Raspberry Pi Pico W ではじめるロボティック・システム

目　　次

1. 環境構築 ＆ デジタル基礎 2
1.1.開発環境を整える....2
1.2.入出力制御 5
1.3.デジタル出力....8
1.4.デジタル入出力....10

2.アナログ I/O....13
2.1.PWM で LED の明るさ調整（アナログ出力） 13
2.2.アナログ入力をしてみよう(温度センサ、照度センサ)....15
2.3.目標の値に出力を操作しよう(LED、照度センサ) 17
2.4.加速度センサを使おう(I²C 通信)....19
2.5.DC モータ制御（PWM 応用） 21

3.ネットワークと Web 連動 24
3.1.Wi-Fi 接続 ＆ IP アドレスとは？....24
3.2.ウェブサイトの基礎を学ぼう....26
3.3.Pico W で Web サーバ経由の LED を制御をしてみよう....31
3.4.Pico W でセンサの状態を web ページから見れるようにしよう....34
3.5.スライダで光の強さを変えよう....35

4.ロボティック・システムを作ろう....38

資料　 MicroPython の基本文法 42
1. インデント....42
2. コメント 42
3. print( ) 43
4. 変数....43
5. 条件分岐 45
6. while ループ 47
7. for ループ 49
8. 関数....52
9. データ型....56
10. 演算子 59
11. リスト....63
12. ロボティック・システムのプログラミング 65

注）Raspberry Pico W の詳細な説明、部品の使い方、応用については、下記の URL が詳しい.

SunFounder Raspberry Pi Pico W Ultimate Starter Kit with Online Tutorials, RoHS Compliant - Kepler Kit | SunFounder

1. 環境構築 & デジタル基礎

みなさんが普段使っている スマートフォン、ゲーム機、家電などの中には必ず小さなコンピュータ（マイコン）が入っています．今回使う Raspberry Pi Pico W も、その仲間です．手のひらサイズですが、プログラムを書き込めば LED を点けたり、スイッチを押したら動いたり、インターネットに接続してパソコンから遠隔で操作できたりします．

この実習ではこのマイコンを使って、外部の状態を計測しながらモータや LED などをコントロールしたり、外部から遠隔で電気をつけたりすることを目指します．そして、最終的には自分でロボティック・システム、IoT（Internet of Things）を作ることを目指しています．オリジナルの作品をぜひ作って楽しんでください．

1.1.開発環境を整える

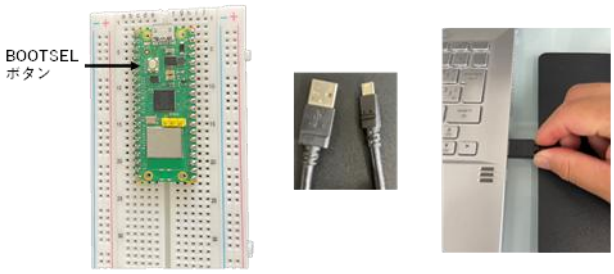
本項では Raspberry Pi Pico W に MicroPython を書き込み、プログラムを実行できる状態にする．Thonny をインストールし、Python のプログラムを作成・実行できるようにすることを目指します．

1) MicroPython のインストール（初回のみ）

MicroPython（マイクロパイソン） とは、小さなコンピュータ（マイコン）上で動くように軽量化された Python プログラミング言語です．普通の Python（PC やスマホで使うもの）は大きくて重いため、メモリや処理能力が限られた Raspberry Pi Pico W では動きません．そこで作られたのが MicroPython で、ほとんどの文法は通常の Python と同じなので、初心者でも学びやすく、マイコンを簡単に動かせるのが特徴です[1]．

① Pico W とパソコンを接続する

まずラズパイ Pico W に USB ケーブルを接続してください．その後、ラズパイ Pico W の BOOSTEL ボタンを押しながらお手持ちのパソコン側に USB ケーブルを接続してください．



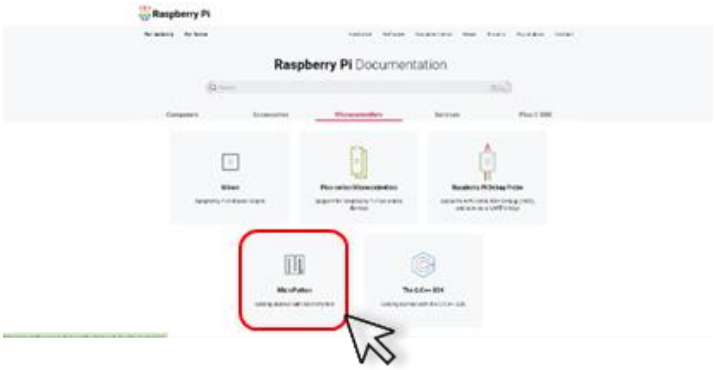
② ファームウェアのページにアクセスする

パソコンで外部デバイスとして認識された、Pico W を開き内部ストレージの HTML ファイルをダブルクリックし、ファームウェアのページにアクセスしてください．

[1] MicroPython はインタプリタ言語であり、実行速度は遅い．ドローンの姿勢制御など、高速に Raspberry Pi Pico W を動かす必要がある場合にはコンパイラ言語である C 言語あるいは C++言語を用いる必要がある．

