの権限を確認してください。

cuu@raspberrypi: ~ $ groups cuu

```
cuu@raspberrypi:~ $ groups cuu
cuu : cuu adm dialout cdrom sudo audio video plugdev games users input rend
er netdev spi i2c gpio lpadmin
```

cuu ユーザー権限の中に、sudo 権限（管理者実行権限）が含まれていることがわかります。

手順⑥～ホームディレクトリ確認～

pwd コマンドを実行して Raspberry Pi を起動したときのディレクトリの場所を確認してください。

cuu@raspberrypi: ~ $ pwd

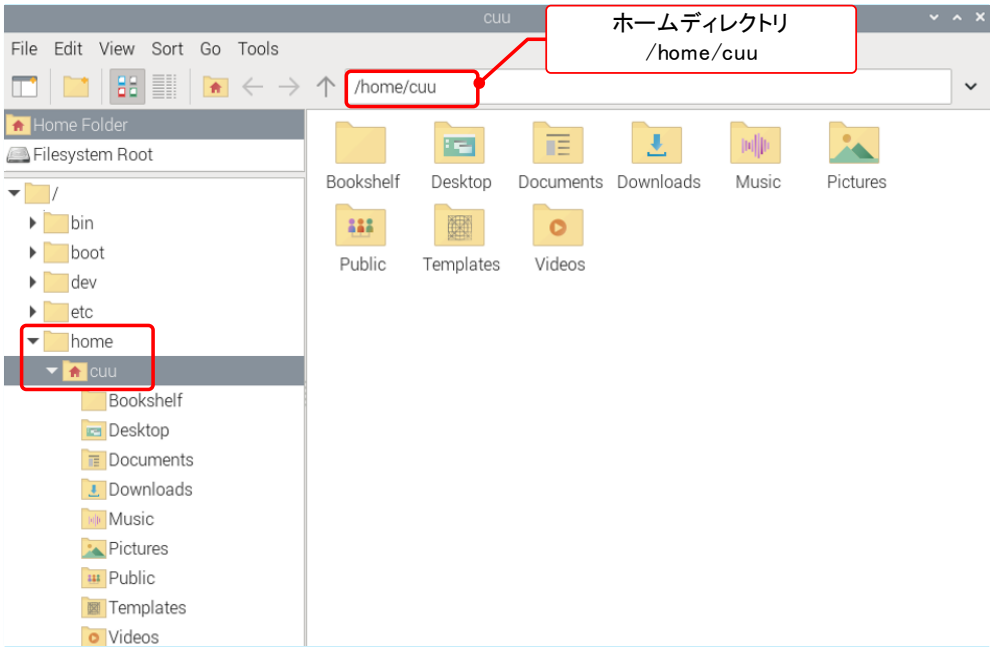
```
cuu@raspberrypi:~ $ pwd
/home/cuu
```

pwd コマンドは、「Print Working Directory」の略称で、現在の作業ディレクトリ（カレントディレクトリ）を表示します。この結果から、カレントディレクトリは、「/home/cuu」であることがわかります。

Raspberry Pi が起動したときの作業ディレクトリを「ホームディレクトリ」と呼びます。

Copyright © 2024 YITJC All Rights Reserved.

ファイルマネージャを起動してホームディレクトリの場所を確認します。

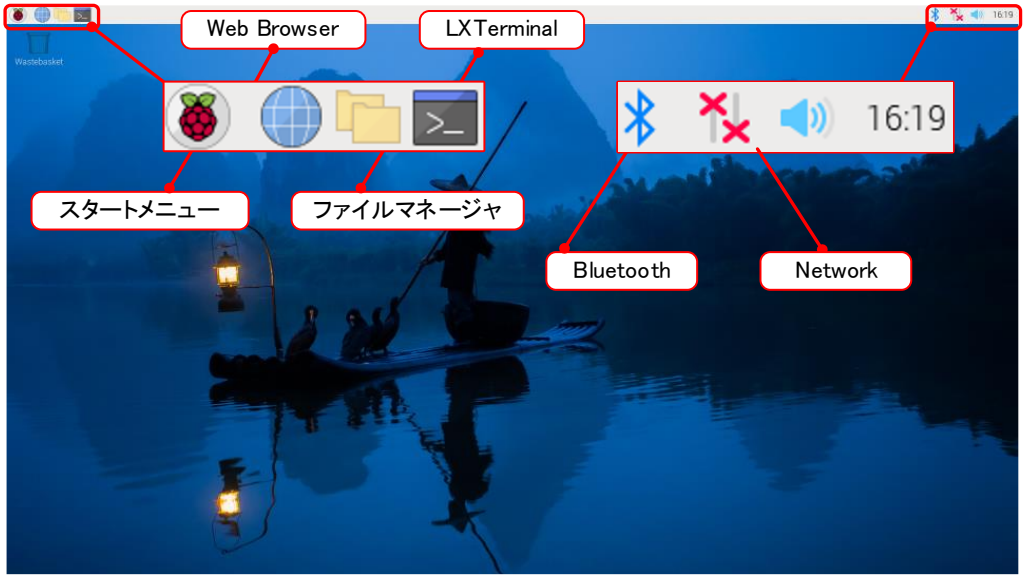


1.6 Raspberry Pi 初期設定

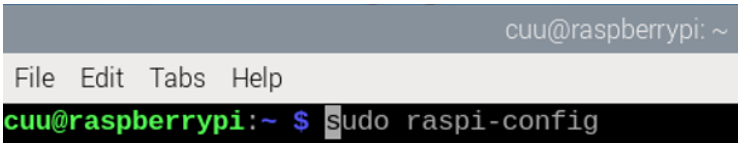
raspi-config コマンドを使用して初期設定を行います。なお、初期設定は GUI 操作で行います。

手順①～LXTerminal 起動

LXTerminal を起動して、raspi-config コマンドを実行してください。



cuu@raspberrypi: ~ $ sudo raspi-config



Copyright © 2024 YITJC All Rights Reserved.

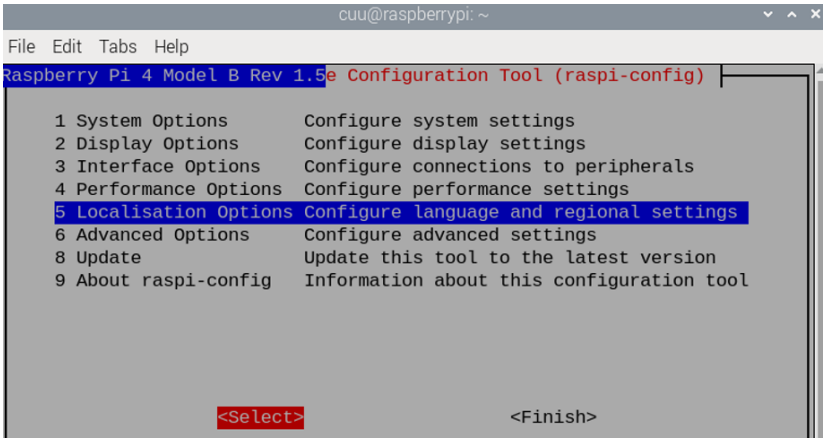
手順②～Local 設定

raspi-config 画面を操作して初期設定を行います。

「5.Localisation Options」を選択し、カーソルを「Select」に移動してから Enter キーを押してください。

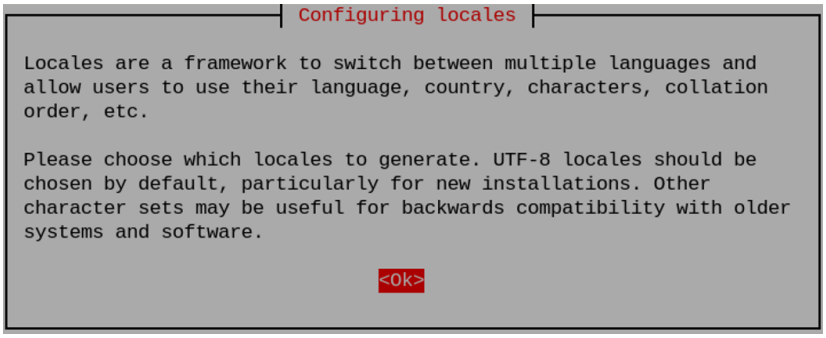
補足～Raspi-config 画面の操作～

Raspi-config 画面においては、「メニュー選択」は「→キー」、「カーソル移動」は「tab キー」、「決定」は「Enter キー」で操作します。



「L1 Locale」を選択してください。

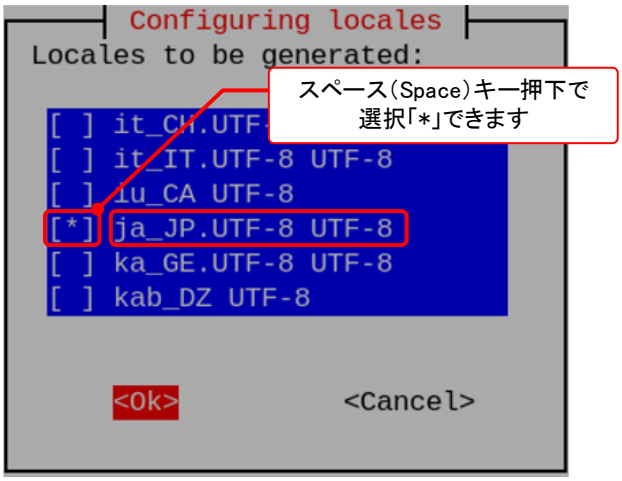
「OK」を選択し、決定（Enter キー）してください。



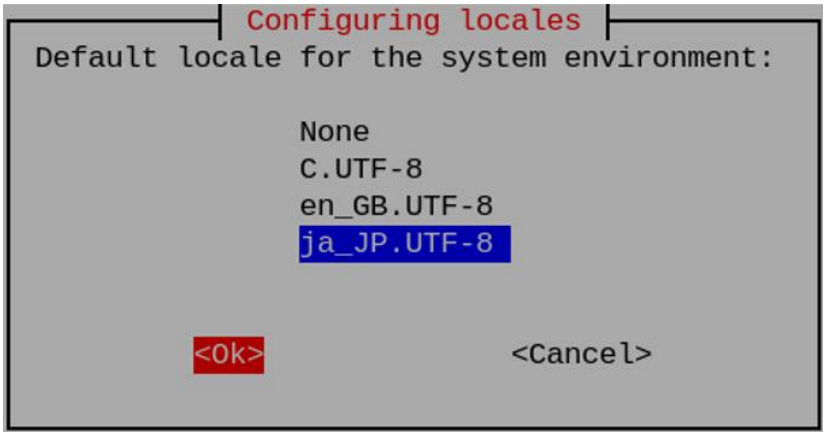
Copyright © 2024 YITJC All Rights Reserved.

「カーソルキー↓」を使用して「ja-JP.UTF-8 UTF-8」を選択してください。

※スペースキーを押すと、選択できます（*印が追加されます）。



「カーソルキー↓」を使用して「ja-JP.UTF-8」を選択してから、カーソルを移動して「OK」してください。



次の画面が表示され、新たな Locale が生成されたことを確認できます。

（この画面は短い時間で表示されるため、確認できない可能性があります。）

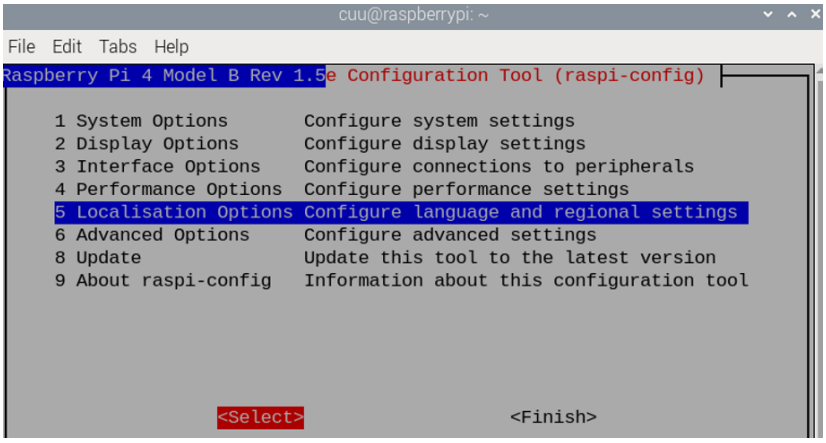
```
cuu@raspberrypi:~ $ sudo raspi-config
Generating locales (this might take a while)...
  en_GB.UTF-8... done
  ja_JP.UTF-8... done
Generation complete.
```

Copyright © 2024 YITJC All Rights Reserved.

手順③～Time Zone 設定～

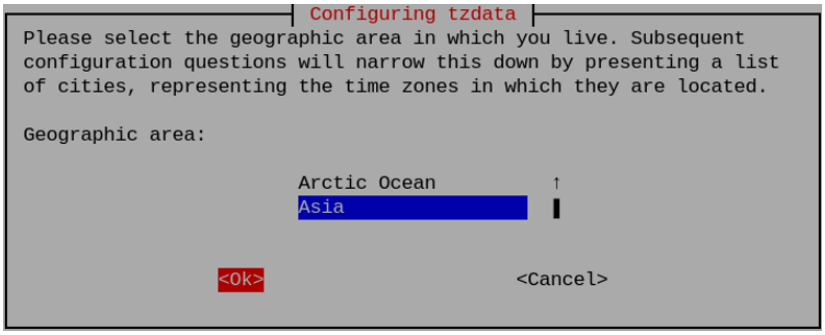
raspi-config 画面を操作して初期設定を行います。

「5.Localisation Options」を選択し、カーソルを「Select」に移動してから Enter キーを押してください。

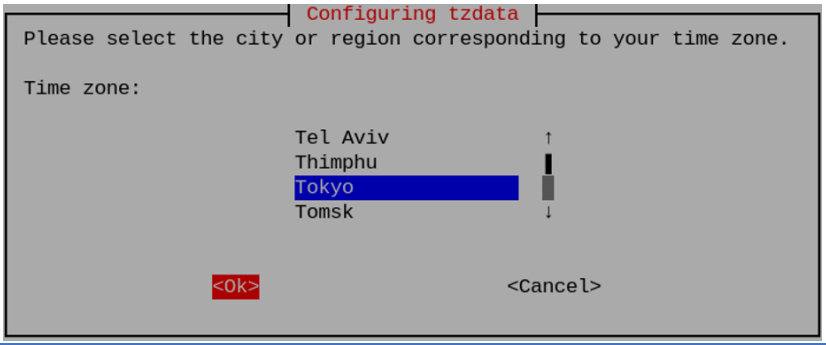


「L2 Time Zone」を選択してください。

「Asia」を選択してください。



「Tokyo」を選択してください。



Copyright © 2024 YITJC All Rights Reserved.

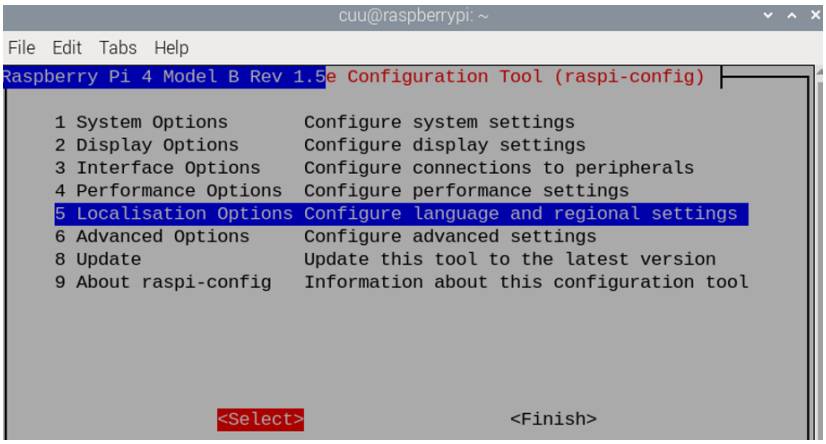
次の画面が表示され、Time Zone が東京（Tokyo）に変更されたことを確認できます。
（この画面は短い時間で表示されるため、確認できない可能性があります。）

```
Current default time zone: 'Asia/Tokyo'
Local time is now:      Sat Mar 16 01:43:40 JST 2024.
Universal Time is now:  Fri Mar 15 16:43:40 UTC 2024.
```

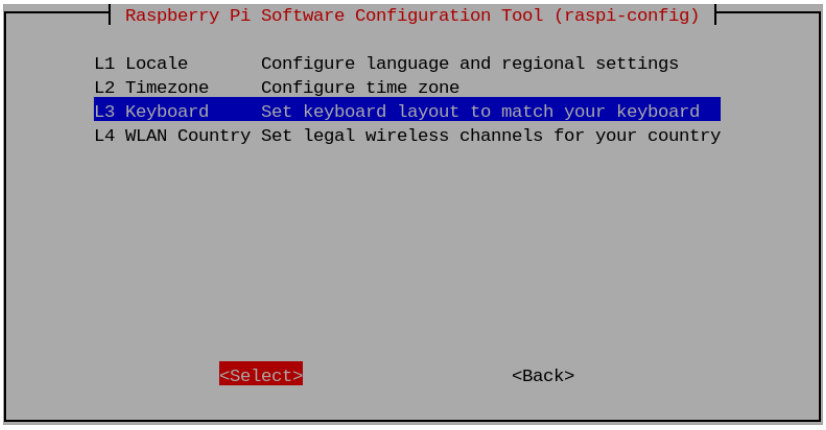
手順④～Keyboard 設定～

raspi-config 画面を操作して初期設定を行います。

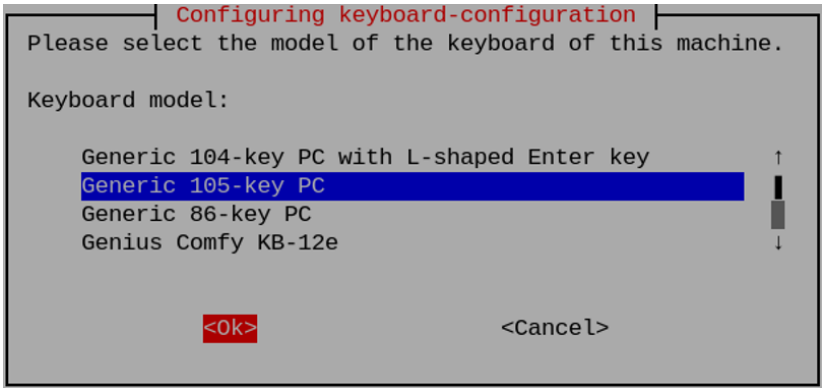
「5.Localisation Options」を選択し、カーソルを「Select」に移動してから Enter キーを押してください。



「L3 Keyboard」を選択してください。

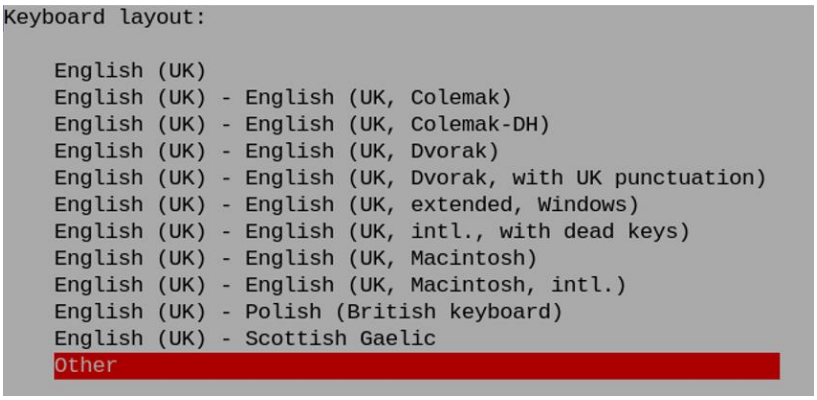


「Generic 105-key PC」を選択してください。

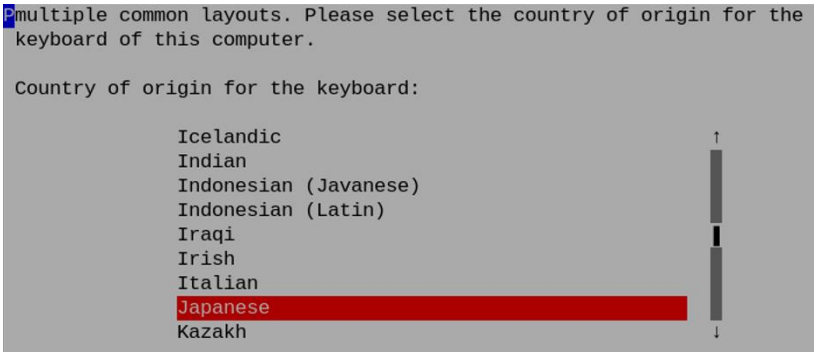


Copyright © 2024 YITJC All Rights Reserved.

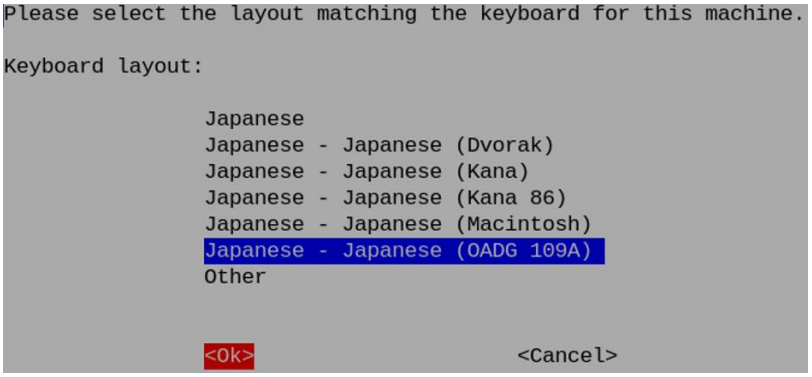
「Other」を選択してください。



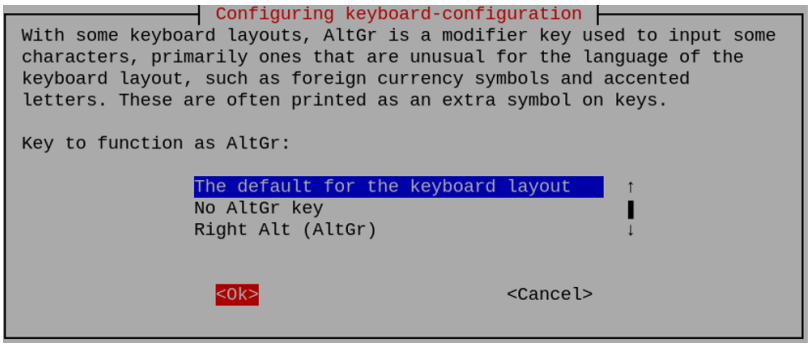
「Japanese」を選択してください。



「Japanese – Japanese (OADG 109A)」を選択してください。

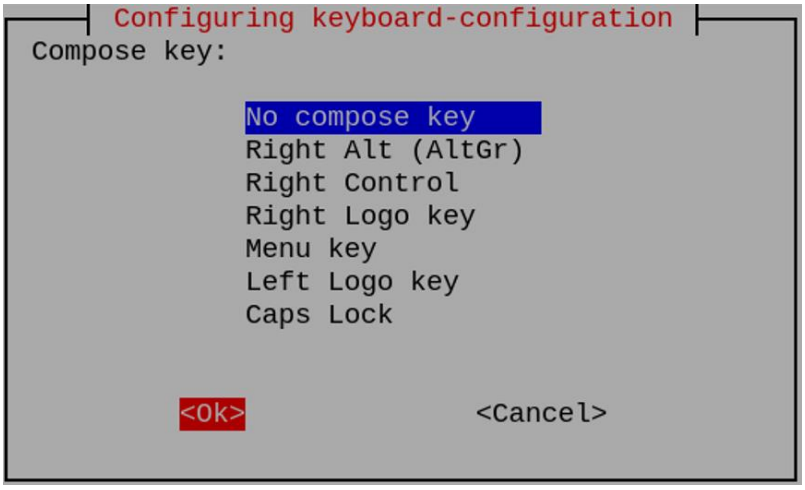


「Ther default for the keyboard layout」を選択してください。

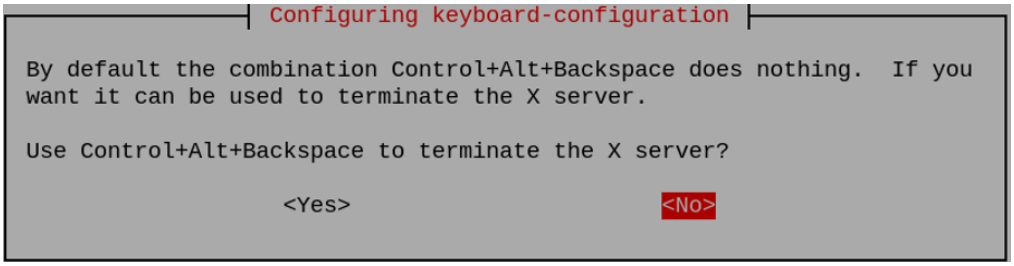


Copyright © 2024 YITJC All Rights Reserved.

「No compose key」を選択してください。

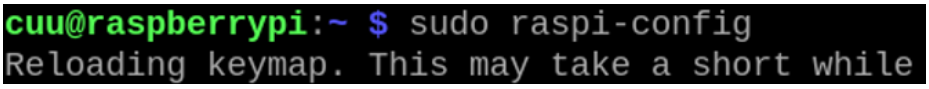


「No」を選択してください。



次の画面が表示され、新しい keymap（キーボード設定）が登録されたことを確認できます。

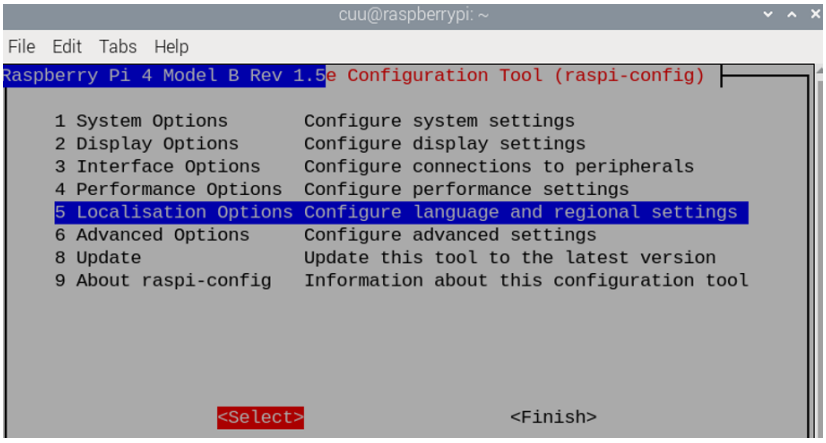
（この画面は短い時間で表示されるため、確認できない可能性があります。）



手順⑤～Wi-Fi 使用国設定～

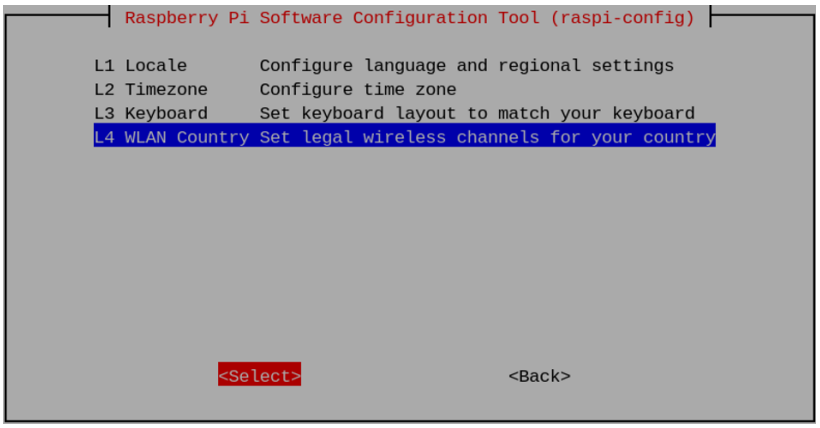
raspi-config 画面を操作して初期設定を行います。

「5.Localisation Options」を選択し、カーソルを「Select」に移動してから Enter キーを押してください。

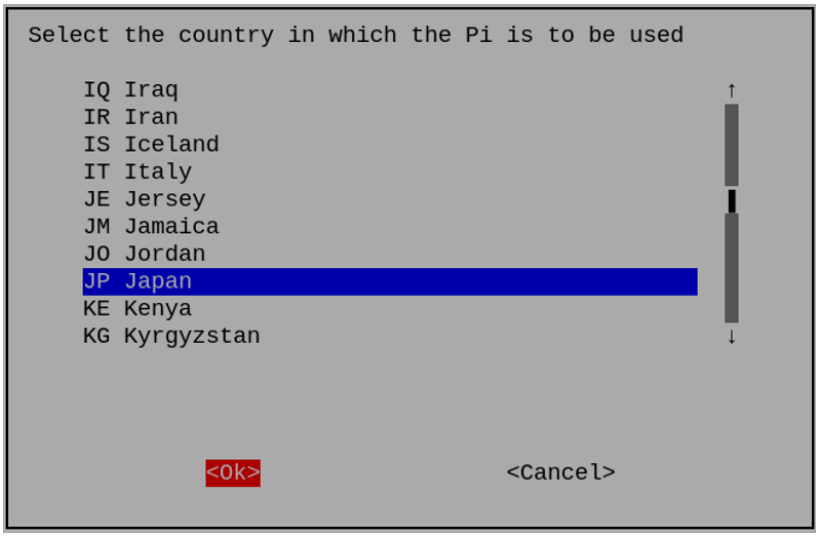


Copyright © 2024 YITJC All Rights Reserved.

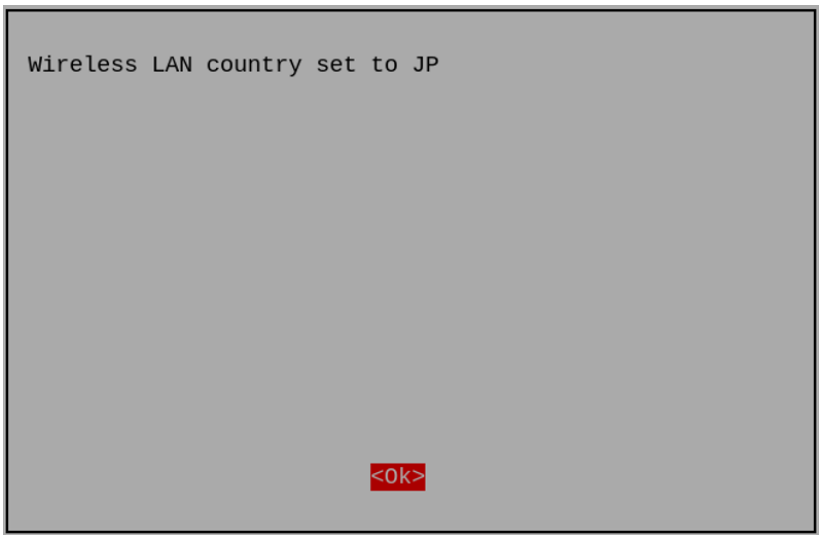
「L4 WLAN Country」を選択してください。



「JP Japan」を選択してください。



「OK」を選択してください。

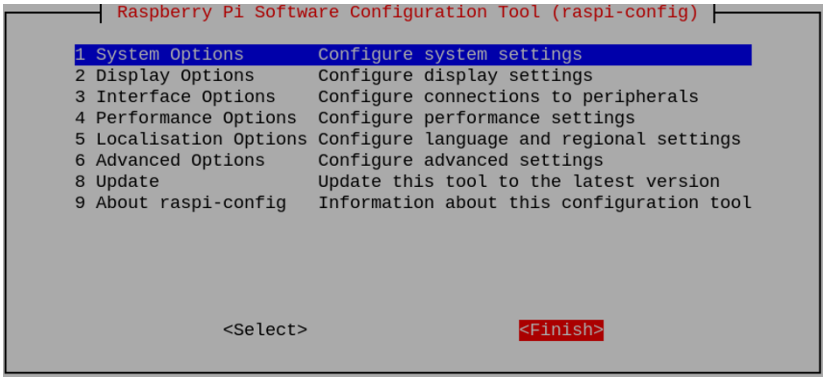


Copyright © 2024 YITJC All Rights Reserved.

手順⑥～再起動～

raspi-config 画面を終了します。

カーソルを「finish」に移動して、Enter キーを押してください。



LXTerminal で reboot コマンドを実行し、Raspberry Pi を再起動します。

cuu@raspberrypi: ~ $ sudo reboot

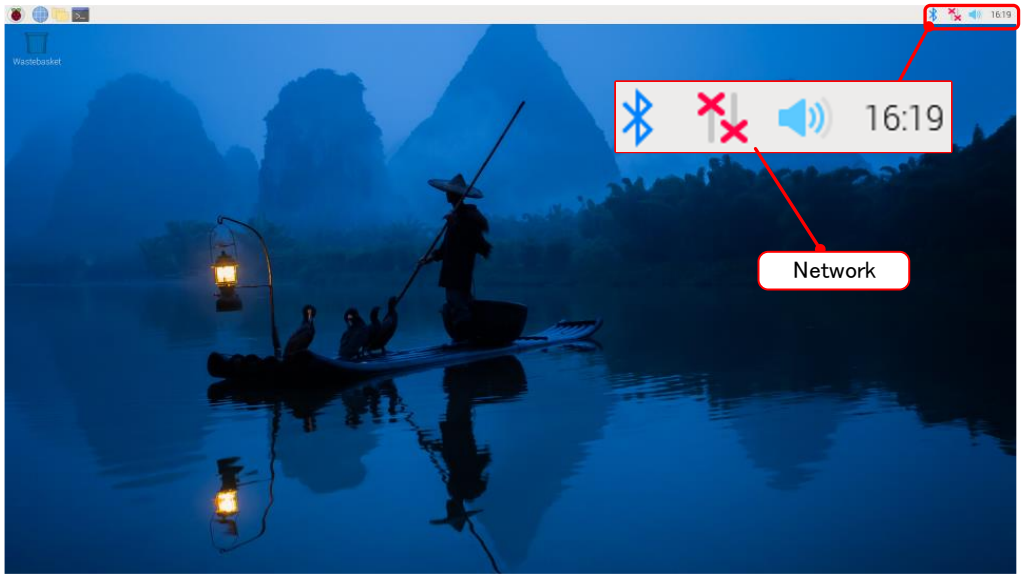
Copyright © 2024 YITJC All Rights Reserved.

1.7 無線 LAN 接続

GUI 操作で Raspberry Pi を無線 LAN（Wi-Fi）に接続します。

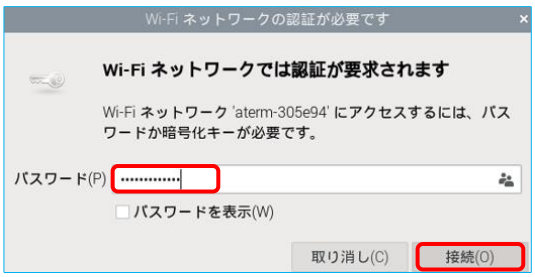
手順①～Wi-Fi への接続～

Network アイコンをクリックすると、Wi-Fi のアクセスポイント識別子（SSID）一覧が表示されます。



接続先となるアクセスポイント識別子（SSID）をクリックしてください。

パスワードを入力し、「接続」をクリックしてください。

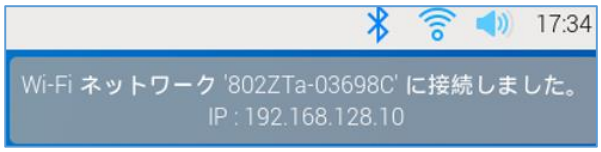


Copyright © 2024 YITJC All Rights Reserved.

画面右上に次のメッセージが表示され、Wi-Fi ネットワークに接続できたことが確認できます。

このとき表示された IP Address は使用するので、メモ帳などに記録してください。

IP Address:



補足～IP Address の確認～

IP Address を記録できなかった場合は、次のコマンドを実行して IP Address を確認してください。

cuu@raspberrypi: ~ $ ip addr

□で囲まれた部分が Raspberry Pi の IP Address です。

```
cuu@raspberrypi:~ $ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
ult qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN grou
p default qlen 1000
    link/ether d8:3a:dd:12:91:d3 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether d8:3a:dd:12:91:d4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.128.198/24 brd 192.168.128.255 scope global dynamic noprefixr
oute wlan0
       valid_lft 86317sec preferred_lft 86317sec
    inet6 fe80::d6ff:ffb5:c020:8ac/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

手順②～ネットワーク設定ファイル～

本講座では、SSH（Secure Shell）と呼ばれる遠隔操作法を使用して開発を進めていきます。具体的には、ノート PC から同じ無線 LAN（Wi-Fi）に接続した Raspberry Pi を遠隔操作して開発を進めます。この際、Raspberry Pi の IP Address を固定する必要があるため、その手順について説明します。

補足～Network サービスについて～

本講座で使用する Raspberry Pi OS（bookworm）からネットワークサービスのデフォルトが「NetworkManager」に変更されました。少し前までは、Raspi-config の設定で DHCP（Dynamic Host Configuration Protocol）を選択することもできましたが、2024 年 4 月時点では NetworkManager がデフォルトサービスに設定されています。

Copyright © 2024 YITJC All Rights Reserved.

cd（change directory）コマンドを実行して、NetworkManager の設定ファイル（Key File 形式）が保存されているディレクトリに移動します。

cuu@raspberrypi: ~ $ cd /etc/NetworkManager/system-connections

cd コマンドの実行により、現在のディレクトリが「/etc/NetworkManager/system-connections」に移動したことがわかります。

```
cuu@raspberrypi:~ $ cd /etc/NetworkManager/system-connections
cuu@raspberrypi:/etc/NetworkManager/system-connections $
```

次に、ls（list）コマンドを実行して、カレントディレクトリにあるファイルやフォルダを確認します。

cuu@raspberrypi:/etc/NetworkManager/system-connections $ ls

```
cuu@raspberrypi:/etc/NetworkManager/system-connections $ ls
802ZTa-03698C.nmconnection  aterm-305e94.nmconnection
```

手順①で接続した Wi-Fi ネットワークの設定ファイルが保存されていることがわかります。
ファイル名は、「ssid 名.nmconnection」です。

手順③～IP Address 固定～

IP Address を固定するためには、手順②で確認した設定ファイルを編集して上書き保存する必要があります。

まず、nano コマンドを実行して設定ファイルを編集します。

nano は、Raspberry Pi にインストールされているテキストエディタです。

cuu@raspberrypi:/etc/NetworkManager/system-connections $ sudo nano 802ZTa-03698C.nmconnection

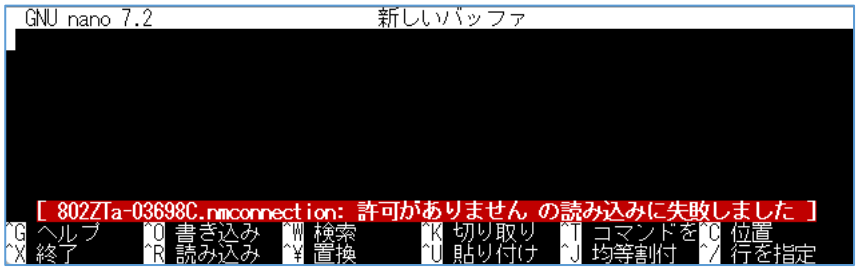
802ZTa-03698C の箇所は、接続した ssid 名に変更してください。

補足～sudo コマンド～

sudo コマンドは、「Super User（管理者）が Do（実行する）」という意味のコマンドです。

手順②で確認した設定ファイルを編集するためには管理者権限が必要になります。そのため、コマンドの冒頭に「sudo」を追加して実行しています。

仮に、「sudo」を追加せず、「nano 802ZTa-03698C.nmconnection」と実行した場合は、編集権限がないと注意されます（次の画面を参照）。



Copyright © 2024 YITJC All Rights Reserved.

nano を使用して、/etc/ディレクトリにある「802ZTa-03698C.nmconnection」ファイルを編集します。

ファイルに赤字の部分を編集（追加）してください。

（緑字・黒字部分は、最初から記載されている内容です。

#はコメントを意味していて、補足の説明を追加しています。

コンピュータ（Raspberry pi）は、#コメントを無視します。

```
[connection]
id=802ZTa-03698C                                    #接続先 ssid
uuid=651b2677-41f7-4254-a853-6e09f1907a48           #Universally Unique Identifier
type=wifi
interface-name=wlan0

[wifi]
mode=infrastructure
ssid=802ZTa-03698C

[wifi-security]
auth-alg=open                                       #Application layer gateway
key-mgmt=wpa-psk                                    #Key management（暗号化 key 管理方式）
psk=9a6fec935633e                                   #Pre shared key（ssid パスワード）

[ipv4]
address1=192.168.128.10/24,192.168.128.1            #設定したい IP Address, Router の IP Address
dns=8.8.8.8;                                        #Google Public DNS
method=manual                                       #IP Address 手動（manual）設定

[ipv6]
Addr-gen-mode=default
method=auto

[proxy]
```

補足～編集内容の説明～

- address1=192.168.128.10/24,192.168.128.1

  address1=（Raspberry Pi の IP アドレス/24),（デフォルトゲートウェイ）

  192.168.128 の部分は、手順①補足で確認した IP Address と同じ値を使用してください。

  10 の部分は、固定したい値（IP Address）を設定してください。

  デフォルトゲートウェイは、通常 1 番が使用されるケースが多いですが、調べる場合は、Windows PC のコマンドプロンプトで「ipconfig」コマンドを実行して確認してください。

- dns=8.8.8.8;

この部分は、テキストと同じ設定にしてください。

Google が運営しているパブリック DNS（Domain Name System）を設定しています。

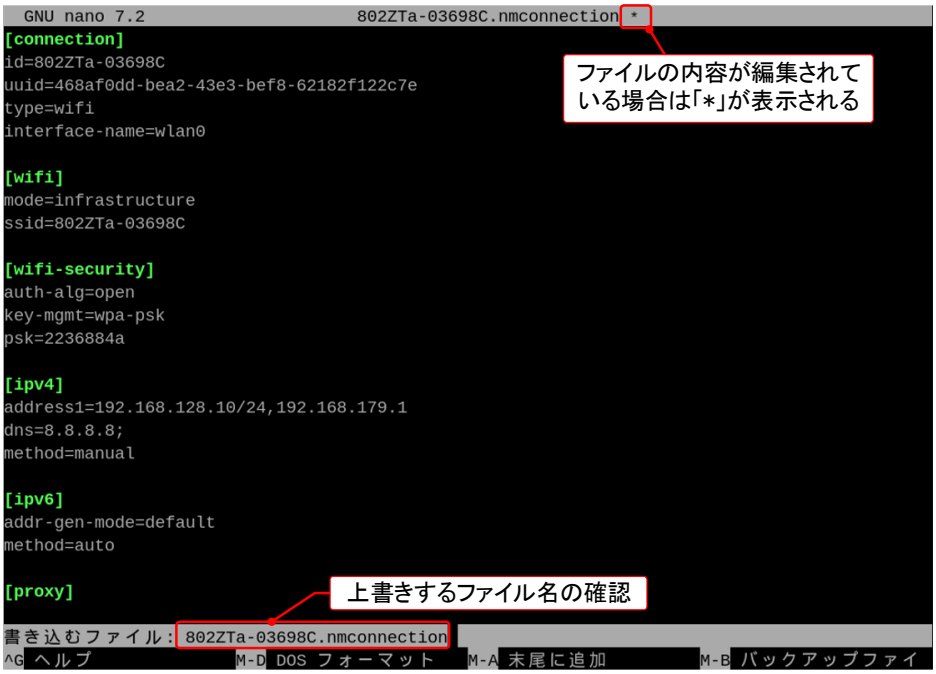
- method=manual

この部分は、「manual」を設定して IP Address を手動で設定できるようにしています。

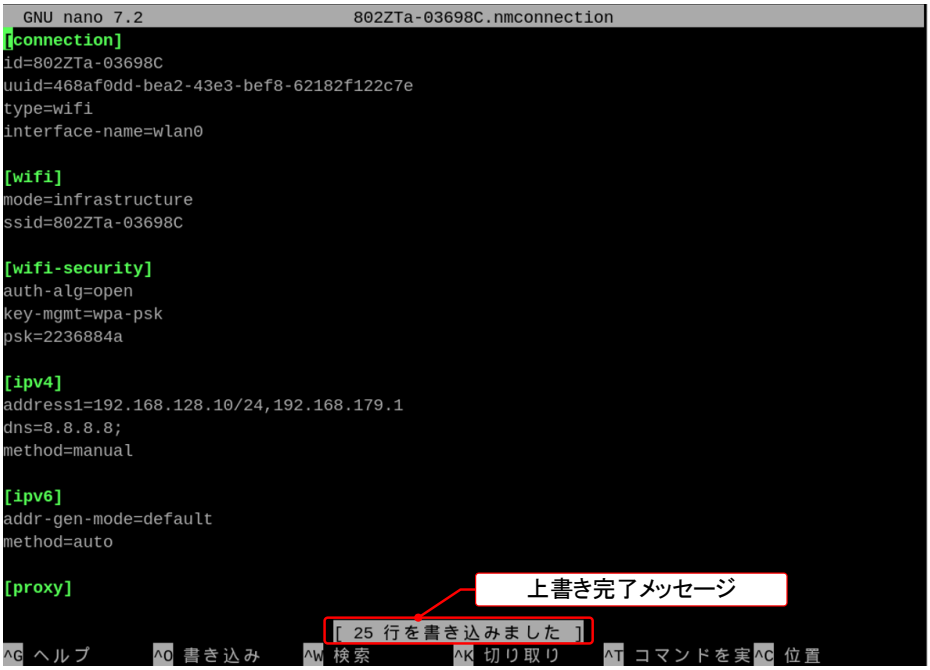
Copyright © 2024 YITJC All Rights Reserved.

ファイルの編集が終了したら、「ctrl + O（オー）」キーを押してください。

画面下部に表示されたファイル名に間違いがなければ、「Enter」を押して上書きを実行してください。



上書きを実行すると、画面下部に上書きが完了したメッセージが表示されます。



最後に、「Ctrl + X」キーを押して、元の画面に戻ります。

補足～Raspberry Pi 内部のファイル編集手順～

Raspberry Pi 内部のファイルを編集する機会が多いので、編集手順を整理しておきます。

① 「sudo nano ファイル名」コマンドを実行

② 編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

③ 「Enter」キーを押下して上書き保存を実行

④ 「Ctrl + X」キーを押して元の画面に戻る

Copyright © 2024 YITJC All Rights Reserved.

手順④～設定内容の反映～

次のコマンドを実行して手順③で設定した内容を反映させます。

cuu@raspberrypi: ~ $ sudo nmcli connection reload

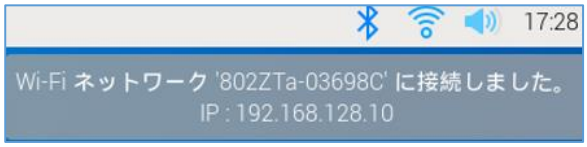
cuu@raspberrypi: ~ $ sudo nmcli connection up 802ZTa-03698C

```
cuu@raspberrypi:~ $ sudo nmcli connection reload
cuu@raspberrypi:~ $ sudo nmcli connection up 802ZTa-03698C
接続が正常にアクティベートされました (D-Bus アクティブパス: /org/freedesktop/NetworkManager/
ActiveConnection/4)
```

802ZTa-03698C の部分は、接続する Wi-Fi ネットワークの ssid を設定してください。

コマンドを実行してから IP Address の変更が反映されるまで、少し時間が掛かります。

IP Address が変更されると、画面左上に新たに設定した IP Address が表示されます。



補足～nmcli コマンド～

nmcli コマンドは、「NetworkManager Command Line Interface」の略称です。

コマンドを実行することで、NetworkManager の制御や情報の表示することができます。

例えば、次のコマンドを実行するとネットワークの接続状況を確認できます。

cuu@raspberrypi: ~ $ nmcli device status

```
cuu@raspberrypi:~ $ nmcli device status
DEVICE         TYPE      STATE           CONNECTION
wlan0          wifi      接続済み        802ZTa-03698C
lo             loopback  接続済み (外部) lo
p2p-dev-wlan0  wifi-p2p  切断済み        --
eth0           ethernet  利用不可        --
```

IP Address の変更表示を確認できなかった場合は、次のいずれかのコマンドを実行して IP Address が変更されていることを確認してください。

cuu@raspberrypi: ~ $ ip addr

```
cuu@raspberrypi:~ $ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen
 1000
    link/ether d8:3a:dd:12:91:d3 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
 qlen 1000
    link/ether d8:3a:dd:12:91:d4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.128.10/24 brd 192.168.128.255 scope global noprefixroute wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::d6ff:ffb5:c020:8ac/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Copyright © 2024 YITJC All Rights Reserved.

cuu@raspberrypi: ~ $ ifconfig

```
cuu@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d8:3a:dd:12:91:d3  txqueuelen 1000  (イーサネット)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (ローカルループバック)
        RX packets 211  bytes 18219 (17.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 211  bytes 18219 (17.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.128.10  netmask 255.255.255.0  broadcast 192.168.128.255
        inet6 fe80::d6ff:ffb5:c020:8ac  prefixlen 64  scopeid 0x20<link>
        ether d8:3a:dd:12:91:d4  txqueuelen 1000  (イーサネット)
        RX packets 122  bytes 113872 (111.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 252  bytes 28555 (27.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Copyright © 2024 YITJC All Rights Reserved.

1.8 Raspberry Pi 操作方法（CUI）

Raspberry Pi の操作方法「CUI」を紹介します。CUI（Character User Interface）は、Raspberry Pi を文字（Character）だけで操作する方法です。CUI は、CLI（Command Line Interface）と呼ばれることもあります。

本講座は、これ以降 CUI 操作で Raspberry Pi の開発を進めます。CUI 操作で開発を進めるため、2つのソフトウェアを使用します。いずれも SSH（Secure Shell）と呼ばれる遠隔操作法を使用して Raspberry Pi を遠隔操作します。

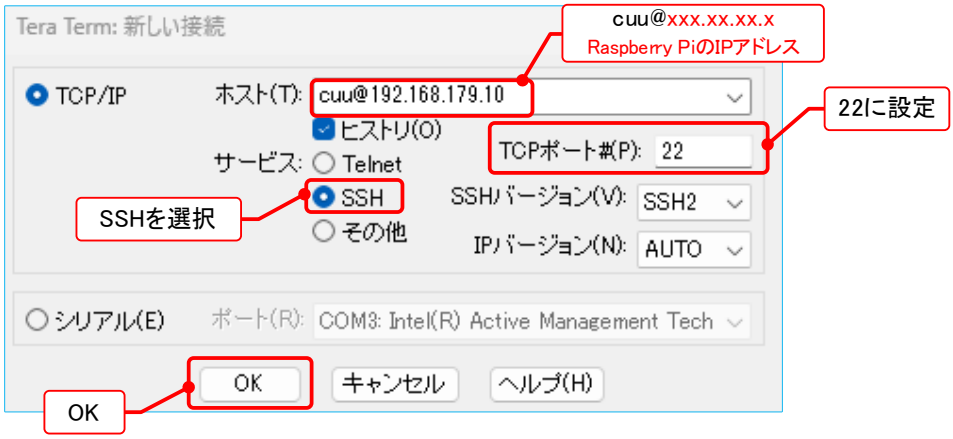
手順①～Tera Term～

Tera Term は、Tera Term Project が開発・公開しているターミナルエミュレータです。Windows PC からコマンドライン方式で Raspberry Pi を遠隔操作します。

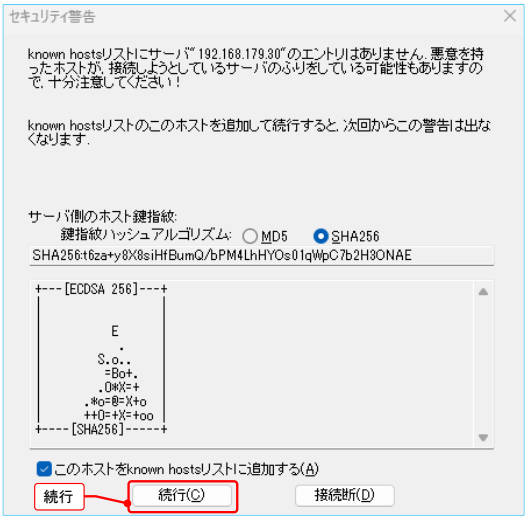
まず、Tera Term を起動します。

1.7「無線 LAN 接続」で設定した IP Address を使用して SSH 接続します。

ホストは、ユーザー名（cuu）@IP Address の形で入力します。入力が終了したら、「OK」をクリックしてください。

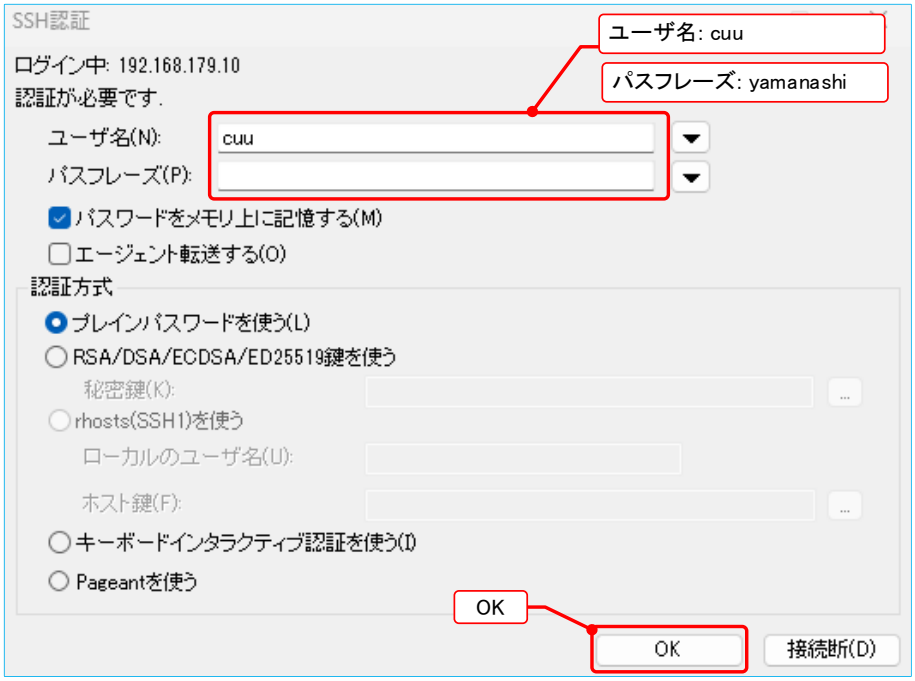


初めて接続する際はセキュリティ警告のダイアログが表示されますが、「続行」ボタンをクリックしてください。

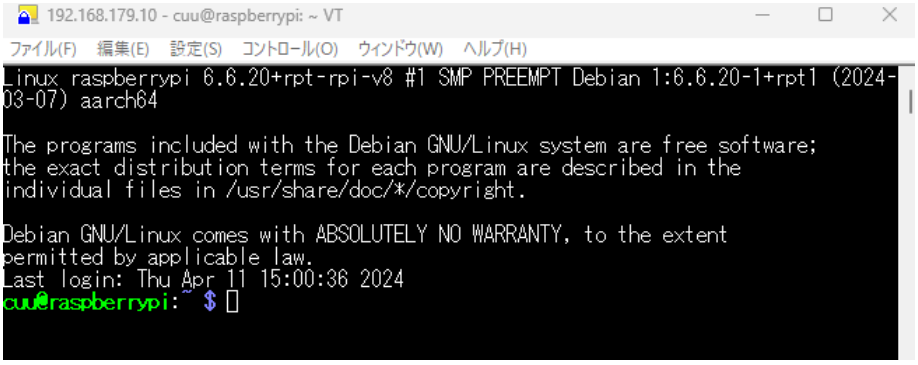


Copyright © 2024 YITJC All Rights Reserved.

ユーザー名を確認し、パスワードを入力してから「OK」をクリックしてください。



次の画面が表示されれば、SSH 接続成功です。



Tera Term の画面で次のコマンドを実行し、Raspberry Pi の CUI 操作ができることを確認してください。
コマンド（緑字部分）をコピーした後、Tera Term 画面上で「右クリック」するとコマンドを貼り付けできます。

cuu@raspberrypi: ~ $ cat /etc/os-release

補足～cat コマンド～

cat コマンドは、本来複数のファイルを連結（conCATenate）するために使用するコマンドですが、ファイルの中身を表示させる場合に使用することが多いコマンドです。

上記コマンドは、「/etc」ディレクトリにある os-release ファイルの中身を表示させています。OS のコードネームが「bookworm」であることを確認できます。

Copyright © 2024 YITJC All Rights Reserved.

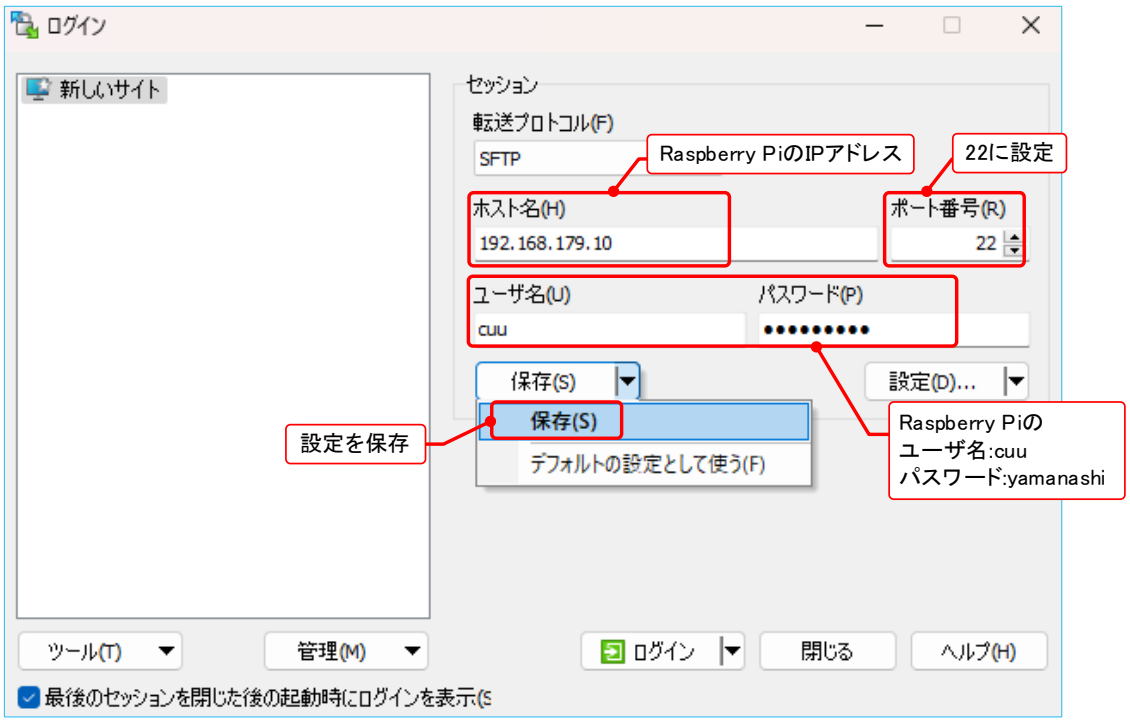
手順②～WinSCP～

Windows PC から WinSCP を使用して Raspberry Pi に SSH 接続します。

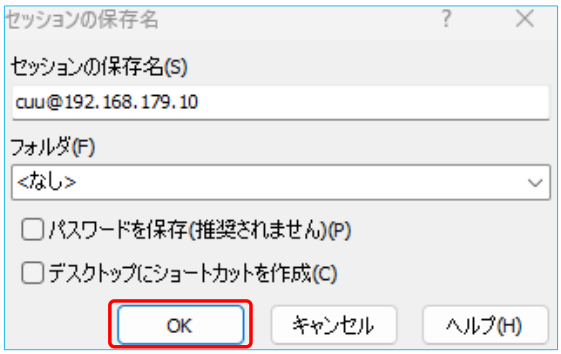
まず、WinSCP を起動してください。

1.7「無線 LAN 接続」で設定した IP Address をホスト名として入力してください。

ユーザー名（cuu）、パスワード（yamanashi）を入力した後、「保存」をクリックして設定を保存してください。

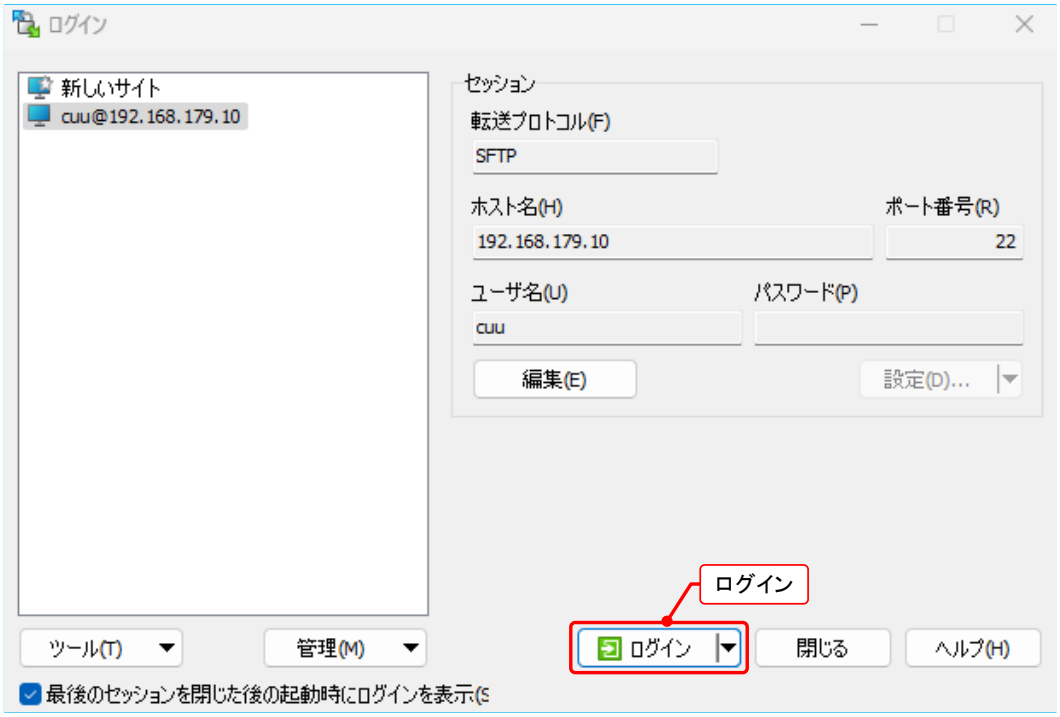


保存内容を確認して「OK」をクリックしてください。

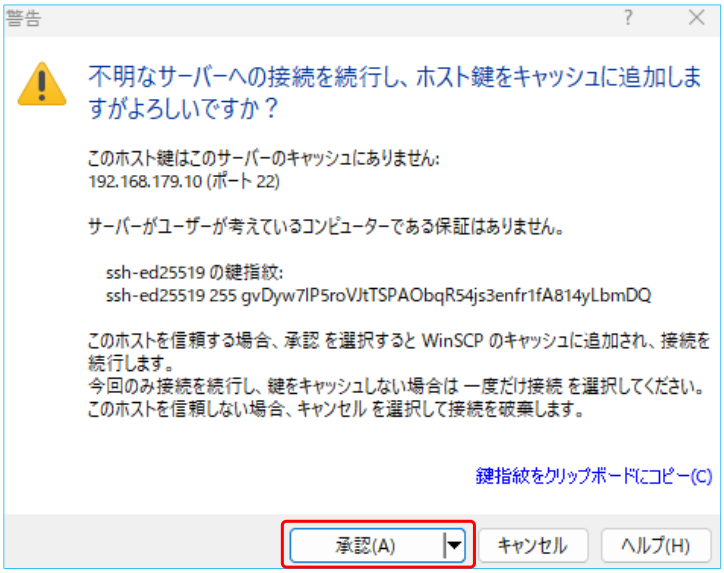


Copyright © 2024 YITJC All Rights Reserved.

「ログイン」をクリックしてください。

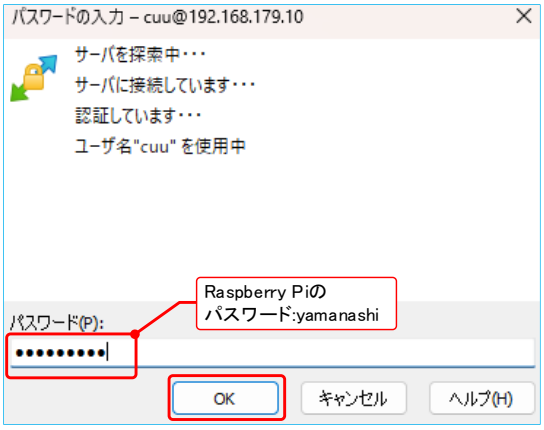


初めてログインする際は警告が表示されます。「承認」をクリックすると、鍵情報が保存されて、次回から警告が表示されません。

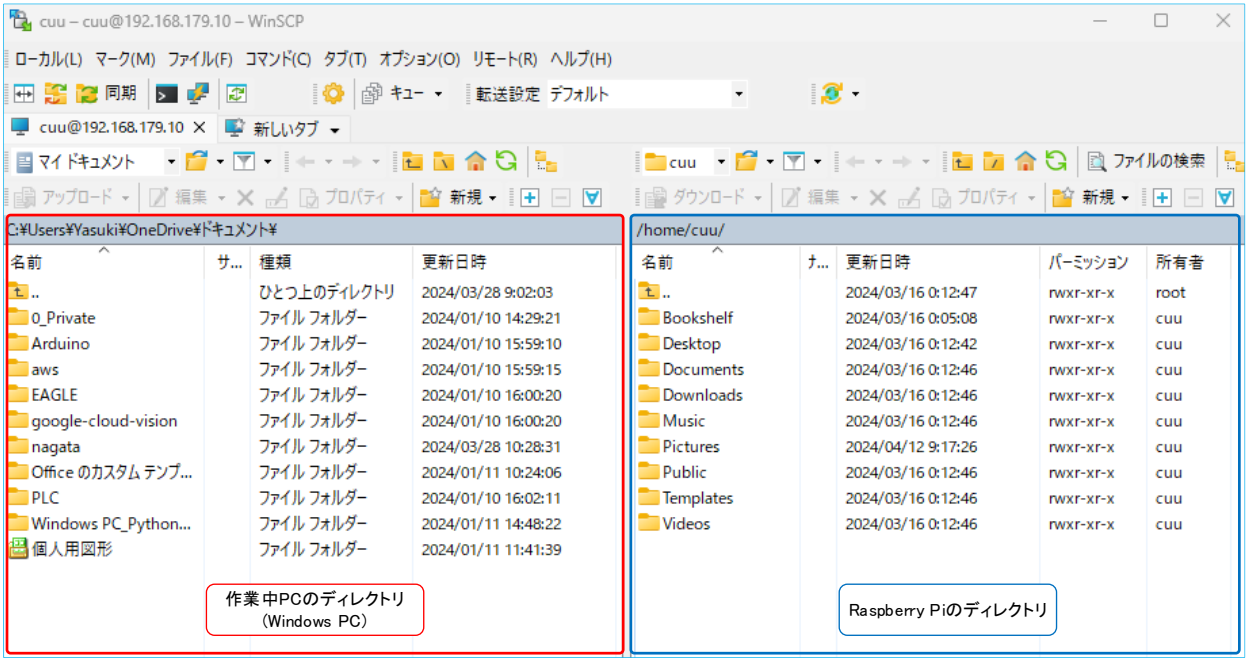


Copyright © 2024 YITJC All Rights Reserved.

再びパスワードを入力して「OK」をクリックしてください。

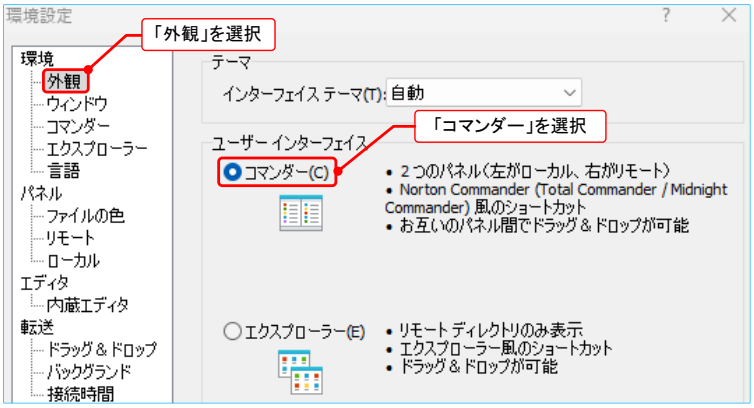


ログインに成功すると、画面左側に Windows PC のディレクトリ、右側に Raspberry Pi のディレクトリが表示されます。これにより、Windows PC と Raspberry Pi との間でファイルのやり取りを「ドラッグ&ドロップ」で行うことができます。



～参考 WinSCP の画面設定～

WinSCP にログインした際、Raspberry Pi 側のディレクトリしか見られない場合は、「オプション」-「環境設定」から画面の設定を変更できます。



Copyright © 2024 YITJC All Rights Reserved.

1.9 ディレクトリの作成

Raspberry Pi に Python プログラムを保存するディレクトリを作成します。

mkdir コマンドは、「Make Directory」の略称で、ディレクトリ（フォルダ）を作成するという意味です。

ホームディレクトリから次のコマンドを実行します。

```
cuu@raspberrypi: ~ $ mkdir dev
```

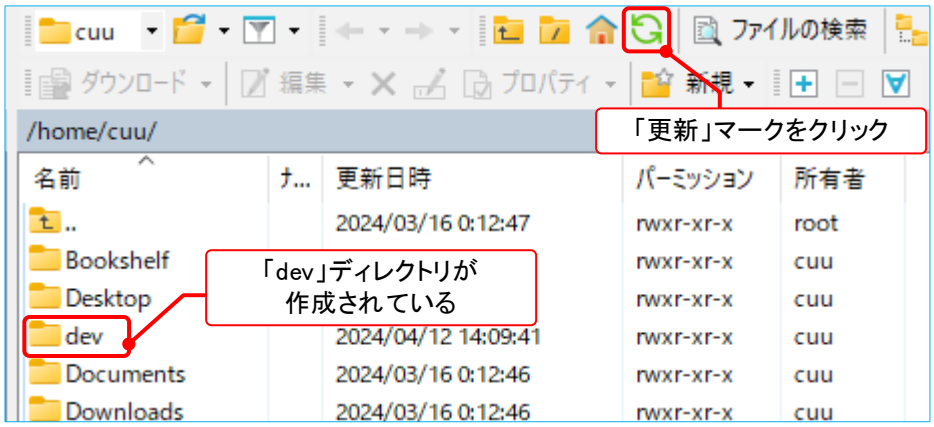
続いて、dev ディレクトリに移動します。

cd コマンドは、「Change Directory」の略称で、カレントディレクトリを移動するという意味です。

```
cuu@raspberrypi: ~ $ cd dev
cuu@raspberrypi: ~/dev $
```

ディレクトリを移動すると、カレントディレクトリが「~/dev」に変化したことがわかります。

WinSCP の画面上でも確認してみます。



本講座で使用するするプログラムなどは、「dev」ディレクトリに保存します。

Copyright © 2024 YITJC All Rights Reserved.

1.10 APT ライブラリの利用

Raspberry Pi はさまざまなソフトウェアがインストールされています。ここでは、パッケージ管理システム APT（Advanced Package Tool）ライブラリを利用して Raspberry Pi にインストールされているパッケージを更新します。

手順①～apt-get update～

cuu@raspberrypi: ~ $ sudo apt-get update

インストール可能なパッケージの「一覧」を更新します。

パッケージのインストール、アップグレードなどは行われません。

```
cuu@raspberrypi: ~ $ sudo apt-get update
ヒット:1 http://deb.debian.org/debian bookworm InRelease
取得:2 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
取得:3 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
取得:4 http://deb.debian.org/debian-security bookworm-security/main arm64 Packages [147 kB]
取得:5 http://deb.debian.org/debian-security bookworm-security/main armhf Packages [144 kB]
取得:6 http://archive.raspberrypi.com/debian bookworm InRelease [23.6 kB]
取得:7 http://archive.raspberrypi.com/debian bookworm/main armhf Packages [384 kB]
取得:8 http://archive.raspberrypi.com/debian bookworm/main arm64 Packages [374 kB]
1,177 kB を 10秒 で取得しました (123 kB/s)
パッケージリストを読み込んでいます... 完了
```

手順②～apt-get upgrade～

cuu@raspberrypi: ~ $ sudo apt-get upgrade

Raspberry Pi にインストール済みのパッケージ更新を行い、新しいバージョンにアップグレードします。

「有効なパッケージ一覧」を元に実行されるので、 apt-get update と組み合わせて実行します。

コマンドを実行すると、アップグレードの続行確認がされます。

「y」と入力し、「Enter」を押して続行してください。

```
cuu@raspberrypi: ~ $ sudo apt-get upgrade
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
アップグレードパッケージを検出しています... 完了
以下のパッケージは保留されます:
  rpi-eeprom
以下のパッケージはアップグレードされます:
  arandr bsdextrautils bsdutils eject fdisk firefox gui-pkinst libblkid1 libfdisk1 libmount1 libsmartcols1 libuuid1 libwf-utils0 mount
_pi-bluetooth piclone pipanel piwiz raspberrypi-ui-mods raspi-utils rc-gui rfkill rp-prefapps util-linux util-linux-extra wayfire
アップグレード: 26 個、新規インストール: 0 個、削除: 0 個、保留: 1 個。
67.6 MB のアーカイブを取得する必要があります。
この操作後に追加で 1,221 kB のディスク容量が消費されます。
続行しますか? [Y/n]
```

アップグレードが必要なパッケージの数が多い場合、パッケージの更新に長時間掛かることがあります。

Copyright © 2024 YITJC All Rights Reserved.

1.11 Python バージョン

プログラミング言語は「Python」を使用します。

まず、Raspberry Pi にインストールされている Python のバージョンを確認します。

```
cuu@raspberrypi: ~ $ python -V
Python 3.11.2
cuu@raspberrypi: ~ $ python3 -V
Python 3.11.2
```

続いて、インストール先を確認すると、別々のディレクトリに同じバージョンの Python がインストールされていることがわかります。

```
cuu@raspberrypi: ~ $ which python
/usr/bin/python
cuu@raspberrypi: ~ $ which python3
/usr/bin/python3
```

参考～Python2 系の廃止～

Raspbian は 2012 年のリリース以来 Python2 系と Python3 系が共存していました。そのため、Python2 系は「/usr/bin/python」、Python3 系は「usr/bin/python3」とインストール先が区別されていました。

しかし、Python2 系は 2020 年 4 月に最後のリリースが行われ、現在は Google や AWS など、さまざまなクラウドサービスで Python2 系のサポートが終了しています。

Raspberry Pi OS も 2021 年にリリースされた OS 「Bullseye」から Python2 系が廃止され、Python3 系がデフォルトの Python となったため、現在は同じバージョンの Python が 2 つのディレクトリにインストールされています。

Raspberry Pi で Python プログラムを実行する方法は、2 つあります。

「python」コマンドで実行する場合は、「/usr/bin/python」にインストールされている Python を使用します。

```
cuu@raspberrypi:~ $ python  プログラム.py
```

「python3」コマンドで実行する場合は、「/usr/bin/python3」にインストールされている Python を使用します。

Python3.11.2 で Python プログラムを実行する場合は、python3 と指定する必要があります。

```
cuu@raspberrypi:~ $ python3  プログラム.py
```

いずれのコマンドも同じバージョンの Python で実行されますが、コマンド指定方法「python」、「python3」の違いによって、実行される Python の環境（プログラムを実行する Python のインストール先）が異なるということを理解しておいてください。

Copyright © 2024 YITJC All Rights Reserved.

1.12 Raspberry Pi シャットダウン

Raspberry Pi には電源スイッチがありません。電源を供給している USB Type-C ケーブルを抜けば強制的に電源を OFF させることができますが、好ましくありません。MicroSD Card のデータ破損や故障が発生する可能性があります。Raspberry Pi は OS が稼働している「コンピュータ」であるため、電源を OFF する際は、正しい方法でシャットダウンする必要があります。

手順①～シャットダウンコマンド実行～

電源を OFF するときのシャットダウンコマンドが用意されています。

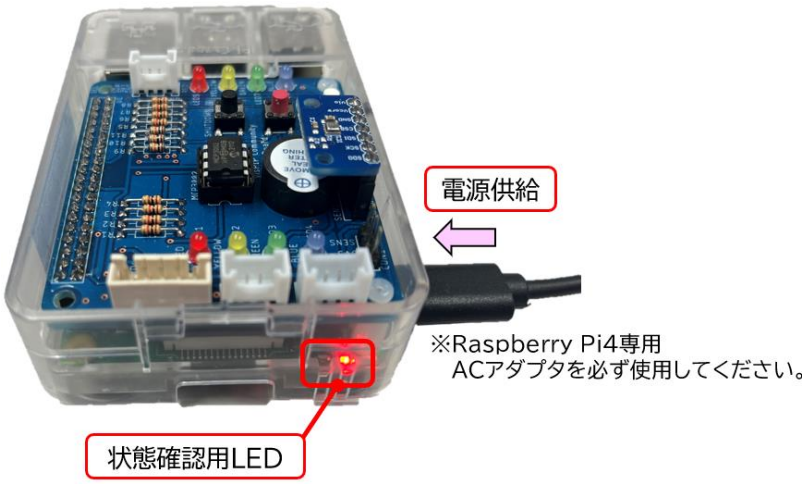
cuu@raspberrypi: ~ $ sudo shutdown -h now

または

cuu@raspberrypi: ~ $ sudo halt

手順②～シャットダウン確認～

シャットダウンコマンドを実行後、しばらくすると状態確認用の緑 LED（MicroSD Card へアクセスする際に点滅）が消灯します。完全に消灯したことを確認してから、USB Type-C ケーブルを抜いてください。



Copyright © 2024 YITJC All Rights Reserved.

1.13 バックアップファイル作成

Raspberry Pi の開発環境やデータは、すべて MicroSD Card に保存されています。一度開発した Raspberry Pi の複製や、万が一 MicroSD Card のデータが消失してしまった場合に備えて、バックアップファイルを作成する手順を説明します。

手順①～シャットダウン～

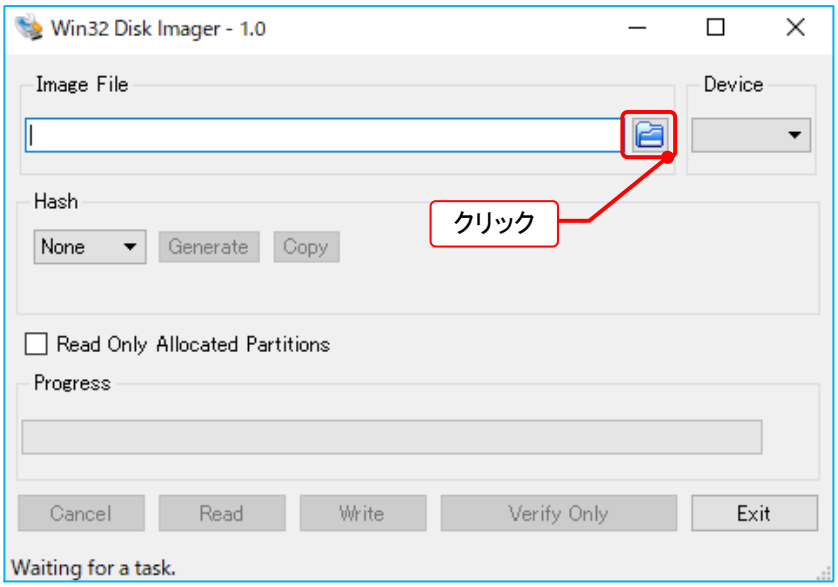
Raspberry Pi をシャットダウンし、電源ケーブル（USB Type-C ケーブル）を抜いてください。

手順②～Windows PC に MicroSD Card 装着～

カードリーダなどを利用して、Raspberry Pi から外した MicroSD Card を Windows PC に装着してください。

手順③～Win32DiskImager を起動～

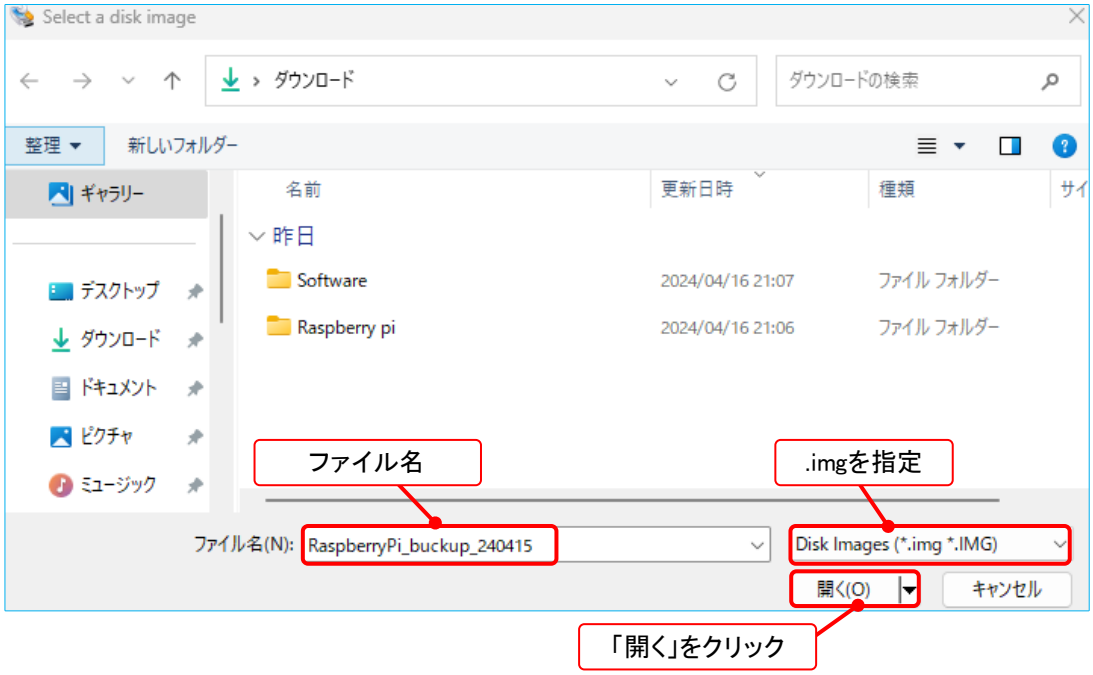
「Win32DiskImager」を起動し、フォルダマークをクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

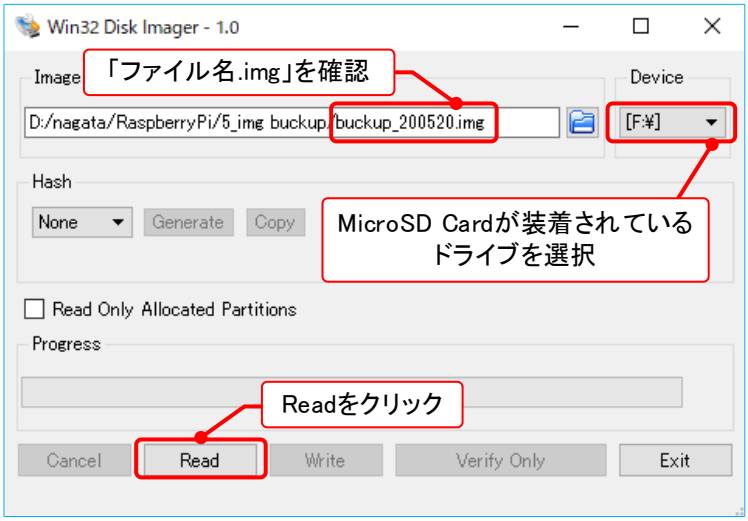
手順④～img ファイル作成～

「.img ファイル形式」を指定してファイルを作成してください。



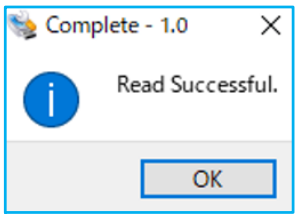
手順⑤～img ファイル Read～

Micro SD Card のデータを img（イメージ）ファイルとして読み込み、バックアップファイルを作成してください。



「Read Successful」と表示されれば、バックアップファイルの作成は完了です。

保存先ディレクトリへ移動して「ファイル名.img」ファイルが保存されていることを確認してください。



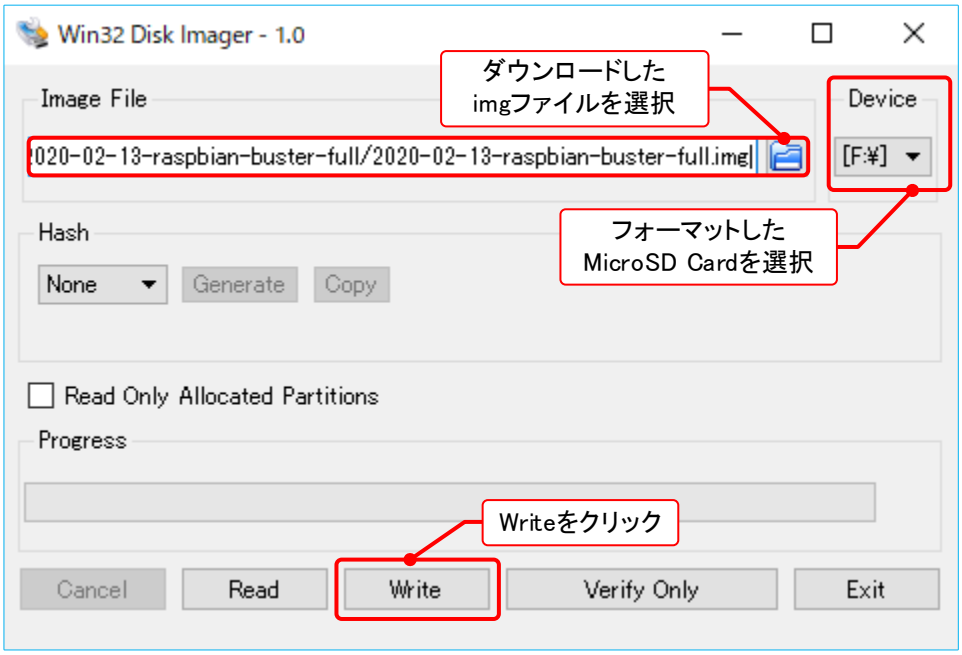
万が一、MicroSD Card のデータを消失してしまった場合は、このバックアップファイルを MicroSD Card へ書き込むことでバックアップファイル作成時点のデータ（Raspberry Pi の環境すべて）を復元できます。

Copyright © 2024 YITJC All Rights Reserved.

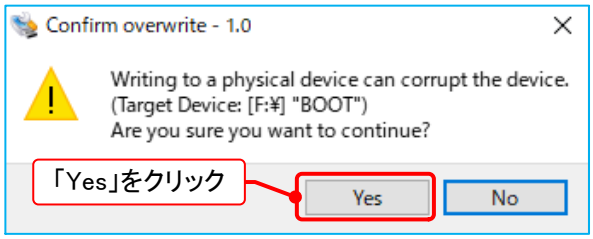
1.14 バックアップファイル書き込み

「1.13 バックアップファイル作成」で作成したバックアップファイル（.img ファイル）を Micro SD Card へ書き込む手順について説明します。まず、「Win32DiskImager」をダブルクリックします。

続いて、次の画面にしたがって OS の書き込みを実行します。

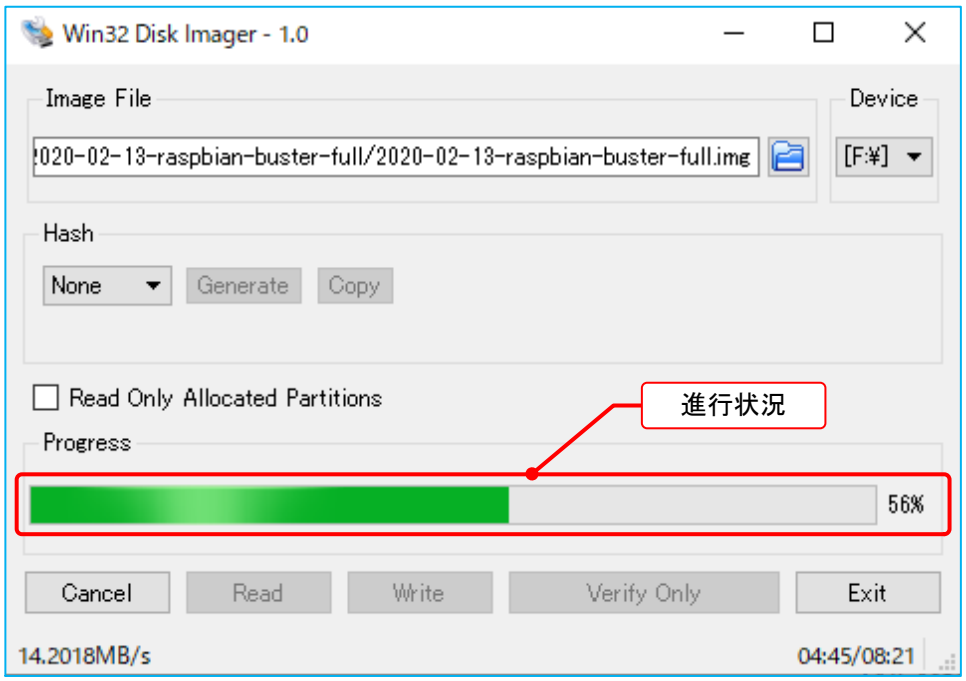


確認メッセージが表示されるので、「Yes」をクリック。

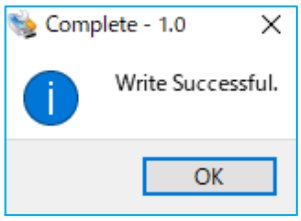


Copyright © 2024 YITJC All Rights Reserved.

書き込みの進行状況が表示されます。10 分程度かかります。



Write Successful と表示されたら、書き込み完了です。



Copyright © 2024 YITJC All Rights Reserved.

1.15 システム開発環境

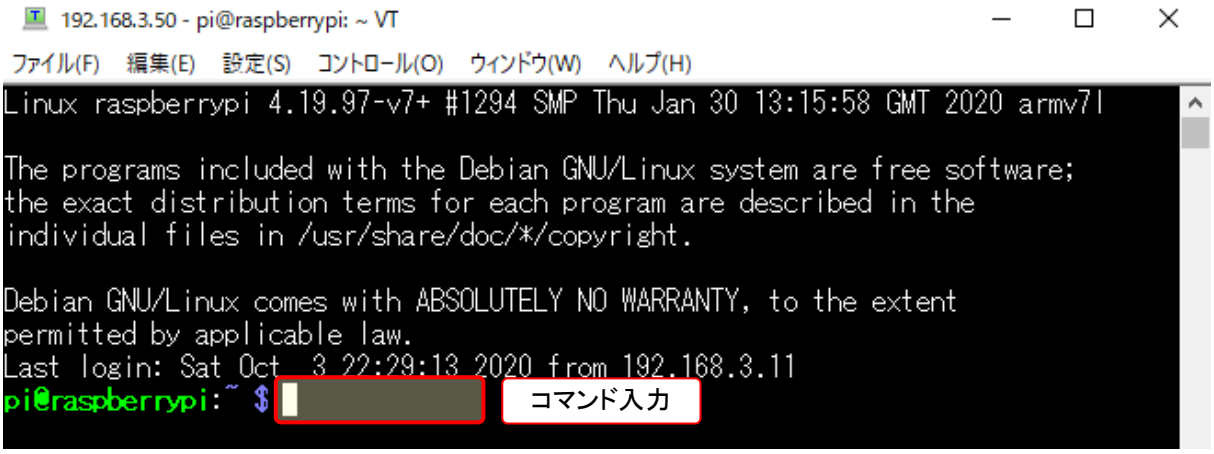
本講座は Raspberry Pi を使用して「生産性向上 IoT システム」を構築します。ここでは、Raspberry Pi を使用して「生産性向上 IoT システムを開発する」流れについて説明します。

手順①～Raspberry Pi 起動～

Type-C USB ケーブルを挿して Raspberry Pi を起動してください。

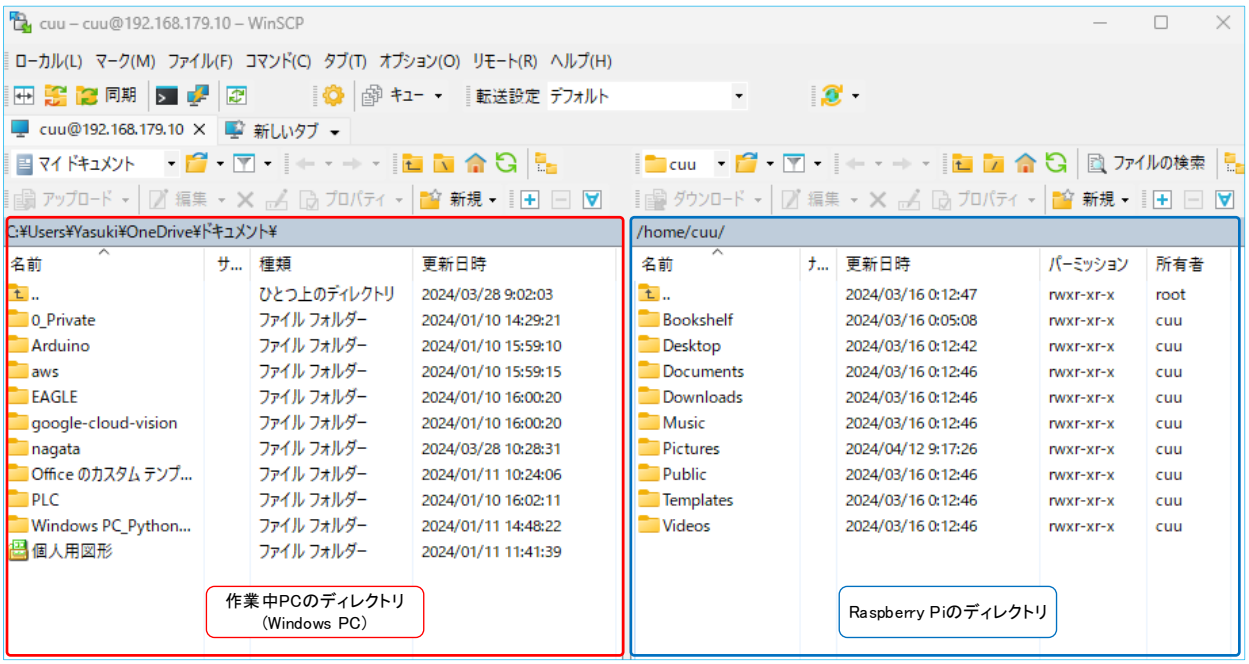
手順②～Tera Term 起動～

Tera Term を起動し、Raspberry Pi に SSH 接続します。コマンドを入力する形で開発を進めます。



手順③～WinSCP 起動～

WinSCP を起動し、Raspberry Pi に SSH 接続します。プログラムファイルなどは、Windows PC から直接ドラッグ＆ドロップでコピーできます。逆に、Raspberry Pi 内のファイルを Windows PC へコピーすることも可能です。



本講座は、「Tera Term」と「WinSCP」2 つのソフトウェアを起動した状態で開発を進めます。

Copyright © 2024 YITJC All Rights Reserved.

手順④～シャットダウン～

作業が終了したら、Raspberry Pi をシャットダウンします。必ずシャットダウンコマンドを実行してから、電源ケーブル（USB Type-C ケーブル）を抜くようにしてください。

シャットダウンコマンドは、次のいずれかを使用してください。

```
cuu@raspberrypi: ~ $ sudo shutdown -h now
```

または

```
cuu@raspberrypi: ~ $ sudo halt
```

第 2 章「GPIO 制御」において、HAT Board 上のスイッチに「シャットダウン」および「再起動」の機能を実装します。機能を実装したあとは、シャットダウンコマンドを実行する必要はなく、HAT Board 上に実装されているスイッチを押下して Raspberry Pi をシャットダウンできます。

Copyright © 2024 YITJC All Rights Reserved.

参考～エディタの使い方～

本講座は、エディタソフトウェア「Visual Studio Code」（以降、VS Code）を使用します。

※すでに、業務で使用されている別のエディタがある場合は、そちらをご利用ください。

注意～改行コード～

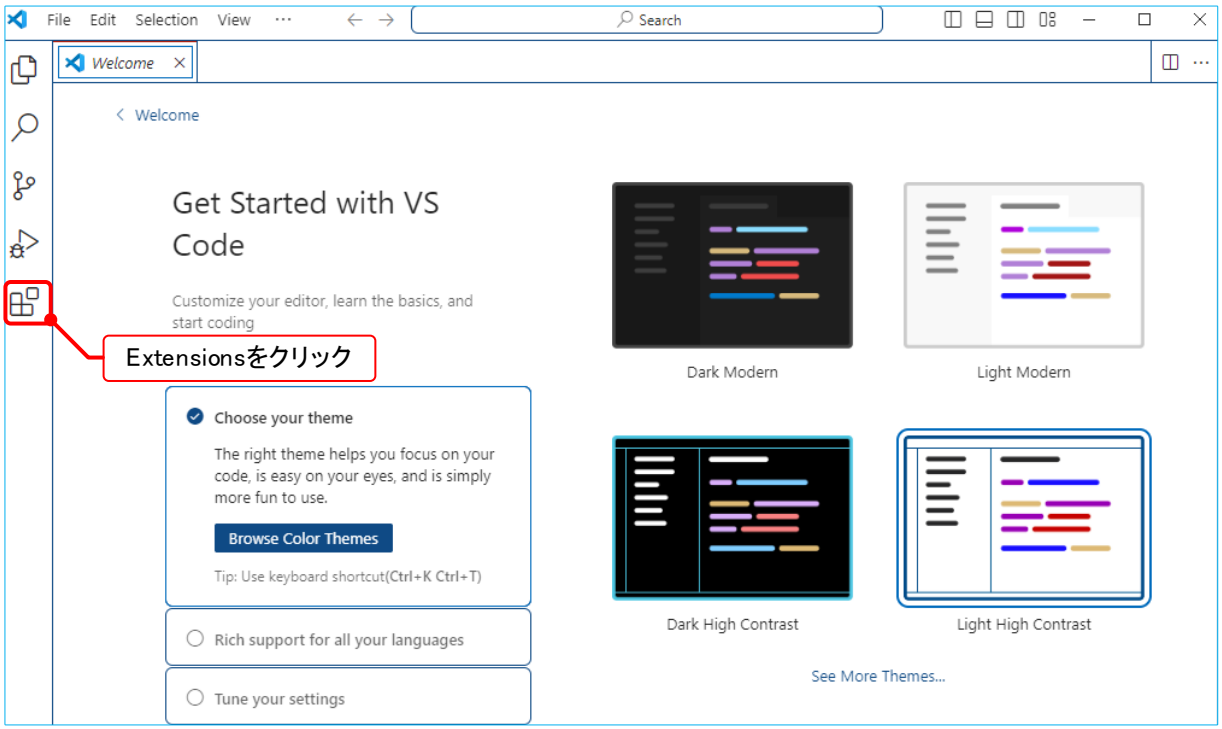
本講座は、Windows PC（OS: Windows）から Raspberry Pi（OS: Raspbian）に ssh 接続して IoT システムを開発します。そのため、Raspberry Pi で使用する Python プログラムや設定ファイルなどは、Windows PC で作成・編集します。Windows PC で作成する際に注意しなければいけない点が「改行コード」です。誤った改行コードで Python プログラムや設定ファイルを作成した場合、エラーが発生してプログラムは正常に動作しません。

Raspberry Pi の OS である Raspbian（Linux Debian OS がベースとなっている）は、Windows と改行コードが異なります。Raspberry Pi に保存する Python プログラムや設定ファイルなどは、改行コード「LF」で作成しなければなりません。

OS	改行コード
Raspbian	LF
Windows	CR+LF

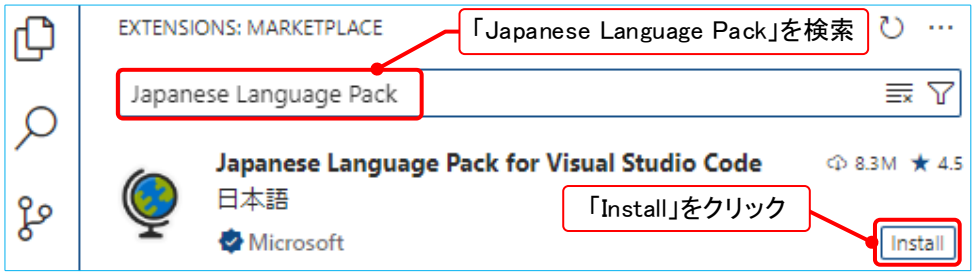
手順①～VS Code 日本語化～

VS Code を日本語化する手順を説明します。「Extensions」をクリックしてください。

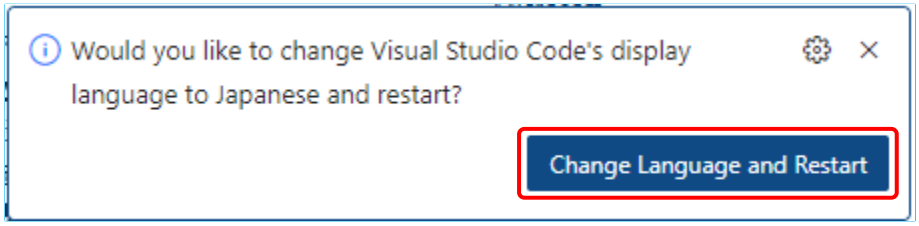


Copyright © 2024 YITJC All Rights Reserved.

「Japanese Language Pack」を検索し、インストールしてください。



「Change Language and Restart」をクリックし、VS Code を再起動してください。

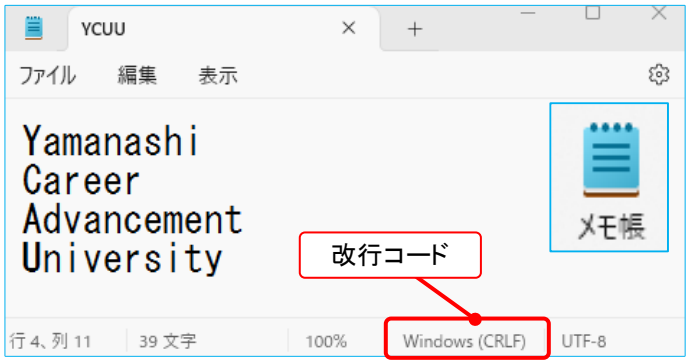


再起動後、日本語化されていることを確認してください。

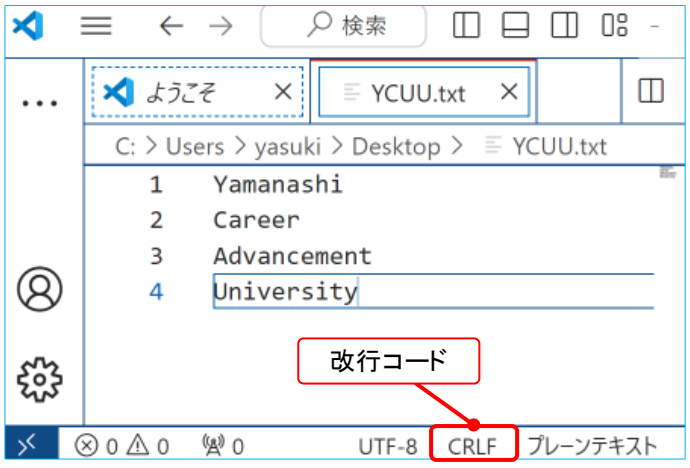
手順②～改行コードの可視化～

改行コードを可視化するため、「code-eol」をインストールします。

「code-eol」をインストールする前に、Windows PC の「メモ帳」で作成した文書を VS Code で開きます。
メモ帳で確認すると、画面下に改行コード「Windows（CRLF）」と表示されていることを確認できます。

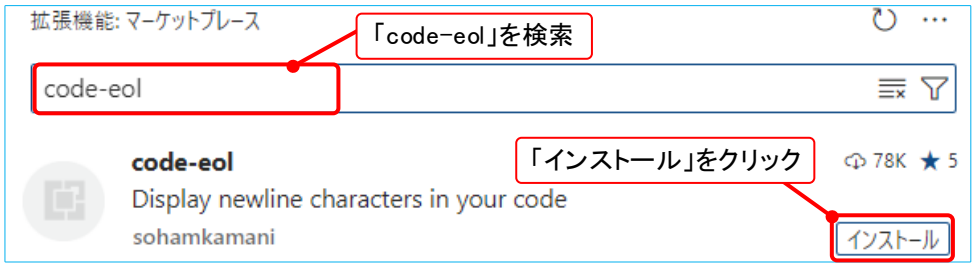


VS Code も画面下に改行コード「CRLF」が表示されていることを確認できます。

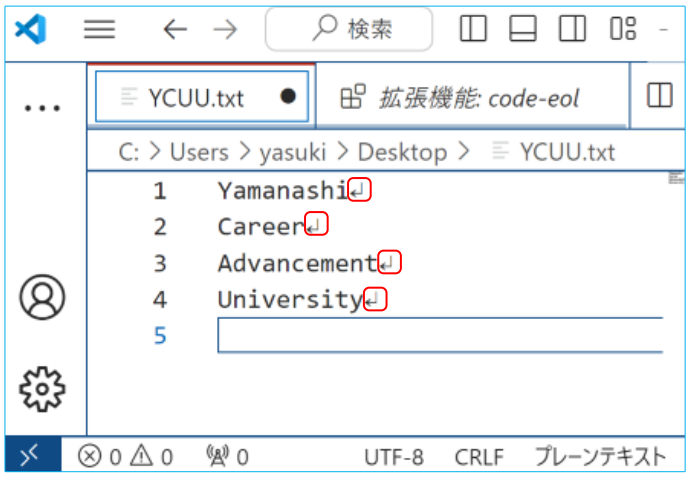


Copyright © 2024 YITJC All Rights Reserved.

続いて、「code-eol」をインストールします。

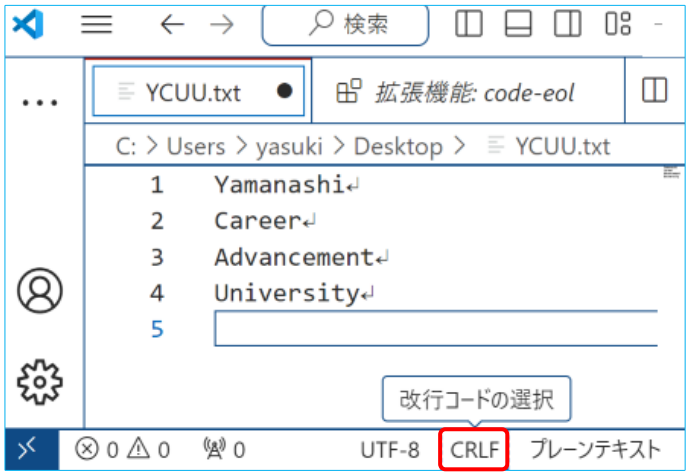


「code-eol」をインストールした後、「メモ帳」で作成した文書を VS Code で開くと、改行コードが可視化(⏎)されていることがわかります。

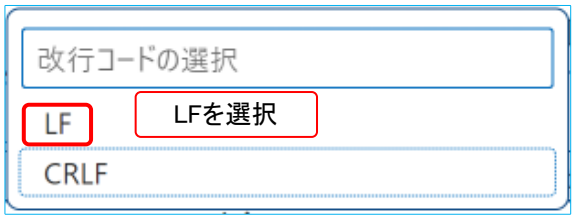


手順③～改行コードの置換～

画面下「改行コードの選択」をクリックします。

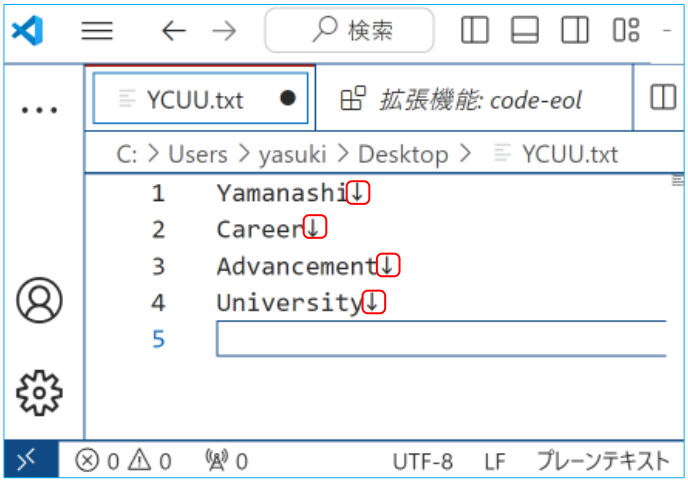


改行コード「LF」を選択してください。



Copyright © 2024 YITJC All Rights Reserved.

可視化された改行コードが「LF」(↓)に変更されたことがわかります。



繰り返しになりますが、Raspberry Pi で使用する Python プログラムや設定ファイルなどを Windows PC で編集した場合は、必ず改行コードが「LF」であることを確認してください。改行コードが「CRLF」の場合は、改行コードを置換して保存してください。

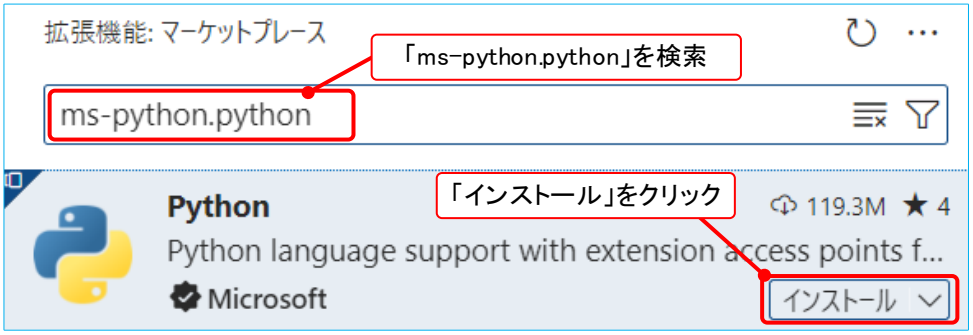
補足～VS Code 画面表示の拡大・縮小～

画面表示を拡大する場合は、「Ctrl」＋「Shift」＋「+」を同時に押すと拡大できます。

画面表示を縮小する場合は、「Ctrl」＋「Shift」＋「-」を同時に押すと縮小できます。

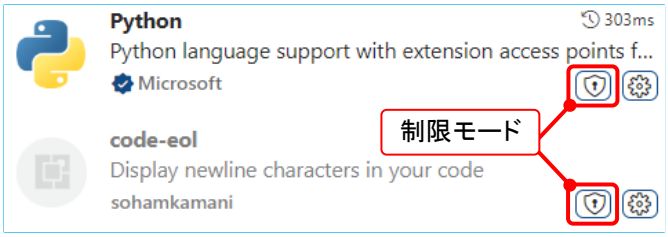
手順④～Python 拡張機能のインストール～

VS Code で Python のプログラミング環境を充実させるために拡張機能をインストールします。



補足～制限モードの解除～

「code eol」や「Python 拡張機能」が制限モードになっている場合があります。



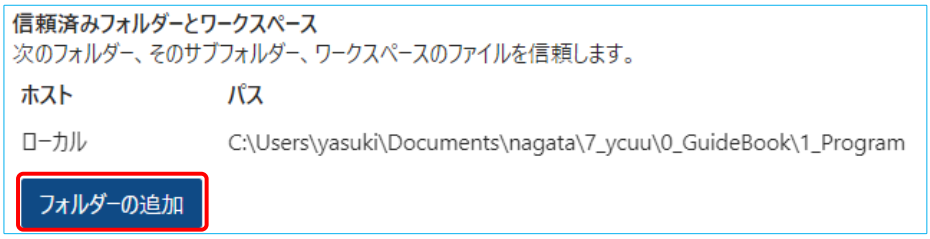
Copyright © 2024 YITJC All Rights Reserved.

制限モードになっている場合は、「信頼する」をクリックして機能を有効にしてください。

制限モードを「信頼する」作業は、毎回実施しなければなりません。

しかし、信頼済みフォルダを登録しておくことで、信頼済みのフォルダにあるデータ（例：Python プログラムなど）は、VS Code の拡張機能が「有効」になります。

信頼済みフォルダを登録する際は、「フォルダの追加」をクリックしてフォルダを登録します。

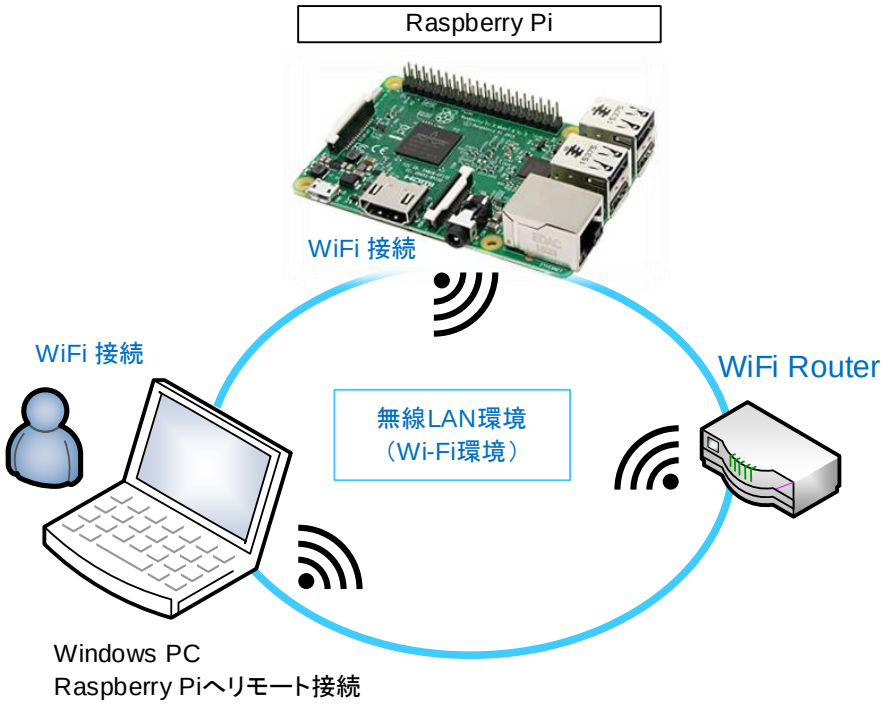


Copyright © 2024 YITJC All Rights Reserved.

参考～Raspberry Pi の IP Address 調査～

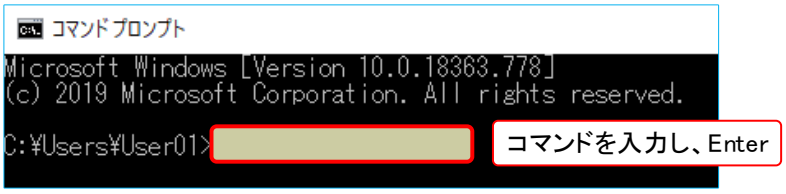
Windows PC から Raspberry Pi の IP Address を調べる方法を説明します。

「1.3 ネットワーク環境」で説明したとおり、Windows PC と Raspberry Pi は同じ無線 LAN（Wi-Fi）に接続しているため、Windows PC から同じネットワークに接続している Raspberry Pi（の IP Address）を探すことができます。



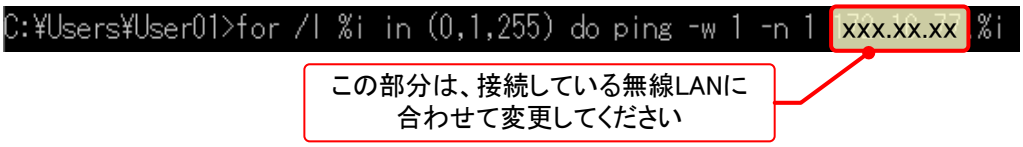
まず、Windows PC にインストールされているコマンドプロンプトを起動します。

起動すると、次の画面が表示されるので、□部にコマンドを入力して実行（Enter キー）します。



コマンドプロンプトに次の**コマンド**を入力して実行します。**コマンド**に含まれる IP アドレス（xxx.xx.xx）は、接続している無線 LAN（Wi-Fi）環境に合わせて変更してください。この**コマンド**は実行完了となるまで数分掛かります。

C:¥Users¥User01>for /l %i in (0,1,255) do ping -w 1 -n 1 xxx.xx.xx.%i

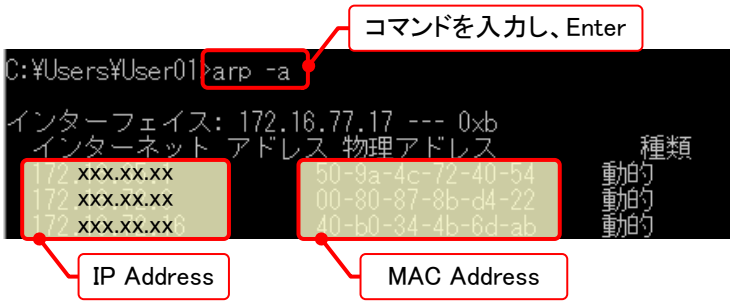


（参考）Windows PC の IP アドレスが「192.168.179.10」だった場合、「xxx.xx.xx」の部分は、「192.168.179」とします。

Copyright © 2024 YITJC All Rights Reserved.

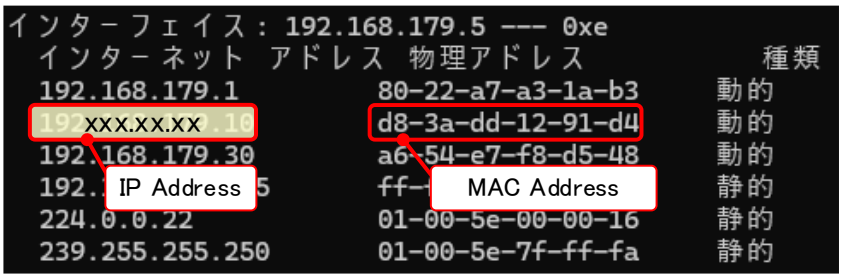
次のコマンドを実行し、無線 LAN（Wi-Fi）内に接続されているネットワーク機器一覧を arp テーブルに記録します。このコマンドは、Windows PC と同じネットワークに接続されている機器の IP アドレスと MAC アドレスの対照表を表示します。この一覧に、Raspberry Pi が含まれているはずです。

C:¥Users¥User01>arp -a



表示されたネットワーク機器から Raspberry Pi を探します。

Raspberry Pi の MAC Address は、d8:3a:dd から始まるアドレスとなっているので、arp テーブルの中から Raspberry Pi を見つけることができます。



～参考 Raspberry Pi4 の MAC アドレス～

Raspberry Pi3 は、b8:27:eb から始まる MAC アドレスで製造されていました。

Raspberry Pi4 は、新たに「dc:a6:32」、「d8:3a:dd」、「e4:5f:01」、「28:cd:c1」から始まる MAC アドレスが追加されています（2024 年 4 月時点）。

セキュリティ対策（デバイスの特定防止）のため、Raspberry Pi の MAC アドレスをランダム化が設定されているケースがあります。ランダム化が有効に設定されている場合、Raspberry Pi 本来の MAC アドレスは表示されず、ランダムに生成された MAC アドレスが表示されます。

ランダム化の設定は、次のコマンドで確認できます。

cuu@raspberrypi: ~ $ sudo nano /etc/NetworkManager/NetworkManager.conf

```
[device]
wifi.scan-rand-mac-address=no
```

「no」となっている場合は、ランダム化の設定が「無効」になっています。

Copyright © 2024 YITJC All Rights Reserved.

参考～Raspberry Pi の時計事情～

Raspberry Pi は、fake-hwclock という機能が Raspberry Pi の終了時刻を記録して、再開時に、その時刻をセットしますが、停止（電源 OFF）していた時間だけ誤差が生じてしまいます。そのため、通常はインターネットに接続していて、NTP（Network Time Protocol）でネットワーク時刻を取得する構成が必要になります。

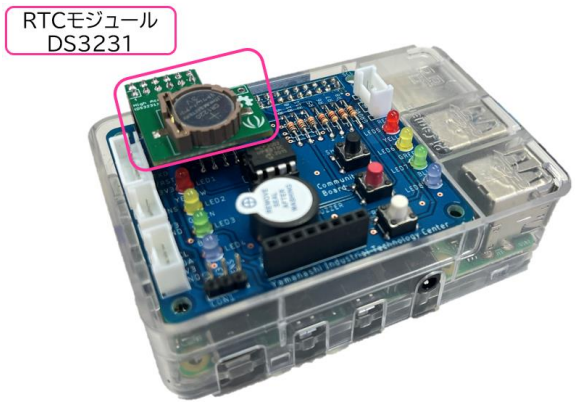
仮に、インターネットに接続していない環境で Raspberry Pi を使用する場合は、手動で時刻設定を行う必要があります。この問題を回避するため、Raspberry Pi 5 から RTC（Real Time Clock）が搭載され、専用のバックアップバッテリーを使用すれば、Raspberry Pi が停止（電源 OFF）している間も時刻を維持できるようになりました。

しかし、Raspberry Pi4 までは、RTC（Real Time Clock）が搭載されていないため、インターネットに接続しない環境で使用する場合は、外付け RTC モジュールを追加する必要があります。

ここでは、RTC モジュールを使用して時刻を設定する手順について説明します。

手順①～RTC モジュール装着～

Raspberry Pi（または HAT Board）に RTC モジュールを装着します。ここで使用している RTC モジュールは、DS3231（Seed Studio 社製）です。



手順②～I2C アドレス確認～

DS3231 の I2C アドレスを確認します。

cuu@raspberrypi: ~ $ sudo i2cdetect -y 1

```
cuu@raspberrypi:  $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

手順③～現在時刻確認～

次のコマンド実行し、現在の時刻を確認します。

cuu@raspberrypi: ~ $ date

```
cuu@raspberrypi:~ $ date
2024年  4月 15日 月曜日 13:55:17 JST
```

Copyright © 2024 YITJC All Rights Reserved.

手順④～RTC 有効化～

次のコマンド実行し、「config.txt」ファイルを編集します。

cuu@raspberrypi: ~ $ sudo nano /boot/firmware/config.txt

末尾に「dtoverlay=i2c-rtc,ds3231」を追加して上書き保存します。

手順⑤～再起動～

次のコマンドを実行し、Raspberry Pi を再起動します。

cuu@raspberrypi: ~ $ sudo reboot

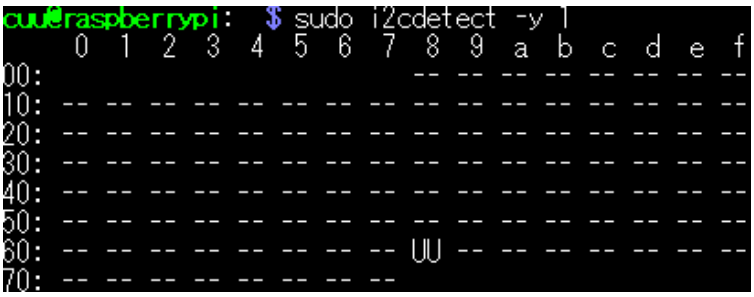
手順⑥～I2C アドレス再確認～

DS3231 の I2C アドレスを再確認します。

cuu@raspberrypi: ~ $ sudo i2cdetect -y 1

DS3231 の I2C アドレス（0x68）の部分が「UU」に変化していれば OK です。

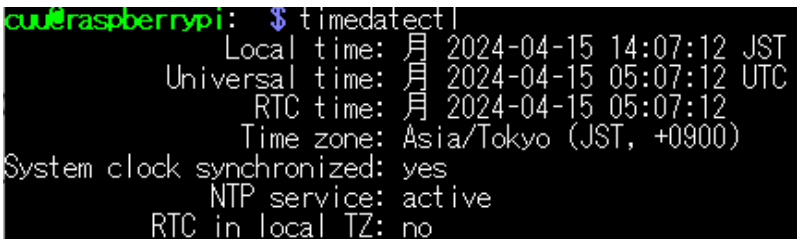
「UU」の表示は、RTC モジュール（DS3231）がシステムに管理されていることを示しています。



手順⑦～時刻確認～

RTC モジュールが使用可能となったため、次のコマンドで RTC モジュールの時刻を確認します。

cuu@raspberrypi: ~ $ timedatectl



Copyright © 2024 YITJC All Rights Reserved.

手順⑧～RTC 時刻設定～

次のコマンドを実行し、RTC モジュールに時刻を設定します。

cuu@raspberrypi: ~ $ sudo hwclock --set --date "2024-04-15 14:15:00"

```
cuu@raspberrypi:~ $ sudo hwclock --set --date "2024-04-15 14:15:00"
```

続いて、RTC モジュールの時刻をシステムの時刻として登録するサービスを作成します。Type=Oneshot 設定で作成するため、Raspberry Pi 起動時に RTC モジュールの時刻をシステムの時刻として登録します。

手順⑨～サービスファイル作成～

次のコマンドを実行し、ディレクトリを移動します。

cuu@raspberrypi: ~ $ cd /etc/systemd/system

次のコマンドを実行し、サービスファイル（.service）を作成します。

cuu@raspberrypi:/etc/systemd/system $ sudo nano RTC-start.service

次の内容を記述して保存します。

```
  GNU nano 7.2                          RTC-start.service *
[Unit]
Description=hwclock-start to read rtc and write to systemclock
After=sysinit.target

[Service]
Type=oneshot
ExecStart=/sbin/hwclock --hctosys --utc

[Install]
WantedBy=basic.target
```

手順⑩～サービスファイル起動～

次の順番でコマンドを実行し、手順⑧で作成したサービスファイルを起動します。

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable RTC-start.service

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start RTC-start.service

```
cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable RTC-start.service
Created symlink /etc/systemd/system/basic.target.wants/RTC-start.service → /etc/systemd/
system/RTC-start.service.
cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start RTC-start.service
```

手順⑪～再起動～

次のコマンドを実行し、Raspberry Pi を再起動します。

cuu@raspberrypi: ~ $ sudo reboot

Copyright © 2024 YITJC All Rights Reserved.

手順⑫～システム時刻確認～

時刻を確認すると、RTC モジュールに設定した時刻がシステムの時刻として登録されていることを確認します。

cuu@raspberrypi: ~ $ timedatectl

```
cuu@raspberrypi:~ $ timedatectl
               Local time: 月 2024-04-15 14:31:16 JST
           Universal time: 月 2024-04-15 05:31:16 UTC
                 RTC time: 月 2024-04-15 05:31:16
                Time zone: Asia/Tokyo (JST, +0900)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

また、次のコマンドでサービスファイルの状態を確認すると、再起動時に「RTC-start.service」の起動が成功（Deactivated successfully）していることを確認できます。

cuu@raspberrypi: ~ $ sudo systemctl status RTC-start.service

```
cuu@raspberrypi:~ $ sudo systemctl status RTC-start.service
○ RTC-start.service - hwclock-start to read rtc and write to systemclock
     Loaded: loaded (/etc/systemd/system/RTC-start.service; enabled; preset: enabled)
     Active: inactive (dead) since Mon 2024-04-15 14:30:04 JST; 2min 53s ago
    Process: 555 ExecStart=/sbin/hwclock --hctosys --utc (code=exited, status=0/SUCCESS)
   Main PID: 555 (code=exited, status=0/SUCCESS)
        CPU: 10ms

4月 15 14:30:04 raspberrypi systemd[1]: Starting RTC-start.service - hwclock-start to read
4月 15 14:30:04 raspberrypi systemd[1]: RTC-start.service: Deactivated successfully.
4月 15 14:30:04 raspberrypi systemd[1]: Finished RTC-start.service - hwclock-start to read
```

補足～RTC 時刻について～

Raspberry Pi は、通常 RTC に UTC（協定世界時）を設定して利用します。Universal time と RTC time が同じ（タイミングによって 1 秒程度ずれる可能性があります）であれば問題ありません。Local タイムには、初期設定「raspi-config」で設定した time zone に対応したローカル時刻が表示されます。

System clock synchronized は、NTP サーバーから時刻を取得してシステムに反映されている場合は、「yes」と表示されます。

Copyright © 2024 YITJC All Rights Reserved.

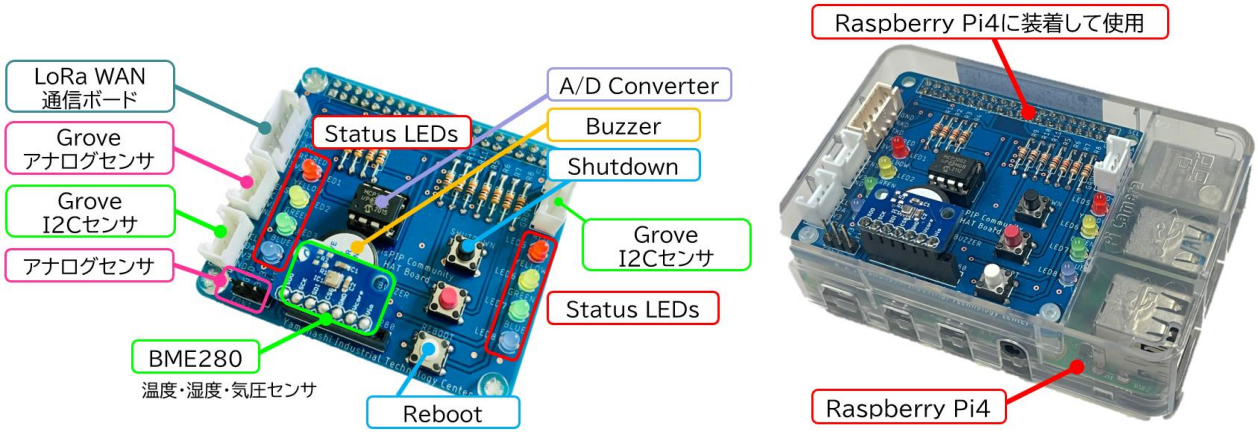
第 2 章 GPIO 制御

GPIO（General-purpose input/output: 汎用入出力）制御について学習します。

2.1 HAT Board

Raspberry Pi にはプログラムで制御できる GPIO ピンが 26 本あります。プログラムで制御する際、GPIO ピンの「番号」を指定して制御します。例えば「何番のピンの出力電圧を 3.3V にする」、「何番のピンの電圧の状態を読み取る」といった制御が可能になります。

本講座は、センサや LED などを搭載し、Raspberry Pi に装着可能な「HAT Board」を使用します。そのため、GPIO 制御やセンシングなどのプログラミングを効率的に学習できます。



HAT Board のピン割付（GPIO の何番がどこに接続されているかを示す表）は、次のとおりです。
例えば、GPIO6 ピンの出力電圧を 3.3V にすると、赤色 LED が点灯します。この表と照らし合わせて、GPIO 制御プログラムの学習を進めます。

Description	Device	Function		Pin		Function		Device	Description
		3.3V		1	2	5V			
	BME280	I2C1 SDA1	GPIO2	3	4	5V			
	BME280	I2C1 SCL1	GPIO3	5	6	GND			
From MQTT	LED赤	GCLK0	GPIO4	7	8	GPIO14	UART0 TXD	ES920LR	TXD
		GND		9	10	GPIO15	UART0 RXD	ES920LR	RXD
From MQTT	LED黄	GEN0	GPIO17	11	12	GPIO18	GEN1, PWM0	Buzzer	
From MQTT	LED緑	GEN2	GPIO27	13	14	GND			
From MQTT	LED青	GEN3	GPIO22	15	16	GPIO23	GEN4	Tact SW黒	Shutdown
		3.3V (max 50mA)		17	18	GPIO24	GEN5	Tact SW白	reboot
LM35DZ	MCP3002	SPI0 MOSI	GPIO10	19	20	GND			
LM35DZ	MCP3002	SPI0 MISO	GPIO9	21	22	GPIO25	GEN6	Tact SW赤	BZ_Stop
LM35DZ	MCP3002	SPI0 SCKL	GPIO11	23	24	GPIO8	SPI0 CS0	MCP3002	LM35DZ
		GND		25	26	GPIO7	SPI1 CS1		
		I2C ID EEPROM	ID_SD	27	28	ID_SC	I2C ID EEPROM		
/RESET	ES920LR	-	GPIO5	29	30	GND			
Status LEDs	LED赤	-	GPIO6	31	32	GPIO12	PWM0		
Status LEDs	LED黄	PWM1	GPIO13	33	34	GND			
Status LEDs	LED緑	SPI1 MISO, PWM1	GPIO19	35	36	GPIO16	-		
Status LEDs	LED青	-	GPIO26	37	38	GPIO20	SPI1 MOSI		
		GND		39	40	GPIO21	SPI1 SCLK		

Copyright © 2024 YITJC All Rights Reserved.

2.2 GPIO 出力制御

GPIO ポート出力制御プログラムを実行します。

[対象プログラム]

1_GPIO.py

※本講座で使用するプログラムは、共有サイト「1_Python」に保存されています。

[プログラム内容]

プログラムを実行すると、青色 LED（GPIO26）が 1 秒間点灯した後、消灯します。

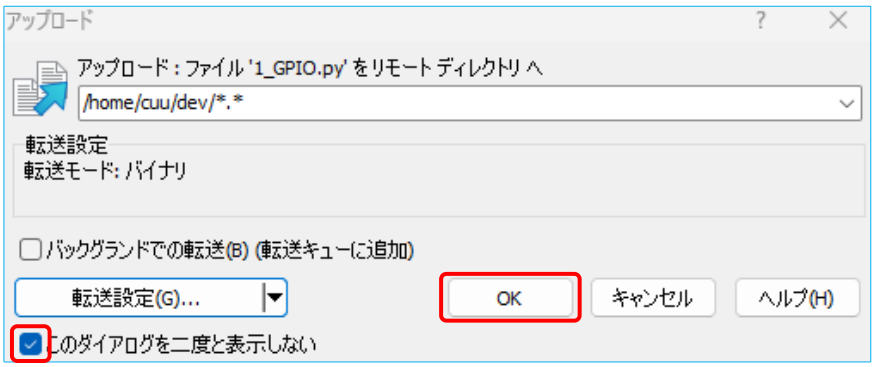
手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

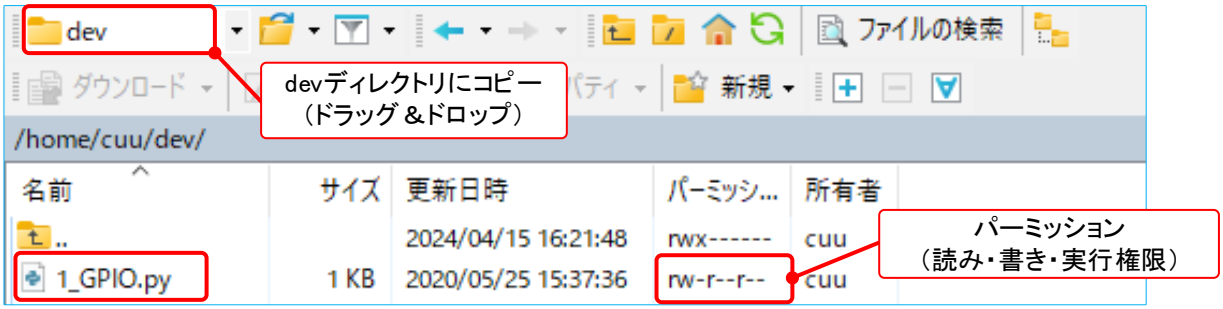
WinSCP を使用して初めてデータ（Python プログラム）を転送する際、次のメッセージが表示されます。

データの転送（アップロード）を確認する画面ですが、「OK」をクリックしてください。

今後この画面を表示させたくない場合は、「このダイアログを二度と表示しない」にチェックを入れてください。



データ転送が完了すると、Raspberry Pi の「dev」ディレクトリに「1_GPIO.py」が配置されます。



手順②～プログラム実行権限追加～

Tera Term を起動し、次のコマンドを実行してください。

```
cuu@raspberrypi: ~ $ cd dev
cuu@raspberrypi: ~/dev $
```

Copyright © 2024 YITJC All Rights Reserved.

「dev」ディレクトリへ移動した後、ls コマンドを実行してプログラムが保存されていることを確認してください。

ls コマンドは、「list segments」の略称でファイルやディレクトリの情報を表示します。

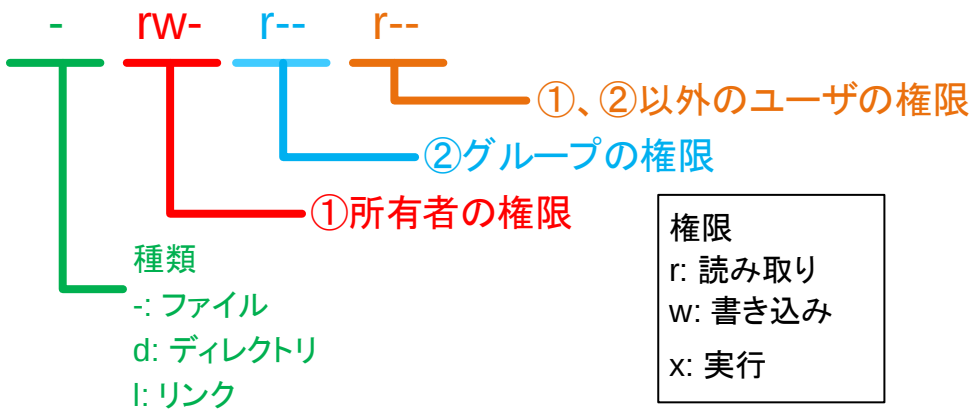
cuu@raspberrypi: ~/dev $ ls

```
cuu@raspberrypi:~/dev $ ls
1_GPIO.py
```

続いて、保存したプログラムの実行権限を確認します。WinSCP の画面と同じく「-rw-r—r--」が表示されていますが、これはパーミッション（読み・書き・実行権限）を表しています。今の状態は、実行権限「x」が追加されていないことが分かります。

cuu@raspberrypi: ~/dev $ ls -l 1_GPIO.py

```
cuu@raspberrypi:~/dev $ ls -l 1_GPIO.py
-rw-r--r-- 1 cuu cuu 570  5月 25  2020 1_GPIO.py
           ①  ②
```



次のコマンドを実行して、実行権限を追加します。

cuu@raspberrypi: ~/dev $ chmod +x 1_GPIO.py

実行権限を確認すると実行権限「x」が追加され、1_GPIO.py プログラムも緑で表示されていることが分かります。

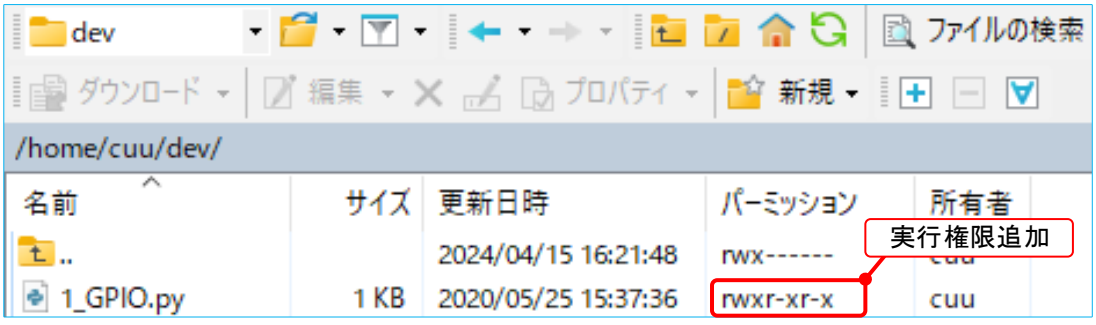
```
cuu@raspberrypi:~/dev $ ls -l 1_GPIO.py
-rwxr-xr-x 1 cuu cuu 570  5月 25  2020 1_GPIO.py
```

～参考 ファイルへのアクセス権追加～

ファイルへのアクセス権を設定するには、chmod（Change Mode）コマンドを実行します。何か権限を追加するときは「+」、削除する場合は「-」を指定し、その後に、読み取り「r」、書き込み「w」、実行「x」などの種類を指定します。

ここでは実行権限を追加したいので「x」を指定しています。

WinSCP の画面からも実行権限が追加されていることが分かります。



Copyright © 2024 YITJC All Rights Reserved.

～参考 コマンドの再入力～

Tera Term でコマンドを入力する場合は、コマンドの入力履歴から選択すると便利です。コマンドの入力履歴は、カーソルキー「↑」、「↓」または「Ctrl」＋「P」、「Ctrl」＋「N」で呼び出すことができます。

手順③～プログラム実行～

対象プログラムを実行します。ここでは、2 つの実行方法を説明します。

① プログラムを指定して実行（python3 を指定）

cuu@raspberrypi: ~/dev $ python3 1_GPIO.py

② プログラムを指定せず、直接ファイルを実行する

cuu@raspberrypi: ~/dev $ ./1_GPIO.py

「.」は、現在のディレクトリを意味しています。

いずれの方法も、プログラムの実行結果（青色 LED が 1 秒間点灯した後、消灯する様子）を確認することができます。

①の方法は、python3 を指定してプログラムを実行しています。

②の方法は、直接ファイル（1_GPIO.py）を実行しています。直接実行する場合は、1_GPIO.py プログラムの中に Shebang（シバン、シェバン）と呼ばれる記述が必要になります。（プログラム解説で説明します。）

なお、②の方法は、実行権限が追加されていないと実行することができません。実行権限が追加されていない状態で②の方法を実行すると、次の画面のように警告メッセージが表示されます。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/usr/bin/python3
2	# coding: utf-8
3	
4	import RPi.GPIO as GPIO
5	from time import sleep
6	
7	#GPIO端子初期設定関数
8	def setup_gpio():
9	#GPIO.BOARD:ピン番号指定
10	#GPIO.BCM: GPIO番号指定（こちらでプログラムを記述）
11	GPIO.setmode(GPIO.BCM)
12	#青LED
13	#出力「OUT」、「LOW(0V出力)」設定
14	GPIO.setup(26, GPIO.OUT)
15	GPIO.output(26, GPIO.LOW)
16	
17	#setup_gpio関数を実行
18	setup_gpio()
19	#青LED点灯
20	GPIO.output(26, GPIO.HIGH)
21	sleep(1.0)
22	#青LED消灯
23	GPIO.output(26, GPIO.LOW)
24	sleep(1.0)
25	#GPIOポートを初期化
26	GPIO.cleanup()

・ 1 行目: Shebang（シバン、シェバン）

#から始まるがコメントではない。2 通りの記述がある。

#!/usr/bin/python3

#!/usr/bin/env python3

Unix/Linux 系の環境では、スクリプトの 1 行目に Shebang を記載しておき、スクリプト自体を実行することで、Shebang に記載されたプログラムを実行しそのプログラムに、スクリプトの内容を渡して動作させることができます。

つまり、1_GPIO.py を

cuu@raspberrypi: ~/dev $ ./1_GPIO.py

と実行すると、Shebang によって python3 プログラムが実行され、python3 プログラムに 1_GPIO.py の内容が渡され、処理されます。

Copyright © 2024 YITJC All Rights Reserved.

～参考 Shebang が必要ないケース～

cuu@raspberrypi: ~/dev $ python3 1_GPIO.py

と実行する場合は、python3 と指定してプログラムを実行しているので、Shebang の記述は必要ありません。

- 2 行目: マジックコメント

  # coding: utf-8

  ソースファイルの文字コードを指定しています。

  スクリプトが実行される際、python のインタプリタ（Python 言語の記述を解釈して動作するプログラム）は、そのスクリプト内に記載されている文字が、指定された文字エンコーディングであることを期待して実行されます。

  つまり、マジックコメント内に utf-8 という記載があると、Python のインタプリタは、スクリプトファイル内の文字のエンコーディングが utf-8 であると期待してスクリプトファイルを解釈しながらプログラムを処理していきます。

  1 行目: #!/usr/bin/python
  2 行目: # coding: utf-8
  の記述は、今後作成するプログラムにおいても同様に記述していきます。

- 4 行目:

  import RPi.GPIO as GPIO

  Raspberry Pi の GPIO を制御するライブラリとして、

  - Pigpio
  - RPi.GPIO
  - WiringPi

  がありますが、今回は Rpi.GPIO を使用します。ここでは、RPi.GPIO ライブラリを使用することを宣言しています。（Rpi.GPIO は、Raspberry Pi にあらかじめインストールされているライブラリです。）

～参考 ライブラリ～

RPi.GPIO のようにあらかじめ内蔵（インストール）されているライブラリを標準ライブラリと呼びます。標準ライブラリは、公式サイトで探すことができます。

https://docs.python.org/ja/3/library/index.html

標準ライブラリ以外にも公開されているライブラリ（外部ライブラリ）は多数あり、Raspberry Pi にインストール（インポート）することで利用できるようになります。本講座も外部ライブラリをインストールして IoT システムを構築します。外部ライブラリのインストール方法については、外部ライブラリを使用するときに説明します。

- 5 行目: sleep 関数インポート

  from time import sleep

  プログラムを一時停止させたいときに便利な「Sleep 関数」を使用するための記述です。

  今回のプログラムにおいては、time ライブラリの中から sleep 関数のみをインポートする形で記述しています。

  Sleep 関数をインポートすると、

  sleep(1.0)　　#1 秒停止

  といった記述で、プログラムを一時停止させることができます。

Copyright © 2024 YITJC All Rights Reserved.

～参考 別の記述方法 Sleep～

import time

time ライブラリだけをインポートする形で記述した場合は、

time.sleep(1.0) #1 秒停止

といった記述で、プログラムを一時停止させることができます。

- 8 行目～15 行目: GPIO 端子初期設定関数

  def とは、英語の define（定義する）を省略しています。

  関数は次のように def 文のブロックで定義します。

```
def 関数名():
        処理
```

  呼び出すときは、関数名をそのまま記述します。

```
関数名()
```

- 11 行目: GPIO 端子を指定する方法を記述しています。

  2 つの方法があり、Raspberry Pi 上には 40pin のヘッダーピンが用意されています。

  そのうち、26 ピンは GPIO 端子として使用することができます。GPIO 端子には、ピン番号とは別に GPIO 番号が付けられています。

  GPIO.BOARD:ピン番号指定

  GPIO.BCM: GPIO 番号指定

  ここでは、GPIIO.BCM を使用して GPIO 端子を指定します。

- 14 行目: 入出力設定

  GPIO 端子を入力端子として使用する場合は、GPIO.setup(6, GPIO.IN)

  GPIO 端子を出力端子として使用する場合は、GPIO.setup(6, GPIO.OUT)

  と記述します。

- 15 行目: 出力電圧設定

  GPIO 端子から 0V を出力する場合は、GPIO.output(6, GPIO.LOW)

  GPIO 端子から 3.3V を出力する場合は、GPIO.output(6, GPIO.HIGH)

  ここでは、初期設定として 0V を出力し、LED を消灯させています。

- 17 行目～26 行目: プログラム実行
- 18 行目: GPIO 端子初期設定関数実行
- 20 行目: GPIO26 番 3.3V 出力
- 21 行目: 1 秒間停止
- 23 行目: GPIO26 番 0V 出力
- 24 行目: 1 秒間停止
- 26 行目: GPIO.cleanup()

Copyright © 2024 YITJC All Rights Reserved.

注意～GPIO.cleanup()～

GPIO を使用した後は必ず GPIO.cleanup()を実行しなければなりません。cleanup メソッドは、GPIO ピンのシステムクリアを実行します。つまり、プログラム内でセットした OUTPUT ポートを INPUT に切り替えます。仮に、OUTPUT のまま放置し、かつ HIGH に設定していた場合、回路の短絡などの危険が伴います。そのため、プログラムの最後に cleanup メソッドを実行させることで、次回実行時に初期状態からスタートすることができます。

プログラム中に GPIO.cleanup()を記述せず、プログラムを 2 回目に実行すると、次のワーニングメッセージが表示されます。1 回目にプログラムを実行した際に、cleanup メソッドを実行していないため、GPIO が「使用中」のままプログラムを終了しています。その後、2 回目のプログラム実行の際、「使用中（すでに使われている）」というワーニングメッセージが表示されています。

～ワーニングメッセージ～

./1_GPIO.py:27: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.

GPIO.setup(26, GPIO.OUT)

def setup_gpio()関数内で、次の記述を追加するとワーニングメッセージを「非表示」に設定できます。

GPIO.setwarning(False)

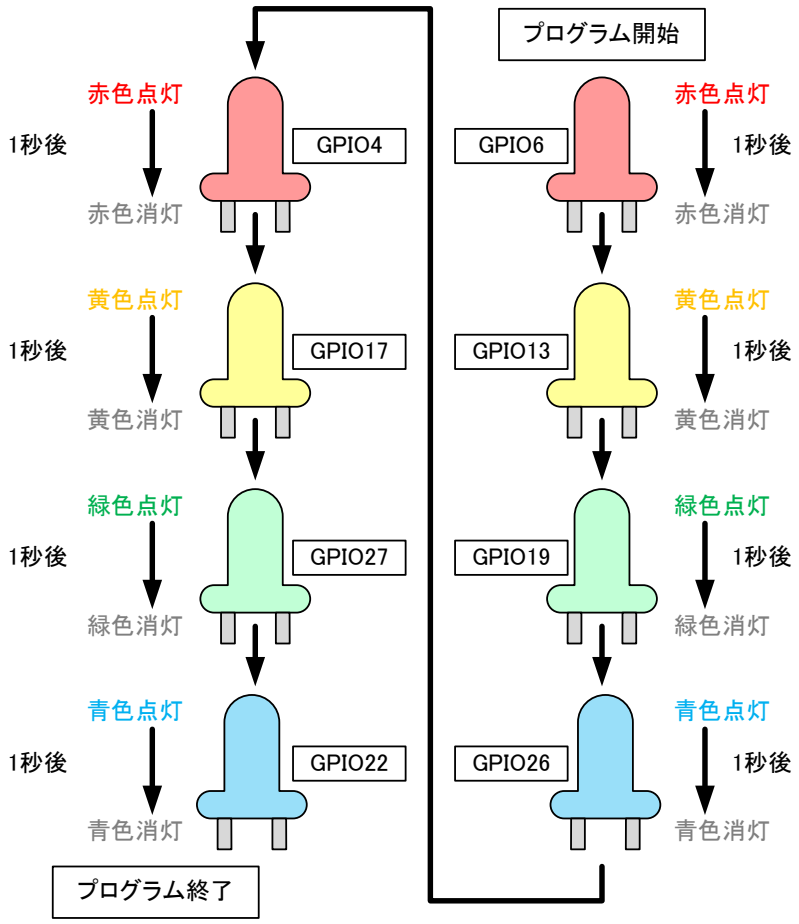
この記述を追加した場合、すべてのワーニングメッセージが非表示となるため、エラーにつながる警告メッセージを見落とす可能性があります。そのため、この記述の追加はお勧めしません。

Copyright © 2024 YITJC All Rights Reserved.

演習 1～GPIO ポート出力制御～

仕様

・ HAT Board に実装されている 8 個の LED を順番に点灯させるプログラムを作成してください。
点灯仕様は、次の図に示します。



プログラム名

2_GPIO.py

補足

プログラムを実行する前に、実行権限を追加する手順を忘れないようにしてください。

Copyright © 2024 YITJC All Rights Reserved.

2.3 GPIO 入出力制御

GPIO 入出力制御プログラムを実行してみます。

[対象プログラム]

3_GPIO.py

[プログラム内容]

プログラムを実行すると、青色 LED（GPIO26）が点滅を開始します。

赤色スイッチ（GPIO25）を押すと、プログラムは終了します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーしてください。（ドラッグ＆ドロップ）

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
cuu@raspberrypi: ~/dev $ chmod +x 3_GPIO.py
```

手順③～プログラム実行～

対象プログラムを実行します。

```
cuu@raspberrypi: ~/dev $ ./3_GPIO.py
```

プログラム実行後、青色 LED が点滅することを確認してください。

そのときの画面を見ると、次の画面のようにプログラムが実行中である（次のコマンド入力ラインが表示されていない）ことが分かります。

この状態で赤色スイッチを押すと、プログラムが終了し、次の画面の表示になります。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/usr/bin/python3
2	# coding: utf-8
3	
4	import RPi.GPIO as GPIO
5	from time import sleep
6	
7	#GPIO端子初期設定関数
8	def setup_gpio():
9	#GPIO.BOARD:ピン番号指定
10	#GPIO.BCM: GPIO番号指定（こちらでプログラムを記述）
11	GPIO.setmode(GPIO.BCM)
12	#赤色スイッチを「入力IN」に設定
13	GPIO.setup(25, GPIO.IN)
14	#青LED
15	#出力「OUT」、「LOW(0V出力)」設定
16	GPIO.setup(26, GPIO.OUT)
17	GPIO.output(26, GPIO.LOW)
18	
19	#setup_gpio関数を実行
20	setup_gpio()
21	#while True以下の内容をループ処理（繰り返し実行）します
22	while True:
23	#赤色SWの状態をsw_state変数に代入
24	sw_state = GPIO.input(25)
25	#sw_stateの状態がTrue(赤色SWが押されていない状態)
26	if sw_state == True:
27	#Trueと判定した場合の処理
28	#青LED点灯
29	GPIO.output(26, GPIO.HIGH)
30	sleep(1.0)
31	#青LED消灯
32	GPIO.output(26, GPIO.LOW)
33	sleep(1.0)
34	else:
35	#True以外(False)
36	#つまり、赤色SWが押された時の処理
37	#cleanupを実行します

Copyright © 2024 YITJC All Rights Reserved.

```
38          GPIO.cleanup()
39          #続いて、breakでwhileループ処理を抜けます(=プログラム終了)
40          break
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang(シバン、シェバン)
- 2 行目: マジックコメント
- 4 行目: RPi.GPIO ライブラリを import
- 5 行目: sleep 関数インポート
- 7 行目～17 行目: GPIO 端子初期設定関数

  赤色スイッチ(GPIO25)の電圧の状態を確認するので、赤色スイッチの GPIO ポートは「入力」として設定します。

  GPIO.setup(25, GPIO.IN)

  赤色スイッチの回路は、HAT Board 上で 10kΩの抵抗でプルアップ処理してありますが、Raspberry Pi は、「プルアップ抵抗」、「プルダウン抵抗」の 2 つを内蔵しており、ソフトウェアから有効・無効を切り替えることができます。内蔵プルアップ抵抗を設定する場合は、次の記述になります。

  GPIO.setup(25, GPIO.IN, pull_up_down = GPIO.PUD_UP)
- 20 行目: GPIO 端子初期設定関数実行
- 22 行目: while 文によるループ処理となっています

  Python における while 文の構造としては、

  ```
  while 条件式:
      処理
  ```

  となりますが、今回のプログラムを確認すると、条件式の部分に「True」と書かれています。

  ```
  while True:
      処理
  ```

  条件式が「True」ということは、「常に条件式が真の状態」(条件を満たしている状態)であるため、ループ処理が無限に繰り返されます。これを「**無限ループ**」と呼びます。

  無限ループから抜け出す(終了する)ためには、ループ処理内に「break」を用意します。

  「break」がない場合は、強制終了(Ctrl + c)するしかなくなるので注意が必要です。

- 24 行目: 赤色スイッチの状態を sw_state 変数に代入
- 26 行目: if 文を使用して赤色スイッチの状態を判定しています
- 27 行目～33 行目: 赤色スイッチが True の状態(押されていない状態)のときに実行する処理を記述
- 34 行目～40 行目: 赤色スイッチが True の状態(押された状態)のときに実行する処理を記述

  40 行目に break と記述してあることで、while 文のループ処理から抜け出すことができます。

Copyright © 2024 YITJC All Rights Reserved.

演習 2～GPIO ポート入出力制御～

仕様

・ プログラムを実行すると、8 個の LED がすべて点灯します。
赤色スイッチ（GPIO25）を押すと、8 個の LED をすべて消灯させた後、プログラムが終了します。

プログラム名

4_GPIO.py

補足

① 各 LED が接続されている GPIO ポート番号を確認するには、HAT Board のピン割付を確認してください。

② プログラムを実行する前に、実行権限を追加する手順を忘れないようにしてください。

Copyright © 2024 YITJC All Rights Reserved.

2.4 シャットダウン機能の追加

1 章「開発環境構築」で Raspberry Pi を再起動・シャットダウンするコマンドを学習しました。

- 再起動

  cuu@raspberrypi: ~ $ sudo reboot

- シャットダウン

  cuu@raspberrypi: ~ $ sudo shutdown -h now

  cuu@raspberrypi: ~ $ sudo halt

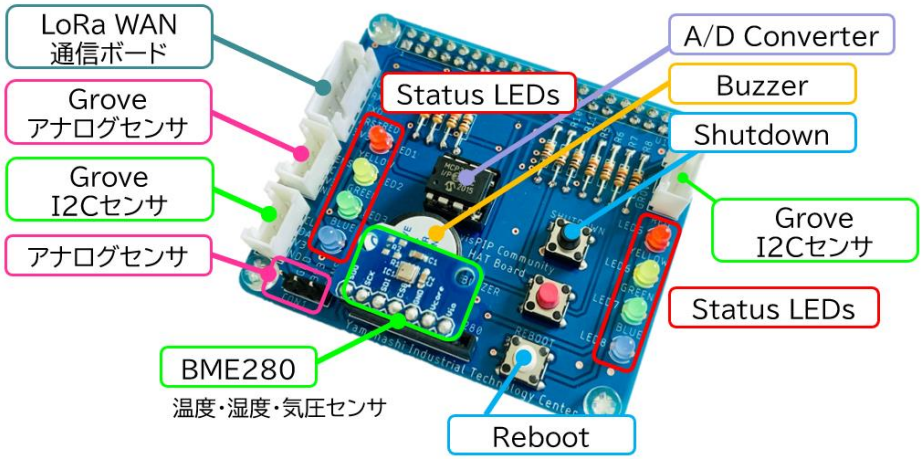
再起動・シャットダウンを行うたびにコマンドを実行する手間を省くため、HAT Board に実装されているスイッチを利用して再起動・シャットダウン機能を追加します。

手順①～GPIO ポート番号確認～

HAT Board の白色スイッチ、黒色スイッチを利用します。

白色スイッチ（GPIO24）： 再起動スイッチ（Reboot スイッチ）

黒色スイッチ（GPIO23）： シャットダウンスイッチ（Shutdown スイッチ）



手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーしてください。（ドラッグ＆ドロップ）

[対象プログラム]

5_Shutdown.py

[プログラム内容]

白色スイッチを押すと、ただちに「再起動」します。

黒色スイッチを押すと、ただちに「シャットダウン」します。

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

cuu@raspberrypi: ~/dev $ chmod +x 5_Shutdown.py

Copyright © 2024 YITJC All Rights Reserved.

手順④～プログラム実行～

シャットダウンプログラムを実行します。

cuu@raspberrypi: ~/dev $ ./5_Shutdown.py

プログラムを実行すると、次の画面のようにプログラムが実行中である（次のコマンド入力ラインが表示されていない）ことが分かります。（プログラム 27 行目～34 行目の無限ループにより実行中の状態になっています。）

この状態で、

白色スイッチを押すと、Raspberry Pi は再起動します。（Tera Term の画面は自動的に閉じられます。）

黒色スイッチを押すと、Raspberry Pi はシャットダウンします。（Tera Term の画面は自動的に閉じられます。）

参考～自動起動の必要性～

ここで気づかれる方もいると思いますが、HAT Board 上にあるシャットダウン・再起動スイッチを使用してシャットダウン・再起動を実行するためにはプログラム（5_Shutdown.py）を実行中しなければなりません。

Raspberry Pi にあるプログラム（5_Shutdown.py）を実行するには、

① Windows パソコンと Raspberry Pi を SSH 接続

② Windows パソコンからプログラム（5_Shutdown.py）を実行

といった手順が必要になりますが、この手順を毎回実施するのは大変です（Windows PC が毎回必要になってしまいます）。

本講座は、さまざまなプログラムを実行して「生産性向上 IoT システム」を構築しますが、システムを稼働させるたびに、Raspberry Pi にあるプログラムを実行することは現実的ではありません。Raspberry Pi は、小型「コンピュータ」ですので、単独で動作させることに価値があります。

Windows PC を使用せず、Raspberry Pi 単独で動作させる方法として「自動起動」があります。

再起動・シャットダウン機能を例に挙げて、「自動起動」について説明します。

Raspberry Pi が電源 ON した後、プログラム（5_Shutdown.py）を自動で実行（自動起動）するように設定すれば、Windows PC から手動で Python プログラムを実行しなくても、再起動・シャットダウン機能が有効になります。つまり、Raspberry Pi を単独で動作させることができます。

➔ 「自動起動」の方法について次節で説明します。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/usr/bin/python3
2	# coding: utf-8
3	
4	import RPi.GPIO as GPIO
5	import time
6	import os
7	
8	# GPIO番号設定
9	shutdown_pin = 23
10	reboot_pin = 24
11	
12	# GPIO端子初期設定関数
13	GPIO.setmode(GPIO.BCM)
14	GPIO.setup(shutdown_pin, GPIO.IN)
15	GPIO.setup(reboot_pin, GPIO.IN)
16	
17	#シャットダウンコマンド実行
18	def shutdown():
19	os.system("sudo shutdown -h now")
20	
21	#再起動コマンド実行
22	def reboot():
23	os.system("sudo reboot")
24	
25	try:
26	#無限ループ処理(スイッチの状態を常に監視)
27	while True:
28	# シャットダウンスイッチの状態を監視
29	if GPIO.input(shutdown_pin) == GPIO.LOW:
30	shutdown()
31	# 再起動スイッチの状態を監視
32	if GPIO.input(reboot_pin) == GPIO.LOW:
33	reboot()
34	time.sleep(0.1)
35	
36	except KeyboardInterrupt:
37	# Ctrl+C が押されたらプログラムを終了する

Copyright © 2024 YITJC All Rights Reserved.

```
38      pass
39
40  finally:
41      GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: RPi.GPIO ライブラリを import
- 5 行目: time ライブラリ import
- 6 行目: os ライブラリ import
- 8 行目～10 行目: シャットダウン、再起動に使用する GPIO 番号を設定
  シャットダウン:GPIO23
  再起動:GPIO24
- 12 行目～15 行目: GPIO 端子初期設定関数
  シャットダウンプログラムに使用するスイッチの初期設定を行っています。
- 17 行目～19 行目: シャットダウンプログラム関数
- 21 行目～23 行目: 再起動プログラム関数
  6 行目で import した os ライブラリが使用されています。
  os.system()は Unix のコマンドを Python で記述するためのライブラリ（関数）です。
  シャットダウンを実行させたいときは、
  os.system("sudo shutdown -h now")
  というように、os.system()内にシャットダウンコマンドを記述します。

～参考～

subprocess ライブラリを import してシャットダウンコマンドを実行する方法についても紹介します。
シャットダウンコマンドは、subprocess ライブラリを使用すると次の記述で実行することができます。

```
import subprocess
subprocess.call (″sudo shutdown -h now ″)
```

- 25 行目～41 行目: 例外処理
  プログラム実行中に検出されたエラーを例外（Exception）と呼びます。
  ここでは、例外処理（except）、例外の発生によらず必ず最後に実行する処理（finally）を記述しています。
- 25 行目～34 行目:try ブロック
  while true の無限ループ内で、シャットダウンスイッチ（GPIO23）と再起動スイッチ（GPIO24）の状態を監視しています。
  スイッチの状態が Low になると、shutdown()関数または reboot()関数が呼び出され、シャットダウンまたは再起動が実行されます。

Copyright © 2024 YITJC All Rights Reserved.

- 36 行目～38 行目：except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文を記述しています。
  「pass」は、何も処理を行わないことを意味しています。
- 40 行目～41 行目：finally ブロック
  例外の発生によらず必ず実行される処理を記述しています。
  本プログラムは GPIO を使用しているため、GPIO.cleanup()を記述しています。これにより、例外の発生によらず GPIO ピンのシステムクリアが必ず実行されます。

～参考 ポーリング制御と割り込み制御～

- **ポーリング制御**

ポーリング制御は、無限ループ内でシステム（スイッチ）の状態を定期的に監視する方法です。
休止期間（sleep）を設定することで、CPU の負荷を軽減し、Raspberry Pi の動作を安定させます。
休止（sleep）時間を長くすると CPU の負荷を減らすことができますが、スイッチの反応は鈍くなります。

- **割り込み制御**

割り込み制御は、CPU の負荷を減らし、スイッチ（の状態変化）に対する応答性を向上させる方法です。
指定した GPIO ピンの状態が変化すると、CPU が待機状態から起動して割り込みイベントを処理します。
この方法を使用すると、CPU はスイッチの状態変化を待機する間に他の処理を実行できるため、CPU のリソースを効率的に使用できます。

GPIO エッジ検出コールバック関数「GPIO.add_event_detect()」を使用した割り込み制御の例を記載します。

```
#黒色スイッチが押されたら、シャットダウンコマンドを実行
GPIO.add_event_detect(23, GPIO.FALLING, callback = shutdown, bouncetime = 2000)
#白色スイッチが押されたら、再起動コマンドを実行
GPIO.add_event_detect(24, GPIO.FALLING, callback = reboot, bouncetime = 2000)
```

シャットダウンコマンドを例に説明すると、
GPIO.add_event_detect(23, GPIO.FALLING, callback = shutdown, bouncetime = 2000)

① GPIO 番号: 23
② GPIO.FALLING: 立ち下がりエッジでイベント（黒色スイッチ押下）を検出
③ Callback = shutdown: コールバック（呼び出す）関数名を指定
④ Bouncetime = 2000: チャタリング防止のため、イベントを検出すると、指定時間（単位：ミリ秒）は次のイベントを呼び出さないように制御しています。

GPIO.add_event_detect()を使用することで、割り込み処理を実行できます。
このプログラムが稼働中であればいつでも、スイッチの押下を検出できます。

Copyright © 2024 YITJC All Rights Reserved.

参考～Python の例外処理～

ここでは、Python の例外処理の構文を紹介します。

例 1 は、すべての例外を補足して処理を行います。

どのような例外が発生しても同じ処理を行う場合に適しています。

～例 1～

```
try:
    # 実行するコード
    # 例外が発生した場合、ここから except ブロックに移る
except:
    # 例外が発生したときに実行されるコード
```

例 2 は、例外を特定して処理を行います。

特定の例外（ここでは、ValueError や IOError）に対する処理が決まっている場合に適しています。

～例 2～

```
try:
    # 実行するコード
    # 例外が発生した場合、ここから except ブロックに移る
except ValueError:
    # ValueError が発生したときに実行されるコード
except IOError as err:
    print('catch IOError:', err)
    print('catch IOError:', type(err))
    # IOError が発生したときに実行されるコード
    # 変数 err には例外発生時に出力されるエラーメッセージなどが格納され、エラー内容を確認できる
    # print(err)は、err のメッセージを表示
    # print(type(err))は、err のクラス（型情報）を表示
except:
    # 上記以外の例外が発生したときに実行されるコード
```

例 3 は、例外の発生によらず必ず最後に行う処理を「finally」ブロックに指定しています。

例外の発生によらず必ず実行したい処理がある場合に適しています。

～例 3～

```
try:
    # 実行するコード
    # 例外が発生した場合、ここから except ブロックに移る
except:
    # 例外が発生したときに実行されるコード
finally:
    # 必ず実行されるコード
```

Copyright © 2024 YITJC All Rights Reserved.

2.5 自動起動

Raspberry Pi には、プログラムを自動起動する方法として、

- /etc/rc.local
- Autostart
- crontab
- /etc/init.d
- Systemd

がありますが、本講座は「Systemd」を使用します。ここでは、「Systemd」という仕組みを使用してプログラムを自動起動する手順について説明します。

参考～Systemd について～

Systemd は、Raspbian Jessie（2015 年 4 月リリース）から推奨されている自動起動の方法です。サービスを作成して、プログラムの起動、シャットダウン、再起動を制御できます。

個々のサービスなどの設定は「Unit ファイル」と呼ばれ、「ファイル名.service」の形式で「/etc/systemd/system」ディレクトリに保存します。

参考～Unit ファイルの保存先～

Unit ファイルが保存されているディレクトリは、2 箇所あります。

/usr/lib/systemd/system/ : システムデフォルトの Unit ファイル保存先（基本的に使用しない）

/etc/systemd/system/ : 管理者がカスタマイズする場所（ここに作成・保存します）

同じ名前の設定ファイルがある場合、 /etc/systemd/system/ が優先されます

システムデフォルトの Unit ファイルを変更する場合は、 /usr/lib/systemd/system/ から /etc/systemd/system に設定ファイルをコピーして編集します。

ここからは、「シャットダウンサービス」を作成するための手順について説明します。

手順①～Unit ファイル作成～

「/etc/systemd/system」ディレクトリへ移動し、Unit ファイルを作成します。今回作成するファイルは、シャットダウンプログラム（5_Shutdown.py）を自動起動させるための Unit ファイルになります。

まず、「/etc/systemd/system」ディレクトリへ移動します。

cuu@raspberrypi: ~ $ cd /etc/systemd/system

```
cuu@raspberrypi:~ $ cd /etc/systemd/system
cuu@raspberrypi:/etc/systemd/system $
```

Copyright © 2024 YITJC All Rights Reserved.

続いて、Unit ファイルを作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit ファイルの中身を書き込んでください。

cuu@raspberrypi:/etc/systemd/system $ sudo nano shutdown.service

～参考 Unit ファイルの書き込み手順～

※本講座で使用する Unit ファイルは、共有サイト「1_Python¥etc_systemd_system」に保存されています。

① VS Code で Unit ファイルを開きます

② 必要に応じて編集します

※このとき、VS Code の画面で改行コードが LF「↓」になっていることを確認します。

```
1   [Unit]↓
2   Description=Shutdown/Reboot raspberry pi by GPIO button input↓
3   Wants=network.target↓
4   ↓
5   [Service]↓
6   ExecStart=/home/cuu/dev/5_Shutdown.py↓
7   Restart=on-failure↓
8   RestartSec=10s↓
9   ↓
10  [Install]↓
11  WantedBy=multi-user.target↓
```

③ VS Code からコピーして、Tera Term の画面上で貼り付けます

Tera Term 画面上で「右クリック」すると次の画面が表示されるので、「OK」をクリックしてください。

④ Tera Term 画面上で上書き保存を実行します

・編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

・「Enter」キーを押下して上書き保存を実行

・「Ctrl + X」キーを押して元の画面に戻る

Copyright © 2024 YITJC All Rights Reserved.

～Unit ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Shutdown/Reboot raspberry pi by GPIO button input
3	Wants=network.target
4	
5	[Service]
6	ExecStart=/home/cuu/dev/5_Shutdown.py
7	Restart=on-failure
8	RestartSec=10s
9	
10	[Install]
11	WantedBy=multi-user.target

- 1 行目～3 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文(動作に影響しませんが、分かりやすく記述するようにしてください。)
- 3 行目: 起動順序の指定
  Wants=network.target
  今回作成する Unit ファイルの起動条件として、ネットワーク接続を設定してあります。

Option	説明
After	起動順序の指定 ここで指定したサービスの後に実行される
Before	起動順序の指定 ここで指定したサービスの前に実行される
Requires	依存関係の指定 ここで指定したサービスが必要(Requires) 指定したサービス Unit が起動に失敗すると自分も起動しない ※Requires は After や Before の様な順序関係ではないので同時に並列で起動しようとするため、システム全体の起動時間を短縮する設計になっている
Wants	依存関係の指定 Requires との違いは、指定したサービスが起動に失敗しても自分は起動する

- 5 行目～8 行目: Service セクション
  起動するプログラム、起動条件など Unit ファイルの中で最も設定項目が多いセクションです。
  サービスの内容(どんな処理を実行させるか)を記述するセクションです。
- 6 行目: 自動起動するプログラムを指定

Copyright © 2024 YITJC All Rights Reserved.

- 7 行目: プロセスが停止した場合、プロセスの再起動条件を設定
  - ✓ Always　　常に再起動する
  - ✓ No　　再起動しない
  - ✓ on-success　　終了コードが 0 で再起動する
  - ✓ on-failure　　終了コードが 0 以外で再起動する
- 8 行目: 再起動するまでの待ち時間（秒）を設定
- 10 行目～11 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 11 行目: サービス Unit の依存関係を設定
  multi-user.target は、複数ユーザー環境のベースとなるターゲットで、多くのサービスがこのターゲットに紐付いて起動されます。（shutdown サービス Unit もこのターゲットに紐付けられます）
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

次のコマンドを実行すると、Raspberry Pi で動作しているサービスの依存関係が表示されます。
依存関係が表示された画面を閉じるには、「q」を押してください。

```
cuu@raspberrypi: ~ $ sudo systemctl list-dependencies
```

●：起動成功したサービス
●：起動失敗したサービス
○：その他の状態（無効化されたサービスなど）

Copyright © 2024 YITJC All Rights Reserved.

手順②～サービス再読み込み～

手順①で Unit ファイルを作成した後は、次のコマンドを実行してサービスを再読み込みします。

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順③～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable shutdown.service

手順④～サービス開始～

次のコマンドを実行してサービスを開始します。

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start shutdown.service

手順②～④まで実行したときの画面を表示します。

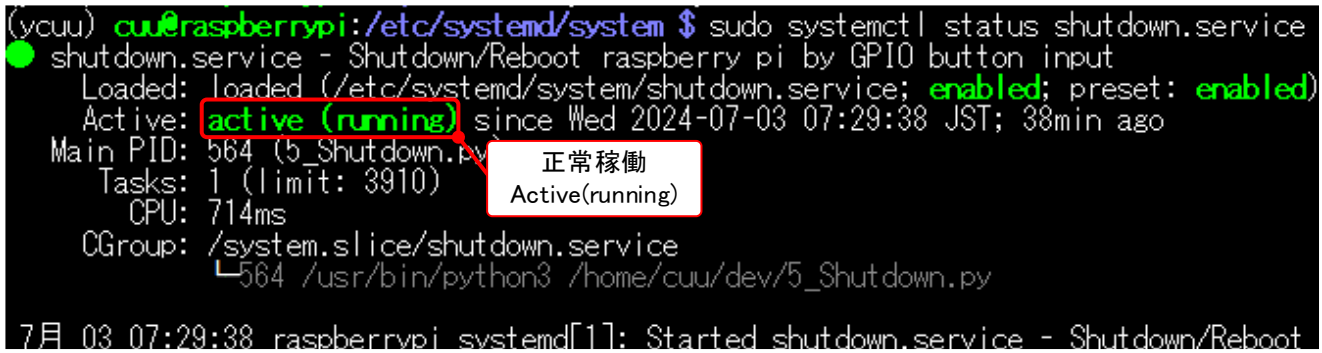
```
cuu@raspberrypi:~ $ sudo systemctl daemon-reload
cuu@raspberrypi:~ $ sudo systemctl enable shutdown.service
Created symlink /etc/systemd/system/multi-user.target.wants/shutdown.service
cuu@raspberrypi:~ $ sudo systemctl start shutdown.service
cuu@raspberrypi:~ $
```

手順⑤～サービス動作状況確認～

次のコマンドを実行してサービスの動作状況を確認します。

cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status shutdown.service

正常に稼働（自動起動）していれば、次の画面のようにステータスが active（runnning）と表示されます。



この画面を閉じるには、「q」を押してください。

手順⑥～動作確認～

5_Shutdown.py プログラムを実行しない状態で、再起動・シャットダウンが実行されることを確認してください。

なお、HAT Board における再起動・シャットダウンスイッチは、次のとおりです。

- ✓ 白色スイッチ: Raspberry Pi 再起動
- ✓ 黒色スイッチ: Raspberry Pi シャットダウン

今後はシャットダウンを実行する際、シャットダウンコマンドを入力する必要はなく、黒色スイッチを押すだけで Raspberry Pi をシャットダウンできます。

Copyright © 2024 YITJC All Rights Reserved.

参考～systemctl コマンド～

サービスファイルを稼働させる際に使用した systemctl コマンドについて、よく使用されるコマンドを紹介します。

- サービス再読み込み

cuu@raspberrypi: ~ $ sudo systemctl daemon-reload

- サービス開始（起動）

cuu@raspberrypi: ~ $ sudo systemctl start shutdown.service

- サービス再起動

cuu@raspberrypi: ~ $ sudo systemctl restart shutdown.service

- サービス停止

cuu@raspberrypi: ~ $ sudo systemctl stop shutdown.service

- サービス動作状況確認

cuu@raspberrypi: ~ $ sudo systemctl status shutdown.service

- サービスがアクティブ状態であるか確認

cuu@raspberrypi: ~ $ sudo systemctl is-active shutdown.service

- サービス自動起動有効化

cuu@raspberrypi: ~ $ sudo systemctl enable shutdown.service

- サービス自動起動無効化

cuu@raspberrypi: ~ $ sudo systemctl disable shutdown.service

Copyright © 2024 YITJC All Rights Reserved.

参考～改行コードの確認方法～

サービスファイルが正常に稼働しない場合、原因の 1 つとしてプログラムまたは Unit ファイルの改行コードが CR+LF（Windows）になっていることが考えられます。VS Code（Windows PC）で改行コードを確認できますが、ここでは Raspberry Pi（Tera Term 画面）で改行コードを確認する方法を紹介します。

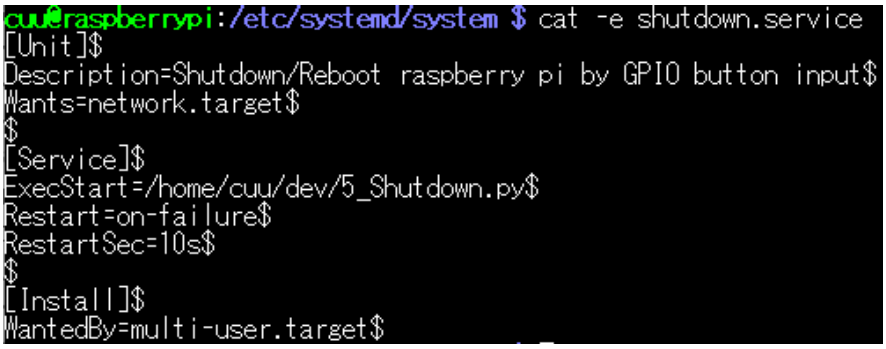
cat -e コマンドを実行すると、Raspberry Pi に保存されているプログラムやファイルなどの改行コードを確認することができます。

cuu@raspberrypi:/etc/systemd/system $ cat -e shutdown.service

cat -e コマンドを実行した結果を示します。

末尾に「$」が表示されていますが、これは改行コードが「LF」であることを意味しています。

改行コードが CR+LF の場合は、末尾に「^M$」と表示されるので、注意してください。



Copyright © 2024 YITJC All Rights Reserved.

第 3 章 データ取得～センサ～

あらゆるモノがインターネットにつながる IoT（Internet of Things）は、家電や自動車、住宅など、さまざまな産業で急速に普及しています。IoT が急速に普及した要因の一つとして、安価なセンサデバイスの普及が挙げられます。IoT 化に最適な条件（安価・小型・省電力）を満足するだけでなく、センサデバイスを制御するためのサンプルプログラムもデバイスメーカから提供されることが多いため、センサデバイスを使用した IoT システムの開発が効率的かつ迅速に行える環境が整っています。

ここでは、センサデバイスからデータを取得する方法について学習します。

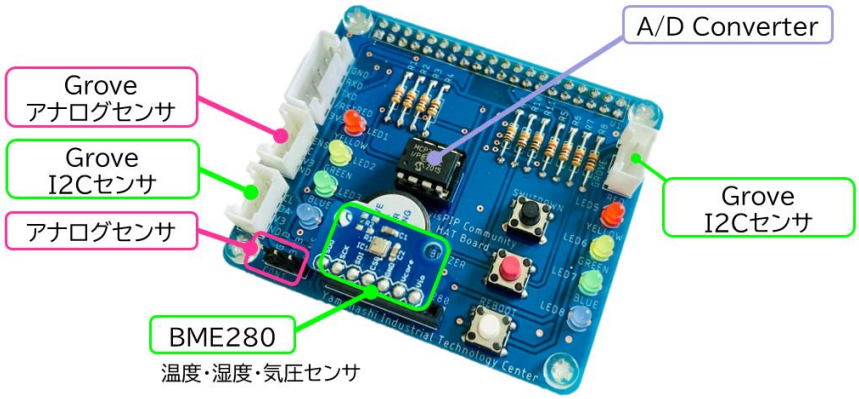
3.1 インターフェース

センサデバイスを扱うためには、Raspberry Pi と接続するための通信規格（通信インターフェース）について理解する必要があります。Raspberry Pi とセンサデバイスを接続する際に使用する代表的な通信規格を示します。

通信規格		説明
I2C	Inter Integrated Circuit	2 線式 同期式シリアル通信
SPI	Serial Peripheral Interface	4 線式 同期式シリアル通信
UART	Universal Asynchronous Receiver Transmitter	2 線式 非同期シリアル通信
1-Wire	One Wire Interface	1 線式 非同期シリアル通信

HAT Board は、Raspberry Pi に最大 5 個のセンサを接続できる構成になっています。

- ✓ BME280 （I2C: 温度・湿度・気圧センサ）
- ✓ アナログセンサ温度センサ LM35（SPI: アナログ式温度センサ）
- ✓ Grove I2C センサ（I2C: I2C インターフェースをもつ任意のセンサデバイスを接続可能）
- ✓ Grove アナログセンサ



参考～Grove シリーズ～

Groves シリーズは、Seed Studio 社が開発し、「センサ」や「スイッチ」などのデバイスを Grove 規格の 4 ピンコネクタで接続するだけで簡単に利用できるようにした規格です。現在、数十種類以上のデバイスが販売されています。HAT Board は、使用目的にあった Grove デバイスを購入し、接続するだけで簡単に利用できるように設計しています。

Copyright © 2024 YITJC All Rights Reserved.

I2C～Inter Integrated Circuit～

- 通信方式

  1つのマスタで複数のスレーブデバイスと通信することが可能です。同じ同期式シリアル通信の SPI と大きく異なる点は、個々のスレーブがアドレスを持っている点です。データの中にアドレスが含まれていて、マスタからアドレスを指定することで、複数のセンサデバイスとの通信を可能としています。

- 通信速度

  I2C は信号線が少ない点が特徴ですが、データ線（SDA）1 本で双方向通信を行うため、SPI 通信に比べて通信速度が遅くなります。

  ✓ 100kbps（デフォルト設定値）

  cuu@raspberrypi: ~ $ sudo nano /boot/firmware/config.txt

  コマンドを実行し、config.txt を編集し、

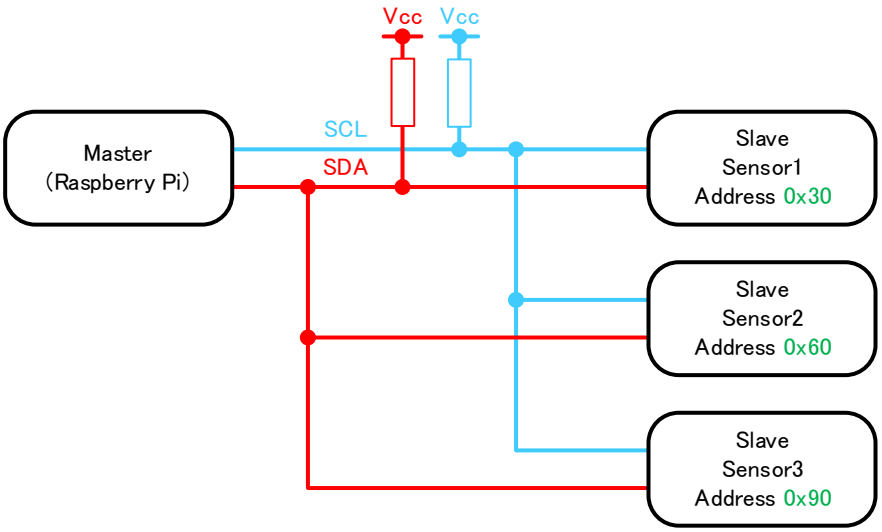
  dtparam=i2c_baudrate=400000

  を追加することで、通信速度を変更できます。（変更した場合は、再起動してください。）

- 通信に使用するポート（端子）

  ✓ SCL（Serial Clock: GPIO3）

  ✓ SDA（Serial Data: GPIO2）



Copyright © 2024 YITJC All Rights Reserved.

SPI～Serial Peripheral Interface～

- 通信方式

  1つのマスタで複数のスレーブデバイスと通信することが可能です。同じ同期式シリアル通信の I2C と大きく異なる点は、SS（Slave Select）によりセンサデバイスを指定する点です。Raspberry Pi には SS 端子が 2 個しかないため、SPI 通信で接続できるセンサデバイスは最大で 2 個です。

- 通信速度

  データ通信は、MISO/MOSI 信号線を使用して「片方向ずつ」行われるため、I2C 通信に比べて通信速度が速くなります。

  ✓ 125MHz（SCK 最大設定値）

  プログラム中に SPI 速度を記述することで、通信速度を設定できます。

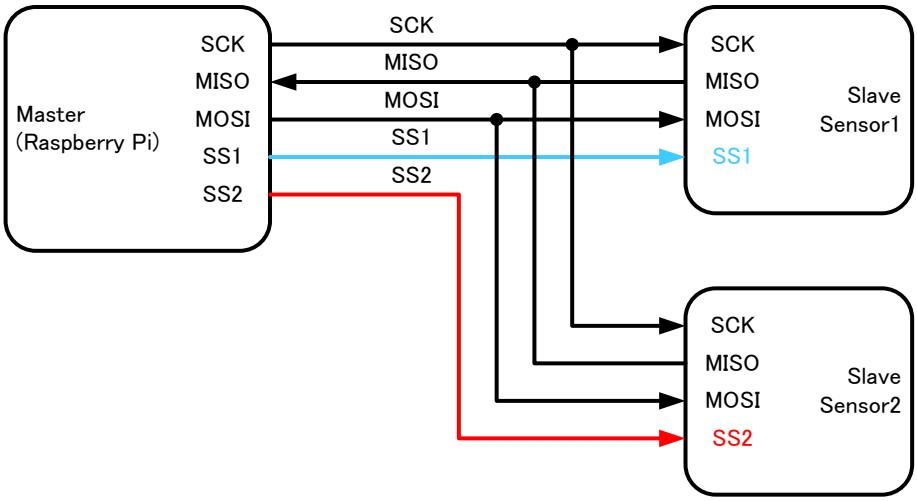
- 通信に使用する通信線

  ✓ SCK (Serial Clock: GPIO11)

  ✓ MISO (Master In Slave Out: GPIO9)

  ✓ MOSI (Master Out Slave In: GPIO10)

  ✓ SS (Slave Select: GPIO8)



Copyright © 2024 YITJC All Rights Reserved.

UART～Universal Asynchronous Receiver Transmitter～

- 通信方式

  I2C, SPI 通信とは異なり、非同期でシリアル通信を行います。そのため、信号線は送信用のデータラインと、受信用のデータラインの 2 線で通信を行い、クロック用の信号線は存在しません。

- 通信速度

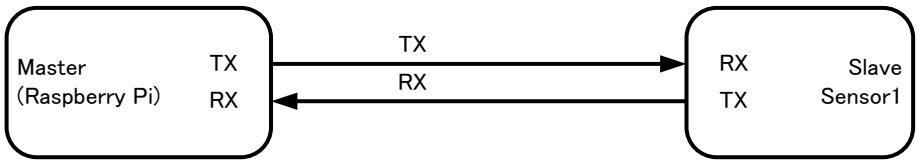
  UART にはクロック信号が無いため、通信する前にあらかじめ送受信するデバイス間で、通信速度（Baud Rate: ボーレート）を決めておきます。よく使用される Baud Rate を記載します。

  - ✓ 460800bps（最大設定値）
  - ✓ 115200bps
  - ✓ 19200bps
  - ✓ 9600bps

  プログラム中に Baud Rate を記述することで、通信速度を設定できます。

- 通信に使用する通信線

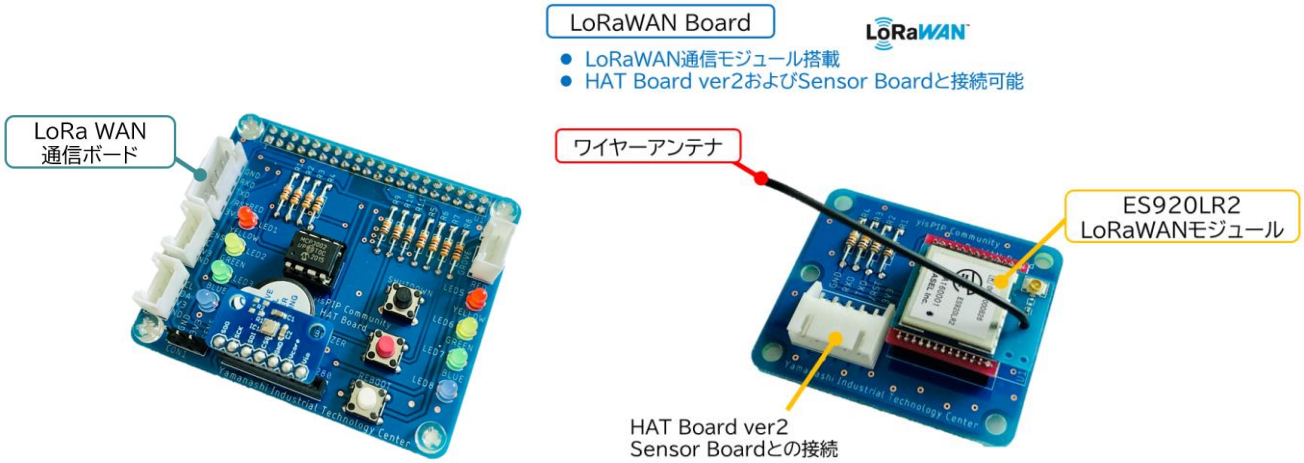
  - ✓ TX（Transmitter: GPIO14）
  - ✓ RX（Receiver: GPIO15）



- HAT Board の UART 通信ポート

  HAT Boad は、LPWA 通信の一種である「LoRa WAN」通信規格に対応した通信モジュールに接続するため、UART 通信用コネクタを用意しています。

  LPWA（Low Power Wide Area Network）は、省電力で数キロ～数十キロメートルの広範囲で通信を可能とする通信方法です。



Copyright © 2024 YITJC All Rights Reserved.

1-Wire～One Wire Interface～

- 通信方式

  1 本の信号線（電源供給を兼ねる）だけで双方向通信を行います。1-Wire を使用するセンサデバイスは端子数を削減できるため、パッケージの小型化が可能です。1-Wire の技術仕様によると、通信距離は最大で 500m まで可能とされています。I2C や SPI, UART 通信を採用するセンサデバイスは、PCB（基板内）や近距離での使用を前提としていますが、1-Wire は長距離通信可能な利点を生かして、遠隔地に複数設置することができます。

  ROM Code と呼ばれる重複しない固有の 64bit ID が割り当てられているため、同じ種類のセンサデバイスを同じ 1-wire 上に複数接続できます。（I2C の場合は、アドレスが重複するデバイスを I2C 通信ライン上に接続することはできません。）

- 通信速度

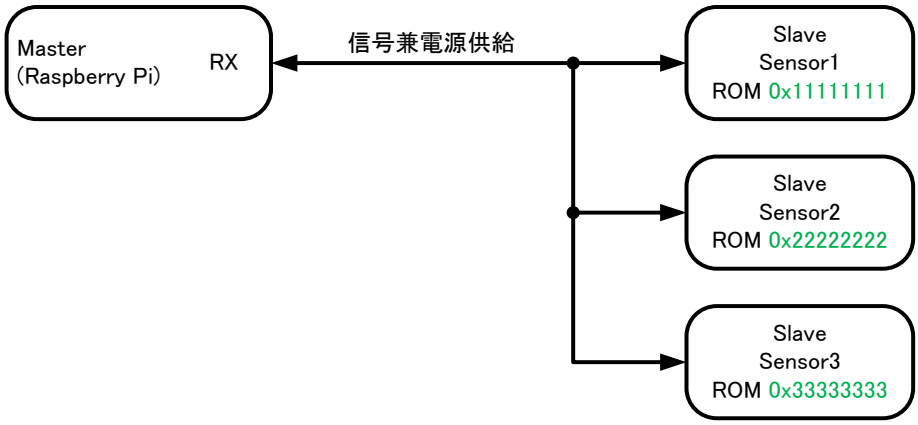
  1-Wire の通信速度は 2 種類用意されていて、センサデバイスごとに速度が決まっています。

  ✓ Standard-mode（約 15kbps）

  ✓ Overdrive-mode（約 100kbps）

- 通信に使用する通信線

  ✓ 1-Wire（GPIO14）



Copyright © 2024 YITJC All Rights Reserved.

3.2 Python ライブラリ

Python は、AI（Artificial Intelligence）や IoT（Internet Of Things）を含む多くの分野で最も利用されているプログラミング言語です。シンプルかつ読みやすい構文であるため、プログラミングの初心者でも学習しやすいプログラミング言語です。特に、Python には AI や IoT を含むさまざまな分野で使用できる「ライブラリ」が豊富です。豊富なライブラリをインストールして使用することで、高度な機能を簡単に実装できます。

ここでは、「cowsay」というライブラリを使用したプログラムを例に挙げて「ライブラリ」のインストールや使用方法を学びます。「cowsay」は、メッセージをつぶやく牛のアスキーアートを生成するライブラリです。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

6_cowsay.py

[プログラム内容]

プログラムを実行すると、メッセージをつぶやく牛のアスキーアートが表示されます。

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

cuu@raspberrypi: ~/dev $ chmod +x 6_cowsay.py

手順③～プログラム実行～

対象プログラムを実行します。

cuu@raspberrypi: ~/dev $ ./6_cowsay.py

プログラムを実行すると、エラーメッセージ「ModuleNotFoundError」が表示されます。

このエラーは、「cowsay」ライブラリが見つからない（インストールされていない）ことを意味しています。

```
cuu@raspberrypi: ~/dev $ ./6_cowsay.py
Traceback (most recent call last):
  File "/home/cuu/dev/./6_cowsay.py", line 4, in <module>
    import cowsay
ModuleNotFoundError: No module named 'cowsay'
```

「6_cowsay.py」を実行し、正しい結果（メッセージをつぶやく牛のアスキーアート）を得るためには、「cowsay」ライブラリをインストールする必要があります。続いて、ライブラリのインストール方法を説明します。

Copyright © 2024 YITJC All Rights Reserved.

参考～pip ツールの使用禁止～

Python のバージョン 3.9 以前は、パッケージのインストールを簡単にするために、pip（Pip Installs Package）という公式のパッケージ管理ツールが用意されていました。しかし、本講座で使用する Python のバージョン 3.11 は、pip ツールを使用したライブラリのインストールが禁止されています。

pip ツールを使用したインストール方法の禁止は、PEP（Python Enhancement Proposal）668 に記載されています。pip ツールを使用してインストールを試みると、次のエラーメッセージが表示されます。

```
cuu@raspberrypi:~/dev $ pip3 install cowsay
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    For more information visit http://rptl.io/venv

note: If you believe this is a mistake, please contact your Python installation
or OS distribution provider. You can override this, at the risk of breaking your
 Python installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.
```

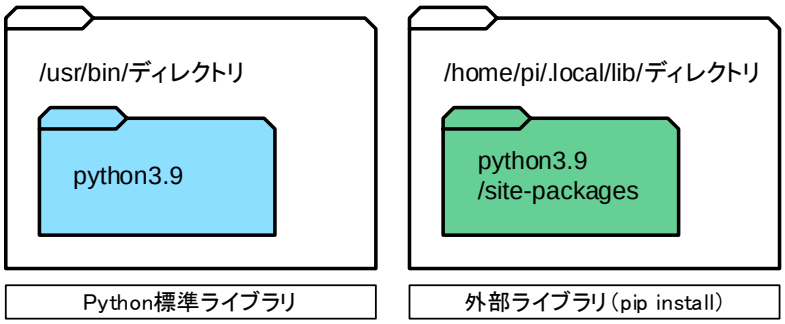
PEP668 で pip を使用したライブラリのインストールを禁止した理由は、Python の「標準ライブラリ」と「外部ライブラリ」の依存関係を管理して Python の安定性と信頼性を向上させることが目的です。

- 標準ライブラリ
  Python をインストールしたときから含まれているライブラリ
- 外部ライブラリ
  Python にもともとインストールされておらず、外部からインストールする必要があるライブラリ

従来の OS（Bullseye）における Python ライブラリのインストール先を図に示します。Bullseye は、Python3.9 が標準でインストールされていたため、pip コマンドによる外部ライブラリのインストールが禁止されていませんでした。

そのため、もともと Raspberry Pi にインストールされていた Python の標準ライブラリは、「usr/bin/pyhton3.9」ディレクトリに格納されています。

しかし、pip install コマンドを使用してインストールされた Python の外部ライブラリは、「/home/pi/.local/lib/python3.9/site-packages」に格納されています。



Copyright © 2024 YITJC All Rights Reserved.

すでに標準ライブラリにインストールされているライブラリを pip install コマンドを使用してインストールした場合、Raspberry Pi 内の別々のディレクトリに同じライブラリが存在することになります。また、標準ライブラリと同じ名前のライブラリ（機能は異なるライブラリ）をインストールしてしまう可能性もあります。

ライブラリの競合や衝突が発生した場合、Python で開発したシステムの安定性は失われ、信頼性が著しく低下します。このような状況を避けるために、PEP668 で pip install コマンドによるインストール法が廃止されました。

参考～ライブラリインストール方法～

PEP668 では、pip install コマンドによるエラー回避策として、次の提案がされています。

提案①：apt install python3-xyz でインストール

提案②：pip install xyz --break-system-packages でインストール

提案③：仮想環境を構築してインストール

xyz は、インストールするライブラリの名前です。

提案①のインストールコマンド「apt install python3」は、システム全体にライブラリをインストールする方法です。システム全体にインストールするため、ライブラリの重複や競合を避けることができます。しかし、特定のプロジェクトに特化したライブラリの管理はできません。（システム全体で同じライブラリを使用するため、特定のプログラムで特定のバージョンのライブラリを使用するといった管理ができません。）

「pip」コマンドは、Pypl（Python Package Index）で公開されているライブラリをインストールするコマンドです。

しかし、「apt install python3」コマンドは、システム全体のソフトウェア管理ツールを利用して「Python3」に関連したライブラリをインストールする方法です。そのため、Pypl で公開されているライブラリすべてをカバーしておらず、pip コマンドでなければインストールできないライブラリが数多くあります。

Pypl で公開されている豊富なライブラリを活用してシステムを構築することを考えると、提案①の内容は推奨できません。

Pypl: https://pypi.org/

提案②のインストール方法は、PEP668 でも明らかに「リスク」がある方法として紹介されています。具体的には、「--break-system-packages」オプションを追加してインストールするため、同じ名前の標準ライブラリが存在する場合、標準ライブラリが強制的に上書きされます。そのため、標準ライブラリを使用している別のプログラムが動作しなくなってしまうなどの問題が発生する可能性があります。

提案③は、仮想環境を構築してライブラリをインストールする方法です。仮想環境は、プロジェクトごとに独立して構築できるため、それぞれのプロジェクトで異なるバージョンのライブラリを使用することができます。これにより、ライブラリの競合や衝突を回避し、プロジェクト（システム）の安定性を確保し、信頼性を向上することができます。

本講座では、**提案③**の内容でライブラリのインストール環境を構築します。

Copyright © 2024 YITJC All Rights Reserved.

手順④～仮想環境構築～

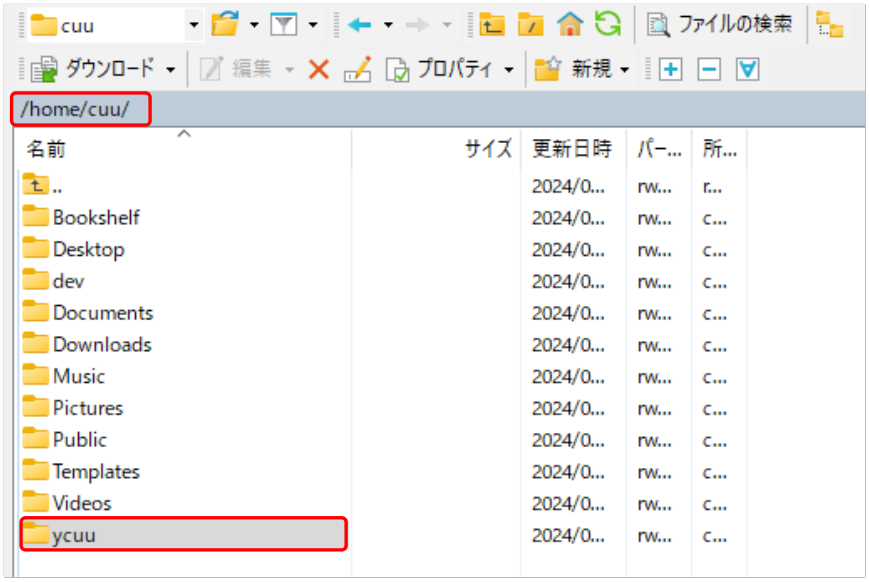
次のコマンドを実行し、Python3.11 の仮想環境を構築します。

注意

ホームディレクトリ（/home/cuu）に仮想環境を構築するため、必ずホームディレクトリでコマンドを実行してください。

cuu@raspberrypi: $ python3.11 -m venv ycuu

このコマンドを実行すると、ホームディレクトリに仮想環境「ycuu」が構築されます。



さらに、「/home/cuu/ycuu/bin」ディレクトリを確認すると、pip ツールや Python の環境が整備されていることがわかります。

続いて、次のコマンドを実行して構築した仮想環境をアクティブにします。

cuu@raspberrypi: $ source ycuu/bin/activate

このコマンドを実行すると、仮想環境（ycuu）の名前が表示されるようになります。

（ycuu）の表示があるときは、構築した仮想環境のもとで実行していることになります。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～ライブラリインストール～

仮想環境から pip3 コマンドを使用して「cowsay」ライブラリをインストールします。

(ycuu) cuu@raspberrypi: $ pip3 install cowsay

「Successfully installed」の表示でインストールされたことを確認できます。

```
(ycuu) cuu@raspberrypi:~ $ pip3 install cowsay
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting cowsay
  Downloading cowsay-6.1-py3-none-any.whl (25 kB)
Installing collected packages: cowsay
Successfully installed cowsay-6.1
```

参考～pip と pip3～

仮想環境にインストールされた pip ツールを確認すると、「pip」コマンドと「pip3」コマンドがあります。

「pip」、「pip3」コマンドを使用する場合は、使用している「Python」によって使い分けなければいけません。

具体的に説明すると、

- pip コマンドでパッケージ（xxxxx）をインストールした場合

  (ycuu) cuu@raspberrypi: ~ $ pip install xxxxx

  pip コマンドでインストールされたパッケージは、「/home/cuu/ycuu/bin/」ディレクトリにインストールされている「python」に対してインストールされたことになります。

  そのため、「/home/cuu/ycuu/bin/」ディレクトリにインストールされている「python3」でプログラムを実行する場合は、pip コマンドでインストールしたパッケージ（xxxxx）を使用することができません。

- pip3 コマンドでパッケージ（xxxxx）をインストールした場合

  (ycuu) cuu@raspberrypi: ~ $ pip3 install xxxxx

  pip3 コマンドでインストールされたパッケージは、「/home/cuu/ycuu/bin/」ディレクトリにインストールされている「python3」に対してインストールされたことになります。

  そのため、「python3」環境で開発することを明示するため、本講座は「pip3」コマンドを使用して必要なパッケージをインストールします。

手順⑥～プログラム実行～

「6_cowsay.py」プログラムを実行するために必要な「cowsay」ライブラリをインストールしたので、あらためてプログラムを実行します。

(ycuu) cuu@raspberrypi: $ cd dev

(ycuu) cuu@raspberrypi: ~/dev $ ./6_cowsay.py

プログラムを実行すると、次の画面が表示されます。

```
(ycuu) cuu@raspberrypi:~ $ cd dev
(ycuu) cuu@raspberrypi:~/dev $ ./6_cowsay.py
| Y-CUU |
  =====
     ¥
      ¥
       ^__^
       (oo)¥_______
       (__)¥       )¥/¥
           ||----w |
           ||     ||
```

Copyright © 2024 YITJC All Rights Reserved.

手順④で仮想環境を構築した際、次のコマンドを実行して仮想環境をアクティブに設定しました。

(ycuu) cuu@raspberrypi: $ source ycuu/bin/activate

このコマンドで設定した「アクティブ」な状態は、Raspberry Pi の電源を切ると仮想環境から抜けてしまいます。

Raspberry Pi に構築したシステムを自動起動するためには、Raspberry Pi が起動したときに自動で仮想環境をアクティブに設定する必要があります。

次の手順では、Raspberry Pi が起動したときに自動で仮想環境をアクティブに設定する方法を説明します。

参考

次のコマンドを実行すると仮想環境から抜けることができます。

(ycuu) cuu@raspberrypi: ~/dev $ deactivate

```
(ycuu) cuu@raspberrypi:~/dev $ deactivate
cuu@raspberrypi:~/dev $
```

手順⑦～.bashrc ファイル確認～

ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

```
cuu@raspberrypi:~ $ ls -a
.               .bash_logout   .config                  .xsession-errors      Documents  Public
..              .bash_profile  .local                   .xsession-errors.old  Downloads  Templates
.Xauthority     .bashrc        .profile                 Bookshelf             Music      Videos
.bash_history   .cache         .sudo_as_admin_successful Desktop              Pictures   dev
```

「.bashrc」はログイン（起動）時に一度だけ実行するコマンドを記述します。ここに「source ycuu/bin/activate」コマンドを記述しておくと、Raspberry Pi が起動したときに仮想環境が自動で「アクティブ」に設定されます。

(ycuu) cuu@raspberrypi: $ nano .bashrc

「.bashrc」ファイルの画面が起動したら、末尾に「source ycuu/bin/activate」コマンドを追加（記述）します。

記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

```
  GNU nano 7.2                                .bashrc *
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

source ycuu/bin/activate
```

Copyright © 2024 YITJC All Rights Reserved.

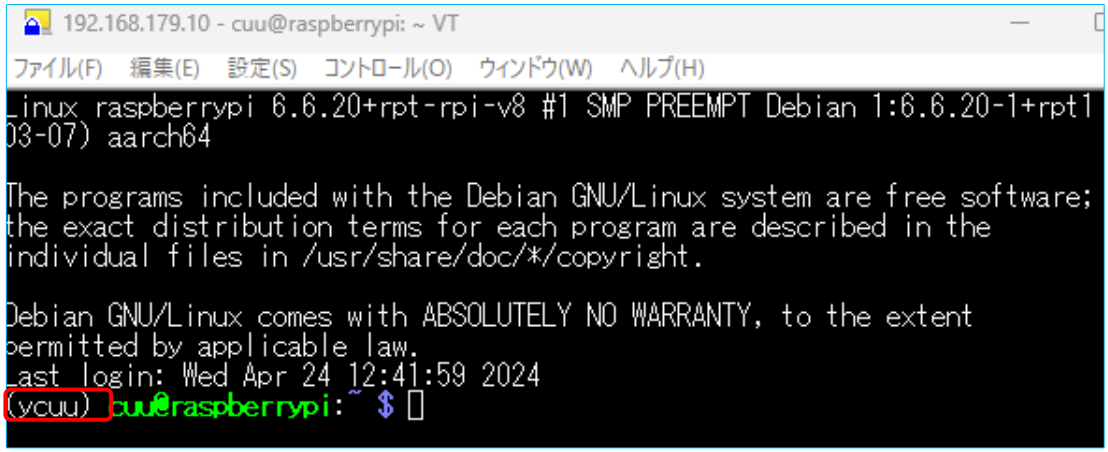
手順⑧～再起動～

次のコマンドを実行し、Raspberry Pi を再起動します。

(ycuu) cuu@raspberrypi: ~ $ sudo reboot

```
(ycuu) cuu@raspberrypi:~ $ sudo reboot
```

再起動後、Tera Term を使用して Raspberry Pi と ssh 接続したときに仮想環境（ycuu）になっていることを確認してください。



手順⑨～環境変数 PATH 確認～

次のコマンドを実行して、環境変数（システムがコマンドを実行する際に検索するディレクトリのリスト）を表示します。この中に、「/home/cuu/ycuu/bin」が追加されていることを確認してください。

(ycuu) cuu@raspberrypi: $ echo $PATH

```
(ycuu) cuu@raspberrypi:~ $ echo $PATH
/home/cuu/ycuu/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/local/games:/usr/games
```

手順⑩～ライブラリー覧確認～

次のコマンドを実行し、pip3 コマンドでインストールしたライブラリー覧を確認します。

(ycuu) cuu@raspberrypi: $ pip3 list

手順⑤でインストールした cowsay が追加されることを確認できます。

```
(ycuu) cuu@raspberrypi:~ $ pip3 list
Package    Version
---------- -------
cowsay     6.1
pip        23.0.1
setuptools 66.1.1
```

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding: utf-8
3	
4	import cowsay
5	
6	def main():
7	#牛がつぶやくメッセージを設定
8	message = "Y-CUU"
9	#cowsayを使用してメッセージをつぶやく
10	cowsay.cow(message)
11	
12	if __name__ == "__main__":
13	main()

・ 1 行目: Shebang（シバン、シェバン）

このプログラムを実行する Python 環境を指定します。

#!/home/cuu/ycuu/bin/python3.11

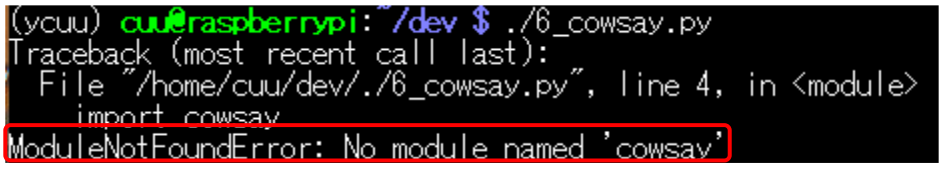
pip コマンドによるライブラリインストールを可能とするため、仮想環境を構築しました。

ここでは、仮想環境にある「Python」を指定する必要があります。

第 1 章で学習したとおり、Raspberry Pi にあらかじめインストールされている「Python」は、「/usr/bin/python3」ディレクトリにあります。第 2 章までは、外部ライブラリをインストールする必要がなかったため、Shebang の指定は次のとおりでした。

#!/usr/bin/python3

この指定（#!/usr/bin/python3）でプログラムを実行すると、「ModuleNotFoundError」が発生します。



参考

Python3 を指定してプログラムを実行した場合は、「ModuleNotFoundError」が発生しません。

cuu@raspberrypi:~/dev $ python3 6_cowsay.py

python3 と指定することで、Raspberry Pi デフォルトの Python（/usr/bin/python3）を使用することを宣言していますが、プログラム中に Shebang が記載されている場合は、Shebang に記載されている環境が優先されるため、エラーは発生しません。

Copyright © 2024 YITJC All Rights Reserved.

- 2 行目: マジックコメント
- 4 行目: cowsay ライブラリを import
- 6 行目～10 行目: main 関数
- 8 行目: 牛がつぶやくメッセージを設定
- 10 行目: cowsay ライブラリを使用してメッセージを牛がつぶやく
- 12 行目～13 行目: main 関数実行

Copyright © 2024 YITJC All Rights Reserved.

参考～pipx コマンドの利用～

PEP668 では、pip install コマンドによるエラー回避策として、仮想環境を構築してライブラリをインストールする方法が提案されています。ここでは、pipx コマンドを使用した仮想環境の構築について解説します。

pipx コマンドは、仮想環境を構築してライブラリをインストールする方法です。仮想環境をプロジェクトごとに独立して構築するため、それぞれのプロジェクトで異なるバージョンのライブラリを使用することができます。

これにより、ライブラリの競合や衝突を回避し、プロジェクトの安定性と信頼性を確保できます。

手順①～pipx インストール～

仮想環境にライブラリをインストールするツール「pipx」をインストールします。

cuu@raspberrypi: $ sudo apt update

cuu@raspberrypi: $ sudo apt install pipx

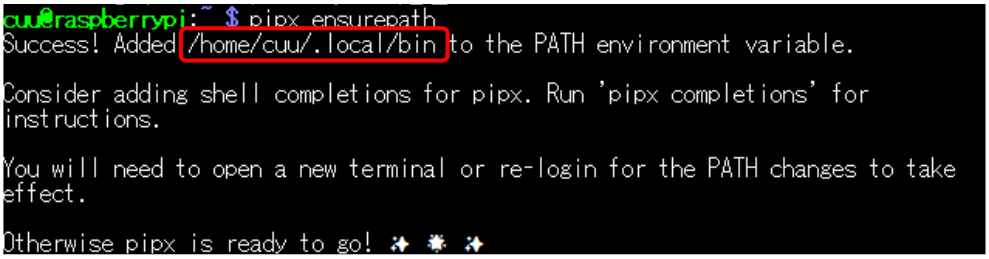
途中でインストールの続行を聞かれるので、「y」を入力して実行（Enter）してください。

手順②～仮想環境構築～

仮想環境を構築します。

cuu@raspberrypi: $ pipx ensurepath

「/home/cuu/.local/bin」ディレクトリが構築されました。



重要

注意すべき点は、このときに「Python」の仮想環境が同時に作成されているという点です。つまり、ライブラリをインストールしたときに作成された仮想環境に、そのライブラリを実行するための「Python」の環境が構築されます。

「/home/cuu/.local/bin」ディレクトリの構造を使用して、pipx を使用してライブラリをインストールした場合の「Python」実行環境の違いを説明します。

このディレクトリ構造は、メッセージをつぶやく牛のアスキーアートを生成するライブラリ「cowsay」と、

テキストを Ascii アートで表示するライブラリ「pyfiglet」を pipx コマンドでインストールした後のディレクトリ構造を示しています。

手順②「pipx ensurepath」コマンドを実行すると、仮想環境「/venvs」の下層にライブラリ毎に Python の実行環境が生成されます。これにより、ライブラリに対応した Python のバージョン管理が可能になります。

しかし、ライブラリを使用したプログラムを実行する際は、必ずそのライブラリをインストールした（仮想）環境に構築された Python で実行しなければなりません。

Copyright © 2024 YITJC All Rights Reserved.

