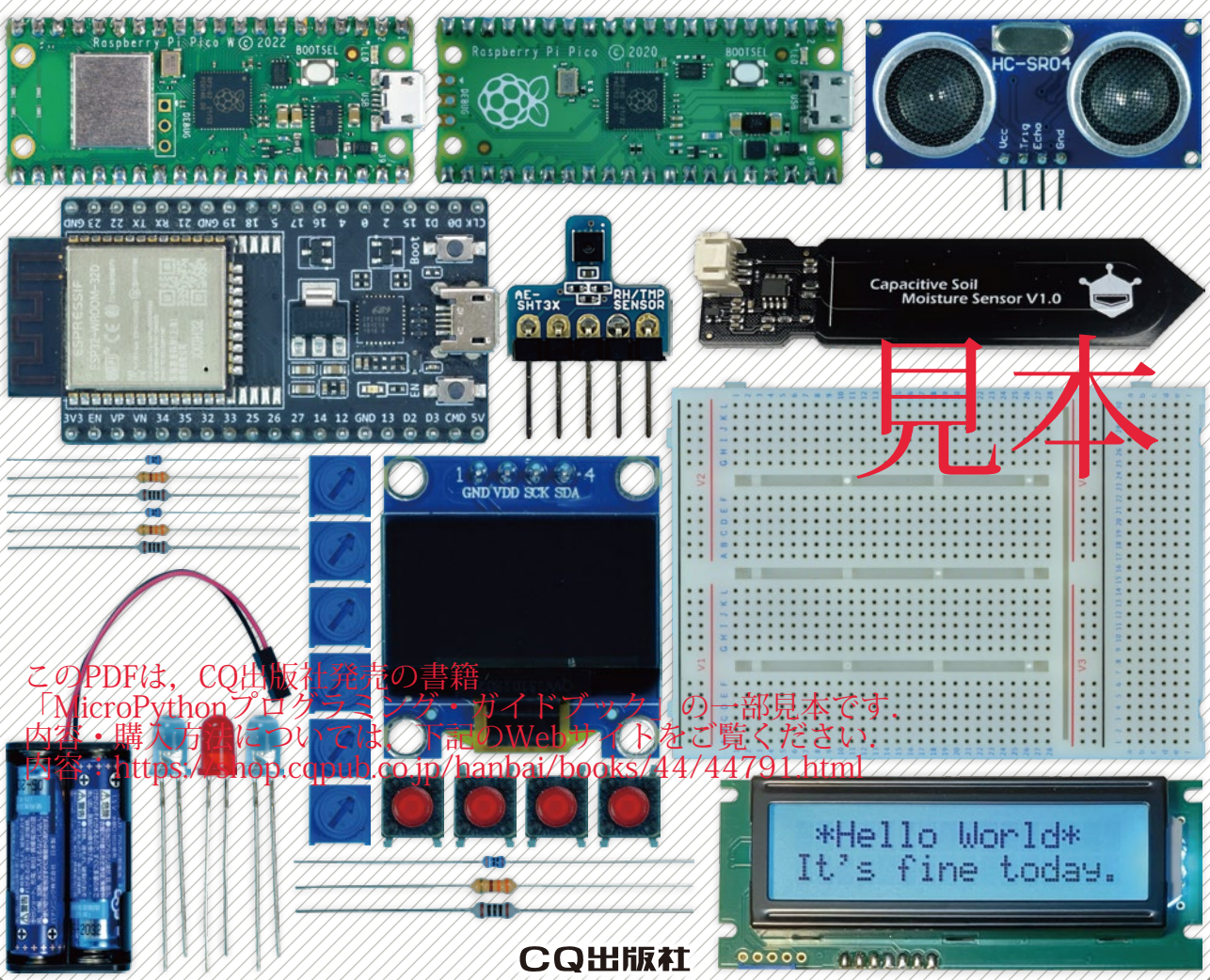
ボード・コンピュータ・シリーズ

ラズパイPico/Pico W & ESP32完全対応

マイコン向けプロトタイピング言語の新定番

MicroPython プログラミング・ガイドブック

宮田 賢一，角 史生 共著



CQ出版社

はじめに

本書の構成とMicroPythonの使いどころ

本書はプログラミング言語MicroPythonの言語仕様やプログラミングの仕方を初心者向けに解説した入門書です．本書の前半ではMicroPythonの言語仕様を一通り解説し，後半では実践編として各種デバイスをMicroPythonで制御する方法を紹介します．

前半の言語仕様のパートは，ArduinoやC言語は分かるけれどもPython系の言語は初めてという方はもちろん，マイコンでのプログラミング自体が初めての方でも読み進められるように，なぜその言語仕様が必要なのかというそもそもの考え方から解説します．さらに，シンプルにプログラムを書くための便利な記法やMicroPython処理系内部の動作についての解説も盛り込み，MicroPythonによるプログラミング経験者であっても新たな気づきが得られるようにしました．

また，文法の解説のすべてに実際に実行した結果を引用しています．これにより，ラズベリー・パイPicoやESP32といったマイコン・ボード上で読者自身が打ち込んでみて，結果が一致することを確認しながら読み進めていくことができ，自然にMicroPythonの言語仕様が身につくようになっています．

ところでMicroPythonはどんな場面で使えるものでしょうか．MicroPythonはマイコンを搭載する組み込み機器向けにチューニングされたプログラミング言語です．その中でも特にナノ秒やマイクロ秒単位での正確なタイミング管理が求められないような機器の制御に向いています．そのような使い方の例としては，たとえば各種センサからの情報の取得や小型ディスプレイへの文字・画像の表示，ポンプの開閉による水の流量の制御などがあります．それに加えて近年では，マイコンをインターネットに接続して，センサ情報をクラウドに送信したり，屋外から自宅の家電製品を制御したりという，いわゆるInternet of Things（IoT）機器への応用例も増えています．MicroPythonはこのような使い方で威力を発揮するものです．

本書の後半では，センサやディスプレイ，ネットワーク・モジュールなどのデバイスをMicroPythonで制御するために必要なプログラミング・テクニックを解説します．これらのデバイスには，便利なライブラリがメーカや有志によってすでに提供されている場合が多く，本来はそれらのライブラリを活用する方が手っ取り早く楽です．しかし本書では，デバイスの仕様書を読み取るところから始めて，MicroPythonの基本機能だけでプログラミングする方法を解説しています．これによりMicroPython自体の理解が進むだけではなく，まだライブラリが存在しない最新のデバイスであっても自分自身でライブラリが作れるようになることが期待できます．とはいえ初心者が一から制御プログラムを書き始めるのは敷居が高いので，実際の応用例からの逆引きでMicroPythonのプログラムを見ることのできる特別付録も収録しています．

私がMicroPythonに初めて触れたのはmicro:bitというマイコン・ボードでした．MicroPythonはPython譲りのシンプル・高機能な言語仕様であるところが気に入ったものの，当時の私はマイコン自体が初心者ということもあり，LEDの点滅はできても外付けの温度センサをどう扱って良いのかから分からず途方に暮れていました．しかしMicroPythonは，「まず使ってみる」という使い方が簡単に行えるのが楽しく，コマンドを入力するとすぐに応答が返ってくるのが学習にとても役立ちました．また，micro:bitからほとんど書き換えることなく他のマイコン・ボードでも動作するところも良いと感じています．

本書を手に取った皆さんも「まず使ってみる」から始めてみてください．あなたのプログラミング・ライフが楽しいものになることを願っています．

宮田 賢一

イントロダクション

言語機能／対話型／メモリ管理／データ構造／
ライブラリの取り込み

MicroPythonをお勧めする５つの理由

宮田 賢一

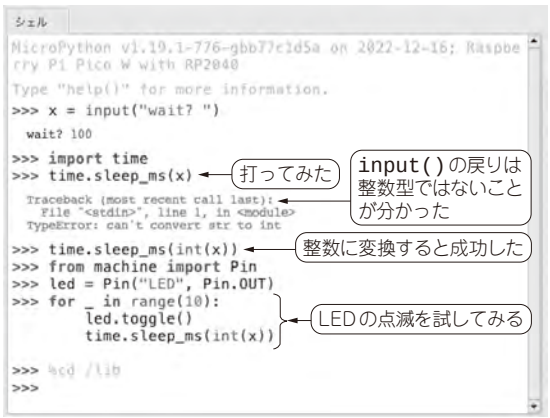


図1　MicroPythonはPythonと同じように対話型でエラーを修正しながらプログラム開発ができる

MicroPythonはマイコン向けにチューニングしたPythonです．

Pythonは世界で人気のある言語で，ユーザも多く，ビギナでも始めやすいとされています．MicroPythonは，制約の多い条件（CPU性能，メモリ量）であっても動作するように，マイコンではあまり使わない言語仕様をサポートしていません．つまりマイコンに特化した言語と言えるでしょう．しかし，MicroPythonはOSの支援無しに動作するので，マイコンが持つさまざまなデバイスのインターフェースを直接制御でき，物作りの楽しさを実感できます．また多くの基本ライブラリがMicroPython用に移植されており，Pythonと比べて遜色のないプログラミングが可能です．

● 理由①…Pythonの良いところを取り込み続ける

MicroPythonは，Pythonバージョン3.4をベースとして開発が始まりました．その後もPythonの便利機能をMicroPythonに取り込む活動が継続的に行われており，現在では次のようなPython機能が使えるようになっています．