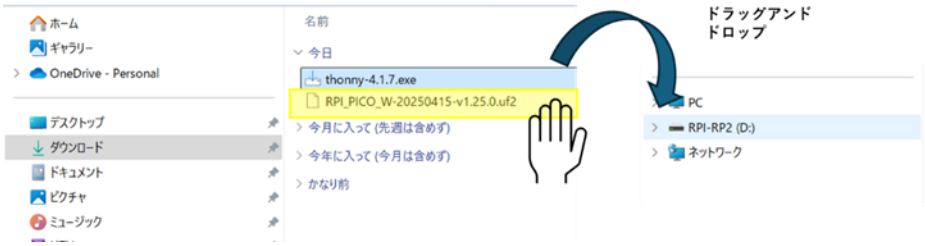
③ MicroPython のファームウェアをダウンロード

Pico W を python で開発するために必要な Micro Python ファームウェアをダウンロードします．先ほど、開いた HTML ウェブサイトから MicroPython をクリックします．すると自動でダウンロードが始まるのでダウンロードが終了するまで待ってください．



④ Pico W に MicroPython のファームウェアを書き込む

ダウンロードフォルダに先ほどダウンロードした MicroPython ソフトウェアがあります．これを Pico W ストレージにドラッグアンドドロップし、Pico W にファームウェア[2]を書き込んでください．



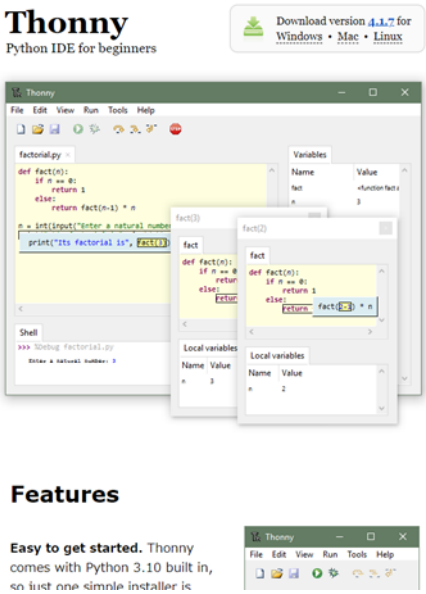
2) Thonny のインストール

Thonny とは、Raspberry Pi Pico W のような小さなコンピュータに、Python でプログラムを書いて、動かすための簡単で使いやすい IDE ソフトです．ここからプログラムを Pico W に書き込むことで LED を光らせることができます．

① Thonny の公式サイトにアクセス

[2] 機器内部のハードウェアを直接制御するために組み込まれた基本ソフトウェアで、ここでは Raspberry Pi Pico W のハードウェアをプログラムで制御しやすくするための基本的なツールを提供する．

Google などのブラウザの検索エンジンを用いて Thonny のダウンロードページ(https://thonny.org/)にアクセスします.

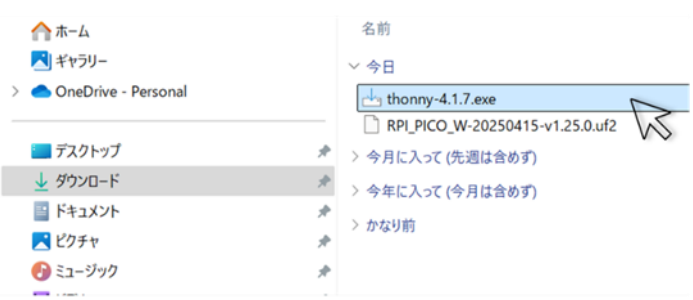


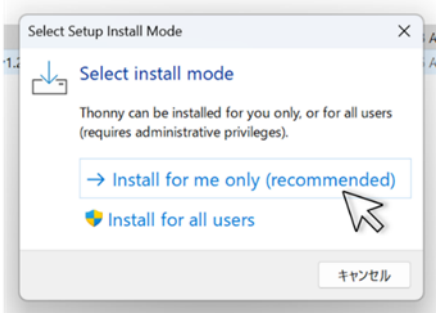
② OS に合ったインストーラをダウンロードしてインストール

Download の文字の近くに Windows・Mac・Linux という文字があります. このうち自分のパソコンの OS と会う文字をクリックしてください. recommended for you と書かれた青字のリンクをクリックすることでインストーラをダウンロードすることができます.

③ Thonny をインストール

ダウンロードフォルダ内の Thonny のインストーラをダブルクリックして Thonny をパソコンにインストールしてください.





④ デスクトップアイコンから Thonny を起動

以上で Pico W の開発環境の設定は終了です.

1.2.入出力制御

コンピュータやマイコンは、入力（Input）と出力（Output）を扱うことで外の世界とやり取りをします. キーボードやセンサは「入力」、ディスプレイや LED、モータは「出力」の代表例です. Raspberry Pi Pico W も同じで、プログラムを通じて外部との入出力を制御することができます. この項では、最も基本的な入出力の体験として Pico W に搭載されているスイッチと LED のコントロール方法を学びます.

1) Hello World を出力する

プログラムを学ぶ際初めに”Hello World!”を出力するのが慣習となっています. さあ、プログラムを始めましょう！これができたらあなたもプログラマです.

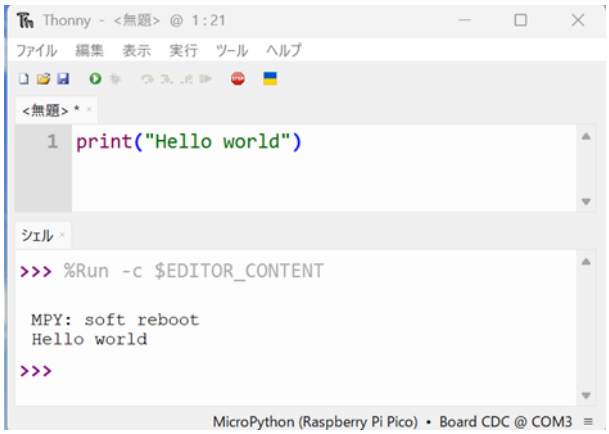
① Thonny とラズパイ Piko W を接続する

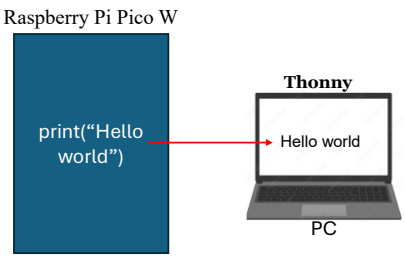
② Thonny の設定を行う

右下にある「Python バージョン表示」をクリックして、「MicroPython (Raspberry Pi Pico)」を選択します. 接続されている Pico W が認識されていれば、Thonny のシェルに「>>>」と表示されているのでそこから選択します.