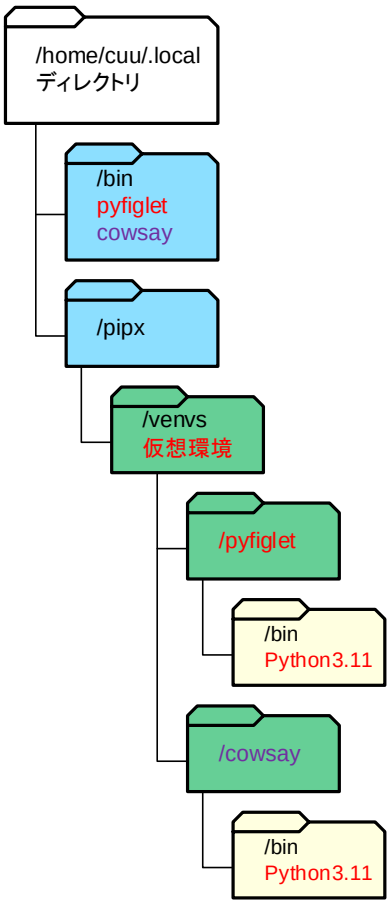
例えば、「pyfiglet」ライブラリを使用したプログラムを実行するためには、プログラムの1行目 Shebang（シェバン）に、Python の環境を指定しなければなりません。

#!/home/cuu/.local/pipx/venvs/pyfiglet/bin/python3.11

別のライブラリ「cowsay」を使用したプログラムを実行する場合は、次のとおり Python の環境を指定します。

#!/home/cuu/.local/pipx/venvs/cowsay/bin/python3.11

さらに、複数の外部ライブラリを含むプログラムの場合は、そのプログラム専用の仮想環境を構築してからライブラリをインストールする必要があります。



手順③～.bashrc ファイル確認～

ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を確認します。

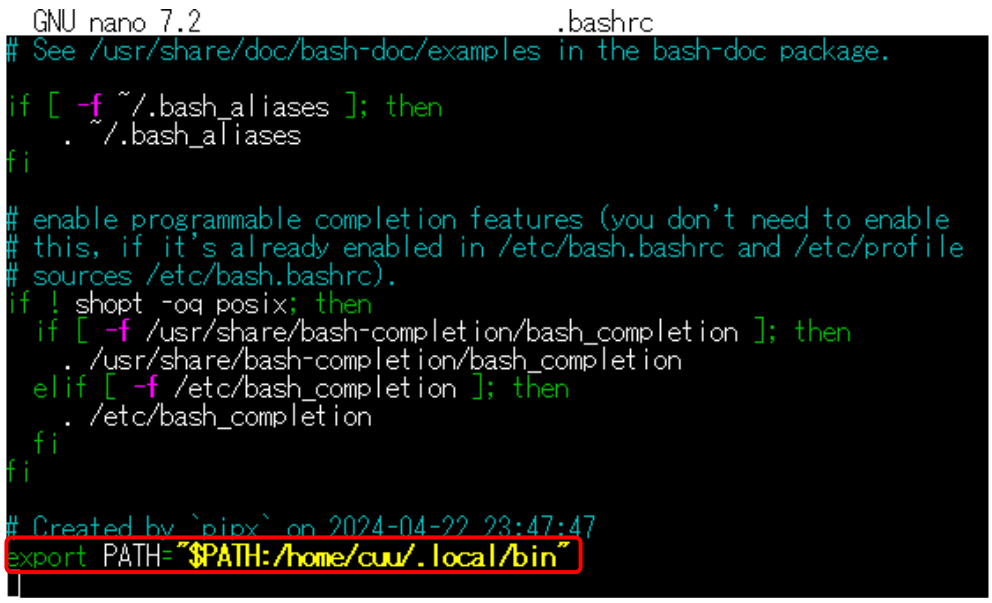
```
cuu@raspberrypi:~ $ ls -a
.               .bash_logout   .config                  .xsession-errors      Documents  Public
..              .bash_profile  .local                   .xsession-errors.old  Downloads  Templates
.Xauthority     .bashrc        .profile                 Bookshelf             Music      Videos
.bash_history   .cache         .sudo_as_admin_successful Desktop              Pictures   dev
```

「.bashrc」は、ホームディレクトリ（/home/cuu）に保存されているファイルで、環境変数などが設定されており、起動時に読み込まれます。

Copyright © 2024 YITJC All Rights Reserved.

cuu@raspberrypi: $ nano .bashrc

「.bashrc」ファイルの画面が起動したら、末尾に「pipx」の PATH が追加されていることを確認します。



「Ctrl」+「x」でこの画面を閉じます。

続いて、次のコマンドを実行して設定した内容を反映します。

cuu@raspberrypi: $ source ~/.bashrc

「~」の部分（チルダ）は、「Shift」+「～」で入力します。

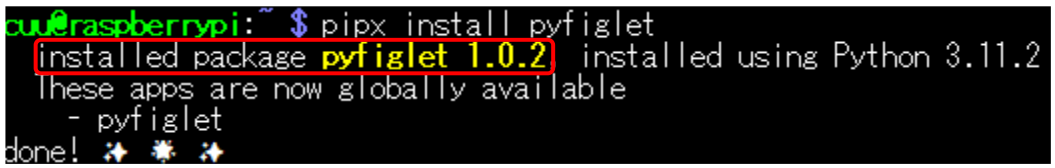
最後に、次のコマンドを実行して、環境変数（システムがコマンドを実行する際に検索するディレクトリのリスト）を表示します。この中に、「/home/cuu/.local/bin」が追加されていることを確認してください。

cuu@raspberrypi: $ echo $PATH

手順④～pyfiglet ライブラリインストール～

次のコマンドを実行し、pyfiglet ライブラリをインストールします。

cuu@raspberrypi: $ pipx install pyfiglet



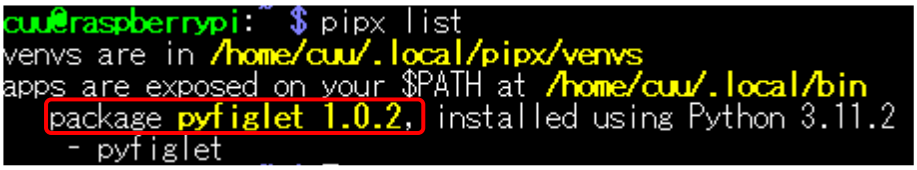
Copyright © 2024 YITJC All Rights Reserved.

手順⑤～ライブラリー覧確認～

次のコマンドを実行し、pipx でインストールしたライブラリの一覧を確認します。

cuu@raspberrypi: $ pipx list

手順⑦でインストールした pyfiglet が追加されることを確認できます。



手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

7_pyfiglet.py

[プログラム内容]

プログラムを実行すると、メッセージをつぶやく牛のアスキーアートが表示されます。

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

cuu@raspberrypi: $ cd dev

cuu@raspberrypi: ~/dev $ chmod +x 7_pyfiglet.py

手順⑧～プログラム実行～

「7_pyfiglet.py」プログラムを実行するために必要な「pyfiglet」ライブラリをインストールしたので、あらためてプログラムを実行します。

cuu@raspberrypi: ~/dev $ ./7_pyfiglet.py

プログラムを実行するとテキスト(Y-CUU)が Ascii アートで表示されます。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/.local/pipx/venvs/pyfiglet/bin/python3.11
2	# coding:utf-8
3	
4	import pyfiglet
5	
6	def main():
7	#ASCIIアートに変換するテキストを指定する
8	text_to_convert = "Y-CUU"
9	#テキストをASCIIアートに変換して表示する
10	ascii_art = pyfiglet.figlet_format(text_to_convert)
11	print(ascii_art)
12	
13	if __name__ == "__main__":
14	main()

- 1 行目: Shebang（シバン、シェバン）
  このプログラムを実行する Python 環境を指定します。
  #!/home/cuu/.local/pipx/venvs/pyfiglet/bin/python3.11
  pipx コマンドで構築した仮想環境を指定しないと、インストールしたはずのライブラリが「見つからない」というエラーが発生します。
  #!/usr/bin/env python3 と指定した場合は、「ModuleNotFoundError」が発生します。
  この場合は、「pyfiglet」ライブラリをインストールしていない「Python」、つまり「/usr/bin/python3」ディレクトリにある「Python」でプログラムを実行しているため、エラーが発生しています。

```
cuu@raspberrypi:~/dev $ ./7_pyfiglet.py
Traceback (most recent call last):
  File "/home/cuu/dev/./7_pyfiglet.py", line 4, in <module>
    import pyfiglet
ModuleNotFoundError: No module named 'pyfiglet'
```

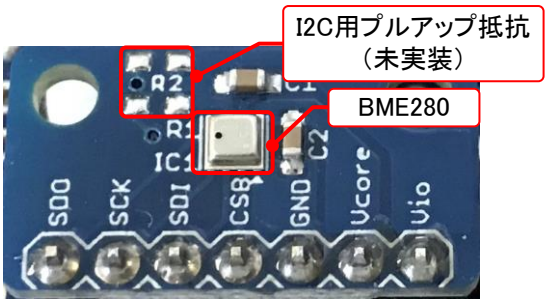
- 2 行目: マジックコメント
- 4 行目: pyfiglet ライブラリを import
- 6 行目～11 行目: main 関数
- 8 行目: Ascii アートに変換するテキストを設定
- 10 行目: テキストデータを Ascii アートに変換
- 11 行目: Ascii アートに変換したデータを print 表示
- 13 行目～14 行目: main 関数実行

Copyright © 2024 YITJC All Rights Reserved.

3.3 I2C～BME280 センサ～

HAT Board に実装されている「温度・湿度・気圧センサ BME280」のデータ取得を行います。BME280 は、BOSCH 社の MEMS センサです。Raspberry Pi と I2C 通信インターフェースで接続します。

I2C 通信用のプルアップ抵抗（R1, R2）は、未実装の状態で使用します。Raspberry Pi は、基板上に I2C のプルアップ抵抗が実装済であるため、センサデバイス側でプルアップ抵抗を実装する必要はありません。



手順①～I2C アドレスの確認～

I2C 接続するセンサデバイスは、データシートなどに I2C のアドレス情報が記載されています。今回使用する BME280 は、

- ✓ SDO 端子を GND 接続: 0x76（HAT Board の回路は、0x76 に設定しています）
- ✓ SDO 端子を Vio 接続: 0x77

と 2 種類のアドレスを設定することができます。

同じセンサデバイスにアドレスを 2 種類設定することで、Raspberry Pi に同じセンサデバイスを 2 個接続することができます。

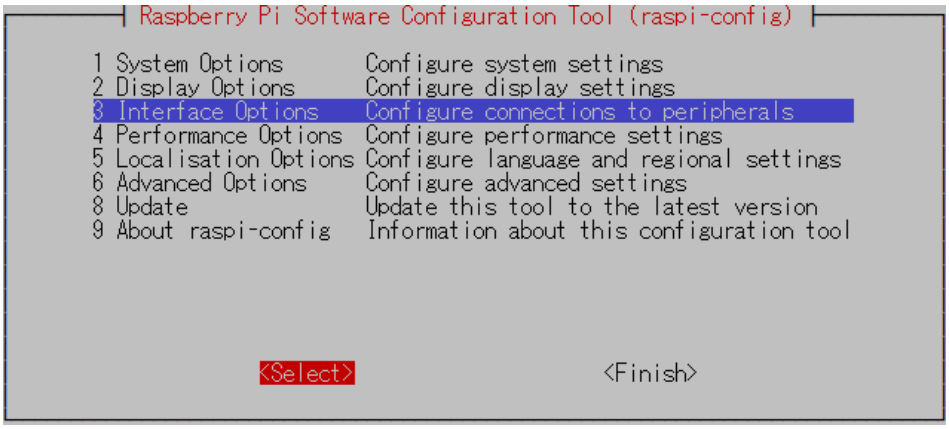
手順②～I2C 有効設定～

raspi-config コマンドを実行し、Raspberry Pi の I2C 通信インターフェースを有効にします。

(ycuu) cuu@raspberrypi: ~ $ sudo raspi-config

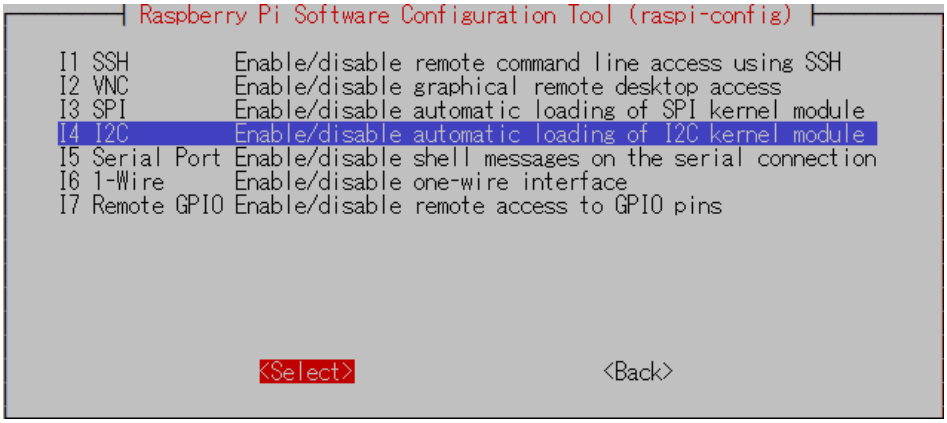
「3.Interface Options」を選択します。

Raspi-config 画面においては、「メニュー選択」は「→キー」、「カーソル移動」は「tab キー」、「決定」は「Enter キー」で操作します。

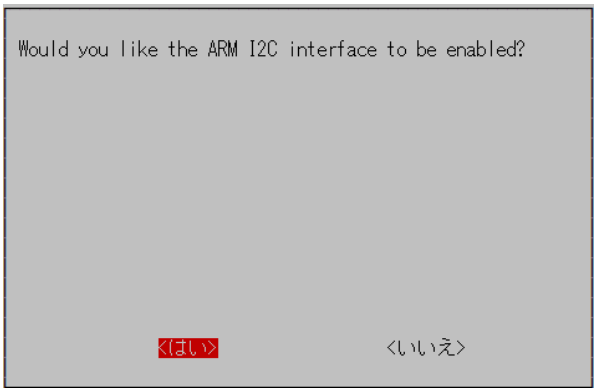


Copyright © 2024 YITJC All Rights Reserved.

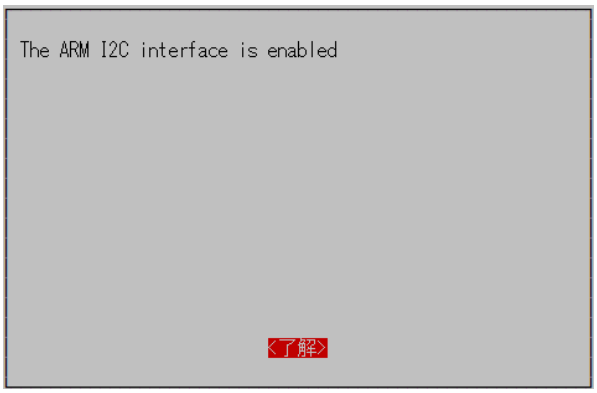
「I4 I2C」を選択してください。



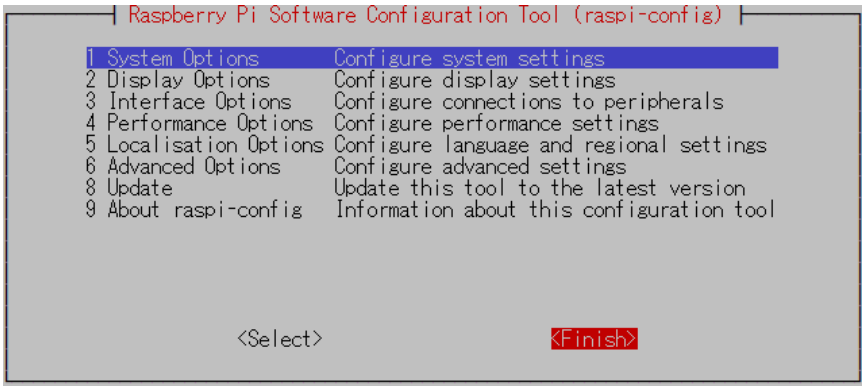
「はい」を選択し、「Enter」を押してください。



「了解」を選択し、「Enter」を押してください。



最初の画面に戻り、「Finish」を選択(Enter)して Raspi-config 画面を終了してください。



以上で、Raspberry Pi の I2C 通信インターフェースが有効(使用可能な状態)になりました。

Copyright © 2024 YITJC All Rights Reserved.

手順③～I2C 接続確認～

次のコマンドを実行し、Raspberry Pi に接続されている I2C デバイスを表示します。

(ycuu) cuu@raspberrypi: ~ $ sudo i2cdetect -y 1

アドレス 0x76 に I2C デバイスが接続されていることが分かります。

```
(ycuu) cuu@raspberrypi:~/dev $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

手順④～I2C レジスタ値確認～

次のコマンドを実行し、I2C デバイスのレジスタ値を出力します。

(ycuu) cuu@raspberrypi: ~ $ i2cdump -y 1 0x76

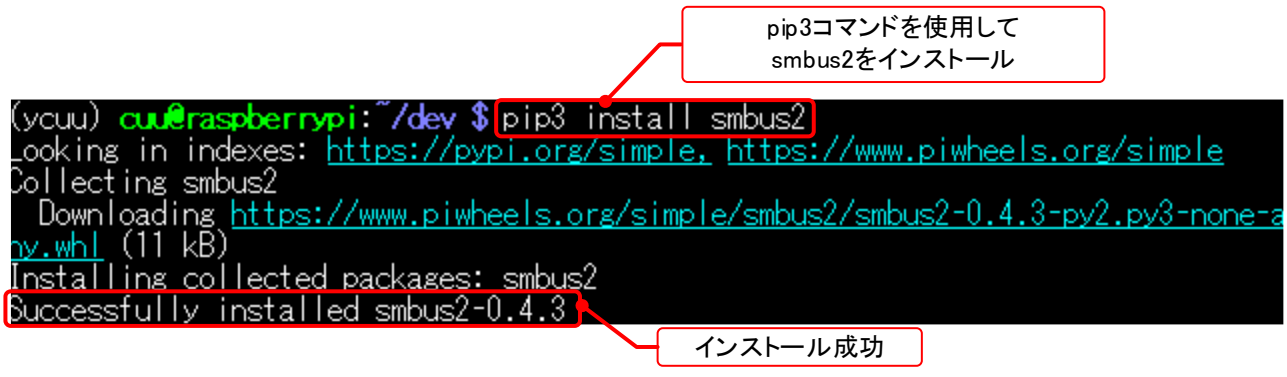
アドレス 0x80～0xff に何らかのデータが書き込まれていることを確認できますが、この状態では温度、湿度、気圧の値を読み取ることができません。

```
(ycuu) cuu@raspberrypi:~/dev $ i2cdump -y 1 0x76
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: 82 70 89 54 8a 5a 02 06 12 6f 10 69 32 00 92 8f    ?p?T?Z???o?i2.??
90: b2 d6 d0 0b 60 16 a9 00 f9 ff ac 26 0a d8 bd 10    ????`??.?.?&????
a0: 00 4b c8 00 00 00 00 00 00 00 00 33 00 00 c0    .K?.........3..?
b0: 00 54 00 00 00 00 60 02 00 01 ff ff 1f 60 03 00    .T....`?.?..?`?.
c0: 00 00 00 ff 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    `...............
e0: 00 93 01 00 0e 27 03 1e a7 41 ff ff ff ff ff ff    .??.?'???A......
f0: ff 00 00 00 00 00 00 80 00 00 80 00 00 80 00 80    .......?..?..?.?
```

手順⑤～smbus2 インストール～

I2C 通信を制御するためのパッケージ smbus2（System Management Bus2）をインストールします。

(ycuu) cuu@raspberrypi: ~ $ pip3 install smbus2



Copyright © 2024 YITJC All Rights Reserved.

手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

8_BME280.py

[プログラム内容]

プログラムを実行すると、画面上に温度、湿度、気圧データが表示されます。

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

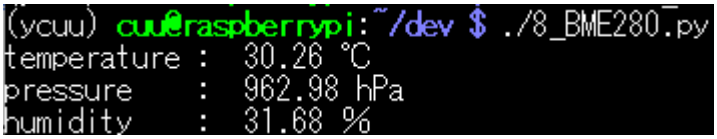
(ycuu) cuu@raspberrypi: ~/dev $ chmod +x 8_BME280.py

手順⑧～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./8_BME280.py

温度、湿度、気圧データが表示されることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding:utf-8
3	
4	from smbus2 import SMBus
5	
6	#I2C Address設定
7	bus_number = 1
8	i2c_address = 0x76
9	bus = SMBus(bus_number)
10	
11	#データ格納変数定義
12	digT = []
13	digP = []
14	digH = []
15	t_fine = 0.0
16	
17	#config, ctrl_meas, ctrl_muレジスタ設定
18	def init():
19	#レジスタ書き込み値
20	osrs_t = 1 #Temperature oversampling x 1
21	osrs_p = 1 #Pressure oversampling x 1
22	osrs_h = 1 #Humidity oversampling x 1
23	mode = 3 #Normal mode
24	t_sb = 5 #Tstandby 1000ms
25	filter = 0 #Filter off
26	spi3w_en = 0 #3-wire SPI Disable
27	#レジスタ設定
28	ctrl_meas_reg = (osrs_t << 5) | (osrs_p << 2) | mode
29	config_reg = (t_sb << 5) | (filter << 2) | spi3w_en
30	ctrl_hum_reg = osrs_h
31	#レジスタアドレス指定
32	writeReg(0xF2,ctrl_hum_reg)
33	writeReg(0xF4,ctrl_meas_reg)
34	writeReg(0xF5,config_reg)
35	
36	#Data(レジスタ設定値)書き込み
37	def writeReg(reg_address, data):

Copyright © 2024 YITJC All Rights Reserved.

```
38      bus.write_byte_data(i2c_address,reg_address,data)
39
40  #Calibration (補正データ取得)
41  def get_calib_param():
42      calib = []
43
44      for i in range (0x88,0x88+24):
45          calib.append(bus.read_byte_data(i2c_address,i))
46      calib.append(bus.read_byte_data(i2c_address,0xA1))
47      for i in range (0xE1,0xE1+7):
48          calib.append(bus.read_byte_data(i2c_address,i))
49
50      digT.append((calib[1] << 8) | calib[0])
51      digT.append((calib[3] << 8) | calib[2])
52      digT.append((calib[5] << 8) | calib[4])
53      digP.append((calib[7] << 8) | calib[6])
54      digP.append((calib[9] << 8) | calib[8])
55      digP.append((calib[11]<< 8) | calib[10])
56      digP.append((calib[13]<< 8) | calib[12])
57      digP.append((calib[15]<< 8) | calib[14])
58      digP.append((calib[17]<< 8) | calib[16])
59      digP.append((calib[19]<< 8) | calib[18])
60      digP.append((calib[21]<< 8) | calib[20])
61      digP.append((calib[23]<< 8) | calib[22])
62      digH.append( calib[24] )
63      digH.append((calib[26]<< 8) | calib[25])
64      digH.append( calib[27] )
65      digH.append((calib[28]<< 4) | (0x0F & calib[29]))
66      digH.append((calib[30]<< 4) | ((calib[29] >> 4) & 0x0F))
67      digH.append( calib[31] )
68
69      for i in range(1,2):
70          if digT[i] & 0x8000:
71              digT[i] = (-digT[i] ^ 0xFFFF) + 1
72
73      for i in range(1,8):
74          if digP[i] & 0x8000:
75              digP[i] = (-digP[i] ^ 0xFFFF) + 1
76
```

Copyright © 2024 YITJC All Rights Reserved.

```
77      for i in range(0,6):
78          if digH[i] & 0x8000:
79              digH[i] = (-digH[i] ^ 0xFFFF) + 1
80
81  #温度・湿度・気圧データ格納レジスタRead
82  def readData():
83      data = []
84      for i in range (0xF7, 0xF7+8):
85          data.append(bus.read_byte_data(i2c_address,i))
86      pres_raw = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4)
87      temp_raw = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4)
88      hum_raw  = (data[6] << 8)  |  data[7]
89
90      compensate_T(temp_raw)
91      compensate_P(pres_raw)
92      compensate_H(hum_raw)
93
94  #温度データtemperature算出
95  def compensate_T(adc_T):
96      global t_fine
97      v1 = (adc_T / 16384.0 - digT[0] / 1024.0) * digT[1]
98      v2 = (adc_T / 131072.0 - digT[0] / 8192.0) * (adc_T / 131072.0 - digT[0] / 8192.0) * digT[2]
99      t_fine = v1 + v2
100     temperature = t_fine / 5120.0
101     print("temperature : %6.2f ℃" % (temperature))
102
103 #湿度データhum算出
104 def compensate_H(adc_H):
105     global t_fine
106     hum = t_fine - 76800.0
107     if hum != 0:
108         hum = (adc_H - (digH[3] * 64.0 + digH[4]/16384.0 * hum)) * (digH[1] / 65536.0 * (1.0 + digH[5] / 67108864.0 *
    hum * (1.0 + digH[2] / 67108864.0 * hum)))
109     else:
110         return 0
111     hum = hum * (1.0 - digH[0] * hum / 524288.0)
112     if hum > 100.0:
113         hum = 100.0
114     elif hum < 0.0:
```

Copyright © 2024 YITJC All Rights Reserved.

```
115             hum = 0.0
116         print("humidity       : %6.2f  %" % (hum))
117
118 #Raspi_pressure
119 def compensate_P(adc_P):
120         global    t_fine
121         pressure = 0.0
122         v1 = (t_fine / 2.0) - 64000.0
123         v2 = (((v1 / 4.0) * (v1 / 4.0)) / 2048) * digP[5]
124         v2 = v2 + ((v1 * digP[4]) * 2.0)
125         v2 = (v2 / 4.0) + (digP[3] * 65536.0)
126         v1 = (((digP[2] * (((v1 / 4.0) * (v1 / 4.0)) / 8192)) / 8)  + ((digP[1] * v1) / 2.0)) / 262144
127         v1 = ((32768 + v1) * digP[0]) / 32768
128
129         if v1 == 0:
130             return 0
131         pressure = ((1048576 - adc_P) - (v2 / 4096)) * 3125
132         if pressure < 0x80000000:
133             pressure = (pressure * 2.0) / v1
134         else:
135             pressure = (pressure / v1) * 2
136         v1 = (digP[8] * (((pressure / 8.0) * (pressure / 8.0)) / 8192.0)) / 4096
137         v2 = ((pressure / 4.0) * digP[7]) / 8192.0
138         pressure = pressure + ((v1 + v2 + digP[6]) / 16.0)
139
140         print("pressure       : %7.2f hPa" % (pressure/100))
141
142 init()
143 get_calib_param()
144
145 if __name__ == '__main__':
146         try:
147             readData()
148         except KeyboardInterrupt:
149             pass
```

Copyright © 2024 YITJC All Rights Reserved.

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: 手順⑤でインストールした smbus2 ライブラリをインポート
- 6 行目～9 行目: I2C Address 設定
- 11 行目～15 行目: データ格納変数定義
- 17 行目～34 行目: BME 内部レジスタ設定

  ここでは、BME280 内部レジスタへの設定値、I2C Address、データ格納変数などを定義しています。

  BME280 レジスタに関する詳細は、Bosch 社 BME280 のデータシートに詳細が書かれています。

  BME280 の Memory map を示します。

Register Name	Register Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset
hum_lsb	0xFE	hum_lsb[7:0]								0x00
hum_msb	0xFD	hum_msb[7:0]								0x80
temp_xlsb	0xFC	temp_xlsb[7:4]				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb[7:0]								0x00
temp_msb	0xFA	temp_msb[7:0]								0x80
press_xlsb	0xF9	press_xlsb[7:4]				0	0	0	0	0x00
press_lsb	0xF8	press_lsb[7:0]								0x00
press_msb	0xF7	press_msb[7:0]								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26~calib41	0xE1~0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00~calib25	0x88~0xA1	calibration data								individual

Registers	Reserved	Calibration Data	Control registers	Data registers	Status registers	Chip ID	Reset
Type	do not change	read only	read/write	read only	read only	read only	write only

Control registers と呼ばれるレジスタに書き込むための設定値を設定しています。

- 19 行目～26 行目: レジスタへの書き込み値を設定
- 27 行目～30 行目: Control registers へ書き込む値を設定
- 31 行目～34 行目: レジスタアドレス、レジスタ書き込み値の設定
- 36 行目～38 行目: レジスタへのデータ書き込み

  Control registers へデータを書き込みます

Copyright © 2024 YITJC All Rights Reserved.

- 40 行目～79 行目: キャリブレーション（補正データ取得）

  キャリブレーションを実施しています。Register Address 0x88～0xE6

Register Address	Register content	Data type
0x88/0x89	dig_T1[7:0]/[15:8]	unsigned short
0x8A/0x8B	dig_T2[7:0]/[15:8]	signed short
0x8C/0x8D	dig_T3[7:0]/[15:8]	signed short
0x8E/0x8F	dig_P1[7:0]/[15:8]	unsigned short
0x90/0x91	dig_P2[7:0]/[15:8]	signed short
0x92/0x93	dig_P3[7:0]/[15:8]	signed short
0x94/0x95	dig_P4[7:0]/[15:8]	signed short
0x96/0x97	dig_P5[7:0]/[15:8]	signed short
0x98/0x99	dig_P6[7:0]/[15:8]	signed short
0x9A/0x9B	dig_P7[7:0]/[15:8]	signed short
0x9C/0x9D	dig_P8[7:0]/[15:8]	signed short
0x9E/0x9F	dig_P9[7:0]/[15:8]	signed short
0xA1	dig_H1[7:0]	unsigned short
0xE1/0xE2	dig_H2[7:0]/[15:8]	signed short
0xE3	dig_H3[7:0]	unsigned short
0xE4/0xE5[3:0]	dig_H4[11:4]/[3:0]	signed short
0xE5[7:4]/0xE6	dig_H5[3:0]/[11:4]	signed short
0xE7	dig_H6	signed char

- 81 行目～92 行目: 温度・湿度・気圧データが格納されているレジスタ Read

  Register Address 0xF7 から 8 バイト分のデータを読み込んでいます。
- 94 行目～101 行目: 温度データ算出
- 103 行目～116 行目: 湿度データ算出
- 118 行目～140 行目: 気圧データ算出

  Data registers から読み込んだデータをもとに、温度、湿度、気圧データを計算により求めています。

  算出方法（プログラム）は、データシートに記載されています。
- 142 行目: init()関数実行
- 143 行目: get_calib_param()実行
- 145 行目～149 行目:

  このブロックで行っている処理は、readData()関数の実行です。

  if __name__ == '__main__':

  に記述されている処理は、このプログラムを直接実行した場合のみ実行されます。

  このプログラムを他のプログラムから読み込んで（import）して実行する場合は、if __name__ == '__main__':

  に記述されている処理は行われません。

  他のプログラムから読み込む（import する）方法については、次項「3.4 クラス（Class）」で説明します。

Copyright © 2024 YITJC All Rights Reserved.

3.4 クラス（Class）

クラスについて説明する前に、「オブジェクト指向プログラミング（Object Oriented Programming: OOP）」について説明します。

オブジェクト指向プログラミングは、プログラムを手順ではなく、オブジェクト（モノ）の作成として考えます。

例えば、「スマートフォン」というモノを操作するとき、中でどのようなプログラムが動いているのか知る必要がありません。画面をタッチすれば、「スマートフォン」というモノは動作してくれます。つまり、スマートフォンというモノの中身を知らなくても、モノの使い方さえ理解していれば、メール送信やインターネット閲覧、動画の再生など、さまざまな機能を使用できるという考え方です。

オブジェクト指向プログラミングには「クラス」という考え方があります。クラスとは、これから作成する「モノの定義」です。別の言い方をすると、これから作成するモノの「設計図」と呼ぶこともできます。

今回紹介する「BM280_Class.py」プログラムを例に挙げて説明すると、クラスとして 6 行目～145 行目にモノを定義しています。ここ定義されているモノは、BME280 センサから温度・湿度・気圧のデータを取得する機能をもっています。オブジェクト指向プログラミングの考えでは、このプログラム（の中身）を知らなくても、温度・湿度・気圧データを取得することができます。つまり、プログラムの中身を知らなくても、クラス（Class BME280）の使い方さえ理解していれば、温度・湿度・気圧のデータを取得できます。

ここではオブジェクト指向プログラミングにおけるクラスの考え方について理解するために、BME280 クラスを定義したプログラムを紹介します。

[対象プログラム]

BME280_Class.py

[プログラム内容]

プログラムを実行すると、画面上に温度、湿度、気圧データが表示されます。

手順①～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x BME280_Class.py
```

手順②～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./BME280_Class.py
```

温度、湿度、気圧データが表示されることを確認してください。

```
(ycuu) cuu@raspberrypi:~/dev $ ./BME280_Class.py
temperature :  27.35 ℃
humidity    :  38.07 %
pressure    :  956.66 hPa
```

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding:utf-8
3	
4	from smbus2 import SMBus
5	
6	#BME280データ取得クラス定義
7	class BME280:
8	#コンストラクタ定義
9	#config, ctrl_meas, ctrl_muレジスタ設定
10	def __init__(self, busNumber=1, i2cAddress=0x76):
11	#I2C Address設定
12	#データ格納変数定義
13	self.bus = SMBus(busNumber)
14	self.i2cAddress = i2cAddress
15	self.digT = []
16	self.digP = []
17	self.digH = []
18	self.timeFine = 0.0
19	self.presRaw = 0.0
20	self.tempRaw = 0.0
21	self.humRaw = 0.0
22	#レジスタ書き込み値
23	osrsT = 1 #Temperature oversampling x 1
24	osrsP = 1 #Pressure oversampling x 1
25	osrsH = 1 #Humidity oversampling x 1
26	mode = 3 #Normal mode
27	tSb = 5 #Tstandby 1000ms
28	filter = 0 #Filter off
29	spi3wEn = 0 #3-wire SPI Disable
30	#レジスタ設定
31	ctrlMeasReg = (osrsT << 5) | (osrsP << 2) | mode
32	configReg = (tSb << 5) | (filter << 2) | spi3wEn
33	ctrlHumReg = osrsH
34	#レジスタアドレス指定
35	self.writeReg(0xF2,ctrlHumReg)
36	self.writeReg(0xF4,ctrlMeasReg)
37	self.writeReg(0xF5,configReg)

Copyright © 2024 YITJC All Rights Reserved.

```
38              self.getCalibParam()
39              #温度・湿度・気圧データ格納レジスタRead
40              self.readData()
41
42      #Data(レジスタ設定値)書き込み
43      def writeReg(self, regAddress, data):
44              self.bus.write_byte_data(self.i2cAddress, regAddress, data)
45
46      #Calibration (補正データ取得)
47      def getCalibParam(self):
48              calib = []
49
50              for i in range (0x88,0x88+24):
51                      calib.append(self.bus.read_byte_data(self.i2cAddress,i))
52              calib.append(self.bus.read_byte_data(self.i2cAddress,0xA1))
53              for i in range (0xE1,0xE1+7):
54                      calib.append(self.bus.read_byte_data(self.i2cAddress,i))
55
56              self.digT.append((calib[1] << 8) | calib[0])
57              self.digT.append((calib[3] << 8) | calib[2])
58              self.digT.append((calib[5] << 8) | calib[4])
59              self.digP.append((calib[7] << 8) | calib[6])
60              self.digP.append((calib[9] << 8) | calib[8])
61              self.digP.append((calib[11]<< 8) | calib[10])
62              self.digP.append((calib[13]<< 8) | calib[12])
63              self.digP.append((calib[15]<< 8) | calib[14])
64              self.digP.append((calib[17]<< 8) | calib[16])
65              self.digP.append((calib[19]<< 8) | calib[18])
66              self.digP.append((calib[21]<< 8) | calib[20])
67              self.digP.append((calib[23]<< 8) | calib[22])
68              self.digH.append( calib[24] )
69              self.digH.append((calib[26]<< 8) | calib[25])
70              self.digH.append( calib[27] )
71              self.digH.append((calib[28]<< 4) | (0x0F & calib[29]))
72              self.digH.append((calib[30]<< 4) | ((calib[29] >> 4) & 0x0F))
73              self.digH.append( calib[31] )
74
75              for i in range(1,2):
76                      if self.digT[i] & 0x8000:
```

Copyright © 2024 YITJC All Rights Reserved.

```
77                      self.digT[i] = (-self.digT[i] ^ 0xFFFF) + 1
78
79              for i in range(1,8):
80                  if self.digP[i] & 0x8000:
81                      self.digP[i] = (-self.digP[i] ^ 0xFFFF) + 1
82
83              for i in range(0,6):
84                  if self.digH[i] & 0x8000:
85                      self.digH[i] = (-self.digH[i] ^ 0xFFFF) + 1
86
87      #温度・湿度・気圧データ格納レジスタRead
88      def readData(self):
89          data = []
90          for i in range (0xF7, 0xF7+8):
91              data.append(self.bus.read_byte_data(self.i2cAddress,i))
92          self.presRaw = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4)
93          self.tempRaw = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4)
94          self.humRaw    = (data[6] << 8)   |   data[7]
95
96      #気圧データpressure算出
97      def getPressure(self):
98          #変数pressure global定義
99          global pressure
100         pressure = 0.0
101
102         v1 = (self.timeFine / 2.0) - 64000.0
103         v2 = (((v1 / 4.0) * (v1 / 4.0)) / 2048) * self.digP[5]
104         v2 = v2 + ((v1 * self.digP[4]) * 2.0)
105         v2 = (v2 / 4.0) + (self.digP[3] * 65536.0)
106         v1 = (((self.digP[2] * (((v1 / 4.0) * (v1 / 4.0)) / 8192)) / 8)   + ((self.digP[1] * v1) / 2.0)) / 262144
107         v1 = ((32768 + v1) * self.digP[0]) / 32768
108
109         if v1 == 0:
110             return 0
111         pressure = ((1048576 - self.presRaw) - (v2 / 4096)) * 3125
112         if pressure < 0x80000000:
113             pressure = (pressure * 2.0) / v1
114         else:
115             pressure = (pressure / v1) * 2
```

Copyright © 2024 YITJC All Rights Reserved.

```
116             v1 = (self.digP[8] * (((pressure / 8.0) * (pressure / 8.0)) / 8192.0)) / 4096
117             v2 = ((pressure / 4.0) * self.digP[7]) / 8192.0
118             pressure = pressure + ((v1 + v2 + self.digP[6]) / 16.0)
119             return pressure/100
120
121         #温度データtemperature算出
122         def getTemperature(self):
123             #変数temperature global定義
124             global temperature
125             v1 = (self.tempRaw / 16384.0 - self.digT[0] / 1024.0) * self.digT[1]
126             v2 = (self.tempRaw / 131072.0 - self.digT[0] / 8192.0) * (self.tempRaw / 131072.0 - self.digT[0] / 8192.0) *
    self.digT[2]
127             self.timeFine = v1 + v2
128             temperature = self.timeFine / 5120.0
129             return temperature
130
131         #湿度データvarH算出
132         def getHumidity(self):
133             #変数varH global定義
134             global varH
135             varH = self.timeFine - 76800.0
136             if varH != 0:
137                 varH = (self.humRaw - (self.digH[3] * 64.0 + self.digH[4]/16384.0 * varH)) * (self.digH[1] / 65536.0 * (1.0 +
    self.digH[5] / 67108864.0 * varH * (1.0 + self.digH[2] / 67108864.0 * varH)))
138             else:
139                 return 0
140             varH = varH * (1.0 - self.digH[0] * varH / 524288.0)
141             if varH > 100.0:
142                 varH = 100.0
143             elif varH < 0.0:
144                 varH = 0.0
145             return varH
146
147 if __name__ == '__main__':
148         sensor = BME280()
149         try:
150             #温度・湿度・気圧データ表示
151             print("temperature : %6.2f ℃" % sensor.getTemperature())
152             print("humidity      : %6.2f %" % sensor.getHumidity())
```

Copyright © 2024 YITJC All Rights Reserved.

```
153             print("pressure      : %7.2f hPa" % sensor.getPressure())
154         except KeyboardInterrupt:
155             pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: 手順⑤でインストールした smbus2 ライブラリをインポート
- 6 行目～145 行目: BME280 温度・湿度・気圧データ取得クラス定義
  ここでは、BME280 という名前のクラスを定義し、モノが行う仕事内容を記述しています。
- 8 行目～40 行目: コンストラクタ定義
  コンストラクタは、次のように記述し、クラスのインスタンスが生成されるときに自動的に呼び出されます。
  def __init__(self, 引数 2, 引数 3, ...) :
  コンストラクタの名前は「__init__」に決まっています。名前の前後に「__」が付いているメソッドは「特殊メソッド」と呼ばれます。コンストラクタは特殊メソッドの 1 つです。
  ここでは、BME280 内部レジスタへの設定値、I2C Address、データ格納変数などを定義しています。
  BME280 レジスタに関する詳細は、Bosch 社 BME280 のデータシートに詳細が書かれています。
- 22 行目～29 行目: レジスタへの書き込み値を設定
- 30 行目～33 行目: Control registers へ書き込む値を設定
- 34 行目～38 行目: レジスタアドレス、レジスタ書き込み値の設定
- 42 行目～44 行目: レジスタへのデータ書き込み
  Control registers へデータを書き込みます
- 46 行目～85 行目: キャリブレーション（補正データ取得）
  キャリブレーションを実施しています。Register Address 0x88～0xE6
- 87 行目～94 行目: 温度・湿度・気圧データが格納されているレジスタ Read
  Register Address 0xF7 から 8 バイト分のデータを読み込んでいます。
- 96 行目～119 行目: 気圧データ算出
- 121 行目～129 行目: 温度データ算出
- 131 行目～145 行目: 湿度データ算出
  Data registers から読み込んだデータをもとに、気圧、温度、湿度データを計算により求めています。
  算出方法（プログラム）は、データシートに記載されています。
- 147 行目～155 行目:
  if __name__ == '__main__':
  このブロックで行っている処理は、
  148 行目: 変数 Sensor に BME280 クラスのインスタンスを代入しています。
  クラスが作成するモノの「設計図」と説明しましたが、インスタンス（instance）は、設計図によって得られる「実体」と呼ぶことができます。ここでは、BME280 センサから得られた「温度・湿度・気圧データ」がインスタンスに該当します。
  続いて、if ブロックの中で、BME280 クラスを使用して取得した温度・湿度・気圧のデータを print 関数で画面に表示しています。

Copyright © 2024 YITJC All Rights Reserved.

手順③～モジュールのインポート（import）～

Python のコード（プログラム）はスクリプトファイルとして保存し、他のプログラムから利用できます。このプログラムファイルを「**モジュール**」と呼びます。これまでに学習したプログラムにおいても Python で標準ライブラリや外部ライブラリを読み込む際に「import」を使用しましたが、自作したモジュールをインポート（import）して読み込むことで、そのモジュールで定義した関数やクラスを他のプログラムから利用できます。

ここでは、手順②で作成した Python プログラム（モジュール）で定義したクラスを読み込んで利用する方法を説明します。

対象プログラムを実行します。

[対象プログラム]

9_BME280.py

[プログラム内容]

プログラムを実行すると、画面上に温度、湿度、気圧データが表示されます。

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加してから実行します。

```
(ycuu) cuu@raspberrypi:$ cd dev
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 9_BME280.py
(ycuu) cuu@raspberrypi:~/dev $ ./9_BME280.py
```

温度、湿度、気圧データが表示されることを確認してください。

BME280_Class.py で定義したクラス「BME280」を読み込んで（import して）使用する方法について、プログラムの解説で説明します。

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	# 自作ライブライのimport
5	import BME280_Class
6	
7	if _name_ == '_main_':
8	try:
9	bme = BME280_Class.BME280()
10	bme.readData()

Copyright © 2024 YITJC All Rights Reserved.

```
11            #温度・湿度・気圧データ表示
12            print("temperature : %6.2f ℃" % bme.getTemperature())
13            print("humidity      : %6.2f  %" % bme.getHumidity())
14            print("pressure      : %7.2f hPa" % bme.getPressure())
15        except KeyboardInterrupt:
16            pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 5 行目: BME280_Class モジュールをインポート（import）

  注意

  モジュール名（ここでは、BME280_Class）には、次のルールがあるので注意してください。

  ① 数字から始まるモジュール名は使用禁止

  ② Python の予約語（def, class, import など）は使用禁止

  ③ 特殊文字（#, $, @, !など）は使用禁止

  ④ アンダースコア「_」とハイフン「-」の混在は禁止

  ⑤ 大文字と小文字を区別する

  また、インポートするモジュールは同じディレクトリ（ここでは、/home/cuu/dev）に保存してください。

  別のディレクトリからモジュールをインポートする場合は、次の記述に倣ってください。

```
import sys
sys.path.append('/your/module/path')


import your_module
```

- 7 行目～16 行目: main プログラム
- 8 行目～14 行目 :try ブロック

  モジュール（BME280_Class）で定義されているクラス（BME280）を読み込んで、温度・湿度・気圧のデータを表示する。
- 9 行目: クラス生成

  5 行目でインポート（import）したモジュールからクラスを生成し、変数「bme」に代入しています。

  クラスを生成する場合は、「**モジュール名.クラス名**」としてください。
- 10 行目: クラスで定義されている readData()関数の実行
- 12 行目: クラスで定義されている getTemperature()関数の実行
- 13 行目: クラスで定義されている getHumidity()関数の実行
- 14 行目: クラスで定義されている getPressure()関数の実行
- 15 行目～16 行目 :except ブロック

  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

参考～if _name_ == '_main_':～

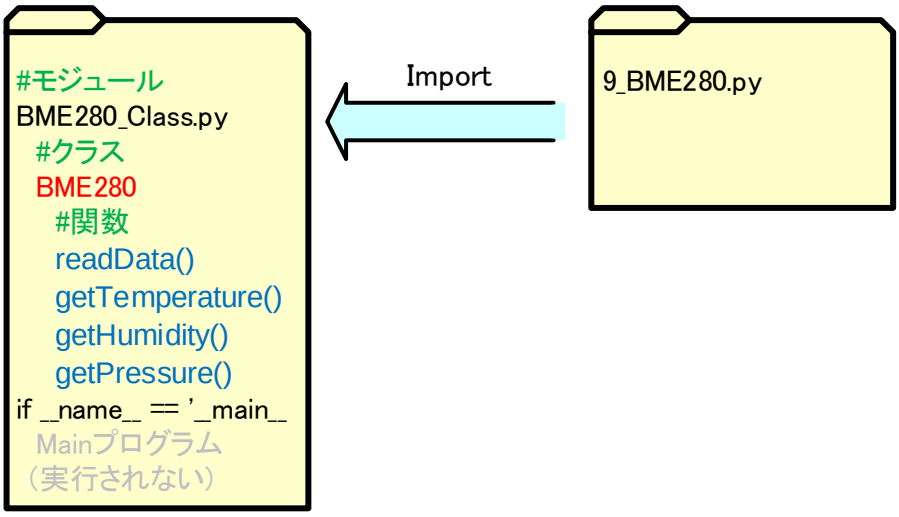
if __name__ == '__main__':に記述されている処理は、このプログラムを直接実行した場合のみ実行されます。

このプログラムを他のプログラムから読み込んで（import）して実行する場合は、if __name__ == '__main__':に記述されている処理は行われません。

「9_BME280.py」でモジュール（BME280_Class.py）を読み込んだケースを例に挙げて説明します。

この場合、「if __name__ == '__main__':」の変数「__name__」には、「9_BME280」が代入されるため、if 文が「False」となり、if 文の中にある Main プログラムは実行されません。

つまり、if __name__ == '__main__':ブロック内に記述された関数は、他のスクリプト（9_BME280.py）から実行することはできません。



参考～プログラムとスクリプトの違い～

本講座は、（第 2 章で少し触れましたが）スクリプトという表現は避けてきました。初めて学ぶ方にもわかりやすいように、Python「プログラム」と表現しています。ここで、プログラムとスクリプトの違いについて説明します。

1. プログラムとスクリプトの区別

   プログラムとスクリプトは、基本的に同じようなものを指します。両者は実行可能なコードを含むファイルであり、コンピュータが処理できる形式に変換されます。「プログラム」は一般的にバイナリ形式の実行可能ファイルを指し、「スクリプト」はテキスト形式の実行可能ファイルを指します。Python は、テキスト形式のコードを直接実行することができるため、スクリプトと呼ばれることが多いです。

2. コンパイラとインタプリタの違い

   コンパイラは、ソースコード全体をバイナリ形式に変換し、実行可能ファイルを生成します。一方、インタプリタは、ソースコードを逐次的に読み込み（翻訳して）、それを実行します。したがって、コンパイラによって生成された実行可能ファイルは、ソースコードとは異なる形式で保存される場合がありますが、インタプリタによって実行されるスクリプトは通常、ソースコードそのものの形式で保存されます。

3. スクリプト言語とコンパイル言語の区別

   プログラムがスクリプト言語またはコンパイル言語であるかは、言語の特性によって決まります。スクリプト言語は、「インタプリタ」と呼ばれるものがスクリプト言語を翻訳して実行します。スクリプト言語としては、Python や Ruby などが代表的です。一方、C、C++、Java などは通常、コンパイラによってバイナリ形式に変換されるため、コンパイル言語と呼ばれます。

Copyright © 2024 YITJC All Rights Reserved.

3.5 SPI～アナログセンサ LM35～

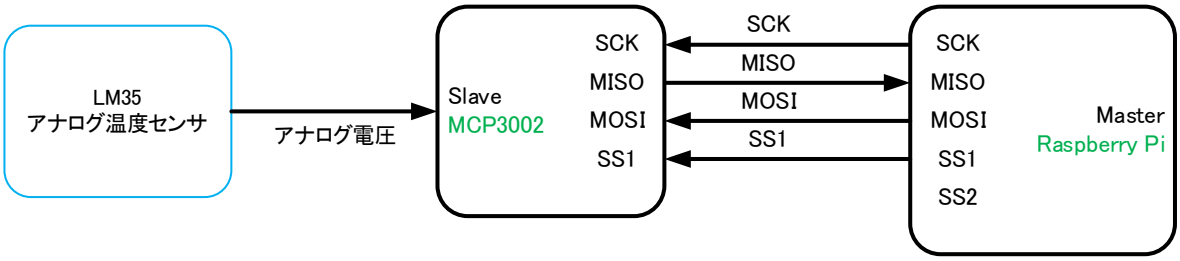
Raspberry Pi は残念ながらアナログ入力端子がないため、アナログセンサ（アナログ電圧出力センサ）を直接接続してデータ（アナログ値）を取得できません。そのため、アナログセンサの値を Raspberry Pi が扱えるデジタル値に変換する AD コンバータ（ADC: Analog Digital Converter）が必要です。

HAT Board は Microchip 社製 10bit 2ch A/D コンバータ「MCP3002」を搭載し、アナログセンサを最大 2 個接続してデータを取得できます。

Raspberry Pi は A/D コンバータ「MCP3002」と SPI 通信を行い、デジタル変換されたアナログセンサのデータを取得します。ここでは、「MCP3002」を使用した A/D 変換プログラムについて学習します。

A/D コンバータ～MCP3002～

アナログセンサ（LM35）から出力されるアナログ信号は、A/D コンバータ「MCP3002」によって 10bit のデジタル値に変換されます。デジタル変換されたアナログ温度センサのデータは、SPI インターフェースを使用して Raspberry Pi に送られます。

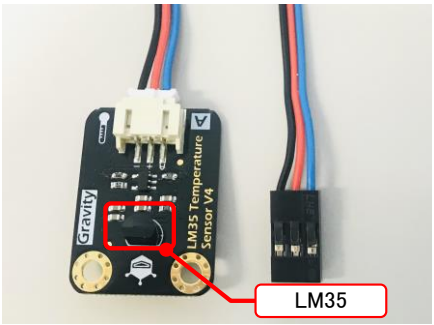


仕様

- 2ch 入力（アナログ入力 2ch）
- 解像度 10bit
- 電源電圧: 2.7V～5.5V
  ※HAT Board は、3.3V 電源を使用しています。
- SPI インターフェース

アナログセンサ～LM35～

アナログセンサは、T.I 社製高精度摂氏直読温度センサ LM35 を搭載した温度センサモジュールを使用します。



Copyright © 2024 YITJC All Rights Reserved.

仕様

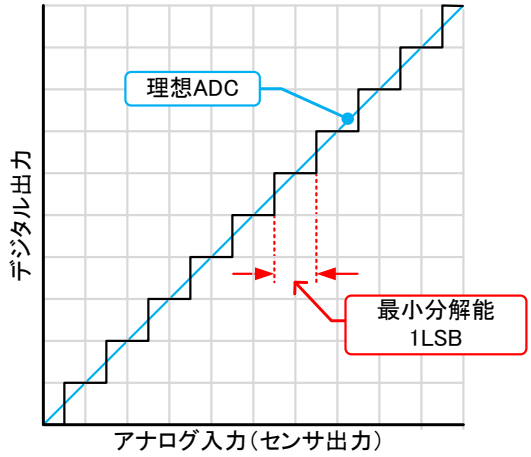
- 測定温度範囲: 0℃～100℃
- 精度: ±1℃
- 電源電圧: 3.3V～5V
  ※HAT Board は、3.3V 電源で使用しています。
- 温度係数: 40mV/℃ (※1)
  ※1: 10mV/℃（購入時）の設定が 40mV/℃になるように回路定数を変更しています。

参考～A/D 変換における最小分解能～

Raspberry Pi はアナログ入力端子がないため、アナログセンサの信号を直接受信できません。しかし、外付け AD コンバータ（ADC）を使用することでデジタル変換されたアナログセンサのデータを受信できます。

ここでは、A/D 変換の基礎知識として「最小分解能」について説明します。ADC は、連続するアナログ信号を等時間間隔でサンプリング（離散データ化）し、デジタル信号に変換します。A/D 変換されたデジタル信号のビット数を「**分解能**」と呼びます。最上位ビットを MSB（Most Significant Bit）、最下位ビットを LSB（Least Significant Bit）と呼びます。

次のグラフは、アナログ入力（センサ出力）とデジタル出力の関係を示したものです。デジタル出力値を 1LSB 変化させるアナログ信号値を「**最小分解能**」と呼びます。



MC3002 は分解能が「10bit」の ADC であるため、アナログ入力を 1024（2 の 10 乗）に分解します。電源電圧が 3.3V であれば、最小分解能は約 3.2mV になります。つまり、MCP3002 は、約 3.2mV の（アナログ）電圧変化を検出できます。MCP3002 より解像度が高い ADC を使用すれば、さらに小さな電圧（変化）を検出してデジタル値に変換できます。

Copyright © 2024 YITJC All Rights Reserved.

注意～測定分解能～

アナログセンサを扱うときに注意すべき点は、「**測定分解能**」です。この「測定分解能」は、アナログセンサから出力される電圧レベルに依存します。

～その① アナログセンサの出力電圧が小さい場合～

今回使用する温度センサモジュールの設定(購入時)は、次のとおりです。

- 温度係数: 10mV/℃(1℃あたり 10mV の電圧変化)

MCP3002(電源電圧 3.3V)の最小分解能は、約 3.2mV なので、
1℃あたりの分解能は、10mV÷3.2mV＝約 3.125LSB です。
したがって、デジタル最小値 1LSB あたりの温度は約 0.32℃になります。
➔ つまり、購入時の設定における「測定分解能」は、0.32℃と言えます。

～その② アナログセンサの出力電圧が大きい場合～

続いて、温度センサモジュールの回路定数を次のとおり変更しました。

- 温度係数: 40mV/℃(1℃あたり 40mV の電圧変化)

MCP3002(電源電圧 3.3V)の最小分解能は、約 3.2mV ですので、
1℃あたりの分解能は、40mV÷3.2mV＝約 12.5LSB です。
したがって、デジタル最小値 1LSB あたりの温度は約 0.08℃になります。
➔ つまり、回路定数を変更した後の「測定分解能」は、0.08℃と言えます。
なお、温度係数 40mV/℃時の(測定可能)温度範囲は、電源電圧が 3.3V のため、
3.3V÷40mV ＝ 82.5℃になります。

同じ温度センサを使用しても、アナログセンサから出力される電圧値によって「測定分解能」が大きく異なることがわかります。アナログセンサから出力される電圧レベルが小さい場合は、アナログセンサの信号を直接 ADC に送信するのではなく、増幅回路により電圧値を増幅してから ADC に送信する必要があります。

最後に、測定分解能を上げる方法を 2 つ紹介します。

① 高解像度の ADC を使用する
高解像度の ADC を使用することで、測定分解能を向上させることができます。例えば、HAT Board に実装されている A/D コンバータ「MCP3002」の解像度は 10bit ですが、より高解像度の A/D コンバータを使用することで、測定分解能が向上してより微細な変化を捉えることができます。

② アナログセンサの出力電圧値を増幅する
アナログセンサの出力電圧値を増幅回路などで増幅することで、測定分解能を上げることができます。ただし、測定上限値にも影響が及ぶため、測定対象の「測定範囲」と「測定分解能」のバランスを考慮して、増幅回路の仕様を決定する必要があります。

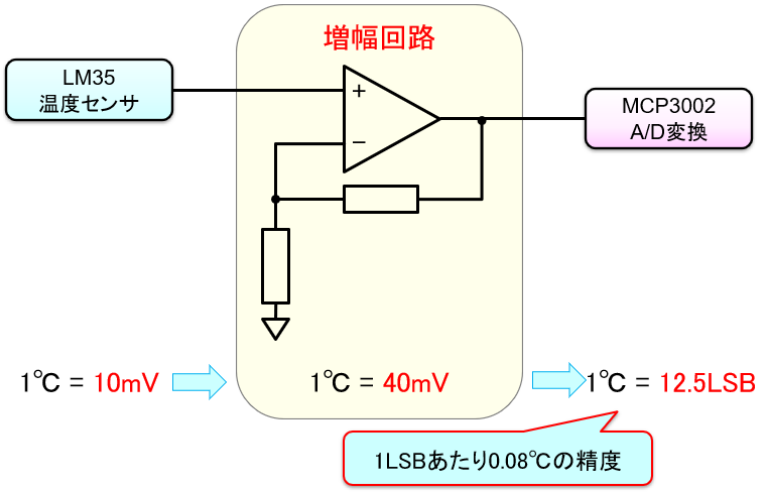
Copyright © 2024 YITJC All Rights Reserved.

参考～アナログ温度センサモジュールの増幅回路～

本講座で使用するアナログ温度センサモジュールは、アナログ温度センサ（LM35）と非反転増幅回路（オペアンプ LM321、抵抗×2 個）が基板上に実装されています。増幅回路を構成する抵抗の値を変更することで、アナログ温度センサから出力されるアナログ電圧を増幅できます。

購入時、増幅回路は 1 倍に設定されているため、「1℃あたりの電圧変化は 10mV」です。そのため、「測定分解能」は、0.32℃です。

本講座は測定分解能を向上させるため、増幅回路を 4 倍に変更して「1℃あたりの電圧変化を 40mV」に設定しました。この結果、「測定分解能」は、0.08℃となっています。



それでは、SPI インターフェースを使用してアナログ温度センサモジュールの値を取得する手順について説明します。

手順①～シャットダウン・電源ケーブル外す～

アナログ温度センサモジュールを接続するため、Raspberry Pi をシャットダウンしてから電源ケーブルを外してください。シャットダウンは、第 2 章で実装したシャットダウンスイッチ（HAT Board の黒スイッチ）を押してください。

注意～活線挿抜について～

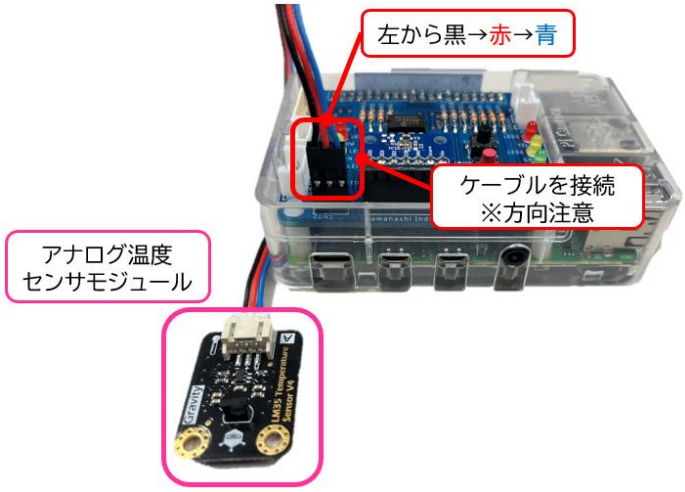
電源を入れた状態でケーブルを抜き挿しすることを「活線挿抜」と言います。活線挿抜は、電子部品（センサや IC など）の故障につながる危険性があります。電子部品に電源が供給された状態で部品を挿抜すると、急激な電流変化やスパイク電圧が発生する可能性があります。これらは電子部品に対して「過電流」や「過電圧」を引き起こし、電子部品内の微細な回路や構成要素を破壊する恐れがあります。特に IC（集積回路）などの高度な電子部品は、微小な電気的変化でも影響を受けやすく、部品の故障や損傷につながる可能性が高まります。そのため、電子部品の取り扱いや交換を行う際には、必ず電源を切って（基板への電源供給を遮断して）から作業を行うことが重要です。

Copyright © 2024 YITJC All Rights Reserved.

手順②～温度センサモジュールの取り付け～

HAT Board にアナログ温度センサモジュールを接続します。ケーブルの接続に注意してください。

アナログ温度センサモジュールを接続したあと、USB Type-C ケーブルを挿して Raspberry Pi に電源を供給してください。



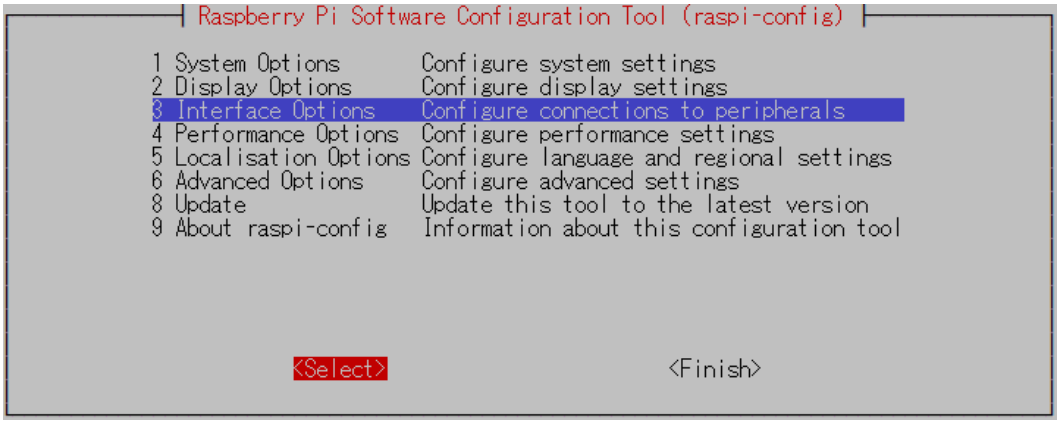
手順③～SPI 有効設定～

raspi-config コマンドを実行し、Raspberry Pi の SPI 通信インターフェースを有効にします。

(ycuu) cuu@raspberrypi:~ $ sudo raspi-config

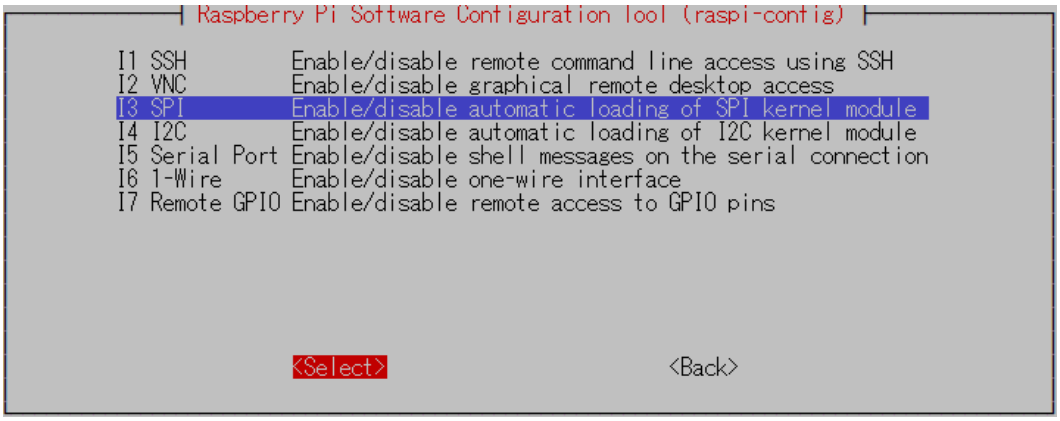
「3.Interface Options」を選択します。

Raspi-config 画面においては、「メニュー選択」は「→キー」、「カーソル移動」は「tab キー」、「決定」は「Enter キー」で操作します。

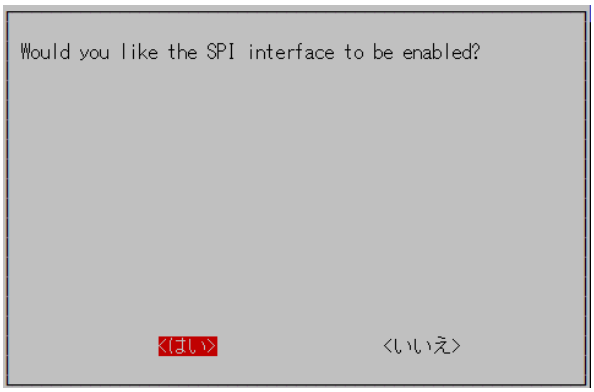


Copyright © 2024 YITJC All Rights Reserved.

「I3 SPI」を選択してください。

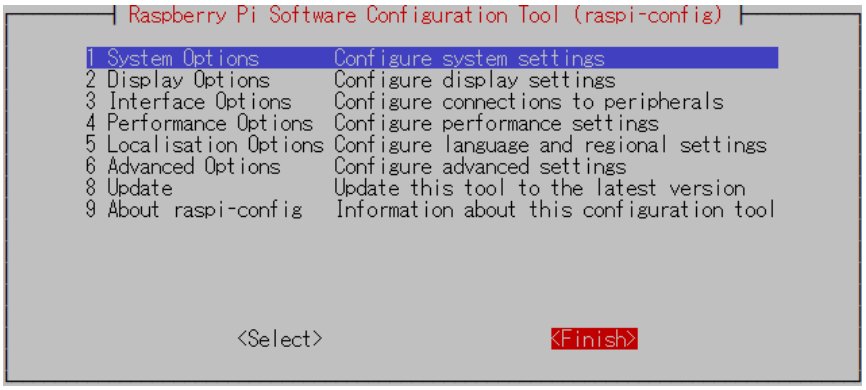


「はい」を選択し、「Enter」を押してください。



「了解」を選択し、「Enter」を押してください。

最初の画面に戻り、「Finish」を選択（Enter）して Raspi-config 画面を終了してください。



以上で、Raspberry Pi の I2C 通信インターフェースが有効（使用可能な状態）になりました。

Copyright © 2024 YITJC All Rights Reserved.

手順④～SPI 設定確認～

次のコマンドを実行し、カーネル（Kernel）に組み込まれている SPI 関連のモジュールを確認します。

(ycuu) cuu@raspberrypi:~ $ lsmod | grep spi

```
spidev                 20480  0
spi_bcm2835            20480  0
```

「spidev」は、SPI デバイス専用のカーネルモジュールです。

「spi bcm2835」は、Raspberry Pi の GPIO ピンを使用して SPI 通信を行うためのモジュールです。

補足～カーネル（Kernel）～

カーネル（Kernel）は、オペレーティングシステム（OS）の中核を担う重要なソフトウェアです。システムのリソースやソフトウェアとハードウェアの連携を管理します。Raspberry Pi OS は、カーネルに組み込まれている I2C や SPI などの通信モジュールを使用して外部のデバイスと通信を行います。

補足～lsmod, grep コマンド～

「lsmod」は、カーネルに組み込まれているモジュールの一覧を表示します。

「|（パイプ）」は、コマンドとコマンドを橋渡し（パイプライン）する役割があります。

ここでは、lsmod コマンドの実行結果を「|」の後の「grep」コマンドに渡しています。

「grep」は、テキストを検索する際に使用するコマンドです。grep コマンドの後にテキストを指定して検索します。

ここでは、カーネルに組み込まれているモジュールの一覧から、「spi」に関連するモジュールを表示しています。

続いて、SPI が有効になっていることを確認します。

(ycuu) cuu@raspberrypi:~ $ cat /boot/firmware/config.txt | grep dtparam

```
dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on
dtparam=audio=on
```

「dtparam」は、Device Tree Parameter の略称称で、Raspberry Pi のデバイスツリーパラメータを設定するために使用されます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～spidev インストール～

SPI 通信を制御するためのパッケージ spidev（SPI Device）をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install spidev

手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

10_SPI.py

[プログラム内容]

プログラムを実行すると、A/D 変換後の「デジタル値」、「電圧換算値」、「温度データ」が表示されます。

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 10_SPI.py

手順⑧～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./10_SPI.py

5 秒おきに A/D 変換後の「デジタル値」、「電圧換算値」、「温度データ」が表示されることを確認してください。

動作確認が終了したら、「Ctrl」+「c」を押してプログラムを終了してください。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import time
5	import sys
6	import spidev
7	
8	spi = spidev.SpiDev()
9	#Bus,CE選択
10	spi.open(0,0)
11	#通信速度設定　1MHz
12	spi.max_speed_hz = 1000000
13	
14	#A/D変換後のデジタル値(data)取得
15	def readAdc():
16	#MCP3002への送信信号(Din)
17	#spi.xfer2メソッドを使用して16進数を送信
18	#CH0から取得する場合　0b01101000=0x68、0b00000000=0x00
19	#CH1から取得する場合　0b01111000=0x78、0b00000000=0x00
20	adc = spi.xfer2([0x68,0x00])
21	
22	#MCP3002からの応答(Dout: 10bitデジタル値取得)
23	#data = ((adc[0] & 3) << 8) | adc[1]
24	data = (adc[0] * 256 + adc[1]) & 0x3ff
25	return data
26	
27	#デジタル値(data)を電圧値(volts)に換算
28	def convertVolts(data):
29	volts = (data * 3.3) / float(1024)
30	volts = round(volts, 6)
31	return volts
32	
33	#電圧値(volts)を温度データ(temp)に換算
34	def convertTemp(volts):
35	temp = (25 * volts)
36	temp = round(temp,4)
37	return temp

Copyright © 2024 YITJC All Rights Reserved.

```
38
39  if __name__ == '__main__':
40      try:
41          while True:
42              data = readAdc()
43              print("adc    : {:8} ".format(data))
44              volts = convertVolts(data)
45              temp = convertTemp(volts)
46              print("volts: {:8.6f}".format(volts))
47              print("temp : {:8.4f}".format(temp))
48              time.sleep(5)
49      except KeyboardInterrupt:
50          spi.close()
51          sys.exit(0)
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: time ライブラリインポート
- 5 行目: sys ライブラリインポート （51 行目「sys.exit(0)」で使用）
- 6 行目: SPI ライブラリインポート
- 8 行目～12 行目: SPI 設定

  8 行目:spidev ライブラリの SpiDev()関数を使用して SPI 通信を行うため、インスタンスを生成しています。

  9 行目: SPI バス、CE 選択

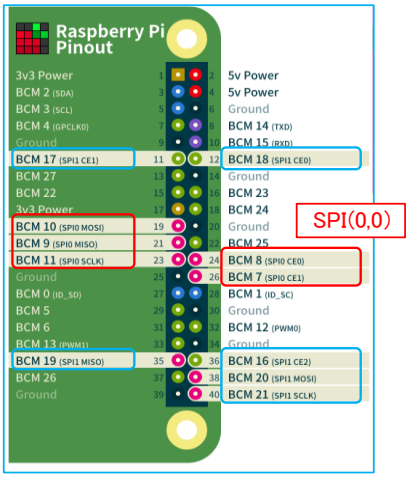
  次のコマンドを実行すると、使用可能な SPI デバイスを確認することができます。

  cuu@raspberrypi:~/dev $ ls -l /dev/spi*

```
(ycuu) cuu@raspberrypi: ~/dev $ ls -l /dev/spi*
crw-rw---- 1 root spi 153, 0  4月 30 09:11 /dev/spidev0.0
crw-rw---- 1 root spi 153, 1  4月 30 09:11 /dev/spidev0.1
```

Copyright © 2024 YITJC All Rights Reserved.

HAT Board に実装されている AD コンバータ（MCP3002）は、Raspberry Pi と SPI 通信するために SPI0, CE0 を使用しています。そのため、ここでは spi.open(0,0)と記述します。



- 12 行目: 通信速度を設定します

～補足～

2018 年頃から speed の指定が必須になっています。

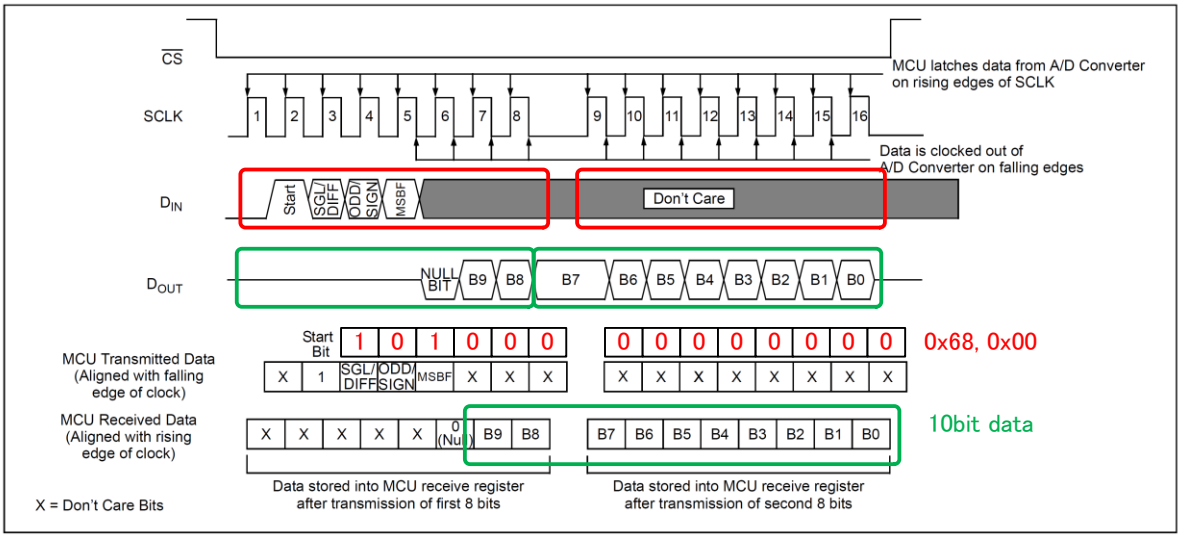
Speed を設定しない場合、data=0 が返ってくるので、注意してください。

- 14 行目～25 行目: A/D 変換後のデジタル値（data）取得
- 20 行目: spi.xfer2 メソッドを使用して A/D コンバータ（MCP3002）の Configurations bits を設定します

設定する内容は、MCP3002 のデータシートに従って設定します。

#CH0（アナログセンサ 1）から取得する場合　0b01101000=0x68、0b00000000=0x00

#CH1（アナログセンサ 2）から取得する場合　0b01111000=0x78、0b00000000=0x00

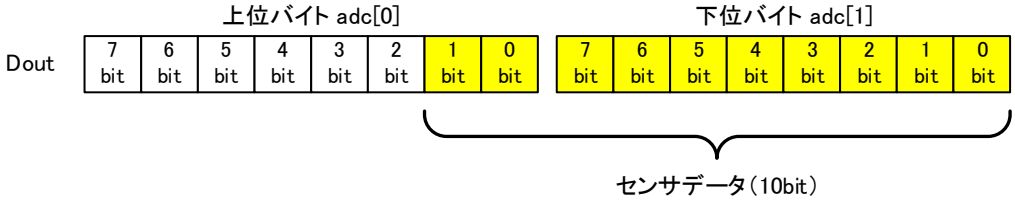


	CONFIG BITS		CHANNEL SELECTION		GND
	SGL/ DIFF	ODD/ SIGN	0	1	
SINGLE ENDED MODE	1	0	+		-
	1	1		+	-
PSEUDO-DIFFERENTIAL MODE	0	0	IN+	IN-	
	0	1	IN-	IN+	

ch0使用

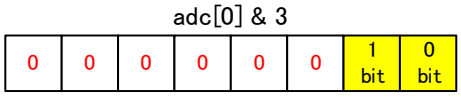
Copyright © 2024 YITJC All Rights Reserved.

・ 24 行目: A/D コンバータ（MCP3002）からの応答（A/D 変換後のデジタル値 10bit データ）を取得
コメントアウト（#）している 23 行目、24 行目いずれも、MCP3002 から受信した 16 ビット（2 バイト）のデータ Dout から 10 ビットデータ「data」を取り出しています。

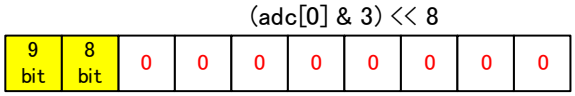


● data = ((adc[0] & 3) << 8) | adc[1]:

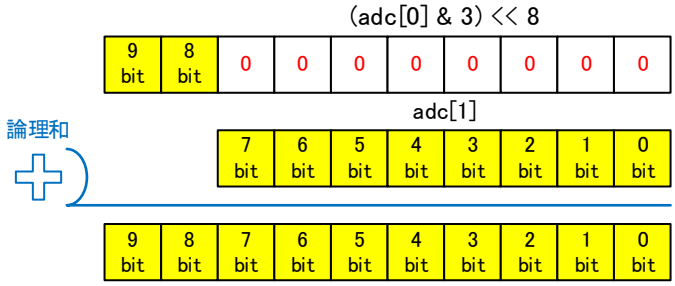
① adc[0] & 3:
上位バイトのデータ adc[0]と 3（0b00000011）の論理積（&）によって、adc[0]の下位 2 ビット以外のデータを 0 にしています。



② ((adc[0] & 3) << 8):
①のデータを 8 ビット左にシフトしています。これにより、10 ビットのデータを作成しています。



③ ((adc[0] & 3) << 8) | adc[1]:
②のデータと下位バイトのデータ adc[1]との論理和（|）によって、センサデータの 10bit を取り出しています。

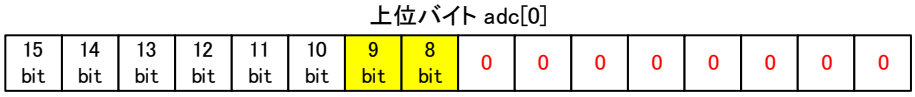


Copyright © 2024 YITJC All Rights Reserved.

- data = (adc[0] * 256 + adc[1]) & 0x3ff:

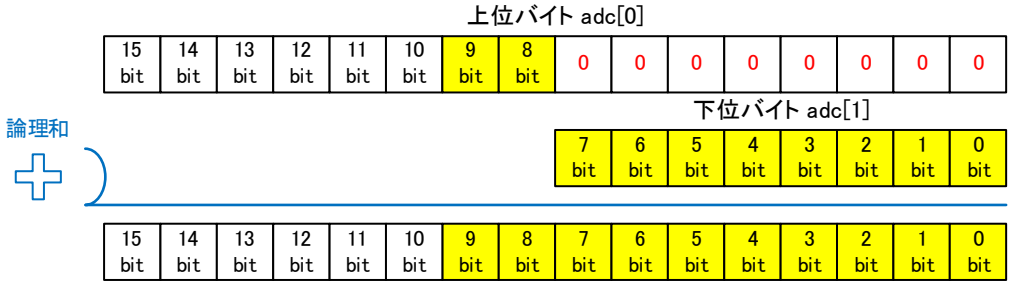
① adc[0] * 256：

上位バイトのデータ adc[0]を 256 倍して 16 ビットのデータに変換しています。



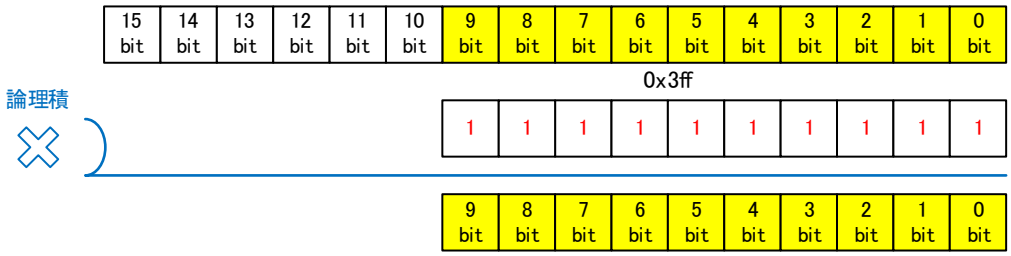
② adc[0] * 256 + adc[1]：

①のデータと下位バイトのデータ adc[1]との論理和（|）を取っています。



③ (adc[0] * 256 + adc[1]) & 0x3ff：

②のデータと 0x3ff（0b0011 1111 1111）の論理積（&）によって、センサデータの 10bit を取り出しています。



- 27 行目～31 行目：デジタル値（data）を電圧値（volts）に換算します
- 29 行目：電圧値への換算式

得られたデジタル値「data」を電圧値に換算するための式です。

A/D コンバータ（MCP3002）の基準電圧（VREF）3.3V で電圧値に換算するため、3.3V で掛けた値を 10bit（2 の 10 乗=1024）で除しています。

30 行目：算出した電圧値を小数 6 桁に丸めています

- 33 行目～37 行目：電圧値（volts）を温度データ（temp）に換算します
- 35 行目：温度データへの換算式

温度センサモジュールからの信号出力は、「1℃あたり 40mV の電圧変化」に設定してあります。

この設定から単位ボルト（1V）あたりの温度を計算し、計算した値（25）と電圧値「volts」を掛け算して温度データ（temp）に変換しています。

ここでは、単位ボルト（1V）あたりの温度は 25℃（1V÷40mV=25）と計算しています。

- 36 行目：算出した温度データを小数 4 桁に丸めています

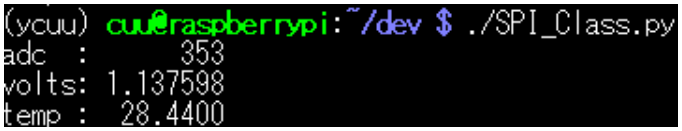
Copyright © 2024 YITJC All Rights Reserved.

- 39 行目～51 行目: デジタル値(data)、電圧値(volts)、温度データ(temp)の表示
- 41 行目～48 行目: print 関数を使用して 5 秒おきに画面表示
- 49 行目～51 行目: 例外処理
  「Ctrl」＋「c」をプログラム実行中に押すと、50 行目以降の処理(例外処理)が実行されます。
- 50 行目: spi クローズ
- 51 行目: プログラム終了 sys.exit(0)で正常終了 （参考: sys.exit(1)は異常終了とされています。）

参考～SPI クラス～

次のプログラムを共有サイトに公開します。

- SPI_Class.py
  AD コンバータ(MCP3002)から SPI 通信経由でデータを取得するためのクラス「MCP3002」を定義しています。



- 11_SPI.py
  SPI_Class.py で定義したクラス「MCP3002」を import してアナログ温度センサモジュールから取得した温度データを表示するプログラム。

Copyright © 2024 YITJC All Rights Reserved.

第 4 章 データ取得～生産設備～

生産設備がインターネットにつながることで、遠隔地から生産設備の稼働状況を把握できます。生産設備の IoT 化は、生産設備稼働状況のデータ収集や生産設備異常監視などの業務を自動化し、生産ラインの効率化や異常の早期発見・復旧、生産設備稼働計画の最適化などの効果が期待できます。しかし、生産設備の更新には費用の負担や更新に時間が掛かるため、生産設備の IoT 化は停滞しています。

そこで、本講座では「低コスト」かつ「短期間」で生産設備の IoT 化を実現するため、現在稼働中の生産設備（既存の生産設備）を活用した IoT 化を提案します。具体的には、Raspberry Pi と既存の生産設備との連携により IoT 化を推進し、生産性向上につながるシステムを構築します。

4.1 生産設備との連携

Raspberry Pi を使用して生産設備と連携する方法を 2 つ提案します。

- PLC（シーケンサ）との連携

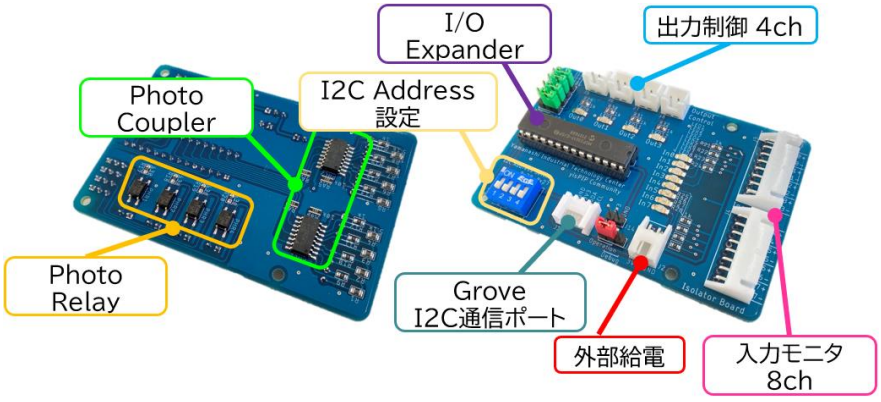
  既存の生産設備を制御している PLC に注目し、Raspberry Pi と PLC との間で通信制御を行います。通信プロトコルには、各 PLC メーカに対応した「Modbus」通信を使用します。

  Modbus 通信は，1979 年に Modicon 社が PLC 向けに策定した通信プロトコルであり、PLC メーカ各社から Modbus 通信に対応した機器が数多く提供されています。Master/Slave 方式で通信が行われ、Master（Raspberry Pi）から Slave（PLC）に対して指令メッセージを送信します。送信メッセージの一部に Function Code と呼ばれる処理命令が含まれており、PLC は Raspberry Pi から命令された処理を実行したあと、応答メッセージを返します。

  通信モードは，Modbus TCP（Transmission Control Protocol）と Modbus RTU（Remote Terminal Unit）に対応しています。Ethernet ポートを持たない PLC は、RS-485 シリアルインターフェースを使用して Modbus RTU による通信を行うことができます。

- Isolator Board の活用

  通信ポートをもたない PLC や PLC を使用しない生産設備などの IoT 化を推進するため、「Isolator Board」を活用します。Isolator Board は、フォトカプラやフォトリレーを使用した入出力回路を備えているため、Raspberry Pi や Sensor Board による生産設備の入出力制御を行うことができます。



Copyright © 2024 YITJC All Rights Reserved.

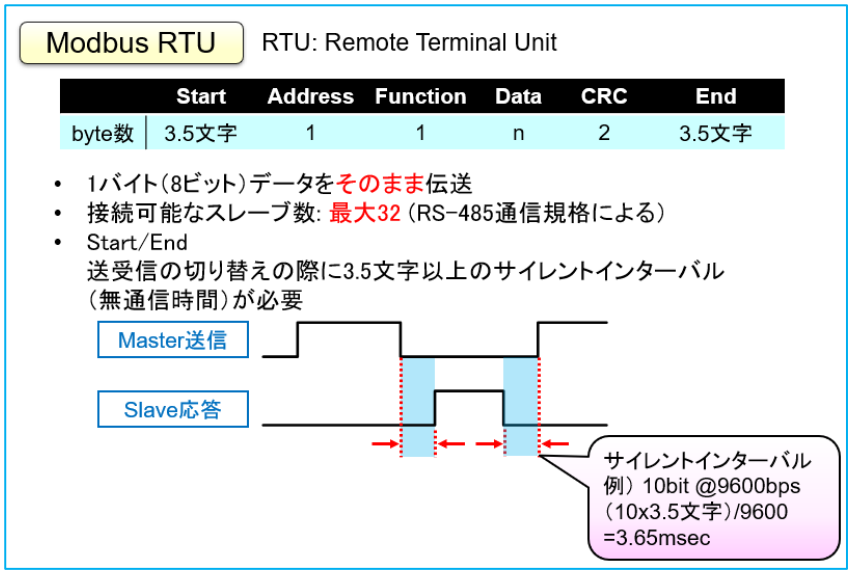
4.2 Modbus 通信

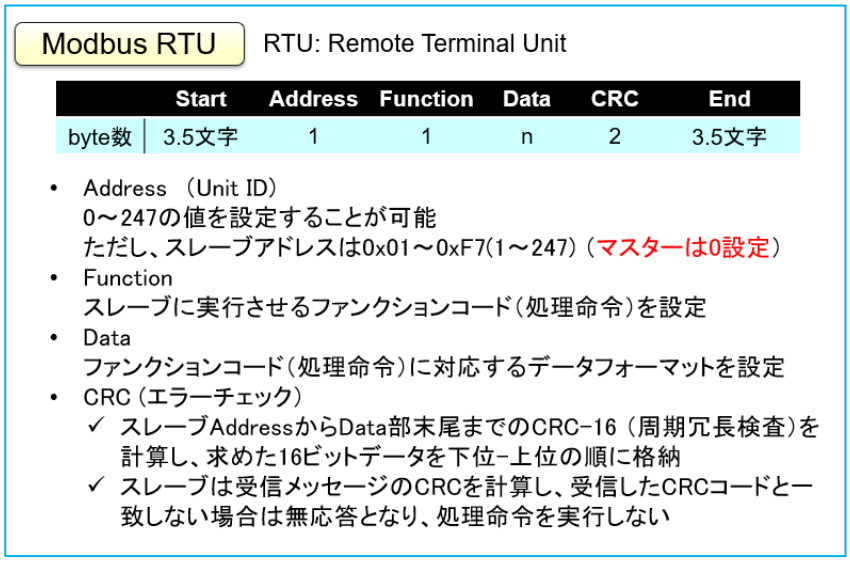
Modbus 通信はプロトコル仕様が一般公開されているため、無料で利用できます。Python には Modbus 通信を行うためのライブラリ「pymodbus」があるため、比較的簡単に Modbus 通信を実装できます。

pymodbus がサポートしている通信モードは、Modbus RTU と Modbus TCP です。

Modbus RTU 通信概要

Modbus RTU 通信は、RS-485 の通信規格を使用します。1Byte データをそのまま伝送しますが、送受信の切り替えの際に 3.5 文字以上のサイレントインターバルを設定する必要があります。接続機器のアドレスは、unit ID で指定します。Master（Raspberry Pi）は必ず 0 番に設定します。Slave（PLC）は 1～247 番を設定します。

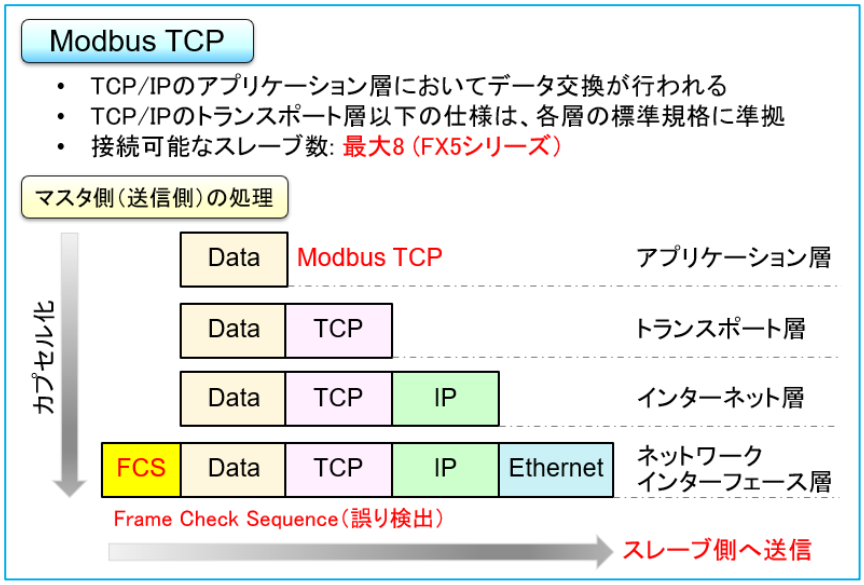


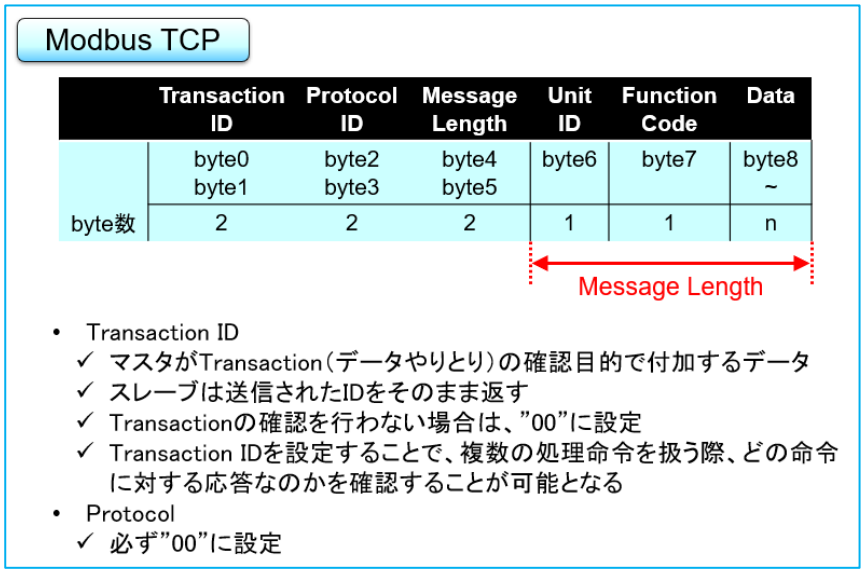


Copyright © 2024 YITJC All Rights Reserved.

Modbus TCP 通信概要

Modbus TCP 通信は、Ethernet の通信規格を使用します。Modbus TCP 通信を行う際、IP Address を指定して接続を確立した後は、トランスポート層以下の仕様は気にする必要はありません。(Raspberry Pi と PLC を同じネットワークに接続して、IP Address を指定するだけで簡単に接続することができます。)





Copyright © 2024 YITJC All Rights Reserved.

Modbus TCP

	Transaction ID	Protocol ID	Message Length	Unit ID	Function Code	Data
	byte0 byte1	byte2 byte3	byte4 byte5	byte6	byte7	byte8 ~
byte数	2	2	2	1	1	n

Message Length

- Message Length
  - ✓ 上位byte (byte4)は必ず0 （メッセージは256byte以下）
  - ✓ Unit ID以降のメッセージ長さ(byte単位)を下位byteに設定
- Unit ID
  - ✓ マスタ側は、0x01～0xF7 (1～247) の値を設定
  - ✓ スレーブ側は、設定された値をそのまま応答
  - ✓ 0x00, 0xF8～0xFF を設定しても、エラーとはならない

Modbus TCP

	Transaction ID	Protocol ID	Message Length	Unit ID	Function Code	Data
	byte0 byte1	byte2 byte3	byte4 byte5	byte6	byte7	byte8 ~
byte数	2	2	2	1	1	n

Message Length

- Function
  スレーブに実行させるファンクションコード(処理命令)を設定
- Data
  ファンクションコード(処理命令)に対応するデータフォーマットを設定

Function Code

Modbus 通信にはさまざまな Function Code(機能)が用意されています。Function Code を指定して Modbus 通信を行うことで、PLC に接続されているすべての入出力機器および PLC 内部データへのアクセス(Read/Write)が可能になります。

Function Code	Function 名称	詳細	複数対応
1 (0x01)	Read Coils	コイル読み出し	○
5 (0x05)	Write Single Coil	コイル書き込み	
3 (0x03)	Read Holding Registers	保持レジスタ読み出し	○
16 (0x10)	Write Multiple Holding Registers	保持レジスタ書き込み	○
2 (0x02)	Read Discrete Input	入力リレー読み出し	
4 (0x04)	Read Input Registers	入力レジスタ読み出し	○
6 (0x06)	Write Single Holding Register	保持レジスタ書き込み	
15 (0x0F)	Write Multiple Coils	コイル書き込み	○

Copyright © 2024 YITJC All Rights Reserved.

Function Code 0x01～Read Coils～

出力コイルの状態を読み取る機能です。

- Read Coils

デバイス割り付け

項目	コイル
デバイス	Y0
リファレンス番号	0
割付点数	1024

マスタ（送信側）

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x01
byte2-3	byte8-9	リファレンス番号
byte4-5	byte10-11	読み出しビット数（1～2000）

送信例）出力コイルY0から連続して10コイル分の状態をRead

byte7	byte8	byte9	byte10	byte11
01	00	00	00	0A

- Read Coils

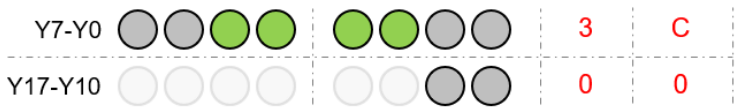
スレーブ（受信側）

- 正常応答

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x01
byte2	byte8	レスポンスバイト数 （読み出しビット数+7）÷8
byte3～	byte9～	読み出しデータ

応答例）出力コイルY0から連続して10コイル分の状態をReadしたときの応答

※Y2-Y5のみ"ON"状態とする。



byte7	byte8	byte9	byte10
01	02	3C	00

Copyright © 2024 YITJC All Rights Reserved.

Function Code 0x05～Write Single Coil～

出力コイルに書き込む機能です。Modbus 通信により出力コイルに「True」または「Faluse」の状態の書き込むことで、出力機器を制御できます。ただし、PLC 制御と Modbus 通信制御で 1 つのコイルを二重で制御しないように注意してください。

- Write Single Coil

マスタ（送信側）

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x05
byte2-3	byte8-9	リファレンス番号
byte4	byte10	1を書き込む場合”FF” 0を書き込む場合”00”
byte5	byte11	“00”固定

送信例）出力コイルY2へ”1” (True) をWrite

byte7	byte8	byte9	byte10	byte11
05	00	02	FF	00

～注意～

Write Single Coilで書き込む出力コイルは、ラダー図上で出力コイルとして記述しない。（ローカル制御とModbus通信制御の2重制御とならないように注意）

- Write Single Coil

スレーブ（受信側）

- 正常応答

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x05
byte2-3	byte8-9	リファレンス番号
byte4	byte10	1を書き込む場合”FF” 0を書き込む場合”00”
byte5	byte11	“00”固定

応答例）出力コイルY2へ”1” (True) をWriteしたときの応答

byte7	byte8	byte9	byte10	byte11
05	00	02	FF	00

Copyright © 2024 YITJC All Rights Reserved.

Function Code 0x03～Read Holding Registers～

保持レジスタの値を読み取る機能です。保持レジスタには、PLC に接続されているセンサのデータや生産数量のカウント値など、生産設備のさまざまなデータが保存されています。

- Read Holding Registers

デバイス割り付け

項目	保持レジスタ
デバイス	D0
リファレンス番号	0
割付点数	8000

マスタ（送信側）

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x03
byte2-3	byte8-9	リファレンス番号
byte4-5	byte10-11	読み出しワード数（1～125）

送信例）保持レジスタ（データレジスタ）D0から2ワードRead

byte7	byte8	byte9	byte10	byte11
03	00	00	00	02

- Read Holding Registers

スレーブ（受信側）

- 正常応答

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x03
byte2	byte8	レスポンスバイト数 （読み出しワード数×2）
byte3～	byte9～	読み出しデータ

応答例）保持レジスタ（データレジスタ）D0から2ワードReadしたときの応答

読み出し値は、
D0: 1054 (0x041E)
D1: 26 (0x001A) とする。

byte7	byte8	byte9	bte10	byte11	byte12
03	04	04	1E	00	1A
		データレジスタ D0		データレジスタ D1	

Copyright © 2024 YITJC All Rights Reserved.

Function Code 0x10～Write Multiple Holding Registers～

保持レジスタに値を書き込む機能です。

- Write Multiple Holding Registers

マスタ（送信側）

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x10
byte2-3	byte8-9	リファレンス番号
byte4-5	byte10-11	書き込みワード数（1～123）
byte6	byte12	書き込みバイト数 （書き込みワード数×2）
byte7～	byte13～	書き込みデータ

送信例）保持レジスタD5, D6に次の値をWrite

書き込み値は、
D5: 555 (0x022B)
D6: 666 (0x029A)　　とする。

byte 7	byte 8	byte 9	bte1 0	byte 11	byte 12	byte 13	byte 14	byte 15	byte 16
10	00	05	00	02	04	02	2B	02	9A

- Write Multiple Holding Registers

スレーブ（受信側）

- 正常応答

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x10
byte2-3	byte8-9	リファレンス番号
byte4～	byte10～	書き込みワード数

応答例）保持レジスタD5, D6に次の値をWriteしたときの応答

書き込み値は、
D5: 555 (0x022B)
D6: 666 (0x029A)　　とする。

byte7	byte8	byte9	byte10	byte11
10	00	05	00	02

Copyright © 2024 YITJC All Rights Reserved.

Function Code 0x02～Read Discrete Input～

入力リレーの状態を読み取る機能です。PLC に接続されているすべての入力機器の状態を読み取ることができます。

- Read Discrete Input

デバイス割り付け

項目	入力リレー
デバイス	X0
リファレンス番号	0
割付点数	1024

マスタ（送信側）

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x02
byte2-3	byte8-9	リファレンス番号
byte4-5	byte10-11	読み出しビット数（1～2000）

送信例）入力リレーX0から連続して10リレー分の状態をRead

byte7	byte8	byte9	byte10	byte11
02	00	00	00	0A

- Read Discrete Input

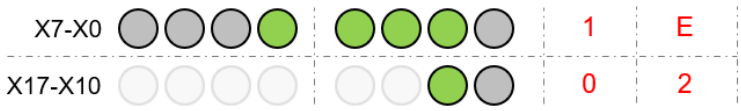
スレーブ（受信側）

- 正常応答

Modbus RTU	Modbus TCP	Data
byte1	byte7	Function Code = 0x02
byte2	byte8	レスポンスバイト数 （読み出しビット数+7）÷8
byte3～	byte9～	読み出しデータ

応答例）入力リレーX0から連続して10リレー分の状態をReadしたときの応答

※X1-X4およびX11が”ON”状態とする。



byte7	byte8	byte9	byte10
02	02	1E	02

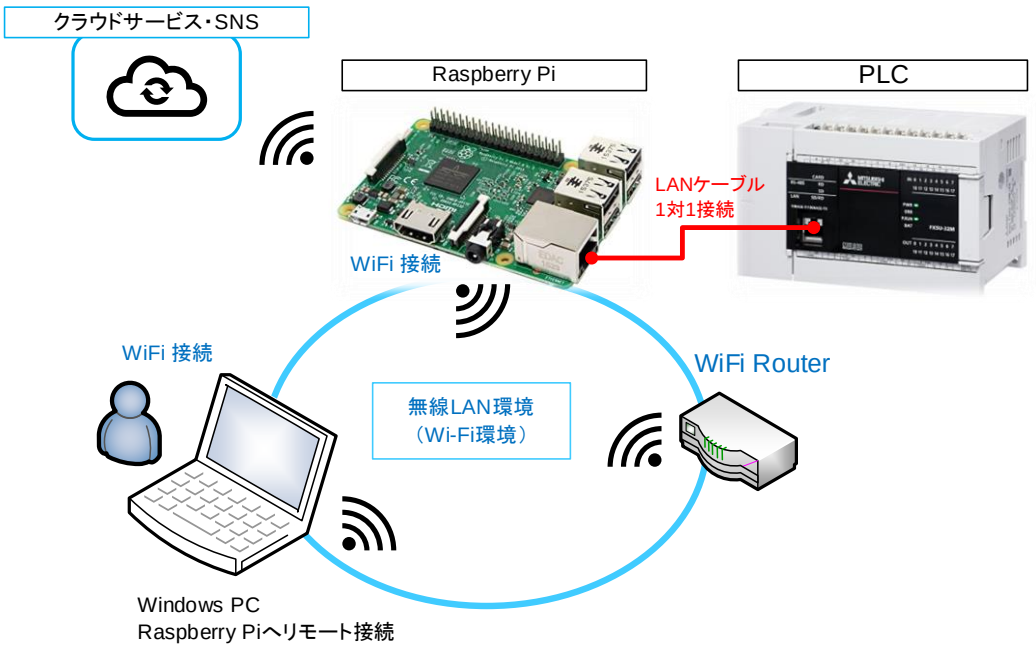
Copyright © 2024 YITJC All Rights Reserved.

4.3 有線ネットワークの設定

PLC と Modbus TCP 通信を行うため、Raspberry Pi と PLC を直接 LAN ケーブルで接続します。本講座は、これまで Windows PC と Raspberry Pi を同じ無線 LAN に接続したネットワーク環境で学習してきましたが、ここでは Raspberry Pi と PLC を直接 LAN ケーブルで接続した状態で Modbus TCP 通信制御について学習します。

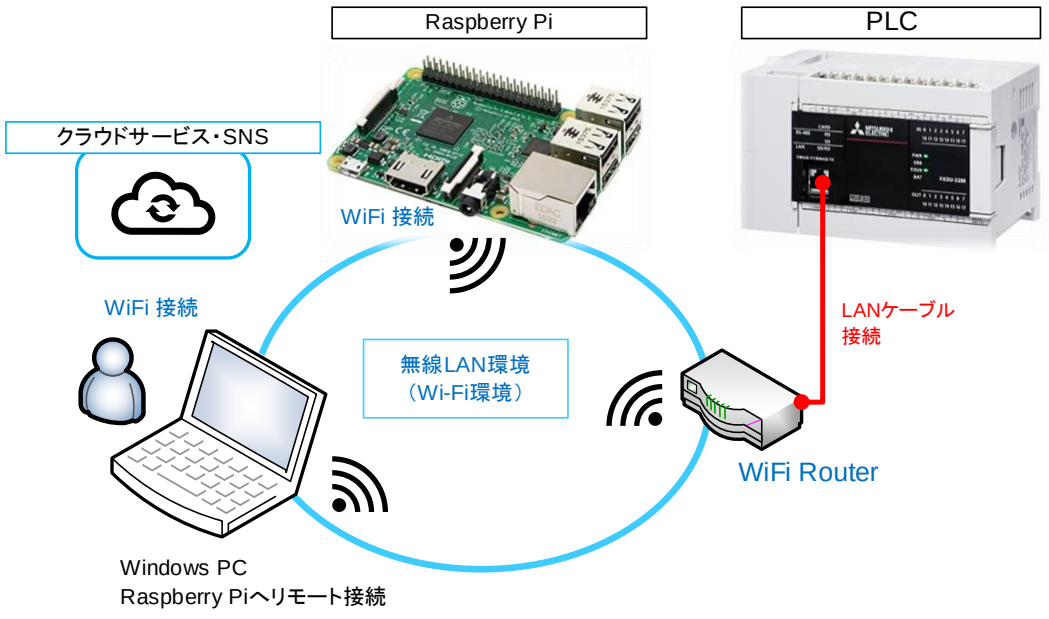
～補足～

本講座は、Raspberry Pi と PLC を直接 LAN ケーブルで接続した状態で実習を行います。つまり、PLC はインターネット環境に接続していない状態です。



参考～PLC が同一ネットワークに接続している場合～

PLC がネットワークに接続している場合は、このあとの手順①～⑦の設定は必要ありません。Raspberry Pi から同一ネットワークに接続されている PLC の IP アドレスを指定して、Modbus 通信制御ができます。



Copyright © 2024 YITJC All Rights Reserved.

手順①～ネットワーク管理リスト表示～

次のコマンドを実行し、NetworkManager が管理しているネットワーク一覧を表示します。

(ycuu) cuu@raspberrypi:~ $ nmcli connection show

～参考～

NetworkManager を使用してネットワークの設定を行う場合、通常「NAME」を指定して設定します。代表的な設定コマンドを紹介します。

- 「NAME」で指定したネットワーク接続の設定をすべて表示
  (ycuu) cuu@raspberrypi:~ $ nmcli connection show NAME

- 「NAME」で指定したネットワーク接続の設定のうち、IPv4 関連の設定を表示
  -f ipv4 と指定することで、IPv4 関連の情報のみを指定しています。
  (ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show NAME

- 「NAME」で指定したネットワーク接続の IP アドレスを変更
  (ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify NAME ipv4.addresses 192.168.10.77/24

- 「NAME」で指定したネットワーク接続の Gateway を変更
  (ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify NAME ipv4.gateway 192.168.10.1

- 「NAME」で指定したネットワーク接続の DNS を変更
  (ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify NAME ipv4.dns 192.168.10.1,8.8.8.8

- 「NAME」で指定したネットワーク接続の方法（自動・手動）を設定
  (ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify NAME ipv4.method manual

Ethernet 接続の「NAME」は、デフォルトで「有線接続1」と日本語表示になっているため、この日本語名を使用してコマンドを実行するとエラーが発生します。この場合は、「NAME」の代わりに「UUID」を使用してください。

各接続には UUID（Universally Unique Identifier）が割り当てられるため、この UUID を使用してネットワークの設定を行います。

Copyright © 2024 YITJC All Rights Reserved.

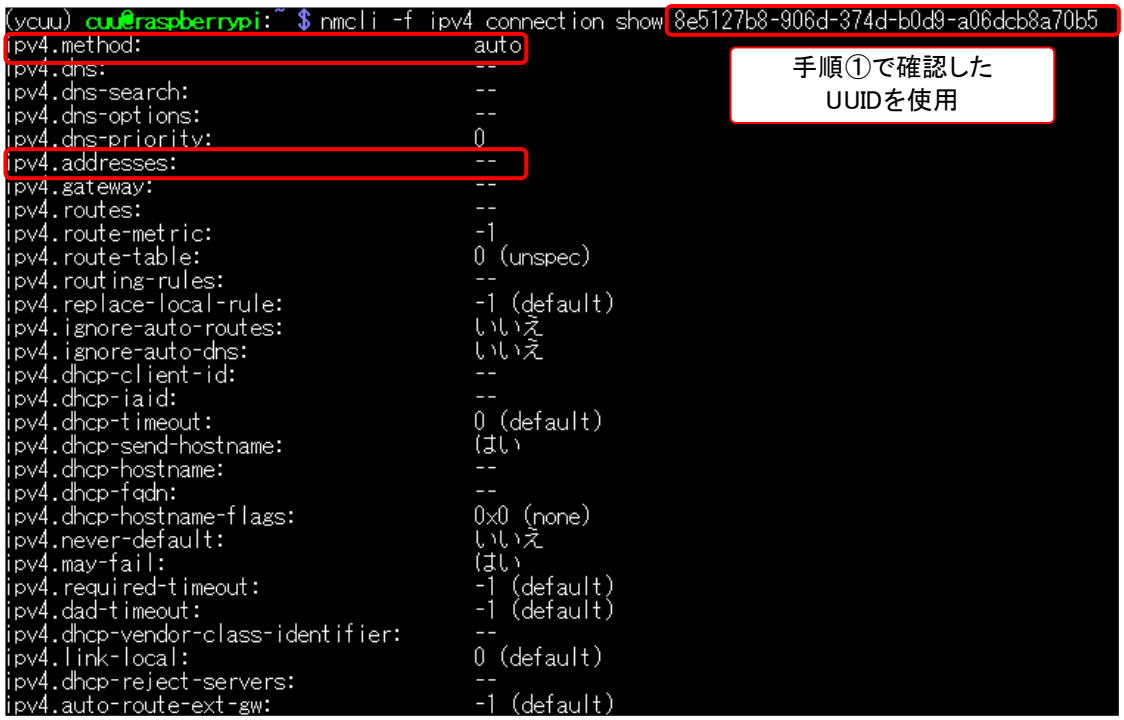
手順②～有線ネットワークの設定確認～

Raspberry Pi 有線ネットワークの設定を確認します。

(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show UUID

UUID は、手順①で確認した値を使用してください。

この画面表示を終了する場合は、「q」を押してください。



手順③～有線ネットワークの IP アドレス設定～

PLC と同じネットワークにするため、Raspberry Pi 有線ネットワークの IP アドレスを変更します。

本講座で使用する PLC の IP アドレスは、172.16.77.72 です。

自社の PLC と接続する場合も、PLC と同じネットワーク接続になるように、IP アドレス（172.16.77 の部分）を合わせるようにしてください。

次のコマンドを実行し、有線ネットワークの IP アドレスを設定します。

本講座では、このコマンドで指定する IP アドレスを使用してください。

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify UUID ipv4.addresses 172.16.77.77/24

次のコマンドを実行し、有線ネットワークの接続方法を「手動（Manual）」に設定します。

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify UUID ipv4.method manual

Copyright © 2024 YITJC All Rights Reserved.

再び有線ネットワークの設定を表示して、IP アドレスと接続方法（Method）が変更されていることを確認してください。

(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show UUID

UUID は、手順①で確認した値を使用してください。

この画面表示を終了する場合は、「q」を押してください。

```
(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify 8e5127b8-906d-374d-b0d9-a06dcb8a70b5 ipv4.addresses 172.16.77.77/24
(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify 8e5127b8-906d-374d-b0d9-a06dcb8a70b5 ipv4.method manual
(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show 8e5127b8-906d-374d-b0d9-a06dcb8a70b5
ipv4.method:                            manual
ipv4.dns:                               --
ipv4.dns-search:                        --
ipv4.dns-options:                       --
ipv4.dns-priority:                      0
ipv4.addresses:                         172.16.77.77/24
ipv4.gateway:                           --
ipv4.routes:                            --
ipv4.route-metric:                      -1
ipv4.route-table:                       0 (unspec)
ipv4.routing-rules:                     --
ipv4.replace-local-rule:                -1 (default)
ipv4.ignore-auto-routes:                いいえ
ipv4.ignore-auto-dns:                   いいえ
ipv4.dhcp-client-id:                    --
ipv4.dhcp-iaid:                         --
ipv4.dhcp-timeout:                      0 (default)
ipv4.dhcp-send-hostname:                はい
ipv4.dhcp-hostname:                     --
ipv4.dhcp-fqdn:                         --
ipv4.dhcp-hostname-flags:               0x0 (none)
ipv4.never-default:                     いいえ
ipv4.may-fail:                          はい
ipv4.required-timeout:                  -1 (default)
ipv4.dad-timeout:                       -1 (default)
ipv4.dhcp-vendor-class-identifier:      --
ipv4.link-local:                        0 (default)
ipv4.dhcp-reject-servers:               --
ipv4.auto-route-ext-gw:                 -1 (default)
```

手順④～PLC との接続確認～

Raspberry Pi と PLC を直接 LAN ケーブルで接続（1 対 1 接続）してください。

続いて、次のコマンドを実行して PLC との接続を確認してください。

(ycuu) cuu@raspberrypi:~ $ ping 172.16.77.72

172.16.77.72 は、本講座で使用する PLC の IP アドレスです。

「ping」コマンドは、Packet Internet Groper の略称で、ネットワーク上のデバイスとの接続をテストするコマンドです。ネットワーク上のデバイス（PLC）に対して、Raspberry Pi から ICMP（Internet Control Message Protocol）エコーリクエスト信号を送信し、デバイス（PLC）からの応答を受信します。これにより、デバイスとの接続や応答時間（通信の遅延）を確認できます。

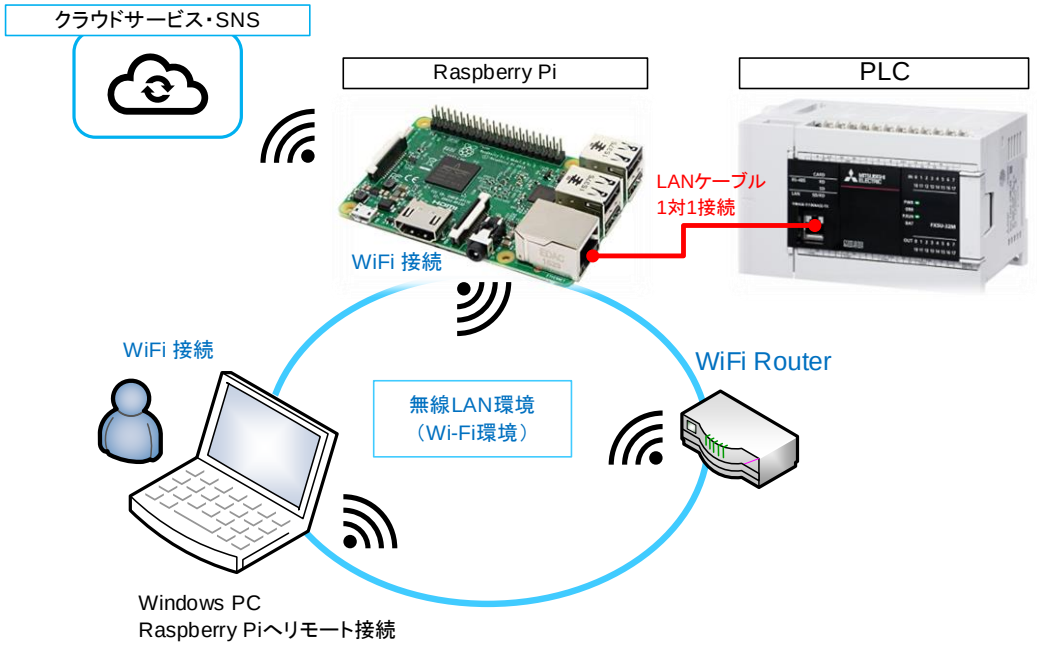
```
(ycuu) cuu@raspberrypi:~ $ ping 172.16.77.72
PING 172.16.77.72 (172.16.77.72) 56(84) bytes of data.
64 bytes from 172.16.77.72: icmp_seq=1 ttl=64 time=2.45 ms
64 bytes from 172.16.77.72: icmp_seq=2 ttl=64 time=0.373 ms
64 bytes from 172.16.77.72: icmp_seq=3 ttl=64 time=0.259 ms
64 bytes from 172.16.77.72: icmp_seq=4 ttl=64 time=0.216 ms
64 bytes from 172.16.77.72: icmp_seq=5 ttl=64 time=0.239 ms
^C
--- 172.16.77.72 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.216/0.708/2.454/0.874 ms
```

ping コマンドを終了する場合は、「Ctrl」+「c」を押してください。

Copyright © 2024 YITJC All Rights Reserved.

手順④まで終了したところで、あらためて Raspberry Pi のネットワーク環境を確認します。

Raspberry Pi は、有線 LAN（PLC との 1 対 1 接続）と無線 LAN（Wi-Fi）の 2 つのネットワーク接続を確立している状態です。



注意～有線 LAN 接続時の注意～

Raspberry Pi と PLC との間を LAN ケーブルで 1 対 1 接続した場合の注意点を説明します。

Raspberry Pi は、有線 LAN と無線 LAN の両方が接続されている場合、有線 LAN による通信の優先順位が高く設定されています。つまり、PLC と通信するための LAN ケーブルを接続した瞬間に有線 LAN による通信が優先され、インターネット接続が遮断されてしまいます。

➔ Raspberry Pi の有線接続は、PLC と 1 対 1 で直接接続しているため、インターネット環境に接続していない点に注意してください。

本講座は、Raspberry Pi とクラウドサービスを連携してさまざまなシステムを開発しますが、有線 LAN による通信が優先されてしまうと、クラウドサービスと連携したシステムを構築できません。そのため、無線 LAN による通信を常に優先する設定を行います。

Copyright © 2024 YITJC All Rights Reserved.

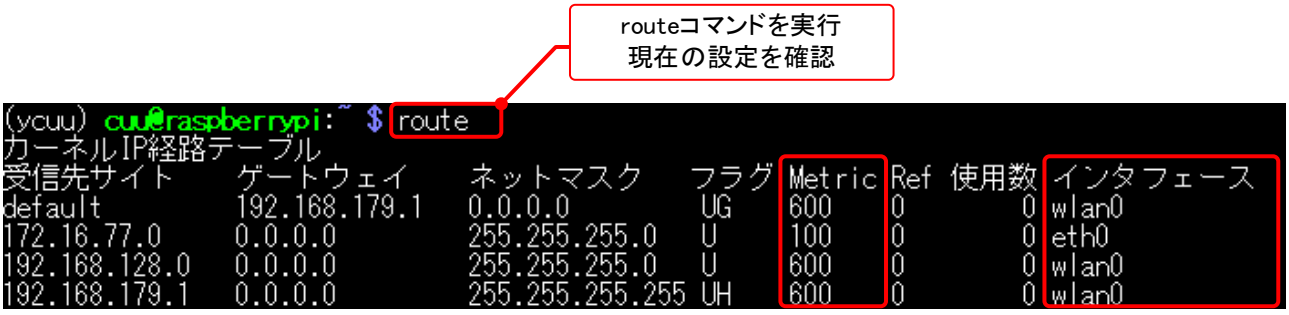
手順⑤～ネットワーク接続の優先順位～

route コマンドを実行し、Raspberry Pi のネットワーク接続優先順位を確認します。

(ycuu) cuu@raspberrypi:~ $ route

Metric の値に注目してください。Metric の値が小さい方の接続が優先されます。

PLC との通信用に設定した「eth0」の Metric 値が 100 に設定されており、無線 LAN（wlan0）の Metric 設定値（600）より小さいことがわかります。この場合、有線 LAN（eth0）による通信が優先されます。



手順⑥～Metric 設定値の変更～

手順①で確認した情報を使用してネットワーク接続の Metric 設定値を変更します。

(ycuu) cuu@raspberrypi:~ $ nmcli connection show

まず、無線 LAN（802ZTa-03698C）の Metric 設定値を変更します。

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify 802ZTa-03698C ipv4.route-metric 100

802ZTa-03698C の部分は、接続している無線 LAN（Wi-Fi）の「NAME」を使用してください。

続いて、有線 LAN の Metric 設定値を変更します。

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection modify UUID ipv4.route-metric 200

UUID は、手順①で確認した値を使用してください。

Copyright © 2024 YITJC All Rights Reserved.

最後に、無線 LAN および有線 LAN のネットワーク設定を表示し、Metric 設定値が変更されていることを確認してください。

- 無線 LAN のネットワーク設定表示

(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show 802ZTa-03698C

```
(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show 802ZTa-03698C
ipv4.method:                            manual
ipv4.dns:                               8.8.8.8
ipv4.dns-search:                        --
ipv4.dns-options:                       --
ipv4.dns-priority:                      0
ipv4.addresses:                         192.168.128.10/24
ipv4.gateway:                           192.168.179.1
ipv4.routes:                            --
ipv4.route-metric:                      100
```

- 有線 LAN のネットワーク設定表示

(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show UUID

```
(ycuu) cuu@raspberrypi:~ $ nmcli -f ipv4 connection show 8e5127b8-906d-374d-b0d9-a06dcb8a70b5
ipv4.method:                            manual
ipv4.dns:                               --
ipv4.dns-search:                        --
ipv4.dns-options:                       --
ipv4.dns-priority:                      0
ipv4.addresses:                         172.16.77.77/24
ipv4.gateway:                           --
ipv4.routes:                            --
ipv4.route-metric:                      200
```

手順⑦～ネットワーク設定変更の反映～

手順⑥で設定（変更）したネットワーク情報をシステムに適用します。

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection reload

- 無線 LAN のネットワーク設定適用

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection up 802ZTa-03698C

- 有線 LAN のネットワーク設定適用

(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection up UUID

```
(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection reload
(ycuu) cuu@raspberrypi:~ $ sudo nmcli connection up 802ZTa-03698C
接続が正常にアクティベートされました (D-Bus アクティブパス: /org/freedesktop/NetworkManager/ActiveConnection/4)
(ycuu) cuu@raspberrypi:~ $  sudo nmcli connection up 8e5127b8-906d-374d-b0d9-a06dcb8a70b5
接続が正常にアクティベートされました (D-Bus アクティブパス: /org/freedesktop/NetworkManager/ActiveConnection/5)
```

route コマンドを実行し、Raspberry Pi のネットワーク接続優先順位が変更されたことを確認します。

(ycuu) cuu@raspberrypi:~ $ route

```
(ycuu) cuu@raspberrypi:~ $ route
カーネルIP経路テーブル
受信先サイト    ゲートウェイ    ネットマスク    フラグ Metric Ref 使用数 インタフェース
default         192.168.179.1   0.0.0.0         UG    100    0        0 wlan0
172.16.77.0     0.0.0.0         255.255.255.0   U     200    0        0 eth0
192.168.128.0   0.0.0.0         255.255.255.0   U     100    0        0 wlan0
192.168.179.1   0.0.0.0         255.255.255.255 UH    100    0        0 wlan0
```

無線LAN（wlan0）通信が優先される設定

Copyright © 2024 YITJC All Rights Reserved.

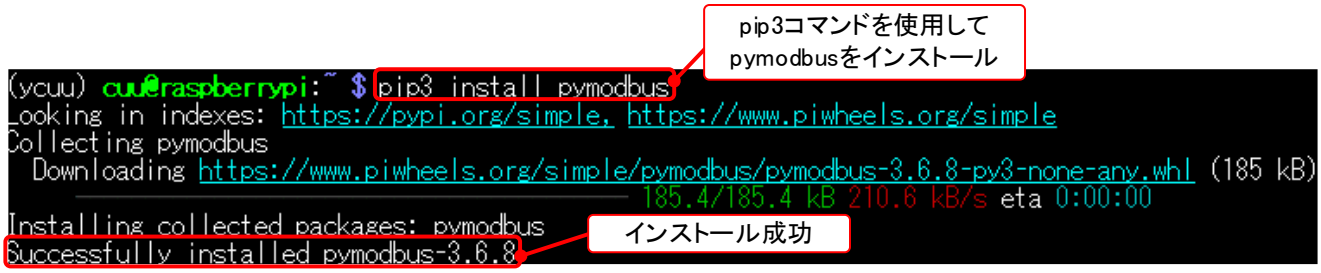
4.4 Modbus TCP 通信

ここでは、Modbus TCP 通信を利用して Raspberry Pi による PLC の通信制御を学習します。

手順①～pymodbus インストール～

Modbus 通信制御を行うために必要なライブラリ「pymodbus」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install pymodbus



手順②～PLC と接続～

Modbus 通信制御を行う PLC と LAN ケーブルで接続します。

本講座で使用する PLC のラダープログラムは次のとおりです。

[対象プログラム]

Raspberry Pi Control.gx3

[プログラム内容]

- ✓ Modbus 通信を使用して Raspberry Pi から制御盤（スイッチやパイロットランプなど）の入出力制御を行います。
- ✓ 制御盤の緊急停止ボタンを押すと、制御内容をすべてリセットします。
- ✓ PLC に接続しているアナログ温度センサのデータを制御盤のディスプレイに表示します。

手順③～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

12_ModbusTCP.py

手順④～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 12_ModbusTCP.py

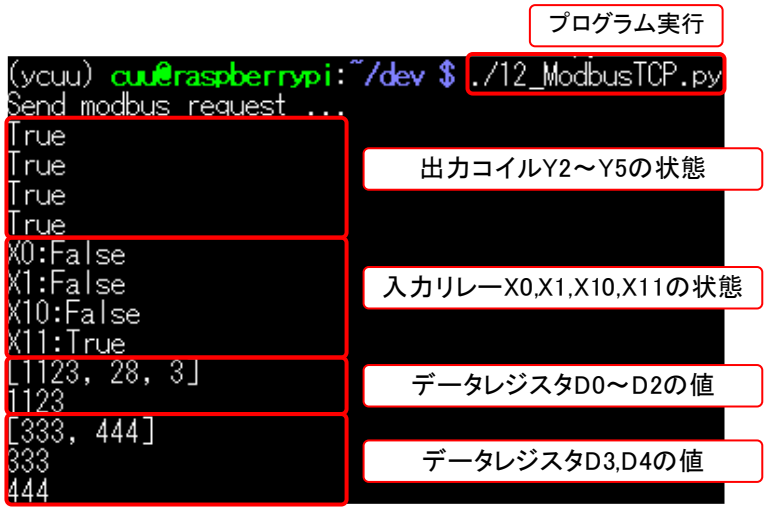
手順⑤～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./12_ModbusTCP.py

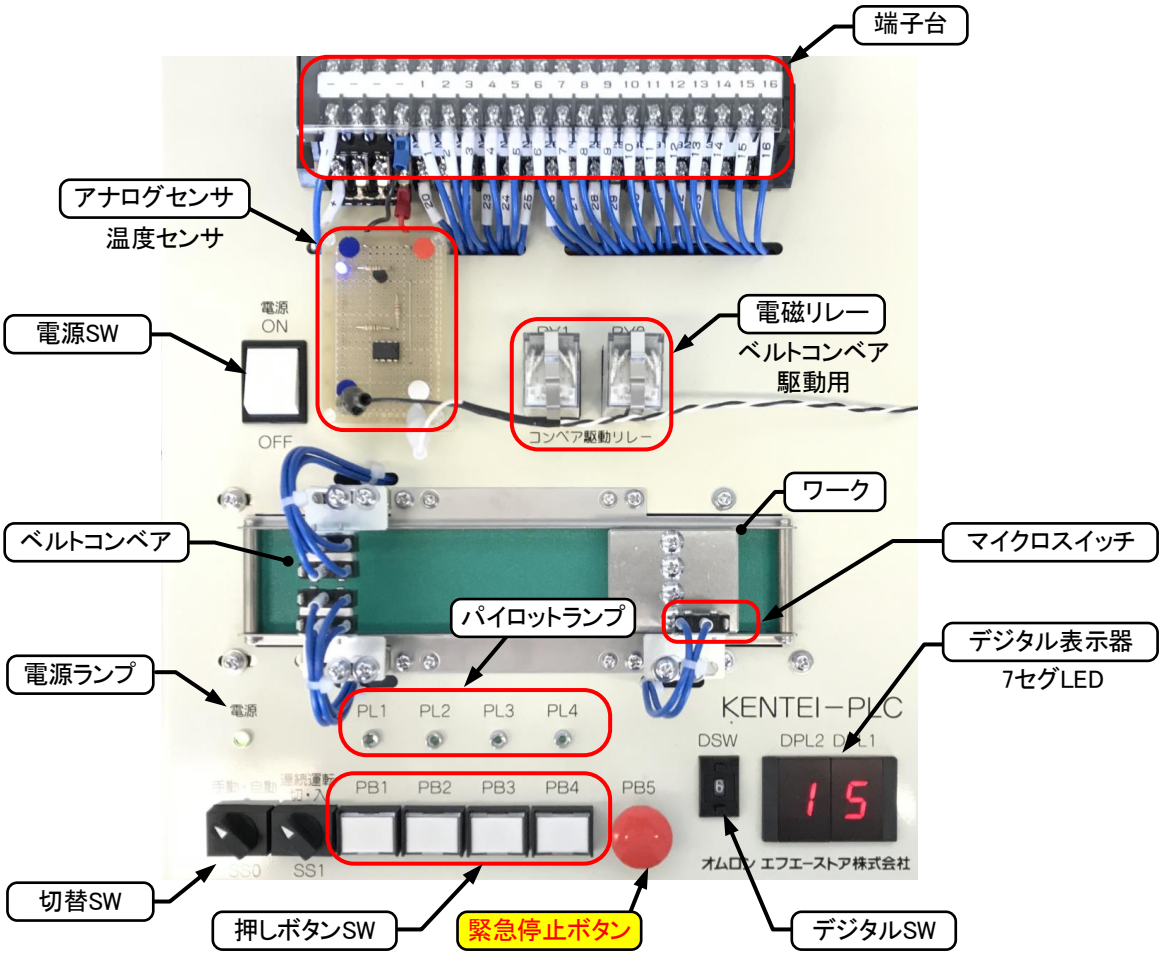
次の画面（実行結果）が表示されることを確認してください。

Copyright © 2024 YITJC All Rights Reserved.



～注意～

プログラムを実行した後は PLC の状態を初期状態に戻すため、必ず制御盤の緊急停止ボタン(PB5:赤いボタン)を押してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説 Modbus TCP 通信～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from pymodbus.client import ModbusTcpClient
5	#PLCのIPアドレスを設定
6	client = ModbusTcpClient('172.16.77.72', port=502)
7	client.connect()
8	
9	try:
10	print("Send modbus request ...")
11	#Write Coil Function Code 0x05
12	#出力コイルY2～Y5に1「True」をWrite
13	client.write_coil(2, True)　#PL1:Y2
14	client.write_coil(3, True)　#PL2:Y3
15	client.write_coil(4, True)　#PL3:Y4
16	client.write_coil(5, True)　#PL4:Y5
17	
18	#Read Coils Function Code 0x01
19	#出力コイルY2～Y5の値をRead
20	result = client.read_coils(2,4)
21	print(result.bits[0])
22	print(result.bits[1])
23	print(result.bits[2])
24	print(result.bits[3])
25	
26	#Read Discrete Input Function Code 0x02
27	#入力リレーX0～X7, X10～X17の値をRead
28	inputrelay = client.read_discrete_inputs(0,16)
29	#異常Responseが0x80以下<であれば、正常Response
30	assert(inputrelay.function_code < 0x80)　　#Error check
31	print ('X0:' + str(inputrelay.bits[0]))
32	print ('X1:' + str(inputrelay.bits[1]))
33	print ('X10:' + str(inputrelay.bits[8]))
34	print ('X11:' + str(inputrelay.bits[9]))
35	
36	#Read Holding Registers　Function Code 0x03
37	#データレジスタD0～D2の値をRead

Copyright © 2024 YITJC All Rights Reserved.

```
38        request = client.read_holding_registers(0,3)
39        #異常Responseが0x80以下<であれば、正常Response
40        assert(request.function_code < 0x80)        #Error check
41        print (request.registers)
42        data = request.registers[0]
43        print(data)
44
45        #Write Multiple Holding Registers    Function Code 0x10
46        #データレジスタD3に"333"、D4に"444"をWrite
47        send = client.write_registers(3,333) #D3
48        send = client.write_registers(4,444) #D4
49
50        #Read Holding Registers Function Code 0x03
51        #D3～D4へ書き込んだデータの確認
52        request2 = client.read_holding_registers(3,2) #D3～D4 Read
53        #異常Responseが0x80以下<であれば、正常Response
54        assert(request2.function_code < 0x80)        #Error check
55        #print request
56        print (request2.registers)
57        data1 = request2.registers[0]
58        data2 = request2.registers[1]
59        print(data1)
60        print(data2)
61
62    finally:
63        client.close()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: ModbusTCPClient ライブラリインポート
- 6 行目: PLC の IP アドレス、ポート番号を指定

  Modbus TCP 通信で使用されるポート番号は、一般的に 502 番が使用されています。
- 9 行目～60 行目: Modbus TCP 通信制御
- 10 行目～16 行目:

  Function Code 0x05～Write Coil～

  制御内容

  出力コイル Y2～Y5 に 1「True」を Write します

  制御結果

  パイロットランプ PL1～PL4 が点灯します

Copyright © 2024 YITJC All Rights Reserved.

- 18 行目～24 行目:
  Function Code 0x01～Read Coils～
  制御内容
  出力コイル Y2～Y5 の状態を Read します
  制御結果
  出力コイル Y2～Y5 の状態を Print 表示します

- 26 行目～34 行目:
  Function Code 0x02～Read Discrete Input～
  制御内容
  入力リレーX0～X7、X10～X17 の状態を Read します
  制御結果
  入力リレーX0, X1, X10, X11 の状態を Print 表示します

- 36 行目～43 行目:
  Function Code 0x03～Read Holding Registers～
  制御内容
  データレジスタ D0～D2 の値を Read します
  制御結果
  データレジスタ D0～D2 の値を Print 表示します
  D0: アナログ温度センサ AD 変換後のデジタル値
  D1: D0 を 40 で割り算した結果（商）
  D2: D0 を 40 で割り算した時の余り

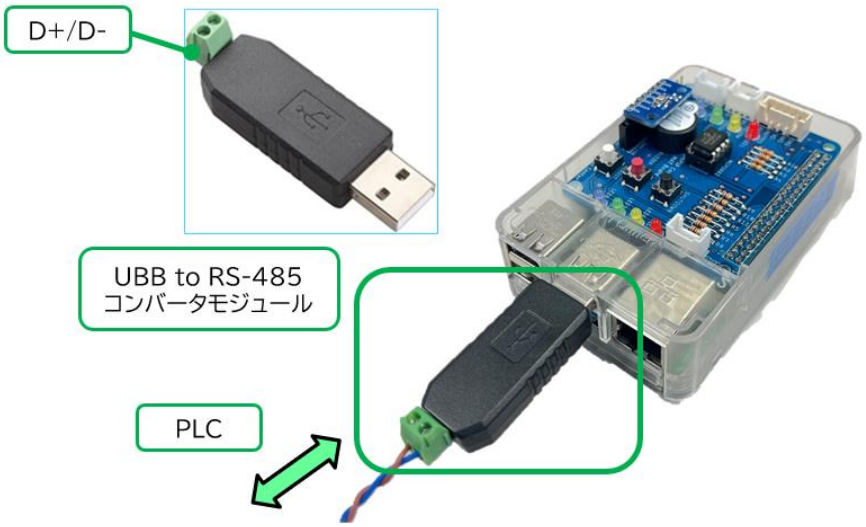
- 45 行目～48 行目:
  Function Code 0x10～Write Multiple Holding Registers～
  制御内容
  データレジスタ D3 へ”333”、D4 へ”444”を Write します

- 50 行目～60 行目:
  Function Code 0x03～Read Holding Registers～
  制御内容
  データレジスタ D3, D4 の値を Read します
  制御結果
  データレジスタ D3, D4 の値を Print 表示します
- 63 行目: 通信ポートクローズ

Copyright © 2024 YITJC All Rights Reserved.

4.5 Modbus RTU 通信

PLC と Modbus RTU 通信を行うためには、Raspberry Pi と PLC との間を RS-485 ケーブルで接続する必要があります。PLC は Modbus RTU 通信を行うために必要な RS-485 ポートを内蔵していますが、Raspberry Pi には RS-485 ポートがないため、UBB to RS-485 コンバータモジュールを使用します。Raspberry Pi の USB ポートにモジュールを接続することで、PLC と Modbus RTU 通信を行うことができます。



ここでは、Modbus RTU 通信を利用した（Raspberry Pi による）PLC の通信プログラムを紹介します。

[対象プログラム]

13_ModbusRTU.py

～プログラム解説 Modbus RTU 通信～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from pymodbus.client import ModbusSerialClient
5	
6	#Modbus RTU通信パラメータ設定
7	#PLC側の設定と一致させる
8	client = ModbusSerialClient(method='rtu',port='/dev/ttyUSB0',parity='N',stopbits=1,bytesize=8,baudrate=9600,timeout=3)
9	
10	try:
11	#マスタ(PLC)側に接続した場合の処理
12	if client.connect():
13	print ("Connected")
14	
15	#Write Coil Function Code 0x05
16	#出力コイルY2～Y5に1「True」をWrite

Copyright © 2024 YITJC All Rights Reserved.

```
17          client.write_coil(2, True)   #PL1:Y2
18          client.write_coil(3, True)   #PL2:Y3
19          client.write_coil(4, True)   #PL3:Y4
20          client.write_coil(5, True)   #PL4:Y5
21
22          #Read Coils Function Code 0x01
23          #出力コイルY2～Y5の値をRead
24          #unitは、PLC側の自局番号
25          result = client.read_coils(address=0x02, count=4,unit=0x01)
26          print(result.bits[0])
27          print(result.bits[1])
28          print(result.bits[2])
29          print(result.bits[3])
30
31          #Read Discrete Input Function Code 0x02
32          #入力リレーX0～X7, X10～X17の値をRead
33          #unitは、PLC側の自局番号
34          inputrelay = client.read_discrete_inputs(address=0x00, count=16,unit=0x01)
35          #異常Responseが0x80以下<であれば、正常Response
36          assert(inputrelay.function_code < 0x80)       #Error check
37          print ('X0:' + str(inputrelay.bits[0]))
38          print ('X1:' + str(inputrelay.bits[1]))
39          print ('X10:' + str(inputrelay.bits[8]))
40          print ('X11:' + str(inputrelay.bits[9]))
41
42          #Read Holding Registers    Function Code 0x03
43          #データレジスタD0～D2の値をRead
44          #unitは、PLC側の自局番号
45          request = client.read_holding_registers(address=0x00, count=3,unit=0x01)
46          #異常Responseが0x80以下<であれば、正常Response
47          assert(request.function_code < 0x80)       #Error check
48          #print request
49          print (request.registers)
50          data = request.registers[0]
51          print(data)
52
53          #Write Multiple Holding Registers    Function Code 0x10
54          #データレジスタD3に"333"、D4に"444"をWrite
55          send = client.write_registers(3,333) #D3
```

Copyright © 2024 YITJC All Rights Reserved.

```
56              send = client.write_registers(4,444) #D4
57
58              #Read Holding Registers Function Code 0x03
59              #D3～D4へ書き込んだデータの確認
60              request2 = client.read_holding_registers(address=0x03, count=2,unit=0x01) #D3～D4 Read
61              #異常Responseが0x80以下<であれば、正常Response
62              assert(request2.function_code < 0x80)       #Error check
63              #print request
64              print (request2.registers)
65              data1 = request2.registers[0]
66              data2 = request2.registers[1]
67              print(data1)
68              print(data2)
69              client.close()
70
71  except:
72      print("Unknown Exception")
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: ModbusSerialClient ライブラリインポート
- 6 行目～8 行目: RTU 通信パラメータ設定（PLC 側の設定と合わせる）
- 10 行目～69 行目: Modbus RTU 通信制御
- 12 行目: PLC 接続
- 13 行目: 接続が成功したら、”Connected”と print 表示
- 15 行目～20 行目:

  Function Code 0x05～Write Coil～

  制御内容

  出力コイル Y2～Y5 に 1「True」を Write します

  制御結果

  パイロットランプ PL1～PL4 が点灯します

- 22 行目～29 行目:

  Function Code 0x01～Read Coils～

  制御内容

  出力コイル Y2～Y5 の状態を Read します

  制御結果

  出力コイル Y2～Y5 の状態を Print 表示します

Copyright © 2024 YITJC All Rights Reserved.

- 31 行目～40 行目:

  Function Code 0x02～Read Discrete Input～

  制御内容

  入力リレーX0～X7、X10～X17 の状態を Read します

  制御結果

  入力リレーX0, X1, X10, X11 の状態を Print 表示します

- 42 行目～51 行目:

  Function Code 0x03～Read Holding Registers～

  制御内容

  データレジスタ D0～D2 の値を Read します

  制御結果

  データレジスタ D0～D2 の値を Print 表示します

  D0: アナログ温度センサ AD 変換後のデジタル値

  D1: D0 を 40 で割り算した結果（商）

  D2: D0 を 40 で割り算した時の余り

- 53 行目～56 行目:

  Function Code 0x10～Write Multiple Holding Registers～

  制御内容

  データレジスタ D3 へ”333”、D4 へ”444”を Write します

- 58 行目～68 行目:

  Function Code 0x03～Read Holding Registers～

  制御内容

  データレジスタ D3, D4 の値を Read します

  制御結果

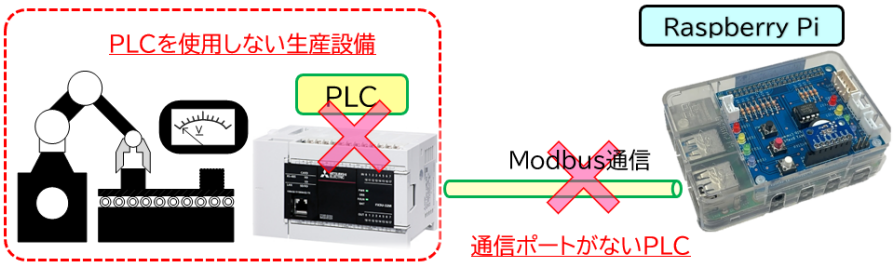
  データレジスタ D3, D4 の値を Print 表示します

- 69 行目: 通信ポートクローズ
- 71 行目～72 行目: 例外処理

Copyright © 2024 YITJC All Rights Reserved.

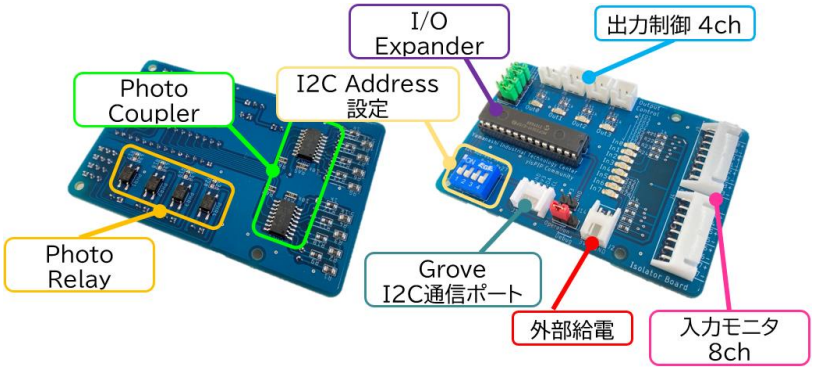
4.6 Isolator Board の活用

生産設備と連携する方法として「PLC との Modbus 通信」を提案しました。しかし、生産設備の中には、PLC を使用しない生産設備や通信ポートがない PLC も存在します。「Isolator Board」は、このような生産設備を IoT 化するために開発した基板です。ここでは、Isolator Board の機能と使い方について学習します。



回路構成と機能

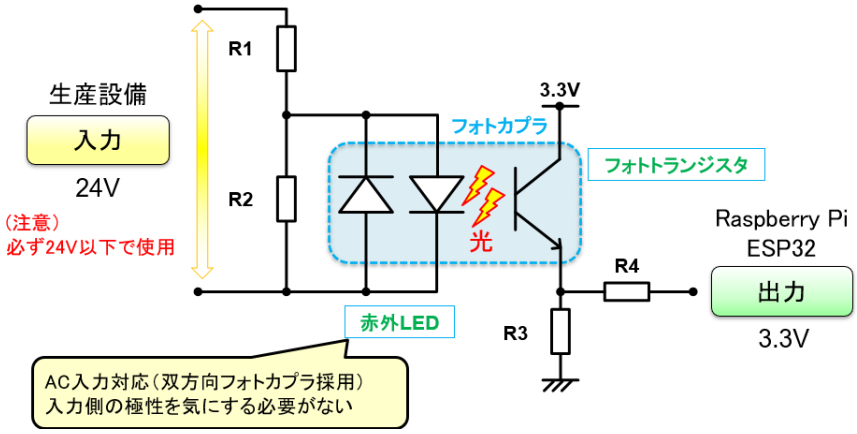
Isolator Board は、フォトカプラやフォトリレーを使用した入出力回路を備えているため、Raspberry Pi による生産設備の入出力制御を行うことができます。Raspberry Pi と Isolator Board との間は Grove ケーブルで接続し、I2C 通信を行います。また、拡張性を確保するため、Raspberry Pi（HAT Board）1 台に対して、最大 8 台の Isolator Board を接続できる構成で設計しています。



入力回路

フォトカプラを使用して、Raspberry Pi と生産設備との間を電気的に絶縁（Isolate）した状態で信号伝達します。具体的には、生産設備の「電気信号」をフォトカプラ内部の赤外 LED で「光信号」に変換し、その光信号をフォトカプラ内部のフォトトランジスタで「電気信号」に戻して出力します。

Isolator Board は、生産設備の電気信号が 24V 系であると想定して回路設計しています。そのため、入力電圧は必ず 24V 以下で使用してください。なお、双方向フォトカプラ（赤外 LED が双方向に搭載されているフォトカプラ）を使用しているため、入力側の極性を気にする必要はありません。

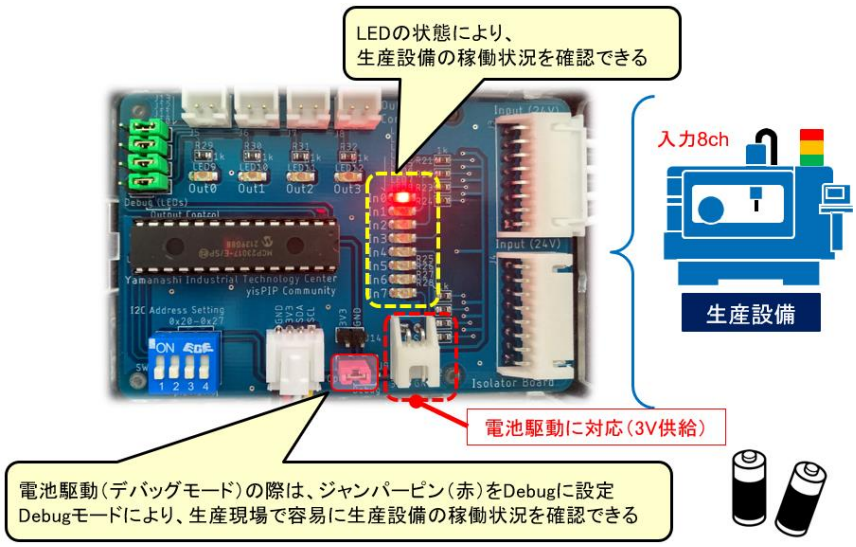


Copyright © 2024 YITJC All Rights Reserved.

入力機能

Isolator Board の入力は、生産設備の稼働信号を受信します。入力は 8ch あり、各 ch に対応した LED を実装しています。そのため、Isolator Board 上に実装されている LED の状態（点灯・消灯）から生産設備の稼働状況を確認できます。

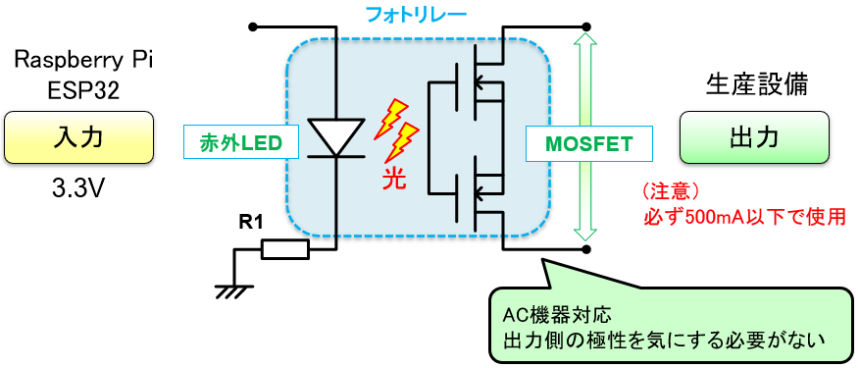
デバッグモードで使用する場合は、電池駆動（3V）で生産設備の稼働状況を確認できます。これにより、生産現場に開発環境（ノート PC や Raspberry Pi）を持ち込まなくても、Isolator Board 団体で生産設備との接続確認やシステムのデバッグを実施できます。



出力回路

フォトリレーを使用して、Raspberry Pi と生産設備との間を電気的に絶縁（Isolate）した状態で信号伝達します。具体的には、Raspberry Pi の「電気信号」をフォトリレー内部の赤外 LED で「光信号」に変換し、その光信号によってフォトリレー内部の MOSFET が導通します。

使用しているフォトリレーは、出力最大電流が 500mA の仕様となっています。そのため、生産設備側の出力回路に流れる電流は、必ず 500mA 以下で使用してください。なお、MOSFET は双方向導通特性があるため、出力側の極性を気にする必要はありません。

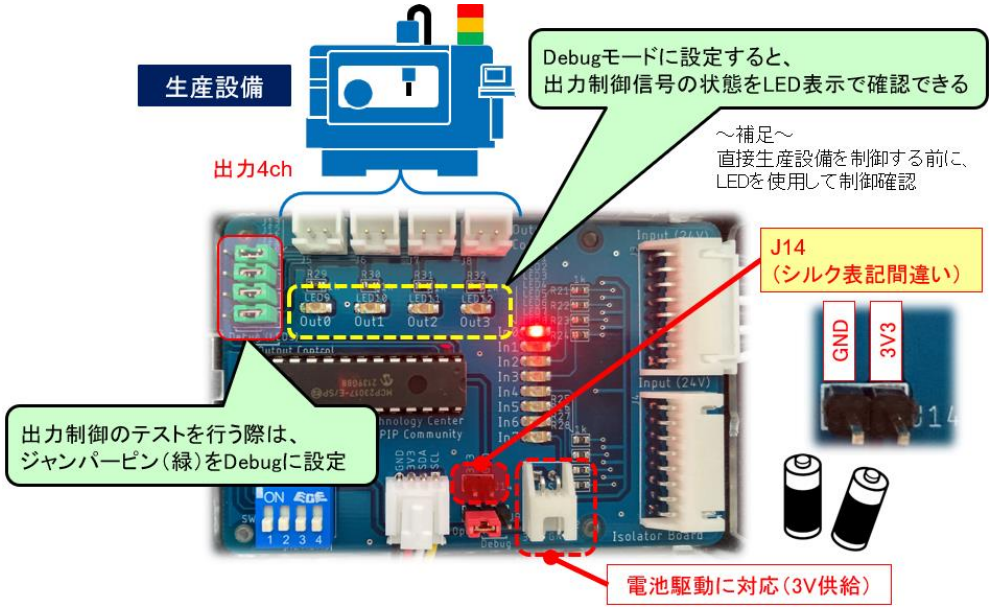


Copyright © 2024 YITJC All Rights Reserved.

出力機能

Isolator Board の出力機能を使用して、Raspberry Pi から生産設備を制御できます。出力は 4ch あり、出力制御のテストを行うためのデバッグモードを用意しています。

デバッグモードで使用する場合は、出力ジャンパーピン（緑）をデバッグに設定します。デバッグモードに設定すると、Raspberry Pi からの出力制御信号の状態（High・Low）を LED で確認できます。そのため、生産設備と接続しない状態でもシステム（出力制御）のデバッグを実施できます。



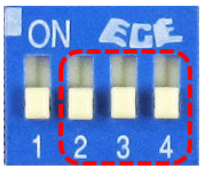
Raspberry Pi との接続

Isolator Boad には、入力 8ch、出力 4ch の合計 12 本の信号線があります。第 2 章で学習したとおり、Raspberry Pi の GPIO（汎用入出力）端子は 26 本しかありません。そのため、Isolator Board の信号線を Raspberry Pi と直接接続しようとすると、GPIO 端子が不足します。この問題を解決するため、Isolator Board には、I/O Expander（IO 拡張 IC）を用意しています。I/O Expander は、Raspberry Pi から I2C 通信制御で I/O Expander の入出力端子を制御できます。また、I2C 通信に必要な信号線は 4 本であるため、Raspberry Pi（HAT Board）と Isolator Board 間の信号線を最小限に抑えることができます。

Copyright © 2024 YITJC All Rights Reserved.

さらに、Isolator Board は、I2C 通信のアドレスを設定するための DIP スイッチを用意しています。DIP スイッチの設定により、I2C 通信のアドレスを 8 パターン設定できるため、Raspberry Pi（HAT Board）1 台に対して、最大 8 台の Isolator Board を接続できます。

DIP SWにより
I2C Address設定



SW2	SW3	SW4	I2C Address
A2	A1	A0	
0	0	0	0x20
0	0	1	0x21
0	1	0	0x22
0	1	1	0x23
1	0	0	0x24
1	0	1	0x25
1	1	0	0x26
1	1	1	0x27

参考～I2C 通信距離延長モジュール～

Isolator Board は生産設備の入出力制御を行うことが目的であるため、生産設備の近くに配置するケースが多いです。そのため、生産設備の近くに配置した Isolator Board と Raspberry Pi（HAT Board）間の通信距離が長くなる可能性があります。I2C 通信の距離が長くなると、信号の品質が劣化して正常な通信ができません。

この問題を解決するため、I2C 通信距離延長モジュールを紹介します。このモジュールにより、LAN ケーブルを使用して、数 m～数十 m の距離で I2C 通信が可能になります。

Copyright © 2024 YITJC All Rights Reserved.

4.7 生産設備との接続

Isolator Board と生産設備との接続方法を学習するため、工作機械を例に挙げて説明します。工作機械は、機械内部の電気系統を管理するため、「端子台」が使用されます。工作機械では、端子台を介して、モータやセンサ、スイッチ、制御ユニットなどの電気部品へ配線されています。そのため、工作機械で稼働している電気部品の制御信号は端子台から取り出すことができます。

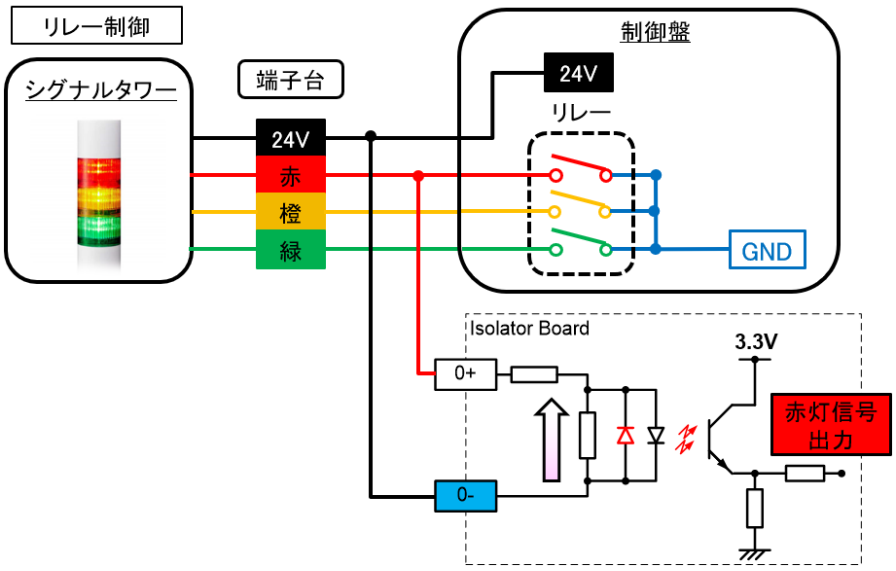
端子台のイメージ

工作機械では、「リレー制御」と「I/O ボード制御」のいずれかの方法で制御が行われています。ここでは、工作機械に設置されているシグナルタワー（信号灯）の制御信号を Isolator Board で取り出す方法を説明します。

- **リレー制御**

  リレー制御は、シグナルタワーへ 24V を供給し、制御盤内部のリレー制御（On/Off）によって各灯の点灯を制御します。Isolator Board と接続する場合は、片方の端子（この図では 0-）を 24V、もう片方の端子（この図では 0+）を取り出したい信号（この場合は赤灯信号）に接続します。これにより、Raspberry Pi は Isolator Board を介して赤灯信号を受信できます。

  なお、Isolator Board の入力回路は、双方向フォトカプラを使用しているため、端子台と接続する場合に極性を気にする必要はありません（0+/0-を逆に接続しても問題ありません）。

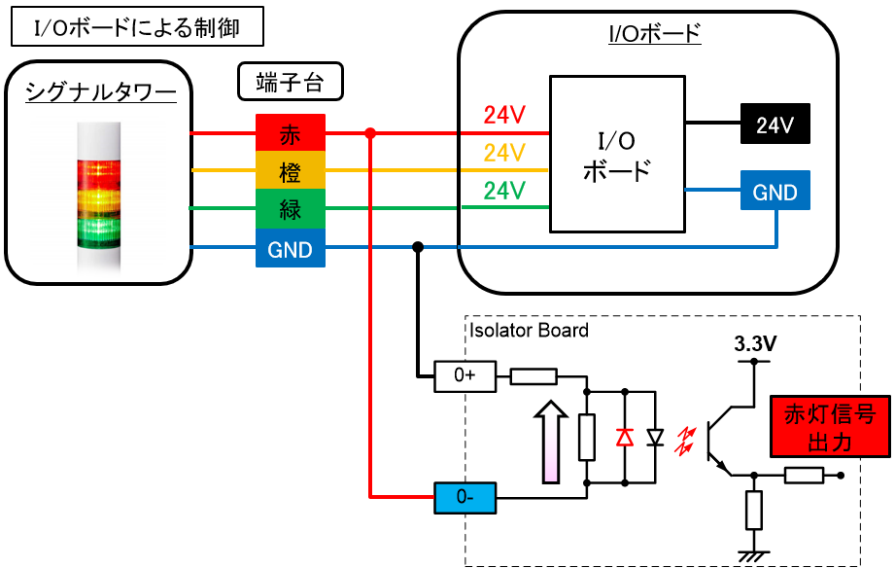


Copyright © 2024 YITJC All Rights Reserved.

- **I/O ボード制御**

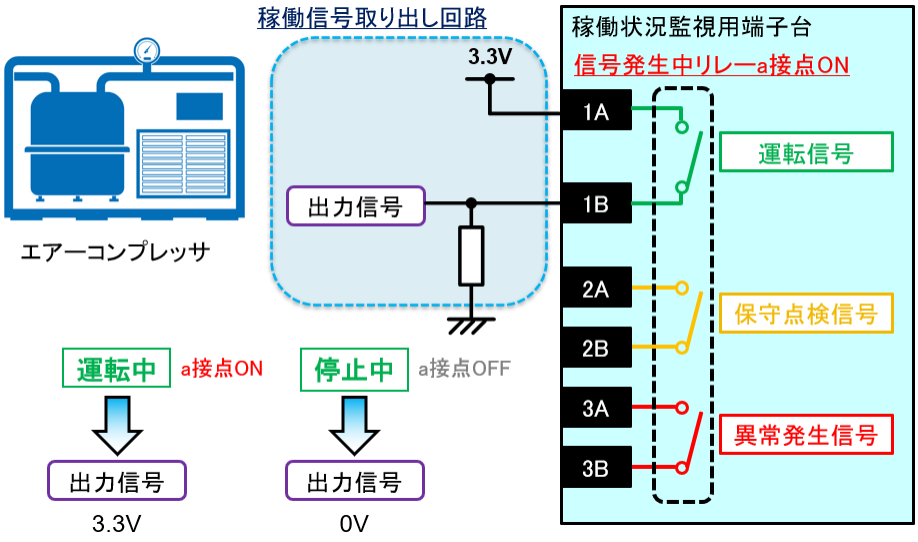
  I/O ボード制御は、24V 信号の入出力が可能な I/O ボードによって各灯の点灯を制御します。Isolator Board と接続する場合は、片方の端子（この図では 0+）を GND、もう片方の端子（この図では 0-）を取り出したい信号（この場合は赤灯信号）に接続します。これにより、Raspberry Pi は Isolator Board を介して赤灯信号を受信できます。

  なお、Isolator Board の入力回路は、双方向フォトカプラを使用しているため、端子台と接続する場合に極性を気にする必要はありません（0+/0-を逆に接続しても問題ありません）。



参考～エアーコンプレッサとの接続～

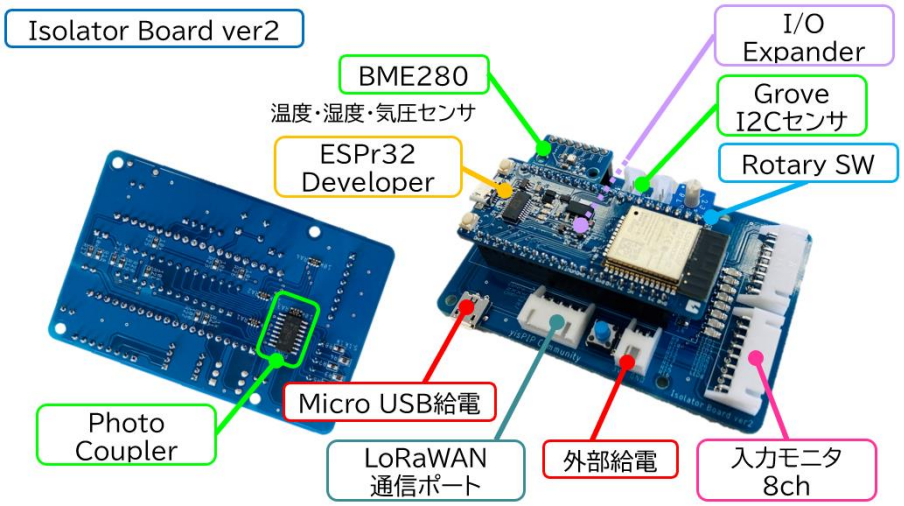
エアーコンプレッサは、稼働状況（運転、保守警告、異常発生）を監視するための端子台が用意されているケースが多いです。しかし、エアーコンプレッサは工作機械などの生産設備と比べて、稼働信号を取り出す回路の構成が異なります。エアーコンプレッサに用意されている端子台は a 接点スイッチで構成されており、稼働信号（運転、保守警告、異常発生）が発生している間だけ a 接点スイッチが ON になる回路構成です。



Copyright © 2024 YITJC All Rights Reserved.

- Isolator Board ver2 の活用

  Isolator Board ver2 は、エアーコンプレッサの稼働信号を取得し、無線通信（Wi-Fi、LPWA など）で稼働信号を送信できます。生産設備の稼働信号を受信する入力回路は 8ch あり、そのうち 4ch はエアーコンプレッサの稼働信号に対応しています。ESP32 を搭載しているため、生産設備やセンサからのデータ取得に加えて、Raspberry Pi やクラウドサービスへのデータ送信もできます。



Copyright © 2024 YITJC All Rights Reserved.

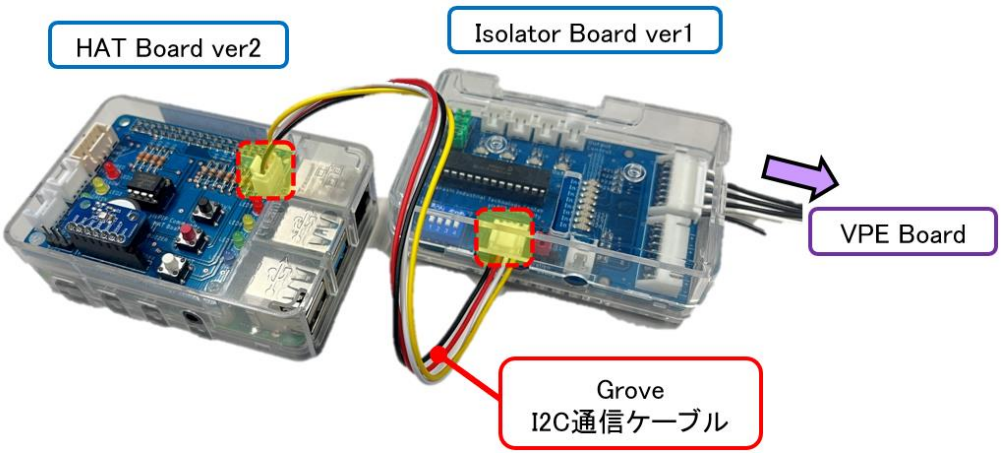
4.8 Isolator Board の制御

Isolator Board を使用して生産設備と連携したシステムを構築する場合、稼働中の生産設備を使用してデバッグを行うことは困難です。そのため、生産設備を想定した VPE Board（仮想生産設備基板：Virtual Production Equipment）を使用して生産設備の入出力制御を学習します。

手順①～Isolator Board, VPE Board の接続～

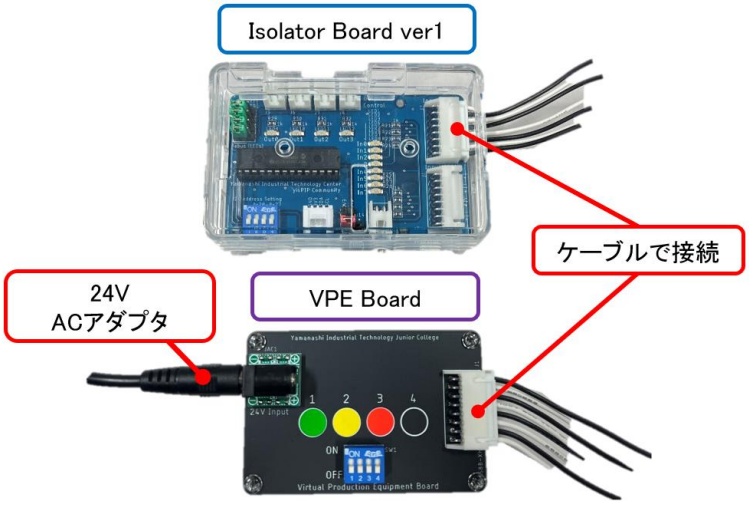
この作業は、必ず Raspberry Pi の電源を切った状態で行ってください。

まず、Raspberry Pi と Isolator Board を Grove I2C 通信ケーブルで接続します。



このとき、Isolator Board の出力ジャンパーピン（緑）をデバッグ側（Debug 側）に設定してください。

続いて、Isolator Board と VPE Board をケーブルで接続してください。ケーブルで接続したあと、24V の AC アダプタを VPE Board に接続して電源を供給してください。



Copyright © 2024 YITJC All Rights Reserved.

手順②～Isolator Board 動作確認～

Raspberry Pi に電源を供給し、Isolator Board の動作を確認します。

Isolator Board の入力 LED（赤色）と VPE Board の DIP スイッチは、次のとおり接続しています。

VPE Board の DIP スイッチの状態（On/Off）に応じて、Isolator Board の LED 表示が変化することを確認してください。

Isolator Board	VPE Board
In0	DIP スイッチ 1
In1	DIP スイッチ 2
In2	DIP スイッチ 3
In3	DIP スイッチ 4
In4	未接続
In5	未接続
In6	未接続
In7	未接続

手順③～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

MCP23017_Class.py

手順④～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

(ycuu) cuu@raspberrypi:~/dev $ chmod +x MCP23017_Class.py

手順⑤～プログラム実行～

対象プログラムを実行します。

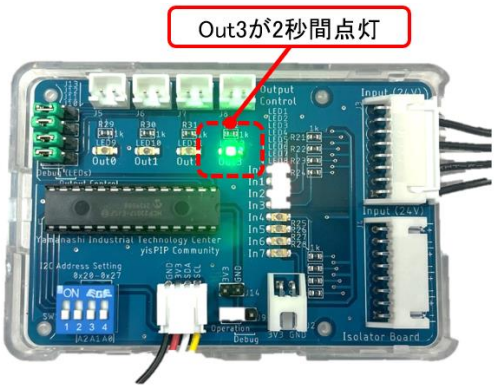
(ycuu) cuu@raspberrypi:~/dev $ ./MCP23017_Class.py

次の画面（実行結果）が表示されることを確認してください。このプログラムでは、次の処理を実行しています。

- 入力 GPIO[1] VPE Board（DIP スイッチ 2 番）の状態を Read
- 出力 GPIO[3]に High を Write

Copyright © 2024 YITJC All Rights Reserved.

プログラムを実行すると、GPIO[3]に High が Write されるため、Out3 の出力 LED（緑）が 2 秒間点灯します。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding:utf-8
3	
4	import time
5	from smbus2 import SMBus
6	
7	#I/O Expander MCP23017データ取得クラス定義
8	class MCP23017:
9	#コンストラクタ定義
10	#config, ctrl_meas, ctrl_muレジスタ設定
11	def __init__(self, busNumber=1, i2cAddress=0x20):
12	#I2C Address設定
13	#データ格納変数定義
14	self.bus = SMBus(busNumber)
15	self.i2cAddress = i2cAddress # Device address (A0-A2)
16	self.IODIRA = 0x00 # Pin direction GPIOA (1:input, 0:output)
17	self.IODIRB = 0x01 # Pin direction GPIOB (1:input, 0:output)
18	self.GPIOA = 0x12 # Register for GPIOA
19	self.GPIOB = 0x13 # Register for GPIOB
20	self.OLATA = 0x14 # Output Latch RegisterA
21	self.OLATB = 0x15 # Output Latch RegisterB
22	
23	def input(self,port):
24	#Pin direction RegisterA (1:input, 0:output)
25	self.bus.write_byte_data(self.i2cAddress, self.IODIRA, 0xFF)
26	#RegisterA for inputs
27	data = self.bus.read_byte_data(self.i2cAddress, self.GPIOA, 1)

Copyright © 2024 YITJC All Rights Reserved.

```
28
29              if port == 0:
30                  print ("GPIOA[0]:" + str((data >>0)&1))
31              elif port == 1:
32                  print ("GPIOA[1]:" + str((data >>1)&1))
33              elif port == 2:
34                  print ("GPIOA[2]:" + str((data >>1)&1))
35              elif port == 3:
36                  print ("GPIOA[3]:" + str((data >>1)&1))
37              elif port == 4:
38                  print ("GPIOA[4]:" + str((data >>1)&1))
39              elif port == 5:
40                  print ("GPIOA[5]:" + str((data >>1)&1))
41              elif port == 6:
42                  print ("GPIOA[6]:" + str((data >>1)&1))
43              elif port == 7:
44                  print ("GPIOA[7]:" + str((data >>1)&1))
45              time.sleep(2)
46              return data
47
48      def output(self,port):
49          #Pin direction RegisterB (1:input, 0:output)
50          self.bus.write_byte_data(self.i2cAddress, self.IODIRB, 0x00)
51          if port == 0:
52              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x01)
53              print ("Writing port0 completed!!")
54              time.sleep(2)
55              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x00)
56          elif port == 1:
57              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x02)
58              print ("Writing port1 completed!!")
59              time.sleep(2)
60              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x00)
61          elif port == 2:
62              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x04)
63              print ("Writing port2 completed!!")
64              time.sleep(2)
65              data =   self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x00)
66          elif port == 3:
```

Copyright © 2024 YITJC All Rights Reserved.

```
67                  data =    self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x08)
68                  print ("Writing port3 completed!!")
69                  time.sleep(2)
70                  data =    self.bus.write_byte_data(self.i2cAddress, self.GPIOB, 0x00)
71
72  if __name__ == '__main__':
73      exp = MCP23017()
74      try:
75          #GPIOA Read
76          exp.input(1)
77          #GPIOB Write
78          exp.output(3)
79      except KeyboardInterrupt:
80          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: time ライブラリをインポート
- 5 行目: smbus2 ライブラリをインポート
- 8 行目～70 行目: I/O Expander（MCP23017）クラス定義
  ここでは、MCP23017 という名前のクラスを定義し、モノが行う仕事内容を記述しています。
- 11 行目～21 行目: コンストラクタ定義
  コンストラクタは次のように記述し、クラスのインスタンスが生成されるときに自動的に呼び出されます。
  def __init__(self, 引数 2, 引数 3, ...) :
  コンストラクタの名前は「__init__」に決まっています。名前の前後に「__」が付いているメソッドは「特殊メソッド」と呼ばれます。コンストラクタは特殊メソッドの 1 つです。
  ここでは、I2C Address、MCP23017 内部レジスタのアドレスなどを定義しています。
  MCP23017 レジスタに関する詳細は、Microchip Technology 社のデータシートに詳細が書かれています
- 16 行目～21 行目: MCP23017 内部レジスタアドレスを設定
  GPIOA（8ch）及び GPIOB（8ch）のレジスタアドレスを設定しています
- 23 行目～46 行目: 入力処理を定義
- 25 行目: GPIOA（8ch）をすべて入力に設定
- 27 行目: GPIOA（8ch）のデータを 1Byte（8bit）読み取り、結果を変数 data に格納
- 29 行目～44 行目: 各ポート（GPIOA[0]～GPIOA[7]）の状態を読み取り、結果を print 表示
- 45 行目: 2 秒間のスリープ
- 46 行目: 読み取った値 data を返しています（data は、この関数の戻り値として返されます）

Copyright © 2024 YITJC All Rights Reserved.

- 48 行目～70 行目: 出力処理を定義
- 50 行目: GPIOB（8ch）をすべて出力に設定
- 51 行目～70 行目: 各ポート（GPIOB[0]～GPIOB[3]）について次の処理を実行
  GPIOB[x]ポートから High（3.3V）を出力　※出力 LED（緑）点灯
  出力したことを print 表示で通知（Writing port completed!!）
  2秒間のスリープ
  GPIOB 全ポートから Low（0V）を出力　※出力 LED（緑）消灯
- 72 行目～80 行目: メイン関数
  if __name__ == '__main__':
  このブロックで行っている処理は、本プログラム（MCP23017_Class.py）を実行したときだけ行われます。
  他のプログラムから MCP23017 クラスをインポート（import）した場合は、メイン関数内の処理は実行されません。
- 73 行目: 変数 exp に MCP23017 クラスのインスタンスを代入しています
- 74 行目～78 行目: try ブロック
- 76 行目: ポート番号 1（GPIOA[1]）の状態を読み取る
  GPIOA[1]は、VPE Board 上の DIP スイッチ 2 番の状態（信号）が入力されています。
- 78 行目: ポート番号 3（GPIOB[3]）の Write 処理（2 秒間 High 出力）を実行
- 79 行目～80 行目: except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～モジュールのインポート(import)～

MCP23017_Class.py で定義したクラス「MCP23017」を読み込んで(import して)使用するプログラムを紹介します。

対象プログラムを実行します。

[対象プログラム]

14_Isolator.py

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加してから実行します。

```
(ycuu) cuu@raspberrypi:$ cd dev
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 14_Isolator.py
(ycuu) cuu@raspberrypi:~/dev $ ./14_Isolator.py
```

プログラムを実行すると、次の画面が表示されます。

手順⑤で実行した「MCP23017_Class.py」と同じ結果(Out3 の出力 LED(緑)が 2 秒間点灯)になることを確認してください。

```
(ycuu) cuu@raspberrypi:~/dev $ ./14_Isolator.py
GPIOA[1]:0
Writing port3 completed!!
```

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	# 自作ライブライのimport
5	import MCP23017_Class
6	
7	if __name__ == '__main__':
8	mcp = MCP23017_Class.MCP23017()
9	try:
10	#GPIOA[1] Read
11	data = mcp.input(1)
12	#GPIOB[3] Write
13	mcp.output(3)
14	except KeyboardInterrupt:
15	pass

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang(シバン、シェバン)
- 2 行目: マジックコメント
- 5 行目: MCP23017_Class モジュールをインポート(import)

Copyright © 2024 YITJC All Rights Reserved.

注意(復習)

モジュール名(ここでは、MCP23017_Class)には、次のルールがあるので注意してください。

① 数字から始まるモジュール名は使用禁止

② Python の予約語(def, class, import など)は使用禁止

③ 特殊文字(#, $, @, !など)は使用禁止

④ アンダースコア「_」とハイフン「-」の混在は禁止

⑤ 大文字と小文字を区別する

また、インポートするモジュールは同じディレクトリ(ここでは、/home/cuu/dev)に保存してください。

別のディレクトリからモジュールをインポートする場合は、次の記述に倣ってください。

```
import sys
sys.path.append('/your/module/path')

import your_module
```

- 7 行目～15 行目: main プログラム
- 8 行目～13 行目:try ブロック

  モジュール(MCP23017_Class)で定義されているクラス(MCP23017)を読み込んで、GPIO ポートの入出力制御を行う。
- 8 行目: クラス生成

  5 行目でインポート(import)したモジュールからクラスを生成し、変数「mcp」に代入しています。

  クラスを生成する場合は、「**モジュール名.クラス名**」としてください。
- 11 行目: ポート番号 1(GPIOA[1])の状態を読み取る

  GPIOA[1]は、VPE Board 上の DIP スイッチ 2 番の状態(信号)が入力されています。
- 13 行目: ポート番号 3(GPIOB[3])の Write 処理(2 秒間 High 出力)を実行
- 14 行目～15 行目:except ブロック

  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文(何も処理しない)を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

第 5 章 データ取得～カメラ・画像処理～

これまでセンサや生産設備からのデータ取得について学習してきました。ここでは、センサや生産設備から得られるデータと比べて情報量が豊富な「画像データ」の取得について学習します。

Raspberry Pi は USB ポートを備えているため、比較的安価な USB カメラ（Web カメラ）を接続して画像データを取得できます。Raspberry Pi が撮影した静止画や動画は、センサや生産設備の稼働信号だけでは実現できない機能を提供できます。例えば、生産設備の稼働監視自動化システムや異常発生前後の様子を動画に記録するドライブレコーダーなど、撮影した画像データを活用して生産性向上につながるシステムを構築できます。

さらに、取得した画像は AI（人工知能：Artificial Intelligence）技術を活用して、製品の欠陥（不良）検出や生産設備の予知保全などに応用できます。

Raspberry Pi を使用して静止画像を取得する方法を 2 つ紹介します。

- Fswebcam

  Fswebcam はコマンド入力だけで静止画像を取得できるコマンドラインツールです。コマンド1行で、画像のファイル形式、サイズ、露光時間などを指定して静止画像を取得できます。しかし、動画の撮影には対応していません。

- OpenCV

  OpenCV（Open Source Computer Vision Library）は静止画、動画の撮影・保存だけでなく、Python を使用して高度な画像処理を実行できます。例えば、画像のフィルタリング、エッジ検出、特徴点抽出、物体検出、顔検出などの処理を行うことが可能です。さらに、OpenCV は機械学習やディープラーニングなどの AI 技術との連携や応用に適しています。例えば、OpenCV と組み合わせて機械学習モデルを使用することで、画像の分類や物体検出、顔認識などのタスクを自動化できます。また、OpenCV を使用して収集された画像データから画像の特徴量を抽出し、機械学習モデルに入力することで、画像分類や物体検出などの精度向上に貢献できます。

5.1 画像データ保存先

Raspberry Pi に画像データを保存するディレクトリを作成します。

mkdir コマンドは、「Make Directory」の略称で、ディレクトリ（フォルダ）を作成するという意味です。

ホームディレクトリから次のコマンドを実行します。

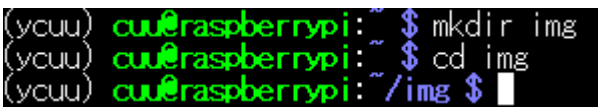
```
(ycuu) cuu@raspberrypi:~ $ mkdir img
```

続いて、img ディレクトリに移動します。

cd コマンドは、「Change Directory」の略称で、カレントディレクトリを移動するという意味です。

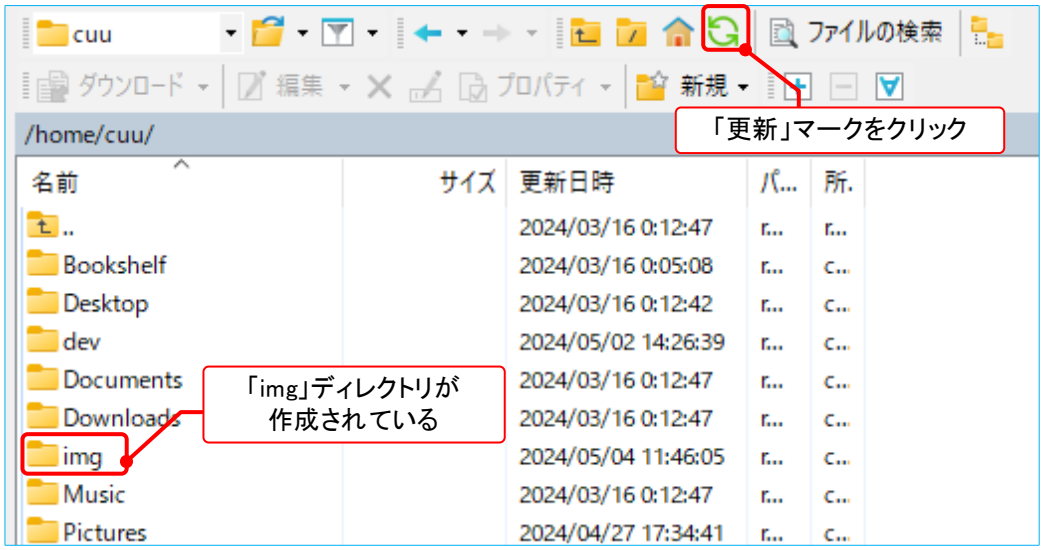
```
(ycuu) cuu@raspberrypi:~ $ cd img
(ycuu) cuu@raspberrypi:~/img $
```

ディレクトリを移動すると、カレントディレクトリが「~/img」に変化したことがわかります。



Copyright © 2024 YITJC All Rights Reserved.

WinSCP の画面上で確認してみます。



画像データ（静止画、動画など）は、「img」ディレクトリに保存します。

Copyright © 2024 YITJC All Rights Reserved.

5.2 静止画像取得～fswebcam～

fswebcam を使用して静止画像を取得します。

手順①～fswebcam インストール～

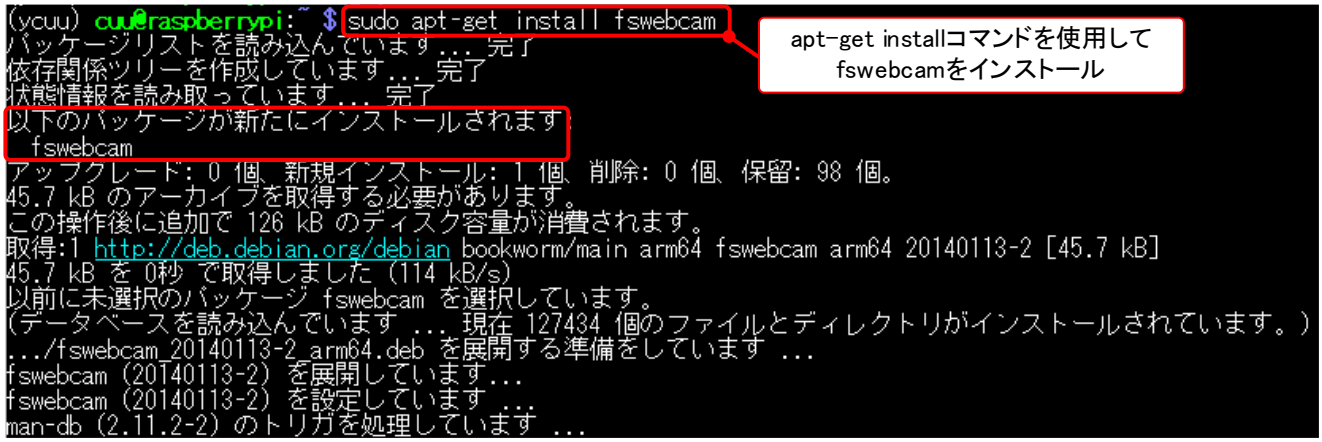
静止画像を取得するためのソフトウェア「fswebcam」をインストールします。

まず、利用可能なソフトウェアのパッケージリストを最新に更新します。

(ycuu) cuu@raspberrypi: ~/img $ sudo apt-get update

続いて、fswebcam をインストールします。

(ycuu) cuu@raspberrypi: ~/img $ sudo apt-get install fswebcam



手順②～Web カメラ接続～

Raspberry Pi の USB ポートに Web カメラを接続し、Raspberry Pi の電源を入れてください。

なお、USB は活線挿抜に対応しているため、通電中も Web カメラの抜き挿しが可能です。

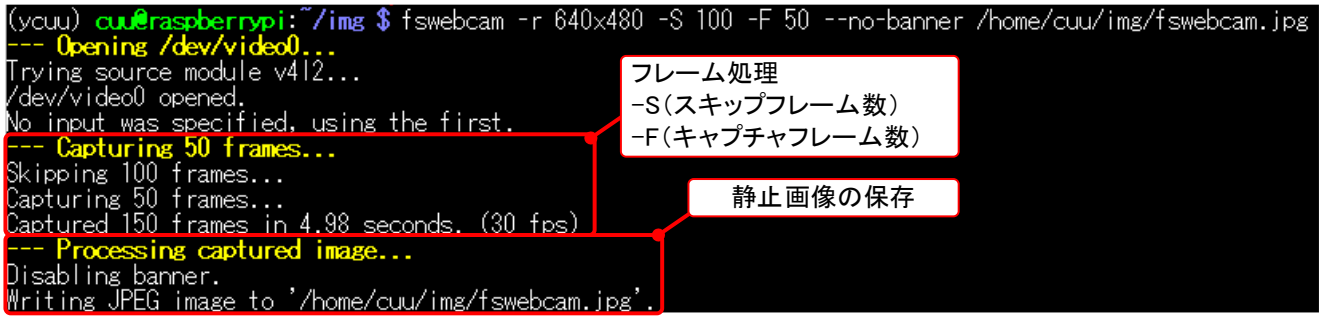
Copyright © 2024 YITJC All Rights Reserved.

手順③～静止画像取得～

次のコマンドを実行し、静止画像を取得してください。

注意 コマンドを途中で改行せずに、1行のコマンドとして実行してください。

(ycuu) cuu@raspberrypi:~/img $ fswebcam -r 640x480 -S 100 -F 50 --no-banner /home/cuu/img/fswebcam.jpg



参考～fswebcam コマンド～

実行した fswebcam コマンドについて説明します。

fswebcam -r 640x480 -S 100 -F 50 --no-banner /home/cuu/img/fswebcam.jpg

- 画像サイズ

  -r 640x480

  -r のあとに画像ファイルのサイズを指定します。

  撮影できる画像サイズを確認する場合は、次のコマンドを実行します。

  (ycuu) cuu@raspberrypi:~/img $ lsusb -v |grep -1 Width

- スキップフレーム数

  -S 100

  -S のあとにスキップするフレームの数を指定します。

  スキップフレームを指定する例として、カメラが起動してからピントが合うまでに時間を要する場合は、スキップフレーム数を調整してピントの合った画像が得られるようにします。

- キャプチャ（撮影）フレーム数

  -F 50

  -F のあとにキャプチャ（撮影）するフレームの数を指定します。

  撮影対象が動いている場合、キャプチャするフレーム数が多いほど画像がぶれやすくなります。

  F 値を変化させて、人差し指を左右に動かしながら HAT Board を撮影した画像を示します。

Copyright © 2024 YITJC All Rights Reserved.

- バーナー表示

  --no-banner

  デフォルト設定では画像ファイルの下部にバーナー（日付・時刻）が表示されます。

  Raspberry Pi に日本語フォントがインストールされていない環境では、バーナーが文字化けしてしまうため、ここでは非表示（--no-banner）に設定しています。

- 画像ファイル形式

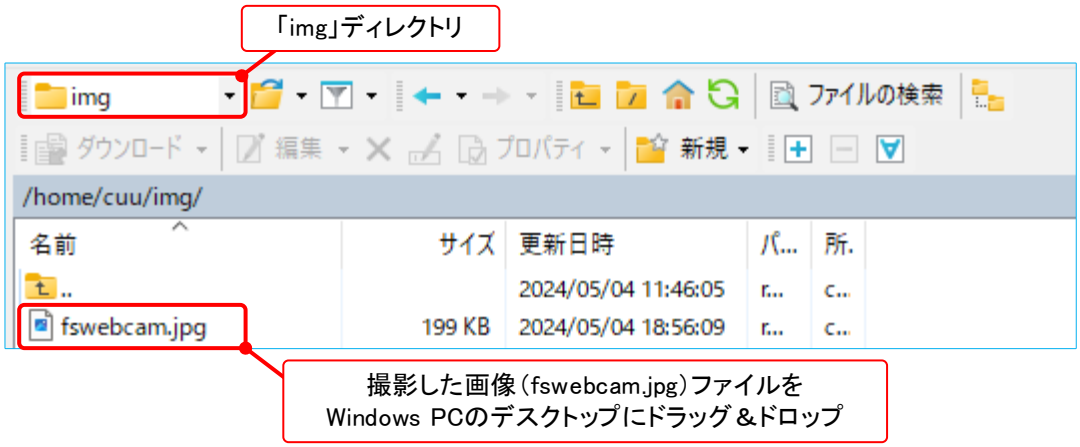
  /home/cuu/img/fswebcam.jpg

  画像ファイルの保存先とファイル形式を指定します。

  ファイル形式は、jpeg と png です。

手順④～静止画像の確認～

WinSCP の画面で「img」ディレクトリを表示し、撮影した画像ファイル「fswebcam.jp」を Windows PC のデスクトップにドラッグ&ドロップしてください。そして、Windows PC 上で撮影した画像ファイルを開いてください。



注意 WinSCP は Raspberry Pi とリモート接続（ssh 接続）してファイルの転送や管理を行うソフトウェアです。そのため、画像ファイルを直接開くための機能は備えていません。

続いて、fswebcam コマンドを Python プログラムの中で実行する方法を説明します。

手順⑤～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

15_fswebcam.py

手順⑥～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:$ cd dev
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 15_fswebcam.py
```

Copyright © 2024 YITJC All Rights Reserved.

手順⑦～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./15_fswebcam.py

次の画面（実行結果）が表示され、画像ファイル「fswebcam_python.jpg」が保存されたことを確認してください。

```
(ycuu) cuu@raspberrypi: ~/dev $ ./15_fswebcam.py
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
--- Capturing 50 frames...
Skipping 100 frames...
Capturing 50 frames...
Captured 150 frames in 4.96 seconds. (30 fps)
--- Processing captured image...
Disabling banner.
Writing JPEG image to '/home/cuu/img/fswebcam_python.jpg'.
```

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import subprocess
5	
6	def photo():
7	cmd = 'fswebcam -r 640x480 -S 100 -F 50 --no-banner /home/cuu/img/fswebcam_python.jpg'
8	subprocess.call(cmd, shell=True)
9	
10	if __name__ == '__main__':
11	try:
12	photo()
13	except KeyboardInterrupt:
14	pass

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Python 標準パッケージ「subprocess」ライブラリをインポート（import）

  ～補足～

  Python 標準パッケージであるため、外部ライブラリのように pip3 コマンドであらかじめライブラリをインストールする必要はありません。

Copyright © 2024 YITJC All Rights Reserved.

- 6 行目～8 行目: 静止画撮影プログラム photo()関数
- 7 行目: Python プログラム内で実行するコマンドを定義して変数 cmd に代入
- 8 行目: subprocess ライブラリを使用して 7 行目で定義したコマンド「cmd」を実行
- 10 行目～14 行目: メイン関数
- 11 行目～12 行目 :try ブロック
  静止画撮影プログラム photo()関数を実行
- 13 行目～14 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

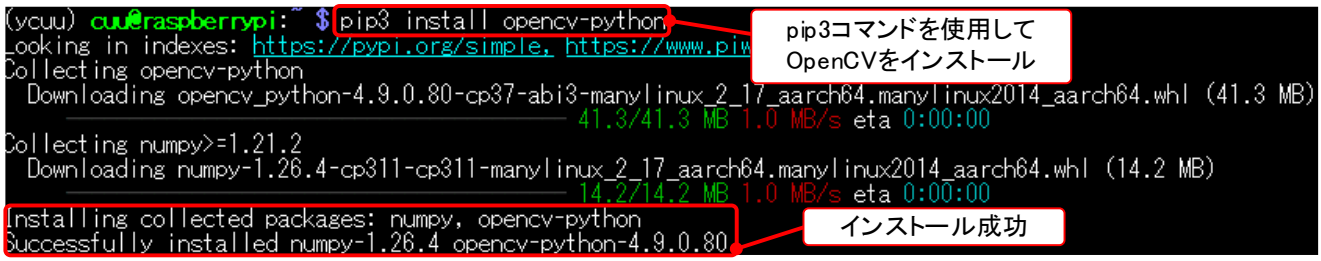
5.3 静止画像取得～OpenCV～

OpenCV を使用して静止画像を取得します。

手順①～OpenCV インストール～

静止画や動画の撮影・保存や画像処理を行うためのライブラリ「OpenCV」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install opencv-python



～補足～

OpenCV と同時に「Numpy（ナムパイ）」がインストールされます。Numpy は Python の数値計算ライブラリです。OpenCV は画像処理で数値処理や配列操作を行う際に Numpy を使用するため、同時にインストールされます。

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

16_OpenCV.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

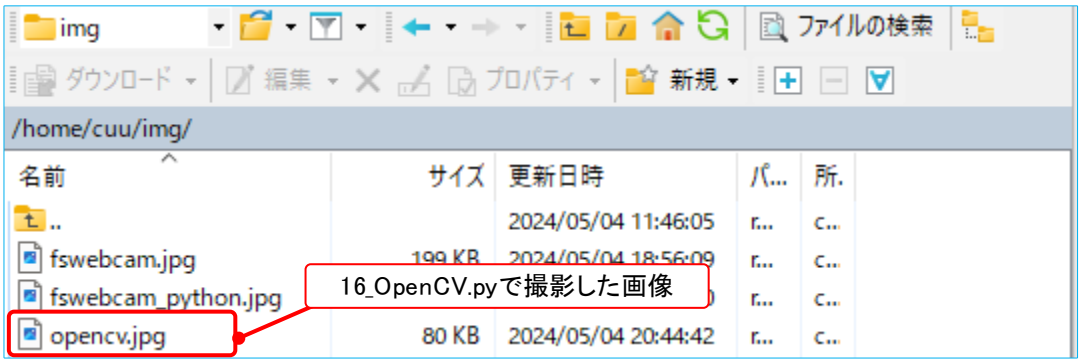
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 16_OpenCV.py

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./16_OpenCV.py

WinSCP の画面で「img」ディレクトリに画像ファイル「opencv.jpg」が保存されたことを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import cv2
5	
6	def pic():
7	deviceid=0
8	capture = cv2.VideoCapture(deviceid)
9	#解像度設定
10	capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
11	capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
12	#画像キャプチャ
13	ret, frame = capture.read()
14	#画像保存
15	cv2.imwrite('/home/cuu/img/opencv.jpg', frame)
16	
17	if __name__ == '__main__':
18	try:
19	pic()
20	except KeyboardInterrupt:
21	pass

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: 手順①でインストールした「OpenCV」ライブラリをインポート（import）
- 6 行目～15 行目: 静止画撮影プログラム pic()関数
- 7 行目: デバイス ID 番号を指定

  ここでは Raspberry Pi に接続している Web カメラのデバイス番号を指定します。

  Web カメラを 1 台だけ接続する場合は「0」番を指定しますが、Web カメラを複数台接続した場合に、各カメラのデバイス番号がわからない場合は、次のコマンドを実行すると番号が表示されます。

  (ycuu) cuu@raspberrypi: ~/dev $ ls -la /dev/video*

```
(ycuu) cuu@raspberrypi:~/dev $ ls -la /dev/video*
crw-rw----+ 1 root video 81, 14  5月  4 12:20 /dev/video0
crw-rw----+ 1 root video 81, 15  5月  4 12:20 /dev/video1
```

  ～補足～

  デバイス ID(/dev/videoX)の番号（X の部分）は、デバイス（Web カメラ）が OS に認識された順番で割り振られます。また、1 台のデバイスに対して 2 つのデバイスファイル（/dev/video0 と/dev/video1）が作成されますが、プログラムで指定できる番号は 1 つだけで（ここでは 0 番）、もう片方を指定するとエラーが発生します。

Copyright © 2024 YITJC All Rights Reserved.

- 8 行目: 4 行目でインポートした cv2 ライブラリから VideoCapture クラスのインスタンスを生成
- 9 行目～11 行目: 解像度を指定
- 13 行目: VideoCapture オブジェクトを使用してフレームを読み取り
  このメソッドは 2 つの値（ret と frame）を返します。
  「ret」は、読み取り操作の結果を示すブール値です（成功した場合は True、失敗した場合は False）。
  「frame」は、読み取られたフレームです。
- 15 行目: 画像データ保存
  13 行目で取得したフレーム（frame）を指定ディレクトリに保存
- 17 行目～21 行目: メイン関数
- 18 行目～19 行目 :try ブロック
  静止画撮影プログラム pic()関数を実行
- 20 行目～21 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

5.4 動画取得～OpenCV～

OpenCV を使用して動画を撮影・保存します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

17_Movie.py

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

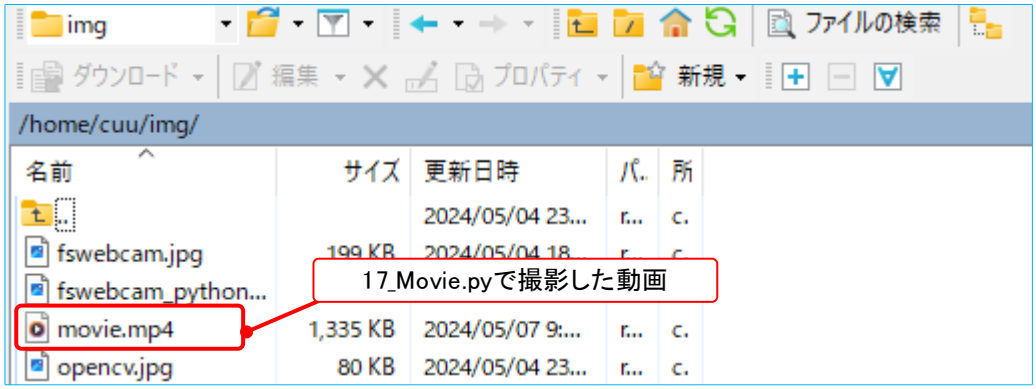
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 17_Movie.py

手順③～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./17_Movie.py

WinSCP の画面で「img」ディレクトリに画像ファイル「movie.mp4」が保存されたことを確認してください。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	#OpenCV
5	import cv2
6	import time
7	from datetime import datetime
8	#Webカメラ fps(frame per second)
9	fps = 30

Copyright © 2024 YITJC All Rights Reserved.

```
10  #撮影時間(sec)
11  m_time = 10
12
13  def movie():
14      deviceid=0
15      capture = cv2.VideoCapture(deviceid)
16      #Size指定
17      width = 640
18      height = 480
19      #ファイル形式: mp4
20      fmt = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
21      writer = cv2.VideoWriter('/home/cuu/img/movie.mp4', fmt ,fps, (width, height))
22
23      for num in range(fps*m_time):
24          #frame撮影開始時刻
25          start_time = datetime.now()
26          #画像キャプチャ
27          ret, frame = capture.read()
28          #フレーム書き込み
29          writer.write(frame)
30          #frame撮影終了時刻
31          end_time = datetime.now()
32          #frame間時間計測
33          diff = end_time - start_time
34          print (diff)
35          #diff = fps(frame per second)
36          #frame数同じ(画像ファイルサイズ同じ)で長時間撮影したい場合はsleep時間を追加
37          #time.sleep(1.0)
38          print(num)
39      else:
40          #ビデオファイルクローズ
41          writer.release()
42          #VideoCaptureオブジェクトが使用しているリソースを解放
43          capture.release()
44
45  if __name__ == '__main__':
46      try:
47          movie()
48      except KeyboardInterrupt:
```

Copyright © 2024 YITJC All Rights Reserved.

49	pass

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 5 行目: OpenCV ライブラリをインポート（import）
- 6 行目: time ライブラリをインポート（import）
- 7 行目: datetime ライブラリから datetime クラスをインポート（import）しています
  現在の日付と時刻は、datetime.now()メソッドを使用して取得できます。
  ～参考～
  Import datetime と記述して、datetime ライブラリをインポート（import）することも可能ですが、
  この場合、datetime クラスを使用して現在の日付と時刻を取得する場合は、
  datetime.datetime.now()と記述しなければなりません。
- 9 行目: Web カメラの fps（frame per second）を指定
  fps は、1 秒間あたりに撮影（キャプチャ）できるフレーム数を表しています。使用する Web カメラの仕様を確認して fps の値を設定してください。
- 11 行目: 撮影時間（秒）を指定
- 13 行目～43 行目: 動画撮影プログラム movie()関数
- 14 行目: デバイス ID 番号を指定
- 15 行目: cv2 ライブラリから VideoCapture クラスのインスタンスを生成
- 17 行目～18 行目: 解像度を指定
- 20 行目: 動画ファイル形式を mp4 に指定
- 21 行目: ビデオファイルに書き込むための VideoWriter オブジェクトを設定
  このオブジェクトの引数には、次の情報を指定しています。
  動画ファイルの保存先: /home/cuu/img/movie.mp4
  動画ファイルの形式: mp4
  動画ファイルの fps
  動画ファイルの解像度: 640 x 480
- 23 行目～38 行目: 繰り返し（for 文）処理
- 23 行目: for 文で繰り返す範囲（回数）を指定
  fps * m_time を繰り返し回数と定義しているため、（fps = 30、m_time = 10sec）
  この for 文は、num の値が 0 から 299 まで（300 回）繰り返し処理を行います。
- 25 行目: datetime.now()メソッドを使用してフレームの撮影開始時刻を取得
- 27 行目: VideoCapture オブジェクトを使用してフレームを読み取り
- 29 行目: 読み取ったフレーム（frame）を書き込み
- 31 行目: datetime.now()メソッドを使用してフレームの撮影終了時刻を取得
- 33 行目: 撮影開始時刻と撮影終了時刻から 1 フレームあたりの撮影時間（diff）を算出
- 34 行目: 1 フレームあたりの撮影時間（diff）を print 表示

Copyright © 2024 YITJC All Rights Reserved.

- 39 行目～43 行目: else ブロック
- 41 行目: cv2.VideoWriter オブジェクトを解放しています（ビデオファイルのクローズ処理を実施しています）。
- 42 行目: cv2.VideoCapture オブジェクトを解放しています（カメラリソースの解放処理を実施しています）。
- 45 行目～49 行目: メイン関数
- 46 行目～47 行目:try ブロック
  動画撮影プログラム movie()関数を実行
- 48 行目～49 行目:except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

参考～Python の繰り返し（for 文）処理～

ここでは、Python の for 文を使用した繰り返し処理の構文を紹介します。

- range で「終了値」のみ指定する場合
  「0」～「終了値-1」まで繰り返し処理が実施されます。

～例 1～

```
for num in range(5):
    print(num)
```

～例 1・実行結果～

```
0
1
2
3
4
```

- range で「開始値」、「終了値」を指定する場合
  「開始値」～「終了値-1」まで繰り返し処理が実施されます。

～例 2～

```
for num in range(3,5):
    print(num)
```

～例 2・実行結果～

```
3
4
```

- range で「開始値」、「終了値」、「インクリメント（増分）値」を指定する場合
  「開始値」～「終了値-1」まで、「増分値」ごとに繰り返し処理が実施されます。
  例 3 の場合は、1 から 10 まで 2 ずつ増分させた結果を print 表示しています。
  デクリメント（減分）する場合は、マイナス（-）を指定します。

～例 3～

```
for num in range(1,11,2):
    print(num)
```

～例 3・実行結果～

```
1
3
5
7
9
```

Copyright © 2024 YITJC All Rights Reserved.

- for 文と else ブロックを組み合わせた使い方
  else ブロックの処理は、for 文の処理がすべて終了した後で実行されます。

～例 4～
```
for num in range(5):
    print(num)
else:
    print(“finish”)
```

～例 4・実行結果～
```
0
1
2
3
4
finish
```

- for 文の繰り返し処理から抜ける方法
  break を使用すると for 文の処理から抜けることができます。
  例 5 の場合は、num の値が「3」になったタイミングで for 文から抜けています。

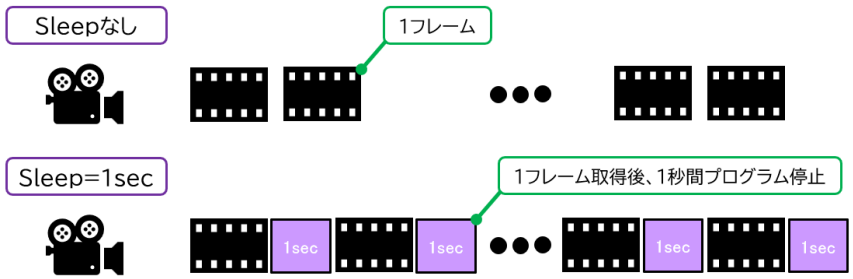
～例 5～
```
for num in range(5):
    print(num)
    if num == 3 :
        break
```

～例 5・実行結果～
```
0
1
2
3
```

参考～動画プログラムの sleep 時間～

動画撮影プログラム 37 行目でコメントアウト(#)している sleep 処理について説明します。

sleep 時間(1 秒と仮定)を設定した場合を例に挙げて説明します。sleep 時間を設定した場合、1 フレーム取得後に毎回 1 秒回プログラムの停止時間(sleep 時間)が入ります。そのため、実際の撮影時間は sleep 時間の設定がない場合と比べて、sleep 時間(秒)×撮影フレーム回数分長くなります。しかし、撮影フレーム数は同じであるため、作成・保存される動画ファイルのサイズはほぼ同じになります。



Sleep 時間	撮影フレーム数	動画ファイルの撮影時間	実際の撮影時間
なし	300	10sec	約 10sec
あり(1 秒)	300	10sec	約 300sec

sleep 時間は、撮影対象の変化に応じて設定する必要があります。例えば、撮影対象の変化が 5 秒ごとのフレーム撮影で十分な場合は、sleep 処理を追加することで、動画ファイルのサイズ(容量)を抑えて長時間の撮影ができます。

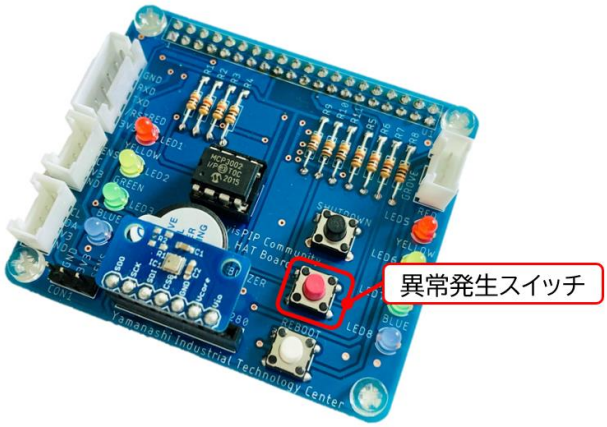
Copyright © 2024 YITJC All Rights Reserved.

5.5 ドライブレコーダー

生産現場において、異常発生時の状況を正確に把握することは、異常の原因分析及び再発防止のために重要です。ここでは、動画撮影プログラムを応用したドライブレコーダー機能を紹介します。

[プログラム内容]

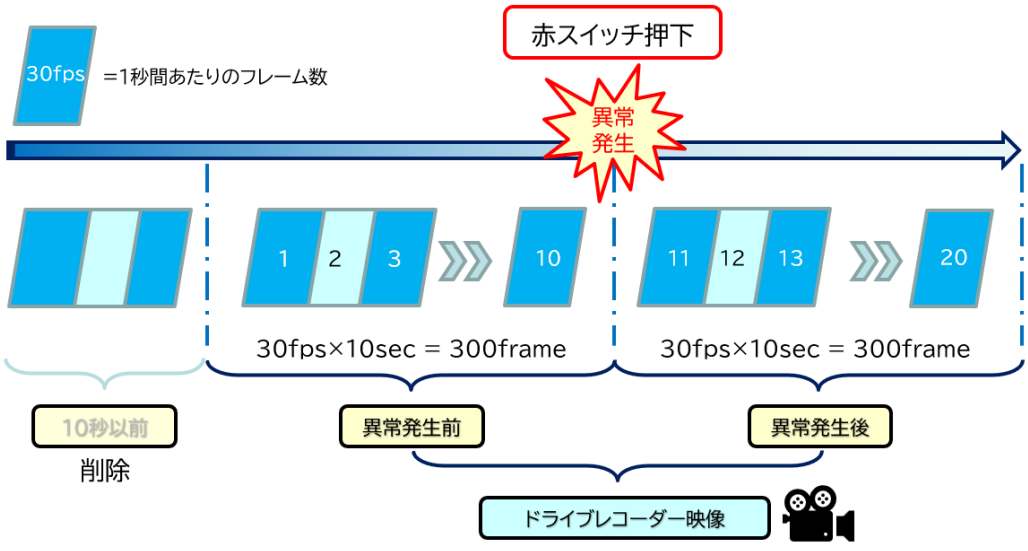
HAT Board 上の赤色 SW を「異常発生」スイッチと仮定します。Web カメラを Raspberry Pi に接続した状態でプログラムの動作確認を行います。



異常発生スイッチが押されると、異常発生前後 20 秒の動画を作成・保存します。

プログラムの流れを説明すると、次のとおりです。

① 常に 10 秒間の動画を撮影（10 秒を超えた場合は、10 秒以前のフレームを自動削除）
② Raspberry Pi の GPIO25（赤 SW）押下で異常発生
③ 異常発生と同時に異常発生 LED 点灯（赤）
④ 異常発生後、10 秒間の動画を撮影
⑤ 異常発生前（10 秒）と異常発生後（10 秒）の動画を 1 つのファイルにまとめて保存
⑥ 動画ファイルの保存完了で LED 点灯（青）
⑦ 動画ファイルを保存した後、動画撮影を開始する（※プログラムを強制終了しない限り、撮影を継続する）



Copyright © 2024 YITJC All Rights Reserved.