- コルーチン，型ヒント（Python 3.4）
- f-string，数値のアンダスコアによる分割表現（Python 3.5）
- 代入式（Python 3.8）

今後の移植候補とされている機能も多数あるので，より使いやすい言語になっていくでしょう．

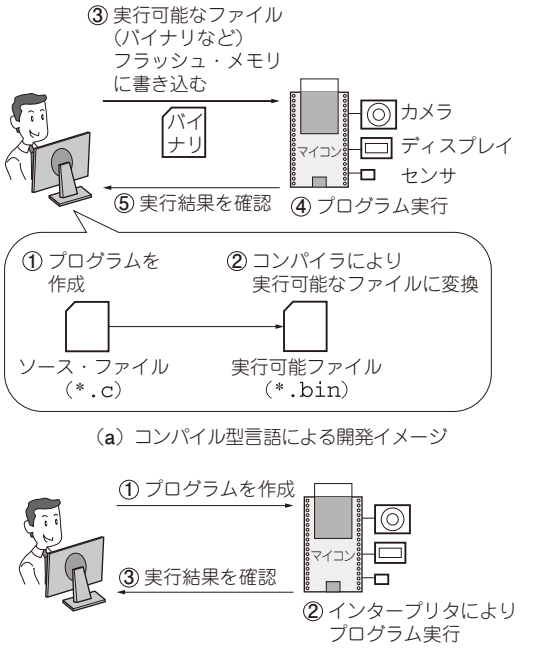


図2　コンパイル型とインタープリタ型の開発イメージ
MicroPythonはインタープリタ型言語なので１行ずつ対話的にプログラムを開発できる

● 理由②…対話型言語で開発効率が高い

MicroPythonはPythonと同じく対話型のインタープリタ型言語です（**図1**，**図2**）．そのためトライ＆エラーによるプログラミングが可能です．マイコンの基本的な使い方は，センサやモータ，小さなグラフィックス・ディスプレイなどのハードウェア・デバイスを接続してデータの送受信をすることです．ハードウェアの制御は目に見えない電気的なやりとりが行われることから，開発初期のプロトタイピングの段階では，いろいろ試しながら「正常に動作する」プログラムを

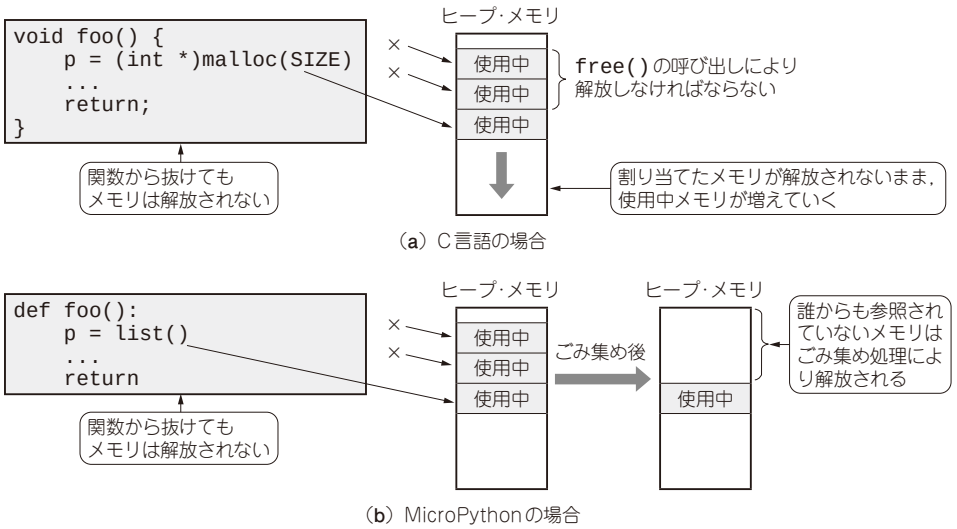


図3
MicroPythonでは使わなくなったメモリ領域が自動的に開放される

積み上げていくことになるでしょう．そのようなときに対話型で動作を試せるというのは，開発効率の向上につながります．

● 理由③…メモリ管理から解放される

プログラムの安全性を維持するための最重要課題はメモリ管理と言ってもよいでしょう．C言語では，動的にメモリが必要になるたびに`malloc`関数でヒープ・メモリからメモリ領域を確保し，不要になったら`free`関数で解放しなければなりません．つまり`malloc`と`free`は必ず1対1で対応させなければならず，注意深いプログラミングが求められます．

一例を**図3**に示します．Cの場合は関数内で`malloc`関数により割り当てたメモリ領域は関数から抜けてもそのままヒープ・メモリ上に残ります．このような関数を何度も呼び出し続けると，ヒープ・メモリ上には誰からも使われることのない使用中メモリが増えていくことになり，いずれメモリが不足してプログラムが異常終了します．一方MicroPythonの場合は，関数から抜けてもメモリが解放されないところは同じですが，ヒープ・メモリの空き容量が逼迫（ひっぱく）すると「ごみ集め」と呼ぶ処理が自動的に行われ，動作中のプログラムの誰からも参照されていないメモリ領域を解放してくれます．

● 理由④…豊富なデータ構造

MicroPythonはPythonと同じように，リストやタプル，辞書，集合といったコレクション型をサポートしていますし，クラスの定義によりユーザ自身で任意の型を作れます．また，数学関数やJSON文字列の処理といった基本ライブラリも最初から組み込まれています．

● 理由⑤…ネット上のライブラリを取り込む仕組み

ネットワークに接続できるマイコン・ボードであれば，必要なライブラリをインターネットからダウンロードしてインストールする仕組みもあります．これにより，プログラミングする前のライブラリ整備が省略でき，開発速度が向上します．

● 便利だけれど…デメリットもある

▶シビアなタイミング制御は苦手

不要になったメモリを自動回収するというメリットは，プログラマによって制御できないタイミングでごみ集め処理が実行されてしまうというデメリットもあります．つまり常に正確なタイミングで信号を出力することが求められる音声やビデオ信号の処理にはMicroPythonは向いていません．ただし一部の機能はマイコンのハードウェアに処理をオフロードするライブラリが用意されていますし，プログラムの工夫によってある程度ごみ集め処理の頻度を下げることもできるので，ハードウェアのサポート状況や，用途を考えてMicroPythonの採用を検討すべきです．

▶デバッグ環境が弱い

CやPythonには対話型デバッグのためのツールやライブラリが用意されているので，ソースコードの1行単位でステップ実行したり，ブレーク・ポイントで中断した時点での変数値を参照したりできます．しかしMicroPythonではまだソースコード・デバッグの仕組みをサポートしていません．そのためプログラムの各所に`print`文を仕込んで実行トレースや変数値を確認したり，異常状態に陥ったことをオンボードのLEDを点灯させて知らせたりといった手法に頼らざるを得ません．デバッグの効率化のためには，MicroPython

コラム　MicroPythonをもっと理解したくなったら

宮田 賢一

本書は次のような読者を想定しています．

- 普段はC/C++で組み込み機器を開発したり，Arduinoで電子工作をしたりしているが，新規システムをサッと試作したいエンジニア
- 製品開発時に，開発のための装置としてデータ記録装置やメカのエージング装置を作りたいエンジニア
- 授業や実験でマイコンを学習する必要のある学生

MicroPythonの特徴として挙げたように，MicroPython用のライブラリは公式/非公式を問わず，インターネット上に多く公開されています．基本的には車輪の再発明は避けた方がよいので「あるものを使う」というスタンスは大事なことです．しかし，それではMicroPythonという言語の学習には適切ではありませんし，なによりMicroPythonの面白さを体験できません．

そこで本書では，ありもののライブラリはなるべく使わず，自前でライブラリを作れるようになるための基本的なプログラミング技術を解説していきます．ただし，Microとは言っても，MicroPythonの言語仕様はかなり大きいので，特集では全てを解説できません．特集でMicroPythonの基本を理解した後は，次の情報を参考にMicroPythonを奥深くまで理解してほしいと思います．

● 公式「MicroPython ドキュメンテーション」

```
https://micropython-docs-ja.readthedocs.io/ja/latest/
```

MicroPythonの公式ドキュメントです．分からないことがあればまずこのウェブ・ページを見に行くのをお勧めします．

● Pythonらしいプログラムを知る「PEP8-ja」

```
https://pep8-ja.readthedocs.io/ja/latest/
```

Pythonのプログラミング・スタイルについてのガイドラインを定めたドキュメントPEP8（Python Enhancement Proposal 8）の日本語訳です．演算子の両端に空白を入れるべきかなど，Pythonらしいプログラムにするにはどうしたらよいかに悩んだときに参照するとよいでしょう．Pythonに対するガイドラインですが，MicroPythonにも当てはまると考えてよいです．

● ハードウェアを動かすのに悩んだときは「Adafruit CircuitPython Library Bundle」

```
https://github.com/adafruit/Adafruit_CircuitPython_Bundle
```

CircuitPythonは，Adafruit社が開発/販売しているマイコン・ボードやデバイス用に開発したMicroPython派生言語です．このCircuitPythonで動くライブラリのソースコードを公開しているサイトです．非常に多数のデバイス用のライブラリが公開されており，仕様の理解に苦しんだり，実装方法に悩んだりしたときに参考になります．ものによってはMicroPythonでもそのまま動く場合もあります．そのまま動くかどうかは，特集の内容を理解すれば自分で判断できるようになるでしょう．