③ Thonny のエディタに、以下のコード「print(“Hello world”)」を入力

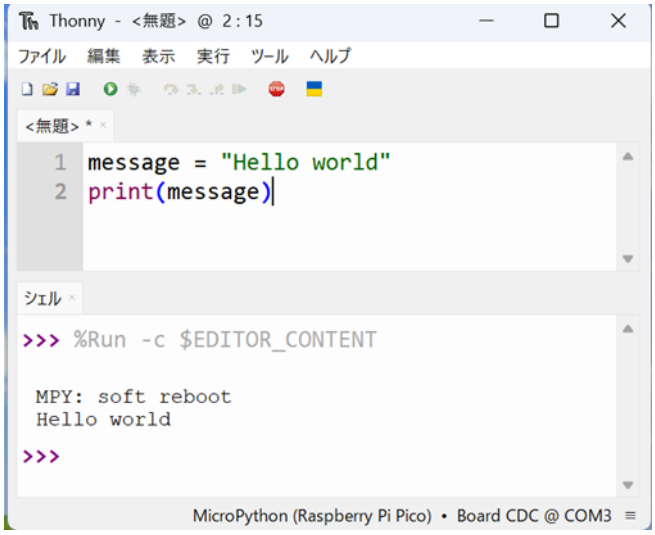
「▶（実行）」ボタンをクリックすると、下のシェルに Hello, World! と表示されます！下図右に示すように、あくまで”print(“Hello world”)”は Raspberry Pi Pico W のメモリ上で実行されていることに注意してください.





④ 変数を活用する

Message という変数を作成し、"Hello World”を変数に格納しましょう. これによって、message を出力することで”Hello World”が出力されます.



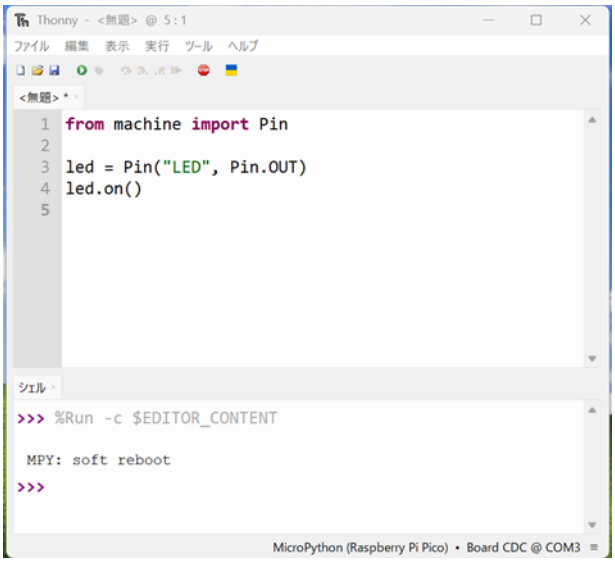
2) LED をコントロールする

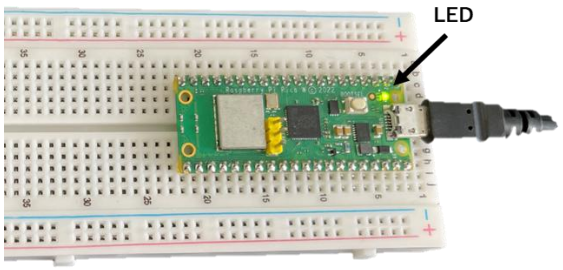
① Thonny のエディタに、以下のコードを入力

「▶（実行）」ボタンをクリックすると、Pico W に搭載された緑色 LED が点灯します.

Pin("LED", Pin.OUT)は Pico W に搭載されている「LED」というピンを扱う準備をします. Pin.OUT は「出力モード」を意味します. つまり「このピンから電気を流す」ということです. 結果として、変数 led は「LED を操作できるスイッチ」のような役割を持ちます.

led.on()は led に対して「電気を流せ」という命令を送っています. これによって実際には Pico W の内部で led に対して電気が流れ、基板上の緑色 LED が光ります.





② LED を 1 秒ごとに点滅させる

Thonny のエディタに、以下のコードを入力.

time.sleep(1)は 1 秒間プログラムを Wait させるコマンド. この数字を変えることによって実行時間をコントロールすることができます.

led.on()によって LED を点ける→1 秒間待つ→ led.off()によって LED を消す

↑これを while 文によって繰り返すことによって、LED を 1 秒ごとに点滅させることができます.