手順①～RPi.GPIO インストール～

GPIO ポートを処理するためのライブラリ「RPi.GPIO」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install RPi.GPIO

～補足～

「RPi.GPIO」ライブラリは第 2 章で使用しています。しかし、第 2 章では仮想環境「ycuu」を構築していなかったため、システムのデフォルト環境で「RPi.GPIO」ライブラリを使用していました。そのため、仮想環境には「RPi.GPIO」ライブラリがまだインストールされていないため、このタイミングでインストールしています。

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

18_Monitoring.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:$ cd dev

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 18_Monitoring.py

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./18_Monitoring.py

次の手順で動作確認してください。

(動作確認手順)

① プログラム実行

② 10 秒以上経過したら、HAT Board 上の赤色 SW を押下
(10 秒経過前でも問題なく動作しますが、その場合異常発生前の動画が 10 秒より短くなります。)
※このとき、赤色 LED が 1 秒間点灯することを確認してください。

③ 赤色 SW 押下から 10 秒後、青色 LED が 2 秒間点灯することを確認

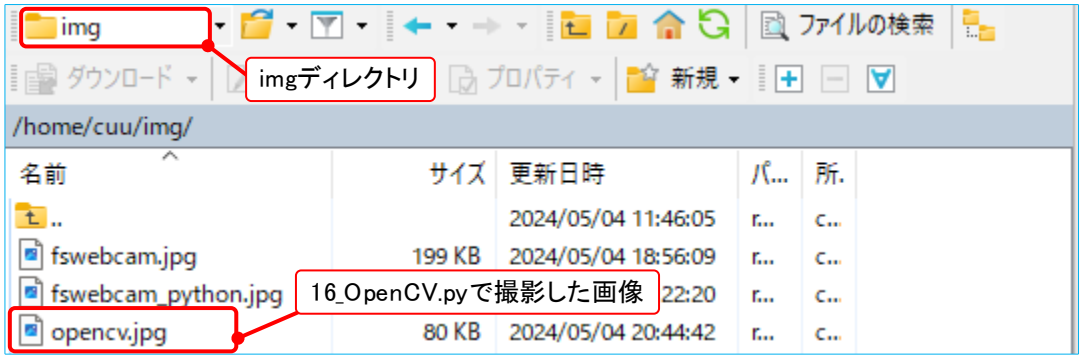
④ 青色 LED 点灯を確認した後、プログラムを強制終了(「Ctrl」+「c」を押下)

⑤ WinSCP の画面を開き、異常発生前後の動画ファイル(error_video.mp4)が保存されていることを確認してください。

⑥ 動画ファイルを再生し、異常発生時の様子(赤色 SW 押下)が撮影されていることを確認してください。

Copyright © 2024 YITJC All Rights Reserved.

WinSCP の画面



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import RPi.GPIO as GPIO
5	import cv2
6	import time
7	
8	#GPIO,撮影パラメータ設定
9	ERROR_SIG = 25　#異常発生スイッチ(赤SW)
10	ALARM_LED = 4　 #異常発生LED(赤)
11	END_LED = 22　　#動画ファイル保存完了LED(青)
12	fps = 30　　#Webカメラ fps(frame per second)
13	m_time = 10　　#撮影時間(sec)
14	
15	#GPIO初期設定
16	def setup_gpio():
17	GPIO.setmode(GPIO.BCM)
18	GPIO.setup(ERROR_SIG, GPIO.IN)
19	GPIO.setup(END_LED, GPIO.OUT, initial=GPIO.LOW)
20	GPIO.setup(ALARM_LED, GPIO.OUT, initial=GPIO.LOW)
21	
22	#ドライブレコーダー関数
23	def monitoring():
24	setup_gpio()
25	deviceid=0
26	global capture
27	capture = cv2.VideoCapture(deviceid)　　#Camera起動

Copyright © 2024 YITJC All Rights Reserved.

```
28      capture.set(cv2.CAP_PROP_AUTOFOCUS, 1)          #Auto Focus ON
29      #Size指定
30      width = 640
31      height = 480
32      fmt = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
33      frame_buffer = []           #動画フレーム保存用バッファ
34      record_flag = False         #録画中Flag
35
36      while True:
37          ret, frame = capture.read()         #Camaraからフレーム取得
38          if not ret:                             #フレームが取得できない場合はフレームの取得を継続
39              continue
40          frame_buffer.append(frame)          #フレームをバッファに保存
41
42          #異常検出(ERROR_SIG受信)前の10秒を録画
43          #バッファが10秒分のフレームを超えたら、古いフレーム(先頭:frame1)から順に削除
44          while len(frame_buffer) > m_time * fps:
45              frame_buffer.pop(0)
46
47          #異常検出 and 録画中でなければ
48          if GPIO.input(ERROR_SIG) == GPIO.LOW and not record_flag:
49              #異常発生LED(赤)
50              GPIO.output(ALARM_LED, GPIO.HIGH)
51              time.sleep(1)
52              GPIO.output(ALARM_LED, GPIO.LOW)
53              record_flag = True    #録画中フラグOn
54
55          if record_flag:                 #録画中フラグOnの処理
56              #異常検出(ERROR_SIG受信)後の10秒を録画
57              for i in range(m_time * fps):
58                  ret, frame = capture.read()
59                  #異常発生前の動画に異常発生後の動画(10sec)を追加
60                  frame_buffer.append(frame)
61              #異常発生前後の動画を1つのファイルで保存
62              writer = cv2.VideoWriter('/home/cuu/img/error_video.mp4', fmt ,fps, (width, height))
63              #freme_bufferにある異常発生前後の動画ファイル(600フレーム)を1フレームずつsaved_frameに書き込む
64              for saved_frame in frame_buffer:
65                  writer.write(saved_frame)
66              writer.release()
```

Copyright © 2024 YITJC All Rights Reserved.

```
67
68                  #撮影終了LED(青)
69                  GPIO.output(END_LED, GPIO.HIGH)
70                  time.sleep(2)
71                  GPIO.output(END_LED, GPIO.LOW)
72
73                  record_flag = False        #録画中フラグOff
74                  frame_buffer = []          #動画フレーム保存用バッファ初期化
75
76  if __name__ == "__main__":
77      try:
78          monitoring()
79      except KeyboardInterrupt:
80          pass
81      finally:
82          capture.release()
83          GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang(シバン、シェバン)
- 2 行目: マジックコメント
- 4 行目: 手順①でインストールした「RPi.GPIO」ライブラリをインポート(import)
- 5 行目: OpenCV ライブラリをインポート(import)
- 6 行目: time ライブラリをインポート(import)
- 9 行目～11 行目: GPIO ピン番号を定義
  異常発生スイッチ(ERROR_SIG=25)
  異常発生 LED 赤(ALARM_LED=4)
  動画ファイル保存完了 LED 青(END_LED=22)
- 12 行目: Web カメラの fps(frame per second)を指定
  fps は、1 秒間あたりに撮影(キャプチャ)できるフレーム数を表しています。使用する Web カメラの仕様を確認して fps の値を設定してください。
- 13 行目: 撮影時間(秒)を指定
  異常発生前及び異常発生後に撮影する時間を指定します。
- 16 行目～20 行目: GPIO 端子初期設定関数
  スイッチや LED などこのプログラムで使用する GPIO 端子の初期設定を行っています。
- 23 行目～74 行目: ドライブレコーダー撮影プログラム monitoring()関数
- 24 行目: GPIO 端子初期設定 setup_gpio()関数を実行
- 25 行目: Web カメラのデバイス番号設定

Copyright © 2024 YITJC All Rights Reserved.

- 26 行目: global 変数「capture」を定義

  ～補足～

  「ローカル変数」は、定義した関数の中でのみ使用できます。

  「グローバル変数」は、関数の外でも使用できる変数です。

  このプログラムでは、27 行目で定義する capture（VideoCapture クラスのインスタンス）を 82 行目のメイン関数で使用しています。そのため、「capture」をグローバル変数で定義する必要があります。
- 27 行目: VideoCapture クラスのインスタンスを生成
- 28 行目: カメラのオートフォーカスを ON に設定

  ～参考～

  OpenCV ではカメラのパラメータを設定できるコマンドが用意されています。以下に例を示します。

分類	パラメータ	内容
解像度	CAP_PROP_FRAME_WIDTH	取得画像の解像度（幅）
	CAP_PROP_FRAME_HEIGHT	取得画像の解像度（高さ）
撮影	CAP_PROP_BRIGHTNESS	明るさ
	CAP_PROP_CONTRAST	コントラスト
	CAP_PROP_GAIN	ゲイン
自動制御	CAP_PROP_AUTO_EXPOSURE	露出制御
	CAP_PROP_AUTOFOCUS	フォーカス制御

- 30 行目～31 行目: 解像度を指定
- 32 行目: 動画ファイル形式を mp4 に指定
- 33 行目: 動画フレーム保存用バッファ（frame_buffer）を用意
- 34 行目: 録画中フラグ（record_flag）を用意し、初期値「False」を設定
- 36 行目～74 行目: ドライブレコーダー撮影の無限ループ処理
- 37 行目: VideoCapture オブジェクトを使用してフレームを読み取り
- 38 行目～39 行目: 37 行目の戻り値「ret」が False の場合は、フレームの取得を継続（continue）

  「ret」は、読み取り操作の結果を示すブール値です（成功した場合は True、失敗した場合は False）

  continue 文を使用した場合、それ以降の処理は行われず、while 文の先頭（36 行目）に戻ります。

Copyright © 2024 YITJC All Rights Reserved.

- Continue 文を使用した例

  例の場合は、num の値が「3」になったタイミングで、continue 以降の処理は行われずに先頭に戻ります。

～例～

```
num = 1                 #num の初期値 1 を設定
while num <= 5:
    if num == 3:        #continue の条件
        num += 1        #num の値に 1 加算
        continue
    print(num)          #num の値を print 表示
    num += 1            #num の値に 1 加算
```

～例・実行結果～

```
1
2
4
5
```

・ 40 行目: 読み取ったフレームをバッファ（frame_buffer）に保存

～異常検出前の動画撮影～

・ 44 行目～45 行目: 異常検出前の 10 秒動画

m_time * fps を取得するフレーム数の上限と定義しているため（fps = 30、m_time = 10sec）、10 秒を超えた場合は、10 秒以前のフレームを順次削除する。

・ 48 行目～53 行目: 異常検出時の GPIO 処理

異常検出（赤 SW 押下）かつ録画中でなければ（record_flag==false）、次の処理を行う。

〇異常発生 LED（赤）を 1 秒間点灯

〇録画中フラグ（record_flag）を録画中（True）に設定

・ 55 行目～66 行目: 異常検出後の動画撮影、異常検出前後の動画ファイル保存

・ 55 行目: 録画中フラグ（record_flag）が録画中（True）である場合の処理

～異常検出後の動画撮影～

・ 57 行目～60 行目: 異常検出後の 10 秒動画

・ 62 行目: ビデオファイルに書き込むための VideoWriter オブジェクトを設定

このオブジェクトの引数には、次の情報を指定しています。

動画ファイルの保存先: /home/cuu/img/error_video.mp4

動画ファイルの形式: mp4

動画ファイルの fps

動画ファイルの解像度: 640 x 480

・ 64 行目～65 行目: 読み取ったフレーム（frame_buffer）を 1 フレームずつ saved_frame に保存

saved_frame のデータ（= 異常検出前後の動画）を書き込み

・ 66 行目: cv2.VideoWriter オブジェクトを解放しています（ビデオファイルのクローズ処理を実施しています）。

・ 68 行目～71 行目: 動画ファイルの保存完了（撮影終了）処理

青色 LED を 2 秒間点灯

・ 73 行目: 録画中フラグ（record_flag）を初期値「False」に設定（フラグを Off）

・ 74 行目: 読み取ったフレーム（frame_buffer）をクリア（保存用バッファを初期化）

Copyright © 2024 YITJC All Rights Reserved.

- 76 行目～83 行目: メイン関数
- 77 行目～78 行目:try ブロック
  ドライブレコーダー撮影プログラム monitoring()関数を実行
- 79 行目～80 行目:except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。
- 81 行目～83 行目:Finally ブロック
- 82 行目: cv2.VideoCapture オブジェクトを解放しています（カメラリソースの解放処理を実施しています）。
- 83 行目: GPIO ピンのシステムクリア

Copyright © 2024 YITJC All Rights Reserved.

5.6 OCR（光学的文字認識）

これまで Raspberry Pi を使用した画像データの取得方法を学習してきました。ここからは、取得した画像データに含まれる文字や数字データを取り出す方法を学習します。文字や数字データを取り出す方法として、OCR（Optical Character Recognition）技術を活用します。

OCR は、生産現場の生産性向上に大きく貢献できる技術です。画像データから必要なデータ（文字や数字データ）を認識・抽出し、テキストデータに変換することで、入力作業の自動化を実現できます。特に、数字データの取り出しが困難な古い生産設備などは、ディスプレイに表示された情報をカメラで撮影することで必要なデータを抽出できます。そのため、既存設備の更新や改造を行う必要がなく、「低コスト」かつ「短期間」で IoT 化を実現できます。

Raspberry Pi を使用してディスプレイに表示された情報を取得する方法を 2 つ紹介します。

- Web カメラ

  ディスプレイを Web カメラで撮影します。ただし、この方法でディスプレイに表示された情報を取得する場合は、次の注意が必要です。

  ① カメラの固定方法

  Raspberry Pi は、Web カメラで撮影した画像から必要な情報が表示されている範囲（座標）を指定してデータを取り出します。そのため、カメラの位置が不安定な場合は、データを取り出す範囲が変化してしまい、データの取り出しに失敗する可能性があります。

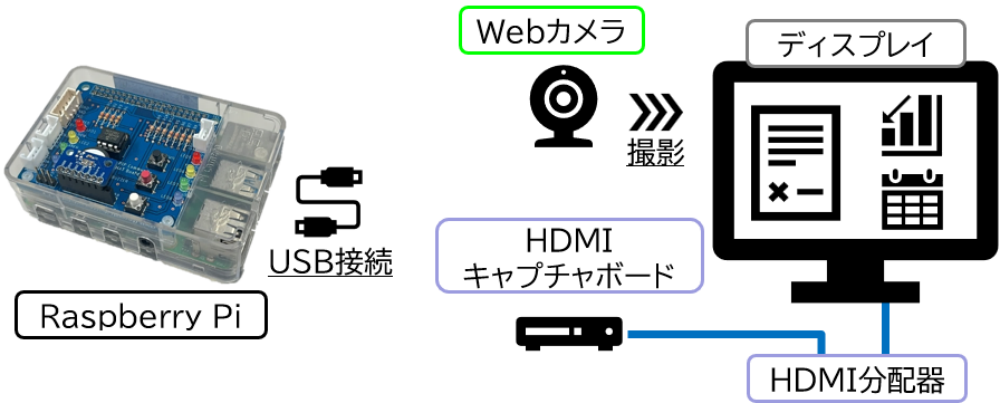
  ② 照明環境

  夜間など Web カメラ周辺の照明環境が変化した場合、Web カメラで撮影した画像から必要な情報を識別できない可能性があります。

- HDMI キャプチャボード

  HDMI 分配器を使用してディスプレイの情報を HDMI キャプチャボードに分配します。HDMI キャプチャボードは、Raspberry Pi と USB ポートで接続できます。Raspberry Pi は、HDMI キャプチャボードに入力された画像データ（ディスプレイに表示されている画像データ）を取得できます。取得するためのプログラムは、Web カメラで画像データを取得するときと同じプログラムを使用できます。

  この方法は、カメラの固定方法や照明環境などを懸念する必要がなく、安定した画像データを取得できるため、ディスプレイに表示されたデータの抽出に最適な方法です。



Copyright © 2024 YITJC All Rights Reserved.

Raspberry Pi を使用して、画像データから必要なデータ（文字や数字データ）を認識・抽出し、テキストデータに変換する方法を 2 つ紹介します。

- Tesseract（テッセラクト）

  Tesseract は、Google が開発したオープンソースの OCR エンジンです。Raspberry Pi に Tesseract をインストールして使用するため、使用する際にインターネット接続環境は必要ありません。

- Google Cloud Vision API

  Google が提供する画像認識 API（Application Programming Interface）です。Google Cloud Vision は、文字や数字の識別・抽出だけでなく、物体検出や顔の表情の自動検出も可能です。

  クラウドサービスのため、画像データを Google のサーバに送信して画像識別処理を行います。そのため、インターネット接続環境が必要であり、従量課金制となっています。

  Google Cloud Vision は、Google が提供する強力な機械学習モデルを使用しているため、文字や数字の識別精度は非常に高いです。

～参考～

Google Cloud Vision API の利用料金（2024 年 7 月 1 日時点）

特徴	1,000 ユニットあたりの料金		
	最初の 1,000 ユニット/月	1,001～5,000,000 ユニット/月	5,000,001 以上のユニット/月
ラベル検出	無料	$1.50	$1.00
テキスト検出	無料	$1.50	$0.60
ドキュメント テキスト検出	無料	$1.50	$0.60
セーフサーチ（不適切なコンテンツ）検出	無料	$1.50（ラベル検出を利用している場合は無料）	$0.60（ラベル検出を利用している場合は無料）
顔検出	無料	$1.50	$0.60
顔検出 - Celebrity Recognition	無料	$1.50	$0.60
ランドマーク検出	無料	$1.50	$0.60

Copyright © 2024 YITJC All Rights Reserved.

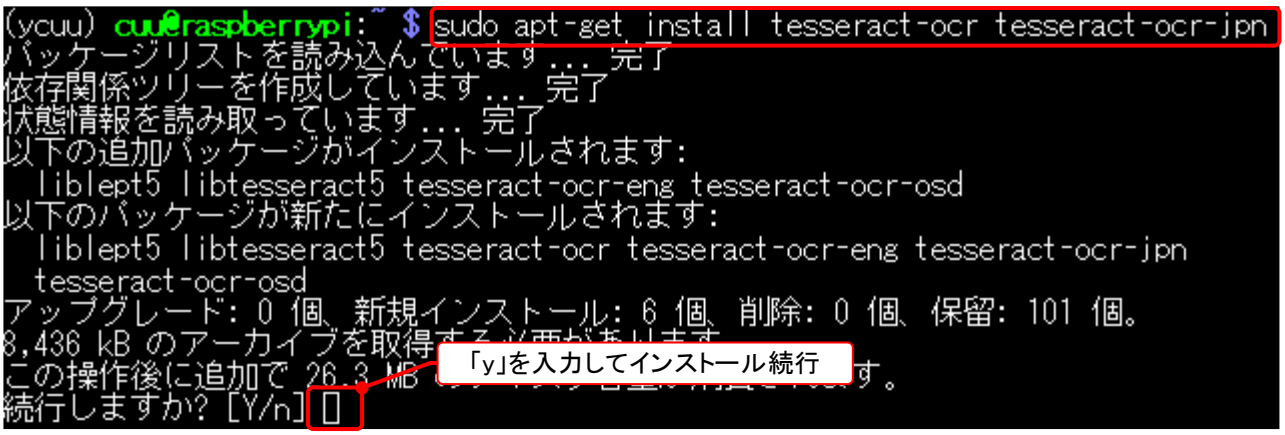
5.7 Tesseract

Tesseract を使用して画像に含まれる数字データの識別・抽出を行い、テキストデータに変換します。

手順①～tesseract インストール～

文字・数字データを識別・抽出してテキストに変換するためのソフトウェア「tesseract」をインストールします。

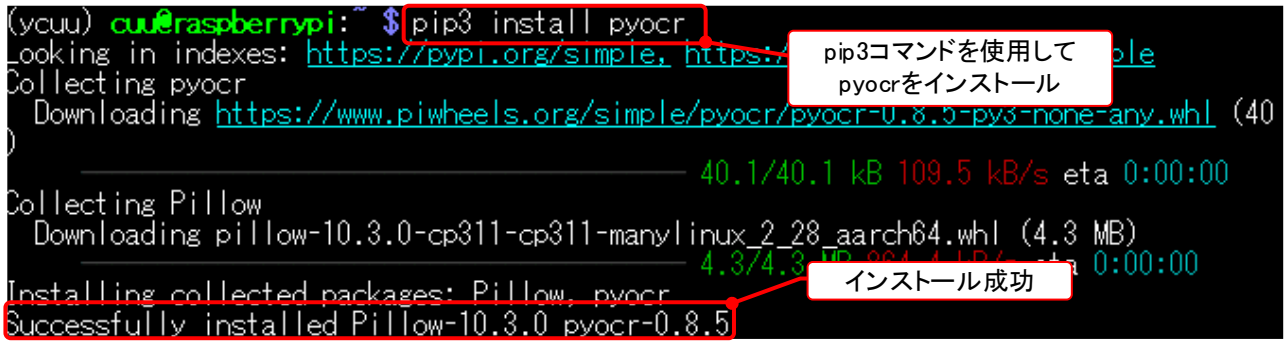
(ycuu) cuu@raspberrypi:$ sudo apt-get install tesseract-ocr tesseract-ocr-jpn



手順②～pyocr インストール～

Python から OCR エンジンを利用可能にするためのライブラリ「pyocr」をインストールします。

(ycuu) cuu@raspberrypi:$ pip3 install pyocr



～補足～

pyocr と同時に「Pillow（パイロウ）」がインストールされます。Pillow は Python の画像処理ライブラリです。Pillow は画像の回転や反転、リサイズやトリミングなどの処理を実行できます。

手順③～OCR 作業ディレクトリ作成～

Raspberry Pi にテキスト変換用プログラムの作業ディレクトリを作成します。

(ycuu) cuu@raspberrypi:~ $ cd img

「img」ディレクトリに移動して「ocr」ディレクトリを作成します。

(ycuu) cuu@raspberrypi:~/img $ mkdir ocr

(ycuu) cuu@raspberrypi: ~/img $ cd ocr

Copyright © 2024 YITJC All Rights Reserved.

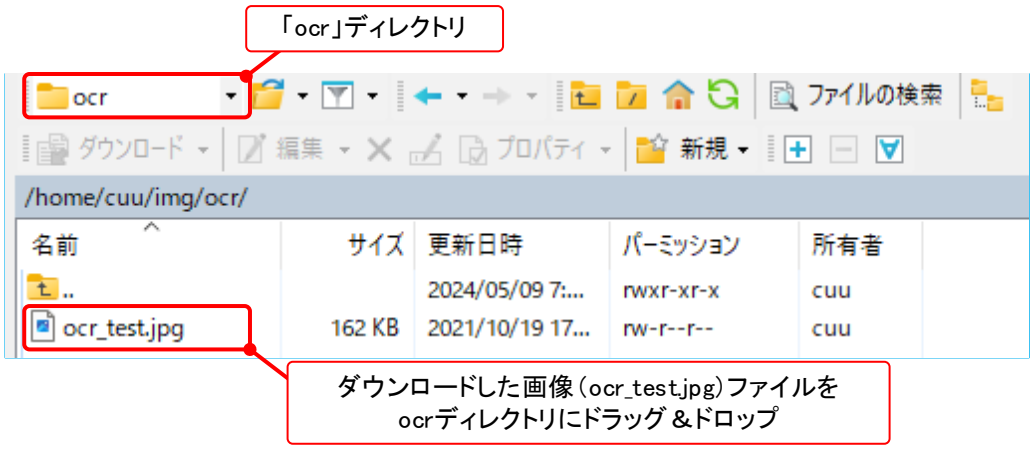
ディレクトリを移動すると、カレントディレクトリが「~/img/ocr」に変化したことがわかります。

```
(ycuu) cuu@raspberrypi:~ $ cd img
(ycuu) cuu@raspberrypi:~/img $ mkdir ocr
(ycuu) cuu@raspberrypi:~/img $ cd ocr
(ycuu) cuu@raspberrypi:~/img/ocr $
```

手順④～テキスト変換用画像の保存～

作成した「ocr」ディレクトリにテキスト変換用の画像データ「ocr_test.jpg」を保存します。

共有サイトにある画像データ「ocr_test.jpg」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「ocr」ディレクトリに保存してください。



テキスト変換に使用する画像データを紹介します。スマートフォンのディスプレイに表示されている時刻（分）をテキストデータに変換します。OCR エンジンの認識精度を確認するため、スマートフォンの画像はあえてピントをずらして撮影しています。

まず、画像データ「ocr_test.jpg」から時刻の「分」の数字を切り取り、切り取った画像「ocr_test_extract.jpg」を保存するプログラムを実行します。

手順⑤～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

［対象プログラム］

19_Extract.py

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi: ~/img/ocr $ cd /home/cuu/dev

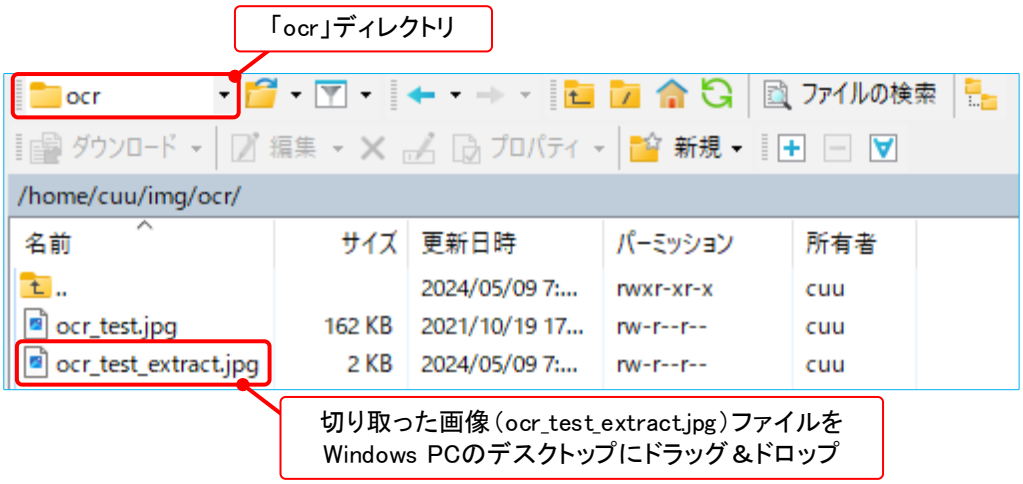
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 19_Extract.py

手順⑦～プログラム実行～

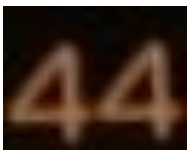
対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./19_Extract.py

WinSCP の画面で「ocr」ディレクトリに画像ファイル「ocr_test_extract.jpg」が保存されたことを確認してください。



Windows PC で切り取った画像ファイルを開いて、時刻の「分」の部分が正確に切り取られていることを確認してください。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import cv2
5	import subprocess
6	
7	#fswebcamで撮影した画像を使用する場合
8	def photo():
9	cmd = 'fswebcam -r 640x480 -S 100 -F 50 --no-banner /home/cuu/img/ocr/ocr_test.jpg'

Copyright © 2024 YITJC All Rights Reserved.

```
10        subprocess.call(cmd, shell=True)
11
12  #OpenCVで撮影した画像を使用する場合
13  def pic():
14        deviceid=0
15        capture = cv2.VideoCapture(deviceid)
16        #解像度設定
17        capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
18        capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
19        #画像キャプチャ
20        ret, frame = capture.read()
21        #画像保存
22        cv2.imwrite('/home/cuu/img/ocr/ocr_test.jpg', frame)
23
24  #画像切り取り（指定範囲切り取り）
25  def extract():
26        #  画像読み込み
27        img = cv2.imread('/home/cuu/img/ocr/ocr_test.jpg')
28
29        # img[top : bottom, left : right]
30        # top < bottom
31        #  切り出し、保存
32        img1 = img[140 : 165, 303 : 335]
33        cv2.imwrite("/home/cuu/img/ocr/ocr_test_extract.jpg", img1)
34
35  if __name__ == '__main__':
36        try:
37              #fswebcamで撮影した画像を使用する場合
38              #photo()
39              #OpenCVで撮影した画像を使用する場合
40              #pic()
41              extract()
42        except KeyboardInterrupt:
43              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: OpenCV ライブラリをインポート（import）
- 5 行目: fswebcam コマンドを実行可能とするため、subprocess ライブラリをインポート（import）

Copyright © 2024 YITJC All Rights Reserved.

- 8 行目～10 行目: fswebcam コマンドを使用した静止画撮影用プログラム photo()関数
- 13 行目～22 行目: OpenCV を使用した静止画撮影用プログラム pic()関数

  ～補足～

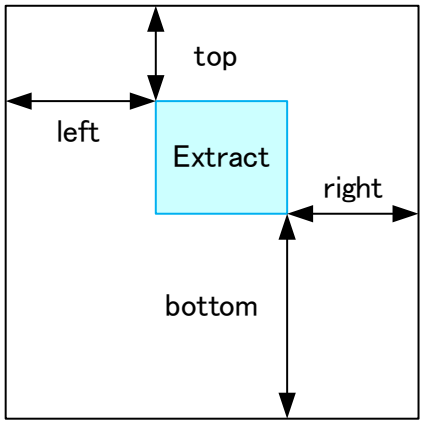
  本プログラムは、あらかじめ用意した画像データ（ocr_test.jpg）を使用しますが、Web カメラで撮影した画像を使用する場合は、photo()関数または pic()関数を使用してください。
- 25 行目～33 行目: 画像切り取りプログラム extract()関数
- 27 行目: 画像データ読み取り（oct_test.jpg）
- 32 行目: 画像切り取り範囲指定

  画像データから時刻（分）の部分を切り取る。

  ～補足～

  画像ファイルから切り取る範囲を「Extract」とした場合、切り取り範囲の指定は次のとおりです。

  img[top : bottom, left : right]



- 33 行目: 切り取った画像「ocr_test_extract.jpg」を保存
- 35 行目～43 行目: メイン関数
- 36 行目～41 行目 :try ブロック
- 38 行目: fswebcam コマンドを使用して撮影した画像データを使用する場合はコメント（#）を外す
- 40 行目: OpenCV を使用して撮影した画像データを使用する場合はコメント（#）を外す
- 41 行目: 画像切り取りプログラム extract()関数を実行
- 42 行目～43 行目 :except ブロック

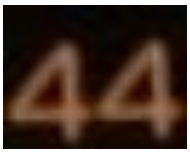
  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

続いて、OCR エンジン「Tesseract」を使用して切り取った画像から時刻の「分」の数字（44）をテキストデータに変換します。



手順⑧～OCR プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

20_Tesseract.py

手順⑨～OCR プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 20_Tesseract.py

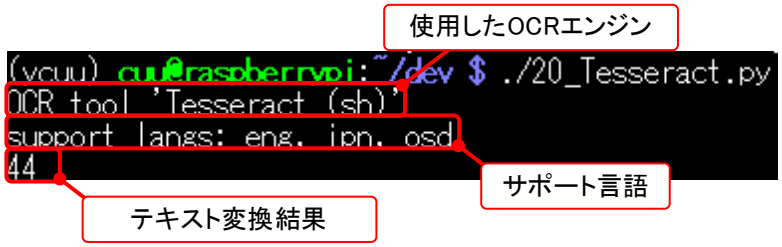
手順⑩～OCR プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./20_Tesseract.py

次の画面が表示され、画像から数字（44）を認識してテキストデータに変換されていることを確認してください。

本プログラムは、テキスト変換結果が「44」である場合、HAT Board 上の青 LED が 1 秒間点灯する構成になっています。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from PIL import Image
5	import sys
6	
7	import pyocr
8	import pyocr.builders

Copyright © 2024 YITJC All Rights Reserved.

```
9
10  import RPi.GPIO as GPIO
11  from time import sleep
12
13  def setup_gpio():
14      GPIO.setmode(GPIO.BCM)
15      #青LED
16      GPIO.setup(26, GPIO.OUT)
17      GPIO.output(26, GPIO.LOW)
18
19  def ocr():
20      global txt
21      tools = pyocr.get_available_tools()
22      if len(tools) == 0:
23          print("No OCR tool found")
24          sys.exit(1)
25
26      tool = tools[0]
27      print("OCR tool '%s'" % (tool.get_name()))
28      langs = tool.get_available_languages()
29      print("support langs: %s" % ", ".join(langs))
30
31      txt = tool.image_to_string(
32          #19_Extract.pyで切り取った画像を指定
33          Image.open("/home/cuu/img/ocr/ocr_test_extract.jpg"),
34          lang="eng",
35          builder=pyocr.builders.DigitBuilder(tesseract_layout=6)
36      )
37      print(txt)
38
39  if __name__ == '__main__':
40      try:
41          setup_gpio()
42          ocr()
43          #OCRの結果が、44であれば、青色LED点滅
44          if (int(txt) == 44):
45              GPIO.output(26, GPIO.HIGH)
46              sleep(1.0)
47              GPIO.output(26, GPIO.LOW)
```

Copyright © 2024 YITJC All Rights Reserved.

```
48        except KeyboardInterrupt:
49            pass
50        finally:
51            GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: PIL（Python Image Library）から Image ライブラリをインポート（import）
  PIL は、画像処理を行うための Python 標準パッケージです。
  Image ライブラリは、画像の読み込みや保存などの機能があります。
- 5 行目: Python のシステム関連の機能を使用するため、sys ライブラリをインポート（import）
- 7 行目: Python から OCR エンジンを利用可能にするため、pyocr ライブラリをインポート（import）
- 8 行目: pyocr ライブラリから builders クラスをインポート（import）
- 10 行目: RPi.GPIO ライブラリをインポート（import）
- 11 行目: time ライブラリから sleep 関数をインポート（import）
- 13 行目～17 行目: GPIO 初期設定関数
  青色 LED（GPIO26）の初期設定を行っています。
- 19 行目～35 行目: Tesseract を使用したテキストデータ変換プログラム ocr()関数
- 20 行目: テキスト変換されたデータを格納する変数「txt」をグローバル変数で定義
- 21 行目: pyocr ライブラリを使用して利用可能な OCR ツール（OCR エンジン）を取得
- 22 行目～24 行目: 利用可能な OCR ツールがない場合の処理
  エラーメッセージ「No OCR tool found」を表示してプログラムを終了します。
- 26 行目～27 行目: 使用する OCR ツールを print 表示
  利用可能な OCR ツールのうち、最初（tools[0]）の OCR ツールを print 表示します。
- 28 行目～29 行目: OCR ツールのサポート言語を print 表示
  利用可能な OCR ツールがサポートしている言語を print 表示します。
  インストールした Tesseract は、次の 3 つをサポートしています。
  eng：英語
  jpn：日本語
  osd：Orientation and script detection（画像に含まれるテキストの方向や書き方を検出）

Copyright © 2024 YITJC All Rights Reserved.

・ 31 行目～36 行目: OCR ツールの「image_to_string」メソッドを使用して画像データ内のテキストを抽出

○画像ファイルの指定

Image.open("/home/cuu/img/ocr/ocr_test_extract.jpg")

○使用する言語の指定（eng：英語、jpn：日本語）

lang="eng"

○builder の指定

builder=pyocr.builders.DigitBuilder(tesseract_layout=6)

～補足～

builders クラスにはさまざまな機能があり、今回は「DigitBuilder」を使用しています。

Builders	機能
TextBuilder	文字列を認識
WordBoxBuilder	単語単位で文字認識 + BoundingBox
LineBoxBuilder	行単位で文字認識 + BoundingBox
DigitBuilder	数字/記号を認識
DigitLineBoxBuilder	数字/記号を認識 + BoundingBox

～補足～

画像に含まれる文字・数字の認識精度は、tesseract_layout の設定で大きく変わります。

今回は、数字部分を切り取った画像を使用してテキスト変換を試みているため、「単一ブロックとして認識（tesseract_layout=6）」に設定しました。

Layout	Page segmentation modes
0	Orientation and script detection (OSD) only.
1	Automatic page segmentation with OSD.
2	Automatic page segmentation, but no OSD, or OCR. (not implemented)
3	Fully automatic page segmentation, but no OSD. (Default)
4	Assume a single column of text of variable sizes.
5	Assume a single uniform block of vertically aligned text.
6	Assume a single uniform block of text.
7	Treat the image as a single text line.
8	Treat the image as a single word.
9	Treat the image as a single word in a circle.
10	Treat the image as a single character.
11	Sparse text. Find as much text as possible in no particular order.
12	Sparse text with OSD.
13	Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.

これらの情報は、次のコマンドで確認できます。

(ycuu) cuu@raspberrypi: ~/dev $ tesseract --help-extra

Copyright © 2024 YITJC All Rights Reserved.

- 37 行目: 画像データから抽出したテキストデータを print 表示
- 39 行目～51 行目: メイン関数
- 40 行目～47 行目:try ブロック
- 41 行目: GPIO 初期設定関数 setuo_gpio()実行
- 38 行目: fswebcam コマンドを使用して撮影した画像データを使用する場合はコメント(#)を外す
- 40 行目: OpenCV を使用して撮影した画像データを使用する場合はコメント(#)を外す
- 41 行目: 画像切り取りプログラム extract()関数を実行
- 42 行目: テキストデータ変換プログラム ocr()関数実行
- 44 行目～47 行目: テキストデータの判定処理
  変換したテキストデータが「44」だった場合、青色 LED を 1 秒間点灯
- 48 行目～49 行目:except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文(何も処理しない)を記述しています。
- 50 行目～51 行目:Finally ブロック
- 51 行目: GPIO ピンのシステムクリア

Copyright © 2024 YITJC All Rights Reserved.

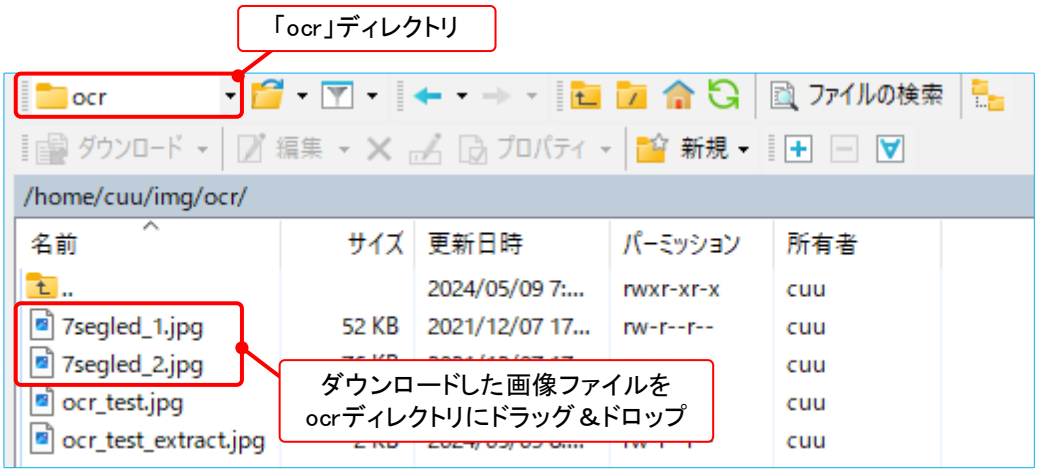
5.8 7 セグメント LED

7 セグメント LED を使用した表示器は、生産設備や計測器などで数多く使用されています。ここでは、Tesseract を使用して 7 セグメント LED に表示された数字をテキストデータに変換します。

手順①～7 セグメント LED 画像の保存～

7 セグメント LED を使用した表示器の画像データを保存します。

共有サイトにある画像データ「7segled_1.jpg」、「7segled_2.jpg」を Windows PC にダウンロードしたあと、WinSCP を使用して（ドラッグ＆ドロップ）、「ocr」ディレクトリに保存してください。



テキスト変換に使用する画像データを紹介します。あらかじめ 7 セグ LED 表示器の部分を切り取った画像を使用します。

7segled_1.jpg

7segled_2.jpg

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

21_Tesseract.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 21_Tesseract.py
```

Copyright © 2024 YITJC All Rights Reserved.

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./21_Tesseract.py

2 つの 7 セグメント LED 画像の変換結果が表示されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./21_Tesseract.py
7segLED Test#1 '135'
7segLED Test#2 '0 5'
```

「7segled_1.jpg」画像は正しい値（135）に変換されましたが、「7segled_2.jpg」画像は間違えた結果（0 5）に変換されてしまいました。

画像ファイル	撮影フレーム数	学習データ	変換結果
7segled_1		eng	135
7segled_2		eng	0 5

～補足～

21_Tesseract.py のプログラムは、20_Tesseract.py の内容とほぼ同じであるため、プログラムの解説は省略します。

参考～Tesseract の学習データ～

Tesseract は 3 つの言語（eng、jpn、osd）をサポートしています。各言語には「.traineddata」と呼ばれる学習データが用意されており、テキスト変換するときに使用されます。7 セグメント LED 画像のテキスト変換に使用した学習データは、「eng.traineddata」です。

次のコマンドを実行すると、Raspberry Pi にインストールされている学習データを確認できます。

(ycuu) cuu@raspberrypi:$ cd /usr/share/tesseract-ocr/5/tessdata

(ycuu) cuu@raspberrypi:/usr/share/tesseract-ocr/5/tessdata $ ls

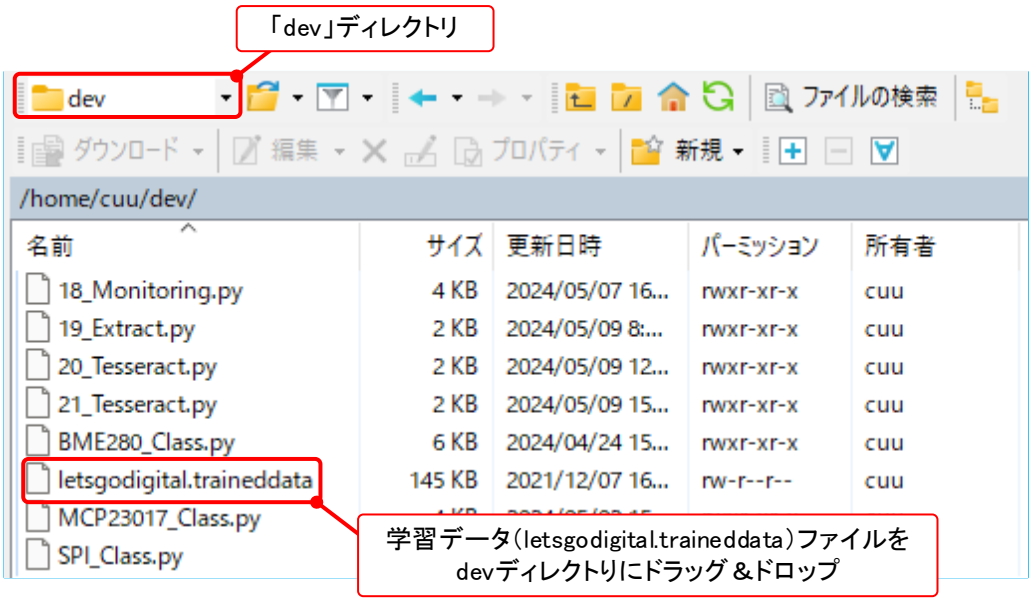
```
(ycuu) cuu@raspberrypi:~ $ cd /usr/share/tesseract-ocr/5/tessdata
(ycuu) cuu@raspberrypi:/usr/share/tesseract-ocr/5/tessdata $ ls
configs  eng.traineddata  jpn.traineddata  osd.traineddata  pdf.ttf  tessconfigs
```

7 セグメント LED は 7 つの細長い LED パターンで数字や英字を表現するため、LED と LED との間に生じる隙間や点灯していない LED パターンの影響などにより数字の認識精度が低くなります。そこで、7 セグメント LED 専用の学習データを使用してテキストデータへの変換を試してみます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～7 セグメント LED 専用学習データ保存～

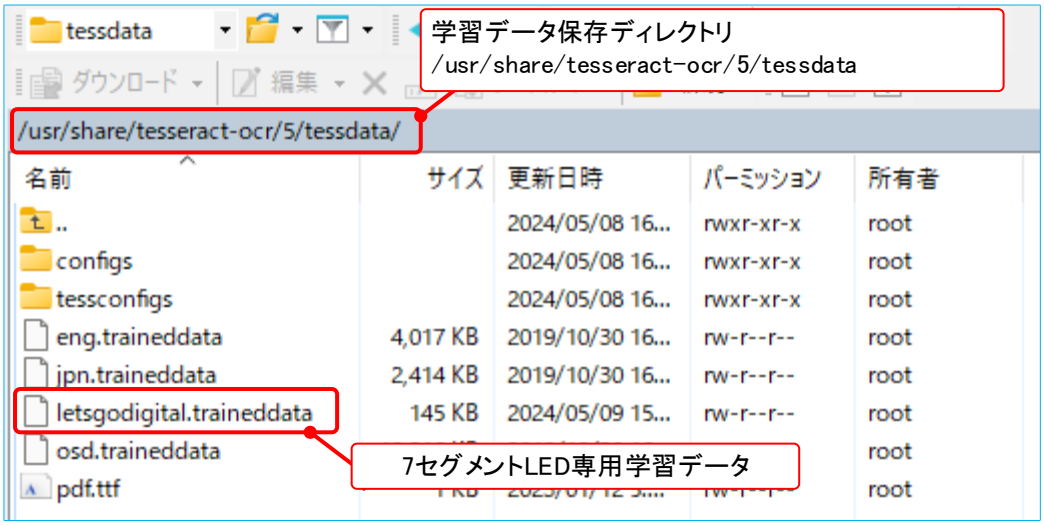
共有サイトにある学習データ「letsgodigital.traineddata」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「dev」ディレクトリに保存してください。



「dev」ディレクトリに保存した学習データ「letsgodigital.traineddata」を Tesseract の学習データ保存先にコピーします。「dev」ディレクトリへ移動して、次のコマンドを実行してください。

(ycuu) cuu@raspberrypi:~/dev $ sudo cp ./letsgodigital.traineddata /usr/share/tesseract-ocr/5/tessdata

WinSCP を使用して、学習データ保存ディレクトリ（/usr/share/tesseract-ocr/5/tessdata）に 7 セグメント LED 専用の学習データが追加されたことを確認してください。



手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

22_Tesseract.py

Copyright © 2024 YITJC All Rights Reserved.

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 22_Tesseract.py

手順⑧～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./22_Tesseract.py

2 つの 7 セグメント LED 画像の変換結果が表示されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./22_Tesseract.py
7segLED Test#1 '135'
7segLED Test#2 '-1 15'
```

「7segled_2.jpg」画像は、7 セグメント LED 専用学習データを使用しても正しい数字に変換されません。

OCR エンジンを使用してテキスト変換を行う場合、学習データだけでなく、画像処理も重要です。通常、画像処理は OCR の前処理として実施します。2 値化やノイズ除去などの画像処理により OCR 性能が向上するため、正確なテキスト変換を実現するうえで画像処理は重要です。

画像ファイル	使用画像	学習データ	変換結果
7segled_1		letsgodigital	135
7segled_2		letsgodigital	-1 15

～補足～

22_Tesseract.py のプログラムは、20_Tesseract.py の内容とほぼ同じであるため、プログラムの解説は省略します。

Copyright © 2024 YITJC All Rights Reserved.

5.9 OpenCV 画像処理

7 セグメント LED 画像「7segled_2.jpg」を使用して OpenCV による画像処理を行います。

画像ファイル	使用画像	学習データ	変換結果
7segled_2		eng	0 5
		letsgodigital	−1 15

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

23_OpenCV.py

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 23_OpenCV.py

手順③～プログラム実行～

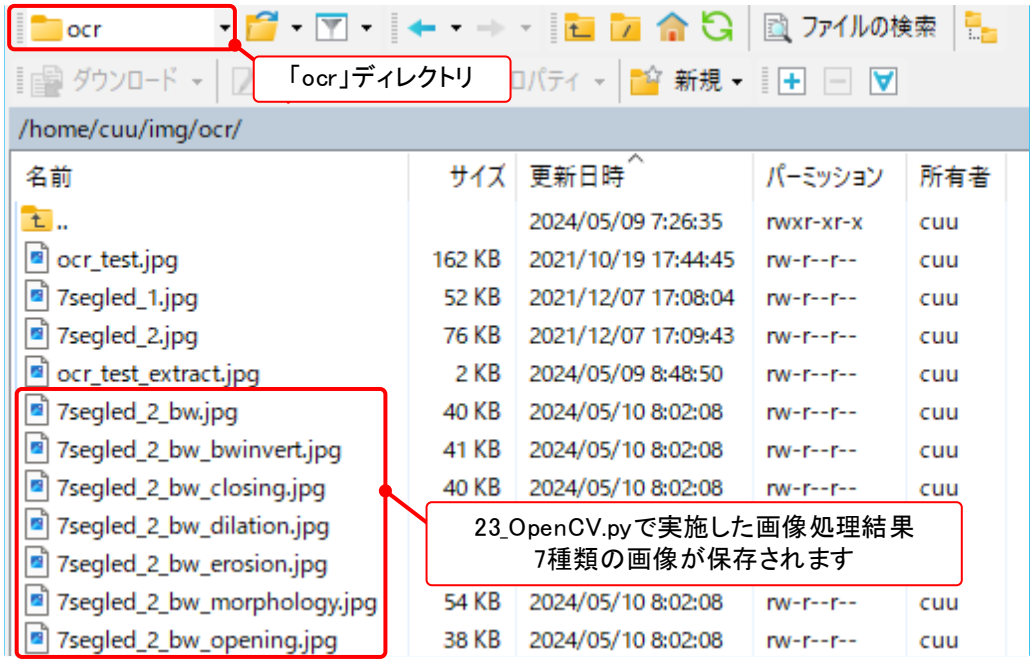
対象プログラムを実行します。

(ycuu) cuu@raspberrypi: ~/dev $ ./23_OpenCV.py

Tera Term 画面上には、2 値化処理したときの閾値が表示されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./23_OpenCV.py
ret: 144.0
```

WinSCP 画面で次の画像が保存されていることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

参考～画像処理解説～

OCR エンジンを使用したテキスト変換の「前処理」として実施される画像処理について説明します。

- 2 値化

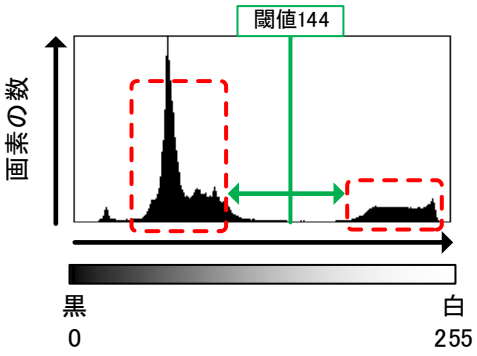
  2 値化処理は、対象の画像を「白」と「黒」の 2 値のみの画像に変換することです。

  2 値化処理は画像の中の対象物と背景の境界を明確にする目的で使用されます。今回の例で説明すると、7 セグメント LED 表示された数字とそれ以外の部分の境界を明確にします。

  「大津の 2 値化」は、2 値化の閾値（白と黒の境界）を自動で求める手法です。具体的には、画像のヒストグラムから、クラス間分離度が最大になる値を閾値に設定します。

  「7segled_2.jpg」画像（グレー画像に変換）のヒストグラムを紹介します。

7segled_2.jpg（グレー変換）



- カーネル（kernel）について

  収縮、膨張、Gradient（勾配）処理には、カーネル（kernel）と呼ばれる行列式を使用します。3x3 行列や 5x5 行列、7x7 行列が使用されます。

```
kernel = np.ones((3, 3), np.uint8)
```

  カーネルの範囲は自由に設定できます。行列のサイズに加えて、行列のデータを「0」に設定して範囲を設定できます。「0」に設定した場合、「0」に設定した個所に黒があっても、対象画像は黒に変換されません。

  23_OpenCV.py で使用しているカーネルは、配列の要素がすべて 1（全方向）の 3x3 行列です。

1	1	1
1	1	1
1	1	1

全方向

0	1	0
1	1	1
0	1	0

縦・横方向

0	0	0
1	1	1
0	0	0

横方向

0	1	0
0	1	0
0	1	0

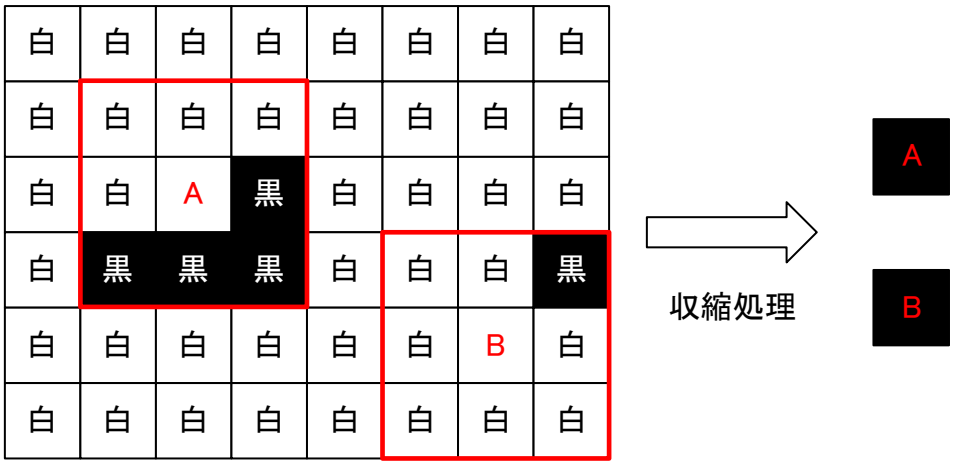
縦方向

Copyright © 2024 YITJC All Rights Reserved.

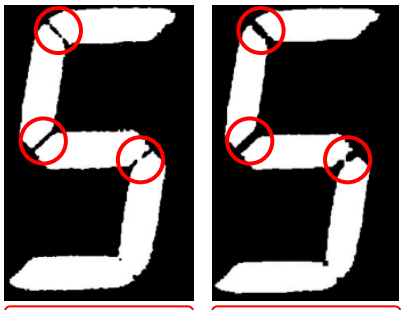
- 収縮処理

収縮処理は、2 値画像の白画素領域を収縮（減少）させる処理です。収縮処理は、対象画素の周辺（カーネルで指定した範囲）に黒があれば黒に変換します。具体的には、今回使用するカーネル（要素がすべて 1 の 3x3 行列）の場合、対象画素を中心として 3x3 の範囲すべての画素を確認し、1 つでも黒があれば黒に変換します。

次の図は、1 枚の画像（6x8 画素）と考えてください。赤い四角（□）は、カーネル（3x3 行列）の範囲です。画素 A、画素 B ともにカーネルの範囲に黒があるため、収縮処理の後、画素 A 及び画素 B は黒に変換されます。



行列の要素はすべて 1 の条件のもと、カーネルサイズだけを変更して収縮処理を実施した結果を示します。カーネルサイズを大きくすると、白画素の領域が収縮（減少）している様子がわかります。

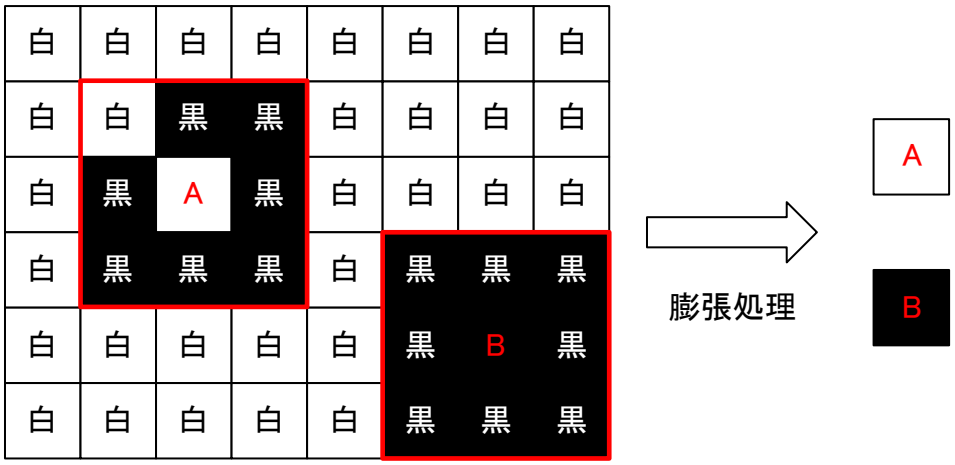


Copyright © 2024 YITJC All Rights Reserved.

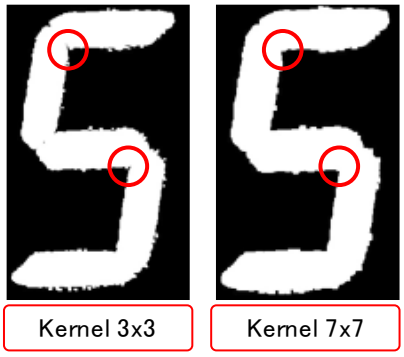
- 膨張処理

膨張処理は、2 値画像の白画素領域を膨張（増加）させる処理です。膨張処理は、対象画素の周辺（カーネルで指定した範囲）に白があれば白に変換します。具体的には、今回使用するカーネル（要素がすべて 1 の 3x3 行列）の場合、対象画素を中心として 3x3 の範囲すべての画素を確認し、1 つでも白があれば白に変換します。

次の図は、1 枚の画像（6x8 画素）と考えてください。赤い四角（□）は、カーネル（3x3 行列）の範囲です。画素 A はカーネルの範囲に白画素があるため、膨張処理後、画素 A は白に変換されます。画素 B はカーネルの範囲に白画素がないため、膨張処理後、画素 B は黒画素のままです。



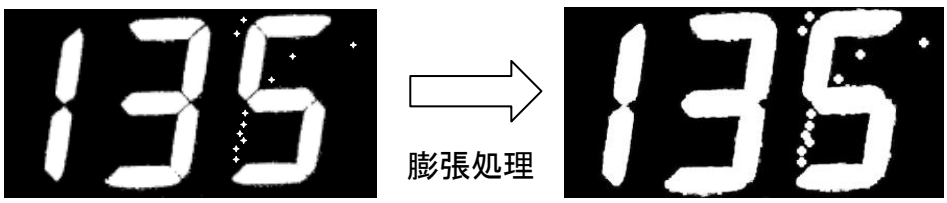
行列の要素はすべて 1 の条件のもと、カーネルサイズだけを変更して収縮処理を実施した結果を示します。カーネルサイズを大きくすると、白画素の領域が膨張（増加）している様子がわかります。



- オープニング処理

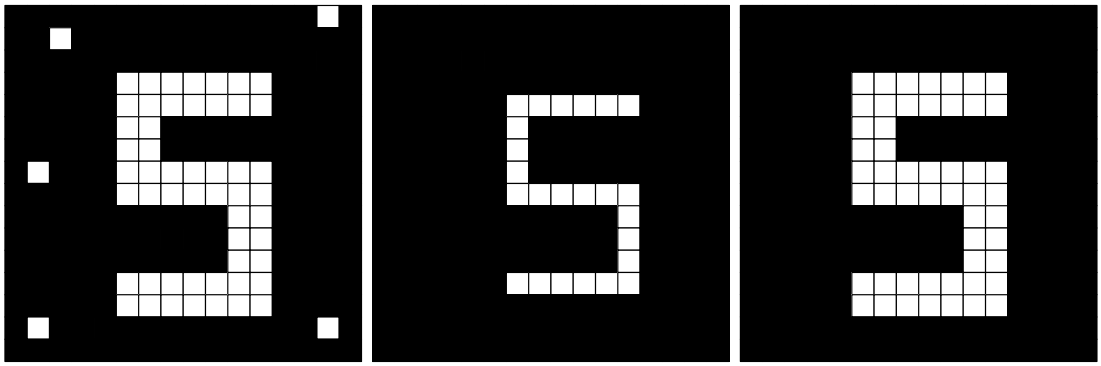
収縮処理、膨張処理の順番で行われる画像処理です。

例えば、次の 2 値画像をオープニング処理する場合について考えてみます。この画像は、数字「5」の周辺にノイズが含まれています。この画像を膨張処理した場合、ノイズを含む白画素領域が膨張（増加）するため、数字「5」が「6」と識別されてしまう可能性があります。



Copyright © 2024 YITJC All Rights Reserved.

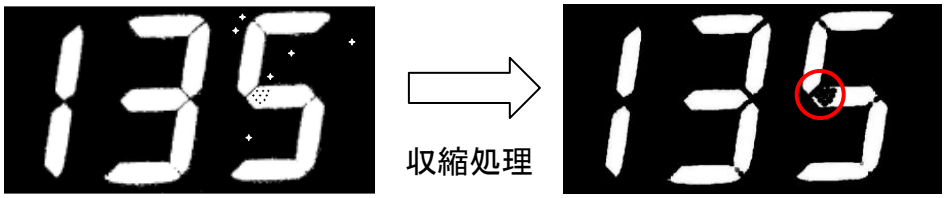
オープニング処理は、最初に収縮処理が行われるため、画像中の小さなノイズを除去できます。しかし、収縮処理を行うことで元の画像の輪郭（形状）が失われてしまうため、そのあとに膨張処理を実施して元の画像の輪郭（形状）を維持します。そのため、オープニング処理は必ず収縮処理と膨張処理を同じ回数実施します。



- クロージング処理

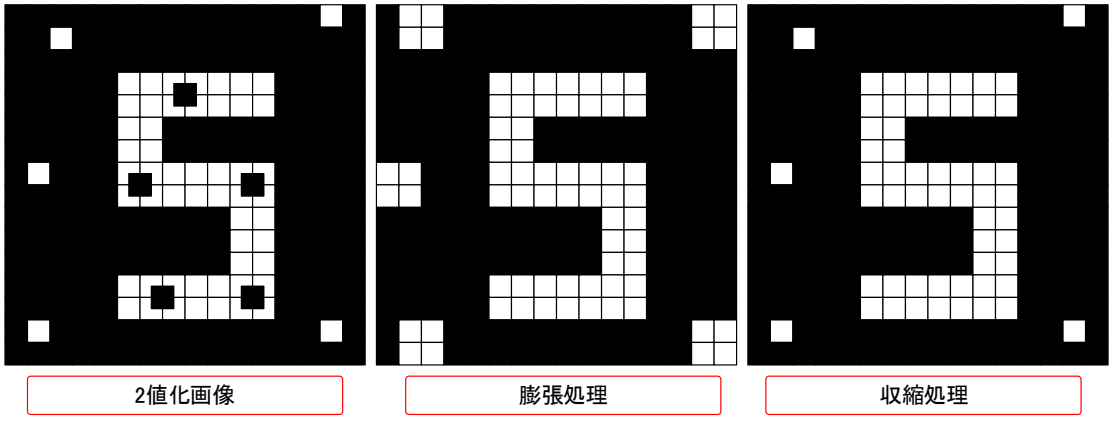
膨張処理、収縮処理の順番で行われる画像処理です。

例えば、次の 2 値画像をオープニング処理する場合について考えてみます。この画像は、数字「5」の周辺と「5」の中心付近にノイズが含まれています。この画像を収縮処理した場合、黒画素が膨張（増加）するため「5」周辺の白画素ノイズは除去されますが、「5」の中心にある黒画素が膨張して数字の「5」が分断されています。



クロージング処理は、最初に膨張処理が行われるため、元の画像の輪郭の内部に含まれるノイズ（黒画素）を除去できます。しかし、輪郭の周辺にあるノイズ（白画素）は範囲が広がってしまいます。その後、収縮処理を実施しても、輪郭の周辺にあるノイズ（白画素）は除去されず、ノイズが残った画像になります。

クロージング処理単体ではノイズの除去が難しいケースが多いため、ノイズを完全に除去するためには、クロージング処理を実施したあとにオープニング処理を実施して元の画像の特徴を正確に再現します。



Copyright © 2024 YITJC All Rights Reserved.

- Morphology 勾配（Gradient）処理

  勾配処理は、膨張した画像と収縮した画像の差分を取る処理です。この処理により、対象物の外郭であるエッジが強調されます。エッジが強調されるため、対象物の輪郭抽出が可能となり、物体検出の前処理に使用されています。

- 白黒反転処理

  画像データの各画素について、ビット反転処理を行います。2 値画像の場合は、黒画素（0）を白画素（255）に変換し、白画素は黒画素に変換します。この処理により、画像の白黒が反転するため、対象物または背景を強調する効果があります。

参考～テキスト変換用プログラム～

「23_OpenCV.py」で画像処理した各画像を使用してテキストデータに変換するプログラムを共有サイトで公開します。

［対象プログラム］

24_Tesseract.py

- 学習データを「letsgodigital」に設定したときの結果

```
(ycuu) cuu@raspberrypi:~/dev $ ./24_Tesseract.py
7segLED Test#1 BW '135'
7segLED Test#2 Erosion '131-
1.13'
7segLED Test#3 Dilation '135'
7segLED Test#4 Opening '135'
7segLED Test#5 Closing '135'
7segLED Test#6 Morphology '535'
7segLED Test#7 BWinvert '135'
```

- 学習データを「eng」に設定したときの結果

```
(ycuu) cuu@raspberrypi:~/dev $ ./24_Tesseract.py
7segLED Test#1 BW '145'
7segLED Test#2 Erosion ''
7segLED Test#3 Dilation '145'
7segLED Test#4 Opening '145'
7segLED Test#5 Closing '145'
7segLED Test#6 Morphology ''
7segLED Test#7 BWinvert '14'
```

「24_Tesseract.py」の学習データは「letsgodigital」に設定しています。

```
lang="letsgodigital"      #デフォルト設定
```

学習データを「eng」を使用する場合は、次のとおり書き換えて実行してください。

```
lang="eng" #デフォルト設定
```

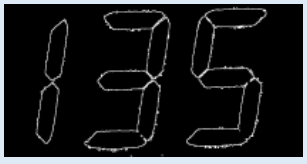
Copyright © 2024 YITJC All Rights Reserved.

手順④～画像処理結果の確認～

手順③で得られた画像処理結果（画像）の確認と各画像のテキスト変換結果をまとめます。

学習データ「eng」の場合は、すべての画像で「誤変換」という結果になりました。

学習データ「letsgodigital」の場合は、収縮処理及び Morphology 処理以外は正しい数字（135）に変換されました。このように、OCR 変換の前処理（画像処理）と学習データの組み合わせによって結果が大きく異なります。

画像ファイル	使用画像	学習データ	変換結果
7segled_2 元画像		eng	0 5
		letsgodigital	−1 15
7segled_2_bw 白黒 2 値化		eng	145
		letsgodigital	135
7segled_2_bw_erosion 収縮処理		eng	認識せず
		letsgodigital	131−1.13
7segled_2_bw_dilation 膨張処理		eng	145
		letsgodigital	135
7segled_2_bw_opening Opening 処理		eng	145
		letsgodigital	135
7segled_2_bw_closing Closing 処理		eng	145
		letsgodigital	135
7segled_2_bw_morphology Gradient（勾配）処理		eng	認識せず
		letsgodigital	535
7segled_2_bw_bwinvert 白黒反転処理		eng	14
		letsgodigital	135

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import cv2
5	#morphology変換用
6	import numpy as np
7	
8	#2値化
9	#7segled_2.jpgを数値化(Tessaract)するための前処理は、2値化処理だけでOK
10	def bw():
11	img = cv2.imread('/home/cuu/img/ocr/7segled_2.jpg',cv2.IMREAD_COLOR)
12	img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13	
14	#任意の閾値(threshold)を指定する場合
15	#threshold = 100
16	#二値化(閾値100を超えた画素を255にする)
17	#ret, img_thresh = cv2.threshold(img, threshold, 255, cv2.THRESH_BINARY)
18	
19	#閾値自動設定
20	#第2引数 (0: grayscale, 6: color)
21	ret, img_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_OTSU)
22	#自動設定された閾値の確認
23	print("ret: {}".format(ret))
24	#2値化画像の保存
25	cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw.jpg", img_otsu)
26	
27	#収縮(erosion)処理
28	def erosion():
29	#2値化画像読み込み
30	img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
31	kernel = np.ones((3, 3), np.uint8)
32	erosion = cv2.erode(img,kernel,iterations = 1)
33	cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_erosion.jpg", erosion)
34	
35	#膨張(dilation)処理
36	def dilation():
37	#2値化画像読み込み

Copyright © 2024 YITJC All Rights Reserved.

```
38        img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
39        kernel = np.ones((3, 3), np.uint8)
40        dilation = cv2.dilate(img,kernel,iterations = 1)
41        cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_dilation.jpg", dilation)
42
43    #Opening処理(収縮処理→膨張処理)
44    def opening():
45        #2値化画像読み込み
46        img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
47        kernel = np.ones((3, 3), np.uint8)
48        opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
49        cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_opening.jpg", opening)
50
51    #Closing処理(膨張処理→収縮処理)
52    def closing():
53        #2値化画像読み込み
54        img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
55        kernel = np.ones((3, 3), np.uint8)
56        closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
57        cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_closing.jpg", closing)
58
59    #Morphology変換(膨張画像と収縮画像の差分)
60    def morphology():
61        #2値化画像読み込み
62        img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
63        kernel = np.ones((3, 3), np.uint8)
64        morphology = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
65        cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_morphology.jpg", morphology)
66
67    #白黒反転処理
68    def bwinvert():
69        #2値化画像読み込み
70        img = cv2.imread('/home/cuu/img/ocr/7segled_2_bw.jpg')
71        bwinvert = cv2.bitwise_not(img)
72        cv2.imwrite("/home/cuu/img/ocr/7segled_2_bw_bwinvert.jpg", bwinvert)
73
74    if __name__ == '__main__':
75        try:
76            bw()
```

Copyright © 2024 YITJC All Rights Reserved.

```
77              erosion()
78              dilation()
79              opening()
80              closing()
81              morphology()
82              bwinvert()
83          except KeyboardInterrupt:
84              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 5 行目: OpenCV ライブラリをインポート（import）
- 6 行目: Morphology 処理を行うために必要な Numpy（ナムパイ）ライブラリをインポート（import）
- 10 行目～25 行目: 2 値化処理プログラム bw()関数
- 11 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 12 行目: 対象画像（7segled_2.jpg）をグレースケールの画像に変換
- 21 行目: グレースケールに変換した画像に対して大津の 2 値化処理を実行
- 23 行目: 大津の 2 値化処理で自動設定された閾値の値を print 表示
- 25 行目: 大津の 2 値化処理を実施した画像を保存（7segled_2_bw.jpg）

  ～補足～

  大津の 2 値化処理（閾値を自動で設定）を使用しない場合は、15 行目～17 行目の処理を実施します。

  まず、15 行目で 2 値化の閾値（threshold）を設定します。

  続いて、17 行目で 2 値化処理を実行します。このとき、閾値を超えた画素は 255 に変換されます。

- 28 行目～33 行目: 収縮処理プログラム erosion()関数
- 30 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 31 行目: カーネルを定義（配列の要素がすべて 1（全方向）の 3x3 行列）
- 32 行目: 2 値化画像に対して収縮処理を実行
- 33 行目: 収縮処理を実施した画像を保存（7segled_2_bw_erosion.jpg）

- 36 行目～41 行目: 膨張処理プログラム dilation()関数
- 38 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 39 行目: カーネルを定義（配列の要素がすべて 1（全方向）の 3x3 行列）
- 40 行目: 2 値化画像に対して膨張処理を実行
- 41 行目: 膨張処理を実施した画像を保存（7segled_2_bw_dilation.jpg）

- 44 行目～49 行目: オープニング処理プログラム opening()関数
- 46 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 47 行目: カーネルを定義（配列の要素がすべて 1（全方向）の 3x3 行列）

Copyright © 2024 YITJC All Rights Reserved.

- 48 行目: 2 値化画像に対してオープニング処理を実行
- 49 行目: オープニング処理を実施した画像を保存（7segled_2_bw_opening.jpg）

- 52 行目～57 行目: クロージング処理プログラム closing()関数
- 54 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 55 行目: カーネルを定義（配列の要素がすべて 1（全方向）の 3x3 行列）
- 56 行目: 2 値化画像に対してクロージング処理を実行
- 57 行目: クロージング処理を実施した画像を保存（7segled_2_bw_closing.jpg）

- 60 行目～65 行目: Gradient（勾配）処理プログラム morphology()関数
- 62 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 63 行目: カーネルを定義（配列の要素がすべて 1（全方向）の 3x3 行列）
- 64 行目: 2 値化画像に対して勾配処理を実行
- 65 行目: 勾配処理を実施した画像を保存（7segled_2_bw_morphology.jpg）

- 68 行目～72 行目: 白黒反転処理プログラム bwinvert()関数
- 70 行目: 画像処理の対象画像（7segled_2.jpg）を読み込み
- 71 行目: 2 値化画像に対して勾配処理を実行
- 72 行目: 勾配処理を実施した画像を保存（7segled_2_bw_bwinvert.jpg）

- 74 行目～84 行目: メイン関数
- 77 行目～78 行目 :try ブロック
- 76 行目: 2 値化処理プログラム bw()関数実行
- 77 行目: 収縮処理プログラム erosion()関数実行
- 78 行目: 膨張処理プログラム dilation()関数実行
- 79 行目: オープニング処理プログラム opening()関数実行
- 80 行目: クロージング処理プログラム closing()関数実行
- 81 行目: Gradient（勾配）処理プログラム morphology()関数実行
- 82 行目: 白黒反転処理プログラム bwinvert()関数実行
- 83 行目～84 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

5.10 Google Cloud Vision

Google Cloud Vision は、文字や数字の識別・抽出だけでなく、物体検出や顔の表情などを自動で検出できるクラウドサービスです。Python から Google Cloud Vison を利用するためのライブラリも用意されているため、Raspberry Pi が撮影した画像から文字や数字の識別・抽出できます。

Python から Google Cloud Vision を利用する方法については、9 章「クラウドサービスの活用」で詳しく説明します。ここでは、Google が提供している「Google Cloud Vision API テストサイト」を利用して、Google Cloud Vision サービスの機能・性能を確認します。

手順①～テスト用画像データ確認～

共有サイトに保存されている「7seg_test_GCV.jpg」ファイルをダウンロードしてください。

今回使用する画像は、4 種類の 7 セグメント LED 表示器が 1 枚の画像ファイルになっています。

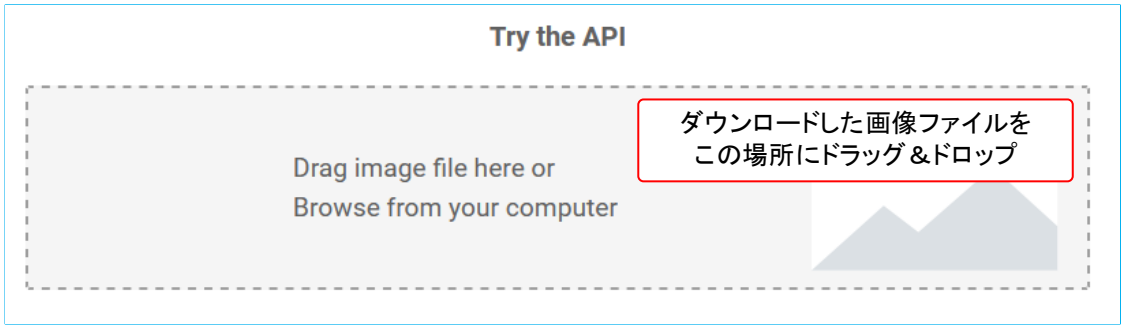
数字領域の抽出性能も確認するため、各 7 セグメント LED 表示器の切り取りは行いません。

手順②～テストサイトアクセス～

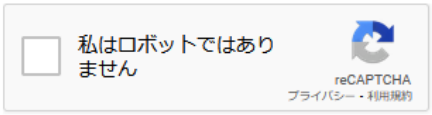
次のリンクをクリックして「Google Cloud Vision API テストサイト」にアクセスします。

URL: https://cloud.google.com/vision/docs/drag-and-drop

リンクを開いたら、手順①でダウンロードした画像をアップロードします（ドラッグ＆ドロップ）。



CAPTCHA（キャプチャ）認証画面が表示されるので、認証を済ませてください。



～補足～

CAPTCHA 認証は、「Completely Automated Public Turing test to tell Computers and Humans Apart」の略称で、コンピュータと人間を区別する完全に自動化された公開チューリングテストです。つまり、コンピュータによるプログラムではないことを確認するテストです。

Copyright © 2024 YITJC All Rights Reserved.

手順③～テスト結果の確認～

「text」タブをクリックすると、抽出した領域とテキストデータへの変換結果が表示されます。

7 セグメント LED 表示器の領域抽出は正確に行われていますが、テキストデータへの変換結果には一部誤りがあります。しかし、画像の切り取りや画像処理を全く実施していない画像であることを考えると、Tesseract と比べて数字の認識・抽出およびテキストデータへの変換性能が高いことがわかります。

なお、Tesseract でテキスト変換に使用した画像データ（7segled_2jpg）のテスト結果は次のとおりです。

画像処理を実施していない元画像でテストしていますが、7 セグメント LED 表示器の数字（135）が正確にテキストデータに変換されています。

Copyright © 2024 YITJC All Rights Reserved.

参考～Google アカウント取得～

次のリンクをクリックして「Google Cloud Vision API テストサイト」にアクセスします。
URL: https://www.google.com/

画面右上にある「ログイン」をクリックしてください。

「アカウント作成」から「個人で使用」をクリックしてください。

名前を入力して「次へ」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

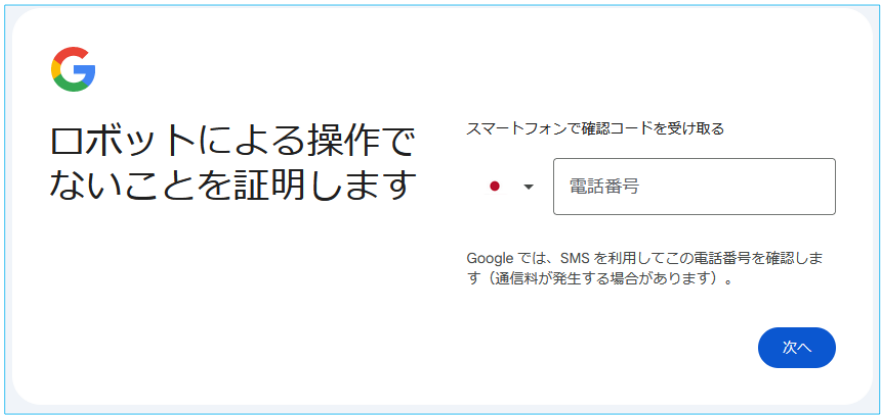
基本情報を入力して「次へ」をクリックしてください。

Google から提案された「メールアドレスの選択」または「自分で Gmail アドレスを作成」してから「次へ」をクリックしてください。

パスワードを設定して「次へ」をクリックしてください。

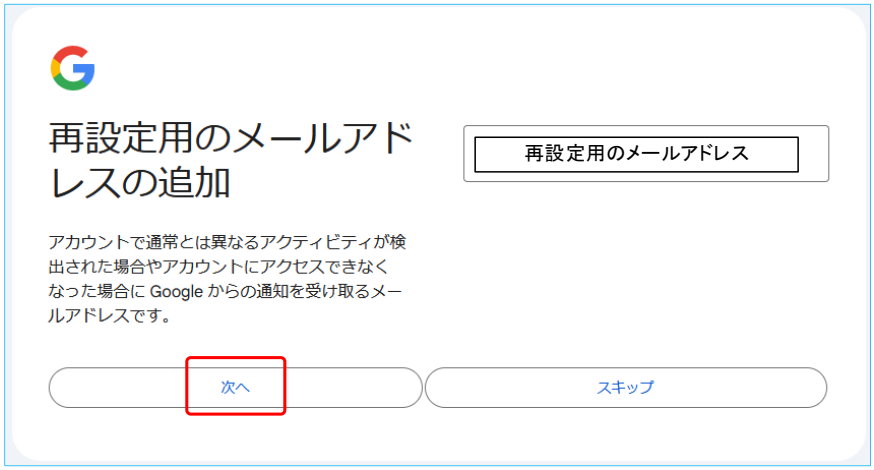
Copyright © 2024 YITJC All Rights Reserved.

スマートフォンを使用して CAPTCHA（キャプチャ）認証を行います。



スマートフォン（SMS）で受信したコード（6 桁の数字）を入力して「次へ」をクリックしてください。

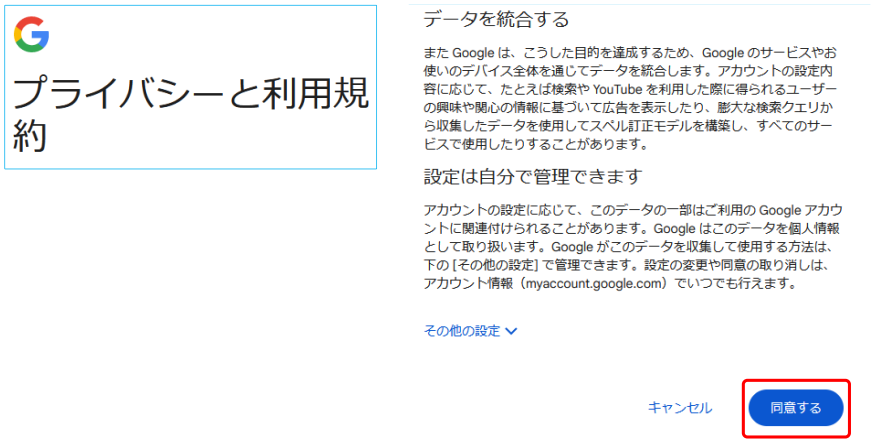
再設定用のメールアドレスを登録して「次へ」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

登録したアカウント情報を確認して「次へ」をクリックしてください。

最後に利用規約を確認してから「同意する」をクリックしてください。



以上で Google アカウントの作成が完了しました。作成したアカウントで Google にログインすると、クラウドサービスやアカウントの管理ができます。

Copyright © 2024 YITJC All Rights Reserved.

第 6 章 データ収集と活用

3 章から 5 章では、センサや生産設備、画像からのデータ取得について学習しました。本講座で構築する IoT システムは、取得したデータを Raspberry Pi で収集・可視化し、オンラインストレージや SNS を利用してリアルタイムに情報共有します。Raspberry Pi を使用してデータの収集や可視化を自動化することで、データ集計やグラフ作成に係る業務を削減できるだけでなく、データの特徴や傾向を瞬時に把握できるメリットも得られます。

ここでは、取得したデータを有効活用するために、データ収集や蓄積、可視化、分析するためのプログラミングについて学習します。

6.1 データ収集

データ収集について学習するため、HAT Board にある温湿度・気圧センサ（BME280）のデータを csv ファイルに収集（保存）する方法を説明します。

手順①～ディレクトリ作成～

csv ファイルを保存するためのディレクトリ「data」を作成します。

ホームディレクトリから次のコマンドを実行します。

(ycuu) cuu@raspberrypi:~ $ mkdir data

続いて、data ディレクトリに移動します

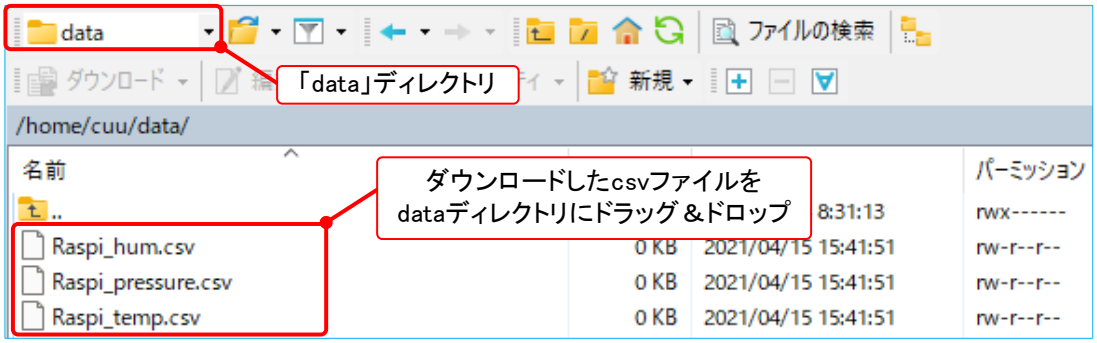
(ycuu) cuu@raspberrypi:~ $ cd data

手順②～csv ファイル保存～

共有サイトから csv ファイルを Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「data」ディレクトリに保存してください。

共有サイトからダウンロードする csv ファイルは、次の 3 つです。

- ✓ Raspi_temp.csv
- ✓ Raspi_hum.csv
- ✓ Raspi_pressure.csv



Copyright © 2024 YITJC All Rights Reserved.

手順③～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

25_CSV.py

手順④～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/data $ cd ..

※「cd ..」コマンドを実行すると、1 つ上のディレクトリに移動できます。

(ycuu) cuu@raspberrypi:$ cd dev

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 25_CSV.py

```
(ycuu) cuu@raspberrypi:~/data $ cd ..
(ycuu) cuu@raspberrypi:~ $ cd dev
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 25_CSV.py
```

手順⑤～プログラム実行～

対象プログラムを実行します。

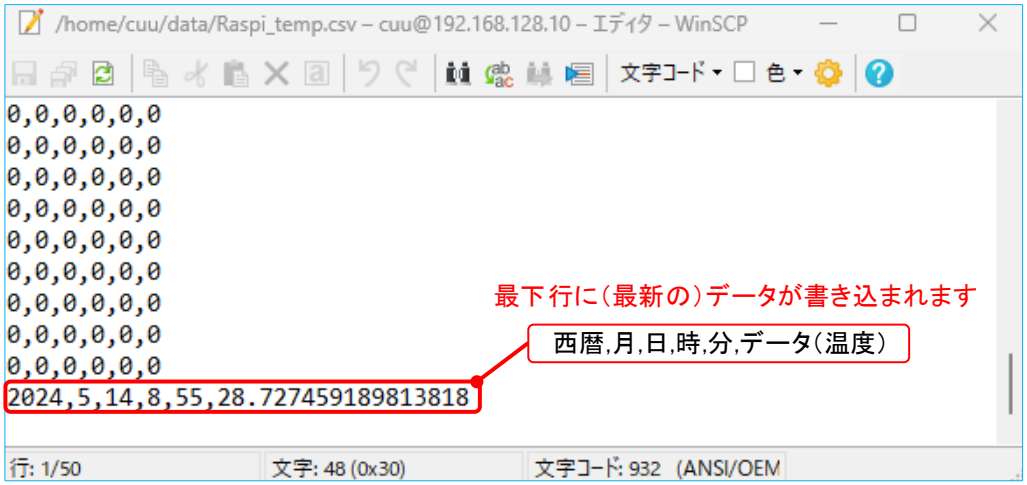
(ycuu) cuu@raspberrypi:~/dev $ ./25_CSV.py

プログラムを実行すると、温度、湿度、気圧データが print 表示され、csv ファイルにデータが書き込まれます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./25_CSV.py
temperature :  28.73 ℃
humidity    :  32.09 %
pressure    :  968.46 hPa
```

「data」ディレクトリにある csv ファイルを開いてください(csv ファイルは WinSCP 画面上でダブルクリックすると開くことができます)。

csv ファイルの最下行(50 行目)にデータが書き込まれていることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding:utf-8
3	
4	import BME280_Class
5	import datetime
6	import csv
7	
8	#データ格納数定義
9	MAX_DATA=50
10	#日付・時刻取得
11	date=datetime.datetime.today()
12	
13	#温度データcsv記録
14	def Temp_csv():
15	#温度データ格納用配列
16	ary_temp=["0,0,0,0,0,0¥n" for i in range(MAX_DATA)]
17	#csvファイルオープン(読み込み専用"r")
18	fp=open("/home/cuu/data/Raspi_temp.csv","r")
19	line=fp.readline()
20	while line:
21	ary_temp.pop(0)
22	ary_temp.append(line)
23	line=fp.readline()
24	ary_temp.pop(0)
25	ary_temp.append(str(date.year)+","+str(date.month)+","+str(date.day)+","+str(date.hour)+","+str(date.minute)+","+str(temperature)+"¥n")
26	fp.close()
27	#csvファイルオープン(書き込み専用"w")
28	fp=open("/home/cuu/data/Raspi_temp.csv","w")
29	fp.writelines(ary_temp)
30	fp.close()
31	
32	#湿度データcsv記録
33	def Hum_csv():
34	#湿度データ格納用配列
35	ary_hum=["0,0,0,0,0,0¥n" for i in range(MAX_DATA)]
36	#csvファイルオープン(読み込み専用"r")

Copyright © 2024 YITJC All Rights Reserved.

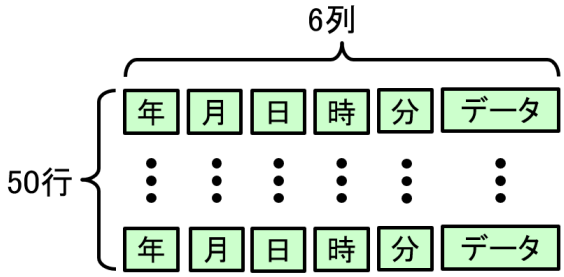
```
37      fp=open("/home/cuu/data/Raspi_hum.csv","r")
38      line=fp.readline()
39      while line:
40          ary_hum.pop(0)
41          ary_hum.append(line)
42          line=fp.readline()
43      ary_hum.pop(0)
44  ary_hum.append(str(date.year)+","+str(date.month)+","+str(date.day)+","+str(date.hour)+","+str(date.minute)+","+str(hum)
    +"¥n")
45      fp.close()
46      #csvファイルオープン(書き込み専用"w")
47      fp=open("/home/cuu/data/Raspi_hum.csv","w")
48      fp.writelines(ary_hum)
49      fp.close()
50
51  #気圧データcsv記録
52  def Press_csv():
53      #気圧データ格納用配列
54      ary_pressure=["0,0,0,0,0,0¥n" for i in range(MAX_DATA)]
55      #csvファイルオープン(読み込み専用"r")
56      fp=open("/home/cuu/data/Raspi_pressure.csv","r")
57      line=fp.readline()
58      while line:
59          ary_pressure.pop(0)
60          ary_pressure.append(line)
61          line=fp.readline()
62      ary_pressure.pop(0)
63  ary_pressure.append(str(date.year)+","+str(date.month)+","+str(date.day)+","+str(date.hour)+","+str(date.minute)+","+str(
    pressure)+"¥n")
64      fp.close()
65      #csvファイルオープン(書き込み専用"w")
66      fp=open("/home/cuu/data/Raspi_pressure.csv","w")
67      fp.writelines(ary_pressure)
68      fp.close()
69
70  if __name__ == '__main__':
71      try:
72          bme = BME280_Class.BME280()
73          bme.readData()
```

Copyright © 2024 YITJC All Rights Reserved.

```
74              temperature = bme.getTemperature()
75              hum = bme.getHumidity()
76              pressure = bme.getPressure()
77              #温度・湿度・気圧データ表示
78              print("temperature : %6.2f ℃" % temperature)
79              print("humidity      : %6.2f  %" % hum)
80              print("pressure      : %7.2f hPa" % pressure)
81              Temp_csv()
82              Hum_csv()
83              Press_csv()
84          except KeyboardInterrupt:
85              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: BME280_Class モジュールをインポート（import）
- 5 行目: 日付時刻取得用 datetime ライブラリをインポート
- 6 行目: csv ファイル読み書き用 csv ライブラリをインポート
- 9 行目: データ格納数定義
  csv ファイルに保存するデータ数を設定
- 11 行目: 日付・時刻取得
  現在の日付・時刻を取得するためには、
  datetime.date.today()
  datetime.datetime.now()
  を使用します。
  これらを使用するためには、datetime ライブラリのインポートが必要です（5 行目）。
- 14 行目～30 行目: 温度データをファイルへ書き込むプログラム Temp_csv()関数
- 16 行目: 温度データ格納用配列定義（50 行 6 列）
  西暦、月、日、時、分、データ:6 列
  50 行は、変数 MAXDATA で定義（9 行目）



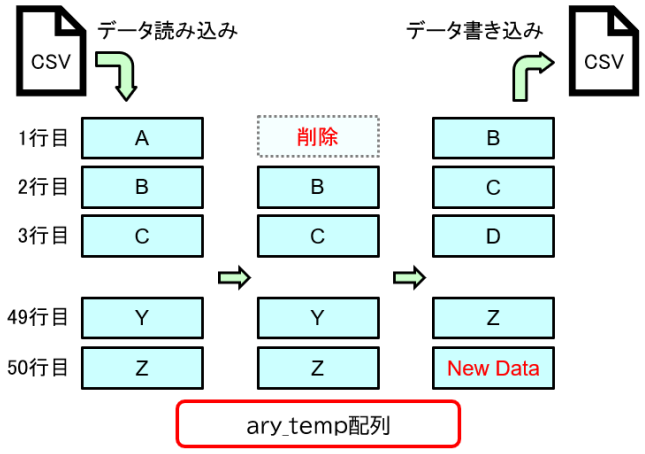
- 18 行目: csv ファイルを「読み取り専用」でオープン
- 19 行目: 先頭行読み取り

Copyright © 2024 YITJC All Rights Reserved.

- 20 行目～23 行目: csv ファイルのデータを 1 行ずつ読み取り ary_temp 配列に格納
- 24 行目: ary_temp 配列の先頭（最も古いデータ）を削除
- 25 行目: ary_temp 配列の末尾に日付、温度データを追加
- 26 行目: csv ファイルをクローズ
- 28 行目: csv ファイルを「書き込み専用」でオープン
- 29 行目: ary_temp 配列のデータを csv ファイルに書き込み
- 30 行目: csv ファイルをクローズ

  csv ファイルへのデータ読み込み・書き込みイメージを図に示します。

  最新のデータを記録する場合、先頭行（最も古いデータ）を削除してから最新のデータを末尾に追加します。



- 33 行目～49 行目: 湿度データを csv ファイルへ書き込むプログラム Hum_csv()関数
- 52 行目～68 行目: 気圧データを csv ファイルへ書き込むプログラム Press_csv()関数
- 70 行目～85 行目: main プログラム
- 71 行目～83 行目 :try ブロック

  モジュール（BME280_Class）で定義されているクラス（BME280）を読み込んで、温度・湿度・気圧のデータを表示、記録（csv ファイル）する。
- 72 行目: クラス生成

  4 行目でインポート（import）したモジュールからクラスを生成し、変数「bme」に代入しています。
- 73 行目: クラスで定義されている readData()関数の実行
- 74 行目: クラスで定義されている getTemperature()関数の実行し、温度データ（temperature）取得
- 75 行目: クラスで定義されている getHumidity()関数の実行し、湿度データ（hum）取得
- 76 行目: クラスで定義されている getPressure()関数の実行し、気圧データ（pressure）取得
- 78～80 行目: 温度、湿度、気圧データを print 表示
- 81 行目: 温度データをファイルへ書き込むプログラム Temp_csv()関数実行
- 82 行目: 湿度データを csv ファイルへ書き込むプログラム Hum_csv()関数実行
- 83 行目: 気圧データを csv ファイルへ書き込むプログラム Press_csv()関数実行
- 84 行目～85 行目 :except ブロック

  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

6.2 データ収集の自動化

6.1「データ収集」で使用したプログラム「25_CSV.py」は、プログラムを実行すると csv ファイルにデータが記録されます。このプログラムを定期的に実行することで、データ収集の自動化が実現します。定期的に実行する仕組みとして、第 2 章で学習した「Systemd」という仕組みを使用します。

手順①～Service ファイル作成～

ここで作成するファイルは、センサデータを csv ファイルに書き込むプログラム（25_CSV.py）を自動起動させるための Unit ファイル（.service ファイル）です。

まず、「/etc/systemd/system」ディレクトリへ移動します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

```
(ycuu) cuu@raspberrypi:~/dev $ cd ..
(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system
(ycuu) cuu@raspberrypi:/etc/systemd/system $
```

続いて、Unit（.service）ファイルを作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit（.service）ファイルの中身を書き込んでください。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano accumulation.service

```
  GNU nano 7.2                    accumulation.service *
[Unit]
Description=Data accumulation
After=network.target

[Service]
ExecStart=/home/cuu/dev/25_CSV.py
User=cuu
Group=cuu

[Install]
WantedBy=multi-user.target
```

25_CSV.pyを指定

※本講座で使用する Unit ファイルは、共有サイト「1_Python¥etc_systemd_system」に保存されています。

① VS Code で Unit ファイルを開きます

② 必要に応じて編集します

※このとき、VS Code の画面で改行コードが LF「↓」になっていることを確認します。

③ VS Code からコピーして、Tera Term の画面上で貼り付けます

Tera Term 画面上で「右クリック」すると次の画面が表示されるので、「OK」をクリックしてください。

④ Tera Term 画面上で上書き保存を実行します

・編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

・「Enter」キーを押下して上書き保存を実行

・「Ctrl + X」キーを押して元の画面に戻る

Copyright © 2024 YITJC All Rights Reserved.

～Unit（サービス）ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Data accumulation
3	After=network.target
4	
5	[Service]
6	ExecStart=/home/cuu/dev/25_CSV.py
7	User=cuu
8	Group=cuu
9	
10	[Install]
11	WantedBy=multi-user.target

灰色部は、これまでに説明した内容です。

- 1 行目～3 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文（動作に影響しませんが、分かりやすく記述するようにしてください。）
- 3 行目: 起動順序の指定
  network.target サービスの後（After）に起動する設定にしています。
- 5 行目～9 行目: Service セクション
- 6 行目: 自動起動するプログラムを指定 （25_CSV.py）
- 8 行目～9 行目: 実行する User 及び Group を指定
  ユーザー（User）及びユーザーが所属しているグループ（Group）は、次のコマンドで確認することできます。
  id のあとに本講座で使用しているユーザー名「cuu」を入力してください。
  (ycuu) cuu@raspberrypi:/etc/systemd/system $ id cuu

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ id cuu
uid=1000(cuu) gid=1000(cuu) groups=1000(cuu),4(adm),20(dialout),24(cdrom)
audio),44(video),46(plugdev),60(games),100(users),102(input),105(render),
95(spi),994(i2c),993(gpio),115(lpadmin)
```

- 11 行目～12 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 12 行目: サービス Unit の依存関係を設定
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

手順②～サービス再読み込み～

手順①で Unit ファイルを作成した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl daemon-reload

Copyright © 2024 YITJC All Rights Reserved.

手順③～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl enable accumulation.service

手順④～サービス開始～

次のコマンドを実行してサービスを開始します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl start accumulation.service

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.service
Created symlink /etc/systemd/system/multi-user.target.wants/accumulation.service → /etc/sys
temd/system/accumulation.service.
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.service
```

参考～サービスファイルの動作状況確認～

この状態でサービスファイル（.service ファイル）の動作状況を確認すると、次の画面が表示されます。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl status accumulation.service

「dead」状態になっている理由は、accumulation.service をタイマー制御するためのファイル「accumulation.timer」がまだ有効になっていない（作成していない）からです。

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.service
Created symlink /etc/systemd/system/multi-user.target.wants/accumulation.service → /etc/syst
emd/system/accumulation.service.
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.service
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status accumulation.service
○ accumulation.service - Data accumulation
     Loaded: loaded (/etc/systemd/system/accumulation.service; enabled; preset: enabled)
     Active: inactive (dead) since Tue 2024-05-14 14:25:49 JST; 5s ago
   Duration: 132ms
    Process: 2567 ExecStart=/home/cuu/dev/25_CSV.py (code=exited, status=0/SUCCESS)
   Main PID: 2567 (code=exited, status=0/SUCCESS)
        CPU: 110ms
```

➔ 次の手順で timer ファイルを作成します。

手順⑤～Timer ファイル作成～

手順①で作成した「accumulation.service」をタイマー制御するための Unit ファイル（.timer ファイル）を作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit ファイル（.timer ファイル）の中身を書き込んでください。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo nano accumulation.timer

Copyright © 2024 YITJC All Rights Reserved.

書き込んだ後は、「ctrl + O（オー）」、「Enter」を押して上書き保存してください。
「書き込みました」の表示がでたら、「Ctrl + X」を押して、元の画面に戻ります。

～Unit（タイマー）ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Accumulate data every 30 seconds
3	
4	[Timer]
5	OnUnitActiveSec=30sec
6	Unit=accumulation.service
7	
8	[Install]
9	WantedBy=multi-user.target

灰色部は、これまでに説明した内容です。

- 1 行目～2 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文（動作に影響しませんが、分かりやすく記述するようにしてください。）
- 4 行目～6 行目: timer セクション
  自動起動するファイル（accumulation.service）および
  起動日時（起動間隔）を指定するセクションです。
  対象のユニットが起動するまでの経過時間や間隔を指定します。
  - ✓ OnActiveSec=: タイマーが active になってから
  - ✓ OnBootSec=: システムがブートしてから
  - ✓ OnStartupSec=: systemd がスタートしてから
  - ✓ OnUnitActiveSec=: 対象のユニットが最後に起動になってから
  - ✓ OnUnitInactiveSec=: 対象のユニットが最後に停止になってから

  設定例（ブートしてから 1 時間 45 分後、その後は 2 ヶ月ごとに実行）
  [Timer]
  OnBootSec=1hour 45min
  OnUnitActiveSec=2month
- 5 行目: 起動間隔（実行間隔）の指定
- 6 行目: 自動起動するファイルの指定
- 8 行目～9 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 9 行目: サービス Unit の依存関係を設定
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～タイマー再読み込み～

手順⑤で Unit ファイル（.timer ファイル）を作成した後は、次のコマンドを実行してタイマーを再読み込みします。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl daemon-reload

手順⑦～タイマー自動起動有効化～

次のコマンドを実行してタイマーの自動起動を有効化します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl enable accumulation.timer

手順⑧～タイマー開始～

次のコマンドを実行してタイマーを開始します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl start accumulation.timer

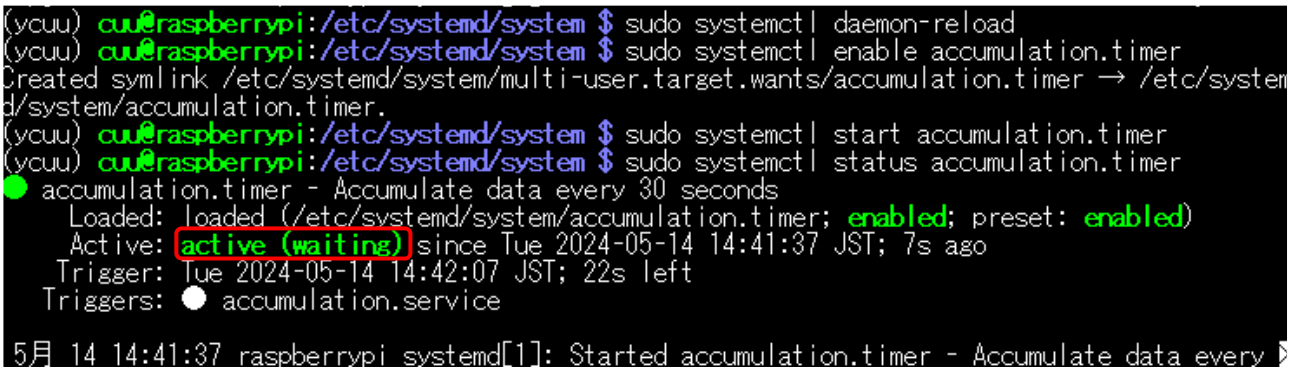
手順⑨～タイマー動作状況確認～

次のコマンドを実行してタイマーの動作状況を確認します。

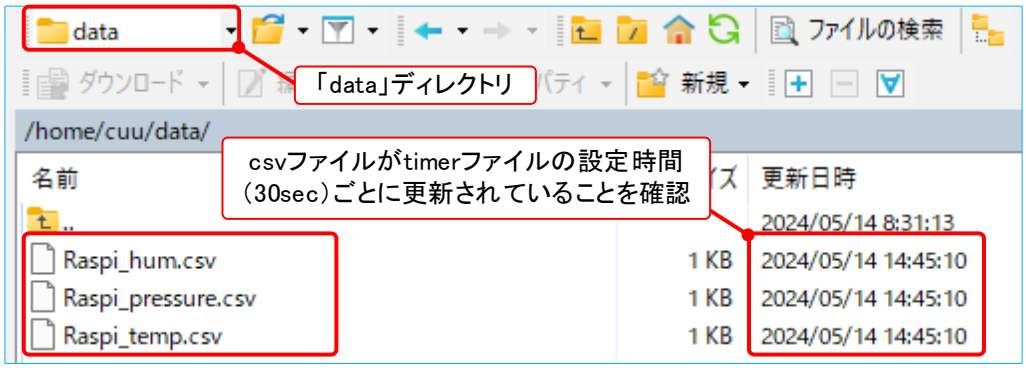
(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl status accumulation.timer

正常に稼働（自動起動）していれば、次の画面のようにステータスが active（waiting）と表示されます。

この画面表示を終了する場合は、「q」を押してください。



WinSCP を使用して csv ファイルにデータが書き込まれていることを確認してください。



注意～csv ファイルの確認～

WinSCP 画面上で csv ファイルを開かないでください。手順⑨で確認したとおり、Raspberry Pi は一定時間（30 秒）ごとに自動で温度、湿度、気圧データを csv ファイルに書き込んでいます。そのため、csv ファイルが開いた状態にある場合、データの書き込みエラーが発生します。

Copyright © 2024 YITJC All Rights Reserved.

参考～システムログの確認～

Tera Term でコマンドを入力してプログラムを実行（手動で実行）した場合、エラーの情報は画面に表示されます。

```
(ycuu) cuu@raspberrypi:~ $ ./25_CSV.py
-bash: ./25_CSV.py: そのようなファイルやディレクトリはありません
```

しかし、サービスファイル「.service」や「.timer」で自動起動しているプログラムは、Raspberry Pi が自動でプログラムを実行しているため、手動で実行したときと同じようにエラーの情報が表示されません。そのため、自動起動プログラムが正常に動作していない場合は、システムのログを確認する必要があります。

Raspberry Pi OS には、システムのログを収集・管理する「systemd-journald.service」があります。

```
(ycuu) cuu@raspberrypi:~ $ systemctl status systemd-journald.service
● systemd-journald.service - Journal Service
     Loaded: loaded (/lib/systemd/system/systemd-journald.service; static)
     Active: active (running) since Fri 2024-05-10 16:54:40 JST; 3 days ago
TriggeredBy: ● systemd-journald.socket
             ● systemd-journald-audit.socket
             ● systemd-journald-dev-log.socket
       Docs: man:systemd-journald.service(8)
```

「journalctl」コマンドは、「systemd-journald.service」が収集したログを表示するためのコマンドです。

次のコマンドを実行すると、「accumulation.service」のログを確認できます。

(ycuu) cuu @raspberrypi:~ $ journalctl -u accumulation.service -n 10

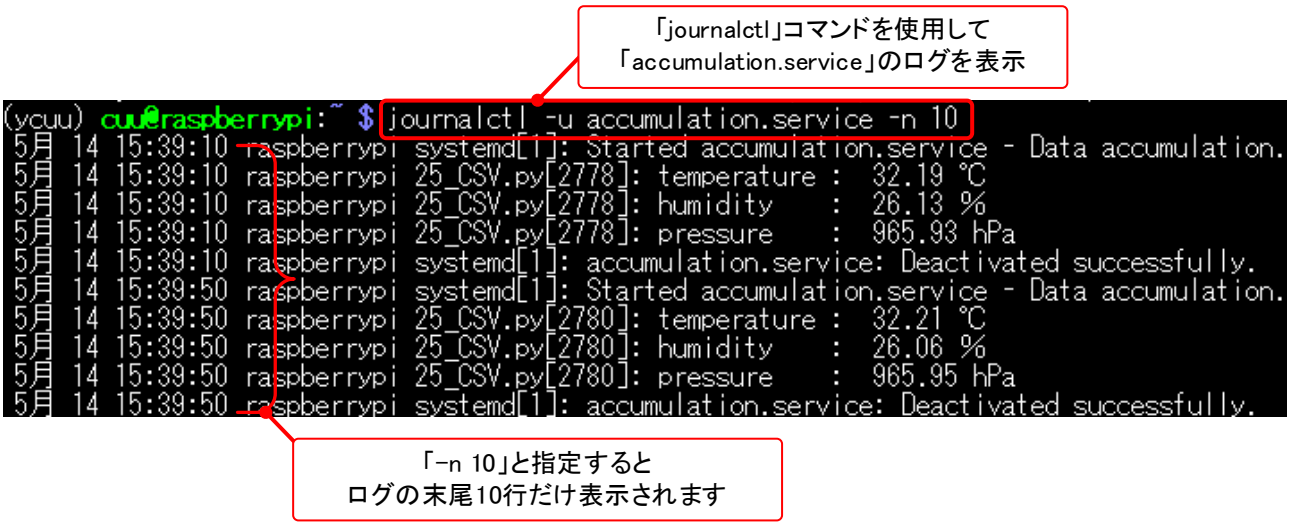
- **オプションの設定（例）**

「-u Unit 名」：指定した Unit のログを表示

「-n 数」：末尾（直近）から指定した行数のログを表示

「-f」：リアルタイム監視（デバッグに便利）

「-xe」：詳細情報を表示



ログを確認すると、「25_CSV.py」プログラムで温度、湿度、気圧のデータを正しく取得できている様子がわかります。このように、プログラムの中で「コメント」や「日付・時刻」を print 表示（記録）することで、システムの不具合解析に役立つ場合があります。そのため、「いつ、どのタイミングで、どのプログラム・コマンドが動作を開始し、終了したか」という情報を記録しておくことが大切です。

Copyright © 2024 YITJC All Rights Reserved.

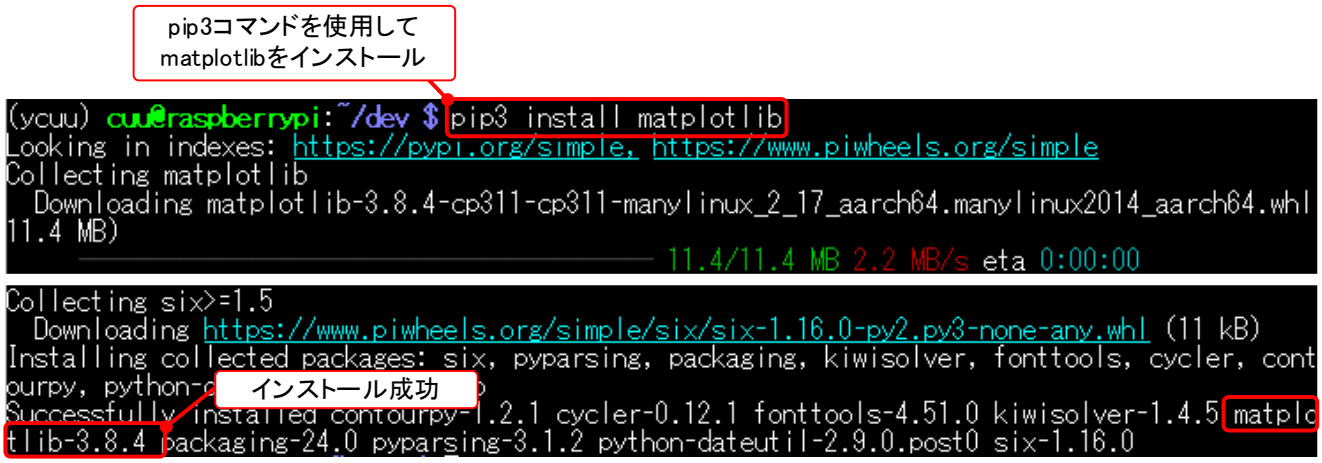
6.3 データ可視化

csv ファイルに自動収集したデータをグラフ化（可視化）する方法について学習します。

手順①～matplotlib インストール～

グラフを描画するためのライブラリ「matplotlib」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install matplotlib



～補足～

Matplotlib と同時に依存関係のあるパッケージもインストールされます。cycler（色や線種などのスタイルを定義）や kiwisolver（プロットレイアウトの最適化）など、グラフ生成に関連するライブラリがインストールされます。

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

26_GRAPH.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

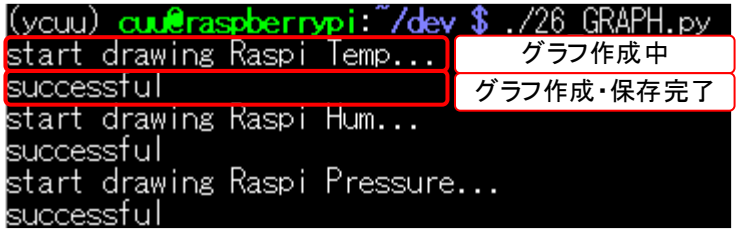
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 26_GRAPH.py

手順④～プログラム実行～

対象プログラムを実行します。

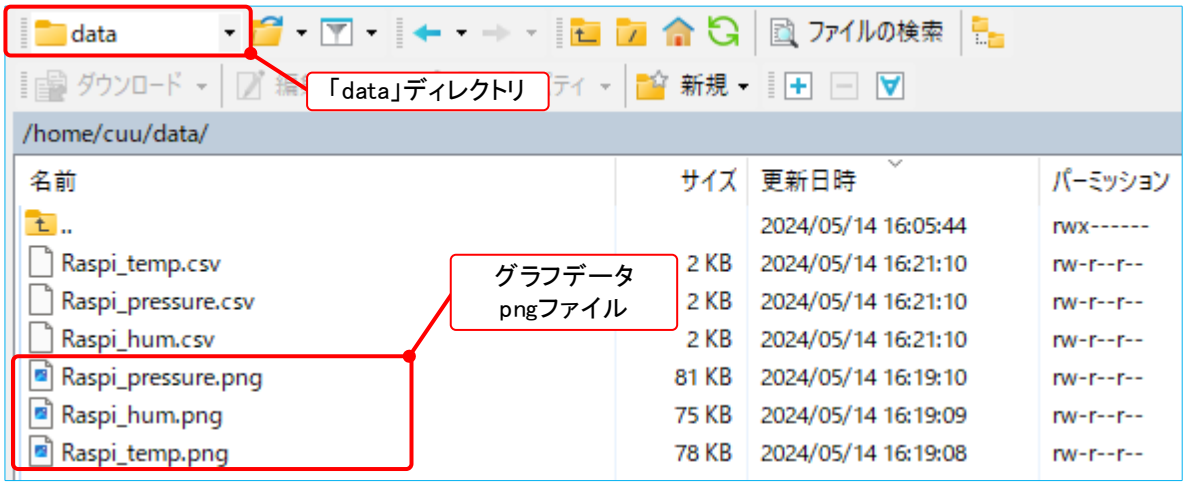
(ycuu) cuu@raspberrypi:~/dev $ ./26_GRAPH.py

プログラムを実行すると次の画面が表示され、csv ファイルのデータを使用してグラフが生成されます。



Copyright © 2024 YITJC All Rights Reserved.

グラフデータは、画像(png)ファイルとして「data」ディレクトリに保存されます。

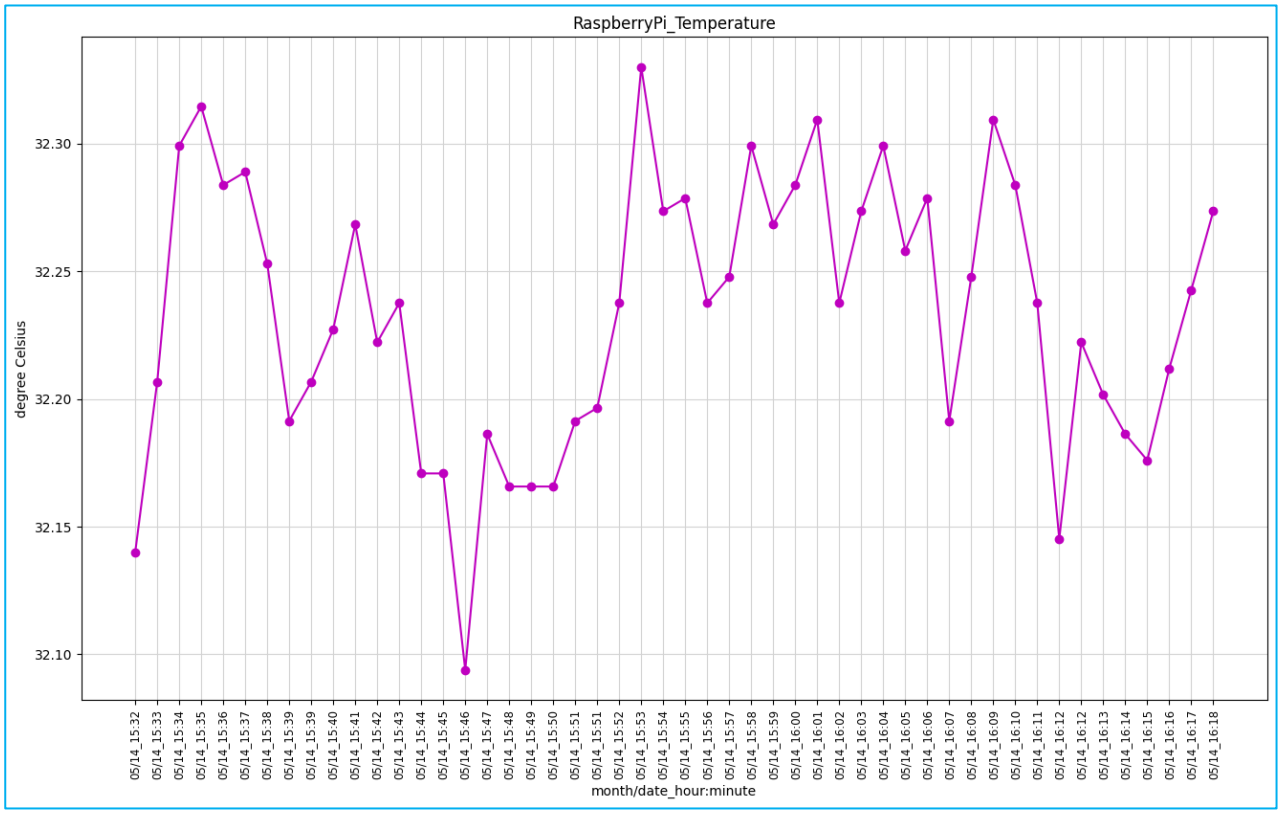


WinSCP を使用してグラフデータ(png ファイル)を Windows PC のデスクトップにドラッグ&ドロップしてください。そして、Windows PC 上でグラフデータを開いて確認してください。

RaspberryPi_Temp のグラフ

X 軸: csv ファイルに記録されている日付・時刻データ

Y 軸: csv ファイルに記録されている温度データ



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	#グラフ表示
5	import matplotlib
6	matplotlib.use("Agg")
7	import matplotlib.pyplot as plt
8	import csv
9	
10	#データ格納数定義
11	MAX_RANGE=50
12	#csv・グラフデータ保存先
13	URL_PATH="/home/cuu/data/"
14	
15	#Raspi_temp
16	def drawingTemp():
17	temp=[0 for i in range(MAX_RANGE)]
18	date=[0 for i in range(MAX_RANGE)]
19	fp=open(URL_PATH+"Raspi_temp.csv","r")
20	reader=csv.reader(fp)
21	for row in reader:
22	temp.pop(0)
23	temp.append(float(row[5]))
24	date.pop(0)
25	#month/date_hour:minute
26	date.append("%02d"%int(row[1])+"/"+"%02d"%int(row[2])+"_"+"%02d"%int(row[3])+":"+"%02d"%int(row[4]))
27	fp.close()
28	
29	fig=plt.figure(figsize=(16,9))
30	ax=fig.add_subplot(1,1,1)
31	plt.title("RaspberryPi_Temperature")
32	plt.gca().get_yaxis().get_major_formatter().set_useOffset(False)
33	plt.xlabel("month/date_hour:minute")
34	plt.ylabel("degree Celsius")
35	plt.plot(temp,color="m",marker='o')
36	plt.xticks(range(MAX_RANGE),date)
37	plt.grid(which='major',color='lightgrey',linestyle='-')

Copyright © 2024 YITJC All Rights Reserved.

```
38        labels=ax.set_xticklabels(date,rotation=90,fontsize="small")
39        plt.savefig(URL_PATH+"Raspi_temp.png",bbox_inches="tight")
40
41    #Raspi_hum
42    def drawingHum():
43        hum=[0 for i in range(MAX_RANGE)]
44        date=[0 for i in range(MAX_RANGE)]
45        fp=open(URL_PATH+"Raspi_hum.csv","r")
46        reader=csv.reader(fp)
47        for row in reader:
48            hum.pop(0)
49            hum.append(float(row[5]))
50            date.pop(0)
51            #month/date_hour:minute
52            date.append("%02d"%int(row[1])+"/"+"%02d"%int(row[2])+"_"+"%02d"%int(row[3])+":"+"%02d"%int(row[4]))
53        fp.close()
54
55        fig=plt.figure(figsize=(16,9))
56        ax=fig.add_subplot(1,1,1)
57        plt.title("RaspberryPi_Humidity")
58        plt.gca().get_yaxis().get_major_formatter().set_useOffset(False)
59        plt.xlabel("month/date_hour:minute")
60        plt.ylabel("percent")
61        plt.plot(hum,color="g",marker='s')
62        plt.xticks(range(MAX_RANGE),date)
63        plt.grid(which='major',color='lightgrey',linestyle='-')
64        labels=ax.set_xticklabels(date,rotation=90,fontsize="small")
65        plt.savefig(URL_PATH+"Raspi_hum.png",bbox_inches="tight")
66
67    #Raspi_pressure
68    def drawingPressure():
69        pressure=[0 for i in range(MAX_RANGE)]
70        date=[0 for i in range(MAX_RANGE)]
71        fp=open(URL_PATH+"Raspi_pressure.csv","r")
72        reader=csv.reader(fp)
73        for row in reader:
74            pressure.pop(0)
75            pressure.append(float(row[5]))
76            date.pop(0)
```

Copyright © 2024 YITJC All Rights Reserved.

```
77              #month/date_hour:minute
78              date.append("%02d"%int(row[1])+"/"+"%02d"%int(row[2])+"_"+"%02d"%int(row[3])+":"+"%02d"%int(row[4]))
79          fp.close()
80
81          fig=plt.figure(figsize=(16,9))
82          ax=fig.add_subplot(1,1,1)
83          plt.title("RaspberryPi_Pressure")
84          plt.gca().get_yaxis().get_major_formatter().set_useOffset(False)
85          plt.xlabel("month/date_hour:minute")
86          plt.ylabel("Hectopascals")
87          plt.plot(pressure,color="c",marker='^')
88          plt.xticks(range(MAX_RANGE),date)
89          plt.grid(which='major',color='lightgrey',linestyle='-')
90          labels=ax.set_xticklabels(date,rotation=90,fontsize="small")
91          plt.savefig(URL_PATH+"Raspi_pressure.png",bbox_inches="tight")
92
93  if __name__ == '__main__':
94          try:
95              print("start drawing Raspi Temp...")
96              drawingTemp()
97              print("successful")
98              print("start drawing Raspi Hum...")
99              drawingHum()
100             print("successful")
101             print("start drawing Raspi Pressure...")
102             drawingPressure()
103             print("successful")
104         except KeyboardInterrupt:
105             pass
```

Copyright © 2024 YITJC All Rights Reserved.

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 5 行目: グラフを描画するためのライブラリ matplotlib をインポート（import）
- 6 行目: バックエンドを「Agg」に設定

～補足～

○バックエンドは、グラフの描画先や管理方法を指します。具体的には、画面上にプロット表示する方法やファイルに保存する方法などを決定します。

○「Agg」は、Anti-Grain Geometry の略称で、グラフ画像を生成するために使用します。

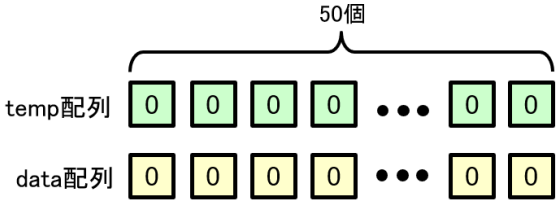
本講座の開発環境 CUI（Character User Interface）は、Tera Term を使用してコマンドで Raspberry Pi を操作しています。そのため、グラフ（画像）をディスプレイ上に描画できません。しかし、「Agg」を使用することで、グラフを画像データに変換・保存できるため、CUI 環境でもグラフを効果的に利用できます。

～参考～

Matplotlib のバックエンドとして指定可能な renderer 一覧（Matplotlib Usage Guide）

Renderer	Filetypes	Description
AGG	png	raster graphics -- high quality images using the Anti-Grain Geometry engine
PDF	pdf	vector graphics -- Portable Document Format
PS	ps, eps	vector graphics -- Postscript output
SVG	svg	vector graphics -- Scalable Vector Graphics
PGF	pgf, pdf	vector graphics -- using the pgf package
Cairo	png, ps, pdf, svg	raster or vector graphics -- using the Cairo library

- 7 行目: matplotlib ライブラリから pyplot クラスをインポート（import）して「plt」と定義
- 8 行目: csv ファイル読み書き用 csv ライブラリをインポート
- 11 行目: データ格納数を定義
  csv ファイルに保存されているデータ数を設定
- 13 行目: データ保存先を定義
  csv ファイルが保存されているディレクトリを指定
  グラフ画像（png ファイル）の保存先ディレクトリとして指定
- 16 行目～42 行目: BME280 温度データグラフ作成プログラム drawingTemp()関数
- 17 行目: 温度データを格納するための配列 temp[]を初期化
- 18 行目: 日付データを格納するための配列 data[]を初期化
  temp[]、data[]ともに 50 個の要素（初期値 0）で構成される配列



Copyright © 2024 YITJC All Rights Reserved.

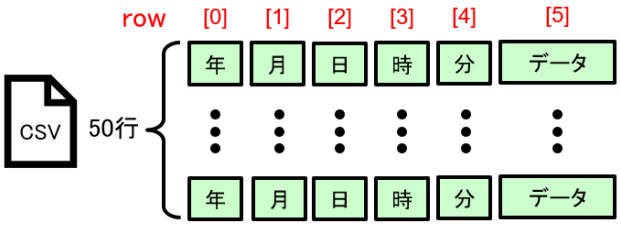
- 19 行目: 温度データが保存されている csv ファイルを開いてファイルオブジェクト「fp」を定義
- 20 行目: csv ファイルを読み取る reader オブジェクト「reader」を定義
- 21 行目～26 行目: csv ファイルから読み取った値を temp[]、data[]配列に格納

  ○X 軸（date）: month/date_hour:minute（csv ファイルから読み取った月日・時刻）

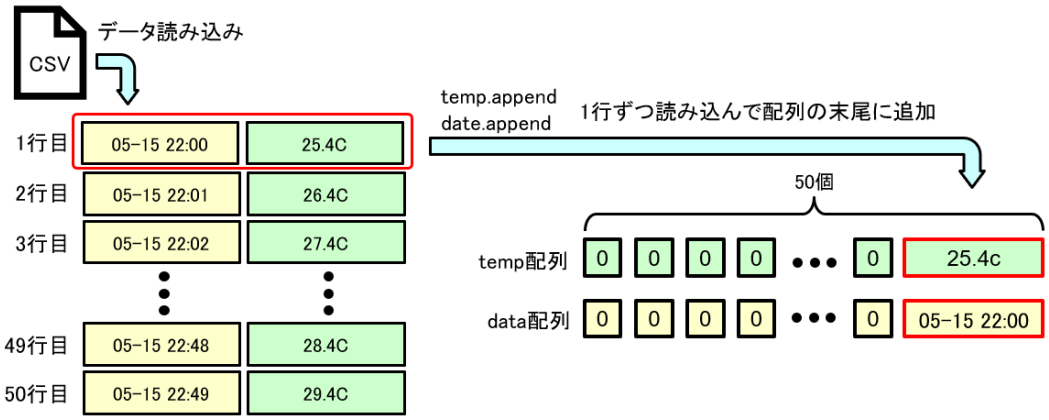
  日時データ（row[1]～row[4]）を data 配列の末尾に追加

  ○Y 軸（temp）: degree Celsius（csv ファイルから読み取った温度データ）

  温度データ（row[5]）を temp 配列の末尾に追加



csv ファイルから 50 行分のデータを読み取って、配列に格納する流れを図に示します。

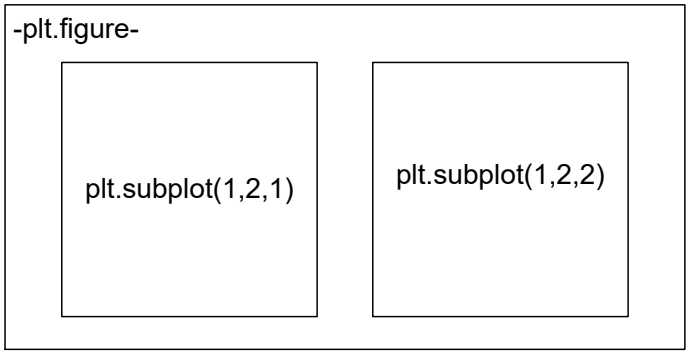


- 29 行目～39 行目: グラフ描画設定
- 29 行目: figure の縦・横サイズ設定
- 30 行目: figure 内の配置設定

～補足～

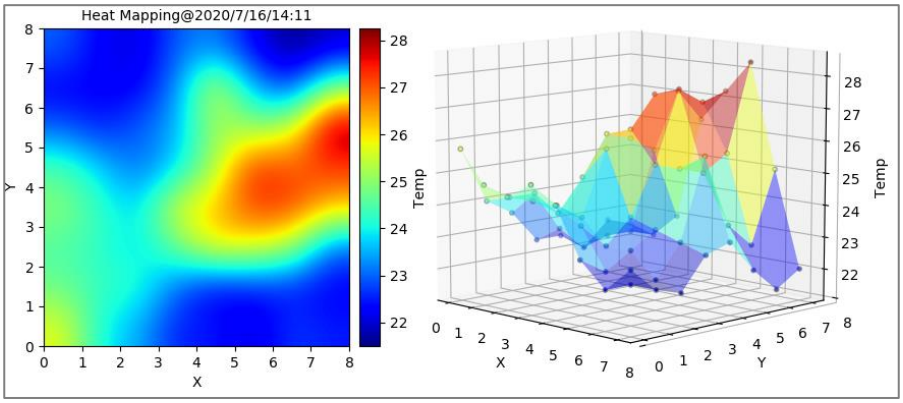
fig.add_subplot(1,1,1)は、1 行目 1 列の 1 番目に配置するという意味

例）1 行 2 列の 1 番目、2 番目にグラフを配置



Copyright © 2024 YITJC All Rights Reserved.

この時の実行結果は、次のようにグラフが表示されます。



- 31 行目:グラフタイトル
- 32 行目: y 軸の数値にオフセットを使わずに表現
- 33 行目: x 軸ラベル
- 34 行目: y 軸ラベル
- 35 行目: 温度データ temp をプロット、マーカ色、マーカ種類を設定
- 36 行目: x 軸目盛に変数 data の値を設定
- 37 行目: 主目盛線のグリッド線を設定
- 38 行目: x 軸目盛を 90° 回転して配置
- 39 行目: 作成したグラフデータを png ファイル形式で保存
- 42 行目～65 行目: BME280 湿度データグラフ作成プログラム drawingHum()関数
- 68 行目～91 行目: BME280 気圧データグラフ作成プログラム drawingPressure()関数
- 93 行目～105 行目: main プログラム
- 94 行目～103 行目:try ブロック

  BME280 温度、湿度、気圧データのグラフ作成
- 96 行目: 温度データグラフ作成プログラム drawingTemp()関数実行
- 99 行目: 湿度データグラフ作成プログラム drawingHum()関数実行
- 102 行目: 気圧データグラフ作成プログラム drawingPressure()関数実行
- 104 行目～105 行目:except ブロック

  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

6.4 データ分析

収集したデータの特徴や傾向を瞬時に把握するためには、データ分析が重要です。データ分析を行うことで、収集したデータを効果的に利用できます。例えば、収集したデータを分析し、データの最大値、最小値、平均値などを算出し、予め設定した閾値と比較することで異常値を瞬時に検出できます。また、特定の機械や生産ライン、生産期間ごとの不良率から品質低下の原因分析を行うことで、品質の改善効果が期待できます。

ここでは、Python のデータ分析ライブラリ「pandas（パンダス）」を使用してデータ分析する方法について学習します。

手順①～pandas インストール～

csv データの分析を行うため、Python のデータ分析用ライブラリ pandas をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install pandas

～補足～

Pandas と同時に依存関係のあるパッケージもインストールされます。pytz（タイムゾーンを扱うライブラリ）と tzdata（タイムゾーンデータベースを提供するライブラリ）がインストールされます。

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

27_csvData.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 27_csvData.py

Copyright © 2024 YITJC All Rights Reserved.

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./27_csvData.py

BME280 センサのデータ（温度、湿度、気圧）の分析結果（最大、最小、平均の 3 つの値）が表示されます。

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import pandas as pd
5	
6	#csvデータ保存先
7	URL_PATH="/home/cuu/data/"
8	
9	def temp_analysis():
10	df = pd.read_csv(URL_PATH+"Raspi_temp.csv", names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
11	tmax=df['temperature'].max()
12	tmin=df['temperature'].min()
13	tmean=df['temperature'].mean()
14	print ('最高温度:'+ '{:.3f}'.format(tmax))
15	print ('最低温度:'+ '{:.3f}'.format(tmin))
16	print ('平均温度:'+ '{:.3f}'.format(tmean))
17	
18	def hum_analysis():
19	df = pd.read_csv(URL_PATH+"Raspi_hum.csv", names=['year', 'month', 'day', 'hour', 'minute', 'hum'])
20	hmax=df['hum'].max()
21	hmin=df['hum'].min()
22	hmean=df['hum'].mean()
23	print ('最高湿度:'+ '{:.3f}'.format(hmax))

Copyright © 2024 YITJC All Rights Reserved.

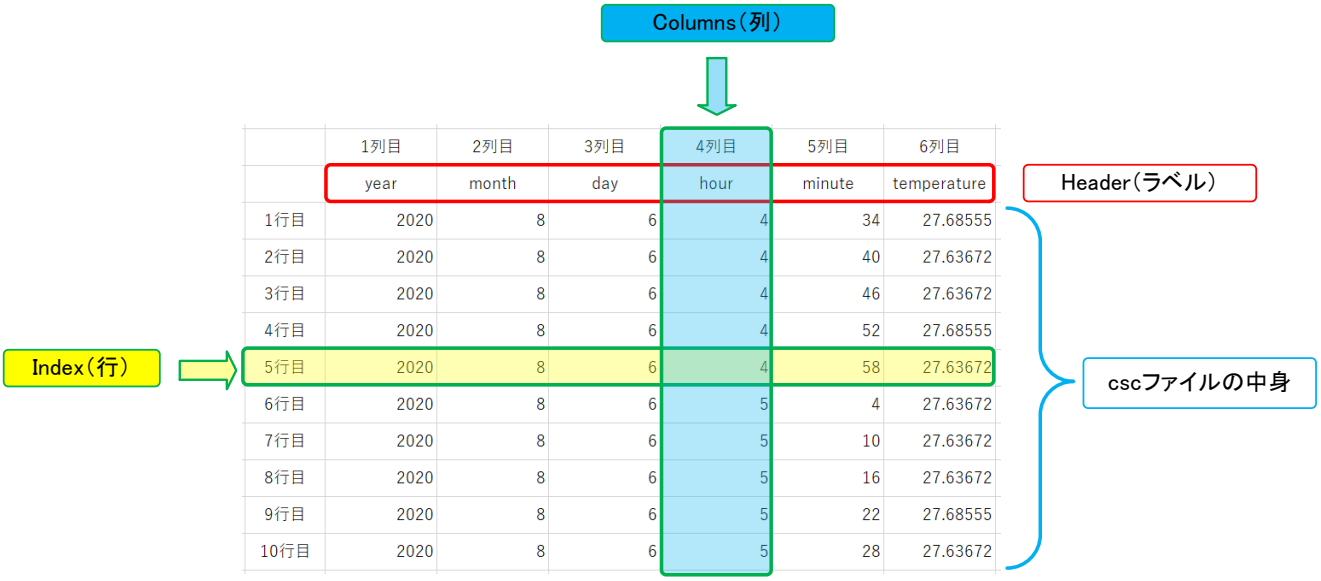
```
24        print ('最低湿度:'+ '{:.3f}'.format(hmin))
25        print ('平均湿度:'+ '{:.3f}'.format(hmean))
26
27  def press_analysis():
28        df = pd.read_csv(URL_PATH+"Raspi_pressure.csv", names=['year', 'month', 'day', 'hour', 'minute', 'pressure'])
29        pmax=df['pressure'].max()
30        pmin=df['pressure'].min()
31        pmean=df['pressure'].mean()
32        print ('最高気圧:'+ '{:.3f}'.format(pmax))
33        print ('最低気圧:'+ '{:.3f}'.format(pmin))
34        print ('平均気圧:'+ '{:.3f}'.format(pmean))
35
36  if __name__ == '__main__':
37        try:
38              print('温度データ分析結果')
39              temp_analysis()
40              print('湿度データ分析結果')
41              hum_analysis()
42              print('気圧データ分析結果')
43              press_analysis()
44        except KeyboardInterrupt:
45              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: データ分析を行うためのライブラリ pandas をインポート（import）
- 7 行目: csv データ保存先定義
  csv ファイルが保存されているディレクトリを指定
- 9 行目～16 行目: BME280 温度データ分析プログラム temp_analysis()関数
- 10 行目: csv データ読み取り
  Header に名前を付けるため、names で定義
  names=['year', 'month', 'day', 'hour', 'minute', 'temperature']
  ここで設定した名前を使用して、データの分析を行います。
- 11 行目: csv ファイルの温度データ"temperature"から最大値を取得
- 12 行目: csv ファイルの温度データ"temperature"から最小値を取得
- 13 行目: csv ファイルの温度データ"temperature"から平均値を取得

Copyright © 2024 YITJC All Rights Reserved.

Pandas にはさまざまなメソッドが用意されていて、データフレーム(csv ファイル)のラベル(Header に付けた名前)を指定することで、必要な値を取得することができます。
データフレームの見方を紹介します。



	1列目	2列目	3列目	4列目	5列目	6列目
	year	month	day	hour	minute	temperature
1行目	2020	8	6	4	34	27.68555
2行目	2020	8	6	4	40	27.63672
3行目	2020	8	6	4	46	27.63672
4行目	2020	8	6	4	52	27.68555
5行目	2020	8	6	4	58	27.63672
6行目	2020	8	6	5	4	27.63672
7行目	2020	8	6	5	10	27.63672
8行目	2020	8	6	5	16	27.63672
9行目	2020	8	6	5	22	27.68555
10行目	2020	8	6	5	28	27.63672

メソッドを使用する際は、
DataFrame[ラベル名].メソッド名
と記述して使用します。

他に用意されているメソッドとして、

- ✓ Count: 要素の個数を取得
- ✓ Unique: ユニーク(重複しないデータ)要素の個数を取得
- ✓ Top: 最頻値
- ✓ Freq: 最頻値の出現回数
- ✓ Std: 標準偏差
- ✓ Median: 中央値(50%)

などがあります。

- 14 行目～16 行目: 最大・最小・平均値を print 表示
- 18 行目～25 行目: BME280 湿度データ分析プログラム hum_analysis()関数
- 27 行目～34 行目: BME280 気圧データ分析プログラム press_analysis()関数
- 36 行目～45 行目: main プログラム
- 37 行目～43 行目:try ブロック
  BME280 温度、湿度、気圧データの分析
- 39 行目: 温度データ分析プログラム temp_analysis()関数実行
- 41 行目: 湿度データ分析プログラム hum_analysis()関数実行
- 43 行目: 気圧データ分析プログラム press_analysis()関数実行

Copyright © 2024 YITJC All Rights Reserved.

- 44 行目～45 行目：except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

6.5 ヒストグラム作成

生産現場では、工程能力や製品品質などを可視化するために「ヒストグラム」が使用されます。ヒストグラムからは、データの中心傾向や散布度（ばらつき具合）、異常値の検出などの情報が得られるため、生産工程や製品品質を改善するために重要な情報になります。

ここでは、グラフを描画するためのライブラリ「matplotlib」とデータ分析ライブラリ「pandas」を使用してヒストグラムを作成する方法について学習します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

28_Histogram.py

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 28_Histogram.py
```

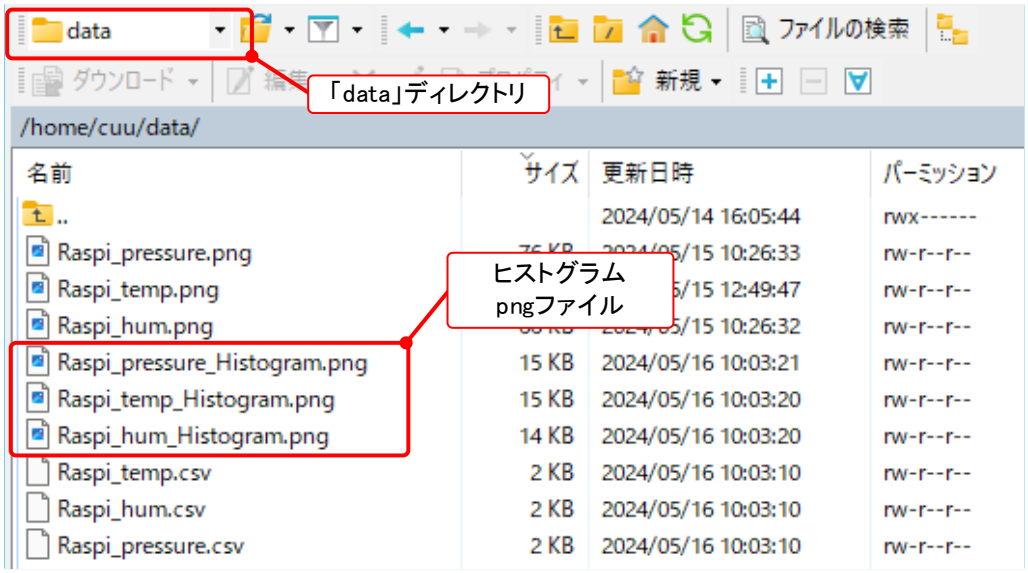
手順③～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./28_Histogram.py
```

プログラムを実行すると次の画面が表示され、csv ファイルのデータを使用してヒストグラムが生成されます。

グラフデータは、画像（png）ファイルとして「data」ディレクトリに保存されます。



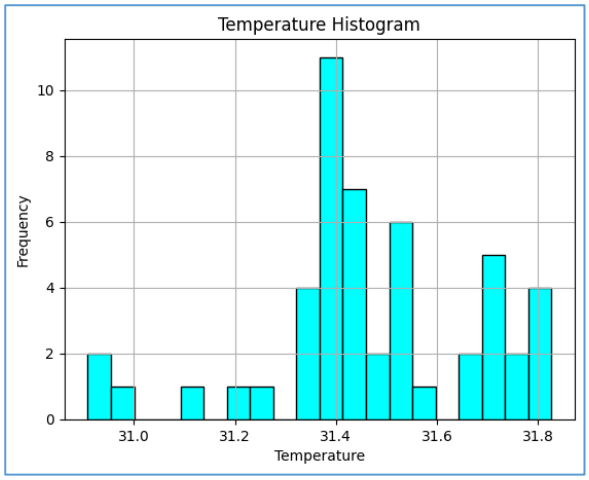
Copyright © 2024 YITJC All Rights Reserved.

WinSCP を使用してヒストグラムの画像データ（png ファイル）を Windows PC のデスクトップにドラッグ&ドロップしてください。そして、Windows PC 上でグラフデータを開いて確認してください。

RaspberryPi Temp のグラフ

X 軸: 温度データの「階級」

Y 軸: 温度データの「度数」



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	#グラフ表示
5	import matplotlib
6	matplotlib.use("Agg")
7	import matplotlib.pyplot as plt
8	import csv
9	import pandas as pd
10	
11	#Histogramデータ保存先
12	URL_PATH="/home/cuu/data/"
13	
14	#Raspi_temp
15	def histogram_Temp():
16	df = pd.read_csv(URL_PATH+"Raspi_temp.csv", names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
17	
18	plt.hist(df['temperature'], bins=20, color='cyan', edgecolor='black')
19	plt.xlabel('Temperature')
20	plt.ylabel('Frequency')

Copyright © 2024 YITJC All Rights Reserved.

```
21        plt.title('Temperature Histogram')
22        plt.grid(True)
23        plt.savefig(URL_PATH+"Raspi_temp_Histogram.png",bbox_inches="tight")
24        plt.close()
25
26    #Raspi_hum
27    def histogram_Hum():
28        df = pd.read_csv(URL_PATH+"Raspi_hum.csv", names=['year', 'month', 'day', 'hour', 'minute', 'hum'])
29
30        plt.hist(df['hum'], bins=20, color='magenta', edgecolor='black')
31        plt.xlabel('Humidity')
32        plt.ylabel('Frequency')
33        plt.title('Humidity Histogram')
34        plt.grid(True)
35        plt.savefig(URL_PATH+"Raspi_hum_Histogram.png",bbox_inches="tight")
36        plt.close()
37
38    #Raspi_pressure
39    def histogram_Press():
40        df = pd.read_csv(URL_PATH+"Raspi_pressure.csv", names=['year', 'month', 'day', 'hour', 'minute', 'pressure'])
41
42        plt.hist(df['pressure'],bins=20, color='yellow', edgecolor='black')
43        plt.xlabel('Pressure')
44        plt.ylabel('Frequency')
45        plt.title('Pressure Histogram')
46        plt.grid(True)
47        plt.savefig(URL_PATH+"Raspi_pressure_Histogram.png",bbox_inches="tight")
48        plt.close()
49
50    if __name__ == '__main__':
51        try:
52            print("start drawing Histogram Raspi Temp...")
53            histogram_Temp()
54            print("successful")
55            print("start drawing Histogram Raspi Hum...")
56            histogram_Hum()
57            print("successful")
58            print("start drawing Histogram Raspi Pressure...")
59            histogram_Press()
```

Copyright © 2024 YITJC All Rights Reserved.

```
60              print("successful")
61          except KeyboardInterrupt:
62              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 5 行目: グラフを描画するためのライブラリ matplotlib をインポート（import）
- 6 行目: バックエンドを「Agg」に設定
- 7 行目: matplotlib ライブラリから pyplot クラスをインポート（import）して「plt」と定義
- 8 行目: csv ファイル読み書き用 csv ライブラリをインポート
- 9 行目: データ分析を行うためのライブラリ pandas をインポート（import）
- 11 行目: データ保存先を定義
  ヒストグラムデータ（png ファイル）の保存先ディレクトリとして指定
- 15 行目～24 行目: BME280 温度データヒストグラム作成プログラム histogram_Temp()関数
- 16 行目: csv データ読み取り
  Header に名前を付けるため、names で定義
  names=['year', 'month', 'day', 'hour', 'minute', 'temperature']
  ここで設定した名前（'temperature'）を使用して、ヒストグラムを作成します。
- 18 行目: 温度データのヒストグラムを作成
  matplotlib の hist()関数を使用してヒストグラムを作成しています。
  plt.hist(df['temperature'], bins=20, color='cyan', edgecolor='black')
  ○ヒストグラムを作成するためのデータを指定 :df['temperature']
  ○ヒストグラムのビン（bins）の数を指定 :bins=20
  ～補足～
  ヒストグラムの階級（棒）をビン（bins）と呼びます。ここでは、階級（棒）の数を「20」に設定しているので、温度データを 20 の階級に区切ってヒストグラムを作成します。
  ○ヒストグラムの色を指定 :color='cyan'
  ○ヒストグラム（棒）の境界線の色を指定 :edgecolor='black'
- 19 行目: x 軸ラベルを定義
- 20 行目: y 軸ラベルを定義
- 21 行目: ヒストグラムタイトルを定義
- 22 行目: グリッド線を設定
- 23 行目: 作成したヒストグラムを png ファイル形式で保存
- 24 行目: ヒストグラムをクローズ
- 27 行目～36 行目: 湿度データヒストグラム作成プログラム histogram_Hum()関数
- 39 行目～48 行目: 気圧データヒストグラム作成プログラム histogram_Press()関数

Copyright © 2024 YITJC All Rights Reserved.

- 50 行目～62 行目: main プログラム
- 51 行目～60 行目 :try ブロック

  BME280 温度、湿度、気圧データのヒストグラム作成
- 53 行目: 温度データヒストグラム作成プログラム histogram_Temp()関数実行
- 56 行目: 湿度データヒストグラム作成プログラム histogram_Hum()関数実行
- 59 行目: 気圧データヒストグラム作成プログラム histogram_Press()関数実行
- 61 行目～62 行目 :except ブロック

  KeyboardInterrupt 例外を定義しています。

  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。

  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

6.6 データ抽出・連結

csv ファイルに記録されたデータを有効活用するため、データの抽出・連結について学習します。

例えば、csv ファイルに記録されている時刻データの抽出・連結によりデータの記録範囲を示すことができます。

```
11/01_07:58から11/01_09:12のデータ
```

また、最高値および最高値が記録された時刻データの抽出・連結により、次のような情報を共有できます。

```
最高温度:21.631 :11/01_09:15に記録
```

手順①～プログラム配置～

対象プログラムを「dev」フォルダにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

29_csvData.py

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 29_csvData.py

手順③～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./29_csvData.py

次のデータが表示されることを確認してください。

プログラムを実行すると、「data」ディレクトリに保存されている csv データ(Raspi_temp.csv)を読み取り、画面上に抽出・連結したデータを print 表示します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./29_csvData.py
   month  day  hour  minute
0      5   16    13      31
    month  day  hour  minute
49      5   16    14      18
    year  month  day  hour  minute  temperature
49  2024      5   16    14      18    33.104849
最高温度:33.105
05/16_13:31
05/16_14:18
05/16_13:31から05/16_14:18のデータ
05/16_14:18
最高温度:33.105 :05/16_14:18に記録
```

1行目データ(最古データ)

50行目データ(最新データ)

最高値を記録したときのデータ

最高値

データ記録開始時刻
データ記録終了時刻

開始時刻から終了時刻

最高値を記録した時刻

最高値と最高値を記録した時刻

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

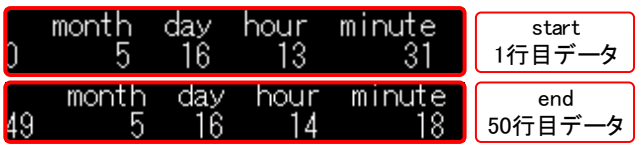
行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import pandas as pd
5	
6	#csvデータ保存先
7	URL_PATH="/home/cuu/data/"
8	
9	def Raspi_temp():
10	df = pd.read_csv(URL_PATH+"Raspi_temp.csv", names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
11	#最高値取得
12	max=df['temperature'].max()
13	#1行目2列～5列のデータ抽出
14	start=df.iloc[0:1,1:5]
15	#50行目2列～5列のデータ抽出
16	end=df.iloc[49:50,1:5]
17	#抽出したデータpint表示
18	print (start)
19	print (end)
20	
21	#最高値を記録した時のデータをprint表示
22	print (df[df['temperature'] == max])
23	#最高値print表示
24	print ('最高温度:'+ '{:.3f}'.format(max))
25	
26	#記録開始時刻抽出(1行目データ)
27	sday="%02d"%int(df.loc[0, 'month'])+"/"+"%02d"%int(df.loc[0, 'day'])+"_"+¥
28	"%02d"%int(df.loc[0, 'hour'])+":"+"%02d"%int(df.loc[0, 'minute'])
29	#記録開始時刻print表示
30	print (sday)
31	
32	#記録終了時刻抽出(50行目データ)
33	eday="%02d"%int(df.loc[49, 'month'])+"/"+"%02d"%int(df.loc[49, 'day'])+"_"+¥
34	"%02d"%int(df.loc[49, 'hour'])+":"+"%02d"%int(df.loc[49, 'minute'])
35	#記録終了時刻print表示
36	print (eday)
37	

Copyright © 2024 YITJC All Rights Reserved.

```
38      #データ記録時間(範囲)print表示
39      print (sday + 'から' + eday + 'のデータ')
40
41      #最高値を記録したデータ(行)抽出
42      tmax=df[df['temperature'] == max]
43      #最高値を記録したデータ(行)から日付・時刻抽出
44      mday="%02d"%int(tmax.loc[tmax.index[0], 'month'])+"/"+"%02d"%int(tmax.loc[tmax.index[0], 'day'])+"_"+¥
45      "%02d"%int(tmax.loc[tmax.index[0], 'hour'])+":"+"%02d"%int(tmax.loc[tmax.index[0], 'minute'])
46      #最高値を記録した日付・時刻print表示
47      print (mday)
48      #最高値記録データprint表示
49      print ('最高温度:' + '{:.3f}'.format(max) + ' :' + mday + 'に記録')
50
51  if __name__ == '__main__':
52      try:
53          Raspi_temp()
54      except KeyboardInterrupt:
55          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: データ分析を行うためのライブラリ pandas をインポート（import）
- 7 行目: csv データ保存先定義
  csv ファイルが保存されているディレクトリを指定
- 9 行目～49 行目: BME280 温度データ抽出・連結プログラム Raspi_temp()関数
- 10 行目: csv データ読み取り
  Header に名前を付けるため、names で定義
  names=['year', 'month', 'day', 'hour', 'minute', 'temperature']
  ここで設定した名前を使用して、データの抽出・連結を行います。
- 12 行目: csv ファイルの温度データ”temperature”から最大値を取得
- 14 行目: 1 行目 2 列～5 列のデータ抽出
- 16 行目: 50 行目 2 列～5 列のデータ抽出
- 17 行目～19 行目: 抽出したデータの print 表示



Copyright © 2024 YITJC All Rights Reserved.

- 22 行目: 最高値を記録した時のデータ行を抽出・print 表示
  12 行目で定義した変数 max(csv ファイル内の temperature データの最大値)を使用して、
  temperature データが最高値(max)を記録した行のデータを抽出

```
    year  month  day  hour  minute  temperature
49  2024      5   16    14      18    33.104849
```

- 24 行目: 最高値 print 表示

```
最高温度:33.105
```

- 27 行目～28 行目: 記録開始時刻抽出
  記録開始、つまり csv ファイルの 1 行目の時刻データを抽出
- 30 行目: 記録開始時刻 print 表示

```
05/16 13:31
```

- 33 行目～34 行目: 記録開始時刻抽出
  記録開始、つまり csv ファイルの 50 行目の時刻データを抽出
- 36 行目: 記録開始時刻 print 表示

```
05/16 14:18
```

- 39 行目: 記録開始時刻と記録終了時刻を連結して print 表示

```
05/16 13:31から05/16 14:18のデータ
```

- 42 行目: temperature データが最高値(max)を記録した行のデータを抽出
- 44 行目～45 行目: temperature データが最高値を記録したデータ(行)から日付・時刻抽出
- 47 行目: 最高値を記録した日付・時刻を print 表示

```
05/16 14:18
```

- 49 行目: 最高値と最高値を記録した時刻を連結して print 表示

```
最高温度:33.105 :05/16 14:18に記録
```

- 51 行目～55 行目: main プログラム
- 52 目～53:try ブロック
- 53 行目: 温度データ抽出・連結プログラム Raspi_temp()関数実行
- 54～55:except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文(何も処理しない)を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

参考～Python のデータの取り出し～

リストや文字列、タプルなどのデータから特定の要素を取り出す機能を「スライス」と呼びます。

ここでは、データ（データフレーム）から特定の列や行、要素を取り出す方法を紹介します。

- データフレームの定義

  「29_csvData.py」で定義したデータフレームを例に挙げて「スライス」の方法を説明します。

```
df = pd.read_csv(URL_PATH+"Raspi_temp.csv", names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
```

- 特定の列を取り出す場合

  この場合、データフレームから temperature（温度）列（データ）だけをスライスします。

```
temp=df['temperature']
```

- 特定の列を複数取り出す場合

  この場合、データフレームから month（月）と minute（分）の列（データ）だけをスライスします。

```
date = df[['month','minute']]
```

- 特定の行を取り出す場合

  この場合、0 番目の要素（行）から 2 番目の要素（行）がスライスされます。

  「行」で言い換えると…

  1行目から3行目までのデータをスライスします。

```
row = df[0:3]
```

～補足～

行、列ともに要素番号は 0 番から開始します。

そのため、1 行目の要素番号は 0、50 行目の要素番号は 49 になります。

df[開始行要素:終了行要素]

開始

取り出しを開始する位置です。省略すると先頭の要素を指定します。先頭は 0 から始まります。

マイナスの値を指定すると末尾から数えた位置を指定する意味になります。

終了

取り出しを終える位置です。省略すると最後の要素まで取り出しを行います。

終了-1 までの要素が取り出されることに注意してください。

こちらもマイナスの値を指定すると末尾から数えた位置を指定する意味になります。

Copyright © 2024 YITJC All Rights Reserved.

- 特定の行にある要素を取り出す場合（loc メソッド）

「loc」メソッドは、行名や列名を使用してデータフレームの要素を取り出すことができます。
この場合は、0 番目の行要素（1 行目）から month（月）データをスライスします。

```
df.loc[0, 'month']
```

次のように指定することで、最高温度が記録されている時刻データを取り出すことができます。

```
max=df['temperature'].max()                  #'temperature'列から最高温度 max を抽出
tmax=df[df['temperature'] == max]            #最高温度 max を記録している行を取り出す
tmax.loc[tmax.index[0], 'month'              #最高温度 max を記録したときの' month' データを取り出す
tmax.loc[tmax.index[0], 'day'                #最高温度 max を記録したときの' day' データを取り出す
tmax.loc[tmax.index[0], 'hour']              #最高温度 max を記録したときの' hour' データを取り出す
tmax.loc[tmax.index[0], 'minute']            #最高温度 max を記録したときの' minute' データを取り出す
```

- 特定の行および列にある要素を取り出す場合（iloc メソッド）

「iloc」メソッドは、行番号及び列番号を使用してデータフレームの要素を取り出すことができます。
この場合は、0 番目の行要素（1 行目）から1番目の列要素（2 列目）～4 番目の列要素（5 列目）をスライスします。

```
start=df.iloc[0:1,1:5]
```

～補足～

df.iloc[開始行要素:終了行要素,開始列要素:終了列要素]

開始

取り出しを開始する位置です。先頭は 0 から始まります。

終了

取り出しを終える位置です。終了-1 までの要素が取り出されることに注意してください。

Copyright © 2024 YITJC All Rights Reserved.

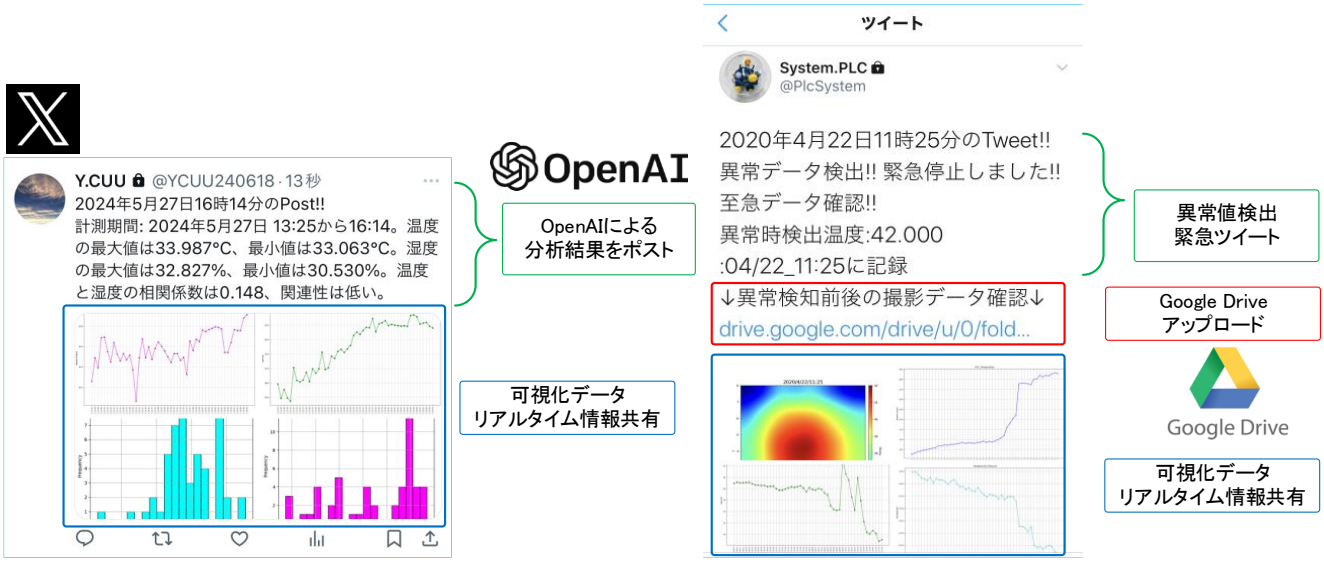
参考～データ活用例～

第 6 章「データ収集と活用」で学習した技術を活用して、オンラインストレージや SNS を利用してリアルタイムに情報共有できます。これらの内容は、8 章「オンラインストレージ」、9 章「通知機能・AI」、10 章「対話型システム構築」で学習します。

X（旧 Twitter）による情報共有

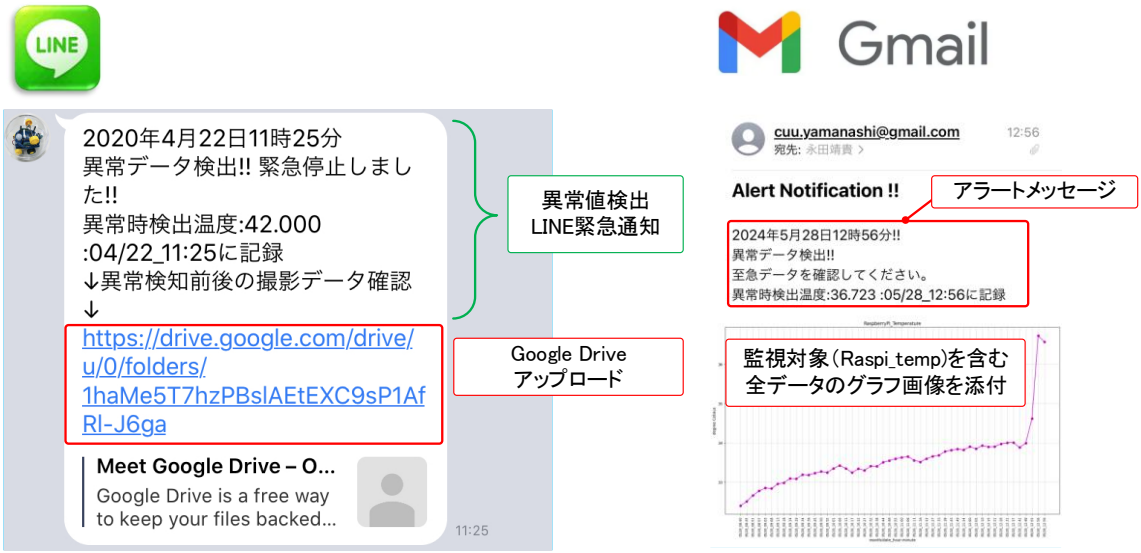
✓ 定期的なデータ更新

✓ 異常検出時のアラート通知



LINE や Gmail による情報共有

✓ 異常検出時のアラート通知



Copyright © 2024 YITJC All Rights Reserved.

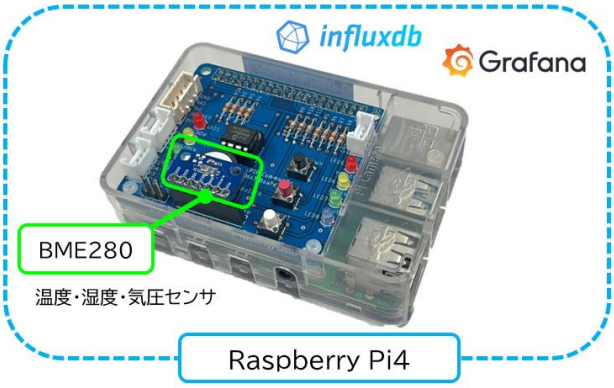
第 7 章 データベース構築・可視化

センサや生産設備、カメラなどから収集したデータを有効活用することで工程能力や製品品質の改善が期待できます。しかし、収集したデータの特徴や傾向を正確に把握して有効活用するためにはデータ量を確保する必要があり、長期的なデータ収集が必要です。「データベース」は、大量のデータを効率的に収集、保存、管理できるため、長期的なデータ収集に適しています。

ここでは、データベースを構築し、データベースに蓄積されたデータを可視化ソフトウェアで可視化する方法について学習します。

7.1 システム構成

本講座はデータベースおよび可視化ソフトウェアの使用方法を学習するため、Raspberry Pi にデータベースを構築し、HAT Board にある温湿度・気圧センサ（BME280）のデータを蓄積します。そして、可視化ソフトウェアを使用して蓄積されたデータを可視化します。



- **データベース（InfluxDB）** influxdb

  本講座で構築するデータベースは、「InfluxDB（インフラックスデービー）」です。InfluxDB は、InfluxData 社が提供するオープンソースの時系列データベースです。

  InfluxDB は、センサや生産設備、カメラなどから送られる時系列データをリアルタイムで取り込み、長期的なデータ収集（蓄積）を実現します。また、データ検索の高速化やデータの圧縮によるストレージ容量の節約、解析時に使用するメモリ量の削減など、時系列データに特化して最適化されているため、大量データの管理に適したデータベースです。

- **可視化ソフトウェア（Grafana）** Grafana

  本講座で使用する可視化ソフトウェアは、「Grafana（グラファナ）」です。Grafana は、Grafana Labs 社が開発したデータ可視化ソフトウェアです。Grafana は、さまざまな種類のグラフや表、パネルなどを使用して時系列データを可視化できます。また、InfluxDB を含むさまざまなデータソース（データベース）に接続できます。他に接続可能なデータソースとして、Prometheus、MySQL、PostgreSQL、Elasticsearch などがあります。また、アラート（警告）機能も備えているため、あらかじめ設定した条件に応じてメールや SNS に警告メッセージを送信することもできます。

Copyright © 2024 YITJC All Rights Reserved.

InfluxDB と Grafana を組み合わせることで、時系列データの収集（蓄積）から可視化まで対応したシステムを構築できます。本講座では、Raspberry Pi4 に InfluxDB と Grafana をインストールしてシステムを構築しますが、将来的にはクラウドサービス（AWS）や Sensor Board を活用することで、より「低コスト」かつ「安全」なシステムを構築できます。

- Sensor Board の活用

  Sensor Board は、ESPr® Developer 32 を搭載し、各種センサとの接続、Raspberry Pi やクラウドサービスへの無線データ通信に対応しています。また、I2C 通信を使用して Isolator Board の入出力制御も可能です。そのため、複数の Sensor Board や Isolator Board から送信されたデータを 1 台の Raspberry Pi に集約できます。

  ～主な機能～

  ○センサ接続

  BME280（温度、湿度、気圧）センサに加えて、Grove シリーズの I2C センサを 4 個、アナログセンサを 2 個接続できます。

  ○無線通信

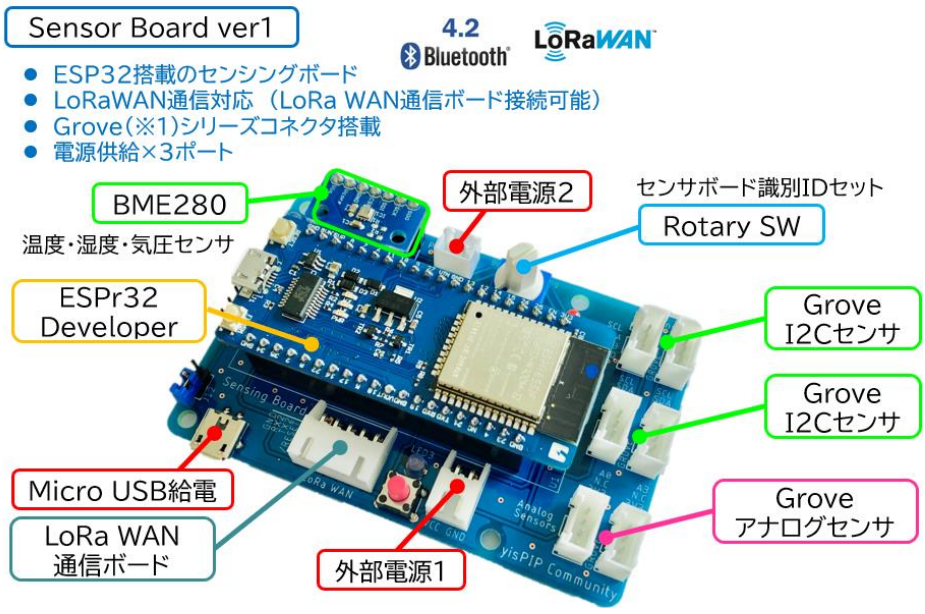
  Bluetooth、Wi-Fi、LoRa（別途 LoRa 通信モジュールを接続して使用）の各種無線通信規格に対応しています。

  ○電源供給

  Micro USB 給電に加えて、2 ヵ所の給電端子を備えています。そのため、モバイルバッテリーや電池などを使用して給電できます。また、各給電ポートの状況は ESP32 が監視しているため、モバイルバッテリーの充電切れや電池の残量不足をリアルタイムで検出できます。

  ○超低消費電力

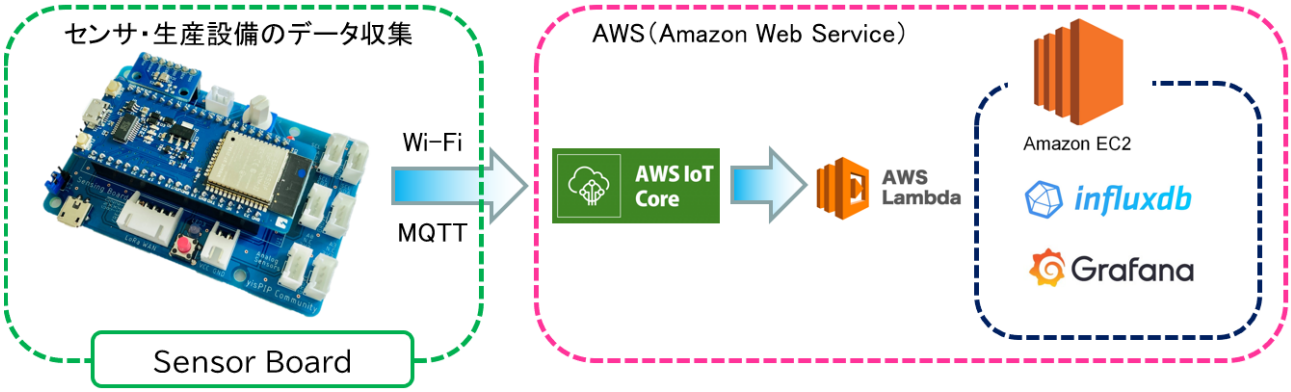
  ESP32 は Deep Sleep モード時の消費電流が数マイクロ A と低く、Deep Sleep モードを使用した間欠動作により、モバイルバッテリーや電池駆動による長期稼働が可能です。



Copyright © 2024 YITJC All Rights Reserved.

- **クラウドサービスの活用**

  Wi-Fi 通信に対応した Sensor Board を活用することで、直接クラウドサービス（AWS）にデータ送信できます。データベース（InfluxDB）や可視化ソフトウェア（Grafana）は、クラウドサービス上のコンピュータ（EC2）にインストールして可視化システムを構築します。これにより、Raspberry Pi を使用しないシステムを構築できます。



Copyright © 2024 YITJC All Rights Reserved.

7.2 InfluxDB インストール

Raspberry Pi に InfluxDB をインストールし、データベースを構築します。

手順①～InfluxDB インストール～

次のコマンドを実行して InfluxDB をインストールしてください。

(ycuu) cuu@raspberrypi:~ $ sudo apt update

```
(ycuu) cuu@raspberrypi:~ $ sudo apt update
ヒット:1 http://deb.debian.org/debian bookworm InRelease
ヒット:2 http://deb.debian.org/debian-security bookworm-security InRelease
取得:3 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
ヒット:4 http://archive.raspberrypi.com/debian bookworm InRelease
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
アップグレードできるパッケージが 113 個あります。表示するには 'apt list --up
able' を実行してください。
```

(ycuu) cuu@raspberrypi:~ $ sudo apt install influxdb influxdb-client

```
(ycuu) cuu@raspberrypi:~ $ sudo apt install influxdb influxdb-client
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下のパッケージが新たにインストールされます:
  influxdb influxdb-client
アップグレード: 0 個、新規インストール: 2 個、削除: 0 個、保留: 113 個。
6,517 kB のアーカイブを取得する必要があります。
この操作後に追加で 24.8 MB のディスク容量が消費されます。
取得:1 http://deb.debian.org/debian bookworm/main arm64 influxdb arm64 1.6.7~
-1+b13 [4,483 kB]
取得:2 http://deb.debian.org/debian bookworm/main arm64 influxdb-client arm64
6.7~rc0-1+b13 [2,034 kB]
6,517 kB を 10秒 で取得しました (666 kB/s)
```

```
Created symlink /etc/systemd/system/influxd.service → /lib/systemd/system/influ
xdb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/influxdb.service →
/lib/systemd/system/influxdb.service.
man-db (2.11.2-2) のトリガを処理しています ...
```

手順②～InfluxDB サービス自動化～

influxdb サービスを起動(start)し、今後自動起動するように設定(enable)します。

(ycuu) cuu@raspberrypi:~ $ sudo systemctl start influxdb.service

(ycuu) cuu@raspberrypi:~ $ sudo systemctl enable influxdb.service

```
(ycuu) cuu@raspberrypi:~ $ sudo systemctl start influxdb.service
(ycuu) cuu@raspberrypi:~ $ sudo systemctl enable influxdb.service
Synchronizing state of influxdb.service with SysV service script with /lib/syste
md/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable influxdb
```

Copyright © 2024 YITJC All Rights Reserved.

手順③～データベース構築～

InfluxDB（データベース）を構築します。

ユーザー名、パスワード、データベース名は、次の名前を使用してください。

- ✓ ユーザー名：cuu
- ✓ パスワード：yamanashi
- ✓ データベース名：sensor

influx コマンドを実行して、InfluxDB を操作します。

(ycuu) cuu@raspberrypi:~ $ influx

InfluxDB に接続（http://localhost:8086）に接続し、データベース（InfluxDB）の操作ができます。

```
(ycuu) cuu@raspberrypi:~ $ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
>
```

次のコマンドを実行し、データベース「sensor」を構築します。

> create database sensor

> use sensor

```
(ycuu) cuu@raspberrypi:~ $ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
> create database sensor
> use sensor
Using database sensor
```

データベース「sensor」を使用するユーザー名「cuu」とパスワード「yamanashi」を設定します。

> create user cuu with password 'yamanashi' with all privileges

> grant all privileges on sensor to cuu

> show users

```
> create user cuu with password 'yamanashi' with all privileges
> grant all privileges on sensor to cuu
> show users
user admin
---- -----
cuu  true
```

注意

InfluxDB 操作画面を終了する場合は、「exit」と入力してください。

```
> show users
user admin
---- -----
cuu  true
> exit
```

Copyright © 2024 YITJC All Rights Reserved.

7.3 Grafana インストール

Raspberry Pi に InfluxDB に蓄積されたデータを可視化するためのソフトウェア「Grafana」をインストールします。

手順①～Grafana インストール～

次のコマンドを実行して InfluxDB をインストールしてください。

注意 コマンドを途中で改行せずに、1行のコマンドとして実行してください。

(ycuu) cuu@raspberrypi:~ $ sudo wget -q -O /usr/share/keyrings/grafana.key https://packages.grafana.com/gpg.key

(ycuu) cuu@raspberrypi:~ $ sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/grafana.key] https://packages.grafana.com/oss/deb stable main" >> /etc/apt/sources.list.d/grafana.list'

```
(ycuu) cuu@raspberrypi:~ $ sudo wget -q -O /usr/share/keyrings/grafana.key https
://packages.grafana.com/gpg.key
(ycuu) cuu@raspberrypi:~ $ sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/
grafana.key] https://packages.grafana.com/oss/deb stable main" >> /etc/apt/sourc
es.list.d/grafana.list'
```

(ycuu) cuu@raspberrypi:~ $ sudo apt update

```
(ycuu) cuu@raspberrypi:~ $ sudo apt update
ヒット:1 http://deb.debian.org/debian bookworm InRelease
ヒット:2 http://deb.debian.org/debian-security bookworm-security InRelease
ヒット:3 http://deb.debian.org/debian bookworm-updates InRelease
取得:4 https://packages.grafana.com/oss/deb stable InRelease [7,661 B]
ヒット:5 http://archive.raspberrypi.com/debian bookworm InRelease
取得:6 https://packages.grafana.com/oss/deb stable/main armhf Packages [181 kB]
取得:7 https://packages.grafana.com/oss/deb stable/main arm64 Packages [200 kB]
389 kB を 2秒 で取得しました (181 kB/s)
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
アップグレードできるパッケージが 113 個あります。表示するには 'apt list --upgrad
able' を実行してください。
```

(ycuu) cuu@raspberrypi:~ $ sudo apt install -y grafana

```
(ycuu) cuu@raspberrypi:~ $ sudo apt install -y grafana
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
状態情報を読み取っています... 完了
以下の追加パッケージがインストールされます:
  musl
以下のパッケージが新たにインストールされます:
  grafana musl
アップグレード: 0 個、新規インストール: 2 個、削除: 0 個、保留: 113 個。
111 MB のアーカイブを取得する必要があります。
この操作後に追加で 424 MB のディスク容量が消費されます。
```

```
### NOT starting on installation, please execute the following statements to con
figure grafana to start automatically using systemd
 sudo /bin/systemctl daemon-reload
 sudo /bin/systemctl enable grafana-server
### You can start grafana-server by executing
 sudo /bin/systemctl start grafana-server
man-db (2.11.2-2) のトリガを処理しています ...
(ycuu) cuu@raspberrypi:~ $
```

Copyright © 2024 YITJC All Rights Reserved.

参考～apt リポジトリ～

InfluxDB と Grafana のインストール手順が違うことに気づいた方がいると思います。

InfluxDB のインストールは、

(ycuu) cuu@raspberrypi:~ $ sudo apt update

(ycuu) cuu@raspberrypi:~ $ sudo apt install influxdb influxdb-client

Grafana のインストールは、

(ycuu) cuu@raspberrypi:~ $ sudo wget -q -O /usr/share/keyrings/grafana.key https://packages.grafana.com/gpg.key

(ycuu) cuu@raspberrypi:~ $ sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/grafana.key] https://packages.grafana.com/oss/deb stable main" >> /etc/apt/sources.list.d/grafana.list'

この 2 つのコマンドを実行したあと、apt-install を実行しています。

(ycuu) cuu@raspberrypi:~ $ sudo apt update

(ycuu) cuu@raspberrypi:~ $ sudo apt install -y grafana

この違いを説明するため、「apt リポジトリ」について説明します。

「apt リポジトリ」は、公式ソフトウェアのパッケージ情報が含まれており、システム（Raspberry Pi OS）がパッケージをインストール、更新、削除するための情報源として機能します。

InfluxDB の場合、

公式ソフトウェアとして登録されていたため（apt リポジトリにパッケージ情報が含まれていたため）、apt コマンドを実行するだけでインストールできました。

Grafana の場合、

公式ソフトウェアとして登録されていないため（apt リポジトリにパッケージ情報が含まれていないため）、次の手順でインストールしました。

① Grafana 公式サイト（https://packages.grafana.com/gpg.key）から GPG キー（GNU Privacy Guard）と呼ばれる公式鍵をダウンロード

② ダウンロードした公式鍵を使用して、リポジトリ（https://packages.grafana.com/oss/）をダウンロードして、Raspberry Pi 内部の apt リポジトリファイル（/etc/apt/sources.list）に追加

③ apt update コマンドを実行して apt リポジトリファイルの情報を更新

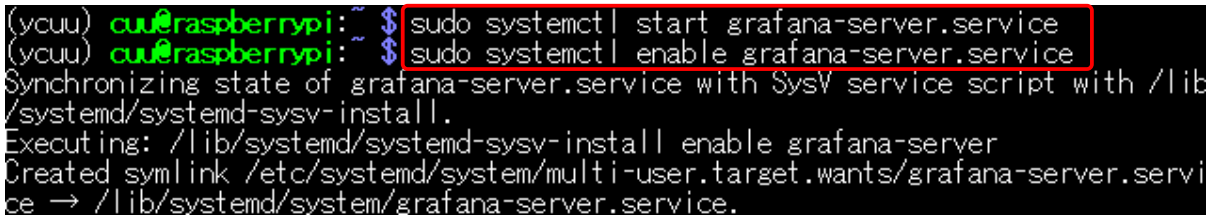
④ apt install コマンドを実行して grafana をインストール

手順②～Grafana サービス自動化～

Grafana サービスを起動（start）し、今後自動起動するように設定（enable）します。

(ycuu) cuu@raspberrypi:~ $ sudo systemctl start grafana-server.service

(ycuu) cuu@raspberrypi:~ $ sudo systemctl enable grafana-server.service



Copyright © 2024 YITJC All Rights Reserved.

手順③～Grafana アクセス確認～

Windows PC の Web ブラウザを使用して、Raspberry Pi にインストールした Grafana にアクセスします。

アドレスバーに次のアドレスを入力して、Enter キーを押してください。

（Raspberry Pi の IP Address）:3000

または

http://（Raspberry Pi の IP Address）:3000

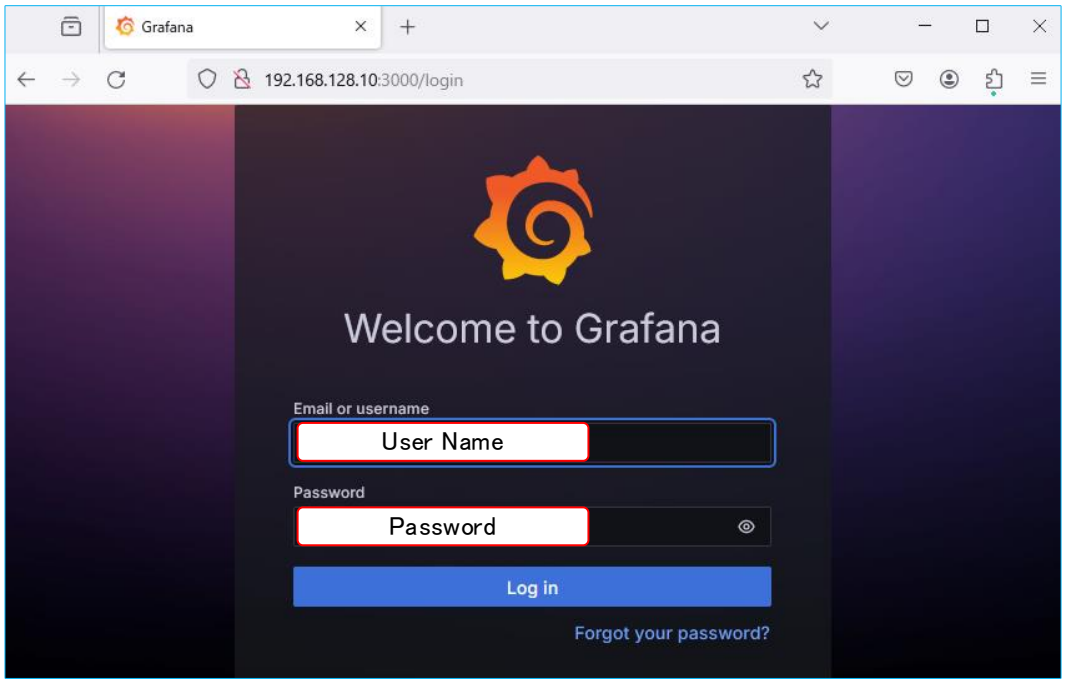
例）Raspberry Pi の IP アドレスが、192.168.128.10 の場合、

192.168.128.10:3000

または

http://192.168.128.10:3000

Grafana へのアクセスが成功すると、次の画面が表示されます。



Copyright © 2024 YITJC All Rights Reserved.

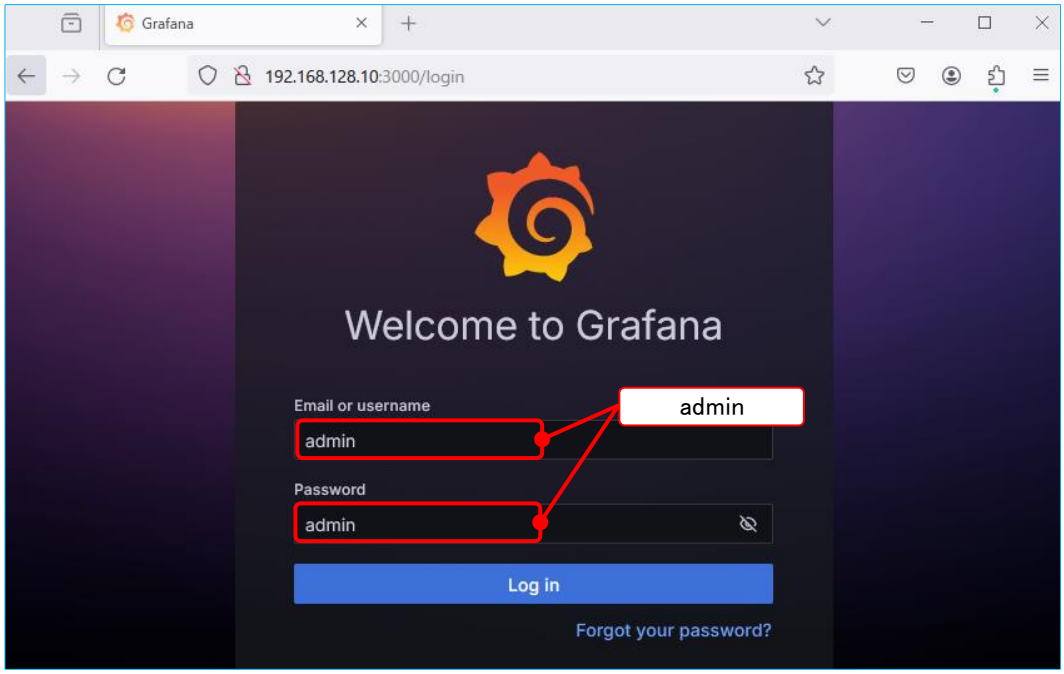
手順④～Grafana パスワード設定～

Grafana にログインしてパスワードを設定します。

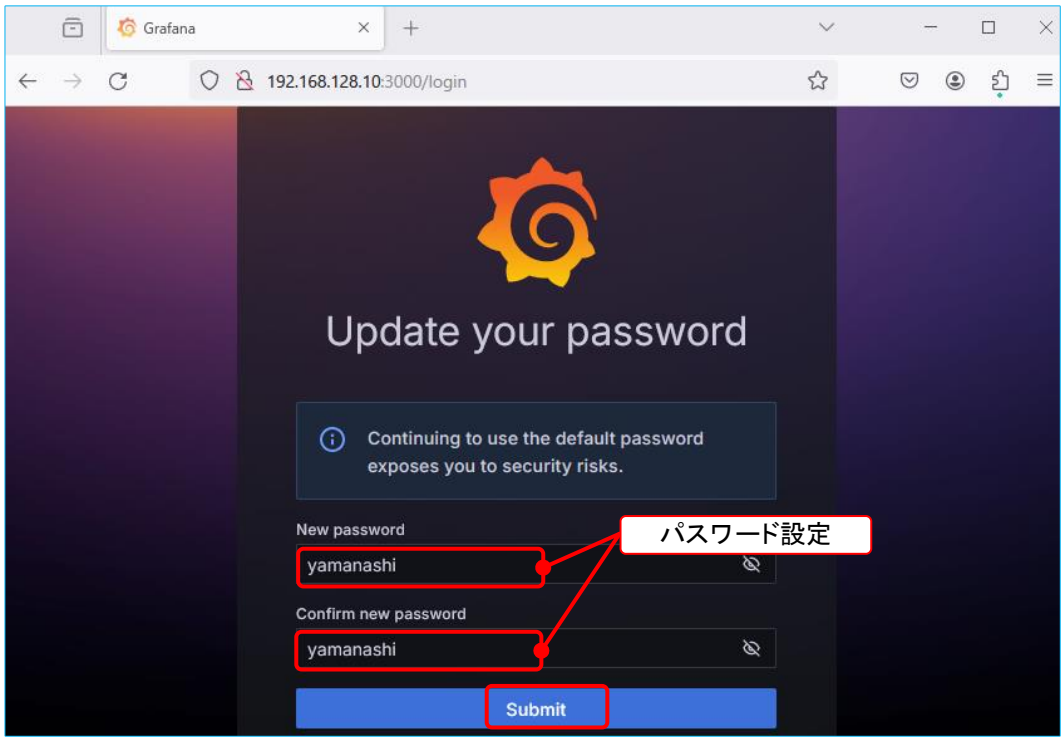
最初の画面でログインする際は、次の初期設定でログインしてください。

User Name: admin

Password: admin

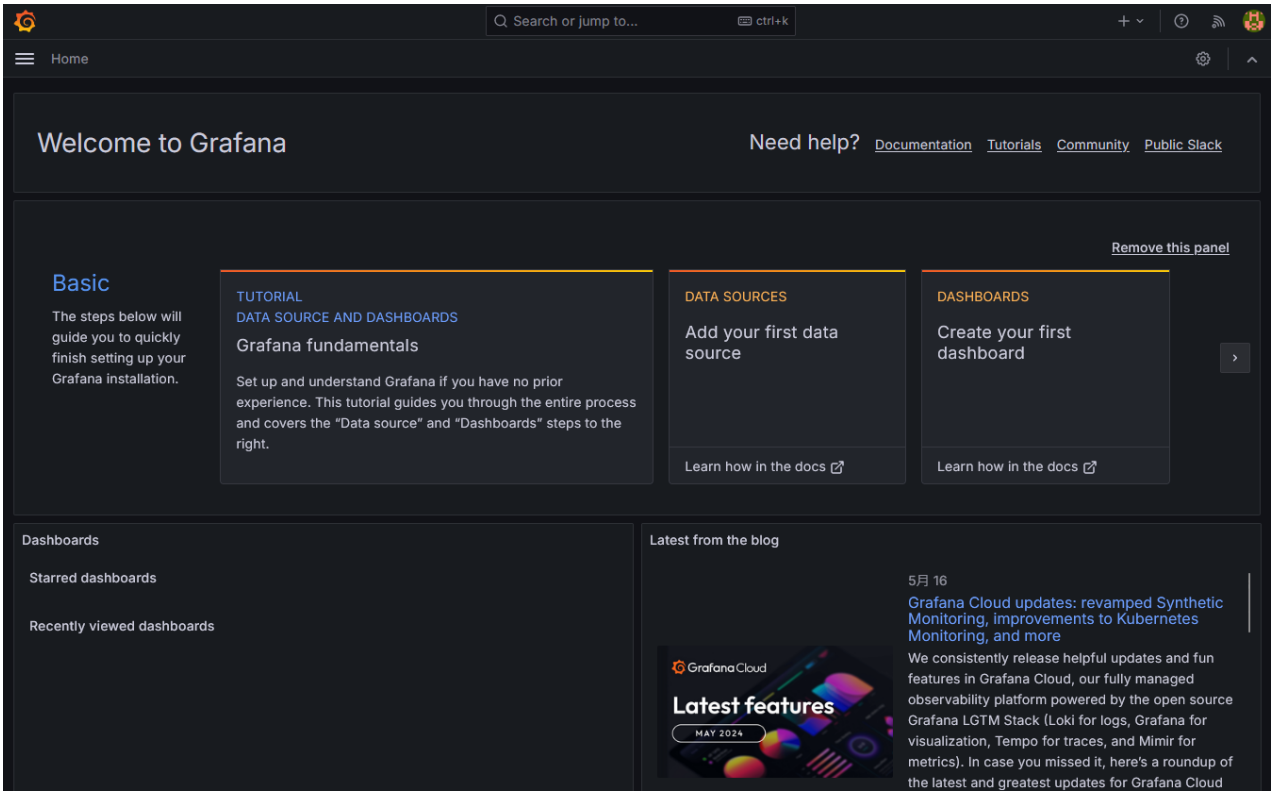


ログインに成功すると、パスワード設定画面が開きます。新たなパスワードを入力してから、「Submit」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

パスワードの設定が完了すると、次の画面が表示されます。



InfluxDB との接続や InfluxDB に蓄積されたデータの可視化については、このあと説明します。

Copyright © 2024 YITJC All Rights Reserved.

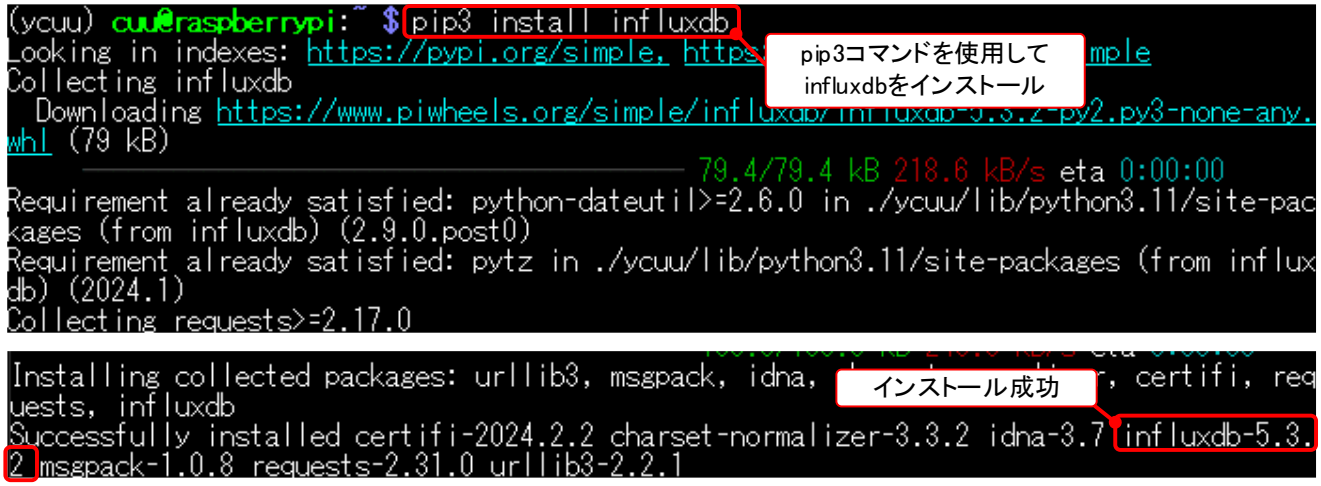
7.4 データ蓄積

Raspberry Pi に構築したデータベースに HAT Board に実装されている温湿度・気圧センサ（BME280）のデータを蓄積する方法について学習します。

手順①～influxdb インストール～

Python を使用して InfluxDB へアクセスするためのライブラリ「influxdb」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install influxdb



～補足～

Influxdb と同時に依存関係のあるパッケージもインストールされます。requests や urllib3（http 通信用）や msgpack（データのシリアライズ）など、データベースへのアクセスや http 通信など、効率的なデータ処理に必要なライブラリがインストールされます。

手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

30_InfluxDB.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 30_InfluxDB.py

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./30_InfluxDB.py

プログラムを実行すると、データベースに記録されたデータ（温度、湿度、気圧）が print 表示されます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～データ確認～

データベース「sensor」に保存されたデータを確認します。

influx コマンドを実行して、データベース「sensor」を操作します。

(ycuu) cuu@raspberrypi:~ $ influx

InfluxDB に接続（http://localhost:8086）に接続し、データベース（InfluxDB）の操作ができます。

```
(ycuu) cuu@raspberrypi:~ $ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
>
```

次のコマンドを実行し、データベース「sensor」を選択します。

> use sensor

```
> use sensor
Using database sensor
```

次のコマンドを実行し、データベース「sensor」に記録されているデータを表示します。

> select * from sensor

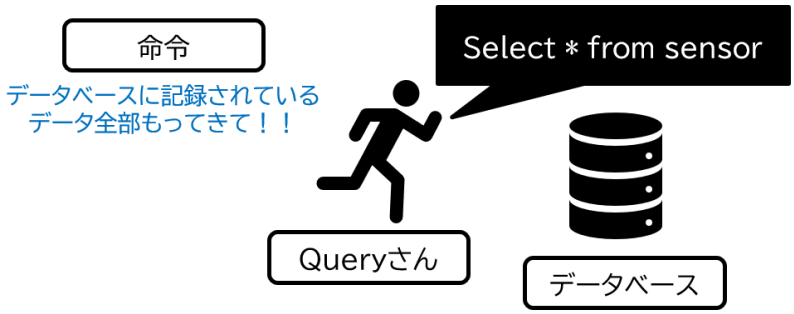
```
> select * from sensor
name: sensor
time                Hum   Press  Temp
----                ---   -----  ----
1715990710446547000 33.38 991.67 32.15
```

Unix時間

データを確認した後は、「exit」と入力して InfluxDB 操作画面を終了してください。

参考～Query（クエリ）～

データベースに対する命令文を「Query（クエリ）」と呼びます。



Query（命令文）で最も有名なのが SQL（Structured English Query Language）です。InfluxDB が使用する Query は InfluxQL（Influx Query Language）と呼ばれ、SQL に似ています。しかし、InfluxDB は時系列データの処理に特化しているため、SQL とは異なる構文や機能があります。

今回実行した Query（命令文）について解説します。

select * from sensor

- select　：データを取得するためのキーワード
- *　：全フィールドを選択するため、ワイルドカード演算子（*）を使用
- from　：データベースを指定するキーワード
- sensor　：データベースを指定

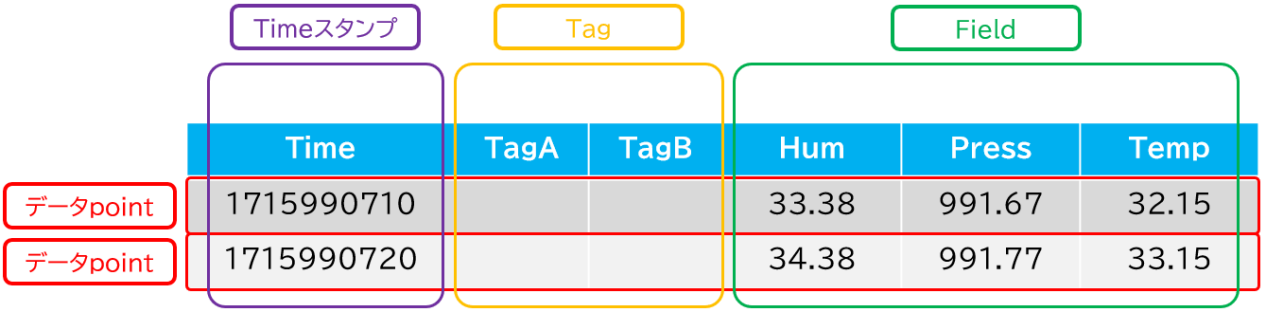
Copyright © 2024 YITJC All Rights Reserved.

参考～InfluxDB の構造～

- メジャメント（Measurement）
  データベースを構成するテーブルです。

Time	TagA	TagB	Hum	Press	Temp
1715990710			33.38	991.67	32.15
1715990720			34.38	991.77	33.15

- データ point
  1 つのデータレコードをデータ point と呼びます。
  データ point は、「Time スタンプ」、「Tag」、「フィールド」で構成されます。
  今回実行したプログラムは、「Tag」情報を記録していません。使い方としては、デバイス ID やセンサロケーション（センサ設置場所）などの情報を記録します。これにより、Tag 付されたデータの高速検索や Tag 付されたデータの分類・可視化が可能になります。



～注意～

フィールド名（field）には次のルールがあるので注意してください。

- フィールド名に「#」を使用してはならない（フィールド名のどの部分にも使用することはできません）
- フィールド名の頭文字に「数字」を使用してはならない

この 2 点が守られていない場合、命令文を実行してもデータを取得できません。

参考～Unix 時間～

InfluxDB の Time スタンプには「Unix 時間」が記録されます。Unix 時間は、コンピュータシステム上での時刻表現の一種であり、1970 年1月1日午前 0 時 0 分 0 秒（協定世界時 UTC（Universal time coordinated））から経過した時間で表現します。

Copyright © 2024 YITJC All Rights Reserved.

参考～InfluxQL（命令文）～

ユーザー名、パスワード、データベース名に次の名前を使用した場合の例を挙げます。

- ✓ ユーザー名：cuu
- ✓ パスワード：yamanashi
- ✓ データベース名：sensor

- データベース登録

  >create database sensor

  >use sensor

  >create user cuu with password 'yamanashi' with all privileges

  >grant all privileges on sensor to cuu

  >show users
- ユーザー表示

  >show users
- データベース表示

  >show databases

- データ確認（全データ表示）

  >use sensor

  >select * from sensor
- データ確認（時刻指定）

  データベースに記録されているデータは UTC 時刻で記録されていますが、tz（Time Zone）を指定することで、InfluxQL がタイムゾーンを考慮してデータを取得します。

  >select * from sensor where time >= '2024-05-18 09:00:00' tz('Asia/Tokyo')
- データ確認（フィールド指定、条件指定）

  10 分前から現在までの”Temp”のデータ表示

  >select "Temp" from sensor where time > now() -10m

- データ登録数確認

  >select count(*) from sensor
- データ登録数確認（フィールド指定）

  >select count("Temp") from sensor
- データ登録数確認（条件指定）

  “Temp”データについて、データの値が 30（度）以上のデータ件数を確認

  >select count("Temp") from sensor where "Temp" >30.0

  “Temp”データについて、10 分前から現在までのデータ件数を確認

  >select count("Temp") from sensor where time > now() -10m

  10 分前から現在までの”Temp”のデータ数 ※ただし、16.6℃より大きいデータ

  >select count("Temp") from sensor where time > now() -10m and "Temp" >16.6

Copyright © 2024 YITJC All Rights Reserved.

- データ削除

  注意 delete 命令は UTC 時刻で指定しなければなりません。(現時刻-9 時間で指定します)

  >delete from sensor where time >= '2024-05-18T09:00:00Z'

- CSV 出力

  (ycuu) cuu@raspberrypi:~/dev $ influx -database 'sensor' -execute 'SELECT * FROM sensor' -format csv > BME280.csv

  「dev」ディレクトリでこのコマンドを実行すると、dev ディレクトリに指定したファイル名(BME280.csv)の csv ファイルが保存されます。

- CSV 出力(時間範囲指定)

  例:2022 年 10 月 1 ヶ月分のデータを取得するコマンド

  UTC 時刻を考慮しているため、コマンドで指定している時刻から「-9:00」で指定します。

  (ycuu) cuu@raspberrypi:~/dev $ influx -database 'sensor' -execute "SELECT * FROM sensor where time >= '2022-09-30 15:00:00' AND time < '2022-10-31 14:59:59'" -format csv > BME280.csv

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import BME280_Class
5	
6	import influxdb
7	import datetime
8	
9	db = influxdb.InfluxDBClient(
10	host='localhost',
11	port=8086,
12	database='sensor'
13	)
14	
15	def write_to_influxdb(sdata):
16	# InfluxDBに書き込むデータを構築
17	points = [{
18	'measurement': 'sensor',
19	'time': datetime.datetime.utcnow(),
20	'fields': sdata
21	}]
22	# InfluxDBにデータを書き込む

Copyright © 2024 YITJC All Rights Reserved.

```
23          db.write_points(points)
24
25  if __name__ == '__main__':
26          bme = BME280_Class.BME280()
27          bme.readData()
28          try:
29                  sdata = {
30                          "Temp": round(bme.getTemperature(), 2),
31                          "Hum": round(bme.getHumidity(), 2),
32                          "Press": round(bme.getPressure(), 2),
33                          }
34                  print(sdata)
35                  write_to_influxdb(sdata)
36          except KeyboardInterrupt:
37                  pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: BME280_Class モジュールをインポート（import）
- 6 行目: InfluxDB へアクセスするためのライブラリ influxdb インポート（import）
- 7 行目: 日付時刻取得用 datetime ライブラリをインポート
- 9 行目～13 行目: InfluxDB に接続するためのクライアントを定義
- 10 行目: host の設定
  'localhost' と指定することで、Raspberry Pi4 をホストとして使用します。
  これにより、Raspberry Pi4 への通信が可能になります。
- 11 行目: ポート番号の設定
  外部（Grafana）から InfluxDB にアクセスする際に必要なポート番号を設定します。
  「8086」は InfluxDB デフォルトのポート番号です。
- 12 行目: データベース名を指定
- 15 行目～23 行目: InfluxDB へのデータ書き込みプログラム write_to_influxdb()関数
- 17 行目～21 行目: InfluxDB に書き込むデータ point を定義
- 18 行目: メジャメント（measurement）にデータベース「sensor」を指定
- 19 行目: time スタンプに UTC 時刻指定
  ～参考～
  InfluxDB の time スタンプに UTC 時刻を指定していますが、Grafana が InfluxDB からデータを取得して可視化する際は、ローカル時刻（日本時間）に自動で変換されます。
- 20 行目: フィールド（field）に sdata（29 行目～33 行目で定義）を指定
- 23 行目: 定義したデータ point を InfluxDB に書き込む

Copyright © 2024 YITJC All Rights Reserved.

- 25 行目～37 行目: main プログラム
- 26 行目: クラス生成
  BME280_Class モジュールからクラスを生成し、変数「bme」に代入しています。
- 27 行目: クラスで定義されている readData()関数の実行
- 28 行目～35 行目 : try ブロック
- 29 行目～33 行目: InfluxDB のフィールド（field）に書き込むデータを定義
- 30 行目： クラスで定義されている getTemperature()関数の実行・取得し、'Temp'データと定義
- 31 行目： クラスで定義されている getHumidity()関数の実行・取得し、'Hum'データと定義
- 32 行目： クラスで定義されている getPressure()関数の実行・取得し、'Press'データと定義
- 35 行目: InfluxDB にデータ（sdata）を書き込む
- 36 行目～37 行目 : except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

7.5 データ蓄積の自動化

「30_InfluxDB.py」は、プログラムを実行するとデータベースにデータを記録します。このプログラムを定期的に実行することで、長期的なデータ収集の自動化が実現します。定期的に実行する方法は、6 章で構築した「データ収集自動システム」と同様に、「Systemd」という仕組みを使用します。

手順①～Service ファイル作成～

ここで作成するファイルは、センサデータをデータベースに書き込むプログラム（30_InfluxDB.py）を自動起動させるための Unit ファイル（.service ファイル）です。

まず、「/etc/systemd/system」ディレクトリへ移動します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

```
(ycuu) cuu@raspberrypi:~/dev $ cd ..
(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system
(ycuu) cuu@raspberrypi:/etc/systemd/system $
```

続いて、Unit（.service）ファイルを作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit（.service）ファイルの中身を書き込んでください。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano database.service

```
 GNU nano 7.2                          database.service *
[Unit]
Description=Data accumulation
After=network.target

[Service]
ExecStart=/home/cuu/dev/30_InfluxDB.py
User=cuu
Group=cuu

[Install]
WantedBy=multi-user.target
```

30_InfluxDB.pyを指定

※本講座で使用する Unit ファイルは、共有フォルダ「1_Python¥etc_systemd_system」に保存されています。

① VS Code で Unit ファイルを開きます

② 必要に応じて編集します

※このとき、VS Code の画面で改行コードが LF「↓」になっていることを確認します。

③ VS Code からコピーして、Tera Term の画面上で貼り付けます

Tera Term 画面上で「右クリック」すると次の画面が表示されるので、「OK」をクリックしてください。

④ Tera Term 画面上で上書き保存を実行します

・編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

・「Enter」キーを押下して上書き保存を実行

・「Ctrl + X」キーを押して元の画面に戻る

Copyright © 2024 YITJC All Rights Reserved.

～Unit（サービス）ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Data accumulation
3	After=network.target
4	
5	[Service]
6	ExecStart=/home/cuu/dev/30_InfluxDB.py
7	User=cuu
8	Group=cuu
9	
10	[Install]
11	WantedBy=multi-user.target

灰色部は、これまでに説明した内容です。

- 1 行目～3 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文（動作に影響しませんが、分かりやすく記述するようにしてください。）
- 3 行目: 起動順序の指定
  network.target サービスの後（After に起動する設定にしています。
- 5 行目～9 行目: Service セクション
- 6 行目: 自動起動するプログラムを指定（30_InfluxDB.py）
- 8 行目～9 行目: 実行する User 及び Group を指定
  11 行目～12 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 12 行目: サービス Unit の依存関係を設定
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

手順②～サービス再読み込み～

手順①で Unit ファイルを作成した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl daemon-reload

手順③～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl enable database.service

Copyright © 2024 YITJC All Rights Reserved.

手順④～サービス開始～

次のコマンドを実行してサービスを開始します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl start database.service

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable database.service
Created symlink /etc/systemd/system/multi-user.target.wants/database.service → /etc/syste
md/system/database.service.
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start database.service
```

手順⑤～Timer ファイル作成～

手順①で作成した「database.service」をタイマー制御するための Unit ファイル（.timer ファイル）を作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit ファイル（.timer ファイル）の中身を書き込んでください。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo nano database.timer

書き込んだ後は、「ctrl + O（オー）」、「Enter」を押して上書き保存してください。

「書き込みました」の表示がでたら、「Ctrl + X」を押して、元の画面に戻ります。

～Unit（タイマー）ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Accumulate data every 30 seconds
3	
4	[Timer]
5	OnUnitActiveSec=30sec
6	Unit=database.service
7	
8	[Install]
9	WantedBy=multi-user.target

灰色部は、これまでに説明した内容です。

- 1 行目～2 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文（動作に影響しませんが、分かりやすく記述するようにしてください。）

Copyright © 2024 YITJC All Rights Reserved.

- 4 行目～6 行目: timer セクション
  自動起動するファイル（database.service）および
  起動日時（起動間隔）を指定するセクションです。
- 5 行目: 起動間隔（実行間隔）の指定
- 6 行目: 自動起動するファイルの指定
- 8 行目～9 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 9 行目: サービス Unit の依存関係を設定
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

手順⑥～タイマー再読み込み～

手順⑤で Unit ファイル（.timer ファイル）を作成した後は、次のコマンドを実行してタイマーを再読み込みします。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl daemon-reload

手順⑦～タイマー自動起動有効化～

次のコマンドを実行してタイマーの自動起動を有効化します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl enable database.timer

手順⑧～タイマー開始～

次のコマンドを実行してタイマーを開始します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl start database.timer

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable database.timer
Created symlink /etc/systemd/system/multi-user.target.wants/database.timer → /etc/systemd
/system/database.timer.
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start database.timer
```

手順⑨～タイマー動作状況確認～

次のコマンドを実行してタイマーの動作状況を確認します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl status database.timer

正常に稼働（自動起動）していれば、次の画面のようにステータスが active（waiting）と表示されます。

この画面表示を終了する場合は、「q」を押してください。

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status database.timer
● database.timer - Accumulate data every 30 seconds
     Loaded: loaded (/etc/systemd/system/database.timer; enabled; preset: enabled)
     Active: active (running) since Sat 2024-05-18 16:16:28 JST; 38s ago
    Trigger: n/a
   Triggers: ● database.service

5月 18 16:16:28 raspberrypi systemd[1]: Started database.timer - Accumulate data every
```

Copyright © 2024 YITJC All Rights Reserved.

Influx コマンドを使用して、データベース「sensor」に記録されているデータを確認します。

(ycuu) cuu@raspberrypi:~ $ influx

> use sensor

次のコマンドを実行し、データベース「sensor」に記録されているデータを表示します。

> select * from sensor

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
> use sensor
Using database sensor
> select * from sensor
name: sensor
time                Hum   Press  Temp
----                ---   -----  ----
1716012997405360000 22.19 987.11 36.44
1716016085863219000 20.74 987.26 36.37
1716016589564420000 20.62 987.28 36.38
1716016627707808000 20.14 987.36 36.36
1716016673887580000 20.29 987.45 36.36
1716016723875301000 20.07 987.37 36.47
1716016783892279000 19.87 987.41 36.41
1716016853873158000 19.98 987.42 36.56
1716016913871797000 20.57 987.38 36.49
```

database.service
database.timerによってデータが自動で蓄積されている

Copyright © 2024 YITJC All Rights Reserved.

7.6 ダッシュボードを活用した IoT システム（ver1）

「7.5 データ蓄積の自動化」でセンサデータを自動的にデータベースに蓄積できるようになりました。ここでは、データベースに蓄積されているデータを可視化して IoT システム ver1 を構築します。具体的には、可視化ソフトウェア「Grafana」を使用して、データベースに蓄積されたセンサデータをグラフやパネルを用いて「ダッシュボード」を作成します。ダッシュボードとは、グラフやチャート、メーターなど可視化された情報を 1 つの画面にまとめて表示するツールのことを言います。

手順①～Grafana アクセス～

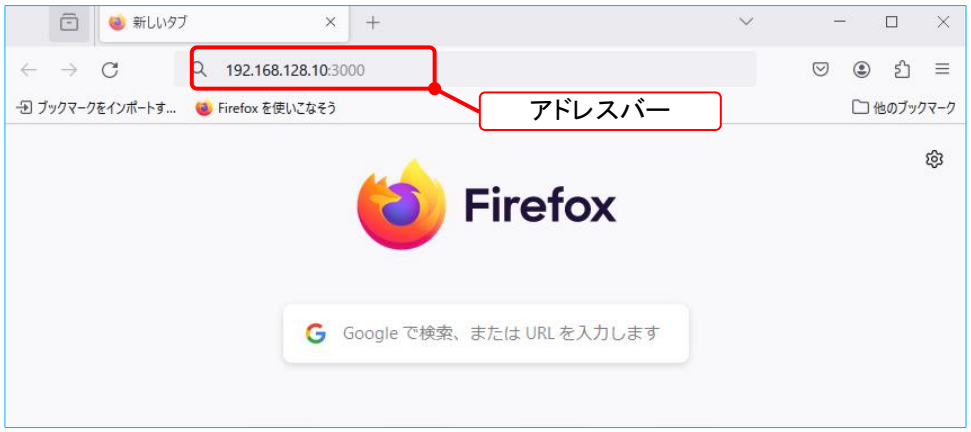
Windows PC の Web ブラウザを使用して、Raspberry Pi にインストールした Grafana にアクセスします。

アドレスバーに次のアドレスを入力して、Enter キーを押してください。

（Raspberry Pi の IP Address）:3000

または

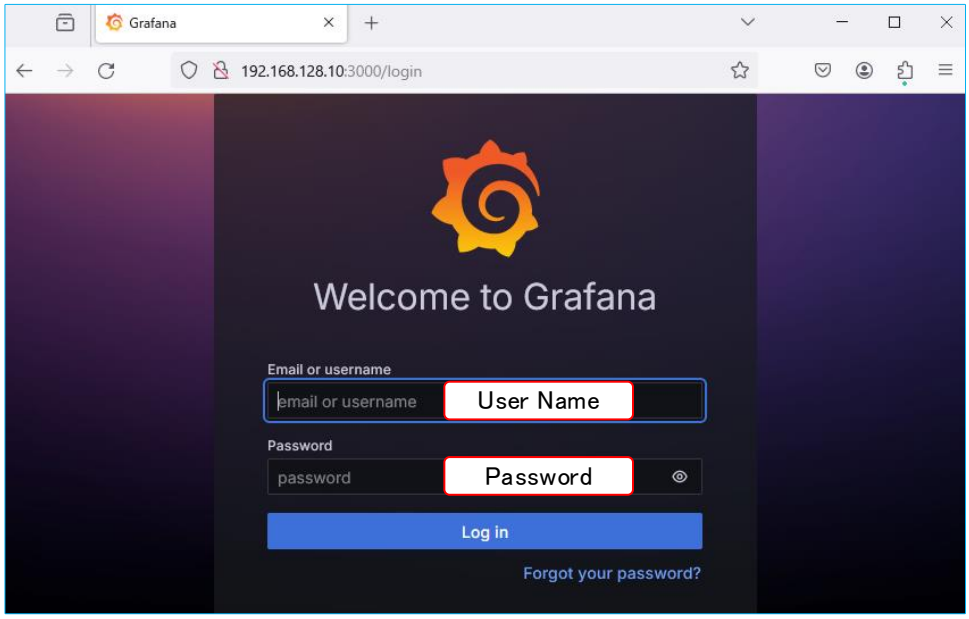
http://（Raspberry Pi の IP Address）:3000



Grafana へのアクセスが成功すると、次の画面が表示されます。

「7.3 Grafana インストール」で設定したパスワードを使用してログインしてください。

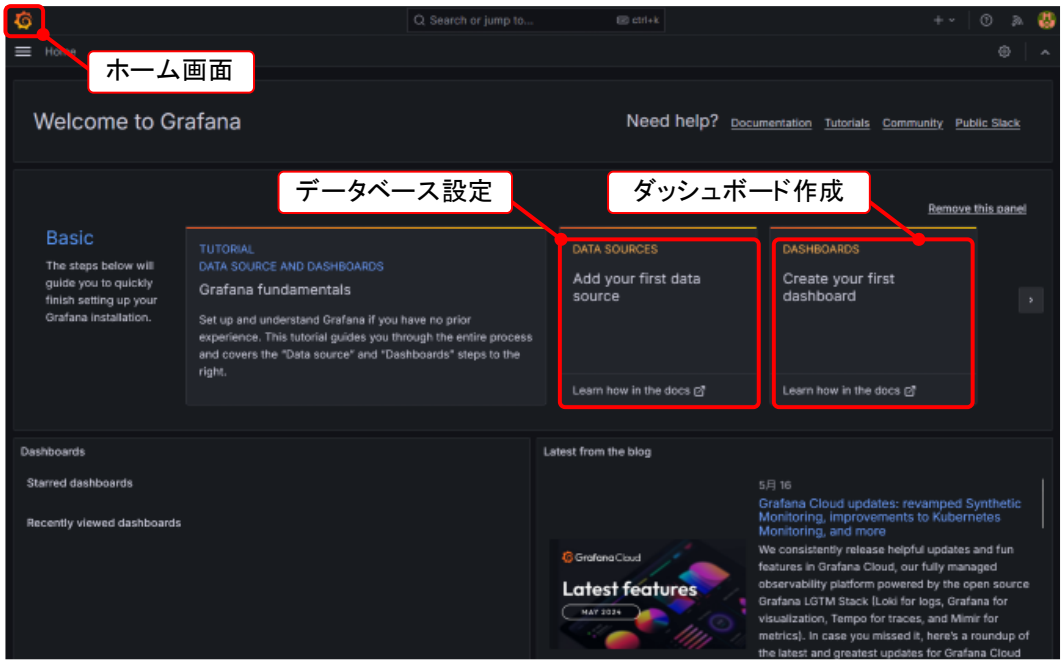
User Name は、「admin」を使用してください。



Copyright © 2024 YITJC All Rights Reserved.