● とにかく作ろう

アイデアを思いついたら実際に作りましょう．例えば，シリアル通信規格I²Cを利用する際に，正しい回路を設計しても，センサとマイコンとの間の結線が長いと途端に動かなくなる，というような不可解な現象はよくあることです．よく調べてみると，物理や電気回路の理論に基づく正しい現象であることが分かるでしょう．つまり「作ってだめなら原因を追及する，そしてまた作る」というプロセスが，エンジニアとしての経験値を何倍にも広げるために重要なのです．

では，何から手を付ければよいのか．もし仕事や研究のような目的が決まっていなければ，デバイスからクラウドまで幅広いスタックをカバーできるホーム・オートメーションがお勧めです．プログラミング言語としては，マイコン制御はMicroPythonやC/C++，一歩先を行くならRust，クラウド側はPythonやJavaScriptを習得できます．テクノロジとしては，デバイス，リアルタイムOS，ネットワーク・プロトコル，データベース，機械学習などを学べます．なにより思い通りに動くと楽しいです．

の言語仕様を正しく理解して，バグの要因を特定する力が必要になります．

みやた・けんいち

第1章

ラズベリー・パイ Pico/Pico W, ESP32-DevKit-C-32Eですぐ試せる!
STM32搭載NUCLEOやRA搭載RA4M1-CLICKERにも対応

本書で使うマイコン・ボード

宮田 賢一

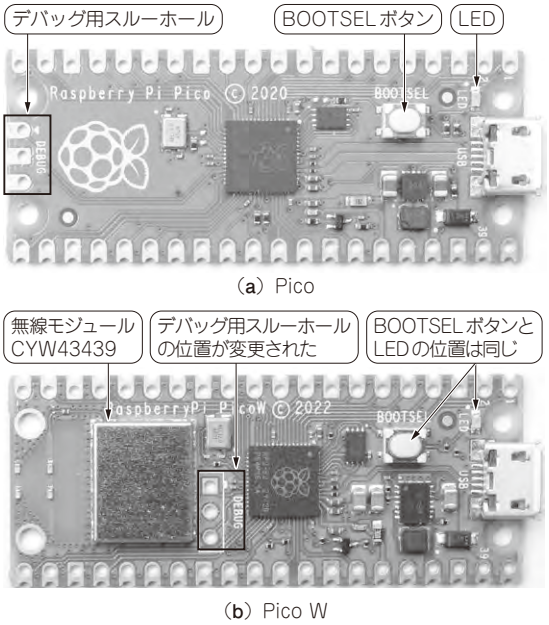


写真1　ラズベリー・パイ Pico，Wi-Fi付きのPico Wの外観
Picoは770円，Pico Wは1300円程度で購入できる点が魅力

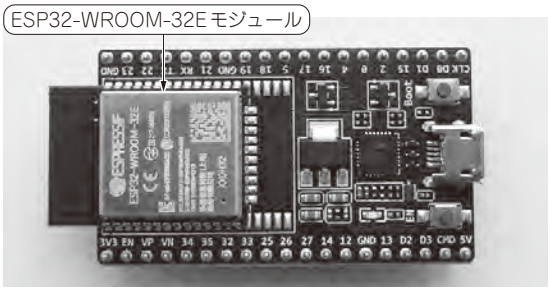


写真2　無線機能付きマイコン・ボードの定番ESP32-DevKitC-32E
Wi-FiとBluetoothを内蔵しながらモジュール単体で500円程度，開発キットでも1600円程度

ハードウェアに依存しないMicroPythonのプログラムの多くは，Pythonでもそのまま動作します．しかし，MicroPythonの学習には，実際にマイコン・ボードを使って試してみるのが一番です．本書では4種類のマイコン・ボードを使ってMicroPythonの学習をします．

表1　ラズベリー・パイ Picoシリーズの仕様

項　目		Pico	Pico W
SoC		RP2040	
CPU	コア	Cortex M0+，2コア	
	動作周波数	最高133MHz	
メモリ	SRAM	264Kバイト	
	フラッシュ・メモリ	2Mバイト（外付けQSPI接続）	
主要ペリフェラル	GPIO	ディジタル専用×26	
		ディジタル，アナログ兼用×4	
	通信	I²C×2	
		SPI×2	
		UART×2	
		USB 1.1（ホスト・デバイス）×1	
	A-Dコンバータ	汎用×4（12ビット，500kSps）内蔵温度センサ専用×1	
	PWM	16チャネル	
	プログラマブルIO	2	
	リアルタイム・クロック	内蔵（バッテリ・バックアップなし）	
無線通信	通信モジュール	—	CYW43439（インフィニオン テクノロジーズ）
	Wi-Fi	—	802.11n（2.4GHz）
外部ポート		USB Micro-B	
動作温度		−20～+85℃	−20～+70℃
電源電圧		1.8～5.5V	
外形寸法		51×21mm	

● ラズベリー・パイPico/Pico W

ラズベリー・パイPico/Pico W（以降，Pico/Pico W）は，ラズベリーパイ財団が開発したマイコン・ボードです（**写真1**）．ラズベリー・パイという名前を冠していますが，Linux OSは動作せず，ボード上のマイコンに直接プログラムを書き込んで実行します．

Picoは，ラズベリーパイ財団が独自開発したRP2040というSoC（System on a Chip）を搭載していま

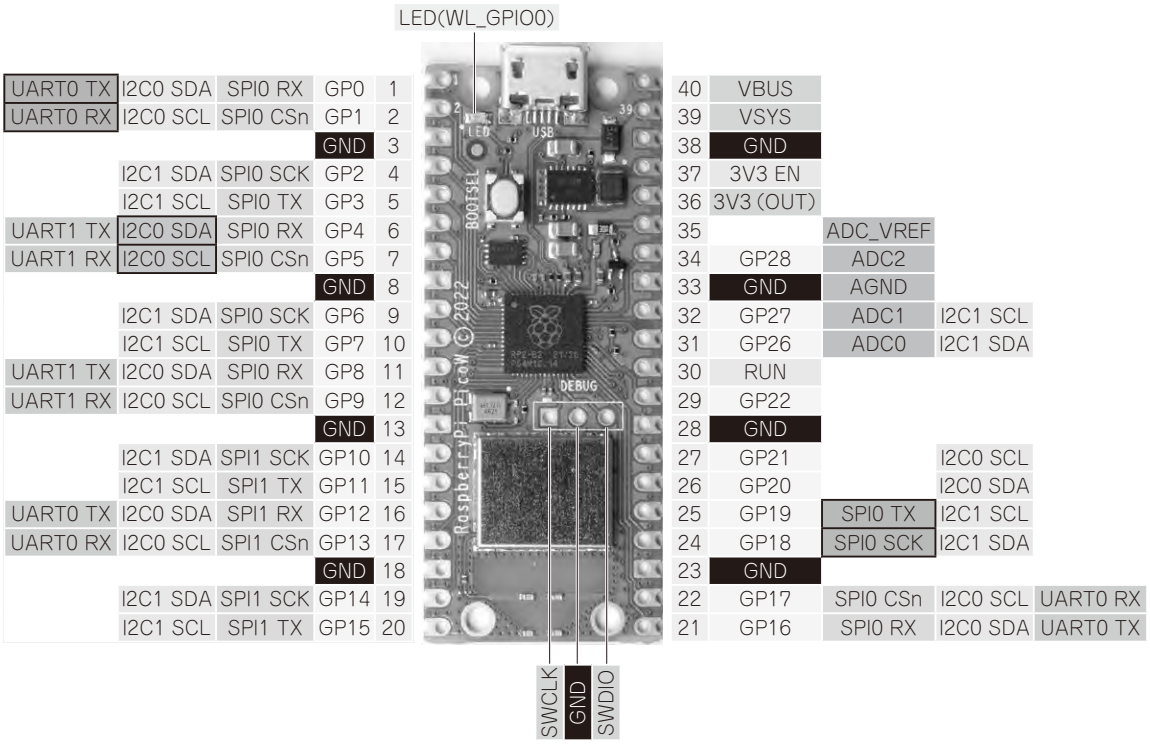


図1(1)　**ラズベリー・パイ Pico Wのピン配置**（枠の付いた信号はデフォルト設定）

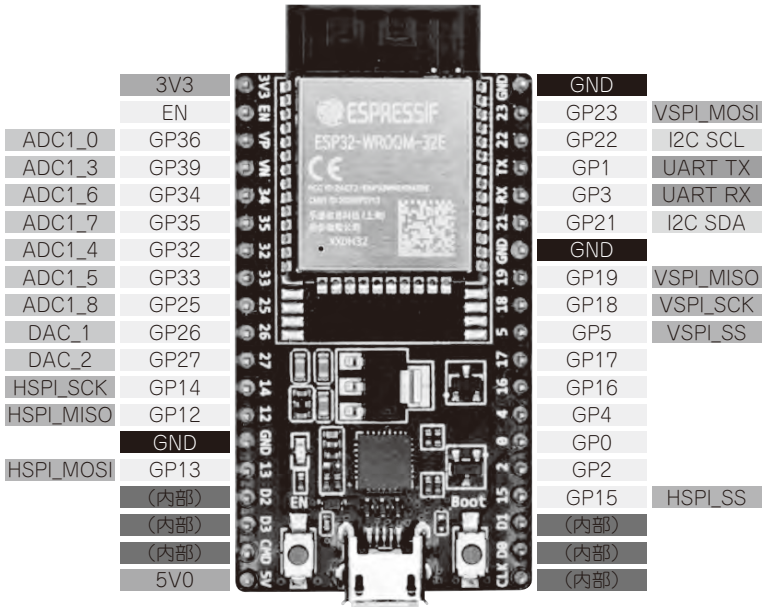


図2(2)
ESP32-DevKitC-32E のピン配置

す．RP2040は2コアのCortex M0+を中心に，I²CやSPI，UARTといった基本ペリフェラルに加え，PIO（Programmable IO）という特徴的なハードウェアを内蔵しています．販売価格が5ドル（販売開始当時）という低価格であることも人気の要因です．