この time.sleep の時間を変えてどのような変化が起こるか観察してみましょう.

```
from machine import Pin
import time

led = Pin("LED", Pin.OUT)

while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

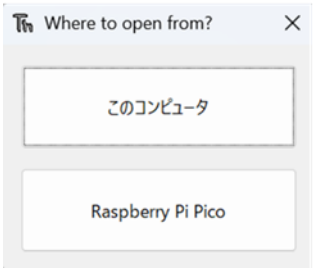
③プログラムをマイコンに保存する

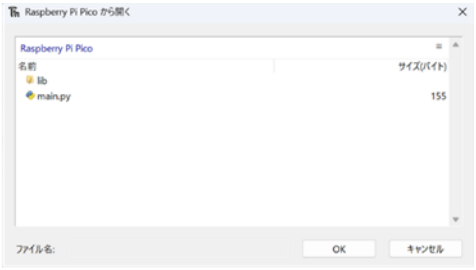
プログラムをマイコンに保存することによって、そのプログラムがパソコンに接続しなくても Pico W に電源が供給されると毎回実行されるようになります.

「💾ボタン」をクリックし保存先をラズパイ Pico W を選択します. この時、"**main.py**"と

いう名前で保存することを忘れないでください. main.py が最初に実行されるプログラムになります.

これによって、「▶(実行)」ボタンをクリックしなくても Pico W に電源を供給すると LED が点滅するようになります.





④ スイッチを使って LED をコントロールする

Thonny のエディタに、以下のコードを入力.

rp2.boostel_button()[3]はボタンが押されていない時は 0 であり、押されると 1 として認識されます.

BOOSTEL ボタンを押すと if 文の中身が実行され、LED が点灯します.

例えば、ボタンが押されているとき、rp2.boostel_button()は 1 なので、上の if 文が実行され、led.on()が実行されます. これによって LED が光ります.

[3] rp2.boostel_button()はどのクラスにも属なさいトップレベルのメソッド(関数)で、BOOTSEL ボタンの押下状態を取得する

```
from machine import Pin
import time

led = Pin(“LED”, Pin.OUT)

while True:
    if rp2.bootsel_button() == 1:
        led.on()
    else:
        led.off()

    time.sleep(0.1)
```

1.3.デジタル出力

これまでの項では、Pico W に標準で搭載されている機能を使い、内蔵 LED の点灯や内蔵スイッチの動作確認を行いました.

しかし、Pico W の本来の使い方は、外部のピンに電子部品を接続し、センサやスイッチを増設したり、LED やモータなどを制御することにあります.

本項では、その第一歩として デジタル入出力の基本 を学びます. 具体的には、外部にスイッチと LED を接続する簡単な回路を組み、Pico W からの制御によって入出力を確認していきます.

1）外部 LED を 1 秒ごとに点滅させよう.

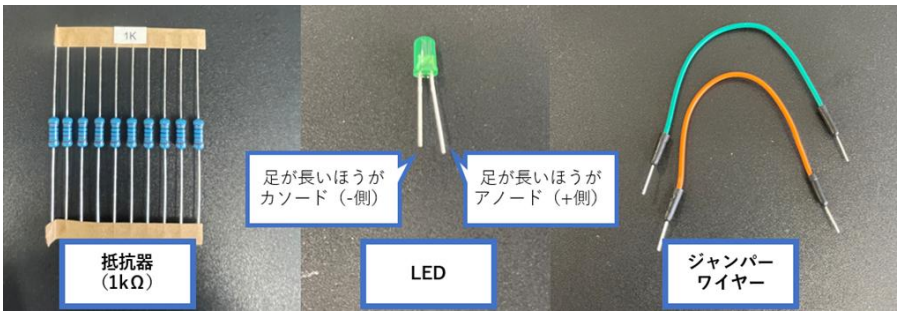
① Thonny で新規ファイルを作成する

② 下記のプログラムを入力する

```
from machine import Pin    # GPIO ピンを使うための machine モジュールの Pin クラスを読込む
import time                #  時間を扱うためのモジュールを読み込む