Grafana のホーム画面を紹介します。

これから使用する項目は、InfluxDB への接続設定を行う「Data Sources」とダッシュボードを作成する「Dash Boards」です。



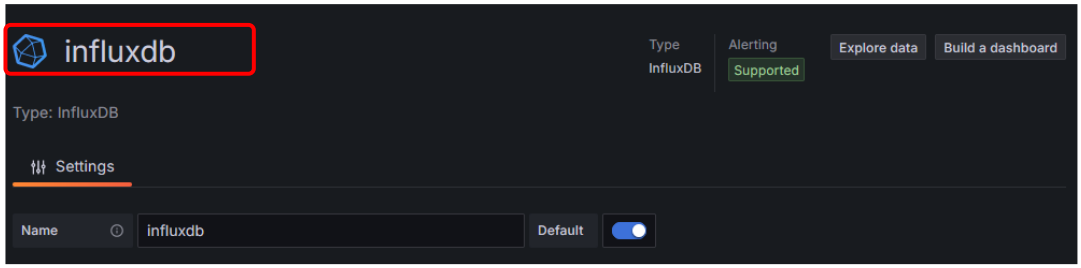
手順②～データベース設定～

画面の「Data Sources（データベース設定）」をクリックして、データベースを設定します。

「Add data source」画面が表示されるので、一覧から「InfluxDB」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

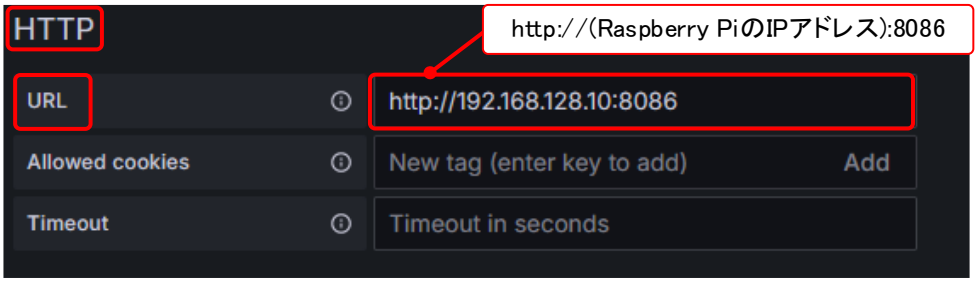
InfluxDB の設定画面に切り替わります。



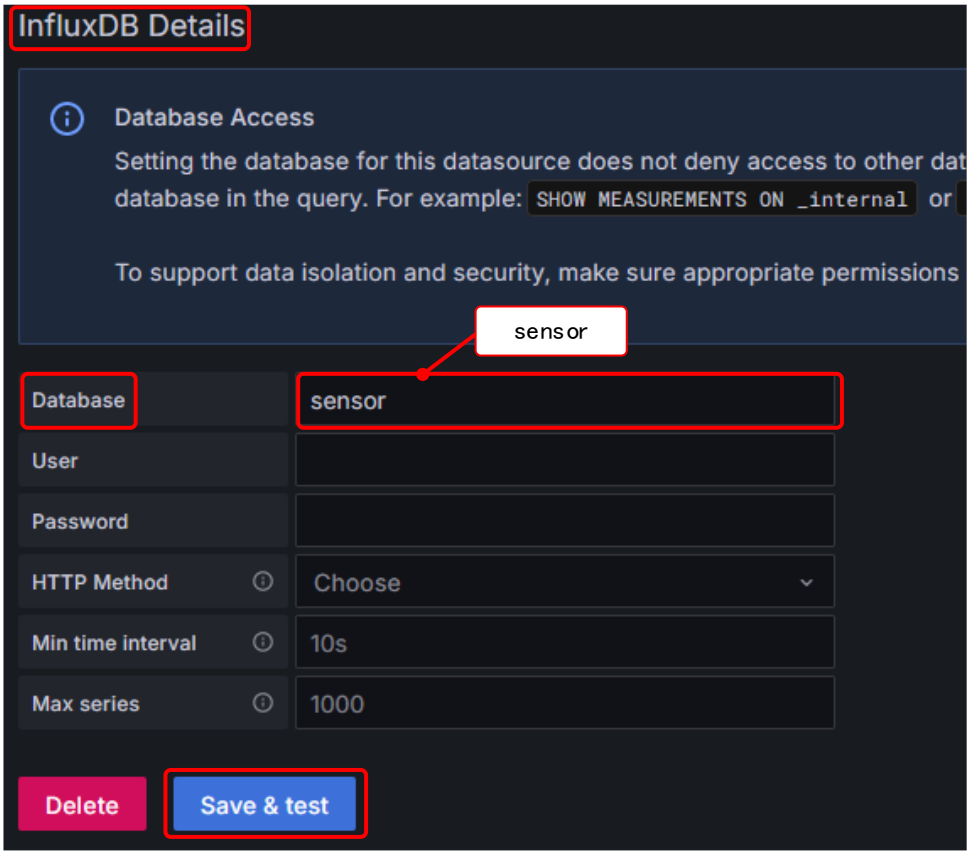
「HTTP」セクションでは、Raspberry Pi に構築したデータベースの URL を設定します。

URL には、次のアドレスを指定します。

http://（Raspberry Pi の IP Address）:8086

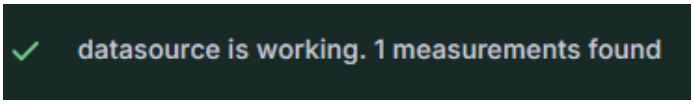


「InfluxDB Details」セクションでは、データベースの名前「sensor」を設定します。



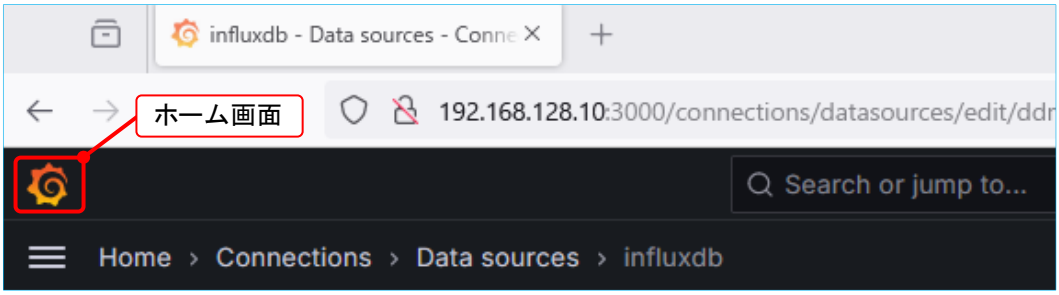
続いて、「Save & test」をクリックして Raspberry Pi に構築したデータベースへの接続テストを行います。

接続に成功すれば、次の画面が表示されます。

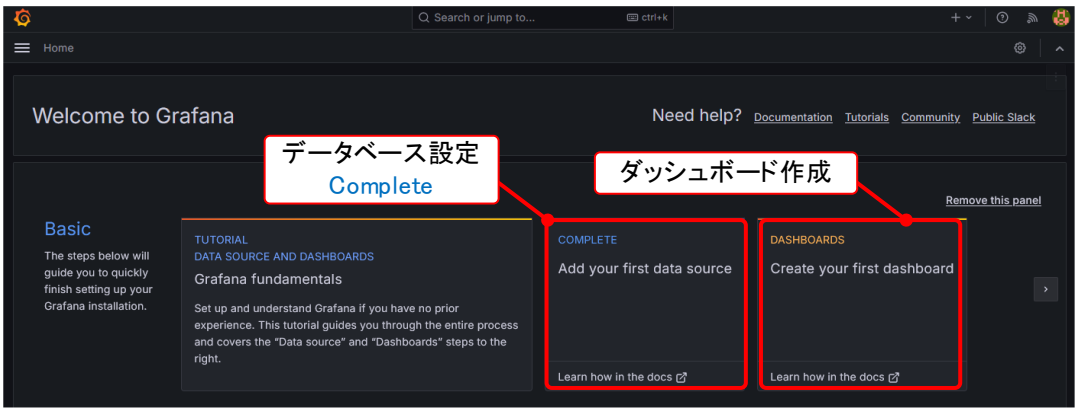


Copyright © 2024 YITJC All Rights Reserved.

データベースとの接続が完了したが、画面左上にある「ホーム画面」をクリックしてホーム画面に戻ってください。



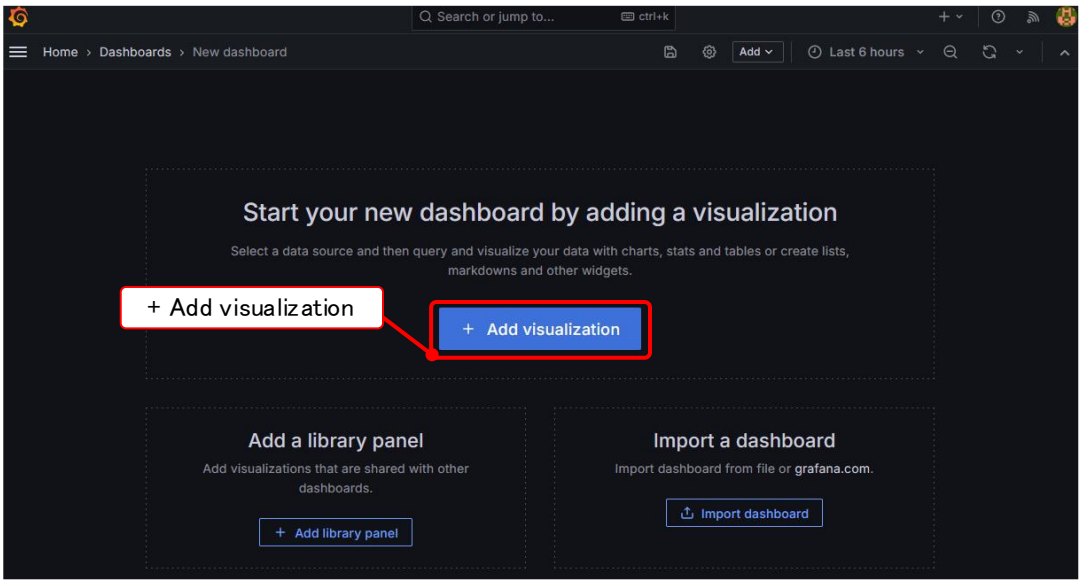
「Data Sources」にはデータベースとの接続設定が完了したことを示す「Complete」が表示されます。



手順③～ダッシュボード作成～

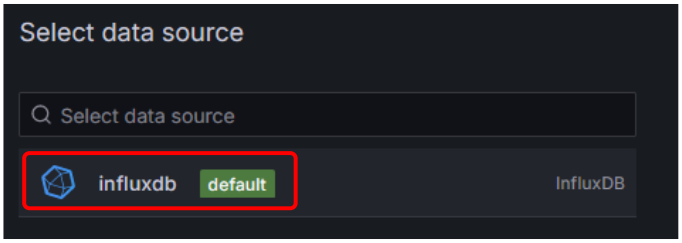
画面の「Dash Boards」をクリックして、ダッシュボードを作成します。

「+ Add visualization」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

続いて、Data Source を選択する画面で、「InfluxDB」をクリックしてください。

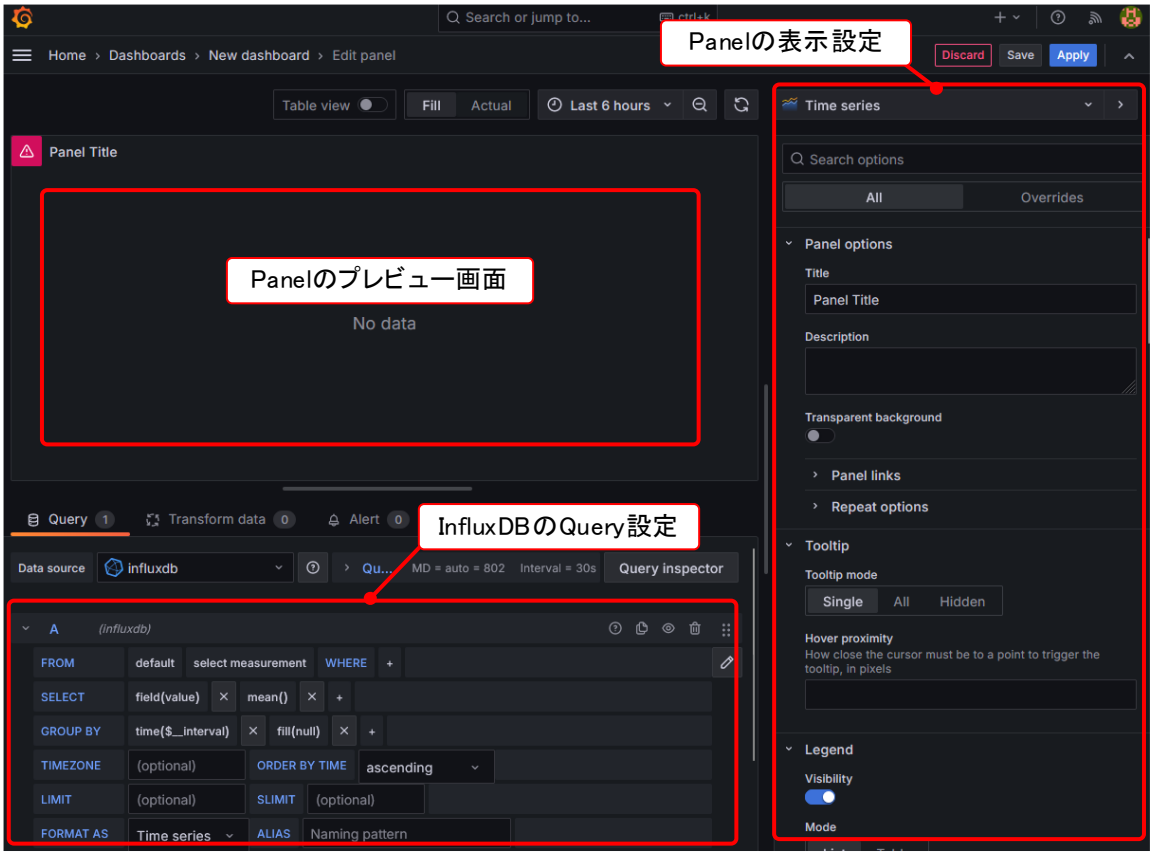


ダッシュボード作成画面について説明します。

ダッシュボードに表示するグラフやチャート、メーターなどを Panel（パネル）と呼びます。

Panel を 1 つずつ完成させて 1 つのダッシュボードを作成します。

- InfluxDB の Query 設定
  Panel に表示するデータを設定します。
- Panel の表示設定
  Panel の表示設定を行います。設定する項目は、グラフ種類、Panel タイトル、軸、表示色の設定などです。
- Panel のプレビュー画面
  選択したデータおよび Panel の表示設定がリアルタイムで確認できます。



Copyright © 2024 YITJC All Rights Reserved.

手順④～Query 設定～

「InfluxDB の Query 設定」セクションで Panel に表示するデータを設定します。

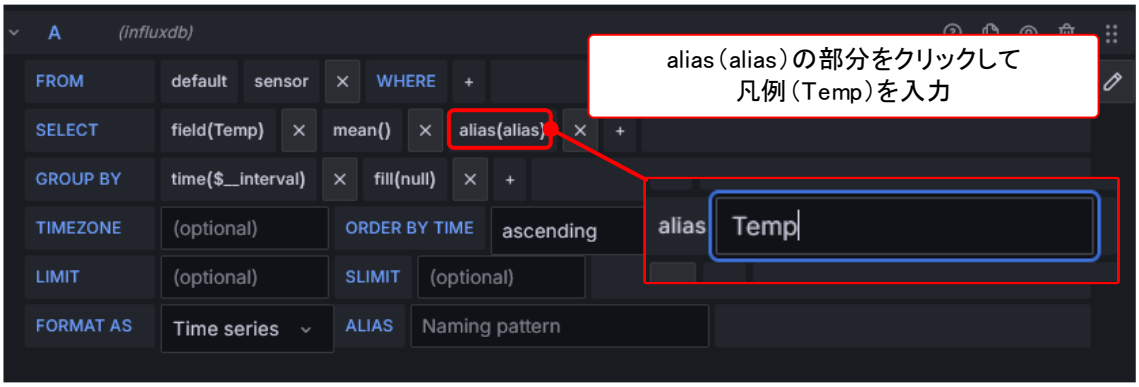
「select measurement」をクリックすると、データベース「sensor」が表示されるので、クリックしてください。

続いて、「field」をクリックすると、「sensor」にある field（'Temp', 'Hum', 'Press'）が表示されるので、「Temp」をクリックしてください。

最後に、「+」をクリックして「alias」と入力してください。

続いて、「alias（alias）」と表示されている部分をクリックして、「凡例」を入力します。

ここでは、凡例として「Temp」を設定しています。



ここまで設定すると、「Panel プレビュー画面」に Temp データがプロットされます。

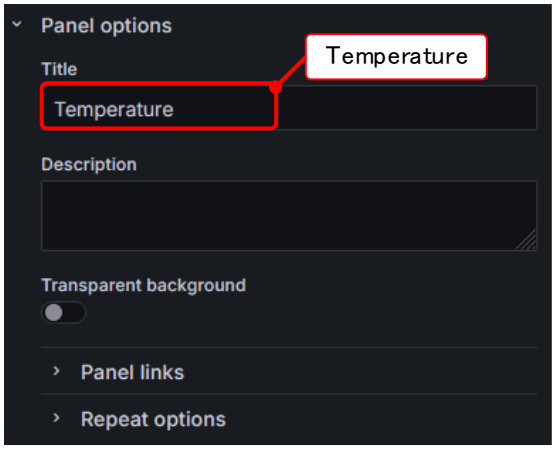
Copyright © 2024 YITJC All Rights Reserved.

手順⑤～Panel 表示設定～

「Panel の表示設定」セクションで Panel の表示設定を行います。

- Panel Options

  ○Title　　: Panel タイトルを入力してください。



- Tooltip

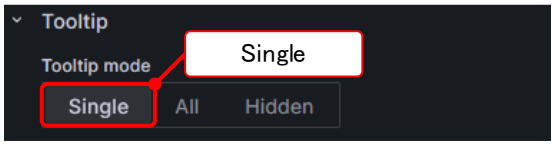
  マウスをデータに合わせたときに表示される情報（Tooltip）の表示方法を設定します。

  次のいずれかを選択してください。

  ○Single　：カーソルを合わせたデータに関する情報だけを表示

  ○All　　：カーソルを合わせたデータに加えて、同じ x 軸上にあるデータの情報も表示
  （複数のデータが同じ Panel に表示されている場合に使用する）

  ○Hidden　：Tooltip を表示しない



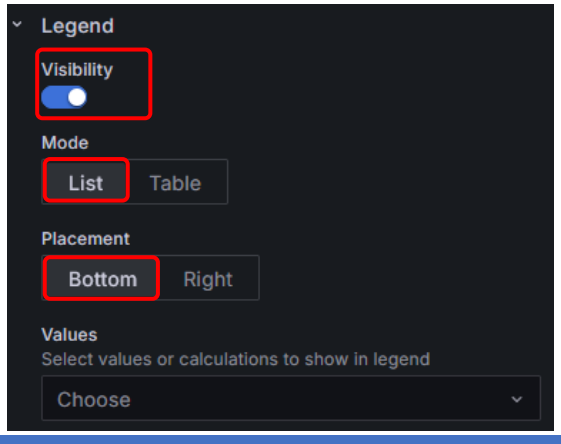
- Legend

  凡例の表示設定を行います。

  ○Visibility　　：「表示」または「非表示」の選択

  ○Mode　　：凡例の表示形式、リスト形式（List）または表形式（Table）を選択

  ○Placement　　：凡例の表示位置、Panel の下（Bottom）または右（Right）を選択



Copyright © 2024 YITJC All Rights Reserved.

- Axis

  軸の設定を行います。

  ○Time Zone　　: x 軸のタイムゾーンを設定

  ○Placement　　: y 軸の表示位置を選択

  左（Left）、右（Right）、表示しない（Hidden）を選択できます。

  ○Label　　: y 軸のラベルを設定

  ○Width　　: y 軸の表示範囲を設定

  小さな値（1～20）に設定した場合、ラベルと軸が重なってしまいます。

  ○Show Grid Lines　: グリッド線の表示設定

  ○Color　　: 軸ラベルの表示色設定

  ○Show border　　: y 軸の目盛線表示設定

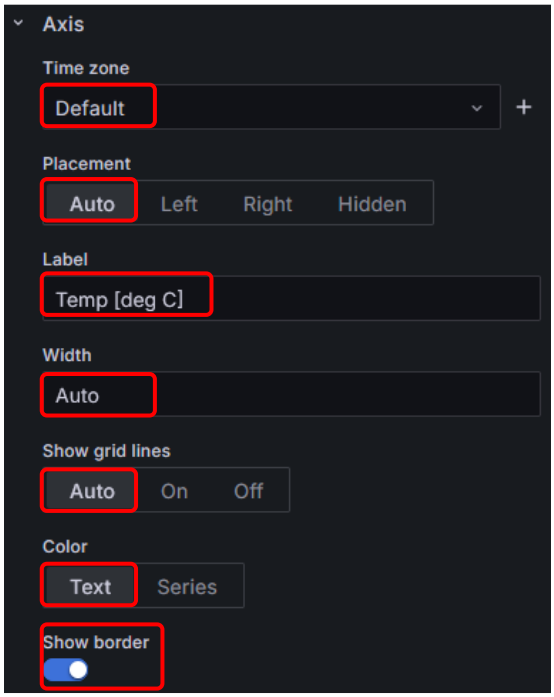
  ○Scale　　: 軸のスケール設定

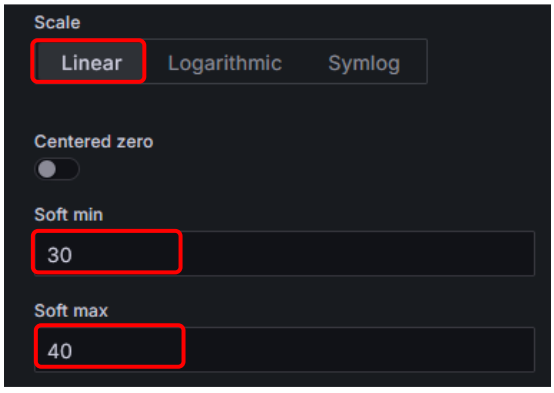
  線形（Linear）、対数（Logarithmic）、対象対数（Symlog）を選択できます。

  ○Centered zero　　: y 軸を「0」中心に設定

  ○Soft min　　: y 軸の最小値を設定

  ○Soft max　　: y 軸の最大値を設定



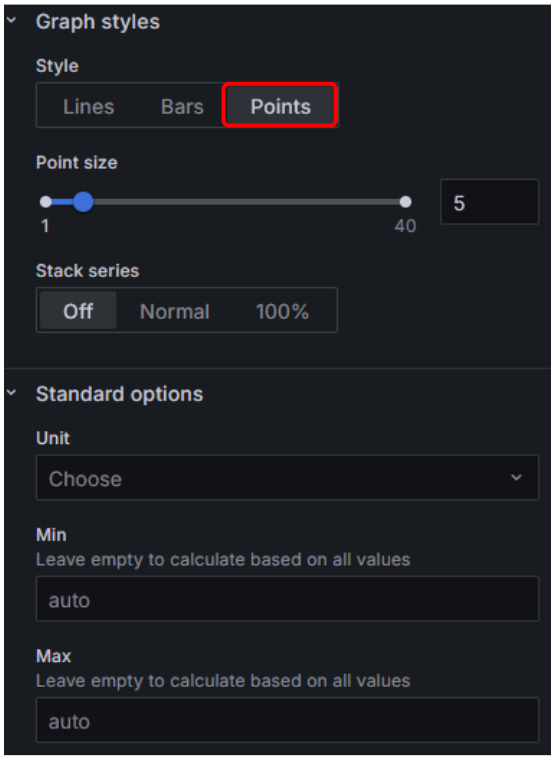


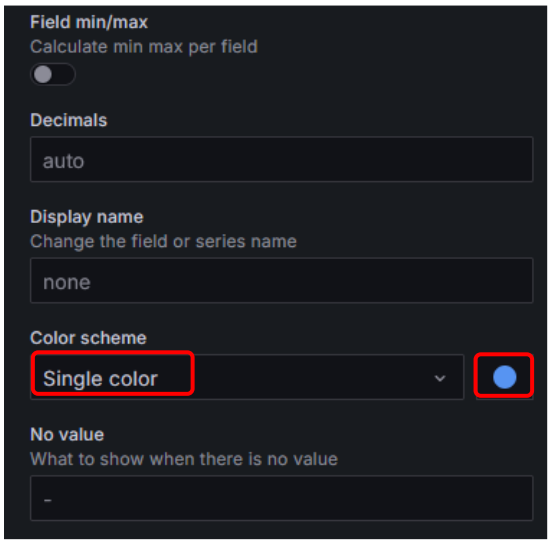
Copyright © 2024 YITJC All Rights Reserved.

- Graph Styles

  ○Style　　：グラフのスタイルを設定
  データを線で接続する「Lines」、データを棒状のバーで表示する「Bars」、
  データをポイント（点）で表示する「Points」を選択できます。

  ○Color Scheme　：データポイントの色を設定
  「Single Color」を選択し、右側○からカラーパレットから色を選択できます。

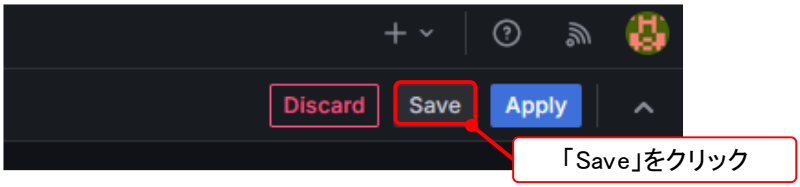




Copyright © 2024 YITJC All Rights Reserved.

手順⑥～ダッシュボード保存～

Panel 表示設定が完了したら、画面右上にある「Save」をクリックしてください。



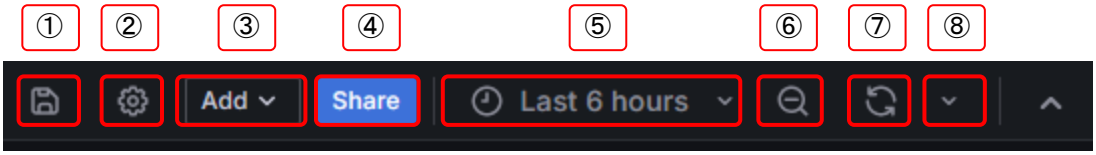
「Save dashboard」の画面が表示されるので、「Title」を入力したあと、「Save」をクリックしてください。

「Save」したあと、ダッシュボードの画面に切り替わります。

Copyright © 2024 YITJC All Rights Reserved.

参考～Dashboard メニュー画面～

「Dashboard メニュー」画面について説明します。

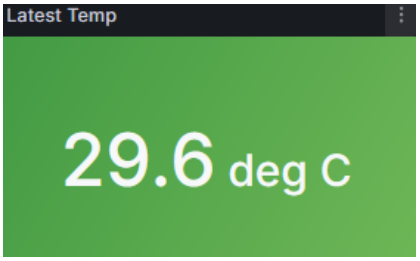


番号	機能
①	ダッシュボード保存
②	ダッシュボード設定
③	Panel の追加（新たに Panel を作成）
④	ダッシュボードの URL や画面の Snap shot を共有
⑤	Panel に表示する期間（日付時刻を指定することも可能） 例） Last 6 hours：6 時間前から現在までのデータを表示
⑥	Panel に表示する時間軸の短縮
⑦	データの更新（自動更新が Off の場合は、クリックするとデータが更新される）
⑧	データ自動更新設定

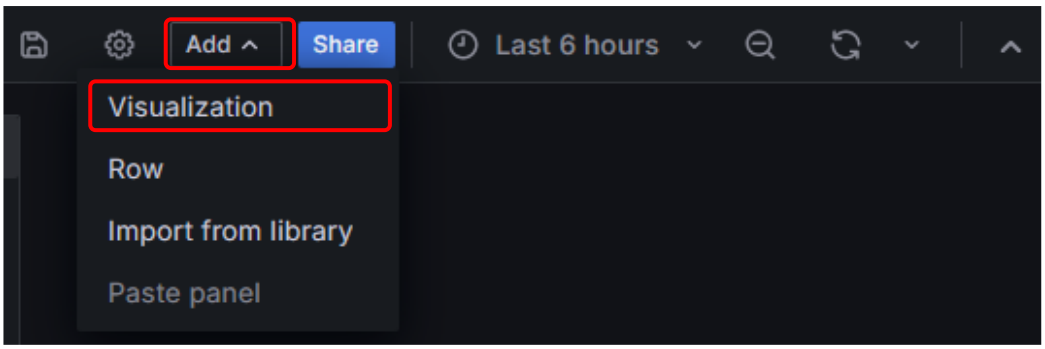
手順⑦～Panel 追加～

作成したダッシュボードに新たな Panel を追加します。

新たに追加する Panel として、最新の温度データを表示する Panel を作成します。



「Dashboard メニュー」画面③の「Add…」をクリックしたあと、「Visualization」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

手順⑧～Query 設定～

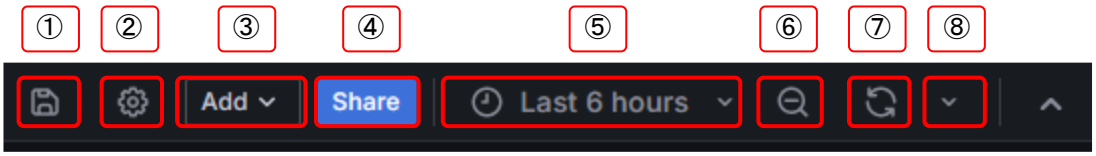
手順④と同じく「Temp」データを Query に設定してください。

続いて、関数を指定します。

デフォルトの関数は「mean()」となっています。

mean()に設定した場合は、ダッシュボード画面⑤で設定した表示期間における「平均値」を表示します。

～参考～

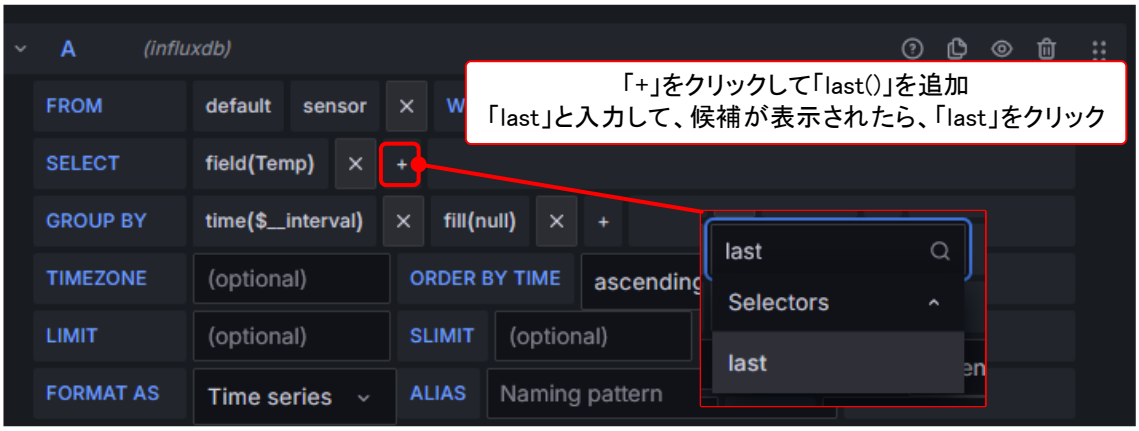


今回は、最新の温度データを表示する Panel を作成するため、「last()」関数を使用します。

そのため、mean()関数を削除してから last()関数を追加します。

続いて、「+」をクリックして「last()」を追加してください。

※「last」と入力すると、候補が表示されるので、その中から「last」をクリックしてください。



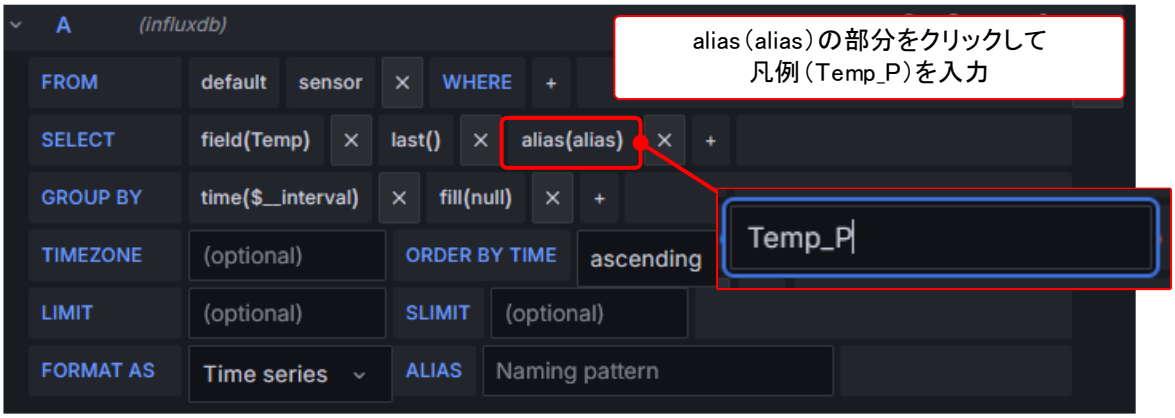
Copyright © 2024 YITJC All Rights Reserved.

「alias（alias）」と表示されている部分をクリックして、「凡例」を入力します。

～注意～

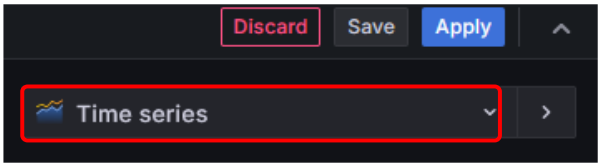
同じデータを複数の Panel で使用する場合は、凡例（alias）を別名にしてください。

ここでは、凡例として「Temp_P」を設定しています。

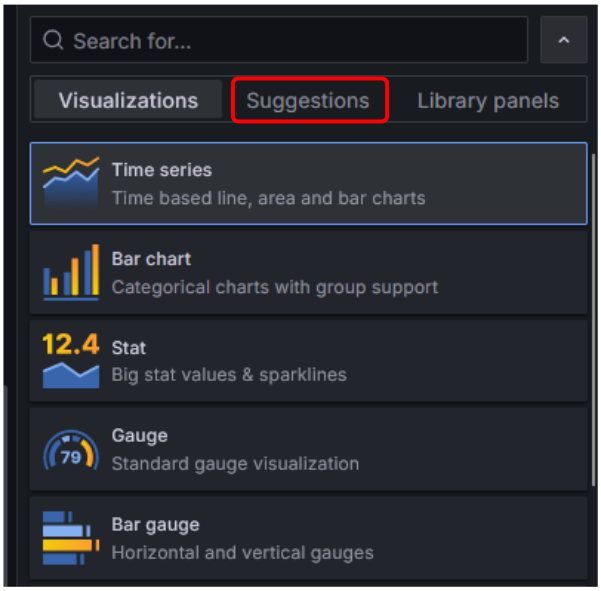


手順⑨～グラフ選択～

画面右上の「Time Series」タブをクリックしてください。

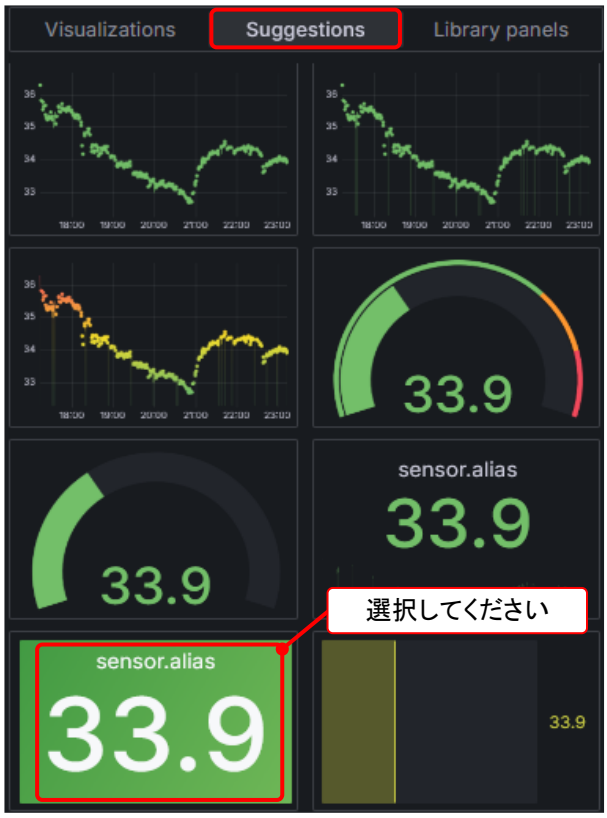


さまざまな種類のグラフが表示されますが、「Suggestions」タブをクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

ここでは、Grafana が Query のデータ傾向からさまざまなグラフを提案（Suggestion）しています。
この中から最新の温度データを表示する Panel を選択してください。

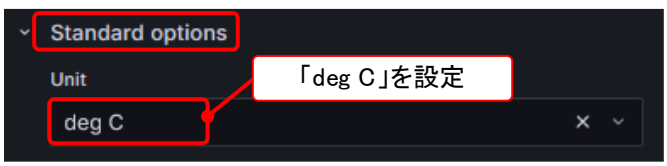


手順⑩～Panel 表示設定～

「Suggestions」から選択した Panel は、ある程度設定が完了しているため、Panel 設定の手間を省略できます。ここでは、次の項目だけ設定して、Panel の設定を完了します。

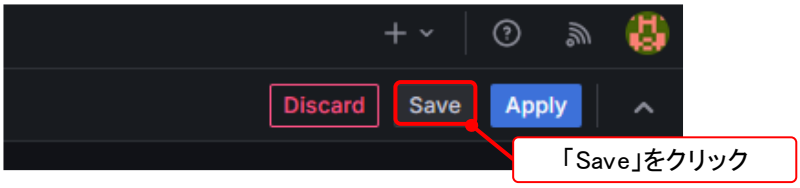
- Panel Options
  Panel のタイトルを「Latest Temp」に設定します。

- Standard Options
  Unit（単位）「deg C」に設定します。

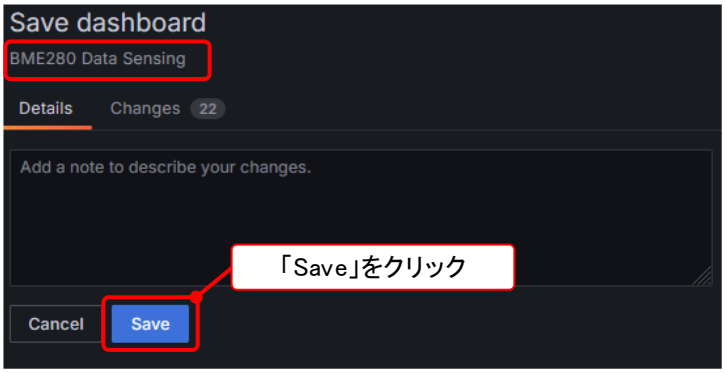


Copyright © 2024 YITJC All Rights Reserved.

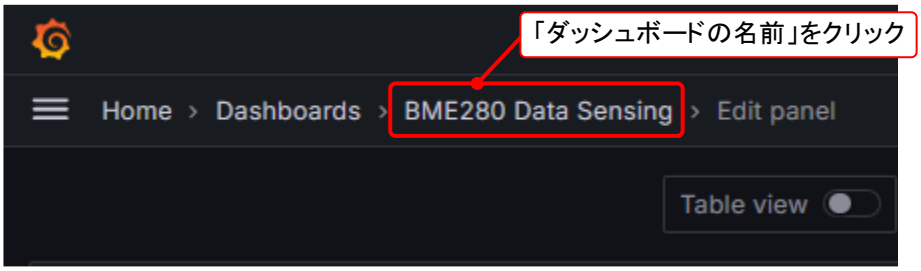
Panel 表示設定が完了したら、画面右上にある「Save」をクリックしてください。



ダッシュボードの名前を確認して「Save」をクリックしてください。



「ダッシュボードの名前」をクリックして、ダッシュボード画面に戻ります。



ダッシュボードには 2 つの Panel が作成され、「時系列データ」と「最新のデータ」が可視化された状態です。

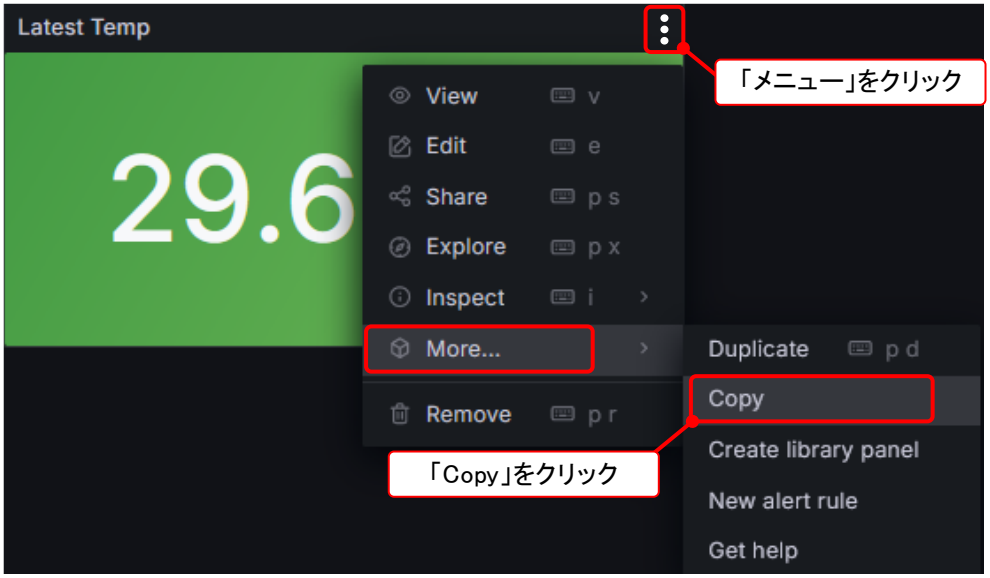
Copyright © 2024 YITJC All Rights Reserved.

手順⑪～Panel 複製～

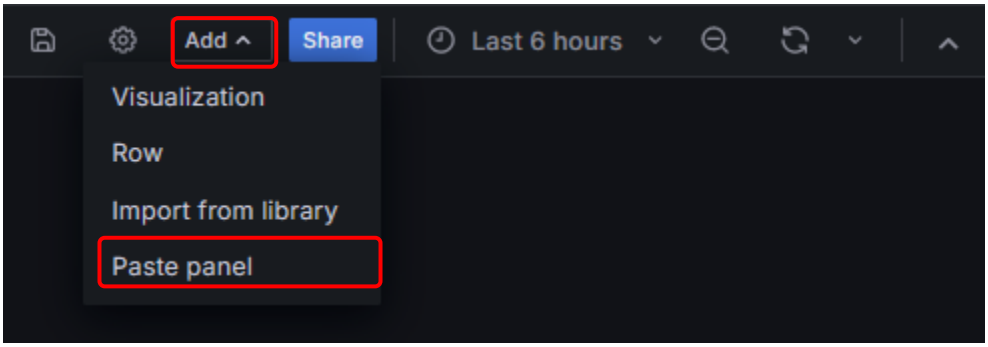
「湿度」や「気圧」のデータも同じ形式の Panel を作成してダッシュボードを完成させます。同じ形式の Panel を作成する場合、Panel の「複製」機能を利用すると、比較的短時間で Panel を作成できます。

複製したい Panel の右上に「メニュー」ボタンがあるので、クリックしてください。

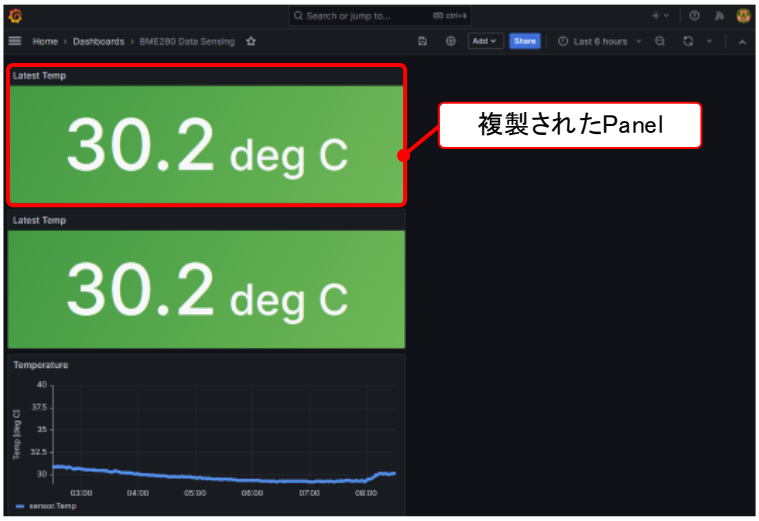
「More…」の中から「Copy」をクリックします。



続いて、「Dashboard メニュー」画面③の「Add…」をクリックしたあと、「Paste Panel」をクリックしてください。



ダッシュボード画面に複製した Panel が表示されます。



Copyright © 2024 YITJC All Rights Reserved.

複製した Panel の「メニュー」ボタンから「Edit」をクリックしてください。

このあとは、ダッシュボード作成画面で必要な編集を行います。

- InfluxDB の Query 設定
  field を編集（湿度（Hum）または気圧（Press）を選択）
  alias（凡例）を編集
- Panel の表示設定
  Panel タイトル、Unit（単位）、軸の最小値・最大値、表示色などを編集します。

手順⑫～ダッシュボード完成～

「湿度」、「気圧」の Panel を追加したら、全体のレイアウトを調整します。

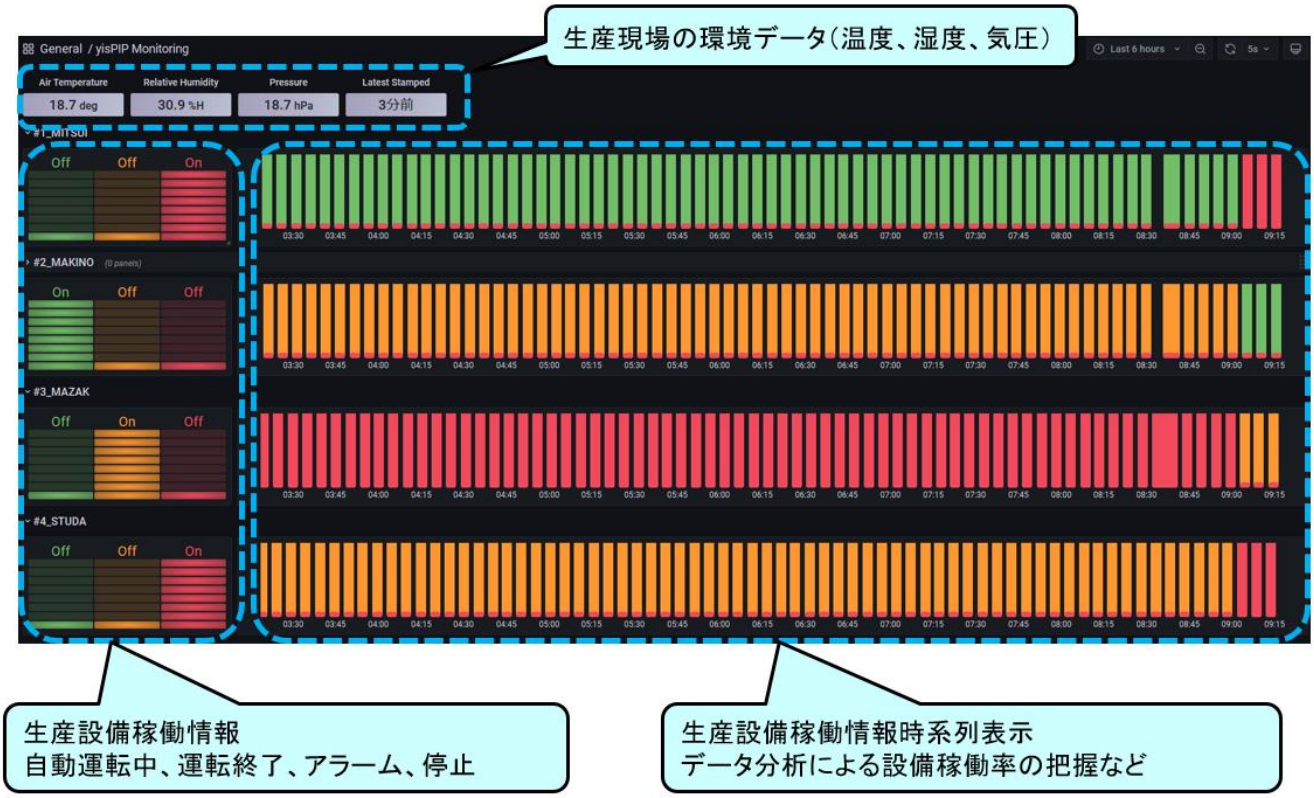
- レイアウト調整
  Panel の移動は、Panel のタイトル付近にカーソルを合わせるとマウスで移動できます。
  Panel のサイズ変更は、Panel 右下をマウスで掴んで調整できます。
- 表示期間、自動更新の設定
  「Dashboard メニュー」画面で「表示期間」と自動更新の時間を設定します。
- ダッシュボードの保存
  ダッシュボードが完成したら忘れずに「保存」をしてください。

Copyright © 2024 YITJC All Rights Reserved.

参考～生産設備稼働状況の監視～

応用例として、生産設備稼働状況のリアルタイム監視を行うダッシュボードを紹介します。生産設備の稼働状況は、生産設備に装備されているパトライトの点灯信号データ（緑灯、橙灯、赤灯の 3 種類の点灯信号）から把握します。パトライトの点灯信号は、Isolator Board 経由で Raspberry Pi がデータ収集してデータベースに蓄積します。蓄積したデータを Grafana で可視化してダッシュボードを作成します。

このダッシュボードを構成している各 Panel の作成方法を紹介します。

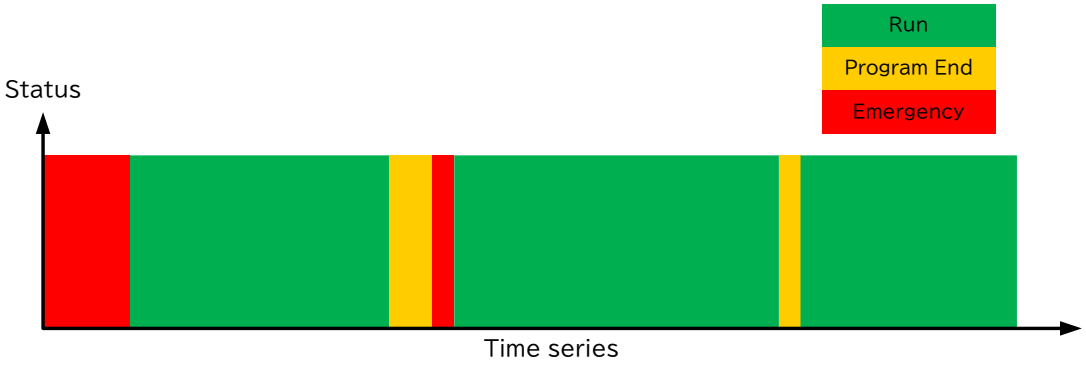


Copyright © 2024 YITJC All Rights Reserved.

参考～Panel 作成例①～

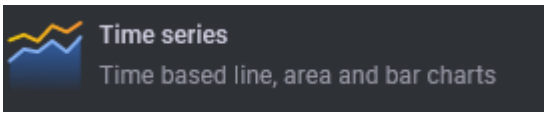
手順①～作成する Panel のイメージ～

ここでは、1 つの Panel に複数のデータを登録する方法について説明します。



手順②～Panel の種類～

「Time Series」を選択してください。



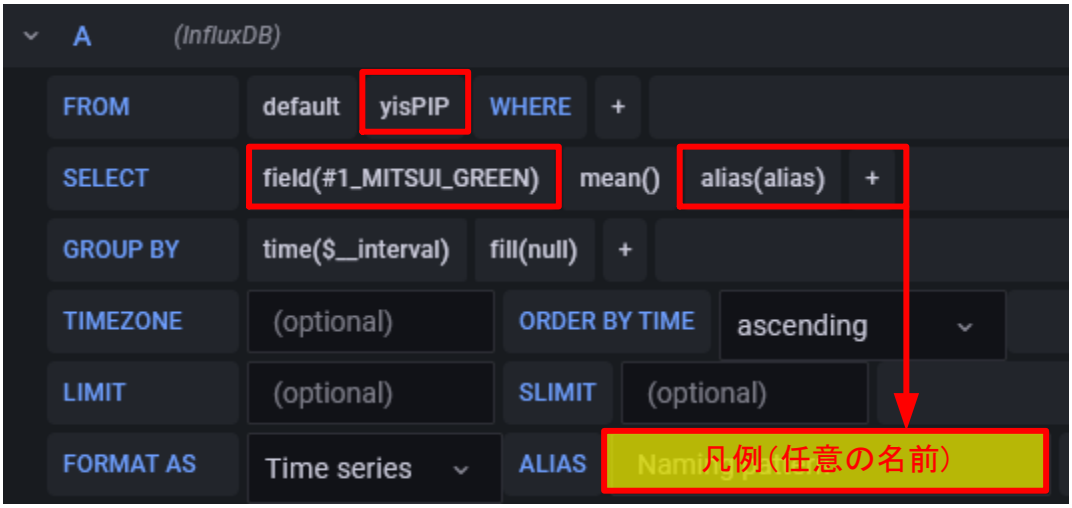
手順③～凡例の設定～

「Alias（エイリアス）」を選択し、任意の名前（凡例）を設定してください。

※Query を追加したとき、凡例には同じ名前が自動的に割り当てられてしまいます。

そのため、データごとに線種、マーカーなどを設定することができません。

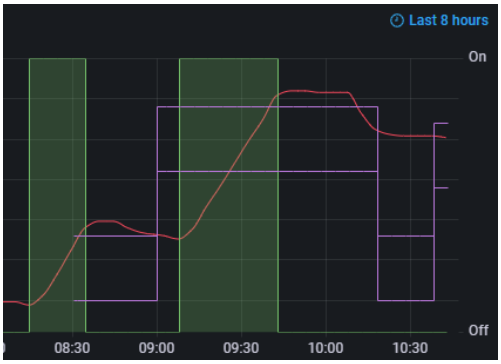
（同じ名前が割り当てられている限り、同じ線種、マーカーが適用されてしまいます。）



Copyright © 2024 YITJC All Rights Reserved.

手順④～Panel の設定～

- Panel Options
  グラフタイトル「Title」を記入
- Tooltip
  マウスをデータに合わせたときに表示される情報（Tooltip）の表示方法を設定します。
- Legend
- 凡例の表示設定を行います。
- Graph Style
  - Style: グラフスタイル
    「Bars」を選択
  - Bar alignment
    「Center」を選択
  - Line interpolation (線形補間)
  - Line width: 線幅
  - Fill opacity: 系列の（塗りつぶし）透明度
    （参考）イメージ↓

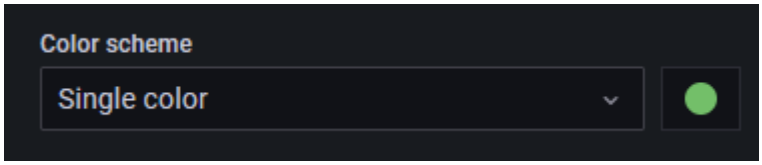


  - Gradient mode: None, Opacity, Hue, Scheme
    （参考）イメージ↓

  - Show points: マーカー（点）表示
    「Never」を選択
  - Point size: マーカーサイズ指定
  - Stack series: 積み上げ表示
    「Off」を選択

Copyright © 2024 YITJC All Rights Reserved.

- Axis
  - Placement: 軸位置
    「Hidden」を選択
  - Label: 軸ラベル
  - Width: 軸の表示幅
  - Soft min: 最小値「0」に設定
  - Soft max: 最大値「1」に設定
    ※Hidden に設定した場合でも、Soft min, Soft max の設定は必要
  - Show grid lines: グリッド表示
    「off」を選択
  - Scale: スケール設定
    「Linear」を選択
- Standard Options
  Color scheme は、線色・マーカー色を変更することができます。
  Single color を選択し、右側〇をクリックし、カラーパレットから色を選択
  ※複数のデータの色分けについては、後述します。



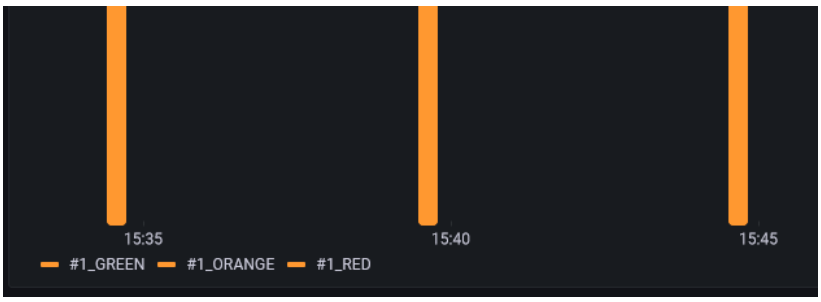
  - Unit: 単位選択
- Thresholds
  閾値の設定
  ※Base の部分の色を変更しても、線色・マーカー色が変化する訳ではないので注意してください。
  Default で 80 が設定されているが、不要であれば、削除（ゴミ箱）をクリック
  追加する場合は、「+ Add threshold」をクリックしてください。
  - Thresholds mode
    Absolute（絶対値）、 Percentage（パーセンテージ）から選択
  - Show thresholds
    閾値の表示設定

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～データごとの書式設定～

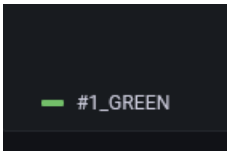
複数のデータを登録した際の色変更

1 つのグラフ上に 3 つのデータを登録すると、同じ色で表示されます。

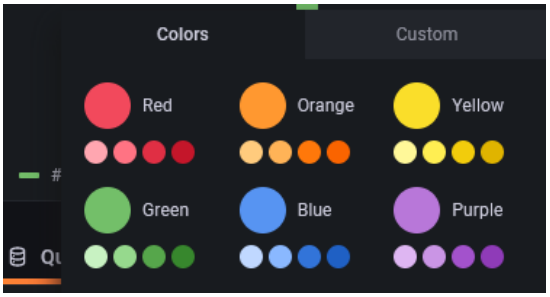


この状態から、各データの色を変更する方法について説明します。

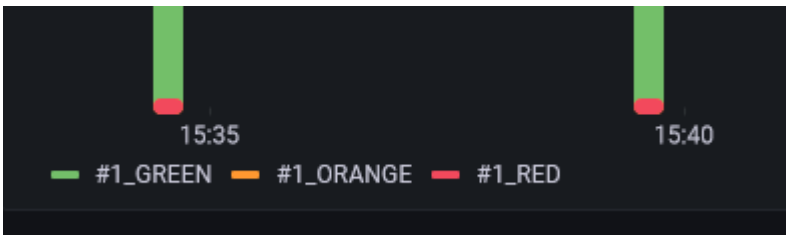
凡例の表示設定を「On」にした状態で、凡例をクリックしてください。



カラーパレットから色を選択できます。



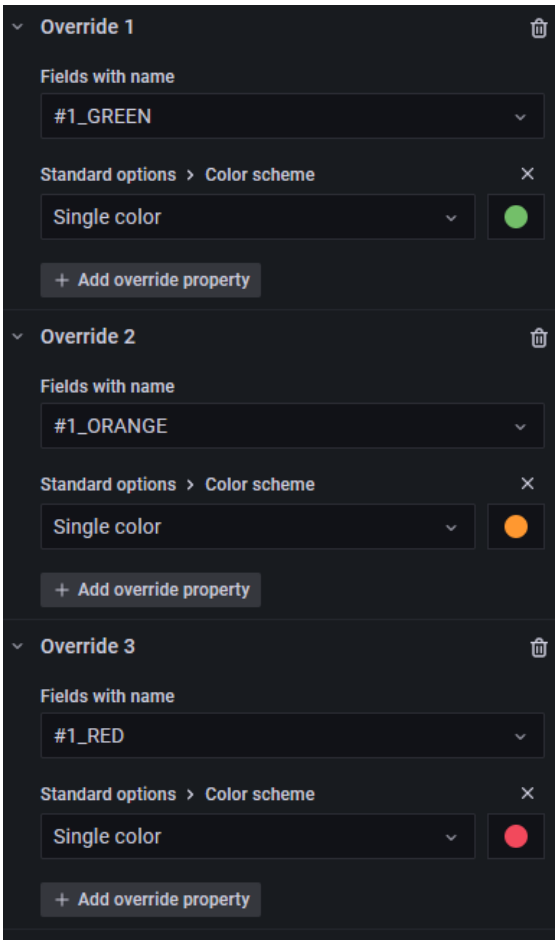
3 つのデータを色分けすると、次のように表示されます。



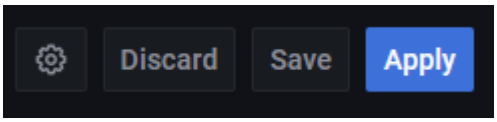
Copyright © 2024 YITJC All Rights Reserved.

色変更した後は、Panel 設定画面に「Override」の設定が追加されます。

※Override の設定から、各データの色分けを行うこともできます。



グラフの設定が完了したら、画面右上から、



Save: 保存

Apply: 保存して、Dash Board の画面に戻る

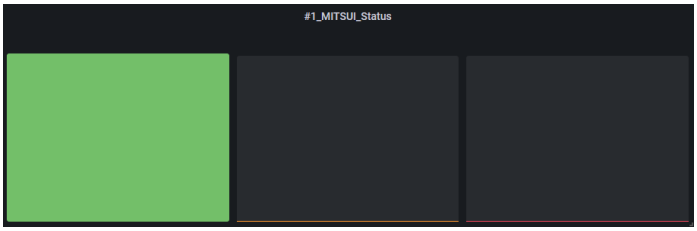
いずれかをクリックして Panel の作成を終了してください。

Copyright © 2024 YITJC All Rights Reserved.

参考～Panel 作成例②～

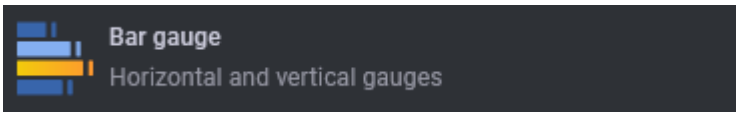
手順①～作成する Panel のイメージ～

ここでは、最新のデータだけを Panel に表示する方法について説明します。



手順②～Panel の種類～

「Bar gauge」を選択してください。



手順③～凡例の設定～

「Alias（エイリアス）」を選択し、任意の名前（凡例）を設定してください。

※Query を追加したとき、凡例には同じ名前が自動的に割り当てられてしまいます。

そのため、データごとに線種、マーカーなどを設定することができません。

（同じ名前が割り当てられている限り、同じ線種、マーカーが適用されてしまいます。）

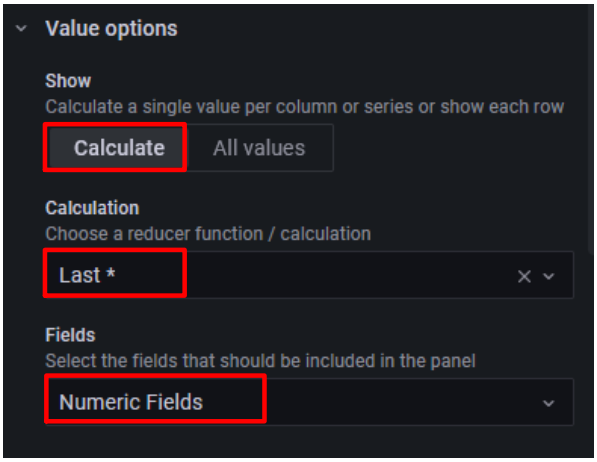
※別のパネルで同じデータを使用する場合は、凡例を別名に設定してください。

Copyright © 2024 YITJC All Rights Reserved.

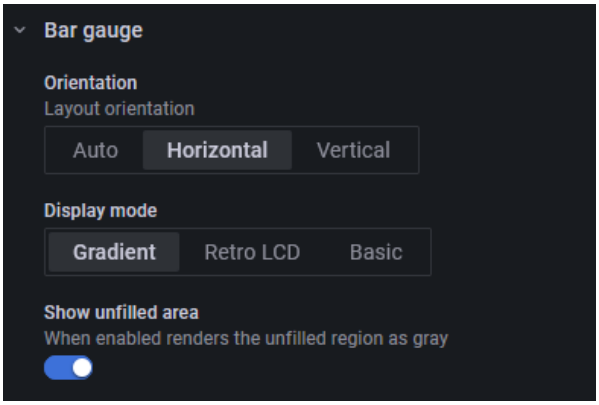
手順④～Panel の設定～

- Panel Options
  グラフタイトル「Title」を記入
- Query 設定
  手順③凡例の設定で信号灯 3 つのデータを登録してください。

- Value options
  3 つのデータについて、最新の状態を表示するために、次の設定をしてください。



- Bar Gauge
  - Orientation: Bar 表示方向設定

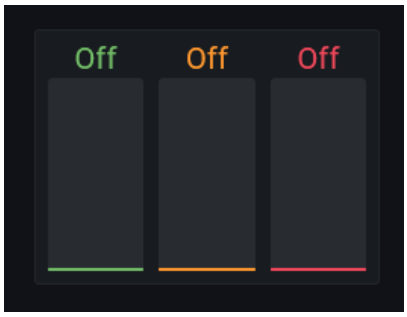


Copyright © 2024 YITJC All Rights Reserved.

- Display Mode
  - Basic に設定

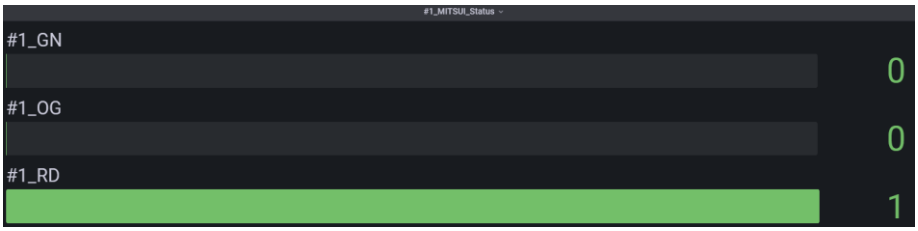
    ※Retro に設定した場合、OFF 表示でもバーの状態が「1」になってしまうことがあります（バグ）。

Retro LCD



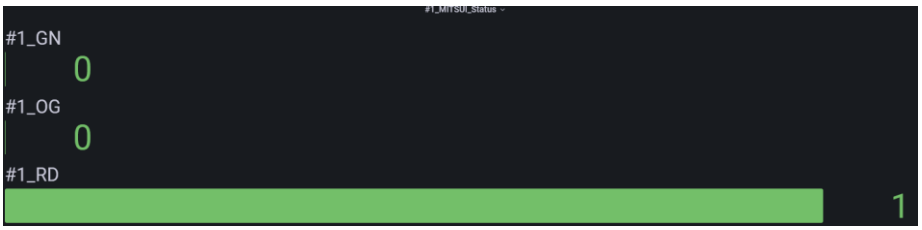
Basic

Horizontal に設定すると、次のように表示されます。



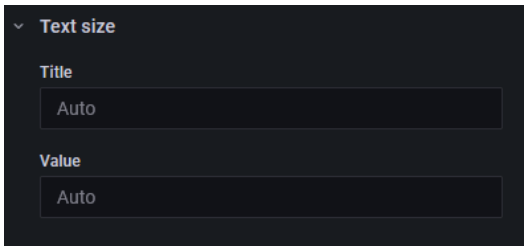
  - Show unfilled area

    チェックを外すと次のように表示されます（データ 0 のときにグレー背景が表示されない）。



- Text Size

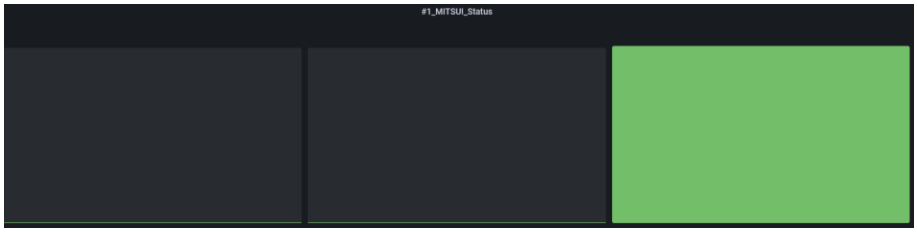
  Title（凡例）、Value（データ）を表示させたくない場合は、「0」を設定してください。



Copyright © 2024 YITJC All Rights Reserved.

（参考）Title: Auto, Value:0 の表示設定

（参考）Title:0, Value: 0 の表示設定



- Standard options

  3 つのデータの色分けについては、手順⑤の「Overrides」機能で行います。

  Color scheme は、「From thresholds (by value)」が選択されていますが、気にせずそのままで OK です。
- Thresholds

  閾値の設定

  ※Base の部分の色を変更しても、線色・マーカー色が変化する訳ではないので注意してください。

  Default で 80 が設定されているが、不要であれば、削除（ゴミ箱）をクリック

  追加する場合は、「+ Add threshold」をクリックしてください。
  - Thresholds mode

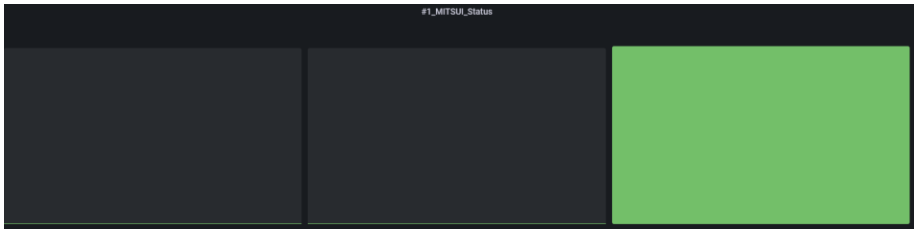
    Absolute（絶対値）、 Percentage（パーセンテージ）から選択
  - Show thresholds

    閾値の表示設定

手順⑤～データごとの書式設定～

- 3 つのデータの色変更

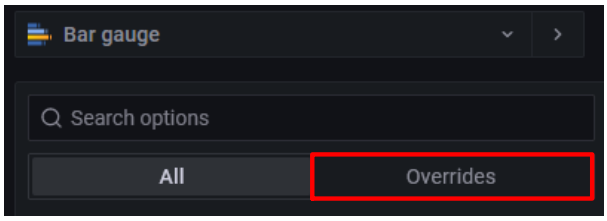
  現在の状態は、同じ色で表示されています。



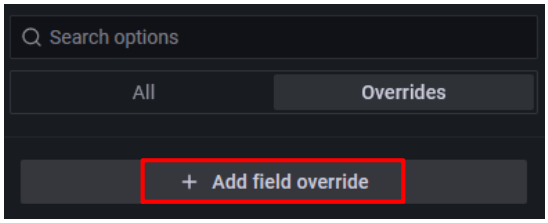
この状態から、各データの色を変更する方法について説明します。

Copyright © 2024 YITJC All Rights Reserved.

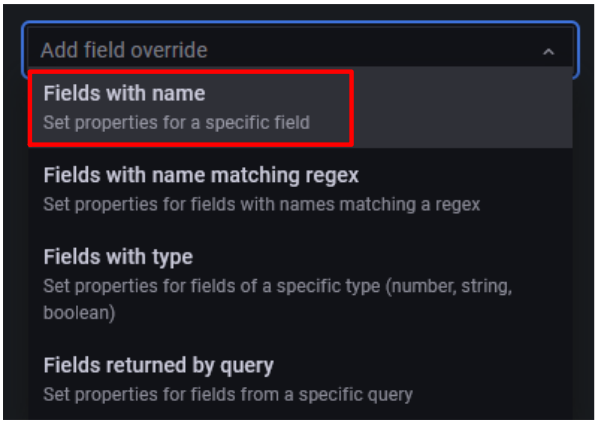
「Panel の表示設定」から「Overrides」をクリックしてください。



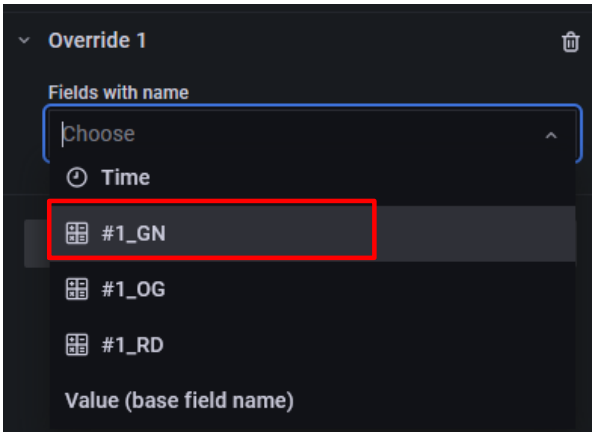
「+ Add field override」をクリックしてください。



「Fields with name」をクリックしてください。



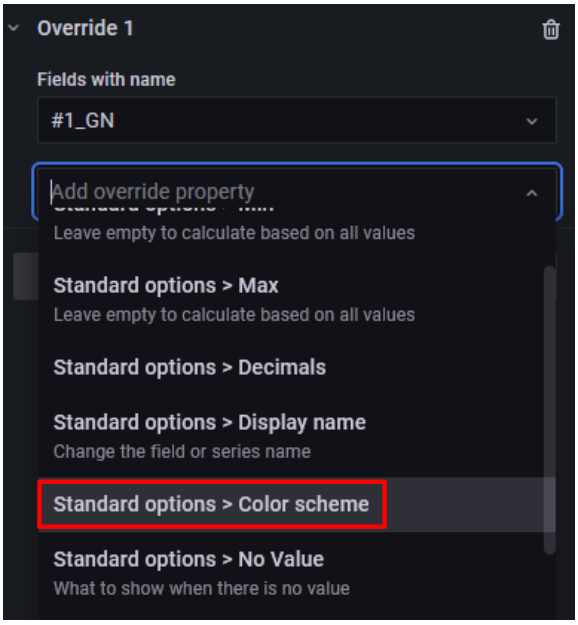
色を設定するデータ（凡例）を選択してください。



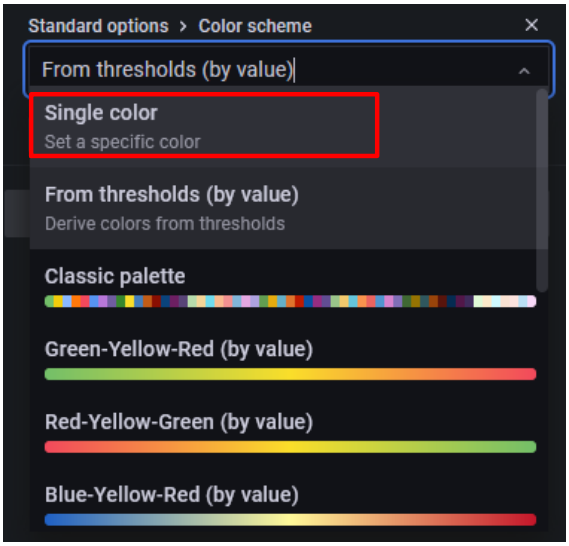
Copyright © 2024 YITJC All Rights Reserved.

「+ Add override property」をクリックして、設定項目を選択します。

色を変更するので、「Color scheme」を選択してください。



「Single Color」を選択し、色を選択してください。



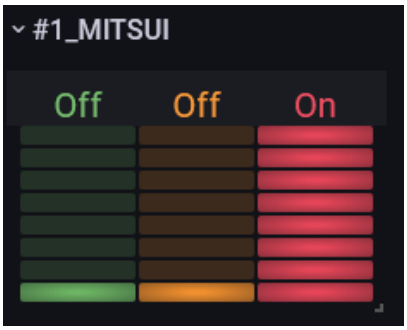
Copyright © 2024 YITJC All Rights Reserved.

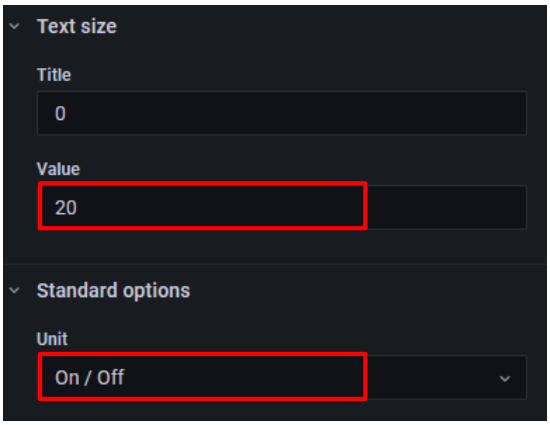
3 つのデータについて色を設定すると、Grafana 上でシグナルタワー（信号灯）の状態をリアルタイム監視できます。

信号の状態が切り替わると、次のように表示されます（赤から緑に変更した状態）。

～参考～

On/Off と表示したい場合は、Unit（単位）から「On/Off」を選択してください。





Copyright © 2024 YITJC All Rights Reserved.

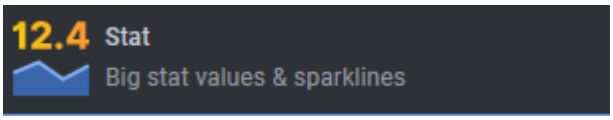
参考～Panel 作成例③～

手順①～作成する Panel のイメージ～

ここでは、最新の数値データを Panel に表示する方法について説明します。

手順②～Panel の種類～

Stat を選択してください。



手順③～凡例の設定～

「Alias（エイリアス）」を選択し、任意の名前（凡例）を設定してください。

※Query を追加したとき、凡例には同じ名前が自動的に割り当てられてしまいます。

そのため、データごとに線種、マーカーなどを設定することができません。

（同じ名前が割り当てられている限り、同じ線種、マーカーが適用されてしまいます。）

※別のパネルで同じデータを使用する場合は、凡例を別名に設定してください。

Copyright © 2024 YITJC All Rights Reserved.

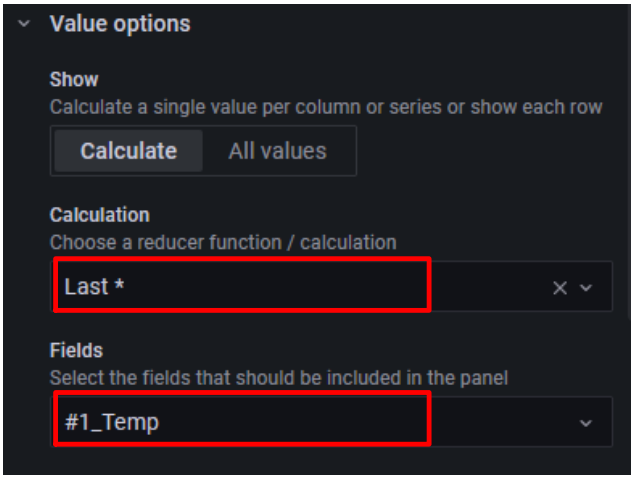
手順④～Panel の設定～

- Panel Options

  グラフタイトル「Title」を記入

- Value options

  最新の状態を表示するために、次の設定をしてください。



- Stat styles
  - Orientation

    グラフ表示を行わないので、何を選択しても構いません。
  - Text mode

    数値だけを表示するので、「Value」を選択
  - Color mode

    「Background」を選択すると、背景色が追加されます。

  - Graph mode

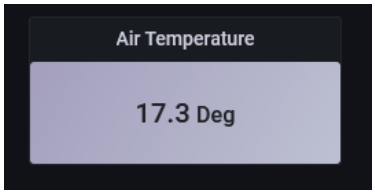
    「None」を選択
  - Text alignment

    「Auto」を選択

- Text size

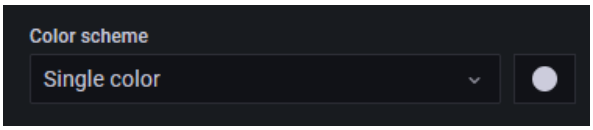
  「Auto」を選択

  タイトルと同じサイズに設定したい場合は、同じ値を設定してください。



Copyright © 2024 YITJC All Rights Reserved.

- Standard options
  - Unit(単位)を設定
  - Color Scheme: テキスト色を設定



Copyright © 2024 YITJC All Rights Reserved.

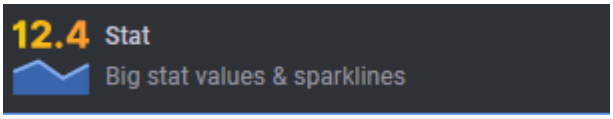
参考～Panel 作成例④～

手順①～作成する Panel のイメージ～

ここでは、最後にデータ更新した時刻を Panel に表示する方法について説明します。

手順②～Panel の種類～

Stat を選択してください。



手順③～凡例の設定～

「Alias（エイリアス）」を選択し、任意の名前（凡例）を設定してください。

ここは、時間の情報が必要となるので、どのデータを選択しても構いません。

※Query を追加したとき、凡例には同じ名前が自動的に割り当てられてしまいます。

そのため、データごとに線種、マーカーなどを設定することができません。

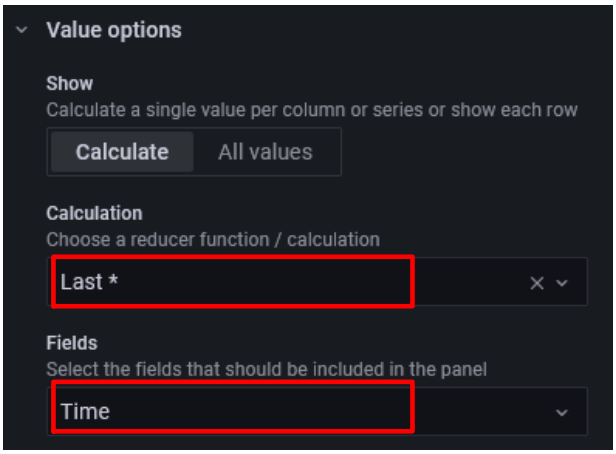
（同じ名前が割り当てられている限り、同じ線種、マーカーが適用されてしまいます。）

※別のパネルで同じデータを使用する場合は、凡例を別名に設定してください。

Copyright © 2024 YITJC All Rights Reserved.

手順④～Panel の設定～

- Panel Options
  グラフタイトル「Title」を記入
- Value options
  最新の更新時刻を表示するために、次の設定をしてください。

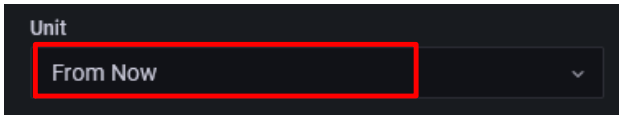


- Stat styles
  - Orientation
    グラフ表示を行わないので、何を選択しても構いません。
  - Text mode
    時間だけを表示するので、「Value」を選択
  - Color mode
    「Background」を選択すると、背景色が追加されます。

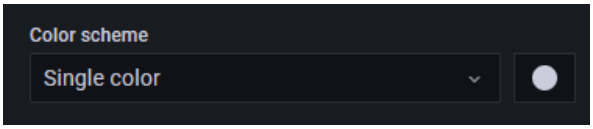
  - Graph mode
    「None」を選択
  - Text alignment
    「Auto」を選択
- Text size
  「Auto」を選択
  タイトルと同じサイズに設定したい場合は、同じ値を設定してください。

Copyright © 2024 YITJC All Rights Reserved.

- Standard options
  - Unit（単位）を設定
    「From Now」を選択してください。



  - Color Scheme: テキスト色を設定



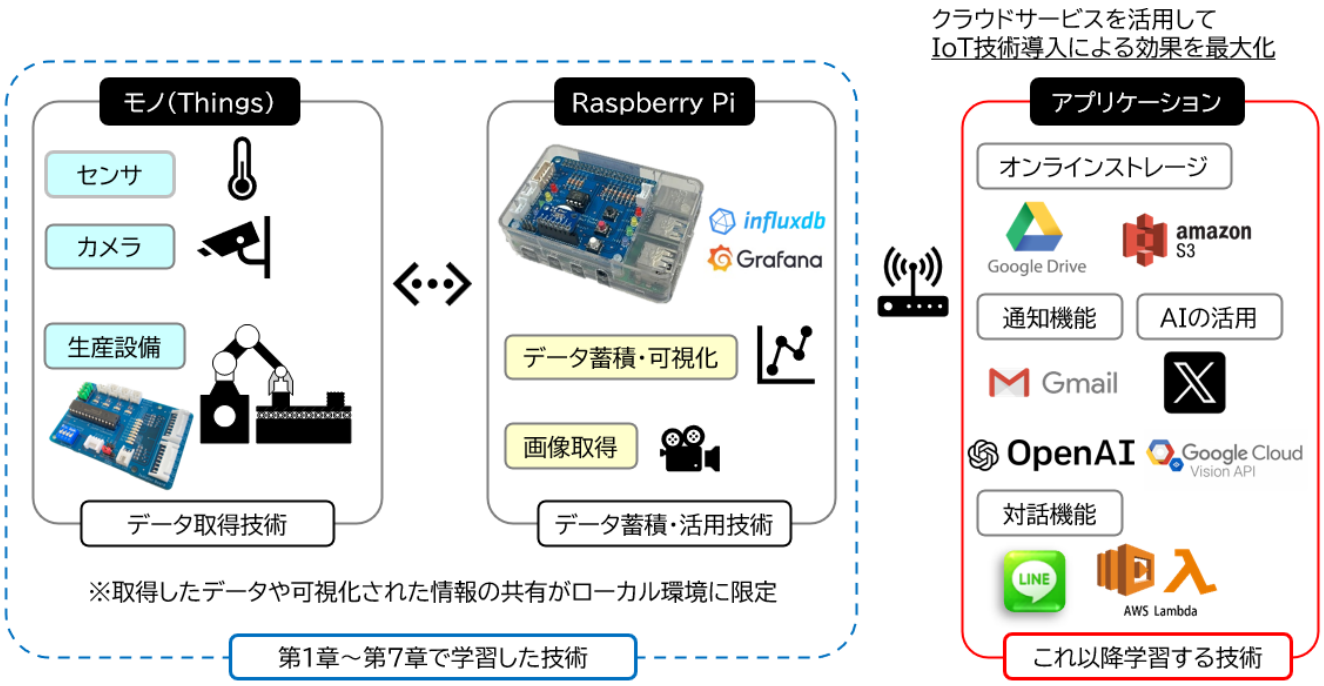
「Panel 作成例①～④」をまとめると、次のようなダッシュボードが完成します。
次のダッシュボードは、生産設備 5 台分の稼働状況をダッシュボード 1 画面で可視化しています。

Copyright © 2024 YITJC All Rights Reserved.

第 8 章 クラウドサービスの活用～オンラインストレージ～

これまでの学習を通して、センサやカメラ、生産設備から「データを取得する技術」や取得した「データを蓄積して活用する技術」を習得しました。しかし、取得したデータや可視化された情報は Raspberry Pi に保存されているため、データ処理や分析を行うためのリソースが Raspberry Pi に限定されています。

ここからは、Raspberry Pi が取得したデータを活用するため、クラウドサービスを活用する方法について学習します。具体的には、「オンラインストレージへのデータアップロード」（第 8 章）や「通知機能」、「OpenAI」、「Google Cloud Vison」（第 9 章）、「モノとの対話機能」（第 10 章）などを実装する方法について学習します。



オンラインストレージを活用することで、IoT システムが収集したデータの「可用性」や「安全性」、「スケーラビリティ」が向上します。具体的には、センサやカメラ、生産設備などのデータがオンラインストレージに保存され、いつでもどこからでもデータにアクセスできます。

本講座では、オンラインストレージに「Google Drive」を使用した IoT システムを構築します。Google Drive は API（Application Programming Interface）が無料公開されているため、API を利用することで開発期間を短縮できます。ここでは、「Google Drive API」を使用して Google Drive にファイルをアップロードおよびダウンロードする方法について学習します。

Copyright © 2024 YITJC All Rights Reserved.

8.1 ファイルアップロード

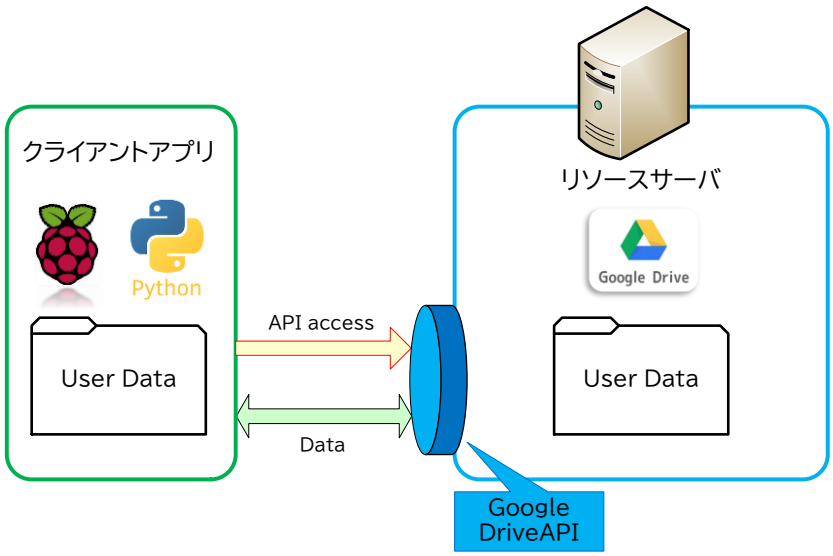
Python プログラムを使用して Raspberry Pi に保存されているさまざまなファイル（データ）を Google Drive にアップロードします。ここでは、「Google Drive API」を使用して Google Drive にファイルをアップロードする方法について学習します。

事前準備～アカウント取得～

Google アカウントでログインする必要があるため、事前に Google アカウントを取得しておいてください。

参考～Google Drive API の役割～

Python プログラム（クライアントアプリ）を使用して、Google Drive（リソースサーバ）にファイルをアップロードします。Google Drive API は、Python プログラムが Google Drive の機能を利用するために用意された「窓口」です。Python プログラムが Google Drive にファイルをアップロードするためには、「窓口」となる Google Drive API の設定が必要です。



手順①～Google アカウント設定～

Google Drive API を利用するため、Google アカウントの設定を行います。

次の URL をクリックし、Google アカウントでログインしてください。

[URL] https://console.developers.google.com/

Copyright © 2024 YITJC All Rights Reserved.

利用規約を確認し、「同意して続行」をクリックしてください。

「プロジェクトを作成」をクリックしてください。

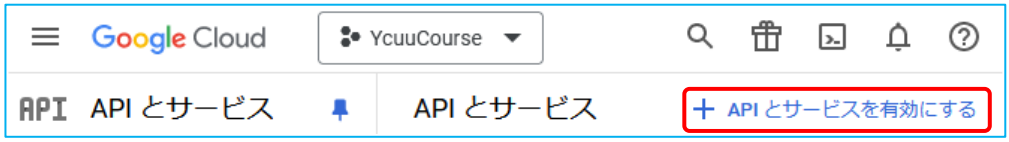
Copyright © 2024 YITJC All Rights Reserved.

プロジェクト名を入力し、「作成」をクリックしてください。

プロジェクトの作成が完了すると、「通知」画面が表示されます。

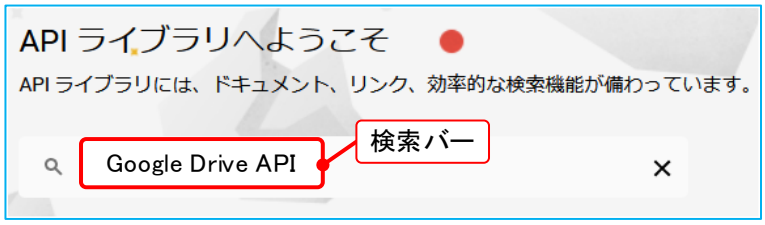
画面左上の「メニューから、「API とサービス」-「有効な API とサービス」をクリックしてください。

「＋ API とサービスを有効化」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

検索バーで「Google Drive API」と入力して検索してください。

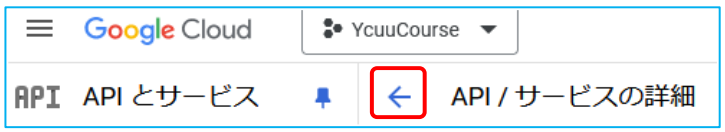


検索結果の中から「Google Drive API」を選択（クリック）してください。

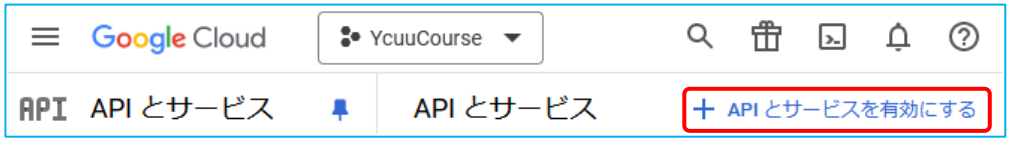
「有効にする」をクリックしてください。

続いて、9 章「通知機能・AI」で使用する「Cloud Vision API」を「有効」に設定します。

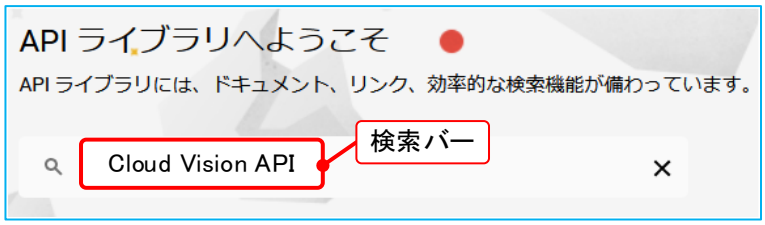
「API とサービス」画面に戻ってください。



「＋ API とサービスを有効化」をクリックしてください。

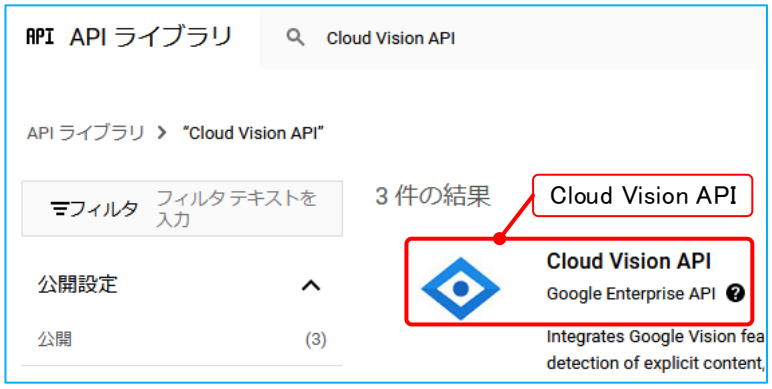


検索バーで「Cloud Vision API」と入力して検索（Enter）してください。



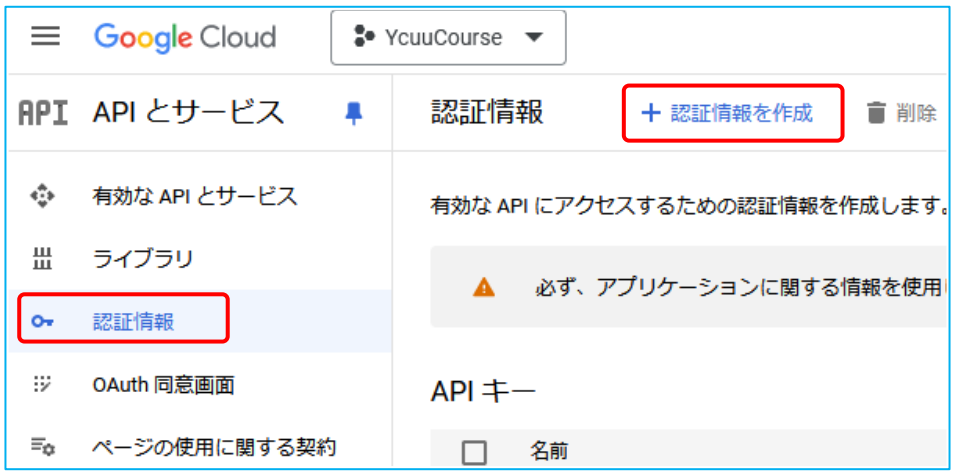
Copyright © 2024 YITJC All Rights Reserved.

検索結果から「Cloud Vison API」を選択（クリック）してください。

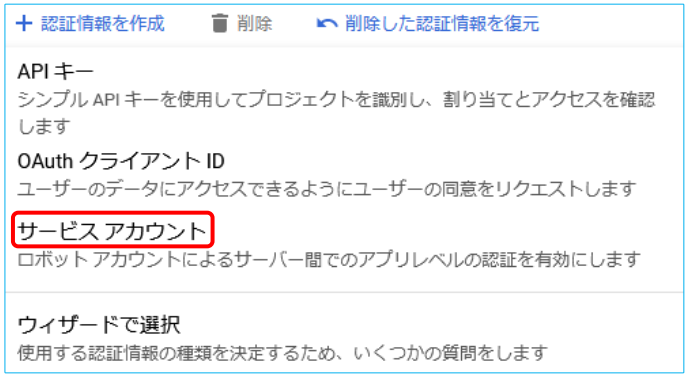


「有効にする」をクリックしてください。

画面左にある「認証情報」タグから「＋認証情報を作成」をクリックしてください。

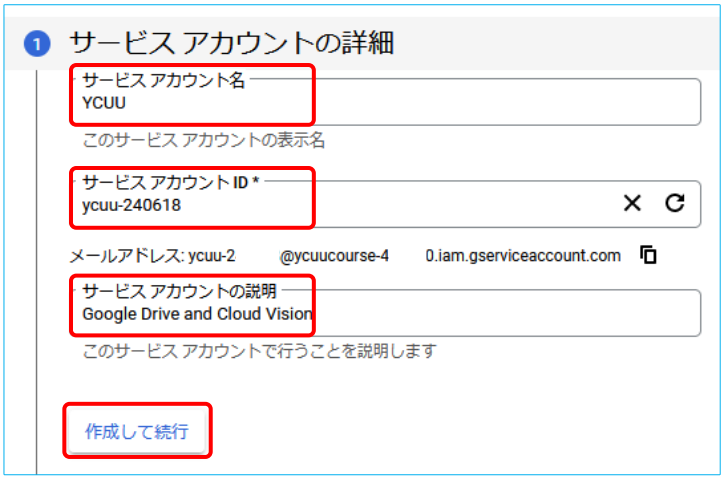


「サービスアカウント」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「サービスアカウント名」、「サービスアカウント ID（自動作成※変更可能）」、「サービスアカウントの説明」を入力したあと、「作成して続行」をクリックしてください。



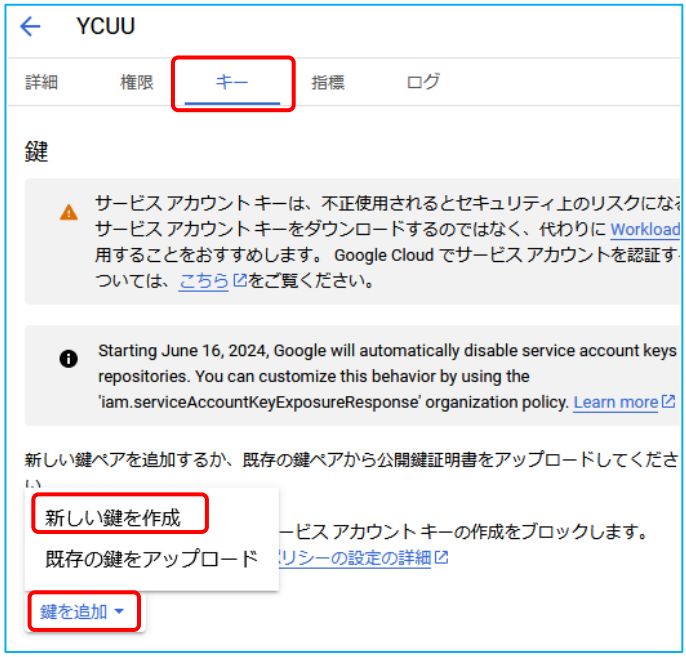
「ロール」から「オーナー」を選択し、「続行」をクリックしてください。

最後に「完了」をクリックして、サービスアカウントの作成を終了してください。

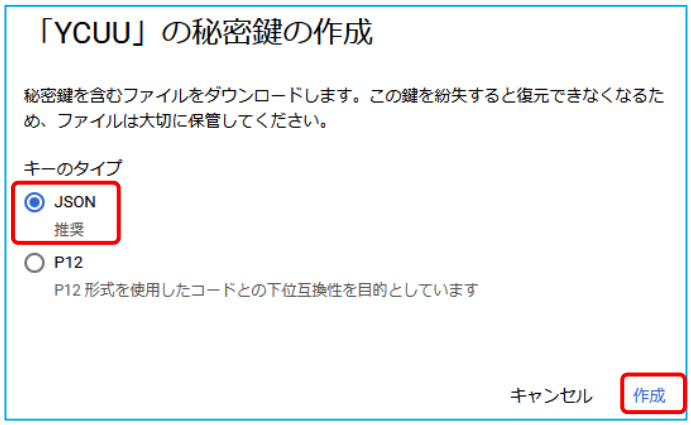
Copyright © 2024 YITJC All Rights Reserved.

続いて、「認証情報」タグを選択し、作成した「サービスアカウント」をクリックしてください。

「キー」タグをクリックして、「鍵を追加」から「新しい鍵を作成」を選択（クリック）してください。

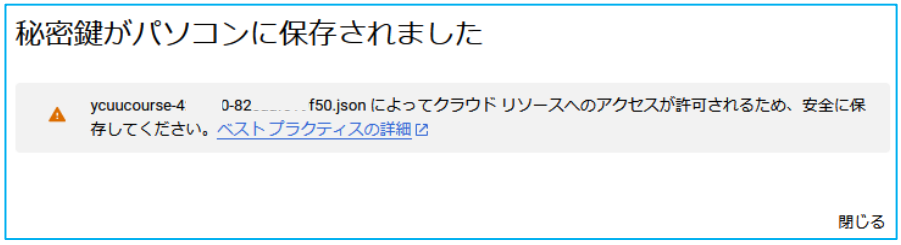


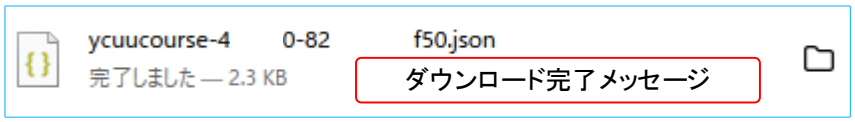
「キーのタイプ」から「JSON」を選択し、「作成」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

秘密鍵（key ファイル）が作成され、Windows PC のダウンロードフォルダに保存されます。

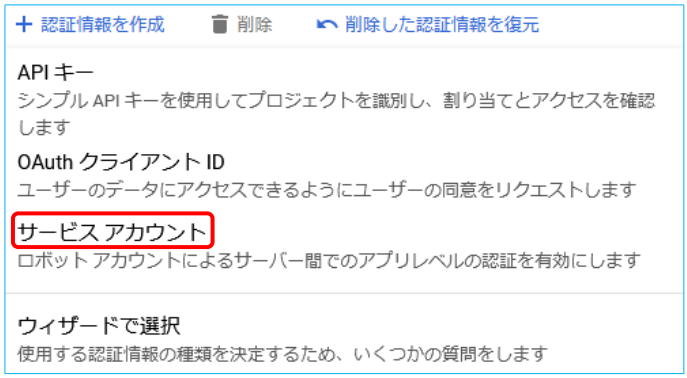




取得した秘密鍵（key ファイル）の名前を「service_account_key」に変更して保存ください。

参考～認証方法の選択～

Python プログラムを使用して Google Drive API を利用する場合、「OAuth 認証」を行う必要があります。ここでは、「サービスアカウントキー」を使用した OAuth 認証を行います。



「OAuth クライアント ID」を使用した認証方法を選択した場合、公式サイトに次の注意事項が記載されています。

[URL] https://developers.google.com/identity/protocols/oauth2?hl=ja

外部ユーザータイプ用に OAuth 同意画面が構成され、公開ステータスが「テスト」である Google Cloud Platform プロジェクトには、7 日後に期限切れになる更新トークンが発行されます。

現在、更新トークンの上限は、OAuth 2.0 クライアント ID ごとに Google アカウントあたり 100 個です。上限に達した場合、新しい更新トークンを作成すると、警告なしで最も古い更新トークンが自動的に無効になります。この上限はサービス アカウントには適用されません。

この情報によると、OAuth クライアント ID を使用した場合、発行されるリフレッシュトークンの有効期限が「7 日」であると明記されています。つまり、Google Drive にファイルをアップロードするシステムを構築してから 7 日後に OAuth 認証エラーが発生し、システムを利用できなくなります。

Copyright © 2024 YITJC All Rights Reserved.

手順②～サービスアカウント確認～

「認証情報」タグを選択し、作成した「サービスアカウント」をクリックしてください。

「詳細」タブを選択し、サービスアカウントの「メールアドレス」を確認してください。

このあとの手順で使用するで、**メモ帳などに記録してください。**

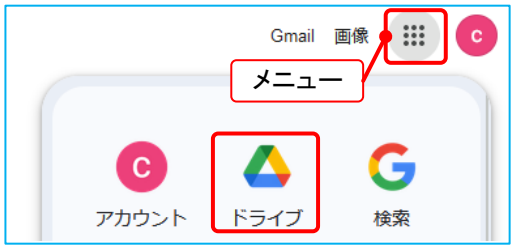
手順③～Google Drive フォルダ作成～

作成した「サービスアカウント」が編集権限をもつ「フォルダ」を Google Drive に作成します。

次の URL をクリックし、Google アカウントでログインしてください。

[URL] https://www.google.com/

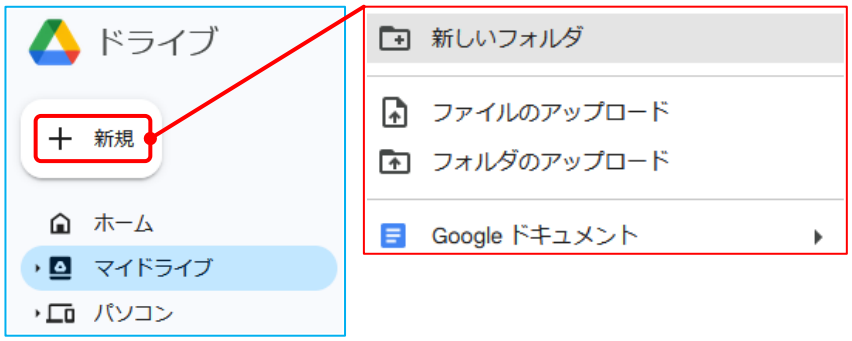
画面右上「メニュー」をクリックしてから、「ドライブ」を選択してください。



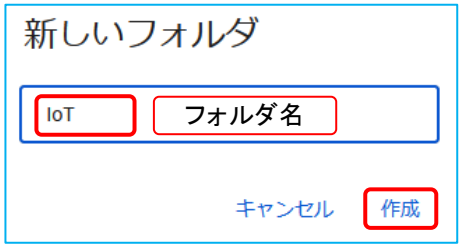
Copyright © 2024 YITJC All Rights Reserved.

画面左のメニューから、「マイドライブ」を選択してください。

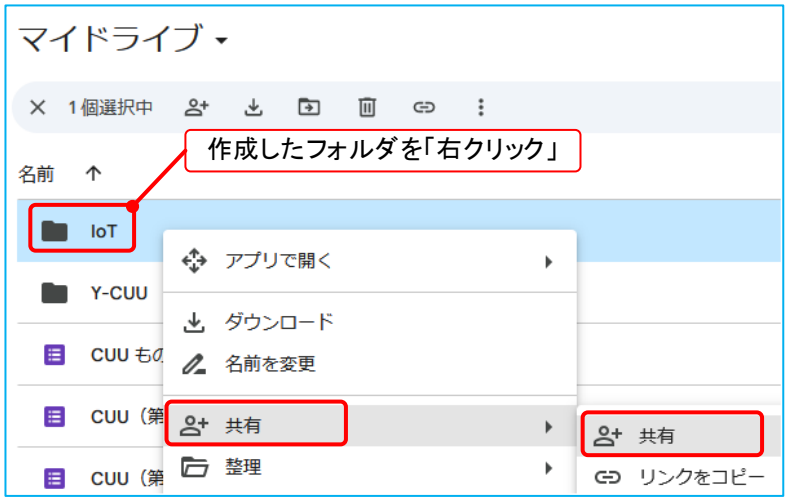
「＋新規」をクリックしてから、「新しいフォルダ」をクリックしてください。



新しいフォルダの名前「IoT」を入力したあと、「作成」をクリックしてください。

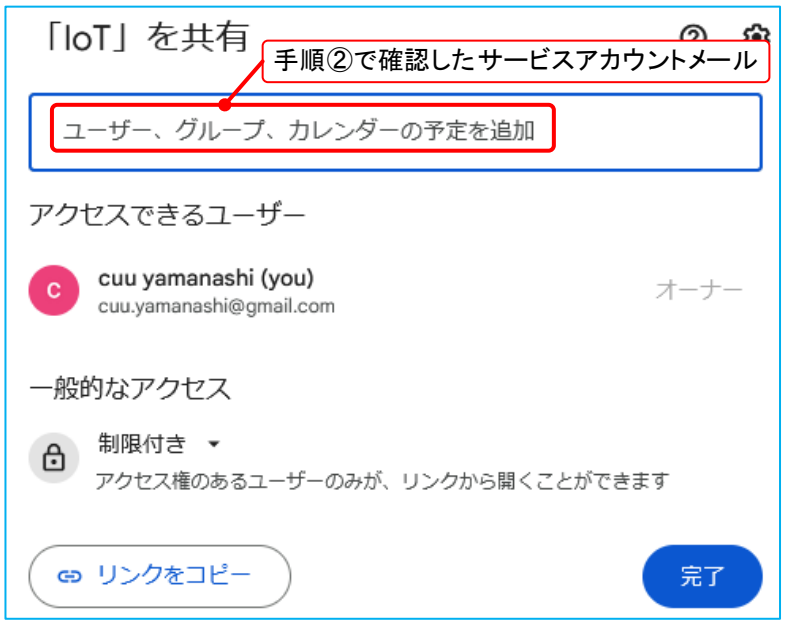


作成したフォルダを右クリックしてから、「共有」を選択してください。



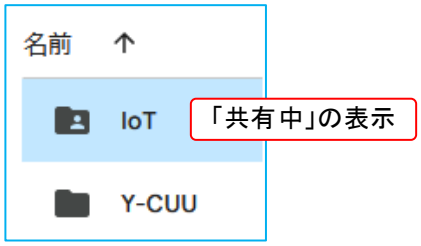
Copyright © 2024 YITJC All Rights Reserved.

手順②で確認したサービスアカウントの「メールアドレス」を入力してください。



サービスアカウントを「編集者」に設定してから、「送信」をクリックしてください。

作成したフォルダ「IoT」が共有されると、ファイルの横に「共有中」の印が表示されます。



以上の設定で、サービスアカウントに作成したフォルダの編集権限を与えることができました。

Copyright © 2024 YITJC All Rights Reserved.

手順④～フォルダ ID 確認～

Raspberry Pi に保存されているファイルを Google Drive にアップロードする場合、アップロード先の「フォルダ ID」を指定します。そのため、手順③で作成・共有した「フォルダ」の ID を確認します。

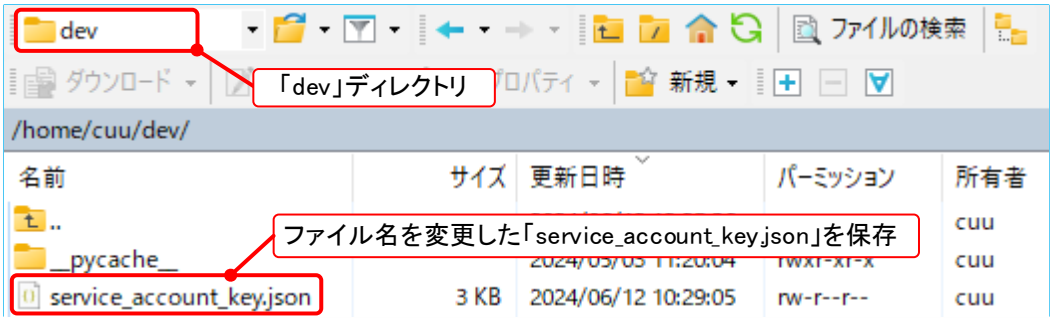
Google Drive で作成したフォルダを開いてください。

URL の一部が「フォルダ ID」になっているので、フォルダ ID をコピーして、メモ帳などに記録してください。

[URL] https://drive.google.com/drive/folders/フォルダ ID

手順⑤～サービスアカウントキーファイル保存～

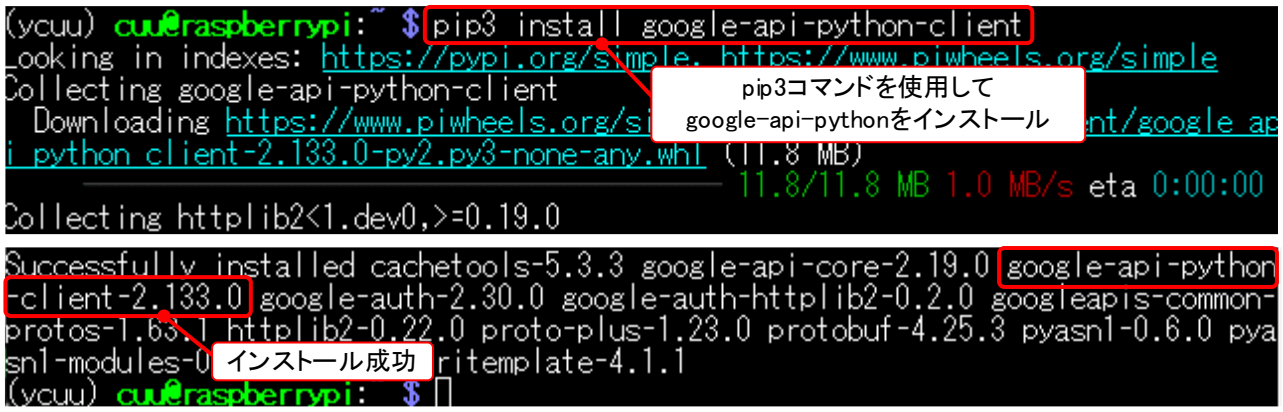
手順①で取得したサービスアカウントキーファイル（serivice_account_key.json）を「dev」ディレクトリに保存してください。このファイルは、OAuth 認証に必要な情報が記載されています。



手順⑥～google-api-client インストール～

Google Drive API を利用するために必要なライブラリ「google-api-python」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install google-api-python-client



Copyright © 2024 YITJC All Rights Reserved.

～補足～

google-api-python-client と同時に依存関係のあるパッケージもインストールされます。google-auth（OAuth2.0 認証用ライブラリ）や httplib2（HTTP クラインとライブラリ）など、API との通信やクラウドサービスの認証に必要なライブラリがインストールされます。

手順⑦～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

31_Pydrive.py

[プログラム編集個所]

```
#15行目　Google Drive　フォルダID指定
parent_folder_id = 'xxxxx'
```

手順⑧～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 31_Pydrive.py
```

手順⑨～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./31_Pydrive.py
```

プログラムを実行すると、次の画面が表示されます。

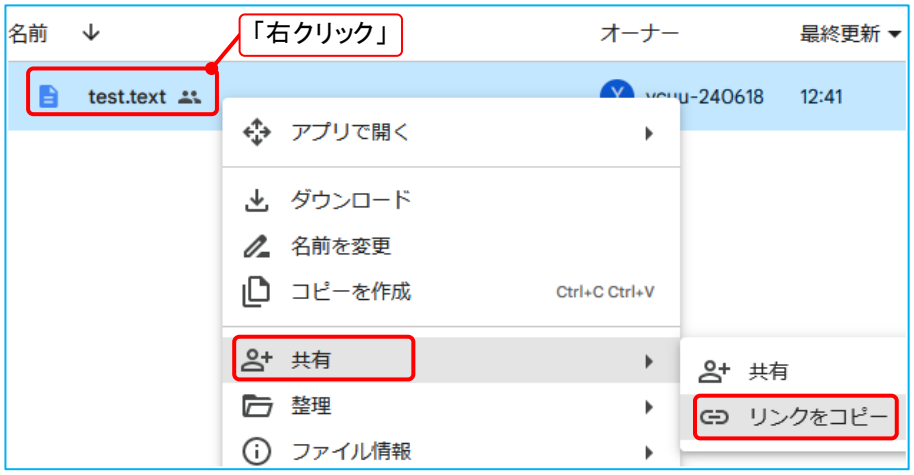
```
(ycuu) cuu@raspberrypi:~/dev $ ./31_Pydrive.py
File ID: 1 -ASyOJ4f8zlywVHY8ByvbGL3ta91GPf
```

「File ID」として記載されている情報は、アップロードしたファイルの「ファイル ID」です。

Google Drive の共有フォルダ（手順③で作成したフォルダ）を確認すると、Raspberry Pi で作成したファイル「test.txt」がアップロードされていることがわかります。

Copyright © 2024 YITJC All Rights Reserved.

「text.txt」ファイルを右クリックしてから、「リンクのコピー」を選択してください。



コピーしたリンクをメモ帳などに貼り付けて確認すると、URL の一部がファイル ID になっています。

https://drive.google.com/file/d/1_-ASyOJ4f8zlywVHY8ByvbGL3ta91GPf/view?usp=drive_link

➔ 「31_Pydrive.py」の実行結果で取得したファイル ID と同じ値になっていることを確認してください。

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	
8	#Google サービスアカウントキーの JSON ファイルへのパス
9	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
10	#Google Drive API のバージョン
11	API_VERSION = 'v3'
12	#Google Drive API のリソース名
13	DRIVE = build('drive', API_VERSION, credentials=service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE))
14	#Google Drive フォルダID指定（サービスアカウントが編集権限をもつフォルダ）
15	parent_folder_id = '1q1eJd202rN-uPuBU7cQLVmqGIjm6o5li'
16	
17	#新しいテキストファイルのパスとファイル名
18	file_path = '/home/cuu/data/test.text'
19	file_name = 'test.text'
20	
21	def upload_file(file_path, file_name, parent_folder_id=None):

Copyright © 2024 YITJC All Rights Reserved.

```
22      file_metadata = {
23          'name': file_name
24      }
25      if parent_folder_id:
26          file_metadata['parents'] = [parent_folder_id]
27      media = MediaFileUpload(file_path, resumable=True)
28      file = DRIVE.files().create(body=file_metadata, media_body=media, fields='id').execute()
29      print('File ID: %s' % file.get('id'))
30
31  def create_text():
32      #テキストファイルの内容を作成
33      with open(file_path, 'w') as f:
34          f.write('This is a test file.')
35
36  if __name__ == '__main__':
37      try:
38          create_text()
39          #ファイルアップロード
40          upload_file(file_path, file_name, parent_folder_id)
41      except KeyboardInterrupt:
42          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Google のサービスアカウント認証を行うため、「service_account」クラスをインポート
- 5 行目: Google Drive API のサービスオブジェクトを構築するため、「build」クラスをインポート
- 6 行目: ファイルを Google Drive にアップロードするために必要なライブラリ「googleapiclient.http」から「MediaFileUpload」クラスをインポート
- 9 行目: サービスアカウントキーのファイル名を定義
- 11 行目: Google Drive API のバージョンを定義
- 13 行目: Google Drive API のサービスオブジェクトを構築
  第一引数で使用する API（drive）、第二引数で API のバージョン（v3）、第三引数でサービスアカウントキーを指定しています。
- 15 行目: ファイルをアップロードするフォルダのフォルダ ID を定義
  サービスアカウントに編集権限を与えたフォルダ（手順④で確認したフォルダ）のフォルダ ID を指定します。
- 18 行目: Google Drive にアップロードする txt ファイルの作成場所を定義
- 19 行目: Google Drive にアップロードするファイル名を定義

Copyright © 2024 YITJC All Rights Reserved.

- 21 行目～29 行目: ファイルをアップロードするプログラム upload_file()関数
- 22 行目～24 行目: ファイルのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「ファイル名」を定義
- 25 行目～26 行目: 保存先ディレクトリ（親ディレクトリ: parent_folder_id）が指定されている場合は、保存先ディレクトリの「フォルダ ID」をメタデータに追加
- 27 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 28 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 29 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示
- 31 行目～34 行目: テキストファイルを作成するプログラム create_text()関数
- 33 行目～34 行目: 34 行目で指定した内容の「text.txt」ファイルを作成
- 36 行目～42 行目: main プログラム
- 37 行目～40 行目 :try ブロック
- 38 行目: テキストファイルを作成するプログラム create_text()関数実行
- 40 行目: ファイルをアップロードするプログラム upload_file()関数実行
- 41 行目～42 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

参考～Oauth2.0（オーオース 2.0）～

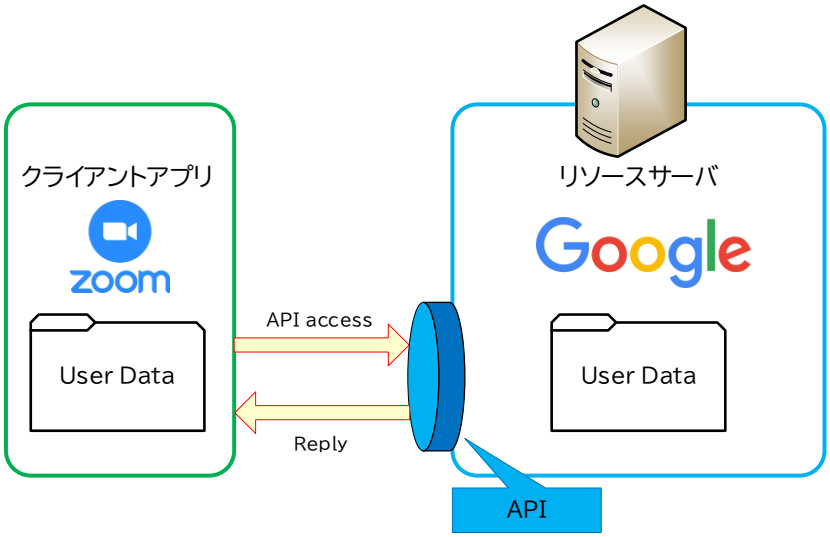
Google Drive の API を利用するためには、API の有効化や Oauth2.0 認証の設定が必要です。

ここでは、OAuth2.0 認証について説明します。OAuth（オーオース）は、SNS や Web サービス間で「アクセス権限の認可」を行うためのプロトコルです。現在では 2012 年に発行された OAuth 2.0 が標準化されています。

OAuth は、身近な存在です。次の図は Zoom のログイン画面ですが、Zoom アカウントがない場合でも、Google や Facebook など、別のアカウントでログインすることが可能です。これは、Google などのユーザー認証を信頼したうえで、Zoom がログイン情報を共有するという OAuth プロトコルの利用例になります。

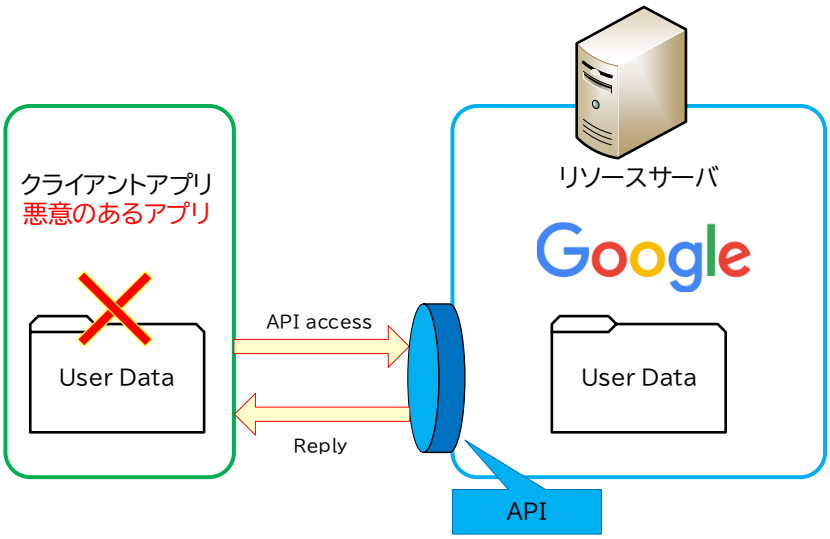
OAuth の仕組みを説明します。クライアントアプリとリソースサーバとの間でデータをやり取りする窓口を API と呼びます。クライアントアプリ側から API を介してデータを要求すると、リソースサーバ側から User Data が渡されて連携が行われます。

例）Zoom（クライアントアプリ）から Google（リソースサーバ）に連携する場合

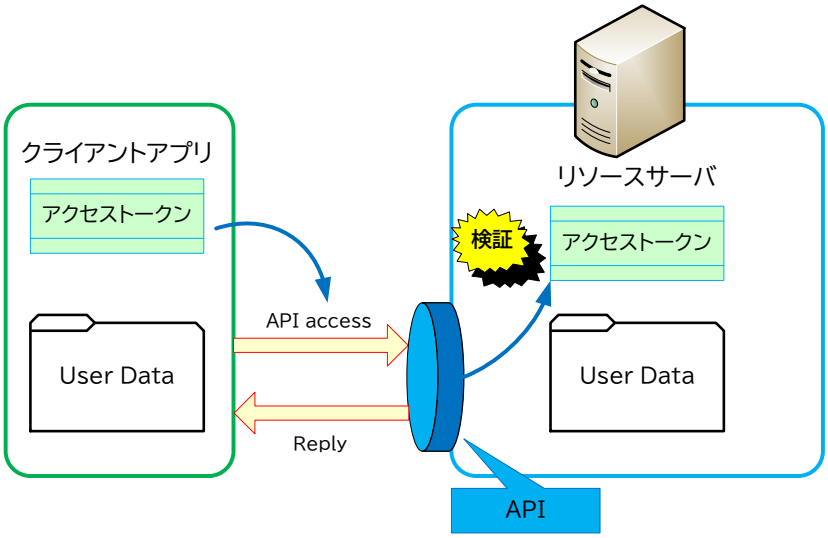


Copyright © 2024 YITJC All Rights Reserved.

ただし、このデータのやり取りには問題があります。仮にクライアントアプリが『悪意のあるアプリ』だった場合、悪意のあるアプリ側にも User Data が渡ってしまいます。User Data を守るためには何らかの仕組みが必要です。



User Data を守るための仕組みとして、クライアントアプリにあらかじめ『**アクセストークン**』を持たせる仕組みが考えられました。アクセストークンとは、認証済みユーザーを識別するための文字列であり、クライアントアプリが User Data を利用できることを証明するものです。



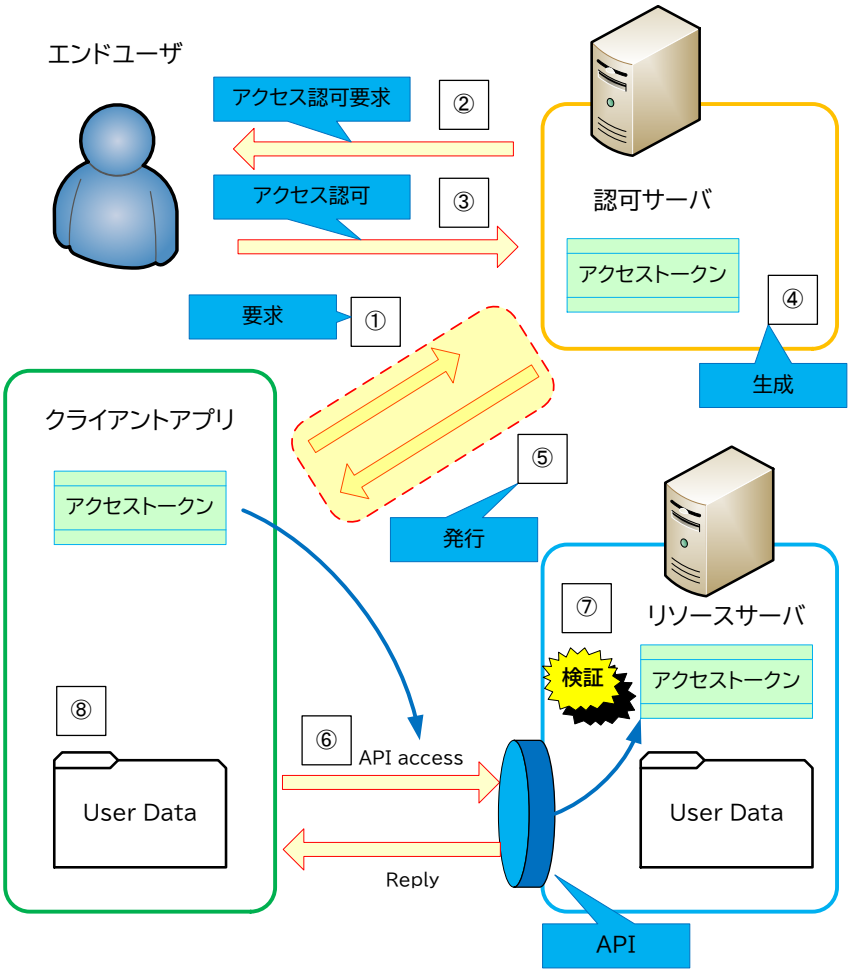
この仕組みにより、必要な権限をもつクライアントアプリだけが User Data を受け取ることができるようになります。ただし、この仕組みが機能するためには、あらかじめクライアントアプリに『アクセストークン』を渡しておく必要があります。

➔ あらかじめクライアントアプリに『アクセストークン』を渡す（発行する）係として『**認可サーバー**』を設置します。

Copyright © 2024 YITJC All Rights Reserved.

これまでの流れをまとめて説明します。

① アクセストークンの要求
② クライアントアプリへのアクセス認可要求
③ クライアントアプリにアクセス権限許可
④ アクセストークン生成
⑤ クライアントアプリにアクセストークン発行
⑥ User Data 要求
⑦ アクセストークン検証（User Data の利用権限確認）
⑧ User Data 受け取り



OAuth2.0 は、①（アクセストークンの要求）と⑤（アクセストークン発行）の部分を標準化したもので、技術文書 RFC 6749 に定義されています。

Copyright © 2024 YITJC All Rights Reserved.

8.2 フォルダ作成

Google Drive に「フォルダ」を作成する方法について説明します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

32_Pydrive.py

[プログラム編集個所]

```
#15行目  Google Drive  フォルダID指定
parent_folder_id = 'xxxxx'
```

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 32_Pydrive.py
```

手順③～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./32_Pydrive.py
```

プログラムを実行すると、作成したフォルダの「フォルダ ID」が表示されます。

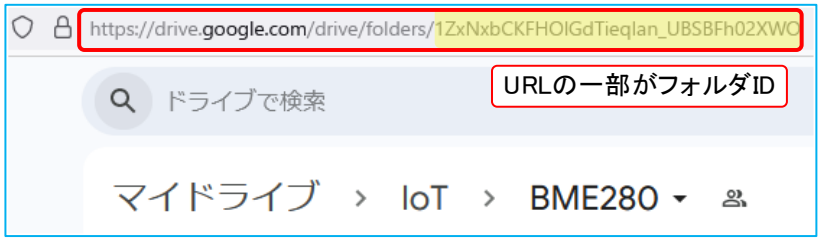
表示された「フォルダ ID」は、このあとのプログラムで使用するので、メモ帳などに記録してください。

Google Drive を確認すると、「BME280」フォルダが作成されていることを確認できます。

～参考～

作成したフォルダの URL を確認し、「フォルダ ID」が実行結果と同じであることを確認してください。

https://drive.google.com/drive/folders/フォルダ ID



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	
8	#Google サービスアカウントキーの JSON ファイルへのパス
9	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
10	#Google Drive API のバージョン
11	API_VERSION = 'v3'
12	#Google Drive API のリソース名
13	DRIVE = build('drive', API_VERSION,
	credentials=service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE))
14	#Google Drive フォルダID指定（サービスアカウントが編集権限をもつフォルダ）
15	parent_folder_id = '1q1eJd202rN-uPuBU7cQLVmqGIjm6o5li'
16	#作成するフォルダの名前
17	new_folder_name = 'BME280'
18	
19	def create_folder(parent_folder_id, folder_name):
20	folder_metadata = {
21	'name': folder_name,
22	'mimeType': 'application/vnd.google-apps.folder',
23	'parents': [parent_folder_id]
24	}
25	folder = DRIVE.files().create(body=folder_metadata, fields='id').execute()
26	print('Folder ID: %s' % folder.get('id'))
27	return folder.get('id')
28	
29	if __name__ == '__main__':
30	try:
31	#新しいフォルダを作成
32	new_folder_id = create_folder(parent_folder_id, new_folder_name)
33	except KeyboardInterrupt:
34	pass

Copyright © 2024 YITJC All Rights Reserved.

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Google のサービスアカウント認証を行うため、「service_account」クラスをインポート
- 5 行目: Google Drive API のサービスオブジェクトを構築するため、「build」クラスをインポート
- 6 行目: ファイルを Google Drive にアップロードするために必要なライブラリ「googleapiclient.http」から「MediaFileUpload」クラスをインポート
- 9 行目: サービスアカウントキーのファイル名を定義
- 11 行目: Google Drive API のバージョンを定義
- 13 行目: Google Drive API のサービスオブジェクトを構築
  第一引数で使用する API（drive）、第二引数で API のバージョン（v3）、第三引数でサービスアカウントキーを指定しています。
- 15 行目: ファイルをアップロードするフォルダのフォルダ ID を定義
  サービスアカウントに編集権限を与えたフォルダ（手順④で確認したフォルダ）のフォルダ ID を指定します。
- 17 行目: 作成するフォルダの「名前」を定義
- 19 行目～27 行目: フォルダを作成するプログラム create_folder()関数
- 20 行目～24 行目: フォルダのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「フォルダ名」、「MIME タイプ」、「親ディレクトリ」を定義
- 25 行目: DRIVE.files().create メソッドを使用してフォルダを作成し、execute()メソッドでアップロードを実行
- 26 行目: 作成したフォルダの「フォルダ ID」を取得して print 表示
- 29 行目～34 行目: main プログラム
- 30 行目～32 行目 :try ブロック
- 32 行目: フォルダを作成するプログラム create_folder()関数実行
- 33 行目～34 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

参考～MIME タイプ～

Web の世界では拡張子という概念と、もうひとつ「MIME（Multipurpose Internet Mail Extensions）タイプ」という概念があります。そのため、Google Drive にファイルをアップロードする場合は、適切な MIME タイプを設定する必要があります。

MIME タイプは、「タイプ名/サブタイプ名」の形式で表現される文字列です。タイプ名でデータの種類（テキスト、画像、動画など）を、サブタイプ名で具体的なデータ形式を指定します。ここでは、Google Drive に作成するフォルダなので、「application/vnd.google-apps.folder」と指定しています。

Copyright © 2024 YITJC All Rights Reserved.

Raspberry Pi で扱う可能性が高いファイルと MIME タイプの一覧を示します。

ファイル	MIME タイプ
Google Drive フォルダ	application/vnd.google-apps.folder
テキストファイル	text/plain
JPEG ファイル	image/jpeg
PNG ファイル	image/png
csv ファイル	text/csv
csv ファイル（Excel フォーマット）	application/vnd.ms-excel
MPEG 動画ファイル	video/mpeg

Copyright © 2024 YITJC All Rights Reserved.

8.3 指定フォルダへのアップロード

Raspberry Pi に保存されているファイルを Google Drive の「指定フォルダ」にアップロードする方法について説明します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

33_Pydrive.py

[プログラム編集個所]

```
#15行目　Google Drive　フォルダID指定
parent_folder_id = 'xxxxx'
```

～注意～

フォルダ ID は、「32_Pydrive.py」の実行結果で取得した情報を使用してください。

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

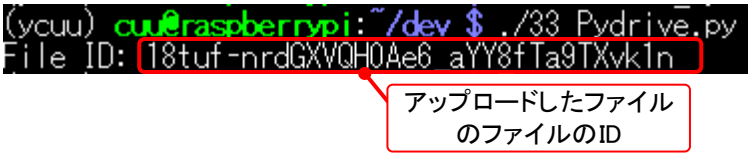
```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 33_Pydrive.py
```

手順③～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./33_Pydrive.py
```

プログラムを実行すると、アップロードしたファイルの「ファイル ID」が表示されます。



Google Drive を確認すると、Raspberry Pi の「/home/cuu/data」ディレクトリに保存されていた「Raspi_temp.csv」ファイルがアップロードされていることを確認できます。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	
8	#Google サービスアカウントキーの JSON ファイルへのパス
9	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
10	#Google Drive API のバージョン
11	API_VERSION = 'v3'
12	#Google Drive API のリソース名
13	DRIVE = build('drive', API_VERSION,
	credentials=service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE))
14	#Google Drive フォルダID指定（サービスアカウントが編集権限をもつフォルダ）
15	parent_folder_id = '1ZxNxbCKFHOlGdTieqIan_UBSBFh02XWO'
16	
17	#アップロードするファイルのパスとファイル名
18	file_path = '/home/cuu/data/Raspi_temp.csv'
19	file_name = 'Raspi_temp.csv'
20	
21	def upload_file(file_path, file_name, parent_folder_id=None):
22	file_metadata = {
23	'name': file_name
24	}
25	if parent_folder_id:
26	file_metadata['parents'] = [parent_folder_id]
27	media = MediaFileUpload(file_path, resumable=True)
28	file = DRIVE.files().create(body=file_metadata, media_body=media, fields='id').execute()
29	print('File ID: %s' % file.get('id'))
30	
31	if __name__ == '__main__':
32	try:
33	upload_file(file_path, file_name, parent_folder_id)
34	except KeyboardInterrupt:
35	pass

Copyright © 2024 YITJC All Rights Reserved.

灰色部は、これまでに説明した内容です。

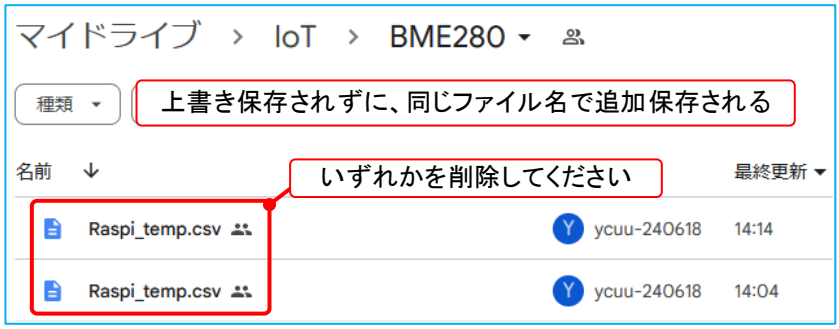
- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Google のサービスアカウント認証を行うため、「service_account」クラスをインポート
- 5 行目: Google Drive API のサービスオブジェクトを構築するため、「build」クラスをインポート
- 6 行目: ファイルを Google Drive にアップロードするために必要なライブラリ「googleapiclient.http」から「MediaFileUpload」クラスをインポート
- 9 行目: サービスアカウントキーのファイル名を定義
- 11 行目: Google Drive API のバージョンを定義
- 13 行目: Google Drive API のサービスオブジェクトを構築
  第一引数で使用する API（drive）、第二引数で API のバージョン（v3）、第三引数でサービスアカウントキーを指定しています。
- 15 行目: ファイルをアップロードするフォルダのフォルダ ID を定義
  「8.2 フォルダ作成」で作成したフォルダ（32_Pydrive.py の実行結果）のフォルダ ID を指定します。
- 18 行目: Google Drive にアップロードするファイルを定義
- 19 行目: Google Drive にアップロードするファイル名を定義
- 21 行目～29 行目: ファイルをアップロードするプログラム upload_file()関数
- 22 行目～24 行目: ファイルのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「ファイル名」を定義
- 25 行目～26 行目: 保存先ディレクトリ（親ディレクトリ: parent_folder_id）が指定されている場合は、保存先ディレクトリの「フォルダ ID」をメタデータに追加
- 27 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 28 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 29 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示
- 31 行目～35 行目: main プログラム
- 32 行目～33 行目 :try ブロック
- 33 行目: ファイルをアップロードするプログラム upload_file()関数実行
- 34 行目～35 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

～参考～

「33_Pydrive.py」を再び実行すると、Google Drive に同じファイルが 2 つ保存されてしまいます。

同じ名前のファイルをアップロードした場合、上書き保存されずに別ファイルとして保存されます。これは、同じファイル名でも、ファイルのメタデータ（ファイルの内容や作成日時、更新日時など）が異なる場合は、Google Drive が異なるファイルと認識して保存します。



注意

この作業をされた方は、いずれかのファイルを削除してファイルが 1 つだけの状態にしてください。

次項から、ファイルを上書き保存する方法について説明します。

Copyright © 2024 YITJC All Rights Reserved.

8.4 ファイル上書き保存

Google Drive に保存されているファイルを「上書き保存」する方法について説明します。

上書き保存する方法は 2 つあります。

① **ファイル ID を指定する方法**

プログラムでフォルダ ID とファイル ID の両方を指定する必要があります。

アップロードするファイルが多い場合、ファイル ID を確認する作業やプログラムに ID を設定する時間が掛かってしまいます。

② **同じファイル名の存在を確認してから上書き保存（同じファイル名がなければ新規保存）**

プログラムで同じ名前のファイルをチェックする機能が必要です。

指定フォルダに「同じ名前のファイル」がある場合は、上書き保存を実行し、

指定フォルダに「同じ名前のファイル」がない場合は、新規保存する方法です。

ここでは、②の方法でアップロードする方法について説明します。

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

34_Pydrive.py

[プログラム編集個所]

```
#15行目  Google Drive  フォルダID指定
parent_folder_id = 'xxxxx'
```

～注意～

フォルダ ID は、「32_Pydrive.py」の実行結果で取得した情報を使用してください。

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

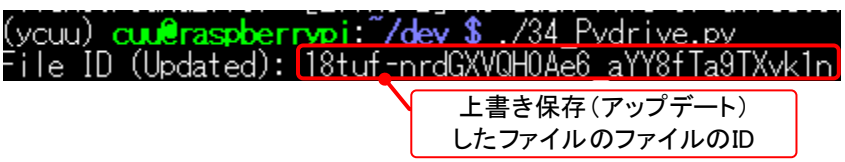
```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 34_Pydrive.py
```

手順③～プログラム実行～

対象プログラムを実行します。

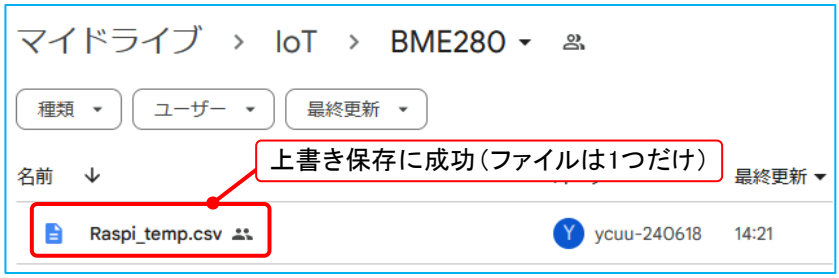
```
(ycuu) cuu@raspberrypi:~/dev $ ./34_Pydrive.py
```

プログラムを実行すると、次の画面が表示されます。



Copyright © 2024 YITJC All Rights Reserved.

Google Drive を確認すると、ファイルの上書き保存（アップデート）が成功し、ファイルは 1 つだけ保存されている状態です。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	
8	#Google サービスアカウントキーの JSON ファイルへのパス
9	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
10	#Google Drive API のバージョン
11	API_VERSION = 'v3'
12	#Google Drive API のリソース名
13	DRIVE = build('drive', API_VERSION,
	credentials=service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE))
14	#Google Drive フォルダID指定（サービスアカウントが編集権限をもつフォルダ）
15	parent_folder_id = '1ZxNxbCKFHOlGdTieqIan_UBSBFh02XWO'
16	
17	#アップロードするファイルのパスとファイル名
18	file_path = '/home/cuu/data/Raspi_temp.csv'
19	file_name = 'Raspi_temp.csv'
20	
21	def upload_file(file_path, file_name, parent_folder_id=None):
22	#既存ファイルのチェック
23	query = f"name='{file_name}'"
24	if parent_folder_id:
25	query += f" and '{parent_folder_id}' in parents"
26	
27	results = DRIVE.files().list(q=query, fields="files(id, name)").execute()

Copyright © 2024 YITJC All Rights Reserved.

```
28          files = results.get('files', [])
29
30          if files:
31              #同じ名前のファイルが存在する場合、そのファイルIDを取得
32              file_id = files[0]['id']
33              #ファイルを更新（上書き）
34              media = MediaFileUpload(file_path, resumable=True)
35              updated_file = DRIVE.files().update(fileId=file_id, media_body=media, fields='id').execute()
36              print('File ID (Updated): %s' % updated_file.get('id'))
37          else:
38              #ファイルが存在しない場合、新規にアップロード
39              file_metadata = {
40                  'name': file_name
41              }
42              if parent_folder_id:
43                  file_metadata['parents'] = [parent_folder_id]
44              media = MediaFileUpload(file_path, resumable=True)
45              file = DRIVE.files().create(body=file_metadata, media_body=media, fields='id').execute()
46              print('File ID (Created): %s' % file.get('id'))
47
48  if __name__ == '__main__':
49      try:
50          upload_file(file_path, file_name, parent_folder_id)
51      except KeyboardInterrupt:
52          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Google のサービスアカウント認証を行うため、「service_account」クラスをインポート
- 5 行目: Google Drive API のサービスオブジェクトを構築するため、「build」クラスをインポート
- 6 行目: ファイルを Google Drive にアップロードするために必要なライブラリ「googleapiclient.http」から「MediaFileUpload」クラスをインポート
- 9 行目: サービスアカウントキーのファイル名を定義
- 11 行目: Google Drive API のバージョンを定義
- 13 行目: Google Drive API のサービスオブジェクトを構築
  第一引数で使用する API（drive）、第二引数で API のバージョン（v3）、第三引数でサービスアカウントキーを指定しています。

Copyright © 2024 YITJC All Rights Reserved.

- 15 行目: ファイルをアップロードするフォルダのフォルダ ID を定義
  「8.2 フォルダ作成」で作成したフォルダ（32_Pydrive.py の実行結果）のフォルダ ID を指定します。
- 18 行目: Google Drive にアップロードするファイルを定義
- 19 行目: Google Drive にアップロードするファイル名を定義
- 21 行目～46 行目: 同名ファイルチェック処理付きアップデートプログラム upload_file()関数
- 23 行目: 変数「query」に検索クエリを定義
- 24 行目～25 行目: 検索フォルダ内で指定ファイル（file_name）と同じファイル名の存在をチェック
  検索フォルダは、15 行目に指定したフォルダです。
- 27 行目: Drive.Files().list メソッド検索クエリに一致するファイルを検索し、結果を変数「result」に格納
- 28 行目: 検索結果（result）が取得したファイル情報を変数「files」に格納
  フォルダ ID（folder_id）、ファイル名（RasPi_temp.csv）を指定して同じファイルの存在チェック
- Google Drive に同じ名前のファイルが存在していた場合は、files に追加

～上書き保存（アップデート）～

- 30 行目: files の空チェック
  files が空でなければ（=同じ名前のファイルが存在していれば）、上書き保存
- 32 行目: 検索結果に同じファイル名がある場合、そのファイル ID を変数「file_id」に格納
- 34 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 35 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 36 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示

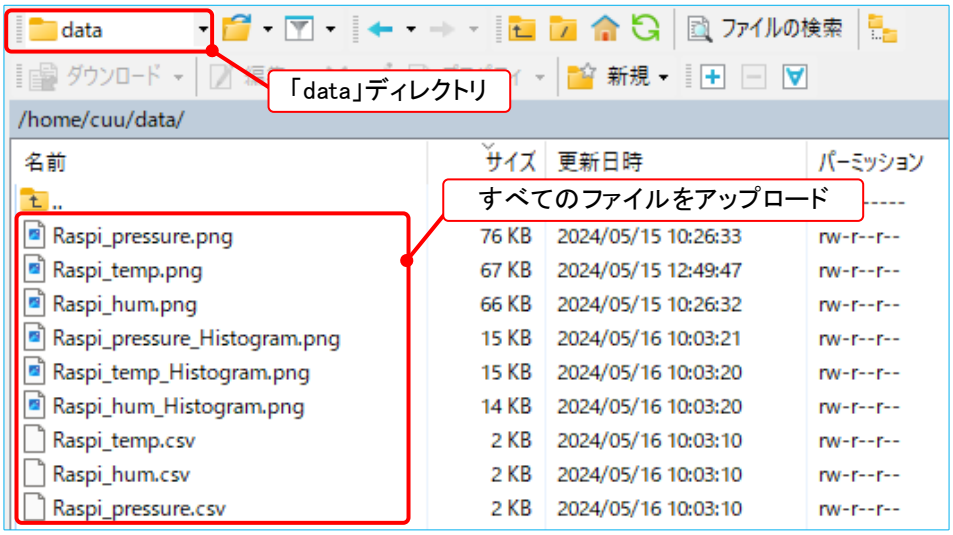
～新規保存（アップロード）～

- 39 行目～41 行目: ファイルのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「ファイル名」を定義
- 42 行目～43 行目: 保存先ディレクトリ（親ディレクトリ: parent_folder_id）が指定されている場合は、保存先ディレクトリの「フォルダ ID」をメタデータに追加
- 44 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 45 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 46 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示
- 48 行目～52 行目: main プログラム
- 49 行目～50 行目 :try ブロック
- 50 行目: ファイルをアップロードするプログラム upload_file()関数実行
- 51 行目～52 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

8.5 フォルダ内のデーター括アップロード

Raspberry Pi の「/home/cuu/data」フォルダに保存されているすべてのデータを Google Drive の「指定フォルダ」に一括アップロードする方法について説明します。



手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

35_Pydrive.py

[プログラム編集個所]

#16行目 Google Drive フォルダID指定

parent_folder_id = 'xxxxx'

～注意～

フォルダ ID は、「32_Pydrive.py」の実行結果で取得した情報を使用してください。

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 35_Pydrive.py

手順③～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./35_Pydrive.py

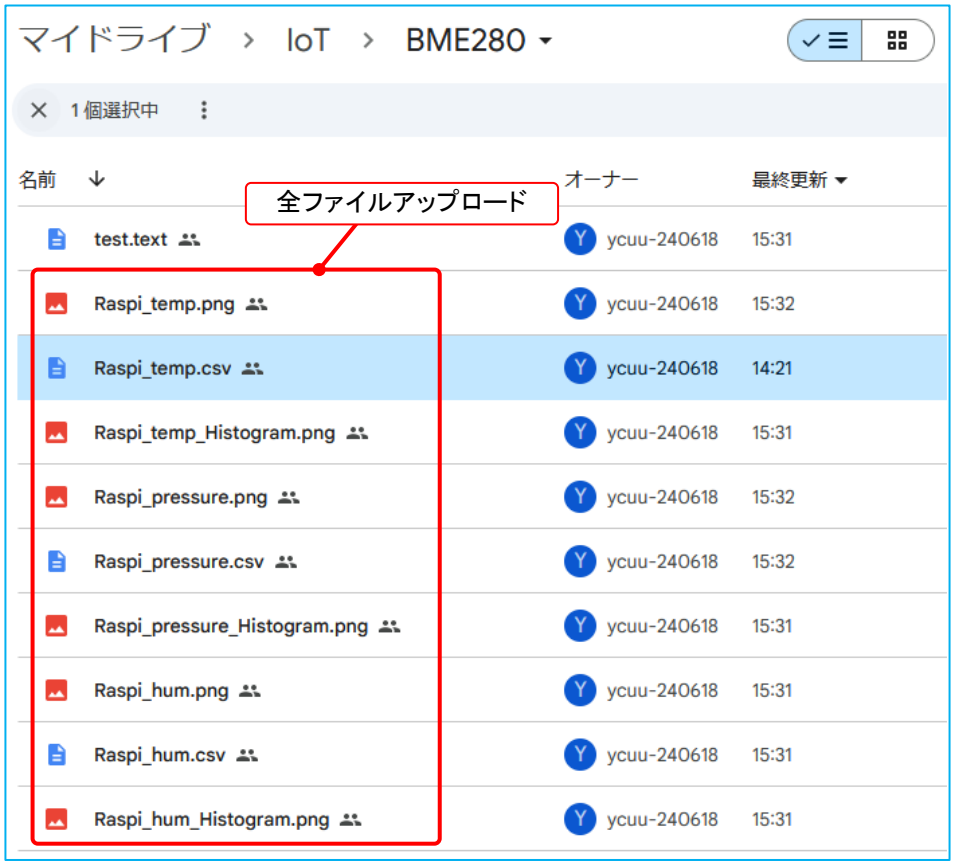
Copyright © 2024 YITJC All Rights Reserved.

プログラムを実行すると、次の画面が表示されます。

Raspberry Pi の「/home/cuu/data」フォルダに保存されていたすべてのファイルが Google Drive にアップロードされたことを確認できます。

～補足～

「Raspi_temp.csv」ファイルはアップデート（上書き保存）されています。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	import os
8	
9	#Google サービスアカウントキーの JSON ファイルへのパス
10	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
11	#Google Drive API のバージョン
12	API_VERSION = 'v3'
13	#Google Drive API のリソース名
14	DRIVE = build('drive', API_VERSION,
	credentials=service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE))
15	#Google Drive フォルダID指定（サービスアカウントが編集権限をもつフォルダ）
16	parent_folder_id = '1ZxNxbCKFHOlGdTieqIan_UBSBFh02XWO'
17	
18	#一括アップロードするディレクトリのパス
19	directory_path = '/home/cuu/data/'
20	
21	def upload_file(file_path, file_name, parent_folder_id=None):
22	# 既存ファイルのチェック
23	query = f"name='{file_name}'"
24	if parent_folder_id:
25	query += f" and '{parent_folder_id}' in parents"
26	
27	results = DRIVE.files().list(q=query, fields="files(id, name)").execute()
28	files = results.get('files', [])
29	
30	if files:
31	# 同じ名前のファイルが存在する場合、そのファイルIDを取得
32	file_id = files[0]['id']
33	# ファイルを更新（上書き）
34	media = MediaFileUpload(file_path, resumable=True)
35	updated_file = DRIVE.files().update(fileId=file_id, media_body=media, fields='id').execute()
36	print('File ID (Updated): %s' % updated_file.get('id'))

Copyright © 2024 YITJC All Rights Reserved.

```
37          else:
38              # ファイルが存在しない場合、新規にアップロード
39              file_metadata = {
40                  'name': file_name
41              }
42              if parent_folder_id:
43                  file_metadata['parents'] = [parent_folder_id]
44              media = MediaFileUpload(file_path, resumable=True)
45              file = DRIVE.files().create(body=file_metadata, media_body=media, fields='id').execute()
46              print('File ID (Created): %s' % file.get('id'))
47
48  if __name__ == '__main__':
49          try:
50              #ディレクトリ内の全ファイルをアップロード
51              for file_name in os.listdir(directory_path):
52                  file_path = os.path.join(directory_path, file_name)
53                  if os.path.isfile(file_path):
54                      upload_file(file_path, file_name, parent_folder_id)
55          except KeyboardInterrupt:
56              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Google のサービスアカウント認証を行うため、「service_account」クラスをインポート
- 5 行目: Google Drive API のサービスオブジェクトを構築するため、「build」クラスをインポート
- 6 行目: ファイルを Google Drive にアップロードするために必要なライブラリ「googleapiclient.http」から「MediaFileUpload」クラスをインポート
- 7 行目: OS と対話するためのライブラリ「os」をインポート
- 10 行目: サービスアカウントキーのファイル名を定義
- 12 行目: Google Drive API のバージョンを定義
- 14 行目: Google Drive API のサービスオブジェクトを構築
  第一引数で使用する API（drive）、第二引数で API のバージョン（v3）、第三引数でサービスアカウントキーを指定しています。
- 16 行目: ファイルをアップロードするフォルダのフォルダ ID を定義
  「8.2 フォルダ作成」で作成したフォルダ（32_Pydrive.py の実行結果）のフォルダ ID を指定します。
- 19 行目: Google Drive に一括アップロードするディレクトリを指定

Copyright © 2024 YITJC All Rights Reserved.

- 21 行目～46 行目: 同名ファイルチェック処理付きアップデートプログラム upload_file()関数
- 23 行目: 変数「query」に検索クエリを定義
- 24 行目～25 行目: 検索フォルダ内で指定ファイル（file_name）と同じファイル名の存在をチェック
  検索フォルダは、15 行目に指定したフォルダです。
- 27 行目: Drive.Files().list メソッド検索クエリに一致するファイルを検索し、結果を変数「result」に格納
- 28 行目: 検索結果（result）が取得したファイル情報を変数「files」に格納
  フォルダ ID（folder_id）、ファイル名（RasPi_temp.csv）を指定して同じファイルの存在チェック
- Google Drive に同じ名前のファイルが存在していた場合は、files に追加

～上書き保存（アップデート）～

- 30 行目: files の空チェック
  files が空でなければ（=同じ名前のファイルが存在していれば）、上書き保存
- 32 行目: 検索結果に同じファイル名がある場合、そのファイル ID を変数「file_id」に格納
- 34 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 35 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 36 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示

～新規保存（アップロード）～

- 39 行目～41 行目: ファイルのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「ファイル名」を定義
- 42 行目～43 行目: 保存先ディレクトリ（親ディレクトリ: parent_folder_id）が指定されている場合は、保存先ディレクトリの「フォルダ ID」をメタデータに追加
- 44 行目: MediaFileUpload クラスを使用してアップロードするファイルのオブジェクトを作成
  「resumable=True」は、アップロードが中断された場合に、アップロードの再開を「有効」に設定しています。
- 45 行目: DRIVE.files().create メソッドを使用してファイルを作成し、execute()メソッドでアップロードを実行
- 46 行目: アップロードしたファイルの「ファイル ID」を取得して print 表示
- 48 行目～56 行目: main プログラム
- 49 行目～54 行目 :try ブロック
- 51 行目～54 行目: 指定ディレクトリに保存されているすべてのファイルについて繰り返し処理
- 52 行目: 「directory_path」と「file_name」から完全なファイルパスを作成
- 53 行目: 指定されたファイルパスが「ファイル」であるかチェック
- 54 行目: ファイルである場合、指定フォルダ（parent_folder_id）にアップロード
- 55 行目～56 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

参考～Google Drive API クラス～

Raspberry Pi と Google Drive との間で「ファイル」や「フォルダ」のアップロードやダウンロードなどを実行する処理（関数）をクラス「GDrive_Class」にまとめました。次のプログラムを共有サイトに公開しておきます。

- GDrive_Class.py
  「GDrive_Class」クラスを定義して「Google Drive API」を使用した各種機能をモジュール化しています。
- 36_Pydrive.py
  「GDrive_Class.py」で定義したクラス「GDrive_Class」をインポート（import）して次の機能を実行します。

関数	機能
google_drive_id_list(self)	全フォルダ・ファイルの ID 取得
get_file_id_in_folder_id(self, folder_id)	指定フォルダ(ID 指定)に含まれる全ファイルの ID 取得
get_folder_id(self, folder_name)	指定フォルダの ID 取得
get_file_id_in_folder_name(self, folder_name)	指定フォルダ(名前指定)に含まれる全ファイルの ID 取得
get_file_info_in_folder_name(self, folder_name)	指定フォルダに含まれる全ファイルの情報(ファイル名、作成日時、ID)取得
upload_file_to_drive(self, local_file_path, folder_name)	ファイルアップデート（上書き）同じ名前のファイルがない場合は、新規アップロード
create_folder_in_drive(self, folder_name, folder_id)	フォルダ作成
upload_folder_to_drive(self, local_folder_path, folder_name)	ローカルフォルダのデータを一括アップデート（上書き）
download_from_file_name(self, folder_name, file_name, target_dir="")	フォルダ名とファイル名を指定してファイルをダウンロード
get_file_id(self, folder_name=None, file_name=None)	指定ファイルの ID 取得

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説（GDrive_Class.py）～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from google.oauth2 import service_account
5	from googleapiclient.discovery import build
6	from googleapiclient.http import MediaFileUpload
7	from googleapiclient.http import MediaIoBaseDownload
8	import os
9	import pandas as pd
10	
11	class GDrive_Class:
12	# Google サービスアカウントキーの JSON ファイルへのパス
13	SERVICE_ACCOUNT_FILE = 'service_account_key.json'
14	# Google Drive API のバージョン
15	API_VERSION = 'v3'
16	
17	def __init__(self):
18	self.credentials = service_account.Credentials.from_service_account_file(self.SERVICE_ACCOUNT_FILE)
19	self.drive = build('drive', self.API_VERSION, credentials=self.credentials)
20	
21	#全フォルダ・全ファイルのID情報取得
22	def google_drive_id_list(self):
23	file_list = self.drive.files().list(q="trashed=false").execute().get('files', [])
24	for file1 in file_list:
25	print('title: %s, id: %s' % (file1['name'], file1['id']))
26	
27	#フォルダ名を指定してフォルダIDを取得
28	def get_folder_id(self, folder_name):
29	query = f"trashed=false and mimeType='application/vnd.google-apps.folder' and name='{folder_name}'"
30	folder_list = self.drive.files().list(q=query).execute().get('files', [])
31	if folder_list:
32	folder_id = folder_list[0]['id']
33	print("Folder ID for '{}' is: {}".format(folder_name, folder_id))
34	return folder_id
35	else:
36	print("Folder '{}' not found.".format(folder_name))
37	return None

Copyright © 2024 YITJC All Rights Reserved.

```
38
39      #指定したフォルダ名に含まれる全ファイルID情報取得
40      def get_file_id_in_folder_name(self, folder_name):
41          folder_id = self.get_folder_id(folder_name)
42          if not folder_id:
43              return None
44          file_list = self.drive.files().list(q=f"'{folder_id}' in parents and trashed=false").execute().get('files', [])
45          file_ids = [(file['name'], file['id']) for file in file_list]
46          for file_id in file_ids:
47              print(file_id)
48          return file_ids
49
50      #指定したフォルダ名に含まれる全ファイルID情報(ファイル名、作成日時、ID)取得
51      def get_file_info_in_folder_name(self, folder_name):
52          folder_id = self.get_folder_id(folder_name)
53          if not folder_id:
54              return pd.DataFrame()
55          file_list = self.drive.files().list(q=f"trashed=false and '{folder_id}' in parents", fields="files(id, name,
    createdTime)").execute().get('files', [])
56          file_id_list = [[file['name'], file['createdTime'], file['id']] for file in file_list]
57          for file_info in file_id_list:
58              print("Title: {}, Created Date: {}, ID: {}".format(file_info[0], file_info[1], file_info[2]))
59          return pd.DataFrame(file_id_list, columns=['title', 'createdDate', 'id'])
60
61      #ファイルアップデート
62      #指定ファイルが存在しない場合はアップロード
63      def upload_file_to_drive(self, local_file_path, folder_name):
64          folder_id = self.get_folder_id(folder_name)
65          if folder_id is not None:
66              file_name = os.path.basename(local_file_path)
67              existing_files = self.drive.files().list(q=f"'{folder_id}' in parents and name='{file_name}' and
    trashed=false").execute().get('files', [])
68              if existing_files:
69                  existing_file_id = existing_files[0]['id']
70                  media = MediaFileUpload(local_file_path, resumable=True)
71                  updated_file = self.drive.files().update(fileId=existing_file_id, media_body=media).execute()
72                  print("Updated file:", local_file_path)
72              else:
74                  file_metadata = {
```

Copyright © 2024 YITJC All Rights Reserved.

```
75                          'name': file_name,
76                          'parents': [folder_id]
77                      }
78                      media = MediaFileUpload(local_file_path, resumable=True)
79                      file = self.drive.files().create(body=file_metadata, media_body=media, fields='id').execute()
80                      print("Uploaded file:", local_file_path)
81              else:
82                  print("Failed to upload file: Folder '{}' not found.".format(folder_name))
83
84      #フォルダ作成
85      def create_folder_in_drive(self, folder_name, parent_folder_id):
86              folder_metadata = {
87                  'name': folder_name,
88                  'parents': [{'id': parent_folder_id}],
89                  'mimeType': 'application/vnd.google-apps.folder'
90              }
91              folder = self.drive.files().create(body=folder_metadata, fields='id').execute()
92              print("フォルダID:", folder['id'])
93              return folder['id']
94
95      #フォルダ内のデーター括アップロード
96      def upload_folder_to_drive(self, local_folder_path, folder_name):
97              for item in os.listdir(local_folder_path):
98                  item_path = os.path.join(local_folder_path, item)
99                  if os.path.isfile(item_path):
100                     self.upload_file_to_drive(item_path, folder_name)
101                 elif os.path.isdir(item_path):
102                     folder_name = os.path.basename(item_path)
103                     subfolder_id = self.create_folder_in_drive(folder_name, folder_name)
104                     self.upload_folder_to_drive(item_path, subfolder_id)
105
106     #フォルダ名とファイル名を指定してダウンロード
107     def download_from_file_name(self, folder_name, file_name, target_dir=""):
108             folder_id = self.get_folder_id(folder_name)
109             if not folder_id:
110                 print("Folder not found")
111                 return
112             file_id = self.get_file_id(folder_name=folder_name, file_name=file_name)
113             if not file_id:
```

Copyright © 2024 YITJC All Rights Reserved.

```
114                 print("File not found")
115                 return
116             f = self.drive.files().get(fileId=file_id).execute()
117             if not f.get('trashed', False):
118                 file_path = os.path.join(target_dir, file_name)
119                 request = self.drive.files().get_media(fileId=file_id)
120                 with open(file_path, 'wb') as file:
121                     downloader = MediaIoBaseDownload(file, request)
122                     done = False
123                     while done is False:
124                         status, done = downloader.next_chunk()
125                         print("Download %d%%." % int(status.progress() * 100))
126 
127     # 指定フォルダ内のファイルIDを取得する
128     def get_file_id(self, folder_name=None, file_name=None):
129         folder_id = self.get_folder_id(folder_name)
130         query = f"trashed=false and '{folder_id}' in parents and mimeType != 'application/vnd.google-apps.folder'"
131         if file_name:
132             query += f" and name='{file_name}'"
133         file_list = self.drive.files().list(q=query).execute().get('files', [])
134         if file_name and file_list:
135             return file_list[0]['id']
136         elif file_list:
137             latest_file = max(file_list, key=lambda f: f['createdTime'])
138             return latest_file['id']
139         else:
140             return None
141 
142 if __name__ == '__main__':
143     gdrive = GDrive_Class()
144     try:
145         #フォルダ名、ファイル名定義
146         folder_name = 'BME280'
147         file_name = 'Raspi_temp.csv'
148         # 親フォルダの ID
149         PARENT_FOLDER_ID = '1xj2Pc5yRlUh5VPr-IT1r3cvJdMN7CU55'
150         local_folder_path = '/home/cuu/data'
151         local_file_path = local_folder_path + '/' + file_name
152 
```

Copyright © 2024 YITJC All Rights Reserved.

```
153             #全フォルダ・全ファイルのID情報取得
154             print("***ID List of All foders and files***")
155             #gdrive.google_drive_id_list()
156
157             #フォルダ名を指定してフォルダIDを取得
158             print("***Get folder ID    by specifying the folder Name***")
159             gdrive.get_folder_id(folder_name)
160
161             #指定したフォルダ名に含まれる全ファイルID情報取得
162             print("***ID List of All files    by specifying the folder Name***")
163             gdrive.get_file_id_in_folder_name(folder_name)
164
165             #指定したフォルダ名に含まれる全ファイルID情報(ファイル名、作成日時、ID)取得
166             print("***Information List of All files    by specifying the folder Name***")
167             gdrive.get_file_info_in_folder_name(folder_name)
168
169             #ファイルアップデート
170             #指定ファイルが存在しない場合はアップロード
171             print("***File update    by specifying the local file path***")
172             gdrive.upload_file_to_drive(local_file_path, folder_name)
173
174             #フォルダ作成
175             #gdrive.create_folder_in_drive(folder_name, folder_id)
176             #print("***Create folder    by specifying the folder Name***")
177             #gdrive.create_folder_in_drive("Test",PARENT_FOLDER_ID)
178
179             #フォルダ内のデーター括アップロード
180             print("***Bulk update by specifying the local folder Path***")
181             gdrive.upload_folder_to_drive(local_folder_path, folder_name)
182
183             #フォルダ名とファイル名を指定してダウンロード
184             print("***File download by specifying the file Name***")
185             gdrive.download_from_file_name(folder_name, file_name)
186
187         except KeyboardInterrupt:
188             pass
```

Copyright © 2024 YITJC All Rights Reserved.

Raspberry Pi と Google Drive 間のファイル交換に必要な機能（関数）について解説します。

～全フォルダ・全ファイルの ID 情報取得～

google_drive_id_list()

- 23 行目: drive.files().list メソッドを使用して Google Drive のフォルダ、ファイルリストを取得
  「trashed=false」にしているため、削除（ゴミ箱に移動）されたフォルダやファイルの情報は取得しません。
- 24 行目～25 行目: 取得したリストに含まれているフォルダ、ファイルの ID 情報を取得し、print 表示

～フォルダ名を指定してフォルダ ID を取得～

get_folder_id(folder_name)

第 1 引数: **フォルダ ID を取得するフォルダ名**

- 29 行目: MIME タイプを（Google Drive フォルダ）に指定して、指定したフォルダ名を検索するクエリを作成
- 30 行目: drive.files().list メソッドを使用して Google Drive のフォルダを検索
- 31 行目: foldr_list が空でない（指定した名前のフォルダが見つかった）場合の処理
- 32 行目～34 行目: 指定した名前のフォルダ ID を取得して print 表示
- 35 行目～36 行目: 指定した名前のフォルダが見つからない場合の print 表示定義
- 37 行目: 指定した名前のフォルダが見つからない場合は「None」を返す

～指定したフォルダ名に含まれる全ファイル ID 情報取得～

get_file_id_in_folder_name(folder_name)

第 1 引数: **ファイル ID 情報を取得するフォルダ名**

- 41 行目: get_folder_id()関数から指定したフォルダ名のフォルダ ID を取得
- 42 行目～43 行目: 指定したフォルダ名のフォルダが見つからない場合は「None」を返す
- 44 行目: drive.files().list メソッドを使用して指定したフォルダ名に含まれるファイルリストを取得
  「trashed=false」にしているため、削除（ゴミ箱に移動）されたフォルダやファイルの情報は取得しません。
- 45 行目: ファイルリストから「ファイル名」と「ファイル ID」を取得
- 46 行目～47 行目: ファイル名とファイル ID を print 表示
- 48 行目: ファイル名とファイル ID のリストを返す

～指定したフォルダ名に含まれる全ファイル ID 情報(ファイル名、作成日時、ID)取得～

get_file_info_in_folder_name(folder_name)

第 1 引数: **ファイル ID 情報を取得するフォルダ名**

- 52 行目: get_folder_id()関数から指定したフォルダ名のフォルダ ID を取得
- 53 行目～54 行目: 指定したフォルダ名のフォルダが見つからない場合は空の DataFrame()を返す
- 55 行目: drive.files().list メソッドを使用して指定したフォルダ名に含まれるファイルリストを取得
  「trashed=false」にしているため、削除（ゴミ箱に移動）されたフォルダやファイルの情報は取得しません。
  fields="files(id, name, createdTime)により、ファイル ID、ファイル名、作成日時を指定します。
- 56 行目: ファイルリストから「ファイル名」、「ファイル ID」、「作成日時」を取得
- 57 行目～58 行目: 「ファイル名」、「ファイル ID」、「作成日時」を print 表示

Copyright © 2024 YITJC All Rights Reserved.

- 59 行目: ファイル情報のリストを DataFrame に変換して返す

～ファイルアップロード（同じ名前のファイルがある場合はアップデート）～

upload_file_to_drive(local_file_path, folder_name)

第 1 引数: アップロードするファイルが保存されているディレクトリ

第 2 引数: アップロード先となる Google Drive のフォルダ名

- 64 行目: get_folder_id()関数から指定したフォルダ名のフォルダ ID を取得
- 65 行目: フォルダ ID が見つかった場合の処理
- 66 行目: local_file_path（Raspberry Pi のディレクトリ）からファイル名を抽出
- 67 行目: drive.files().list メソッドを使用して同じ名前をもつファイルリストを取得
- 68 行目～72 行目: 同じ名前のファイルが存在する場合の処理
  同じ名前のファイル ID を取得してアップデート（上書き保存）
- 73～80 行目: 同じ名前のファイルが存在しない場合の処理
  新規にファイルをアップロード
- 81 行目～82 行目: フォルダ ID が見つからなかった場合のエラーメッセージ表示

～フォルダ作成～

create_folder_in_drive(folder_name, parent_folder_id)

第 1 引数: Google Drive に作成するフォルダ名

第 2 引数: フォルダを作成する場所（親ディレクトリ）

- 86 行目～90 行目: フォルダのメタデータ（ファイルの内容や作成日時、更新日時など）
  ここでは、メタデータとして「フォルダ名」、「MIME タイプ」、「親ディレクトリ」を定義
- 91 行目: DRIVE.files().create メソッドを使用してフォルダを作成し、execute()メソッドでアップロードを実行
- 92 行目: 作成したフォルダの「フォルダ ID」を取得して print 表示
- 93 行目: フォルダ ID を返す

～フォルダ内のデーター括アップロード～

upload_folder_to_drive(local_folder_path, folder_name)

第 1 引数 アップロードするファイルが保存されているディレクトリ

第 2 引数: アップロード先となる Google Drive のフォルダ名

- 97 行目～104 行目: 指定フォルダに保存されているすべての「ファイル」、「フォルダ」について繰り返し処理
- 98 行目: local_folder_path から完全なファイルパスを取得
- 99 行目～100 行目: 「ファイル」である場合の処理を定義
  ファイルの場合は、upload_file_to_drive()関数を使用してアップロード
- 101 行目～104 行目: 「フォルダ」である場合の処理を定義
  まず、フォルダ（サブフォルダ）の名前を取得してから Google Drive に同じ名前のフォルダを作成します。
  次に、Google Drive に作成したフォルダの「フォルダ ID」を取得して、サブフォルダ内のファイルを再帰的に処理します。

Copyright © 2024 YITJC All Rights Reserved.

～フォルダ名とファイル名を指定してダウンロード～

download_from_file_name(folder_name, file_name, target_dir="")

第 1 引数: ダウンロードするファイルが保存されているフォルダ名

第 2 引数: ダウンロードするファイル名

第 3 引数 ダウンロード先のディレクトリ

- 108 行目: get_folder_id()関数から指定したフォルダ名のフォルダ ID を取得
- 109 行目～111 行目: 指定フォルダが見つからなかった場合の処理
- 112 行目: get_file_id()関数から指定したファイルのファイル ID を取得
- 113 行目～115 行目: 指定ファイルが見つからなかった場合の処理
- 116 行目: drive.files().get メソッドを使用してファイル情報を取得
- 117 行目: ファイルがゴミ箱に入っていないことを確認
- 118 行目: ダウンロード先のファイルパスを作成
  target_dir（ダウンロード先ディレクトリ）を指定しない場合は、プログラムを実行したディレクトリにダウンロード
- 119 行目: drive.files().get_media メソッドを使用してファイルのメディアリクエストを作成
- 120 行目: 指定されたパスにバイナリ書き込みモード ('wb') でファイルを開きます。
  ファイルが存在しない場合は新たに作成されます。
- 121 行目: MediaIoBaseDownload()メソッドを使用してファイルをダウンロード
- 122 行目: ダウンロード完了フラグ「done」を初期化
- 123 行目～125 行目: ダウンロードが完了するまでループ処理
- 125 行目: ダウンロードの進捗をパーセント（%）表示

～指定フォルダ内のファイル ID を取得～

get_file_id(folder_name=None, file_name=None)

第 1 引数: ファイル IC を取得するファイルが保存されているフォルダ名

第 2 引数: ファイル ID を取得するファイル名

- 129 行目: get_folder_id()関数から指定したフォルダ名のフォルダ ID を取得
- 130 行目: 検索クエリを測定
  mimeType != 'application/vnd.google-apps.folder'と指定することで、フォルダ以外のファイルを検索
- 131 行目～132 行目: ファイル名（file_name）が指定されている場合は、ファイル名を検索クエリに追加
- 133 行目: 検索クエリを使用してファイルリストを取得
- 134 行目～135 行目: 「ファイル名指定」かつ「ファイルリストにファイルが存在する」場合の処理
  ファイル ID を返す
- 136 行目～138 行目: ファイル名が指定されておらず、ファイルリストにファイルが存在する場合の処理
  最新のファイルを選択してファイル ID を返す
- 139 行目～140 行目: ファイル名が指定されておらず、ファイルリストが空の場合は「None」を返す

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説（36_Pydrive.py）～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import GDrive_Class
5	
6	#フォルダ名、ファイル名定義
7	folder_name = 'BME280'
8	file_name = 'Raspi_temp.csv'
9	#親フォルダの ID
10	PARENT_FOLDER_ID = '1xj2Pc5yRlUh5VPr-IT1r3cvJdMN7CU55'
11	local_folder_path = '/home/cuu/data'
12	local_file_path = local_folder_path + '/' + file_name
13	
14	if __name__ == '__main__':
15	gdrive = GDrive_Class.GDrive_Class()
16	try:
17	#全フォルダ・全ファイルのID情報取得
18	print("***ID List of All foders and files***")
19	gdrive.google_drive_id_list()
20	
21	#フォルダ名を取得してフォルダIDを取得
22	print("***Get folder ID　by specifying the folder Name***")
23	gdrive.get_folder_id(folder_name)
24	
25	#指定したフォルダ名に含まれる全ファイルID情報取得
26	print("***ID List of All files　by specifying the folder Name***")
27	gdrive.get_file_id_in_folder_name(folder_name)
28	
29	#指定したフォルダ名に含まれる全ファイルID情報(ファイル名、作成日時、ID)取得
30	print("***Information List of All files　by specifying the folder Name***")
31	gdrive.get_file_info_in_folder_name(folder_name)
32	
33	#ファイルアップデート
34	#指定ファイルが存在しない場合はアップロード
35	print("***File update　by specifying the local file path***")
36	gdrive.upload_file_to_drive(local_file_path, folder_name)
37	

Copyright © 2024 YITJC All Rights Reserved.

```
38              #フォルダ作成
39              #gdrive.create_folder_in_drive(folder_name, folder_id)
40              #print("***Create folder   by specifying the folder Name***")
41              #gdrive.create_folder_in_drive("Test",PARENT_FOLDER_ID)
42
43              #フォルダ内のデーター括アップロード
44              print("***Bulk update by specifying the local folder Path***")
45              gdrive.upload_folder_to_drive(local_folder_path, folder_name)
46
47              #フォルダ名とファイル名を指定してダウンロード
48              print("***File download by specifying the file Name***")
49              gdrive.download_from_file_name(folder_name, file_name)
50
51          except KeyboardInterrupt:
52              pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: GDrive_Class クラスモジュールをインポート
- 7 行目: フォルダ名設定（folder_name）
- 8 行目: ファイル名定義（file_name）
- 10 行目: 親フォルダ ID 定義（PARENT_FOLDER_ID）
- 11 行目: ローカルディレクトリ設定（local_folder_path）
- 12 行目: ローカルファイルが保存されているディレクトリ設定（local_file_path）
- 14 行目～52 行目: main プログラム
- 15 行目: クラス生成
  4 行目でインポートした GDrive_Class モジュールからクラスを生成し、変数「gdrive」に代入しています。
- 16 行目～49 行目 :try ブロック
- 18 行目～19 行目: 全フォルダ・ファイルの ID 取得
- 22 行目～23 行目: 指定フォルダの ID 取得
- 26 行目～27 行目: 指定フォルダ（フォルダ名指定）に含まれる全ファイルの ID 取得
- 30 行目～31 行目: 指定フォルダ（フォルダ名指定）に含まれる全ファイルの情報(ファイル名、作成日時、ID)取得
- 35 行目～36 行目: ファイルアップデート（上書き）
  同じ名前のファイルがない場合は、新規アップロード
- 40 行目～41 行目: フォルダ作成
- 44 行目～45 行目: ローカルフォルダのデータを一括アップデート（上書き）
- 48 行目～49 行目: フォルダ名とファイル名を指定してファイルをダウンロード

Copyright © 2024 YITJC All Rights Reserved.

- 54 行目～55 行目：except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

8.6 オンラインストレージを活用した IoT システム（ver2）

これまで、「データ取得」、「データ収集・活用」、「データ可視化」、「オンラインストレージへの保存」など、IoT システムを構築するために必要な処理（プログラム）を学習してきました。IoT システムを構築するためには、さまざまな処理（プログラム）を統合する必要があります。しかし、複数の処理機能を 1 つのプログラムに統合してシステムを構築した場合、プログラムの「可読性」やシステムの「保守性」が失われ、システムの変更や修正が困難になります。

ここでは、さまざまな処理機能をもつプログラムを「シェルスクリプト」を使用してシステムを構築する方法について学習します。シェルスクリプトを使用した場合、各機能（プログラム）は独立しているため、変更や修正が容易になります。また、プログラム間の依存関係を最小限に抑えることができるため、システムの安定動作につながります。

～シェルスクリプト～

シェルスクリプトは、複数のコマンドやプログラムを組み合わせて実行するためのせるためのプログラミング言語（スクリプト言語）です。シェルスクリプトは、OS を操作するための「シェル」上で実行されます。

「シェル」は OS に対して命令を出すことができるプログラムであり、Windows PC の場合は「コマンドプロンプト」や「PowerShell」が一般的な例です。

～シェルスクリプト構成～

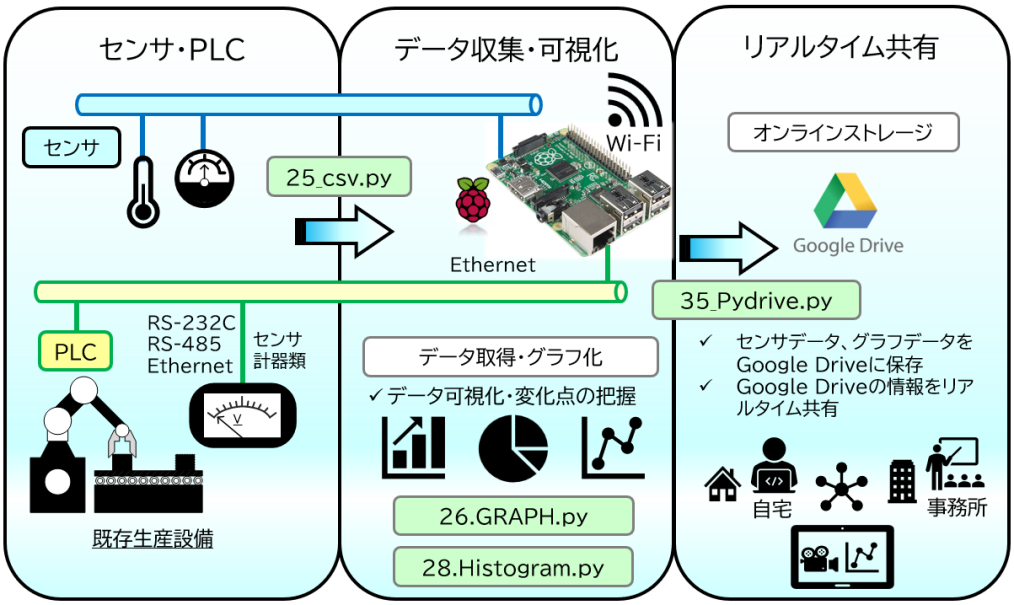
BME280（温度、湿度、気圧）センサのデータを収集・加工（グラフ化）し、定期的にオンラインストレージへアップロードします。

➔ これまでに作成したプログラムを組み合わせて IoT システム ver2 を構築します。

使用するプログラム	処理内容
25_CSV.py	センサデータを csv ファイルに蓄積 ✓ Raspi_temp.csv ✓ Raspi_hum.csv ✓ Raspi_pressure.csv
26_GRAPH.py	CSV ファイルを読み込み、グラフ作成 ✓ Raspi_temp.png ✓ Raspi_hum.png ✓ Raspi_pressure.png
28_Histogram.py	CSV ファイルを読み込み、ヒストグラム作成 ✓ Raspi_temp_Histogram.png ✓ Raspi_hum_Histogram.png ✓ Raspi_pressure_Histogram.png
35_Pydrive.py	「/home/cuu/data」ディレクトリに保存されているデータをすべて Google Drive にアップデート（上書き）

Copyright © 2024 YITJC All Rights Reserved.

シェルスクリプトを使用して構築する IoT システム ver2 のイメージを示します。4 つのプログラムがもつ機能を統合し、センサデータを「リアルタイム共有」する IoT システム ver2 を構築します。



手順①～サービス・タイマー停止～

「25_CSV.py」は、6 章「データ収集と活用」で構築したデータ収集自動化システムに使用されています。

現在も、「accumulation.service」と「accumulation.timer」により、30 秒ごとに「25_CSV.py」が実行されています。新たなシステムを構築するため、このサービス(タイマー)を停止します。

(ycuu) cuu@raspberrypi:$ sudo systemctl stop accumulation.timer

(ycuu) cuu@raspberrypi:$ sudo systemctl stop accumulation.service

(ycuu) cuu@raspberrypi:$ sudo systemctl disable accumulation.timer

(ycuu) cuu@raspberrypi:$ sudo systemctl disable accumulation.service

```
(ycuu) cuu@raspberrypi:~ $ sudo systemctl stop accumulation.timer
(ycuu) cuu@raspberrypi:~ $ sudo systemctl stop accumulation.service
(ycuu) cuu@raspberrypi:~ $ sudo systemctl disable accumulation.timer
Removed "/etc/systemd/system/multi-user.target.wants/accumulation.timer".
(ycuu) cuu@raspberrypi:~ $ sudo systemctl disable accumulation.service
Removed "/etc/systemd/system/multi-user.target.wants/accumulation.service".
```

次のコマンドを実行すると、現在の状態が inactive(dead)になっていることを確認できます。

(ycuu) cuu@raspberrypi:$ sudo systemctl status accumulation.timer

この画面表示を終了する場合は、「q」を押してください。

```
(ycuu) cuu@raspberrypi:~ $ sudo systemctl status accumulation.timer
○ accumulation.timer - Accumulate data every 30 seconds
     Loaded: loaded (/etc/systemd/system/accumulation.timer; disabled; preset:
     Active: inactive (dead)
    Trigger: n/a
   Triggers: ● accumulation.service
```

Copyright © 2024 YITJC All Rights Reserved.

手順②～プログラム配置～

作成したプログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

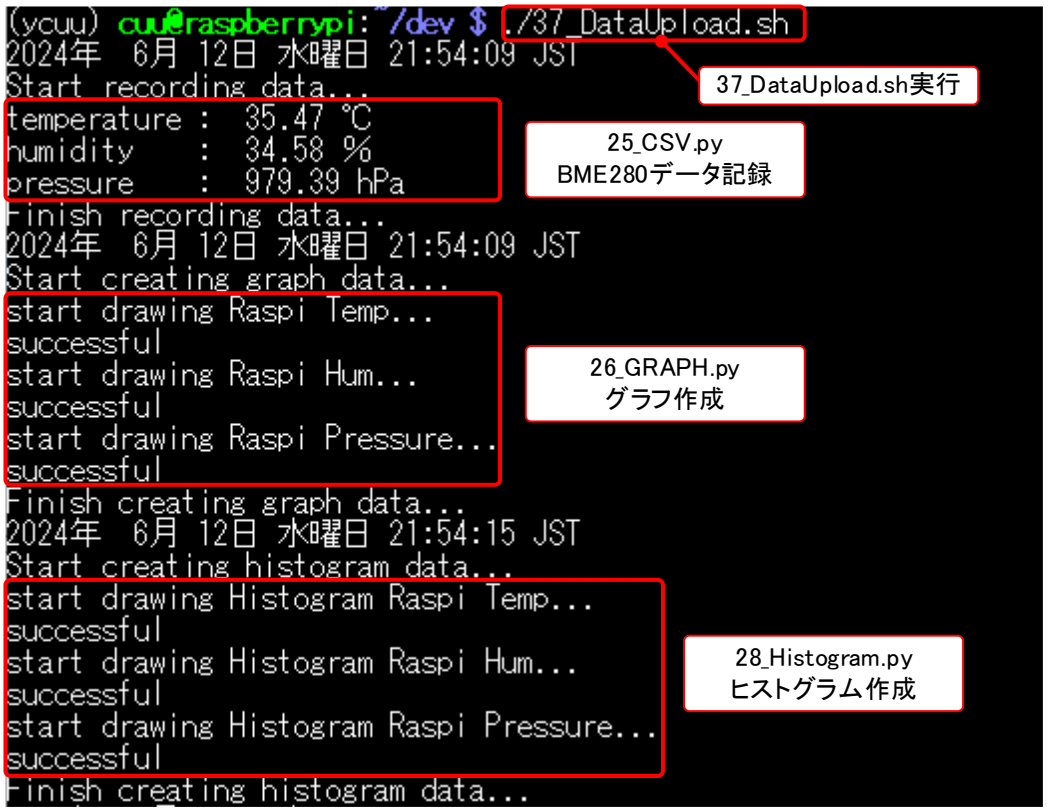
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 37_DataUpload.sh

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./37_DataUpload.sh

次の実行画面が表示されます。この後のプログラム解説で説明しますが、シェルスクリプトが実行している各プログラムの処理履歴(時刻・処理内容)が明確になるようにシェルスクリプトを作成しています。



(次のページへ続く)

Copyright © 2024 YITJC All Rights Reserved.

（続き）

～プログラム解説～

行	プログラム
1	#!/bin/sh
2	#データ記録(from BME280 to csv)
3	date
4	echo Start recording data...
5	/home/cuu/dev/./25_CSV.py
6	echo Finish recording data...
7	
8	#グラフ作成・グラフ画像生成
9	date
10	echo Start creating graph data...
11	/home/cuu/dev/./26_GRAPH.py
12	echo Finish creating graph data...
13	
14	#ヒストグラム作成・グラフ画像生成
15	date
16	echo Start creating histogram data...
17	/home/cuu/dev/./28_Histogram.py
18	echo Finish creating histogram data...
19	
20	#Google Driveアップロード
21	date
22	echo Start uploading all data...
23	/home/cuu/dev/./35_Pydrive.py
24	echo Finish uploading all data...

Copyright © 2024 YITJC All Rights Reserved.

25	date

- 1 行目: Shebang（シバン、シェバン）
  Raspberry Pi OS（Raspbian）は使用するシェルが「sh」、「bash」、「dash」などから選ぶことができます。
  ここでは、「sh」を選択してシェルスクリプトを作成しています。

～BME280（温度・湿度・気圧）センサデータ記録～

- 3 行目～6 行目: センサデータ記録
- 3 行目: 日付・時刻表示
- 4 行目: コメント「Start recording data...」表示
- 5 行目: 25_CSV.py 実行
- 6 行目: コメント「Finish recording data...」表示

～BME280（温度・湿度・気圧）センサデータから時系列グラフ作成～

- 9 行目～12 行目: センサデータからグラフ作成
- 9 行目: 日付・時刻表示
- 10 行目: コメント「Start creating graph data...」表示
- 11 行目: 26_GRAPH.py 実行
- 12 行目: コメント「Finish creating graph data...」表示

～BME280（温度・湿度・気圧）センサデータからヒストグラム作成～

- 15 行目～18 行目: センサデータからヒストグラム作成
- 15 行目: 日付・時刻表示
- 16 行目: コメント「Start creating histogram data...」表示
- 17 行目: 28_Histogram.py 実行
- 18 行目: コメント「Finish creating histogram data...」表示

～Raspberry Pi「/home/cuu/data」ディレクトリのファイルを一括アップロード～

- 21 行目～25 行目: センサデータからヒストグラム作成
- 21 行目: 日付・時刻表示
- 22 行目: コメント「Start uploading all data...」表示
- 23 行目: 35_Pydrive.py 実行
- 24 行目: コメント「Finish uploading all data...」表示
- 25 行目: 日付・時刻表示

Copyright © 2024 YITJC All Rights Reserved.

「37_DataUpload.sh」スクリプトを定期的に実行し、データ記録からオンラインストレージへの保存まで自動で行うシステムを構築します。定期的に自動起動する仕組みとして「Systemd」の仕組みを使用します。

そのため、手順①で停止した「accumulation.service」および「accumulation.timer」の内容を編集します。

手順⑤～Service ファイル編集～

「accumulation.service」ファイルを編集するため、「/etc/systemd/system」ディレクトリへ移動します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

```
(ycuu) cuu@raspberrypi:~/dev $ cd ..
(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system
(ycuu) cuu@raspberrypi:/etc/systemd/system $
```

～参考～

Unit ファイルの「上書き」、「終了」は、次の手順を参考にしてください。

・編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

・「Enter」キーを押下して上書き保存を実行

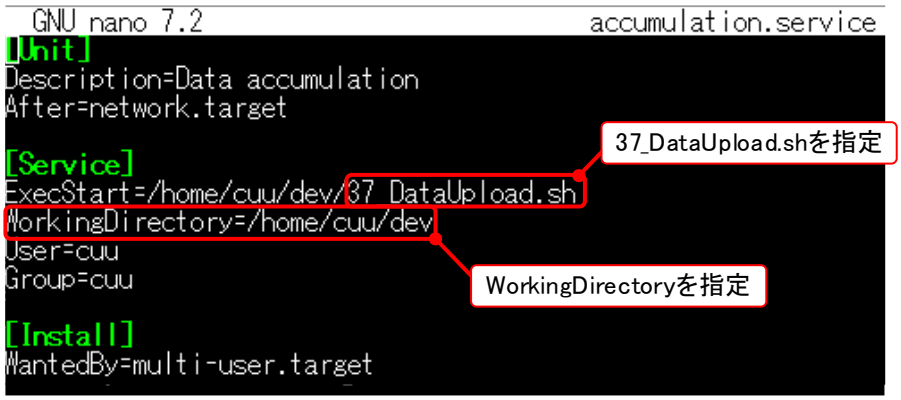
・「Ctrl + X」キーを押して元の画面に戻る

まず、「accumulation.service」ファイルを編集します。

現在の Unit ファイルは自動起動するプログラムが「25_CSV.py」に設定されていますが、

「37_DataUpload.sh」に変更します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano accumulation.service



参考～WorkingDirectory の指定～

「Service」セクションに WorkingDirectory の設定を追加しています。WorkingDirectory は、「Systemd」サービスが起動するときのディレクトリです。

今回実行するスクリプト（37_DataUpload.sh）の中に、「35_Pydrive.py」が含まれています。このプログラムは、「/home/cuu/dev」ディレクトリにある認証ファイル（credentials.json）を使用して Google Drive にファイルをアップロードします。このように、他のファイルを参照してプログラムを実行する場合は WorkingDirectory の設定が重要です。

- WorkingDirectory を設定しない場合…
  ルートディレクトリ「/home/cuu」が WorkingDirectory になります。
  この場合、「/home/cuu/dev」ディレクトリにある認証ファイル（credentials.json）を読み込めないため、
  「Systemd」サービスが「37_DataUploac.sh」を実行すると OAuth 認証エラーが発生します。

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～Timer ファイル編集～

続いて、「accumulation.timer」ファイルを編集します。

現在の Unit ファイルは 30 秒（30sec）に設定されていますが、「37_DataUpload.sh」の処理時間を考慮して、5 分（5min）に変更します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano accumulation.timer

手順⑦～サービス再読み込み～

編集したサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順⑧～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.service

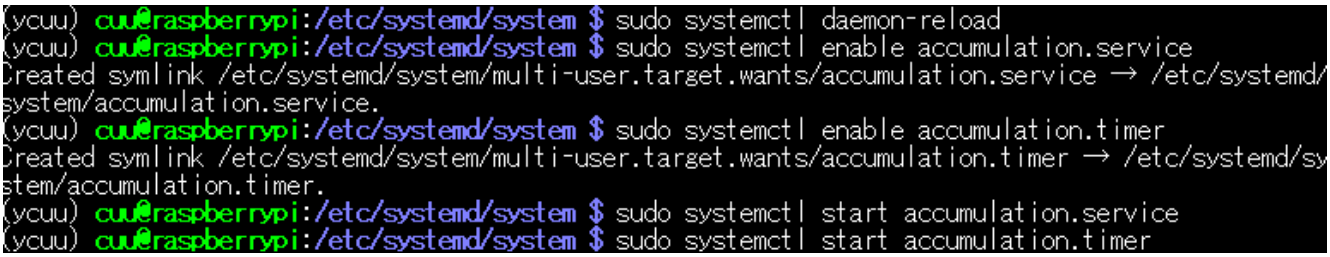
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.timer

手順⑨～サービス開始～

次のコマンドを実行してサービスを開始します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.service

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.timer



Copyright © 2024 YITJC All Rights Reserved.

手順⑩～サービス動作状況確認～

サービスの動作状況を確認します。

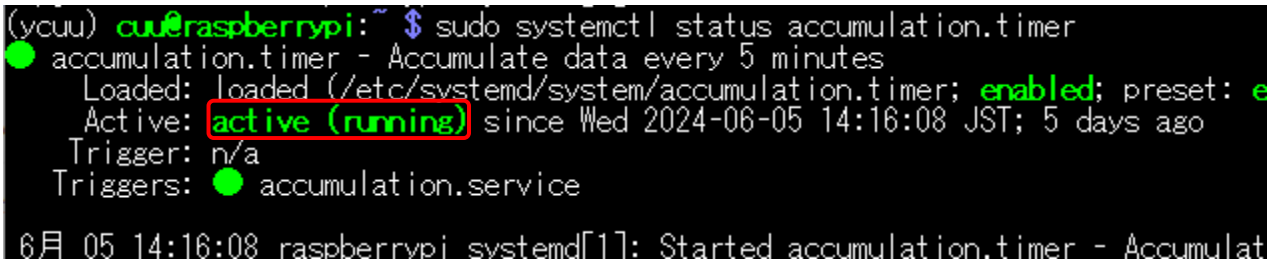
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status accumulation.timer

正常に稼働（自動起動）していれば、次の画面が表示されます。

この画面表示を終了する場合は、「q」を押してください。

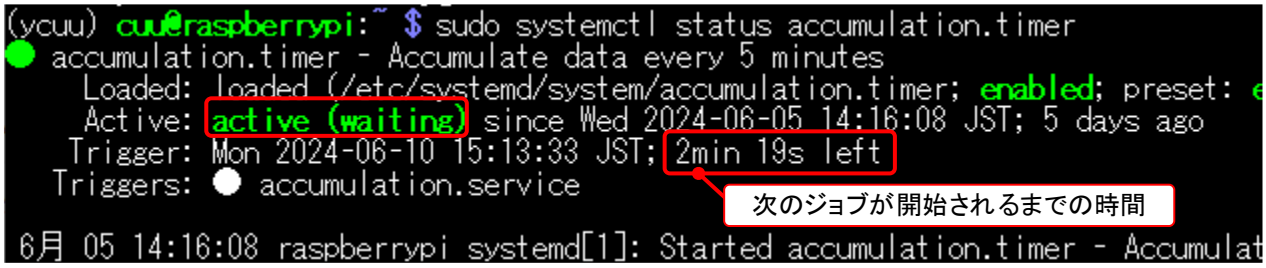
- active（running）の状態

  サービスが現在実行中であることを意味しています。



- active（waiting）の状態

  次のジョブが開始されるのを待っている状態（次のジョブが開催されるまでの時間が表示されます）



手順⑪～Google Drive へのアップロード確認～

Google Drive に定期的なデータ更新がされていることを確認してください。

Copyright © 2024 YITJC All Rights Reserved.

参考～システムログの確認～

第 6 章「データ収集と活用」で学習した「journalctl」コマンドを使用して、「accumulation.service」のログを確認します。

(ycuu) cuu @raspberrypi:~ $ journalctl -u accumulation.service -n 20

- **オプションの設定（例）**

「-u Unit 名」：指定した Unit のログを表示

「-n 数」：末尾（直近）から指定した行数のログを表示

「-f」：リアルタイム監視（デバッグに便利）

「-xe」：詳細情報を表示

サービスで自動起動しているスクリプト（37_DataUpload.sh）でコメントや時刻を表示しているため、システムの稼働状況を正確に把握できます。

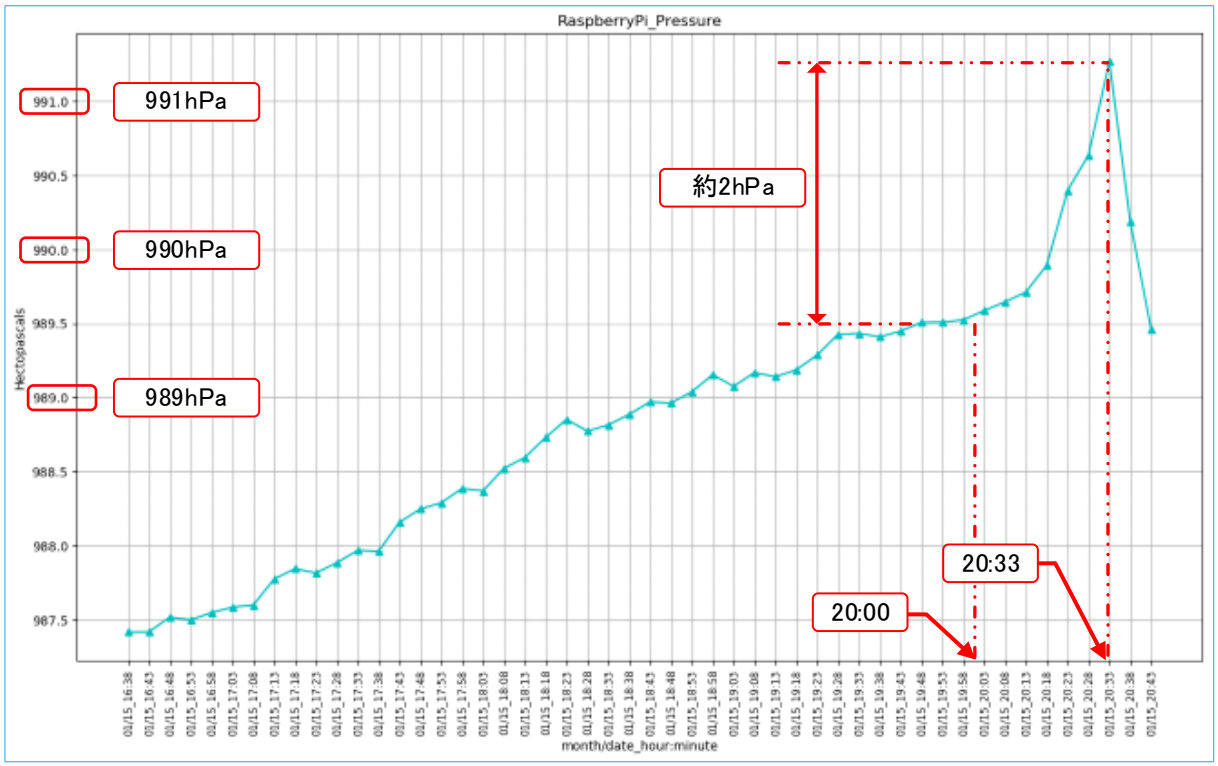
Copyright © 2024 YITJC All Rights Reserved.

参考～フンガ・トンガ噴火の観測～

2022 年 1 月 15 日 13 時頃（日本時間）、南太平洋・トンガ沖の海底火山が大噴火しました。この影響により日本の太平洋岸では、1 メートルを超える高さの潮位変化が観測され、全国 8 県で一時 23 万人近くが避難対象となりました。この「潮位変化」は地震による津波とは異なり、「揺れを伴わない津波」として当時話題になりました。当時のニュースでは、火山による衝撃波の影響から、日本国内で一時的に気圧が 2hPa ほど高まったと報じていました。

この「2hPa」の気圧変化を、本講座で構築した IoT システムが観測していたのでご紹介します。

次の画像は、本講座で使用しているセンサと同じ「BME280」の気圧データを時系列グラフで可視化したものです。午後 8 時台に、ニュースで報じられた情報と同じく「約 2hPa」の気圧変化を観測しています。



以下、簡易的な計算ですが、観測した気圧変化が「噴火の影響」であることを確認しました。

- 噴火した火山～観測地点までの距離

  トンガ-フンガ・ハアパイ火山から観測地点までの距離は、7,889km（Google Earth による測定）

- 噴火発生時刻

  気象衛星「ひまわり 8 号」が撮影した画像から、噴火は、日本時間 15 日 13 時 10 分に発生し、噴火による衝撃波がしばらく続いたことがわかります。また、13 時 50 分～14 時 10 分頃の画像データを確認すると、火山からの衝撃波が同心円の波紋として周囲に広がっている様子がうかがえます。

  （参考: http://agora.ex.nii.ac.jp/digital-typhoon/volcano-eruption/2022-hunga-tonga/）

- 音速

  海面（1 気圧）、気温 15℃と仮定すると、音速は約 340m/sec（= 20.4km/min）

  これらの数字から、検証してみると。

  7,889 [km] ÷ 20.4 [km/min] ≒ 387 [min]

  20 時 00 分頃から気圧の変化が観測されているので、そこから 387 [min]さかのぼると、

  20 時 00 分 - 387 [min] = 13 時 33 分となり、噴火による衝撃波の影響であることが分かります。

Copyright © 2024 YITJC All Rights Reserved.

第 9 章 クラウドサービスの活用～通知機能・AI～

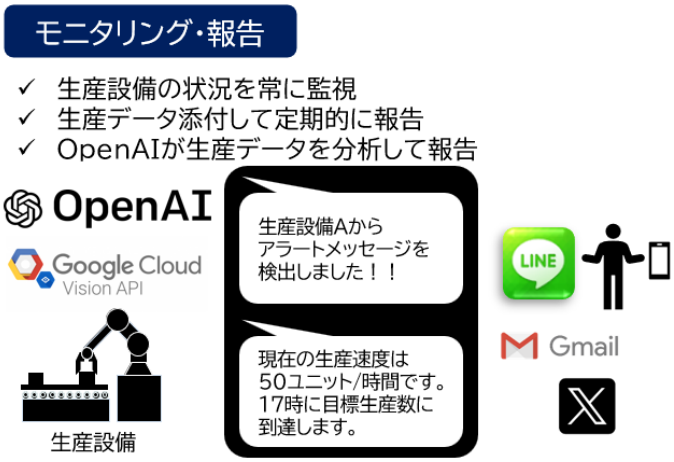
第 8 章では、リアルタイムに情報を共有するためのツールとして「オンラインストレージ」の活用方法を学びました。収集したデータを最大限に活用するためには、アプリケーションが不可欠です。しかし、アプリケーションの「開発期間」や「開発費」、IoT システムの導入に係る「費用対効果」を考慮すると、新規開発は困難な選択です。

そのため、本講座は API（Application Programming Interface）が無料で公開されているクラウドサービスを活用します。API を利用することで開発期間を大幅に短縮できます。また、無料で利用できるクラウドサービスが多いため、運用コストも限りなく抑えることができます。

具体的には、メールや X（旧 Twitter）、LINE などのアプリケーション（通知機能）を使用して、生産設備の「リアルタイム監視」や「異常発生通知」システムを構築する方法について学習します。これにより、生産設備「監視業務の自動化」や「異常の早期発見」を実現し、不良品や材料費のロスを削減します。また、異常発生前後のデータを蓄積・分析することで「予知保全」の実現も期待できます。

また、近年技術の進展が著しい「AI 技術」の活用方法についても学習します。「OpenAI API」は、通常の「ChatGPT」と同様に文章の生成や対話のタスクを実行できますが、さらに OpenAI が提供している他のモデルを利用することで、画像や音声の生成、データ分析などの機能も利用できます。

「Google Cloud Vision」は、画像の中に含まれる対象の抽出や識別に特化しており、オブジェクト認識や顔認識、テキスト検出などの機能を提供しています。これらの「AI 技術」は、生産設備のモニタリングや生産状況の分析報告などの用途で活用が期待できます。



Copyright © 2024 YITJC All Rights Reserved.

9.1 通知機能～Gmail～

Python プログラムを使用して Google が提供する無料のメールサービス「Gmail」を利用する方法について学習します。

手順①～アプリパスワードの発行～

外部アプリ（Python プログラム）が Google のサービスを利用する場合、アプリパスワードを発行する必要があります。アプリパスワードを発行するため、Google アカウントの「2 段階認証」を有効に設定します。

次の URL をクリックし、Google アカウントでログインしてください。

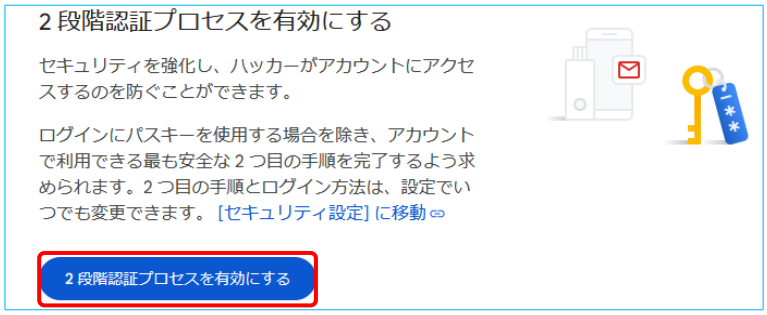
[URL] https://myaccount.google.com/intro/security?hl=ja

「2 段階認証」を有効にするため、「携帯電話番号」を追加されていない方は先に追加してください。

「再設定用」の電話番号を追加したあと、「セキュリティタグ」から「2 段階認証」をクリックしてください。

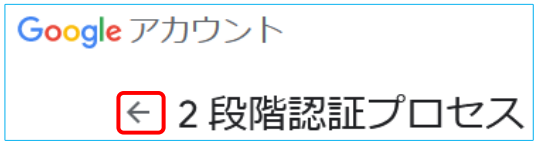
Copyright © 2024 YITJC All Rights Reserved.

「2 段階認証を有効にする」をクリックしてください。



「2 段階認証」の設定に成功すると、次の画面が表示されます。

Google アカウント画面に戻り、「セキュリティ」タグから「アプリパスワード」をクリックしてください。



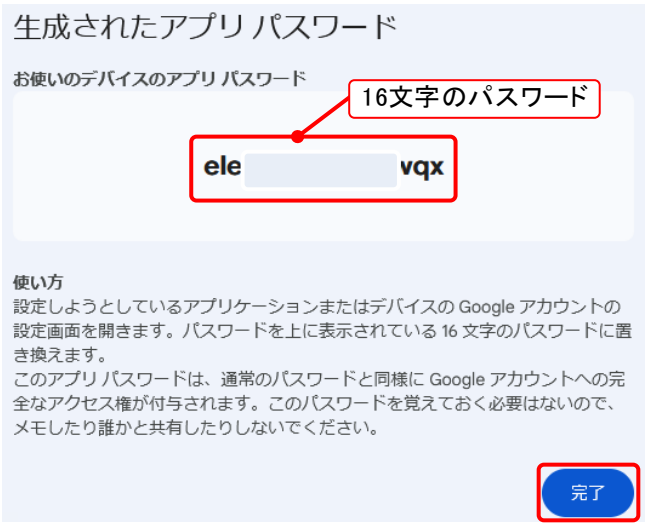
Copyright © 2024 YITJC All Rights Reserved.

「アプリパスワード」が表示されない場合は、次の URL をクリックしてください。

[URL] https://myaccount.google.com/apppasswords?pli=1&rapt=AEjHL4OLT9SmujePkRJd1lB-NBFaNRaFG2THMBlic3KVlu8_aTykSVVtURSJbm6j9pf57-9EGnlC7y0dqP_PqRLVS3hTPIR5Ew

アプリ固有のパスワードを作成するため、「アプリ名」を入力したあと、「作成」をクリックしてください。

生成されたパスワードはプログラムで使用するので、メモ帳などに記録してください。



手順②～環境変数設定～

Gmail アカウントの情報を環境変数に設定します。

参考～環境変数～

環境変数は、Raspberry Pi で動作しているシステムやアプリケーションなどの設定情報を保存するための変数です。シェルやプログラムから利用できる変数です。ユーザー情報やパスワードなどを環境変数に設定することで、これらの情報をプログラム内に記述する必要がなくなり、プログラムの柔軟性やセキュリティ強化につながります。

Copyright © 2024 YITJC All Rights Reserved.

「export」コマンドを使用して環境変数を追加します。