Picoに Wi-Fi機能を追加した新しいモデル Pico Wもリリースされ，インターネットを活用した製作物も作りやすくなりました．

PicoとPico Wの主な仕様を**表1**に示します．またPicoとPico Wのピン配置を**図1**に示します．

表2　ESP32-DevKitC-32Eの仕様

項　目		ESP32-DevKitC-32E
SoCモジュール		ESP32-WROOM-32
CPU	コア	Xtensa LX6
	動作周波数	240MHz
メモリ	SRAM	520Kバイト
	フラッシュ・メモリ	4Mバイト
主要ペリフェラル	GPIO	34
	通信	I²C × 2
		SPI × 4
		UART × 3
		I²S × 2
	A-Dコンバータ	16
	D-Aコンバータ	2
	PWM	モータ用×3 LED用×16
無線通信	通信モジュール	内蔵
	Wi-Fi	IEEE 802.11 b/g/n
	Bluetooth	v4.2 BR/EDR，BLE
外部ポート		USB Micro-B
動作温度		−40〜+80℃
電源電圧		5V
外形寸法		48.2 × 27.9mm

表3　MicroPythonに対応する主要なボード
https://micropython.org/download/を参考に作成

メーカ名/団体名	ボード名(多いときはシリーズ名)
Adafruit	Feather M0/M4 Express, Feather RP2040, Feather nRF52840, ItsyBitsy M0/M4 Express ItsyBitsy RP2040, Qt Py RP2040, Trinket M0
Arduino	Nano 33 BLE Sense, Nano RP2040 Connect, Portenta H7
BBC	micro:bit v1
Espressif Systems	ESP32，ESP32-C3，ESP32-S3，ESP8266
George Robotics	Pyboard v1.0/v1.1, Pyboard D-SF2/SF3/SF6
M5Stack	M5Stack Atom
NXP セミコンダクターズ	MIMXRT EVKシリーズ
PJRC	Teensy 4.0/4.1
Pimoroni	Pico LiPo，Tiny 2040
Raspberry Pi財団	Pico，Pico W
ルネサス エレクトロニクス	RA4M1 Clicker，EK-RA4M1 EK-RA6W1, EK-RA6M1，EK-RA6M2
STマイクロ エレクトロニクス	NUCLEO-F0/F4/F7/G0/G4/H7/L0/L1/L4/WB/WLシリーズ, STM32 Discovery F4/F7/L4シリーズ
Seeed Studio	XIAO nRF52840，XIAO SAMD21, Wio Terminal
Sparkfun	Micromod STM32，Pro Micro RP2040, SAMD51 Thing Plus，Thing Plus RP2040
Wiznet	W5100S-EVB-Pico，W5500-EVB-Pico

● ESP32-DevKitC-32E

ESP32-DevKitC-32Eは，マイコン・モジュールESP32-WROOM-32E（Espressif Systems）とUSB-シリアル変換ICを搭載する開発ボードです（**写真2**）．ESP32-WROOM-32Eは，Xtensa LX6（ケイデンス・デザイン・システムズ）を2コア搭載し，クロック周波数240MHzで動作するという高性能なモジュールです．さらにWi-FiとBluetoothを両方内蔵しながらモジュール単体で500円程度，開発キットでも1600円程度と低価格なので，電子工作用として人気があります．

ESP32-DevKitC-32Eの主な仕様を**表2**に，ピン配置を**図2**にそれぞれ示します．以降ではESP32-DevKitC-32EをESP32と表記します．

● RA4M1-CLICKER（RA4M1-EB）

32ビットCortex-M4マイコン R7FA4M1AB3CFM（ルネサス エレクトロニクス）を搭載したボードです．マイクロテクニカやマルツエレックから入手できます．

● NUCLEO-F446RE

32ビットCortex-M4マイコン STM32F446RET6（STマイクロエレクトロニクス）を搭載したボードです．

● MicroPythonで使えるボードは150種類超!

MicroPythonに対応している国内で入手できる主要なマイコン・ボードを**表3**に示します．これらはボード用にカスタマイズされたMicroPythonのファームウェアが用意されているので，ダウンロードしてボードに書き込めばすぐにMicroPythonを使えます．**表3**に載っているボード以外でも，ソースコードからビルドできるスキルがあれば，Windows上で動くものや，プロセッサ・エミュレータQEMUで仮想化されたArmプロセッサで動作するファームウェアを作ることもできます．

◆参考・引用*文献◆

(1) Raspberry Pi Pico W Pinout，Raspberry Pi財団.
https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf
(2) ESP32-DevKitC V4 Getting Started Guide，Espressif Systems.
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

みやた・けんいち

第3章

ビギナ向け，Pythonの開発環境としても使える

開発環境 Thonny（ソニー）の使い方

宮田 賢一

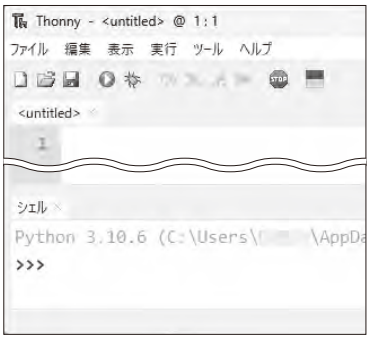


図1　Thonnyを立ち上げたときの画面

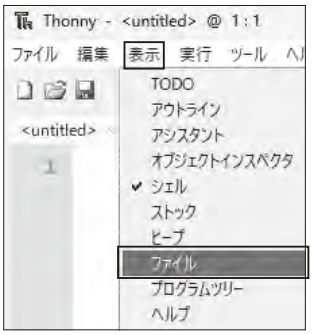


図2　［表示］-［ファイル］とたどる

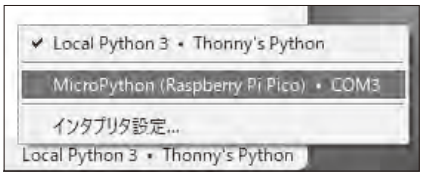


図3　トップ画面右下からマイコン・ボードを指定する

図4　Thonnyの画面（プログラムが入力されている状態）

Pythonの統合開発環境Thonnyの使い方を説明します．最初に，GitHubから筆者提供プログラムをダウンロードします．

```
https://github.com/kemusiro/MicroPythonGuide.git
```

［Code］ボタンをクリックしたらプルダウン・メニューが表示されるので，［Download ZIP］を選択してプログラムをダウンロードします．ダウンロードしたプログラムはZIP形式で圧縮されているので，解凍し，任意の場所に置きます．

Thonnyを起動すると，**図1**の画面が開きます．［表示］-［ファイル］（**図2**）とすることで，トップ画面左側にファイル・タブが表示されます．ファイル・タブ直下には，Cドライブが表示されていると思います．その状態から，ダウンロードしたデータを，C¥…任意の場所…¥MicroPythonGuideとたどり指定します．

次に前章でMicroPythonファームウェアを書き込んだマイコンをPCに接続します（Picoの場合，BOOTSELボタンは押さない）．

Thonnyトップ画面の右下でPicoを選択します（**図3**）．すると，**図4**の画面が開きます．画面は大きく次のパートに分かれます．

● エディタ画面

ユーザがプログラムを入力する画面です．複数のファイルを同時に開いた場合はエディタ画面上部のタ

第1章

基本要素から四則演算，リスト，for文の使い方，
インデントのルールまで

MicroPython チュートリアル

宮田 賢一

この章ではMicroPythonのプログラミングの基本を一通り体験します．

基本となる要素：関数呼び出し，代入，変数

プログラミング言語の学習で大切なのは，動くプログラムを作り，動作を理解することだと思います．まずは簡単なプログラムを通じて，MicroPythonの基本的な使い方を理解しましょう．

● print

Thonnyのコンソール画面で，プログラムの入力待ちを意味する>>>に対して，次のように1行を入力し，行末でEnterキーを押します．

```
>>> print("Hello, MicroPython")
 Hello, MicroPython
>>>
```

すると直後の行にHello, MicroPythonと表示され，次の入力待ちとなりました．MicroPythonではこのように，入力したプログラムがすぐに実行されるので，対話的にプログラムの実行ができます．