led = Pin(14, Pin.OUT)     # 14 番ピンを「出力モード」で使うように設定、led オブジェクトとして保存

while True:                # 「ずっと繰り返す」ためのループ
    led.on()               # led インスタンス内の on 関数を使い、LED を点灯する
    time.sleep(1)          # 1 秒間待つ（LED が 1 秒間ついたままになる）
    led.off()              # off 関数を使い、LED を消灯する（電気を止める）
    time.sleep(1)          # 1 秒間待つ（LED が 1 秒間消えたままになる）
```

注）led 変数を使わずに、7 行目を Pin(14, Pin.OUT).on()としても良いが、呼び出す毎にインスタンスの再設定がされるため、

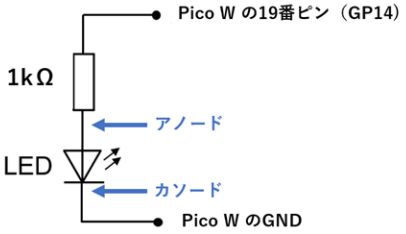
③ 下記の部品を準備する

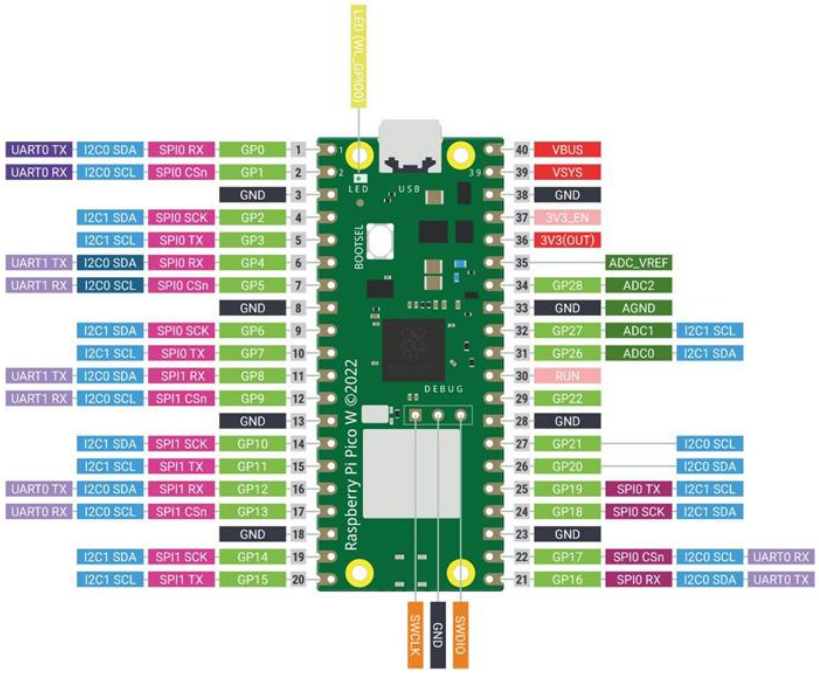


④ 下図の回路をブレッドボード上で作成します．

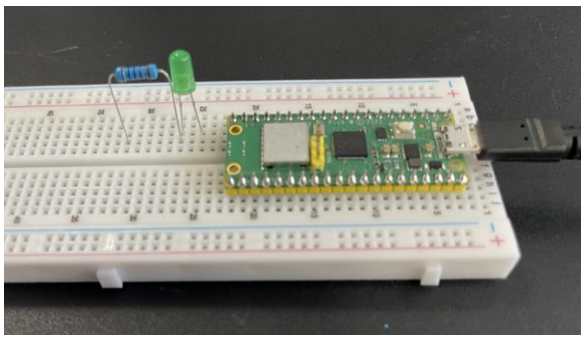
LED（発光ダイオード）は、電流が アノード（＋側）→カソード（－側） に流れると光ります．LED はダイオードであるため向きが逆だと電流が流れないため光りません．これは LED やダイオードは PN 接合といわれる P 型半導体と N 型半導体 を接合して作られています．順方向に電圧をかけると、P 型の正孔と N 型の電子が引き寄せられ、結合して電流が流れます．LED の場合、このときのエネルギーの一部が光として放出されます．そのため、LED は必ず極性（向き）を考えて回路に接続する必要があります．基本的に LED は足が長いほうがアノード（＋側）です．

LED は直接電源につなぐと、大きな電流が流れて破損してしまいます．これは LED は抵抗が非常に小さいためです．そこで 直列に抵抗（ここでは 1kΩ）を入れて、流れる電流を制限 します．

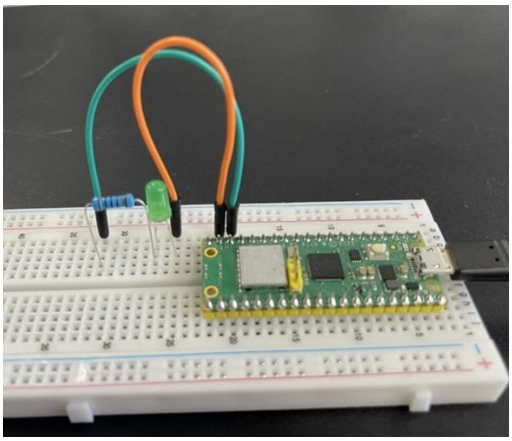




Pico W のピン配列を上記に示します．上から数えて 19 番目のピン GP14 につないでください．LED、抵抗器をブレッドボード上に写真のように刺してください．



その後、ジャンパーワイヤを Pico W の右から 19 番目のピン(GP14) ー LED（アノード）間、抵抗器（LED の反対側）ー Pico W GND ピンにさしてください.



⑤ プログラムを書き込み実行する

現在のスクリプト（プログラム）を実行をクリックし LED が 1 秒ごとに点滅することを確認してください.

これは、プログラム上の 14 ピンが、Pico W の 19 番ピンである GP14 と対応しており、led.on()を実行すると、GP14 が+3.3 V がかかるため、3.3V→LED→GND という電気の流れ道ができることで LED が光ります. 逆に led.off()を実行すると GND→LED→GND となり、LED は光りません. 実際にテスターなどを用いて、LED にかかっている電圧を図ってみるといいでしょう.

【課題 1-1】 LED の点滅の間隔を，現在の半分にしてみよう.

【課題 1-2】 ブレッドボード上の3個の LED を，1 秒ごとに順番に点灯させるプログラムを作成し，動作を確認してみよう.

1.4.デジタル入出力

外部プッシュスイッチの状態を読み取り、プッシュスイッチが押されている間のみ LED を点灯させよう.

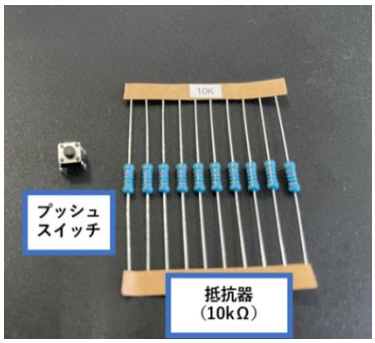
1）例題 2-1 のプログラムを入力する

この時 switch は Pin.IN でデジタル入力であることを示しています. Switch は GP17 に電圧がかかった時は“1”、電圧がかかっていない時は“0”になります. すなわち、GP17 に電圧がかかると led.on()が実行されることで GP14 につながれた LED が光ります.

例題 2-1 のプログラム