```
export 変数名='設定する情報'
```

環境変数を削除する場合は、「-n」オプションを使用します。

```
export -n 変数名
```

環境変数に登録する情報は、次の情報です。

- SMTP_SERVER: Gmail の送信メールサーバーSMTP（Simple Mail Transfer Protocol）のアドレス
- SMTP_PORT: 送信メールサーバーのポート番号
- EMAIL: 送信元メールアドレス
- APP_PASSWORD: Gmail のアプリパスワード

```
export SMTP_SERVER='smtp.gmail.com'
export SMTP_PORT=587
export EMAIL='cuu.yamanashi@gmail.com'
export APP_PASSWORD='elec xxxx xxxx xwqx'
```

Raspberry Pi が起動したときに環境変数を自動的に設定するため、「.bashrc」ファイルに環境変数の設定コマンドを記述します。

「.bashrc」ファイルは、第 3 章「センサデータ取得」で仮想環境をアクティブにする記述を追加しています。

ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

(ycuu) cuu@raspberrypi:$ nano .bashrc

「.bashrc」ファイルの画面が起動したら、末尾に「export」コマンドで変数を 4 つ追加（記述）します。

記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

手順③～環境変数の反映～

続いて、次のコマンドを実行して設定した環境変数を反映します。

(ycuu) cuu@raspberrypi:$ source ~/.bashrc

```
cuu@raspberrypi:~ $ source ~/.bash_profile
```

「~」の部分（チルダ）は、「Shift」+「～」で入力します。

Copyright © 2024 YITJC All Rights Reserved.

手順④～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

38_Gmail.py

[プログラム編集個所]

- 50 行目 (送信先メールアドレス)

  送信先メールアドレスを編集してください。

  ```
  to_email = 'xxxxx@gmail.com'       #送信先メールアドレスを設定
  ```

手順⑤～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

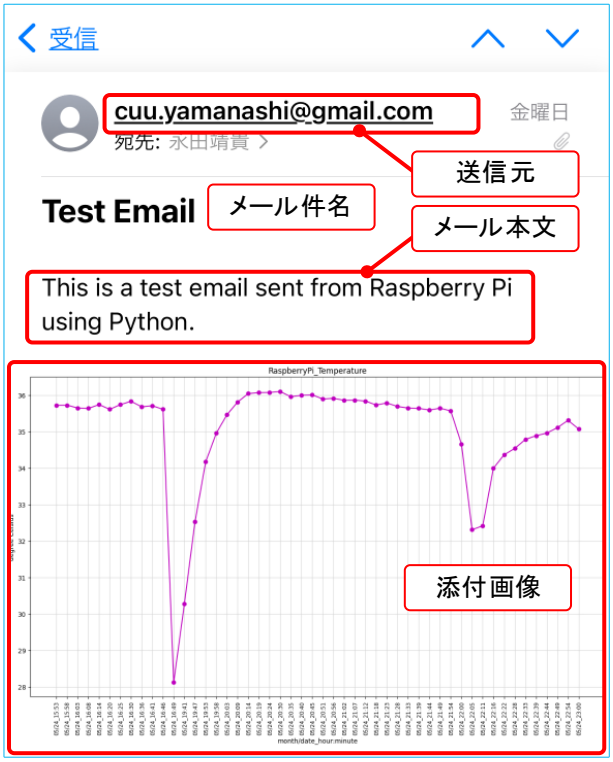
```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 38_Gmail.py
```

手順⑥～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./38_Gmail.py
```

プログラムを実行すると、メールが送信されます。送信先のメールアドレスでメールを受信して内容を確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import smtplib　　#SMTPプロトコルを使用してメールを送信するためのライブラリ
5	import os　　#OSとの対話を提供するライブラリ
6	from email.mime.multipart import MIMEMultipart　#MIME形式のマルチパートメッセージを作成するためのクラス
7	from email.mime.text import MIMEText　　#MIME形式のテキストメッセージを作成するためのクラス
8	from email.mime.base import MIMEBase　　#MIME形式のベースメッセージを作成するためのクラス
9	from email import encoders　　#MIMEメッセージのエンコードを行うためのライブラリ
10	
11	#Gmailアカウントの情報を環境編集から取得
12	def get_gmail_settings():
13	smtp_server = os.getenv('SMTP_SERVER')
14	smtp_port = os.getenv('SMTP_PORT')
15	email = os.getenv('EMAIL')
16	app_password = os.getenv('APP_PASSWORD')
17	return smtp_server, smtp_port, email, app_password
18	
19	#メール作成(添付ファイルは任意=None)
20	def create_email(from_email, to_email, subject, body, attachment_path=None):
21	msg = MIMEMultipart()　　#マルチパートメッセージを作成
22	#MIMEText(本文)
23	msg['From'] = from_email　　#送信元アドレスを設定
24	msg['To'] = to_email　　#送信先アドレスを設定
25	msg['Subject'] = subject　　#メールの件名を設定
26	msg.attach(MIMEText(body, 'plain')) #テキスト形式の本文を添付
27	
28	#MIMEBase(添付ファイル)
29	if attachment_path:
30	attachment = MIMEBase('image', 'png')　　#添付ファイルのMIMEタイプ指定
31	with open(attachment_path, 'rb') as attachment_file:
32	attachment.set_payload(attachment_file.read())　#添付ファイルを読み込み
33	encoders.encode_base64(attachment)　　#ファイルをBase64形式にエンコード
34	attachment.add_header('Content-Disposition', f'attachment; filename={os.path.basename(attachment_path)}')　#添付ファイルのヘッダーを設定
35	msg.attach(attachment)　　#添付ファイルをメールに添付
36	return msg

Copyright © 2024 YITJC All Rights Reserved.

```
37
38  #メール送信関数
39  def send_email(from_email, to_email, msg):
40      #Gmailアカウントの情報を環境編集から取得
41      smtp_server, smtp_port, email, app_password = get_gmail_settings()
42      server = smtplib.SMTP(smtp_server, smtp_port)     #SMTPサーバーに接続
43      server.starttls()                                   #SMTP通信をTLSで暗号化
44      server.login(email, app_password)                  #Gmailにログイン
45      server.sendmail(from_email, to_email, msg.as_string())   #メールを送信
46      server.quit()                                       #SMTPサーバーから切断
47
48  def main():
49      from_email = os.getenv('EMAIL')              #送信元メールアドレスを取得(環境変数に設定)
50      to_email = 'xxxxx@gmail.com'       #送信先メールアドレスを設定
51      subject = 'Test Email'                        #メールの件名を設定
52      body = 'This is a test email sent from Raspberry Pi using Python.'   #メールの本文を設定
53      attachment_path = '/home/cuu/data/Raspi_temp.png'                    #添付ファイルのパスを設定
54      #メール作成
55      msg = create_email(from_email, to_email, subject, body, attachment_path)
56      #メール送信
57      send_email(from_email, to_email, msg)
58
59  if __name__ == '__main__':
60      try:
61          main()
62      except KeyboardInterrupt:
63          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: SMTP プロトコルを使用してメールを送信するためのライブラリ「smtplib」をインポート
- 5 行目: OS と対話するためのライブラリ「os」をインポート
- 6 行目: MIME 形式のマルチパートメッセージを作成するためのクラスモジュールをインポート
- 7 行目: MIME 形式のテキストメッセージを作成するためのクラスモジュールをインポート
- 8 行目: MIME 形式のベースメッセージを作成するためのクラスモジュールをインポート
- 9 行目: MIME メッセージのエンコードを行うためのクラスモジュールをインポート

Copyright © 2024 YITJC All Rights Reserved.

～参考～

- マルチパートメッセージ（MIMEMultipart）
  マルチパートメッセージは、異なる種類のメッセージで構成されるメッセージです。
  ここでは、テキストメッセージと添付ファイルを含むメールを扱うためにマルチパートメッセージを使用します。
- テキストメッセージ（MIMEText）
  テキストメッセージは、メールの本文を表現するためのメッセージです。
- ベースメッセージ（MIMEBase）
  ベースメッセージは、添付ファイルを表現するために使用するメッセージです。ベースメッセージは、特定のファイルタイプや MIME タイプに基づいて、添付ファイルを表現するためのメッセージを作成します。
  添付ファイルは、メールの本文とは異なる形式のデータを含むため、ベースメッセージが使用されます。

- 9 行目: MEIME メッセージのエンコードを行うためのライブラリをインポート
  添付ファイルの形式を「Base64」形式にエンコードするために使用します。

～環境変数から Gmail アカウント情報取得～

- 12 行目～17 行目: 環境変数から Gmail アカウントの情報を取得するプログラム get_gmail_settings()関数
- 13 行目: SMTP_SERVER: Gmail の送信メールサーバーSMTP のアドレス取得
- 14 行目: SMTP_PORT: 送信メールサーバーのポート番号取得
- 15 行目: EMAIL: 送信元メールアドレス取得
- 16 行目: APP_PASSWORD: Gmail のアプリパスワード取得
- 17 行目: 他の関数で使用する変数（smtp_server, smtp_port, email, app_password）を戻り値として返す

～メール作成～

- 20 行目～36 行目: メール作成プログラム create_email()関数
- 21 行目: マルチパートメッセージを作成
- 23 行目～26 行目: テキストメッセージ（MIMEText）定義
- 23 行目: 送信元メールアドレスを定義
- 24 行目: 送信先メールアドレス定義
- 25 行目: メール件名定義
- 26 行目: テキスト形式の本文をメールに添付
- 29 行目～35 行目: ベースメッセージ（MIMEBase）定義
- 29 行目: 添付ファイルの有無を判断
  添付ファイルがある場合は、「attachment_path」でファイルを指定
- 30 行目: 添付ファイルの MIME タイプ指定
- 31 行目～32 行目: 添付ファイルを読み込み
- 33 行目: 添付ファイルを Base64 形式にエンコード
- 34 行目: 添付ファイルのヘッダーを設定
- 35 行目: 添付ファイルをメールに添付
- 36 行目: 他の関数で使用する変数（msg）を戻り値として返す

Copyright © 2024 YITJC All Rights Reserved.

～メール送信～

- 39 行目～47 行目: メール送信プログラム send_email()関数
- 41 行目: get_gmail_settings()関数から Gmail アカウントの情報を取得
- 43 行目: SMTP サーバーに接続
- 44 行目: SMTP 通信を TLS（Transport Layer Security）で暗号化
- 45 行目: Gmail アカウントにログイン
- 46 行目: メール送信
- 47 行目: SMTP サーバーから切断

～メイン関数～

- 49 行目～58 行目: メイン関数（メール作成・メール送信）
- 50 行目: 送信元メールアドレス取得
- 51 行目: 送信先メールアドレスを設定
- 52 行目: メール件名を設定
- 53 行目: メール本文を設定
- 54 行目: 添付ファイルのパス（ディレクトリ+ファイル名）を設定
  ファイルを添付しない場合は、「attachment_path = None」と設定
- 56 行目: メール作成
- 58 行目: メール送信
- 60 行目～64 行目: main プログラム
- 61 行目～62 行目 :try ブロック
- 62 行目:main()関数実行
- 63 行目～64 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

9.2 通知機能～X（旧 Twitter）～

X（旧 Twitter）は、国内における月間アクティブユーザー（MAU）数が 6,000 万人を超える SNS（Social Networking Service）です。X 社が提供する API は無料で利用することができていましたが、2023 年 2 月に有料プランが発表されました。しかし、無料（Free）プランも用意されており、無料利用枠（1,500 ポスト/月）の範囲内であれば、無料で利用可能です（2024 年 5 月末時点）。

ここでは、Python プログラムを使用して X（旧 Twitter）でメッセージをポスト（ツイート）する方法について学習します。

事前準備～アカウント取得～

X アカウントでログインする必要があるため、スマートフォンなどを使用して事前に X アカウントを取得してください。取得したアカウントで X API の利用申請を行います。

API 利用申請を実施し、Python プログラムでメッセージを送信（ポスト）するために必要な情報（4 項目）を取得します。

- API key
- API secret key
- Access token
- Access token secret

手順①～API 利用申請～

X API の利用申請を行います。

次の URL をクリックし、X Account でログインしてください。

[URL] https://developer.x.com/ja

ログインすると、X のトップページが表示されます。

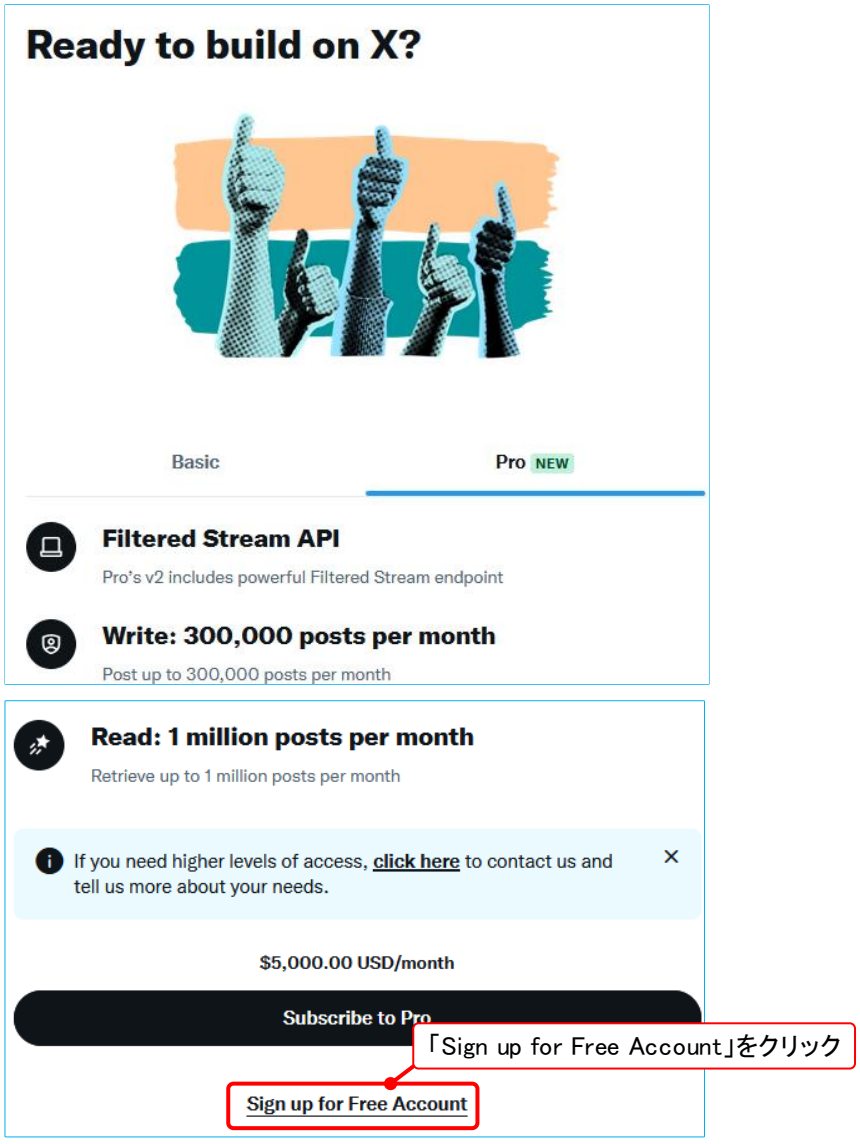
トップページからは API の利用申請ができないため、次のリンクをクリックしてください。

[URL] https://developer.twitter.com/ja

Copyright © 2024 YITJC All Rights Reserved.

次の画面が表示されたら、「開発者ポータル」をクリックしてください。

次の画面が表示されたら、「Sign up for Free Account」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

ここでは、X（旧 Twitter）のデータや API の利用目的を英語で記載しなければなりません。利用目的を明確に記載しないと申請が承認されないので注意してください。

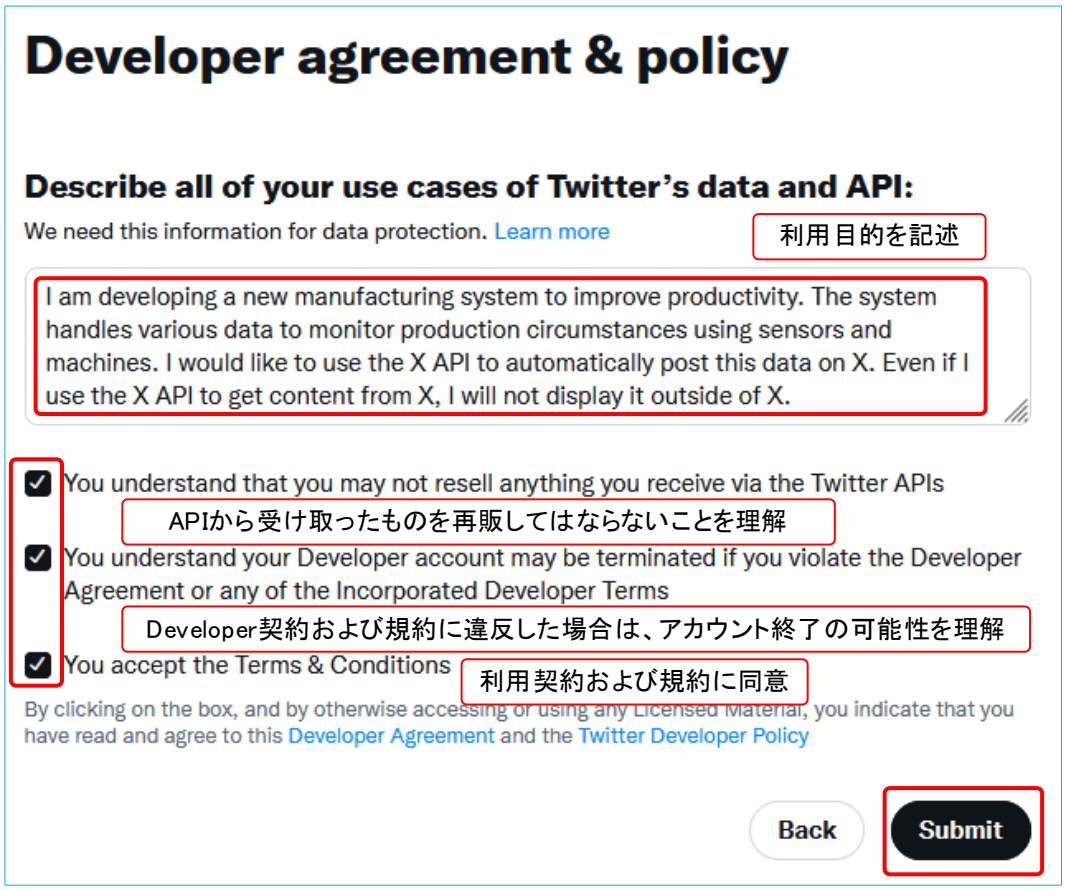
✓ 利用目的

記載例

I am developing a new manufacturing system to improve productivity. The system handles various data to monitor production circumstances using sensors and machines. I would like to use the X API to automatically post this data on X. Even if I use the X API to get content from X, I will not display it outside of X.

利用目的を英語で記述したあとは、チェックボックス（3 項目）の内容を確認してチェックしてください。

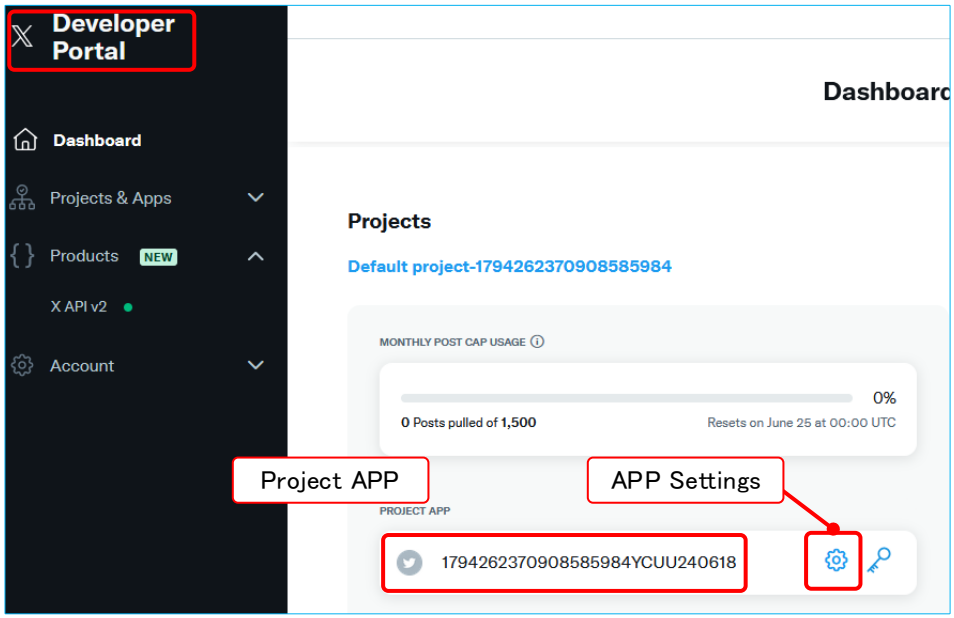
最後に、「Submit」をクリックしてください。



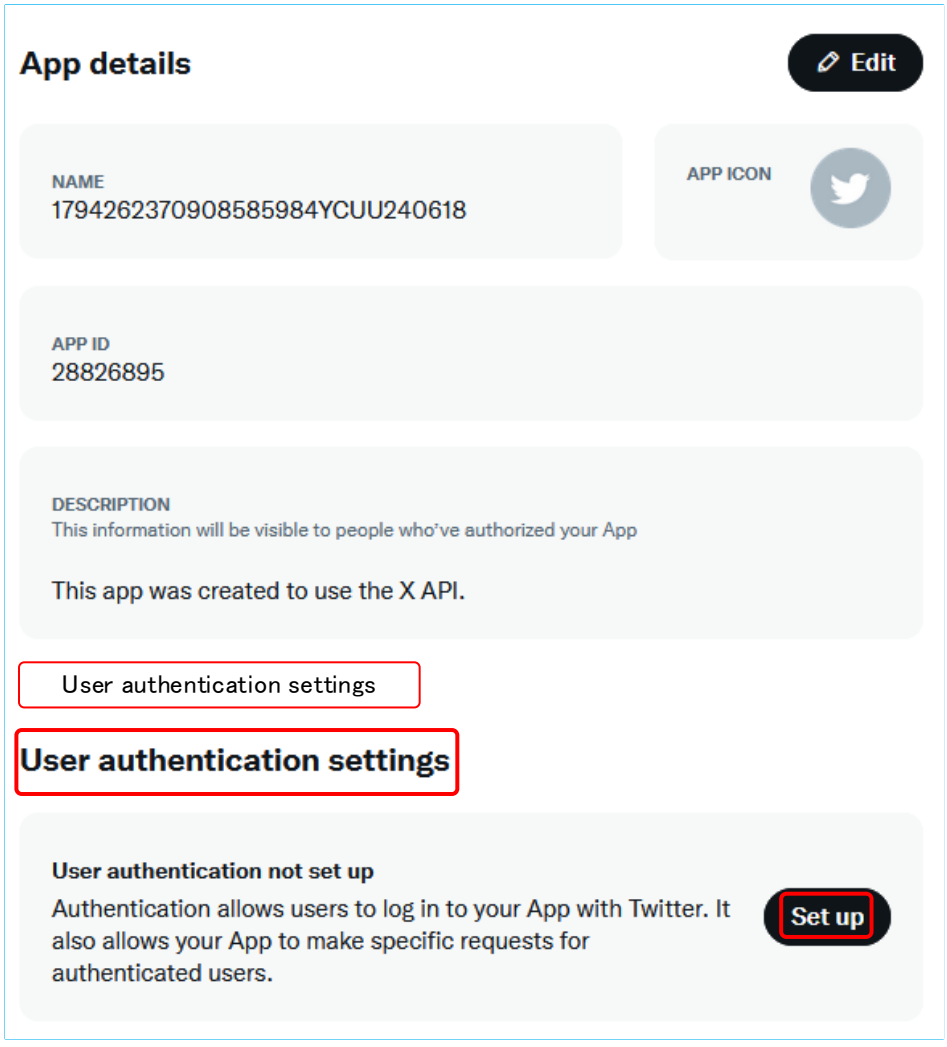
Copyright © 2024 YITJC All Rights Reserved.

「X Developer Portal」画面が開き、プロジェクトアプリ（Project APP）が作成されています。

「APP Settings」をクリックしてください。

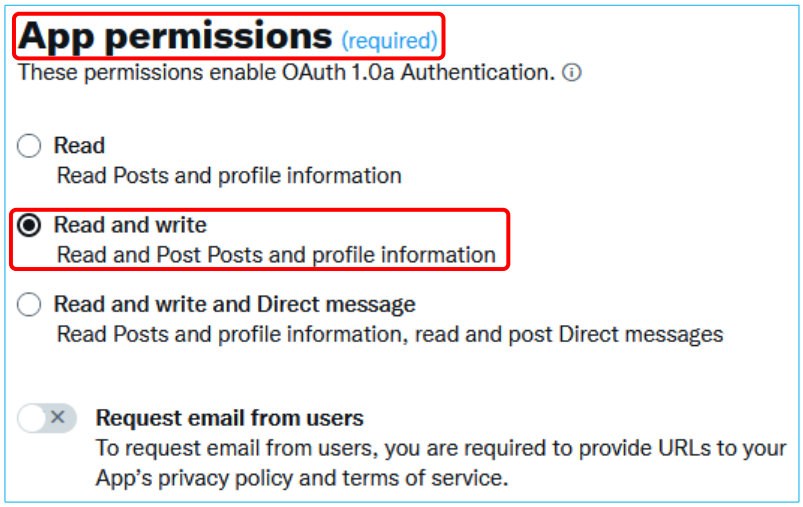


「User authentication settings」の「Set up」をクリックしてください。

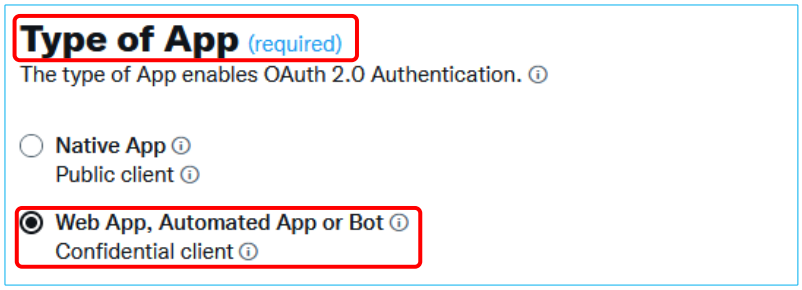


Copyright © 2024 YITJC All Rights Reserved.

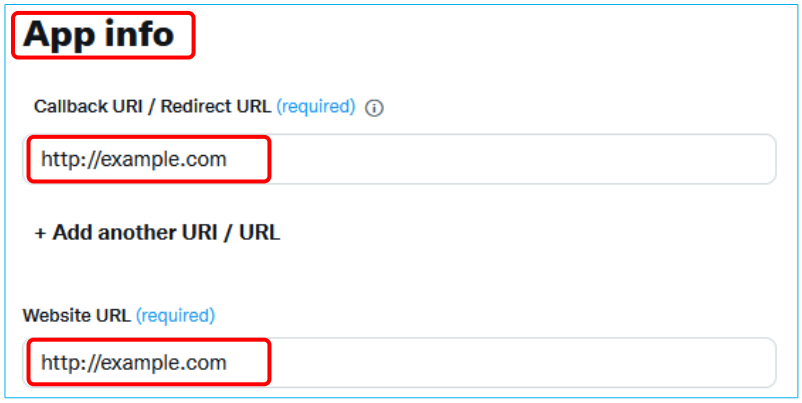
「Read and write」を選択してください。



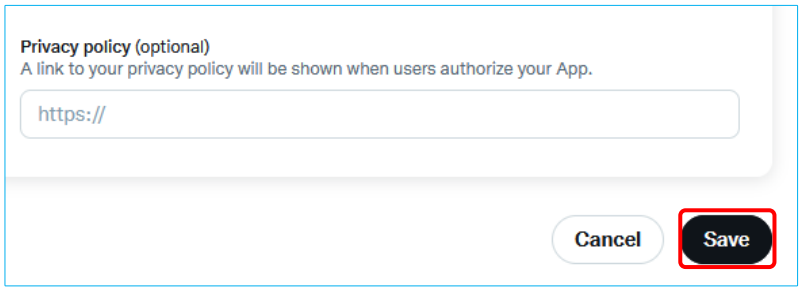
「Web APP, Automated APP or Bot」を選択してください。



「Callback URI」と「Website URL」には適当な URL（例：http://example.com）を指定してください。
指定した URL が実際の Web ページにリンクされる必要はありません。

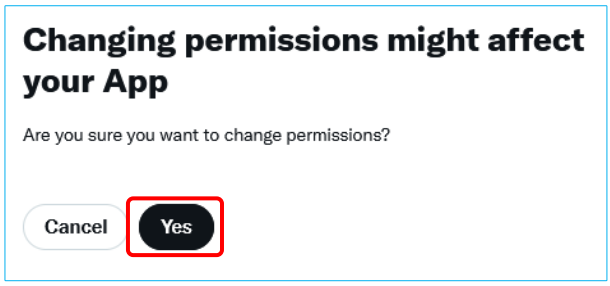


最後に「Save」をクリックしてください。



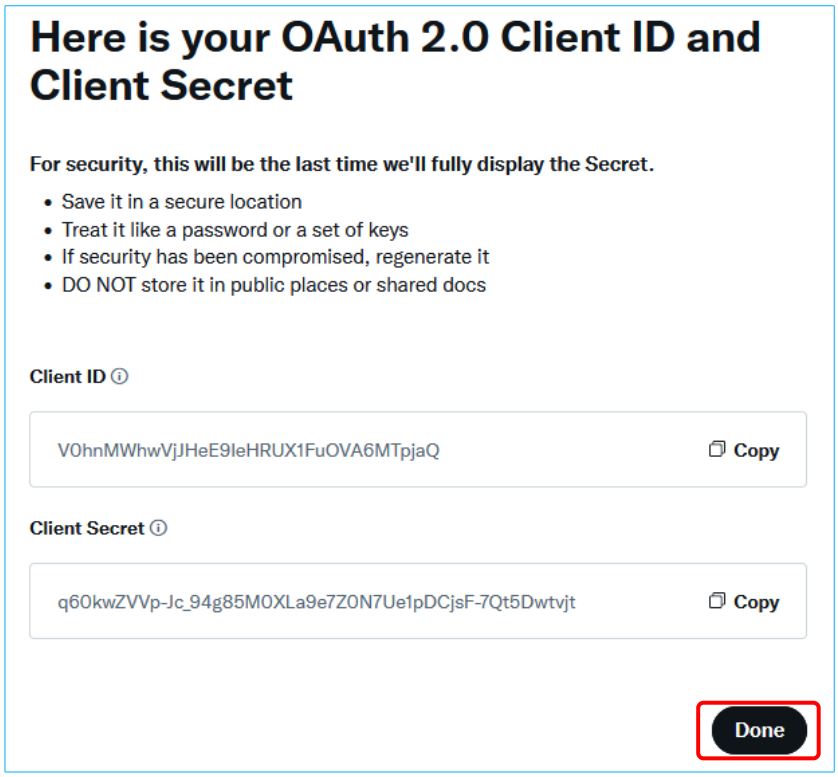
Copyright © 2024 YITJC All Rights Reserved.

「Yes」をクリックしてください。

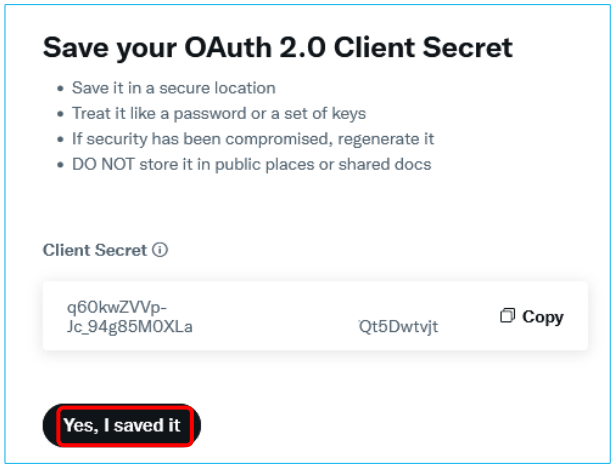


OAuth 認証情報が画面に表示されます。

「Client ID」と「Client Secret」の情報は Python プログラムに必要ないため、「Done」をクリックしてください。

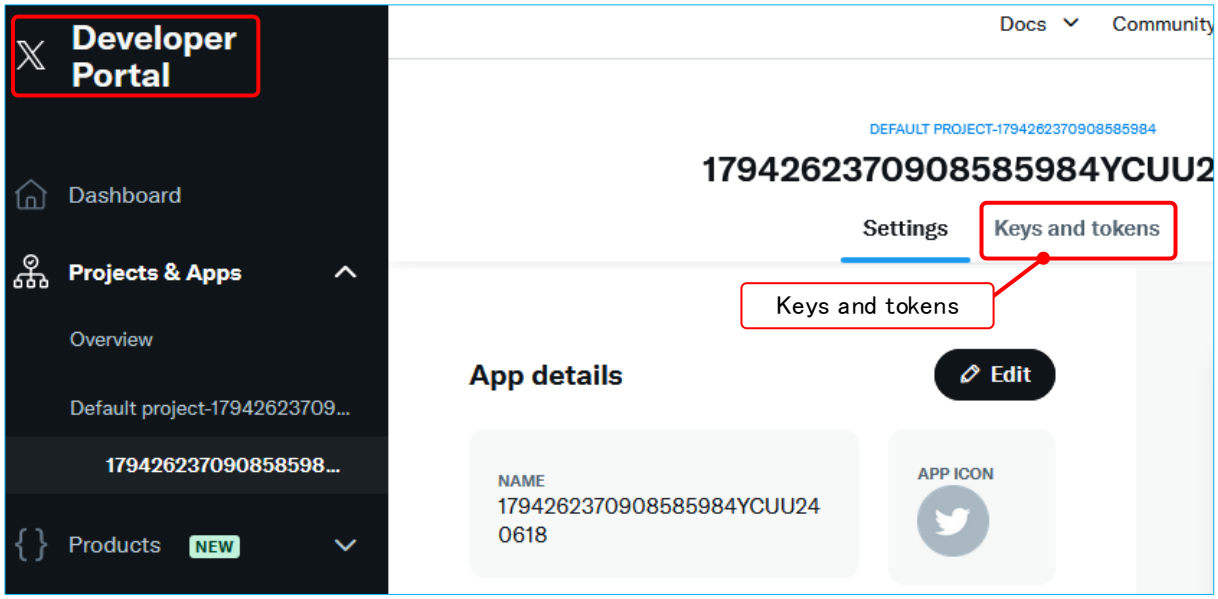


もう一度画面に「Client Secret」が表示されますが、「Yes, I saved it.」をクリックしてください。

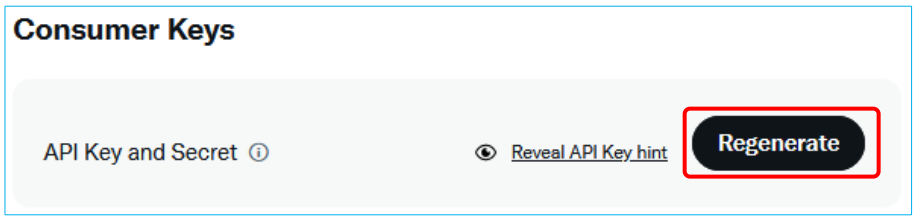


Copyright © 2024 YITJC All Rights Reserved.

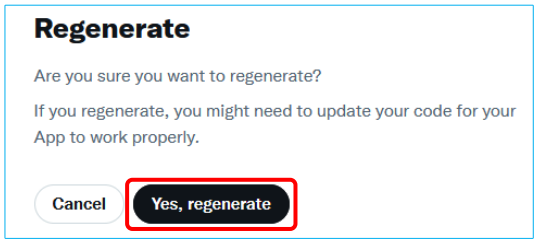
「Keys and tokens」をクリックしてください。



「Consumer Keys」の「Regenerate」をクリックし、「API Key and Secret」を発行します。



「Yes, regenerate」をクリックしてください。



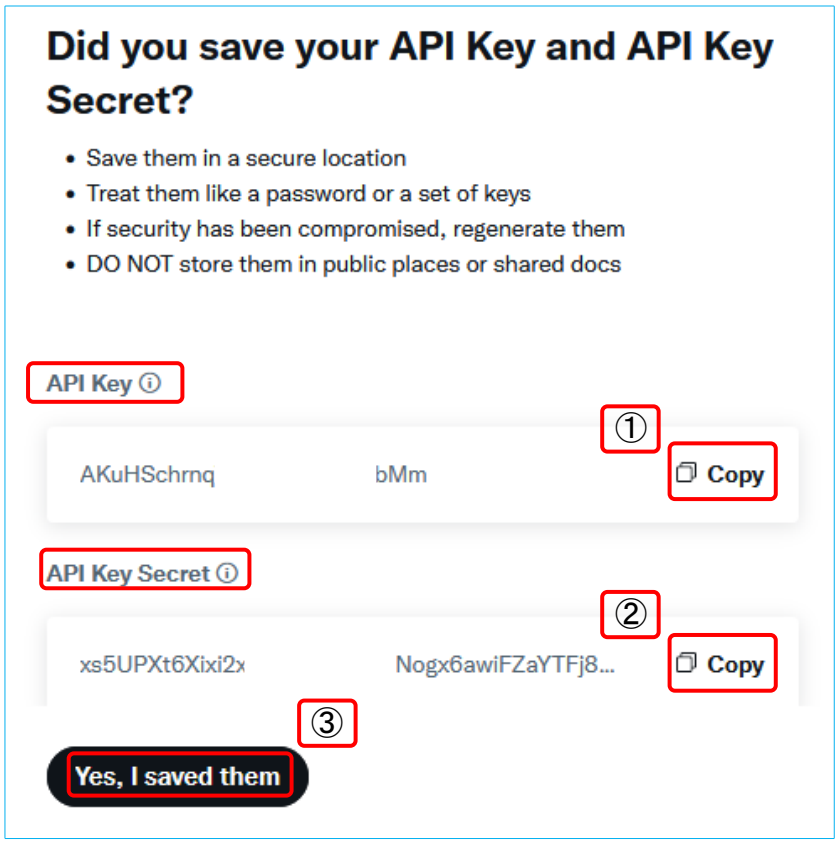
Copyright © 2024 YITJC All Rights Reserved.

「API Key」および「API Key Secret」が画面に表示されます。次の手順を実施してください。

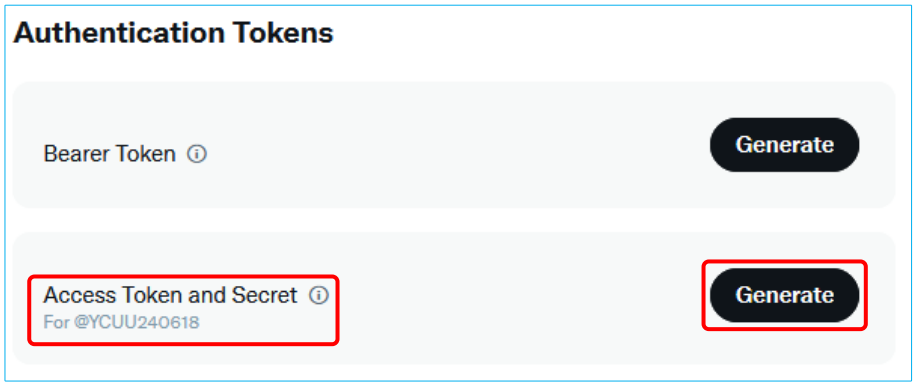
① API Key をコピーして、メモ帳などに記録してください。

② API Key Secret をコピーして、メモ帳などに記録してください。

③ 「Yes, I saved them.」をクリックして API Key および API Key Secret の作成を終了してください。



続いて、「Authentication Tokens」の「Generate」をクリックし、「Access Token and Secret」を発行します



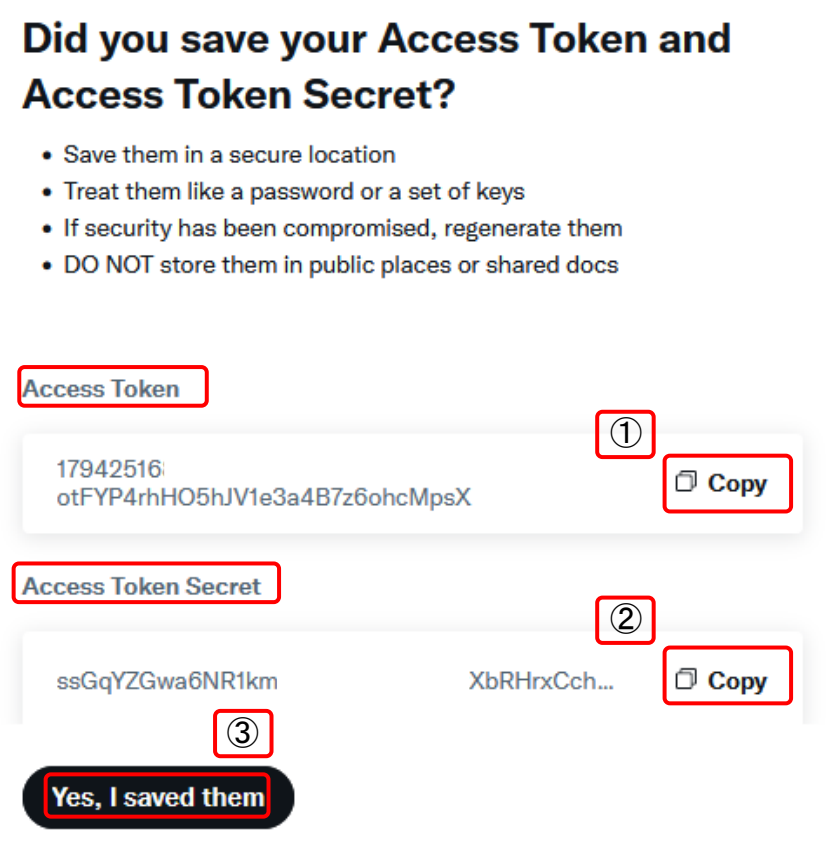
Copyright © 2024 YITJC All Rights Reserved.

「Access Token」および「Access Token Secret」が画面に表示されます。次の手順を実施してください。

① Access Token をコピーして、メモ帳などに記録してください。

② Access Token Secret をコピーして、メモ帳などに記録してください。

③ 「Yes, I saved them.」をクリックして API Key および API Key Secret の作成を終了してください。



Copyright © 2024 YITJC All Rights Reserved.

手順②～tweepy インストール～

Python プログラムを使用して X でポスト（メッセージ送信）するために必要なライブラリ「tweepy」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install tweepy

～補足～

tweepy と同時に依存関係のあるパッケージもインストールされます。oauthlib（OAuth2.0 認証用）や requests-oauthlib（OAuth2.0 認証を requests ライブラリと統合）がインストールされます。

手順③～環境変数設定～

X API を利用するために必要な情報（手順①で取得した情報）を環境変数に設定します。

環境変数に登録する情報は、次の情報です。

```
export API_KEY='AKuHSxxxxxxxxxx04CrdJGbMm'
export API_SECRET='xs5UPXt6Xixi2xxxxxxxxxxx4Nogx6awiFZaYTFj8UBc7XwOqo'
export ACCESS_TOKEN='1794251681510xxxxxxxxxxYP4rhHO5hJV1e3a4B7z6ohcMpsX'
export ACCESS_TOKEN_SECRET='ssGqYZGwa6NR1kmHhjsJxxxxxxxxxxXbRHrxCch3vTPhM'
```

Raspberry Pi が起動したときに環境変数を自動的に設定するため、「.bashrc」ファイルに環境変数の設定コマンドを記述します。

ホームディレクトリに移動し、ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

(ycuu) cuu@raspberrypi:~/dev $ cd ..

※「cd ..」コマンドを実行すると、1 つ上のディレクトリに移動できます。

(ycuu) cuu@raspberrypi:$ nano .bashrc

Copyright © 2024 YITJC All Rights Reserved.

「.bashrc」ファイルの画面が起動したら、末尾に「export」コマンドで変数を 4 つ追加（記述）します。

記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

手順④～環境変数の反映～

続いて、次のコマンドを実行して設定した環境変数を反映します。

(ycuu) cuu@raspberrypi:$ source ~/.bashrc

「~」の部分（チルダ）は、「Shift」+「～」で入力します。

手順⑤～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

39_Xpost.py

手順⑥～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 39_Xpost.py

手順⑦～プログラム実行～

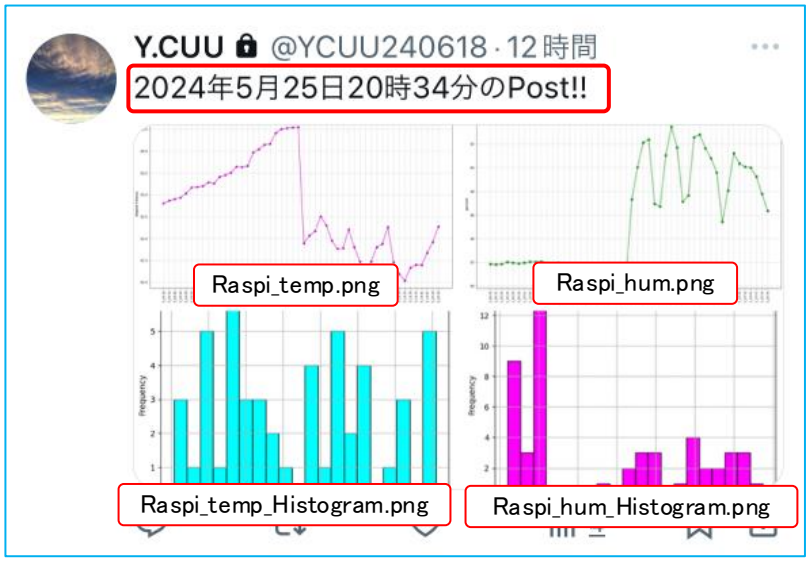
対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./39_Xpost.py

ポスト時刻が表示されます。スマートフォンなどでメッセージが送信（ポスト）されていることを確認してください。

Copyright © 2024 YITJC All Rights Reserved.

SNS アプリ「X」の画面には、メッセージ（ポスト時刻）とともに、BME280 センサの温度、湿度のグラフが添付されていることが確認できます。



参考～非公開設定～

スマートフォンで「X」を開いてください。

「プロフィール画像」-「設定とサポート」-「設定とプライバシー」-「プライバシーと安全」「設定とプライバシー」から「オーディエンスとタグ付け」を開いてください。次の画面が表示されます。

「ポストを非公開にする」にチェックを入れると、ポストを非公開（鍵垢）に設定できます。

「非公開」に設定するとで、フォローリクエストを承認したアカウントしかポストを見ることができません。これによりポスト（メッセージ）の公開範囲を限定できるため、不特定多数のユーザーにポストを見られる心配がありません。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import tweepy
5	import os
6	import datetime
7	
8	# ディレクトリとファイル名の定義
9	URL_PATH = "/home/cuu/data/"
10	file_names = [
11	"Raspi_temp.png",
12	"Raspi_hum.png",
13	"Raspi_temp_Histogram.png",
14	"Raspi_hum_Histogram.png"
15	]
16	
17	#X APIの情報を環境編集から取得
18	def get_xapi_settings():
19	API_KEY = os.getenv('API_KEY')
20	API_SECRET = os.getenv('API_SECRET')
21	ACCESS_TOKEN = os.getenv('ACCESS_TOKEN')
22	ACCESS_TOKEN_SECRET = os.getenv('ACCESS_TOKEN_SECRET')
23	return API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET
24	
25	#X OAuth認証
26	def x_oauth():
27	API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET = get_xapi_settings()
28	auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
29	auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
30	api = tweepy.API(auth)
31	client = tweepy.Client(
32	consumer_key=API_KEY,
33	consumer_secret=API_SECRET,
34	access_token=ACCESS_TOKEN,
35	access_token_secret=ACCESS_TOKEN_SECRET)
36	return api, client
37	

Copyright © 2024 YITJC All Rights Reserved.

```
38  #メッセージpost
39  def post_message():
40      # OAuth認証を行い、apiとclientを取得
41      api, client = x_oauth()
42      #日付・時刻取得
43      now = datetime.datetime.today()
44      message = str(now.year) + u"年" + str(now.month) + u"月" + str(now.day) + u"日" + str(now.hour) + u"時" +
    str(now.minute) + u"分のPost!!"
45      print(message)
46      #画像ファイル添付
47      media_ids = []  # media_idsリストを定義
48      for filename in file_names:
49          res = api.media_upload(URL_PATH + filename)
50          media_ids.append(res.media_id)
51      #メッセージポスト
52      client.create_tweet(text=message, media_ids=media_ids)
53
54  if __name__ == '__main__':
55      try:
56          post_message()
57      except KeyboardInterrupt:
58          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Python プログラムを使用して X でポストするためのライブラリ「tweepy」をインポート
- 5 行目: OS と対話するためのライブラリ「os」をインポート
- 6 行目: 日付・時刻を取得するため datetime ライブラリをインポート
- 9 行目: ポストに添付する画像ファイルが保存されているディレクトリを指定
- 10 行目～15 行目: ポストに添付する画像ファイルを指定

～環境変数から X API の情報取得～

- 18 行目～23 行目: 環境変数から X API の情報を取得するプログラム get_xapi_settings()関数
- 19 行目: API_KEY 取得
- 20 行目: API_SECRET 取得
- 21 行目: ACCESS_TOKEN 取得
- 22 行目: ACCESS_TOKEN_SECRET 取得
- 23 行目: 他の関数で使用する変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を戻り値として返す

Copyright © 2024 YITJC All Rights Reserved.

～OAuth 認証～

- 26 行目～36 行目: OAuth 認証を行うプログラム x_oauth()関数
- 27 行目: OAuth 認証に必要な変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を get_xapi_settings()関数から取得
- 28 行目: tweepy.OAuthHandler クラスのインスタンスを作成
  API_KEY と API_SECRET を使用して認証ハンドラーを初期化します。
  認証ハンドラーを使用して、このあとアクセストークンの設定や tweepi.API クラスのインスタンスを作成します。
- 29 行目: OAuth 認証に必要なアクセストークンとアクセストークンシークレットを設定
- 30 行目: 認証ハンドラー（auth）を使用して tweepi.API クラスのインスタンスを作成
- 31 行目～35 行目: tweepi.Client クラスのインスタンスを作成
  インスタンスの作成には、API 利用申請で取得した変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を使用します。
- 36 行目: 他の関数で使用するため api オブジェクトと client オブジェクトを戻り値として返す

～メッセージ post～

- 39 行目～52 行目: メッセージと画像をポストするプログラム post_message()関数
- 41 行目: api オブジェクトと client オブジェクトを x_oauth()関数から取得
- 43 行目: 現在の日付・時刻取得
- 44 行目: ポストするメッセージを定義
- 45 行目: ポストするメッセージを print 表示
- 47 行目: ポストに添付する画像ファイルのメディア ID を格納するためのリスト「media_ids」を定義
- 48 行目～50 行目: ポストに添付する画像のメディア ID をリストに格納
- 49 行目: api.media_upload()メソッドを使用して添付した画像ファイルを X（Twitter）にアップロードし、アップロードした画像ファイルのメディア情報をオブジェクト res に格納
- 50 行目: アップロードされた画像のメディア ID を「media_ids」リストに追加
- 52 行目: メッセージと画像をポスト
- 54 行目～58 行目: main プログラム
- 55 行目～56 行目 :try ブロック
- 56 行目: メッセージと画像をポストするプログラム post_message()関数実行
- 57 行目～58 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

Copyright © 2024 YITJC All Rights Reserved.

9.3 定期的な通知機能をもつ IoT システム（ver3）

「8 章オンラインストレージ活用」で構築した IoT システム ver2 は、BME280（温度、湿度、気圧）センサのデータを収集・加工（グラフ化）し、定期的にオンラインストレージへアップロードしています。

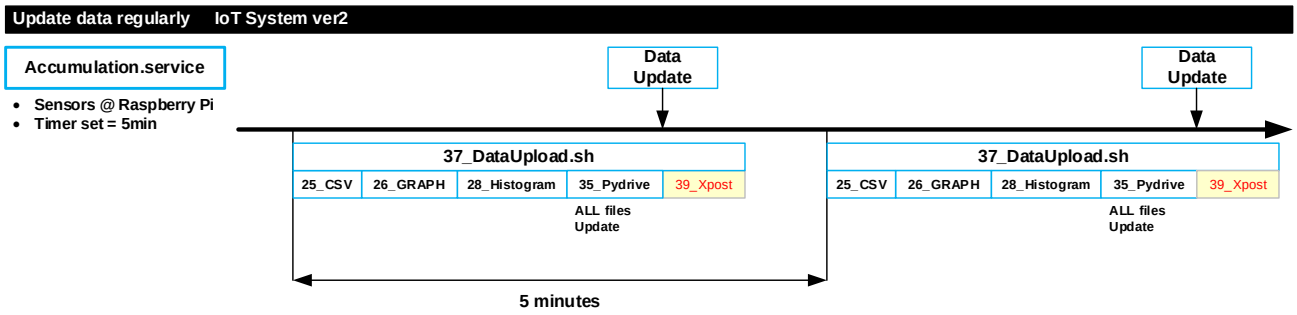
ここでは、IoT システム ver2 に定期的なポスト機能を追加して ver3 を構築します。

～ver3 システム構成検討～

定期的なポスト機能を追加する前に「IoT システム ver2」の構成を復習します。

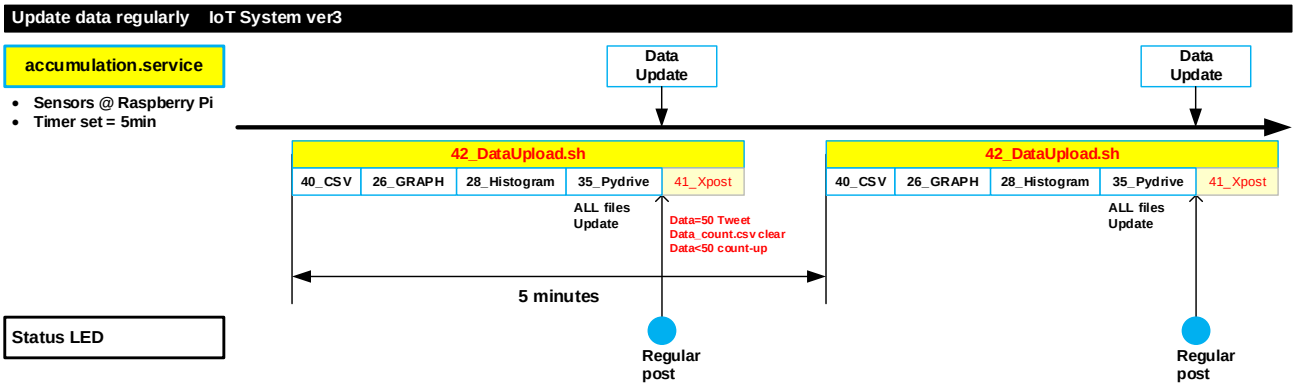
IoT システム ver2 は、センサデータ記録から Google Drive へのアップロードまでのプログラムをシェルスクリプト「37_DataUpload.sh」にまとめ、5 分ごとに実行しています。

例えば、「37_DataUpload.sh」の最後にポスト機能「39_Xpost.py」を追加した場合、システムの構築は簡単ですが、5 分ごとにポストが届くため通知を煩わしく感じます。



IoT システム ver3 は、csv ファイルに 50 個のデータが蓄積されたタイミングでポストする構成にします。

これにより、250 分周期（データ記録 5 分周期×50 個のデータ蓄積）のポストになるため、1 日当たりのポストは約 6 回になります。この頻度であれば、X API の無料利用枠（1,500 ポスト/月）の範囲内に収まります。



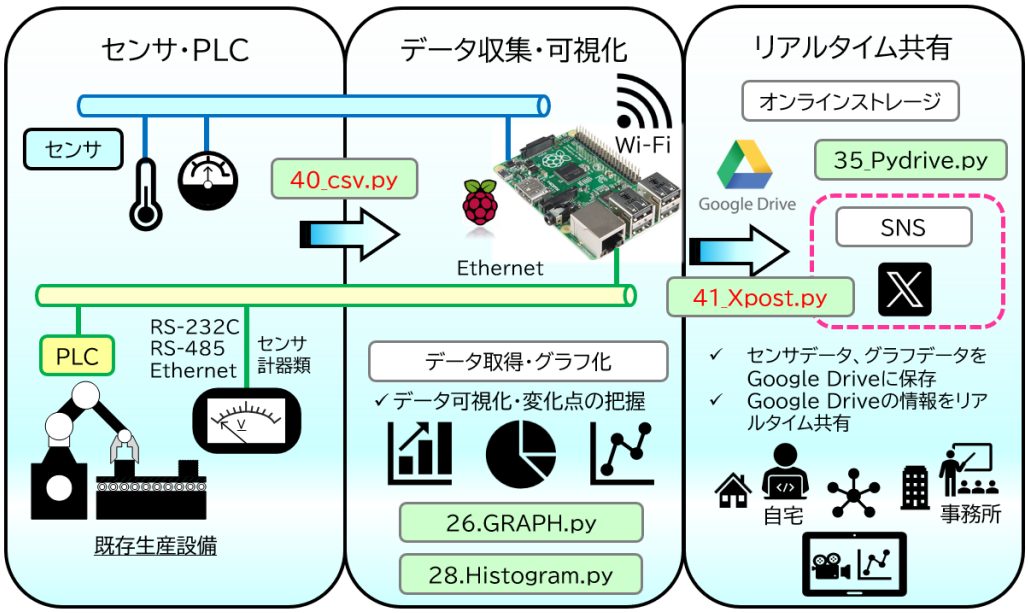
IoT システム ver3 を構築するため、これまで使用してきたプログラムに次の変更を加えます。

① データ収集プログラム「25_CSV.py」を編集して「40_CSV.py」を作成
- データ数カウント用ファイルを新たに追加（Data_count.csv）

② ポストプログラム「39_Xpost.py」を編集して「41_Xpost.py」を作成
- データ更新時にデータ数をカウント
- データが 50 個蓄積されたら…次の処理を実施する。
  - ✓ ポスト
  - ✓ ポスト時に青色 LED （GPIO26）を点灯
  - ✓ データ数カウント用ファイルをクリア（データカウントを 0 にリセット）

Copyright © 2024 YITJC All Rights Reserved.

IoT システム ver3 の構成を示します。



～シェルスクリプト構成～

IoT システム ver3 は、シェルスクリプト「42_DataUpload.sh」に次のプログラムを組み合わせて構築します。

赤字部は、変更・追加する項目です。

使用するプログラム	処理内容
40_CSV.py	センサデータを csv ファイルに蓄積 ✓ Raspi_temp.csv ✓ Raspi_hum.csv ✓ Raspi_pressure.csv データ数カウント用 csv ファイルを作成 ✓ Data_count.csv
26_GRAPH.py	CSV ファイルを読み込み、時系列グラフ作成
28_Histogram.py	CSV ファイルを読み込み、ヒストグラム作成
35_Pydrive.py	「/home/cuu/data」ディレクトリに保存されているデータをすべて Google Drive にアップデート（上書き）
41_Xpost.py	✓ データ更新時にデータ数をカウント ✓ データが 50 個蓄積されたら…次の処理を実施する。 ①ポストするデータを分析 ②ポスト ③ポストしたときに青色 LED（GPIO26）を点灯 ④データ数カウント用ファイルをクリア （データカウントを 0 にリセット）

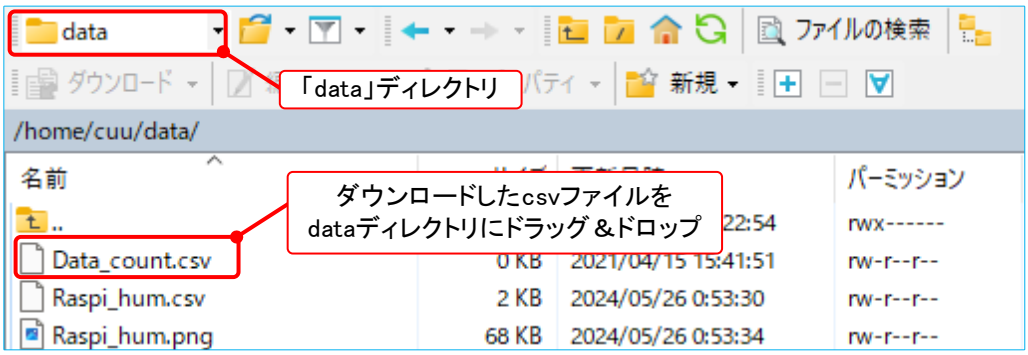
Copyright © 2024 YITJC All Rights Reserved.

手順①～csv ファイル保存～

共有サイトから csv ファイルを Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「data」ディレクトリに保存してください。

共有サイトからダウンロードする csv ファイルは、データカウント用の csv ファイルです。

✓ Data_count.csv



手順②～CSV プログラム作成～

「25_CSV.py」を編集して「40_CSV.py」を作成します。

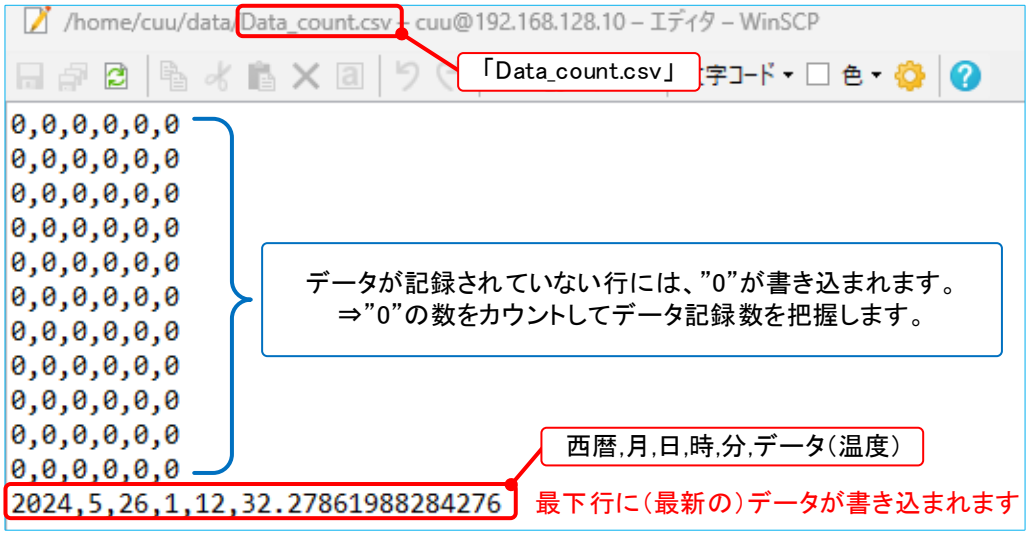
編集内容は、データ数カウント用 csv ファイル「Data_count.csv」にデータを記録するコードを追加することです。「Data_count.csv」ファイルは、記録されているデータ数をカウントすることが目的であるため、何を記録しても問題ありませんが、ここでは「温度データ」を記録しています。

[対象プログラム]

40_CSV.py

[プログラム内容]

「Data_count.csv」ファイルには「Raspi_temp.csv」に書き込まれる温度データと同じデータが書き込まれます。



データは最下行に書き込まれ、データが記録されていない行には「0」が書き込まれています。そのため、「0」の数をカウントすることで、記録されているデータ数を確認できます。

Copyright © 2024 YITJC All Rights Reserved.

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

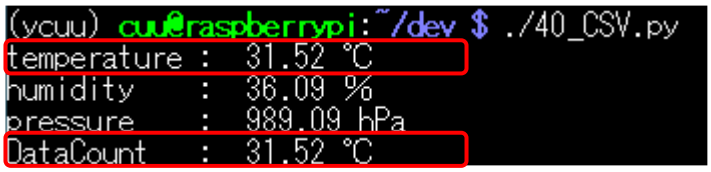
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 40_CSV.py

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./40_CSV.py

Data_count.csv ファイルに Raspi_temp と同じ温度データが書き込まれていることを確認してください。



～補足～

「40_CSV.py」のプログラムは、「25_CSV.py」の内容とほぼ同じであるため、プログラムの解説は省略します。

手順⑤～Post プログラム作成～

[対象プログラム]

41_Xpost.py

新たに作成するポストプログラム「41_Xpost.py」は、「39_Xpost.py」に対して次の処理を追加します。

- ポストするデータ（Raspi_temp.csv）を分析し、最高温度、データ記録時間を抽出
  6 章「データ収集と活用」の復習（29_csvData.py）
- 現在のデータ記録数を把握（Data_count.csv ファイルの「0」をカウント）
- データが 50 個蓄積されたタイミングでポスト
- ポストした時に青色 LED （GPIO26）を点灯
- ポストした時にデータ数カウント用ファイル「Data_count.csv」を初期化（データクリア）

➔ これにより、データ数のカウント値をリセットします。

手順⑥～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 41_Xpost.py

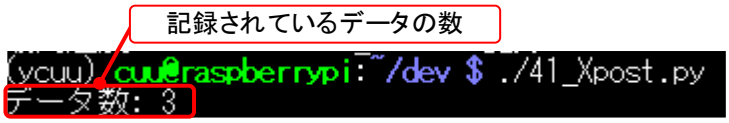
手順⑦～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./41_Xpost.py

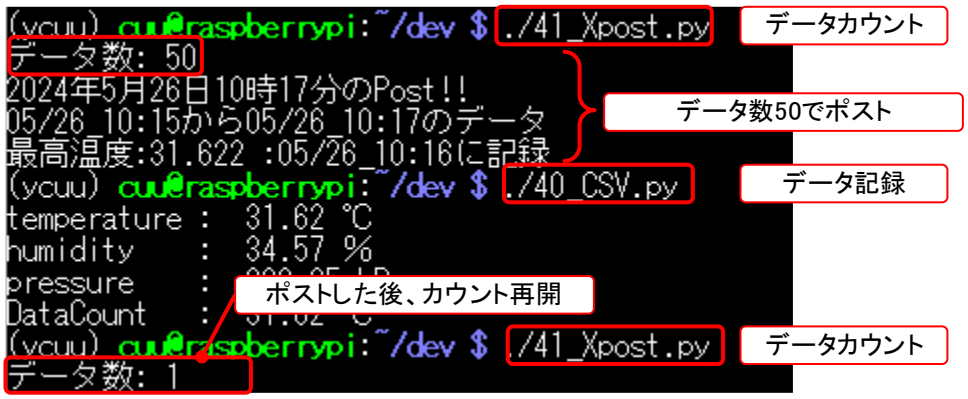
Copyright © 2024 YITJC All Rights Reserved.

プログラムを実行すると、次の画面が表示されます。

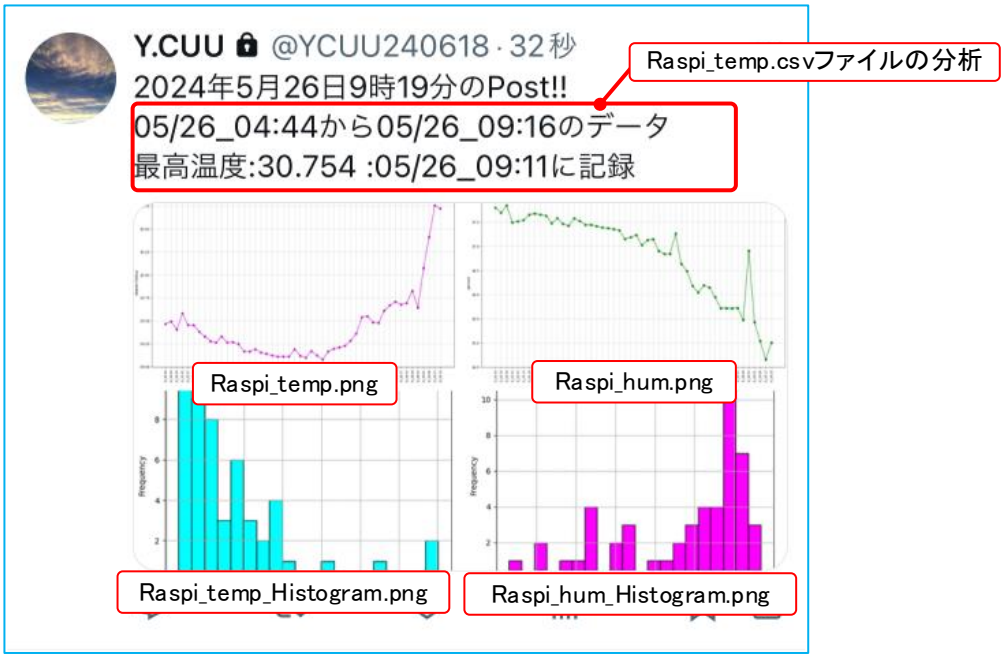


データ記録プログラム「40_CSV.py」を実行するとデータが記録されるため、カウントアップを確認できます。

データ数が 50 個になると、ポストを実行し、「Data_count.csv」ファイルを初期化（データクリア）します。
データ記録⇒カウント再開⇒50 個でポスト＆データクリアが繰り返し行われます。



X の画面を確認すると、「Raspi_temp.csv」ファイルの分析結果が送信されていることがわかります。



Copyright © 2024 YITJC All Rights Reserved.

～Post プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import tweepy
5	import os
6	import time
7	import datetime
8	import csv
9	import pandas as pd
10	import RPi.GPIO as GPIO
11	
12	#csvファイルデータ数定義
13	MAX_RANGE=50
14	# ディレクトリとファイル名の定義
15	URL_PATH = "/home/cuu/data/"
16	file_names = [
17	"Raspi_temp.png",
18	"Raspi_hum.png",
19	"Raspi_temp_Histogram.png",
20	"Raspi_hum_Histogram.png"
21	]
22	
23	#GPIO端子初期設定関数
24	def setup_led():
25	GPIO.setmode(GPIO.BCM)
26	GPIO.setup(26, GPIO.OUT)
27	GPIO.output(26, GPIO.LOW)
28	
29	#データ記録数カウント用csvファイル読み込み
30	def count_data():
31	#global変数定義: csv_clear()内でdf_cを使用
32	global df_c
33	df_c = pd.read_csv(URL_PATH+'Data_count.csv', names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
34	
35	#データ数カウント
36	#カウント用csvファイル'year'列 "0"の数をカウント
37	#'year'=0判定 (0:True, 1:False表示)

Copyright © 2024 YITJC All Rights Reserved.

```
38        y_bool = df_c['year'] == 0
39        global y_sum
40        y_sum = y_bool.sum()
41        count = MAX_RANGE-y_sum
42        print ('データ数:',count)
43
44    #Post用メッセージ作成
45    def create_message():
46        #日付・時刻取得
47        now = datetime.datetime.today()
48        #データ抽出対象となるデータファイル読み込み
49        df_t = pd.read_csv(URL_PATH+'Raspi_temp.csv', names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
50        #温度データ最大値取得
51        tmax=df_t['temperature'].max()
52        #データ記録開始時間の取得
53        start_time="%02d"%int(df_t.loc[0, 'month'])+"/"+"%02d"%int(df_t.loc[0, 'day'])+"_"+¥
54        "%02d"%int(df_t.loc[0, 'hour'])+":"+"%02d"%int(df_t.loc[0, 'minute'])
55        #データ記録終了時間の取得
56        end_time="%02d"%int(df_t.loc[49, 'month'])+"/"+"%02d"%int(df_t.loc[49, 'day'])+"_"+¥
57        "%02d"%int(df_t.loc[49, 'hour'])+":"+"%02d"%int(df_t.loc[49, 'minute'])
58        #最高値を記録したデータ(行)抽出
59        tmax_row=df_t[df_t['temperature'] == tmax]
60        #最高値を記録したデータ(行)から日付・時刻抽出
61        mday = "%02d"%int(tmax_row.iloc[0]['month']) + "/" + "%02d"%int(tmax_row.iloc[0]['day']) + "_" + ¥
62        "%02d"%int(tmax_row.iloc[0]['hour']) + ":" + "%02d"%int(tmax_row.iloc[0]['minute'])
63        #Post用メッセージ作成
64        global message
65        text1 = str(now.year) + u"年" + str(now.month) + u"月" + str(now.day) + u"日" + str(now.hour) + u"時" +
      str(now.minute) + u"分のPost!!"
66        text2 = start_time + 'から' + end_time + 'のデータ' + '¥n最高温度:' + '{:.3f}'.format(tmax) + ' :' + mday + 'に記録'
67        message = text1 + '¥n'+ text2
68
69    #csvデータクリア
70    def csv_clear():
71        #データ削除
72        df = df_c.drop(range(0,50))
73        df.to_csv(URL_PATH+'Data_count.csv', header=False, index=False)
74
75    #X APIの情報を環境編集から取得
```

Copyright © 2024 YITJC All Rights Reserved.

```
76  def get_xapi_settings():
77      API_KEY = os.getenv('API_KEY')
78      API_SECRET = os.getenv('API_SECRET')
79      ACCESS_TOKEN = os.getenv('ACCESS_TOKEN')
80      ACCESS_TOKEN_SECRET = os.getenv('ACCESS_TOKEN_SECRET')
81      return API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET
82
83  #X OAuth認証
84  def x_oauth():
85      API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET = get_xapi_settings()
86      auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
87      auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
88      api = tweepy.API(auth)
89      client = tweepy.Client(
90          consumer_key=API_KEY,
91          consumer_secret=API_SECRET,
92          access_token=ACCESS_TOKEN,
93          access_token_secret=ACCESS_TOKEN_SECRET)
94      return api, client
95
96  #メッセージpost
97  def post_message():
98      # OAuth認証を行い、apiとclientを取得
99      api, client = x_oauth()
100     #画像ファイル添付
101     media_ids = []   # media_idsリストを定義
102     for filename in file_names:
103         res = api.media_upload(URL_PATH + filename)
104         media_ids.append(res.media_id)
105     #メッセージポスト
106     client.create_tweet(text=message, media_ids=media_ids)
107     print(message)
108
109 if __name__ == '__main__':
110     try:
111         setup_led()
112         count_data()
113         create_message()
114         #カウント用csvファイルの"0"が無くなる=50個データが蓄積されたと判定
```

Copyright © 2024 YITJC All Rights Reserved.

```
115             if(y_sum == 0):
116                 post_message()
117                 GPIO.output(26, GPIO.HIGH)
118                 time.sleep(2.0)
119                 GPIO.output(26, GPIO.LOW)
120                 csv_clear()
121         except KeyboardInterrupt:
122             pass
123         finally:
124             GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Python プログラムを使用して X でポストするためのライブラリ「tweepy」をインポート
- 5 行目: OS と対話するためのライブラリ「os」をインポート
- 6 行目: 日付・時刻を取得するため datetime ライブラリをインポート
- 7 行目: csv ファイル読み書き用 csv ライブラリをインポート
- 8 行目: sleep 関数を使用するため time ライブラリをインポート
- 9 行目: データ分析を行うため pandas ライブラリをインポート
- 10 行目: GPIO 制御を行うため RPi.GPIO ライブラリをインポート
- 13 行目: csv ファイルに保存されているデータ数を設定
- 15 行目: ポストに添付する画像ファイルが保存されているディレクトリを指定
- 16 行目～21 行目: ポストに添付する画像ファイルを指定
- 24 行目～27 行目: GPIO 端子初期設定プログラム setup_led()関数
  青色 LED（GPIO26）を制御するため、GPIO 端子の初期設定を行う。

～csv ファイルに記録されているデータ数確認～

- 30 行目～42 行目: データ記録数カウント用プログラム count_data()関数
  他の関数で変数 df_c を使用するため、グローバル変数で定義しています。
- 32 行目: csv ファイルのデータを Pandas のデータフレーム df_c で定義
- 33 行目: csv データ読み取り
  Header に名前を付けるため、names で定義
  names=['year', 'month', 'day', 'hour', 'minute', 'temperature']
  ここで設定した名前を使用して、データの抽出や連結を行います。
- 38 行目～40 行目: データ数カウントファイルの”year”列にある「0」の数をカウント
- 41 行目: 現在のデータ記録数を算出（csv ファイルデータ格納数「50」から「0」カウント値を引き算）
- 42 行目: 現在のデータ記録数を print 表示

Copyright © 2024 YITJC All Rights Reserved.

～post 用メッセージ作成～

- 45 行目～67 行目: post 用メッセージを作成するプログラム create_message()関数
- 47 行目: 現在の日付・時刻取得
- 49 行目: csv ファイルのデータを Pandas のデータフレーム df_t で定義
  ここでは、メッセージ作成対象のデータ「Raspi_temp.csv」を使用しています。
- 51 行目: 温度データの最大値（tmax）を取得
- 53 行目～54 行目: データ記録開始時刻抽出
  csv ファイルの 1 行目の時刻データを抽出
- 56 行目～57 行目: データ記録終了時刻抽出
  csv ファイルの 50 行目の時刻データを抽出
- 59 行目: temperature データが最高値（max）を記録した行のデータを抽出
- 61 行目～62 行目: temperature データが最高値を記録したデータ（行）から日付・時刻抽出
- 64～67 行目: post 用メッセージ作成

～csv ファイルデータクリア～

- 70 行目～73 行目: csv ファイルのデータをクリアするプログラム csv_clear()関数
- 72 行目: データフレーム df_c の先頭 50 行を削除してデータフレーム df に格納
- 73 行目: データフレーム df のデータを csv ファイルに保存
  つまり、この 2 行で csv ファイルに記録されていたデータをクリアしています。
  「header=False」は、CSV ファイルにヘッダー行を含めないことを意味しています。
  「index=False」は、CSV ファイルに行番号を含めないことを意味しています。

～環境変数から X API の情報取得～

- 76 行目～81 行目: 環境変数から X API の情報を取得するプログラム get_xapi_settings()関数
- 77 行目: API_KEY 取得
- 78 行目: API_SECRET 取得
- 79 行目: ACCESS_TOKEN 取得
- 80 行目: ACCESS_TOKEN_SECRET 取得
- 81 行目: 他の関数で使用する変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を戻り値として返す

～OAuth 認証～

- 84 行目～94 行目: OAuth 認証を行うプログラム x_oauth()関数
- 85 行目: OAuth 認証に必要な変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を get_xapi_settings()関数から取得
- 86 行目: tweepy.OAuthHandler クラスのインスタンスを作成
  API_KEY と API_SECRET を使用して認証ハンドラーを初期化します。
  認証ハンドラーを使用して、このあとアクセストークンの設定や tweepi.API クラスのインスタンスを作成します。

Copyright © 2024 YITJC All Rights Reserved.

- 87 行目: OAuth 認証に必要なアクセストークンとアクセストークンシークレットを設定
- 88 行目: 認証ハンドラー（auth）を使用して tweepi.API クラスのインスタンスを作成
- 89 行目～93 行目: tweepi.Client クラスのインスタンスを作成
  インスタンスの作成には、API 利用申請で取得した変数（API_KEY, API_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET）を使用します。
- 94 行目: 他の関数で使用するため api オブジェクトと client オブジェクトを戻り値として返す

～メッセージ post～

- 97 行目～107 行目: メッセージと画像をポストするプログラム post_message()関数
- 99 行目: api オブジェクトと client オブジェクトを x_oauth()関数から取得
- 101 行目: ポストに添付する画像ファイルのメディア ID を格納するためのリスト「media_ids」を定義
- 102 行目～104 行目: ポストに添付する画像のメディア ID をリストに格納
- 103 行目: api.media_upload()メソッドを使用して添付した画像ファイルを X（Twitter）にアップロードし、アップロードした画像ファイルのメディア情報をオブジェクト res に格納
- 104 行目: アップロードされた画像のメディア ID を「media_ids」リストに追加
- 106 行目: メッセージと画像をポスト
- 107 行目: ポストしたメッセージを print 表示
- 109 行目～124 行目: main プログラム
- 110 行目～120 行目 :try ブロック
- 111 行目: GPIO 端子初期設定プログラム setup_led()関数実行
- 112 行目: データ記録数カウント用プログラム count_data()関数実行
- 113 行目: post 用メッセージを作成するプログラム create_message()関数実行
- 115 行目～120 行目: データ数が 50 になった場合の処理を定義
- 115 行目: データ数=50 の判定
- 116 行目: ポストを実行
- 117 行目～119 行目: 青色 LED を 2 秒間点灯
- 120 行目: データ数確認用 csv ファイルのデータクリア実行
- 121 行目～122 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。
- 123 行目～124 行目: finally ブロック
  例外の発生によらず必ず実行される処理を記述しています。
  本プログラムは GPIO を使用しているため、GPIO.cleanup()を記述しています。これにより、例外の発生によらず GPIO ピンのシステムクリアが必ず実行されます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑧～シェルスクリプト作成～

IoT システム ver3 のシェルスクリプトを作成します。

[対象プログラム]

42_DataUpload.sh

シェルスクリプトに含まれる Python プログラムは、次のとおり。

- ✓ 40_CSV.py
- ✓ 26_GRAPH.py
- ✓ 28_Histogram.py
- ✓ 35_Pydrive.py
- ✓ 41_Xpost.py

手順⑨～プログラム配置～

作成したプログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

手順⑩～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

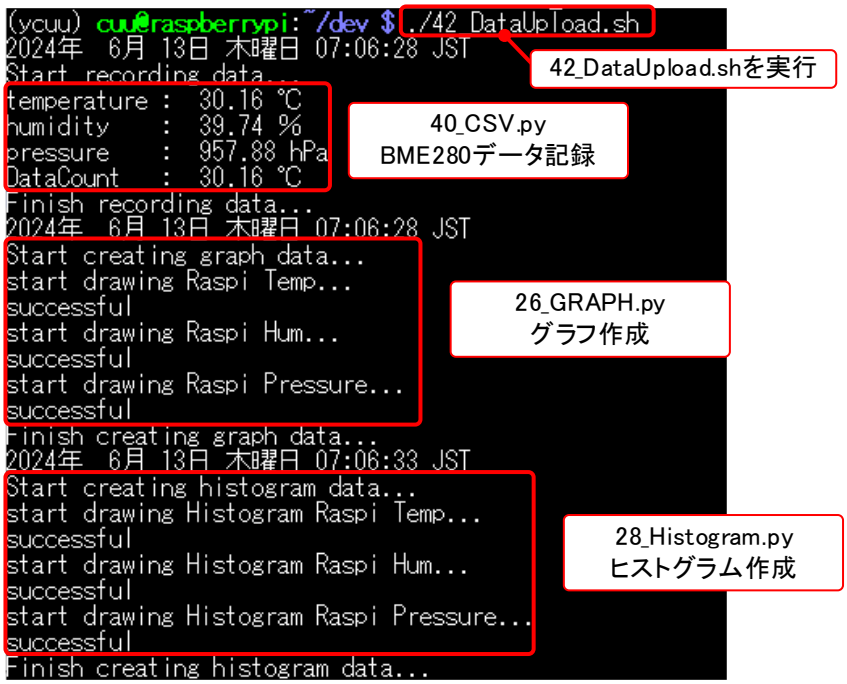
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 42_DataUpload.sh

手順⑪～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./42_DataUpload.sh

次の実行画面が表示されます。



(次のページへ続く)

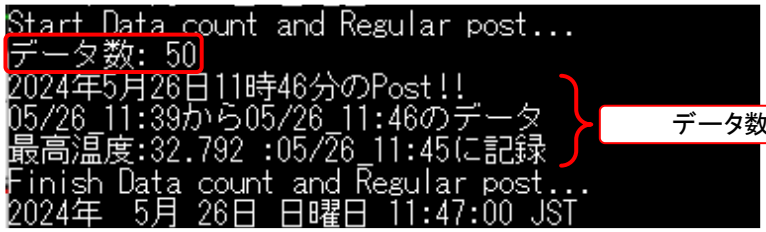
Copyright © 2024 YITJC All Rights Reserved.

（続き）

35_Pydrive.py
データアップロード

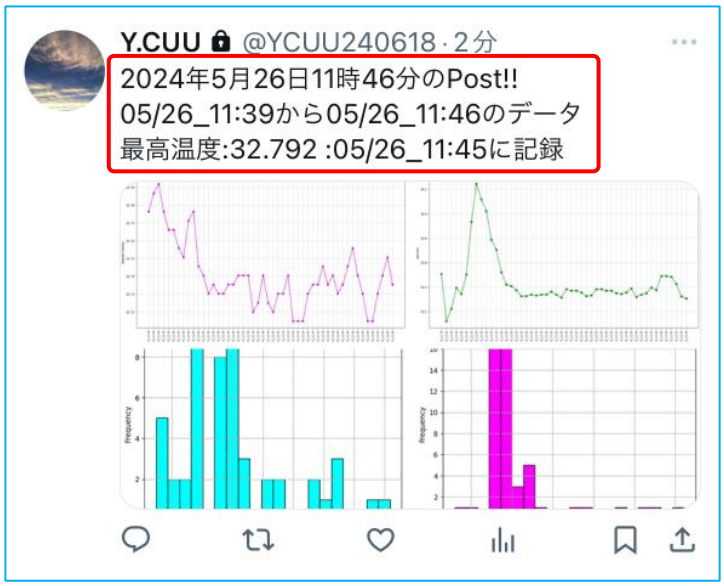
41_Xpost.py （データ記録数確認）

データ数が 50 個のタイミングでポストした場合は、次の画面が表示されます。



データ数50でポスト

X の画面でもポストを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/bin/sh
2	#データ記録(from BME280 to csv)
3	date
4	echo Start recording data...
5	/home/cuu/dev/./40_CSV.py
6	echo Finish recording data...
7	
8	#グラフ作成・グラフ画像生成
9	date
10	echo Start creating graph data...
11	/home/cuu/dev/./26_GRAPH.py
12	echo Finish creating graph data...
13	
14	#ヒストグラム作成・グラフ画像生成
15	date
16	echo Start creating histogram data...
17	/home/cuu/dev/./28_Histogram.py
18	echo Finish creating histogram data...
19	
20	#Google Driveアップロード
21	date
22	echo Start uploading all data...
23	/home/cuu/dev/./35_Pydrive.py
24	echo Finish uploading all data...
25	
26	#定期post
27	date
28	echo Start Data count and Regular post...
29	/home/cuu/dev/./41_Xpost.py
30	echo Finish Data count and Regular post...
31	date

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
  Raspberry Pi OS（Raspbian）は使用するシェルが「sh」、「bash」、「dash」などから選ぶことができます。
  ここでは、「sh」を選択してシェルスクリプトを作成しています。

Copyright © 2024 YITJC All Rights Reserved.

～BME280（温度・湿度・気圧）センサデータ記録～

- 3 行目～6 行目: センサデータ記録
- 3 行目: 日付・時刻表示
- 4 行目: コメント「Start recording data...」表示
- 5 行目: 40_CSV.py 実行
- 6 行目: コメント「Finish recording data...」表示

～BME280（温度・湿度・気圧）センサデータから時系列グラフ作成～

- 9 行目～12 行目: センサデータからグラフ作成
- 9 行目: 日付・時刻表示
- 10 行目: コメント「Start creating graph data...」表示
- 11 行目: 26_GRAPH.py 実行
- 12 行目: コメント「Finish creating graph data...」表示

～BME280（温度・湿度・気圧）センサデータからヒストグラム作成～

- 15 行目～18 行目: センサデータからヒストグラム作成
- 15 行目: 日付・時刻表示
- 16 行目: コメント「Start creating histogram data...」表示
- 17 行目: 28_Histogram.py 実行
- 18 行目: コメント「Finish creating histogram data...」表示

～Raspberry Pi「/home/cuu/data」ディレクトリのファイルを一括アップロード～

- 21 行目～25 行目: センサデータからヒストグラム作成
- 21 行目: 日付・時刻表示
- 22 行目: コメント「Start uploading all data...」表示
- 23 行目: 35_Pydrive.py 実行
- 24 行目: コメント「Finish uploading all data...」表示

～データ記録数確認＆定期ポスト実行～

- 27 行目～31 行目: 記録されているデータの数を確認し、データ 50 個のタイミングで定期ポスト実行
- 27 行目: 日付・時刻表示
- 22 行目: コメント「Start Data count and Regular post...」表示
- 23 行目:41_Xpost.py 実行
- 24 行目: コメント「Finish Data count and Regular post...」表示
- 31 行目: 日付・時刻表示

Copyright © 2024 YITJC All Rights Reserved.

現在稼働している IoT システム ver2 は、「accumulation.service」と「accumulation.timer」によって自動起動しています。この IoT システムを ver3 にアップグレードするため、サービス（タイマー）を停止します。

手順⑫～サービス・タイマー停止～

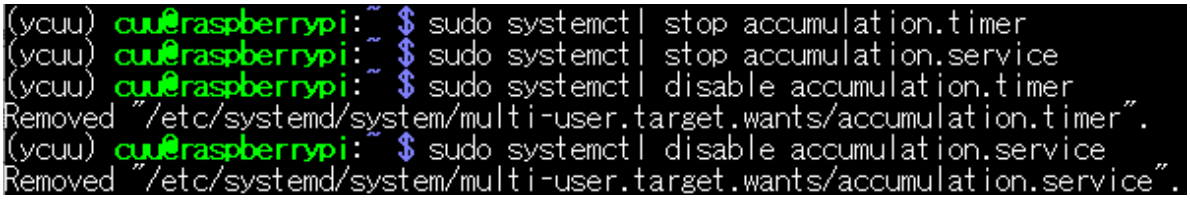
次のコマンドを実行してください。

(ycuu) cuu@raspberrypi:$ sudo systemctl stop accumulation.timer

(ycuu) cuu@raspberrypi:$ sudo systemctl stop accumulation.service

(ycuu) cuu@raspberrypi:$ sudo systemctl disable accumulation.timer

(ycuu) cuu@raspberrypi:$ sudo systemctl disable accumulation.service



手順⑬～Service ファイル編集～

「accumulation.service」ファイルを編集するため、「/etc/systemd/system」ディレクトリへ移動します。

「accumulation.timer」ファイルは、IoT システム ver2 と同じ時間間隔（5 分）で稼働させるため編集する必要はありません。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

～参考～

Unit ファイルの「上書き」、「終了」は、次の手順を参考にしてください。

・編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認

・「Enter」キーを押下して上書き保存を実行

・「Ctrl + X」キーを押して元の画面に戻る

続いて、「accumulation.service」ファイルを編集します。

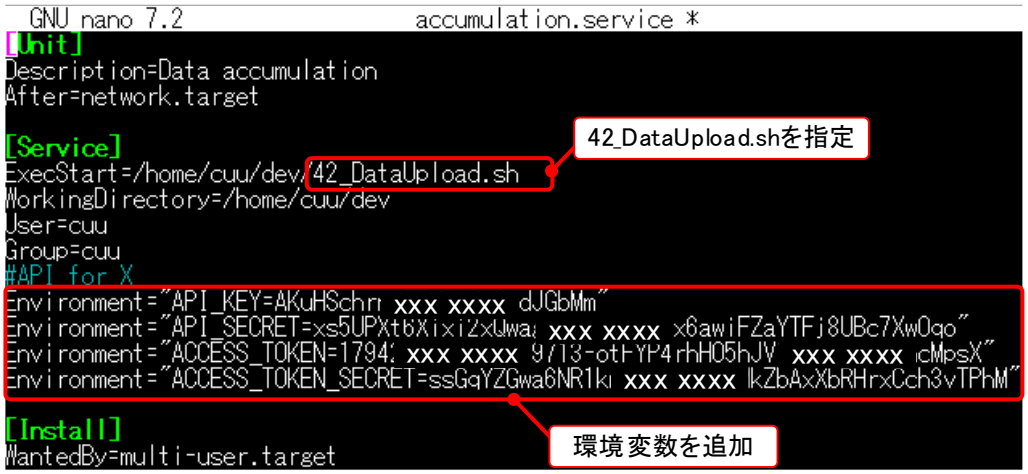
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano accumulation.service

Copyright © 2024 YITJC All Rights Reserved.

編集する内容は、次の 2 点です。

- 自動起動プログラムの変更:「42_DataUpload.sh」に変更
- 環境変数追加
  X を利用するために必要な情報（4 項目）を Unit ファイルの[Service]セクションに記載してください。

-記載方法-
Environment="環境変数名=設定値"



手順⑭～サービス再読み込み～

編集したサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順⑮～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.service

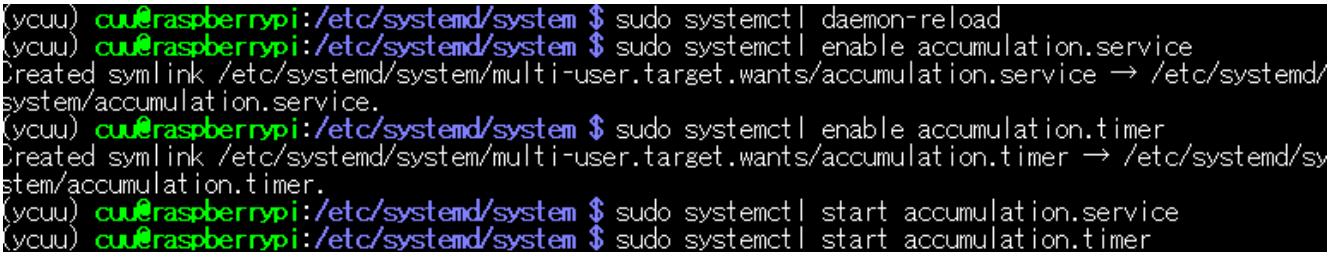
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable accumulation.timer

手順⑯～サービス開始～

次のコマンドを実行してサービスを開始させます。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.service

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start accumulation.timer



Copyright © 2024 YITJC All Rights Reserved.

手順⑰～サービス動作状況確認～

サービスの動作状況を確認します。

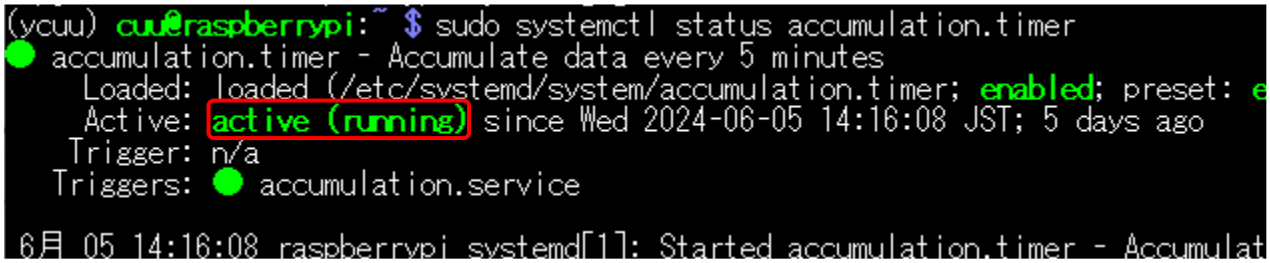
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status accumulation.timer

正常に稼働（自動起動）していれば、次の画面が表示されます。

この画面表示を終了する場合は、「q」を押してください。

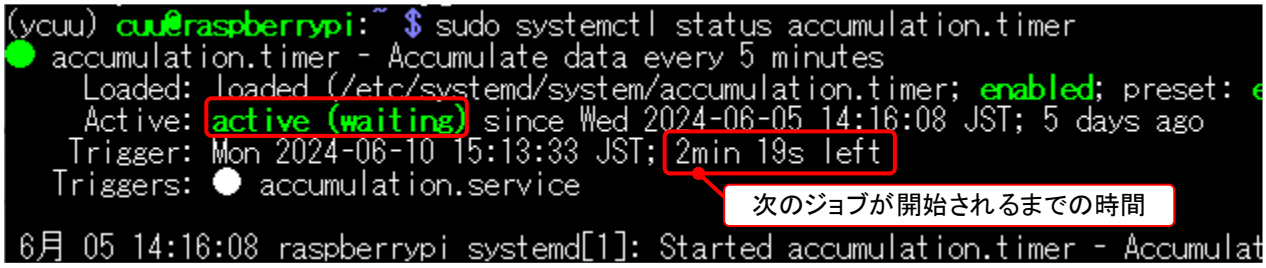
- active（running）の状態

  サービスが現在実行中であることを意味しています。



- active（waiting）の状態

  次のジョブが開始されるのを待っている状態（次のジョブが開催されるまでの時間が表示されます）



手順⑱～システムログ確認～

IoT システム ver3 の動作確認を行うため、次のコマンドを実行してシステムのログを確認してください。

(ycuu) cuu @raspberrypi:~ $ journalctl -u accumulation.service -n 20

- **オプションの設定（例）**

「-u Unit 名」：指定した Unit のログを表示

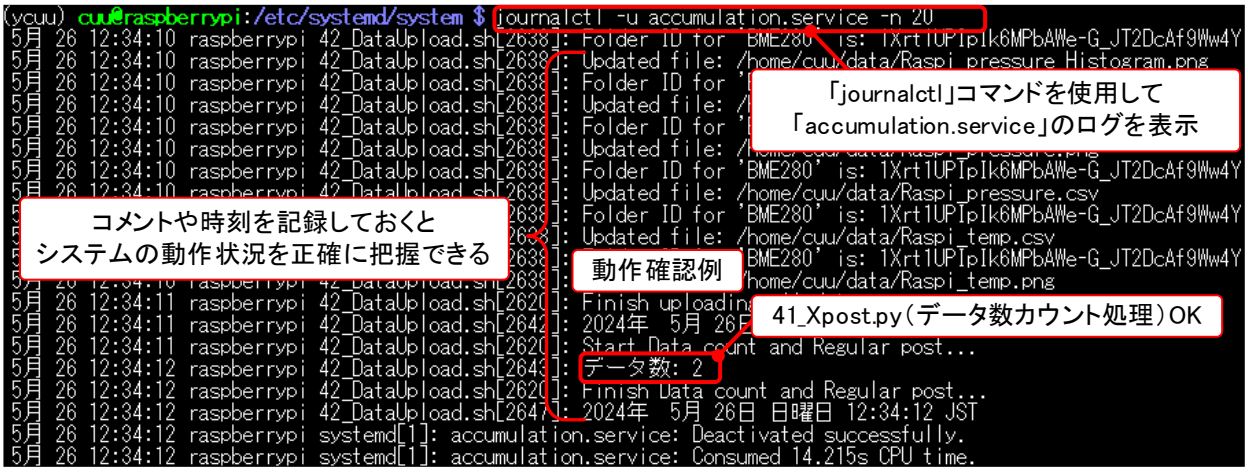
「-n 数」：末尾（直近）から指定した行数のログを表示

「-f」：リアルタイム監視（デバッグに便利）

「-xe」：詳細情報を表示

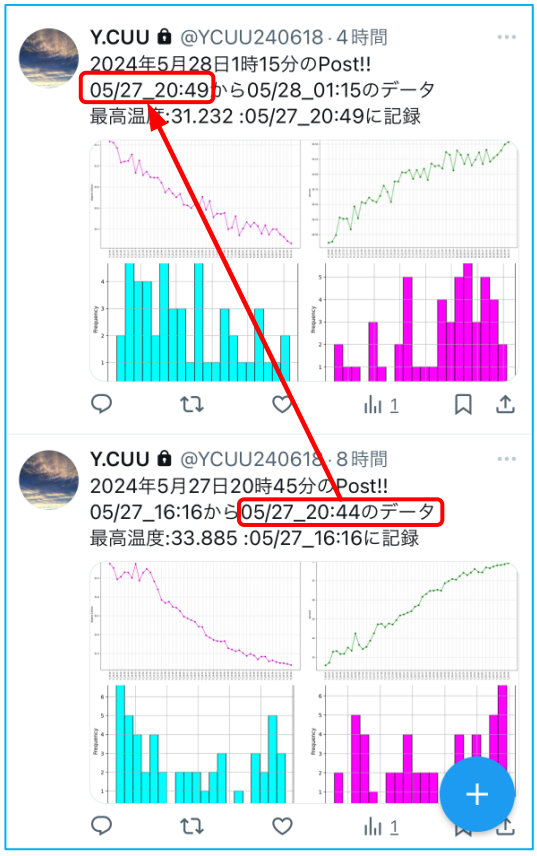
Copyright © 2024 YITJC All Rights Reserved.

定期ポストの周期は約 4 時間（長時間）であるため、システムのログを活用して動作確認を行ってください。



X の画面上でセンサの情報が定期的なポストで通知されていることを確認できます。

データカウント機能により、データが途切れることなく継続して情報が通知されています。



Copyright © 2024 YITJC All Rights Reserved.

9.4 生成 AI「ChatGPT」の活用

OpenAI による「ChatGPT」は 2020 年に公開され、自然な文章で高い品質の回答を幅広いジャンルで「生成」できることから、世界中で利用拡大が進んでいます。ChatGPT は、人工知能技術の「大規模言語モデル（LLM：Large Language Model）：GPT（Generative Pre-trained Transformer）」をベースにした対話型の AI です。

ChatGPT が公開されてから、「生成 AI」の技術は急速に進展し、文章だけでなく、画像や音楽、ビデオなどのデジタルコンテンツも自動で生成できるようになりました。このような背景から、ビジネスシーンでも生成 AI の導入が進み、文書・画像作成、顧客対応の自動化（AI チャット bot）や企画のアイデア創出、プログラミングのソースコード生成など、多岐に渡る業務で生成 AI の活用が進んでいます。

OpenAI 社は、さまざまな AI モデルへのアクセスを可能とする OpenAI API を公開しています。OpenAI API は、通常の「ChatGPT」と同様に文章の生成や対話のタスクを実行できますが、さらに OpenAI が提供している他のモデルを利用することで、画像や音声の生成、データ分析などの機能も利用できます。

例えば、IoT システムが取得したデータを OpenAI API に渡し、そのデータの解析を依頼することもできます。

ここでは、Python プログラムを使用して OpenAI API によるデータ解析結果を取得する方法について学習します。

手順①～API Key 作成～

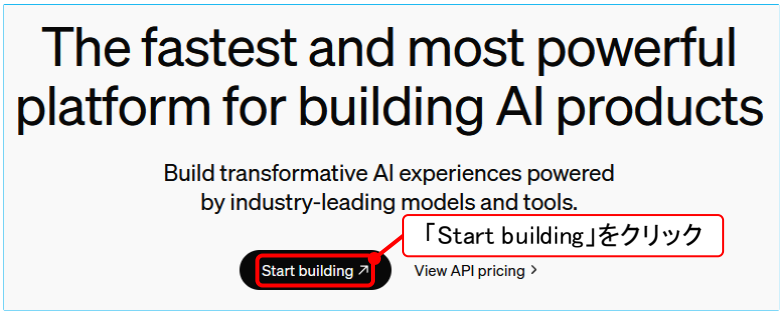
OpenAI API を利用するためには「API Key」を作成する必要があります。

次の URL をクリックしてログインしてください。

～補足～

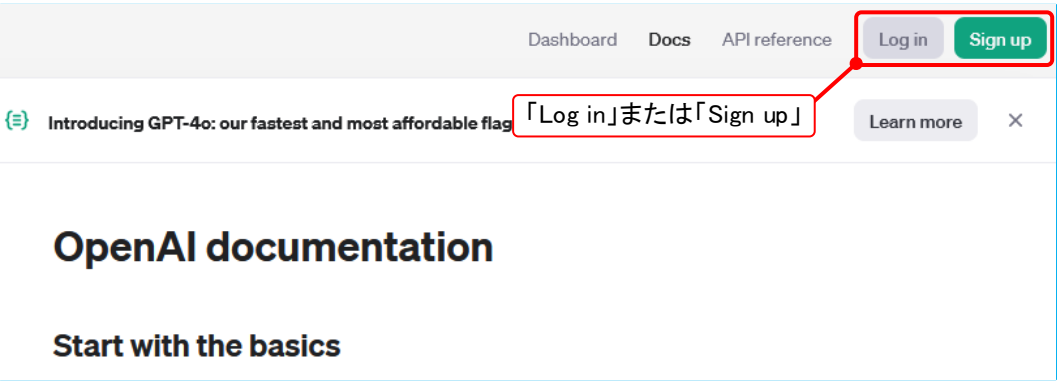
Google アカウントでもログインできます。

[URL] https://openai.com/api/



画面左上にある「Log in」または「Sign up」をクリックしてください。

ここでは、Google アカウントを使用して「Log in」する手順で説明します。



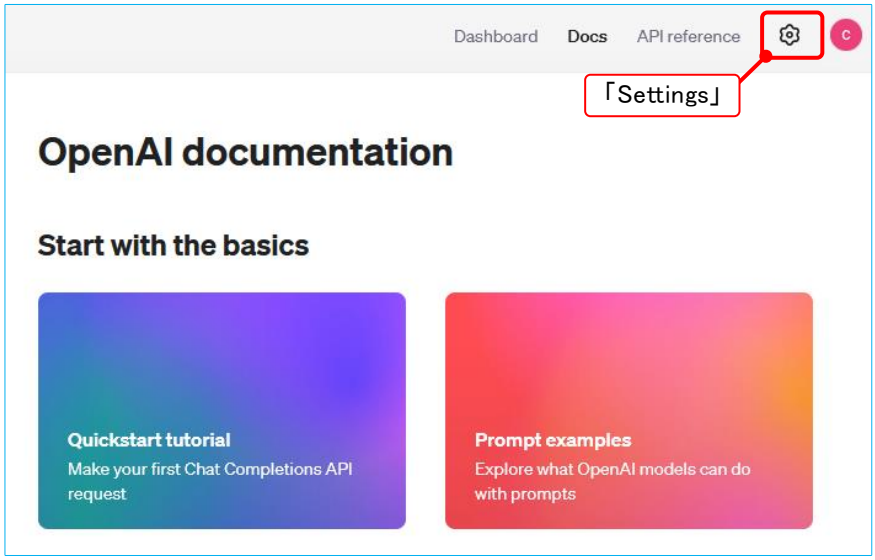
Copyright © 2024 YITJC All Rights Reserved.

「Google で続行」を選択し、Google アカウントでログインしてください。

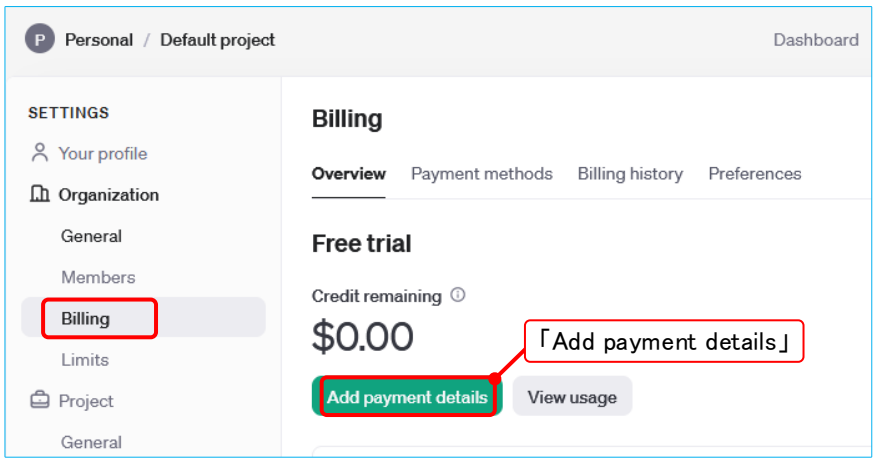
「名前」、「生年月日」を入力して「同意する」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

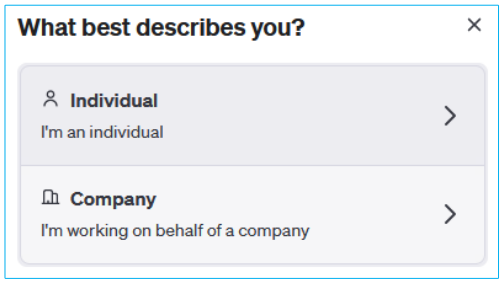
画面左上にある「Settings」をクリックしてください。



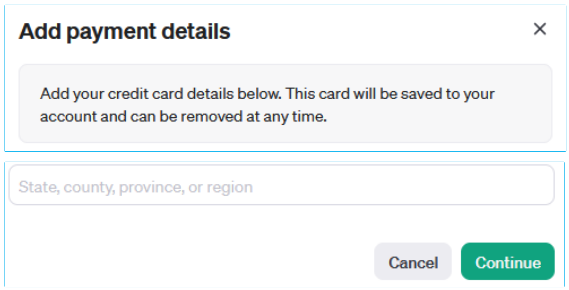
画面左のメニューから「Billing」をクリックしてください。続いて、「Add payment details」をクリックしてください。



個人「Individual」または企業「Company」を選択して次に進んでください。



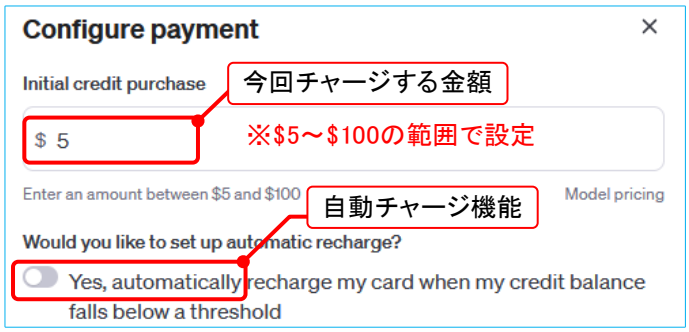
クレジットカード情報を入力して「Continue」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「Initial credit purchase」で今回チャージする金額（USドル）を入力してください。

続いて、「Automatic recharge（自動チャージ）」のチェックを外してください。



参考～OpenAI API の料金体系～

OpenAI API は「従量課金制度」を採用しています。「自動チャージ機能」を無効にしている場合、チャージした金額を超えると OpenAI API は利用できなくなります。OpenAI API の利用料金は、次の公式サイトで確認できます。

[URL] https://openai.com/api/pricing/

言語モデルの料金体系を例に挙げます（2024 年 5 月 31 日時点の情報）。

GPT-4o は、2024 年 5 月 13 日に発表されたモデルです。

モデル	入力	出力
GPT-4 o	$5 / 1M tokens	$15 / 1M tokens
GPT-4 Turbo	$10 / 1M tokens	$30 / 1M tokens
GPT-3.5 Turbo	$0.5 / 1M tokens	$1.5/ 1M tokens

参考～トークン数の確認～

入力した文章の「トークン数」を確認したい場合、OpenAI 公式サイトで提供されている「Tokenizer」で入力した文章からトークン数を確認できます。

[URL] https://platform.openai.com/tokenizer

次の質問を「ChatGPT」に送信した場合のトークン数は、「30」です。

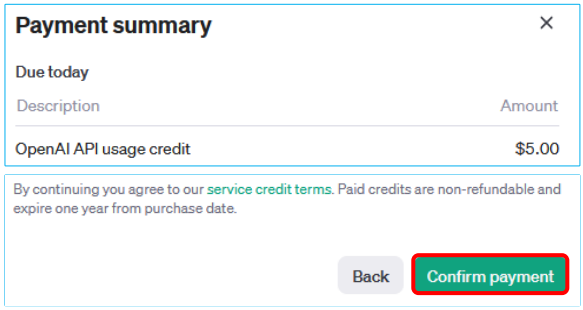
GPT-4 Turbo で入力の料金を計算すると、US$10 ×（30/ 1M token）= US$0.0003 になります。

-GPT4-

やまなしキャリアアップ・ユニバーシティについて教えてください。

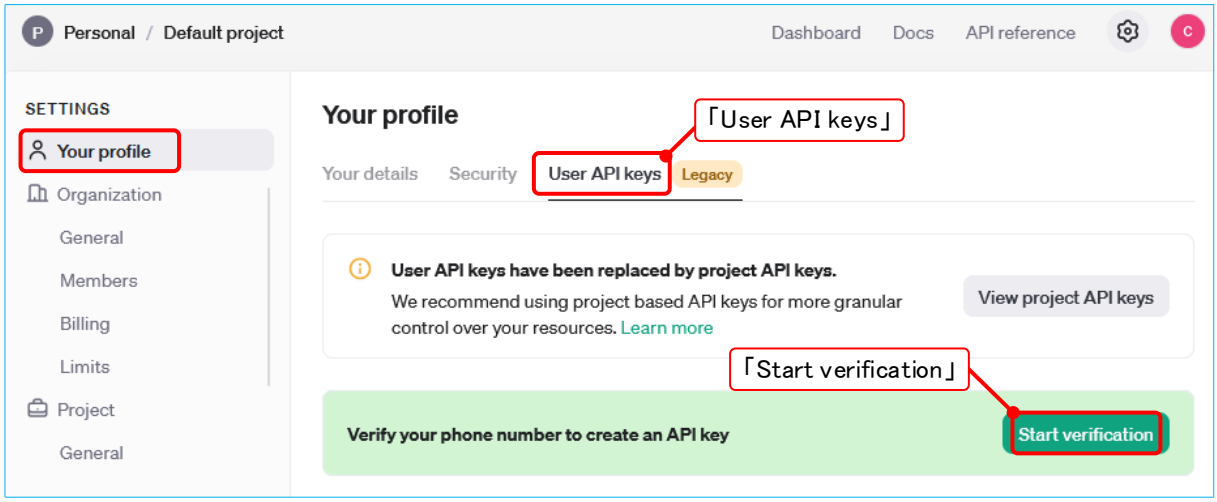
Copyright © 2024 YITJC All Rights Reserved.

続いて、「Confirm payment」をクリックしてください。

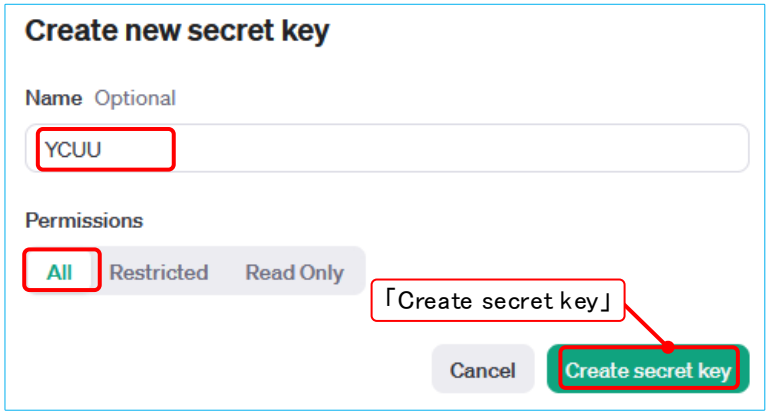


利用料金をチャージした後は、「Billing」画面で残高を確認できます。

画面左のメニューから「Your profile」をクリックし、「User API keys」をクリックしてください。
API Key を発行するため、電話番号を登録する必要があります。「Start verification」をクリックしてください。

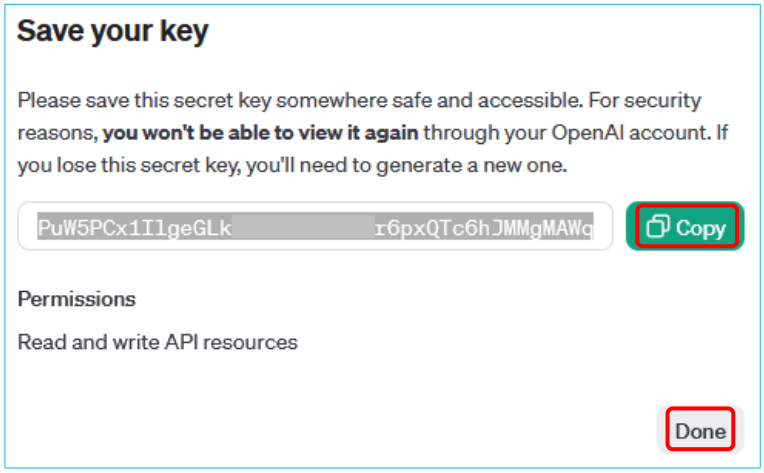


スマートフォンを使用した電話番号認証が完了したあと、「名前」を入力してから「Create secret key」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

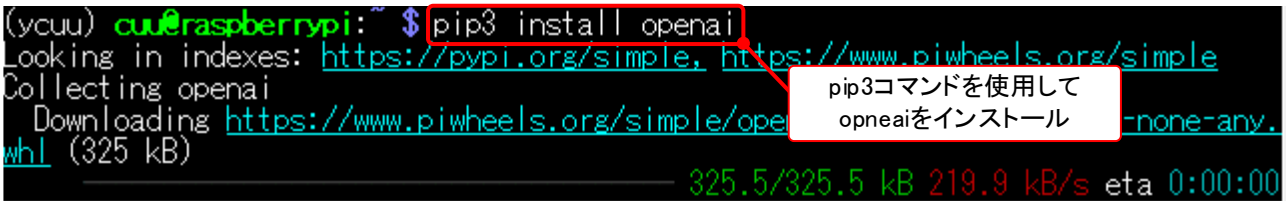
生成された「API key」はプログラムで使用するので、メモ帳などに記録してください。

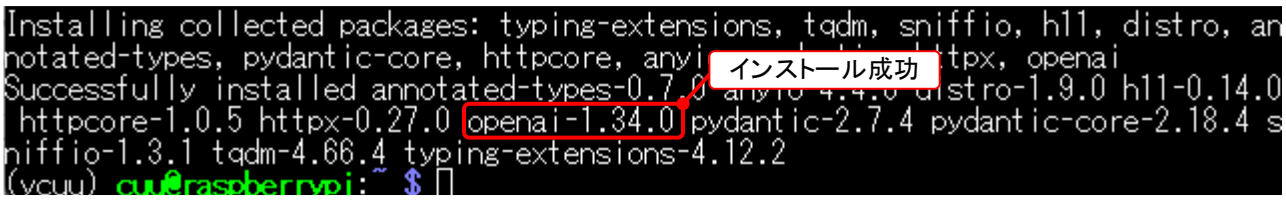


手順②～openai インストール～

Python プログラムを使用して OpenAI API を利用するために必要なライブラリ「openai」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install openai





～補足～

openai と同時に OpenAI API の動作に必要なライブラリがインストールされます。

手順③～環境変数設定～

OpenAI API を利用するために必要な情報（手順①で取得した情報）を環境変数に設定します。

環境変数に登録する情報は、次の情報です。

```
export OPENAI_API_KEY='sk-qoYcFPuW5PCxxxxxxxxxx3BlbkFJBhfr6pxQTc6hJMMgMAWq'
```

Raspberry Pi が起動したときに環境変数を自動的に設定するため、「.bashrc」ファイルに環境変数の設定コマンドを記述します。

ホームディレクトリに移動し、ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

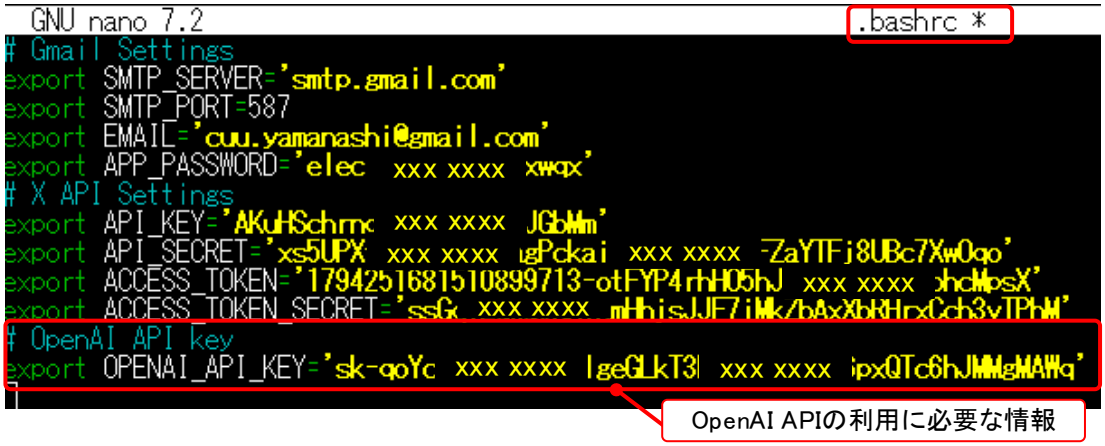
(ycuu) cuu@raspberrypi:~/dev $ cd ..

※「cd ..」コマンドを実行すると、1 つ上のディレクトリに移動できます。

(ycuu) cuu@raspberrypi:$ nano .bashrc

Copyright © 2024 YITJC All Rights Reserved.

「.bashrc」ファイルの画面が起動したら、末尾に「export」コマンドで変数を 1 つ追加（記述）します。



記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

手順④～環境変数の反映～

続いて、次のコマンドを実行して設定した環境変数を反映します。

(ycuu) cuu@raspberrypi:$ source ~/.bashrc

```
cuu@raspberrypi:~ $ source ~/.bash_profile
```

「~」の部分（チルダ）は、「Shift」+「～」で入力します。

手順⑤～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

43_OpenAI.py

手順⑥～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 43_OpenAI.py

手順⑦～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./43_OpenAI.py

実行結果については、プログラム解説の中で紹介します。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from openai import OpenAI
5	import pandas as pd
6	import os
7	
8	# CSVファイルのURLパス
9	FILE_PATH = "/home/cuu/data/"
10	
11	#解析用CSVファイルをテキストデータに変換
12	def analysis_data():
13	# CSVファイルをデータフレームとして読み込む
14	df_t = pd.read_csv(FILE_PATH + "Raspi_temp.csv", names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
15	#温度と時間の関係をテキスト形式で出力する
16	text_tdata = ""
17	for index, row in df_t.iterrows():
18	text_tdata += f"日付時刻: {row['year']}-{row['month']}-{row['day']} {row['hour']}:{row['minute']}, 温度: {row['temperature']}¥n"
19	# CSVファイルをデータフレームとして読み込む
20	df_h = pd.read_csv(FILE_PATH + "Raspi_hum.csv", names=['year', 'month', 'day', 'hour', 'minute', 'humidity'])
21	#湿度と時間の関係をテキスト形式で出力する
22	text_hdata = ""
23	for index, row in df_h.iterrows():
24	text_hdata += f"日付時刻: {row['year']}-{row['month']}-{row['day']} {row['hour']}:{row['minute']}, 湿度: {row['humidity']}¥n"
25	return text_tdata, text_hdata
26	
27	#OpenAIに解析依頼
28	def open_ai():
29	#解析用テキストデータ取得
30	text_tdata, text_hdata = analysis_data()
31	#API Keyを使用してインスタンス作成
32	client = OpenAI(api_key=os.getenv('OpenAI_API_KEY'))
33	#Message format
34	#messages=[{"role": "user", "content": prompt}]
35	completion = client.chat.completions.create(

Copyright © 2024 YITJC All Rights Reserved.

```
36              #model="gpt-3.5-turbo",
37              model = "gpt-4-turbo",
38              #model = "gpt-4o",
39              max_tokens = 500,
40              messages=[
41                  {"role": "system", "content": "あなたはデータ分析の専門家です。温度と湿度のデータを正確に分析し、報告することが業務です。"},
42                  {"role": "user", "content": "分析結果には、データ測定期間、最大値、最小値のデータおよび温度と湿度の相関関係を含めてください。"},
43                  {"role": "user", "content": text_tdata},
44                  {"role": "user", "content": text_hdata},
45              ]
46      )
47      result_text = completion.choices[0].message.content
48      print(result_text)
49
50  if __name__ == '__main__':
51      try:
52          open_ai()
53      except KeyboardInterrupt:
54          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: OpenAI API を使用するために必要なライブラリ「openai」をインポート
- 5 行目: データ分析を行うため pandas ライブラリをインポート
- 6 行目: OS と対話するためのライブラリ「os」をインポート
- 9 行目: 分析用データ（csv ファイル）が保存されているディレクトリを指定
- 10 行目～15 行目: ポストに添付する画像ファイルを指定

～OpenAI 解析用テキストデータ作成～

- 12 行目～24 行目: 解析用テキストデータ作成プログラム analysis_data()関数
  BME280 センサデータ（csv ファイル）を読み取り、テキストデータに変換する。
- 14 行目: 温度 csv ファイルを読み取り、データフレーム（df_t）に格納
- 16 行目: 解析用テキストデータ（温度データ）を格納する変数「text_tdata」を初期化
- 17 行目～18 行目: データフレーム（df_t）を 1 行ずつ読み取りテキストデータに変換
- 20 行目: 温度 csv ファイルを読み取り、データフレーム（df_h）に格納
- 22 行目: 解析用テキストデータ（温度データ）を格納する変数「text_hdata」を初期化
- 23 行目～24 行目: データフレーム（df_h）を 1 行ずつ読み取りテキストデータに変換
- 25 行目: 他の関数で使用するため「text_tdata」と「text_hdata」を戻り値として返す

Copyright © 2024 YITJC All Rights Reserved.

～OpenAI へのリクエスト～

- 28 行目～50 行目: OpenAI にリクエストを送信しレスポンスを表示するプログラム open_ai()関数
- 30 行目: 解析用テキストデータを analysis_data()関数から取得
- 32 行目: API key を使用して OpenAI クラスのインスタンスを作成
  環境変数（OPENAI_API_KEY）は os.getenv()メソッドを使用して取得
- 35 行目～48 行目: OpenAI にリクエスト（messages）を送信し応答を生成
- 38 行目: 応答の生成に使用するモデルを選択

～参考～

使用できるモデル一覧を掲載します。（参考：https://platform.openai.com/docs/models）

Model	概要
GPT-4o	最も高速で価格も手頃なフラグシップモデル
GPT-4 Turbo and GPT-4	一世代前の高性能モデル
GPT-3.5 Turbo	高速・低価格（シンプルなタスク向け）
DALL·E	自然言語の記述からデジタル画像を生成
TTS	Text to Speech（テキストを自然な音声データに変換）
Whisper	音声データからテキストデータを生成（文字起こし）
Embeddings	テキストデータを数値ベクトルに変換
Moderation	テキストからセンシティブまたは不適切な内容を検出
GPT base	自然言語やコードを生成

「43_OpenAI.py」では、OpenAI に対して次のリクエストを送信しています。

-リクエストメッセージ-

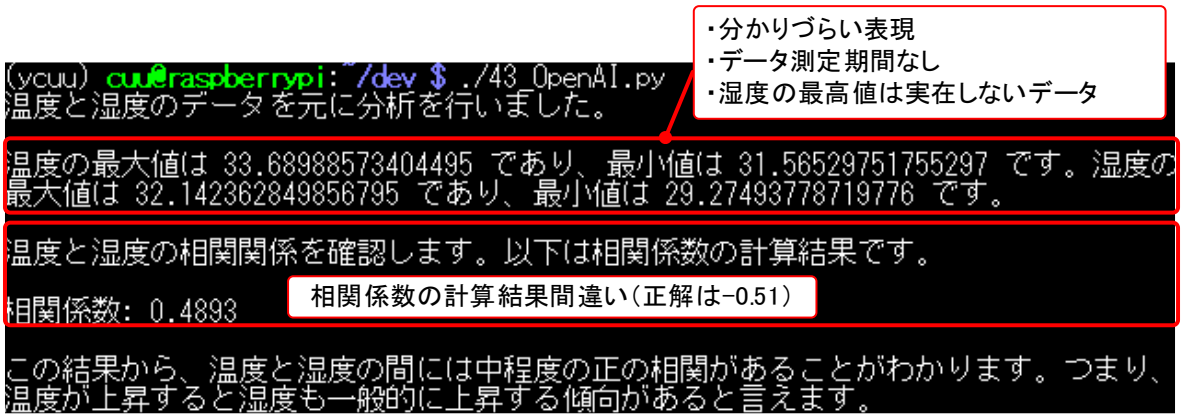
あなたはデータ分析の専門家です。温度と湿度のデータを正確に分析し、報告することが業務です。

分析結果には、データ測定期間、最大値、最小値のデータおよび温度と湿度の相関関係を含めてください。

温度データ

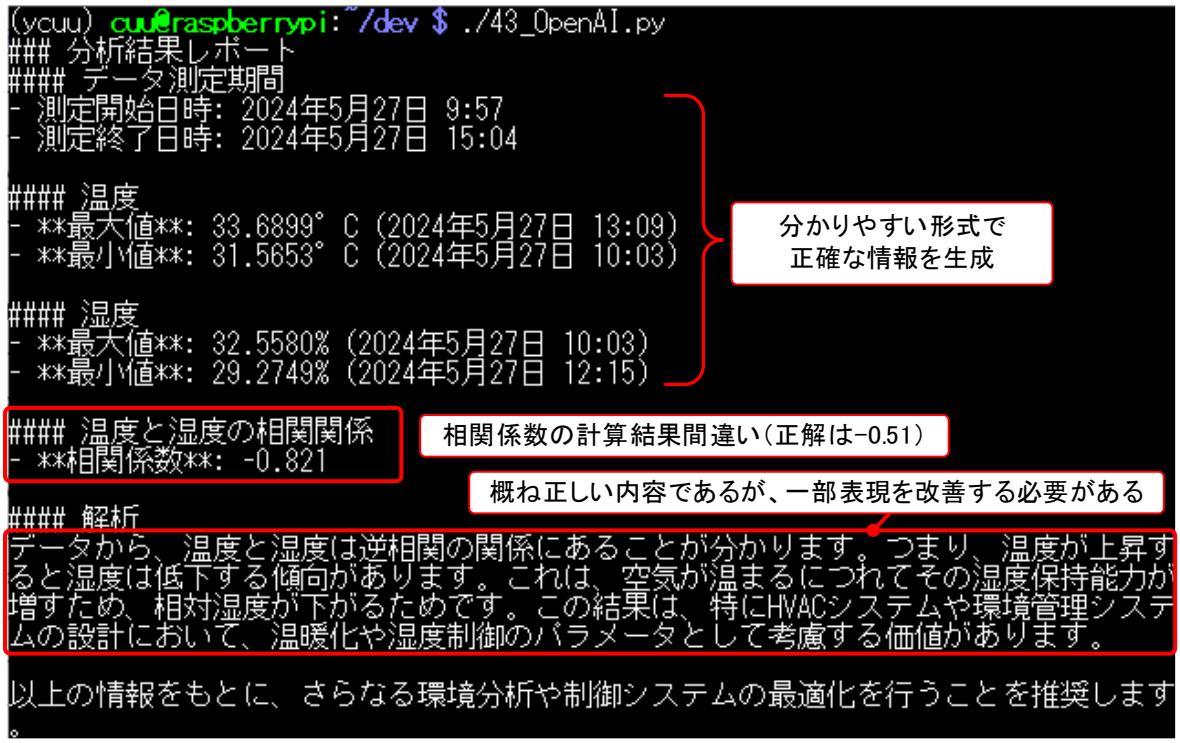
湿度データ

- GPT-3.5-Turbo
  - ✓ リクエストに対して、「データ測定期間」の応答が欠落している
  - ✓ 湿度データの最高値として報告されているデータは実在しない
  - ✓ 相関係数の計算が間違えているため、「正の相関」という間違えた分析結果になっている



Copyright © 2024 YITJC All Rights Reserved.

- GPT-4-Turbo
  - ✓ リクエストに対して分かりやすい形式で正確な情報を応答している
  - ✓ 相関係数の計算が間違えている
  - ✓ 「解析」の内容は概ね正しいが、改善が必要な箇所がある



- 39 行目: 生成するテキストの最大トークン数を定義（スペースや句読点を含む）
- 40 行目～45 行目: OpenAI にリクエストするメッセージ（messages）を作成

～参考～

messages リストの構造

messages=[{"role": "user", "content": prompt}]

- role: メッセージの役割は以下の 3 つがあります。
  - ✓ system

    OpenAI リクエストに対して model に役割を与えると精度の高い回答が得られると言われています。

    system メッセージは model に役割や応答の条件を与えるために使用されます。
  - ✓ assistant

    OpenAI からの応答を出力します。

    assistant で回答例を示すことで、リクエストに対して望ましい回答を得るために使われます。
  - ✓ user

    リクエストメッセージを送信するユーザーを表します。

「system」と「user」を組み合わせてメッセージを送信した場合、GPT-4-Turbo では「system」のリクエストが優先されると言われています。しかし、生成 AI の技術は日々進化しているため、role の設定やリクエストメッセージの内容など、さまざまなパターンを組み合わせてテストしてください。

- content: リクエストメッセージ

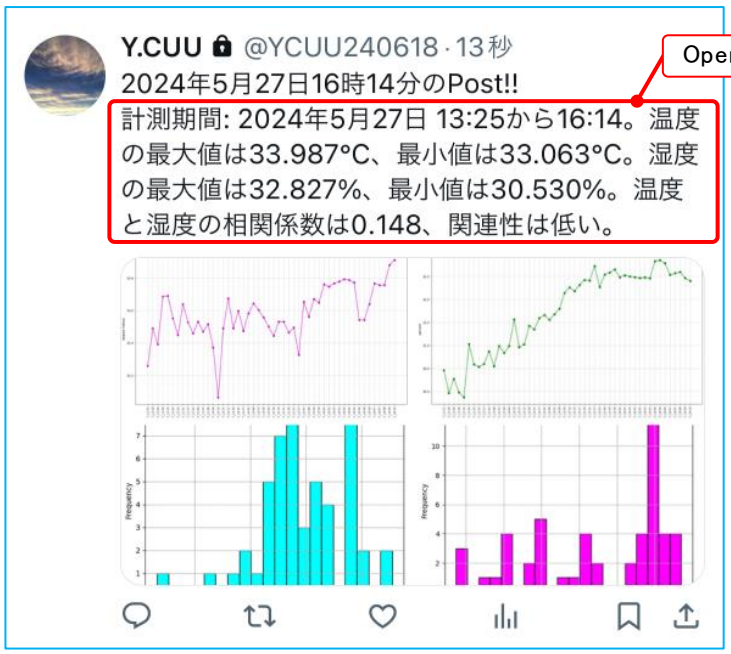
Copyright © 2024 YITJC All Rights Reserved.

- 47 行目: OpenAI が生成した応答を変数「result_text」に格納
- 48 行目: OpenAI が生成した応答を print 表示
- 50 行目～54 行目: main プログラム
- 51 行目～52 行目 :try ブロック
- 52 行目: OpenAI にリクエストを送信しレスポンスを表示するプログラム open_ai()関数実行
- 53 行目～54 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。

参考～生成 AI を活用した Post～

IoT システム ver3 は定期的なポストを実行していますが、定期ポストで発信するメッセージに OpenAI が生成した解析データを使用することもできます。

この例で使用したプログラム「44_Xpost.py」を共有サイトに公開しておきます。



Web サイトやアプリで使用する Chat GPT と比較して、OpenAI API は model や role など、さまざまな設定を組み合わせて最適な応答を生成します。ただし、現時点では GPT-4-Turbo を使用した場合でも、要求する応答レベルに達しないケースが多いです。しかし、生成 AI の技術は急速に進展しているため、API の機能や生成される応答の精度を定期的に確認することが重要です。

Copyright © 2024 YITJC All Rights Reserved.

9.5 アラート通知機能をもつ IoT システム（ver4）

IoT システム ver3 に対して「異常監視」および「アラート通知」機能を追加して ver4 を構築します。

IoT システム ver3 は、オンラインストレージに保存されたデータや定期通知で送信される情報から生産状況を把握できるシステムです。しかし、このシステムでは情報の収集が定期的にしか実施されないため、予期せぬ異常の検出が遅れるケースがあります。そのため、生産現場で発生した異常を早期に発見し、異常発見遅れに伴う損失（不良品、材料費のロス）を抑える必要があります。

ここでは、生産設備の「異常監視」や「アラート通知」システムを構築する方法について学習します。また、アラート通知には異常発生前後のデータを添付して通知するため、異常原因の早期特定と復旧指示の迅速化が期待できます。

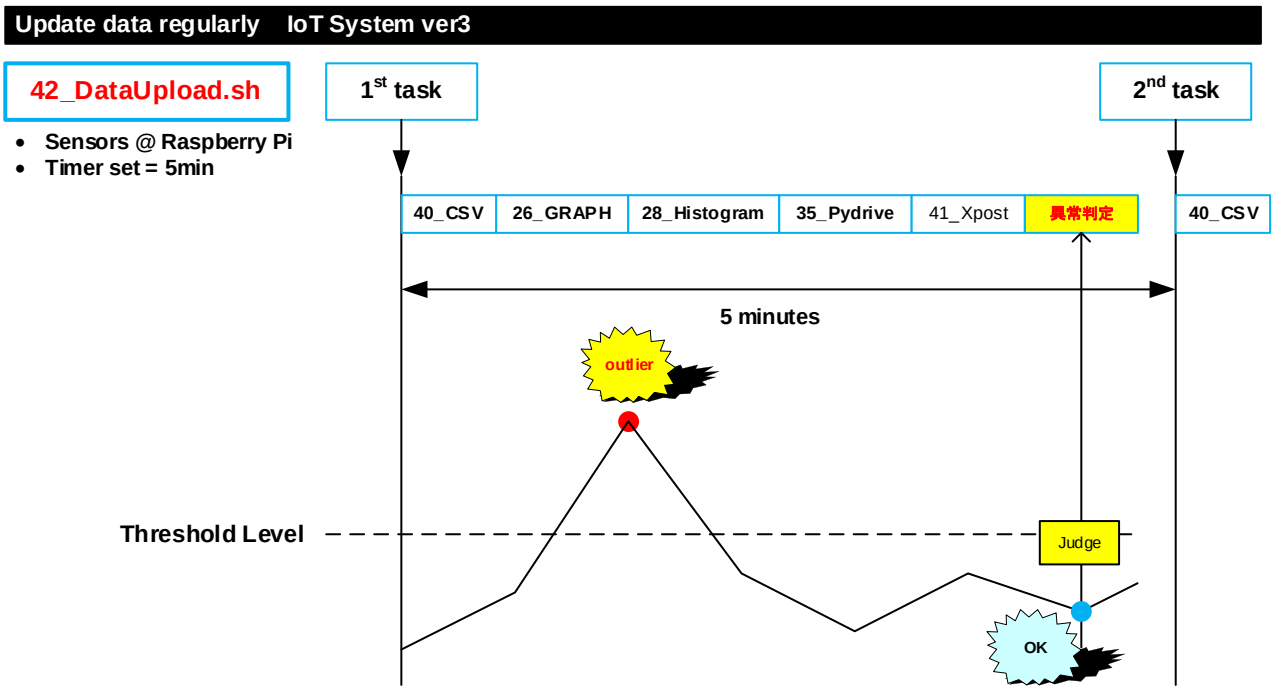
～ver4 システム構成検討～

「異常監視」および「アラート通知」機能を追加する前に、「IoT システム ver3」の構成を復習します。

IoT システム ver3 では、センサデータ記録から Google Drive へのアップロード、定期通知プログラムをシェルスクリプト「42_DataUpload.sh」にまとめ、5 分ごとに実行しています。

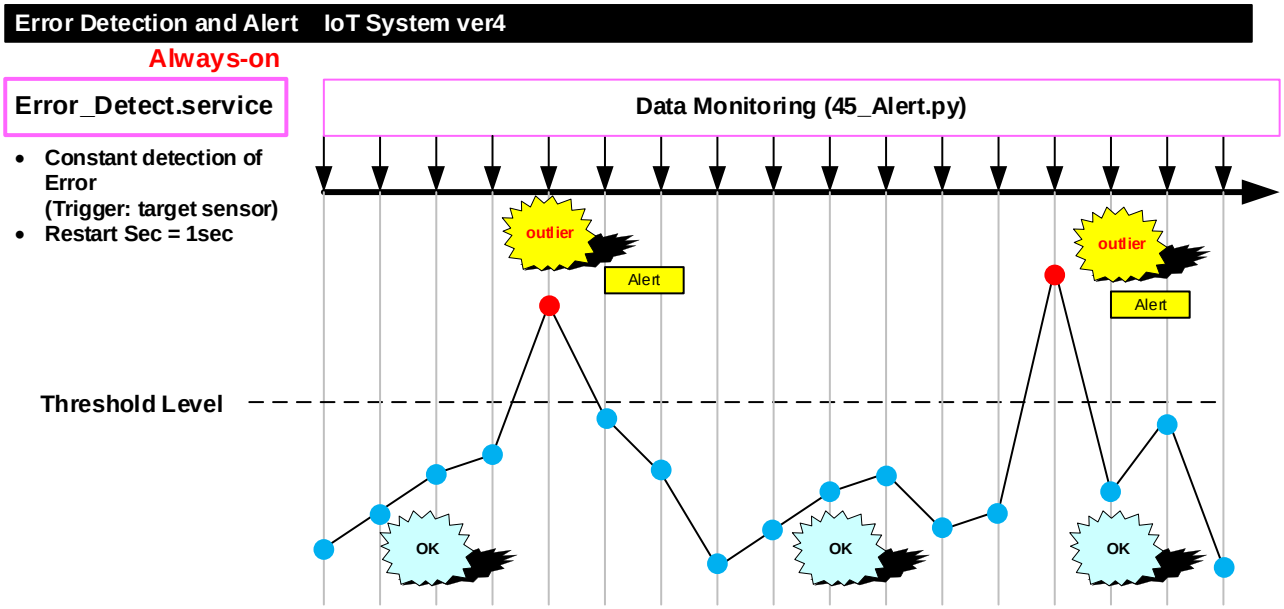
例えば、「42_DataUpload.sh」の最後に「異常判定」を実施した場合、システムの構築は簡単ですが、このシェルスクリプトは 5 分毎に実行されるため、異常判定のタイミングで異常を検出できるとは限りません。

次の図は、異常判定プログラムが動作していないタイミングで異常（outlier）が発生している様子を示しています。異常判定プログラムが動作しているときはデータが正常値に戻っているため、異常検出とはなりません。

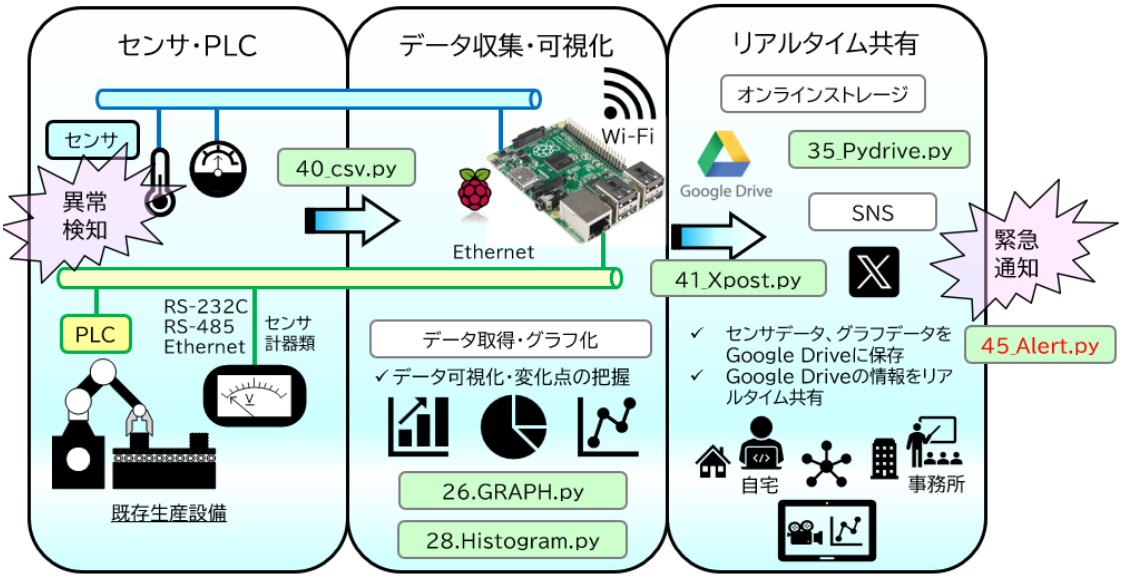


Copyright © 2024 YITJC All Rights Reserved.

IoT System ver4 は、異常検知プログラムを常時稼働して異常データを検出します。また、異常を検出した場合は、異常検出前後のデータを添付して直ちにアラートメールを送信します。

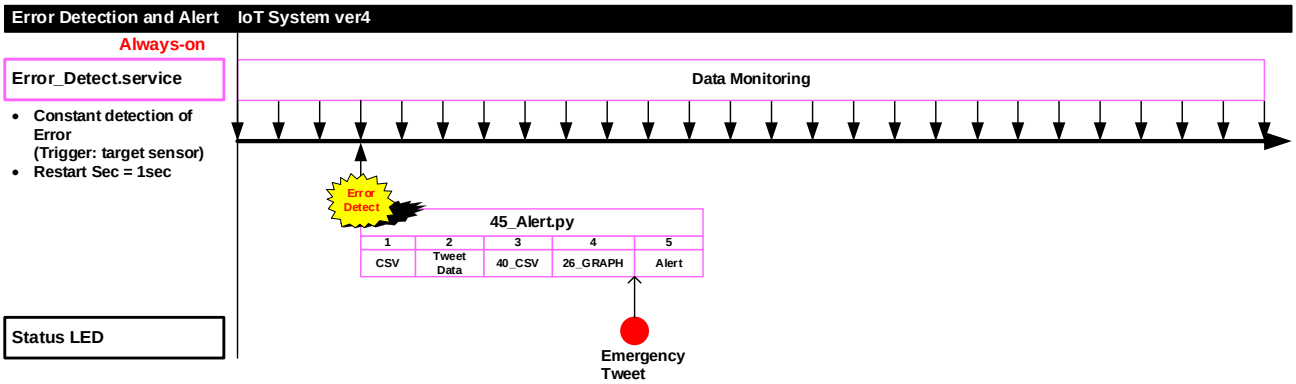


IoT システム ver4 の構成を示します。



～プログラム構成～

IoT システム ver4 のプログラム構成を示します。



Copyright © 2024 YITJC All Rights Reserved.

1 つのプログラムでデータ異常監視からアラート通知までの処理を行います。

③および④の処理は、異常検出前後のデータを分析するために実施している処理です。

➔ ③および④の処理内容は、以前作成したプログラムを呼び出して実行します。

使用するプログラム	処理内容
45_Alert.py	✓ データ常時監視（対象データ: BME280 Temp） ✓ 異常データを検出した場合…次の処理を実施する。 ① 異常検出対象データ（BME280 Temp）を CSV ファイルに書き込む ② アラート通知メッセージに必要な情報取得（検出値・検出時刻） ③ センサデータ CSV ファイルに書き込み（40_CSV.py プログラム呼び出し） ④ センサデータグラフ作成（26_GRAPH.py プログラム呼び出し） ⑤ アラートメール送信（赤色 LED 点灯）

手順①～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

45_Alert.py

[プログラム編集個所]

- 27 行目（異常判定の閾値設定）

  異常判定の閾値を編集してください。

  ```
  #異常判定値
  Judge = 36.0
  ```

- 126 行目（送信先メールアドレス）

  送信先メールアドレスを編集してください。

  ```
  to_email = 'xxxxx@gmail.com'      #送信先メールアドレスを設定
  ```

～補足　閾値の設定～

プログラムを実行すると、対象データ（Raspi temp）の監視を開始します。公開しているプログラムは、Raspi temp の値が 36℃以上になると異常検出と判定してアラートメールを送信します。

プログラムを実行する前に、「/home/cuu/data」ディレクトリに記録されている Raspi temp の温度データを確認してください。直前に記録されている温度データに「+1℃」した値を閾値に設定してください。

```
-閾値の設定（27 行目）-
#異常判定値
Judge = 36.0
```

手順②～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 45_Alert.py
```

Copyright © 2024 YITJC All Rights Reserved.

手順③～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./45_Alert.py

プログラムを実行すると、アラートメールが送信されます。送信先のメールアドレスでメールを受信して内容を確認してください。

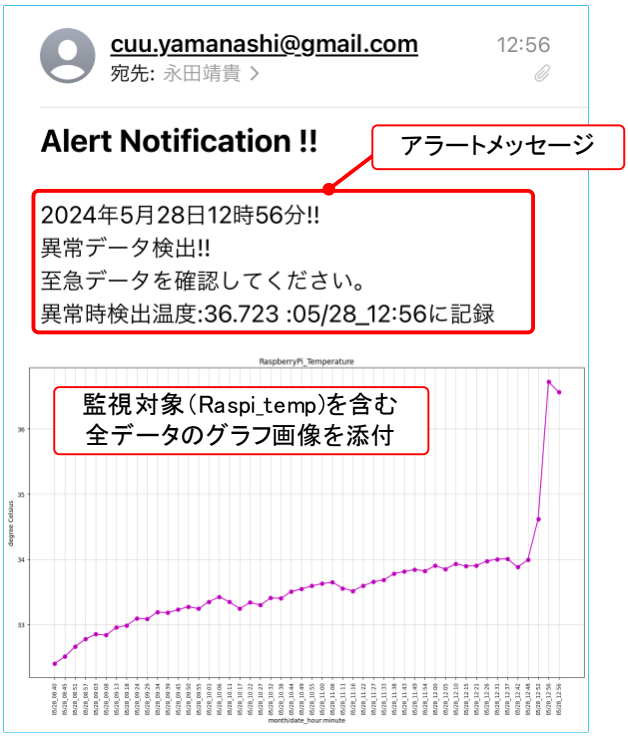
プログラムを実行すると、次の画面が表示されます。

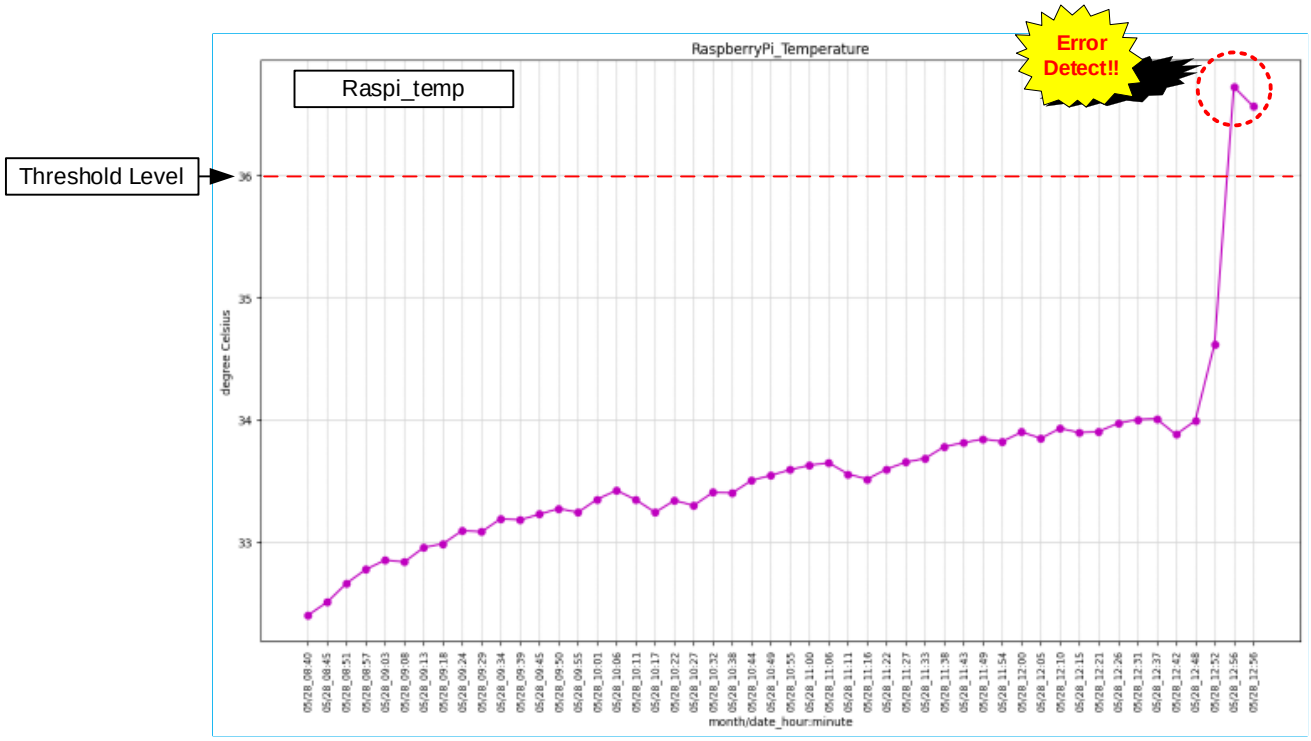
最初に 3 秒間隔で監視対象の温度データ（Raspi_temp）が表示されます。監視対象データが閾値（36℃）を超えると、次の処理が行われます。

① 検出した異常値（閾値を超えたデータ）を CSV ファイルに書き込む
② アラート通知メッセージに必要な情報（検出値・検出時刻）を取得し、アラートメールのメッセージを作成
③ 全センサデータ CSV ファイルに書き込み (40_CSV.py プログラム呼び出し)
④ 全センサデータグラフ作成 (26_GRAPH.py プログラム呼び出し)
⑤ アラートメール送信（赤色 LED 点灯))

Copyright © 2024 YITJC All Rights Reserved.

アラートメールが送信されたメールアドレスで「アラートメッセージ」と「異常検出前後のデータ(グラフ画像)」を確認できます。





Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import BME280_Class
5	import time
6	import datetime
7	import csv
8	import pandas as pd
9	import RPi.GPIO as GPIO
10	import subprocess
11	import smtplib　　#SMTPプロトコルを使用してメールを送信するためのライブラリ
12	import os　　#OSとの対話を提供するライブラリ
13	from email.mime.multipart import MIMEMultipart　#MIME形式のマルチパートメッセージを作成するためのクラス
14	from email.mime.text import MIMEText　　#MIME形式のテキストメッセージを作成するためのクラス
15	from email.mime.base import MIMEBase　　#MIME形式のベースメッセージを作成するためのクラス
16	from email import encoders　　#MIMEメッセージのエンコードを行うためのライブラリ
17	
18	#CSVファイルデータ数・保存先
19	MAX_RANGE=50
20	URL_PATH="/home/cuu/data/"
21	attachment_list = [
22	URL_PATH + "Raspi_temp.png",
23	URL_PATH + "Raspi_hum.png",
24	URL_PATH + "Raspi_pressure.png"
25	]
26	#異常判定値
27	Judge = 36.0
28	
29	#Sets up pins as outputs
30	def setup_led():
31	GPIO.setmode(GPIO.BCM)
32	GPIO.setup(6, GPIO.OUT)
33	GPIO.output(6, GPIO.LOW)
34	
35	#異常検出対象となるデータ取得
36	def Raspi_temp():
37	global raspi_temp

Copyright © 2024 YITJC All Rights Reserved.

```
38      bme = BME280_Class.BME280()
39      bme.readData()
40      raspi_temp =    bme.getTemperature()
41
42  #温度データcsv記録
43  def Temp_csv():
44      date=datetime.datetime.today()
45      ary_temp=["0,0,0,0,0,0¥n" for i in range(MAX_RANGE)]
46      fp=open(URL_PATH+'Raspi_temp.csv',"r")
47      line=fp.readline()
48      while line:
49          ary_temp.pop(0)
50          ary_temp.append(line)
51          line=fp.readline()
52      ary_temp.pop(0)
53  ary_temp.append(str(date.year)+","+str(date.month)+","+str(date.day)+","+str(date.hour)+","+str(date.minute)+","+str(ras
    pi_temp)+"¥n")
54      fp.close()
55      fp=open(URL_PATH+'Raspi_temp.csv',"w")
56      fp.writelines(ary_temp)
57      fp.close()
58
59  #アラートメール用text作成
60  def read_temp():
61      #異常検出対象となるデータファイル読み込み
62      df_t = pd.read_csv(URL_PATH+'Raspi_temp.csv', names=['year', 'month', 'day', 'hour', 'minute', 'temperature'])
63      #データ記録終了時間の取得(末尾tail 1行のデータ取得)
64      end_time="%02d"%int(df_t.loc[49, 'month'])+"/"+"%02d"%int(df_t.loc[49, 'day'])+"_"+"%02d"%int(df_t.loc[49,
    'hour'])+":"+"%02d"%int(df_t.loc[49, 'minute'])
65      #緊急ツイート用text作成
66      global alert_text
67      text1 = str(now.year) + u"年" + str(now.month) + u"月" + str(now.day) + u"日" + str(now.hour) + u"時" +
    str(now.minute) + u"分!!"
68      text2 = '異常データ検出!!'+ '¥n'+'至急データを確認してください。'
69      text3 = '異常時検出温度:' + '{:.3f}'.format(df_t.loc[49, 'temperature']) + ' :' + end_time + 'に記録'
70      alert_text = text1 + '¥n'+ text2 + '¥n'+ text3
71      print (alert_text)
72
73  #Gmailアカウントの情報を環境編集から取得
```

Copyright © 2024 YITJC All Rights Reserved.

```
74  def get_gmail_settings():
75      smtp_server = os.getenv('SMTP_SERVER')
76      smtp_port = os.getenv('SMTP_PORT')
77      email = os.getenv('EMAIL')
78      app_password = os.getenv('APP_PASSWORD')
79      return smtp_server, smtp_port, email, app_password
80
81  #メール作成(添付ファイルは任意=None)
82  def create_email(from_email, to_email, subject, body, attachment_path=None):
83      msg = MIMEMultipart()                    #マルチパートメッセージを作成
84      #MIMEText(本文)
85      msg['From'] = from_email                 #送信元アドレスを設定
86      msg['To'] = to_email                     #送信先アドレスを設定
87      msg['Subject'] = subject                 #メールの件名を設定
88      msg.attach(MIMEText(body, 'plain')) #テキスト形式の本文を添付
89
90      # MIMEBase(添付ファイル)
91      if attachment_list:
92          for attachment_path in attachment_list:
93              try:
94                  maintype, subtype = 'image', 'png'
95                  #  添付ファイルを読み込み
96                  with open(attachment_path, 'rb') as attachment_file:
97                      attachment = MIMEBase(maintype, subtype)
98                      attachment.set_payload(attachment_file.read())
99                  #  添付ファイルをBase64形式にエンコード
100                 encoders.encode_base64(attachment)
101                 #  添付ファイルのヘッダーを設定
102                 attachment.add_header(
103                     'Content-Disposition',
104                     f'attachment; filename={os.path.basename(attachment_path)}'
105                 )
106                 #  添付ファイルをメールに添付
107                 msg.attach(attachment)
108             except Exception as e:
109                 print(f'Could not attach file {attachment_path}: {str(e)}')
110     return msg
111
112 #メール送信関数
```

Copyright © 2024 YITJC All Rights Reserved.

```
113 def send_email(from_email, to_email, msg):
114     #Gmailアカウントの情報を環境編集から取得
115     smtp_server, smtp_port, email, app_password = get_gmail_settings()
116 
117     server = smtplib.SMTP(smtp_server, smtp_port)     #SMTPサーバーに接続
118     server.starttls()                                   #SMTP通信をTLSで暗号化
119     server.login(email, app_password)                  #Gmailにログイン
120     server.sendmail(from_email, to_email, msg.as_string())   #メールを送信
121     server.quit()                                       #SMTPサーバーから切断
122 
123 #Alert mail送信
124 def alert_mail():
125     from_email = os.getenv('EMAIL')             #送信元メールアドレスを取得(環境変数に設定)
126     to_email = 'xxxxx@gmail.com'       #送信先メールアドレスを設定
127     subject = 'Alert Notification !!'                  #メールの件名を設定
128     body = alert_text                                  #メールの本文を設定
129     #attachment_path = attachment_paths
130     #メール作成
131     msg = create_email(from_email, to_email, subject, body, attachment_list)
132     #メール送信
133     send_email(from_email, to_email, msg)
134     print ('Alert mail has been sent!!')
135 
136 
137 if __name__ == '__main__':
138     try:
139         setup_led()
140         while True:
141             Raspi_temp()                    #データ監視
142             raspi_max = raspi_temp          #判定用データ
143             print ("Raspi_temp : %6.3f ℃" %raspi_max)          #デバッグ用（温度が判定値を超えた瞬間↓のプロ
    グラムが動くことを確認）
144             time.sleep(3.0)                 #データ監視間隔
145 
146             if(raspi_max >= Judge):
147                 now=datetime.datetime.today()    #現在時刻取得(=異常検出時刻)
148                 print (now)
149                 Temp_csv()                          #異常検出データcsv書き込み
150                 print ('Error Detect!!')
```

Copyright © 2024 YITJC All Rights Reserved.

```
151                     read_temp()                         #緊急ツイート用データ取得
152                     #全データをcsvファイルに書き込み
153                     #subprocessを使用して、40_CSV.pyを呼び出す
154                     subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/40_CSV.py'])
155                     #全データをグラフ化
156                     #subprocessを使用して、26_GRAPH.pyを呼び出す
157                     subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/26_GRAPH.py'])
158                     #グラフがすべて揃ったところで緊急ツイート
159                     alert_mail()                        #アラートメール（異常検出前後のデータ添付）
160                     GPIO.output(6, GPIO.HIGH)
161                     time.sleep(2.0)
162                     GPIO.output(6, GPIO.LOW)
163                     break
164         except KeyboardInterrupt:
165             pass
166         finally:
167             GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: BME280_Class モジュールをインポート（import）
- 5 行目: sleep 関数を使用するため time ライブラリをインポート
- 6 行目: 日付・時刻を取得するため datetime ライブラリをインポート
- 7 行目: csv ファイル読み書き用 csv ライブラリをインポート
- 8 行目: データ分析を行うため pandas ライブラリをインポート
- 9 行目: GPIO 制御を行うため RPi.GPIO ライブラリをインポート
- 10 行目: 別のプログラムやコマンドを実行するためのライブラリ「subprocess」をインポート
- 11 行目: SMTP プロトコルを使用してメールを送信するためのライブラリ「smtplib」をインポート
- 12 行目: OS と対話するためのライブラリ「os」をインポート
- 13 行目: MIME 形式のマルチパートメッセージを作成するためのクラスモジュールをインポート
- 14 行目: MIME 形式のテキストメッセージを作成するためのクラスモジュールをインポート
- 15 行目: MIME 形式のベースメッセージを作成するためのクラスモジュールをインポート
- 16 行目: MIME メッセージのエンコードを行うためのクラスモジュールをインポート
- 19 行目: BME280 センサデータを記録している csv ファイルのデータ数を定義
- 20 行目: メールに添付する画像ファイルが保存されているディレクトリを定義
- 21 行目～25 行目: メールに添付する画像ファイルを指定

Copyright © 2024 YITJC All Rights Reserved.

- 27 行目: **異常判定値を定義**
- 30 行目～33 行目: GPIO 端子初期設定プログラム setup_led()関数
  赤色 LED（GPIO6）を制御するため、GPIO 端子の初期設定を行う。
- 36 行目～40 行目: Raspi_temp データ取得プログラム Raspi_temp()関数
  異常監視対象となる Raspi_temp の温度データを取得
  参考プログラム： 9_BME280.py
- 43 行目～57 行目: Raspi_temp データを csv ファイルに書き込むプログラム Temp_csv()関数
  異常検出後、検出した異常値データを csv ファイルに書き込みます。
  参考プログラム： 40_CSV.py

～アラートメール用テキストデータ作成～

- 60 行目～77 行目: アラートメール用テキスト作成プログラム read_temp()関数
  異常検出後、アラートメール用の本文テキストを作成します。
- 62 行目: csv ファイルのデータを Pandas のデータフレーム df_t で定義
  ここでは、メッセージ作成対象のデータ「Raspi_temp.csv」を使用しています。
- 64 行目: 異常検出時刻を抽出（csv ファイル 50 行目のデータ抽出）
  異常検出後、異常値データは csv ファイルに書き込まれているため、csv ファイルの末尾に記録されているデータが検出された異常値です。
  ここでは、末尾に記録されている時刻を抽出し、異常検出時刻を表す変数「end_time」に格納しています。
- 66 行目: アラートメールの本文テキストデータを格納する「alert_text」をグローバル変数で定義
- 67 行目: 現在の日付・時刻取得（text1）
- 68 行目: アラートメッセージ（text2）
- 69 行目: 検出した異常値および検出時刻（text3）
- 70 行目: アラートメール用テキストデータ作成（alert_text）
- 71 行目: アラートメール用テキストデータを print 表示

～環境変数から Gmail アカウント情報取得～

- 82 行目～88 行目: 環境変数から Gmail アカウントの情報を取得するプログラム get_gmail_settings()関数
  参考プログラム： 38_Gmail.py

～メール作成～

- 91 行目～110 行目: メール作成プログラム create_email()関数
- 83 行目: マルチパートメッセージを作成
- 85 行目～88 行目: テキストメッセージ（MIMEText）定義
- 85 行目: 送信元メールアドレスを定義
- 86 行目: 送信先メールアドレス定義
- 87 行目: メール件名定義
- 88 行目: テキスト形式の本文をメールに添付
- 91 行目～109 行目: ベースメッセージ（MIMEBase）定義
- 91 行目: 添付ファイルの有無を判断
  添付ファイルがある場合は、「attachment_path」でファイルを指定

Copyright © 2024 YITJC All Rights Reserved.

- 92 行目: attachment_list で指定したファイルを 1 つずつ読み込みメールに添付
- 94 行目: 添付ファイルの MIME タイプ指定
- 96 行目～98 行目: 添付ファイルを読み込み
- 100 行目: 添付ファイルを Base64 形式にエンコード
- 102 行目～105 行目: 添付ファイルのヘッダーを設定
- 107 行目: 添付ファイルをメールに添付
- 108 行目～109 行目: 添付ファイルが見つからなかった場合の例外処理
- 110 行目: 他の関数で使用する変数(msg)を戻り値として返す

～メール送信～

- 113 行目～121 行目: メール送信プログラム send_email()関数

  参考プログラム: 38_Gmail.py

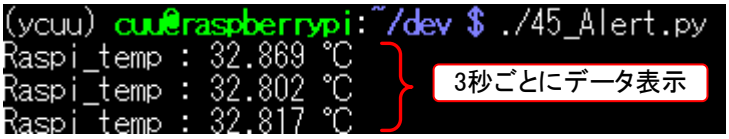
～Alert メール送信～

- 124 行目～134 行目: アラートメール送信プログラム alert_mail()関数
- 125 行目: 送信元メールアドレス取得
- 126 行目: **送信先メールアドレスを設定**
- 127 行目: メール件名を設定
- 128 行目: メール本文を設定
- 131 行目: メール作成
- 133 行目: メール送信
- 134 行目: アラートメール送信完了 print 表示

～メインプログラム～

- 137 行目～167 行目: main プログラム
- 138 行目～163 行目 :try ブロック
- 139 行目: GPIO 端子初期設定プログラム setup_led()関数実行
- 140 行目～163 行目: 異常検出プログラム(ループ処理)
- 141 行目: Raspi_temp データ取得プログラム Raspi_temp()関数実行
- 142 行目: 判定用データ「raspi_max」に測定データ「raspi_temp」を代入
- 143 行目: 判定用データ「raspi_max」を print 表示

  プログラムを実行すると 3 秒(異常監視間隔)ごとにデータが表示されます。

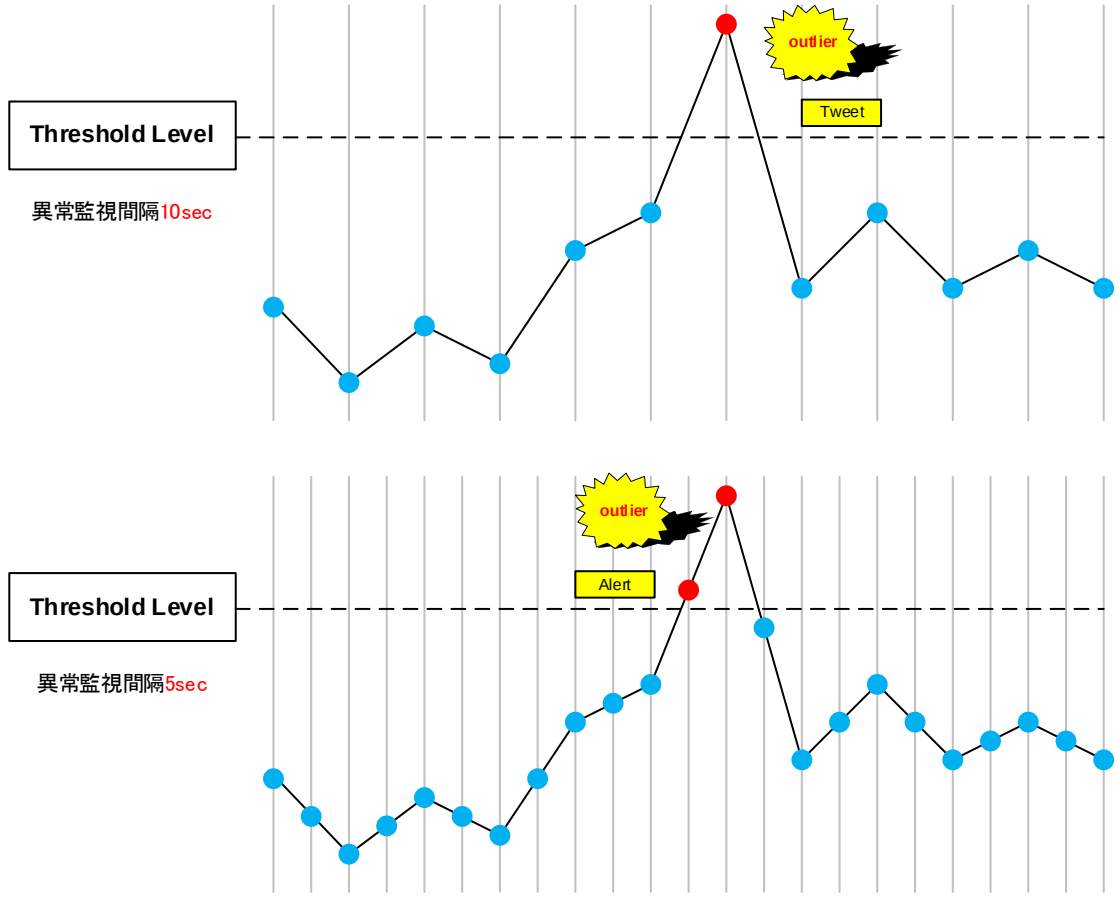


Copyright © 2024 YITJC All Rights Reserved.

- 144 行目: **データ監視間隔**

  異常監視間隔の時間設定により検出値に差が生じることを理解してください。

  次の図は、異常監視(判定)タイミングを縦線で表現しています。異常監視間隔で設定した時間の違いによって検出される値やタイミングが異なることが分かります。異常監視の時間間隔が短ければ短いほど、検出値は閾値(Threshold Level)に近づきます。



- 146 行目: データ判定
- 147 行目～163 行目: 異常検出時に実行する処理
- 147 行目: 現在時刻取得(=異常検出時刻)
- 148 行目: 異常検出時刻 print 表示
- 149 行目: 異常検出データ csv ファイルへ書き込み
- 150 行目: 異常検出 print 表示
- 151 行目: アラートメール用テキスト作成プログラム read_temp()関数実行
- 154 行目: subprocess ライブラリを使用して、「40_CSV.py」を実行
- 157 行目: subprocess ライブラリを使用して、「26_Graph.py」を実行

  異常監視対象以外のデータについても最新の状況を把握するため、データ記録・グラフ化処理を実行する。

  注意 構築した仮想環境にある Python(/home/cuu/ycuu/bin/python3.11)を指定してください
- 159 行目: アラートメール送信プログラム alert_mail()関数実行
- 160 行目～162 行目: 赤 LED を 2 秒間点灯
- 163 行目: ループ処理を抜ける

Copyright © 2024 YITJC All Rights Reserved.

- 164 行目～165 行目：except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。
- 166 行目～167 行目: finally ブロック
  例外の発生によらず必ず実行される処理を記述しています。
  本プログラムは GPIO を使用しているため、GPIO.cleanup()を記述しています。これにより、例外の発生によらず GPIO ピンのシステムクリアが必ず実行されます。

Copyright © 2024 YITJC All Rights Reserved.

ここからは、「アラート通知プログラム」を自動起動（常時稼働）するための手順について説明します。

手順④～Unit ファイル作成～

ここで作成するファイルは、「異常監視」および「アラート通知」を行うプログラム（45_Alert.py）を自動起動させるための Unit ファイル（.service ファイル）です。

まず、/etc/systemd/system ディレクトリへ移動します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

```
(ycuu) cuu@raspberrypi:~/dev $ cd ..
(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system
(ycuu) cuu@raspberrypi:/etc/systemd/system $
```

続いて、Unit ファイルを作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit ファイルの中身を書き込んでください。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano Error_Detect.service

～参考～

Unit ファイルの「上書き」、「終了」は、次の手順を参考にしてください。

- 編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認
- 「Enter」キーを押下して上書き保存を実行
- 「Ctrl + X」キーを押して元の画面に戻る

～Unit ファイル解説～

Unit ファイルの中身について解説します。

行	Unit ファイルの中身
1	[Unit]
2	Description=Outlier Detect
3	After=network.target
4	
5	[Service]

Copyright © 2024 YITJC All Rights Reserved.

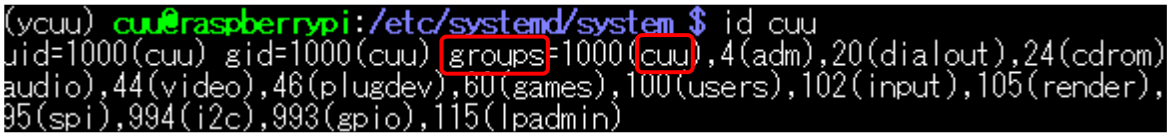
```
6  ExecStart=/home/cuu/dev/45_Alert.py
7  Restart=always
8  RestartSec=5sec
9  Type=simple
10 WorkingDirectory=/home/cuu/dev
11 User=cuu
12 Group=cuu
13 #App Password for Google
14 Environment="SMTP_SERVER=smtp.gmail.com"
15 Environment="SMTP_PORT=587"
16 Environment="EMAIL=cuu.yamanashi@gmail.com"
17 Environment="APP_PASSWORD=elec xxxx xxxx xwqx"
18
19 [Install]
20 WantedBy=multi-user.target
```

灰色部は、これまでに説明した内容です。

- 1 行目～3 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文（動作に影響しませんが、分かりやすく記述するようにしてください。）
- 3 行目: 起動順序の指定
  After=network.target
- 5 行目～17 行目: Service セクション
  起動するプログラム、起動条件など Unit ファイルの中で最も設定項目が多いセクションです。
  サービスの内容（どんな処理を実行させるか）を記述するセクションです。
- 6 行目: 自動起動するプログラムを指定 （45_Alert.py）
- 7 行目: プロセスが停止した場合、プロセスの再起動条件を設定
  Always: 常に再起動
- 8 行目: 再起動するまでの待ち時間（秒）を設定
  ～補足～
  異常検出後、「45_Alert.py」はアラートメールを送信してプログラムを終了します。
  再び、異常監視プログラム「45_Alert.py」を稼働させるため、Always（常に再起動）設定にしています。
- 9 行目: サービスプロセスの起動完了の判定方法
  デフォルト設定値「Simple」に設定
  Simple は、コマンドを実行したタイミングで起動完了と判断します。
- 10 行目: WorkingDirectory を指定
  ～補足～
  「45_Alert.py」プログラムは、他のプログラム「40_CSV.py」や「26_GRAPH.py」を呼び出して実行します。
  他のファイルを参照してプログラムを実行する場合は WorkingDirectory を必ず設定してください。

Copyright © 2024 YITJC All Rights Reserved.

- 11 行目～12 行目: 実行時の User, Group を指定
  ユーザー（User）及びユーザーが所属しているグループ（Group）は、次のコマンドで確認することできます。
  id のあとに本講座で使用しているユーザー名「cuu」を入力してください。
  (ycuu) cuu@raspberrypi:/etc/systemd/system $ id cuu



- 14 行目～17 行目: Gmail アカウントの情報を環境変数として設定
  ～補足～
  Gmail アカウントの情報は、ホームディレクトリ（/home/cuu）にある「.bashrc」で環境変数に設定しています。しかし、systemd のサービスは「.bashrc」ファイルを読み込むことができないため、Unit ファイルで環境変数を定義する必要があります。
- 19 行目～20 行目: Install セクション
  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 20 行目: サービス Unit の依存関係を設定
  multi-user.target は、複数ユーザー環境のベースとなるターゲットで、多くのサービスがこのターゲットに紐付いて起動されます。（shutdown サービス Unit もこのターゲットに紐付けます）
  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

手順⑤～サービス再読み込み～

手順④で Unit ファイルを作成した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl daemon-reload

手順⑥～サービス自動起動有効化～

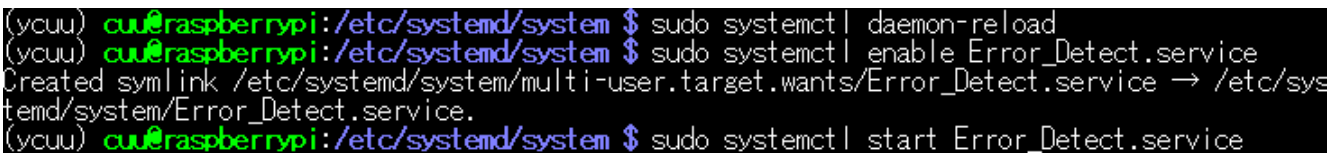
次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl enable Error_Detect.service

手順⑦～サービス開始～

次のコマンドを実行してサービスを開始させます。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl start Error_Detect.service



Copyright © 2024 YITJC All Rights Reserved.

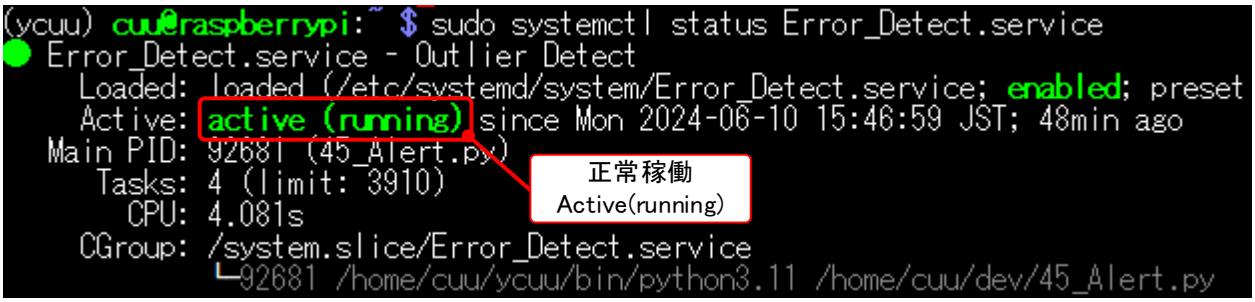
手順⑧～サービス動作状況確認～

次のコマンドを実行してサービスの動作状況を確認します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl status Error_Detect.service

正常に稼働（自動起動）していれば、次の画面のようにステータスが active（runnning）と表示されます。

この画面表示を終了する場合は、「q」を押してください。

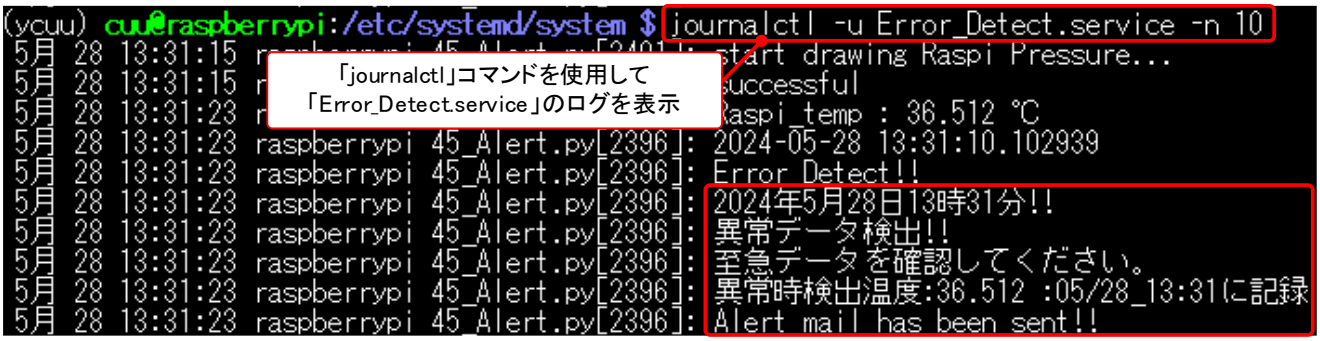


手順⑨～IoT システム ver4 動作確認～

「Raspi_temp」の温度を上昇させて、アラートメールが送信されることを確認してください。

システムのログから IoT システム ver4 の動作を確認する場合は、次のコマンドを実行してシステムのログを確認してください。

(ycuu) cuu @raspberrypi:~ $ journalctl -u Error_Detect.service -n 20



～補足～

今後は常に「Raspi_temp」の温度データが監視され、温度データ閾値を超えて異常と判定された場合は、アラートメールが送信されます。IoT システム ver4（温度監視）を停止させたい方は、次のコマンドを実行してサービスを無効にしてください。

次のコマンドを実行してサービスを停止します。

(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl stop Error_Detect.service

次のコマンドを実行してサービスの自動起動を無効にします。

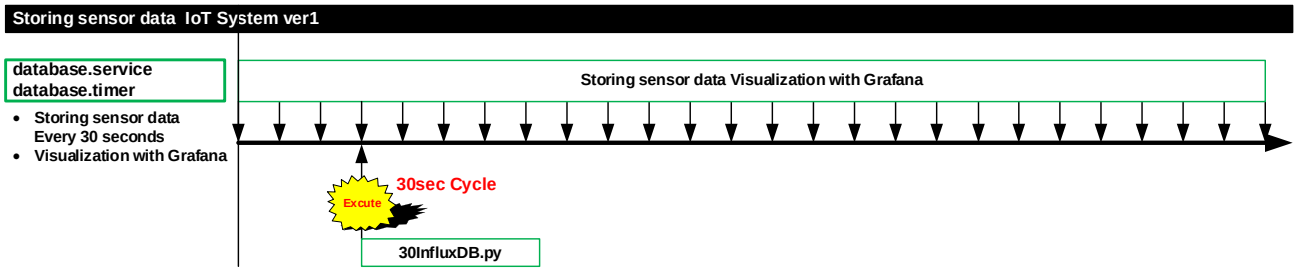
(ycuu) cuu@raspberrypi: /etc/systemd/system $ sudo systemctl disable Error_Detect.service

Copyright © 2024 YITJC All Rights Reserved.

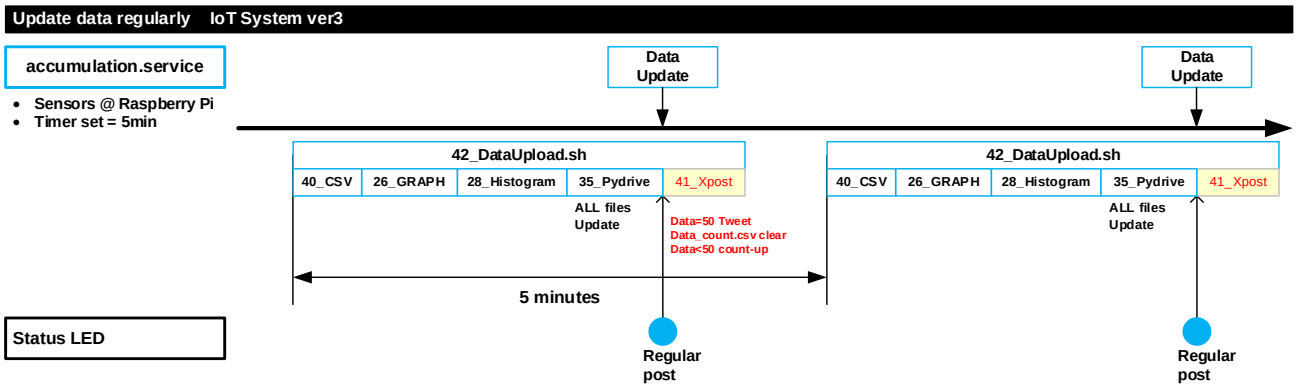
参考～IoT システム稼働状況～

現在稼働している IoT システムは、ver1 と ver3、ver4 の 3 つのサービスが稼働しています。

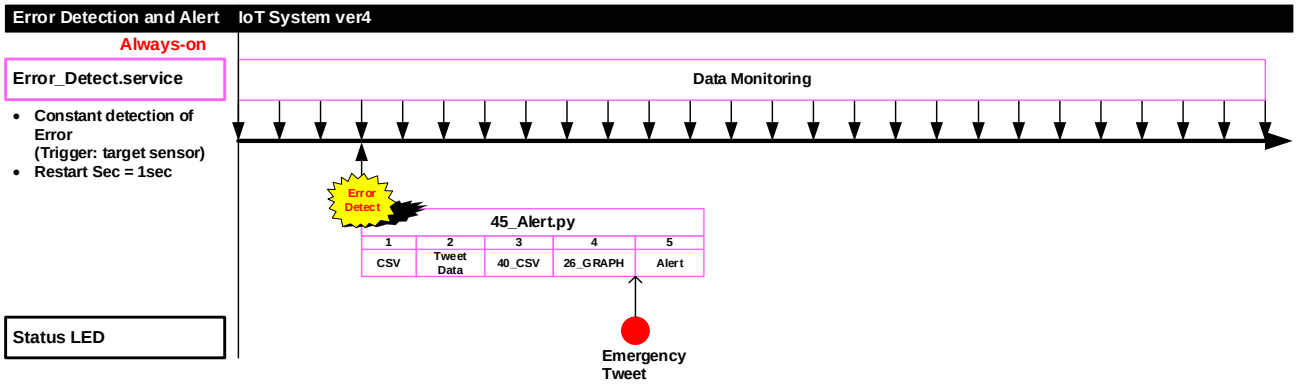
IoT システム ver1 は、30 毎にデータを取得し、取得したデータをデータベースに蓄積・可視化しています。



IoT システム ver2 は、5 分毎にデータを取得し、センサ情報とグラフデータを定期ポストしています。



IoT システム ver4 は、センサデータを常時監視し、異常を検出すると直ちにアラートメールを送信します。



Copyright © 2024 YITJC All Rights Reserved.

参考～IoT システム適用例～

これまでに構築した「IoT システム」の適用例を紹介します。

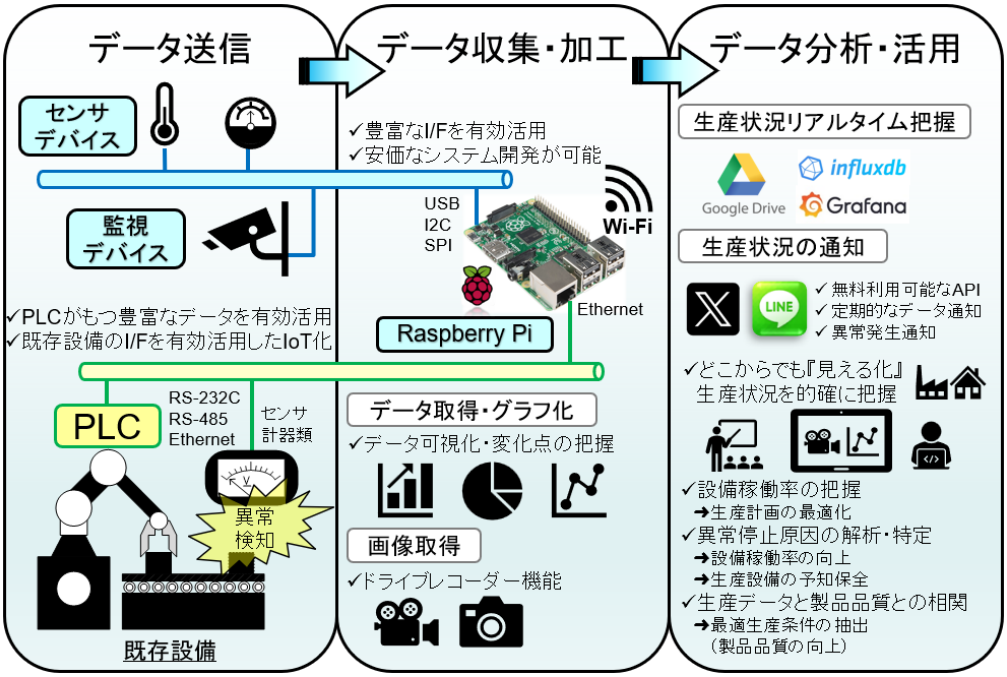
現在稼働している IoT System を既存の生産設備に導入した場合に期待される効果を紹介します。

IoT System ver1、ver3

- 生産状況のリアルタイム更新
  - ✓ 遠隔地から「いつでも」生産状況を把握
    - ➔ 定期ツイートだけでなく、Google Drive 上には 5 分毎に最新のデータが更新されるため、リアルタイムに生産状況を把握
  - ✓ 可視化（グラフ化）されたデータにより生産状況や「変化」を的確に把握
  - ✓ PLC がもつ有効な生産データにより設備稼働率を正確に把握し、生産計画および人員配置を最適化
  - ✓ 生産データと製品品質との相関を分析することにより、最適な生産条件を抽出して製品品質を向上

IoT System ver4

- 生産状況の通知
  - ✓ 遠隔地から「いつでも」生産設備の異常の発生を的確に把握
  - ✓ 異常検出後直ちに生産設備を緊急停止制御（Raspberry Pi から直接 PLC を制御可能）
  - ✓ 遠隔地から現場作業員へ「迅速」かつ「的確」な復旧指示
    - ➔ 生産設備の早期復旧、現場作業者の負担軽減
  - ✓ 異常停止前後の生産データ・センサデータ・画像データを分析することにより、異常停止の原因特定に掛かる時間を削減
    - ➔ 「予知保全」を実現し、ダウンタイム（生産設備停止時間）を最小限に抑えて設備稼働率を向上



Copyright © 2024 YITJC All Rights Reserved.

参考～生産設備の保全方法～

本講座で構築した「IoT システム」の活用により、設備保全業務の改善が期待できます。具体的には、従来から実施している「事後保全」や「予防保全」の業務を「IoT システム」を活用にして「予知保全」にステップアップすることで保全業務の改善が期待できます。

「事後保全」…

生産設備が故障、または不具合が生じた時に、その現象に応じて補修や部品の交換を実施する保全方法です。突発的な故障が発生した際、交換部品が手元になく、稼働停止時間が長時間になると損失が非常に大きくなるという課題があります。

「予防保全」…

生産設備の状態にかかわらず、定期的に点検、修理、部品交換などのメンテナンスを実施する保全方法です。部品交換を一定期間で交換する「時間計画保全」(TBM: Time Based Maintenance)と、部品の劣化具合に応じて交換する「状態監視保全」(CBM: Condition Based Maintenance)があります。まだ寿命に達していない部品を交換するケースがあるため、定期的な保全には無駄が多いという課題があります。

「予知保全」…

生産設備の IoT 化に伴い最も注目を集めている保全方法であり、生産設備の不具合や故障を「生産データ」や「センサデータ」によりあらかじめ「予知」し、生産設備を最適な状態に管理できます。

最適なタイミングで部品交換や補修を行うことができるため、無駄が少なくメンテナンスコストを抑えることが可能です。具体的な利点として次の項目が挙げられます。

- ✓ 保全作業者の経験やスキルを問わない保全活動が可能
- ✓ 交換部品の在庫を効率的に管理できる
- ✓ 部品交換時期を的確に判断することができる
- ✓ ダウンタイム(生産設備停止時間)を最小限に抑えて設備稼働率を向上

Copyright © 2024 YITJC All Rights Reserved.

9.6 Google Cloud Vision～利用環境整備～

「5 章 画像データ取得」で紹介した「Google Cloud Vision」を Python プログラムで使用する方法について紹介します。Google Cloud Vision を利用するためには「Cloud Vision API」を有効に設定する必要がありますが、この作業は「8 章 オンラインストレージ活用」で完了しています。また、Google Cloud Vision を利用するために必要な「サービスアカウントキー」は、「8 章 オンラインストレージ活用」で取得したキーを使用します。

Google Cloud Vision の設定に関する情報は、次の URL を参考にしてください。

[URL] https://cloud.google.com/vision/docs/setup?hl=ja

手順①～支払い情報登録～

Google Cloud Vison は「従量課金制度」を採用しています。ただし、毎月 1,000 ユニットまでは無料で利用することができます。Cloud Vision API の利用料金は、次の公式サイトで確認できます。

[URL] https://cloud.google.com/vision/pricing?hl=ja

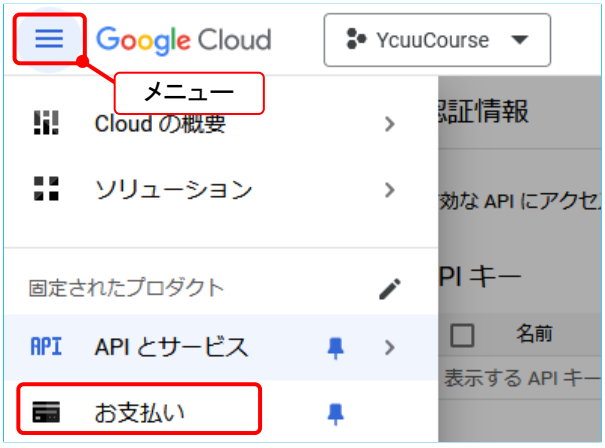
特徴	1,000 ユニットあたりの料金		
	最初の 1,000 ユニット/月	1,001～5,000,000 ユニット/月	5,000,001 以上のユニット/月
ラベル検出	無料	$1.50	$1.00
テキスト検出	無料	$1.50	$0.60
ドキュメント テキスト検出	無料	$1.50	$0.60
セーフサーチ（不適切なコンテンツ）検出	無料	$1.50（ラベル検出を利用している場合は無料）	$0.60（ラベル検出を利用している場合は無料）
顔検出	無料	$1.50	$0.60
顔検出 - Celebrity Recognition	無料	$1.50	$0.60
ランドマーク検出	無料	$1.50	$0.60
ロゴ検出	無料	$1.50	$0.60
画像プロパティ検出	無料	$1.50	$0.60
クロップヒント検出	無料	$1.50（画像プロパティ検出を利用している場合は無料）	$0.60（画像プロパティ検出を利用している場合は無料）
ウェブ検出	無料	$3.50	Google にお問い合わせください
オブジェクトのローカライズ	無料	$2.25	$1.50

次の URL をクリックし、Google アカウントでログインしてください。

[URL] https://console.developers.google.com/

Copyright © 2024 YITJC All Rights Reserved.

画面左上の「メニュー」をクリックし、「お支払」をクリックしてください。



「請求先アカウントをリンク」をクリックしてください。

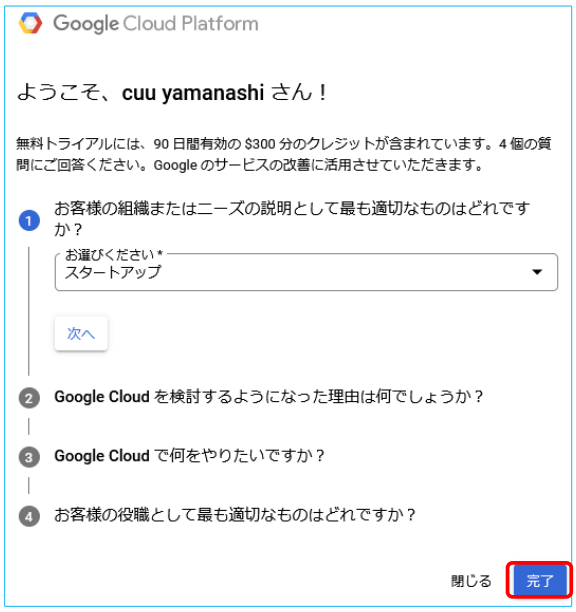
「請求先アカウントを作成」をクリックしてください。

「同意して続行」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「お支払プロファイル」と「お支払方法」を入力したあと、「無料トライアルを開始」をクリックしてください。

最後にアンケートに回答したあと、「完了」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

手順②～google-cloud-vision インストール～

Python プログラムを使用して Google Cloud Vision を利用するために必要なライブラリ「google-cloud-vision」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install google-cloud-vision

～補足～

google-cloud-vision と同時に依存関係のあるパッケージもインストールされます。gRPC（Google Remote Procedure Call）関連のライブラリがインストールされます。

手順③～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

46_CloudVision.py

[プログラム内容]

プログラムを実行すると、画像ファイルに写っているものを識別して結果をテキストで表示します。

手順④～ディレクトリ作成・画像保存～

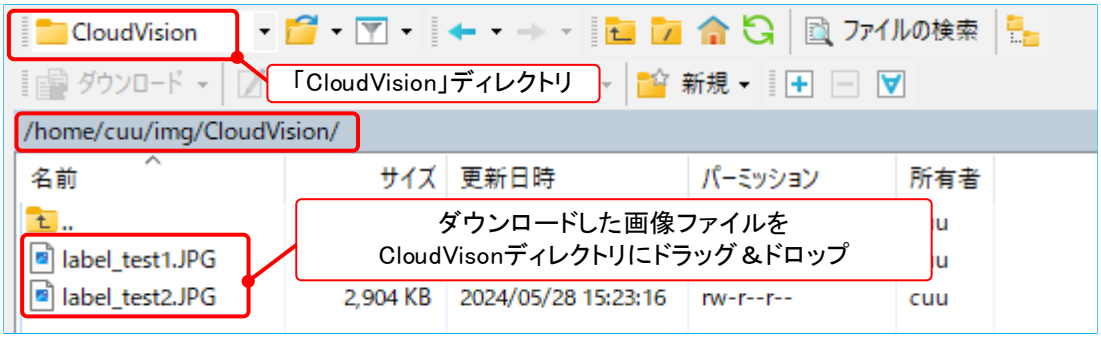
Google Cloud Vision のテスト用画像を保存するためのディレクトリ「CloudVision」を作成します。

このディレクトリを作成するため、「img」ディレクトリに移動した後、「CloudVision」ディレクトリを作成します。

(ycuu) cuu@raspberrypi:$ cd img

(ycuu) cuu@raspberrypi:~/img $ mkdir CloudVision

共有サイトにある画像データ「label_test1.jpg」および「label_test2.jpg」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「CloudVision」ディレクトリに保存してください。



Copyright © 2024 YITJC All Rights Reserved.

Google Cloud Vision を使用して次の画像に写っているものを識別します。

[画像ファイル]

label_test1.jpg（パンダの画像）

label_test2.jpg（アルパカの画像）

手順⑤～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/img $ cd ..
(ycuu) cuu@raspberrypi:$ cd dev
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 46_CloudVision.py
```

手順⑥～プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./46_CloudVision.py
```

「label_test1」画像は、植物（Plant）とパンダ（panda）のスコアが高い結果になっています。

「label_test2」画像は、脊椎動物（Vertebrate）と哺乳類（Mammal）のスコアが高い結果になっています。

```
(ycuu) cuu@raspberrypi:~/dev $ ./46_CloudVision.py
Labels for /home/cuu/img/CloudVision/label_test1.JPG:
Plant: 0.9337413311004639
Panda: 0.9287206530570984
Carnivore: 0.859151303768158
Organism: 0.8565475940704346
Biome: 0.8282283544540405
Terrestrial animal: 0.7972740530967712
Flower: 0.782882809638977
Groundcover: 0.7392410039901733
Natural material: 0.7363053560256958
Snout: 0.734133243560791
Labels for /home/cuu/img/CloudVision/label_test2.JPG:
Vertebrate: 0.9191992878913879
Mammal: 0.8587700724601746
Road surface: 0.8562642335891724
Yellow: 0.8480928540229797
Asphalt: 0.8299442529678345
Fawn: 0.8153670430183411
Hat: 0.7531706690788269
Dog breed: 0.7514606714248657
Companion dog: 0.7318578958511353
Human leg: 0.6969577074050903
```

画像に写っているものを識別するケースはあまりないと思いますので、プログラムの解説は省略します。

以上の手順で、Google Cloud Vision の利用環境が整備できました。

Copyright © 2024 YITJC All Rights Reserved.

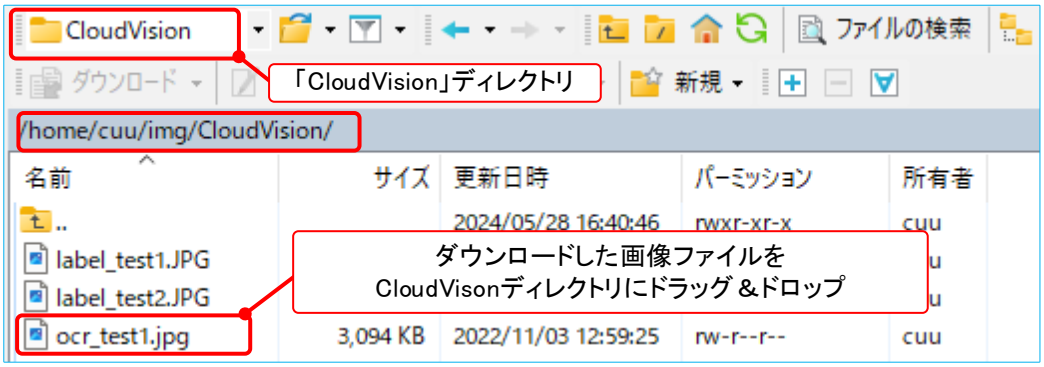
9.7 Google Cloud Vision～OCR 機能活用～

Google Cloud Vision の OCR 機能を「5 章 画像データ取得」でテストした結果、画像内の数字データを抽出・識別する精度が OCR ソフトウェア「Tesseract」と比べて高いことがわかりました。

ここでは、「Google Cloud Vision」を Python プログラムで使用して「OCR 機能」を利用する方法について学習します。

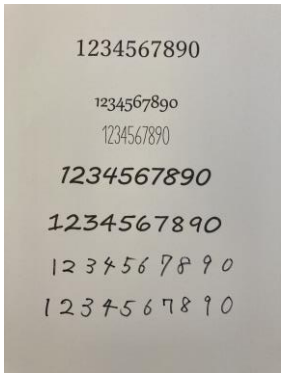
手順①～画像保存～

共有サイトにある画像データ「ocr_test1.jpg」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「CloudVision」ディレクトリに保存してください。



[画像ファイル]

ocr_test1.jpg　（手書きを含む 7 種類の数字画像）※下 2 行は、手書き数字



手順②～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

47_CloudVision.py

手順③～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

```
(ycuu) cuu@raspberrypi:~/dev $ chmod +x 47_CloudVision.py
```

Copyright © 2024 YITJC All Rights Reserved.