この1行プログラムではprint関数を呼び出しています．あらためて用語の意味を定義しておきます．

- 関数…何らかの機能を提供するプログラムをひとまとめにして名前を付け，繰り返し呼び出せるようにしたもの
- 引数（ひきすう）…関数が提供する機能に与えるパラメータ
- 戻り値…関数を実行して得られる結果

print関数は与えられた引数を画面に表示するという機能を提供する，MicroPythonに最初から組み込まれている関数です．戻り値はありません[注1]．

● input

別の関数を使ってみましょう．

```
>>> x = input("Enter a number: ")
 Enter a number:
```

ここで使っているinput関数は，ユーザからの入力を受け取って，その値を戻り値として返す関数です．input関数に文字列の引数を指定すると，ユーザの入力待ちを意味するプロンプトとして表示されます．MicroPythonでは文字列を二重引用符（"Enter a number: "）または一重引用符（'Enter a number'）のいずれかで囲って表します．どちらの形式を使っても意味的な違いはありません．

"Enter a number: "が表示されたのに続いて，10と入力してEnterキーを押します．

```
>>> x = input("Enter a number: ")
 Enter a number: 10
>>>
```

今回は結果が何も表示されずにMicroPythonのプロンプトが表示されました．このプログラムは代入文を実行するものです．代入文は，

変数 = 式

という形をしており，等号 = の右辺の式の計算結果を左辺の変数に代入する働きがあります．この例の場合，xが変数，input関数が式に当たります．

実際に，input関数に対して入力した値10が変数xに代入されていることを確認してみましょう．

```
>>> print ("Entered number is", x)
 Entered number is 10
```

この例のように，print関数にカンマで区切って複数の引数を指定した場合は，それらを空白で挟んで順番に連結して表示してくれます．この場合は確かに変数xの値が10であることを示しています．

● 変数の型は固定でない

C言語のプログラマであれば，変数を使用する前に

注1：厳密には戻り値を参照するとNoneという値が得られますが，今のところは意味がないものとして戻り値なしと表現しました．

第2章

数値型 / 文字列型 / 論理値型 / None 型

基本的なデータ型

宮田 賢一

ここからは，MicroPythonの言語仕様について，プログラム例や実行結果を示しながら説明します．実際にThonnyで打ちながら学んでいきましょう．

1.1 リテラル…プログラムに値を直接表記したもの

データ型の説明を調べていると，リテラルという文法用語を目にすることがあると思います．リテラルとは，プログラム中にデータの値を直接表記したものであり，データ型によって表記方法が決まっています．以下はMicroPythonのリテラルの一例です．

- 整数リテラル：`100, -20, 0x10, 100_000`
- 浮動小数点リテラル：`1.0, .5, 1e3`
- 虚数リテラル：`2j, 3.0j`
- 文字列リテラル：`"abc", f"{param}"`
- バイト列リテラル：`b"abc", b"¥x12¥x34"`

一方，変数名のようにデータの値そのものを表していないものはリテラルではありません．本稿でも必要に応じてリテラルという用語を使いますので，覚えておいてください．

1.2 数値型

MicroPythonでは，次の3種類の数値型を扱えます．

- 整数
- 浮動小数
- 複素数

● 整数型…無限精度で使える

整数は，3や15のように小数点がない数値です．C言語の場合では，整数のビット幅に応じて複数の整数型が用意されていますが，MicroPythonには1種類しかなく，メモリが許す限り無限の精度で整数を扱えます．

例えば，べき乗を求める演算子**を使って，2の1000乗と3の1000乗の足し算も実行できてしまいます（**コマンド1**）．無限精度の整数が扱えるとはいえ，マイコンのCPUで扱える32ビットや64ビットのレジスタ範囲をはるかに超えるデータになるので，32ビット値や64ビット値の計算に比べて実行時間は遅くなります．実際に使用する場合は，適切な用途かどうかを考慮すべきでしょう．

```
>>> 2 ** 1000 + 3 ** 1000
1322070819480806636890455259…
7783138506080619639097776968 7
2080388872986030082751448387 4
7403057464132562505635672985 6
```

コマンド1　2の1000乗と3の1000乗の足し算
メモリが許す限り無限の精度で整数を扱える

数値リテラルには，数字列の任意の場所にアンダスコア_を挿入できます．これにより，人間にとって読みやすい形で桁数の大きな数値を表現できます（**コマンド2**）．

```
>>> 1_000_000    # 1M: 3けた区切り表記
1000000
>>> 1_0000_0000  # 1億: 4けた区切り表記
100000000
```

コマンド2　数字列の任意の場所にアンダスコア_を挿入できる

整数については特別な文字を前に置くことにより基数表現が可能です．

- `0b`または`0B`…基数2（2進数）
- `0o`または`0O`（ゼロ，オー）…基数8（8進数）
- `0x`または`0X`…基数16（16進数）

16進数で10 ～ 15を表現する場合は`a` ～ `f`，または`A` ～ `F`の文字を使います．さらに，基数表現でもアンダスコアによる数値区切りができるので，桁数が大きくなりがちな2進数表記で効果を発揮するでしょう（**コマンド3**）．

```
>>> 0x2b4f
11087
>>> 0b0010_1011_0100_1111
11087
```

コマンド3
特別な文字を前に置くことにより基数表現が可能

第11章

疑似的なマルチタスクのメカニズム

非同期処理

宮田 賢一

10.1 非同期処理とは

ネットワークにリクエストを送信して応答を待ったり，定期的にLEDを点滅させるためにウェイトを入れたりといった処理が必要なプログラムでは，待っている間CPUは何も処理をしていません．

この待っている間に別の処理ができれば，CPUの時間を有効に活用でき，プログラム全体として見ると実行時間の短縮につながることが期待できます．そのために使えるのが非同期処理です．

非同期処理と対比されるのは同期処理ですが，これらの違いを次に示します．

- 同期処理：要求を発行した後，結果が得られるのを待ってから後続の処理を実行する方式
- 非同期処理：要求を発行した後，結果が得られるのを待たずに後続の処理を実行し，結果が得られたかどうかは任意のタイミングで確認する方式

MicroPythonでは，非同期処理のための仕組みとしてコルーチンを採用していて，データの入出力だけでなく，待ち状態が発生しうる処理全般に対して効果を発揮します．

10.2 コルーチン

● 一時中断/再開が可能な処理構造で疑似的なマルチタスクを実現できる

コルーチンとは，途中で一時中断が可能で，かつ中断したところから再開可能な処理構造です．コルーチンが一時中断している間は，別のコルーチンを実行できます．つまり，複数のコルーチンがあるとき，それらが協調して中断と継続を繰り返すことにより，複数のコルーチンを並行して動作させられるということから，疑似的なマルチタスクを実現しているとみることができます．

MicroPythonでは，コルーチンの機能が言語仕様に組み込まれているとともに，uasyncioモジュールをインポートすることにより，コルーチンを扱うためのサポート関数をプログラム中で利用できるようになります．

● 定義

コルーチンを定義するには，次に示すように通常の関数定義をするdef文にasyncというキーワードを付けます．

```
async def コルーチン名(引数, …):
    本体
```

コルーチンは，関数と同じように実行できますが，単純に実行してもコルーチン本体は実行されず，コルーチン・オブジェクトが返されるだけです．

● コルーチンが実行される流れ

いつコルーチン本体が実行されるのかを，非同期処理のライフサイクル（**図1**）を追いながら確認しましょう．

▶①run関数を使って非同期処理の世界に入る

MicroPythonを最初に起動した状態を「同期の世界」とします．同期の世界から「非同期の世界」に入るには，uasyncio.run（コルーチン・オブジェクト）を呼び出します．

```
uasyncio.run(coro1)
```

▶②非同期の世界はタスクで管理される

非同期の世界では，タスクによって実行状態が管理されます．タスクの状態には，コルーチン本体を実行している実行状態と，別のコルーチン本体処理が完了するのを待つ，待ち状態があります．2つ以上のタスクが同時に実行状態になることはありません．

前述のuasyncio.runを実行すると，指定したコルーチン・オブジェクトcoro1に対する新しいタスク（タスク1）が割り当てられて，コルーチン本体の処理を開始します．

▶③タスクの中から新しいタスクを生成する

新しいタスクは，実行中のコルーチン内からuasyncio.create_task関数を使って生成できます．生成された直後のタスクは，待ち状態になります．この関数が返すタスク・オブジェクトを使って，

第1章

Pico W/ESP32単体でネットワークに接続する方法

接続編①…直接Wi-Fiに接続する

宮田 賢一

リスト1　Wi-Fi接続のプログラム`wlan.py`

```
 1: try:
 2:     import network
 3:     import time
 4:
 5:     def init_wlan(ssid, password):
 6:         # ステーション・モード接続用のオブジェクトを生成
 7:         wlan = network.WLAN(network.STA_IF)
 8:         # Wi-Fiインターフェースを有効化
 9:         wlan.active(True)
10:         if not wlan.isconnected():
11:             # Wi-Fiアクセス・ポイントに接続する
12:             wlan.connect(ssid, password)
13:             # IPアドレスを取得するまで待つ
14:             while wlan.status() != network.
                                        STAT_GOT_IP:
15:                 print('waiting...')
16:                 # 1秒待つ
17:                 time.sleep(1)
18:         return wlan
19: except ImportError:
20:     # networkモジュールを持たないボードの場合は何もしない
                                        関数を定義する
21:     def init_wlan(ssid, password):
22:         pass
```