```
from machine import Pin
import time

led = Pin(14, Pin.OUT)
switch = Pin(17, Pin.IN)

while True:
    if switch.value() != 1:
        led.on()
    else:
        led.off()
```

2) 部品を準備する(プッシュスイッチ、抵抗器(10kΩ))



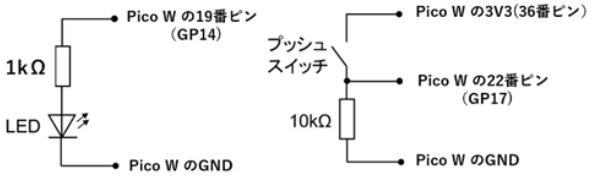
プッシュスイッチはスイッチを押すと対角のピンが通電し、それ以外の時は対角のピンが断線しているという構造になっています.

3) 回路を作成する

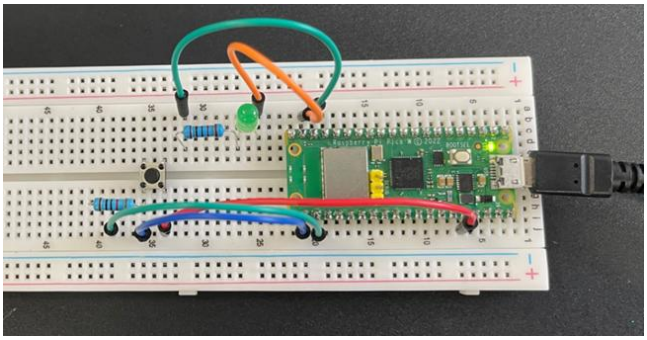
下図の回路をブレッドボード上で作成する

マイコンの入力ピン(GP17)は、何にもつながっていないとき電圧がフラフラして 0 にも 1 にもなります(＝フローティング). そこで 10 kΩ を GND に接続して常に 0 側に引っ張ります(プルダウン). これによって、スイッチを押したときだけ GP17 が 3.3 V と直結されて 1 になります.

ここで、なぜ 10 kΩ につなぐかというと、もし 10 kΩ につないでいない場合、3.3V から直接マイコンに電流が流れます. この時、回路内の抵抗は 0 なので、過電流が流れてマイコンに負荷がかかり、最悪破損してしまいます.



4) プログラムを実行し、ボタンを押しているときに LED が光ることを確認しよう



【課題 1-5】プッシュスイッチを押している間だけ LED が消灯し、離すと点灯するプログラムを作成してみよう.

【課題 1-6】プッシュスイッチを押すごとに、LED の点灯状態が切り替わる（トグル動作する）プログラムを作成してみよう.

＜ヒント＞

- 変数を 1 つ用意して、スイッチが押されるたびに +1 していこう.
- time.sleep(0.1) を入れることで、スイッチを 1 回押したときにカウントが何回も増えるのを防げる.
- “%2”を使えば、押された回数が偶数か奇数かを判定できる. 例：if count % 2 == 0: なら「偶数回目のとき」だけ LED を ON にできる.

【課題 1-7】プッシュスイッチを押すたびに, 3 個の LED が順番に点灯する.

2.アナログ I/O

これまでのデジタル I/O では、ON(1)か OFF(0)かの二つの状態しか扱えませんでした. しかし、私たちの身の回りには、明るさ、色、温度、音量など、滑らかに変化する「アナログ」な情報がたくさんあります. この章では、マイコンでアナログ情報を扱う方法を学びます. LED の明るさをじんわりと変えたり、センサで温度を読み取ったり、モータの回転速度を自由にコントロールしたりすることを目指します. デジタルとアナログの違いを理解し、表現の幅を広げていきましょう.

2.1.PWM で LED の明るさ調整（アナログ出力）

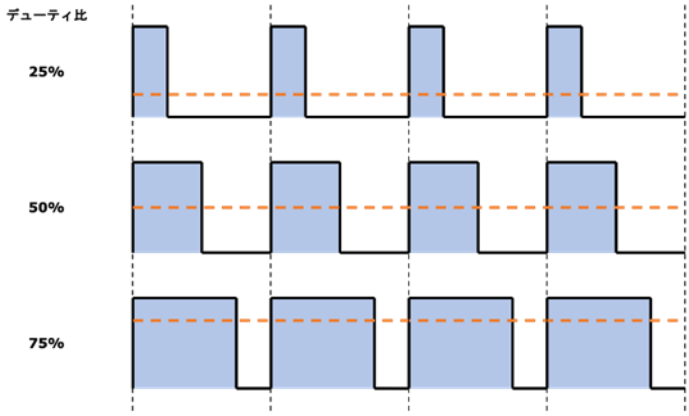
LED をただ点けたり消したりするだけでなく、ホタルのようにふわっと光らせたり、スマートフォンの通知のようにゆっくり点滅させたりする方法を学びます. ここでは PWM（Pulse Width Modulation）という技術を使います.

1) PWM とは? (https://ensatellite.com/ja/pwm/)

本当にアナログの電圧（例えば 1.7V とか 2.3V とか）を作ろうと思うと、可変抵抗や DAC（Digital to Analog Converter）を用いる必要があります. しかしながらこれはとても手間のかかることです. そこで、デジタルな例えば 3.3V の信号を出せる装置のみでアナログ信号を再現する手法が PWM です.

PWM（パルス幅変調）は、電気の ON/OFF を人間には知覚できないほど高速で繰り返し、その ON の時間と OFF の時間の「割合」を変化させることで、出力される電力（電圧）を擬似的にコントロールする技術です. この ON の時間の割合を「デューティ比（Duty Ratio）」と呼びます.

これによって、3.3V の信号を使って、1.7V や 2.3V などの信号を再現することができます. たとえば、デューティ比が 0%なら常に OFF（消灯）、100%なら常に ON（最大輝度）、50%なら ON と OFF の時間が半分ずつになり、半分の明るさに見えます. この仕組みを使うことで、LED の明るさを滑らかに調整できます.



2) 呼吸ランプを作ろう

それでは、実際に PWM 信号を使って、LED の明るさを段階的に変えてみましょう.

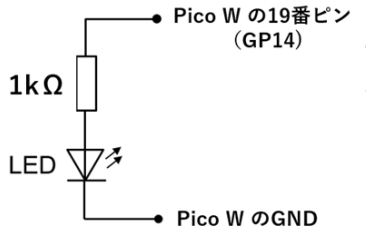
① 回路を組む

前章で作成した外部 LED の回路をそのまま使用します. ただし、Pico W では、PWM を扱えるピンが決まっ

ています.

・Pico W の GP14 (19 番ピン) → 抵抗(1kΩ) → LED のアノード(+)

・LED のカソード(-) → Pico W の GND (18 番ピンなど)



② コードを入力して実行する

Thonny に以下のコードを入力し、実行してみましょう. Pico W では、Duty 比を 16 bit(0～65535)の幅で指定することができます. led.duty_u16(32768)は半分の大きさで光ります. この値を変えることで、段階的な光を表現することができます.