手順④～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./47_CloudVision.py

プログラムを実行すると、次の画面が表示されます。手書きを含む 7 種類の数字をすべて正確に抽出・識別できています。

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import io
5	import os
6	from google.cloud import vision
7	from google.oauth2 import service_account
8	
9	#Service Account Key(jsonファイル)読み込み
10	#devディレクトリにservice_account_key.jsonファイルを保存
11	credentials = service_account.Credentials.from_service_account_file('service_account_key.json')
12	client = vision.ImageAnnotatorClient(credentials=credentials)
13	
14	# 画像ファイル(OCR対象)指定
15	file_name = os.path.abspath('/home/cuu/img/CloudVision/ocr_test1.jpg')
16	
17	# Loads the image into memory
18	with io.open(file_name, 'rb') as image_file:
19	content = image_file.read()
20	
21	image = vision.Image(content=content)
22	
23	# Text Detetion (OCR) on the image file
24	response = client.document_text_detection(

Copyright © 2024 YITJC All Rights Reserved.

```
25              image=image,
26              image_context={'language_hints': ['en']}
27          )
28
29  # レスポンスからテキストデータを抽出
30  output_text = ''
31  for page in response.full_text_annotation.pages:
32          for block in page.blocks:
33              for paragraph in block.paragraphs:
34                  for word in paragraph.words:
35                      output_text += ''.join([
36                          symbol.text for symbol in word.symbols
37                      ])
38                  output_text += '¥n'
39  print(output_text)
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: 画像ファイルをバイナリデータとして読み込むために必要なライブラリ「io」をインポート
- 5 行目: OS と対話するためのライブラリ「os」をインポート
- 6 行目: Cloud Vision API を使用するため google.cloud クラスから「vision」モジュールをインポート
- 7 行目: OAuth2 認証するため google.oauth2 クラスから「service_account」モジュールをインポート
- 11 行目: サービスアカウントのキーファイル（service_account_key.json）を指定して読み込み
- 12 行目: サービスアカウントのキーワイルを使用して Client をインスタンス（実体）化
- 15 行目: OCR テストを実施する画像ファイルを指定
- 18～19 行目: 「io」ライブラリを使用して画像ファイルを rb（read binary）モードで読み込み、image_file に格納データとして読み込み
- 19 行目: image_file のデータ（バイナリデータとして読み込んだ画像のデータ）を content に格納
- 21 行目: Cloud Vision API に画像データを渡すためのクラス vision.Image()を使用して image をインスタンス（実体）化

Copyright © 2024 YITJC All Rights Reserved.

- 24～27 行目: 機能選択

  response = client.document_text_detection

  今回使用した機能は「document_text_detection」です。

機能	説明
type_unspecified	機能タイプを指定しない
face_detection	顔検出
landmark_detection	ランドマーク検出
logo_detection	ロゴ検出
label_detection	ラベル検出
text_detection	単純なテキスト検出・光学式文字認識（OCR）を実行 テキスト検出は、より大きな画像のテキスト領域用に最適化されています。 画像がドキュメントの場合は、代わりに「document_text_detection」を使用します。
document_text_detection	高密度テキストドキュメントの OCR を実行
safe_search_detection	セーフサーチを実行 安全でない可能性のあるコンテンツや不適切なコンテンツを検出

- 26 行目: image_context 画像解析に必要な情報を追加
  - 文字解析の言語指定

    image_context={'language_hints': ['en']}

    en: 英語

    ja: 日本語

    対象となる言語は、次のリンク先から確認できます。

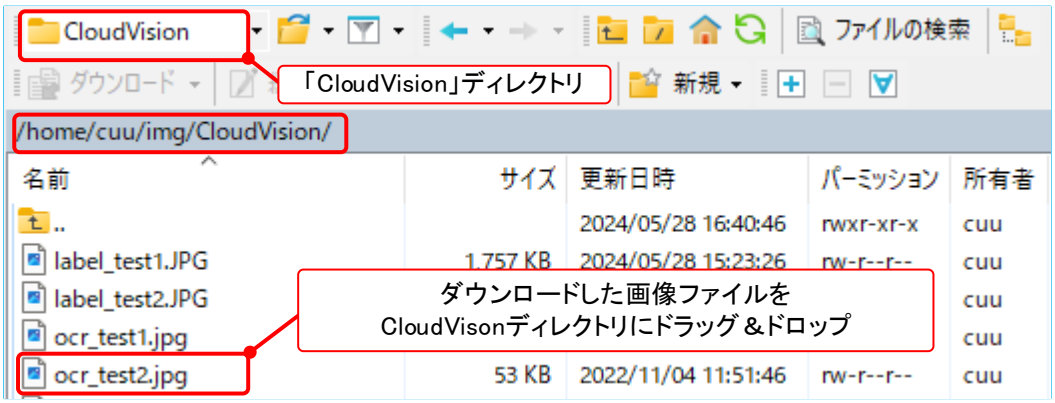
    https://cloud.google.com/vision/docs/languages
- 30 行～39 行: OCR の結果からテキストデータを抽出し、結果を Print 表示
- 31 行目: Cloud Vision API のレスポンスからテキストが含まれるすべてのページにアクセス
- 32 行目: 各ページのテキストブロックにアクセス
- 33 行目: 各ブロックの段落にアクセス
- 34 行目: 各段落の単語にアクセス
- 35 行目～37 行目: 各単語のシンボル（文字・数字）アクセスし、そのテキストを取得

  各単語のシンボルを連結して単語を生成（ourput_text）
- 38 行目～39 行目: ブロック抽出した単語に改行を加えて print 表示

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～7 セグメント LED 画像保存～

共有サイトにある画像データ「ocr_test2.jpg」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「CloudVision」ディレクトリに保存してください。



[画像ファイル]

ocr_test2.jpg（7 セグメント LED 画像）

手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

48_CloudVision.py

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 48_CloudVision.py

手順⑧～プログラム実行～

対象プログラムを実行します。

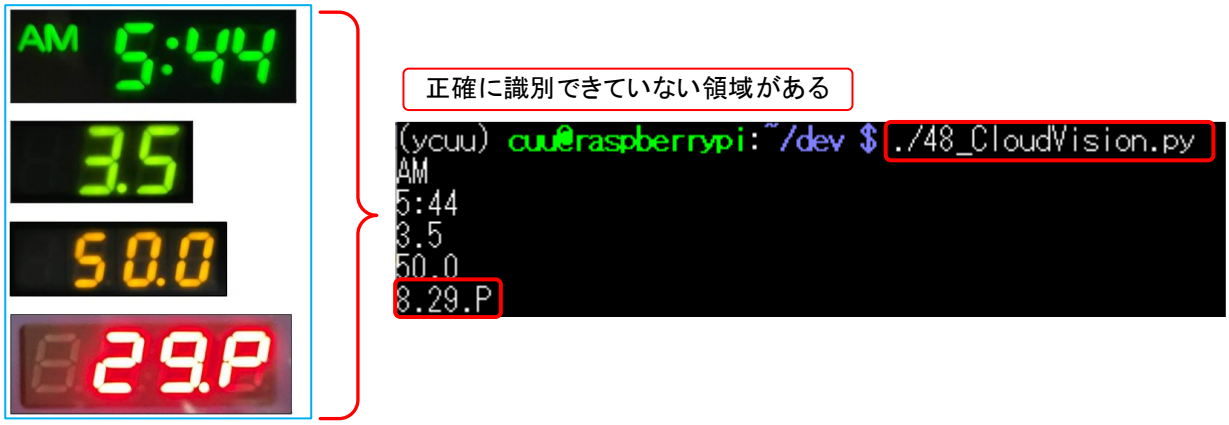
(ycuu) cuu@raspberrypi:~/dev $ ./48_CloudVision.py

プログラムを実行すると、次の画面が表示されます。

Copyright © 2024 YITJC All Rights Reserved.

4 種類の 7 セグメント LED 表示器の識別結果が表示されます。

左の画像と比較すると、数値の認識が正確に行われていない箇所があることが分かります。



「ocr_test2.jpg」ファイルに含まれる次の 7 セグメント LED 表示器は、数値の認識が正確に行われませんでした。「5 章　画像データ取得」で学習したとおり、数値認識の精度を向上させるためには、適切な画像処理が必要になります。

この画像対してさまざまな画像処理を実施し、画像処理された画像ファイルに対して Google Cloud Vision による OCR を実施した結果を紹介します。

画像ファイル	OCR 使用画像	変換結果
ocr_test2 元画像		
ocr_test2_bw 白黒 2 値化		
ocr_test2_bw_erosion 収縮処理		
ocr_test2_bw_dilation 膨張処理		

Copyright © 2024 YITJC All Rights Reserved.

画像ファイル	OCR 使用画像	変換結果
ocr_test2_bw_opening Opening 処理		
ocr_test2_bw_closing Closing 処理		
ocr_test2_bw_morphology Gradient（勾配）処理		
ocr_test2_bw_bwinvert 白黒反転処理		

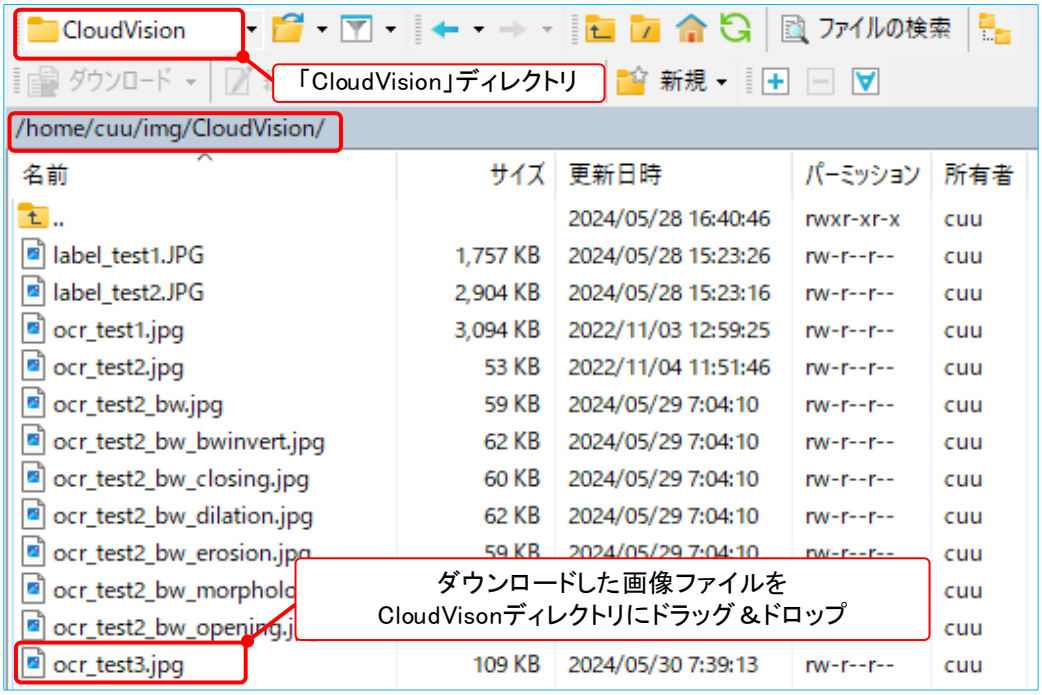
生産設備や工場の設備で使用されている表示器には、明るさや表示色など、さまざまな種類の 7 セグメント LED が使用されています。画像処理の内容によって Google Cloud Vision の領域抽出および数字の識別精度が変わるため、実際に使用する場合は、対象となる 7 セグメント LED 表示器の画像に対して最適な画像処理の組み合わせを選択してください。

※）画像処理に使用したプログラム「49_OpenCV.py」は、共有サイトに公開しておきます。

Copyright © 2024 YITJC All Rights Reserved.

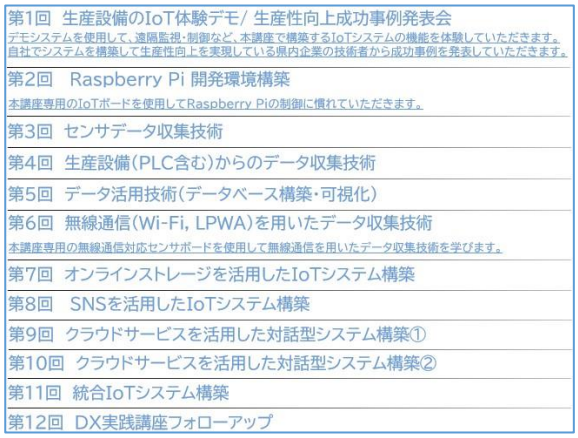
手順⑨～日本語画像保存～

共有サイトにある画像データ「ocr_test3.jpg」を Windows PC にダウンロードした後、WinSCP を使用して（ドラッグ＆ドロップ）、「CloudVision」ディレクトリに保存してください。



[画像ファイル]

ocr_test3.jpg（日本語ドキュメント画像）



手順⑩～プログラム配編集～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

48_CloudVision.py

[プログラム編集個所]

```
#15行目 画像ファイル(OCR対象)指定
file_name = os.path.abspath('/home/cuu/img/CloudVision/ocr_test3.jpg')
#16行目 言語指定(jp: 日本語)
image_context={'language_hints': ['jp']}
```

Copyright © 2024 YITJC All Rights Reserved.

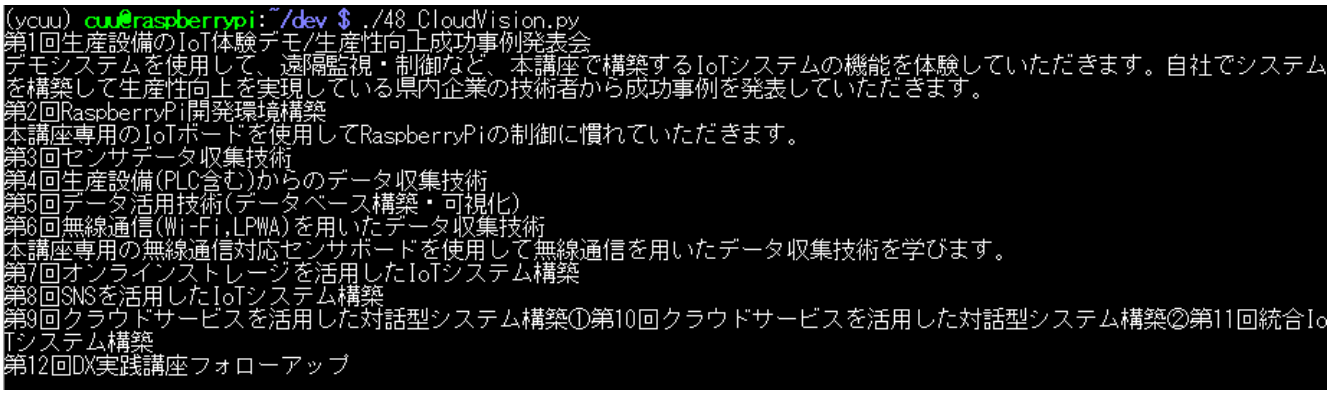
手順⑪～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./48_CloudVision.py

プログラムを実行すると、次の画面が表示されます。

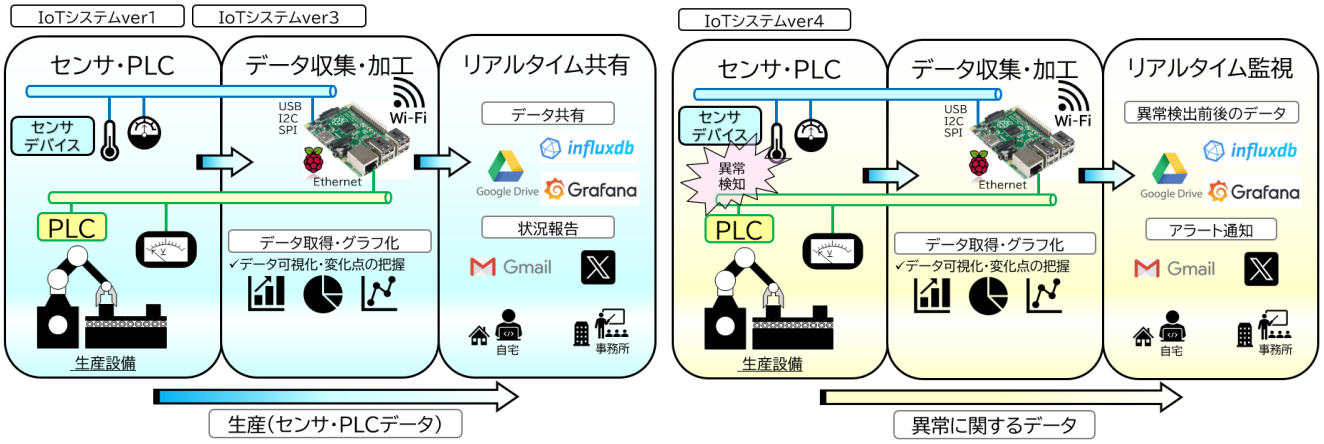
日本語ドキュメントにはさまざまな文字（漢字、英数字、ひらがな）が含まれていますが、文字領域の抽出および文字認識が正確に行われていることがわかります。このため、Google Cloud Vision を活用すれば、帳票類のデジタルデータ化も実現できると考えています。



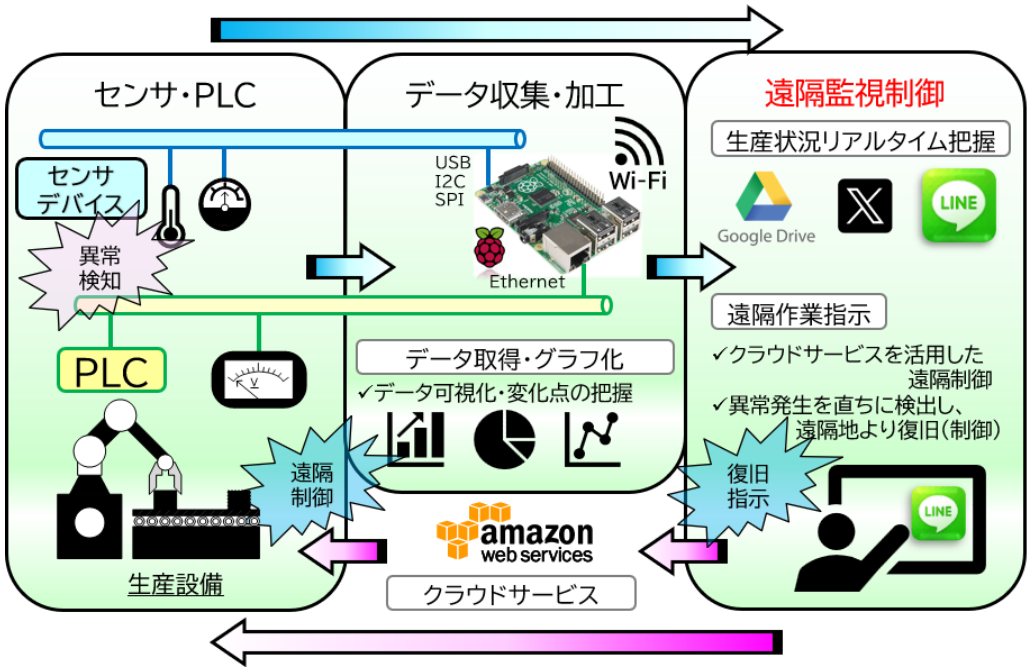
Copyright © 2024 YITJC All Rights Reserved.

第 10 章 クラウドサービスの活用～対話型システム～

現在稼働している IoT システムは、「ダッシュボード」や「オンラインストレージ」、「通知機能」を利用して、生産現場の状況をリアルタイムに把握できます。しかし、いずれのシステムもデータの流れが「一方向」で構成されているため、従業員が生産設備に対してデータをリクエストする機能がありません。そのため、従業員が生産設備に対して何らかのアクションを実行するためには、わざわざ生産現場に赴かなければなりません。この問題を解決するためには、双方向（インタラクティブ: Interactive）通信が可能な「**対話型システム**」を構築する必要があります。



「対話型システム」は、生産設備（IoT システム）と従業員（LINE ユーザー）との間で双方向コミュニケーションを可能にします。この対話型システムを構築するために、LINE Messaging API とクラウドサービスを活用します。LINE Messaging API は LINE が無料で公開している API です。クラウドサービスには AWS（Amazon Web Service）を使用します。AWS（クラウド上）に対話機能の中心となる Web アプリケーションを配置し、対話型システムを構築します。



Copyright © 2024 YITJC All Rights Reserved.

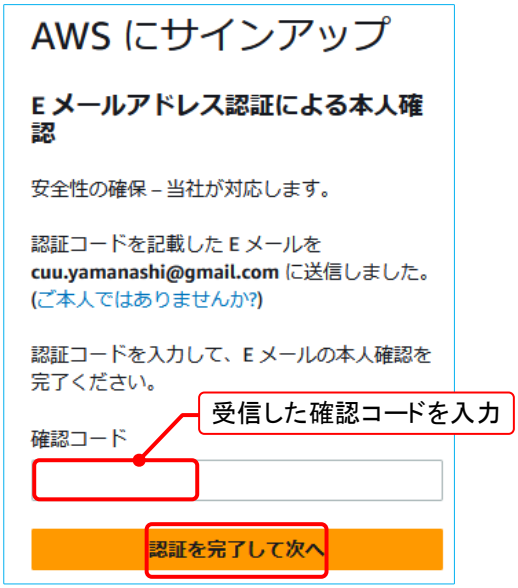
10.1 AWS アカウント取得

次のリンクをクリックして「AWS（Amazon Web Service）」にアクセスしてください。

[URL] https://portal.aws.amazon.com/billing/signup#/start/email

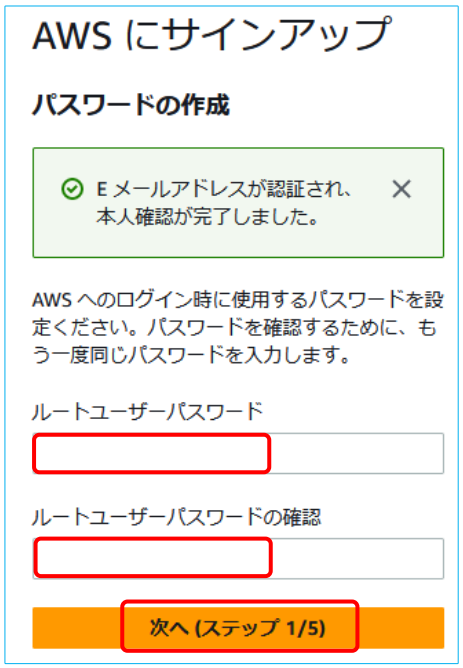
「E メールアドレス」、「AWS アカウント名」を入力し、「認証コードを E メールアドレスに送信」をクリックしてください。

受信した確認コードを入力したあと、「認証を完了して次へ」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「パスワード」を入力したあと、「次へ」をクリックしてください。



続いて、

✓ 連絡先登録

✓ クレジットカード情報登録（※無料枠の利用でも登録が必要になります。）

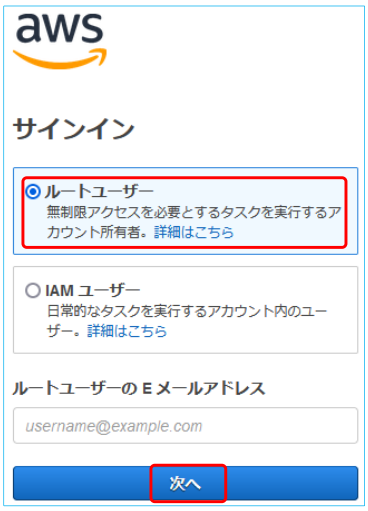
✓ 本人確認

を実施したあと、「ベーシックサポート（**無料**）」を選択し、「サインアップを完了」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

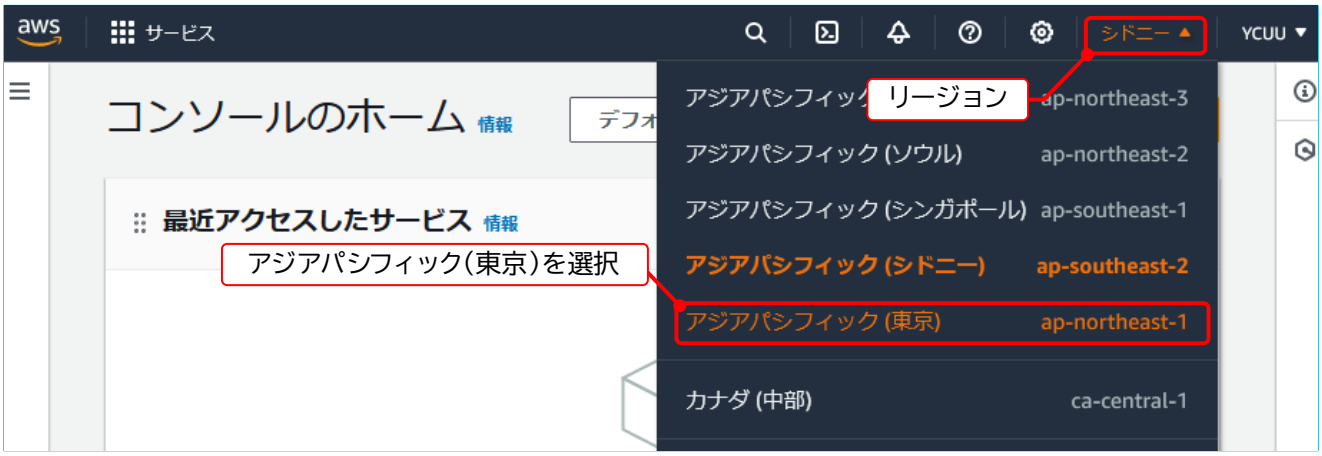
サインアップが完了したら、「AWS マネジメントコンソールにお進みください」をクリックしてください。

「ルートユーザー（管理者）」を選択し、E メールアドレスを入力したあと、「次へ」をクリックしてください。



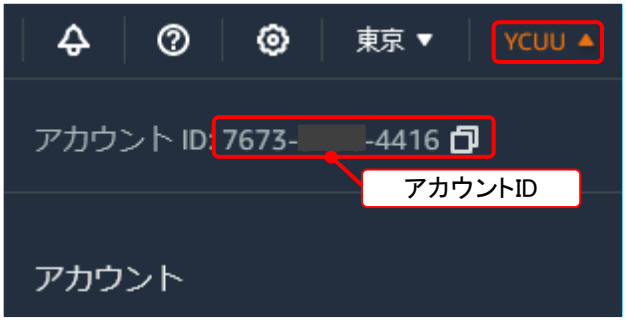
ログインすると、次の画面が表示されます。

AWS は世界各地にデータセンター（サーバー）を保有しています。データセンターは、リージョン（地域）という単位で地理的に分割して管理されています。画面右上に現在選択しているリージョンが表示されていますが、サービスを利用する場合は、地理的に近いリージョンを選択するようにしてください。



Copyright © 2024 YITJC All Rights Reserved.

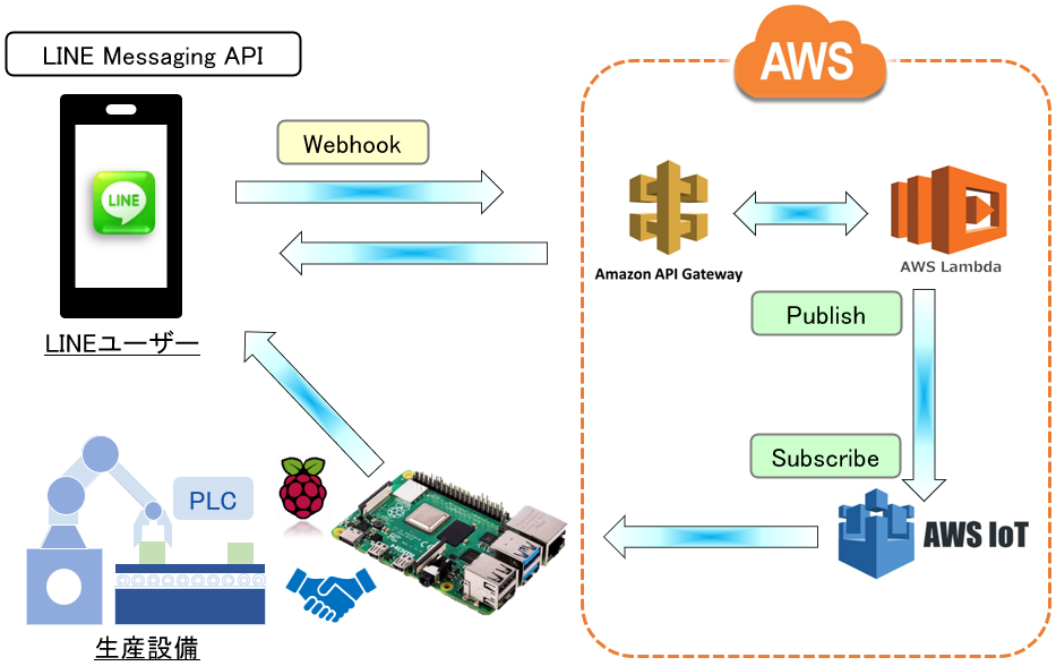
最後に、画面右上の「アカウント名」をクリックして「アカウント ID」を確認してください。
AWS ログイン画面で要求されることがありますので、メモ帳などに記録してください。



Copyright © 2024 YITJC All Rights Reserved.

10.2 対話型システム構成～クラウドサービス～

AWS（Amazon Web Service）を使用して「対話型システム」を構築します。AWS にはさまざまな種類のサービスが用意されていますが、対話型システムの構築に必要なサービスについて説明します。



～補足～

AWS アカウントを取得するとサインアップした日から 12 カ月分の「無料利用枠」が利用できますが、無料利用枠の上限を超えた使用は、料金が発生します。12 カ月経過後は、システムの使用状況（リクエスト数、実行時間等）により料金が発生します。参考として有料枠の金額も記載していますが、AWS は料金体系が非常に複雑であるため、正確な料金を知りたい場合は、AWS が提供する「AWS 料金計算ツール」を利用してください。

- Amazon API Gateway

  REST API（※1）を作成、公開、管理、モニタリング、保護するためのサービスです。

  Lambda 関数を Invoke（呼び出す）ために使用します。

  ✓ API コール数と接続時間により課金されるシステム

  ✓ リージョンごとに料金が異なる

  -REST API コール数は、1,000,000 件/月の無料利用枠（有料枠: US$4.25/1,000,000 件）

  -接続時間は、750,000 分/月の無料利用枠（有料枠: US$0.315/1,00,000 分）

  （※1）Representational State Transfer API

  「REST（レスト）」と呼ばれる設計原則にしたがって策定された API です。Web システムを外部から利用するための API として使用されます。「対話型システム」の構成で考えると、外部（LINE Messaging API）から Web システム（Lambda 関数）を呼び出すための API（Amazon API Gateway）ということになります。

Copyright © 2024 YITJC All Rights Reserved.

- AWS Lambda

  AWS Lambda は、クラウド上で Function（関数・機能）の実行環境を提供するサービスで、FaaS（Function as a Service）と呼ばれています。クラウド上にプログラムを定義しておき、インターネットを通じて実行できるサービスです。サーバー等の管理は AWS が行うため、サーバーレス FaaS とも呼ばれています。

  ✓ 対応言語: Python, Node.js, Java, Ruby, C#, Go, PowerShell

  ✓ 関数に対するリクエストの数とコードの実行時間により課金されるシステム

  -リクエスト数は、1,000,000 件/月の無料利用枠（有料枠: US$0.2/1,000,000 件）

  -コンピューティング時間は、毎秒 400,000GB の無料利用枠

  （有料枠（128MB）: US$0.0000002083/100msec）

- AWS IoT Core

  IoT デバイス（モノ: Raspberry Pi）とクラウドサービスを安全に接続するためのサービスです。

  IoT デバイスがクラウドサービスにアクセスする際のエンドポイントや認証処理などを管理します。

  ✓ IoT デバイスが AWS IoT Core に接続した時間とメッセージ数により課金されるシステム

  -接続時間は、2,250,000 分の無料利用枠（有料枠: US$0.096/1,000,000 分）

  -メッセージ数は、500,000 件の無料利用枠（有料枠: 10 億件まで US$1.20）

- AWS Identity and Access Management（IAM）

  AWS のサービスやリソースへのアクセスを安全に制御するための仕組みです。IAM は料金設定がなく、AWS アカウントに提供されている機能です。AWS Lambda が AWS IoT Core にアクセスする際に利用します。

- CloudWatch

  AWS で実行されるアプリケーションを監視し、ログファイルの監視、保存、アクセスが可能です。Raspberry Pi 上でプログラムを実行していたときは、実行結果、発生したエラーなどの情報はコンソール（Tera Term）画面で確認できていましたが、「対話型システム」においては、Lambda 関数をクラウド上で実行するため、実行結果や発生したエラーなどの情報は CloudWatch logs サービスを使用して確認します。

  ✓ 初期費用や最低利用料金の設定はなく、従量課金制となっています。

  -ログデータは、5GB 分の無料利用枠

Copyright © 2024 YITJC All Rights Reserved.

10.2 対話の流れ

「対話型システム」の仕組みについて説明します。次の図は、LINE ユーザーからメッセージを送信し、受信したメッセージに応じて Raspberry Pi が応答（処理を実行）するまでの流れを表しています。

まず、LINE ユーザーからメッセージが送信されると、

① 「Webhook」と呼ばれる仕組みを利用して、LINE ユーザーからのリクエストを API Gateway が受信します。

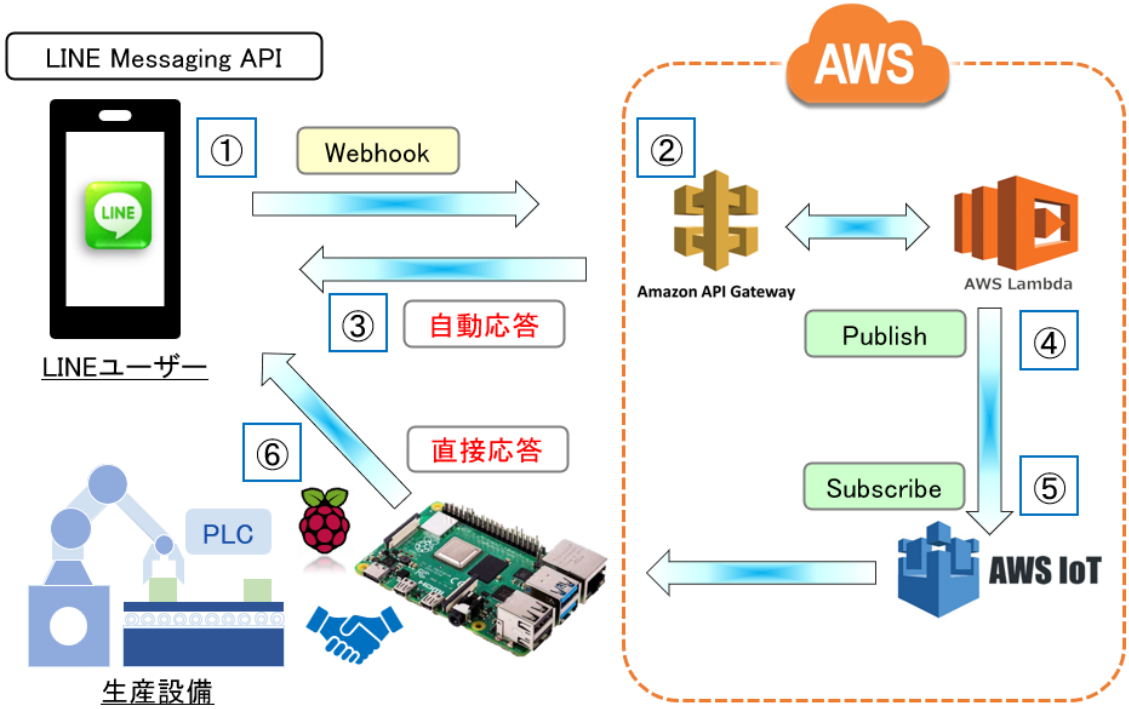
② API Gateway が AWS Lambda 関数（Python プログラム）を呼び出します（invoke）。
クラウド上の Lambda 関数によって、LINE ユーザーからのメッセージに応じた処理が行われます。

③ LINE ユーザーからのメッセージに応じて「自動応答」します。
いわゆる「LINE Bot」と呼ばれる動作を行います。LINE ユーザーからのメッセージを受信すると、あらかじめプログラムに設定されているメッセージを自動応答します。

④ LINE ユーザーからのメッセージに応じて「Publish」します。
受信先となる AWS IoT との間で、Publish/Subscribe 通信を行います。

⑤ AWS IoT Core サービスで、Raspberry Pi を IoT デバイス（Things: モノ）として登録します。
Publish/Subscribe 通信により AWS Lambda からのメッセージを受信します。

⑥ Publish/Subscribe 通信により受信したメッセージに応じた処理を Raspberry Pi が実行します。
このとき、Raspberry Pi は LINE ユーザーに対して直接メッセージを送信できます。
つまり、Raspberry Pi がセンサや生産設備から取得した情報を、AWS を経由せずに直接応答できます。
また、Raspberry Pi が生産設備や PLC と接続している場合、センサや生産設備のデータを取得するだけでなく、LINE ユーザーから送信されたメッセージで生産設備や PLC を制御することもできます。



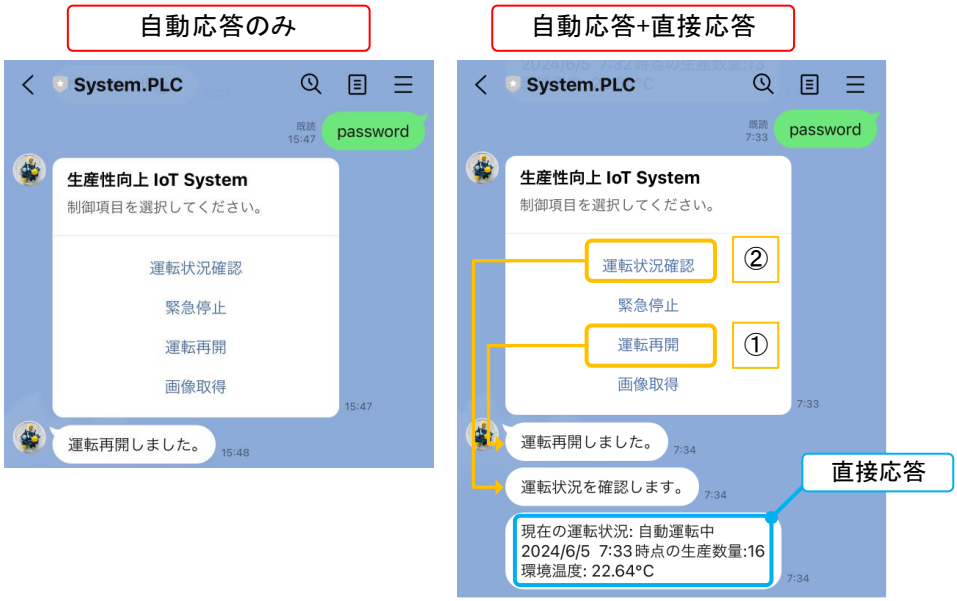
「対話型システム」の構成の中で重要なポイントは、Raspberry Pi から LINE ユーザーに対して直接応答ができる点です。AWS サービスの活用により、LINE ユーザーからのメッセージに対して自動応答が実現します。いわゆる、LINE Bot と呼ばれる機能です。しかし、LINE Bot はあらかじめ登録しておいたメッセージを送信するだけで、「対話」とは呼べません。Raspberry Pi から LINE ユーザーに対して直接応答することで「対話」が実現します。

Copyright © 2024 YITJC All Rights Reserved.

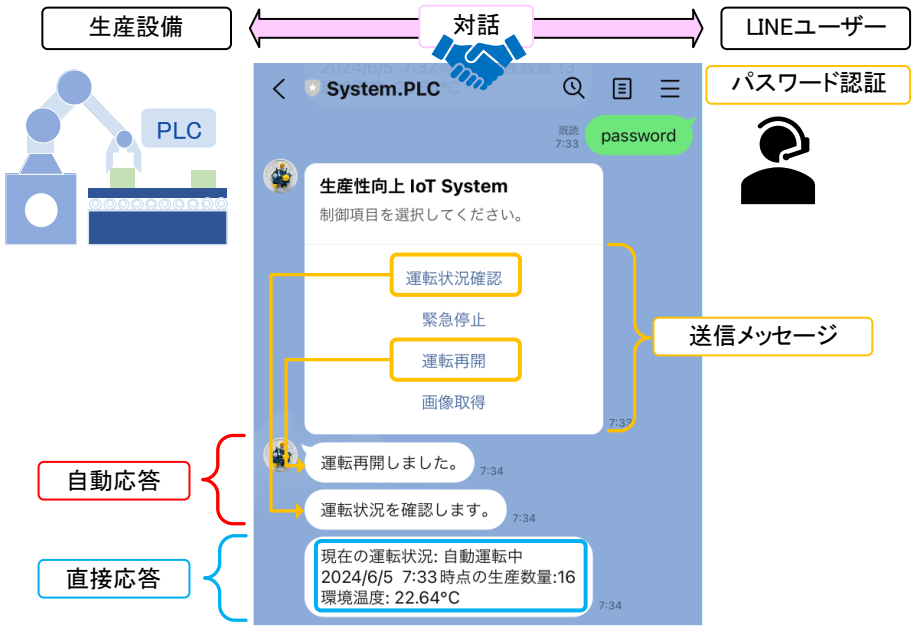
LINE ユーザーから生産設備に対して「運転再開」の指示を出す場面を例に挙げて、「対話」の例を紹介します。

左図は、自動応答だけで構成したシステムです。LINE ユーザーから「運転再開」の指示を行っていますが、実際に生産設備の運転が再開されたことを確認するためには、生産現場に赴かなければなりません。

右図は、自動応答に加えて Raspberry Pi から LINE ユーザーに対して直接応答しています。LINE ユーザーから「運転再開」の指示①を行ったあと、「運転状況確認」の指示②を出すことで、Raspberry Pi が生産設備（PLC）の状況を確認して、正確な情報を LINE ユーザーに対して直接応答しています。



「対話型システム」は、AWS による「自動応答」と Raspberry Pi による「直接応答」を組み合わせて「**対話**」を実現しています。



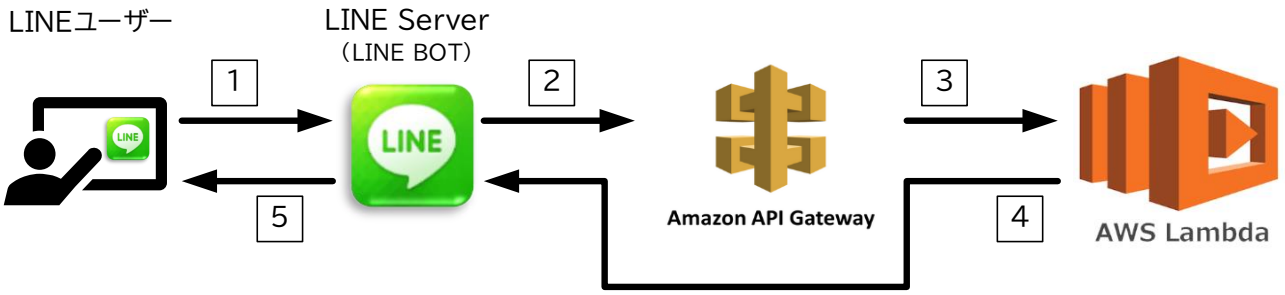
Raspberry Pi が生産設備と連携することで生産設備のデータを取得するだけでなく、生産設備を制御することもできます。これまでに構築した「IoT システム」に「対話型システム」を組み合わせることで、人と生産設備がいつでも「対話」可能なシステムが実現します。

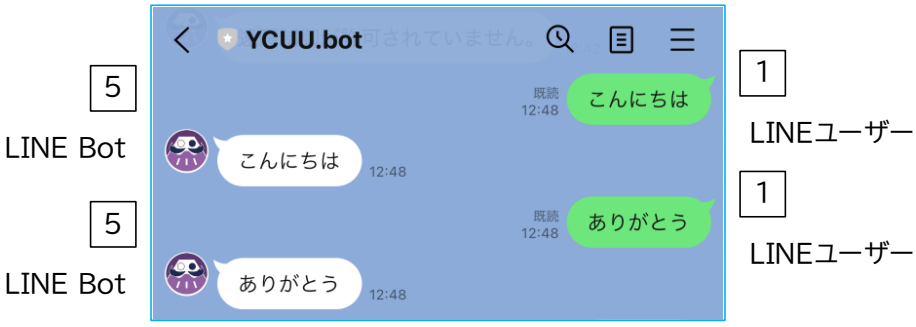
Copyright © 2024 YITJC All Rights Reserved.

10.3 自動応答の仕組み

LINE ユーザーが送信したメッセージに対して、AWS が「自動応答」を返信する仕組みについて説明します。

1. LINE ユーザーがメッセージを送信
2. LINE Server（LINE Bot）からのリクエストを API Gateway が受信
   API Gateway が発行する URL を LINE Messaging API の Webhook URL として設定します。
3. API Gateway が AWS Lambda 関数（Python プログラム）を呼び出します（invoke）。
   Lambda 関数は、LINE Messaging API の「WebhookHandler」を使用して LINE ユーザーからメッセージイベントを処理します。WebhookHandler は、テキストメッセージだけでなく、画像や音声が含まれる内容も処理できます。
4. WebhookHandler の Reply メソッドを使用して、LINE ユーザーから受信したメッセージをそのまま返信します。
5. LINE Server（LINE Bot）が Lambda 関数から受け取ったメッセージを LINE ユーザーに送信します。





～補足～

Amazon API Gateway は、LINE からのリクエストを「Webhook」と呼ばれる仕組みを利用して受信しています。「Webhook」はイベントが実行されたときに、外部サービスに HTTPS 通信で通知する仕組みです。「対話型システム」における「Webhook」を利用した動作は、次のとおりです。

① LINE ユーザーからメッセージ送信（イベント発生）
② イベント発生後、API Gateway が指定する宛先（URL）に HTTPS 通信で通知

Copyright © 2024 YITJC All Rights Reserved.

10.4 Pub/Sub メッセージングモデル

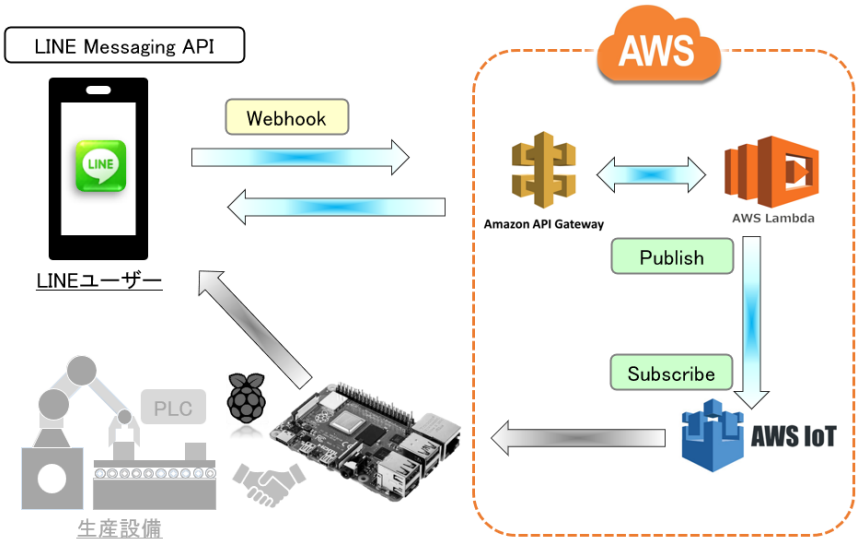
「対話型システム」を構築するためには、LINE ユーザーから送信されたメッセージを Raspberry Pi に届ける必要があります。ここでは、AWS Lambda に届けられた LINE ユーザーからのメッセージを AWS IoT に届ける仕組みについて説明します。

～補足～

「対話型システム」は、離れた場所にいる従業員（LINE ユーザー）から生産現場に設置した Raspberry Pi に対して、遠隔監視・制御命令を送信し、Raspberry Pi がさまざまな処理を実行するシステムです。

AWS IoT Core サービスは LINE ユーザーからのメッセージを受信すると、Raspberry Pi に対してさまざまな処理を命令できます。対話による遠隔監視制御を実現するためには、LINE ユーザーからのメッセージを AWS IoT Core サービスに届ける仕組みが重要です。

AWS IoT Core サービスと Raspberry Pi との連携については、「10.5 Subscribe 実装」で説明します。

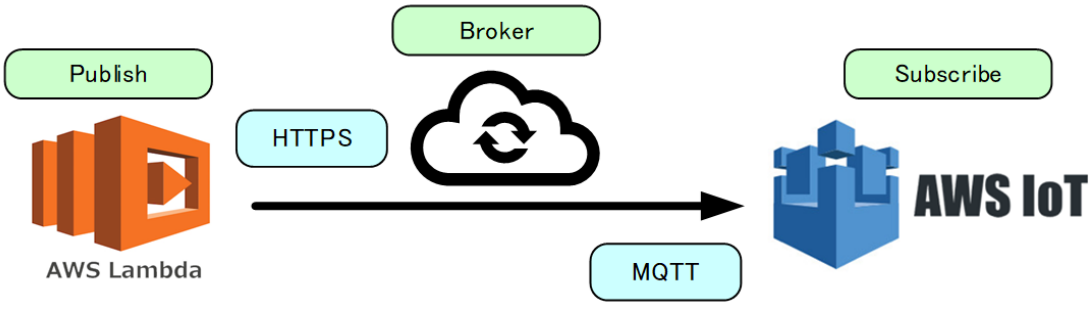


AWS Lambda と AWS IoT の通信は、「Pub/Sub **メッセージングモデル**」を使用します。

Pub/Sub メッセージングモデルは、Publish-Subscribe 方式でメッセージを送受信するためのモデルです。

メッセージの送信側は Publisher（出版社）、受信側は Subscriber（購読者）、仲介役を Broker と呼びます。

「対話型システム」の構成では、AWS Lambda が Publisher、AWS IoT が Subscriber および Broker になります。通信プロトコルは、Publish 側が「HTTPS」、Subscribe 側が「MQTT」を使用します。



Copyright © 2024 YITJC All Rights Reserved.

参考～HTTPS 通信と MQTT 通信～

HTTPS 通信と MQTT 通信の特徴について説明します。

- HTTPS（Hyper Text Transfer Protocol Secure）
  - ✓ HTTP に対してセキュリティ機能を追加したもので、通信を暗号化
  - ✓ 同期プロトコル（1 つのリクエストに対して 1 つのレスポンスを返す）
  - ✓ 多数のヘッダーとルールを伴う重量級プロトコルであり、オーバーヘッド（※1）が大きい

    ※1: 目的の処理に付随して発生する付加的な処理。通信においては、送信データそのものとは別に必要となるヘッダーなどのデータや、それを処理、伝送するために発生する負荷や時間のことを言います。

- MQTT（Message Queueing Telemetry Transport）
  - ✓ Pub/Sub メッセージングモデルにもとづく軽量プロトコルであり、オーバーヘッドが小さい

    （ヘッダーサイズは最小 2 バイト）
  - ✓ 非同期に 1 対 n の通信が可能（n 対 n 通信も可能）
  - ✓ 到達保証（QoS: Quality of Service）を指定可能

    AWS IoT Core は、QoS0 および QoS1 をサポートしています。

    （使用する Broker によって対応している QoS が異なるので注意してください。）

    QoS0: At most once

    -メッセージは最高 1 回配信される

    -メッセージが送信先に確実に届くかは保証されない

    QoS1: At least once

    -メッセージは最低 1 回配信される

    -メッセージが送信先に届くことが保証されるが、重複して届く可能性がある

    QoS2: Exactly once

    -メッセージは正確に 1 回配信される

    -メッセージが重複なく送信先に届くことが保証される
  - ✓ Broker（※2）と呼ばれる仲介役（中継サーバー）が必要

    ※2: 情報を発信する Publisher は、情報のカテゴリー（種類）を表す「topic」を付けて Broker にメッセージを送信します。情報を受信する Subscriber は「topic」を指定して Broker からメッセージを受信します。
  - ✓ Keep Alive 機能

    一定の時間内に Publisher からのメッセージ送信がない場合、コネクションが切れていないか確認する機能があります。Keep alive timer に監視時間を秒で設定し、監視時間内にメッセージ送信がないとコネクションは自動的に切断されます。
  - ✓ LWT（Last Will and Testament）

    Publisher は LWT（遺言）という情報を追加することができます。何らかの原因で Publisher と通信できなくなった場合、Broker は LWT（遺言）を Subscriber に送信します。

Copyright © 2024 YITJC All Rights Reserved.

参考～対話型システムの通信プロトコル～

AWS Lambda と AWS IoT で異なる通信プロトコルを採用している理由について説明します。

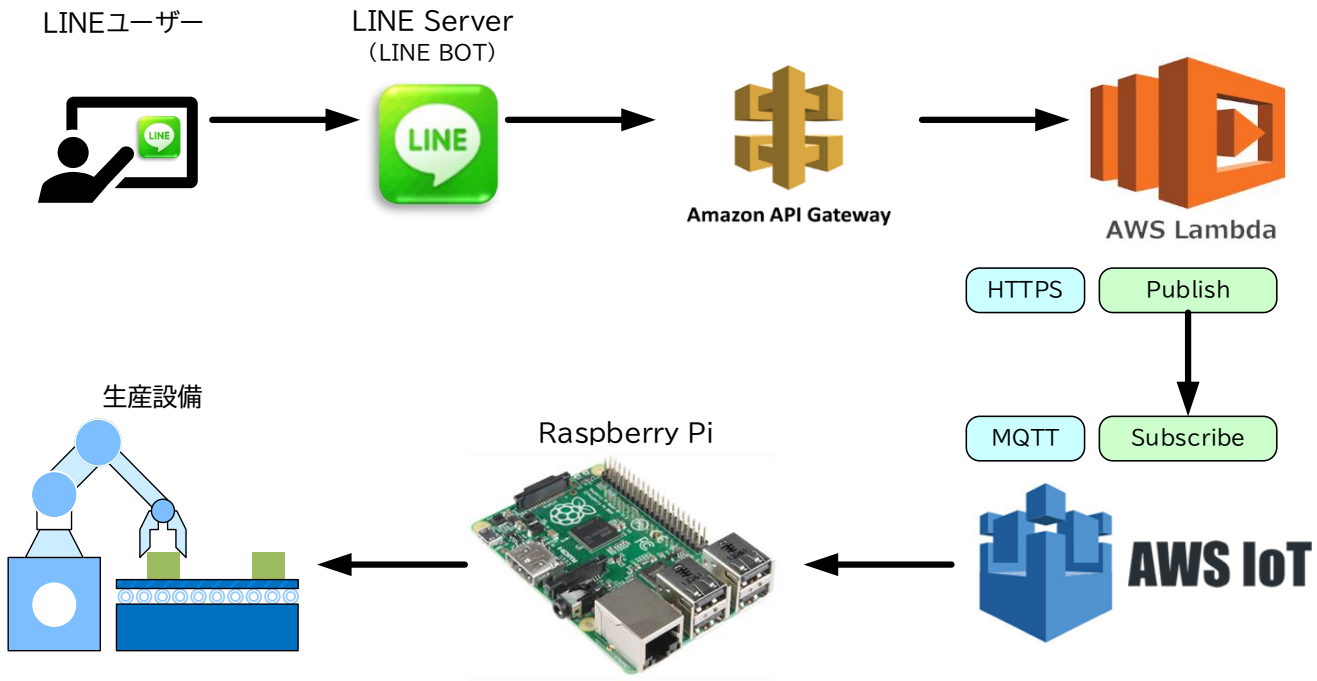
「MQTT」通信は「Pub/Sub メッセージングモデル」に基づく軽量プロトコルであり、IoT 時代を支えている通信プロトコルです。しかし、「対話型システム」の Publisher である AWS Lambda は、「HTTPS」通信プロトコルを採用しています。

これは、AWS Lambda 関数のタイムアウト時間（最大実行時間: 900 秒）が影響しています。MQTT は「常時接続」を想定した通信プロトコルです。AWS Lambda 関数に MQTT 通信プロトコルを実装してもタイムアウト時間の規定により常時接続ができず、通信が切れてしまいます。そのため、HTTPS 通信プロトコルを採用しました。

ここで、「対話型システム」における Publisher および Subscriber の働きを再確認します。

「Publisher」は、LINE ユーザーから送信されるメッセージを受信したら Publish する仕組みです。つまり、「常時接続」の必要はなく、LINE ユーザーからの「リクエスト」に対して「レスポンス」する HTTPS 通信プロトコルにより構成できます。

「Subscriber」は、Publisher から送信されるメッセージをいつでも受信できるように待機するために「常時接続」の必要があります。そのため、軽量プロトコルであり常時接続可能な MQTT 通信プロトコルを採用しています。



Copyright © 2024 YITJC All Rights Reserved.

参考～メッセージの送受信の流れ～

Pub/Sub メッセージングモデルによるメッセージ送受信の流れを説明します。

情報を発信する「Publisher」は、情報のカテゴリー（種類）を表す「topic」を付けて Broker にメッセージを送信します。「topic」はメッセージの種類を指定するもので、「/（スラッシュ）」区切りによって階層構造を表現できます。

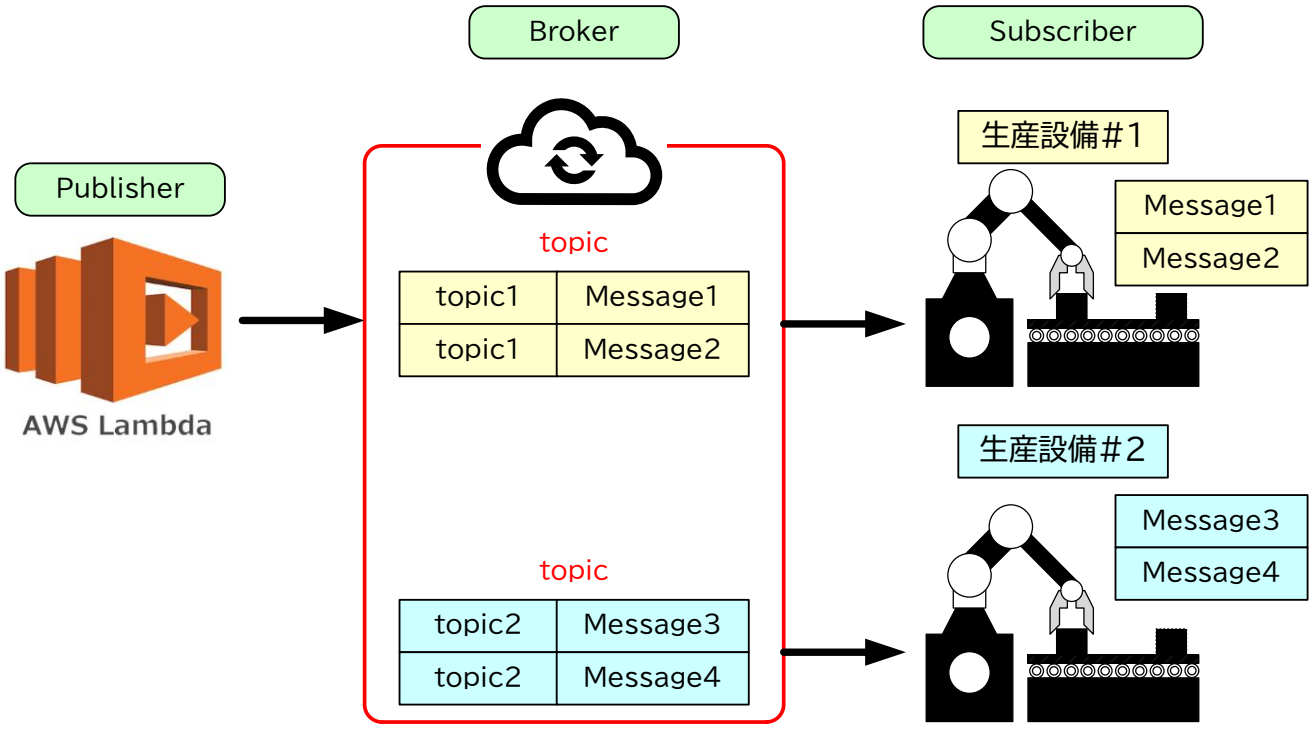
メッセージを Publish（送信）する時も Subscribe（受信）する時も、必ず「topic」トピックを指定する必要があります。

topic の例

- yamanashi/1F/temp　　山梨工場 1F の温度
- yamanashi/1F/hum　　山梨工場 1F の湿度
- yamanashi/1F/pres　　山梨工場 1F の気圧
- nagano/2F/temp　　長野工場 2F の温度
- nagano/2F/hum　　長野工場 2F の湿度
- nagano/2F/pres　　長野工場 2F の気圧

Pub/Sub メッセージングモデルによるメッセージ送受信の流れを図に示します。Publisher から送信されたメッセージは、Broker に送信され、「topic」ごとに分類されます。Broker に登録されたメッセージは、「topic」を指定して配信を申し込んできた Subscriber に配信されます。

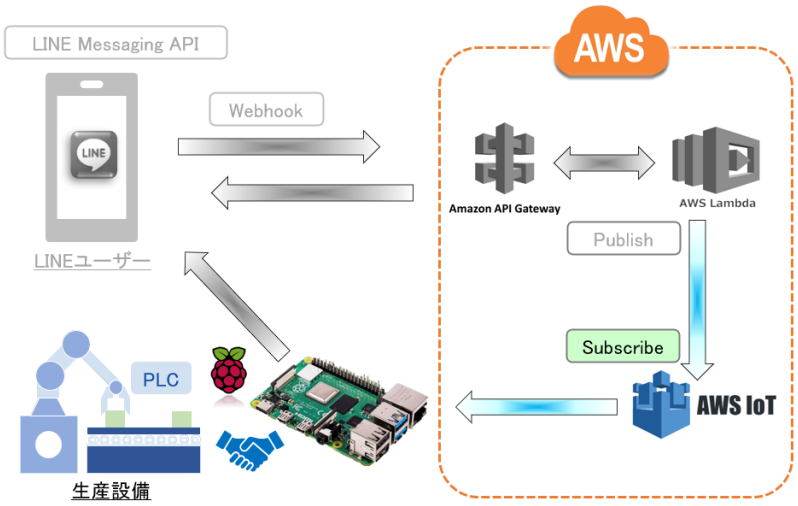
この図の例では、topic1 に登録された Message1 と Message2 は、「生産設備#1」に配信されます。topic2 に登録された Message3 と Message4 は、「生産設備#2」に配信されます。このように配信先の指定は「topic」を指定して行います。



Copyright © 2024 YITJC All Rights Reserved.

10.5 Subscribe 実装

「対話型システム」を構築するため、Subscribe を実装する手順について学習します。AWS IoT は、IoT デバイスとクラウドサービスを安全に接続するためのサービスです。ここでは、AWS IoT サービスを利用して Raspberry Pi を IoT デバイス(Thing: モノ)として登録します。モノとして登録された Raspberry Pi は、「Publish/Subscribe」通信によって、AWS Lambda から受信したメッセージに応じた処理を実行します。



次の URL をクリックし、AWS にログインしてください。

[URL] https://portal.aws.amazon.com

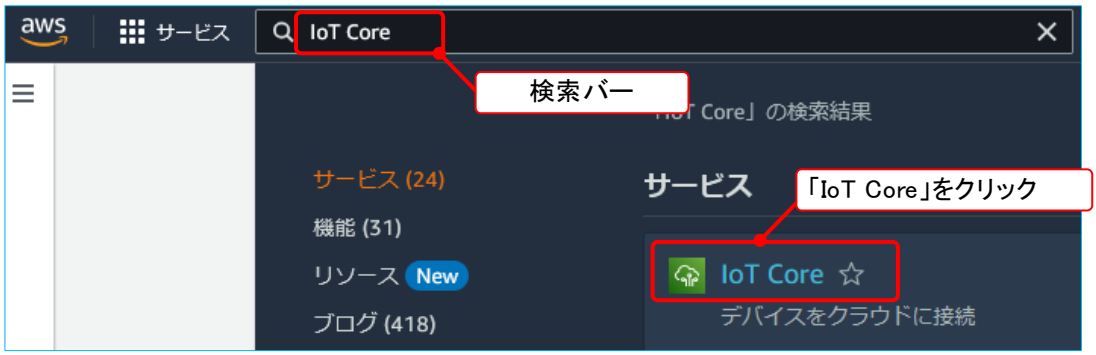
事前準備で取得したアカウントを使用してログインしてください。

いずかの画面を選択してログインしてください。

Copyright © 2024 YITJC All Rights Reserved.

手順①～IoT Core～

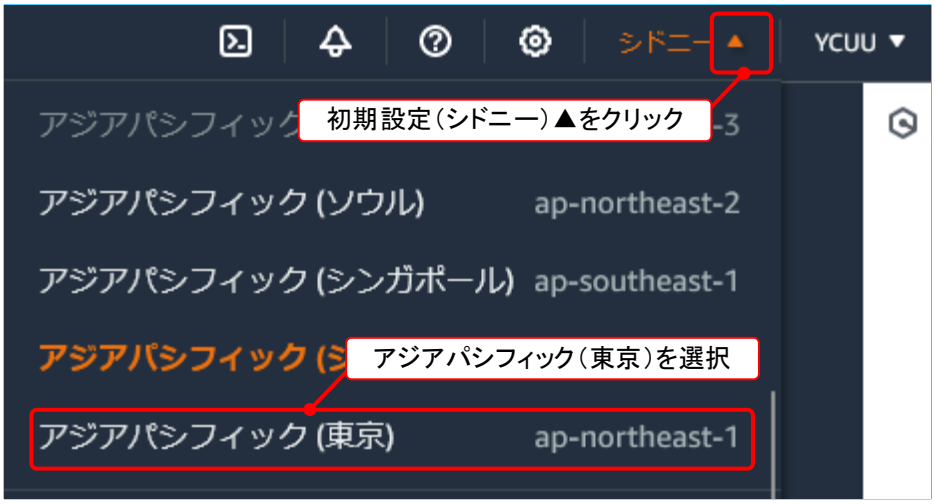
画面上部の検索バーを使用して、「IoT Core」を検索してください。



「IoT Core」をクリックすると、次の画面が表示されます。

手順②～リージョンの変更～

画面右上に表示されているリージョン（初期設定：シドニー）を変更します。▲をクリックすると、リージョンの一覧が表示されるので、「アジアパシフィック（東京）」を選択してください。



Copyright © 2024 YITJC All Rights Reserved.

東京を選択すると、画面右上のリージョン表示が「東京」に変わります。

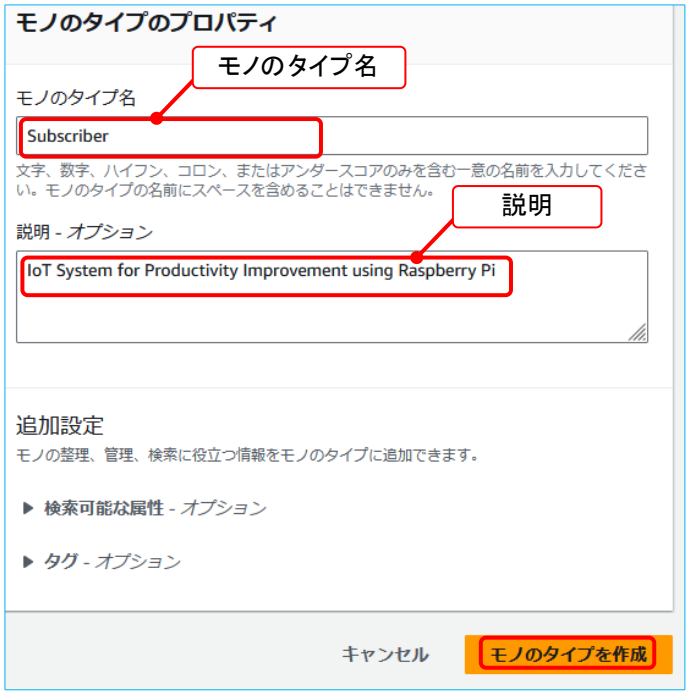
AWS は世界各地にデータセンター（サーバー）を保有しており、データセンターは、リージョン（地域）という単位で地理的に分割して管理しているため、リージョンを変更した場合は、すべての設定を最初からやり直す必要があります。

手順③～モノのタイプ作成～

このあと（手順⑤）作成する「モノ」のタイプを作成します。

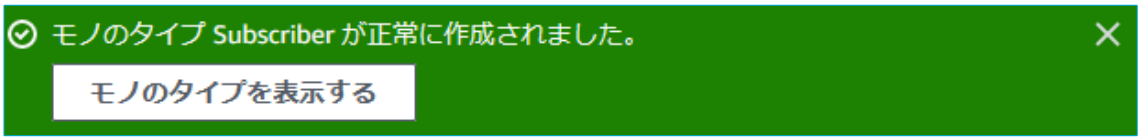
画面左のメニューから「すべてのデバイス」-「モノのタイプ」を選択し、「モノのタイプを作成」をクリックしてください。

「モノのタイプ名」、「説明」を入力したあと、「モノのタイプを作成」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

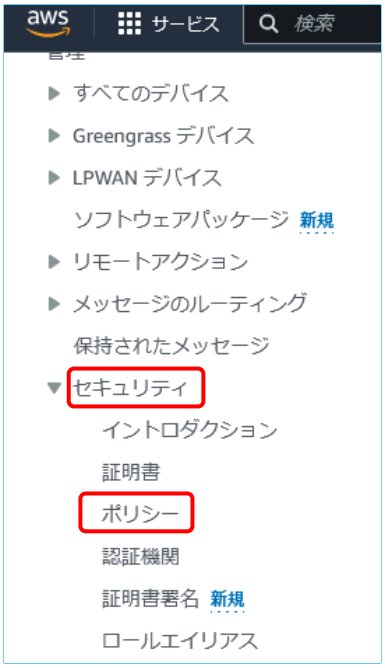
「モノのタイプ」の作成が成功すると、次の画面が表示されます。



手順④～ポリシーの作成～

このあと(手順⑤)作成する「モノ」に対するアクセス許可を設定するため、「ポリシー」を作成します。

画面左のメニューから「セキュリティ」-「ポリシー」を選択し、「ポリシーを作成」をクリックしてください。

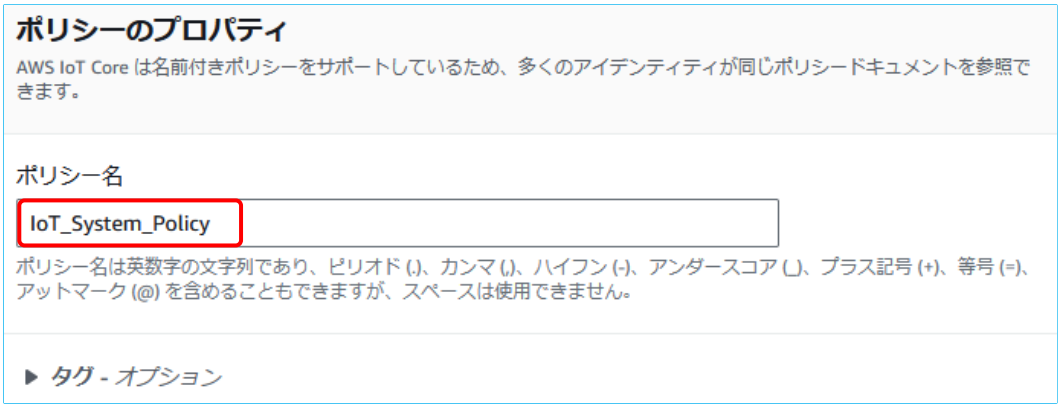


～参考～

AWS IoT Core ポリシー関する詳しい情報は、次の URL に記載されています。

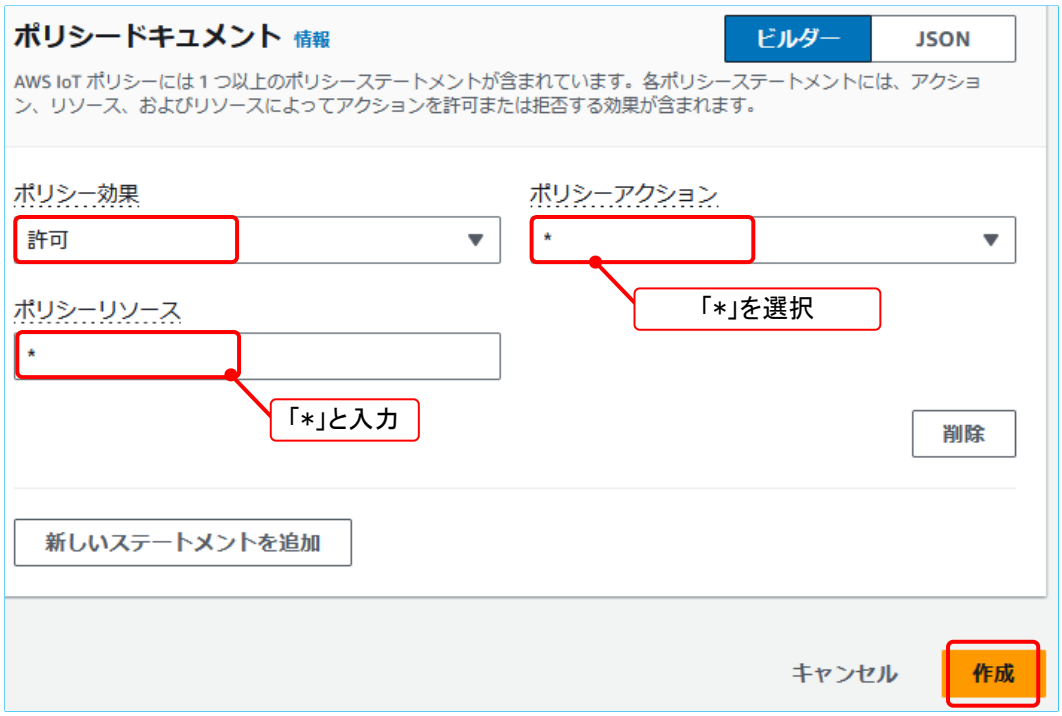
[URL] https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/iot-policies.html

ポリシーのプロパティで「ポリシー名」を入力してください。



Copyright © 2024 YITJC All Rights Reserved.

ポリシー効果「許可」、ポリシーアクションに「*」をそれぞれ選択し、ポリシーリソースに「*」を入力したあと、「作成」をクリックしてください。



ここでの設定は「Full Access」を許可していますが、「アクションを限定」および「リソースを指定」してセキュリティを強化することも可能です。

- アクション:「*」
  すべてのアクションを許可
- リソース ARN(Amazon Resource Name):「*」
  すべてのリソースからのアクセスを許可

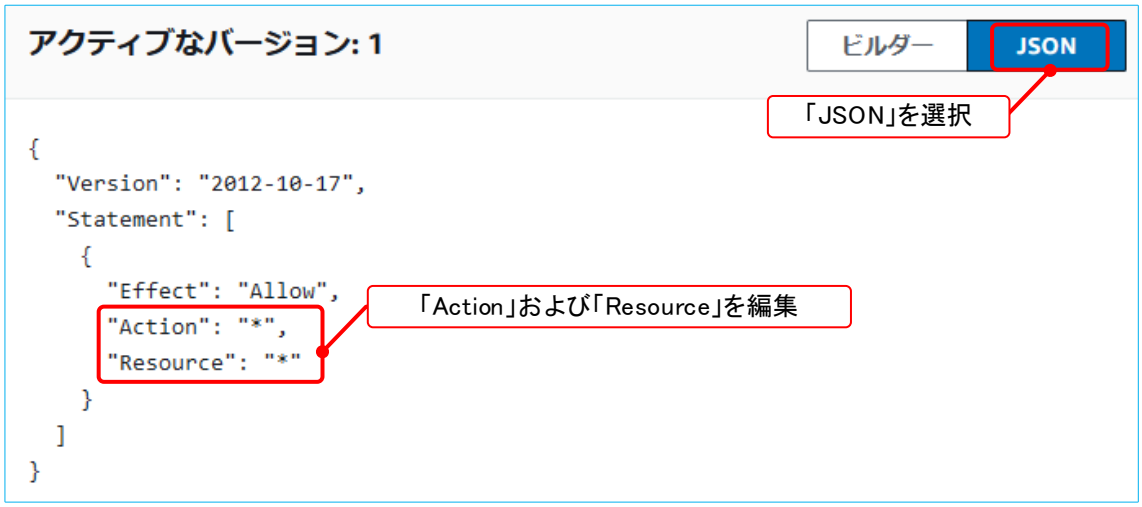
ポリシーの作成が成功すると、次の画面が表示されます。

～参考～

「アクション」および「リソース」の設定を編集する場合は、作成した「ポリシー名」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

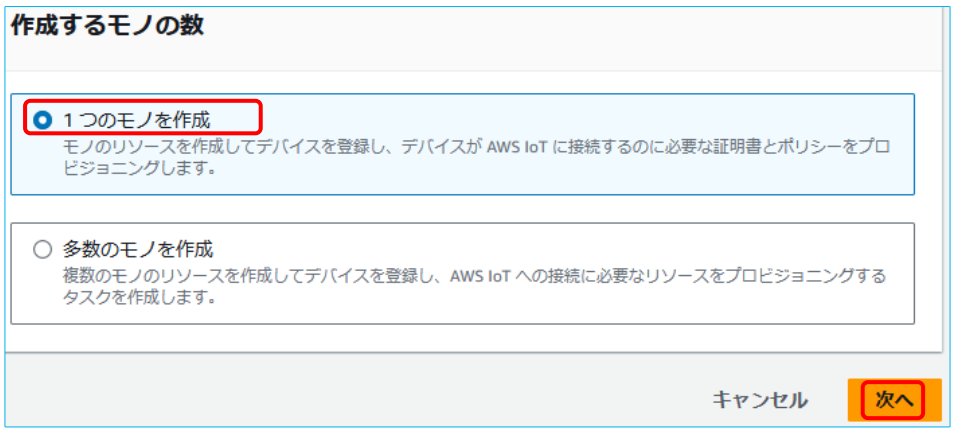
「JSON」をクリックし、「Action」および「Policy」の設定を編集できます。



手順⑤～モノの登録～

画面左のメニューから「すべてのデバイス」-「モノ」を選択し、「モノを作成」をクリックしてください。

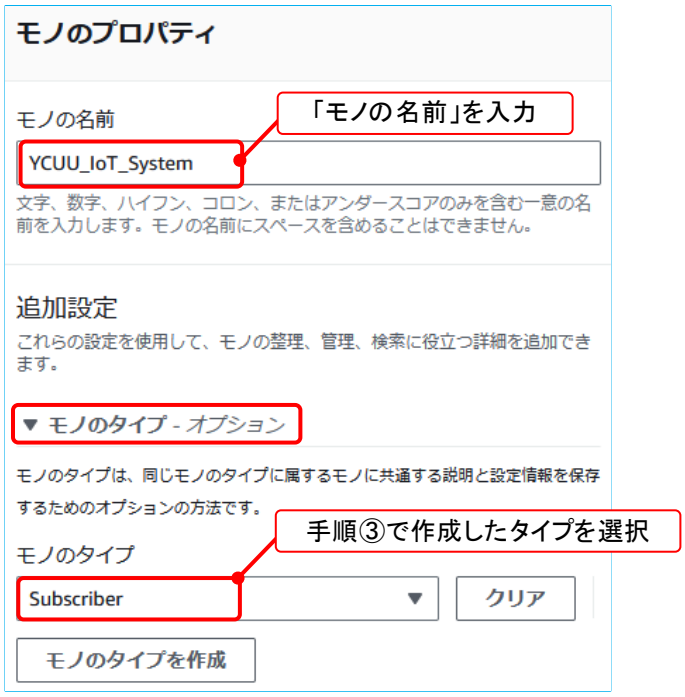
「1 つのモノを作成」を選択し、「次へ」をクリックしてください。



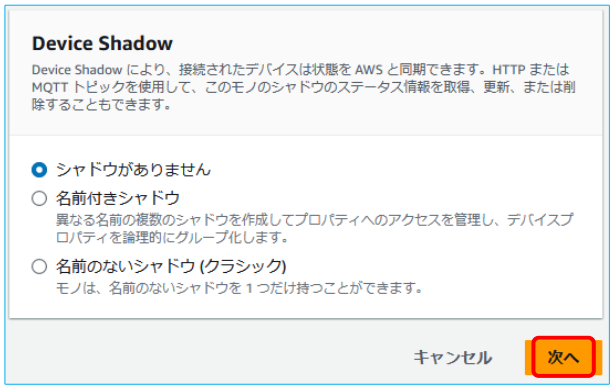
Copyright © 2024 YITJC All Rights Reserved.

「モノの名前」を入力してください。

また、モノのタイプ（オプション）で手順③で作成した「モノのタイプ」を選択してください。



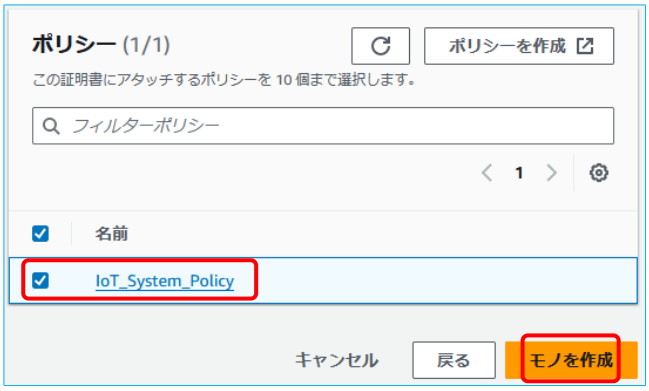
「次へ」をクリックしてください。



「新しい証明書を自動生成（推奨）」を選択し、「次へ」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

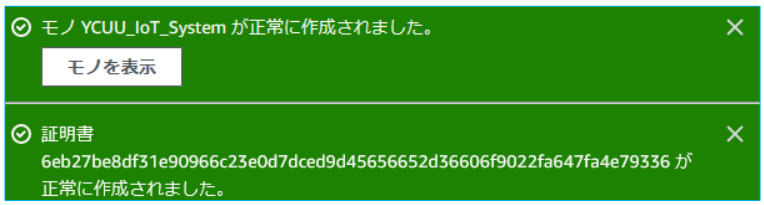
「証明書」にアタッチするポリシーとして、手順④で作成したポリシーを選択（チェック）してください。
続いて、「モノを作成」をクリックしてください。



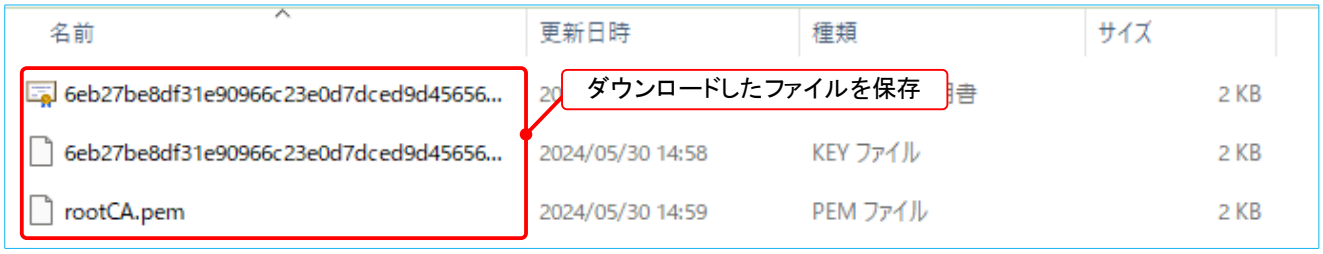
1～3まで 3 つのファイルをダウンロードしてください。
ダウンロードしたあと、「完了」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

ここまでの手順に問題がなければ、画面上部に「モノの作成」および「証明書の作成」の成功メッセージが表示されます。



ダウンロードしたファイル「AmazonRootCA1.pem」については、「rootCA.pem」にファイル名を変更してください。このあと（手順⑦）で使用するため、ダウンロードしたファイルをまとめて保存しておいてください。

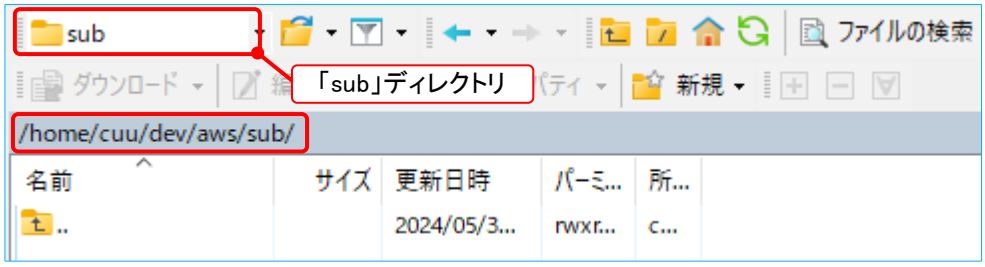


ここからは、Raspberry Pi の開発環境で作業を進めていきます。

手順⑥～ディレクトリ作成～

Subscribe プログラムを保存するディレクトリを作成してください。

```
(ycuu) cuu@raspberrypi:~ $ cd dev
(ycuu) cuu@raspberrypi:~/dev $ mkdir aws
(ycuu) cuu@raspberrypi:~/dev $ cd aws
(ycuu) cuu@raspberrypi:~/dev/aws $ mkdir sub
(ycuu) cuu@raspberrypi:~/dev/aws $ cd sub
```



手順⑦～証明書配置～

作成した「sub」ディレクトリに証明書を保存するディレクトリ「cert」を作成してください。

```
(ycuu) cuu@raspberrypi:~/dev/aws/sub $ mkdir cert
```

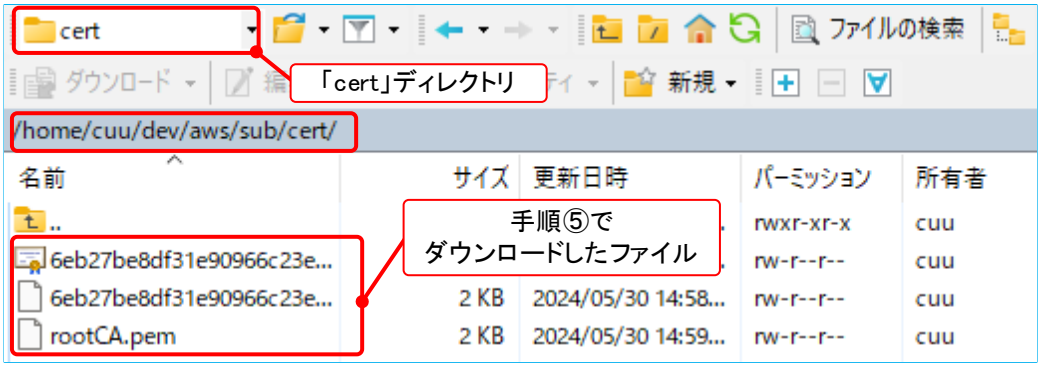
Copyright © 2024 YITJC All Rights Reserved.

手順⑤でダウンロードした証明書を「cert」ディレクトリに保存（ドラッグ＆ドロップ）します。

［保存するファイル］

- xxxxxx-certificate.pem.crt
- xxxxxx-private.pem.key
- rootCA.pem

xxxxxx の部分は、手順⑤でダウンロードした固有のファイル名です。



手順⑧～エンドポイント確認～

subscribe プログラムで指定する「エンドポイント」の情報を確認します。エンドポイント（リンク先）は、Raspberry Pi が AWS IoT Core に接続するために必要な情報です。

画面左のメニューから「設定」をクリックしてください。

「エンドポイント」の情報が表示さます。プログラムで使用するので、メモ帳などに記録してください。

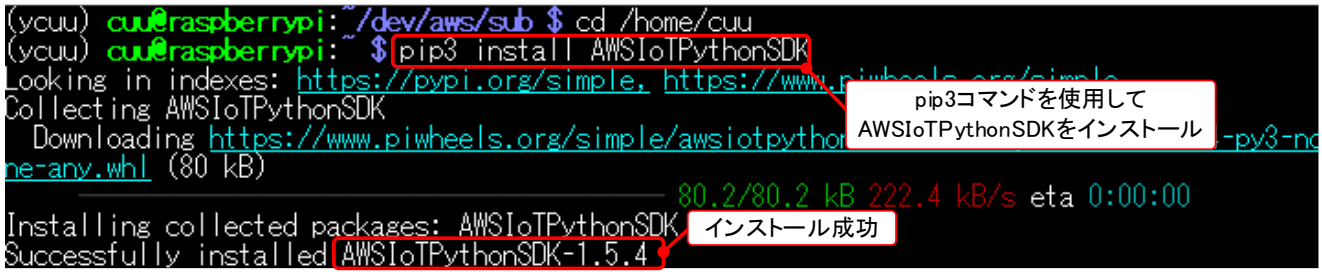
Copyright © 2024 YITJC All Rights Reserved.

手順⑨～AWSIoTPythonSDK インストール～

Python プログラムを使用して AWS IoT Core を利用するために必要なライブラリ「AWSIoTPythonSDK」をインストールします。

(ycuu) cuu@raspberrypi:~/dev/aws/sub $ cd /home/cuu

(ycuu) cuu@raspberrypi:~ $ pip3 install AWSIoTPythonSDK



手順⑩～環境変数設定～

AWS IoT Core を利用するために必要な情報（手順⑤および⑧で取得した情報）を環境変数に設定します。

環境変数に登録する情報は、次の情報です。

黄色で示した部分は、手順⑤および手順⑧で取得した情報を記載してください。

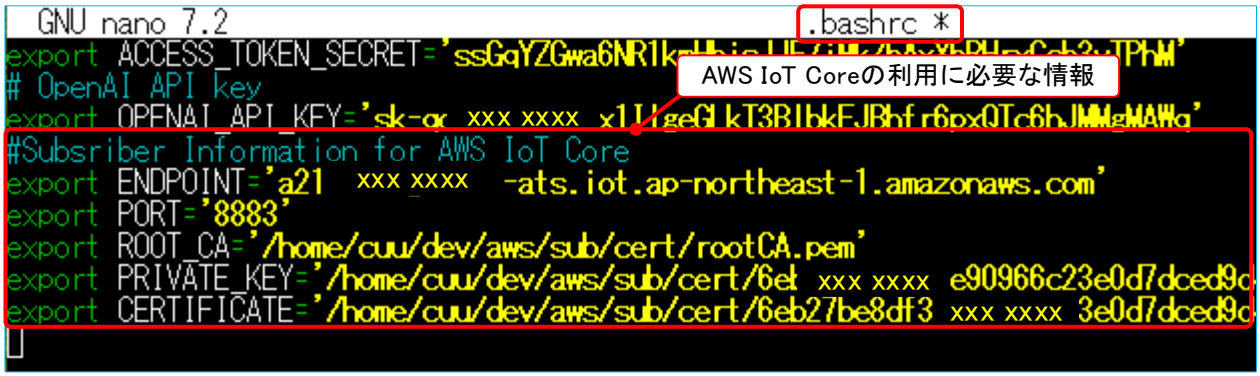
```
export ENDPOINT='a21xxxxxxxxxd3-ats.iot.ap-northeast-1.amazonaws.com'
export PORT='8883'
export ROOT_CA='/home/cuu/dev/aws/sub/cert/rootCA.pem'
export PRIVATE_KEY='/home/cuu/dev/aws/sub/cert/6eb27be8df31e90xxxxxxxxxx022fa647fa4e79336-private.pem.key'
export CERTIFICATE='/home/cuu/dev/aws/sub/cert/6eb27be8df31e90966c2xxxxxxxxxx647fa4e79336-certificate.pem.crt'
```

Raspberry Pi が起動したときに環境変数を自動的に設定するため、「.bashrc」ファイルに環境変数の設定コマンドを記述します。

ホームディレクトリに移動し、ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

(ycuu) cuu@raspberrypi:$ nano .bashrc

「.bashrc」ファイルの画面が起動したら、末尾に「export」コマンドで変数を 4 つ追加（記述）します。



記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑪～環境変数の反映～

続いて、次のコマンドを実行して設定した環境変数を反映します。

(ycuu) cuu@raspberrypi:$ source ~/.bashrc

```
cuu@raspberrypi:~ $ source ~/.bash_profile
```

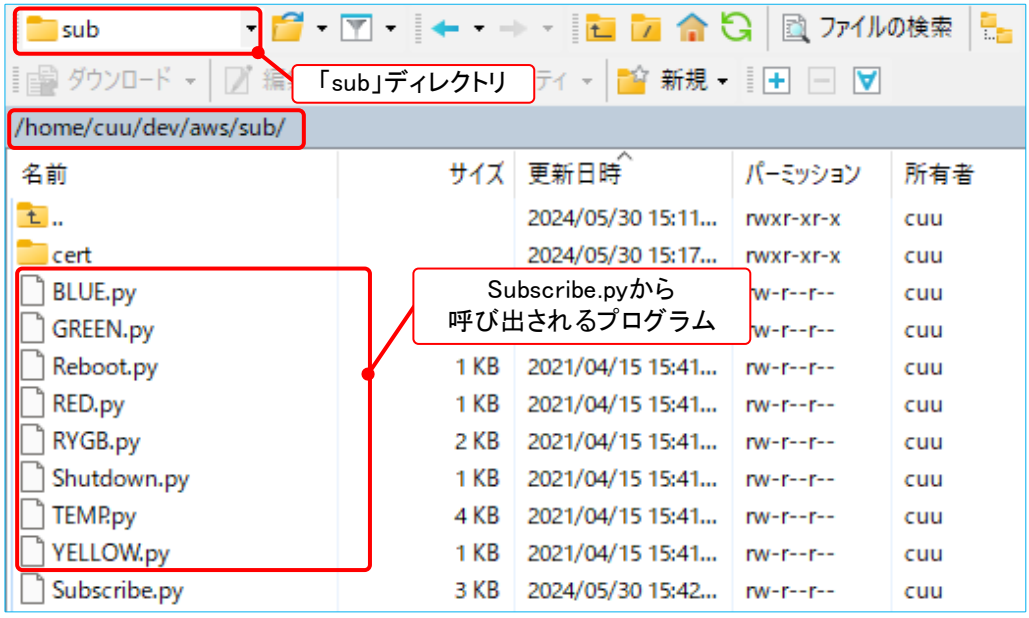
「~」の部分（チルダ）は、「Shift」+「～」で入力します。

手順⑫～プログラム配置～

対象プログラムを「sub」ディレクトリにコピーします。（ドラッグ＆ドロップ）。Subscribe.py 以外のプログラムは、Subscribe プログラムから呼び出される（subprocess ライブラリ使用して呼び出す）プログラムです。

［対象プログラム］

- ✓ Subscribe.py
- ✓ RED.py
- ✓ YELLOW.py
- ✓ GREEN.py
- ✓ BLUE.py
- ✓ RYGB.PY
- ✓ Shutdown.py
- ✓ Reboot.py
- ✓ TEMP.py



手順⑬～プログラム実行権限追加～

「sub」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev/aws/sub $ chmod +x Subscribe.py

～補足～

「Subscribe.py」以外のプログラムについては、実行権限を追加する必要はありません。

Copyright © 2024 YITJC All Rights Reserved.

手順⑭～プログラム実行～

Subscribe プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev/aws/sub $ ./Subscribe.py

プログラムを実行すると MQTT メッセージ「受信待ち」の状態になります。

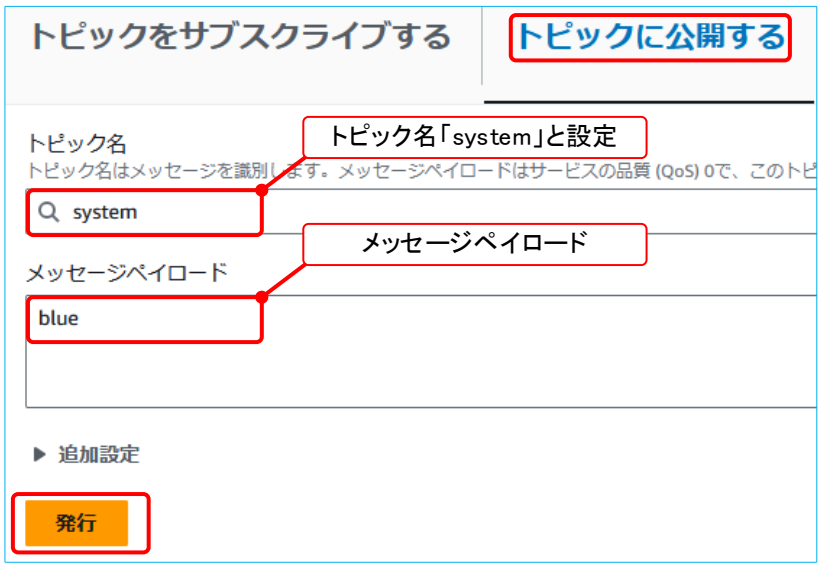
「MQTT テストクライアント」を使用して、Raspberry Pi に実装した「Subscribe」に MQTT メッセージを送信するとプログラムが実行されます。

画面左のメニューから「テスト」-「MQTT テストクライアント」を選択し、「トピックに公開する」をクリックしてください。

「MQTT テストクライアント」は、「Publish」と「Subscribe」両方のテストを実施できます。

今回は、Raspberry Pi に実装した「Subscribe」の動作確認を実施するため、「MQTT テストクライアント」は「Publish 機能（トピックに公開する）」を使用します。

トピック名「system」、「ペイロードメッセージ」を入力してから「発行」をクリックすると、Subscribe 側メッセージが発行（Publish）されます。



Copyright © 2024 YITJC All Rights Reserved.

Publish 側から送信されるメッセージ「トピック（topic）」と「ペイロード（payload）」に応じて、「Subscribe.py」プログラムは実行する（呼び出す）プログラムを設定しています。

「トピック」と「ペイロード」を指定して、Subscribe 側の動作を確認してください。

プログラム	動作説明	topic	payload
RED.py	赤色 LED2 秒間点灯	system	red
YELLOW.py	黄色 LED2 秒間点灯	system	yellow
GREEN.py	緑色 LED2 秒間　点灯	system	green
BLUE.py	青色 LED2 秒間点灯	system	blue
RYGB.py	赤・黄・緑・青 LED 順次点滅+ブザーON ※赤色 SW を押すまで動作継続	system	xxxx
Shutdown.py	シャットダウン実行	system	shutdown
Reboot.py	再起動実行	system	reboot

Subscribe 実行例

「トピック」と「ペイロード」を指定して Publish すると、受信した「ペイロード」に応じた処理が行われます。

動作確認が終了したら、「Ctrl」+「c」を押してプログラムを終了してください。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
5	import os
6	import time
7	import RPi.GPIO as GPIO
8	import subprocess
9	
10	#TOPIC指定（大文字・小文字の区別に注意）
11	#Publish(AWS Lambda)で設定しているTOPICと同じに設定
12	TOPIC = "system"
13	CLIENT_ID = "YCUU_IoT_System"
14	
15	#X APIの情報を環境編集から取得
16	def get_iotcore_settings():
17	IoTCore_Endpoint = os.getenv('ENDPOINT')
18	IoTCore_Port = os.getenv('PORT')
19	IoTCore_RootCA = os.getenv('ROOT_CA')
20	IoTCore_Privatekey = os.getenv('PRIVATE_KEY')
21	IoTCore_Certificate = os.getenv('CERTIFICATE')
22	return IoTCore_Endpoint, IoTCore_Port, IoTCore_RootCA, IoTCore_Privatekey, IoTCore_Certificate
23	
24	def set_MQTT():
25	global client
26	IoTCore_Endpoint, IoTCore_Port, IoTCore_RootCA, IoTCore_Privatekey, IoTCore_Certificate = get_iotcore_settings()
27	# setting for MQTT
28	client = AWSIoTMQTTClient(CLIENT_ID)
29	client.configureEndpoint(IoTCore_Endpoint, int(IoTCore_Port))
30	client.configureCredentials(IoTCore_RootCA, IoTCore_Privatekey, IoTCore_Certificate)
31	# AWSIoTMQTTClient connection configuration
32	client.configureAutoReconnectBackoffTime(1, 32, 20)
33	# -1:infinite
34	client.configureOfflinePublishQueueing(-1)
35	# Draining: 2 requests/second
36	client.configureDrainingFrequency(2)
37	# 10 sec

Copyright © 2024 YITJC All Rights Reserved.

```
38      client.configureConnectDisconnectTimeout(10)
39      # 600 sec
40      client.configureMQTTOperationTimeout(600)
41      #client.connect(600)
42      # Connect to AWS IoT with default keepalive set to 600 seconds
43      # keepAliveIntervalSecond : Time in seconds for interval of sending MQTT ping request. Default set to 600 seconds.
44
45  def mycallback(client, userdata, message):
46      payload = message.payload.decode("utf-8")
47      print("Received message '" + payload + "' on topic '" + message.topic + "' with QoS " + str(message.qos))
48      if payload == "red":
49          print("red")
50          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/RED.py'])
51      elif payload == "blue":
52          print("blue")
53          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/BLUE.py'])
54      elif payload == "yellow":
55          print("yellow")
56          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/YELLOW.py'])
57      elif payload == "green":
58          print("green")
59          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/GREEN.py'])
60      elif payload == "xxxx":
61          print("xxxx")
62          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/RYGB.py'])
63      elif payload == "shutdown":
64          print("shutdown")
65          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/Shutdown.py'])
66      elif payload == "reboot":
67          print("reboot")
68          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/Reboot.py'])
69      elif payload == "temp":
70          print("temp")
71          subprocess.check_call(['/home/cuu/ycuu/bin/python3.11','/home/cuu/dev/aws/sub/TEMP.py'])
72
73  if __name__ == "__main__":
74      try:
75          set_MQTT()
76          # Connect to AWS IoT
```

Copyright © 2024 YITJC All Rights Reserved.

```
77              client.connect()
78              while True:
79                  client.subscribe(TOPIC, 1, mycallback)
80                  time.sleep(5)
81          except KeyboardInterrupt:
82              GPIO.cleanup()
83          finally:
84              GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: AWS IoT Core を利用するために必要なライブラリ「AWSIoTPythonSDK」をインポート
- 5 行目: OS と対話するためのライブラリ「os」をインポート
- 6 行目: sleep 関数を使用するため time ライブラリをインポート
- 7 行目: GPIO 制御を行うため RPi.GPIO ライブラリをインポート
- 8 行目: 別のプログラムやコマンドを実行するためのライブラリ「subprocess」をインポート
- 12 行目: topic 設定
  大文字・小文字の区別があるので注意してください。
  Publish で指定した topic と同じに設定する必要があります。
- 13 行目: CLIENT_ID 設定
  デバイス（今回のケースは Raspberry Pi）を識別するため、CLIENT_ID を設定できます。
  CLIENT_ID には任意の値を割り当てることができますが、CLIENT ID が重複した場合は、MQTT3.1 の仕様に次の記述があります。
  If a client with the same Client ID is already connected to the server, the "older" client must be disconnected by the server before completing the CONNECT flow of the new client.
  同じ CLIENT_ID を使用して接続した場合は、先に接続していたクライアントは切断されて、新しいクライアントの接続が有効になります。

～環境変数から AWS IoT Core の情報取得～

- 16 行目～22 行目: 環境変数から AWS IoT Core の情報を取得するプログラム get_iotcore_settings()関数
- 17 行目: エンドポイント取得（手順⑧で確認したエンドポイント情報）
- 18 行目: ポート番号情報取得
- 19 行目: AWS IoT Core ルート CA ファイルを絶対パス情報取得
- 20 行目: Private key ファイルを絶対パス情報取得
- 21 行目: Certificate ファイルを絶対パス情報取得
- 22 行目: 他の関数で使用する変数（IoTCore_Endpoint, IoTCore_Port, IoTCore_RootCA, IoTCore_Privatekey, IoTCore_Certificate）を戻り値として返す

～参考～

AWSIoTPythonSDK の MQTT Moule に関する詳しい内容は、次の URL に記載されています。

[URL] https://s3.amazonaws.com/aws-iot-device-sdk-python-docs/sphinx/html/index.html

Copyright © 2024 YITJC All Rights Reserved.

～AWS IoT Core に接続するための MQTT Client 設定～

- 24 行目～40 行目: MQTT Client を設定するプログラム set_MQTT()関数
- 25 行目: グローバル変数で「client」を定義
- 26 行目: get_iotcore_settings()関数から AWS IoT Core の接続に必要な情報を取得
- 28 行目: 13 行目で定義した「CLIENT_ID」を使用して AWSIoTMQTTClient()クラスのインスタンス作成
- 29 行目: MQTT client の接続先エンドポイントとポート番号を設定
- 30 行目: セキュアな通信を確立するため、ルート CA 証明書、プライベートキーおよびクライアント証明を設定
- 32 行目: 自動再接続のバックオフ時間設定

  初回は 1 秒、最大で 32 秒、時間の増加は 20%ずつの設定です。

  Parameters

  baseReconnectQuietTimeSecond –

  The initial back off time to start with, in seconds. Should be less than the stableConnectionTime.

  maxReconnectQuietTimeSecond –

  The maximum back off time, in seconds.

  stableConnectionTimeSecond –

  The number of seconds for a connection to last to be considered as stable. Back off time will be reset to base once the connection is stable.
- 34 行目: オフライン時の Publish キューイング設定

  無限にキューイングする設定「-1」にしています。

  Parameters

  queueSize –

  Size of the queue for offline publish requests queueing.

  If set to 0, the queue is disabled. If set to -1, the queue size is set to be infinite.

  dropBehavior –

  the type of drop behavior when the queue is full.
- 36 行目: クライアントがオフライン時に蓄積されたメッセージの再送頻度設定

  「2 リクエスト/秒」に設定しています。

  ここでは、クライアントがオフライン中に蓄積されたメッセージをオンラインに戻ったときに再送する頻度を設定しています。オフラインキューイングにより、接続が復旧した際に蓄積されたデータを確実に送信できます。

  Parameters

  frequencyInHz –

  The draining speed to clear the queued requests, in requests/second.
- 38 行目: 接続または切断が完了するまでの待機時間設定

  10 秒に設定しています。

  Parameters

  timeoutSecond –

  Time in seconds to wait for a CONNACK or a disconnect to complete.

Copyright © 2024 YITJC All Rights Reserved.

・ 40 行目: MQTT QoS1 Publish, subscribe, unsubscribe 動作のタイムアウト時間設定
ここでは Subscriber の動作を行っているので、subscribe 動作のタイムアウト時間を設定しています。
「対話型システム」の構成において、このタイムアウト設定時間が重要なパラメータです。
詳しくは、このあとの「Subscriber 動作」で説明します。
Parameters
timeoutSecond –
Time in seconds to wait for a PUBACK/SUBACK/UNSUBACK.

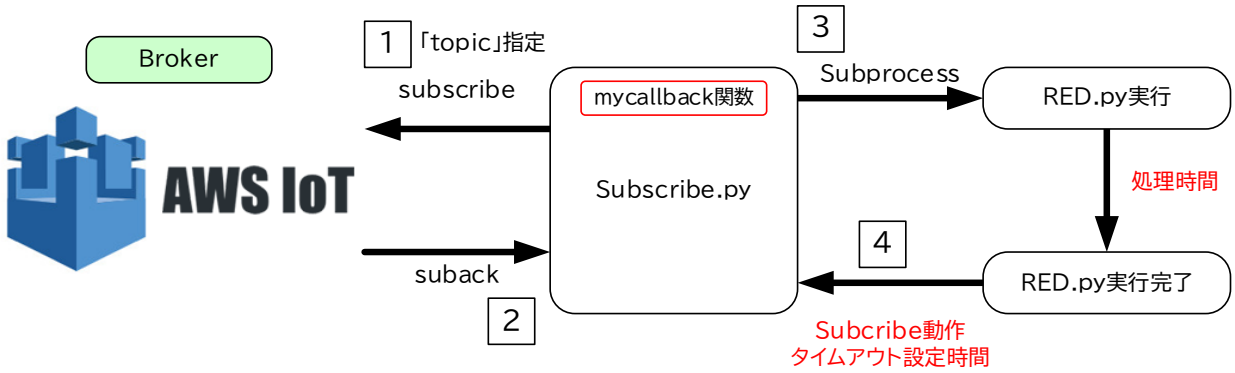
重要～Subscriber 動作～

Subscribe 動作について説明します。
76 行目: MQTT 接続確立
78 行目: 「topic」を指定して MQTT Broker からメッセージ受信
78 行目の記述を確認すると、subscribe メソッドの引数に「mycallback」があります。
これは、MQTT Broker からのメッセージを「mycallback」関数で受信することを意味しています。

```
client.subscribe(TOPIC, 1, mycallback)
```

つまり、Subscriber の動作は次の図で表すことができます。

1. MQTT Broker に対して「topic」を指定してペイロードを受信します。
2. Subscriber が正常にペイロードを受信すると、MQTT Broker は「suback」を返します。
3. Subscribe.py プログラム内の「mycallback」関数は、subprocess ライブラリを使用して、受信したペイロードに対応した処理（プログラム）を実行します。
4. subprocess ライブラリを使用して呼び出したプログラムの処理が完了



40 行目で設定している「client.configureMQTTOperationTimeout()」は、Subscriber がペイロードを受信してから mycallback 関数による処理が完了するまでの上限時間（タイムアウト時間）を設定しています。

例えば、subscribe 動作のタイムアウト設定時間を 5 秒に設定した場合を考えてみます。
subprocess モジュールを使用して呼び出すプログラムの処理に 5 秒以上要する場合は、mycallback 関数の処理が 5 秒以内に終了しないため、「タイムアウトエラー」が発生します。
「タイムアウトエラー」の発生を防ぐためには、「タイムアウト設定時間」を「処理時間」より長く設定する必要があります。

Copyright © 2024 YITJC All Rights Reserved.

- 41 行目: Keep Alive
  MQTT 通信には一定の時間内に Publisher からのメッセージ送信がない場合、コネクションが切れていないか確認する機能（Keep Alive）があります。
  client.connect(keep alive timer)メソッドを使用します。監視時間を「keep alive timer」秒で設定し、監視時間内にメッセージ送信がないとコネクションは自動的に切断されます。
  プログラム内で Keep Alive タイムの設定は行っていませんが、設定しない場合はデフォルトの 600 秒が設定されます。Keep Alive タイムは最大 1200 秒まで設定することができます。

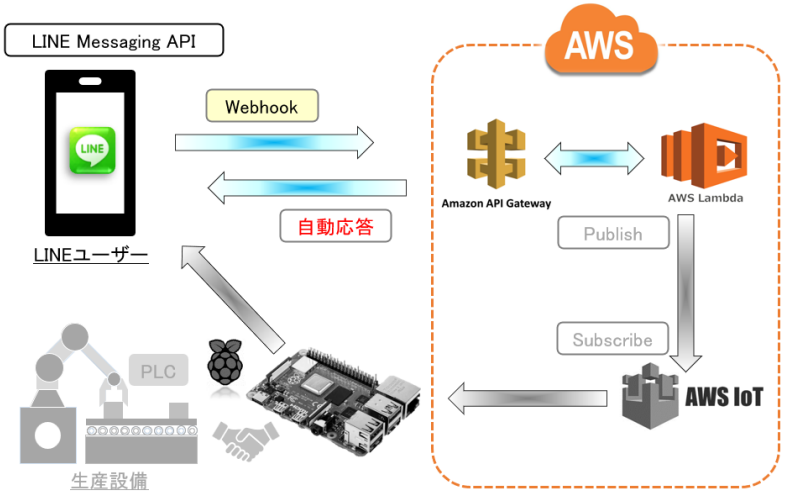
～mycallback 関数～

- 45 行目～71 行目: callback 関数
- 46 行目: MQTT で受信するメッセージの中身「payload」の定義
  バイト列を文字列に変換するため、decode メソッドを使用しています。
- 47 行目: 受信した「topic」、「payload」、「QoS」を print 表示
- 48 行目～71 行目: 受信した「payload」に応じて実行プログラムを呼び出し（subprocess モジュール使用）
- 73 行目～84 行目: メイン関数
- 74 行目～80 行目 :try ブロック
- 75 行目: MQTT Client を設定するプログラム set_MQTT()関数実行
- 77 行目: MQTT 接続確立
- 78 行目～80 行目: subscribe 動作を 5 秒間隔で実行
- 79 行目: 「topic」を指定して MQTT Broker からメッセージ受信
- 81 行目～82 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。
- 83 行目～84 行目: finally ブロック
  例外の発生によらず必ず実行される処理を記述しています。
  本プログラムは GPIO を使用しているため、GPIO.cleanup()を記述しています。これにより、例外の発生によらず GPIO ピンのシステムクリアが必ず実行されます。

Copyright © 2024 YITJC All Rights Reserved.

10.6 LINE Bot～LINE Messaging API 設定～

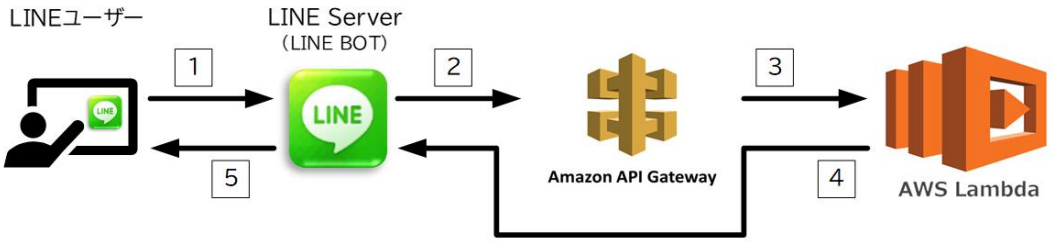
LINE Bot とは、LINE 上で動作する「自動応答ロボット（LINE Robot）」です。LINE ユーザーからの発言に対して自動応答するプログラムで構成されています。ここでは、AWS サービスを利用した「LINE Bot」システムの構築について学習します。



～復習～

LINE ユーザーが送信したメッセージに対して、AWS が自動応答を返信する仕組みついて復習します。

1. LINE ユーザーがメッセージを送信
2. LINE Server（LINE Bot）からのリクエストを API Gateway が受信
   API Gateway が発行する URL を LINE Messaging API の Webhook URL として設定します。
3. API Gateway が AWS Lambda 関数（Python プログラム）を呼び出します（invoke）。
   Lambda 関数は、LINE Messaging API の「WebhookHandler」を使用して LINE ユーザーからメッセージイベントを処理します。WebhookHandler は、テキストメッセージだけでなく、画像や音声が含まれる内容も処理できます。
4. WebhookHandler の Reply メソッドを使用して、LINE ユーザーから受信したメッセージをそのまま返信します。
5. LINE Server（LINE Bot）が Lambda 関数から受け取ったメッセージを LINE ユーザーに送信します。



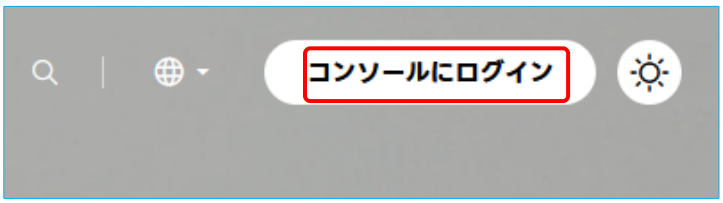
Copyright © 2024 YITJC All Rights Reserved.

手順①～LINE Developers ログイン～

LINE Messaging API の設定を行うため、次の URL にアクセスして LINE Developers にログインしてください。

[URL] https://developers.line.biz/ja/

画面右上の「コンソールにログイン」をクリックしてください。



「LINE アカウントでログイン」をクリックしてください。

「メールアドレス」と「パスワード」を入力したあと、「ログイン」をクリックしてください。

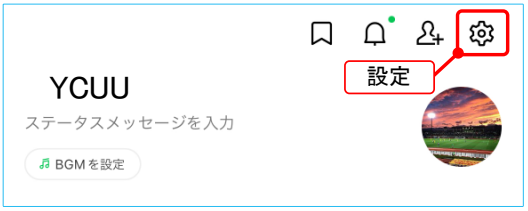
「ログイン」をクリックしたあと、スマートフォンの LINE アプリと連携が行われます。

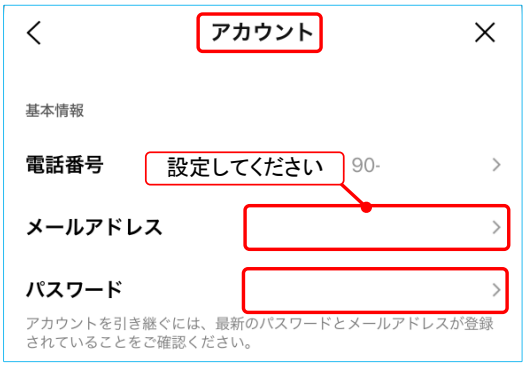
お手元にスマートフォンを用意してログインしてください。

Copyright © 2024 YITJC All Rights Reserved.

～補足～

LINE アカウントで「メールアドレス」および「パスワード」を設定していない場合は、スマートフォンの LINE アプリから設定してください。「設定」-「アカウント」から設定できます。





手順②～LINE Business ID 作成～

手順①でログインした LINE アカウントを使用して LINE Business ID を作成します。

「LINE ビジネス ID を作成」をクリックしてください。

「開発者名」を入力してください。

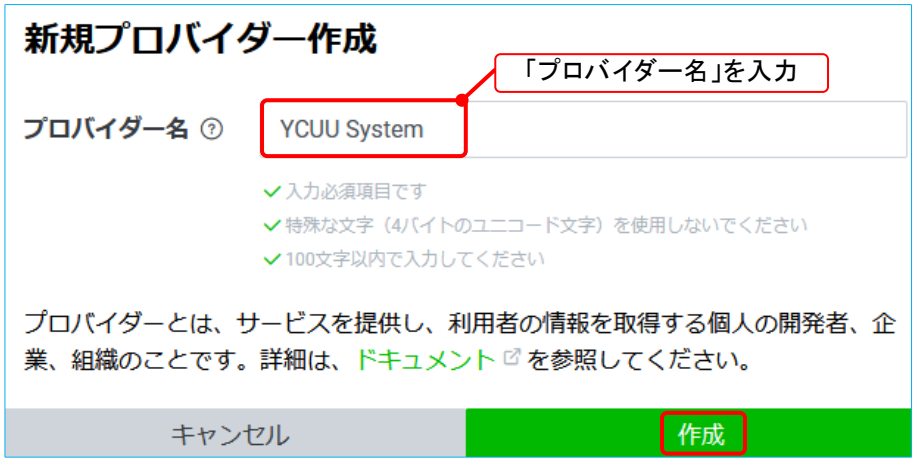
Copyright © 2024 YITJC All Rights Reserved.

「メールアドレス」を入力し、LINE 開発者契約の内容を確認したあと、「アカウントを作成」をクリックしてください。

手順③～新規プロバイダー作成～

「新規プロバイダーを作成」をクリックしてください。

「プロバイダー名」を入力し、「作成」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

手順④～チャネル作成～

Messaging API チャネルを作成します。

「Messaging API」をクリックしてください。

チャネル作成に必要な情報を入力してください。

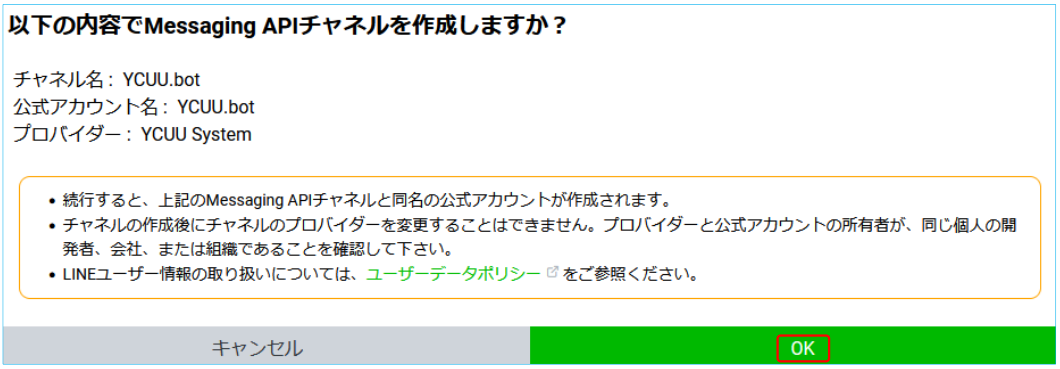
Copyright © 2024 YITJC All Rights Reserved.

引き続き、チャネル作成に必要な情報を入力してください。

最後に、「メールアドレス」を入力し、「LINE 公式アカウント利用規約」および「LINE 公式アカウント API 利用規約」の内容を確認してから、「作成」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「Messaging API チャネル」作成確認画面が表示されるので、「OK」をクリックしてください。



このあと、「情報利用に関する用意」のメッセージが表示されますが、内容を確認したうえで、「同意」をクリックしてください。

手順⑤～チャネルシークレット・ユーザーID 取得～

「チャネル基本設定」から LINE Bot プログラムで使用する「Channel Secret」を確認します。

「チャネルシークレット」と「ユーザーID」の情報は、プログラムで使用するので**メモ帳などに記録してください。**

Copyright © 2024 YITJC All Rights Reserved.

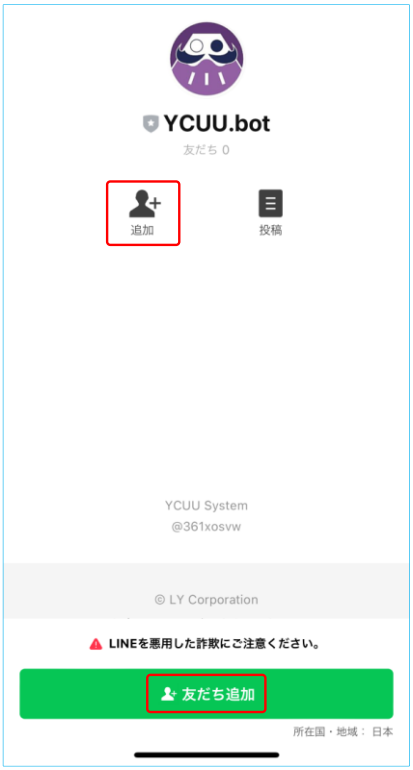
手順⑥～LINE Bot 登録～

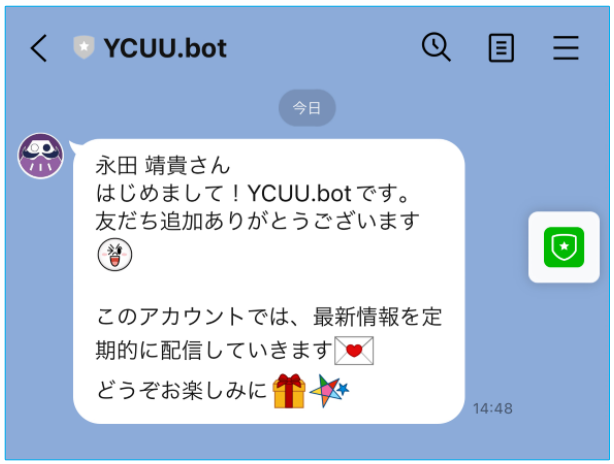
作成したチャネルを「LINE Bot」としてスマートフォンの LINE アプリに登録します。

「Messaging API 設定」タブをクリックしてください。

画面に表示されている「QR code」をスマートフォンの LINE アプリで読み取り、「友達」として追加してください。

スマートフォンの LINE アプリで「友達追加」のメッセージを受信できれば LINE Bot の登録完了です。





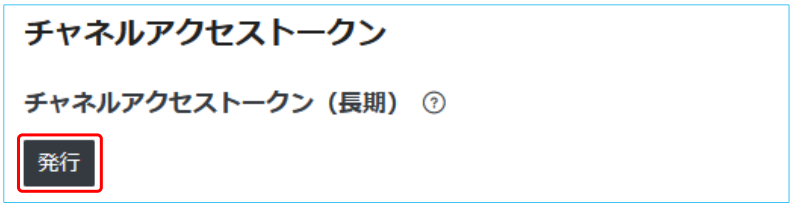
Copyright © 2024 YITJC All Rights Reserved.

手順⑦～チャネルアクセストークン取得～

「Messaging API 設定」タブから LINE Bot プログラムで使用する「チャネルアクセストークン」を取得します。

「Messaging API 設定」タブをクリックしてください。

この時点ではまだ発行されていないので、「発行」をクリックしてください。



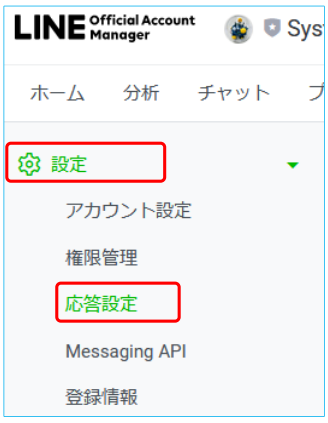
「チャネルアクセストークン」の情報は、プログラムで使用するのでメモ帳などに記録してください。

Copyright © 2024 YITJC All Rights Reserved.

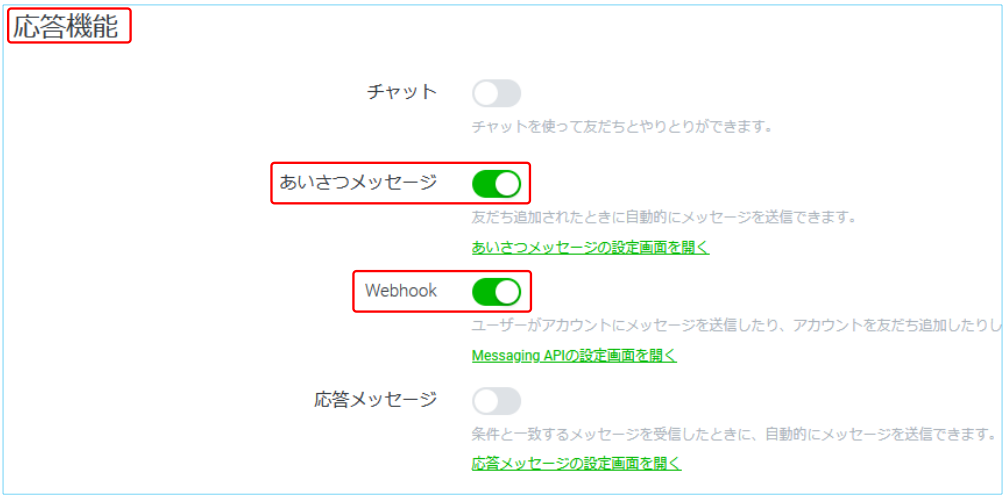
手順⑧～LINE Manager 応答設定～

「チャネル基本設定」タブをクリックして、「LINE Official Account Manager」のページを開いてください。

画面左のメニューから「設定」-「応答設定」を選択してください。



「応答設定」および「チャットの応答方法」を次の画面と同じ設定にしてください。



Copyright © 2024 YITJC All Rights Reserved.

Copyright © 2024 YITJC All Rights Reserved.

10.7 LINE Bot～Python 環境構築（Windows）～

AWS を使用した LINE Bot プログラムは Windows PC で開発します。ここでは、Windows PC に Python 開発環境を構築する手順について説明します。

手順①～Python インストール～

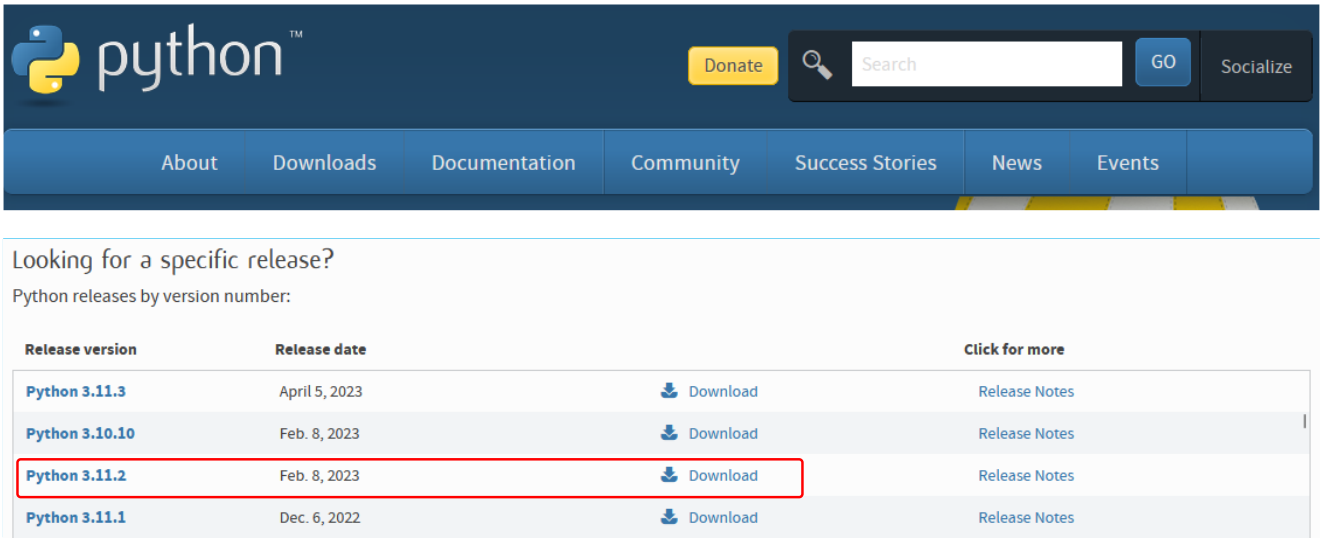
Windows PC に Python 開発環境を構築するため、Python をインストールします。

次の URL にアクセスしてインストーラをダウンロードします。

[URL] https://www.python.org/downloads/

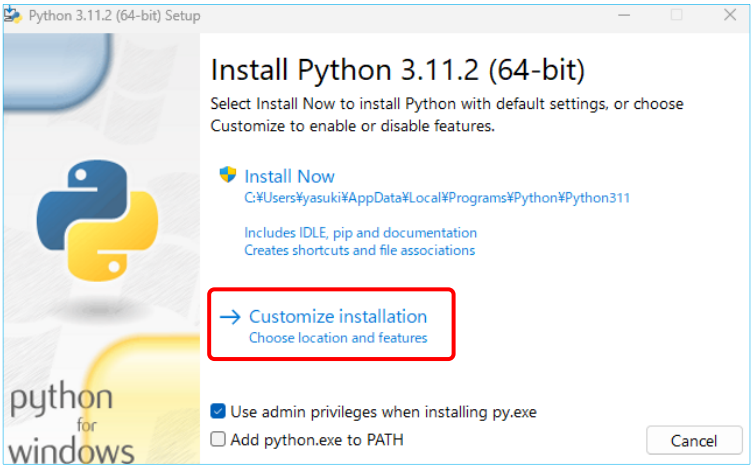
Raspberry Pi と同じバージョン「Python3.11.2」をダウンロードします。

なお、ダウンロードしたファイルを共有サイト「3_Software」に保存しておきます。



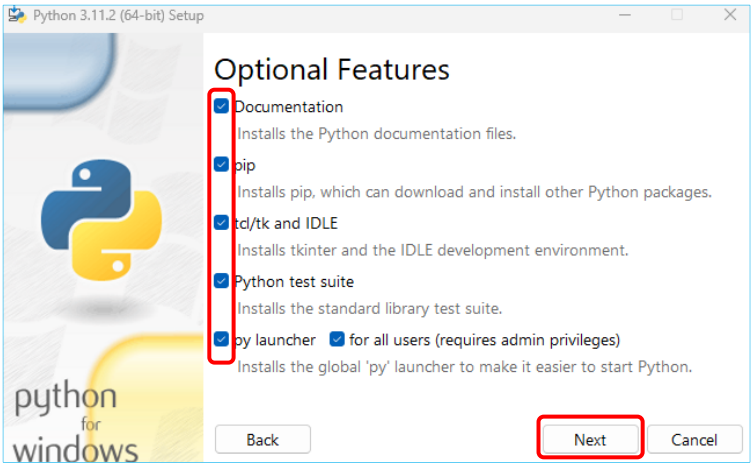
ダウンロードしたインストーラ「python-3.11.2-amd64.exe」ファイルを実行してください。

「Customize installation」を選択してください。

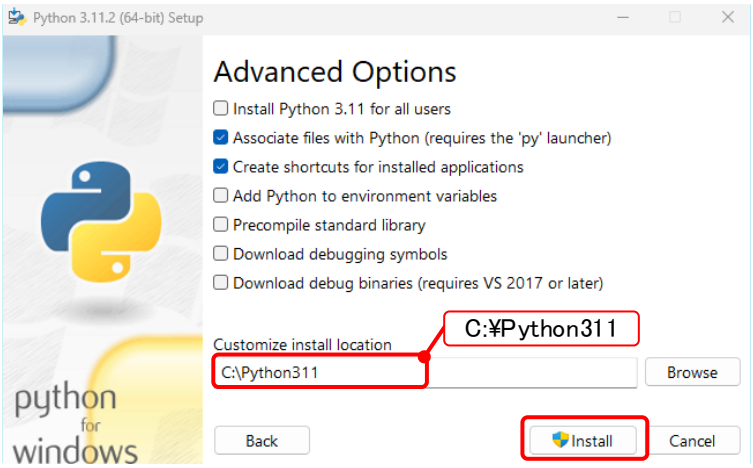


Copyright © 2024 YITJC All Rights Reserved.

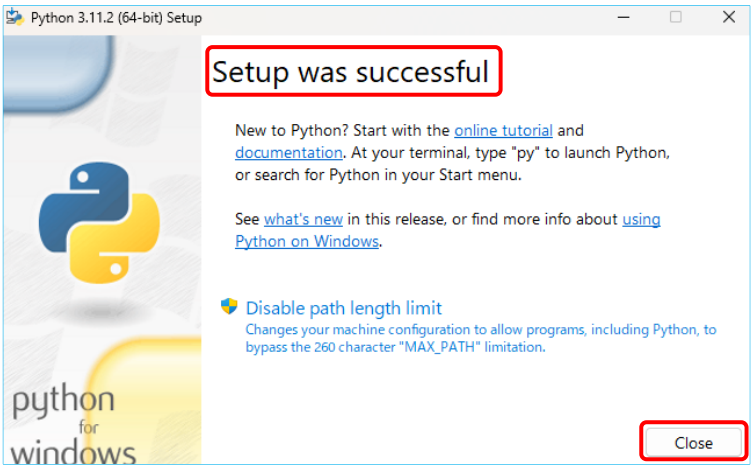
項目がすべてチェックされていることを確認してから「Next」をクリックしてください。



インストール先（Customize install location）を「C:¥Python311」に変更してから「Install」をクリックしてください。



「Python3.11」のインストールが成功すれば、次の画面が表示されます。



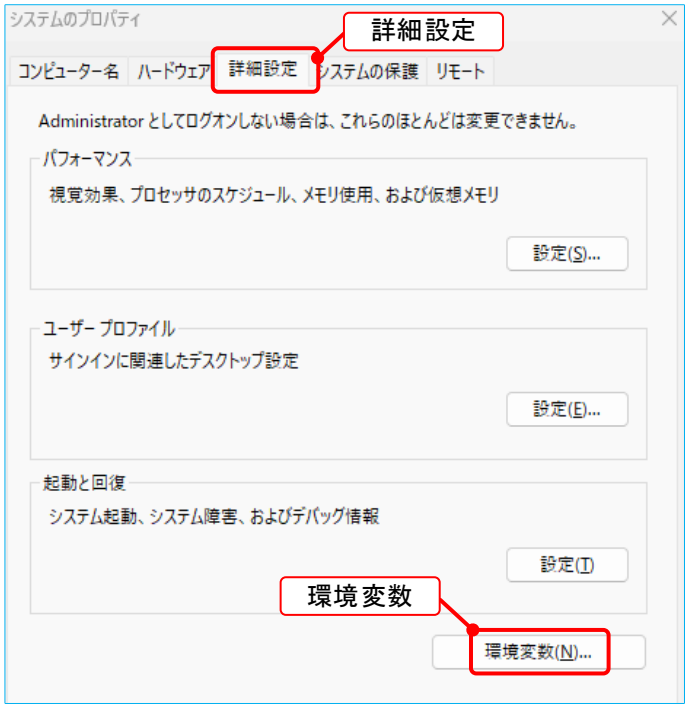
Copyright © 2024 YITJC All Rights Reserved.

手順②～環境変数設定～

Windows PC の環境変数を編集するため、「システムの詳細設定」をクリックしてください。

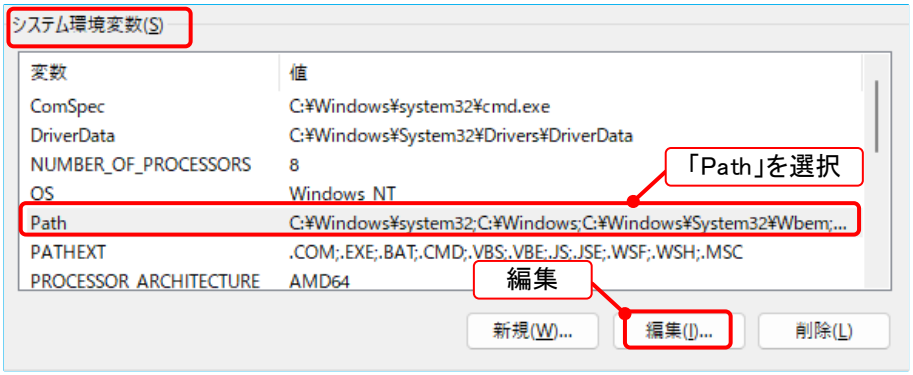
次の画面は Windows スタートボタンを「右クリック」-「システム」を選択すると、表示されます。

「詳細設定」タブの「環境変数」をクリックしてください。

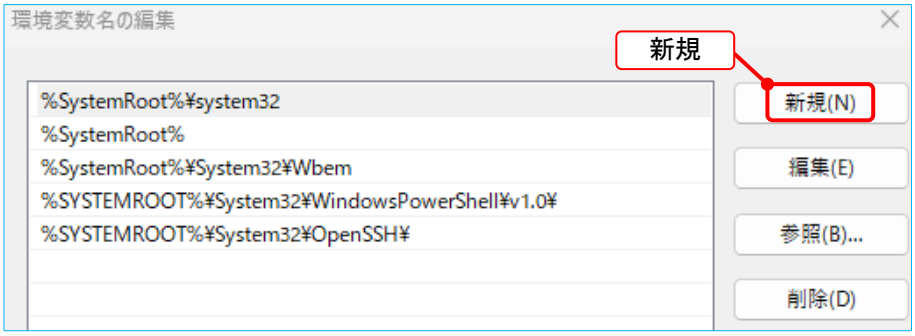


Copyright © 2024 YITJC All Rights Reserved.

「Path」を選択し、「編集」をクリックしてください。

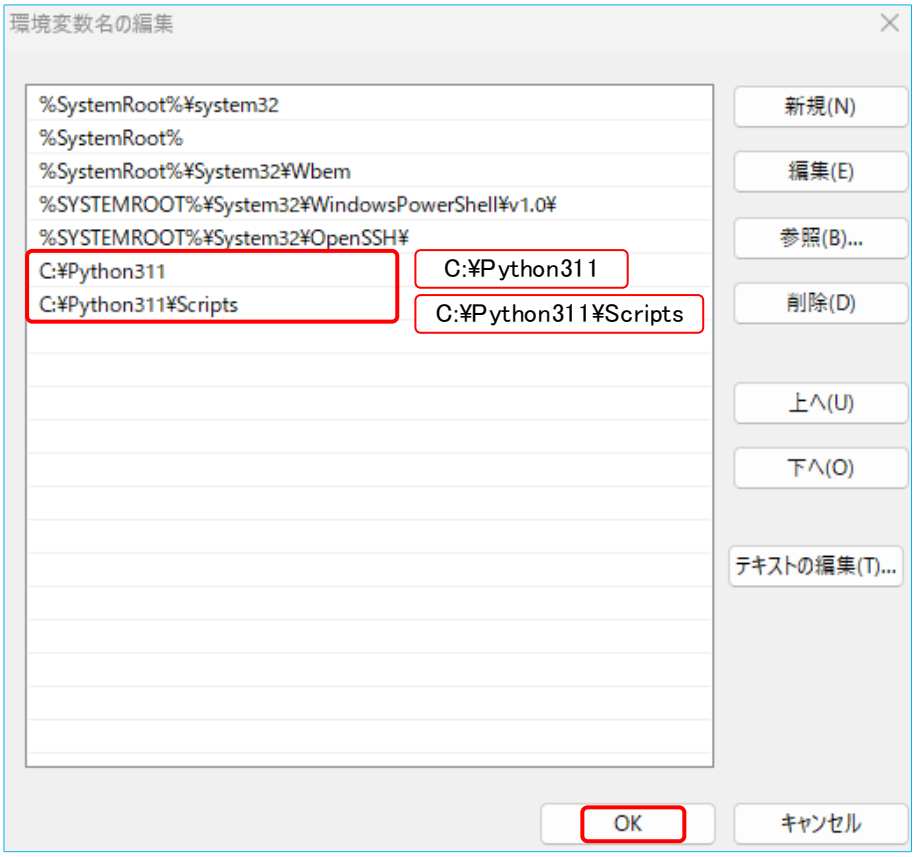


「新規」をクリックしてください。



「C:¥Python311」

「C:¥Python311¥Scripts」を追加してから「OK」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

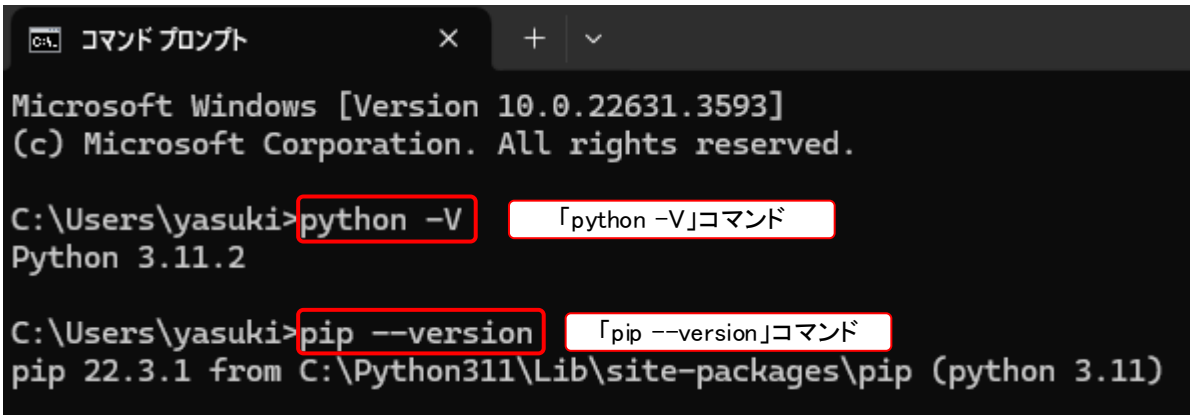
手順③～Python インストール確認～

Windows PC スタートメニューから「コマンドプロンプト」を起動してください。

Python および pip のバージョンを確認することができれば、インストール完了です。

C:¥Users¥User >python -V

C:¥Users¥User >pip --version



Copyright © 2024 YITJC All Rights Reserved.

10.8 LINE Bot～Bot プログラム作成～

LINE Bot を開発するため Messaging API 対応の公式 SDK（ソフトウェア開発キット）を使用します。

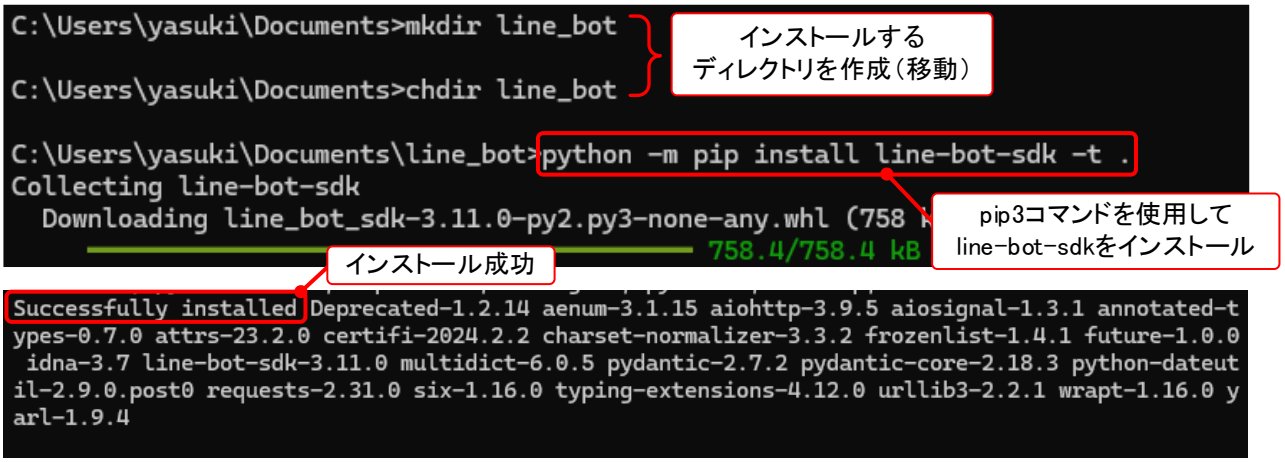
LINE Messaging API SDK には、ライブラリやツール、サンプルコードなどが含まれています。

公式 SDK を使用することで、LINE Bot の開発期間を短期できます。

手順①～LINE Messaging API SDK インストール～

インストール先のフォルダへ移動して、次のコマンドを実行します。

C:¥Users¥User¥Documents¥line_bot>python -m pip install line-bot-sdk -t .



手順②～LINE Bot プログラム作成～

手順①で「line-bot-sdk」をインストールしたフォルダに AWS Lambda で実行する LINE Bot プログラムを保存します。

[対象プログラム]

lambda_function.py

（共有サイトにある「lambda_function_bot.py」ファイルを編集して使用してください。）

～注意～

Lambda プログラム中に「チャネルシークレット」と「チャネルアクセストークン」を指定する箇所があります。

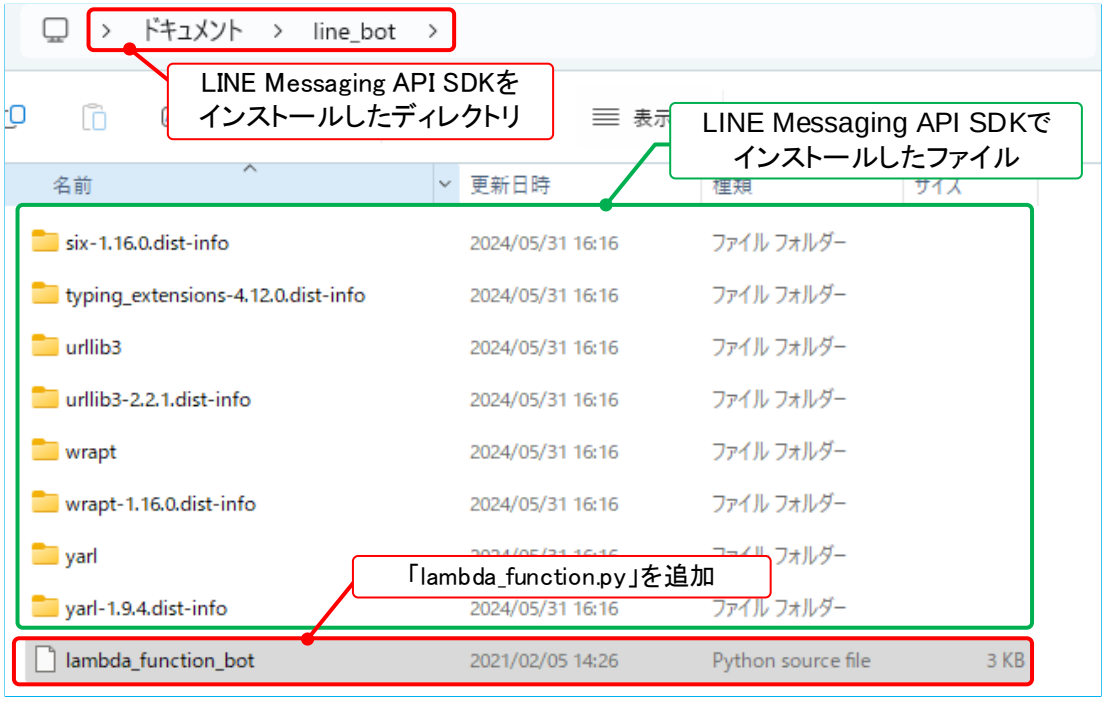
「10.6 LINE Messaging API 設定」で取得した値に書き換えてください。

[プログラム編集個所]

```
#12行目　チャネルアクセストークン
CHANNEL_ACCESS_TOKEN = 'xxxxxxx'
#14行目　チャネルシークレット
CHANNEL_SECRET = 'xxxxxxx'
```

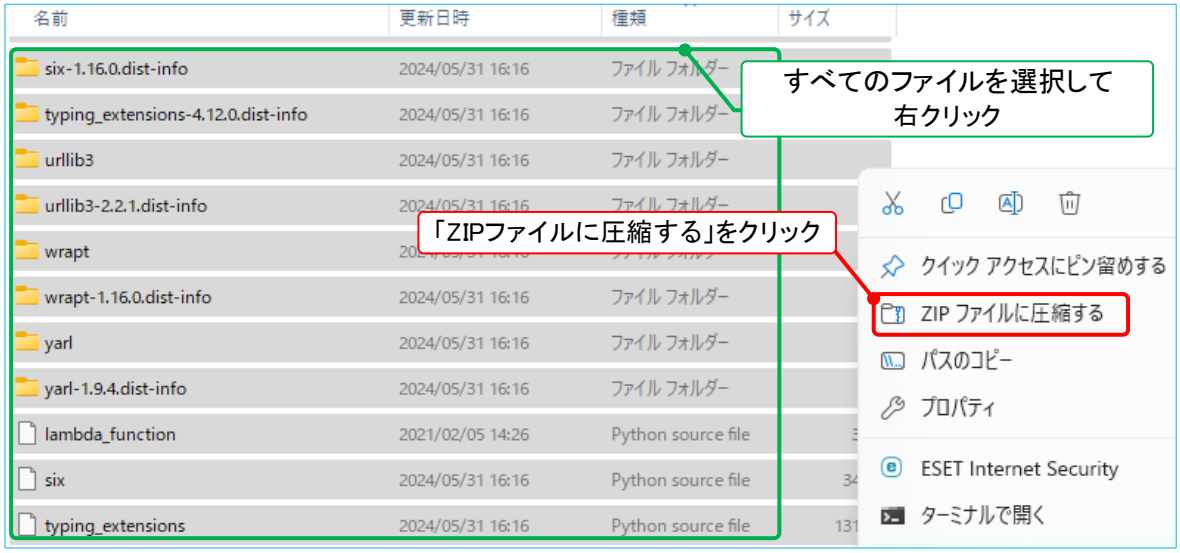
Copyright © 2024 YITJC All Rights Reserved.

手順①で「line-bot-sdk」をインストールしたフォルダに「lambda_function.py」を追加した状態です。

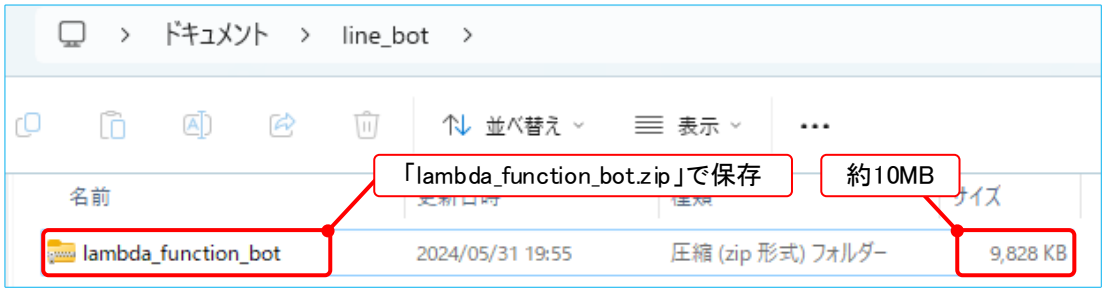


手順③～zip ファイル作成～

AWS Lambda にアップロードするため、zip ファイルを作成します。作業フォルダにあるファイルすべてを選択して右クリックしてください、続いて「ZIP ファイルに圧縮する」をクリックしてください。



zip ファイルは、「lambda_function_bot.zip」のファイル名で保存してください。



Copyright © 2024 YITJC All Rights Reserved.

10.9 LINE Bot～Bot プログラム保存～

Windows PC で作成した Lambda 関数ファイル一式「lambda_function_bot.zip」は、ファイルサイズが約 10MB です。10MB を超えるファイルは、AWS Lambda に直接アップロードできません。そのため、AWS S3（Simple Storage Service）経由でアップロードする必要があります。

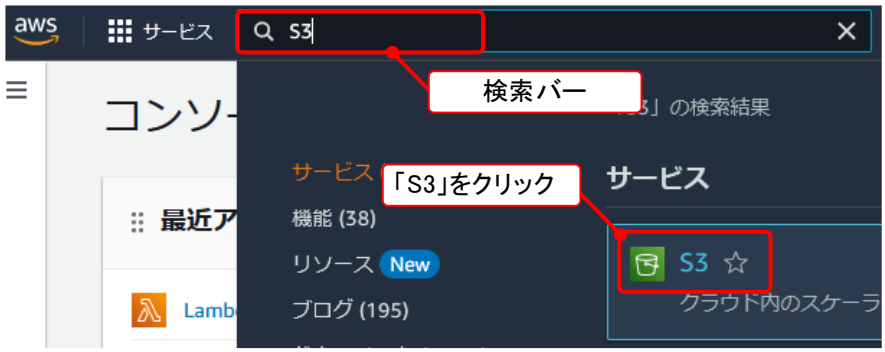
ここでは、Lambda 関数ファイルを AWS S3 にアップロードする手順について説明します。

手順①～AWS S3 バケット作成～

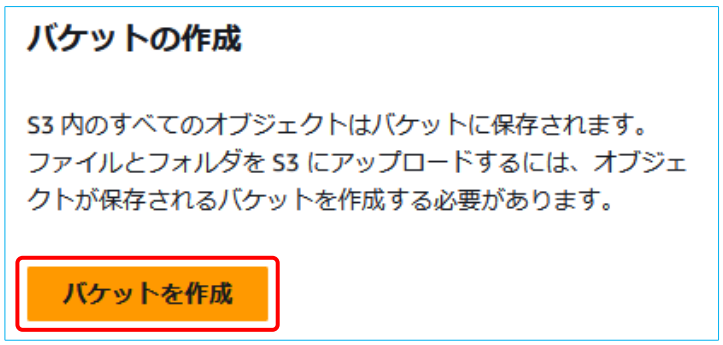
次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「S3」を検索してください。



「S3」をクリックすると次の画面が表示されるので、「バケットを作成」をクリックしてください。

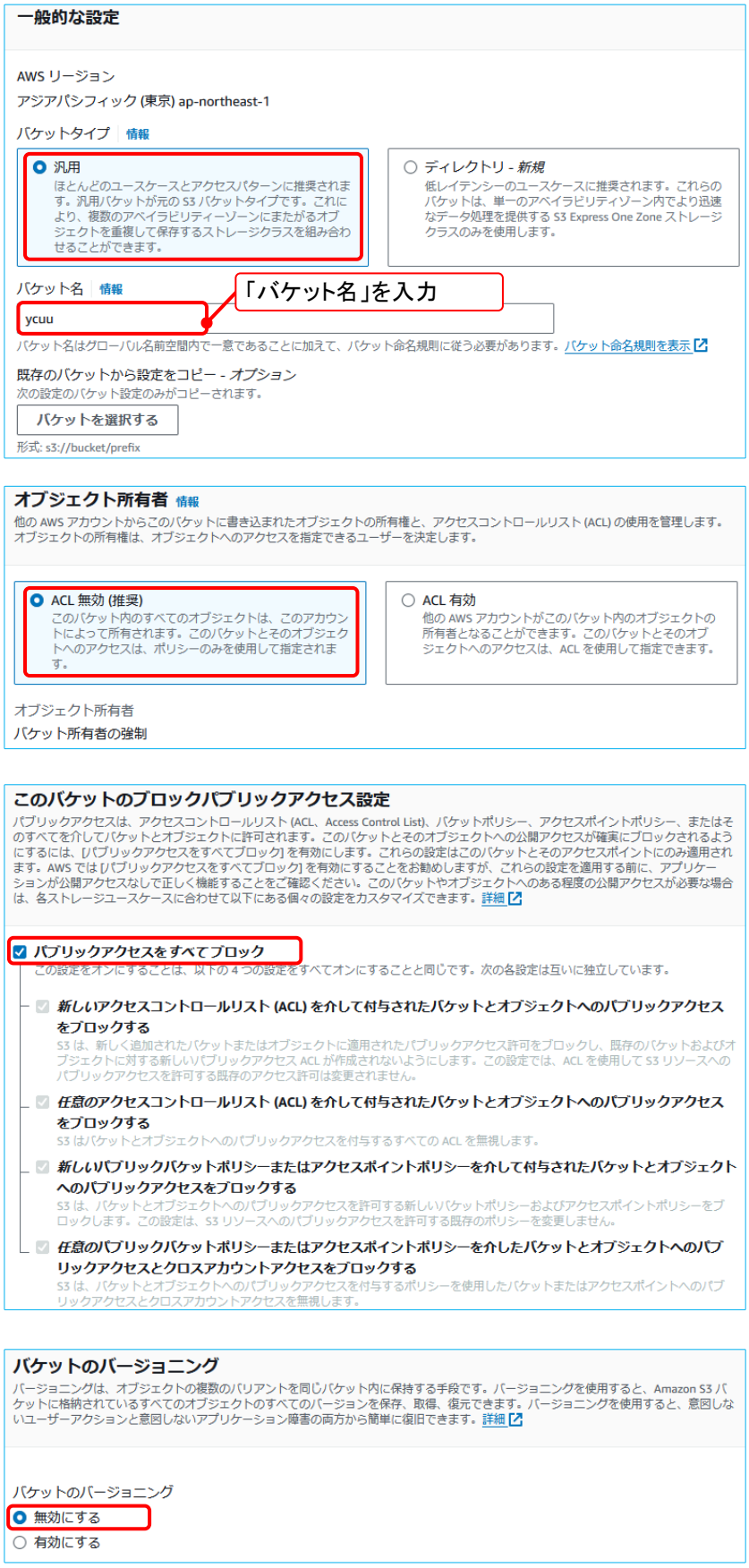


～補足～

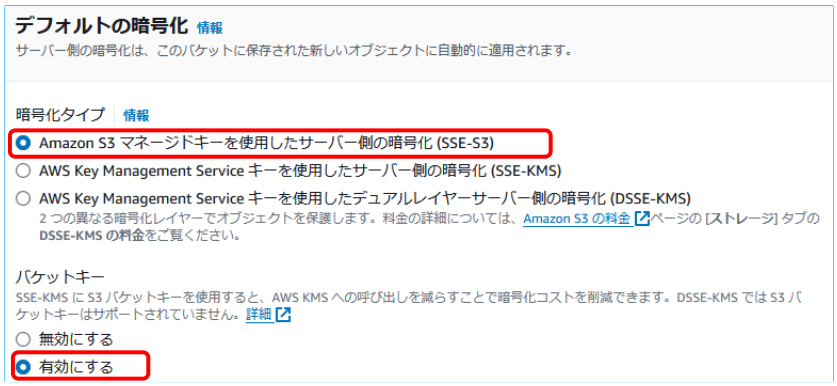
「バケット」とは、AWS S3 に保存されるオブジェクト（ファイル）のコンテナです。フォルダやディレクトリとは異なり、入れ子構造にはできません。1 つのアカウントで最大 100 個のバケットを保存できます。

Copyright © 2024 YITJC All Rights Reserved.

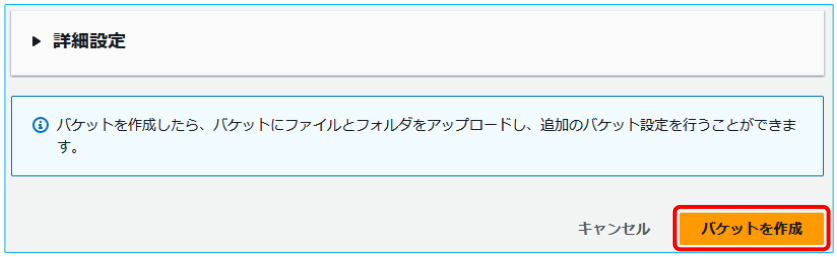
次の画面の内容にしたがって作成する「バケット」の設定をしてください。



Copyright © 2024 YITJC All Rights Reserved.

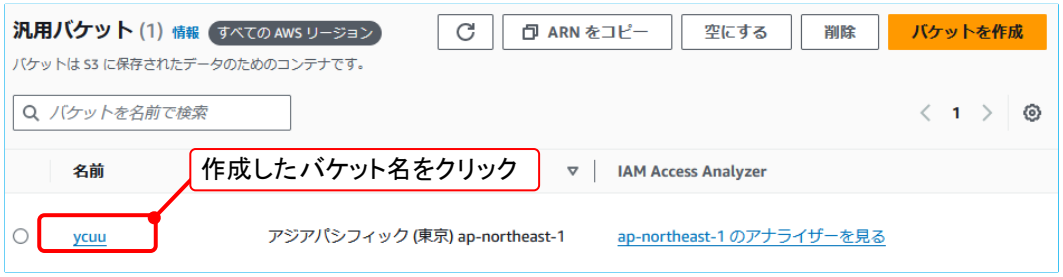


最後に「バケットを作成」をクリックしてください。



手順②～ファイルアップロード～

「作成したバケット名」をクリックしてください。

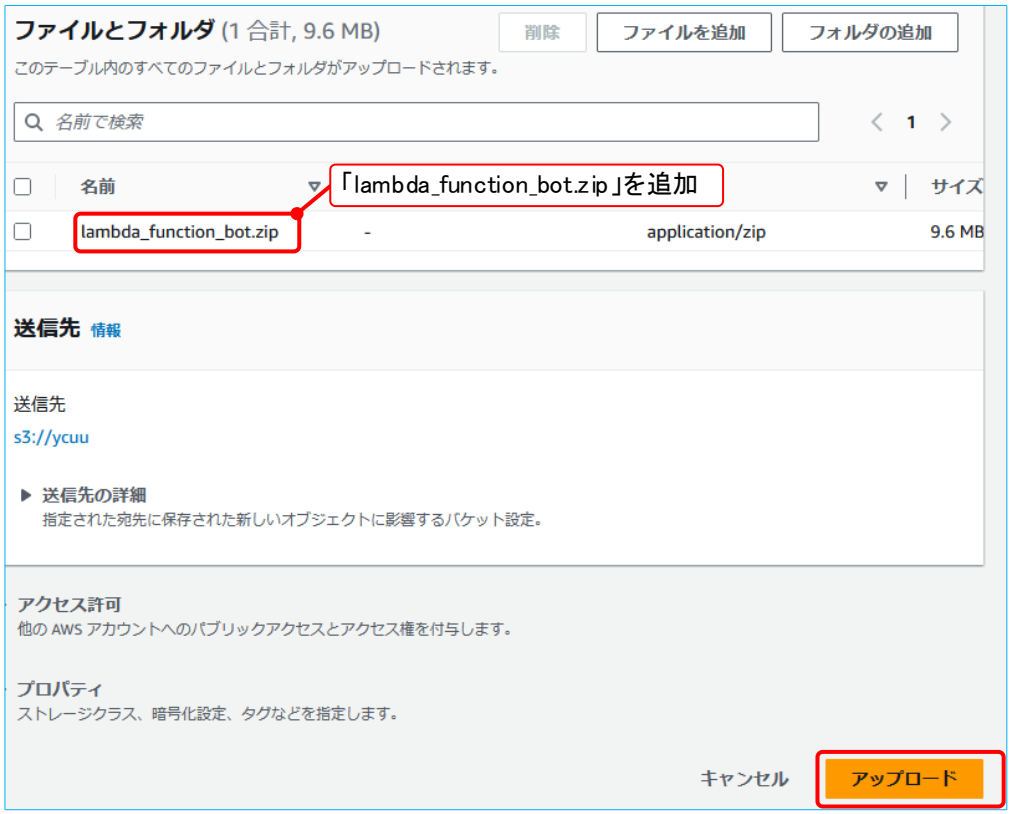


「アップロード」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「ファイルを追加」をクリックしてください。

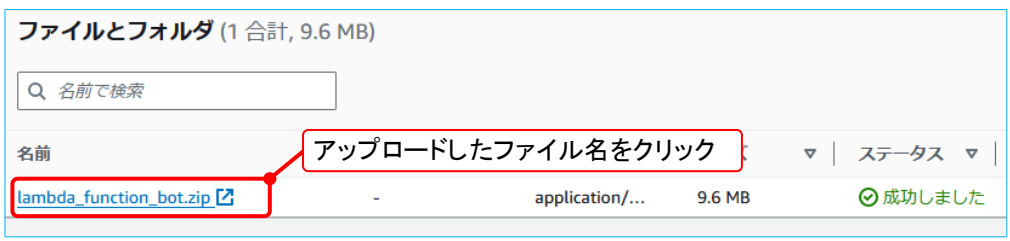
「lambda_function_bot.zip」ファイルを追加してから「アップロード」をクリックしてください。



手順③～オブジェクト URL 取得～

AWS S3 にアップロードしたオブジェクト（zip ファイル）の URL 情報を取得します。

アップロードしたファイル名をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「オブジェクト URL」の情報は使用するので、**メモ帳などに記録してください**。

Copyright © 2024 YITJC All Rights Reserved.

10.10 LINE Bot～Bot プログラムデプロイ～

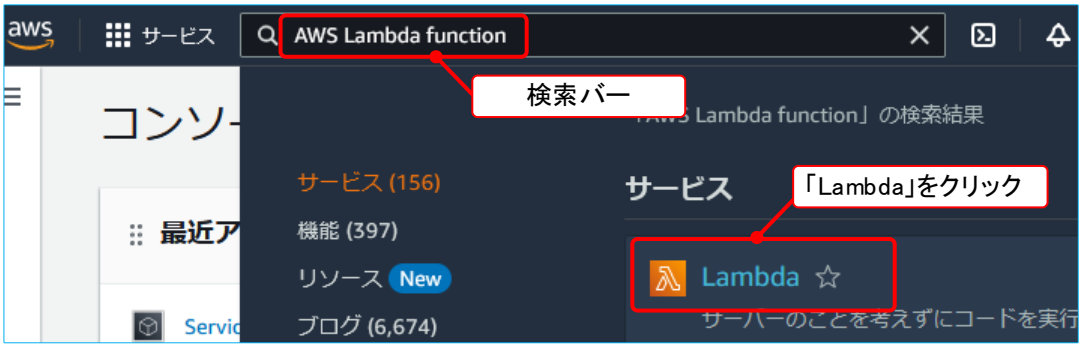
Lambda 関数のファイル一式「lambda_function_bot.zip」を Lambda に配置して利用できる状態にすることをデプロイ（Deploy）と呼びます。AWS Lambda は、クラウドで Function（関数・機能）の実行環境を提供するサービスです。ここでは AWS Lambda に LINE Bot プログラムをデプロイする方法について説明します。

手順①～Lambda 関数作成～

次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「AWS Lambda Function」を検索してください。

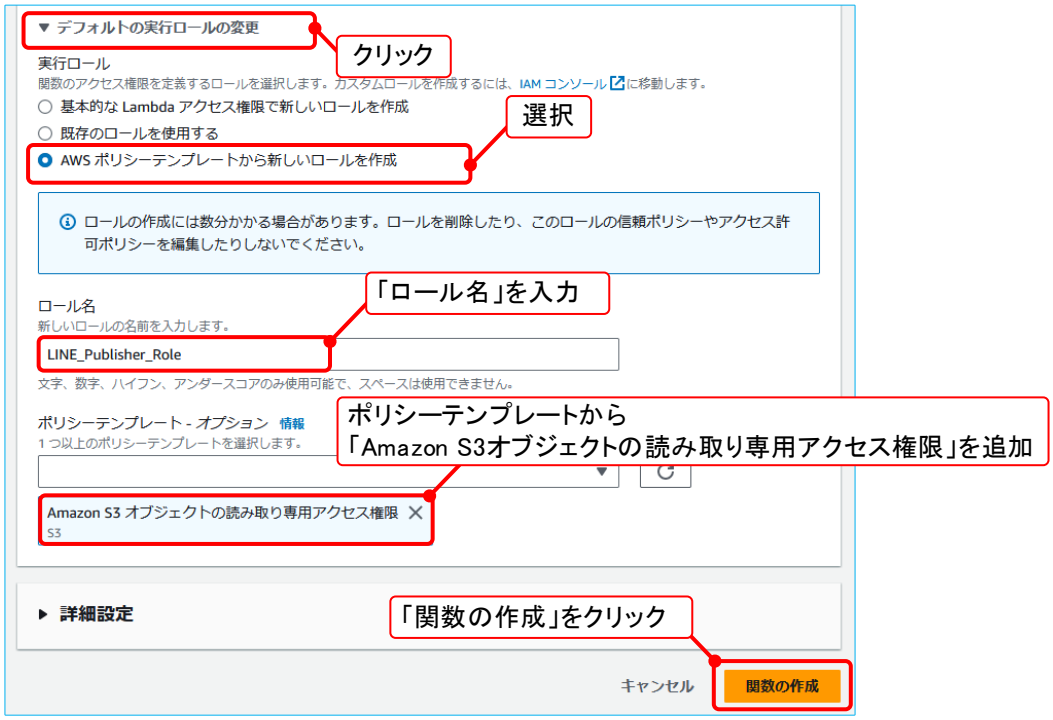


「Lambda」をクリックすると次の画面が表示されるので、「関数の作成」をクリックしてください。

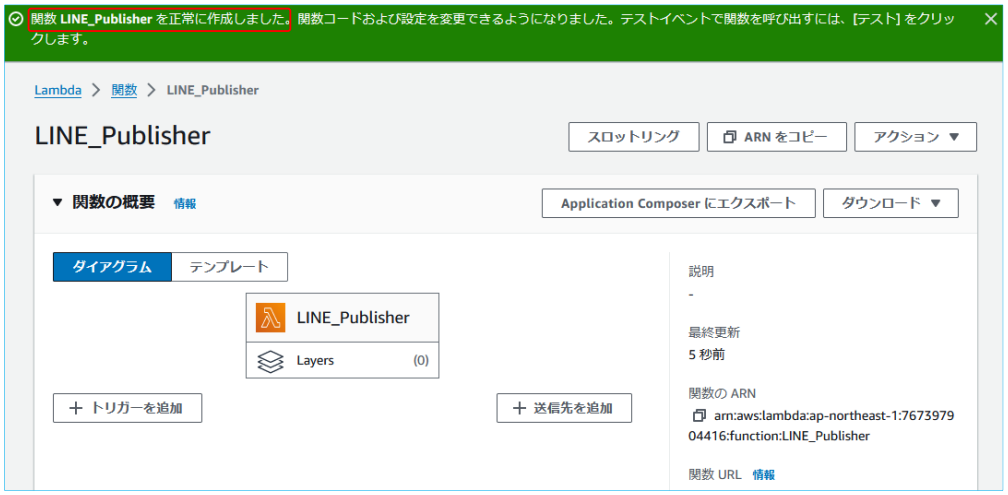
画面の内容にしたがって、関数を作成してください。

Copyright © 2024 YITJC All Rights Reserved.

「デフォルトの実行ロールの変更」をクリックし、「AWS ポリシーテンプレートから新しいロールを作成」を選択してください。「ロール名」を入力し、ポリシーテンプレートから「Amazon S3 オブジェクトの読み取り専用アクセス制限」を選択してください。最後に「関数の作成」をクリックしてください。



関数が正常に作成されると、次の画面が表示されます。



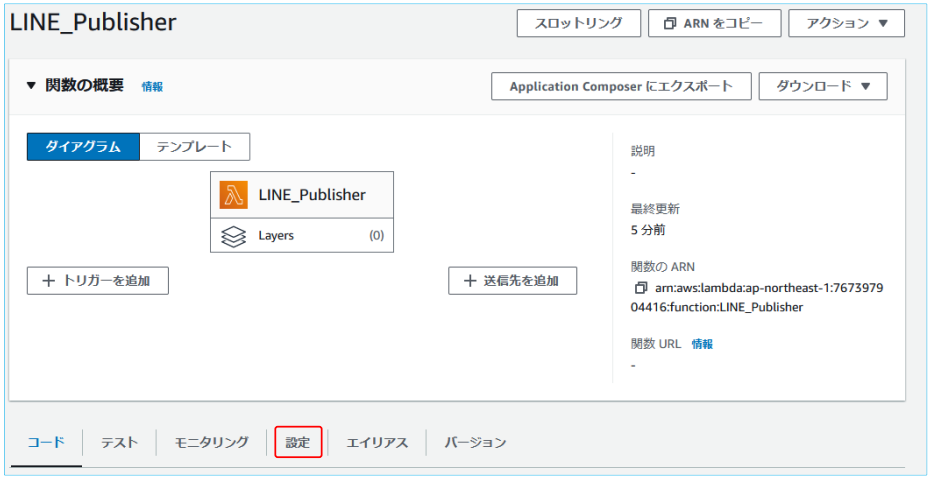
Copyright © 2024 YITJC All Rights Reserved.

手順②～IAM Role の作成～

「IAM（Identity and Access Management） Role」は、Lambda 関数が別のサービスにアクセスするときのアクセス権限を設定（管理）するサービスです。

「対話型システム」の構成では、Lambda 関数は AWE IoT サービスにアクセスします。AWS IoT サービスには Raspberry Pi がモノとして登録されているため、Lambda 関数の処理命令にしたがって Raspberry Pi を制御できます。ここでは、IAM Role を作成する手順について説明します。

「設定」タブをクリックしてください。



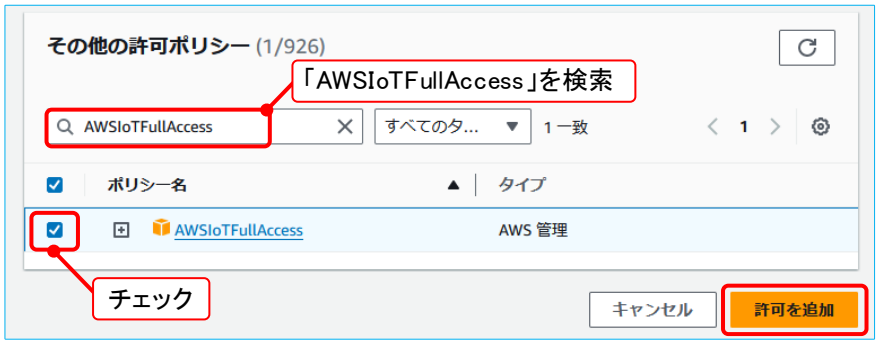
「アクセス権限」を選択してから「ロール名」をクリックしてください。

新たに「Identity and Access Management（IAM）」のウィンドウが開きます。

「許可を追加」を選択してから「ポリシーをアタッチ」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「AWSIoTFullAccess」を検索してください。
検索結果に表示された「AWSIoTFullAccess」のチェックボックスをチェックしてください。
最後に「許可を追加」をクリックしてください。



ポリシー「AWSIoTFullAccess」が正常にアタッチされると次の画面が表示され、「許可ポリシー」の一覧に「AWSIoTFullAccess」が追加されます。

ポリシーの追加が完了したので、「Identity and Access Management（IAM）」のウィンドウは閉じてください。

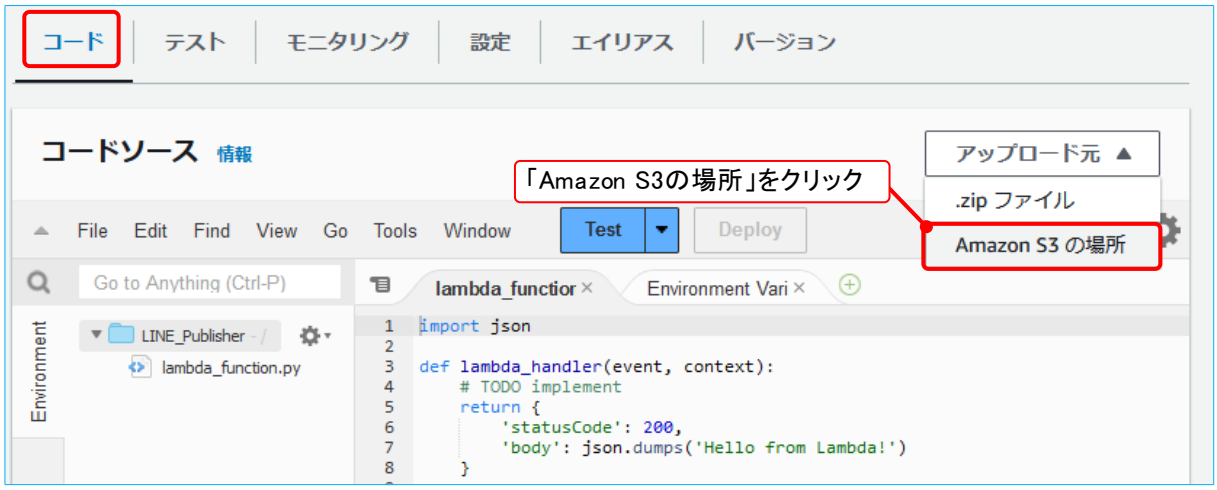
手順③～zip ファイルアップロード～

Lambda 関数ファイル一式「lambda_function_bot.zip」を AWS Lambda にアップロードします。

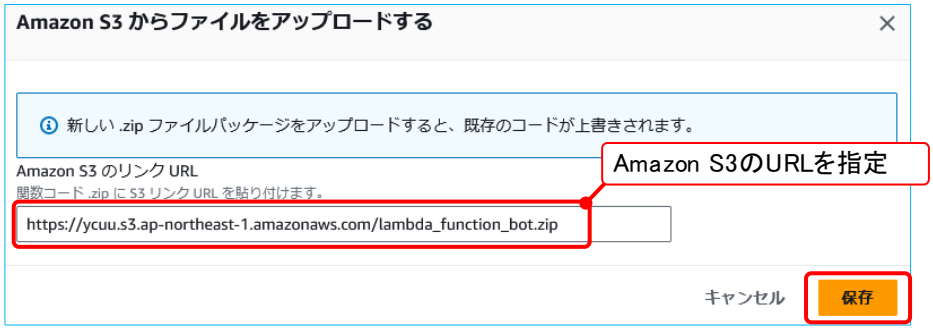
アップロードする前に、「実行ロール」に「AWS IoT」が追加されていることを確認します。
AWS Lambda のウィンドウに戻り、リロード（ブラウザを更新）してください。

Copyright © 2024 YITJC All Rights Reserved.

「コード」タブを選択し、「アップロード元」から「Amazon S3 の場所」をクリックしてください。



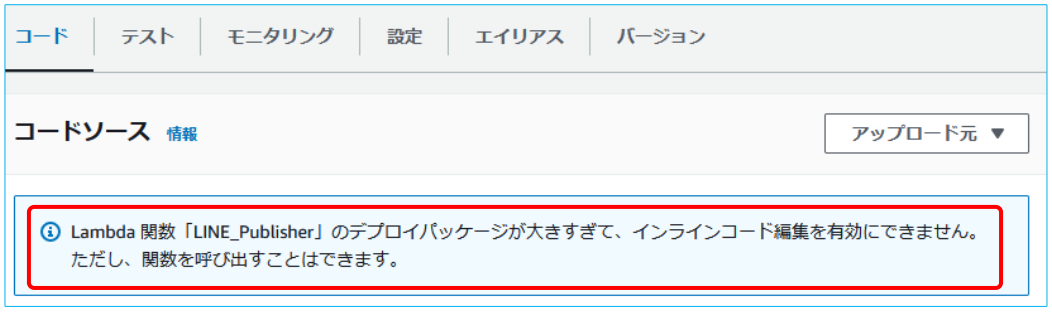
「10.9 Bot プログラム保存」で取得した Amazon S3 の URL を入力してから「保存」をクリックしてください。



Lambda 関数ファイル一式のアップロードにより Lambda 関数は利用できる状態になりました。つまり、「デプロイ」は完了です。

重要～Lambda 関数の編集～

Lambda 関数「lambda_function.py」は、AWS Lambda の画面で編集できます。しかし、今回のケースはアップロードした関数ファイル一式の容量が大きいため、次のメッセージが表示されてしまいます。そのため、ローカル環境（Windows PC）で Lambda 関数を編集しなければなりません。



編集の手順を確認しておきます。

手順① 「lambda_function.py」関数を編集

手順② 「lambda_function.py」およびライブラリ一式を Zip ファイルに保存

手順③ Zip ファイルを AWS S3 にアップロードし、オブジェクトファイルの URL 情報を取得

手順④ AWS Lambda に Lambda 関数一式（zip）ファイルをアップロード

Copyright © 2024 YITJC All Rights Reserved.

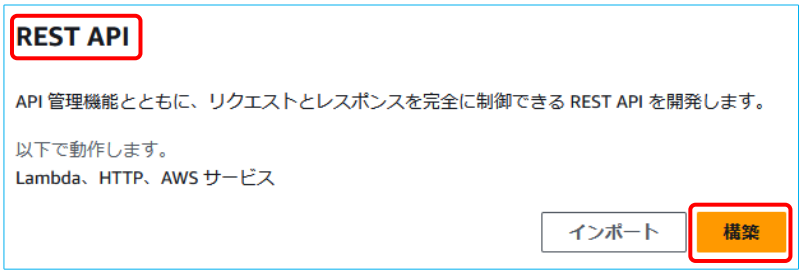
10.11 LINE Bot～API Gateway～

API Gateway は Lambda 関数を Invoke（呼び出す）ために使用します。ここでは、API Gateway の設定方法について説明します。

手順①～REST API 構築～

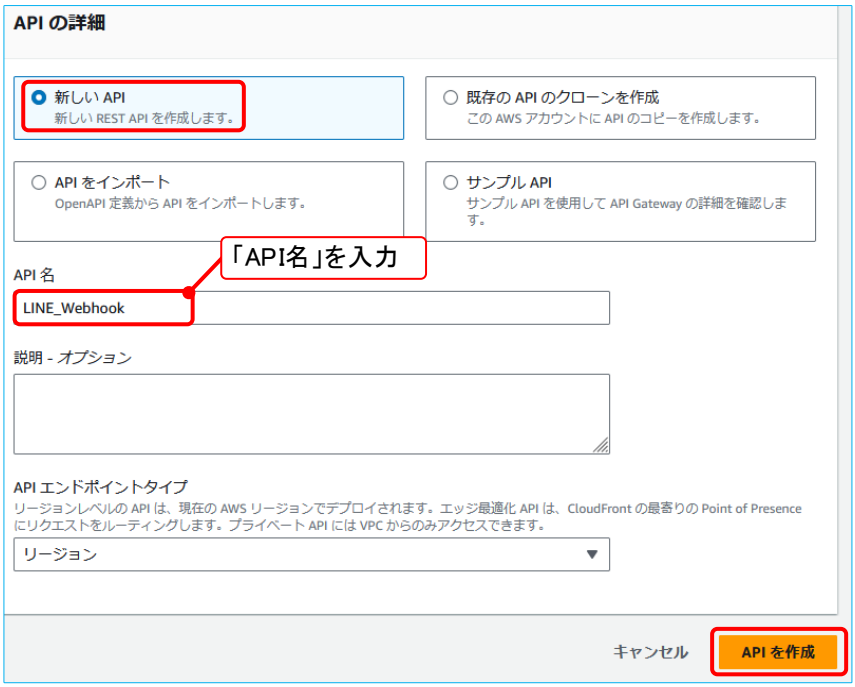
画面上部の検索バーを使用して、「API Gateway」を検索してください。

「API Gateway」をクリックすると次の画面が表示されるので、REST API の「構築」をクリックしてください。



手順②～API 作成～

「新しい API」を選択し、「API 名」を入力してから「API を作成」をクリックしてください。



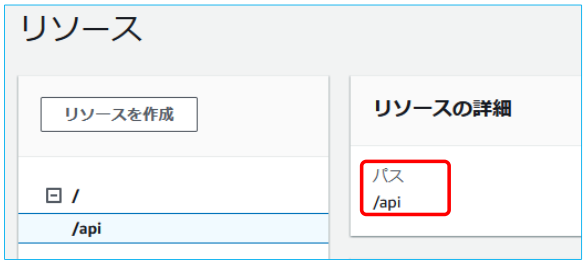
Copyright © 2024 YITJC All Rights Reserved.

手順③～リソース作成～

「リソースを作成」をクリックしてください。

リソース名に「api」と入力してから「リソースを作成」をクリックしてください。

リソースパスは何も入力しない状態で、「/api」が自動で作成されます。



リソースパスは、LINE Messaging API で設定する「Webhook URL」の末尾に使用します。

手順④～メソッド作成～

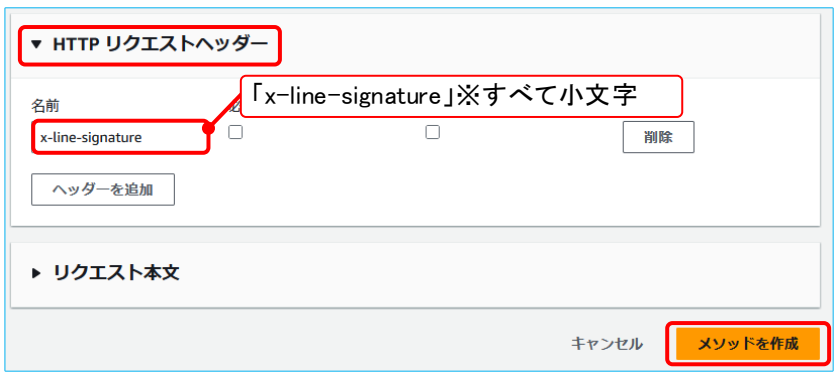
「メソッドを作成」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「POST」を選択したあと、次の設定を行ってください。

- 統合タイプ:「Lambda 関数」を選択
- Lambda プロキシ統合の使用: チェック
  ※プロキシ統合の使用に関しては、プログラム解説で説明します。
- Lambda 関数:作成した「Lambda 関数」の名前を入力

「HTTP リクエストヘッダー」を選択し、「x-line-signature」を追加してください。
最後に「メソッドを作成」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

補足～LINE Messaging API 署名検証～

HTTP リクエストヘッダーの「x-line-signature」は、「**署名検証**」のために設定しています。LINE Messaging API Reference に次の記述があります。

x-line-signature Request Header に入っている Signature を検証することによって、
リクエストが LINE Platform から送信されたものであることを確認する必要があります。
検証は以下の手順で行います。

1. *Channel Secret を秘密鍵として、HMAC-SHA256 アルゴリズムにより Request Body のダイジェスト値を得る。*
2. *ダイジェスト値を BASE64 エンコードした文字列が、Request Header に付与された Signature と一致することを確認する。*

リクエストが LINE ユーザーから送信されたことを確認するため、LINE Server（Bot Server）でリクエストヘッダーの「x-line-signature」に含まれる署名を検証しています。つまり、LINE ユーザーの LINE Messaging API を偽装して、システム側（AWS サービス）からのレスポンスを受け取るという悪意を防いでいます。

注意～x-line-signature～

2021 年 2 月から「LINE Messaging API Reference」に次の情報が掲載されました。
HTTP リクエストヘッダーの「x-line-signature」は、大文字小文字区別なく扱う（すべて小文字）ように指示されています。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～API デプロイ～

「API をデプロイ」をクリックしてください。

- デプロイされるステージ:「新しいステージ」を選択
- ステージ名を入力

最後に「デプロイ」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～Webhook URL 取得～

LINE Messaging API の Webhook に設定する URL 情報を取得します。

このあと（手順⑦）で使用するので、メモ帳などに記録してください

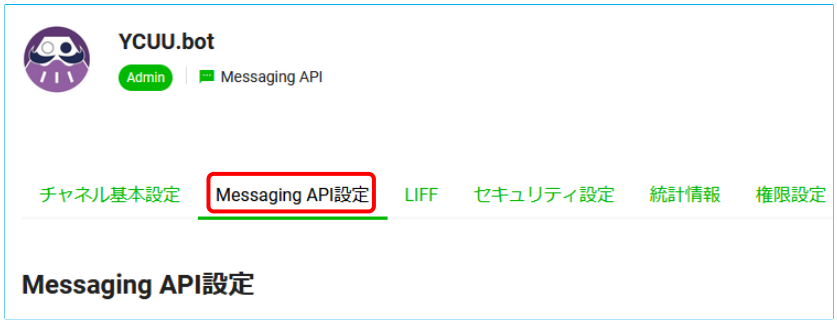
手順⑦～Webhook URL 設定～

LINE Messaging API に手順⑥で取得した Webhook URL を設定します。

次の URL にアクセスして LINE Developers にログインしてください。

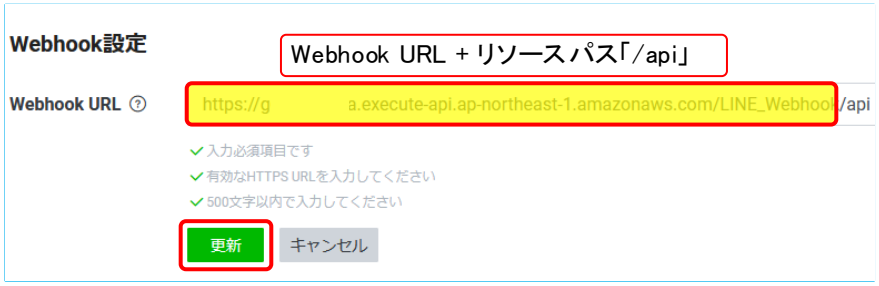
[URL] https://developers.line.biz/ja/

「Messaging API」タブから「Webhook 設定」を選択し、Webhook URL を入力してください。



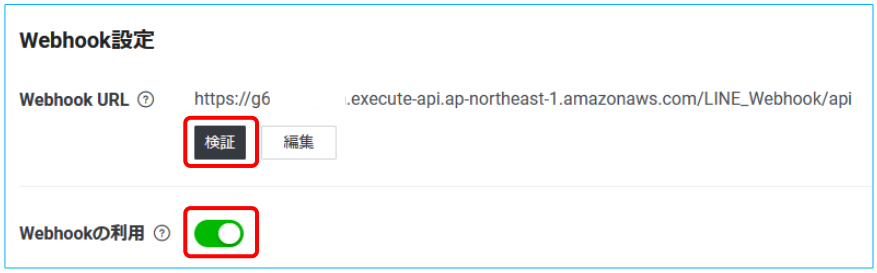
注意～リソースパス末尾追加～

手順④で設定したリソースパス「/api」を Webhook URL の末尾に追加してください。

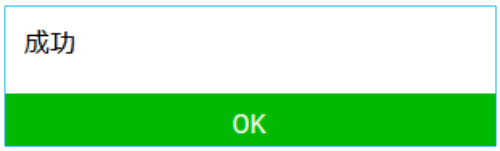


Copyright © 2024 YITJC All Rights Reserved.

「Webhook の利用」をチェックしてから「検証」をクリックしてください。

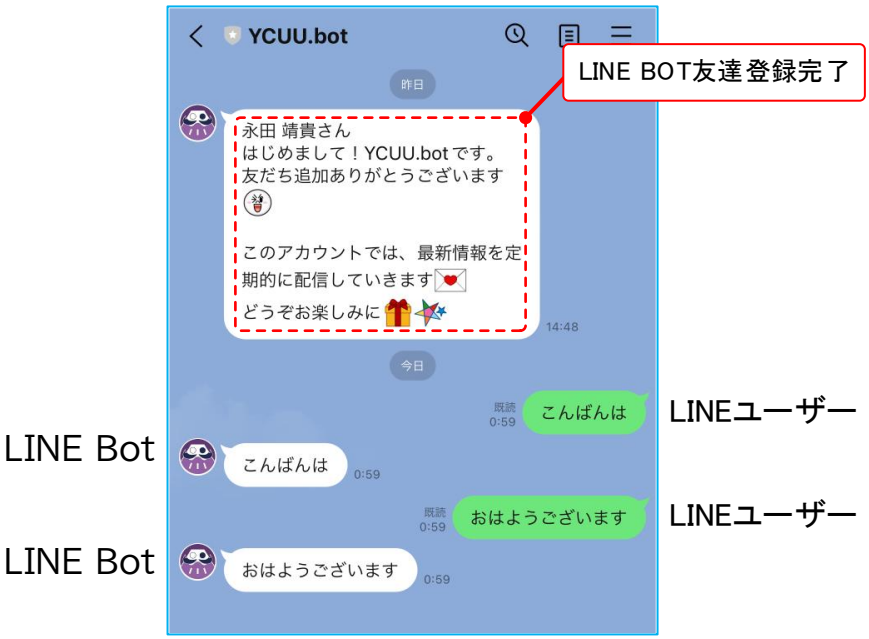


次の画面が表示されれば、Webhook URL の登録完了です。



手順⑧～動作確認～

友達登録した LINE Bot にメッセージを送信してください。LINE ユーザーからの発言に対して自動応答（オウム返し）されることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	from linebot.v3 import WebhookHandler
2	from linebot.v3.exceptions import InvalidSignatureError
3	from linebot.v3.messaging import (
4	ApiClient, Configuration, MessagingApi,
5	ReplyMessageRequest, TextMessage
6	)
7	from linebot.v3.webhooks import (
8	MessageEvent, TextMessageContent
9	)
10	
11	#チャネルアクセストークン
12	CHANNEL_ACCESS_TOKEN = 'Q6/cA/gUE6yOYR7+xxxxxxxxxxbZ9cahLc0TK4/71gdRm0v42W2f+JrlP6TjVQ+OQIOf29CAU/Hetv/t7+oWfNWV/eBXirpzgh1fB99QQTayCTuAcvqEO8iH2L7jACAcSKywehFZq3j/6UdZhhXnmAdB04t89/1O/w1cDnyilFU='
13	#チャネルシークレット
14	CHANNEL_SECRET = 'e6bdaxxxxxxxxxx963eee04486668ba4'
15	
16	configuration = Configuration(access_token=CHANNEL_ACCESS_TOKEN)
17	api_client = ApiClient(configuration=configuration)
18	line_bot_api = MessagingApi(api_client)
19	handler = WebhookHandler(CHANNEL_SECRET)
20	
21	import logging
22	logger = logging.getLogger()
23	logger.setLevel(logging.ERROR)
24	
25	def lambda_handler(event, context):
26	#Line Uer ID取得するためeventを表示
27	print(f"{event}")
28	signature = event["headers"]["x-line-signature"]
29	body = event["body"]
30	ok_json = {"isBase64Encoded": False,
31	"statusCode": 200,
32	"headers": {},
33	"body": ""}
34	error_json = {"isBase64Encoded": False,
35	"statusCode": 403,

Copyright © 2024 YITJC All Rights Reserved.

```
36                          "headers": {},
37                          "body": "Error"}
38
39      try:
40          handler.handle(body, signature)
41      except InvalidSignatureError:
42          return error_json
43      return ok_json
44
45  @handler.add(MessageEvent, message=TextMessageContent)
46  def handle_message(event):
47      text = event.message.text
48      #オウム返し
49      line_bot_api.reply_message(
50          ReplyMessageRequest(
51              replyToken=event.reply_token,
52              messages=[TextMessage(text=text)]
53          )
54      )
```

～参考～

ここでは、LINE Messaging API SDK for Python に含まれるさまざまなモジュールを使用しています。モジュールの使い方など、詳しい情報は次の URL に記載されています。

[URL] https://github.com/line/line-bot-sdk-python

～注意～

2023 年 3 月に LINE Messaging API SDK version3（以降 v3）がリリースされました。v3 以降のライブラリは、「linebot.v3」として提供されます。v3 以前のライブラリは「linebot」として提供され、クラスやメソッドの使用方法が異なるため、注意が必要です。本講座は最新のライブラリ「linebot.v3」を使用して対話型システムを構築します。

灰色部は、これまでに説明した内容です。

- 1 行目: linebot.v3 ライブラリの「WebhookHandler」をインポート
- 2 行目: linebot.v3.exceptions ライブラリの「InvalidSignatureError」クラスをインポート
- 3 行目～6 行目: linebot.v3.messaging ライブラリの「ApiClient」、「Configuration」、「MessagingApi」、「ReplyMessageRequest」、「TextMessage」クラスをインポート
  このほか、画像（イメージ）ファイルを扱う「ImageMessage」クラス、ボタンテンプレートを扱う「ButtonsTemplate」クラス、ビデオメッセージを扱う「VideoMessage」クラスなどがあります。
- 7 行目～9 行目: linebot.v3.webhook ライブラリの「MessageEvent」、「TextMessageContent」クラスとインポート

Copyright © 2024 YITJC All Rights Reserved.

- 12 行目: チャネルアクセストークン設定
- 14 行目: チャネルアシークレット設定

  ※「チャネルアクセストークン」および「チャネルシークレット」は、AWS Lambda で環境変数として保存することも可能ですが、ここではプログラム内に記述しています。
- 16 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 17 行目: ApiClient クラスを使用して API クライアントを初期化
- 18 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 19 行目: WebhookHandler クラスを使用してチャネルシークレットを設定
- 21 行目: プログラム実行時に発生したイベントを追跡するためのライブラリ「logging」をインポート
- 22 行目: 現在のロギング情報取得（引数はファイル名を設定）
- 23 行目: ロギングのレベルを「ERROR」に設定

～参考～

Logging モジュールは、プログラム実行時に発生したイベントを追跡するためのライブラリです。ログレベルが 5 段階に設定されており、各レベルの追跡目的は次表のとおりです。デフォルトのレベルは WARNING です。

Level	追跡（記録）目的
DEBUG	動作確認・デバッグ用途
INFO	正常動作の記録用
WARNING	警告を出しておきたい動作の記録
ERROR	エラーなど重大の問題の記録
CRITICAL	システム停止など致命的な問題の記録

- 25 行目～43 行目: Lambda 関数 Handler
- 27 行目: リクエストデータ print 表示

  「f」は、フォーマット済み文字リテラル（f-string）です。

～参考～

リクエストデータについて説明します。

API Gateway の設定を行う際、手順④「メソッドの作成」において、Lambda プロキシ統合を「使用」に設定しました。Lambda プロキシ統合を「使用」した場合、LINE Server（LINE Bot）からのリクエストを API Gateway が受信すると、API Gateway は「統合された」Lambda 関数にリクエストをそのまま渡します。

ただし、リクエストに含まれるパラメータ（情報）の順序は保持されません。（API Gateway が受け取ったリクエストと AWS Lambda が受け取ったリクエストの中身は同じですが、順序が異なります。）

リクエストデータには次の情報が含まれています。

- ✓ リクエストヘッダー
- ✓ クエリ文字列パラメータ
- ✓ URL
- ✓ パス変数
- ✓ Payload
- ✓ API 設定データ（現在のデプロイステージ名、ステージ変数、ユーザーID など）

Copyright © 2024 YITJC All Rights Reserved.

- 28 行目: リクエストヘッダーの「x-line-signature」に含まれる値（署名検証のための値）を取得
- 29 行目: リクエストボディを取得
- 30 行目～33 行目: ok_json 定義
- 34 行目～37 行目: error_json 定義

～参考～

API Gateway が Lambda 関数の応答（レスポンス）を処理するためには、応答が次の JSON 形式である必要があります。ok_json と error_json は、Lambda 関数から API Gateway にレスポンスする出力です。

```
{
    "isBase64Encoded": true|false,
    "statusCode": httpStatusCode,
    "headers": { "headerName": "headerValue", ... },
    "body": "..."
}
```

～参考～

JSON 形式（JavaScript Object Notation）

よく利用される形式は、{ , , , } の中に、変数名と値をコロン(:)で区切ります。

```
{ "name": "Nagata" }
```

カンマ(,)で連結することにより、複数の変数名と値のペアを指定することができます。変数名(name や age)はダブルクォーテーション(")で囲みます。

```
{ "name": "Nagata", "age": 43 }
```

読みやすさのために、インデントをつけて表記することが多いです。ok_json と error_json もインデントをつけて表記しています。インデントは 4 文字スペースや 2 文字スペースがよく使用されます。

```
{
  "name": "Nagata",
  "age": 43
}
```

- 39 行目～40 行目: try ブロック
- 40 行目: WebhookHandler の「handler.handle」メソッドを使用
  署名「Signature」を検証し、問題なければ handle に定義されている関数を呼び出します。
- 41 行目～42 行目: except ブロック
- 42 行目: 署名検証が失敗した場合、error_json を API Gateway にレスポンス
- 43 行目: handle の処理が問題なく終了すれば、ok_json を API Gateway にレスポンス

Copyright © 2024 YITJC All Rights Reserved.

- 45 行目:「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。ここでは、「MessageEvent」、「TextMessage」を指定しています。
  ～参考～
  @handler.add(MessageEvent, message=TextMessageContent)
  def handle_message(event):
  @handler.add(MessageEvent, message=TextMessageContent)は、「**デコレータ**」と呼ばれる記述です。
  handle_message()関数を装飾（decoration）する役割があります。具体的には、特定の条件（メッセージイベント発生かつテキストメッセージ）をトリガとして、handle_message()関数を実行するように設定しています。
  このプログラムの場合、
  LINE ユーザーからメッセージが送信されると、MessageEvent が発生します。
  なおかつ、受信したメッセージがテキストメッセージ（TextMessageContent）である場合に、
  def handle_message(event):関数を実行します。

- 47 行目: LINE ユーザーからの送信メッセージを変数「text」に格納
- 49 行目～54 行目: LINE Bot 動作
  LINE Bot の応答（オウム返し）は LineBotApi の「line_bot_api.reply_message」メソッドを使用します。
  第一引数の「event.reply_token」はイベントの応答に用いるトークンです。
  第二引数には linebot.v3.messaging に定義されている返信用の TextMessage オブジェクトを渡しています。

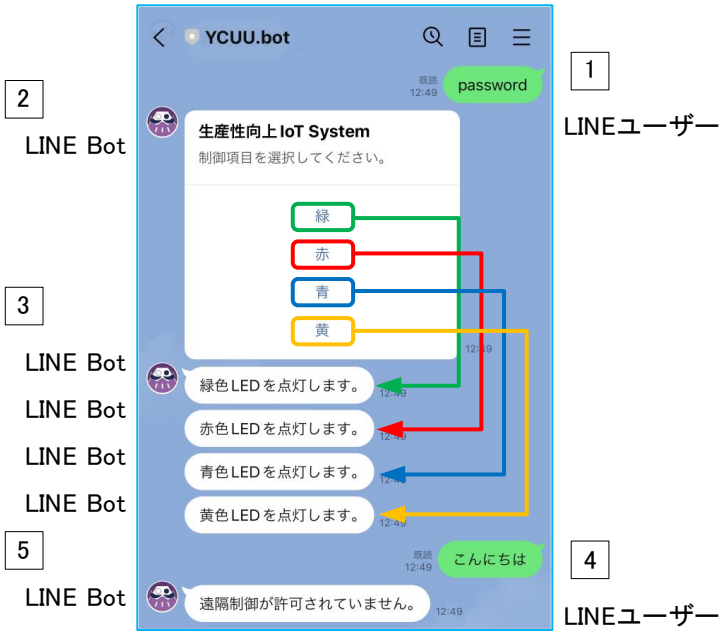
Copyright © 2024 YITJC All Rights Reserved.

10.12 LINE Bot～ボタンテンプレート～

LINE Bot（オウム返しプログラム）を改良して「ボタンテンプレート」を使用した LINE Bot を作成します。ボタンテンプレートを使用した LINE Bot の画面を説明します。

～メッセージの流れ～

1. あらかじめ指定したメッセージ「password」を送信
2. あらかじめ指定したメッセージ「password」を受信すると、ボタンテンプレートが表示されます。
   ※ボタンテンプレートは、最大 4 個のボタンを登録できます。
3. ボタンテンプレートに表示された「ボタン」を押すと、各ボタンに対応したメッセージが自動応答されます。
   ※この段階では、まだ Raspberry Pi による処理は実行されません。自動応答のみです。
4. 「password」以外のメッセージを送信。
5. 「password」以外のメッセージを受信すると、「遠隔制御が許可されていません。」と自動応答されます。



LINE ユーザーからのメッセージに対して LINE Bot が「自動応答」する構成に変わりはありませんが、新たな要素として「ボタンテンプレート」を使用しています。「ボタンテンプレート」は、最大 4 個のボタンを登録できます。

ここでは、各ボタンに対応したメッセージが自動応答されるプログラムについて学習します。

LINE メッセージタイプ、ボタンテンプレートなど、LINE に関する詳しい情報は、次の URL にアクセスして確認してください。

- LINE Messaging API リファレンス
  [URL] https://developers.line.biz/ja/reference/messaging-api/#template-messages
- メッセージタイプ
  [URL] https://developers.line.biz/ja/docs/messaging-api/message-types/

Copyright © 2024 YITJC All Rights Reserved.

手順①～Lambda プログラム作成～

「10.8 Bot プログラム作成」と同じ手順で AWS Lambda で実行する LINE Bot プログラムを作成・保存します。

[対象プログラム]

lambda_function.py

（共有サイトにある「lambda_function_button.py」ファイルを編集して使用してください。）

～注意～

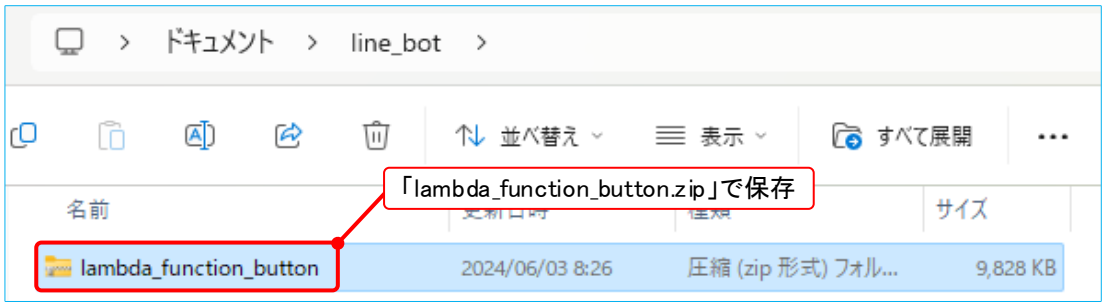
Lambda プログラム中に「チャネルシークレット」と「チャネルアクセストークン」を指定する箇所があります。

「10.6 LINE Messaging API 設定」で取得した値に書き換えてください。

[プログラム編集個所]

```
#12行目　チャネルアクセストークン
CHANNEL_ACCESS_TOKEN = 'xxxxxxx'
#14行目　チャネルシークレット
CHANNEL_SECRET = 'xxxxxxx'
```

zip ファイルは、「lambda_function_button.zip」のファイル名で保存してください。

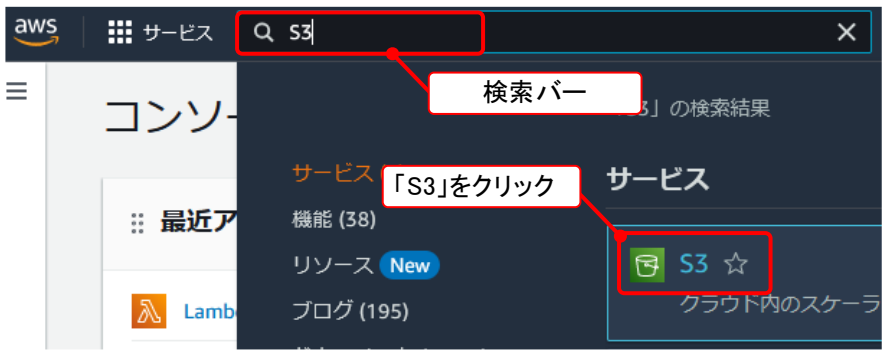


手順②～Lambda 関数ファイルアップロード～

次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

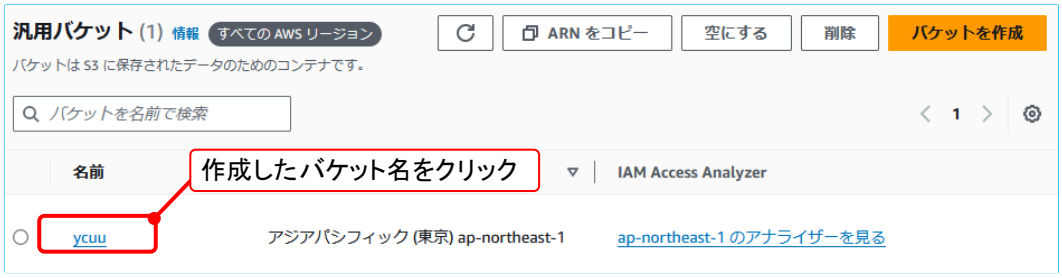
画面上部の検索バーを使用して、「S3」を検索してください。



「10.9 Bot プログラム保存」と同じ手順で AWS S3 に Lambda 関数ファイル一式をアップロードします。

Copyright © 2024 YITJC All Rights Reserved.

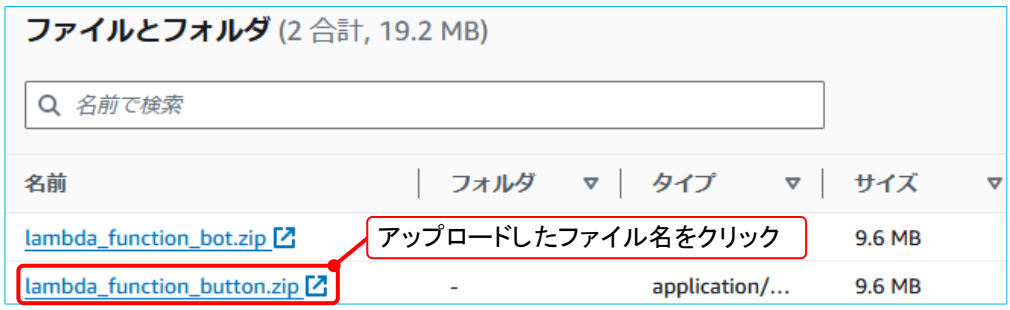
「作成したバケット名」をクリックしてください。



「アップロード」をクリックしてください。

「ファイルを追加」をクリックしてください。

「lambda_function_button.zip」ファイルをアップロードしたあと、アップロードしたファイルの URL 情報を取得します。アップロードしたファイル名をクリックして URL 情報を取得してください。



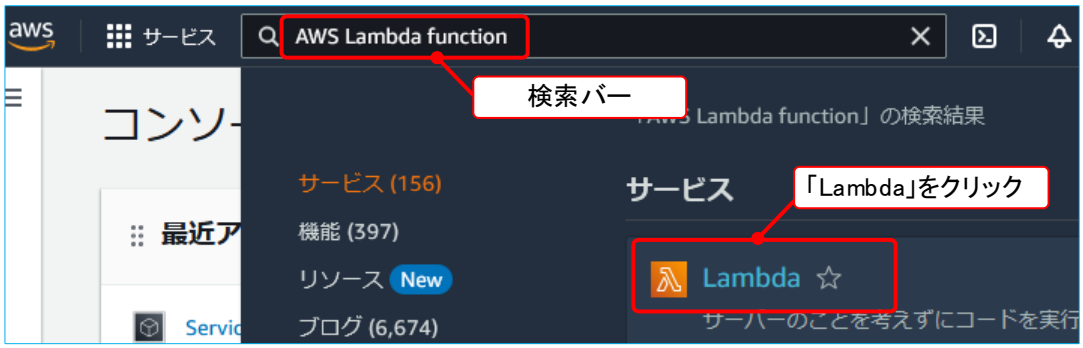
Copyright © 2024 YITJC All Rights Reserved.

手順③～Lambda 関数デプロイ～

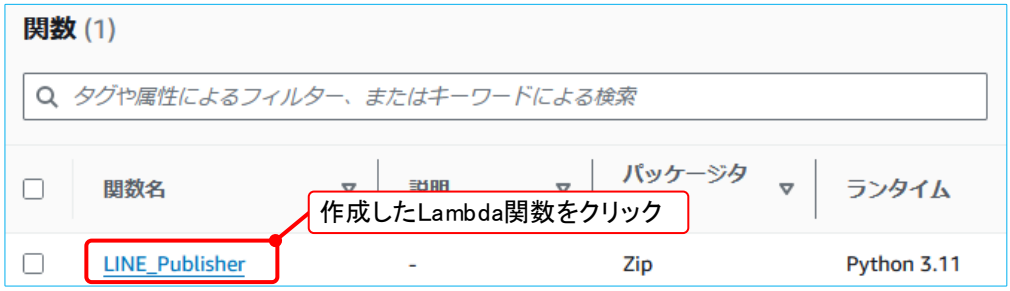
次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「AWS Lambda Function」を検索してください。



作成した「Lambda 関数」をクリックしてください。



「コード」タブを選択し、「アップロード元」から「Amazon S3 の場所」をクリックしてください。

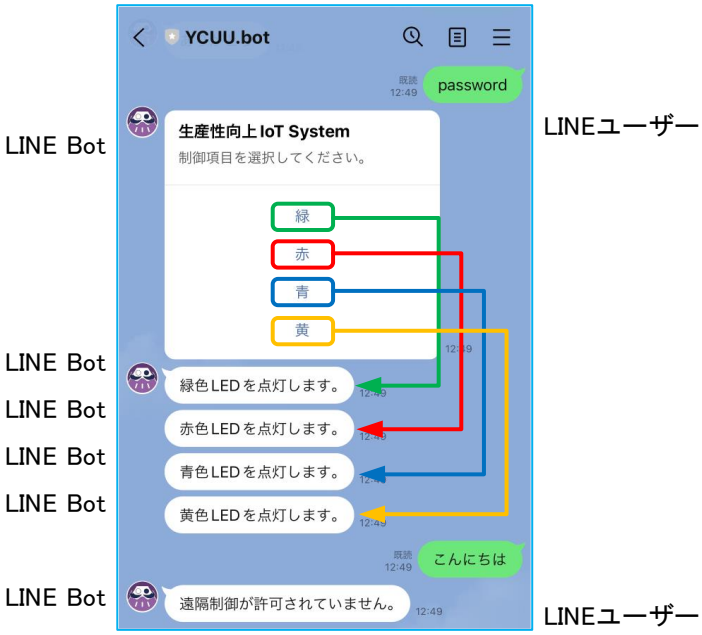
手順②で取得した Amazon S3 の URL を入力してから「保存」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

手順④～動作確認～

LINE Bot にメッセージを送信し、次の動作を確認してください。

- ✓ 指定メッセージ「password」を送信すると、ボタンテンプレートが表示される。
- ✓ ボタンテンプレートを押すと、各ボタンに応じたメッセージが自動応答される。
- ✓ 「password」以外のメッセージを送信すると、「遠隔制御が許可されていません。」と自動応答される。



～プログラム解説～

行	プログラム
1	from linebot.v3 import WebhookHandler
2	from linebot.v3.exceptions import InvalidSignatureError
3	from linebot.v3.messaging import (
4	ApiClient, Configuration, MessagingApi,
5	ReplyMessageRequest, TextMessage, TemplateMessage, ButtonsTemplate, PostbackAction
6	)
7	from linebot.v3.webhooks import (
8	MessageEvent, TextMessageContent, PostbackEvent
9	)
10	
11	#チャネルアクセストークン
12	CHANNEL_ACCESS_TOKEN = 'Q6/cA/gUE6yOYR7+xxxxxxxxxxbZ9cahLc0TK4/71gdRm0v42W2f+JrlP6TjVQ+OQIOf29CAU/Hetv/t7+oWfNWV/eBXirpzgh1fB99QQTayCTuAcvqEO8iH2L7jACAcSKywehFZq3j/6UdZhhXnmAdB04t89/1O/w1cDnyilFU='
13	#チャネルシークレット
14	CHANNEL_SECRET = 'e6bdaxxxxxxxxxx963eee04486668ba4'
15	

Copyright © 2024 YITJC All Rights Reserved.

```
configuration = Configuration(access_token=CHANNEL_ACCESS_TOKEN)
api_client = ApiClient(configuration=configuration)
line_bot_api = MessagingApi(api_client)
handler = WebhookHandler(CHANNEL_SECRET)

import logging
logger = logging.getLogger()
logger.setLevel(logging.ERROR)

def lambda_handler(event, context):
    #Line Uer ID取得するためeventを表示
    print(f"{event}")
    signature = event["headers"]["x-line-signature"]
    body = event["body"]
    ok_json = {"isBase64Encoded": False,
                "statusCode": 200,
                "headers": {},
                "body": ""}
    error_json = {"isBase64Encoded": False,
                  "statusCode": 403,
                  "headers": {},
                  "body": "Error"}

    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        return error_json
    return ok_json

@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    text = event.message.text
    if text == 'password':
        send_buttons_template(event.reply_token)
    else:
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=event.reply_token,
                messages=[TextMessage(text='遠隔制御が許可されていません。')]
```

Copyright © 2024 YITJC All Rights Reserved.

```
55                  )
56              )
57
58  def send_buttons_template(reply_token):
59      buttons_template = ButtonsTemplate(
60          title='生産性向上IoT System', text='制御項目を選択してください。',
61          actions=[
62              PostbackAction(label='緑', data='GREEN'),
63              PostbackAction(label='赤', data='RED'),
64              PostbackAction(label='青', data='BLUE'),
65              PostbackAction(label='黄', data='YELLOW'),
66          ]
67      )
68      template_message = TemplateMessage(alt_text='Buttons alt text', template=buttons_template)
69      line_bot_api.reply_message(
70          ReplyMessageRequest(
71              replyToken=reply_token,
72              messages=[template_message]
73          )
74      )
75
76  @handler.add(PostbackEvent)
77  def handle_postback(event):
78      if event.postback.data == 'GREEN':
79          line_bot_api.reply_message(
80              ReplyMessageRequest(
81                  replyToken=event.reply_token,
82                  messages=[TextMessage(text='緑色LEDを点灯します。')]
83              )
84          )
85      elif event.postback.data == 'RED':
86          line_bot_api.reply_message(
87              ReplyMessageRequest(
88                  replyToken=event.reply_token,
89                  messages=[TextMessage(text='赤色LEDを点灯します。')]
90              )
91          )
92      elif event.postback.data == 'BLUE':
93          line_bot_api.reply_message(
```

Copyright © 2024 YITJC All Rights Reserved.

```
94                  ReplyMessageRequest(
95                      replyToken=event.reply_token,
96                      messages=[TextMessage(text='青色LEDを点灯します。')]
97                  )
98              )
99          elif event.postback.data == 'YELLOW':
100             line_bot_api.reply_message(
101                 ReplyMessageRequest(
102                     replyToken=event.reply_token,
103                     messages=[TextMessage(text='黄色LEDを点灯します。')]
104                 )
105             )
```

灰色部は、これまでに説明した内容です。

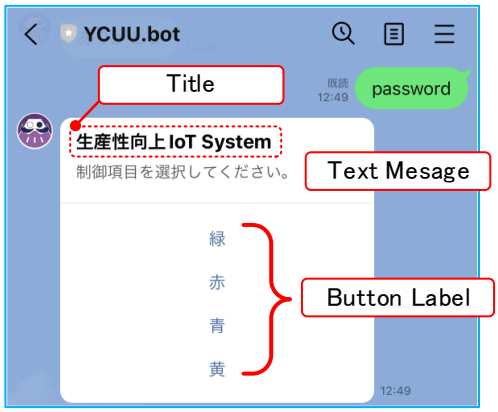
- 1 行目: linebot.v3 ライブラリの「WebhookHandler」をインポート
- 2 行目: linebot.v3.exceptions ライブラリの「InvalidSignatureError」クラスをインポート
- 3 行目～6 行目: linebot.v3.messaging ライブラリの「ApiClient」、「Configuration」、「MessagingApi」、「ReplyMessageRequest」、「TextMessage」、「TemplateMessage」、「ButtonsTemplate」、「PostbackAction」クラスをインポート
  このほか、画像（イメージ）ファイルを扱う「ImageMessage」クラス、ボタンテンプレートを扱う「ButtonsTemplate」クラス、ビデオメッセージを扱う「VideoMessage」クラスなどがあります。
- 7 行目～9 行目: linebot.v3.webhook ライブラリの「MessageEvent」、「TextMessageContent」クラスとインポート
- 12 行目: チャネルアクセストークン設定
- 14 行目: チャネルアシークレット設定
  ※「チャネルアクセストークン」および「チャネルシークレット」は、AWS Lambda で環境変数として保存することも可能ですが、ここではプログラム内に記述しています。
- 16 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 17 行目: ApiClient クラスを使用して API クライアントを初期化
- 18 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 19 行目: WebhookHandler クラスを使用してチャネルシークレットを設定
- 21 行目: プログラム実行時に発生したイベントを追跡するためのライブラリ「logging」をインポート
- 22 行目: 現在のロギング情報取得（引数はファイル名を設定）
- 23 行目: ロギングのレベルを「ERROR」に設定
- 25 行目～43 行目: Lambda 関数 Handler
- 27 行目: リクエストデータ print 表示
  「f」は、フォーマット済み文字リテラル（f-string）です。
- 28 行目: リクエストヘッダーの「x-line-signature」に含まれる値（署名検証のための値）を取得

Copyright © 2024 YITJC All Rights Reserved.

- 29 行目: リクエストボディを取得
- 30 行目～33 行目: ok_json 定義
- 34 行目～37 行目: error_json 定義
- 39 行目～40 行目: try ブロック
- 40 行目: WebhookHandler の「handler.handle」メソッドを使用
  署名「Signature」を検証し、問題なければ handle に定義されている関数を呼び出します。
- 41 行目～42 行目: except ブロック
- 42 行目: 署名検証が失敗した場合、error_json を API Gateway にレスポンス
- 43 行目: handle の処理が問題なく終了すれば、ok_json を API Gateway にレスポンス
- 45 行目～56 行目:「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。ここでは、「MessageEvent」、「TextMessage」を指定しています。
- 46 行目: LINE ユーザーからの送信メッセージを変数「text」に格納
- 48 行目～49 行目: 指定メッセージ「password」を受信したときの処理
- 49 行目: ボタンテンプレート表示関数 send_buttons_template()を呼び出し
- 50 行目～56 行目: 指定メッセージ以外のメッセージを受信したときの処理
- 51 行目～56 行目: LINE Bot 動作
  「遠隔制御が許可されていません。」と自動応答

- 58 行目～74 行目: ボタンテンプレート表示プログラム send_buttons_template()関数
- 59 行目～67 行目: ボタンテンプレート設定
  - ✓ Title: 生産性向上 IoT System
  - ✓ Text Message: 制御項目を選択してください。
  - ✓ Button Label / data
    data は、ボタンが押されたときに出力されます。

Button Label	data
緑	GREEN
赤	RED
青	BLUE
黄	YELLOW



- 68 行目: template_message 定義
- 69 行目～74 行目: LINE Bot 動作
  LINE Bot の応答（オウム返し）は LineBotApi の「line_bot_api.reply_message」メソッドを使用します。
  第一引数の「line_event.reply_token」はイベントの応答に用いるトークンです。
  第二引数には 68 行目で定義した template_message を渡しています。
- 76 行目:「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。
  ここでは、ボタンが押されたときに「data」が出力される「PostbackAction」を指定しています。

Copyright © 2024 YITJC All Rights Reserved.

- 78 行目～84 行目: data=「GREEN」を受け取った時の処理
  「緑色 LED を点灯します。」と自動応答
- 85 行目～91 行目: data=「RED」を受け取った時の処理
  「赤色 LED を点灯します。」と自動応答
- 92 行目～98 行目: data=「BLUE」を受け取った時の処理
  「青色 LED を点灯します。」と自動応答
- 99 行目～105 行目: data=「YELLOW」を受け取った時の処理
  「黄色 LED を点灯します。」と自動応答

Copyright © 2024 YITJC All Rights Reserved.