WLANクラスのオブジェクトを返す

SSIDとパスワードを用意

マイコン・ボードをネットワークに接続するとマイコン活用の幅が格段に広がります．第3部では，MicroPythonの最初の実践として，ネットワーク接続の仕方を解説します．

Wi-Fiは家庭内LANやインターネット上のサービスに接続できる汎用的な無線規格です．ボード上にWi-Fi接続用の通信モジュールを搭載しているラズベリー・パイPico W（以降，Pico W）とESP32では，MicroPythonからWi-Fi通信を直接使えます．通信モジュールを持たないラズベリー・パイPico（以降，Pico）でも，他の通信モジュールと組み合わせることでWi-Fiネットワークへの接続が可能となります．本章ではまず前者の直接接続型のプログラムを説明します．

● Wi-Fiへの接続の仕方

Wi-Fiアクセス・ポイントに接続するための関数`init_wlan`を定義します（**リスト1**）．この関数は，指定したアクセス・ポイントのSSIDとパスワードを用いてアクセス・ポイントへの接続を試み，成功すると設定情報を格納したWLANクラスのオブジェクトを返します．プログラムのポイントを見ていきます．

7行目：Wi-Fiアクセス・ポイントへの接続は，`network.WLAN`コンストラクタで作成した`WLAN`オブジェクトを介して行います（関数の詳細は付録第1章を参照）．

12行目：この`WLAN`オブジェクトを使って，最初にWi-Fiアクセス・ポイントに接続します．

14～15行目：Wi-Fiアクセス・ポイントへの接続の完了は，`wlan.status`関数が`STAT_GOT_IP`（IPアドレスを取得した）という戻り値を返すまで，ポーリングで待ちます．

● プログラムの書き込み

`wlan.py`をモジュールとしてインポートできるように，Pico WまたはESP32のプログラム格納用フラッシュ・メモリ上で，`lib/common/network`フォルダに`wlan.py`というファイル名で格納します（**図1**）．

なお`lib/common/network`フォルダ配下の`__init__.py`は，`common.network`モジュールをインポートしたときに自動的に実行されるファイルですが，この内容には次の行を含みます．

```
from .wlan import init_wlan
```

この文は`__init__.py`と同じフォルダにある`wlan`モジュールから`init_wlan`属性（この場合は関数）でインポートすることを意味します．`from`で始まる`import`文は，モジュール名を介さずに属性名を直接参照可能にするので，`init_wlan`関数は`common.network.wlan.init_wlan`ではなく`common.network.init_wlan`として参照できます．

Wi-Fiのアクセス・ポイントに接続するために必要なSSIDとパスワードの情報は，セキュリティ上外部に漏れないように管理すべきものです．そこでこれらの情報を個別のファイル`secrets.py`（**リスト2**）に格納し，ユーザのプログラムからはその定義情報をインポートします．実験環境の準備（第1部第4章）で既にファイルが作られていると思いますが，改めてファイル内容を確認してください．

第4章

双方向通信でLEDのON/OFFを制御

通信編①…MQTT

宮田 賢一

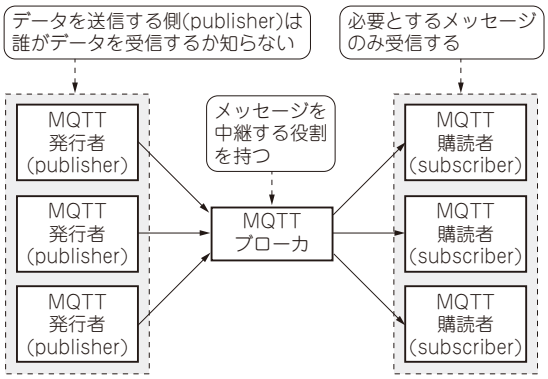


図1　MQTTによる通信方式

本章からは，インターネットを介した通信プロトコルの活用方法を説明します．外付けモジュールなしに直接インターネットに接続できるラズベリー・パイPico W（以降，Pico W）とESP32を使います．

軽量かつシンプルな通信プロトコル「MQTT」

● 通信方式

MQTTとは，軽量・シンプルなデータ転送プロトコルです．軽量というのは送りたいデータ本体以外に必要なメッセージ・ヘッダが最小2バイトと小さいことを意味していて，通信状況が十分とは言えない場所に設置されたセンサから情報をインターネットに送信するような使い方でその効果を発揮します．

MQTTは発行者/購読者モデル（publisher/subscriberモデル）を採用しています．発行者が送信したデータはMQTTブローカが受け取った後，MQTTブローカから購読者にデータが配信されるという仕組みです（**図1**）．この方式のポイントは，発行側は誰が受信するかを気にする必要がなく，購読者側も直接発行者を特定せずにデータを受け取れるということです．

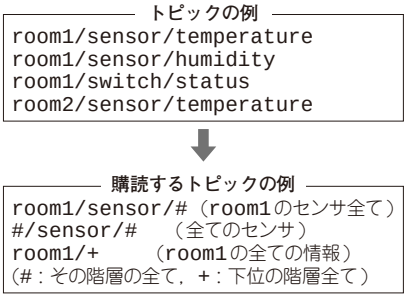


図2　MQTTのトピックの例

● トピックとメッセージ

MQTTでやりとりされるデータは，トピックとメッセージからなります．トピックはメッセージに対するタグのようなものです．購読者側は自分自身をMQTTブローカに登録するときに，受信したいトピックを指定します．**図2**にトピックの例を示します．発行者側はメッセージのトピックを細かく指定するのに対して，購読者側はワイルドカード（#や+）を使って特定の分類に属するメッセージのみ受け取ることができます．

ステップ①…MQTTクライアントのインストール

MicroPythonでMQTTを使うには，最初にMQTTクライアント`umqtt.simple`をインストールする必要があります．MicroPythonではPythonの`pip`モジュールと同じように，ネットワーク上のモジュールを検索してインストールしてくれる専用のモジュール`mip`が用意されているので，これを活用します．`mip`はPico WまたはESP32がネットワークに接続している状態で使用します．

インストールの様子を**図3**に示します．これによりプログラム格納用のフラッシュ・メモリ上に，`lib/umqtt.simple`というフォルダと，関連するMicroPythonファイルが格納されるので，これ以降は`mip`を実行しなくても`umqtt.simple`モジュールを使えるようになります．

第1章

デバイスの仕様書から必要な情報を読み取って
MicroPythonでプログラム化する

シリアル通信（I²C）で出力するセンサ

宮田 賢一

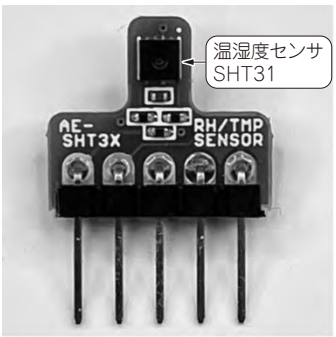


写真1 温湿度センサ・モジュールAE-SHT31（秋月電子通商）

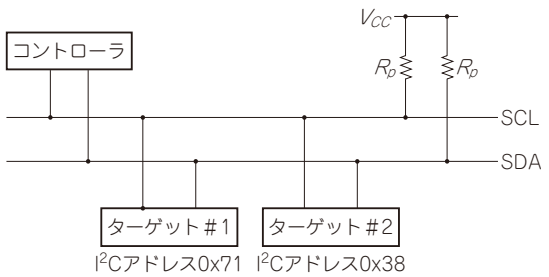


図1　I²C通信のための結線

マイコンとセンサを組み合わせて周辺の状況を数値化することは，マイコンの使い方としては基本的かつ応用範囲も広いものです．また，MicroPythonによるプログラミングとしても得意分野です．

取得できるデータには大きく分けて2種類あります．

- ディジタル・データ：センサからのデータがディジタル数値として直接得られるもの．直接マイコンに取り込める
- アナログ・データ：センサからのデータがアナログ量（電圧値，抵抗値など）であるもの．マイコンに取り込むためにディジタル値に変換しなければならない

MicroPythonで個々のセンサを扱うためのライブラリは，センサ提供元や一般の有志によって公開されているものが多くあります．しかし本章では，MicroPythonの学習のためにライブラリを使わず，自分自身で実装する方法を学びます．手法をマスタすれば，未知のデバイスや誰も触ったことがないレアなデバイスであっても，仕様書を読めばプログラミングできるようになるでしょう．

ここでは，**写真1**の温湿度センサ・モジュールを例に，MicroPythonでセンサを制御する方法を体験します．

I²Cのあらまし

● 2線式のシリアル通信規格

シリアル通信の規格の一種として，I²C（Inter-Integrated Circuit）があります．シリアル通信とは送信するデータをビット単位に順に転送する方式で，複数のビットをまとめて送るパラレル通信と対比される方式です．

I²Cは2本の信号線を使ってデータの送受信を行います．信号線の1つはクロック信号を伝達するSCL（Serial Clock），もう1つはデータを伝達するSDA（Serial Data）と呼びます．データはSCLのクロックに同期して1ビットずつSDAを流れていきます．