```
from machine import Pin, PWM
import time

# GPIO14 を PWM 出力に設定
pwm  pin = Pin(14)
led = PWM(pwm_pin)

# 周波数を設定（1kHz くらいが LED にはちょうどいい）
led.freq(1000)

# デューティ比を 50% に設定 (0～65535 の範囲)
led.duty_u16(32768)

# そのまま点灯を続ける
while True:
      time.sleep(1)
```

③ 段階的に光る強さを変えることで、呼吸ランプを作ろう

Python では for 文を使って段階的な数字を作って実行することができます. for duty in range(0, 65535, 1024): という書き方は、range(0, 65535, 1024) という部分が「0 から始る→1024 ずつ増やす→65535 に到達するまで繰り返す」ことを表します. 実際には duty という変数「0, 1024, 2048, 3072, …」というように続いていきます.

これによって、段階的に duty 比が変わります. Time.sleep を入れているのはこれを入れないとプログラムの実行が速すぎて一瞬で 65535 に達し、0 に戻るを繰り返すためです. それでは、time.sleep の値を変えて、動作を見てみましょう.

```
from machine import Pin, PWM
import time

# GPIO14 を PWM 出力に設定
```

```
pwm_pin = Pin(14)
led = PWM(pwm_pin)

#  周波数を設定
led.freq(1000)

while True:
    #  明るくする(0 →  最大)
    for duty in range(0, 65535, 1024):    # 1024 ずつ増やす
        led.duty_u16(duty)
        time.sleep(0.01)

    #  暗くする(最大  → 0)
    for duty in range(65535, -1, -1024):    # 1024 ずつ減らす
        led.duty_u16(duty)
        time.sleep(0.01)
```

【課題 2-1】明るさの増減を SIN カーブにする

＜ヒント＞ Python で正弦波を再現するには math.sin(math.radians(rad))を使います．Rad には 0~2π の値を入れてください．

【課題 2-2】LED を二つに増やし、交互に明るさを増減させる．

2.2.アナログ入力をしてみよう(温度センサ、照度センサ)

これまで、スイッチといったある電圧を超えたら 1、それ以下なら 0 というデジタル入力については学びました．しかし、世の中のセンサは 0 か 1 ではなくアナログな値をとるものがたくさんあります(温度、照度、加速度など)．この節では、マイコンに備わっているアナログ-デジタル変換器(ADC)を使って、センサのアナログ信号を数値として読み取り、その変化をプログラムで扱う方法を学びます．

1) 内部温度センサの値を出力する

Pico W には温度センサが搭載されています．温度はアナログな値であるので ADC を使って読み取ります．ADC(4)というのは 4 チャンネルにかかった電圧を読み込むということです．実際に sensor_temp.read_u16() で温度センサからの生の値(0～65535→0~3.3V)を読み取っています．

conversion_factor = 3.3 / 65535 は、ADC が読み取る値(0～65535 の整数)を実際の電圧(0～3.3V)に変換するための係数です．Pico の ADC は 16 ビットで、最大で 65535 までの値を返すので、この係数を掛けると「何ボルトか」に直せます．

temp = 27 - (ADC_voltage - 0.706) / 0.001721

という式で、電圧から温度に変換しています．ここで使われている「0.706」や「0.001721」という数字は、Pico W に搭載されたセンサの特性を表す定数です．基準温度 27℃のときの電圧が 0.706V で、1℃変化すると電圧が約 0.001721V 変わる、という意味です．その結果を round(temp, 1) で小数点以下 1 桁に丸めて、見やすくしています．

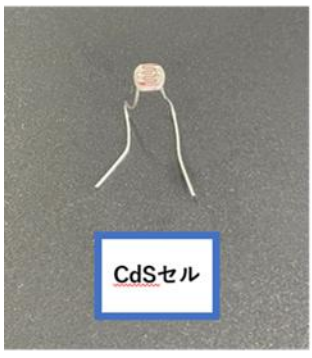
```
from machine import ADC
import time

sensor_temp = ADC(4)
conversion_factor = 3.3 / 65535

while True:
    ADC  voltage = sensor  temp.read  u16() * conversion  factor
    temp = 27 – (ADC_voltage – 0.706) / 0.001721
    temp = round(temp, 1)
    print(temp)