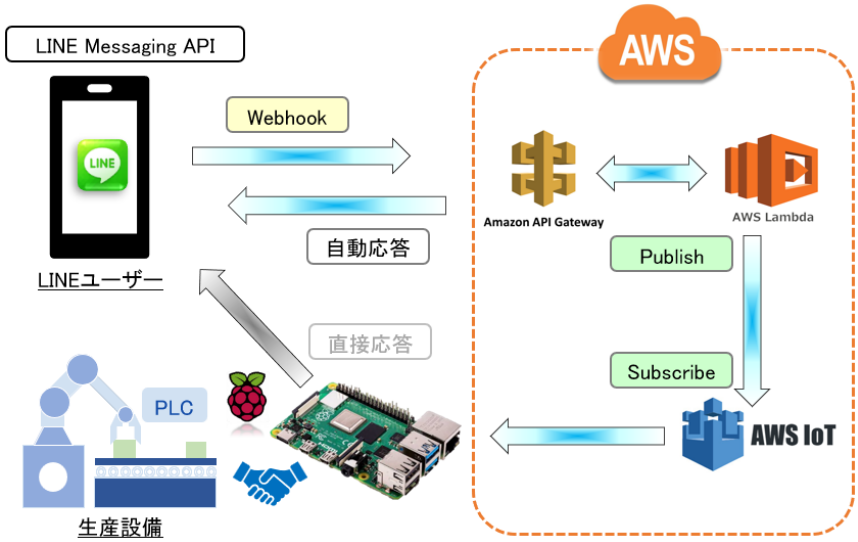
10.13 LINE Bot～Publish 機能実装～

「Subscribe」機能は AWS IoT Core サービスを利用して実装が完了しています。Raspberry Pi に実装した「Subscribe.py」は、Publish から送信された「ペイロード」に応じた処理を実行できる状態にあります。そのため、AWS Lambda 関数に「Publish」機能を実装することで、LINE ユーザーから送信されたメッセージに応じた処理を Raspberry Pi が実行できるようになります。

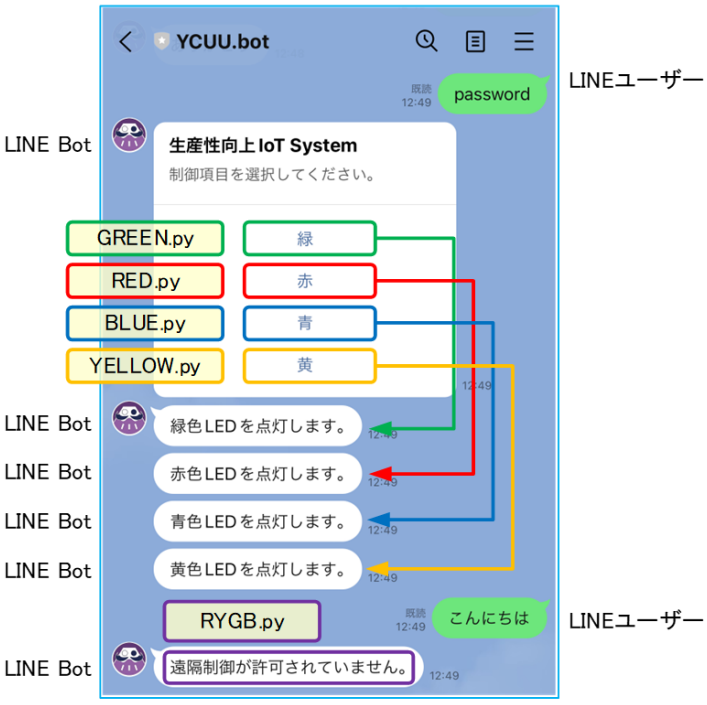
ここでは、LINE Bot（自動応答）に「Publish 機能」を追加する方法について説明します。

～補足～

「10.5 Subscribe 実装」では、AWS IoT Core の「MQTT テストクライアント」を使用して MQTT 通信テストを実施しました。このときは、MQTT テストクライアントの「Publish」機能を使用して Subscribe 側の動作確認を行いましたが、今回は「Publish」機能を Lambda 関数に実装します。



LINE ユーザーからのメッセージ（ボタンテンプレート含む）と Raspberry Pi（Subscriber）に実行させるプログラムの関係を LINE メッセージ画面に示します。



Copyright © 2024 YITJC All Rights Reserved.

手順①～Lambda プログラム作成～

「10.8 Bot プログラム作成」と同じ手順で AWS Lambda で実行する LINE Bot プログラムを作成・保存します。

[対象プログラム]

lambda_function.py

（共有サイトにある「lambda_function_publish.py」ファイルを編集して使用してください。）

～注意～

「lambda_function_publish.py」について、次の 3 か所を編集してください。

[プログラム編集個所]

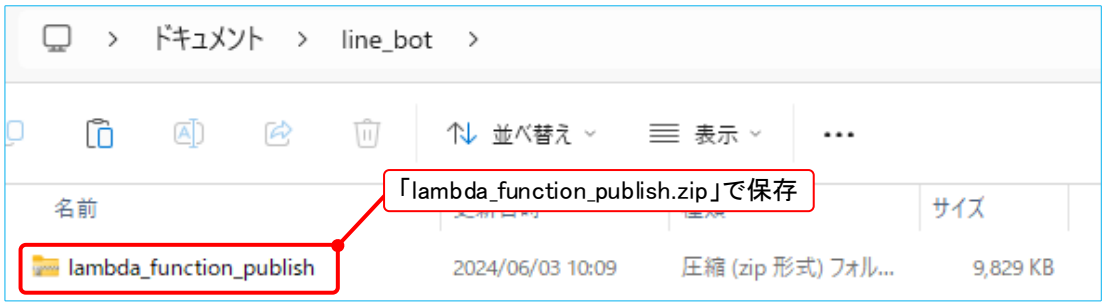
- 「チャネルシークレット」、「チャネルアクセストークン」
  「10.6 LINE Messaging API 設定」で取得した値に書き換えてください。

```
#12行目 チャネルアクセストークン
CHANNEL_ACCESS_TOKEN = 'xxxxxxx'
#14行目 チャネルシークレット
CHANNEL_SECRET = 'xxxxxxx'
```

- Subscriber のエンドポイント指定
  「10.3 Subscribe 実装」手順⑧で取得した「エンドポイント（end point）」を使用します。
  「https://」のあとに「エンドポイント」を追加してください。

```
#27 行目 Subscriber End Point 指定
client = boto3.client('iot-data', endpoint_url='https://end point')
```

zip ファイルは、「lambda_function_publish.zip」のファイル名で保存してください。



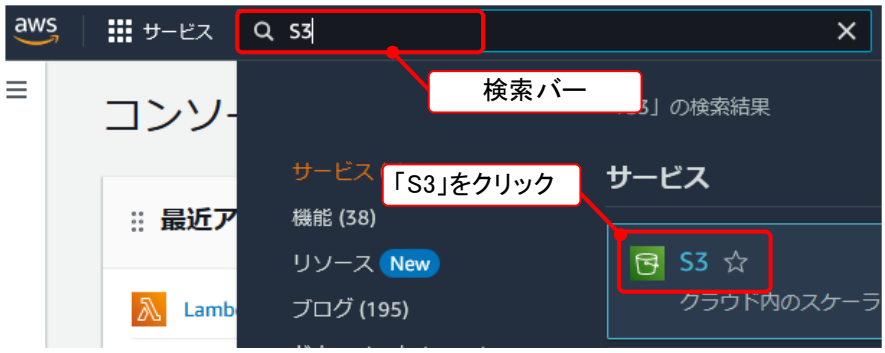
Copyright © 2024 YITJC All Rights Reserved.

手順②～Lambda 関数ファイルアップロード～

次の URL をクリックして AWS にログインしてください。

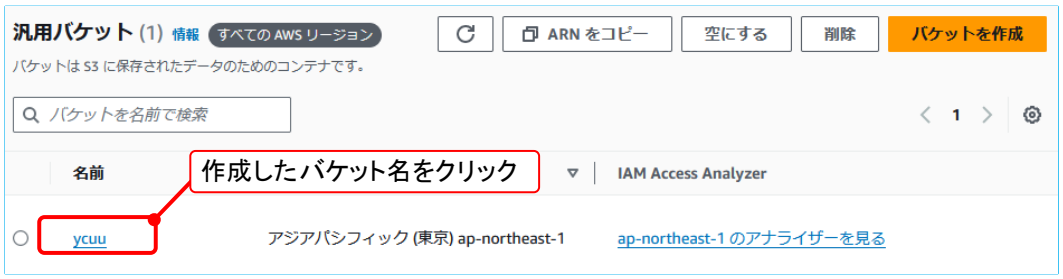
[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「S3」を検索してください。



「10.9 Bot プログラム保存」と同じ手順で AWS S3 に Lambda 関数ファイル一式をアップロードします。

「作成したバケット名」をクリックしてください。

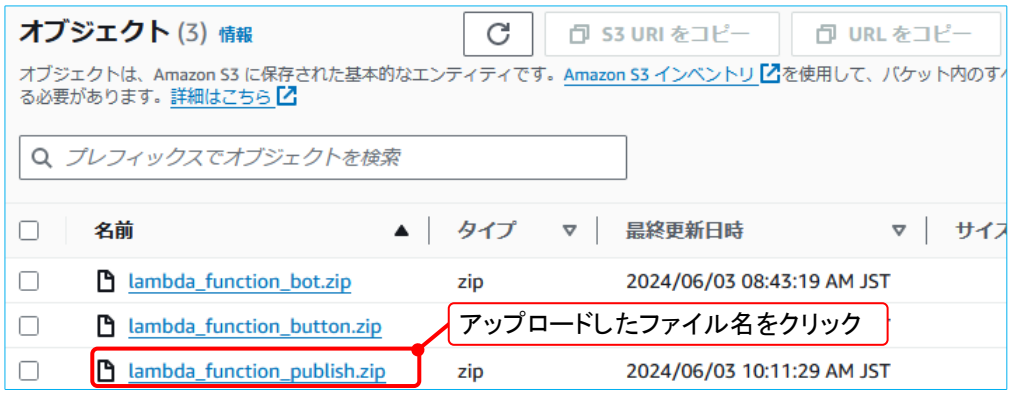


「アップロード」をクリックしてください。

「ファイルを追加」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「lambda_function_publish.zip」ファイルをアップロードしたあと、アップロードしたファイルの URL 情報を取得します。アップロードしたファイル名をクリックして URL 情報を取得してください。

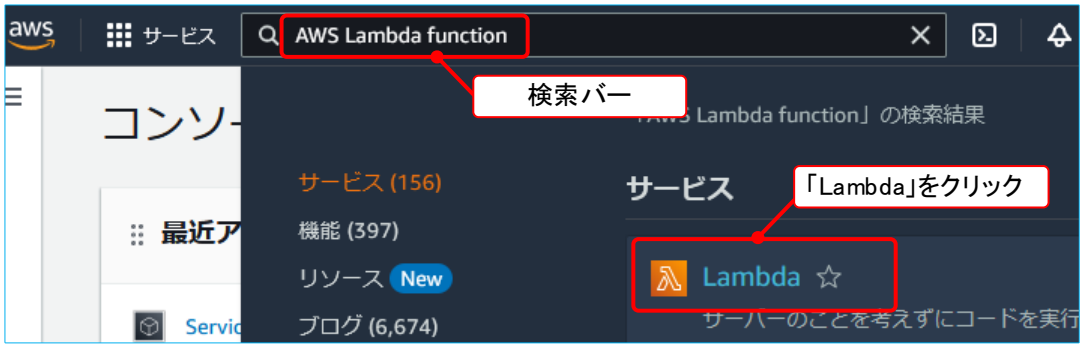


手順③～Lambda 関数デプロイ～

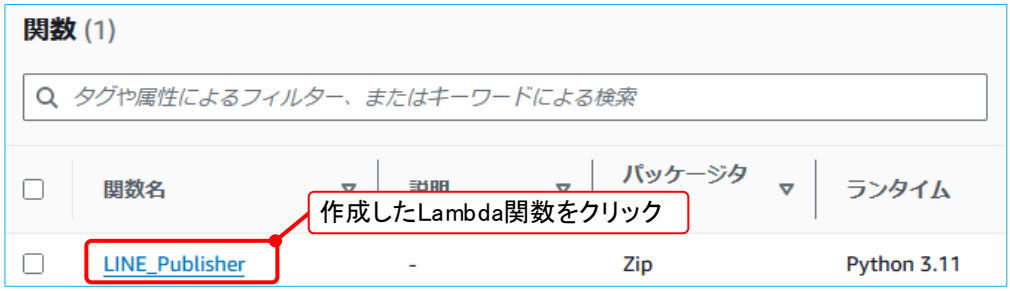
次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「AWS Lambda Function」を検索してください。



作成した「Lambda 関数」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

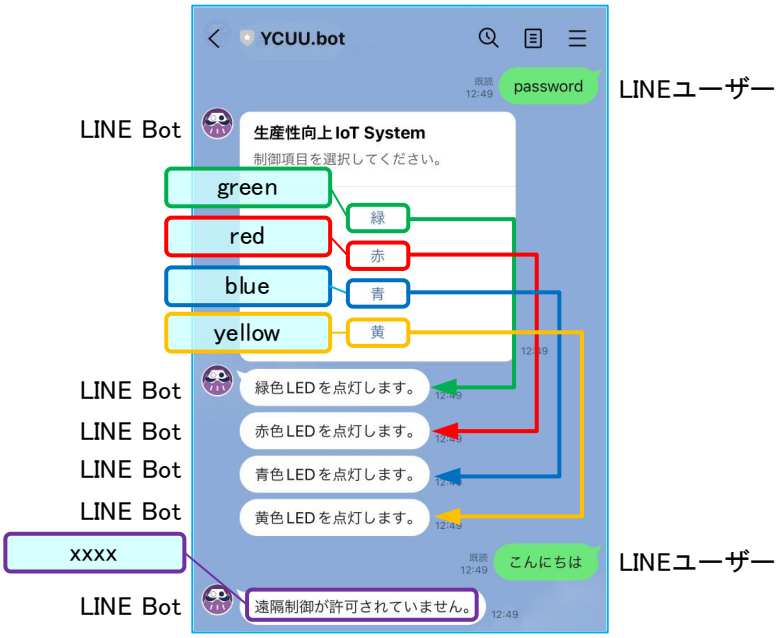
「コード」タブを選択し、「アップロード元」から「Amazon S3 の場所」をクリックしてください。

手順②で取得した Amazon S3 の URL を入力してから「保存」をクリックしてください。

手順④～Publish 動作確認～

AWS IoT Core「MQTT テストクライアント」ツールの「Subscribe 機能」を使用して、Lambda 関数に実装した「Publish」機能をテストします。

LINE ユーザーからのメッセージ（ボタンテンプレート含む）と Publisher が送信するメッセージ（payload）の関係を LINE メッセージ画面に示します。



Copyright © 2024 YITJC All Rights Reserved.

Lambda 関数に実装した Publisher から送信するメッセージに含まれる「topic」および「payload」と、Subscriber が（受信した「payload」に応じて）実行するプログラムの関係を次の表に示します。

Publisher 送信メッセージ		Subscriber 実行プログラム	
topic	payload	プログラム	動作説明
system	red	RED.py	赤色 2 秒間点灯
system	yellow	YELLOW.py	黄色 2 秒間点灯
system	green	GREEN.py	緑色 2 秒間点灯
system	blue	BLUE.py	青色 2 秒間点灯
system	xxxx	RYGB.py	赤・黄・緑・青 LED 順次点滅+ブザーON ※赤色 SW を押すまで動作継続

～補足～

例えば、LINE ユーザーが「緑」ボタンをタップした場合、Publisher が送信するメッセージ（payload）は、「green」です。Subscriber は、「green」を受信すると、「GEEN.py」プログラムを実行し、緑色 LED が 2 秒間点灯します。

次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「IoT Core」を検索してください。

画面左のメニューから「テスト」-「MQTT テストクライアント」を選択してください。

今回は、Lambda 関数に実装した「Publish」の動作確認を行うため、「MQTT テストクライアント」は「Subscribe 機能（トピックをサブスクライブする）」を使用します。

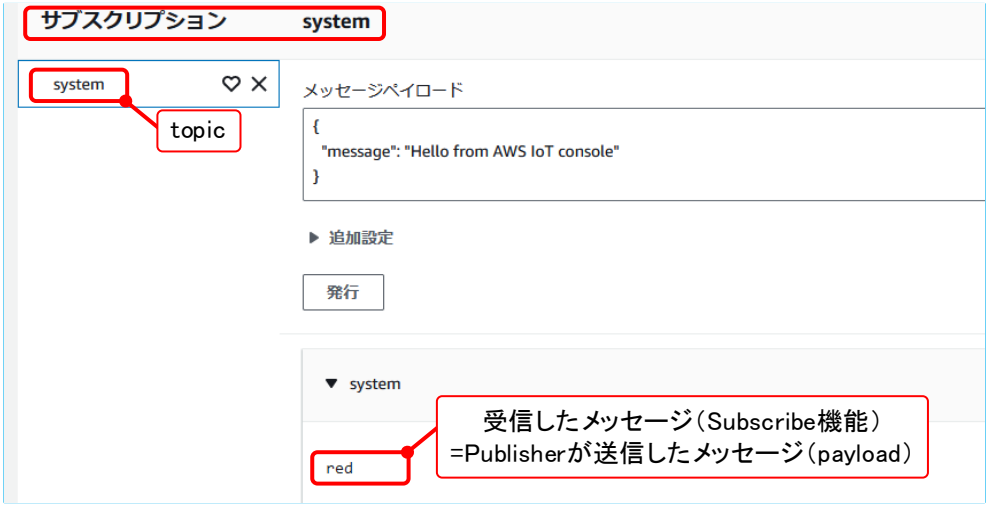
Copyright © 2024 YITJC All Rights Reserved.

次の画面と同じ設定にしてから「サブスクライブ」をクリックしてください。

スマートフォンの LINE アプリを使用して、LINE Bot に「password」を送信したあと、ボタンテンプレートをタップしてください。次の画面は、ボタンテンプレートの「赤」をタップした時の実行結果です。

「赤」をタップすると、Publisher は「red」メッセージを Subscribe 側に送信します。

Subscribe 側が受信したメッセージを IoT Core「MQTT テストクライアント」ツールのサブスクリプション画面で確認できます。



「MQTT テストクライアント」ツールのテスト結果より、Lambda 関数に実装された「Publish」機能が正常に動作していることがわかりました。続いて、Raspberry Pi に実装した「Subscribe」機能を使用して MQTT 通信テストを行います。

Copyright © 2024 YITJC All Rights Reserved.

手順⑤～Subscribe プログラム実行～

Publish から送信されるメッセージ（MQTT メッセージ）の受信テストを実施するため、「Subscribe.py」プログラムを実行して「受信待ち」の状態にします。

(ycuu) cuu@raspberrypi:~$ cd dev/aws/sub

(ycuu) cuu@raspberrypi:~/dev/aws/sub $ ./Subscribe.py

```
(ycuu) cuu@raspberrypi:~ $ cd dev/aws/sub
(ycuu) cuu@raspberrypi:~/dev/aws/sub $ ./Subscribe.py
```

手順⑥～Raspberry Pi 動作確認～

Raspberry Pi で Subscribe プログラムを動作させた状態で、LINE ユーザーからのメッセージを送信し、送信メッセージにより指定されたプログラムが Raspberry Pi によって実行されることを確認します。

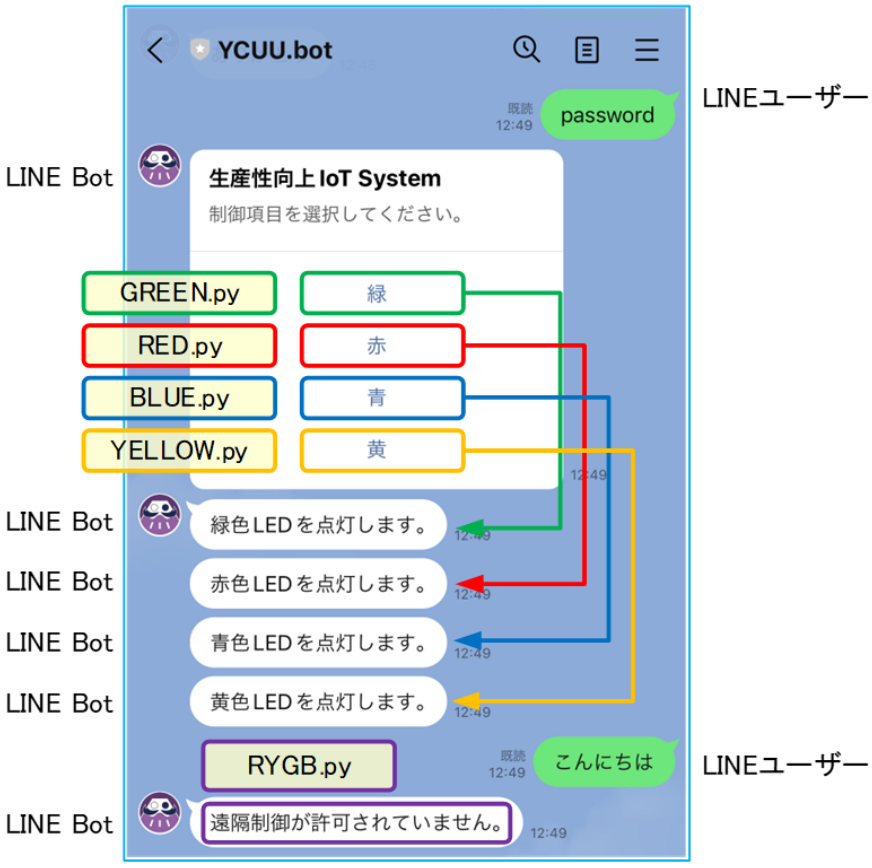
Raspberry Pi を起動して Subscribe プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev/aws/sub $ ./Subscribe.py

プログラムを実行すると MQTT メッセージ「受信待ち」の状態になります。

```
pi@raspberrypi:~/dev/aws/sub $ ./Subscribe.py
```

スマートフォンの LINE アプリを使用してメッセージを送信してください。ボタンテンプレートを表示させる場合は、「password」を送信してください。

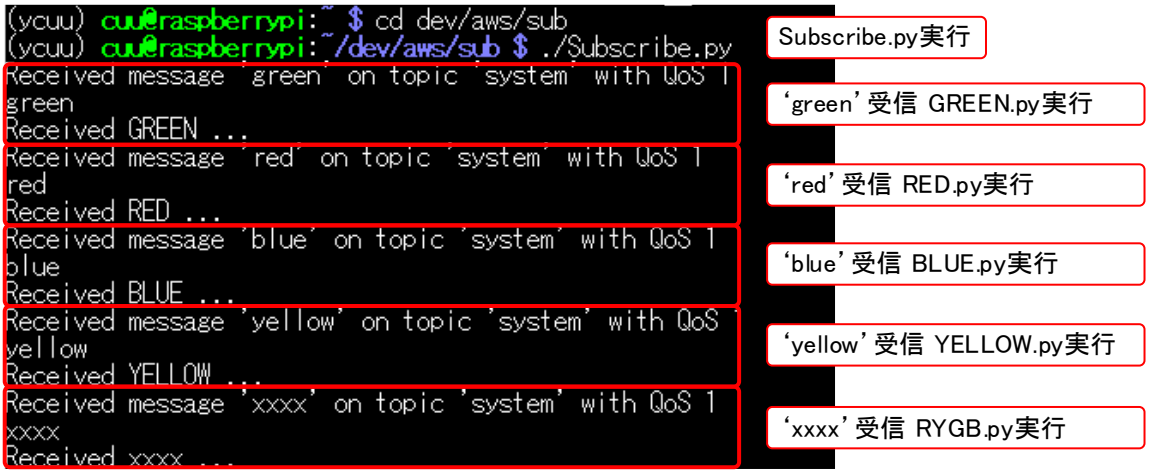


Copyright © 2024 YITJC All Rights Reserved.

LINE 画面の「ボタンテンプレート」と Raspberry Pi が実行するプログラムの関係を次の表に記載します。

LINE 送信メッセージ	Raspberry Pi 実行プログラム	
赤（ボタン）	RED.py	赤色 LED2 秒間点灯
黄（ボタン）	YELLOW.py	黄色 LED2 秒間点灯
緑（ボタン）	GREEN.py	緑色 LED2 秒間点灯
青（ボタン）	BLUE.py	青色 LED2 秒間点灯
「password」以外のメッセージ	RYGB.py	赤・黄・緑・青 LED 順次点滅+ブザーON ※赤色 SW を押すまで動作継続

LINE ユーザーからメッセージが送信されると、画面上に MQTT 通信で受信したメッセージ（topic および payload）が表示されます。Raspberry Pi に実装した「Subscriber」は受信したメッセージ「payload」によって指定されたプログラムを実行します。



動作確認が終了したら、「Ctrl」+「c」を押してプログラムを終了してください。

ここからは、Raspberry Pi に実装した「Subscriber」を常時稼働するための手順について説明します。

手順⑦～Unit ファイル作成～

ここで作成するファイルは、Lambda 関数に実装した Publisher からのメッセージを受信するプログラム（Subscribe.py）を自動起動させるための Unit ファイル（.service ファイル）です。

まず、/etc/systemd/system ディレクトリへ移動します。

(ycuu) cuu@raspberrypi: ~/dev/aws/sub $ cd /etc/systemd/system

Copyright © 2024 YITJC All Rights Reserved.

続いて、Unit ファイルを作成します。次のコマンドを実行すると、何も書かれていない画面が表示されるので、Unit ファイルの中身を書き込んでください。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano Subscribe.service

～参考～

Unit ファイルの「上書き」、「終了」は、次の手順を参考にしてください。

- 編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認
- 「Enter」キーを押下して上書き保存を実行
- 「Ctrl + X」キーを押して元の画面に戻る

～Unit ファイル解説～

Unit ファイルの中身について解説します。

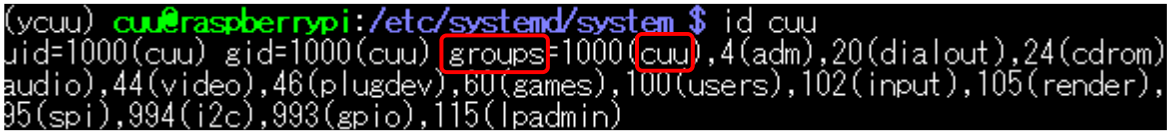
行	Unit ファイルの中身
1	[Unit]
2	Description=Subscriber waiting for MQTT Message
3	After=network.target
4	
5	[Service]
6	ExecStart=/home/cuu/dev/aws/sub/Subscribe.py
7	Restart=always
8	RestartSec=5sec
9	Type=simple
10	WorkingDirectory=/home/cuu/dev/aws/sub
11	User=cuu
12	Group=cuu
13	#Subsriber Information for AWS IoT Core
14	Environment="ENDPOINT=xxxxxxxxxx-ats.iot.ap-northeast-1.amazonaws.com"

Copyright © 2024 YITJC All Rights Reserved.

```
15  Environment="PORT=8883"
16  Environment="ROOT_CA=/home/cuu/dev/aws/sub/cert/rootCA.pem"
17  Environment="PRIVATE_KEY=/home/cuu/dev/aws/sub/cert/xxxxxxxxxx-private.pem.key"
18  Environment="CERTIFICATE=/home/cuu/dev/aws/sub/cert/xxxxxxxxxx-certificate.pem.crt"
19
20  [Install]
21  WantedBy=multi-user.target
```

灰色部は、これまでに説明した内容です。

- 1 行目～3 行目: Unit セクション
  プロセスの説明や依存関係、起動順序などを設定します。
- 2 行目: サービスの説明文(動作に影響しませんが、分かりやすく記述するようにしてください。)
- 3 行目: 起動順序の指定
  After=network.target
- 5 行目～18 行目: Service セクション
  起動するプログラム、起動条件など Unit ファイルの中で最も設定項目が多いセクションです。
  サービスの内容(どんな処理を実行させるか)を記述するセクションです。
- 6 行目: 自動起動するプログラムを指定 (Subscribe.py)
- 7 行目: プロセスが停止した場合、プロセスの再起動条件を設定
  Always: 常に再起動
- 8 行目: 再起動するまでの待ち時間(秒)を設定
  ～補足～
  「Subscribe.py」は Publisher から送信されるメッセージを常に受信します。
  そのため、受信したメッセージの処理が終了したあと、次のメッセージを受信待機させるため、Always(常に再起動)設定にしています。
- 9 行目: サービスプロセスの起動完了の判定方法
  デフォルト設定値「Simple」に設定
  Simple は、コマンドを実行したタイミングで起動完了と判断します。
- 10 行目: WorkingDirectory を指定
  ～補足～
  「Subscribe.py」プログラムは、他のプログラム「BLUE.py」や「GREEN.py」を呼び出して実行します。
  他のファイルを参照してプログラムを実行する場合は WorkingDirectory を必ず設定してください。
- 11 行目～12 行目: 実行時の User, Group を指定
  ユーザー(User)及びユーザーが所属しているグループ(Group)は、次のコマンドで確認することできます。
  id のあとに本講座で使用しているユーザー名「cuu」を入力してください。
  (ycuu) cuu@raspberrypi:/etc/systemd/system $ id cuu



Copyright © 2024 YITJC All Rights Reserved.

- 14 行目～18 行目: AWS IoT Core を利用するために必要な情報を環境変数として設定

  ～補足～

  AWS IoT Core の情報は、ホームディレクトリ（/home/cuu）にある「.bashrc」で環境変数に設定しています。しかし、systemd のサービスは「.bashrc」ファイルを読み込むことができないため、Unit ファイルで環境変数を定義する必要があります。
- 20 行目～21 行目: Install セクション

  このセクションが無ければ、後で行う自動起動有効化（enable）を実行することができません。
- 20 行目: サービス Unit の依存関係を設定

  multi-user.target は、複数ユーザー環境のベースとなるターゲットで、多くのサービスがこのターゲットに紐付いて起動されます。（shutdown サービス Unit もこのターゲットに紐付けます）

  WantedBy=multi-user.target と記述することで、multi-user.target の起動時に一緒に起動するように要求しています。

手順⑧～サービス再読み込み～

手順⑦で Unit ファイルを作成した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順⑨～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable Subscribe.service

手順⑩～サービス開始～

次のコマンドを実行してサービスを開始します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start Subscribe.service

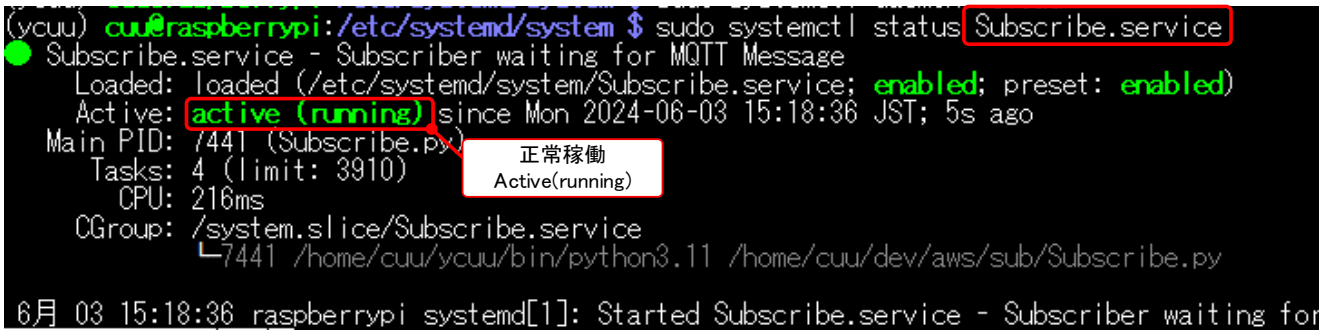
手順⑪～サービス動作状況確認～

次のコマンドを実行してサービスの動作状況を確認します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status Subscribe.service

正常に稼働（自動起動）していれば、次の画面のようにステータスが active（runnning）と表示されます。

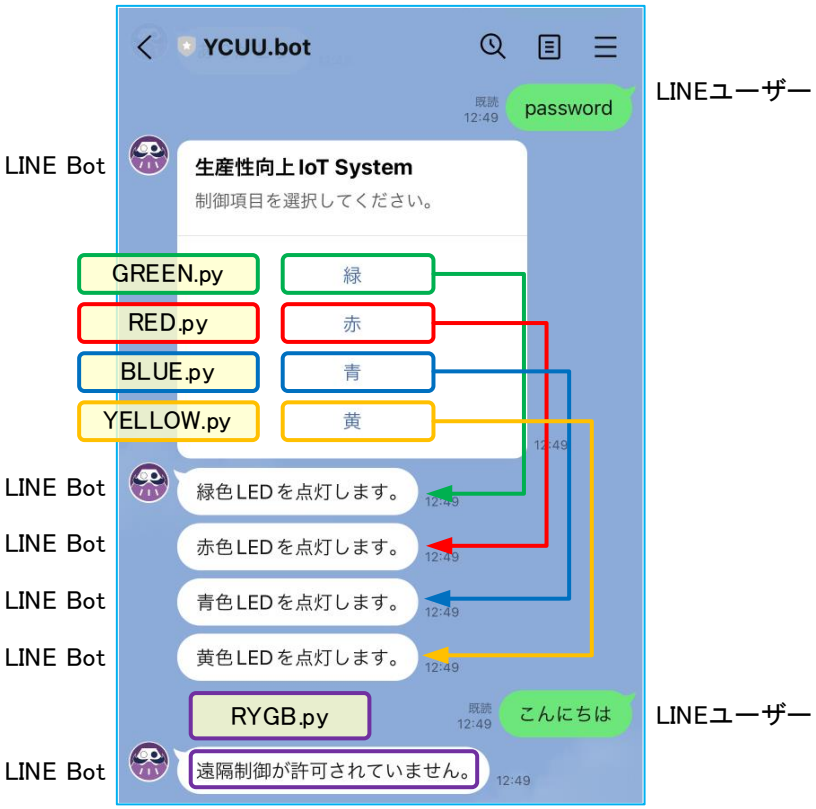
この画面表示を終了する場合は、「q」を押してください。



Copyright © 2024 YITJC All Rights Reserved.

手順⑫～動作確認～

現在の状態は、「Subscribe.service」が自動起動しているので、常に AWS Lambda（Publisher）からのメッセージを受信待機している状態です。スマートフォンを使用して LINE アプリからメッセージを送信し、メッセージ対応したプログラムが Raspberry Pi によって実行されることを確認してください。



手順⑬～システムログ確認～

「Subscribe.service」の動作確認を行うため、次のコマンドを実行してシステムのログを確認してください。

(ycuu) cuu @raspberrypi:/etc/systemd/system $ journalctl -u Subscribe.service -n 10

- **オプションの設定（例）**

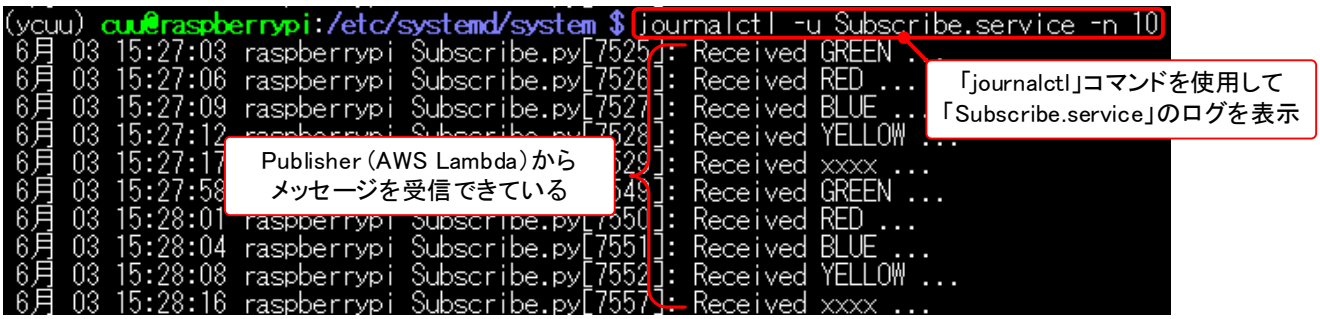
「-u Unit 名」：指定した Unit のログを表示

「-n 数」：末尾（直近）から指定した数文のログを表示

「-f」：リアルタイム監視（デバッグに便利）

「-xe」：詳細情報を表示

Publisher（AWS Lambda 関数）からのメッセージを受信できていることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	from linebot.v3 import WebhookHandler
2	from linebot.v3.exceptions import InvalidSignatureError
3	from linebot.v3.messaging import (
4	ApiClient, Configuration, MessagingApi,
5	ReplyMessageRequest, TextMessage, TemplateMessage, ButtonsTemplate, PostbackAction
6	)
7	from linebot.v3.webhooks import (
8	MessageEvent, TextMessageContent, PostbackEvent
9	)
10	
11	#チャネルアクセストークン
12	CHANNEL_ACCESS_TOKEN = 'Q6/cA/gUE6yOYR7+xxxxxxxxxxbZ9cahLc0TK4/71gdRm0v42W2f+JrlP6TjVQ+OQIOf29CAU/Hetv/t7+oWfNWV/eBXirpzgh1fB99QQTayCTuAcvqEO8iH2L7jACAcSKywehFZq3j/6UdZhhXnmAdB04t89/1O/w1cDnyilFU='
13	#チャネルシークレット
14	CHANNEL_SECRET = 'e6bdaxxxxxxxxxx963eee04486668ba4'
15	
16	configuration = Configuration(access_token=CHANNEL_ACCESS_TOKEN)
17	api_client = ApiClient(configuration=configuration)
18	line_bot_api = MessagingApi(api_client)
19	handler = WebhookHandler(CHANNEL_SECRET)
20	
21	import logging
22	logger = logging.getLogger()
23	logger.setLevel(logging.ERROR)
24	
25	import boto3
26	#Subscriber End Point指定
27	client = boto3.client('iot-data', endpoint_url='https://xxxxxxxxxx-ats.iot.ap-northeast-1.amazonaws.com')
28	
29	def lambda_handler(event, context):
30	#Line Uer ID取得するためeventを表示
31	print(f"{event}")
32	signature = event["headers"]["x-line-signature"]
33	body = event["body"]
34	ok_json = {"isBase64Encoded": False,
35	"statusCode": 200,

Copyright © 2024 YITJC All Rights Reserved.

```
                    "headers": {},
                    "body": ""}
    error_json = {"isBase64Encoded": False,
                    "statusCode": 403,
                    "headers": {},
                    "body": "Error"}

    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        return error_json
    return ok_json

@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    text = event.message.text
    if text == 'password':
        send_buttons_template(event.reply_token)
    else:
        #boto3 Publish
        response = client.publish(topic='system', qos=1, payload='xxxx')
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=event.reply_token,
                messages=[TextMessage(text='遠隔制御が許可されていません。')]
            )
        )

def send_buttons_template(reply_token):
    buttons_template = ButtonsTemplate(
        title='生産性向上IoT System', text='制御項目を選択してください。',
        actions=[
            PostbackAction(label='緑', data='GREEN'),
            PostbackAction(label='赤', data='RED'),
            PostbackAction(label='青', data='BLUE'),
            PostbackAction(label='黄', data='YELLOW'),
        ]
    )
    template_message = TemplateMessage(alt_text='Buttons alt text', template=buttons_template)
```

Copyright © 2024 YITJC All Rights Reserved.

```
75      line_bot_api.reply_message(
76          ReplyMessageRequest(
77              replyToken=reply_token,
78              messages=[template_message]
79          )
80      )
81
82  @handler.add(PostbackEvent)
83  def handle_postback(event):
84      if event.postback.data == 'GREEN':
85          #boto3 Publish
86          response = client.publish(topic='system', qos=1, payload='green')
87          line_bot_api.reply_message(
88              ReplyMessageRequest(
89                  replyToken=event.reply_token,
90                  messages=[TextMessage(text='緑色LEDを点灯します。')]
91              )
92          )
93      elif event.postback.data == 'RED':
94          #boto3 Publish
95          response = client.publish(topic='system', qos=1, payload='red')
96          line_bot_api.reply_message(
97              ReplyMessageRequest(
98                  replyToken=event.reply_token,
99                  messages=[TextMessage(text='赤色LEDを点灯します。')]
100             )
101         )
102     elif event.postback.data == 'BLUE':
103         #boto3 Publish
104         response = client.publish(topic='system', qos=1, payload='blue')
105         line_bot_api.reply_message(
106             ReplyMessageRequest(
107                 replyToken=event.reply_token,
108                 messages=[TextMessage(text='青色LEDを点灯します。')]
109             )
110         )
111     elif event.postback.data == 'YELLOW':
112         #boto3 Publish
113         response = client.publish(topic='system', qos=1, payload='yellow')
```

Copyright © 2024 YITJC All Rights Reserved.

```
114             line_bot_api.reply_message(
115                 ReplyMessageRequest(
116                     replyToken=event.reply_token,
117                     messages=[TextMessage(text='黄色LEDを点灯します。')]
118                 )
119             )
```

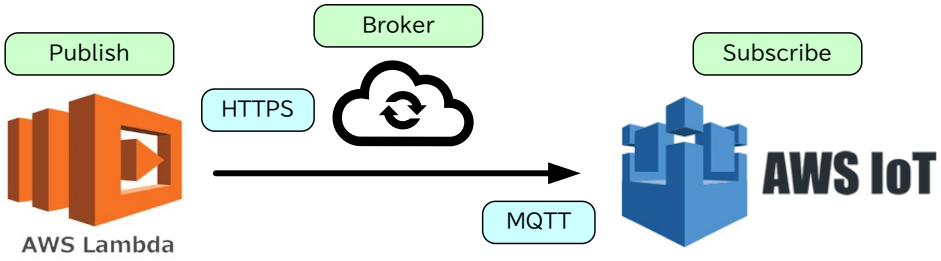
灰色部は、これまでに説明した内容です。

- 1 行目: linebot.v3 ライブラリの「WebhookHandler」をインポート
- 2 行目: linebot.v3.exceptions ライブラリの「InvalidSignatureError」クラスをインポート
- 3 行目～6 行目: linebot.v3.messaging ライブラリの「ApiClient」、「Configuration」、「MessagingApi」、「ReplyMessageRequest」、「TextMessage」、「TextMessage」、「TemplateMessage」、「ButtonsTemplate」、「PostbackAction」クラスをインポート
- クラスをインポート
  このほか、画像（イメージ）ファイルを扱う「ImageMessage」クラス、ボタンテンプレートを扱う「ButtonsTemplate」クラス、ビデオメッセージを扱う「VideoMessage」クラスなどがあります。
- 7 行目～9 行目: linebot.v3.webhook ライブラリの「MessageEvent」、「TextMessageContent」クラスとインポート
- 12 行目: チャネルアクセストークン設定
- 14 行目: チャネルアシークレット設定
  ※「チャネルアクセストークン」および「チャネルシークレット」は、AWS Lambda で環境変数として保存することも可能ですが、ここではプログラム内に記述しています。
- 16 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 17 行目: ApiClient クラスを使用して API クライアントを初期化
- 18 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 19 行目: WebhookHandler クラスを使用してチャネルシークレットを設定
- 21 行目: プログラム実行時に発生したイベントを追跡するためのライブラリ「logging」をインポート
- 22 行目: 現在のロギング情報取得（引数はファイル名を設定）
- 23 行目: ロギングのレベルを「ERROR」に設定
- 25 行目: boto3 ライブラリをインポート
  boto3 は「AWS SDK for Python」の別称であり、Python を介して AWS を操作するためのライブラリです。
  余談ですが、アマゾンカワイルカ（Boto Pink Dolphin）にちなんで名付けられたそうです。
- 27 行目: boto3 ライブラリの client()メソッドを使用してインスタンスを生成
  「https://」のあとに「エンドポイント」を追加

Copyright © 2024 YITJC All Rights Reserved.

～参考～

「対話型システム」の Publisher である AWS Lambda は、「HTTPS」通信プロトコルを採用していますが、具体的には AWS IoT サービスの「Publish」機能を使用しています。



AWS IoT サービスの API は、さまざまな機能が用意されていますが、機能ごとに属している「カテゴリ」が異なります。つまり、「Publish」機能を使用するためには、「Publish」機能が属している「カテゴリ」を指定する必要があります。

AWS IoT サービスのカテゴリは、次の 4 つに分類されています。

- ✓ AWS IoT
- ✓ AWS IoT Data Plane
- ✓ AWS IoT Jobs Data Plane
- ✓ AWS IoT Secure Tunneling

「Publish」機能は、AWS IoT Data Plane カテゴリに属しています。

boto3.client()の第一引数に「iot-data」と指定することで、AWS IoT Data Plane カテゴリを指定しています。

boto3.client()の第二引数に「endpoint_url」を指定することで、AWS IoT Core へのアクセスが可能になります。

```
client = boto3.client('iot-data', endpoint_url='https://xxxx')
```

以下、参考 URL です。

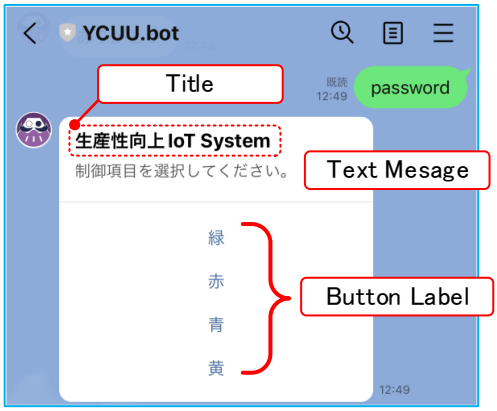
- AWS IoT サービス機能
  Publish メソッド（機能）に関する記述があります。
  https://docs.aws.amazon.com/iot/latest/apireference/API_iotdata_Publish.html
- AWS IoT Data Plane
  AWS IoT Data Plane についての記述があります。
  https://docs.aws.amazon.com/iot/latest/apireference/Welcome.html#Welcome_AWS_IoT_Data_Plane
- Boto3 Documents
  'iot-data'と指定することで、「Publish」メソッド（機能）が利用可能になることが記載されています。
  https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/iot-data.html

- 29 行目～47 行目: Lambda 関数 Handler
- 31 行目: リクエストデータ print 表示
  「f」は、フォーマット済み文字リテラル（f-string）です。
- 32 行目: リクエストヘッダーの「x-line-signature」に含まれる値（署名検証のための値）を取得

Copyright © 2024 YITJC All Rights Reserved.

- 33 行目: リクエストボディを取得
- 34 行目～37 行目: ok_json 定義
- 38 行目～41 行目: error_json 定義
- 43 行目～44 行目: try ブロック
- 44 行目: WebhookHandler の「handler.handle」メソッドを使用
  署名「Signature」を検証し、問題なければ handle に定義されている関数を呼び出します。
- 45 行目～46 行目: except ブロック
- 46 行目: 署名検証が失敗した場合、error_json を API Gateway にレスポンス
- 47 行目: handle の処理が問題なく終了すれば、ok_json を API Gateway にレスポンス
- 50 行目～62 行目: 「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。ここでは、「MessageEvent」、「TextMessage」を指定しています。
- 51 行目: LINE ユーザーからの送信メッセージを変数「text」に格納
- 52 行目～53 行目: 指定メッセージ「password」を受信したときの処理
- 53 行目: ボタンテンプレート表示関数 send_buttons_template()を呼び出し
- 54 行目～62 行目: 指定メッセージ以外のメッセージを受信したときの処理
- 56 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=xxxx」を指定して Publish
- 57 行目～62 行目: LINE Bot 動作
  「遠隔制御が許可されていません。」と自動応答
- 64 行目～80 行目: ボタンテンプレート表示プログラム send_buttons_template()関数
- 65 行目～73 行目: ボタンテンプレート設定
  - ✓ Title: 生産性向上 IoT System
  - ✓ Text Message: 制御項目を選択してください。
  - ✓ Button Label / data
    data は、ボタンが押されたときに出力されます。

Button Label	data
緑	GREEN
赤	RED
青	BLUE
黄	YELLOW



- 74 行目: template_message 定義
- 75 行目～80 行目: LINE Bot 動作
  LINE Bot の応答（オウム返し）は LineBotApi の「line_bot_api.reply_message」メソッドを使用します。
  第一引数の「line_event.reply_token」はイベントの応答に用いるトークンです。
  第二引数には 68 行目で定義した template_message を渡しています。
- 82 行目: 「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。
  ここでは、ボタンが押されたときに「data」が出力される「PostbackAction」を指定しています。

Copyright © 2024 YITJC All Rights Reserved.

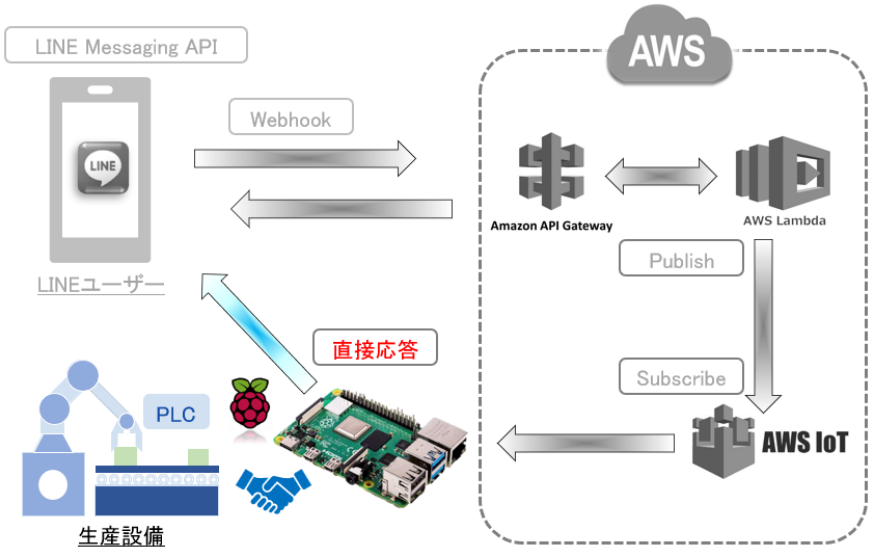
- 84 行目～92 行目: data=「GREEN」を受け取った時の処理
- 86 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=green」を指定して Publish
  「緑色 LED を点灯します。」と自動応答
- 93 行目～101 行目: data=「RED」を受け取った時の処理
- 95 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=red」を指定して Publish
  「赤色 LED を点灯します。」と自動応答
- 102 行目～110 行目: data=「BLUE」を受け取った時の処理
- 104 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=blue」を指定して Publish
  「青色 LED を点灯します。」と自動応答
- 111 行目～119 行目: data=「YELLOW」を受け取った時の処理
- 113 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=yellow」を指定して Publish
  「黄色 LED を点灯します。」と自動応答

Copyright © 2024 YITJC All Rights Reserved.

10.14 直接応答～LINE ユーザーID～

対話型システムの構築に向けて、残る機能は Raspberry Pi による「直接応答」です。Raspberry Pi から LINE ユーザー直接応答できるこの「直接応答」機能は、LINE ユーザーと Raspberry Pi との間で「対話」を実現するうえで重要な機能です。

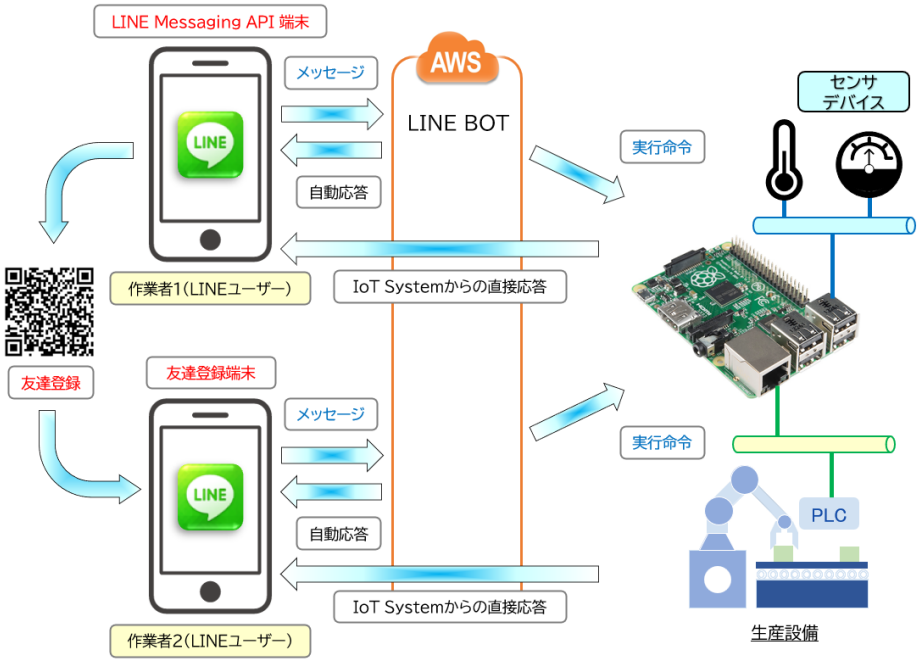
Raspberry Pi が LINE ユーザーに対して直接応答するためには、LINE ユーザーの「ユーザーID」を取得する必要があります。ここでは、ユーザーID の取得方法とユーザーID を使用した直接応答の方法について学習します。



参考～開発した LINE Bot の共有～

開発した LINE Bot システムは、開発者の端末（LINE Messaging API アカウントを取得した端末）以外でも利用できます。LINE Bot を「友達登録」した端末であれば、LINE Bot 機能を使用できます。

Raspberry Pi が行う「直接応答」については、端末ごとに異なるユーザーID をあらかじめ確認しておくことで、各端末宛に Raspberry Pi から「直接応答」メッセージを送信できます。



Copyright © 2024 YITJC All Rights Reserved.

ここでは、LINE ユーザーID の取得方法を 2 つ紹介します。

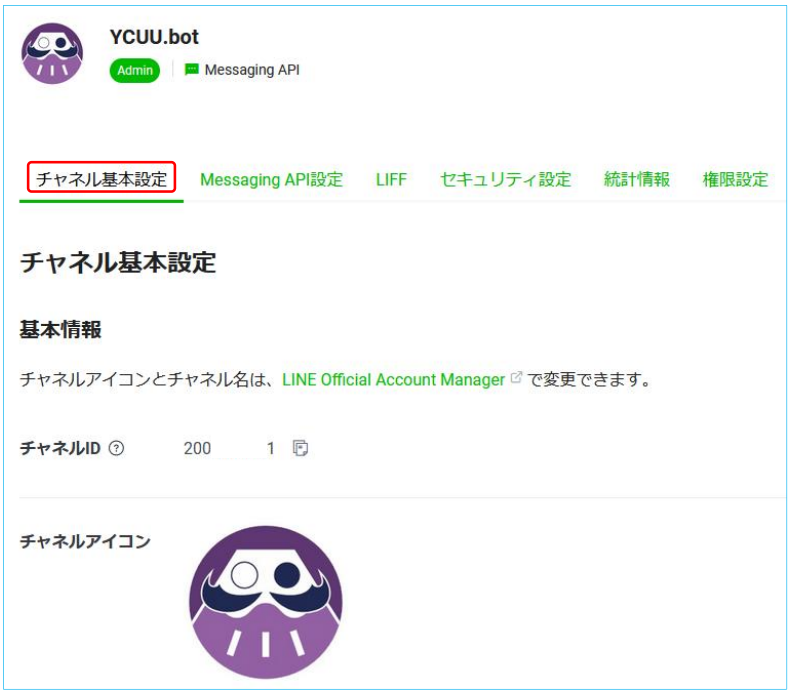
取得方法①～LINE Developers（復習）～

LINE Bot 開発者の端末（LINE Messaging API アカウントを取得した端末）については、LINE Developers にログインしてユーザーID できます。

次のリンクをクリックして LINE Developers にログインしてください。

[URL] https://developers.line.biz/ja/

「チャネル基本設定」にユーザーID が記載されています。



「ユーザーID」の情報は、プログラムで使用するのでメモ帳などに記録してください。

取得方法②～AWS CloudWatch～

LINE ユーザーID の取得方法 2 つ目は、AWS Cloud Watch サービスを利用します。Cloud Watch サービスに記録されている通信ログを確認することでユーザーID の情報を取得できます。この方法を使用すると、「友達登録」した端末の通信ログからユーザーID を取得できるため、友達登録した端末宛に直接応答メッセージを送信できるようになります。

Copyright © 2024 YITJC All Rights Reserved.

手順①～CloudWatch～

次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「Cloud Watch」を検索してください。

「CloudWatch」をクリックすると次の画面が表示されるので、「ログ」-「ロググループ」をクリックすると、右側の画面に「作成した Lambda 関数のロググループ」が表示されます。

手順②～通信ログ確認～

「作成した Lambda 関数のロググループ」をクリックしてください。

ログストリームに作成した Lambda 関数名のログをクリックすると、ログが時系列に記録されていることがわかります。最新のログは一番上に記録されています。

Copyright © 2024 YITJC All Rights Reserved.

手順③～LINE ユーザーID 取得～

ログをクリックすると「タイムスタンプ」と「メッセージ」が時系列に表示されています。

ここでは古い順に上から記録されているため、最新のログを確認する場合は一番下に記録されているログを確認してください。

「START」と「END」の間にあるログをクリックして開いてください。

このログは、Lambda 関数（Publish プログラム）で定義した print 表示の結果が記録されています。

```
print(f"{event}")
```

ログの記録を確認すると、「userId」の記録に送信元の LINE ユーザーID が記録されています。

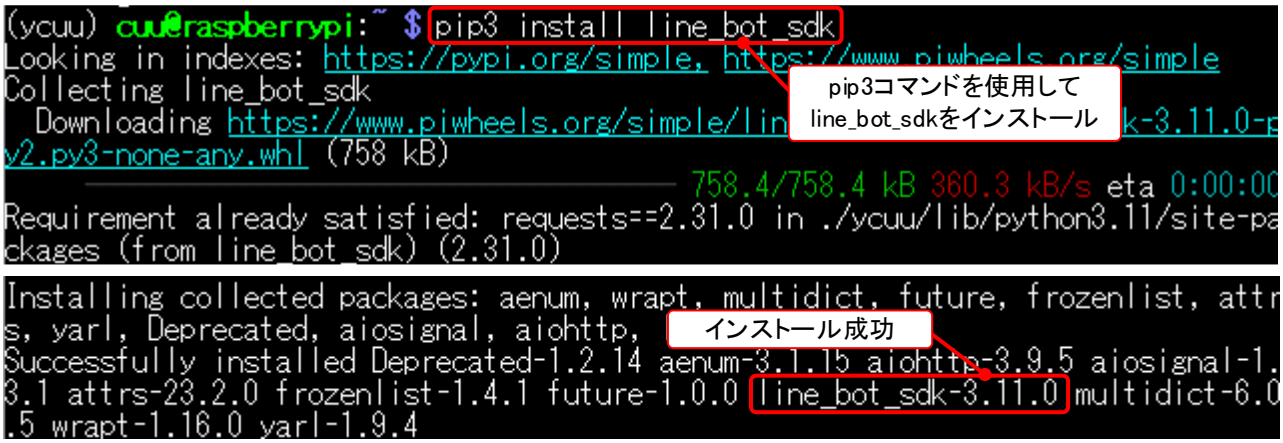
「ユーザーID」の情報は、プログラムで使用するのでメモ帳などに記録してください。

Copyright © 2024 YITJC All Rights Reserved.

手順④～line_bot_sdk インストール～

Python で LINE Messaging API を利用するために必要なライブラリ「line_bot_sdk」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install line_bot_sdk



～補足～

line_bot_sdk と同時に依存関係のあるパッケージもインストールされます。Deprecated（廃止予定のクラスやメソッドを通知）や aiohttp（非同期 HTTP クライアント・サーバー機能を提供）などがインストールされます。

手順⑤～環境変数設定～

LINE を利用するために必要な情報（チャネルアクセストークン）を環境変数に設定します。

環境変数に登録する情報は、次の情報です。

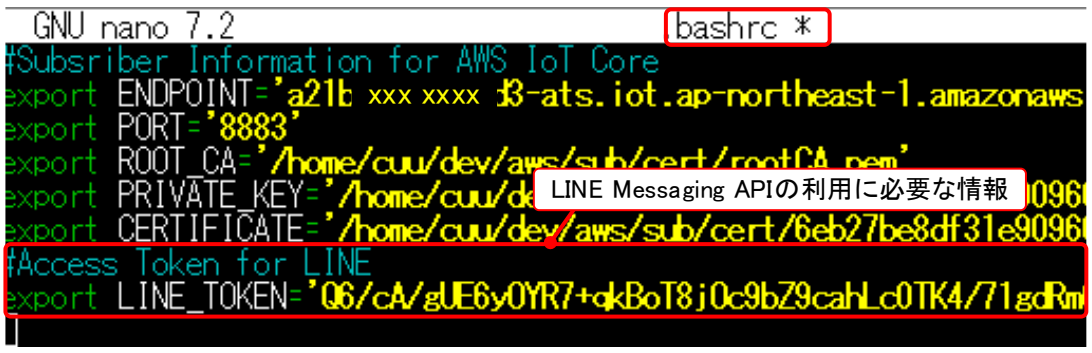
```
export LINE_TOKEN=' Q6/cA/gUE6yOYR7+qkBoxxxxxxxxxx'
```

Raspberry Pi が起動したときに環境変数を自動的に設定するため、「.bashrc」ファイルに環境変数の設定コマンドを記述します。

ホームディレクトリ（/home/cuu）にある「.bashrc」の内容を編集します。

(ycuu) cuu@raspberrypi:$ nano .bashrc

「.bashrc」ファイルの画面が起動したら、末尾に「export」コマンドで変数（チャネルアクセストークン）を追加（記述）します。



記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～環境変数の反映～

続いて、次のコマンドを実行して設定した環境変数を反映します。

(ycuu) cuu@raspberrypi:$ source ~/.bashrc

```
cuu@raspberrypi:~ $ source ~/.bash_profile
```

「~」の部分（チルダ）は、「Shift」+「～」で入力します。

手順⑦～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

50_LINEpush.py

[プログラム編集個所]

- 送信先「ユーザーID」

  「取得方法①」または「取得方法②」で取得したユーザーID に書き換えてください。

  ```
  #24行目 ユーザーID
  user_id_1 = 'U9ebf2641xxxxx3464455002d83'
  ```

手順⑧～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 50_LINEpush.py

手順⑨～プログラム実行～

対象プログラムを実行します。

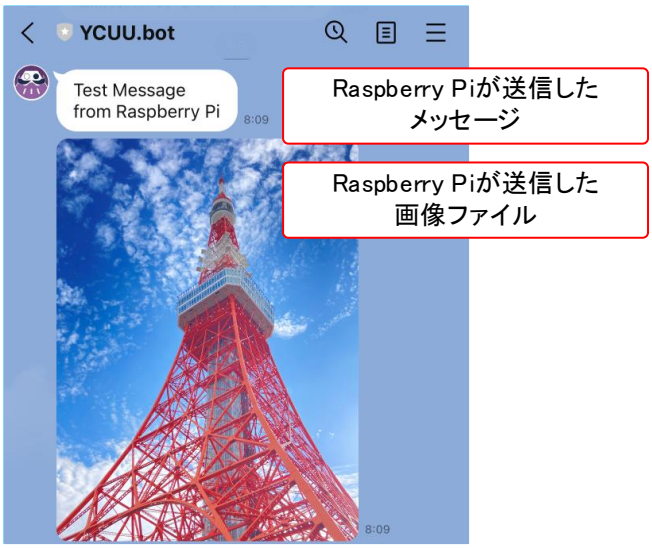
(ycuu) cuu@raspberrypi:~/dev $ ./50_LINEpush.py

次の画面が表示され、プログラムの中で指定した LINE ユーザー宛にメッセージが送信されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./50_LINEpush.py
LINE push Mission accomplished!
```

Copyright © 2024 YITJC All Rights Reserved.

スマートフォンの LINE アプリでメッセージの受信を確認してください。



～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from linebot.v3.messaging import MessagingApi, Configuration, ApiClient
5	from linebot.v3.messaging.models import (
6	TextMessage, PushMessageRequest, ImageMessage
7	)
8	import os
9	
10	#LINEアカウントの情報を環境変数から取得
11	def get_line_settings():
12	line_token = os.getenv('LINE_TOKEN')
13	return line_token
14	
15	#LINE push送信関数
16	def push_message():
17	#LINEアカウントの情報を環境変数から取得
18	line_token = get_line_settings()
19	configuration = Configuration(access_token=line_token)
20	api_client = ApiClient(configuration=configuration)
21	messaging_api = MessagingApi(api_client)
22	
23	#メッセージを送信するユーザーID

Copyright © 2024 YITJC All Rights Reserved.

```
24      user_id_1 = 'U9ebf264109089eee5ac3464455002d83'
25      #user_id_2 = 'U2bce1b65188281f0062f8bfa1b955c43'
26
27      text_message = TextMessage(text='Test Message ¥nfrom Raspberry Pi')
28      image_message = ImageMessage(
29          original_content_url='https://drive.google.com/uc?id=1vWlOKvK-8EjY5ofpRe0HWrPRNb5AbXkY',
30          preview_image_url='https://drive.google.com/uc?id=1vWlOKvK-8EjY5ofpRe0HWrPRNb5AbXkY'
31      )
32      #User1 (LINE Developerへ送信)
33      messaging_api.push_message(PushMessageRequest(to=user_id_1, messages=[text_message]))
34      messaging_api.push_message(PushMessageRequest(to=user_id_1, messages=[image_message]))
35
36      #User2 (LINE Friendへ送信)
37      #messaging_api.push_message(PushMessageRequest(to=user_id_2, messages=[text_message]))
38      #messaging_api.push_message(PushMessageRequest(to=user_id_2, messages=[image_message]))
39
40  if __name__ == '__main__':
41      try:
42          push_message()
43          print('LINE push Mission accomplished!')
44      except Exception as e:
45          print('Transmission Error:', e)
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: linebot.v3.messaging ライブラリから「MessagingApi」、「Configuration」、「ApiClient」クラスをインポート
- 5 行目～7 行目: linebot.v3.messaging.model ライブラリから「TextSendMessage」、「PushMessageRequest」、「ImageMessage」クラスをインポート
- 8 行目: OS と対話するためのライブラリ「os」をインポート

～環境変数から LINE Messaging API（チャネルアクセストークン）の情報取得～

- 11 行目～13 行目: 環境変数から LINE Messaging API の情報を取得するプログラム get_line_settings()関数
- 12 行目: LINE_TOKEN 取得
- 13 行目: 他の関数で使用する変数（line_token）を戻り値として返す

～メッセージ push～

- 16 行目～38 行目: Raspberry Pi が直接応答する push_message()関数
  指定した LINE ユーザーに対してメッセージと画像を送付
- 18 行目: line_token を get_line_settings()関数から取得

Copyright © 2024 YITJC All Rights Reserved.

- 19 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 20 行目: ApiClient クラスを使用して API クライアントを初期化
- 21 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 24 行目～25 行目: 送信先の LINE ユーザーID 設定
- 27 行目: 送信するテキストメッセージを定義
- 28 行目～31 行目: 送信する画像メッセージを定義
  画像メッセージを送信する場合は、URL を 2 つ指定します。
  1 つ目は、トーク画面に表示される「プレビュー画像」の URL
  2 つ目は、画像をタップしたときに表示する「フルサイズ画像」の URL です。同じ画像ファイル（URL）を設定できます。なお、URL には HTTPS（TLS 1.2 以降）の URL のみ指定できます。
- 33 行目～38 行目: LINE ユーザーにメッセージを送信
  messaging_api クラスの push_message メソッドを使用して送信します。
  第一引数に「ユーザーID」を指定します。
  第二引数に「テキストメッセージ」または「送信する画像ファイルの URL」を指定します。
- 40 行目～45 行目: メイン関数
- 41 行目～43 行目 :try ブロック
- 42 行目: Raspberry Pi が直接応答する push_message()関数実行
- 43 行目: 「メッセージ送信完了」の print 表示
- 44 行目～45 行目 :except ブロック
  LineBotApiError が発生した場合、print 表示します。

～参考～

LINE メッセージタイプに関する詳しい情報は、次の URL にアクセスして確認してください。

- メッセージタイプ

[URL] https://developers.line.biz/ja/docs/messaging-api/message-types/

Copyright © 2024 YITJC All Rights Reserved.

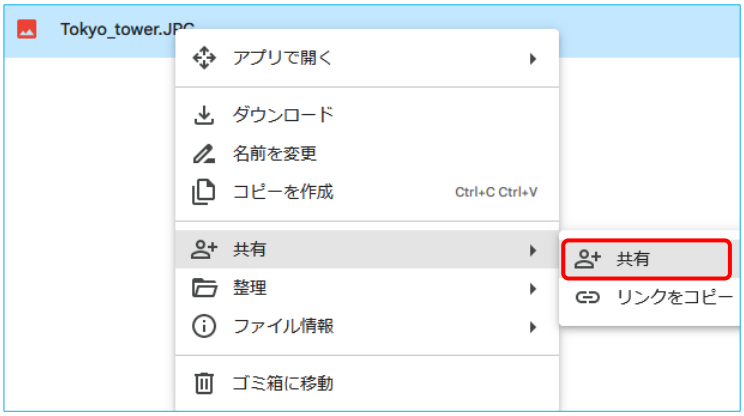
参考～画像ファイル URL 取得～

Google Drive に保存されている「画像ファイル」の URL を取得する方法について説明します。

手順①～ファイル共有設定～

Google Drive 上に保存されている画像ファイルを「共有」に設定します。

画像ファイルを右クリックし、「共有」をクリックしてください。

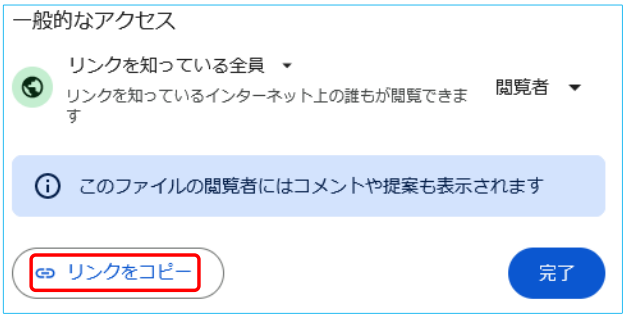


手順②～共有設定変更～

「リンクを知っている全員」に変更してから「完了をクリックしてください。

手順③～共有リンク（URL）取得～

「リンクをコピー」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

コピーした URL を確認すると、URL の一部にファイル ID が含まれています。

ファイルID

https://drive.google.com/file/d/1PdrBNgt6rA9-z9ThB7ruPIz8G4m0-eUq/view?usp=sharing

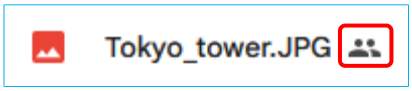
手順④～画像ファイルのリンク先（URL）作成～

手順③で取得した「ファイル ID」を使用して、画像ファイルのリンク先（URL）を作成します。

※赤字の部分を「ファイル ID」に書き換えてください。

https://drive.google.com/uc?id=1PdrBNgt6rA9-z9ThB7ruPIz8G4m0-eUq

画像ファイルが共有されると、ファイルの横に「共有中」の印が表示されます。



Copyright © 2024 YITJC All Rights Reserved.

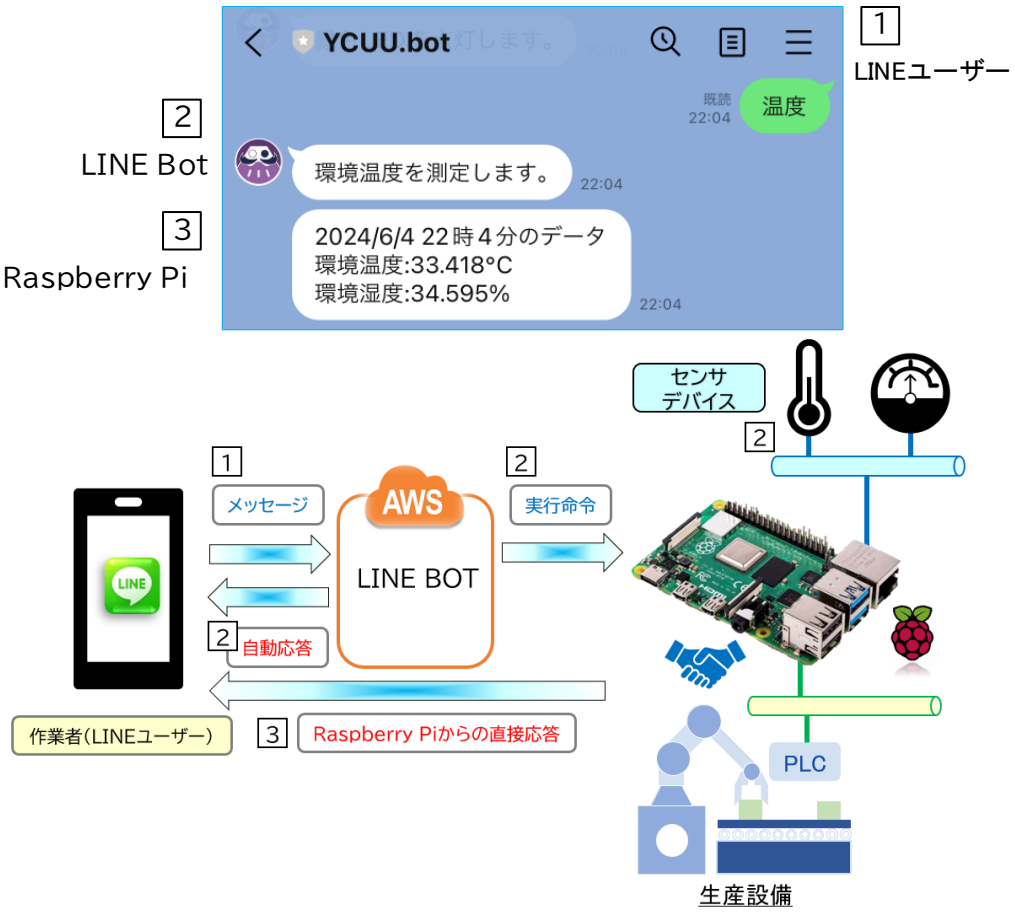
10.15 対話機能をもつ IoT システム（ver5）

対話機能をもつ IoT システム ver5 を構築します。具体的には、LINE ユーザーから送信されたメッセージに対して Raspberry Pi が直接応答する「対話型システム」を構築します。

「対話型システム」は、生産設備（IoT システム）と従業員（LINE ユーザー）との間で双方向コミュニケーションを可能にするシステムです。IoT システムの中心である Raspberry Pi は、センサやカメラ、生産設備からデータを取得できます。また、取得したデータの蓄積や可視化、クラウドサービスとの連携など、Raspberry Pi により実現できる機能は多岐にわたります。そのため、「対話機能」を活用して、これまで生産現場の従業員が担っていた業務を IoT システム（Raspberry Pi）に置き換えることで、生産性向上や従業員の負担軽減などの効果が期待できます。

～対話型システム～

ここで構築する「対話型システム」は、LINE ユーザーから送信されるメッセージ「温度」をトリガにして、BME280 センサ（温度・湿度）の値を取得し、LINE ユーザーに直接応答するシステムです。「対話システム」の構成を LINE メッセージ画面およびシステム構成図に示します。



～メッセージのやり取り～

1. 指定メッセージ「温度」を送信
2. 指定メッセージ「温度」を受信
   - ✓ LINE Bot が「環境温度を測定します。」と自動応答
   - ✓ Publish/Subscribe 通信により Raspberry Pi はメッセージを受信
   - ✓ Raspberry Pi は温度センサのデータ取得（命令実行）
3. Raspberry Pi は、アナログ温度センサの測定データおよび測定時刻を LINE ユーザーに対して直接応答

Copyright © 2024 YITJC All Rights Reserved.

手順①～Lambda プログラム作成～

「10.8 Bot プログラム作成」と同じ手順で AWS Lambda で実行する LINE Bot プログラムを作成・保存します。

[対象プログラム]

lambda_function.py

（共有サイトにある「lambda_function_temp.py」ファイルを編集して使用してください。）

～注意～

「lambda_function_temp.py」について、次の 3 か所を編集してください。

[プログラム編集個所]

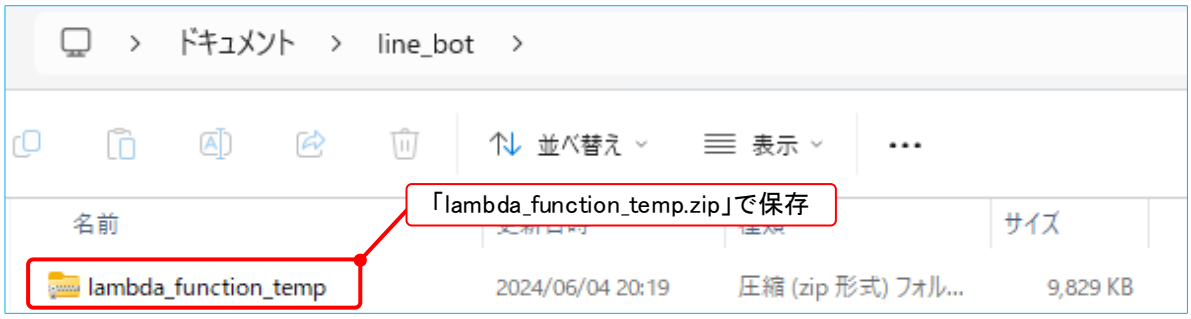
- 「チャネルシークレット」、「チャネルアクセストークン」
  「10.6 LINE Messaging API 設定」で取得した値に書き換えてください。

```
#12行目 チャネルアクセストークン
CHANNEL_ACCESS_TOKEN = 'xxxxxxx'
#14行目 チャネルシークレット
CHANNEL_SECRET = 'xxxxxxx'
```

- Subscriber のエンドポイント指定
  「10.3 Subscribe 実装」手順⑧で取得した「エンドポイント（end point）」を使用します。
  「https://」のあとに「エンドポイント」を追加してください。

```
#27 行目 Subscriber End Point 指定
client = boto3.client('iot-data', endpoint_url='https://end point')
```

zip ファイルは、「lambda_function_temp.zip」のファイル名で保存してください。



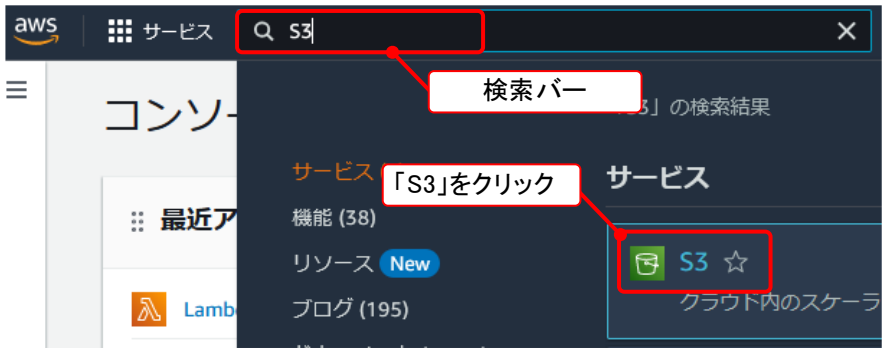
Copyright © 2024 YITJC All Rights Reserved.

手順②～Lambda 関数ファイルアップロード～

次の URL をクリックして AWS にログインしてください。

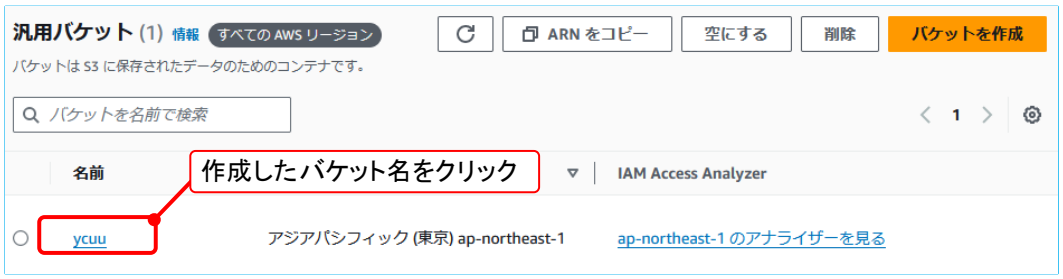
[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「S3」を検索してください。



「10.9 Bot プログラム保存」と同じ手順で AWS S3 に Lambda 関数ファイル一式をアップロードします。

「作成したバケット名」をクリックしてください。

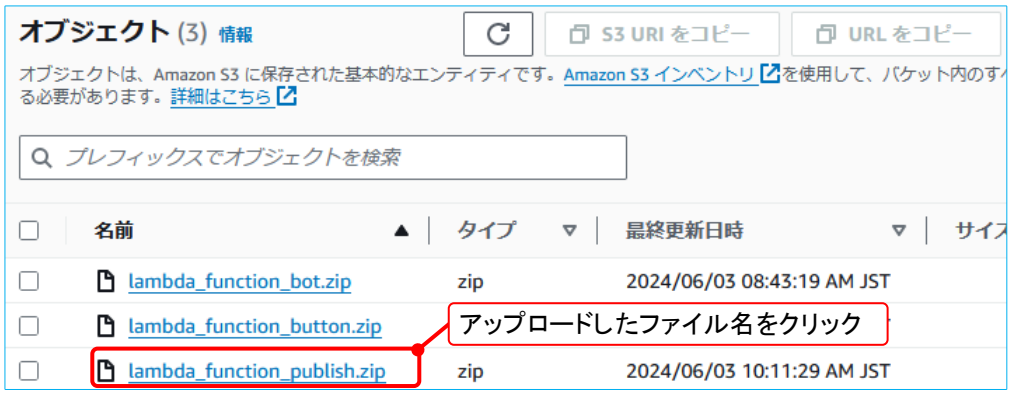


「アップロード」をクリックしてください。

「ファイルを追加」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「lambda_function_temp.zip」ファイルをアップロードしたあと、アップロードしたファイルの URL 情報を取得します。アップロードしたファイル名をクリックして URL 情報を取得してください。

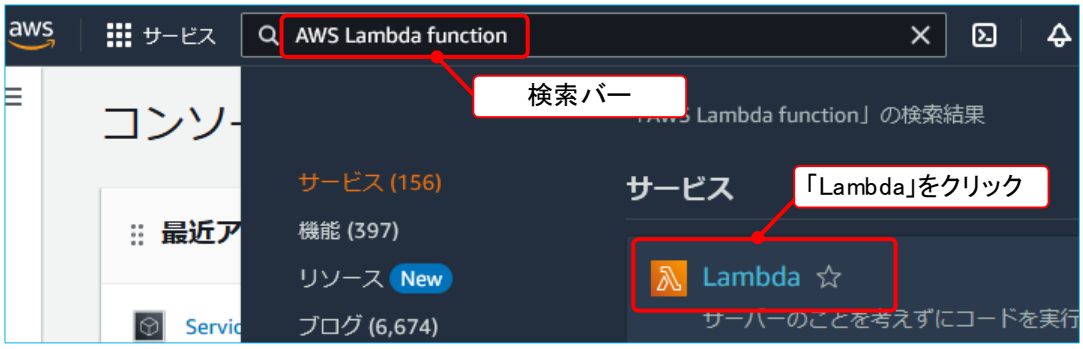


手順③～Lambda 関数デプロイ～

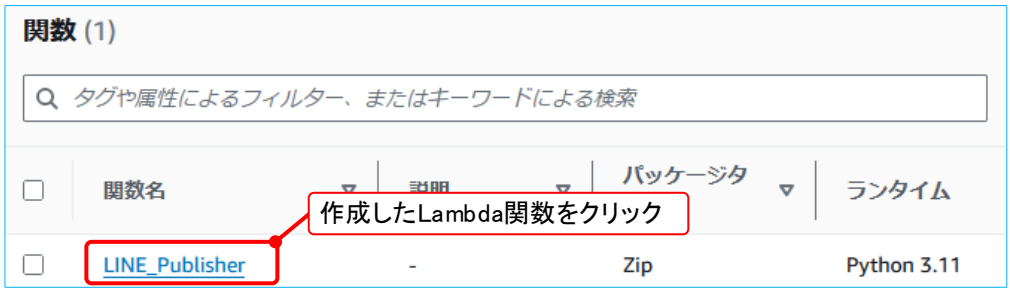
次の URL をクリックして AWS にログインしてください。

[URL] https://portal.aws.amazon.com

画面上部の検索バーを使用して、「AWS Lambda Function」を検索してください。



作成した「Lambda 関数」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「コード」タブを選択し、「アップロード元」から「Amazon S3 の場所」をクリックしてください。

手順②で取得した Amazon S3 の URL を入力してから「保存」をクリックしてください。

手順④～対話プログラム編集～

対象プログラム「TEMP.py」の送信先「ユーザーID」を編集してください。

[対象プログラム]

TEMP.py

[プログラム編集個所]

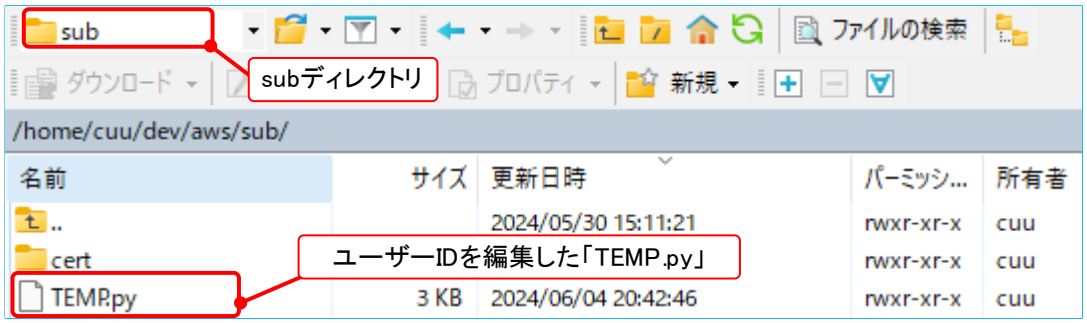
- 送信先「ユーザーID」

  「取得方法①」または「取得方法②」で取得したユーザーID に書き換えてください。

  #42行目 ユーザーID

  user_id_1 = 'U9ebf2641xxxxx3464455002d83'

プログラムの編集が終了したら、編集した「TEMP.py」を「/home/cuu/dev/aws/sub」ディレクトリに配置してください。



Copyright © 2024 YITJC All Rights Reserved.

ここからは、「10.3 Publish 機能実装」で作成したサービスファイル（Subscribe.service）を編集します。Raspberry Pi から LINE ユーザーに対して直接応答するために必要な LINE Messaging API の情報をサービスファイルに追加します。

手順⑤～サービス・タイマー停止～

次のコマンドを実行し、現在稼働している「Subscribe.service」を停止してください。

(ycuu) cuu@raspberrypi:$ sudo systemctl stop Subscribe.service

(ycuu) cuu@raspberrypi:$ sudo systemctl disable Subscribe.service

手順⑥～Service ファイル編集～

「Subscribe.service」ファイルを編集するため、「/etc/systemd/system」ディレクトリへ移動します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/systemd/system

続いて、「Subscribe.service」ファイルを編集します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo nano Subscribe.service

～参考～

Unit ファイルの「上書き」、「終了」は、次の手順を参考にしてください。

- 編集が終了したら、「ctrl + O（オー）」キーを押下して上書きするファイル名を確認
- 「Enter」キーを押下して上書き保存を実行
- 「Ctrl + X」キーを押して元の画面に戻る

Copyright © 2024 YITJC All Rights Reserved.

- 環境変数追加

  LINE Messaging API を利用するために必要な情報（チャネルアクセストークン）を Unit ファイルの [Service]セクションに記載してください。

-記載方法- Environment="環境変数名=設定値"

手順⑦～サービス再読み込み～

編集したサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順⑧～サービス自動起動有効化～

次のコマンドを実行してサービスの自動起動を有効化します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable Subscribe.service

手順⑨～サービス開始～

次のコマンドを実行してサービスを開始します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start Subscribe.service

```
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl enable Subscribe.service
Created symlink /etc/systemd/system/multi-user.target.wants/Subscribe.service → /etc/sy
stemd/system/Subscribe.service.
(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl start Subscribe.service
```

Copyright © 2024 YITJC All Rights Reserved.

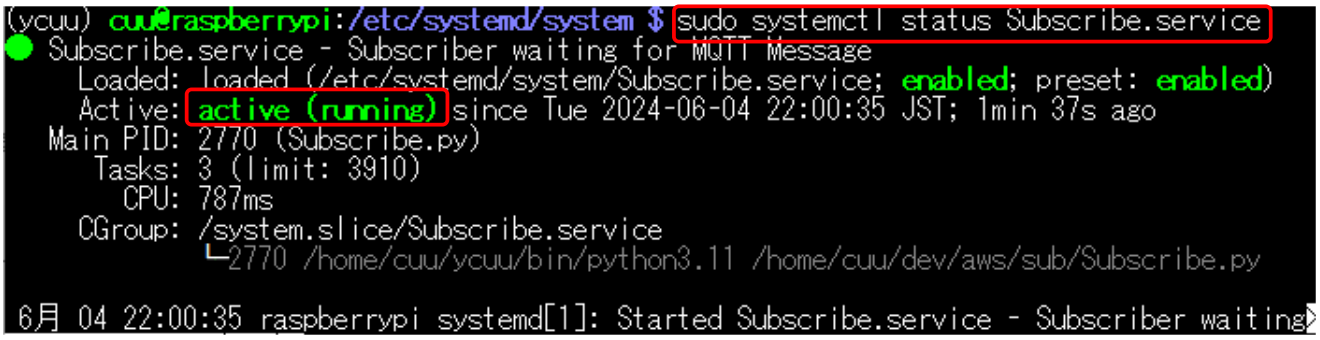
手順⑩～サービス動作状況確認～

サービスの動作状況を確認します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl status Subscribe.service

正常に稼働（自動起動）していれば、次の画面 active（running）が表示されます。

この画面表示を終了する場合は、「q」を押してください。



手順⑪～システムログ確認～

IoT システム ver5 の動作確認を行うため、次のコマンドを実行してシステムのログを確認してください。

(ycuu) cuu @raspberrypi:~ $ journalctl -u Subscribe.service -n 10

- **オプションの設定（例）**

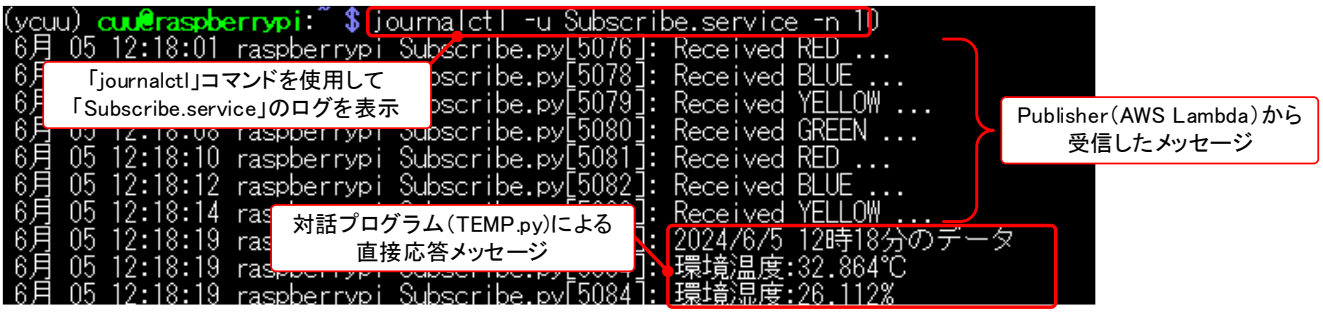
「-u Unit 名」：指定した Unit のログを表示

「-n 数」：末尾（直近）から指定した数文のログを表示

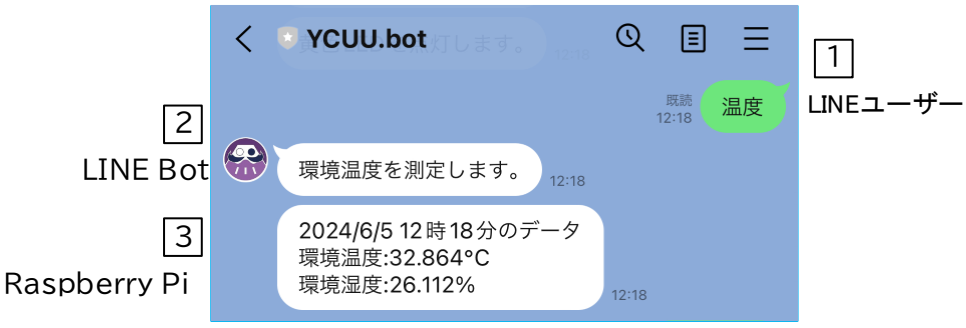
「-f」：リアルタイム監視（デバッグに便利）

「-xe」：詳細情報を表示

対話システムの動作確認を行うため、LINE から「温度」メッセージを送信してからログを確認してください。

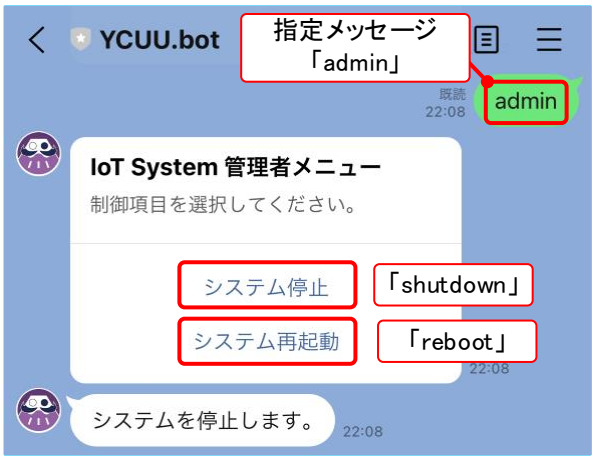


LINE ユーザー、LINE Bot（AWS）、Raspberry Pi の 3 者で「対話」が実現しています。



Copyright © 2024 YITJC All Rights Reserved.

「Subscribe.py」は、「シャットダウン」や「再起動」のコマンドにも対応しています。指定メッセージ「admin」と送信すると、Raspberry Pi のシャットダウンや再起動を実行できます。



～Publish プログラム解説～

行	プログラム
1	from linebot.v3 import WebhookHandler
2	from linebot.v3.exceptions import InvalidSignatureError
3	from linebot.v3.messaging import (
4	ApiClient, Configuration, MessagingApi,
5	ReplyMessageRequest, TextMessage, TemplateMessage, ButtonsTemplate, PostbackAction
6	)
7	from linebot.v3.webhooks import (
8	MessageEvent, TextMessageContent, PostbackEvent
9	)
10	
11	#チャネルアクセストークン
12	CHANNEL_ACCESS_TOKEN = 'Q6/cA/gUE6yOYR7+xxxxxxxxxxbZ9cahLc0TK4/71gdRm0v42W2f+JrlP6TjVQ+OQIOf29CAU/Hetv/t7+oWfNWV/eBXirpzgh1fB99QQTayCTuAcvqEO8iH2L7jACAcSKywehFZq3j/6UdZhhXnmAdB04t89/1O/w1cDnyilFU='
13	#チャネルシークレット
14	CHANNEL_SECRET = 'e6bdaxxxxxxxxxx963eee04486668ba4'
15	
16	configuration = Configuration(access_token=CHANNEL_ACCESS_TOKEN)
17	api_client = ApiClient(configuration=configuration)
18	line_bot_api = MessagingApi(api_client)
19	handler = WebhookHandler(CHANNEL_SECRET)
20	
21	import logging
22	logger = logging.getLogger()

Copyright © 2024 YITJC All Rights Reserved.

```
23  logger.setLevel(logging.ERROR)
24
25  import boto3
26  #Subscriber End Point指定
27  client = boto3.client('iot-data', endpoint_url='https://xxxxxxxxxx-ats.iot.ap-northeast-1.amazonaws.com')
28
29  def lambda_handler(event, context):
30      #Line Uer ID取得するためeventを表示
31      print(f"{event}")
32      signature = event["headers"]["x-line-signature"]
33      body = event["body"]
34      ok_json = {"isBase64Encoded": False,
35                 "statusCode": 200,
36                 "headers": {},
37                 "body": ""}
38      error_json = {"isBase64Encoded": False,
39                    "statusCode": 403,
40                    "headers": {},
41                    "body": "Error"}
42
43      try:
44          handler.handle(body, signature)
45      except InvalidSignatureError:
46          return error_json
47      return ok_json
48
49  @handler.add(MessageEvent, message=TextMessageContent)
50  def handle_message(event):
51      text = event.message.text
52      if text == 'password':
53          #ボタンテンプレート1呼出
54          send_buttons_template1(event.reply_token)
55      elif text == "admin":
56          #ボタンテンプレート2呼出
57          send_buttons_template2(event.reply_token)
58      elif text == "温度":
59          #boto3 Publish
60          response = client.publish(topic='system', qos=1, payload='temp')
61          line_bot_api.reply_message(
```

Copyright © 2024 YITJC All Rights Reserved.

```
62                  ReplyMessageRequest(
63                      replyToken=event.reply_token,
64                      messages=[TextMessage(text='環境温度を測定します。')]
65                  )
66              )
67          else:
68              #boto3 Publish
69              response = client.publish(topic='system', qos=1, payload='xxxx')
70              line_bot_api.reply_message(
71                  ReplyMessageRequest(
72                      replyToken=event.reply_token,
73                      messages=[TextMessage(text='遠隔制御が許可されていません。')]
74                  )
75              )
76
77  def send_buttons_template1(reply_token):
78          buttons_template = ButtonsTemplate(
79              title='生産性向上IoT System', text='制御項目を選択してください。',
80              actions=[
81                  PostbackAction(label='緑', data='GREEN'),
82                  PostbackAction(label='赤', data='RED'),
83                  PostbackAction(label='青', data='BLUE'),
84                  PostbackAction(label='黄', data='YELLOW'),
85              ]
86          )
87          template_message = TemplateMessage(alt_text='Buttons alt text', template=buttons_template)
88          line_bot_api.reply_message(
89              ReplyMessageRequest(
90                  replyToken=reply_token,
91                  messages=[template_message]
92              )
93          )
94
95  def send_buttons_template2(reply_token):
96          buttons_template = ButtonsTemplate(
97              title='IoT System　管理者メニュー', text='制御項目を選択してください。',
98              actions=[
99                  PostbackAction(label='システム停止', data='システム停止'),
100                 PostbackAction(label='システム再起動', data='システム再起動'),
```

Copyright © 2024 YITJC All Rights Reserved.

```
            ]
        )
        template_message = TemplateMessage(alt_text='Buttons alt text', template=buttons_template)
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=reply_token,
                messages=[template_message]
            )
        )

@handler.add(PostbackEvent)
def handle_postback(event):
    if event.postback.data == 'GREEN':
        #boto3 Publish
        response = client.publish(topic='system', qos=1, payload='green')
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=event.reply_token,
                messages=[TextMessage(text='緑色LEDを点灯します。')]
            )
        )
    elif event.postback.data == 'RED':
        #boto3 Publish
        response = client.publish(topic='system', qos=1, payload='red')
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=event.reply_token,
                messages=[TextMessage(text='赤色LEDを点灯します。')]
            )
        )
    elif event.postback.data == 'BLUE':
        #boto3 Publish
        response = client.publish(topic='system', qos=1, payload='blue')
        line_bot_api.reply_message(
            ReplyMessageRequest(
                replyToken=event.reply_token,
                messages=[TextMessage(text='青色LEDを点灯します。')]
            )
        )
```

Copyright © 2024 YITJC All Rights Reserved.

```
140         elif event.postback.data == 'YELLOW':
141             #boto3 Publish
142             response = client.publish(topic='system', qos=1, payload='yellow')
143             line_bot_api.reply_message(
144                 ReplyMessageRequest(
145                     replyToken=event.reply_token,
146                     messages=[TextMessage(text='黄色LEDを点灯します。')]
147                 )
148             )
149         elif event.postback.data == 'システム停止':
150             #boto3 Publish
151             response = client.publish(topic='system', qos=1, payload='shutdown')
152             line_bot_api.reply_message(
153                 ReplyMessageRequest(
154                     replyToken=event.reply_token,
155                     messages=[TextMessage(text='システムを停止します。')]
156                 )
157             )
158         elif event.postback.data == 'システム再起動':
159             #boto3 Publish
160             response = client.publish(topic='system', qos=1, payload='reboot')
161             line_bot_api.reply_message(
162                 ReplyMessageRequest(
163                     replyToken=event.reply_token,
164                     messages=[TextMessage(text='システムを再起動します。')]
165                 )
166             )
```

灰色部は、これまでに説明した内容です。

- 1 行目: linebot.v3 ライブラリの「WebhookHandler」をインポート
- 2 行目: linebot.v3.exceptions ライブラリの「InvalidSignatureError」クラスをインポート
- 3 行目～6 行目: linebot.v3.messaging ライブラリの「ApiClient」、「Configuration」、「MessagingApi」、「ReplyMessageRequest」、「TextMessage」、「TextMessage」、「TemplateMessage」、「ButtonsTemplate」、「PostbackAction」クラスをインポート
- クラスをインポート
  このほか、画像（イメージ）ファイルを扱う「ImageMessage」クラス、ボタンテンプレートを扱う「ButtonsTemplate」クラス、ビデオメッセージを扱う「VideoMessage」クラスなどがあります。
- 7 行目～9 行目: linebot.v3.webhook ライブラリの「MessageEvent」、「TextMessageContent」クラスとインポート

Copyright © 2024 YITJC All Rights Reserved.

- 12 行目: チャネルアクセストークン設定
- 14 行目: チャネルアシークレット設定

  ※「チャネルアクセストークン」および「チャネルシークレット」は、AWS Lambda で環境変数として保存することも可能ですが、ここではプログラム内に記述しています。
- 16 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 17 行目: ApiClient クラスを使用して API クライアントを初期化
- 18 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 19 行目: WebhookHandler クラスを使用してチャネルシークレットを設定
- 21 行目: プログラム実行時に発生したイベントを追跡するためのライブラリ「logging」をインポート
- 22 行目: 現在のロギング情報取得（引数はファイル名を設定）
- 23 行目: ロギングのレベルを「ERROR」に設定
- 25 行目: boto3 ライブラリをインポート

  boto3 は「AWS SDK for Python」の別称であり、Python を介して AWS を操作するためのライブラリです。
  余談ですが、アマゾンカワイルカ（Boto Pink Dolphin）にちなんで名付けられたそうです。
- 27 行目: boto3 ライブラリの client()メソッドを使用してインスタンスを生成

  「https://」のあとに「エンドポイント」を追加
- 29 行目～47 行目: Lambda 関数 Handler
- 31 行目: リクエストデータ print 表示

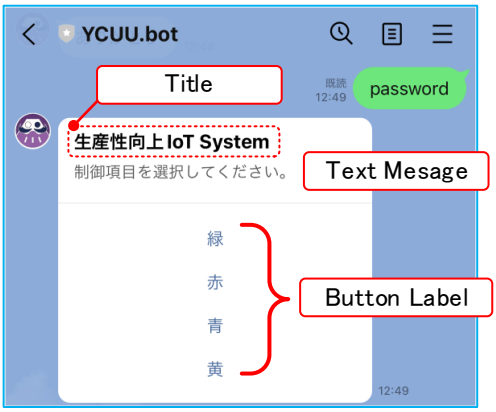
  「f」は、フォーマット済み文字リテラル（f-string）です。
- 32 行目: リクエストヘッダーの「x-line-signature」に含まれる値（署名検証のための値）を取得
- 33 行目: リクエストボディを取得
- 34 行目～37 行目: ok_json 定義
- 38 行目～41 行目: error_json 定義
- 43 行目～44 行目: try ブロック
- 44 行目: WebhookHandler の「handler.handle」メソッドを使用

  署名「Signature」を検証し、問題なければ handle に定義されている関数を呼び出します。
- 45 行目～46 行目: except ブロック
- 46 行目: 署名検証が失敗した場合、error_json を API Gateway にレスポンス
- 47 行目: handle の処理が問題なく終了すれば、ok_json を API Gateway にレスポンス
- 49 行目～75 行目: 「handler.add」デコレータを使用

  引数にリクエストのイベントを指定します。ここでは、「MessageEvent」、「TextMessage」を指定しています。
- 51 行目: LINE ユーザーからの送信メッセージを変数「text」に格納
- 52 行目～54 行目: 指定メッセージ「password」を受信したときの処理
- 54 行目: ボタンテンプレート表示関数 send_buttons_template1()を呼び出し
- 55 行目～57 行目: 指定メッセージ「admin」を受信したときの処理
- 57 行目: ボタンテンプレート表示関数 send_buttons_template2()を呼び出し
- 58 行目～66 行目: 指定メッセージ「温度」を受信したときの処理

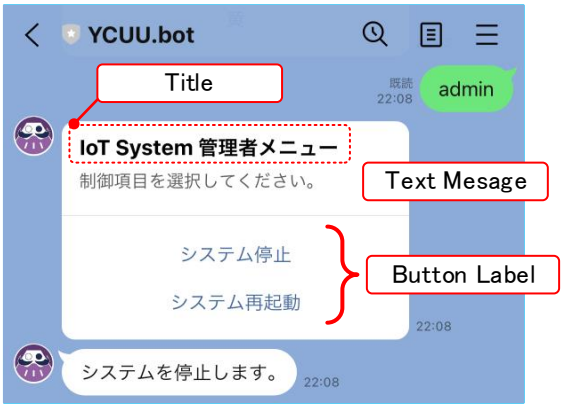
Copyright © 2024 YITJC All Rights Reserved.

- 60 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=temp」を指定して Publish
- 61 行目～66 行目: LINE Bot 動作
  「環境温度を測定します。」と自動応答
- 67 行目～75 行目: 指定メッセージ以外のメッセージを受信したときの処理
- 69 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=xxxx」を指定して Publish
- 70 行目～75 行目: LINE Bot 動作
  「遠隔制御が許可されていません。」と自動応答
- 77 行目～93 行目: ボタンテンプレート表示プログラム send_buttons_template1()関数
- 78 行目～86 行目: ボタンテンプレート設定
  - ✓ Title: 生産性向上 IoT System
  - ✓ Text Message: 制御項目を選択してください。
  - ✓ Button Label / data
    data は、ボタンが押されたときに出力されます。



Button Label	data
緑	GREEN
赤	RED
青	BLUE
黄	YELLOW

- 87 行目: template_message 定義
- 88 行目～93 行目: LINE Bot 動作
  LINE Bot の応答（オウム返し）は LineBotApi の「line_bot_api.reply_message」メソッドを使用します。
  第一引数の「line_event.reply_token」はイベントの応答に用いるトークンです。
  第二引数には 87 行目で定義した template_message を渡しています。
- 95 行目～109 行目: ボタンテンプレート表示プログラム send_buttons_template2()関数
- 96 行目～102 行目: ボタンテンプレート設定
  - ✓ Title: 生産性向上 IoT
  - ✓ Text Message: 制御項目を選択してください。
  - ✓ Button Label / data
    data は、ボタンが押されたときに出力されます。



Button Label	data
システム停止	システム停止
システム再起動	システム再起動

- 103 行目: template_message 定義
- 104 行目～109 行目: LINE Bot 動作
  LINE Bot の応答（オウム返し）は LineBotApi の「line_bot_api.reply_message」メソッドを使用します。
  第一引数の「line_event.reply_token」はイベントの応答に用いるトークンです。
  第二引数には 87 行目で定義した template_message を渡しています。

Copyright © 2024 YITJC All Rights Reserved.

- 111 行目:「handler.add」デコレータを使用
  引数にリクエストのイベントを指定します。
  ここでは、ボタンが押されたときに「data」が出力される「PostbackAction」を指定しています。
- 113 行目～121 行目: data=「GREEN」を受け取った時の処理
- 115 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=green」を指定して Publish
  「緑色 LED を点灯します。」と自動応答
- 122 行目～130 行目: data=「RED」を受け取った時の処理
- 124 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=red」を指定して Publish
  「赤色 LED を点灯します。」と自動応答
- 131 行目～139 行目: data=「BLUE」を受け取った時の処理
- 133 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=blue」を指定して Publish
  「青色 LED を点灯します。」と自動応答
- 140 行目～148 行目: data=「YELLOW」を受け取った時の処理
- 142 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=yellow」を指定して Publish
  「黄色 LED を点灯します。」と自動応答
- 149 行目～157 行目: data=「システム停止」を受け取った時の処理
- 151 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=shutdown」を指定して Publish
  「黄色 LED を点灯します。」と自動応答
- 158 行目～166 行目: data=「システム再起動」を受け取った時の処理
- 160 行目: Publish 動作
  「topic=system」、「qos=1」、「payload=reboot」を指定して Publish
  「黄色 LED を点灯します。」と自動応答

Copyright © 2024 YITJC All Rights Reserved.

～対話プログラム(TEMP.py)解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	from linebot.v3.messaging import MessagingApi, Configuration, ApiClient
5	from linebot.v3.messaging.models import (
6	TextMessage, PushMessageRequest, ImageMessage
7	)
8	import os
9	
10	#LINEアカウントの情報を環境変数から取得
11	def get_line_settings():
12	line_token = os.getenv('LINE_TOKEN')
13	return line_token
14	
15	import time
16	import datetime
17	import RPi.GPIO as GPIO
18	import sys
19	sys.path.append('/home/cuu/dev')
20	import BME280_Class
21	
22	# Sets up pins as outputs
23	def setup_gpio():
24	GPIO.setmode(GPIO.BCM)
25	GPIO.setup(4, GPIO.OUT)
26	GPIO.output(4, GPIO.LOW)
27	GPIO.setup(17, GPIO.OUT)
28	GPIO.output(17, GPIO.LOW)
29	GPIO.setup(27, GPIO.OUT)
30	GPIO.output(27, GPIO.LOW)
31	GPIO.setup(22, GPIO.OUT)
32	GPIO.output(22, GPIO.LOW)
33	
34	def push_message():
35	#LINEアカウントの情報を環境変数から取得
36	line_token = get_line_settings()
37	configuration = Configuration(access_token=line_token)

Copyright © 2024 YITJC All Rights Reserved.

```
38      api_client = ApiClient(configuration=configuration)
39      messaging_api = MessagingApi(api_client)
40
41      #メッセージを送信するユーザーID
42      user_id_1 = 'U9ebf2xxxxxxxxx5ac3464455002d83'
43      #user_id_2 = 'U2bce1bxxxxxxxx062f8bfa1b955c43'
44
45      now=datetime.datetime.today()    #現在時刻取得
46      text1 = str(now.year) + u"/" + str(now.month) + u"/" + str(now.day) + u" " + str(now.hour) + u"時" + str(now.minute)
    + u"分のデータ"
47      text2 = '環境温度:' + '{:.3f}'.format(temp) + u"℃"
48      text3 = '環境湿度:' + '{:.3f}'.format(hum) + u"%"
49      text_message = text1 + '¥n' + text2 + '¥n' + text3
50      print (text_message)
51
52      #TextMessageオブジェクトを作成
53      text_message_obj = TextMessage(text=text_message)
54
55      #User1 (LINE Developerへ送信)
56      messaging_api.push_message(PushMessageRequest(to=user_id_1, messages=[text_message_obj]))
57
58      #User2 (LINE Friendへ送信)
59      #messaging_api.push_message(PushMessageRequest(to=user_id_2, messages=[text_message_obj]))
60
61
62  if __name__ == '__main__':
63      try:
64          setup_gpio()
65          bme = BME280_Class.BME280()
66          bme.readData()
67          temp = bme.getTemperature()
68          hum = bme.getHumidity()
69          GPIO.output(4, GPIO.HIGH)
70          GPIO.output(17, GPIO.HIGH)
71          GPIO.output(27, GPIO.HIGH)
72          GPIO.output(22, GPIO.HIGH)
73          time.sleep(2.0)
74          push_message()
75          GPIO.output(4, GPIO.LOW)
```

Copyright © 2024 YITJC All Rights Reserved.

```
76              GPIO.output(17, GPIO.LOW)
77              GPIO.output(27, GPIO.LOW)
78              GPIO.output(22, GPIO.LOW)
79          except KeyboardInterrupt:
80              pass
81          finally:
82              GPIO.cleanup()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: linebot.v3.messaging ライブラリから「MessagingApi」、「Configuration」、「ApiClient」クラスをインポート
- 5 行目～7 行目: linebot.v3.messaging.model ライブラリから「TextSendMessage」、「PushMessageRequest」、「ImageMessage」クラスをインポート
- 8 行目: OS と対話するためのライブラリ「os」をインポート

～環境変数から LINE Messaging API（チャネルアクセストークン）の情報取得～

- 11 行目～13 行目: 環境変数から LINE Messaging API の情報を取得するプログラム get_line_settings()関数
- 12 行目: LINE_TOKEN 取得
- 13 行目: 他の関数で使用する変数（line_token）を戻り値として返す
- 15 行目: sleep 関数を使用するため time ライブラリをインポート
- 16 行目: 日付・時刻を取得するため datetime ライブラリをインポート
- 17 行目: GPIO 制御を行うため RPi.GPIO ライブラリをインポート
- 18 行目: Python インタプリタを制御するために必要なライブラリ sys をインポート
- 19 行目: Python のモジュール検索パスに「/home/cuu/dev」を追加

  ～補足～

  対話プログラム「TEMP.py」が保存されているディレクトリは「/home/cuu/dev/aws/sub」です。

  同じディレクトリに「BME280_Class.py」がないため、モジュール検索パスを追加しています。
- 20 行目: BME280_Class モジュールをインポート（import）
- 23 行目～32 行目: GPIO 端子初期設定プログラム setup_gpio()関数

～メッセージ push～

- 34 行目～59 行目: Raspberry Pi が直接応答する push_message()関数
  指定した LINE ユーザーに対してメッセージと画像を送付
- 36 行目: line_token を get_line_settings()関数から取得
- 37 行目: Configuration クラスを使用してチャネルアクセストークンを設定
- 38 行目: ApiClient クラスを使用して API クライアントを初期化
- 39 行目: MessagingApi クラスを使用してメッセージング API を初期化
- 42 行目～43 行目: 送信先の LINE ユーザーID 設定

Copyright © 2024 YITJC All Rights Reserved.

- 45 行目: 現在の日付・時刻取得
- 46 行目～49 行目: 送信するテキストメッセージを定義
  text1: 現在の日付・時刻
  text2: 温度測定結果
  text3: 湿度測定結果
  text_message = text1 + '¥n' + text2 + '¥n' + text3
- 50 行目: 送信するメッセージを print 表示
- 53 行目: TextMessage()クラスを使用してテキストメッセージのオブジェクトを作成
- 56 行目: LINE ユーザー#1 にメッセージを送信
- 59 行目: LINE ユーザー#2 にメッセージを送信
  ～補足～
  LINE Bot を友達登録した端末ごとにユーザーID が異なるため、直接応答の送信先を指定できます。
- 62 行目～82 行目: メイン関数
- 63 行目～78 行目 :try ブロック
- 64 行目: GPIO 端子初期設定プログラム setup_gpio()関数実行
- 65 行目: クラス生成
- 66 行目: クラスで定義されている readData()関数の実行
- 67 行目: クラスで定義されている getTemperature()関数を実行して「温度」データを取得
- 68 行目: クラスで定義されている getHumidity()関数を実行して「湿度」データを取得
- 69 行目～73 行目: LED（赤・黄・緑・青）を点灯
- 74 行目: 対話の応答（Raspberry Pi による直接応答）プログラム push_message()関数実行
- 75 行目～78 行目: LED（赤・黄・緑・青）を消灯
- 79 行目～80 行目 :except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文（何も処理しない）を記述しています。
- 81 行目～82 行目: finally ブロック
  例外の発生によらず必ず実行される処理を記述しています。
  本プログラムは GPIO を使用しているため、GPIO.cleanup()を記述しています。これにより、例外の発生によらず GPIO ピンのシステムクリアが必ず実行されます。

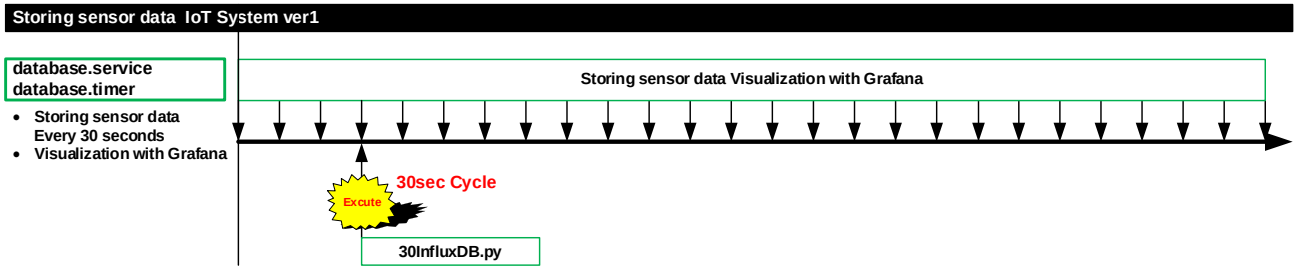
Copyright © 2024 YITJC All Rights Reserved.

参考～IoT システム稼働状況～

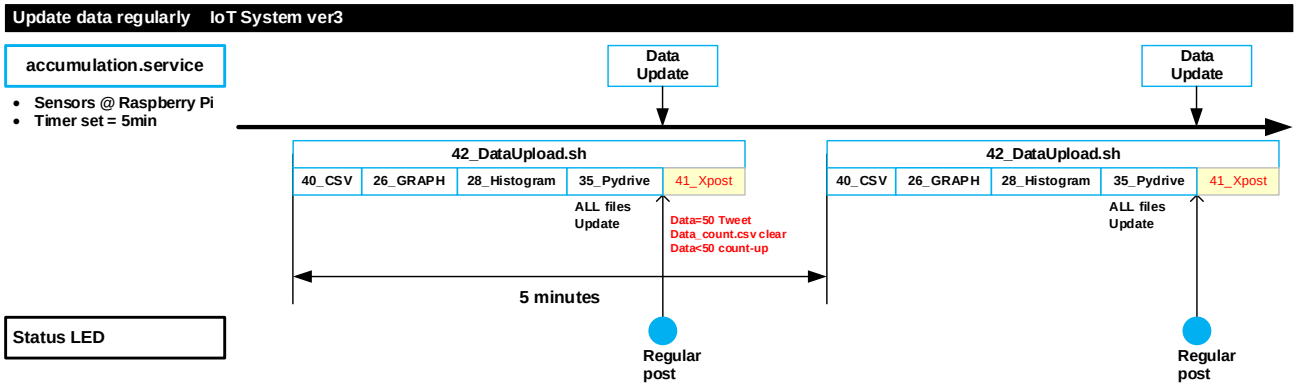
現在稼働している IoT システムは、ver1 と ver3、ver4、ver5 の 4 つのサービスが稼働しています。

- ✓ shutdown.service（再起動・シャットダウン機能）
- ✓ accumulation.service（IoT システム ver3） ※5 分ごとにサービスの起動を管理（accumulation.timer）
- ✓ Error_Detect.service（IoT システム ver4）
- ✓ Subscribe.service（IoT システム ver5）

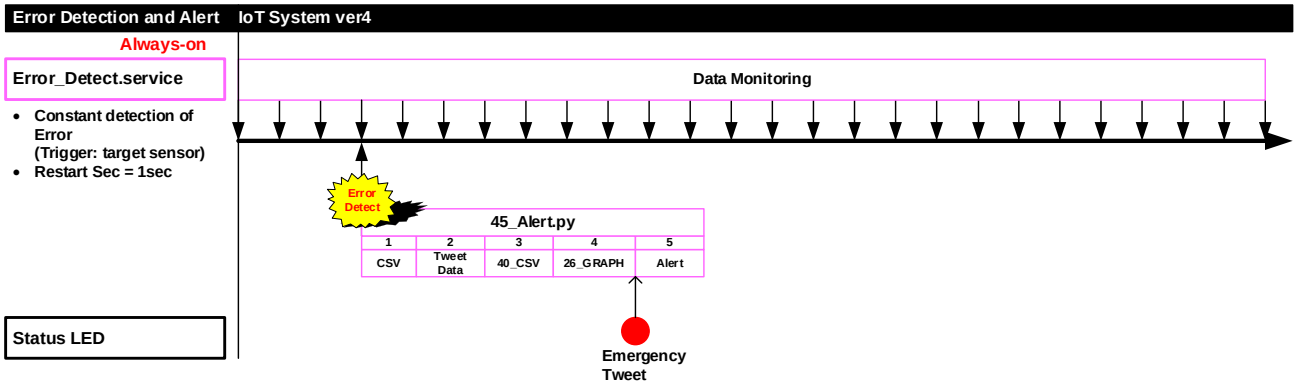
IoT システム ver1 は、30 毎にデータを取得し、取得したデータをデータベースに蓄積・可視化しています。



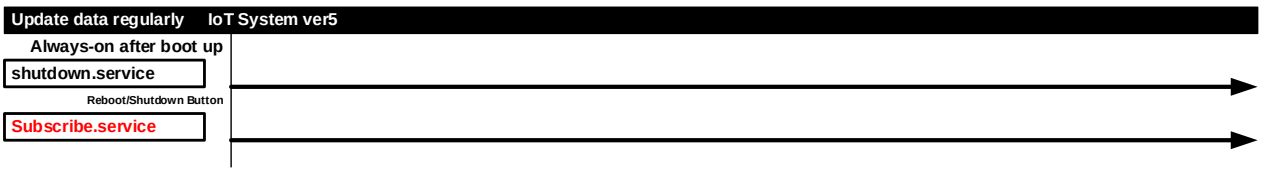
IoT システム ver2 は、5 分毎にデータを取得し、センサ情報とグラフデータを定期ポストしています。



IoT システム ver4 は、センサデータを常時監視し、異常を検出すると直ちにアラートメールを送信します。



IoT システム ver5 は、LINE ユーザーからのメッセージを受信待機し、メッセージに応じた処理を実行します。



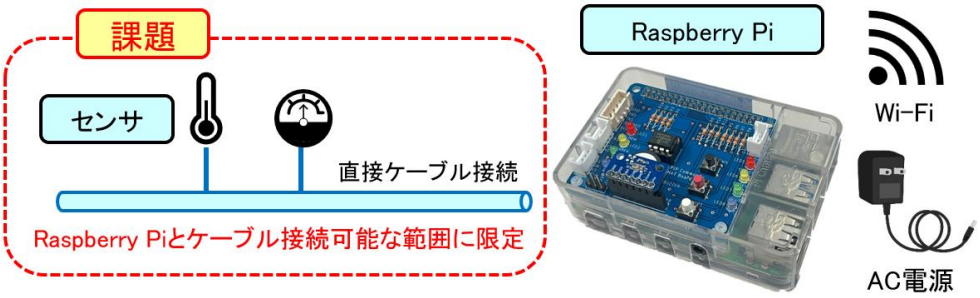
Copyright © 2024 YITJC All Rights Reserved.

第 11 章 システム拡張・機能強化

これまでの学習を通して、センサや既存の生産設備と Raspberry Pi が連携した IoT システムを構築しました。しかし、IoT システムの拡張・機能強化を実現するためには、次の課題を解決する必要があります。

- センサ設置エリアの課題

  Raspberry Pi は消費電力が大きいため、モバイルバッテリーによる稼働は難しく、AC アダプタから電源を供給する必要があります。また、クラウドサービスを利用するためには、Raspberry Pi を Wi-Fi 接続可能な場所に設置する必要があります。そのため、Raspberry Pi の設置場所が限定され、Raspberry Pi と直接ケーブルで接続するセンサの設置エリアに制限が生じます。



- センサ拡張の制限

  HAT Board は、あらかじめ実装されている BME280（温湿度・気圧センサー）を含めて、最大 4 個までしかセンサを接続できません。そのため、IoT システムを構築するうえで拡張性の低さが課題です。また、Raspberry Pi は、AC アダプタなどの周辺部品を含めると約 2 万円と高価であるため、複数の Raspberry Pi を使用して IoT システムを構築することは費用対効果の観点から難しい選択です。

これらの課題を解決するため、無線通信によるデータ送信を可能とするセンサボード（Sensor Board）を開発しました。具体的には、センサボードと Raspberry Pi の間で Wi-Fi 通信によるデータ送受信を実現します。

また、Wi-Fi が整備されていない環境や広い工場の敷地、圃場などにおける通信環境を整備するため、LPWA 通信の一種である「LoRa」通信が可能な LoRa ボードを開発しました。

ここでは、「Sensor Board」や「LoRa Board」を使用した無線通信によるセンシング技術を学習し、IoT システムの拡張・機能強化について説明します。

11.1 Sensor Board ver1 機能

Sensor Board は、ESP32 を搭載し、各種センサとの接続、Raspberry Pi やクラウドサービスへのデータ送信に対応しています。また、4 章「生産設備データ取得」で使用した Isolator Board との接続も可能です。そのため、複数の Sensor Board や Isolator Board から送信されたデータを（1 台の）Raspberry Pi に集約して IoT システムを構築できます。

～主な機能～

○センサ接続

BME280（温度、湿度、気圧）センサに加えて、Grove シリーズの I2C センサを 4 個、アナログセンサを 2 個接続できます。

Copyright © 2024 YITJC All Rights Reserved.

○無線通信

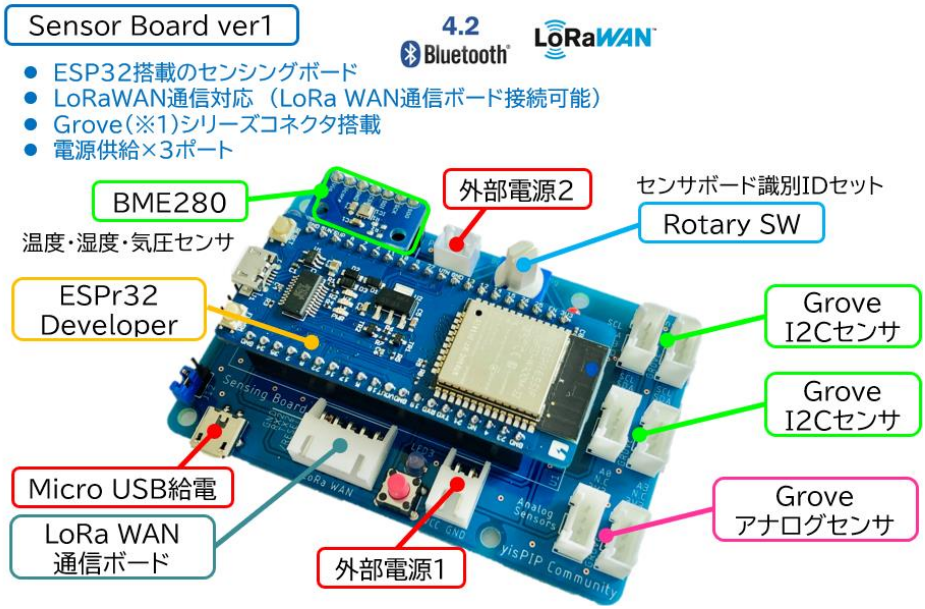
Bluetooth、Wi-Fi、LoRa（別途 LoRa 通信モジュールを接続する必要有）の各種無線通信規格に対応しています。

○電源供給

Micro USB 給電に加えて、2 ヵ所の給電端子を備えています。そのため、モバイルバッテリーや電池などを併用して給電できます。また、各給電ポートの状況は ESP32 が監視しているため、モバイルバッテリーの充電切れや電池の残量不足をリアルタイムで検出できます。

○超低消費電力

ESP32 は Deep Sleep モード時の消費電流が数マイクロ A と低く、Deep Sleep モードを使用した間欠動作を行うことで、モバイルバッテリーや電池による長期稼働が可能です。



○ケーシング

Sensor Board は Raspberry Pi と同じ基板サイズで設計しているため、市販されている Raspberry Pi のケースを利用できます。また、電源供給口などの位置も Raspberry Pi の部品配置を考慮して設計しているため、電源やセンサなどのケーブルを容易に接続できます。なお、第 4 章「生産設備データ取得」で使用した Isolator Board も Raspberry Pi と同じケースを利用できます。

Copyright © 2024 YITJC All Rights Reserved.

11.2 LoRa Board ver1 機能

LoRa Board ver1 は、Raspberry Pi と Sensor Board 間の LPWA 通信（Low Power Wide Area Network）を実現します。通信プロトコルには、LPWA 通信規格の一種であり、無線変調技術に「チャープ・スペクトラム拡散」を使用した LoRa（Long Range）通信を採用しています。

LoRa 通信には、通信事業者が管理するゲートウェイにデータ送信する「LoRaWAN」と自前でゲートウェイを設置して独自の通信ネットワークを構築する「プライベート LoRa」の 2 種類があります。前者は通信事業者との契約が必要であり、データ量に応じて通信料金が発生します。後者は独自のネットワーク網を構築するため、通信料金の発生はなく、運用コストを抑えた IoT システムを実現できます。

本講座では、Raspberry Pi をゲートウェイ、Sensor Board をエンドデバイスとした「プライベート Lora」通信網を構築します。

参考～ES920LR2（EASEL 社）～

LoRa Board は、超低消費電力 LoRa 通信に対応した ES920LR2（EASEL 社）を使用しています。

[URL] https://easel5.com/service/products-information/products/wireless-module/es920lr2/

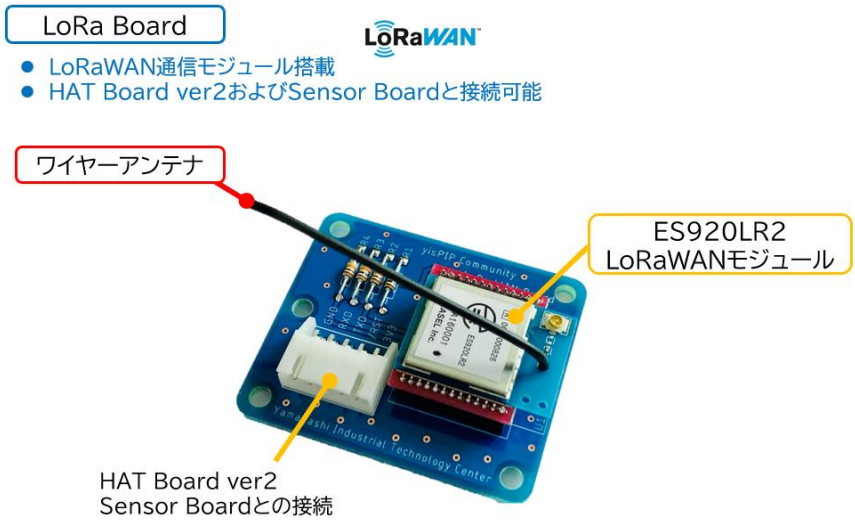
参考～ISM バンド～

LoRa 通信が使用する 920MHz 帯は、電波法で定められている ISM バンドであり、無線局免許がなくても使用できる（特定小電力無線局）電波帯域です。無線局免許が不要な「アンライセンス」系の通信は、ISM バンドと呼ばれる周波数帯を使用します。ISM は「Industry Science Medical」の略称であり、産業・科学・医療向けに免許不要で自由に使える周波数帯として国際電気通信連合（ITU）が割り当てた帯域です。周波数帯は国や地域によって異なり、日本国内では 920MHz 帯、5.7GHz 帯、2.4GHz 帯などがあります。

～主な機能～

○プライベート LoRa 通信

Raspberry Pi（HAT Board ver2）および Sensor Board ver1 の両方に接続し、通信料が発生しない「プライベート LoRa 通信」を実現します。



Copyright © 2024 YITJC All Rights Reserved.

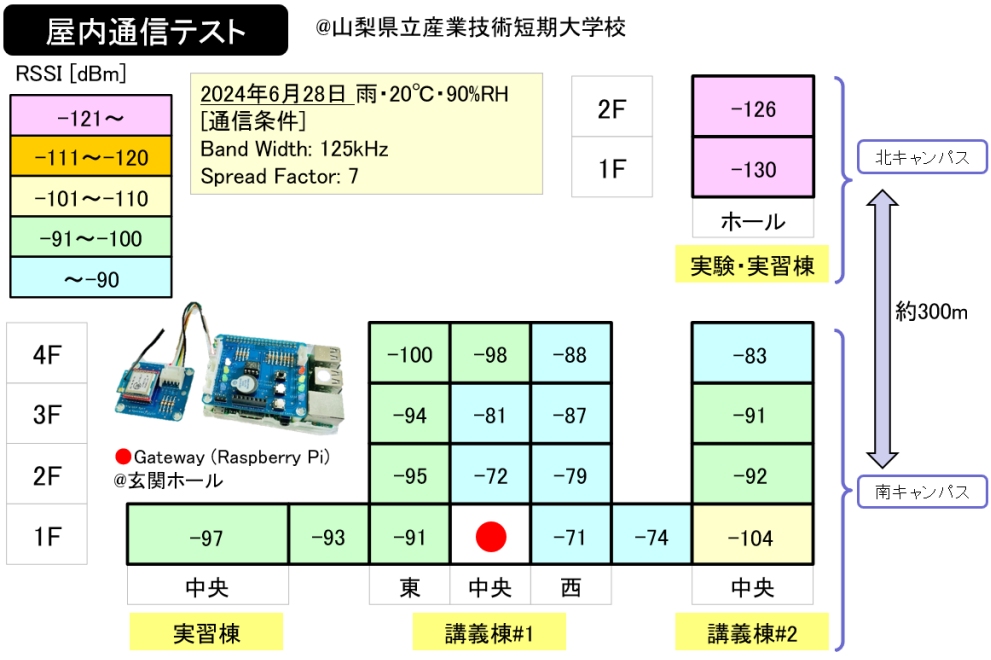
参考～LoRa 通信テスト結果～

- 屋外通信テスト

  山梨県立産業技術短期大学校・塩山キャンパスの屋上に Raspberry Pi（ゲートウェイ）を設置し、屋外環境における通信テストを実施しました。GPS を搭載した送信機（Sensor Board）から位置情報を Raspberry Pi に送信し、Raspberry Pi が受信した位置情報をオンライン地図に記録して通信可能範囲を確認しました。最大約 24km の距離で LoRa 通信が正常に行われることを確認しました。

- 屋内通信テスト

  山梨県立産業技術短期大学校・塩山キャンパスの校舎（鉄筋コンクリート構造）を使用して屋内環境における通信テストを実施しました。920MHz の周波数帯を使用した LoRa 通信は、Wi-Fi 通信（2.4GHz や 5GHz）に比べて波長が長いため、電波の回折や反射が起こりやすく、障害物を回り込んで届きやすい性質があります。そのため、Wi-Fi 通信よりも広い範囲で通信が可能です。



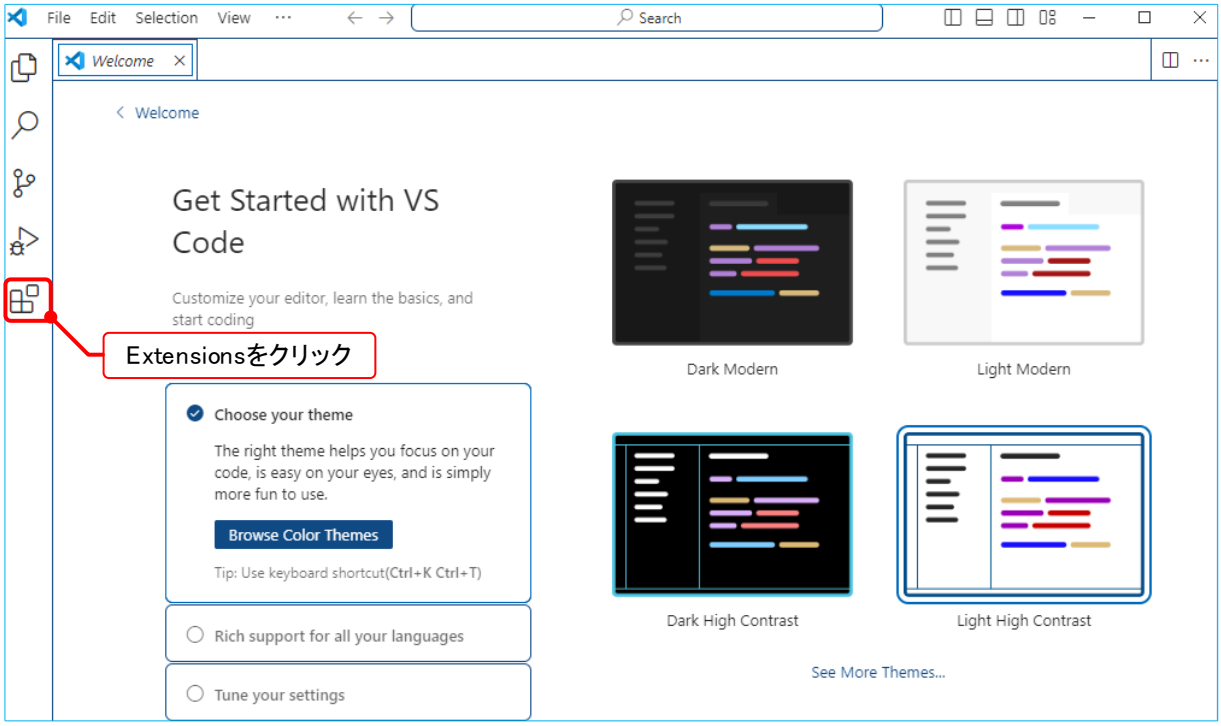
Copyright © 2024 YITJC All Rights Reserved.

11.3 ESP32 開発環境構築

Sensor Board は、Wi-Fi や Bluetooth の通信機能をもつマイコン「ESP32」を搭載しています。ここでは、エディタソフトウェア「VS Code」を使用して EPS32 の開発環境を構築する手順について説明します。

手順①～PlatformIO IDE インストール～

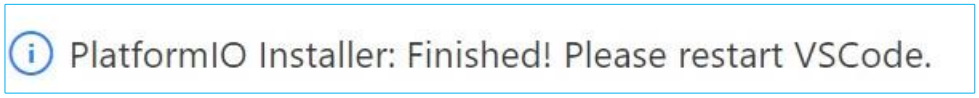
ESP32 のソースコード編集、コンパイル、プログラム書き込みを行うため、「PlatformIO IDE」拡張機能をインストールします。VS Code を起動したあと、「Extensions」をクリックしてください。



「PlatformIO IDE」を検索し、インストールしてください。

※インストール完了までに数分掛かります。

インストールが完了し、次のメッセージが表示されたら、一度 VS Code を終了して再起動してください。

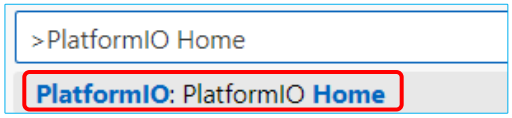


Copyright © 2024 YITJC All Rights Reserved.

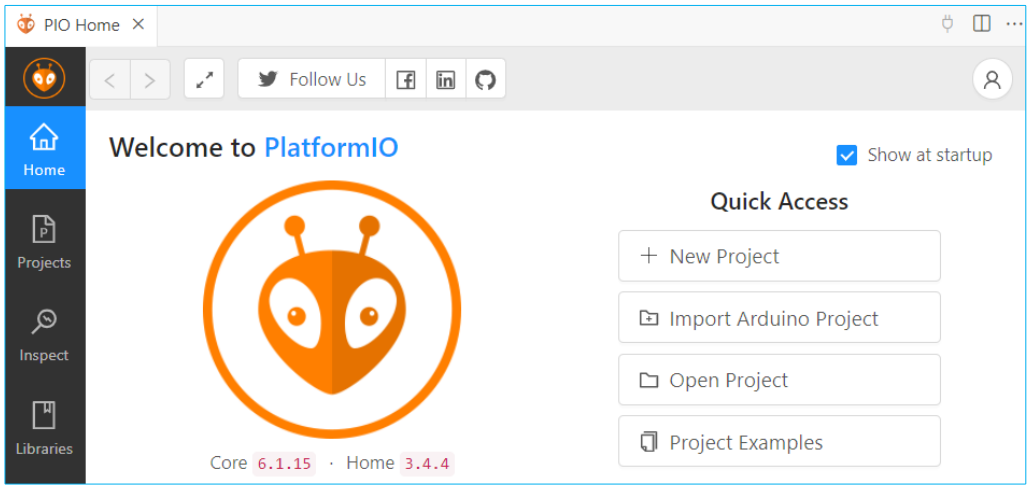
手順②～PlatformIO Home 起動～

VS Code を起動したあと、「Ctrl」＋「Shift」＋「p」を押してください。

検索バーに「PlatformIO Home」と入力し、検索結果に表示される「PlatformIO Home」をクリックしてください。



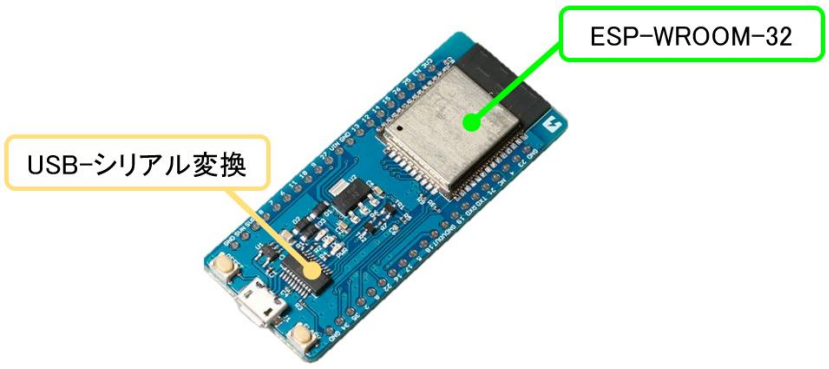
PlatformIO Home が起動すると、次の画面が表示されます。



手順③～プロジェクト作成～

プロジェクトを作成する前に、Sensor Bord に搭載している ESPr® Developer 32（Switch Science 社）について説明します。

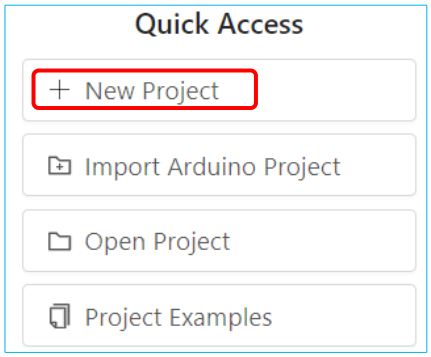
ESPr® Developer 32 は、Espressif 社の無線通信モジュール（Wi-Fi + Bluetooth）ESP-WROOM-32（通称: ESP32）や USB-シリアル変換 IC（FT231XS）などを搭載した開発ボードです。ESP-WROOM-32 を搭載したボードは数多く販売されているため、プロジェクトに適したボードを選択することが重要です。ESPr® Developer 32 を使用する場合は、プロジェクト作成時に「Espressif ESP32 Dev Module」を選択してください。



ESPr® Developer 32

Copyright © 2024 YITJC All Rights Reserved.

PlatformIO Home 画面で「New Project」をクリックしてください。

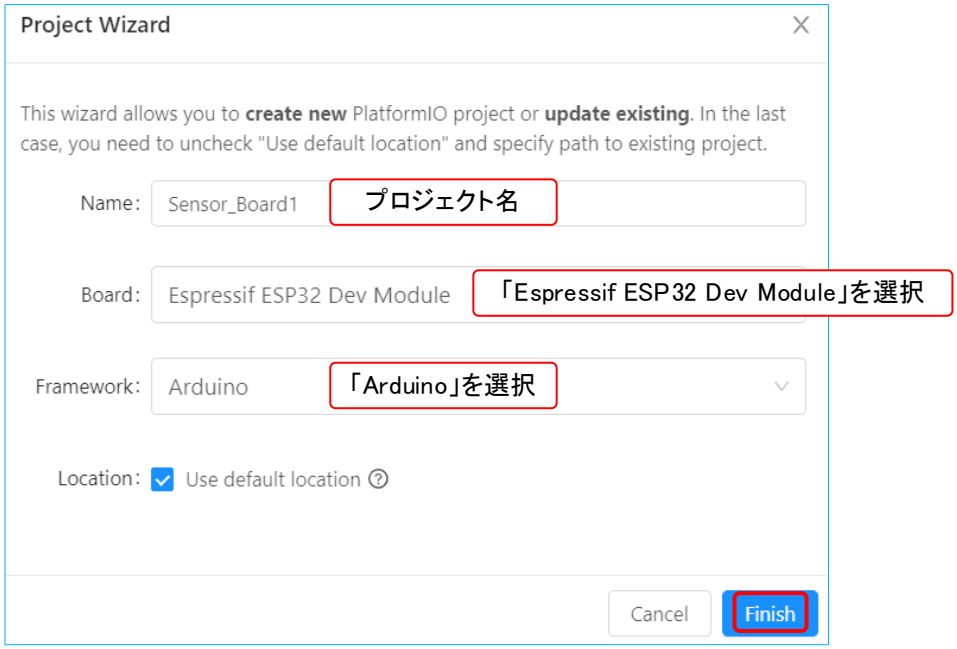


プロジェクトウィザード（Project Wizard）画面で、次の項目を設定してから「Finish」をクリックしてください。

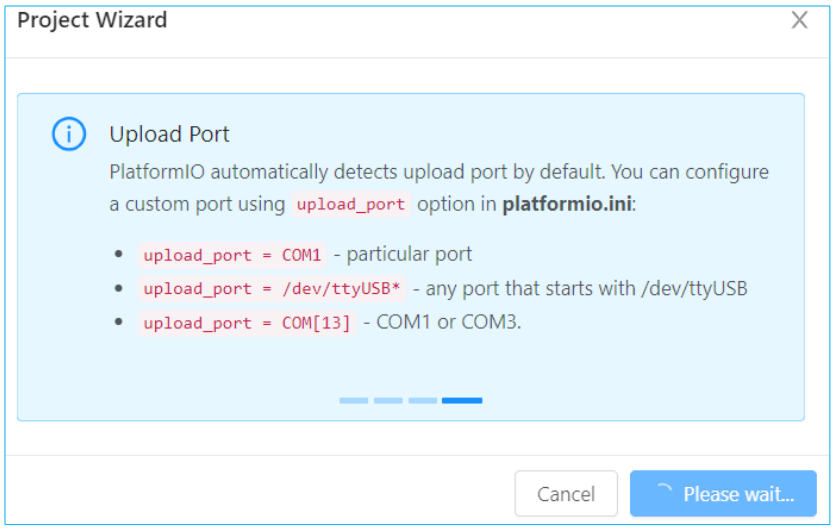
プロジェクト名　: Sensor_Board1

Board　　: Espressif ESP32 Dev Module

Framework　: Arduino



プロジェクトの作成には数分掛かります。プロジェクト作成中は、次の画面が表示されます。

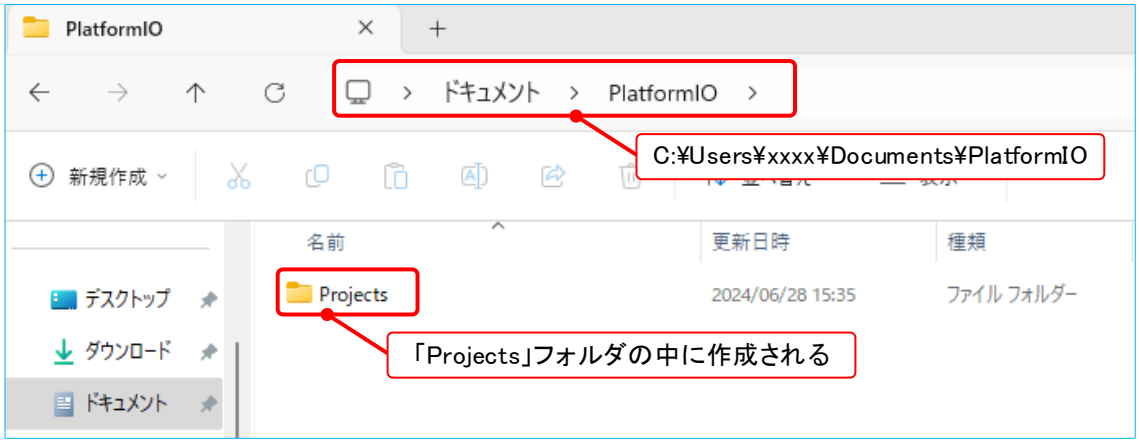


Copyright © 2024 YITJC All Rights Reserved.

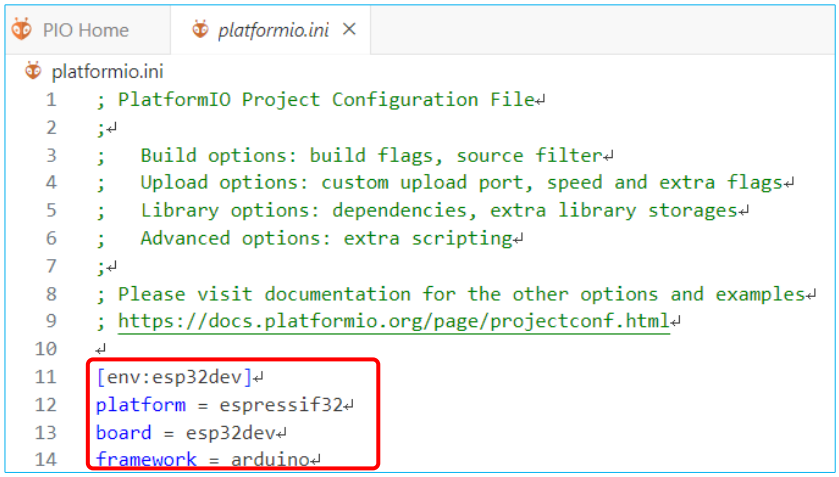
次のメッセージが表示されたら、「親フォルダの作成者を信頼します」にチェックを入れたあと、「はい、作成者を信頼します」をクリックしてください。

～補足～

プロジェクトは次のフォルダに作成されますが、「Projects」フォルダ内すべてのファイルの作成者を信頼するか確認しています。



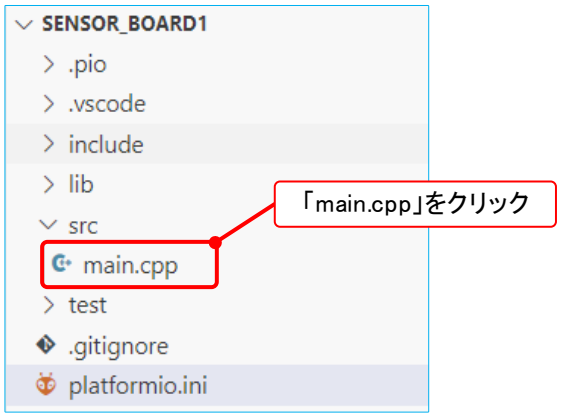
プロジェクトの作成が完了したら、「platformio.ini」ファイルが表示されます。このファイルには、プロジェクト作成時に設定した内容が記載されています。



Copyright © 2024 YITJC All Rights Reserved.

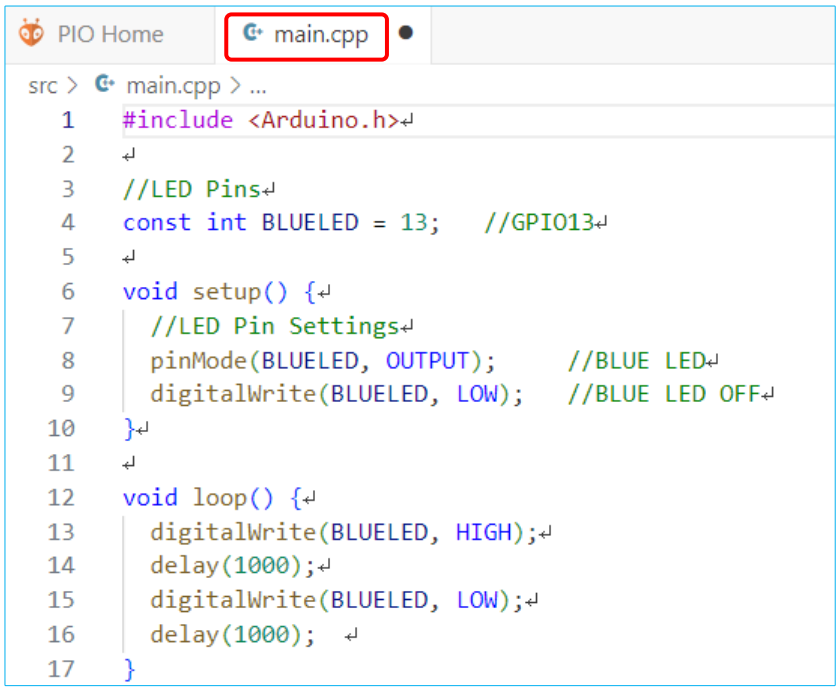
手順④～プログラム作成～

画面左のプロジェクトウィンドウから「src」-「main.cpp」をクリックしてください。



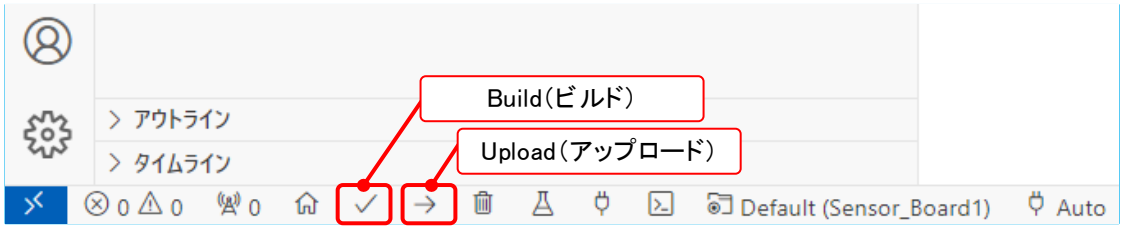
画面右のプログラムウィンドウ「main.cpp」にはプログラムの見本（書き方）が示されています。

このプログラムを編集して、次のコードを記述してください。



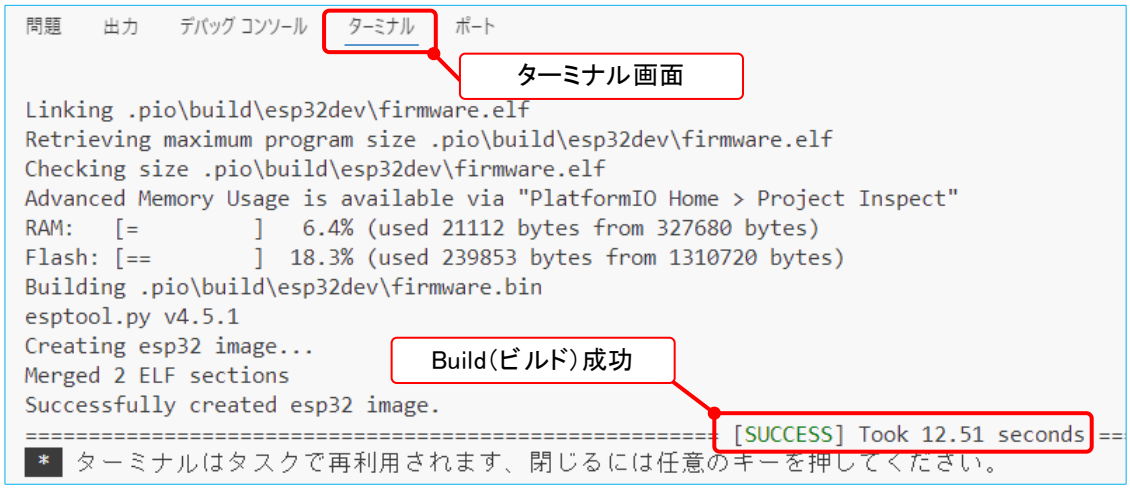
手順⑤～ビルド実行～

画面下ステータスバーに表示されているメニューから「Build（ビルド）」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

ビルドに成功すると、ターミナル画面に次のメッセージが表示されます。

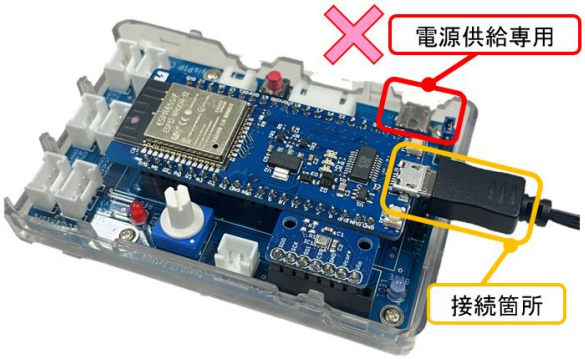


～補足～

「ビルド」は、ソースコードを検証し、問題がなければ、ESP32 が実行できる機械語に翻訳する作業です。このプロセスには、ソースコードをオブジェクトコードに変換する「コンパイル」と、オブジェクトコードを結合して実行可能なバイナリファイルを生成する「リンク」が含まれます。最終的なバイナリファイルは ESP32 に書き込まれ、実行されます。

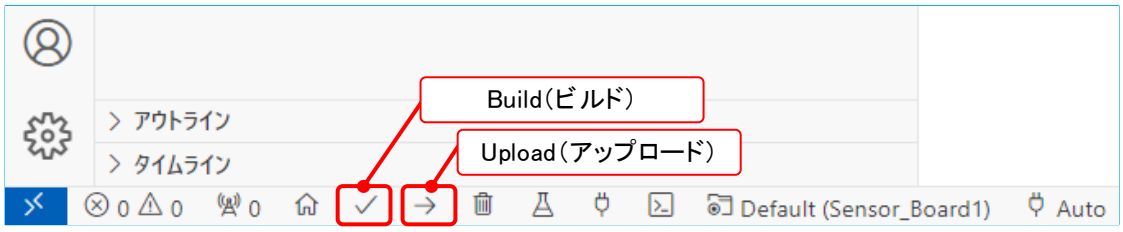
手順⑥～ESP32 接続～

Sensor Board に搭載されている ESPr® Developer 32 と Micro USB ケーブルで接続します。Sensor Board は、モバイルバッテリーなどからの電源供給を想定した「電源供給専用」の Micro USB ポートを用意しています。このポートは電源供給専用であるため、プログラムを書き込む場合は、必ず ESPr® Developer 32 に Micro USB ケーブルを接続してください。



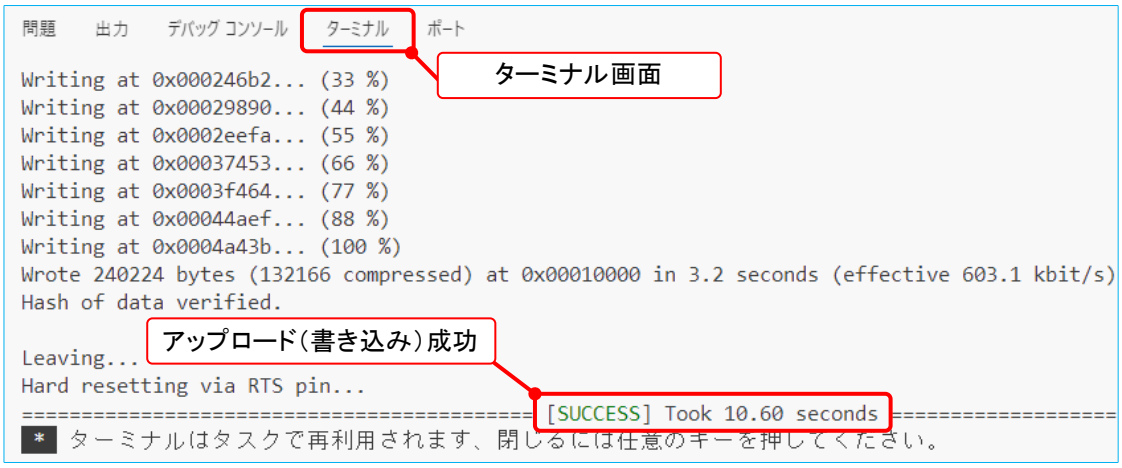
手順⑦～プログラム書き込み～

画面下ステータスバーに表示されているメニューから「Upload（アップロード）」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

プログラムの書き込み（アップロード）に成功すると、ターミナル画面に次の画面が表示されます。



手順⑧～動作確認～

プログラムの書き込みに成功したあと、Sensor Board の青色 LED が 1 秒周期で点滅していることを確認してください。続いて、ESPr® Developer 32 から Micro USB ケーブルを抜き、電源供給専用ポートに Micro USB ケーブルを挿して、同じ動作が行われることを確認してください。

～補足～

ESP32 は、内部にフラッシュメモリを搭載したマイクロコントローラです。一度プログラムを書き込むと、プログラムはフラッシュメモリに保存されます。次回以降は電源が供給されると、自動的にフラッシュメモリからプログラムが読み込まれ、実行されます。そのため、ESP32 に電源を供給するだけでプログラムは自動的に実行されます。

一方、Raspberry Pi は OS（オペレーティングシステム）を搭載したコンピュータです。プログラムを実行するためには、まず OS が起動し、その上でプログラムを実行する必要があります。Raspberry Pi は電源を供給するだけではプログラムが自動的に実行されないため、自動起動の仕組み（systemd など）を使用する必要があります。

Copyright © 2024 YITJC All Rights Reserved.

～プログラム解説～

行	プログラム
1	#include <Arduino.h>
2	
3	//LED Pins
4	const int BLUELED = 13;　　//GPIO13
5	
6	void setup() {
7	//LED Pin Settings
8	pinMode(BLUELED, OUTPUT);　　//BLUE LED
9	digitalWrite(BLUELED, LOW);　　//BLUE LED OFF
10	}
11	
12	void loop() {
13	digitalWrite(BLUELED, HIGH);
14	delay(1000);
15	digitalWrite(BLUELED, LOW);
16	delay(1000);
17	}

- 1 行目: Arduino の標準ライブラリを使用するため「Arduino.h」をインクルード
- 4 行目: 青色 LED が接続されている GPIO ピン番号を定義
- 6 行目～10 行目: setup()関数
  電源供給（またはリセット）時に一度だけ実行される処理を記載しています。
- 8 行目: 青色 LED が接続されている GPIO13 ピンを「出力」モードに設定
- 9 行目: 青色 LED が接続されている GPIO13 ピンの初期状態を「Low」に設定
- 12 行目～17 行目: loop 関数
  プログラムが動作している間、繰り返し実行される処理を記載しています。
- 13 行目: 青色 LED が接続されている GPIO13 ピンに「High」を設定し、青色 LED を点灯
- 14 行目: 1000 ミリ秒（1 秒）プログラムを停止し、青色 LED の点灯状態を維持
- 15 行目: 青色 LED が接続されている GPIO13 ピンに「Low」を設定し、青色 LED を消灯
- 16 行目: 1000 ミリ秒（1 秒）プログラムを停止し、青色 LED の消灯状態を維持

Copyright © 2024 YITJC All Rights Reserved.

参考～Sensor Board ピン割付～

ESPr® Developer 32 のピン割付と主に使用するピンの機能を紹介します。

Description	Device	Function		Pin		Function		Device	Description
		3.3Vout		1	40	GND			
		EN		2	39	GPIO23	-	GPS	GPS_RX
Rotary_SW8	Rotary SW	~~A18~~	GPIO25	3	38	GPIO04	-	GPS	GPS_TX
Rotary_SW4	Rotary SW	~~A19~~	GPIO26	4	37	N.C	-		
Rotary_SW2	Rotary SW	~~A13~~	GPIO15	5	36	I2C_SDA	-	I2C Device	SDA
Rotary_SW1	Rotary SW	~~A16~~	GPIO14	6	35	TX0	-		
Status LEDs	LED赤	~~A15~~	GPIO12	7	34	RX0	-		
Status LEDs	LED青	~~A14~~	GPIO13	8	33	GPIO19	-	タクトSW	SW
		GND		9	32	GND			
		VIN		10	31	5Vout			
USB	USB電源監視	-	GPIO27	11	30	GPIO18	-	ES920LR	/RESET
	N.A	RX1	GPIO09	12	29	GPIO05	-		
	N.A	TX1	GPIO10	13	28	~~GPIO17~~	TX2	ES920LR	TX2
	N.A	CMD	GPIO11	14	27	~~GPIO16~~	RX2	ES920LR	RX2
	N.A	CLK	GPIO06	15	26	I2C_SCL	-	I2C Device	SCL
	N.A	SD0	GPIO07	16	25	GPIO0	A11	N.A	
	N.A	SD1	GPIO08	17	24	GPIO02	A12	N.A	
A0	アナログセンサ	A0	~~GPIO36~~	18	23	~~GPIO35~~	A7	電源#2監視	VBAT2
A3	アナログセンサ	A3	~~GPIO39~~	19	22	~~GPIO34~~	A6	電源#1監視	VBAT1
		GND		20	21	GND			

～主な機能～

接続先	ピン番号	ポート	機能
ロータリースイッチ	3	GPIO25	Sensor Board の識別 ID 設定
	4	GPIO26	ロータリーを使用して 0～9 の値を設定可能
	5	GPIO15	Sensor Board を複数使用した IoT システムを構
	6	GPIO14	築する際に、Sensor Board を容易に識別可能
LED	7	GPIO12	赤色 LED
	8	GPIO13	青色 LED
電源	11	GPIO27	Micro USB 電源供給監視
	22	A6	外部電源 1 残電圧（アナログ値）監視
	23	A7	外部電源 2 残電圧（アナログ値）監視
Grove アナログセンサ	18	A0	アナログセンサデータ入力
	19	A3	アナログセンサデータ入力
Grove I2C センサ	26	SCL	I2C センサ SCL
	36	SDA	I2C センサ SCL
LoRa 通信モジュール ES920LR2	27	RX2	ES920LR2 TX 端子と接続してシリアル通信
	28	TX2	EX920LR2 RX 端子と接続してシリアル通信
	30	GPIO18	/RESET
タクト SW	33	GPIO19	タクト SW
GPS モジュール	38	GPIO4	GPS RX 端子と接続してシリアル通信
	39	GPIO23	GPS TX 端子と接続してシリアル通信

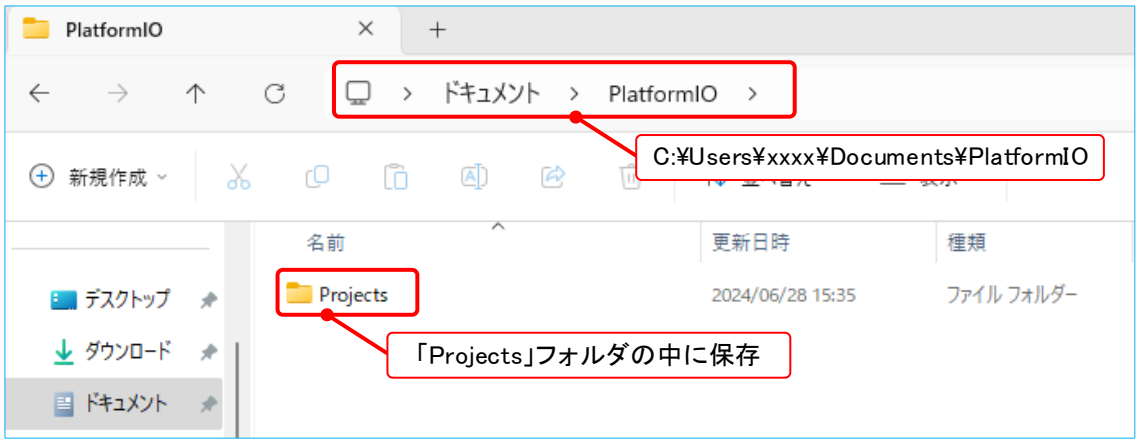
Copyright © 2024 YITJC All Rights Reserved.

11.4 BME280 センサデータ取得

Sensor Board に搭載している BME280（温湿度・気圧センサ）のデータ取得プログラムについて説明します。

手順①～プロジェクト配置～

共有サイトから対象のプロジェクトをダウンロードして、「Projects」ディレクトリに配置してください。



[対象プロジェクト]

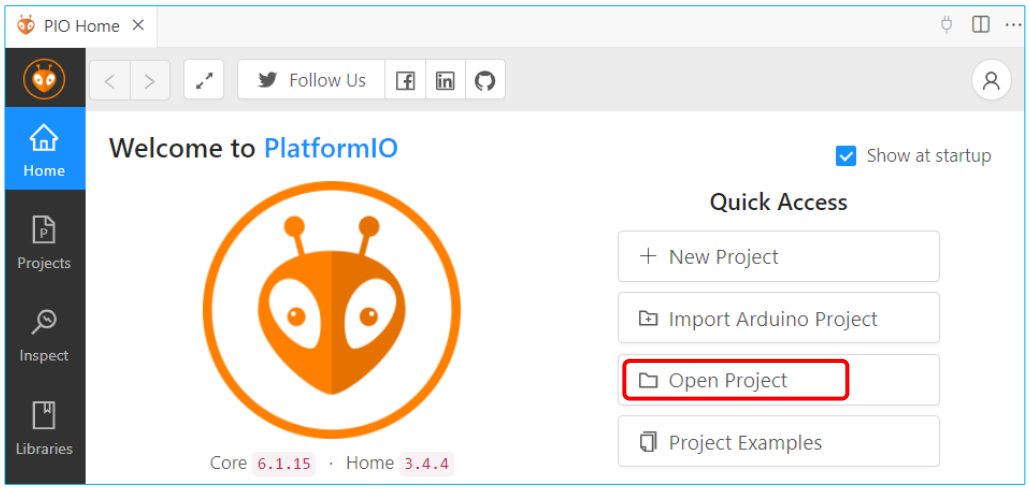
Sensor_Board2

[プログラム内容]

Sensor Board に実装されているタクトスイッチ（赤色）押下で BME280（温湿度・気圧センサ）のデータを表示します。

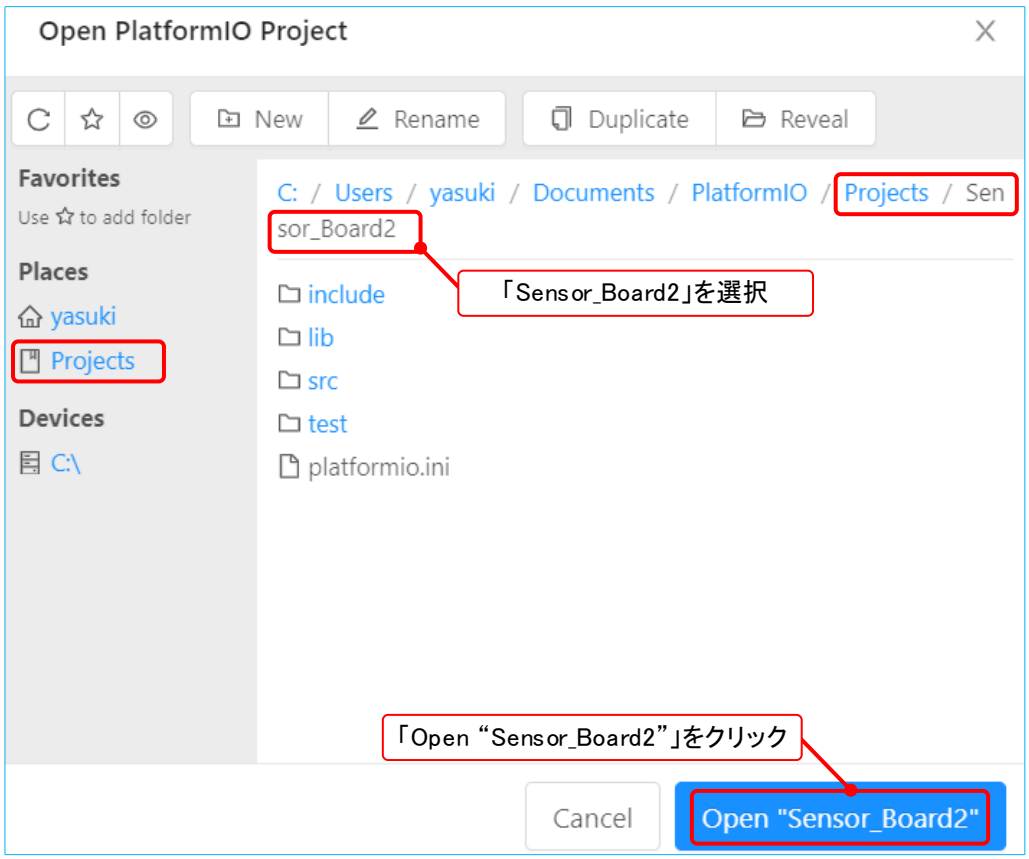
手順②～プロジェクトオープン～

PlatformIO Home 画面で「Open Project」をクリックしてください。



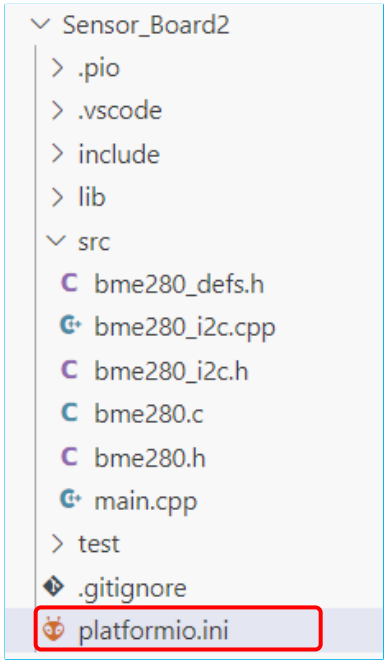
Copyright © 2024 YITJC All Rights Reserved.

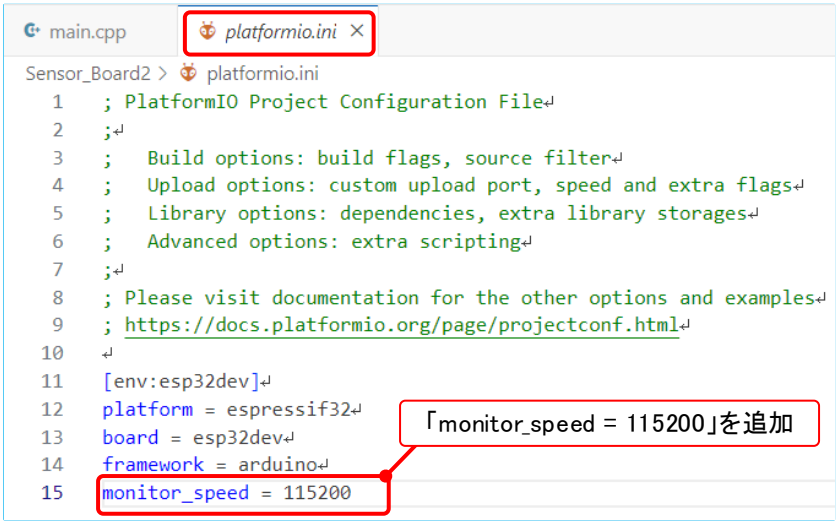
プロジェクト「Sensor_Board2」を選択してから、「Open Projects」をクリックしてください。



手順③～シリアルモニタ設定～

BME280（温湿度・気圧センサ）のデータは、Platform IO のシリアルモニタ（Serial Monitor）に表示します。シリアルモニタの通信速度を 115200bps に設定するため、「platformio.ini」ファイルに通信速度（monitor_speed）の記述を追加してください。





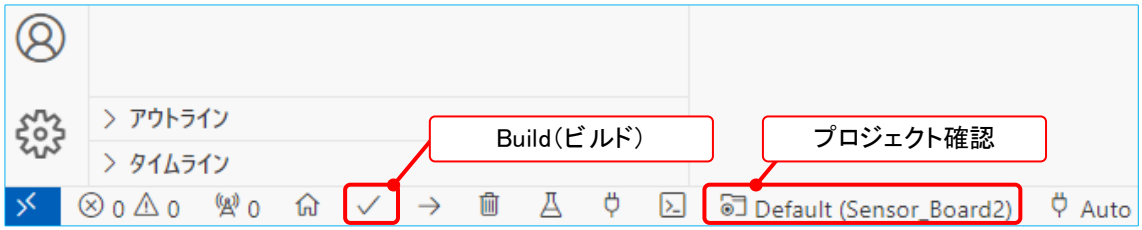
Copyright © 2024 YITJC All Rights Reserved.

手順④～ビルド実行～

画面下ステータスバーに表示されているメニューから「Build（ビルド）」をクリックしてください。

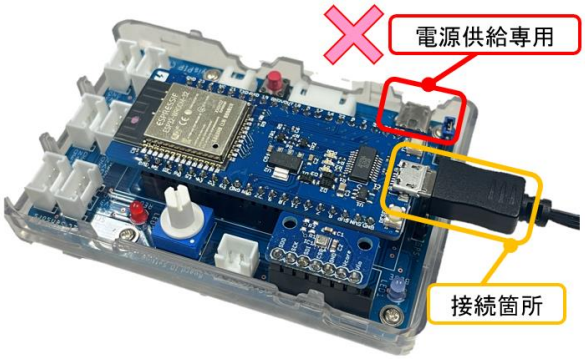
このとき、ビルドを実行するプロジェクトが「Sensor_Board2」であることを確認してください。

※プロジェクトが異なる場合は、「プロジェクト名」をクリックして正しい「プロジェクト」を選択してください。



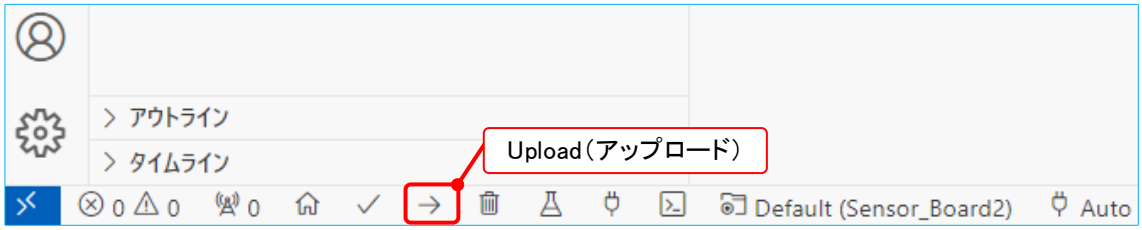
手順⑤～ESP32 接続～

Sensor Board に搭載されている ESPr® Developer 32 と Micro USB ケーブルで接続します。プログラムを書き込む場合は、必ず ESPr® Developer 32 に Micro USB ケーブルを接続してください。



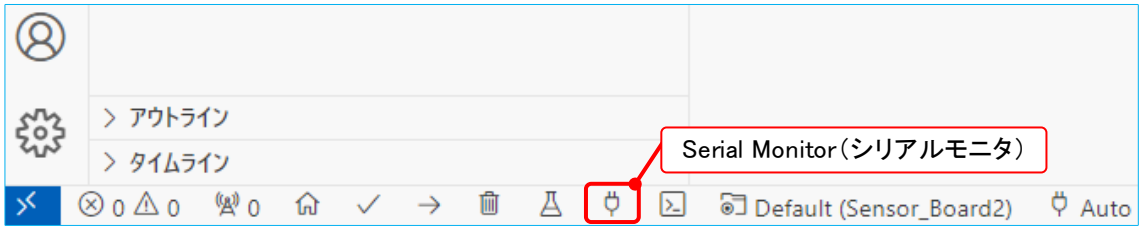
手順⑥～プログラム書き込み～

画面下ステータスバーに表示されているメニューから「Upload（アップロード）」をクリックしてください。



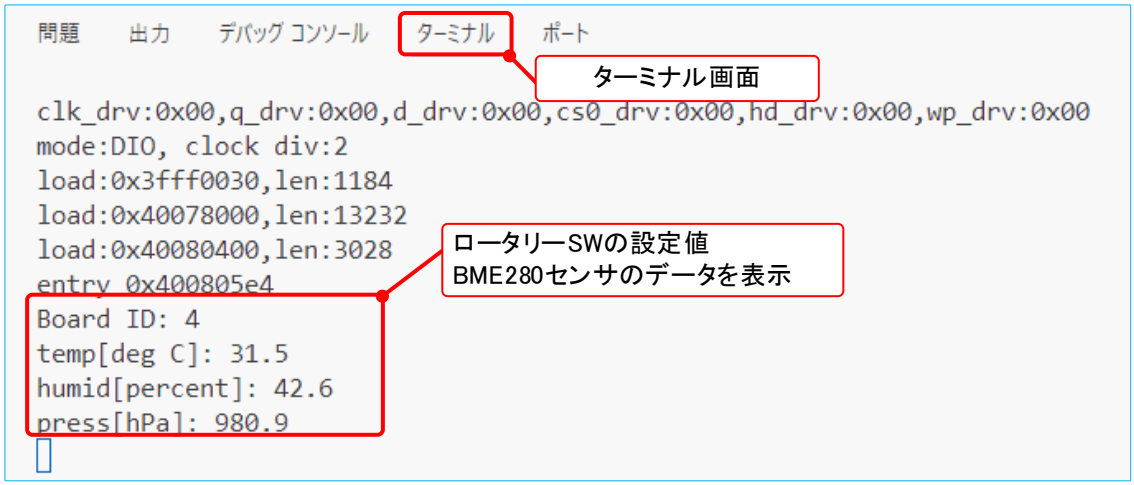
手順⑦～動作確認～

画面下ステータスバーに表示されているメニューから「Serial Monitor（シリアルモニタ）をクリックしてください。

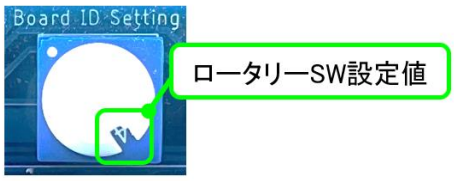


Copyright © 2024 YITJC All Rights Reserved.

続いて、Sensor Board のタクトスイッチ（赤色）を押してください。
ターミナル画面に、「ロータリーSW の設定値」と「BME280 センサのデータ」が表示されます。



ロータリーSW の値を変化させて、動作確認を実施してください。



～プログラム解説～

行	プログラム
1	#include <Arduino.h>
2	#include <Wire.h>
3	#include "bme280_i2c.h"
4	
5	#define SDA 21
6	#define SCL 22
7	
8	//SW Pins
9	const int SW = 19; //GPIO19
10	//Rotary SW Pins
11	const int RSW1 = 14; //GPIO14
12	const int RSW2 = 15; //GPIO15
13	const int RSW4 = 26; //GPIO26
14	const int RSW8 = 25; //GPIO25
15	
16	//Board ID
17	int ID;
18	

Copyright © 2024 YITJC All Rights Reserved.

```
19  BME280 bme280(BME280_I2C_ADDR_PRIM);
20  struct bme280_data data;
21  //タクトSW押下判定時間
22  static const unsigned long PUSH_SHORT = 100;
23
24  void sensor() {
25      //Board ID取得（Rotary SW）
26      ID = digitalRead(RSW8) * 8 + digitalRead(RSW4) * 4 + digitalRead(RSW2) * 2 + digitalRead(RSW1);
27      Serial.printf("Board ID: %d¥r¥n", ID);
28      //BME280データ取得
29      bme280.get_sensor_data(&data);
30      Serial.printf("temp[deg C]: %.1f¥r¥n", data.temperature);
31      Serial.printf("humid[percent]: %.1f¥r¥n", data.humidity);
32      Serial.printf("press[hPa]: %.1f¥r¥n", data.pressure / 100);
33  }
34
35  void setup() {
36      //SW Pin Settings
37      pinMode(SW, INPUT);    //SW
38      //Rotary SW Pin Settings
39      pinMode(RSW1, INPUT);    //RSW1
40      pinMode(RSW2, INPUT);    //RSW2
41      pinMode(RSW4, INPUT);    //RSW4
42      pinMode(RSW8, INPUT);    //RSW8
43      //I2C Settings
44      Wire.begin(SDA, SCL);
45      bme280.begin();
46      Serial.begin(115200);
47      delay(20);
48  }
49
50  void loop() {
51      static unsigned long lastPressTime = 0;
52      unsigned long currentTime = millis();
53      // SWが押された場合の処理
54      if (digitalRead(SW) == LOW) {
55          unsigned long pressDuration = currentTime - lastPressTime;
56          // ショートプレス判定
57          if (pressDuration > PUSH_SHORT) {
```

Copyright © 2024 YITJC All Rights Reserved.

```
58            sensor();   // センサデータ取得
59        }
60        // デバウンス処理
61        while (digitalRead(SW) == LOW) {
62            delay(10);   // SWが安定するまで待機
63        }
64        lastPressTime = currentTime;   // 最後の押下時刻を更新
65    }
66 }
```

灰色部は、これまでに説明した内容です。

- 1 行目: Arduino の標準ライブラリを使用するため「Arduino.h」をインクルード
- 2 行目: I2C 通信用のライブラリ「Wire.h」をインクルード
- 3 行目: BME280 センサ用のライブラリ「bme280_ic2.h」をインクルード
- 5 行目: I2C 通信 SCL のピン番号定義
- 6 行目: I2C 通信 SDA のピン番号定義
- 9 行目: タクトスイッチのピン番号定義
- 11 行目～14 行目: ロータリーSW のピン番号定義
- 17 行目: Sensor Board を識別するための変数「ID」を定義
- 19 行目: BME280 センサの I2C アドレスを指定してインスタンス作成
- 20 行目: BME280 センサから取得するデータを格納するための構造体変数「data」を定義
  「struct bme280_data」は、BME280 センサから取得するデータを格納するための構造体です。
- 22 行目: タクトスイッチ押下（ショートプレス）判定時間を定義
- 24 行目～33 行目: sensor()関数
  ロータリーSW から Sensor Board の識別 ID、BME280 センサのデータを取得します。
- 26 行目: ロータリーSW から Sensor Board の識別 ID を取得
- 27 行目: Sensor Board の識別 ID をシリアルモニタに print 表示
- 29 行目: BME280 センサのデータを取得
- 30 行目～32 行目: BME280 センサのデータをシリアルモニタに print 表示
- 35 行目～48 行目: setup()関数
  電源供給（またはリセット）時に一度だけ実行される処理を記載しています。
- 37 行目: タクト SW が接続されている GPIO19 ピンを「入力」モードに設定
- 39 行目: ロータリーSW が接続されている GPIO14 ピンを「入力」モードに設定
- 40 行目: ロータリーSW が接続されている GPIO15 ピンを「入力」モードに設定
- 41 行目: ロータリーSW が接続されている GPIO26 ピンを「入力」モードに設定
- 42 行目: ロータリーSW が接続されている GPIO25 ピンを「入力」モードに設定
- 44 行目: I2C 通信を初期化
- 45 行目: 作成した BME280 センサのインスタンスを初期化
- 46 行目: 通信速度（115200bps）を指定してシリアル通信を開始

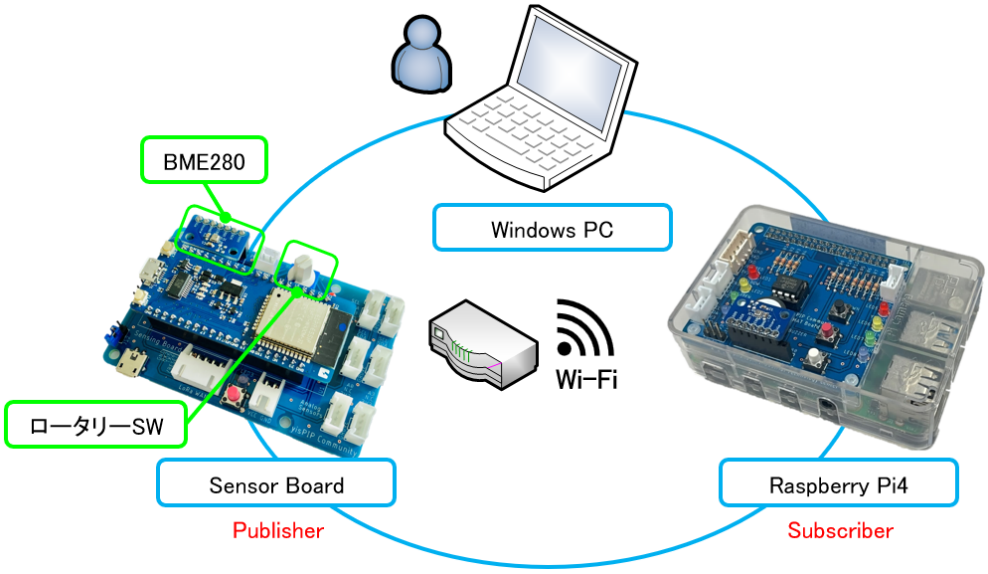
Copyright © 2024 YITJC All Rights Reserved.

- 47 行目: 20 ミリ秒プログラムを停止
  シリアル通信が安定するのを待つため、20 ミリ秒の待機時間を設定しています。
- 50 行目～66 行目: loop 関数
  プログラムが動作している間、繰り返し実行される処理を記載しています。
- 51 行目: スイッチが最後に押された時刻を格納するための変数「lastPressTime」を定義
  初期値として 0 を設定しています。
- 52 行目: プログラムが開始されてからの経過時間をミリ秒単位で取得
  取得した経過時間を変数「currentTime」に格納しています。
- 54 行目～65 行目: タクトスイッチが押された時の処理
- 55 行目: スイッチが押されている時間を格納するための変数「pressDuration」を定義
  currentTime から lastPressTime を引いた値が、スイッチが押されてから経過した時間「pressDuration」です。
- 57 行目: スイッチ押下（ショートプレス）判定
  pressDuration が PUSH_SHORT（押下判定時間）より大きい場合、sensor()関数を呼び出してセンサデータを取得します。この処理によって、一定時間以上押された場合のみ、センサデータを取得しています。
- 61 行目～63 行目: デバウンス処理（チャタリング防止処理）
  スイッチが押されている間、10 ミリ秒の遅延時間を設定してチャタリングが収まるのを待っています。
- 64 行目: 最後にスイッチが押下された時刻を更新

Copyright © 2024 YITJC All Rights Reserved.

11.5 MQTT 通信～データ送信～

MQTT（Message Queueing Telemetry Transport）通信を使用して Sensor Board から Raspberry Pi にデータを送信します。Sensor Board が Raspberry Pi と同じネットワークに接続されていれば、「Topic」を個別に指定してデータ送信できるため、センシング範囲を拡大できます。なお、Sensor Board には、同一ネットワーク内にある複数の Sensor Board を識別するため、ロータリーSW（0～9 の範囲で設定可能なスイッチ）を実装しています。



復習～MQTT 通信～

- MQTT（Message Queueing Telemetry Transport）
  - ✓ Pub/Sub メッセージングモデルにもとづく軽量プロトコルであり、オーバーヘッドが小さい（ヘッダーサイズは最小 2 バイト）
  - ✓ 非同期に 1 対 n の通信が可能（n 対 n 通信も可能）
  - ✓ 到達保証（QoS: Quality of Service）を指定可能

    AWS IoT Core は、QoS0 および QoS1 をサポートしています。

    （使用する Broker によって対応している QoS が異なるので注意してください。）

    QoS0: At most once

    -メッセージは最高 1 回配信される

    -メッセージが送信先に確実に届くかは保証されない

    QoS1: At least once

    -メッセージは最低 1 回配信される

    -メッセージが送信先に届くことが保証されるが、重複して届く可能性がある

    QoS2: Exactly once

    -メッセージは正確に 1 回配信される

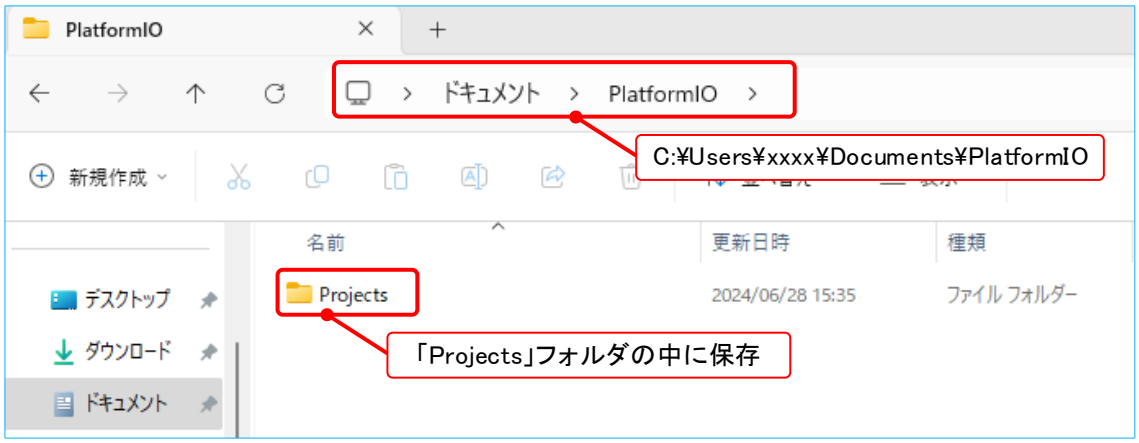
    -メッセージが重複なく送信先に届くことが保証される
  - ✓ Broker（※1）と呼ばれる仲介役（中継サーバー）が必要

    ※1: 情報を発信する Publisher は、情報のカテゴリー（種類）を表す「topic」を付けて Broker にメッセージを送信します。情報を受信する Subscriber は「topic」を指定して Broker からメッセージを受信します。

Copyright © 2024 YITJC All Rights Reserved.

手順①～プロジェクト配置～

共有サイトから対象のプロジェクトをダウンロードして、「Projects」ディレクトリに配置してください。



[対象プロジェクト]

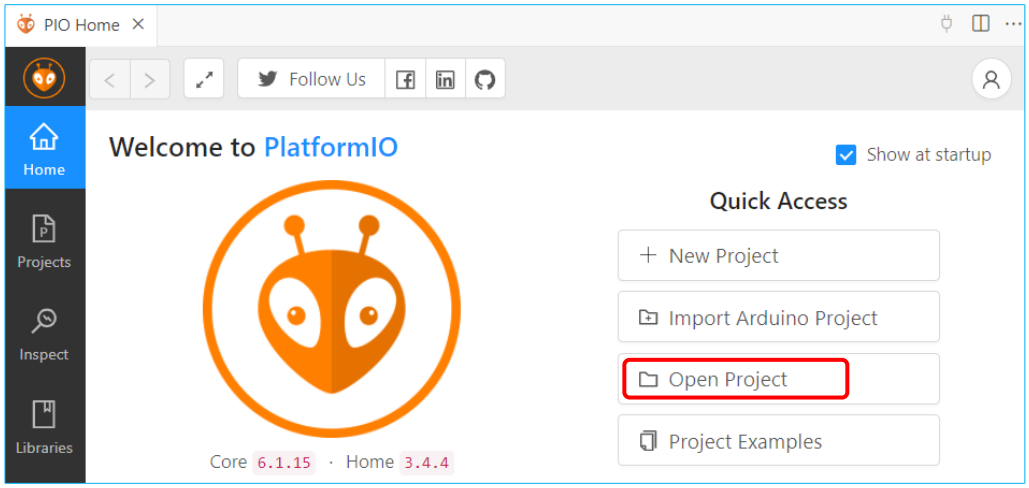
Sensor_Board3

[プログラム内容]

MQTT 通信を使用して、BME280（温湿度・気圧センサ）のデータを Raspberry Pi に送信します。

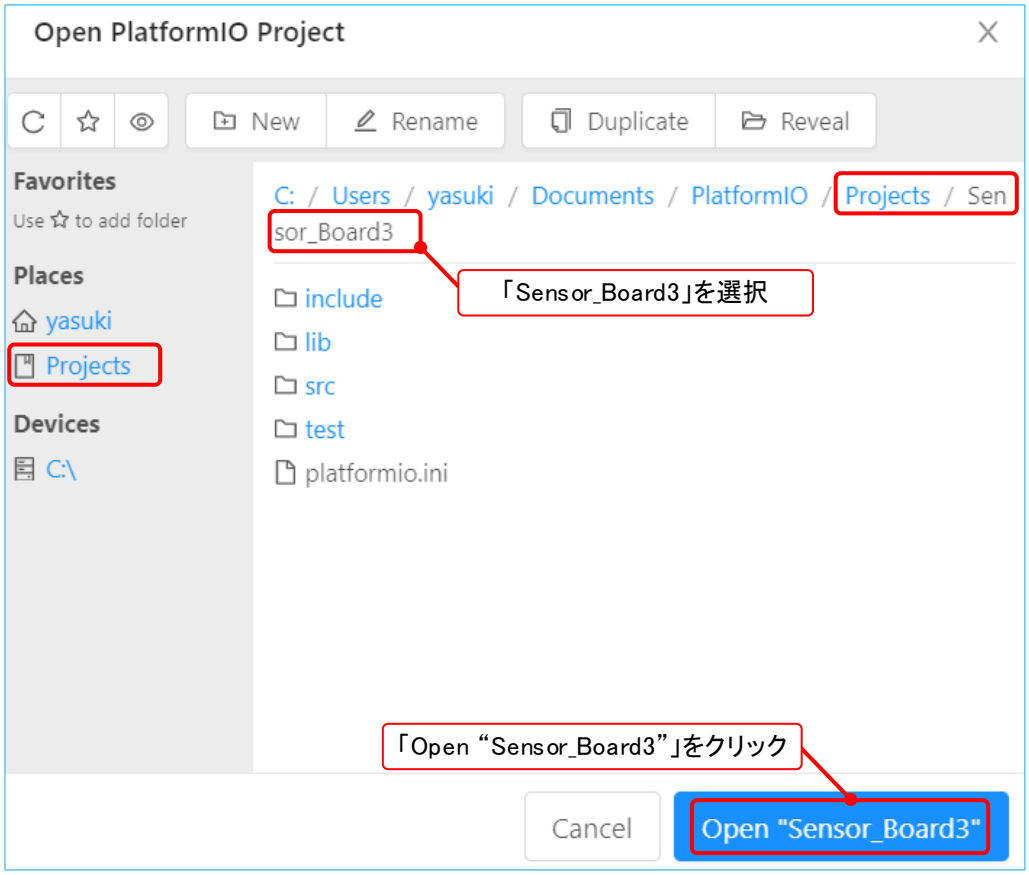
手順②～プロジェクトオープン～

PlatformIO Home 画面で「Open Project」をクリックしてください。



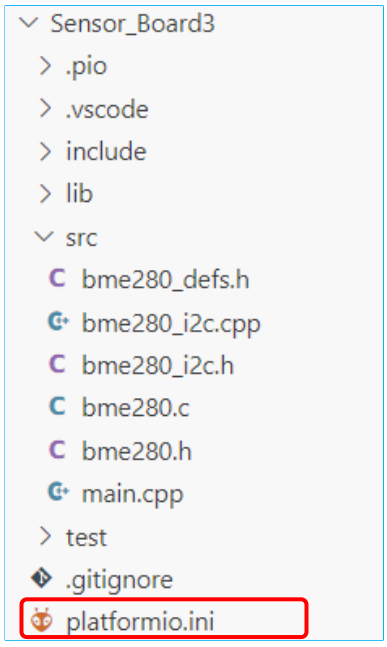
Copyright © 2024 YITJC All Rights Reserved.

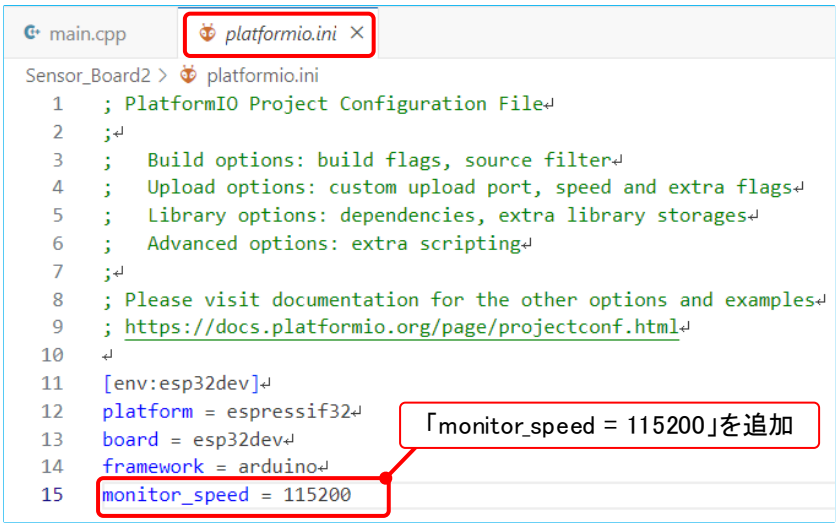
プロジェクト「Sensor_Board3」を選択してから、「Open Projects」をクリックしてください。



手順③～シリアルモニタ設定～

BME280（温湿度・気圧センサ）のデータは、Platform IO のシリアルモニタ（Serial Monitor）に表示します。シリアルモニタの通信速度を 115200bps に設定するため、「platformio.ini」ファイルに通信速度（monitor_speed）の記述を追加してください。

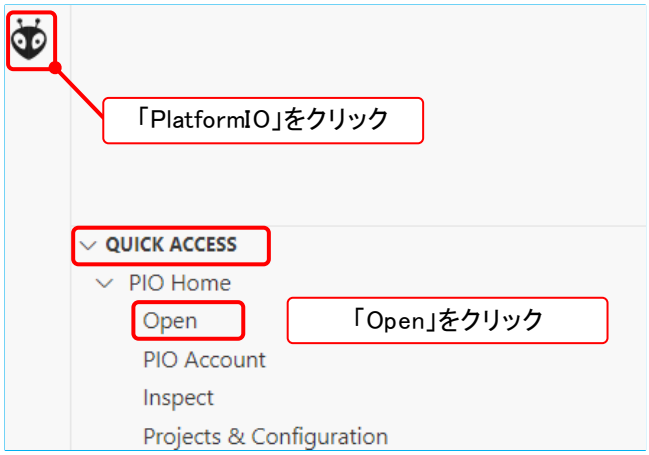




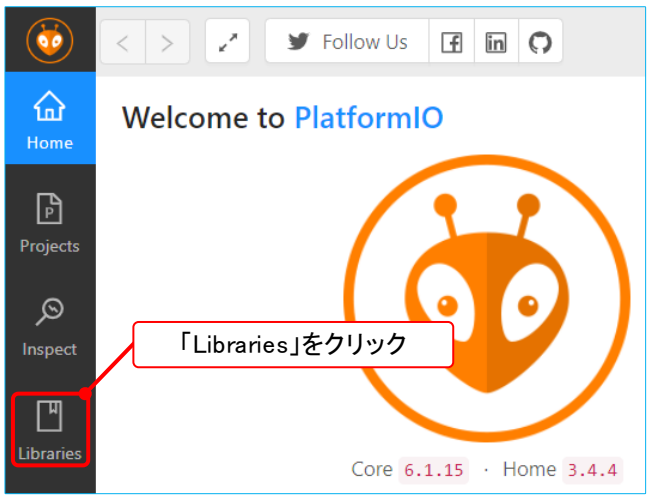
Copyright © 2024 YITJC All Rights Reserved.

手順④～JSON ライブラリ追加～

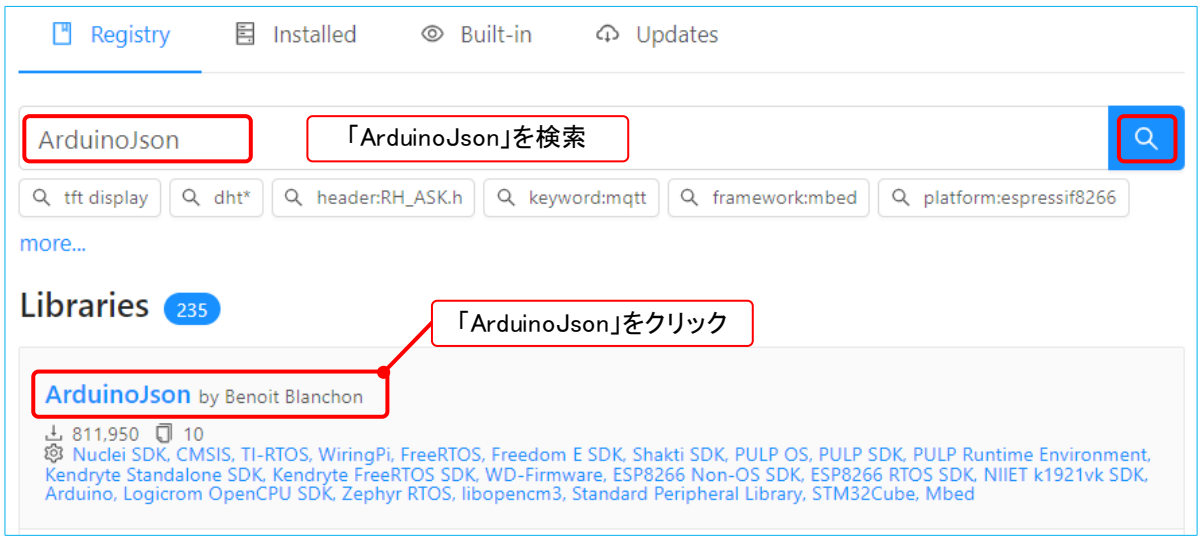
送信データを JSON 形式に変換するために必要な「ArduinoJson」ライブラリをプロジェクトに追加するため、PlatformIO Home 画面に戻ります。画面左のメニューから「PlatformIO」のアイコンをクリックしたあと、Quick Access メニューから「PIO Home」の「Open」をクリックしてください。



PlatformIO Home 画面左のメニューから、「Libraries」をクリックしてください。

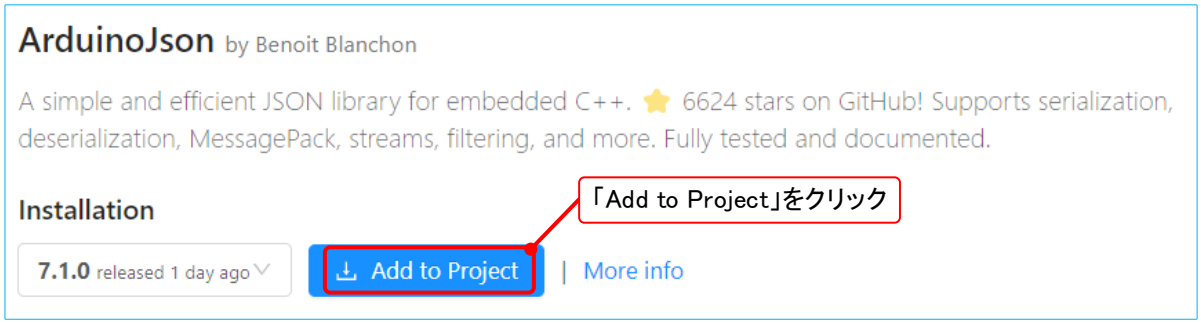


検索バーを使用して「ArduinoJson」を検索し、検索結果に表示された「ArduinoJson」をクリックしてください。

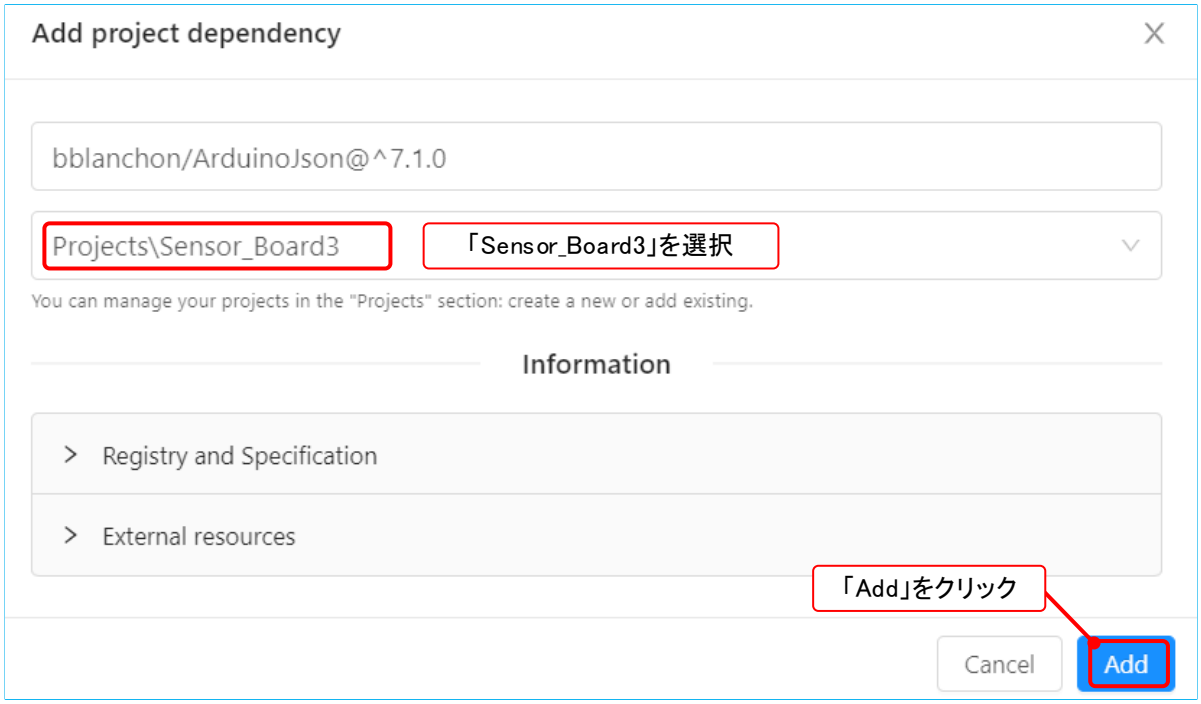


Copyright © 2024 YITJC All Rights Reserved.

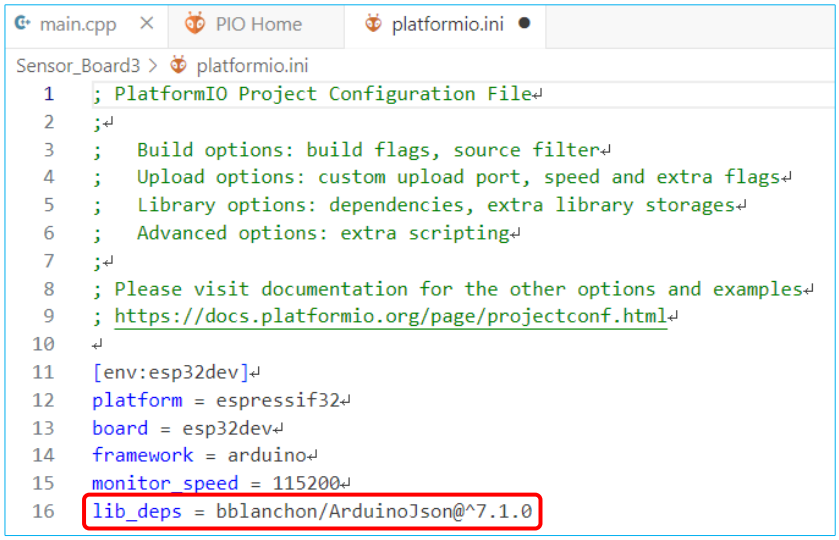
「Add to Project」をクリックしてください。



「Sensor_Board3」プロジェクトを選択してから、「Add」をクリックしてください。



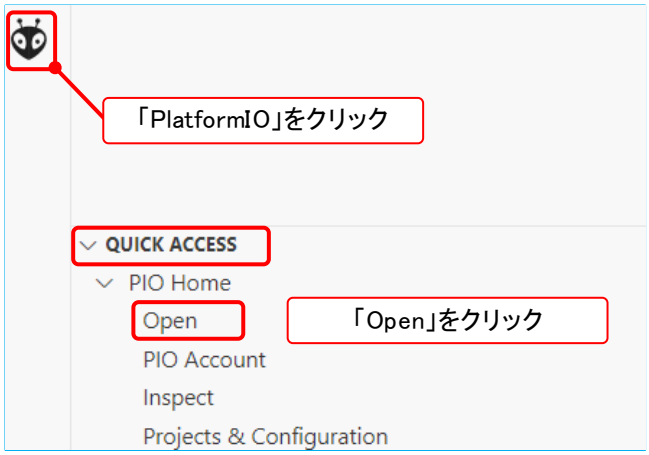
「ArduinoJson」ライブラリの追加に成功したら、「platformio.ini」ファイルに「ArduinoJson」ライブラリの記述が追加されます。



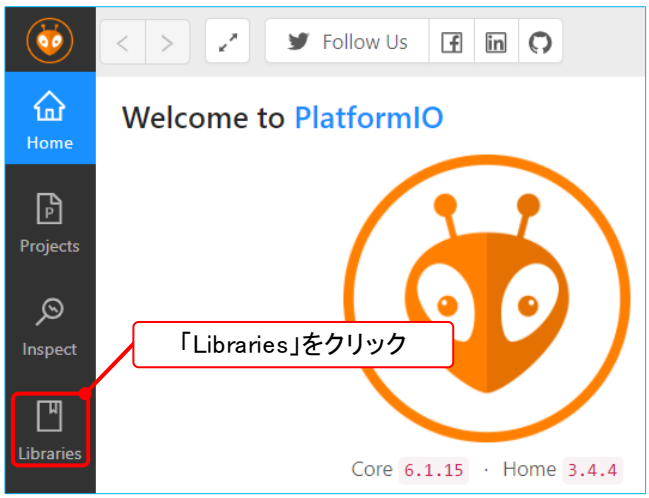
Copyright © 2024 YITJC All Rights Reserved.

手順⑤～PubSubClient ライブラリ追加～

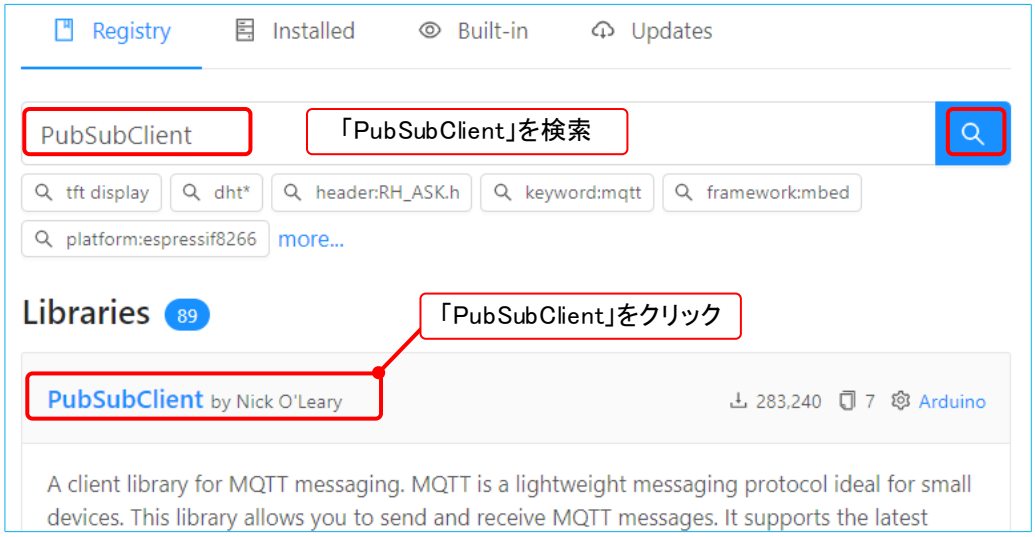
MQTT 通信を行うために必要な「PubSubClient」ライブラリをプロジェクトに追加するため、PlatformIO Home 画面に戻ります。画面左のメニューから「PlatformIO」のアイコンをクリックしたあと、Quick Access メニューから「PIO Home」の「Open」をクリックしてください。



PlatformIO Home 画面左のメニューから、「Libraries」をクリックしてください。

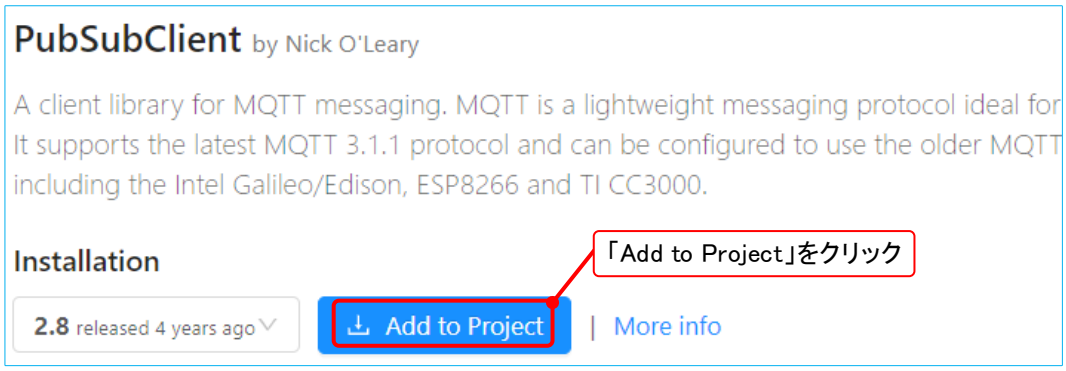


検索バーを使用して「ArduinoJson」を検索し、検索結果に表示された「ArduinoJson」をクリックしてください。

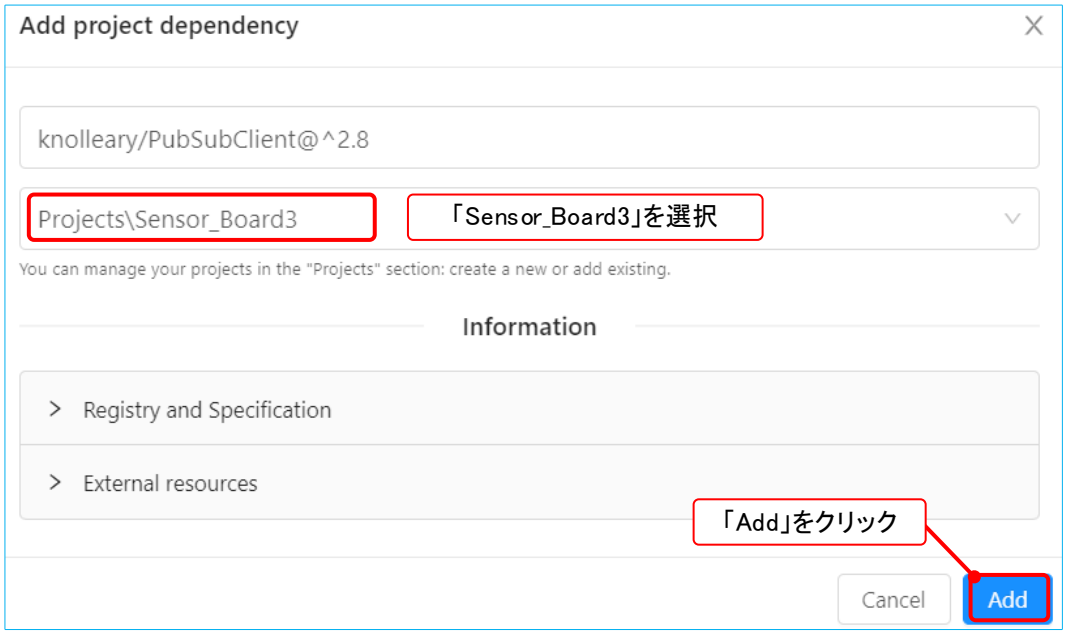


Copyright © 2024 YITJC All Rights Reserved.

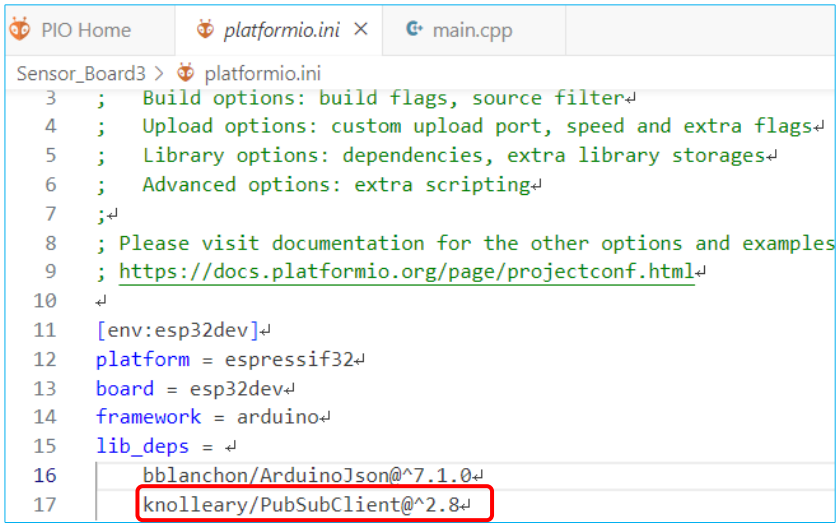
「Add to Project」をクリックしてください。



「Sensor_Board3」プロジェクトを選択してから、「Add」をクリックしてください。



「PubSubClient」ライブラリの追加に成功したら、「platformio.ini」ファイルに「PubSubClient」ライブラリの記述が追加されます。



Copyright © 2024 YITJC All Rights Reserved.

手順⑥～プログラム編集～

Sensor_Board3 プロジェクトの「main.cpp」を編集してください。

[対象プログラム]

main.cpp（Sensor_Board3 プロジェクト）

[プログラム編集個所]

- 31 行目～32 行目（Wi-Fi 接続先）

  Sensor Board と Raspberry Pi が接続する Wi-Fi の情報を記載してください。

```
//Wi-Fi Settings
const char* ssid = "xxxxxxxxxxxxx";
const char* password = "xxxxxxxxx";
```

- 35 行目（Raspberry Pi の IP アドレス）

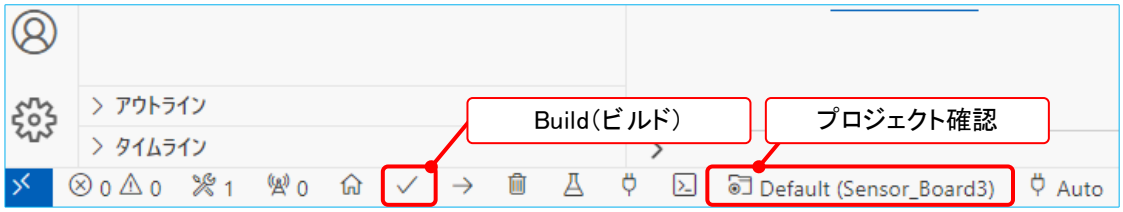
```
//MQTT setting
const char* mqttHost = "192.168.xxx.xxx";     //Raspberry Pi の IP Address
```

手順⑦～ビルド実行～

画面下ステータスバーに表示されているメニューから「Build（ビルド）」をクリックしてください。

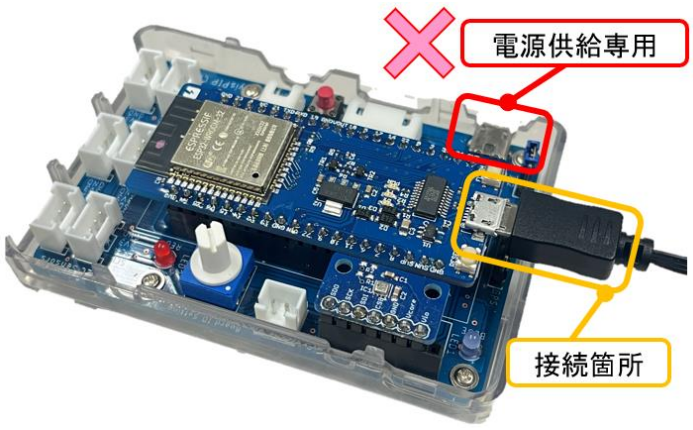
このとき、ビルドを実行するプロジェクトが「Sensor_Board3」であることを確認してください。

※プロジェクトが異なる場合は、「プロジェクト名」をクリックして正しい「プロジェクト」を選択してください。



手順⑧～ESP32 接続～

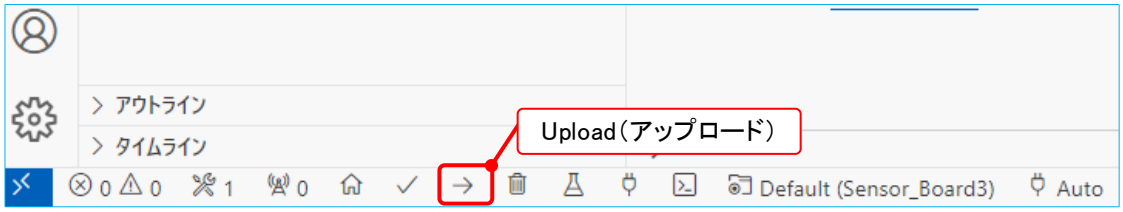
Sensor Board に搭載されている ESPr® Developer 32 と Micro USB ケーブルで接続します。プログラムを書き込む場合は、必ず ESPr® Developer 32 に Micro USB ケーブルを接続してください。



Copyright © 2024 YITJC All Rights Reserved.

手順⑨～プログラム書き込み～

画面下ステータスバーに表示されているメニューから「Upload（アップロード）」をクリックしてください。



このあと、データ受信側（Raspberry Pi）の開発を行うため、プログラムの書き込みが終了したら、Sensor Boardの電源を切ってください。

～プログラム解説～

行	プログラム
1	#include <Arduino.h>
2	#include <Wire.h>
3	#include "bme280_i2c.h"
4	#include <WiFi.h>
5	#include "PubSubClient.h"
6	#include "esp_sleep.h"
7	#include <ArduinoJson.h>
8	
9	#define SDA 21
10	#define SCL 22
11	
12	//Deep Sleep Period [sec]
13	#define S_PERIOD 30
14	
15	//LED Pins
16	const int REDLED = 12; //GPIO12
17	const int BLUELED = 13; //GPIO13
18	//Rotary SW Pins
19	const int RSW1 = 14; //GPIO14
20	const int RSW2 = 15; //GPIO15
21	const int RSW4 = 26; //GPIO26
22	const int RSW8 = 25; //GPIO25
23	
24	//Board ID
25	int ID;
26	
27	BME280 bme280(BME280_I2C_ADDR_PRIM);

Copyright © 2024 YITJC All Rights Reserved.