I²Cを使用するときの結線を**図1**に示します．SCLとSDAはプルアップ抵抗R_pを通して電源V_{cc}に接続しなければなりません．データ転送はいわゆるコントローラ・ターゲット方式であり，コントローラとなるデバイス（マイコン）が主体となって，ターゲットとなるデバイス（センサやキャラクタ・ディスプレイなど）からデータを読み出したり，データを書き込んだりします．

I²Cデバイスは固有のアドレス情報を持っているので，複数のターゲットを同じ信号線上に接続しても，データ送受信時にI²Cアドレスを使って対象のターゲット・デバイスを選択できます．データ転送用の信号線はSDAの1本しかないので，データの送受信が

第2章

SPI接続で定番の制御チップSSD1331を操作

グラフィックスLCDで波形や文字を表示する

宮田 賢一

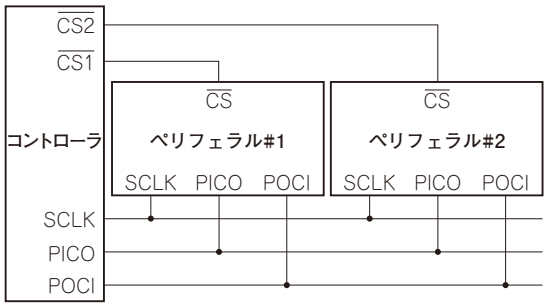


図1　SPIの構成…クロック，データ2本，チップ・セレクトの4本の信号線を使う

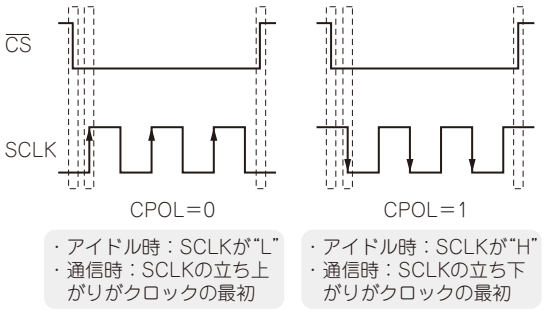


図2　クロック極性（CPOL）の違い

この章ではSPIで制御するグラフィックスLCDのプログラミングを説明します．

シリアル通信規格「SPI」の基礎知識

● 全二重通信が可能な4線式シリアル通信規格

SPI（Serial Peripheral Interface）はシリアル通信の規格の一種です．同じシリアル通信のI²Cとは信号線の数が異なり，I²Cが3本なのに対してSPIはSCLK，PICO，POCI，CSの4本を使用します（**図1**）．SPIにはコントローラとペリフェラルがあります．一般的には1つのマイコンがコントローラになり，これに複数のデバイスがペリフェラルとしてつながります．

SCLK（Serial Clock）はクロック信号をコントローラからペリフェラルに伝達します．PICO（Peripheral In/Controller Out）はコントローラからペリフェラルの方向にデータを伝達し，POCI（Peripheral Out/Controller In）はペリフェラルからコントローラの方向にデータを伝達します．つまりコントローラ-ペリフェラル間は全二重でのデータ送受信が可能です（同時に双方向の通信が行える）．どちらか1方向のみの通信しか必要ない場合でも，SPIバス上は逆方向の通信もダミーで行われています．

SPIでもI²Cと同じように1つの信号伝達バスに複数のペリフェラルを接続できますが，I²CではI²Cアドレスによってペリフェラルを区別するのに対して，SPIではCS（Chip Select）信号を使ってどのペリフェラルと通信するかを選択する方式となります．もしペリフェラルが1台しかなければ，CS信号を固定してしまうことで，3本の信号線で制御できます．

特性としては，I²Cが比較的低速なデータ転送向け（センサからのデータ読み取りなど）なのに対して，SPIは比較的高速なデータ転送（例えばグラフィックス・ディスプレイへのデータ送信など）に向いています．

● デバイスごとに意識すること

SPIでデータ転送を行う場合，デバイスごとに定められているクロック極性とクロック位相の2種類のパラメータを考慮してプログラムを作成しなければなりません．

▶クロックの極性（CPOL）

クロックの極性（polarity）です．SCLKの立ち上がりと立ち下がりのどちらがクロック・パルスの最初になるかを意味するフラグです（**図2**）．通信が行われていないアイドル時にSCLKが"L"の状態で，クロックの立ち上がりでクロックを開始する場合はCPOL=0であり，逆にアイドル時のSCLKが"H"で，クロックの立ち下がりでクロックを開始する場合はCPOL＝1となります．

▶クロックの位相（CPHA）

クロックのどの位相（phase）でPOCIまたはPICOのデータをサンプリングするかを意味するフラグです

第1章

①アナログ・センサ計測，②小型ポンプ制御，
③クラウド連携

自動水やりシステムの製作

宮田 賢一

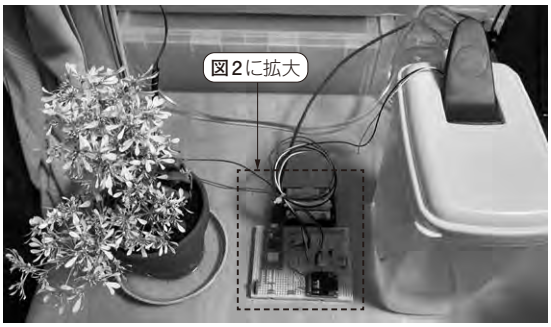


写真1　Wi-Fiマイコンの良さを生かしてクラウド連携システムを作る

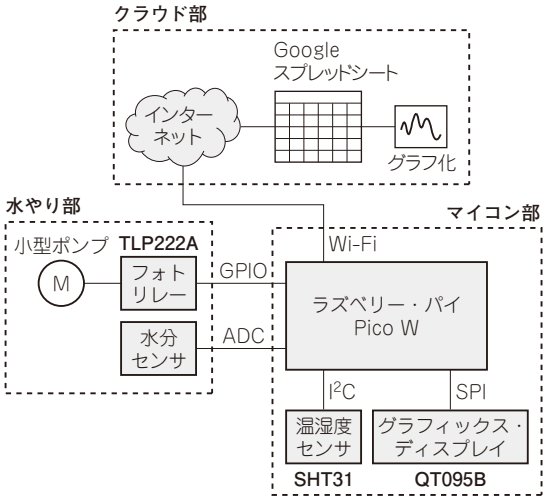


図1　自動水やりシステムの全体構成

Wi-Fi接続やLCD表示，センサ・データ取得など，これまで説明してきたさまざまな要素を組み合わせて，実際に運用できるシステムを構築します．

植物を育てるときの水やりは，やり過ぎてもやらなさすぎても植物の生育にとって良いことではありません．特に屋内の観葉植物の場合は，水やりをつい忘れてしまい，枯らしてしまうこともあるかもしれません．そこで水やりを自動化するシステムを作ります（**写真1**）．

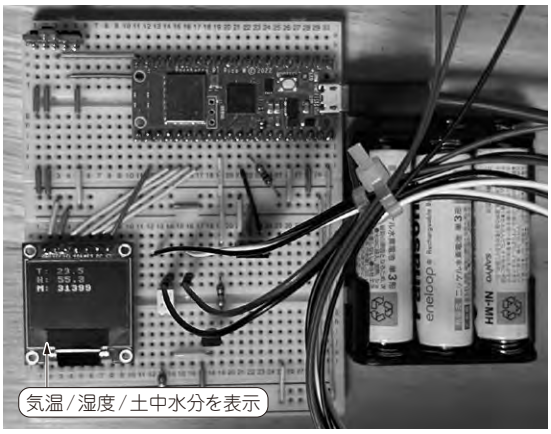


写真2　マイコンと周辺回路
ラズベリー・パイ Pico Wと小型ディスプレイ，バッテリなど

● 作るもの

今回作るシステムに持たせる機能は次の通りです．

- 土の水分量を定期的に監視し「乾いた」と判断したら水をやる
- 育成環境の監視のため，周辺の気温/湿度を計測する
- 計測したデータは画面上に表示すると同時にクラウドに転送し蓄積する
- クラウドに蓄積したデータをグラフ化する

これを具体化したシステムを**図1**に示します．

システム構成

● マイコン部

水やりシステムの全体を制御する部分です（**写真2**）．マイコンはラズベリー・パイ Pico Wを使用し，センサによる計測，小型ポンプの制御，インターネットとのデータ通信を処理します．また，マイコン部には周辺環境の温度と湿度を取得する温湿度センサSHT31（センシリオン）と，取得したデータの現在値を表示するグラフィックス・ディスプレイQT095B（Shenzhen Taida Century Technology）を含みます．

イントロダクション

コンパイル不要，シンプルな言語仕様，
豊富なライブラリ…

MicroPythonがプロトタイプ開発に向く理由

角 史生

表1　本付録で扱うプログラムのテーマ

章番号	テーマ
1	開発環境の準備
2	スイッチ入力/ボリューム入力
3	センサ入力
4	PWM出力で光る/動く/鳴る
5	ディスプレイ出力
6	シリアル通信
7	シリアル通信接続例
8	ファイル操作
9	クラウド通信/サーバ機能
10	スリープ機能
11	タイマ機能
12	コード改善
13	メモリ管理

MicroPythonは，プログラミング言語Pythonの実装の1つで，マイコン上での動作に最適化された言語処理系です．Python3と高い互換性を持っているので，マイコンに慣れていない初心者でも開発しやすいという特徴を持ちます．