```
28  struct bme280_data data;
29
30  //Wi-Fi Settings
31  const char* ssid = "xxxxxxxxxxxx";
32  const char* password = "xxxxxxx";
33
34  //MQTT setting
35  const char* mqttHost = "192.168.xxx.xxx";      //Raspberry PiのIP Address
36  const int mqttPort = 1883;                     //MQTTのポート番号（Raspberry Piと同じ固定）
37  WiFiClient espclient;
38  PubSubClient mqttClient(espclient);
39  const char* topic;          //Topic
40  char* payload;              //Data
41
42  void connectMqtt() {
43    mqttClient.setServer(mqttHost, mqttPort);
44    Serial.println("Connecting to MQTT...");
45    //Board IDをMQTT IDに設定
46    int ID = digitalRead(RSW8) * 8 + digitalRead(RSW4) * 4 + digitalRead(RSW2) * 2 + digitalRead(RSW1);
47    String MQTT_ID = "Sensor" + String(ID);
48    Serial.println("MQTT ID:" + MQTT_ID);
49
50    while ( ! mqttClient.connected() ) {
51      if ( mqttClient.connect((MQTT_ID.c_str())) {
52        Serial.println("connected");
53      }
54      delay(1000);
55    }
56  }
57
58  void sendMQTT() {
59    //Board ID取得 (Rotary SW)
60    ID = digitalRead(RSW8) * 8 + digitalRead(RSW4) * 4 + digitalRead(RSW2) * 2 + digitalRead(RSW1);
61    Serial.printf("Board ID: %d¥r¥n", ID);
62    //BME280データ取得
63    bme280.get_sensor_data(&data);
64    Serial.printf("temp[deg C]: %.1f¥r¥n", data.temperature);
65    Serial.printf("humid[percent]: %.1f¥r¥n", data.humidity);
66    Serial.printf("press[hPa]: %.1f¥r¥n", data.pressure / 100);
```

Copyright © 2024 YITJC All Rights Reserved.

```
67
68      digitalWrite(BLUELED, HIGH);
69      delay(100);
70      //JSONデータの作成
71      StaticJsonDocument<256> doc;
72      doc["temperature"] = data.temperature;
73      doc["humidity"] = data.humidity;
74      doc["pressure"] = data.pressure / 100;
75      char jsonBuffer[256];
76      serializeJson(doc, jsonBuffer);
77      String topic = "Sensor" + String(ID);
78      // MQTTデータ送信  publish
79      mqttClient.publish(topic.c_str(), jsonBuffer);
80
81      digitalWrite(BLUELED, LOW);
82      //データ送信間隔
83      //delay(5000);
84      //送信間隔が長い場合は、Deep Sleepに移行
85      Serial.printf("enter deep sleep¥n");
86      delay(10);
87      esp_deep_sleep(1000000LL * S_PERIOD);                    // S_PERIOD秒Deep Sleepする
88      Serial.printf("in deep sleep¥n");
89  }
90
91  void setup() {
92      //LED Pin Settings
93      pinMode(REDLED, OUTPUT);          //RED LED
94      pinMode(BLUELED, OUTPUT);         //BLUE LED
95      digitalWrite(REDLED, LOW);       //RED LED OFF
96      digitalWrite(BLUELED, LOW);      //BLUE LED OFF
97      //Rotary SW Pin Settings
98      pinMode(RSW1, INPUT);    //RSW1
99      pinMode(RSW2, INPUT);    //RSW2
100     pinMode(RSW4, INPUT);    //RSW4
101     pinMode(RSW8, INPUT);    //RSW8
102     //I2C Settings
103     Wire.begin(SDA, SCL);
104     bme280.begin();
105     Serial.begin(115200);
```

Copyright © 2024 YITJC All Rights Reserved.

```
106     delay(20);
107     //Wi-Fi AP接続
108     WiFi.begin(ssid, password);
109     while (WiFi.status() != WL_CONNECTED) {
110         delay(100);
111     }
112     Serial.print("WiFi connected¥r¥nIP address: ");
113     Serial.println(WiFi.localIP());
114     connectMqtt();
115 }
116
117 void loop() {
118     sendMQTT();
119     delay(100);
120     mqttClient.loop();
121 }
```

灰色部は、これまでに説明した内容です。

- 1 行目: Arduino の標準ライブラリを使用するため「Arduino.h」をインクルード
- 2 行目: I2C 通信用のライブラリ「Wire.h」をインクルード
- 3 行目: BME280 センサ用のライブラリ「bme280_ic2.h」をインクルード
- 4 行目: Wi-Fi 通信用のライブラリ「WiFi.h」をインクルード
- 5 行目: MQTT 通信用のライブラリ「PubSubClient.h」をインクルード
- 6 行目: ESP32 をスリープ制御するためのライブラリ「esp_sleep.h」をインクルード
- 7 行目: MQTT 通信で送信するデータを JSON 形式に変換するためのライブラリ「ArduinoJson.h」をインクルード
- 9 行目: I2C 通信 SCL のピン番号定義
- 10 行目: I2C 通信 SDA のピン番号定義
- 13 行目: Deep Sleep モードの周期（秒）を設定
- 16 行目: 赤色 LED のピン番号を定義
- 17 行目: 青色 LED のピン番号を定義
- 19 行目～22 行目: ロータリーSW のピン番号定義
- 25 行目: Sensor Board を識別するための変数「ID」を定義
- 27 行目: BME280 センサの I2C アドレスを指定してインスタンス作成
- 28 行目: BME280 センサから取得するデータを格納するための構造体変数「data」を定義
  「struct bme280_data」は、BME280 センサから取得するデータを格納するための構造体です。
- 31 行目～32 行目: Sensor Board が接続する Wi-Fi の設定
- 31 行目: Wi-Fi 接続先 ssid を定義
- 32 行目: Wi-Fi 接続先パスワードを定義

Copyright © 2024 YITJC All Rights Reserved.

～MQTT 通信の設定～

- 35 行目～40 行目: MQTT 通信の設定
- 35 行目: 変数「mqttHost」に Raspberry Pi の IP アドレスを定義
  mqttHost には、Raspberry Pi に構築する MQTT Broker「mosquitto」の IP アドレスを指定します。
- 36 行目: MQTT 通信のポート番号を定義
  Raspberry Pi（Subscribe）側も同じポート番号を設定します。
- 37 行目: WiFiClient クラスを使用してインスタンス「espclient」を作成
- 38 行目: MQTT 通信を行うためのクラス「PubSubClient」を使用してインスタンス「mqttClient」を作成
- 39 行目: MQTT 通信用トピックを変数「topic」で定義
- 40 行目: MQTT 通信用データを変数「payload」で定義

～MQTT 通信接続～

- 43 行目～56 行目: MQTT Broker に接続するためのプログラム connectMqtt()関数
- 43 行目: MQTT Broker 設定
  Broker の IP アドレス、ポート番号を指定して MQTT Broker を設定しています。
- 44 行目: シリアルモニタに「Connecting to MQTT…」を表示
- 46 行目: ロータリーSW に設定した値を取得
- 47 行目: ロータリーSW の値を使用してクライアント ID（MQTT_ID）を定義
- 48 行目: シリアルモニタに「MQTT_ID」を表示
- 50 行目～55 行目: MQTT Broker への接続
- 51 行目: MQTT_ID を使用して MQTT Broker に接続
- 52 行目: MQTT Broker への接続に成功したら、シリアルモニタに「connected」を表示
- 53 行目: MQTT 通信にトライする間隔として、1000 ミリ秒（1 秒）を設定

注意～MQTT 通信のクライアント ID～

「クライアント ID」は、MQTT クライアント（ここでは、MQTT Broker に接続している Sensor Board）を識別する ID です。複数の Sensor Board を使用して IoT システムを構築する場合、必ず Sensor Board1 つ 1 つに固有のクライアント ID を設定しなければなりません。仮に、同じクライアント ID を使用して接続した場合、先に接続していたクライアント（Sensor Board）が切断されてしまいます。

この問題を解決するため、Sensor Board には「ロータリーSW」を実装しています。設置場所ごとにロータリーSW の値を変えることで、クライアント ID の重複を避けることができます。

しかし、ロータリーSW で設定できる値は 0～9 であるため、Sensor Board を 11 台以上設置する場合は、クライアント ID の重複に注意してプログラムを作成してください。

～MQTT データ送信～

- 58 行目～89 行目: MQTT Broker にデータ送信するためのプログラム sendMQTT()関数
- 60 行目: ロータリーSW に設定した値を取得
- 61 行目: ロータリーSW の設定値をシリアルモニタに表示
- 63 行目: BME280 センサのデータを取得
- 64 行目～66 行目: BME280 センサのデータをシリアルモニタに print 表示

Copyright © 2024 YITJC All Rights Reserved.

- 68 行目: 青色 LED が接続されている GPIO13 ピンに「High」を設定し、青色 LED を点灯
- 69 行目: 100 ミリ秒プログラムを停止し、青色 LED の点灯状態を維持
- 71 行目: Static メモリ割り当てを使用して JSON データを格納するための変数「doc」を定義
  <256>は割り当てるメモリのサイズ（バイト）を表しています。
- 72 行目: JSON オブジェクト（doc）に BME280 センサの温度データを追加
- 73 行目: JSON オブジェクト（doc）に BME280 センサの湿度データを追加
- 74 行目: JSON オブジェクト（doc）に BME280 センサの気圧データを追加
- 75 行目: JSON データを文字列に変換するためのバッファを定義
  [256]はバッファのサイズ（バイト）を表しています。
- 76 行目: JSON オブジェクト（doc）を文字列に変換し、バッファ（jsonBuffer）に格納
  これにより、Json データを MQTT メッセージとして送信できます。
- 77 行目: MQTT 通信のトピック「topic」を定義
  ロータリーSW の値を使用して Sensor Board ごとに「topic」を設定しています。
- 79 行目: mqttClient オブジェクトを使用して MQTT Broker にデータを送信（Publish）
- 81 行目: 青色 LED が接続されている GPIO13 ピンに「Low」を設定し、青色 LED を消灯
- 85 行目～88 行目: Deep Sleep モード移行処理
- 85 行目: シリアルモニタに「enter deep sleep」を表示
- 86 行目: 10 ミリ秒プログラムを停止し、シリアルモニタへの出力完了を待機
- 87 行目: Deep Sleep モードに移行
  13 行目で定義した時間「S_PERIOD」Deep Sleep モードに移行します。
  “1000000LL”は、1 秒をマイクロ秒単位で表現しています。
- 88 行目: デバッグ用
  Deep Slee モードに移行すると、88 行目のメッセージ「in deep sleep」が表示されることはありません。

参考～ESP32 の Sleep モード～

ESP32 に搭載されている Sleep モードを紹介します。Sensor Board はモバイルバッテリーや電池による電源供給に対応しているため、Sleep モードを活用して電力消費を抑えることで、長期間の自立稼働が期待できます。

Sleep モード	電力消費	機能
Modem Sleep	通常使用より低い （10～20mA 程度）	Wi-Fi および Bluetooth の接続を維持した状態で Wi-Fi および Bluetooth のモデムをスリープ状態に移行
Light Sleep	Modem Sleep より低い （1mA 以下）	Modem Sleep より電力消費が小さい Deep Sleep より復帰時間が短い
Deep Sleep	非常に低い （0.1mA 以下）	内部 RTC（Real Time Clock）タイマーによる復帰が可能 Sleep 復帰時は再起動が必要（setup()関数から実行）
Hibernation （冬眠モード）	最も低い （0.01mA 以下）	電力消費を最大限に抑えることが可能 ただし、Sleep 復帰時はリセットによる再起動が必要

Copyright © 2024 YITJC All Rights Reserved.

～Sensor Board セットアップ～

- 91 行目～115 行目: setup()関数
  電源供給（またはリセット）時に一度だけ実行される処理を記載しています。
- 93 行目: 赤色 LED が接続されている GPIO12 ピンを「出力」モードに設定
- 94 行目: 青色 LED が接続されている GPIO13 ピンを「出力」モードに設定
- 95 行目: 赤色 LED が接続されている GPIO12 ピンの初期状態を「Low」に設定
- 96 行目: 青色 LED が接続されている GPIO13 ピンの初期状態を「Low」に設定
- 98 行目: ロータリーSW が接続されている GPIO14 ピンを「入力」モードに設定
- 99 行目: ロータリーSW が接続されている GPIO15 ピンを「入力」モードに設定
- 100 行目: ロータリーSW が接続されている GPIO26 ピンを「入力」モードに設定
- 101 行目: ロータリーSW が接続されている GPIO25 ピンを「入力」モードに設定
- 102 行目: I2C 通信を初期化
- 103 行目: 作成した BME280 センサのインスタンスを初期化
- 104 行目: 通信速度（115200bps）を指定してシリアル通信を開始
- 105 行目: 20 ミリ秒プログラムを停止
  シリアル通信が安定するのを待つため、20 ミリ秒の待機時間を設定しています。
- 108 行目: Wi-Fi アクセスポイント（AP）に接続
  31 行目～32 行目で定義した ssid と password を使用して接続します。
- 109 行目～111 行目: 通信速度（115200bps）を指定してシリアル通信を開始
- 112 行目: 100 ミリ秒（0.1 秒）ごとに Wi-Fi 接続状態を確認
  Wi-Fi への接続が確立するまで接続状態を確認します。
- 113 行目: Wi-Fi への接続確立後、シリアルモニタに「Wi-Fi connected」を表示
- 114 行目: Wi-Fi への接続確立後、シリアルモニタに「IP アドレス」を表示
- 115 行目: MQTT Broker への接続開始

～繰り返し処理～

- 117 行目～121 行目: loop 関数
  プログラムが動作している間、繰り返し実行される処理を記載しています。
- 118 行目: MQTT Broker にデータ送信するためのプログラム sendMQTT()関数実行
- 119 行目: 100 ミリ秒（0.1 秒）プログラムを停止し、MQTT メッセージ送信完了を待機
- 120 行目: MQTT 通信の接続状態を維持

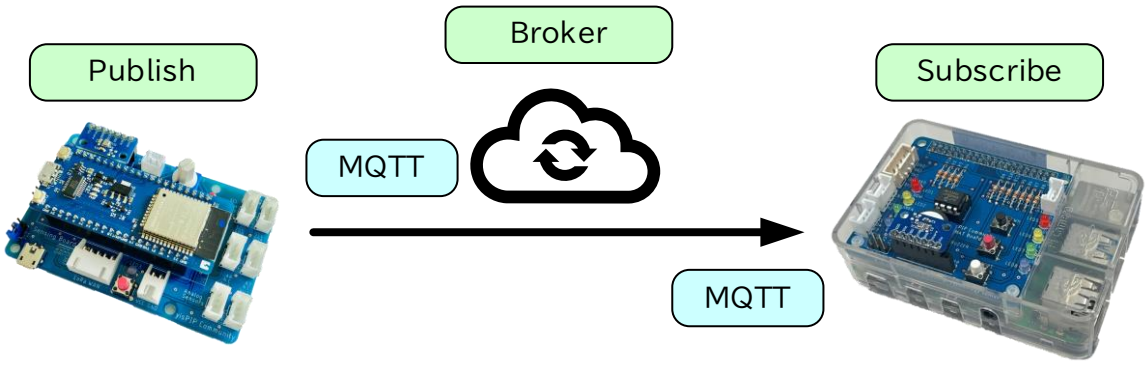
Copyright © 2024 YITJC All Rights Reserved.

11.6 MQTT 通信～データ受信～

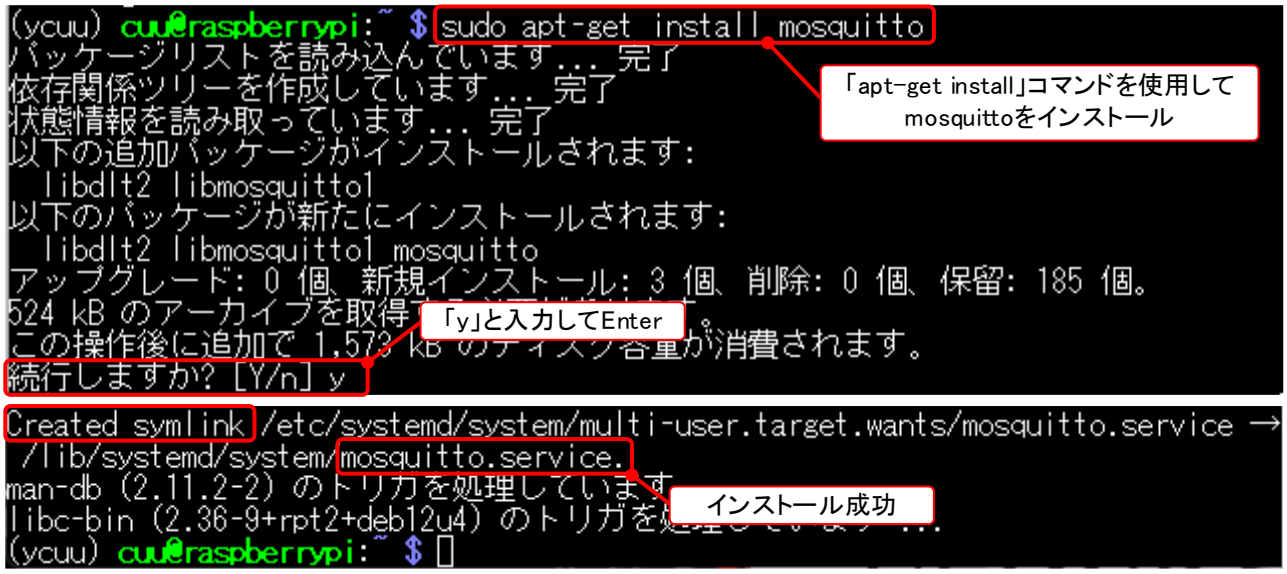
MQTT 通信を使用して Sensor Board から送信された BME280（温湿度・気圧センサ）のデータを Raspberry Pi が受信します。Raspberry Pi は、Sensor Board に対して個別に用意された「topic」を指定してデータを受信します。ここでは、Python プログラムを使用して Sensor Board から送信されたデータを受信する方法について説明します。

手順①～mosquitto インストール～

Sensor Board が Publisher（メッセージ送信）、Raspberry Pi が Subscriber（メッセージ受信）となって MQTT 通信を行います。ここでは、MQTT 通信の仲介役となる Broker として「mosquitto」をインストールします。



(ycuu) cuu@raspberrypi:~ $ sudo apt-get install mosquitto

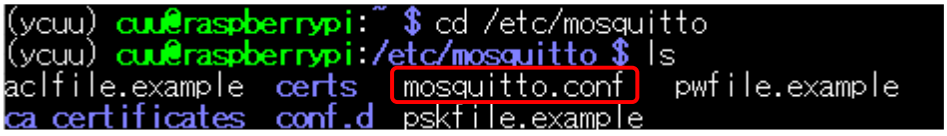


手順②～Broker ファイル編集～

Broker ファイル（mosquitto.conf）を編集して通信ポートやログイン認証の設定を記述します。

(ycuu) cuu@raspberrypi:~ $ cd /etc/mosquitto

(ycuu) cuu@raspberrypi:~/etc/mosquitto $ ls

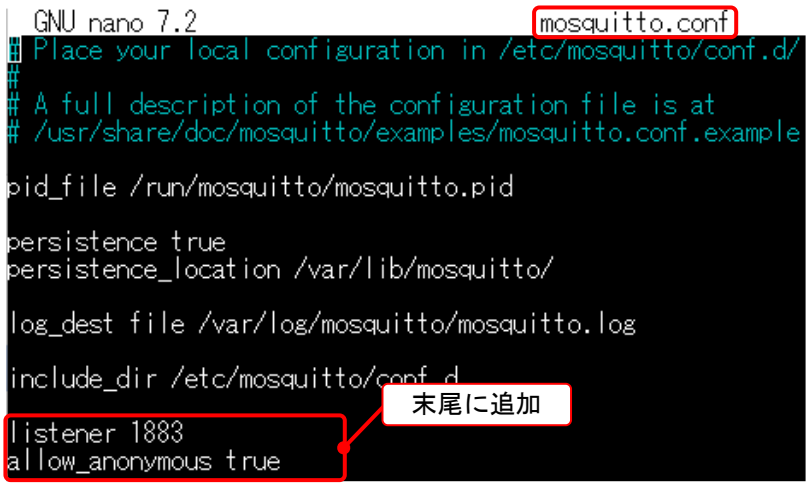


Copyright © 2024 YITJC All Rights Reserved.

(ycuu) cuu@raspberrypi:~/etc/mosquitto $ sudo nano mosquitto.conf

末尾に次の 2 行を追加してください。

```
listener 1883                #MQTT 通信ポート番号
allow_anonymous true         #ログイン認証不要
```



記述した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

手順③～サービス再読み込み～

手順②で「mosquitto.conf」ファイルを編集した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/mosquitto $ sudo systemctl daemon-reload

手順④～サービス再起動～

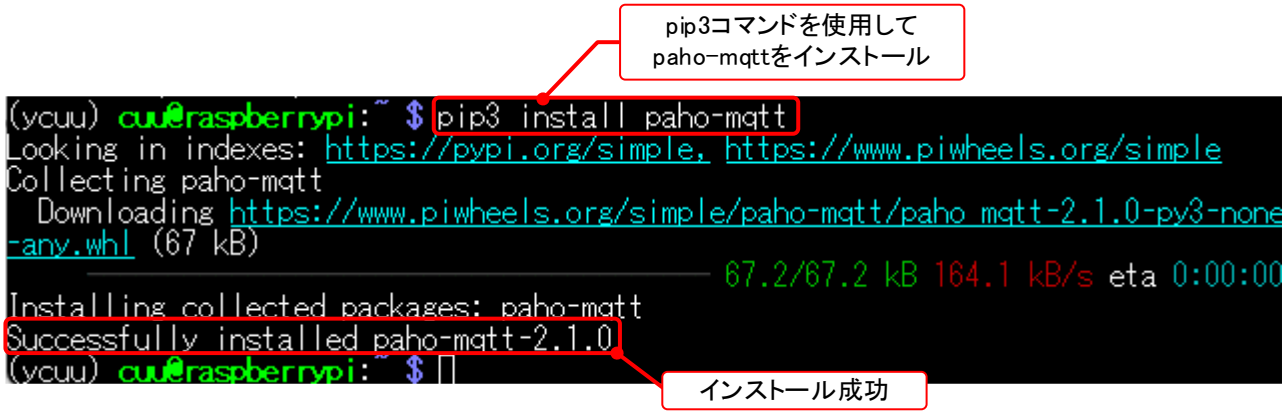
次のコマンドを実行して「mosquitto」サービスを再起動します。

(ycuu) cuu@raspberrypi:/etc/mosquitto $ sudo systemctl restart mosquitto.service

手順⑤～paho インストール～

MQTT 通信を行うために必要なライブラリ「paho」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install paho-mqtt



Copyright © 2024 YITJC All Rights Reserved.

手順⑥～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。(ドラッグ＆ドロップ)

[対象プログラム]

51_MQTT.py

手順⑦～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 51_MQTT.py

手順⑧～プログラム実行～

対象プログラムを実行します。

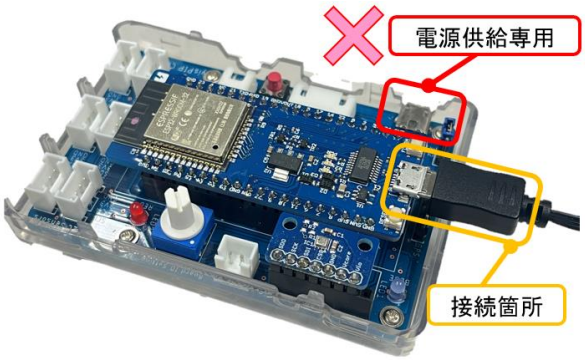
(ycuu) cuu@raspberrypi:~/dev $ ./51_MQTT.py

MQTT Broker への接続に成功すると、次のメッセージ「Status Success」が表示されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./51_MQTT.py
status Success
```

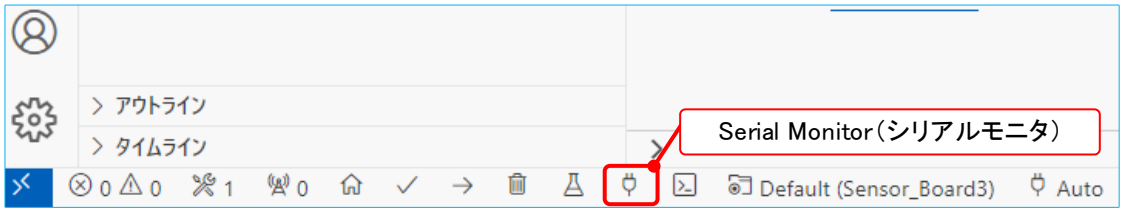
手順⑨～ESP32 接続～

VS Code シリアルモニタを使用して Sensor Board の動作を確認するため、ESPr® Developer 32 に Micro USB ケーブルを接続してください。



手順⑩～動作確認～

VS Code を開いて、画面下ステータスバーに表示されているメニューから「Serial Monitor（シリアルモニタ）をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

- Sensor Board（子機）の動作

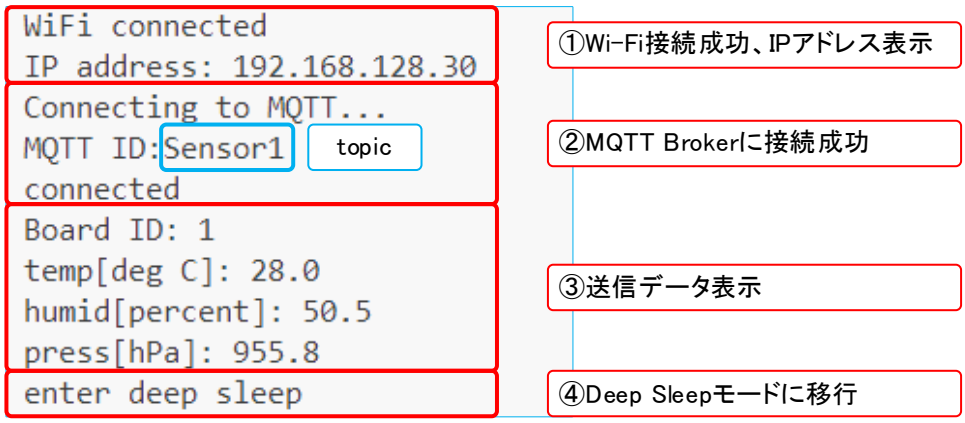
①～④の動作を 30 秒周期で繰り返し処理しています。

① Wi-Fi への接続に成功すると、IP アドレスが表示されます。

② ロータリーSW で設定した識別 ID を使用して「topic」を作成し、MQTT Broker に接続します。

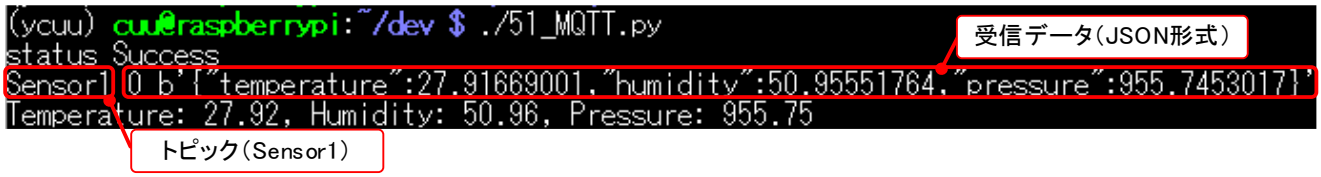
③ Raspberry Pi に送信するデータ print 表示。JSON 形式に変換したあとデータ送信します。

④ データ送信後、Deel Sleep モードに移行します。



- Raspberry Pi（親機）の動作

Tera Term の画面を確認すると、トピック「Sensor1」のデータを受信していることがわかります。



ロータリーSW の値を変化させて、受信データを確認してください。

「51_MQTT.py」は、受信するトピックを次のとおり設定しています。

```
#Sensor Board ロータリーSWでTopic0～4を設定
topics = ['Sensor0', 'Sensor1', 'Sensor2', 'Sensor3', 'Sensor4']
```

このため、Sensor Board に実装されているロータリーの値を「5」以上に設定した場合、MQTT Broker（Raspberry Pi）はデータを受信しません。

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# -*- coding: utf-8 -*-
3	
4	import paho.mqtt.client as mqtt
5	import json
6	
7	host = '127.0.0.1'
8	port = 1883

Copyright © 2024 YITJC All Rights Reserved.

```
9  #topic複数設定
10 #Sensor Board ロータリーSWでTopic0～4を設定
11 topics = ['Sensor0', 'Sensor1', 'Sensor2', 'Sensor3', 'Sensor4']
12
13 def on_connect(client, userdata, flags, respons_code, properties):
14     print('status {0}'.format(respons_code))
15     # 全てのトピックをSubscribe
16     for topic in topics:
17         client.subscribe(topic)
18
19 def on_message(client, userdata, msg):
20     # 受信したtopic/qos/payloadを表示
21     print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))
22
23     if msg.topic in topics:
24         # バイト列を文字列に変換
25         payload_str = msg.payload.decode('utf-8')
26         # JSONデータを解析
27         try:
28             data = json.loads(payload_str)
29             temperature = float(data['temperature'])
30             humidity = float(data['humidity'])
31             pressure = float(data['pressure'])
32             #小数点以下2桁表示
33             temp_str = f"{temperature:.2f}"
34             hum_str = f"{humidity:.2f}"
35             press_str = f"{pressure:.2f}"
36             print(f"Temperature: {temp_str}, Humidity: {hum_str}, Pressure: {press_str}")
37         except ValueError as e:
38             print(f"Error parsing JSON: {e}")
39
40 if __name__ == '__main__':
41     try:
42         client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
43         client.on_connect = on_connect
44         client.on_message = on_message
45         client.connect(host, port=port, keepalive=60)
46         # 待ち受け状態にする
47         client.loop_forever()
```

Copyright © 2024 YITJC All Rights Reserved.

```
48      except KeyboardInterrupt:
49          pass
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: MQTT 通信を使用するためのライブラリ「paho」をインポート
- 5 行目: MQTT 通信データ（JSON 形式）を扱うために必要なライブラリ「json」をインポート
- 7 行目: MQTT Broker（mosquitto）のアドレスを定義
- 8 行目: MQTT Broker（mosquitto）のポート番号を定義
- 11 行目: MQTT Broker（mosquitto）が Subscribe するトピックを定義
  MQTT Broker は指定したトピックのメッセージを受信（Subscribe）します。
- 13 行目～17 行目: MQTT Broker に接続するプログラム on_connect()関数
- 14 行目: 接続結果のステータスコード（response_code）を print 表示
  MQTT Broker への接続に成功すると、「Success」と表示されます。

```
(ycuu) cuu@raspberrypi:~/dev $ ./51_MQTT.py
status Success
```

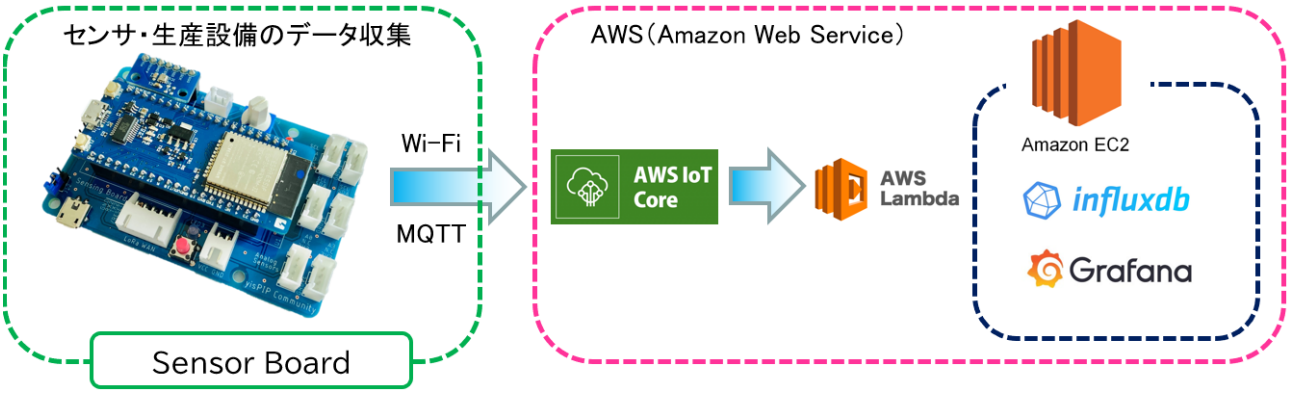
- 16 行目～17 行目: 指定したトピックのメッセージを Subscribe
  11 行目「topics」リストで指定したすべてのトピックに対して Subscribe を実行します。
- 19 行目～38 行目: MQTT 通信で受信したメッセージを表示するプログラム on_message()関数
- 21 行目: 受信した「topic」、「QoS」、「Payload」を表示
- 23 行目～38 行目: 「topics」リストで指定したトピックについて行う処理
  「topics」リストに含まれていないトピックを受信した場合は、何も処理を行いません。
- 25 行目: バイト列の受信データ「msg.payload」を UTF-8 で文字列にデコード
- 27 行目～36 行目：try ブロック
- 28 行目: JSON 形式の文字列を Python オブジェクトに変換し、変数「data」に格納
- 29 行目: 変数「data」から温度データを取得し、float()関数を使用して浮動小数点数に変換
- 30 行目: 変数「data」から湿度データを取得し、float()関数を使用して浮動小数点数に変換
- 31 行目: 変数「data」から気圧データを取得し、float()関数を使用して浮動小数点数に変換
- 33 行目: f"{temperature:.2f}" を使用して、温度データを小数点以下 2 桁までの文字列に変換
- 34 行目: f"{humidity:.2f}" を使用して、湿度データを小数点以下 2 桁までの文字列に変換
- 35 行目: f"{pressure:.2f}" を使用して、気圧データを小数点以下 2 桁までの文字列に変換
- 36 行目: 温度、湿度、気圧データを print 表示
- 37 行目～38 行目：except ブロック
- 40 行目～49 行目: main プログラム
- 41 行目～47 行目：try ブロック
- 42 行目: mqtt.Client クラスを使用してインスタンス「client」を作成
- 43 行目: MQTT Broker に接続した際に実行されるコールバック関数として on_connect を指定
- 44 行目: MQTT Broker がメッセージを受信した際に実行されるコールバック関数として on_message を指定

Copyright © 2024 YITJC All Rights Reserved.

- 45 行目: MQTT Broker に接続
- 47 行目: MQTT クライアント(Sensor Board)からのメッセージ受信待機状態を維持
- 48 行目～49 行目:except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたときに実行される処理として、「pass」文(何も処理しない)を記述しています。

参考～Raspberry Pi レス IoT システム～

Wi-Fi 通信に対応した Sensor Board を活用することで、直接クラウドサービス(AWS)にデータ送信できます。データベース(InfluxDB)や可視化ソフトウェア(Grafana)は、クラウドサービス上のコンピュータ(EC2)にインストールして可視化システムを構築します。これにより、Raspberry Pi を使用しないシステムを構築できます。

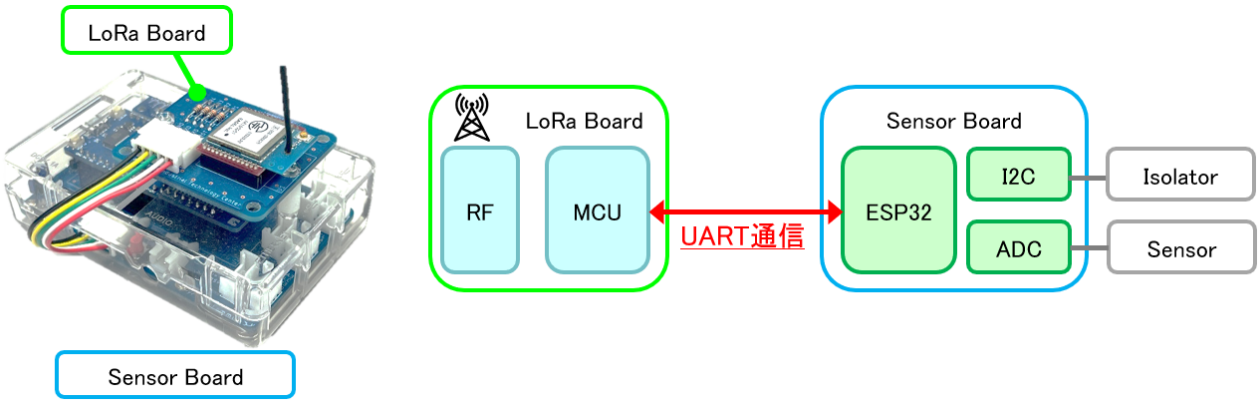


Copyright © 2024 YITJC All Rights Reserved.

11.7 LoRa 通信～データ送信～

Sensor Board は、Wi-Fi が整備されていない環境や広大な工場敷地、圃場などにおける通信環境を整備するため、LPWA（Low Power Wide Area）通信の一種である「LoRa」通信を利用できます。LoRa 通信を使用することで、Wi-Fi 通信に比べて広範囲でのセンシングが可能になります。また、本講座では、Raspberry Pi をゲートウェイ、Sensor Board をエンドデバイスとした「プライベート Lora」通信網を構築するため、通信料金の発生がなく、運用コストを抑えた IoT システムを実現できます。

ここでは、Sensor Board と LoRa Board を組み合わせて、LoRa 通信によるデータ送信の手順について説明します。



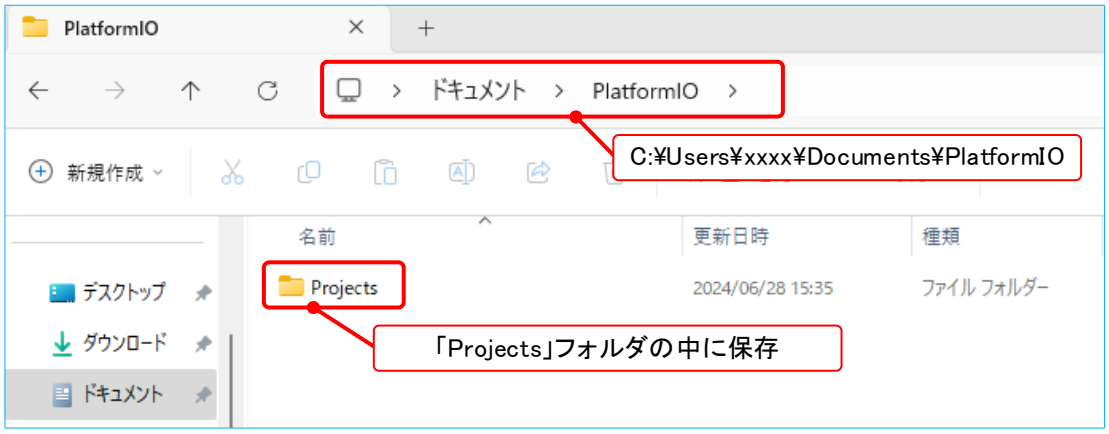
復習～活線挿抜～

LoRa 通信モジュール（ES920LR2）や LoRa Board は、活線挿抜をしないでください。ES920LR2 故障の原因になります。以下の説明は、3 章で説明した「活線挿抜」の復習です。

電源を入れた状態でケーブルを抜き挿しすることを「活線挿抜」と言います。活線挿抜は、電子部品（センサや IC など）の故障につながる危険性があります。電子部品に電源が供給された状態で部品を挿抜すると、急激な電流変化やスパイク電圧が発生する可能性があります。これらは電子部品に対して「過電流」や「過電圧」を引き起こし、電子部品内の微細な回路や構成要素を破壊する恐れがあります。特に IC（集積回路）などの高度な電子部品は、微小な電気的変化でも影響を受けやすく、部品の故障や損傷につながる可能性が高まります。そのため、電子部品の取り扱いや交換を行う際には、必ず電源を切って（基板への電源供給を遮断して）から作業を行うことが重要です。

手順①～プロジェクト配置～

共有サイトから対象のプロジェクトをダウンロードして、「Projects」ディレクトリに配置してください。



Copyright © 2024 YITJC All Rights Reserved.

[対象プロジェクト]

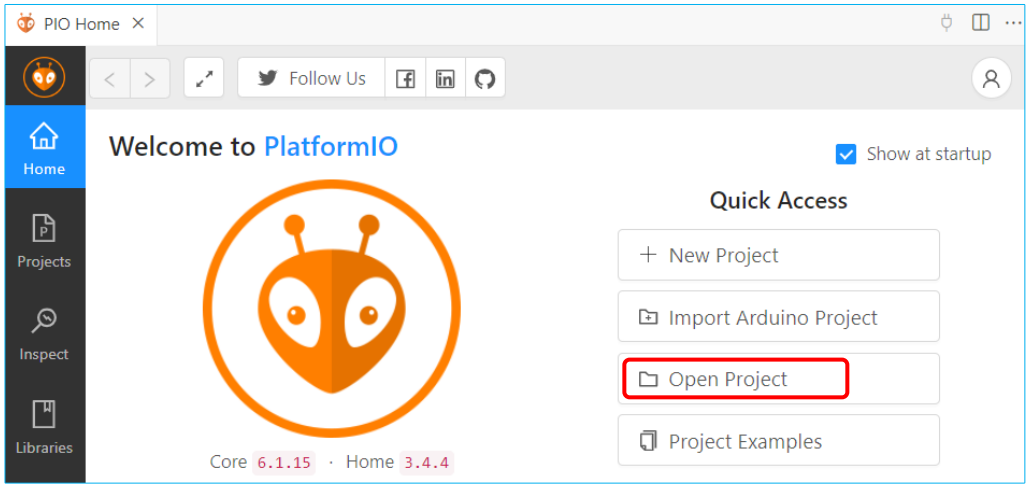
Sensor_Board4

[プログラム内容]

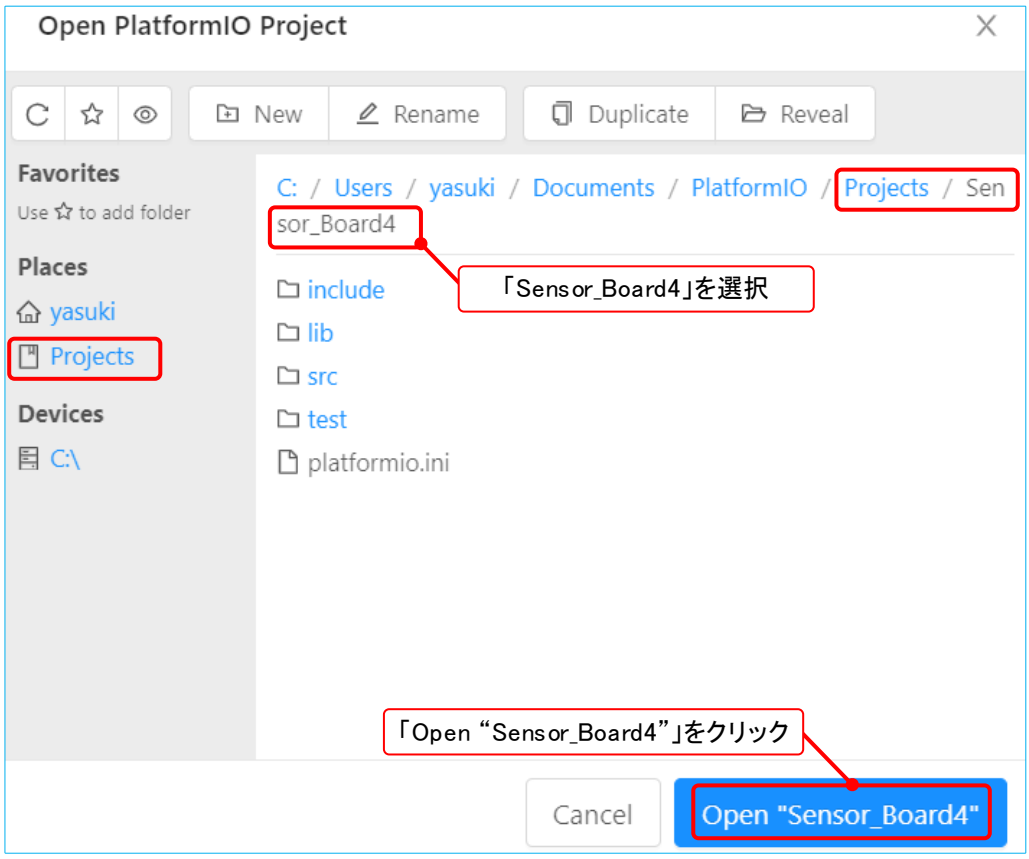
LoRa 通信を使用して、BME280（温湿度・気圧センサ）および外部電源 2 のデータを Raspberry Pi に送信します。

手順②～プロジェクトオープン～

PlatformIO Home 画面で「Open Project」をクリックしてください。



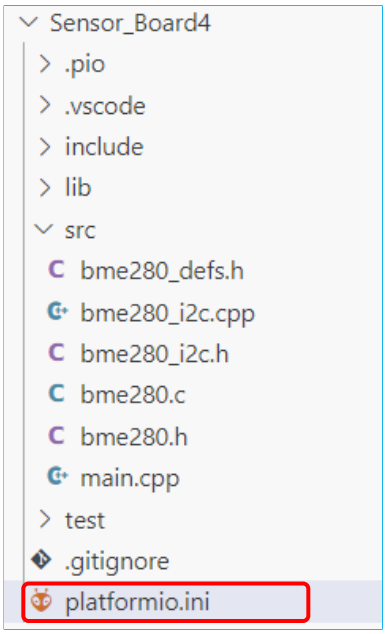
プロジェクト「Sensor_Board4」を選択してから、「Open Projects」をクリックしてください。

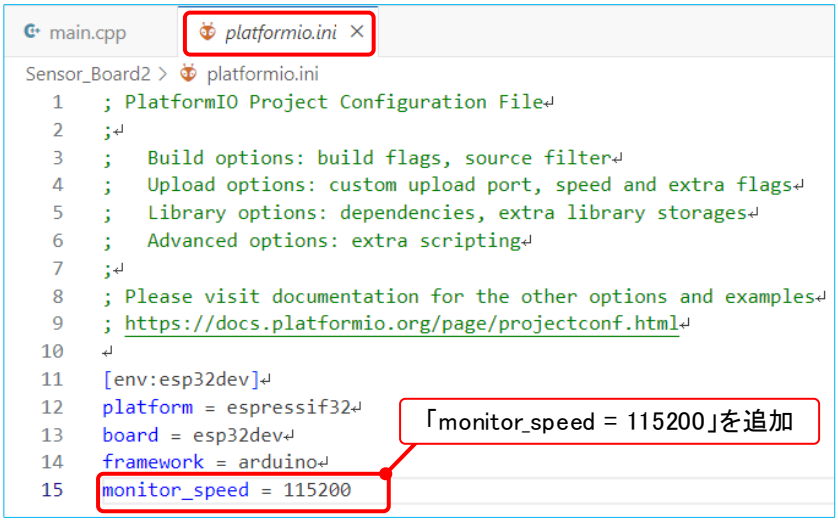


Copyright © 2024 YITJC All Rights Reserved.

手順③～シリアルモニタ設定～

BME280（温湿度・気圧センサ）および外部電源 2 のデータは、Platform IO のシリアルモニタ（Serial Monitor）に表示します。シリアルモニタの通信速度を 115200bps に設定するため、「platformio.ini」ファイルに通信速度（monitor_speed）の記述を追加してください。



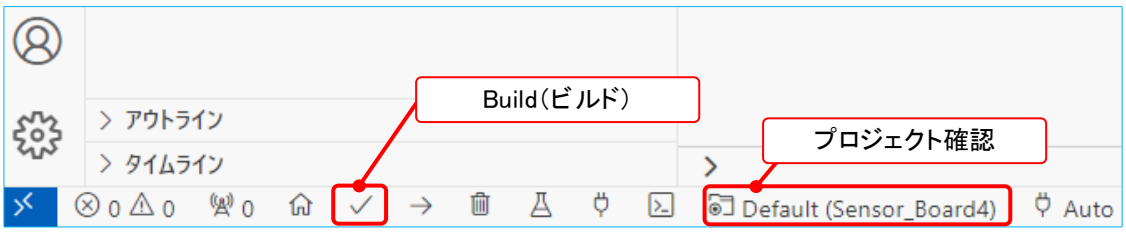


手順④～ビルド実行～

画面下ステータスバーに表示されているメニューから「Build（ビルド）」をクリックしてください。

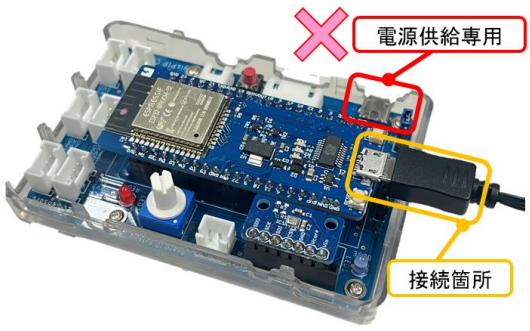
このとき、ビルドを実行するプロジェクトが「Sensor_Board4」であることを確認してください。

※プロジェクトが異なる場合は、「プロジェクト名」をクリックして正しい「プロジェクト」を選択してください。



手順⑤～ESP32 接続～

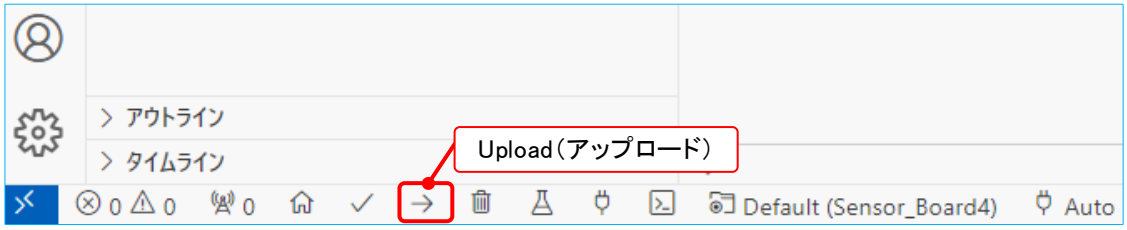
Sensor Board に搭載されている ESPr® Developer 32 と Micro USB ケーブルで接続します。プログラムを書き込む場合は、必ず ESPr® Developer 32 に Micro USB ケーブルを接続してください。



Copyright © 2024 YITJC All Rights Reserved.

手順⑥～プログラム書き込み～

画面下ステータスバーに表示されているメニューから「Upload（アップロード）」をクリックしてください。



このあと、データ受信側（Raspberry Pi）の開発を行うため、プログラムの書き込みが終了したら、Sensor Boardの電源を切ってください。

～プログラム解説～

行	プログラム
1	#include <Arduino.h>
2	#include <Wire.h>
3	#include "bme280_i2c.h"
4	#include "esp_sleep.h"
5	
6	#define SDA 21
7	#define SCL 22
8	
9	//Deep Sleep Period [sec]
10	#define S_PERIOD 30
11	
12	//LED Pins
13	const int REDLED = 12; //GPIO12
14	const int BLUELED = 13; //GPIO13
15	//SW Pins
16	const int SW = 19; //GPIO19
17	//Rotary SW Pins
18	const int RSW1 = 14; //GPIO14
19	const int RSW2 = 15; //GPIO15
20	const int RSW4 = 26; //GPIO26
21	const int RSW8 = 25; //GPIO25
22	//ADC Parameter
23	const int A7Pin = 35; //A7 VBAT2
24	const int VOLT = 3.3; //3.3V
25	const int ANALOG_MAX = 4096; //12bit
26	//LoRa Reset
27	const int LoRa_Rst = 18;

Copyright © 2024 YITJC All Rights Reserved.

```
28  //Board ID
29  int ID;
30
31  BME280 bme280(BME280_I2C_ADDR_PRIM);
32  struct bme280_data data;
33
34  void LoRa_reset(void) {
35      digitalWrite(LoRa_Rst, LOW);
36      delay(100);
37      digitalWrite(LoRa_Rst, HIGH);
38  }
39
40  int LoRa_recv(char *buf) {
41      char *start = buf;
42
43      while (true) {
44          delay(0);
45          while (Serial2.available() > 0) {
46              *buf++ = Serial2.read();
47              if (*(buf - 2) == '¥r' && *(buf - 1) == '¥n') {
48                  *buf = '¥0';
49                  return (buf - start);
50              }
51          }
52      }
53  }
54
55  int LoRa_send(const char * msg) {
56      const char *start = msg;
57      while (*msg != '¥0') {
58          Serial2.write(*msg++);
59      }
60      return (msg - start);
61  }
62
63  int sendcmd(const char *cmd) {
64      char buf[64];
65      //NG002対策（バッファクリア）
66      while (Serial2.available())Serial2.read();
```

Copyright © 2024 YITJC All Rights Reserved.

```
67      LoRa_send(cmd);
68      while (true) {
69          LoRa_recv(buf);
70          //showコマンド実行結果を表示
71          //Serial.print(buf);
72          if (strstr(buf, "OK")) {
73              //Serial.print(buf);
74              return true;
75          } else if (strstr(buf, "NG")) {
76              //Serial.print(buf);
77              return false;
78          }
79      }
80  }
81
82  void setMode() {
83      char buf[64];
84      //NG 002対策  Operation Modeで起動
85      //sendcmd("q 2¥r¥n");              //1:Congig Mode/2:Operation Mode
86      //sendcmd("w¥r¥n");                //saveコマンド
87      LoRa_send("config¥r¥n");
88      delay(200);
89      LoRa_reset();
90      delay(1500);
91
92      while (true) {
93          LoRa_recv(buf);
94          if (strstr(buf, "Mode")) {
95              Serial.print(buf);
96              break;
97          }
98      }
99
100     sendcmd("2¥r¥n");                //1:terminal/2:processor
101     sendcmd("node 2¥r¥n");           //1:Coordinator/2:EndDevice/3:Router
102     sendcmd("panid 1122¥r¥n");    //PAN Network Address
103     sendcmd("ownid 1122¥r¥n");    //自ノード  Network Address
104     sendcmd("dstid 0000¥r¥n");    //送信先  Network Address
105     sendcmd("rssi 1¥r¥n");           //受信電波強度付与  1:ON/2:OFF
```

Copyright © 2024 YITJC All Rights Reserved.

```
106     sendcmd("bw 4¥r¥n");              //帯域幅:125kHz
107     sendcmd("sf 7¥r¥n");              //拡散率: 7
108     sendcmd("channel 1¥r¥n");         //無線channel: 920.6MHz
109     sendcmd("q 2¥r¥n");               //1:Congig Mode/2:Operation Mode
110     sendcmd("retry 0¥r¥n");           //retry : 0
111     sendcmd("ack 1¥r¥n");             //ACK : ON
112     sendcmd("w¥r¥n");                 //saveコマンド
113     //sendcmd("show¥r¥n");            //showコマンド（設定値の確認）
114
115     LoRa_reset();
116     Serial.println("LoRa module setup complete");
117     delay(1000);
118 }
119
120 void send2LoRa() {
121     char obuf[64], ibuf[64];
122     char * buf;
123
124     //Board ID取得（Rotary SW）
125     ID = digitalRead(RSW8) * 8 + digitalRead(RSW4) * 4 + digitalRead(RSW2) * 2 + digitalRead(RSW1);
126     Serial.printf("Board ID: %d¥r¥n", ID);
127     //BME280データ取得
128     bme280.get_sensor_data(&data);
129     Serial.printf("temp[deg C]: %.1f¥r¥n", data.temperature);
130     Serial.printf("humid[percent]: %.1f¥r¥n", data.humidity);
131     Serial.printf("press[hPa]: %.1f¥r¥n", data.pressure / 100);
132     //Analog Read(A7:VBAT2)
133     int reading7 = analogRead(A7Pin);
134     //AD値をmVに変換
135     float vbat7 = ((long)reading7 * VOLT * 1000.0) / ANALOG_MAX;
136     Serial.printf("Voltage[mV]: %.2f¥r¥n", vbat7);
137
138     //LoRa送信データ
139     strcpy(obuf, "data=(");
140     dtostrf(ID, 1, 0, &obuf[strlen(obuf)]);
141     strcat(obuf, ",");
142     dtostrf(data.temperature, 5, 2, &obuf[strlen(obuf)]);
143     strcat(obuf, ",");
144     dtostrf(data.humidity, 5, 2, &obuf[strlen(obuf)]);
```

Copyright © 2024 YITJC All Rights Reserved.

```
145     strcat(obuf, ",");
146     dtostrf(data.pressure / 100, 5, 1, &obuf[strlen(obuf)]);
147     strcat(obuf, ",");
148     dtostrf((vbat7 / 1000) * 2, 5, 2, &obuf[strlen(obuf)]);
149     strcat(obuf, ")¥r¥n");
150
151     digitalWrite(BLUELED, HIGH);
152     delay(100);
153     Serial.print("send to LoRa: ");
154     buf = obuf;
155     Serial.print(buf);
156     LoRa_send(buf);
157
158     while (true) {
159       LoRa_recv(ibuf);
160       if (strstr(ibuf, "OK")) {
161         Serial.print(ibuf);
162         break;
163       } else if (strstr(ibuf, "NG")) {
164         Serial.print(ibuf);
165         while (strstr(ibuf, "NG")) {
166           LoRa_send(buf);
167           LoRa_recv(ibuf);
168           if (strstr(ibuf, "OK")) {
169             Serial.print(ibuf);
170             break;
171           } else if (strstr(ibuf, "NG")) {
172             Serial.print(ibuf);
173             digitalWrite(BLUELED, LOW);
174             digitalWrite(REDLED, HIGH);
175             delay(1000);
176             digitalWrite(REDLED, LOW);
177             delay(1000);
178           }
179         }
180         break;
181       }
182     }
183     digitalWrite(BLUELED, LOW);
```

Copyright © 2024 YITJC All Rights Reserved.

```
184     //データ送信間隔
185     //delay(5000);
186     //送信間隔が長い場合は、Deep Sleepに移行
187     Serial.printf("enter deep sleep¥n");
188     delay(10);
189     esp_deep_sleep(1000000LL * S_PERIOD);                    // S_PERIOD秒Deep Sleepする
190     Serial.printf("in deep sleep¥n");
191 }
192
193 void setup() {
194     //LED Pin Settings
195     pinMode(REDLED, OUTPUT);          //RED LED
196     pinMode(BLUELED, OUTPUT);         //BLUE LED
197     digitalWrite(REDLED, LOW);       //RED LED OFF
198     digitalWrite(BLUELED, LOW);      //BLUE LED OFF
199     //Rotary SW Pin Settings
200     pinMode(RSW1, INPUT);    //RSW1
201     pinMode(RSW2, INPUT);    //RSW2
202     pinMode(RSW4, INPUT);    //RSW4
203     pinMode(RSW8, INPUT);    //RSW8
204     //LoRa Reset (Open Drain)
205     pinMode(LoRa_Rst, OUTPUT);
206     digitalWrite(LoRa_Rst, HIGH);     //ES920LR Reset
207     //I2C Settings
208     Wire.begin(SDA, SCL);
209     bme280.begin();
210     Serial.begin(115200);
211     delay(20);
212     //LoRa Serial通信
213     Serial2.begin(115200);
214     LoRa_reset();
215     //LoRa ResetからESP通信開始まで1秒待機(仕様)
216     delay(1000);
217     setMode();
218     Serial.println(F("¥r¥nStart"));
219 }
220
221 void loop() {
222     send2LoRa();
```

Copyright © 2024 YITJC All Rights Reserved.

```
223 }
```

灰色部は、これまでに説明した内容です。

- 1 行目: Arduino の標準ライブラリを使用するため「Arduino.h」をインクルード
- 2 行目: I2C 通信用のライブラリ「Wire.h」をインクルード
- 3 行目: BME280 センサ用のライブラリ「bme280_ic2.h」をインクルード
- 4 行目: ESP32 をスリープ制御するためのライブラリ「esp_sleep.h」をインクルード
- 6 行目: I2C 通信 SCL のピン番号定義
- 7 行目: I2C 通信 SDA のピン番号定義
- 10 行目: Deep Sleep モードの周期（秒）を設定
- 13 行目: 赤色 LED のピン番号を定義
- 14 行目: 青色 LED のピン番号を定義
- 16 行目: タクト SW（赤色）のピン番号を定義
- 18 行目～21 行目: ロータリーSW のピン番号定義
- 23 行目: 外部電源 2 監視用アナログ端子のピン番号定義
- 24 行目: AD 変換で使用する基準電圧を定義
- 25 行目: AD 変換の分解能を定義
- 27 行目: LoRa 通信モジュールリセット制御端子のピン番号定義
- 29 行目: Sensor Board を識別するための変数「ID」を定義
- 31 行目: BME280 センサの I2C アドレスを指定してインスタンス作成
- 32 行目: BME280 センサから取得するデータを格納するための構造体変数「data」を定義
  「struct bme280_data」は、BME280 センサから取得するデータを格納するための構造体です。

～LoRa 通信モジュールリセット制御～

- 34 行目～37 行目: LoRa 通信モジュールをリセットするためのプログラム LoRa_reset()関数
- 35 行目: リセット端子（GPIO18）から Low を出力して LoRa 通信モジュールをリセット
- 36 行目: 100 ミリ秒プログラムを停止し、LoRa 通信モジュールのリセット状態を維持
- 37 行目: リセット端子（GPIO18）から High を出力して LoRa 通信モジュールのリセットを解除

～LoRa 通信モジュールからの受信データ格納～

- 40 行目～53 行目: LoRa 通信モジュールから受信したデータを格納するプログラム LoRa_recv()関数
- 41 行目: LoRa 通信モジュールからの受信データを格納するためのバッファ「buf」を使用
  受信データの長さを計算するため、ポインタ「star」をバッファ「buf」の先頭に設定しています。
- 43 行目～52 行目: データ受信処理
- 44 行目: スピンロック回避（無駄な CPU リソースの消費を回避）のため delay(0)処理を追加
- 45 行目: シリアルポート（Serial2）のデータ有無チェック
- 46 行目～51 行目: シリアルポートにデータが存在する場合の処理
- 46 行目: Serial2.read()メソッドで 1 バイトのデータを読みより、バッファ「buf」に格納
  バッファをインクリメント「*buf++」してシリアル通信で受信したデータを順次「buf」に格納しています。

Copyright © 2024 YITJC All Rights Reserved.

- 47 行目: 受信データの終了位置判定
  最後に受信したデータ 2 バイトが「CR（キャリッジリターン）」＋「LF（ラインフィード）」で終了しているか確認しています。
- 48 行目: 「CR」＋「LF」で終了している場合の処理
  シリアル通信データの終端文字として null 文字「¥0」を追加します。
- 49 行目: 受信データの長さを計算
  ポインタ「buf」が現在指している位置からポインタ「start」が指している位置を引いた値「buf – start」を、バッファに格納された受信データの長さ（バイト数）として表しています。

～LoRa データ送信～

- 55 行目～61 行目: データ送信するためのプログラム LoRa_send()関数
  Raspberry Pi に送信するデータを LoRa 通信モジュールに送信するため、シリアル通信を使用しています。
- 56 行目: 送信データの長さを計算するため、ポインタ「star」をバッファ「buf」の先頭に設定
- 57 行目: 送信データの末尾（null 文字）まで繰り返し処理を実行
- 58 行目: Serial2.write()メソッドで 1 バイトずつデータを送信
- 60 行目: 送信データの長さを計算
  ポインタ「msg」が現在指している位置からポインタ「start」が指している位置を引いた値「msg – start」を、バッファに格納された送信データの長さ（バイト数）として表しています。

～LoRa コマンド送信～

- 63 行目～80 行目: LoRa 通信モジュールにコマンドを送信するためのプログラム sendcmd()関数
- 64 行目: LoRa 通信モジュールから受信したデータを格納するバッファ「buf」を定義
  バッファサイズは 64 バイトに設定しています。
- 66 行目: シリアル通信受信バッファのデータをクリア
  受信バッファにデータが残った状態でコマンドを送信すると、NG002（オプション値異常）が発生します。
  そのため、コマンド送信する前に受信バッファのデータをクリアしています。

～補足～

過去に NG002 が発生したケースとして、次の状況が報告されています。

- ✓ LoRa の通信環境が NG（配線間違いなど）の状態で何回もプログラムを実行した結果、シリアル受信バッファに、過去の応答を蓄積してしまったケース
- ✓ ノイズやチャタリング等によって LoRa 通信モジュール（ES920LR2）にリセットが複数回発生した結果、受信バッファにゴミデータが蓄積されてしまったケース
- ✓ ノイズを受信した結果、受信バッファにゴミデータが蓄積されてしまったケース

- 67 行目: LoRa_send()関数を使用してコマンド「cmd」を LoRa 通信モジュールに送信
- 68 行目～79 行目: LoRa 通信モジュールからの応答（OK or NG）が返るまで繰り返し処理
- 69 行目: LoRa_recv()関数を使用して LoRa 通信モジュールからの応答を受信
- 72 行目～78 行目: LoRa 通信モジュールからの応答処理
  応答に「OK」が含まれていれば「True」を返し、「NG」が含まれていれば「False」を返しています。

Copyright © 2024 YITJC All Rights Reserved.

～LoRa 動作モード設定～

- 82 行目～117 行目: LoRa 通信モジュールに動作モードを設定するためのプログラム setMode()関数
- 83 行目: LoRa 通信モジュールから受信したデータを格納するバッファ「buf」を定義
  バッファサイズは 64 バイトに設定しています。
- 85 行目～86 行目: NG002 が発生した場合の対策
  NG002 が発生した場合、LoRa 通信モジュールに動作モードを設定できません。対策として、次の手順を実行してください。
  ① 85 行目～86 行目のコメントを外してからプログラムを書き込んでください。
  ② 再び 85 行目～86 行目をコメントアウトしてからプログラムを書き込んでください。
- 87 行目: 「config」コマンドを送信してコンフィグレーションモードに移行
- 88 行目: 200 ミリ秒待機
- 89 行目: LoRa_reset()関数を使用して LoRa 通信モジュールをリセット
- 90 行目: 1500 ミリ秒待機
- 92 行目～98 行目: LoRa 通信モジュールからの応答を受信したときの処理
- 93 行目: LoRa 通信モジュールからの応答を受信しバッファ「buf」に格納
- 94 行目: 受信データに「Mode」が含まれていた場合の処理
- 95 行目: バッファに格納された受信データをシリアルモニタに出力
  LoRa 通信モジュールからの応答を正常に受信できた場合、次の内容がシリアルモニタに出力されます。
  Select Mode [1.terminal or 2.processor]
- 96 行目: ループ処理を抜ける
- 100 行目～112 行目: LoRa 通信モジュールにコマンドを送信
- 100 行目: プロセッサモード「2」を送信
- 101 行目: node コマンドを使用して「2 EndDevice」を送信
- 102 行目: panid コマンドを使用して PAN ネットワークアドレス「1122」を送信
  親機（Coordinator）と同じ「panid」を設定します。
- 103 行目: ownid コマンドを使用して自ノードネットワークアドレス「1122」を送信
  子機（Sensor Board）が複数存在する場合は、子機ごとに固有の「ownid」を設定してください。
- 104 行目: dstid コマンドを使用して送信先ノードネットワークアドレス「0000」を送信
  親機（Coordinator）に送信する場合は、「0000」を設定します。
- 105 行目: rssi コマンドを使用して受信電波強度（RSSI）付与設定「1 ON」を送信
- 106 行目: bw コマンドを使用して帯域幅設定「4 125kHz」を送信
  親機（Coordinator）と同じ値を設定します。
- 107 行目: sf コマンドを使用して拡散率「7」を送信
  親機（Coordinator）と同じ値を設定します。
- 108 行目: channel コマンドを使用して無線チャンネル番号設定「1」を送信
  親機（Coordinator）と同じ値を設定します。
- 109 行目: q コマンドを使用して動作モード「2 Operation Mode」を送信
- 110 行目: ACK コマンドを使用して Acknowledge 使用設定「1 ON」を送信
- 111 行目: retry コマンドを使用してリトライ回数「0」を送信

Copyright © 2024 YITJC All Rights Reserved.

- 112 行目: w コマンドを使用して EEPROM への設定値書き込みを指示
- 113 行目: EEPROM の設定値を確認するためのコマンド（通常はコメントアウト）
- 115 行目: LoRa_reset()関数を使用して LoRa 通信モジュールをリセット
- 116 行目:設定完了メッセージ「LoRa module setup complete」をシリアルモニタに表示
- 117 行目: 1000 ミリ秒待機

～LoRa 送信データ定義・送信～

- 120 行目～191 行目: 送信データを定義し、データを送信するためのプログラム send2LoRa()関数
- 121 行目: 送信データを格納するバッファ「obuf」、受信データを格納するバッファ「ibuf」を定義
  バッファサイズは 64 バイトに設定しています。
- 122 行目: ポインタ「buf」を定義
- 125 行目: ロータリーSW に設定した値を取得
- 126 行目: ロータリーSW の設定値をシリアルモニタに表示
- 128 行目: BME280 センサのデータを取得
- 129 行目～130 行目: BME280 センサのデータをシリアルモニタに print 表示
- 133 行目: 外部電源 2 監視用アナログ端子（A7）のアナログ値を読み取り、デジタル値を「reading7」に格納
  ここでは AD 変換処理を行っています。
- 135 行目: 読み取ったデジタル値「reading7」を電圧値に変換
- 136 行目: 外部電源 2 監視用アナログ端子（A7）の電圧値を mV 単位でシリアルモニタに表示
- 139 行目～149 行目: Raspberry Pi に送信するデータを定義
  次のフォーマットを使用してデータを送信します。
  data = （Board ID, temperature, humidity, pressure, vbat7）¥r¥n
  Board ID : Sensor Board の識別 ID（ロータリーSW の値）
  temperature : BME280 センサの温度データ
  humidity : BME280 センサの湿度データ
  pressure : BME280 センサの気圧データ
  vbat7 : 外部電源 2 に供給している電池の残電圧データ
- 139 行目: strcpy メソッドを使用して「data=(」をバッファ「obuf」にコピー
- 140 行目: dtostrf メソッドを使用して文字列に変換したロータリーSW の値「ID」をバッファ「obuf」の末尾に追加
  dtostrf (ID, 1, 0, &obuf[strlen(obuf)]);
  第 1 引数: 文字列に変換する値（ID）
  第 2 引数: 文字列の桁数
  第 3 引数: 小数点以下の桁数
  第 4 引数: 文字列に変換した結果を格納するバッファのアドレス
- 141 行目: srtcat メソッドを使用してカンマ「,」をバッファの末尾に追加
- 142 行目: dtostrf メソッドを使用して文字列に変換した温度データをバッファ「obuf」の末尾に追加
  温度データ「temperature」を 5 桁（小数点以下 2 桁）の文字列に変換してからバッファの末尾に格納しています。
- 143 行目: srtcat メソッドを使用してカンマ「,」をバッファの末尾に追加

Copyright © 2024 YITJC All Rights Reserved.

- 144 行目: dtostrf メソッドを使用して文字列に変換した湿度データをバッファ「obuf」の末尾に追加
  湿度データ「humidity」を 5 桁（小数点以下 2 桁）の文字列に変換してからバッファの末尾に格納しています。
- 145 行目: srtcat メソッドを使用してカンマ「,」をバッファの末尾に追加
- 146 行目: dtostrf メソッドを使用して文字列に変換した気圧データをバッファ「obuf」の末尾に追加
  気圧データ「pressure」を 100 で割った値を文字列に変換し、5 桁（小数点以下 1 桁）の文字列をバッファの末尾に格納しています。
- 147 行目: srtcat メソッドを使用してカンマ「,」をバッファの末尾に追加
- 148 行目: dtostrf メソッドを使用して文字列に変換した電圧データをバッファ「obuf」の末尾に追加
  電圧データ「vbat7」を 1000 で割ってから 2 倍した値を文字列に変換し、5 桁（小数点以下 2 桁）の文字列をバッファの末尾に格納しています。
- 149 行目: srtcat メソッドを使用してカンマ「)¥r¥n」をバッファの末尾に追加
- 151 行目: 青色 LED が接続されている GPIO13 ピンに「High」を設定し、青色 LED を点灯
- 152 行目: 100 ミリ秒プログラムを停止し、青色 LED の点灯状態を維持
- 153 行目～155 行目: Raspberry Pi に送信するデータをシリアルモニタに表示
  シリアルモニタには、次の内容が表示されます。
  send to LoRa: data = （Board ID, temperature, humidity, pressure, vbat7）
- 156 行目: LoRa_send()関数を使用してデータ送信

～LoRa 送信結果確認・再送処理～

- 158 行目～191 行目: 送信結果の確認・再送処理
- 159 行目: LoRa_recv()関数を使用して LoRa 通信モジュールからの応答を受信
- 160 行目～162 行目: 応答に「OK」が含まれている場合の処理
  シリアルモニタに「OK」と表示し、ループ処理を抜けます。
- 163 行目～164 行目: 応答に「NG」が含まれている場合の処理
- 164 行目: シリアルモニタに「NG」および「レスポンスコード」を表示し、再送処理に移行します。
- 165 行目～179 行目: 再送処理
- 166 行目: LoRa_send()関数を使用してデータ再送信
- 167 行目: LoRa_recv()関数を使用して LoRa 通信モジュールからの応答を受信
- 168 行目～170 行目: 応答に「OK」が含まれている場合の処理
  シリアルモニタに「OK」と表示し、ループ処理を抜けます。
- 171 行目～178 行目: 応答に「NG」が含まれている場合の処理
- 172 行目: シリアルモニタに「NG」および「レスポンスコード」を表示し、再送処理に移行します。
- 173 行目: 青色 LED が接続されている GPIO13 ピンに「Low」を設定し、青色 LED を消灯
- 174 行目: 赤色 LED が接続されている GPIO12 ピンに「High」を設定し、赤色 LED を点灯
- 175 行目: 1000 ミリ秒（1 秒）プログラムを停止し、赤色 LED の点灯状態を維持
- 176 行目: 赤色 LED が接続されている GPIO12 ピンに「Low」を設定し、赤色 LED を消灯
- 177 行目: 1000 ミリ秒（1 秒）プログラムを停止し、赤色 LED の消灯状態を維持
- 180 行目: ループ処理を抜ける
- 183 行目: 青色 LED が接続されている GPIO13 ピンに「Low」を設定し、青色 LED を消灯

Copyright © 2024 YITJC All Rights Reserved.

- 187 行目: シリアルモニタに「enter deep sleep」を表示
- 188 行目: 10 ミリ秒プログラムを停止し、シリアルモニタへの出力完了を待機
- 189 行目: Deep Sleep モードに移行
  10 行目で定義した時間「S_PERIOD」Deep Sleep モードに移行します。
  “1000000LL”は、1 秒をマイクロ秒単位で表現しています。
- 190 行目: デバッグ用
  Deep Slee モードに移行すると、88 行目のメッセージ「in deep sleep」が表示されることはありません。

～Sensor Board セットアップ～

- 193 行目～219 行目: setup()関数
  電源供給（またはリセット）時に一度だけ実行される処理を記載しています。
- 195 行目: 赤色 LED が接続されている GPIO12 ピンを「出力」モードに設定
- 196 行目: 青色 LED が接続されている GPIO13 ピンを「出力」モードに設定
- 197 行目: 赤色 LED が接続されている GPIO12 ピンの初期状態を「Low」に設定
- 198 行目: 青色 LED が接続されている GPIO13 ピンの初期状態を「Low」に設定
- 200 行目: ロータリーSW が接続されている GPIO14 ピンを「入力」モードに設定
- 201 行目: ロータリーSW が接続されている GPIO15 ピンを「入力」モードに設定
- 202 行目: ロータリーSW が接続されている GPIO26 ピンを「入力」モードに設定
- 203 行目: ロータリーSW が接続されている GPIO25 ピンを「入力」モードに設定
- 205 行目: LoRa 通信モジュールリセット端子が接続されている GPIO18 ピンを「出力」モードに設定
- 206 行目: LoRa 通信モジュールリセット端子が接続されている GPIO18 ピンの初期状態を「High」に設定
- 208 行目: I2C 通信を初期化
- 209 行目: 作成した BME280 センサのインスタンスを初期化
- 210 行目: 通信速度（115200bps）を指定してシリアル通信を開始
- 211 行目: 20 ミリ秒プログラムを停止
  シリアル通信が安定するのを待つため、20 ミリ秒の待機時間を設定しています。
- 213 行目: 通信速度（115200bps）を指定して LoRa 通信モジュールとのシリアル通信を開始
- 214 行目: LoRa_reset()関数を使用して LoRa 通信モジュールをリセット
- 216 行目: LoRa 通信モジュールに動作モードを設定するまで 1000 ミリ秒待機
- 217 行目: setMode()関数を使用して LoRa 通信モジュールに動作モードを設定
- 218 行目: シリアルモニタに「Start」を表示

～繰り返し処理～

- 221 行目～223 行目: loop 関数
  プログラムが動作している間、繰り返し実行される処理を記載しています。
- 222 行目: 送信データを定義し、データを送信するためのプログラム send2LoRa()関数実行

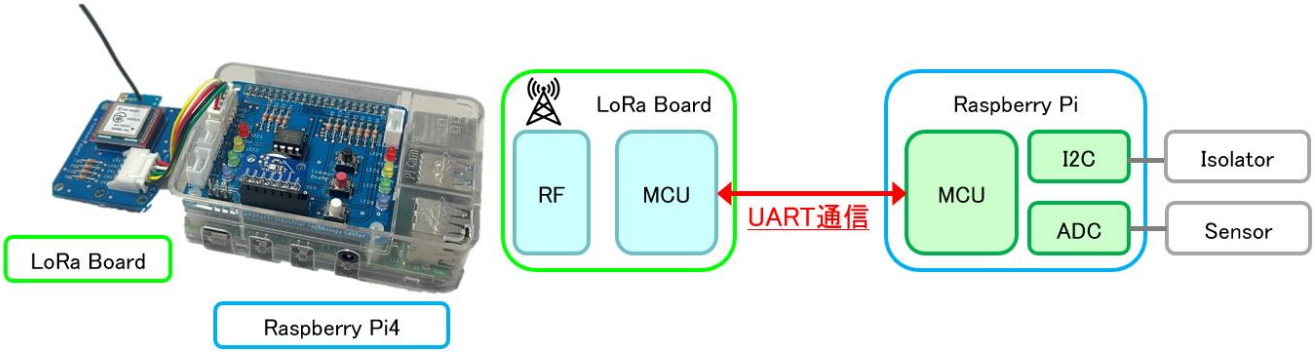
Copyright © 2024 YITJC All Rights Reserved.

11.8 LoRa 通信～データ受信～

LoRa 通信を使用して Sensor Board から送信された BME280（温湿度・気圧センサ）および外部電源 2 のデータを Raspberry Pi で受信します。ここでは、Python プログラムを使用して Sensor Board から送信されたデータを受信する方法について説明します。

手順①～LoRa Board 接続～

HAT Board は、LoRa Board と接続するためのコネクタを用意しています。活線挿抜を避けるため、Raspberry Pi に電源を供給していない状態で LoRa Board を接続してください。



手順②～UART 設定～

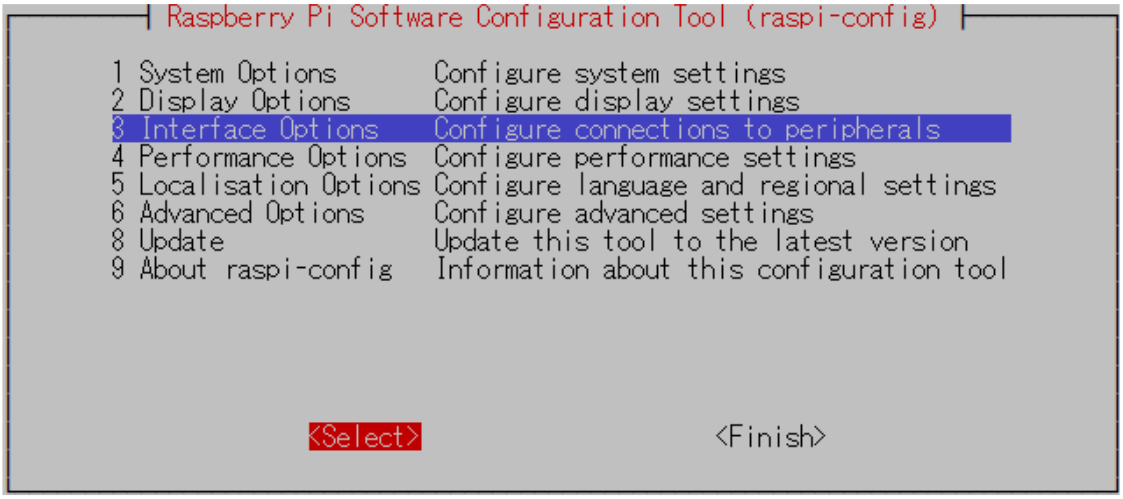
Raspberry Pi は LoRa 通信モジュール（ES920LR2）と UART 通信（非同期シリアル通信）を行います。ここでは、raspi-config 画面を操作して、Raspberry Pi UART 通信ポートの設定を行います。

(ycuu) cuu@raspberrypi:~ $ sudo raspi-config

補足～Raspi-config 画面の操作～

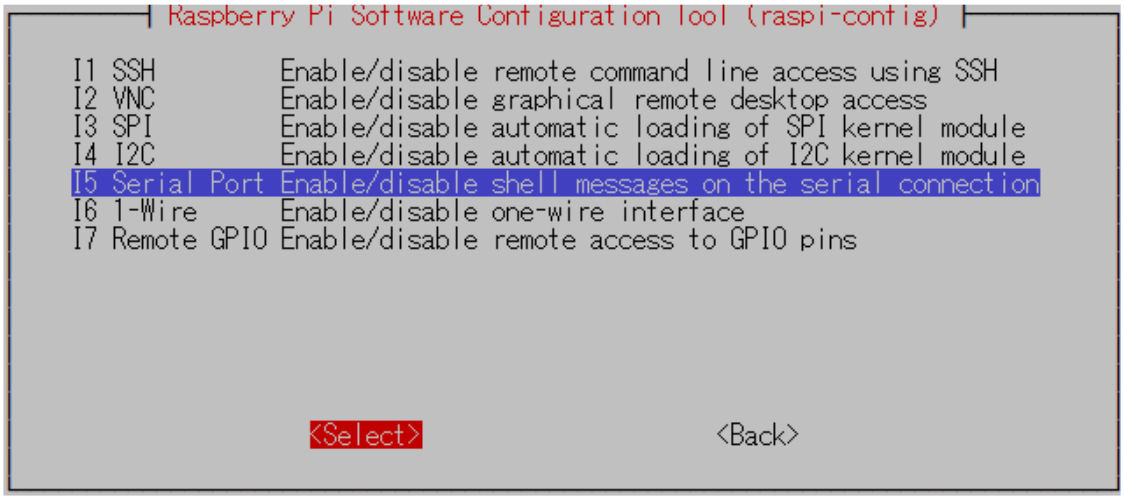
Raspi-config 画面では、「メニュー選択」は「→キー」、「カーソル移動」は「tab キー」、
「決定」は「Enter キー」で操作します。

「3 Interface Options」を選択してください。

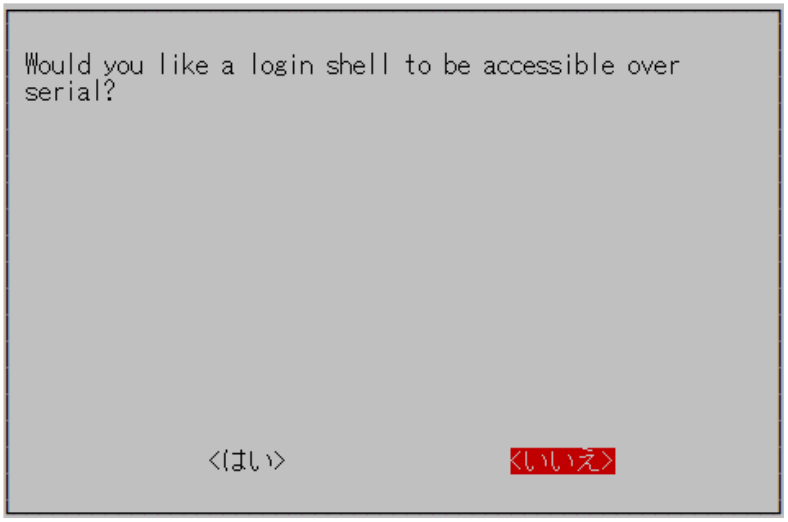


Copyright © 2024 YITJC All Rights Reserved.

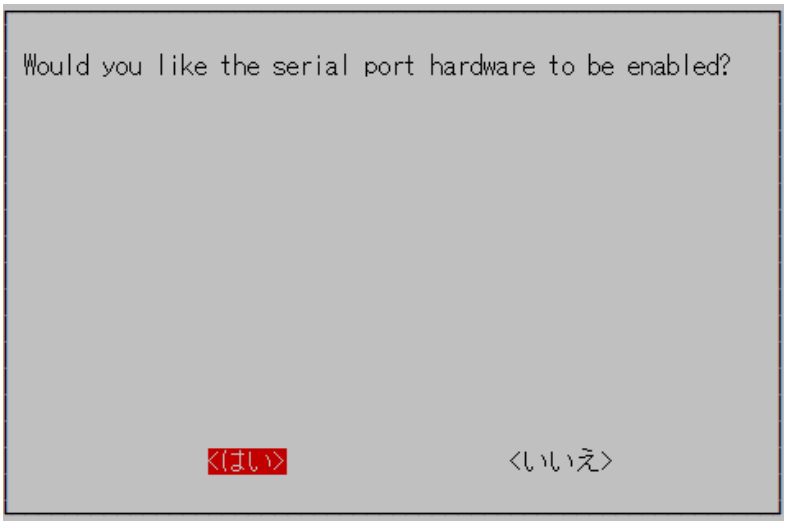
「I5 Serial Port」を選択してください。



「いいえ」を選択してください。

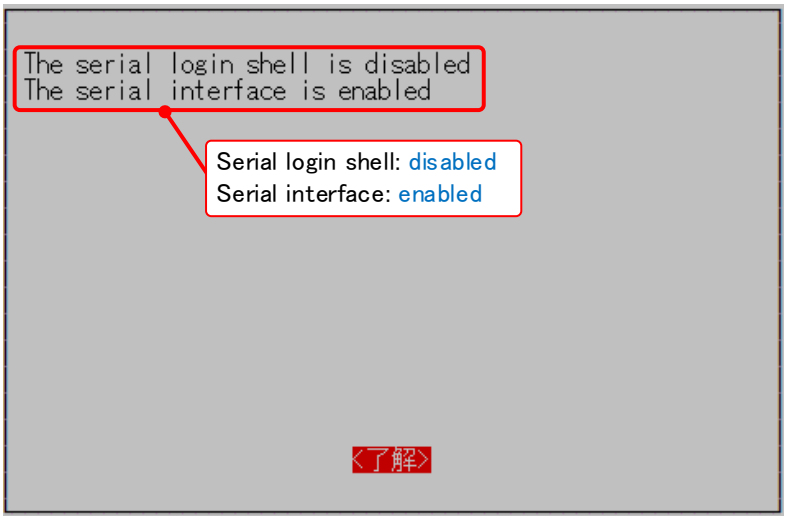


「はい」を選択してください。

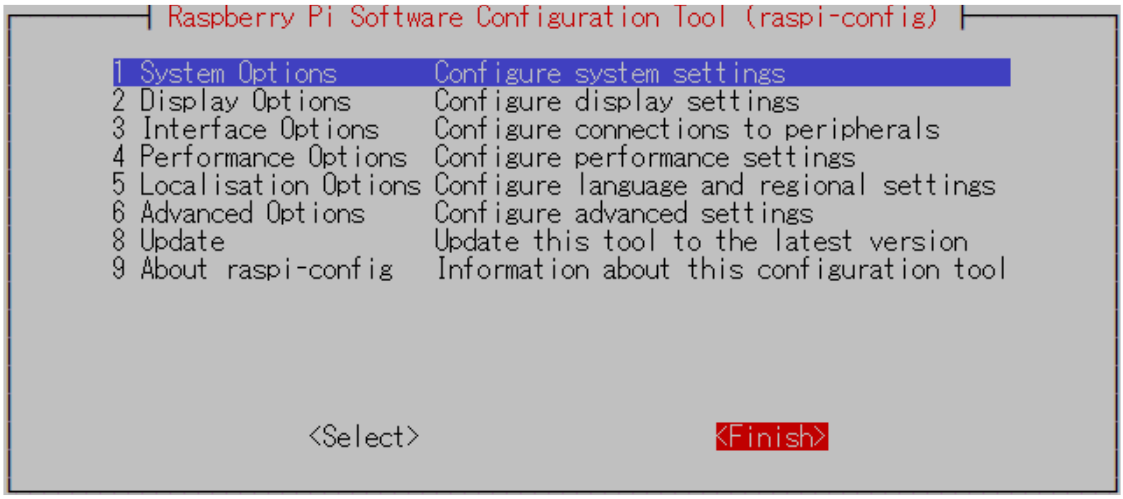


Copyright © 2024 YITJC All Rights Reserved.

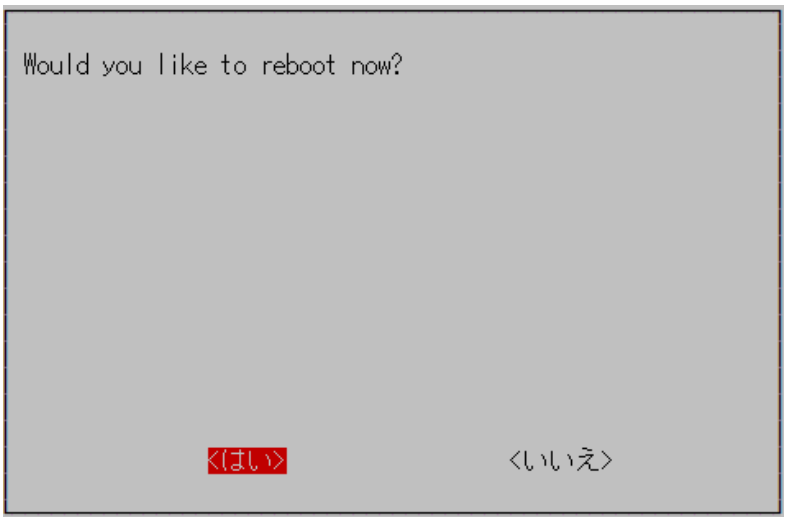
画面と同じ設定になっていることを確認してから、「了解」を選択してください。



カーソルを「finish」に移動して、raspi-config 画面を終了してください。



「はい」を選択して、Raspberry Pi を再起動してください。

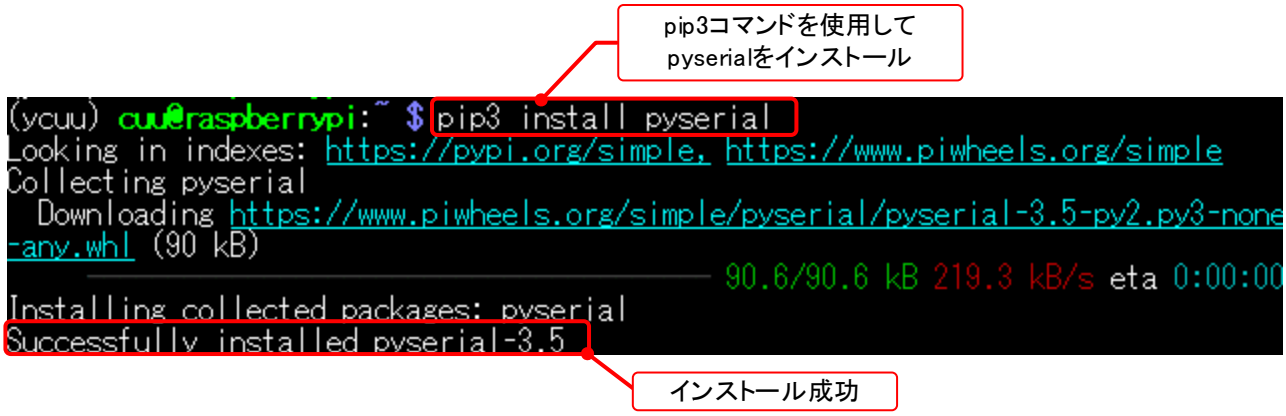


Copyright © 2024 YITJC All Rights Reserved.

手順③～pyserial インストール～

Raspberry Pi と LoRa 通信モジュール（ES920LR2）との間でシリアル通信を行うために必要なライブラリ「pyserial」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install pyserial



手順④～プログラム配置～

対象プログラムを「dev」ディレクトリにコピーします。（ドラッグ＆ドロップ）

[対象プログラム]

LoRa_Class.py

52_LoRa.py

～補足～

「LoRa_Class.py」は、Raspberry Pi と LoRa 通信モジュールとの間で行われるシリアル通信を管理するクラス「LoRa」を含むモジュール（プログラム）です。「52_LoRa.py」は、「LoRa_Class.py」で定義したクラス「LoRa」を使用して、LoRa 通信モジュールとの間でシリアル通信を行っています。

手順⑤～シャットダウンプログラムの更新～

「シャットダウン」および「再起動」の際に LoRa 通信モジュールをリセットする必要があるため、シャットダウンプログラムを更新してください。

[対象プログラム]

5_Shutdown_LoRa.py

～補足～

「5_Shutdown_LoRa.py」は、Raspberry Pi を「シャットダウン」および「再起動」する前に LoRa 通信モジュールをリセットする制御を追加しています。LoRa 通信モジュールをリセット制御する理由については、プログラム解説で説明します。

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

cuu@raspberrypi:~/dev $ chmod +x 5_Shutdown_LoRa.py

Copyright © 2024 YITJC All Rights Reserved.

手順⑥～Service ファイル編集～

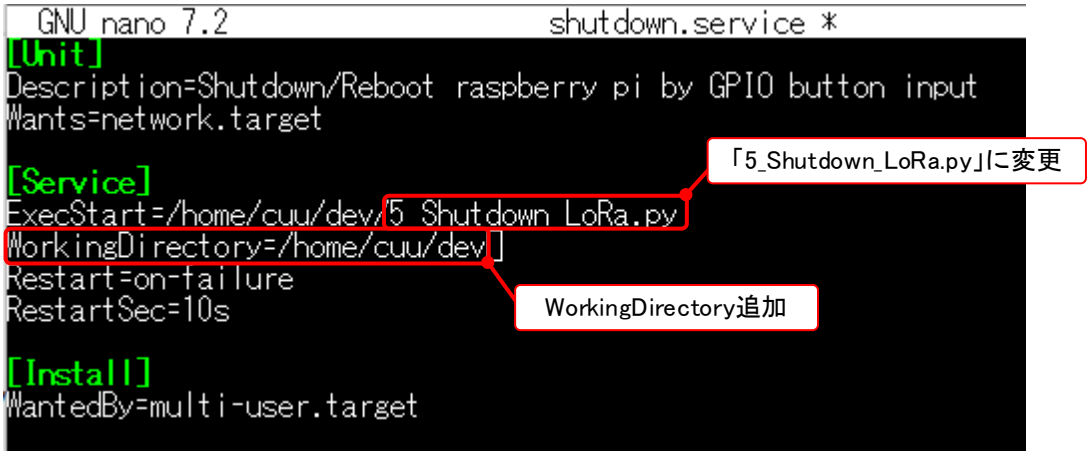
「shutdown.service」ファイルを編集するため、「/etc/systemd/system」ディレクトリへ移動します。

cuu@raspberrypi:~ $ cd /etc/systemd/system

```
cuu@raspberrypi:~ $ cd /etc/systemd/system
cuu@raspberrypi:/etc/systemd/system $
```

続いて、Unit ファイルを編集します。次のコマンドを実行してください。

cuu@raspberrypi:/etc/systemd/system $ sudo nano shutdown.service



編集した後、「Ctrl」+「o」で上書きファイルを確認し、「Enter」で上書きを実行します。

上書きした後は、「Ctrl」+「x」で画面を閉じます。

手順⑦～サービス再読み込み～

手順⑥で「shutdown.service」ファイルを編集した後は、次のコマンドを実行してサービスを再読み込みします。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl daemon-reload

手順⑧～サービス再起動～

次のコマンドを実行して「shutdown」サービスを再起動します。

(ycuu) cuu@raspberrypi:/etc/systemd/system $ sudo systemctl restart shutdown.service

手順⑨～プログラム実行権限追加～

「dev」ディレクトリへ移動して、プログラムが保存されていることを確認します。

続いて、実行権限を追加します。

(ycuu) cuu@raspberrypi:~/dev $ chmod +x 52_LoRa.py

手順⑩～プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./52_LoRa.py

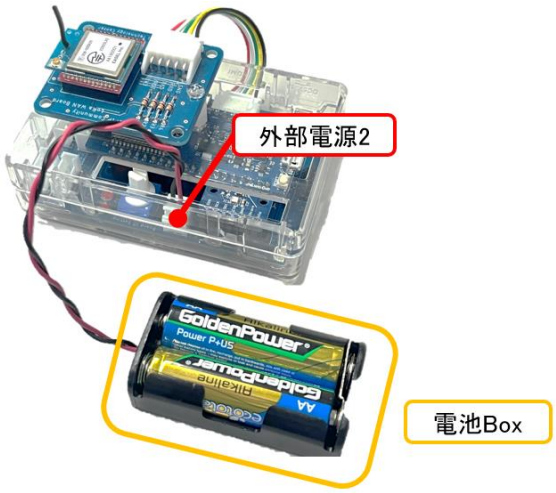
Copyright © 2024 YITJC All Rights Reserved.

LoRa 通信モジュールとの通信に成功すると、次のメッセージが表示され、Sensor Board からのデータ受信を待機します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./52_LoRa.py
b'¥r¥n'
b'Select Mode [1.terminal or 2.processor]¥r¥n'
LoRa module setup complete
```

手順⑪～電池 Box 接続～

Sensor Board の外部電源 2 に電池 Box を接続してください。LoRa 通信プログラムは、BME280（温湿度・気圧センサ）のデータと外部電源 2 に接続された電池 Box の残電圧データを Raspberry Pi に送信します。

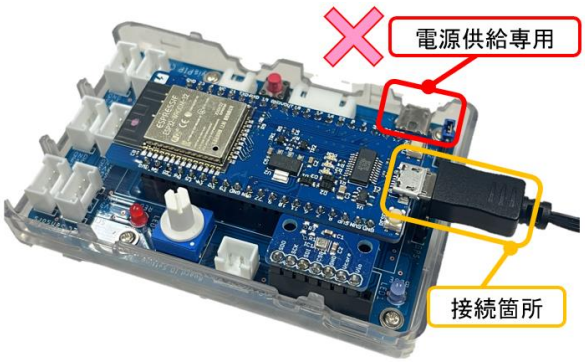


～補足～

Sensor Board は Micro USB 給電に加えて、2 ヵ所の給電端子を備えています。そのため、モバイルバッテリーや電池などを併用して給電できます。また、各給電ポートの状況は ESP32 が監視しているため、モバイルバッテリーの充電切れや電池の残量不足をリアルタイムで検出できます。

手順⑫～ESP32 接続～

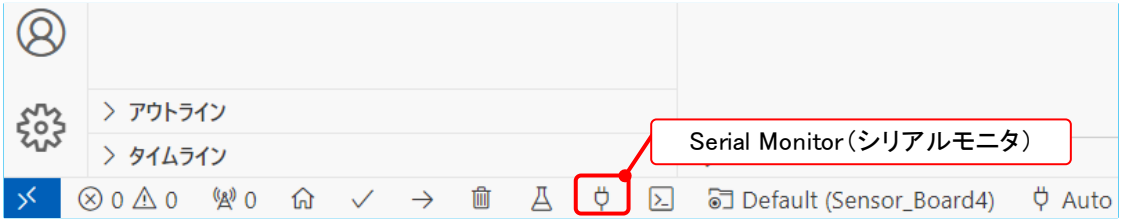
VS Code シリアルモニタを使用して Sensor Board の動作を確認するため、ESPr® Developer 32 に Micro USB ケーブルを接続してください。



Copyright © 2024 YITJC All Rights Reserved.

手順⑬～動作確認～

VS Code を開いて、画面下ステータスバーに表示されているメニューから「Serial Monitor（シリアルモニタ）をクリックしてください。



- Sensor Board（子機）の動作

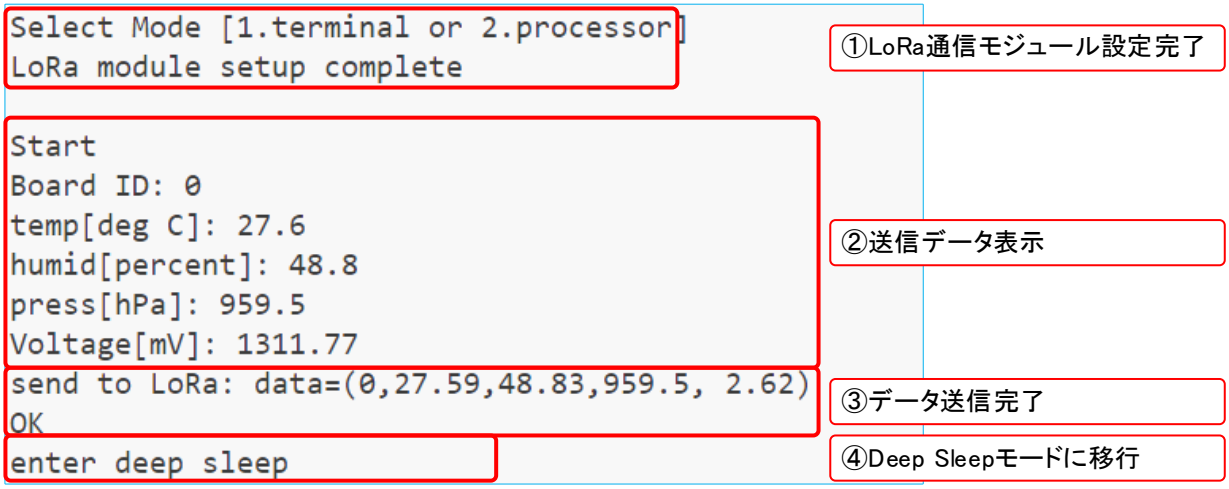
①～④の動作を 30 秒周期で繰り返し処理しています。

① LoRa 通信モジュールとシリアル通信を行い、動作モードを設定します。

② Raspberry Pi に送信するデータを print 表示します。

③ Raspberry Pi にデータを送信し、LoRa 通信モジュールから送信が正常に終了したことを表す信号「OK」を受信しています。

④ データ送信後、Deel Sleep モードに移行します。



参考～レスポンスコード～

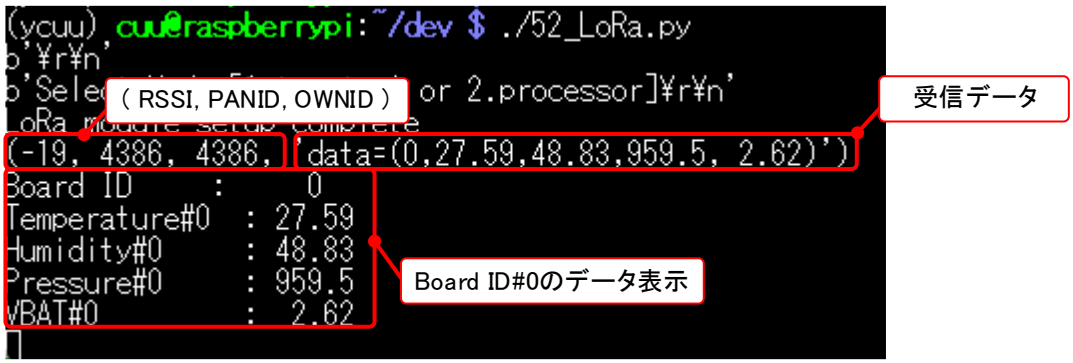
LoRa 通信モジュール（ES920LR2）の代表的なレスポンスコードを紹介します。

レスポンスコード	説明（エラー発生時の対策）
OK	正常応答
NG002	オプション値エラー（Operation モードで起動してから再度プログラム書き込み）
NG100	送信データ長エラー（送信データを 50byte 以下にしてください）
NG102	キャリアセンス検出 （LoRa 通信モジュールアンテナとケーブルなどの干渉を確認）
NG103	ACK 未受信（親機の電源、受信プログラムの動作を確認してください）
NG105	送信時間制限オーバー（4 秒以下になるように bw, sf の設定を見直してください）

Copyright © 2024 YITJC All Rights Reserved.

- Raspberry Pi（親機）の動作

Tera Term の画面を確認すると、Sensor Board からのデータを受信していることがわかります。



Raspberry Pi が受信したデータについて説明します。

(−19, 4386, 4386, 'data=(0, 27.59, 59.48, 959.5, 2.62)')

(RSSI, PANID, OWNID, 'data=(Board ID, Temperature , Humidity, Pressure, Battery)')

- RSSI（受信電波強度）
- PANID（Sensor Board PANID）
- OWNID（Sensor Board OWNID）
- Board ID（Sensor Board ロータリーSW の値）
- Temperature（BME280 センサ温度データ）
- Humidity（BME280 センサ湿度データ）
- Pressure（BME280 センサ気圧データ）
- Battery（Sensor Board 外部電源 2 の残電圧）

～プログラム解説～

行	プログラム
1	#!/home/cuu/ycuu/bin/python3.11
2	# coding:utf-8
3	
4	import time
5	import RPi.GPIO as GPIO
6	import LoRa_Class
7	import ast
8	import struct
9	import sys
10	
11	# Sets up pins as outputs
12	def setup_led():
13	GPIO.setmode(GPIO.BCM)
14	GPIO.setup(26, GPIO.OUT)
15	GPIO.output(26, GPIO.LOW)

Copyright © 2024 YITJC All Rights Reserved.

```
16
17  lora = LoRa_Class.LoRa()
18
19  def printable(l):
20      return ''.join(chr(x) for x in struct.unpack(f'{len(l)}b', l) if x >= 0)
21
22  def sendcmd(cmd):
23      # print(cmd)
24      lora.write(cmd)
25      start_time = time.time()
26      while (True):
27          if (time.time() - start_time) > 5:
28              print('Timeout: %s' % cmd)
29              exit()
30          line = lora.readline()
31          if 'OK' in printable(line):
32              # print(line)
33              return True
34          elif 'NG' in printable(line):
35              # print(line)
36              return False
37
38  def setMode():
39      lora.write('config¥r¥n')
40      lora.s.flush()
41      time.sleep(0.2)
42      lora.s.flushInput()
43      lora.reset()
44      time.sleep(1.5)
45
46      line = lora.readline()
47      print(line)
48      while not ('Mode' in printable(line)):
49          line = lora.readline()
50          if len(line) > 0:
51              print(line)
52
53      sendcmd('2¥r¥n')              #1:terminal/2:processor
54      sendcmd("node 1¥r¥n")         #1:Coordinator/2:EndDevice/3:Router
```

Copyright © 2024 YITJC All Rights Reserved.

```
55        sendcmd("panid 1122¥r¥n")    #PAN Network Address
56        sendcmd("ownid 0000¥r¥n")    #自ノード Network Address
57        sendcmd("dstid 0000¥r¥n")    #送信先 Network Address
58        sendcmd("rssi 1¥r¥n")        #受信電波強度付与 1:ON/2:OFF
59        sendcmd("rcvid 1¥r¥n")       #データ受信時相手ノードのNetwork Address付与
60        sendcmd("bw 4¥r¥n")           #帯域幅:125kHz
61        sendcmd("sf 7¥r¥n")           #拡散率: 7
62        sendcmd("channel 1¥r¥n")     #無線channel: 920.6MHz
63        #Config Modeで起動するための設定（n対1通信:親機取りこぼし検出のため）
64        sendcmd('q 1¥r¥n')            #1:Congig Mode/2:Operation Mode
65        sendcmd('w¥r¥n')              #saveコマンド
66        sendcmd('start¥r¥n')          #Operation モードに移行
67        lora.readline()               #移行完了を待機
68        print('LoRa module setup complete')
69        time.sleep(1)
70        sys.stdout.flush()
71
72    if __name__ == '__main__':
73        try:
74            setup_led()
75            setMode()
76            while (True):
77                time.sleep(2)
78                while (True):
79                    line = lora.readline(None)
80                    #RSSI(4byte)-PANID(4byte)-OWNID(4bytye)-¥r¥n(2byte)
81                    if len(line) >= 14:
82                        break
83                #RSSI(4byte)-PANID(4byte)-OWNID(4bytye)-Payload
84                data = lora.parse(line)
85                print(data)
86
87                if 'data=' in data[3]:
88                    payload = ast.literal_eval(data[3].split('=')[1])
89                    rssi = data[0]
90                    ID = payload[0]
91                #print(rssi)
92                print("Board ID        : {:5} ".format(ID))
93
```

Copyright © 2024 YITJC All Rights Reserved.

```
94                  if ID == 0:
95                      Temp0 = payload[1]
96                      Hum0 = payload[2]
97                      Press0 = payload[3]
98                      VBAT0 = payload[4]
99
100                     print("Temperature#0   : {:5} ".format(Temp0))
101                     print("Humidity#0      : {:5} ".format(Hum0))
102                     print("Pressure#0      : {:5} ".format(Press0))
103                     print("VBAT#0            : {:5} ".format(VBAT0))
104                     sys.stdout.flush()
105
106                 elif ID == 1:
107                     Temp2 = payload[1]
108                     Hum2 = payload[2]
109                     Press2 = payload[3]
110                     VBAT2 = payload[4]
111
112                     print("Temperature#1   : {:5} ".format(Temp1))
113                     print("Humidity#1      : {:5} ".format(Hum1))
114                     print("Pressure#1      : {:5} ".format(Press1))
115                     print("VBAT#1            : {:5} ".format(VBAT1))
116                     sys.stdout.flush()
117
118                 time.sleep(1.0)
119                 GPIO.output(26, GPIO.HIGH)
120                 time.sleep(1.0)
121                 GPIO.output(26, GPIO.LOW)
122                 time.sleep(1.0)
123         lora.reset()
124     except KeyboardInterrupt:
125         lora.reset()
```

灰色部は、これまでに説明した内容です。

- 1 行目: Shebang（シバン、シェバン）
- 2 行目: マジックコメント
- 4 行目: Sleep 制御するため time ライブラリをインポート
- 5 行目: GPIO を制御するため RPi.GPIO ライブラリをインポート
- 6 行目: LoRa 通信モジュールとシリアル通信を行うために必要な「LoRa_Class」モジュールをインポート

Copyright © 2024 YITJC All Rights Reserved.

- 7 行目: 文字列を Python オブジェクトに変換するために必要なライブラリ「ast」をインポート
  ast は、Python の抽象構文木（AST: Abstract Syntax Tree）を扱うための標準ライブラリです。
- 8 行目: バイナリデータを文字列に変換するために必要なライブラリ「struct」をインポート
- 9 行目: Python のシステム関連の機能を使用するために必要なライブラリ「sys」をインポート
- 12 行目～15 行目: GPIO 初期設定プログラム setup_led()関数
  青色 LED（GPIO26）の初期設定を行っています。
- 17 行目: LoRa_Class モジュールの「LoRa」クラスを使用してインスタンス「lora」を作成

～受信データ文字列変換～

- 19 行目～20 行目: バイナリデータを文字列に変換するためのプログラム printable()関数
  `struct.unpack(f'{len(l)}b', l)`
  与えられたバイト列「l」を、符号付き 8 ビット整数(b)としてアンパック（デコード）します。
  各バイトが符号付き整数としてアンパックされます。
  `(chr(x) for x in struct.unpack(f'{len(l)}b', l) if x >= 0)`
  アンパックされた各整数 x に対して、条件「x >= 0」を満たすものだけを取り出し、それを chr(x)で文字列に変換します。
  `join(chr(x) for x in struct.unpack(f'{len(l)}b', l) if x >= 0)`
  `join` によって変換された文字列を一つの文字列に連結します。
  これにより、バイト列「l」の各バイトが文字列に変換され、それらが連結されて一つの文字列になります。
  例えば、バイト列「l」が [72, 101, 108, 108, 111] だった場合
  struct.unpack(f'{len(l)}b', l) によって [72, 101, 108, 108, 111] という整数のリストが作成されます。
  そして、各整数は chr(x)によって文字列に変換され、最終的に'Hello'という文字列が得られます。

～LoRa コマンド送信～

- 22 行目～36 行目: LoRa 通信モジュールにコマンドを送信するためのプログラム sendcmd()関数
- 24 行目: lora.write()関数を使用してコマンド「cmd」を LoRa 通信モジュールに送信
- 25 行目: 現在の時刻情報を変数「start_time」に格納
- 26 行目～36 行目: LoRa 通信モジュールからの応答（OK or NG）が返るまで繰り返し処理
- 27 行目: コマンド送信開始から 5 秒以上経過した場合の処理
- 28 行目～29 行目: 「Timeout」およびコマンド「cmd」を print 表示してからプログラムを終了
- 30 行目: lora.readline()関数を使用して LoRa 通信モジュールからの応答を受信
- 31 行目～36 行目: LoRa 通信モジュールからの応答処理
  応答に「OK」が含まれていれば「True」を返し、「NG」が含まれていれば「False」を返しています。

～LoRa 動作モード設定～

- 38 行目～69 行目: LoRa 通信モジュールに動作モードを設定するためのプログラム setMode()関数
- 39 行目: 「config」コマンドを送信してコンフィグレーションモードに移行
- 40 行目: 送信バッファクリア
- 41 行目: 0.2 秒待機
- 42 行目: 受信バッファクリア
- 43 行目: lora.reset()関数を使用して LoRa 通信モジュールをリセット
- 44 行目: 1.5 秒待機

Copyright © 2024 YITJC All Rights Reserved.

- 46 行目～51 行目: LoRa 通信モジュールからの応答を受信したときの処理
- 46 行目: lora.readline()関数を使用して LoRa 通信モジュールからの応答を受信
- 47 行目: LoRa 通信モジュールからの応答を print 表示
  次の内容が Tera Term に出力されます。
  b'¥r¥n'
- 48 行目～51 行目: LoRa 通信モジュールからの応答に「Mode」が含まれるまで繰り返し処理
- 51 行目: LoRa 通信モジュールからの応答に「Mode」が含まれていた場合の処理
  「Mode」が含まれていた場合、次の内容が Tera Term に出力されます。
  b'Select Mode [1.terminal or 2.processor]¥r¥n'
- 53 行目～65 行目: LoRa 通信モジュールにコマンドを送信
- 53 行目: プロセッサモード「2」を送信
- 54 行目: node コマンドを使用して「1 Coordinator」を送信
- 55 行目: panid コマンドを使用して PAN ネットワークアドレス「1122」を送信
  Sensor Board（子機）と同じ値を設定します。
- 56 行目: ownid コマンドを使用して自ノードネットワークアドレス「0000」を送信
  Coordinator の場合は、ownid を「0000」に設定します。
- 57 行目: dstid コマンドを使用して送信先ノードネットワークアドレス「0000」を送信
  親機（Coordinator）のため、送信先ノードネットワークアドレスにはデフォルト値「0000」を設定します。
- 58 行目: rssi コマンドを使用して受信電波強度（RSSI）付与設定「1 ON」を送信
- 59 行目: rcvid コマンドを使用して相手ノードネットワークアドレス付与設定「1 ON」を送信
- 60 行目: bw コマンドを使用して帯域幅設定「4 125kHz」を送信
  Sensor Board（子機）と同じ値を設定します。
- 61 行目: sf コマンドを使用して拡散率「7」を送信
  Sensor Board（子機）と同じ値を設定します。
- 62 行目: channel コマンドを使用して無線チャンネル番号設定「1」を送信
  Sensor Board（子機）と同じ値を設定します。
- 64 行目: q コマンドを使用して動作モード「1 Configuration Mode」を送信

注意～データ取りこぼし対策～

LoRa 通信モジュールが Operation モードで動作している場合、次のケースで「データの取りこぼし」が発生します。

（取りこぼしが発生するケース）

LoRa 受信プログラムの自動起動より先に、LoRa 通信モジュールが Operation モードで起動しているケースは、「親機がデータ受信していないにもかかわらず、ACK を応答する」問題が発生します。

子機は親機から「ACK」を受信するまで「再送」処理を行いますが、「ACK」を受信したあとは、親機がデータを受信したと判断して次のデータを送信します。つまり、親機側で「データの取りこぼし」が発生します。

この「ケース」は親機に電源が供給された直後に発生するため、対策として、親機が必ず Configuration モードで起動する設定にしています。LoRa 通信モジュールが Configuration モードで起動した場合、LoRa 受信プログラムが動作するまで、「ACK」を応答することはありません。

Copyright © 2024 YITJC All Rights Reserved.

（対策）

Configuration モードで起動するため、64 行目で「1 Configuration Mode」を設定してから保存（65 行目 w コマンド）しています。また、Configuration モードで起動するためには、プログラムを終了する際に必ず Lora 通信モジュールをリセットする必要があります。「52_LoRa.py」はプログラム終了時に必ず LoRa 通信モジュールをリセットする構成にしていますが、Raspberry Pi が「シャットダウン」または「再起動」した場合も LoRa 通信モジュールをリセットするため、手順⑤で「シャットダウンプログラムの更新」を行いました。

- 65 行目: w コマンドを使用して EEPROM への設定値書き込みを指示
- 66 行目: start コマンドを使用して Operation モードへ移行
- 67 行目: LoRa 通信モジュールからの応答を受信して Operation モードへの移行を確認
- 68 行目:設定完了メッセージ「LoRa module setup complete」をシリアルモニタに表示
- 69 行目: 1 秒待機
- 70 行目: Python 標準出力バッファをフラッシュ

～メイン関数～

- 73 行目～123 行目: try ブロック
- 74 行目: GPIO 初期設定プログラム setup_led()関数実行
- 75 行目: LoRa 通信モジュールに動作モードを設定するためのプログラム setMode()関数実行
- 76 行目～122 行目: 繰り返し処理
- 77 行目: 2 秒待機
- 78 行目～82 行目: データ受信処理（繰り返し処理）
- 79 行目: lora.readline()関数を使用して LoRa 通信モジュールからの応答を受信
- 81 行目～82 行目: 受信データが 14byte 以上の場合繰り返し処理（While True）を抜ける
  14byte の内訳は、rssi（4bite）、panid（4byte）、ownid（4byte）、改行コード（2byte）です。
- 84 行目: LoRa 通信モジュールから受信したデータを変数「data」に格納
- 85 行目: 受信したデータ「data」を print 表示
- 87 行目～91 行目: 受信したデータに「data=」が含まれていた場合の処理
- 87 行目: 受信データの配列 data[3]に「data=」が含まれているか判定
  受信データ data[]の配列構成は、次のとおり。
  data[0]: rssi（受信電波強度）
  data[1]: panid
  data[2]: ownid
  data[3]: 'data=(Board ID, Temperature , Humidity, Pressure, Battery)'
- 88 行目: 受信データの文字列からデータを抽出して Python オブジェクトに変換
  「data=」のあとに続く部分を取り出し、Python オブジェクト「payload」に変換しています。
  'data=(Board ID, Temperature , Humidity, Pressure, Battery)'
- 89 行目: data[0]にある受信電波強度の値を変数「rssi」に格納
- 90 行目: payload[0]にある Sensor Board の識別 ID を変数「ID」に格納
- 92 行目: Sensor Board の識別 ID を print 表示

Copyright © 2024 YITJC All Rights Reserved.

- 94 行目～104 行目: Board ID が「0」の場合の処理
- 106 行目～116 行目: Board ID が「1」の場合の処理
- 118 行目～122 行目: データ受信完了を表す LED（青色）点灯
- 123 行目: lora.reset()関数を使用して LoRa 通信モジュールをリセット
- 124 行目～125 行目 : except ブロック
  KeyboardInterrupt 例外を定義しています。
  この例外は、「Ctrl」＋「c」を押すことで発生します。通常はプログラムを終了させる目的で使用します。
  「Ctrl」＋「c」が押されたとき、lora.reset()関数を使用して LoRa 通信モジュールをリセットします。

Copyright © 2024 YITJC All Rights Reserved.

11.9 LoRa 通信パラメータが与える影響

本講座で構築した「プライベート Lora」ネットワークは、屋内環境および屋外環境の両方で広範囲の通信が可能です。通信可能な範囲は、帯域幅（bw）および拡散率（sf）の通信パラメータだけでなく、建物の構造や地形などの通信環境にも大きく影響を受けます。そのため、プライベート LoRa ネットワークを使用した IoT システムを構築する場合は、必ず実際の通信環境で受信電波強度（rssi）を確認する通信テストを実施してください。

- **通信パラメータ～帯域幅（Band Width）～**

帯域幅を広く設定すると、伝送速度（bps）が向上します。

例えば、帯域幅を 500kHz に設定して 1MHz の中心周波数で通信する場合、使用可能な周波数範囲は 0.75MHz（1MHz-250kHz）～1.25MHz（1MHz+250kHz）になります。つまり、帯域幅を設定しない場合（1MHz）に比べて、500kHz 広い周波数範囲で信号を伝送できるため、伝送速度が向上します。

- **通信パラメータ～拡散率（Spread Factor）～**

LoRa 通信は、無線変調技術に「チャープ・スペクトラム拡散」を使用しています。拡散率（sf）は、「チャープ・スペクトラム拡散」の拡散率を設定します。拡散率（sf）の値を大きく設定すると、受信感度が向上します。そのため、建物の構造や地形の影響により受信電波強度（rssi）を確保できない通信環境では、拡散率（sf）の設定が通信に与える影響が大きいです。しかし、拡散率（sf）の値を大きく設定すると、伝送速度（bps）が低下するため、送信時間に注意する必要があります。

注意～送信時間制限（4 秒）～

LoRa 通信（920MHz 帯）を利用する場合、「ARIB STD-T108」と呼ばれる標準規格を順守する必要があります。この規格は、電波法に基づいて技術基準を定義しており、最大送信時間が 4 秒以下と規定しています。つまり、LoRa 通信モジュールを使用して 4 秒以上の通信を行った場合、電波法違反となります。これを回避するため、LoRa 通信モジュール（ES920LR2）は、4 秒を超えるデータ送信となる場合は、レスポンスコード NG105（送信時間制限オーバー）でエラーになります。

- **通信パラメータが伝送速度に与える影響**

LoRa 通信は、帯域幅（bw）および拡散率（sf）の設定により、伝送速度（bps）と通信距離が変動します。伝送速度と通信距離はトレードオフの関係になります。例えば、建物の構造や地形の影響で通信環境が悪い場合、通信距離（受信電波強度）を確保するために拡散率（sf）を高く設定すると、その影響で伝送速度は低くなります。

そのため、実際に使用する通信環境（通信距離）で伝送速度を考慮に入れた通信パラメータ（帯域幅、拡散率）の設定が重要です。

通信パラメータの設定による伝送速度（bps）の違いを次の表に示します。（参照：ES920LR2 データシート）

伝送速度 [bps]		sf							
		5	6	7	8	9	10	11	12
bw	62.5kHz	7813	4688	2734	1563	879	488	269	146
	125kHz	15625	9375	5469	3125	1758	977	537	293
	250kHz	31250	18750	10938	6250	3516	1953	1074	586
	500kHz	62500	37500	21875	12500	7031	3906	2148	1172

Copyright © 2024 YITJC All Rights Reserved.

続いて、50byte のデータを送信するために必要な時間（msec）を表に示します。

帯域幅（bw） 62.5kHz、拡散率（sf） 12 の設定では、送信時間が 5,587msec となり、電波法で規定されている 4 秒を超えてしまうことがわかります。

送信時間 [msec]		sf							
		5	6	7	8	9	10	11	12
bw	62.5kHz	121	187	308	534	903	1642	2957	5587
	125kHz	61	93	154	267	452	821	1479	2793
	250kHz	30	47	77	133	226	411	739	1397
	500kHz	15	23	38	67	113	205	370	698

参考～通信パラメータの設定～

プライベート LoRa 通信を行う場合、親機と子機で通信パラメータの設定に注意してください。

- 親機（Raspberry Pi） － 子機（Sensor Board）共通の設定項目
  - ・bw（帯域幅）
  - ・sf（拡散率）
  - ・channel（チャンネル）
  - ・pan（PAN ネットワークアドレス）
- 子機（Sensor Board）専用の設定項目
  - ・ownid（自ノードネットワークアドレス）
  - ※プライベート LoRa 通信ネットワークに子機が複数存在する場合は、必ず子機ごとにユニークな値（重複しない値）を設定してください。
  - ・dstid（送信先ノードネットワークアドレス）
  - ※Coordinator に送信する場合は、「0000」を設定してください。

プライベート LoRa 通信で設定する通信パラメータを紹介します。

コマンド	ショートコマンド	設定内容	初期設定	設定
bw	b	帯域幅 Band Width 設定	4 （125kHz）	3 62.5kHz 4 125kHz 5 250kHz 6 500kHz
sf	c	拡散率 Spread Factor 設定	7	7～12
channel	d	無線チャンネル番号設定	1	bw 125kHz 以下: 1～38 bw 250kHz: 1～19 bw 500kHz: 1～12
panid	e	PAN ネットワークアドレス設定	0x0001	0x0001～0xFFFE
ownid	f	自ノードネットワークアドレス設定	0x0001	0x0001～0xFFFE
dstid	g	送信先ノードネットワークアドレス設定	0x0000	0x0000～0xFFFF

Copyright © 2024 YITJC All Rights Reserved.

参考～LoRa 通信コマンド～

LoRa 通信モードの設定で主に使用するコマンドを紹介します。

コマンド	ショートコマンド	設定内容	初期設定	設定
node	n	ノード種別設定	EndDevice	1 Coordinator（親機） 2 EndDevice（子機） 3 Router（中継機）
ack	l	Acknowledge 設定	ON	1 ON 2 OFF
retry	m	リトライ回数設定	3	0～10
rcvid	o	相手ノード ネットワークアドレス付与設定	OFF	1 ON 2 OFF
rssi	p	受信電波強度付与設定	OFF	1 ON 2 OFF
operation	q	動作モード設定	Configuration	1 Configuration 2 Operation
baudrate	r	UART 通信速度設定	115200bps	1 9600bps 2 19200bps 3 38400bps 4 57600bps 5 115200bps 6 230400bps
sleep	s	スリープモード設定	No Sleep	1 No Sleep 2 Timer Wakeup 3 INT Wakeup (TX continue) 4 INT Wakeup (One Time Tx) 5 UART Wakeup
show	y	設定値確認	–	–
start	z	Operation モード移行	–	–
save	w	EEPROM への設定値書き込み	–	–

Copyright © 2024 YITJC All Rights Reserved.

参考～LoRa 通信テスト（屋内）～

屋内環境で LoRa 通信テストを実施する手順を紹介します。

手順①～子機準備～

共有サイトから対象のプロジェクトをダウンロードして、子機（Sensor Board）に書き込んでください。

[対象プロジェクト]

Sensor_Board5

[プログラム内容]

LoRa 通信を使用して、BME280（温湿度・気圧センサ）および外部電源 2 のデータを親機（Raspberry Pi）に送信します。

[プログラム編集個所]

通信パラメータを変更する場合は、必ず親機の設定と合わせてください。

```
#104行目　帯域幅（bw）
sendcmd("bw 4¥r¥n");
#105行目　拡散率（sf）
sendcmd("sf 7¥r¥n");
```

手順②～親機準備～

共有サイトから対象のプログラムをダウンロードして、親機（Raspberry Pi）に実装（配置および実行権限追加）してください。

[対象プログラム]

53_LoRa.py

[プログラム内容]

受信電波強度（rssi）および子機（Sensor Board）から受信したデータを指定した LINE ユーザー宛に送信します。

[プログラム編集個所]

```
#27行目　ユーザーID
user_id_1 = 'U9ebf2641xxxxx3464455002d83'
```

通信パラメータを変更する場合は、必ず子機の設定と合わせてください。

```
#101行目　帯域幅（bw）
sendcmd("bw 4¥r¥n")
#102行目　拡散率（sf）
sendcmd("sf 7¥r¥n")
```

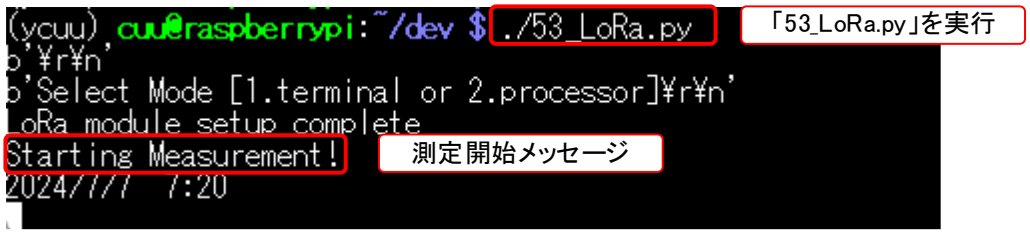
手順③～親機プログラム実行～

対象プログラムを実行します。

```
(ycuu) cuu@raspberrypi:~/dev $ ./53_LoRa.py
```

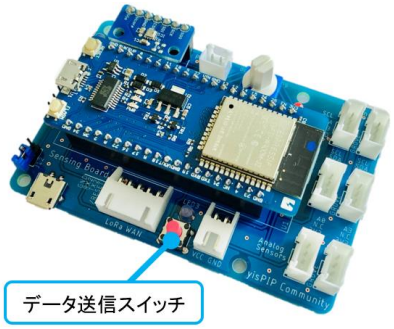
Copyright © 2024 YITJC All Rights Reserved.

「測定開始メッセージ」が表示され、子機（Sensor Board）からのデータ受信を待機します。

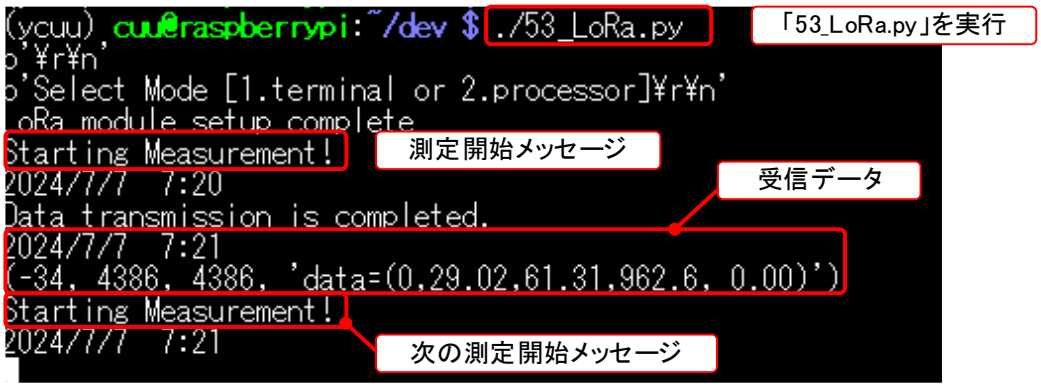


手順④～子機データ送信～

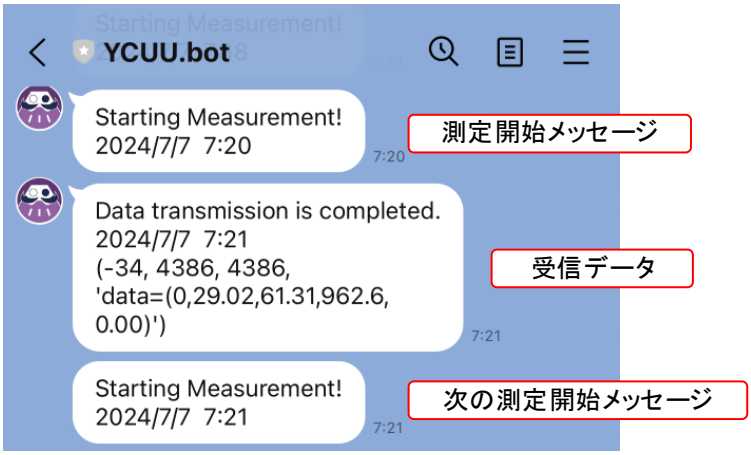
子機（Sensor Board）は、モバイルバッテリーや電池（外部電源ポート）から電源を供給してください。通信テストを実施する場所で、Sensor Board の「データ送信スイッチ」を押すと、Raspberry Pi にデータが送信されます。



Raspberry Pi がデータを受信すると、受信電波強度（rssi）や受信データの情報が表示されます。



LINE ユーザー宛に、次のメッセージが送信されます。



Copyright © 2024 YITJC All Rights Reserved.

参考～LoRa 通信テスト（屋外）～

屋内環境で LoRa 通信テストを実施する手順を紹介します。

手順①～子機準備～

共有サイトから対象のプロジェクトをダウンロードしてください。

[対象プロジェクト]

Sensor_Board6

[プログラム内容]

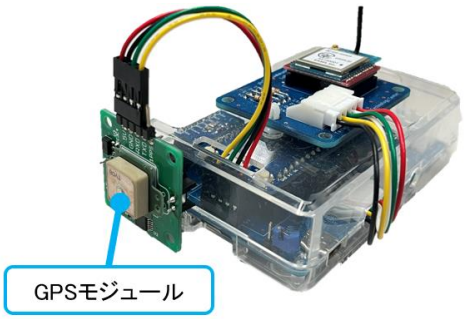
GPS モジュールを使用して子機（Sensor Board）の位置情報（緯度・経度）を取得します。GPS モジュールには、日本の準天頂衛星システム（QZSS）「みちびき」3 機受信に対応し、小型高感度が特徴の「GYSFDMAXB（太陽誘電製）」を使用します。ESP32 と GPS モジュールの間は UART 通信を行い、GPS モジュールが取得した位置情報を ESP32 が LoRa 通信を使用して Raspberry Pi にデータ送信します。

[通信パラメータ]

屋外環境における最適な通信パラメータ（帯域幅および拡散率）を見出すため、すべての組合せ（全 24 パターン）で送信します。

帯域幅（bw）　：62.5kHz、125kHz、250kHz、500kHz（4 パターン）

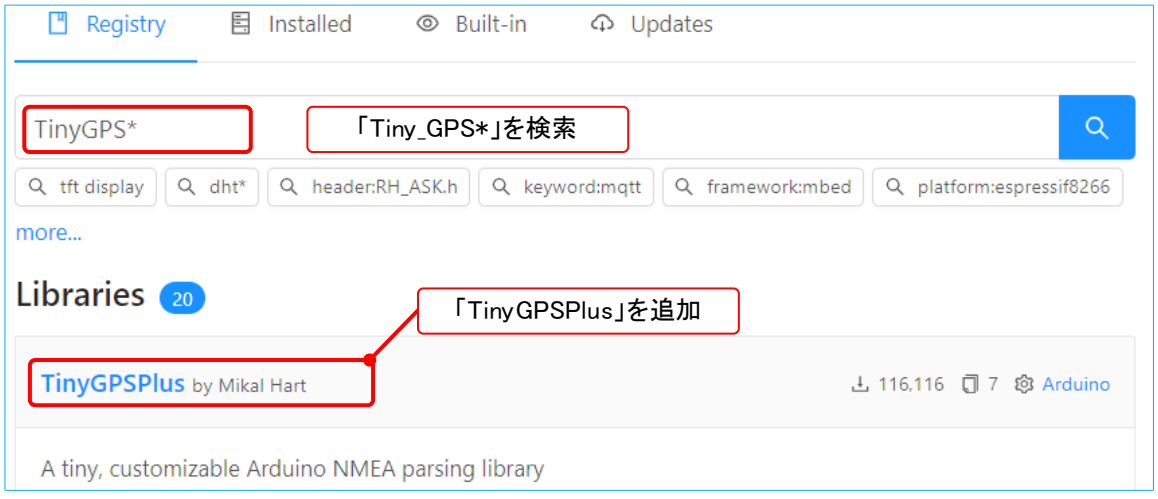
拡散率（sf）　：7～12（6 パターン）



手順②～ライブラリ追加～

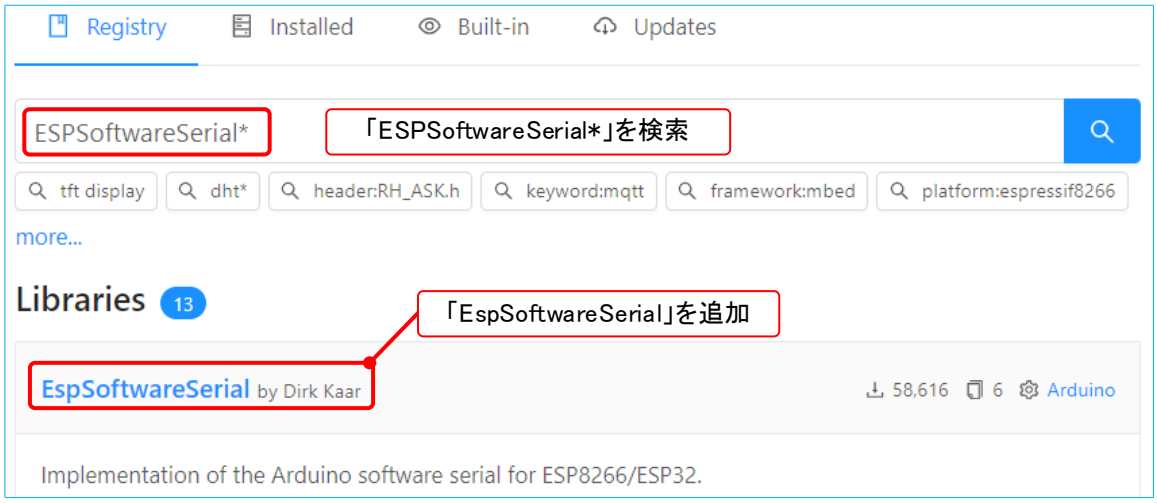
「Sensor_Board6」プロジェクトに必要なライブラリをインストールしてください。

まず、GPS モジュールから位置情報を取得するために必要なライブラリ「TinyGPSPlus」を追加してください。



Copyright © 2024 YITJC All Rights Reserved.

続いて、ESP と GPS モジュールの間で UART 通信を行うために必要なライブラリ「EspSoftwareSerial」を追加してください。



最後に、「platformio.ini」ファイルを確認して、シリアルモニタの速度設定を行ってください。

```
11  [env:esp32dev]
12  platform = espressif32
13  board = esp32dev
14  framework = arduino
15  monitor_speed = 115200
16  lib_deps =
17      mikalhart/TinyGPSPlus@^1.1.0
18      plerup/EspSoftwareSerial@^8.2.0
19
```

このあと、データ受信側（Raspberry Pi）の開発を行うため、プログラムの書き込みが終了したら、Sensor Board の電源を切ってください。

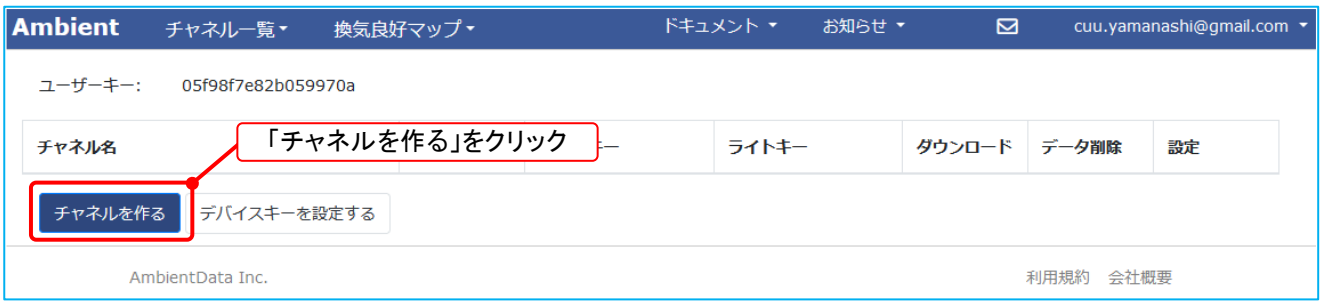
手順③～Ambient 環境設定～

子機（Sensor Board）から受信した位置情報は、IoT データ可視化サービス「Ambient」を使用してオンライン地図に記録します。

次の URL をクリックし、Ambient にログインしてください。

[URL] https://ambidata.io/usr/signup.html

「チャネルを作る」をクリックして、「チャネル」を 4 つ作成してください。



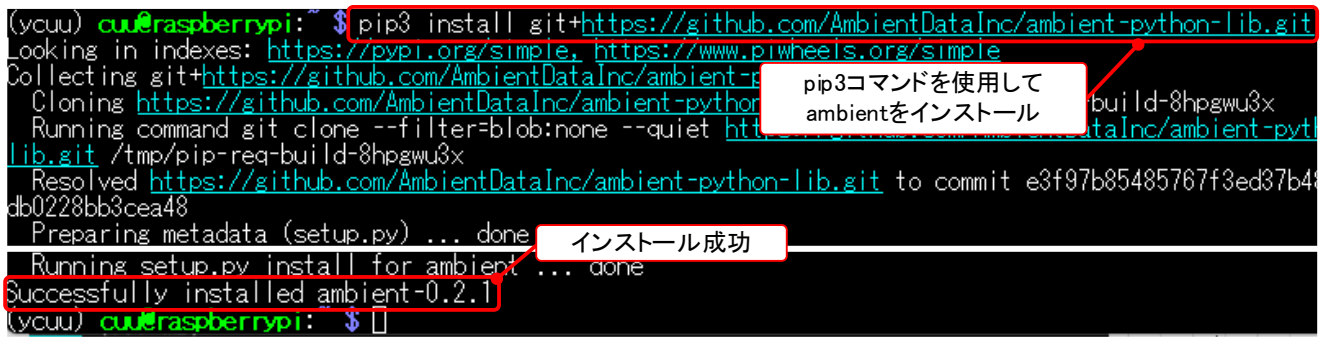
Copyright © 2024 YITJC All Rights Reserved.

「チャネル ID」、「ライトキー」の情報が表示さます。プログラムで使用するので、メモ帳などに記録してください。

手順④～Ambient インストール～

IoT データ可視化サービス「Ambient」に位置情報を記録するために必要なライブラリ「ambient」をインストールします。

(ycuu) cuu@raspberrypi:~ $ pip3 install git+https://github.com/AmbientDataInc/ambient-python-lib.git



手順⑤～親機準備～

共有サイトから対象のプログラムをダウンロードして、親機（Raspberry Pi）に実装（配置および実行権限追加）してください。

[対象プログラム]

54_LoRa.py

[プログラム内容]

子機（Sensor Board）から受信した位置情報（緯度・経度）および受信電波強度（rssi）のデータを IoT データ可視化サービス「Ambient」に送信し、オンライン地図に記録します。

また、通信パラメータ（帯域幅および拡散率）すべての組合せ（全 24 パターン）における受信電波強度（rssi）を LINE ユーザー宛に送信します。

[プログラム編集個所]

```
#28行目 ユーザーID
user_id_1 = 'U9ebf2641xxxxx3464455002d83'
```

Copyright © 2024 YITJC All Rights Reserved.

Ambient の設定情報は、手順③で取得した情報を記載してください。

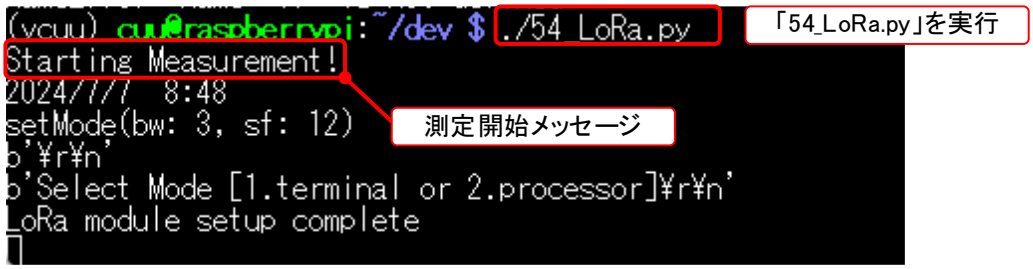
```
#Ambient Settings
channelId1 = 8xxx6
writeKey1 = '2a8dbxxxxx72ab77'
channelId2 = 8xxx7
writeKey2 = '4523cxxxxxbc3773'
channelId3 = 8xxx8
writeKey3 = 'ae885xxxxx5d065d'
channelId4 = 8xxx9
writeKey4 = '7b519xxxxxa0e6b7'
```

手順⑥～親機プログラム実行～

対象プログラムを実行します。

(ycuu) cuu@raspberrypi:~/dev $ ./54_LoRa.py

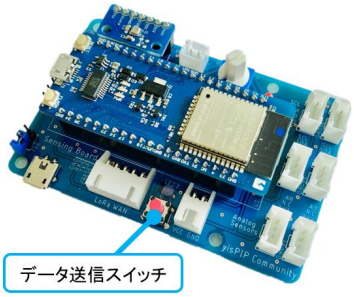
「測定開始メッセージ」が表示され、子機（Sensor Board）からのデータ受信を待機します。



手順⑦～子機データ送信～

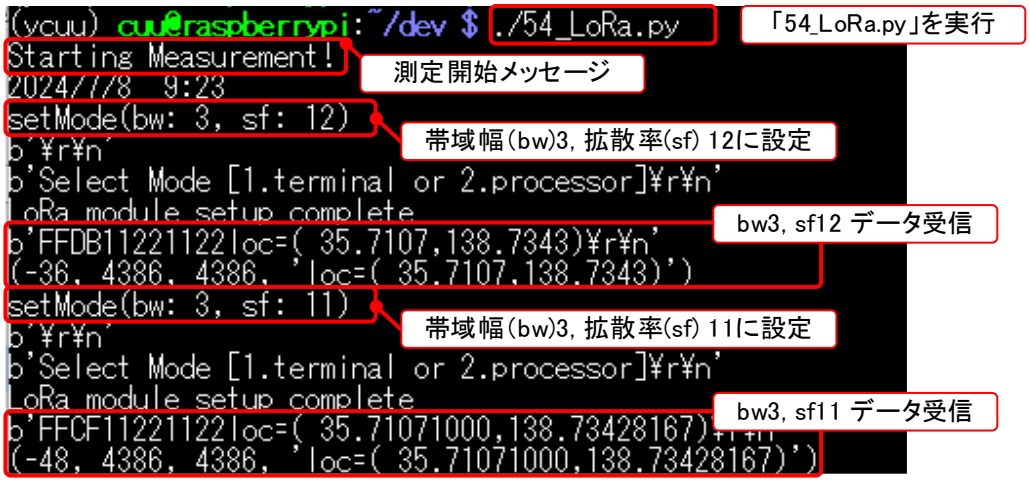
子機（Sensor Board）は、モバイルバッテリーや電池（外部電源ポート）から電源を供給してください。アンテナが上空を向くように設置すると、数分で準天頂衛星システム（QZSS）の追尾を開始し、GPS モジュールに実装されている受信確認 LED（赤色）が 1 秒周期で点滅します。

通信テストを実施する場所で、Sensor Board の「データ送信スイッチ」を押すと、Raspberry Pi に位置情報が送信されます。

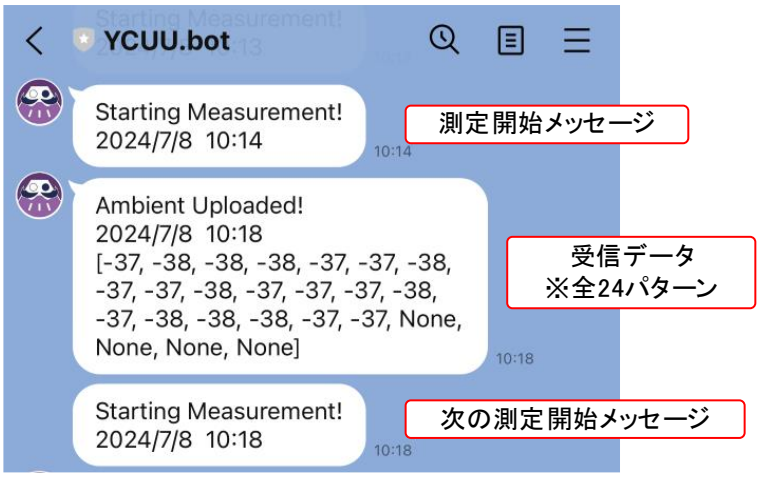


Copyright © 2024 YITJC All Rights Reserved.

Raspberry Pi がデータを受信すると、受信電波強度（rssi）や受信データの情報が表示されます。



通信パラメータ（帯域幅および拡散率）すべての組合せ（全 24 パターン）のデータを受信したあと、LINE ユーザー宛に、次のメッセージが送信されます。



手順⑧～通信範囲確認～

子機（Sensor Board）から受信した位置情報は、IoT データ可視化サービス「Ambient」を使用してオンライン地図で確認できます。

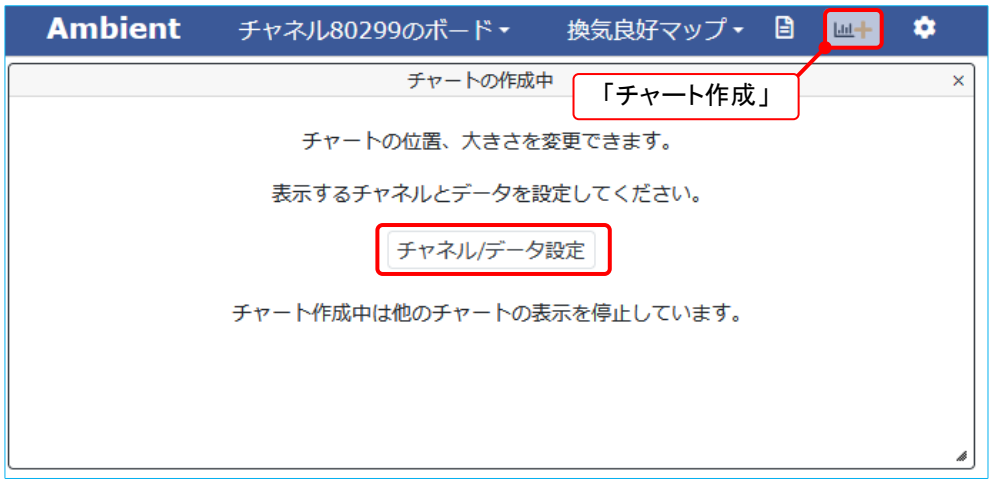
次の URL をクリックし、Ambient にログインしてください。

[URL] https://ambidata.io/usr/login.html

手順③で作成した「チャネル名」をクリックしてください。

Copyright © 2024 YITJC All Rights Reserved.

「チャート作成」アイコンをクリックしたあと、「チャネル／データ設定」をクリックしてください。



チャート種類から「地図」を選択してください。

続いて、d1～d6 を「表示」に設定したあと、「設定する」をクリックしてください。

～補足～

d1～d6 は、拡散率（sf）の値 7～12 に対応しています。

Copyright © 2024 YITJC All Rights Reserved.

手順③で作成した 4 つのチャネルには、帯域幅（bw）ごとの結果が記録されています。

地図上で測定箇所にカーソルを合わせると、拡散率（sf）ごとの受信電波強度が記録されていることを確認できます。

Copyright © 2024 YITJC All Rights Reserved.

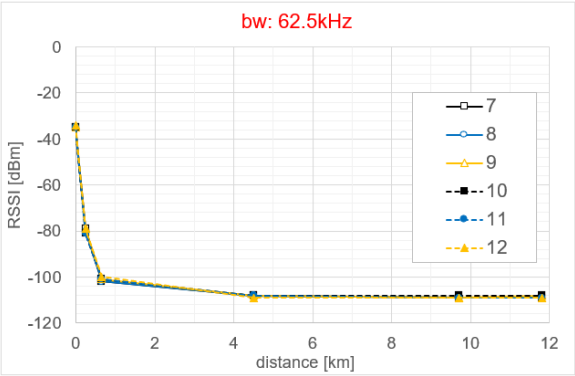
参考～屋外通信テスト（見通しの良い環境）～

見通しの良い屋外環境で、帯域幅（bw）および拡散率（sf）と受信電波強度（rssi）の関係を評価した結果を紹介します。

- **通信テスト結果①～帯域幅 62.5kHz および 125kHz～**

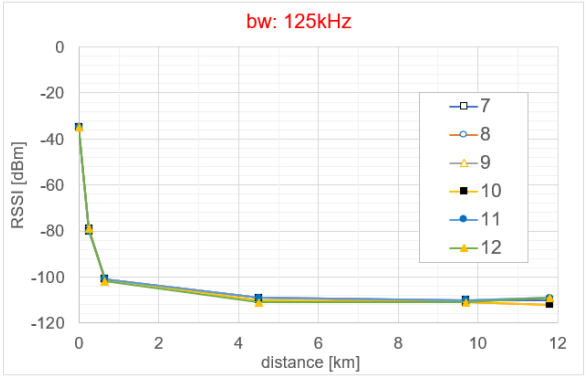
RSSI[dBm]

bw	sf	distance [km]					
		0	0.25	0.65	4.5	9.7	11.8
3	7	-35	-79	-102	-108	-109	-109
	8	-35	-80	-102	-108	-109	-109
	9	-35	-79	-101	-108	-109	-108
	10	-35	-81	-101	-108	-108	-108
	11	-35	-81	-101	-108	-109	-109
	12	-34	-79	-100	-109	-109	-109



RSSI[dBm]

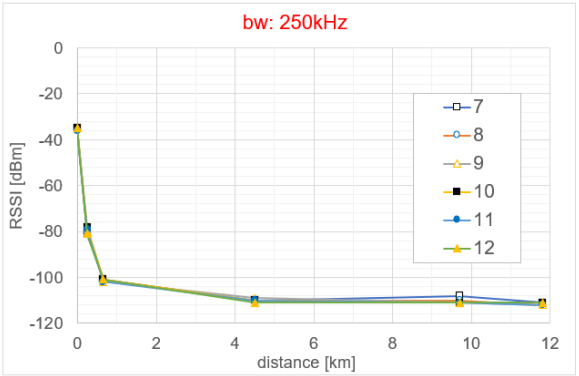
bw	sf	distance [km]					
		0	0.25	0.65	4.5	9.7	11.8
4	7	-35	-79	-101	-109	-110	-110
	8	-35	-79	-101	-109	-111	-112
	9	-35	-79	-101	-109	-110	-109
	10	-35	-80	-101	-110	-111	-112
	11	-35	-80	-101	-109	-110	-109
	12	-35	-79	-102	-111	-111	-109



- **通信テスト結果②～帯域幅 250kHz および 500kHz～**

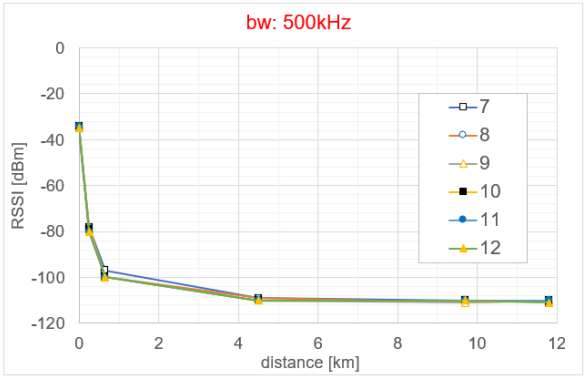
RSSI[dBm]

bw	sf	distance [km]					
		0	0.25	0.65	4.5	9.7	11.8
5	7	-35	-81	-101	-110	-108	-111
	8	-35	-80	-101	-110	-110	-112
	9	-35	-80	-102	-109	-111	-112
	10	-35	-78	-101	-110	-111	-111
	11	-36	-79	-102	-110	-111	-112
	12	-35	-81	-101	-111	-111	-111



RSSI[dBm]

bw	sf	distance [km]					
		0	0.25	0.65	4.5	9.7	11.8
6	7	-35	-78	-97	-109	-110	-110
	8	-34	-78	-100	-109	-111	-110
	9	-34	-79	-100	-110	-111	-110
	10	-34	-79	-100	-110	-110	-111
	11	-34	-80	-100	-110	-110	-110
	12	-35	-80	-100	-110	-110	-111



見通しの良い屋外環境では、帯域幅（bw）と拡散率（sf）の設定にかかわらず、受信電波強度（rssi）はほぼ同じ結果となりました。そのため、この環境で通信する場合は伝送速度を優先して通信パラメータを設定できます。

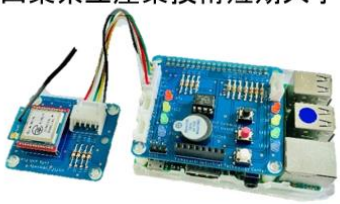
Copyright © 2024 YITJC All Rights Reserved.

参考～屋外通信テスト（見通しの悪い環境）～

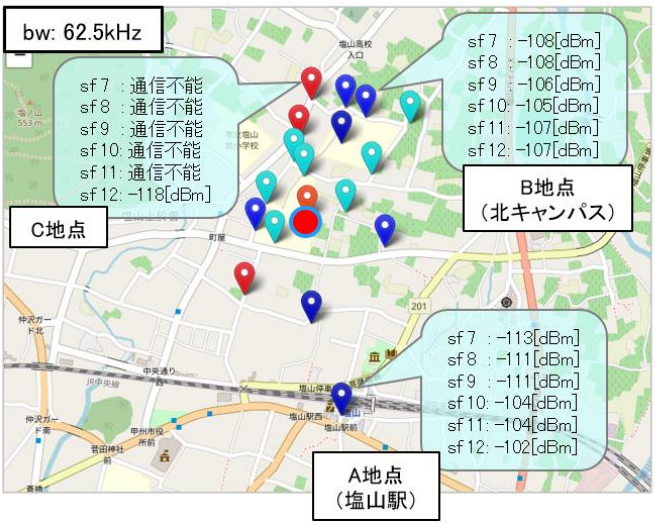
山梨県立産業技術短期大学校（甲州市塩山）周辺の屋外環境で実施した LoRa 通信テストの結果を紹介します。見通しの悪い環境（住宅などの建物が密集する地域）で、通信パラメータ（帯域幅および拡散率）の設定による受信電波強度（rssi）の違いを調査しました。

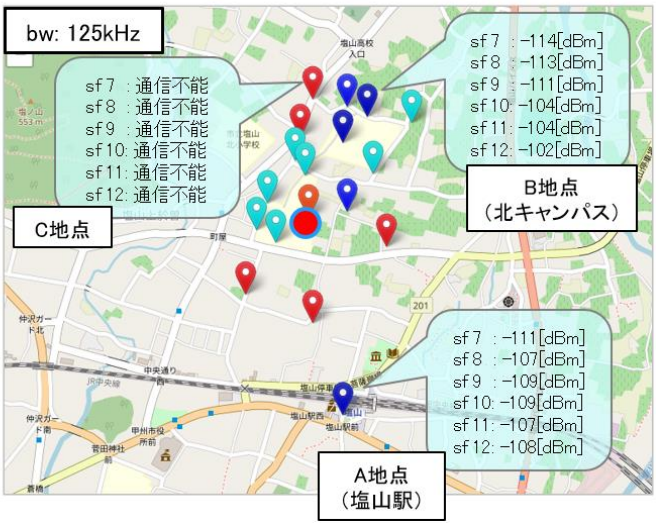
建物や地形などの影響で見通しの悪い環境では、拡散率（sf）を高く設定することで、受信電波強度（rssi）が向上し、通信可能な範囲が拡大することがわかります。

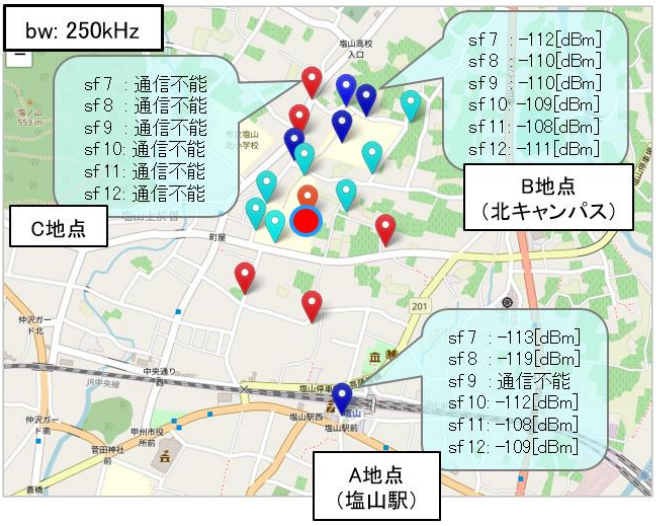
Gateway (Raspberry Pi)
@山梨県立産業技術短期大学校（屋上）



2024年7月8日 曇・30.8℃・60%RH
[通信条件]
Band Width: 62.5kHz～500kHz
Spread Factor: 7～12







～参考～

LoRa 通信モジュール（ES920LR2）を使用して安定した通信を行うためには、受信電波強度（rssi）の値が-120[dBm]以上確保できる環境で、プライベート LoRa ネットワークを構築してください。

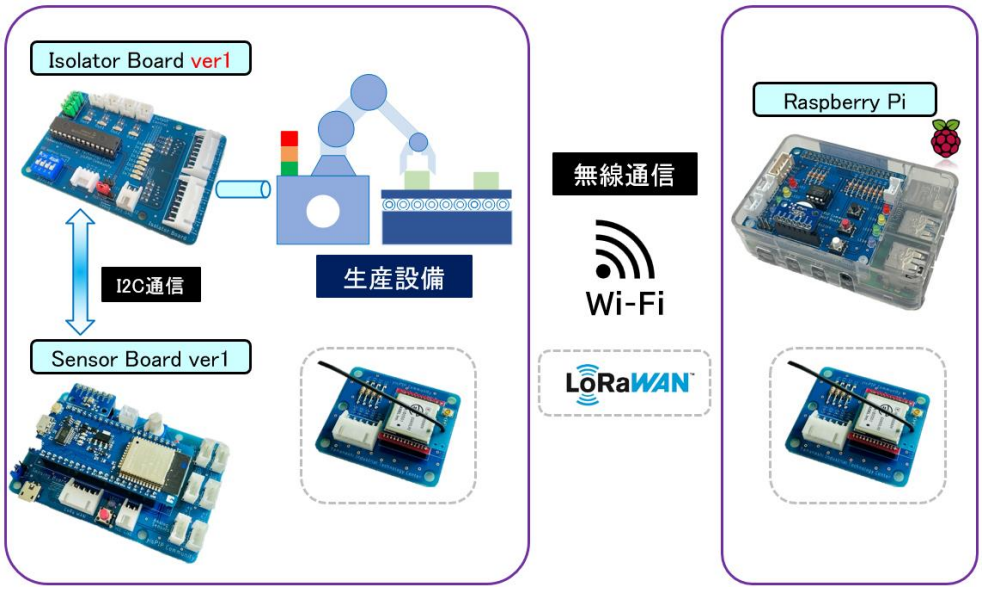
Copyright © 2024 YITJC All Rights Reserved.

11.10 Isolator Board 制御

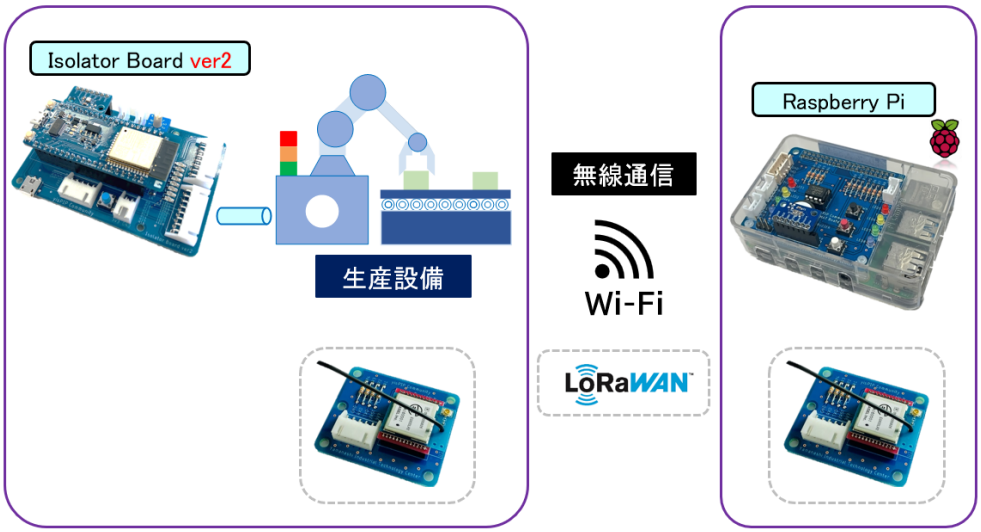
Isolator Board は I2C 通信で制御可能な I/O Expander を実装しているため、Sensor Board から I/O Expander の入出力端子を制御できます。また、ネットワーク環境が整備されていない生産現場においても、Wi-Fi や LoRa 通信などの無線通信機能を備えた Sensor Board と Isolator Board の連携により、生産設備の IoT 化を促進できます。

ここでは、Sensor Board を使用して Isolator Board の入出力制御を行う方法について学習します。

次の構成は、Sensor Board と Isolator Board ver1 が連携して、生産設備の稼働信号を Raspberry Pi に送信するシステムです。このシステムでは、Isolator Board ver1 が取得した生産設備の稼働信号を Sensor Boad が I2C 通信で受信します。Sensor Board は LoRa 通信にも対応しているため、ネットワーク環境が整備されていない生産現場や屋外の工場設備など、広範囲で収集したデータを送信できます。



さらに、Isolator Board ver2 は、ESPr® Developer 32 を搭載しているため、生産設備やセンサから取得したデータを Raspberry Pi やクラウドサービスに直接送信できます。



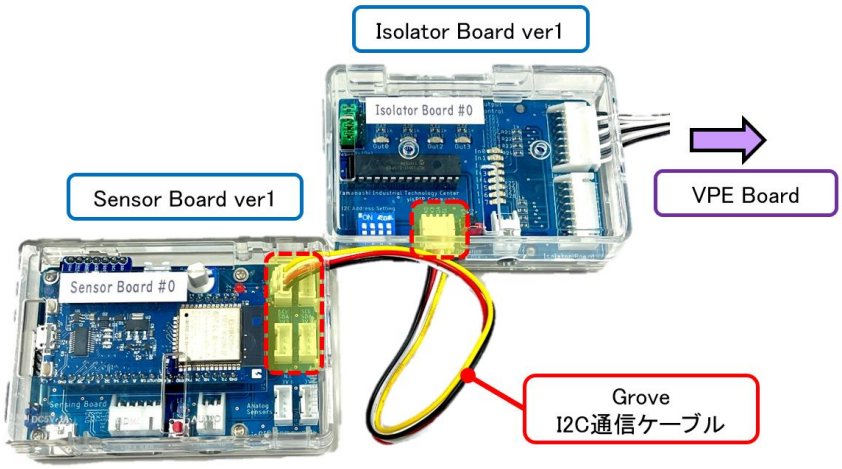
Copyright © 2024 YITJC All Rights Reserved.

Isolator Board を使用して生産設備と連携したシステムを構築する場合、稼働中の生産設備を使用してデバッグを行うことは困難です。そのため、生産設備を想定した VPE Board（仮想生産設備基板：Virtual Production Equipment）を使用して生産設備の入出力制御を学習します。

手順①～Isolator Board, VPE Board の接続～

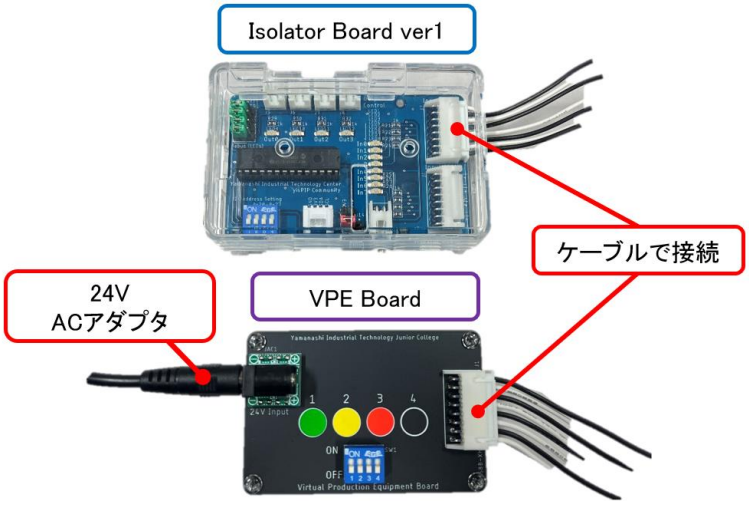
この作業は、必ず Sensor Board の電源を切った状態で行ってください。

まず、Sensor Board と Isolator Board を Grove I2C 通信ケーブルで接続します。



このとき、Isolator Board の出力ジャンパーピン（緑）をデバッグ側（Debug 側）に設定してください。

続いて、Isolator Board と VPE Board をケーブルで接続してください。ケーブルで接続したあと、24V の AC アダプタを VPE Board に接続して電源を供給してください。



Copyright © 2024 YITJC All Rights Reserved.

手順②～Isolator Board 動作確認～

Sensor Board に電源を供給し、Isolator Board の動作を確認します。

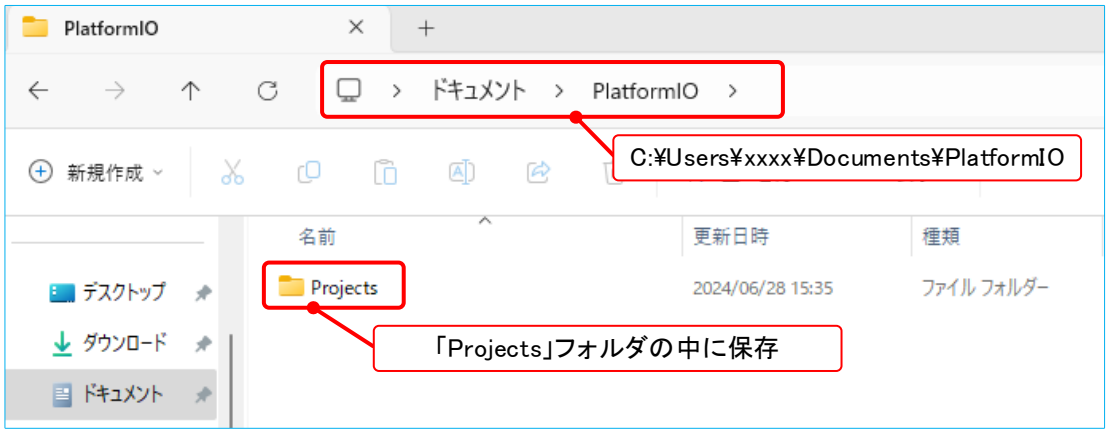
Isolator Board の入力 LED（赤色）と VPE Board の DIP スイッチは、次のとおり接続しています。

VPE Board の DIP スイッチの状態（On/Off）に応じて、Isolator Board の LED 表示が変化することを確認してください。

Isolator Board	VPE Board
In0	DIP スイッチ 1
In1	DIP スイッチ 2
In2	DIP スイッチ 3
In3	DIP スイッチ 4
In4	未接続
In5	未接続
In6	未接続
In7	未接続

手順③～プロジェクト配置～

共有サイトから対象のプロジェクトをダウンロードして、「Projects」ディレクトリに配置してください。



[対象プロジェクト]

Sensor_Board7

[プログラム内容]

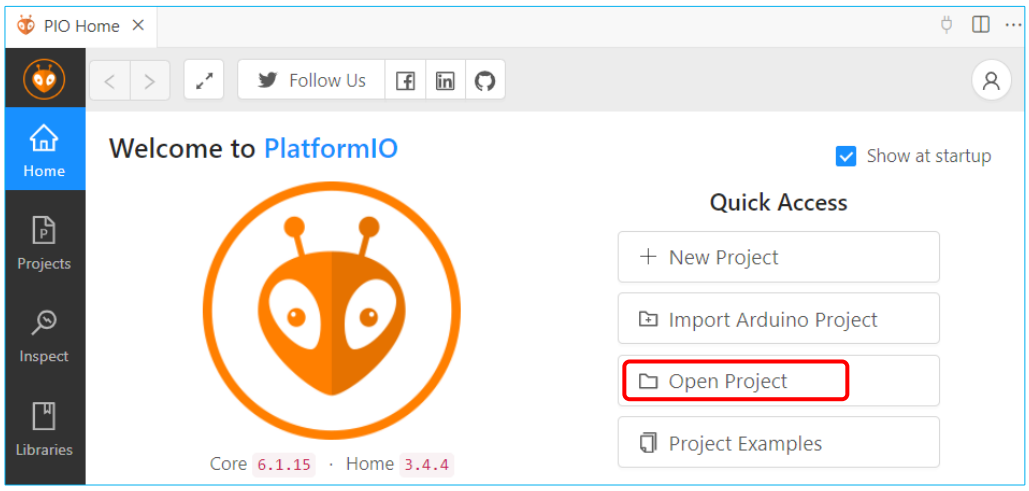
Isolator Board との間で I2C 通信を使用して、次の処理を実行します。

- 入力 GPIO[0-3] VPE Board（DIP スイッチ 1 番～4 番）の状態を Read
- 出力 GPIO[3]に High を Write

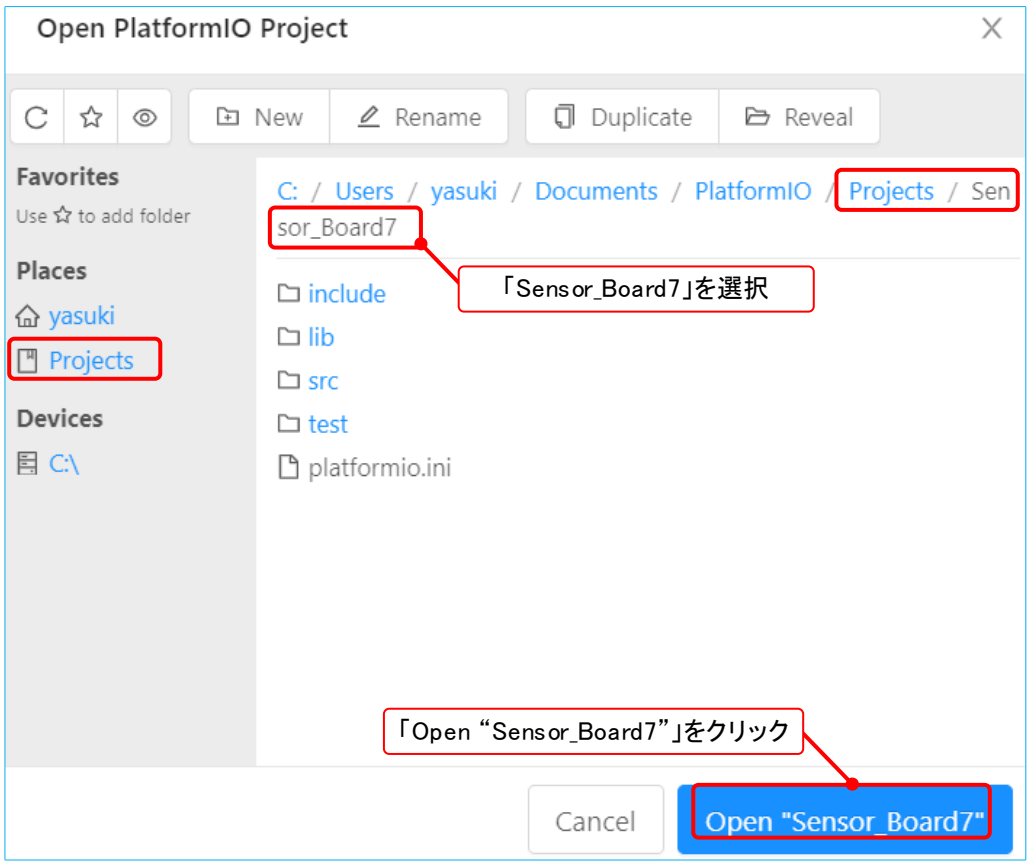
Copyright © 2024 YITJC All Rights Reserved.

手順④～プロジェクトオープン～

PlatformIO Home 画面で「Open Project」をクリックしてください。



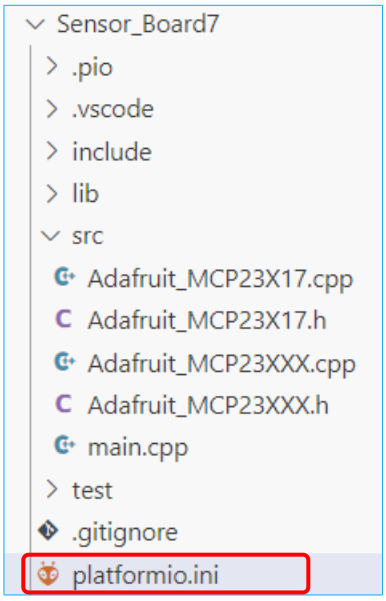
プロジェクト「Sensor_Board7」を選択してから、「Open Projects」をクリックしてください。

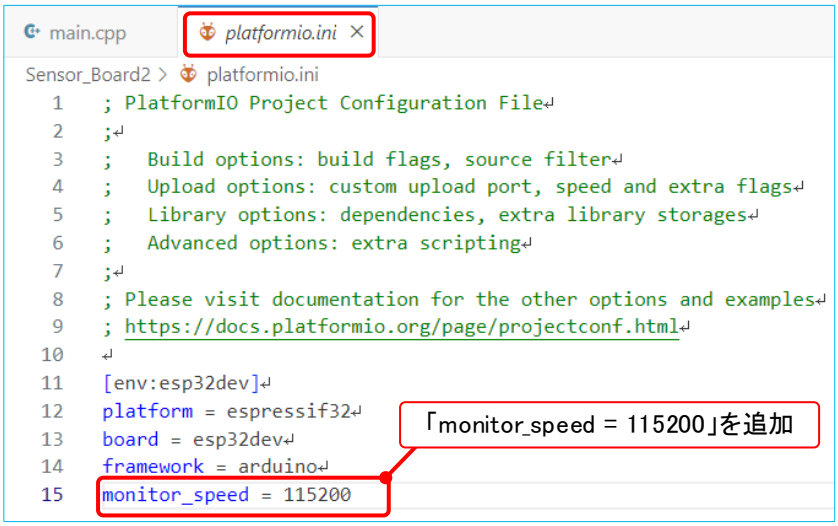


Copyright © 2024 YITJC All Rights Reserved.

手順⑤～シリアルモニタ設定～

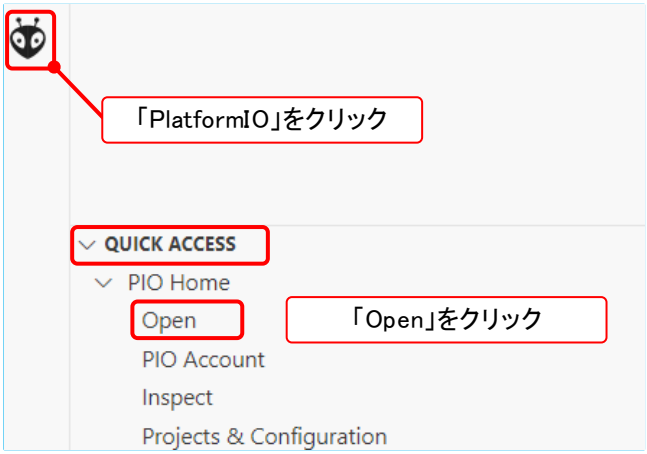
VPE Board に実装されている DIP スイッチ 1 番～4 番の状態は、Platform IO のシリアルモニタ（Serial Monitor）に表示します。シリアルモニタの通信速度を 115200bps に設定するため、「platformio.ini」ファイルに通信速度（monitor_speed）の記述を追加してください。





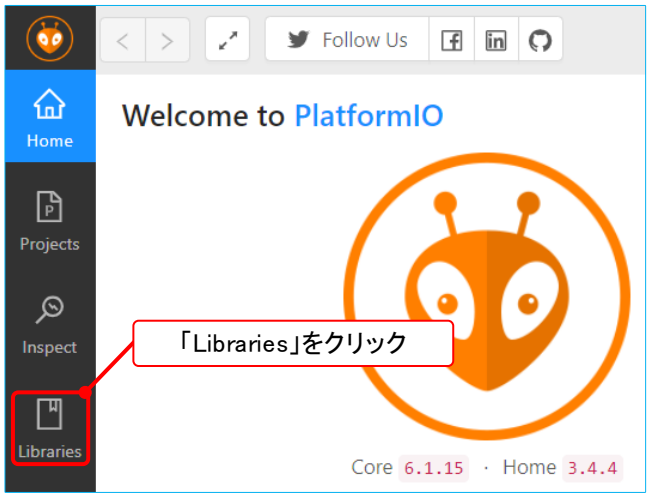
手順⑥～MCP23017 ライブラリ追加～

Isolator Board に実装されている I/O Expander と I2C 通信するために必要な「MCP23017」ライブラリをプロジェクトに追加するため、PlatformIO Home 画面に戻ります。画面左のメニューから「PlatformIO」のアイコンをクリックしたあと、Quick Access メニューから「PIO Home」の「Open」をクリックしてください。

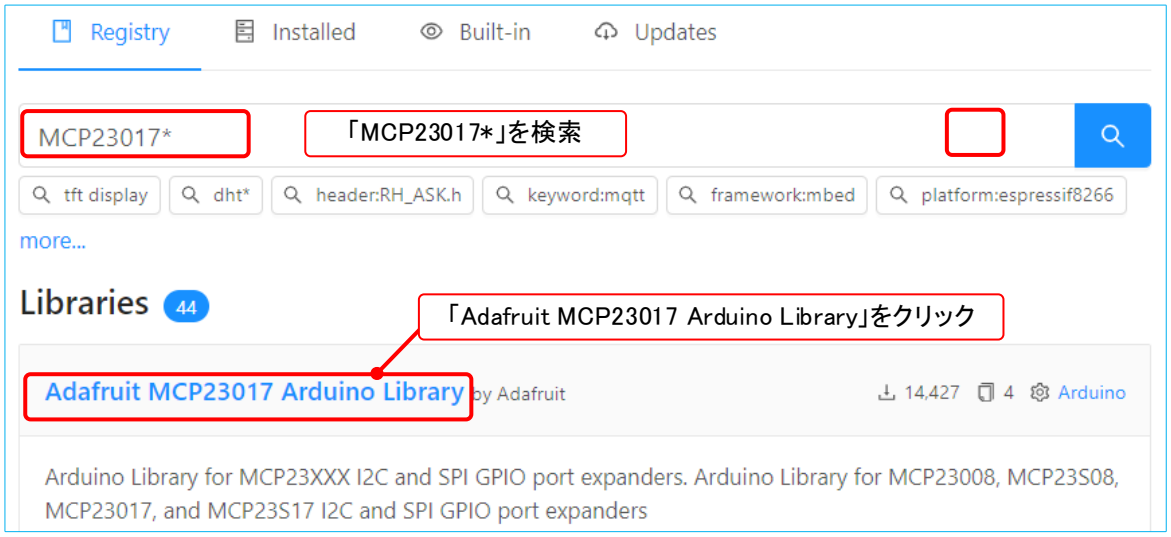


Copyright © 2024 YITJC All Rights Reserved.

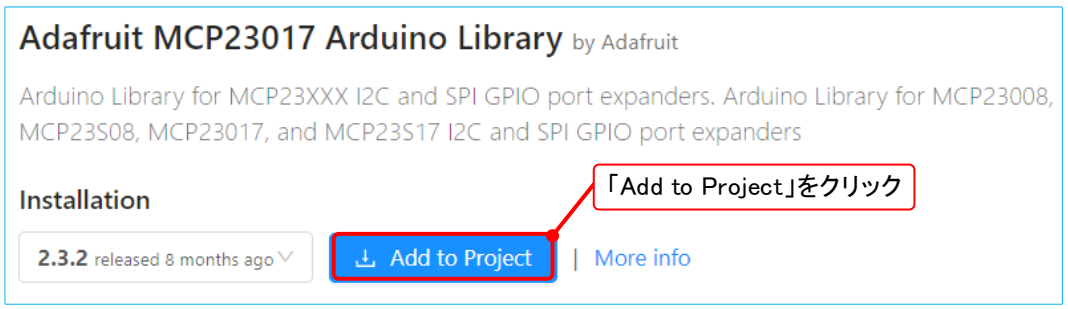
PlatformIO Home 画面左のメニューから、「Libraries」をクリックしてください。



検索バーを使用して「MCP23017*」を検索し、検索結果に表示された「Adafruit MCP23017 Arduino Library」をクリックしてください。



「Add to Project」をクリックしてください。



Copyright © 2024 YITJC All Rights Reserved.

「Sensor_Board7」プロジェクトを選択してから、「Add」をクリックしてください。

「MCP23017」ライブラリの追加に成功したら、「platformio.ini」ファイルに「MCP23017」ライブラリの記述が追加されます。

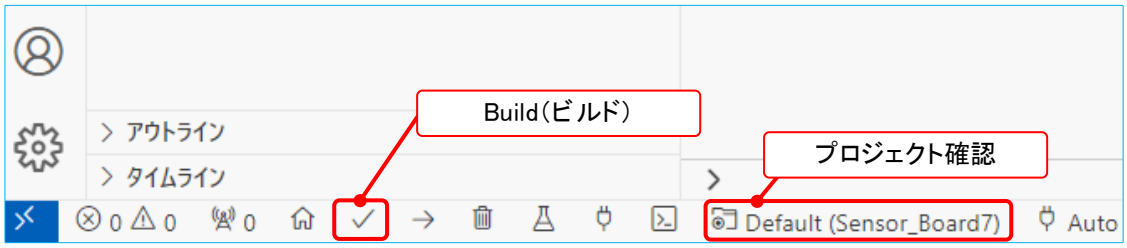
```
11  [env:esp32dev]
12  platform = espressif32
13  board = esp32dev
14  framework = arduino
15  monitor_speed = 115200
16  lib_deps = adafruit/Adafruit MCP23017 Arduino Library@^2.3.2
17
```

手順⑦～ビルド実行～

画面下ステータスバーに表示されているメニューから「Build（ビルド）」をクリックしてください。

このとき、ビルドを実行するプロジェクトが「Sensor_Board7」であることを確認してください。

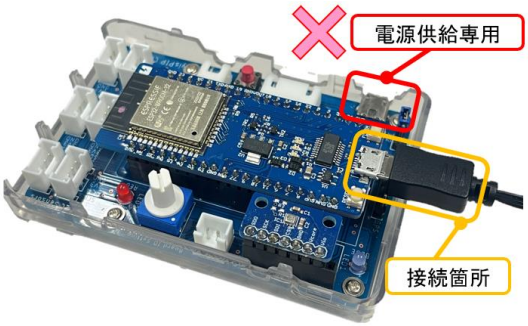
※プロジェクトが異なる場合は、「プロジェクト名」をクリックして正しい「プロジェクト」を選択してください。



Copyright © 2024 YITJC All Rights Reserved.

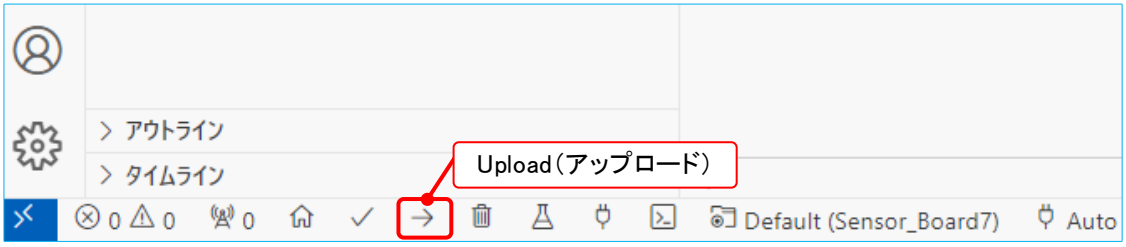
手順⑧～ESP32 接続～

Sensor Board に搭載されている ESPr® Developer 32 と Micro USB ケーブルで接続します。プログラムを書き込む場合は、必ず ESPr® Developer 32 に Micro USB ケーブルを接続してください。



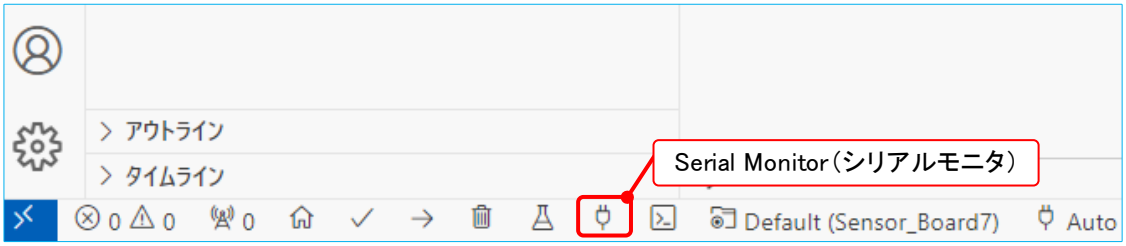
手順⑨～プログラム書き込み～

画面下ステータスバーに表示されているメニューから「Upload（アップロード）」をクリックしてください。

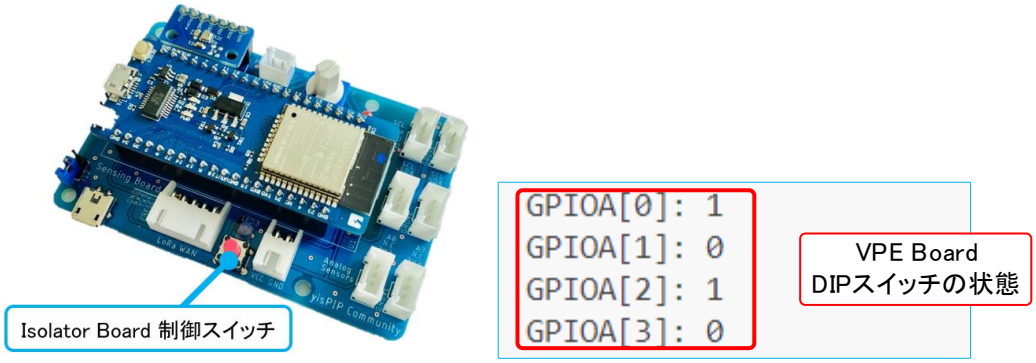


手順⑩～動作確認～

画面下ステータスバーに表示されているメニューから「Serial Monitor（シリアルモニタ）をクリックしてください。

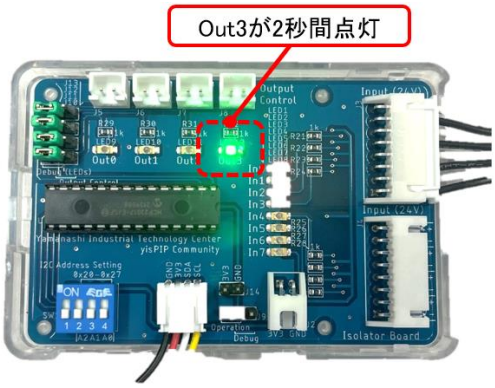


Sensor Board に実装されているタクトスイッチ（赤色）を押したあと、VPE Board に実装されている DIP スイッチ 1 番～4 番の状態が正しく表示されていることを確認してください。



Copyright © 2024 YITJC All Rights Reserved.

シリアルモニタに DIP スイッチの状態が表示されたあと、Isolator Board に実装されている I/O Expander の GPIO[3]に High が Write されるため、Out3 の出力 LED（緑）が 2 秒間点灯します。



～プログラム解説～

行	プログラム
1	#include <Arduino.h>
2	#include <Adafruit_MCP23X17.h>
3	
4	//SW Pins
5	const int SW = 19;　　//GPIO19
6	//タクトSW押下判定時間
7	static const unsigned long PUSH_SHORT = 100;
8	//GPIOA Status
9	int GPA0, GPA1, GPA2, GPA3;
10	
11	Adafruit_MCP23X17 mcp;
12	
13	void isolator() {
14	//Sample for Reading registers (GPIOA)
15	GPA0 = mcp.digitalRead(0);
16	GPA1 = mcp.digitalRead(1);
17	GPA2 = mcp.digitalRead(2);
18	GPA3 = mcp.digitalRead(3);
19	
20	Serial.printf("GPIOA[0]: %d¥r¥n", GPA0);
21	Serial.printf("GPIOA[1]: %d¥r¥n", GPA1);
22	Serial.printf("GPIOA[2]: %d¥r¥n", GPA2);
23	Serial.printf("GPIOA[3]: %d¥r¥n", GPA3);
24	delay(500);
25	//Sample for Output Control (GPIOB[0])
26	mcp.digitalWrite(10, HIGH);

Copyright © 2024 YITJC All Rights Reserved.

```
27      delay(2000);
28      mcp.digitalWrite(11, LOW);
29      delay(1000);
30  }
31
32  void setup() {
33      //SW Pin Settings
34      pinMode(SW, INPUT);   //SW
35      Serial.begin(115200);
36      delay(20);
37      mcp.begin_I2C(0x20);
38      //GPIO A
39      mcp.pinMode(0, INPUT);
40      mcp.pinMode(1, INPUT);
41      mcp.pinMode(2, INPUT);
42      mcp.pinMode(3, INPUT);
43      //GPIO B
44      mcp.pinMode(8, OUTPUT);
45      mcp.pinMode(9, OUTPUT);
46      mcp.pinMode(10, OUTPUT);
47      mcp.pinMode(11, OUTPUT);
48  }
49
50  void loop() {
51      static unsigned long lastPressTime = 0;
52      unsigned long currentTime = millis();
53      // SWが押された場合の処理
54      if (digitalRead(SW) == LOW) {
55          unsigned long pressDuration = currentTime - lastPressTime;
56          // ショートプレス判定
57          if (pressDuration > PUSH_SHORT) {
58              isolator();   //Isolator制御
59          }
60          // デバウンス処理
61          while (digitalRead(SW) == LOW) {
62              delay(10);   // SWが安定するまで待機
63          }
64          lastPressTime = currentTime;   // 最後の押下時刻を更新
65      }
```

Copyright © 2024 YITJC All Rights Reserved.

```
66 }
```

灰色部は、これまでに説明した内容です。

- 1 行目: Arduino の標準ライブラリを使用するため「Arduino.h」をインクルード
- 2 行目: I/O Expander の入出力制御を行うためのライブラリ「Adafruit_MCP23X17.h」をインクルード
- 5 行目: タクトスイッチのピン番号定義
- 7 行目: タクトスイッチ押下（ショートプレス）判定時間を定義
- 9 行目: GPIO[0-3] VPE Board（DIP スイッチ 1 番～4 番）の状態を格納するための変数を定義

変数	VPE Board
GPIOA[0]	DIP スイッチ 1
GPIOA[1]	DIP スイッチ 2
GPIOA[2]	DIP スイッチ 3
GPIOA[3]	DIP スイッチ 4

- 11 行目: I/O Expander「MCP23017」を制御するためのインスタンス「mcp」を作成
- 13 行目～30 行目: isolator()関数
  Sensor Board が I/O Expander「MCP23017」の入出力端子に対して行う制御内容を定義しています。
- 15 行目～18 行目: I/O Expander「MCP23017」の入力端子の状態を読み取り
  mcp.digitalRead()メソッドを使用して GPIO[0-3]の状態を読み取っています。
- 20～23 行目: I/O Expander「MCP23017」の入力端子の状態をシリアルモニタに print 表示
- 24 行目: 500 ミリ秒プログラムを停止
  I2C 通信が安定するのを待つため、500 ミリ秒の待機時間を設定しています。
- 26 行目: I/O Expander「MCP23017」の出力端子（GPIOB[3]）の Write 処理（High 出力）を実行
- 27 行目: 2000 ミリ秒（2 秒）プログラムを停止し、出力端子（GPIOB[3]）の High 状態を維持
- 28 行目: I/O Expander「MCP23017」の出力端子（GPIOB[3]）の Write 処理（Low 出力）を実行
- 27 行目: 1000 ミリ秒（1 秒）プログラムを停止し、出力端子（GPIOB[3]）の Low 状態を維持
- 32 行目～48 行目: setup()関数
  電源供給（またはリセット）時に一度だけ実行される処理を記載しています。
- 34 行目: タクト SW が接続されている GPIO19 ピンを「入力」モードに設定
- 35 行目: 通信速度（115200bps）を指定してシリアル通信を開始
- 36 行目: 20 ミリ秒プログラムを停止
  シリアル通信が安定するのを待つため、20 ミリ秒の待機時間を設定しています。
- 37 行目: I/O Expander「MCP23017」と行う I2C 通信を初期化
- 39 行目: DIP スイッチ 1 が接続されている GPIOA[0]ピンを「入力」モードに設定
- 40 行目: DIP スイッチ 2 が接続されている GPIOA[1]ピンを「入力」モードに設定
- 41 行目: DIP スイッチ 3 が接続されている GPIOA[2]ピンを「入力」モードに設定
- 42 行目: DIP スイッチ 4 が接続されている GPIOA[3]ピンを「入力」モードに設定
- 44 行目: Out0 LED が接続されている GPIOB[0]ピンを「出力」モードに設定
- 45 行目: Out1 LED が接続されている GPIOB[1]ピンを「出力」モードに設定

Copyright © 2024 YITJC All Rights Reserved.

- 46 行目: Out2 LED が接続されている GPIOB[2]ピンを「出力」モードに設定
- 47 行目: Out3 LED が接続されている GPIOB[3]ピンを「出力」モードに設定
- 50 行目～66 行目: loop 関数
  プログラムが動作している間、繰り返し実行される処理を記載しています。
- 51 行目: スイッチが最後に押された時刻を格納するための変数「lastPressTime」を定義
  初期値として 0 を設定しています。
- 52 行目: プログラムが開始されてからの経過時間をミリ秒単位で取得
  取得した経過時間を変数「currentTime」に格納しています。
- 54 行目～65 行目: タクトスイッチが押された時の処理
- 55 行目: スイッチが押されている時間を格納するための変数「pressDuration」を定義
  currentTime から lastPressTime を引いた値が、スイッチが押されてから経過した時間「pressDuration」です。
- 57 行目: スイッチ押下（ショートプレス）判定
  pressDuration が PUSH_SHORT（押下判定時間）より大きい場合、isolator()関数を呼び出して Isolator Board に対する入出力制御を実行します。
- 61 行目～63 行目: デバウンス処理（チャタリング防止処理）
  スイッチが押されている間、10 ミリ秒の遅延時間を設定してチャタリングが収まるのを待っています。
- 64 行目: 最後にスイッチが押下された時刻を更新

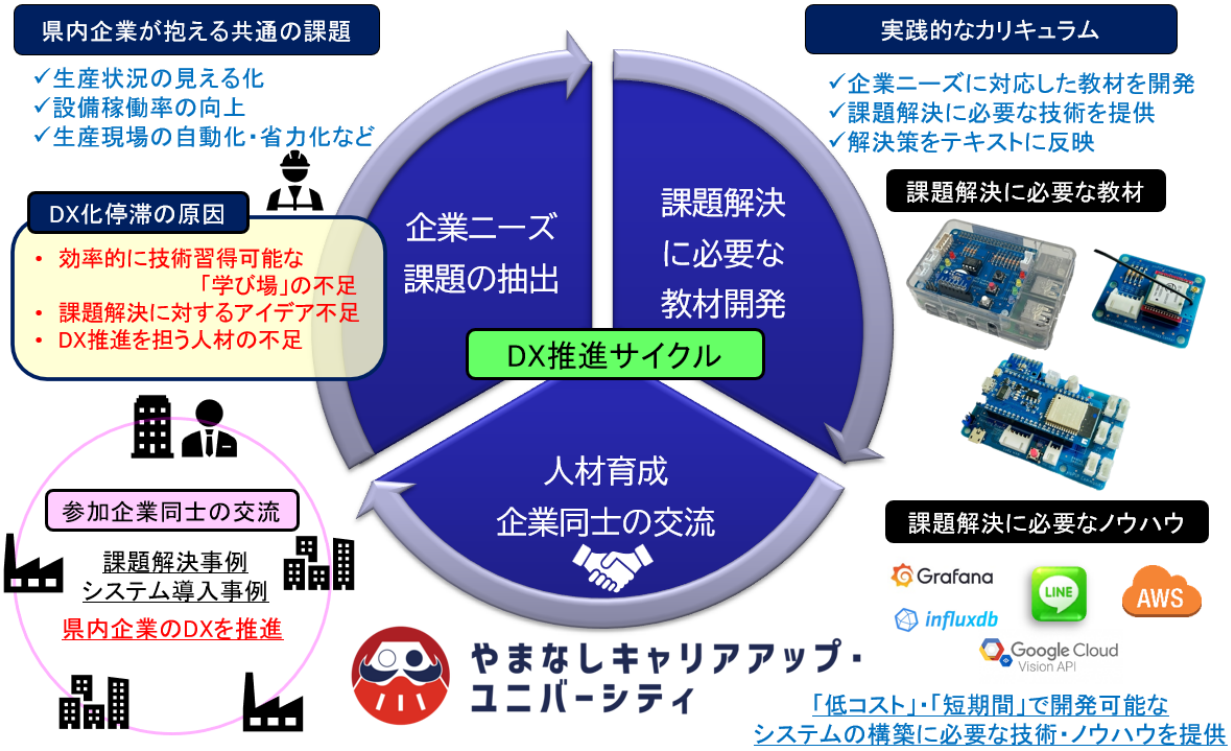
Copyright © 2024 YITJC All Rights Reserved.

やまなしキャリアアップ・ユニバーシティ「ものづくり講座」

やまなしキャリアアップ・ユニバーシティは、働きながら「学べる場」として令和 5 年度に開校しました。働く人のスキルアップ、企業の収益・生産性アップ、働く人の評価・賃金アップの「スリーアップ」を目標としています。山梨県立産業技術短期は「やまなしキャリアアップ・ユニバーシティ」と連携して、ものづくり講座「生産性向上のための IoT 実践技術講座」を開講しました。

本講座は、令和 2 年度に山梨県産業技術センターで実施した研究テーマ「PLC と安価な組み込みコンピュータを用いた生産性向上 IoT システムの開発」の成果に加え、県内企業への支援から得られた技術やノウハウを教材に反映しています。これにより、短期間で効率的に技術を習得できる実践的なカリキュラムを提供しています。

今後も「課題解決に必要な教材開発」、「やまなしキャリアアップ・ユニバーシティによる人材育成」、「企業ニーズの抽出」のサイクルを繰り返すことで、県内企業で働く人のスキルアップに貢献していきます。



Copyright © 2024 YITJC All Rights Reserved.