特別付録では，スイッチをつなぎたい，センサ値を読みたい，LCDに表示したいなどの目的別にMicroPythonプログラムを紹介します（**表1**）．

Wi-Fi/Bluetooth接続機能を持ち，数百円で購入できる無線マイコン・モジュールESP32-WROOM-32E（Espressif Systems）を実際に動かしながら解説します．

MicroPythonがプロトタイプ開発に向いている理由

MicroPythonはリソースの少ないマイコン上でPython3と同じようにプログラミングできる環境の実現を目指して開発されました．MicroPythonの特徴が活かせる開発用途としてプロトタイプ開発が挙げられます．プロトタイプ開発では，試作，テスト，修正を繰り返しながら開発が進みますが，MicroPythonを用いることで得られるメリットを次に整理します．

● 理由1…対話インタープリタ・モードが使える

Python/MicroPythonはインタープリタ型言語です．インタープリタは，マイコンやコンピュータで解釈できるように変換し実行する機能があります．このため，コンパイル不要で実行したいコードや確認したい変数をコンソールから入力すると，すぐに実行され結果が得られます．この機能は対話インタープリタ・モードREPLと呼ばれます（Read，Evaluate，Print，Loop）．対話インタープリタ・モードを活用することでプロトタイプ開発を効率良く進めることができます．

例えばCMOSカメラ・モジュールなどの周辺機器を制御するソフトウェアを開発する際に，コンパイル型言語と，インタープリタ型言語による難易度の違いを比較してみましょう．

まず仕様書やサンプル・コードを参照して，制御レジスタの操作方法や設定パラメータを理解し，テスト・プログラムを作成します．テストを繰り返しながら希望した動作になるまでソフトウェアを修正します．

▶コンパイル型言語での開発の場合

C言語やArduinoなどのコンパイル型言語でのソフトウェア開発を**図1**に示します．

1. PC上でプログラムを作成
2. コンパイラによりマイコンで実行可能なファイルに変換（コンパイル）
3. 変換したバイナリ・ファイルをマイコンのフラッシュ・メモリに書き込む
4. プログラムを実行
5. 実行結果を確認し，問題があれば1に戻る．希望した動きになるまでプログラム修正，コンパイル，テストを繰り返す．

▶インタープリタ型言語による開発の場合

インタープリタ型言語による開発の例を**図2**に示します．

1. プログラムをREPLに入力，または，コピー＆ペースト．
2. インタープリタによりプログラムを実行．
3. 実行結果を確認し，問題があれば1に戻る．希望

第2章

スイッチやボリューム，ロータリ・エンコーダを接続

入力検出

角 史生

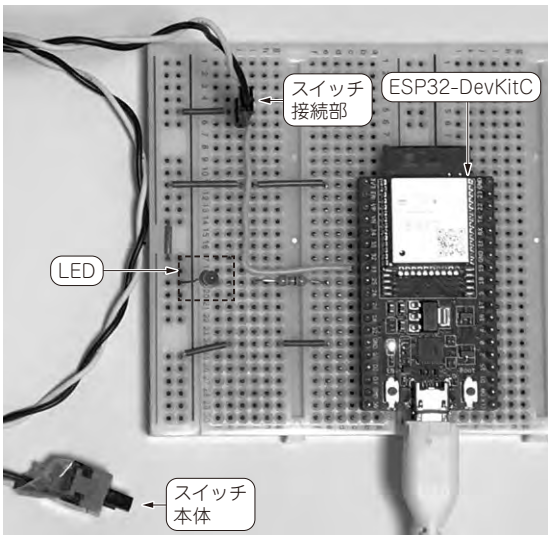


写真1　ESP32-DevKitCとスイッチ，LEDをブレッドボード上で接続した様子

第2章はマイコンへの入力手段をまとめます．具体的には，スイッチによるディジタル入力，ボリュームとA-Dコンバータによるアナログ入力，ロータリ・エンコーダによるディジタル入力を説明します．

1-1 スイッチを押したときだけLEDが点灯する

初めにスイッチ操作に連動してLEDを点灯させる例を示します．GPIOを入力，出力として使う場合の設定を示すとともに，ポーリング方式によるLED点灯および，割り込み方式によるLED点滅を行います．

● 回路

最も基本的な，スイッチを押したときにLEDが点灯する例を示します．回路図を**図1**に示します．**写真1**に示すのはブレッドボードを用いた試作回路です．特別付録で例として示す回路はいずれも，ESP32の電源をUSBケーブルから供給することを前提にしています．

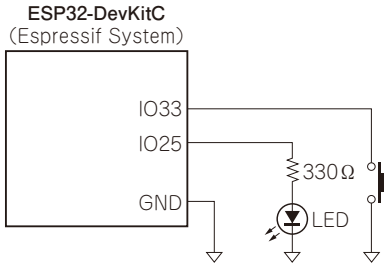


図1　LEDとの接続回路

● プログラム

MicroPythonのプログラムを**リスト1**に示します．LEDを制御するためIO25を出力に設定し，スイッチの状態を取得するためIO33を入力に設定しています．IO33は内蔵プルアップ抵抗を有効に設定することで，スイッチのプルアップ抵抗を省略可能にしています．スイッチが押されていないときは内蔵プルアップ抵抗により3.3Vとなり，スイッチが押されるとGNDに接続され0Vになります．これによりIO33の入力値が1から0に変化します．

スイッチに接続されたIO33の入力値を無限ループ内で取得し，値が0になるとスイッチが押されたと判断して，LEDを点灯させるためにIO25の出力を1に設定します．スイッチの接点が離れた場合，IO33の値が1になり，LEDを消灯させるためにIO25の出力を0に設定します．

1-2 スイッチを押すごとにLEDが点灯/消灯する

● 回路

回路は1-1項と同じものを使用します．プログラムでスイッチを押すごとにLEDが点灯/消灯を繰り返すようにします．

● プログラム

リスト2にプログラムを示します．1-1項のプログラムでは，無限ループ内でスイッチの状態を問い合わ

第11章

現在時刻の表示やウェイト / タイマ処理の使い方

日時の取得と時間管理

角 史生

リスト1　localtime関数を使用して現在時刻を表示するプログラム
UTC（協定世界時）を取得する．取得したUTCに9時間を加算することでJST（日本標準時）を計算している

```
import ntptime
import utime

ntptime.settime()           # NTPによる時刻同期
JST_OFFSET = 9 * 60 * 60    # JST = UTC + 9H(32400秒)

# 時刻を取得、UTCで表示する
(year, month, day, hour, min, sec, wd, yd) =
                                    utime.localtime()
print(f"{year:4d}-{month:02d}-{day:02d}
              {hour:02d}:{min:02d}:{sec:02d} (UTC)")

# 時刻を取得、JSTで表示する
(year, month, day, hour, min, sec, wd, yd) =
          utime.localtime(utime.time() + JST_OFFSET)
print(f"{year:4d}-{month:02d}-{day:02d}
              {hour:02d}:{min:02d}:{sec:02d} (JST)")
```

本章ではMicroPythonで日時の取得，時間管理をする方法を解説します．

9-1 NTPサーバと同期して時刻を表示する

● サーバと同期して正しい時刻を表示する

標準ライブラリのntptimeモジュールを用いることで，時刻管理を行うNTP（Network Time Protocol）サーバとの同期を行えます．NTPを利用するにはESP32がインターネットに接続されている必要があります．接続方法は本付録の第9章か，文献(2)のネットワーキングの章を参照してください．

● プログラム

settime関数によりNTPサーバと同期し，utimeモジュールのlocaltime関数を実行することでUTC（協定世界時）を取得できます．しかし，この段階では，日本の標準時であるJST（日本標準時）とは9時間の時差があります．この時差を加算することでJSTにします．

MicroPythonには地域情報（ロケール機構）のライブラリが実装されていません．そこで，utime.

リスト2　sleep関数を使用したウェイト処理の例

```
import utime
utime.sleep(3)       # 3s待つ
utime.sleep_ms(100) # 100ms待つ
utime.sleep_us(100) # 100μs待つ

print("wait for 1sec...")
utime.sleep(1)
print("done")
```

time関数を用い，取得したUTCの時間に32400秒［60（秒）×60（分）×9（時間）］を加算することでJSTに変換します．

本プログラムを**リスト1**[注1]に示します．実行した結果は次のようになります．上段がUTCで下段がJSTです．

```
2022-04-29 00:09:36 (UTC)
2022-04-29 09:09:36 (JST)
```

9-2 ウェイト処理をする

● 処理の実行を待機したいときに使う

utimeモジュールのsleep関数を使うことで，s単位，ms単位，μs単位で待機します．

● プログラム

プログラムを**リスト2**に示します．utime.sleep，utime.sleep_ms，utime.sleep_usのそれぞれの引数に数字を入力することで，s単位，ms単位，μs単位で待ち時間を指定できます．

9-3 タイマを使って一定周期でLEDを点滅する

● 一定周期で処理を実行する

machineモジュールのTimerクラスを用いるこ

注1：**リスト1**は文字列の整形にformat関数を使わず，フォーマット済み文字列リテラル（f-string）を使っています．f-stringを使うことで文字列の整形指示が簡潔に行えます．2022年4月時点の最新版ファームウェア，v1.18（2022-01-17）で動作確認しています．