    time.sleep(1)
```

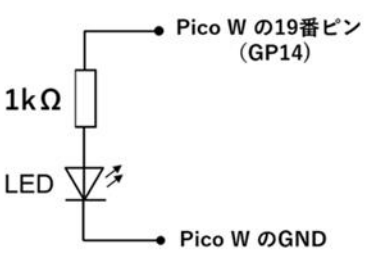
Thonny で「▶（実行）」ボタンを押すと、シェル画面に 1 秒ごとに現在の気温が表示されます．写真の例だと「27.5℃」や「28.0℃」といった値が出ています．

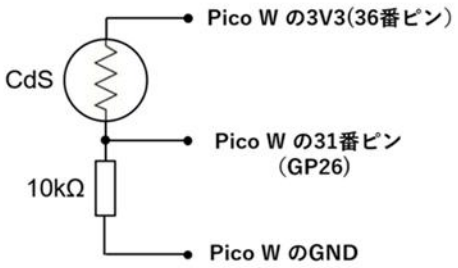
2）照度センサの値を出力する

CdS とは，明るさに応じて抵抗値が変化する素子です．CdS の出力をアナログ電圧として Pico W のアナログ入力から読み込み，CdS のまわりが明るくなると LED も明るくなるようにしよう．

まず、CdS セルから照度を読み取るための回路を作成します．回路図を参考にして LED と CdS セルの回路を作成してください．CdS セルは「硫化カドミウム（CdS）」という半導体を使った光センサの一種で、受ける光の強さによって抵抗値が変わる素子です．暗いときは光が当たらなければ電子があまり動かないため、抵抗値は大きくなり電気が流れにくくなります．光が当たったときは光のエネルギーが CdS の原子に吸収されて、電子が励起されて「キャリア（電子と正孔）」が増えます．これによって電気を運ぶ粒子が増えるので、抵抗値が小さくなります．








```
from machine import Pin, PWM, ADC
import time
```

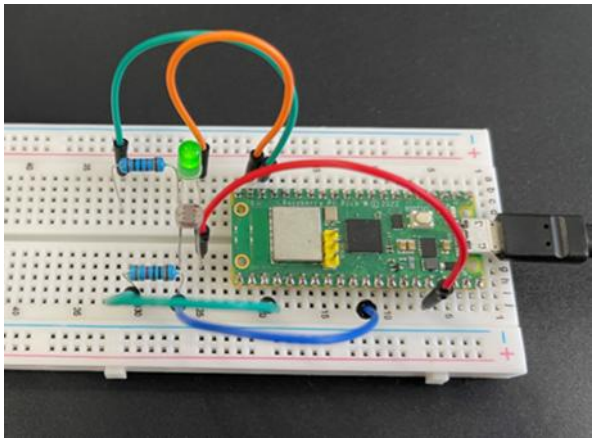
```
led = PWM(Pin(14), freq = 100)
cds = ADC(Pin(26))

while True:
    duty = cds.read_u16()
    led.duty_u16(duty)
    print(duty)
    time.sleep(0.5)
```

プログラムを上記に示します．このプログラムを読んだら、実行する前に下記の問いに答えてください．

＜事前の予想（ここは記入させる）＞

1. 暗くすると CdS の抵抗は大きくなる／小さくなる（どちら？）
2. そのとき分圧点（GP26）の電圧は高くなる／低くなる（どちら？）
3. それによって PWM の duty は増える／減る（どちら？）
4. LED は明るくなる／暗くなる（どちら？）

理由も含めて考察して下さい．



予想を記入し終えたら、上記のコードを入力して、CdS センサを手で覆ったりして変化を確かめてみましょう．

【課題 2-3】手で覆った時の変化からどのような流れで LED の光る強さが変わっているのか、考察しよう．

＜ヒント＞ 抵抗が直列につながっており、電圧は分圧されます．CdS の抵抗が大きいほど、GP26 の電圧はどうなるでしょうか？

【課題 2-4】回路の上下を入れ替えてみましょう．どんな変化をするかそれはなぜか考察してください．

2.3.目標の値に出力を操作しよう(LED、照度センサ)

前節では、照度センサ（CdS）の値を読み取り、そのまま LED の明るさに反映させるプログラムを作りました．これだと、照度が暗くなると、LED も暗くなってしまいます．本来の LED は部屋の明るさを一定に保ってほしいですよね？

そこで、この節では、照度センサで明るさを計測し、それに応じて LED の強さを制御することによって明るさを一定に保つ方法を学びます．

まずは、2.2 節の回路を組んでください. この際、LED と CdS は近くに設置してください. それでは実際に、CdS センサに対する明かりを一定にするにはどうしたらいいでしょうか.

1) バンバン制御とは?

例えば、部屋の明かりを「明るさ 100」にしたいとします.

・暗ければ LED を最大点灯

・明るければ LED を消灯

このように「基準を超えたらオン、下回ったらオフ」という単純な制御を「バンバン制御」と呼びます. この方法でも、明るさは 100 になるかもしれません.

しかしこの方法だと、目標値の周りで点いたり消えたりを繰り返し、安定しません.

2) P 制御(比例制御)の考え方

そこで、目標値との差(偏差)に応じて出力を変える「P 制御(比例制御)」を導入します. 出力=Kp×(目標値−現在の値)

ここで、目標値は狙いたい明るさです. 例えば目標値に現在の値の値が近ければ出力は小さくなり、差が大きければ出力は大きくなります.

偏差が大きければ LED を強く光らせ、小さければ弱くすることで、目標値に近づけることができます. Kp の値が大きいほど、目標値へ近づくための出力は大きくなります. Kp の値を小さくすると滑らかに変化します.

3) 実際に、LED を制御してみよう

まずは、CdS の目標明るさを 32000 にしてみます. これは、CdS が 32000 の明るさと感じるように LED を制御するという意味です. error = TARGET – y によって、現在の明るさが 32000 に対して明るいかどうかを数値化します. 明るければ LED は暗くなり、暗ければ LED は明るくなります.

```
from machine import Pin, PWM, ADC
import time

led = PWM(Pin(14), freq=1000)       # 1 kHz
cds = ADC(Pin(26))                  # 16bit (0..65535)

# --- P 制御パラメータ ---
TARGET = 32000        # 目標"明るさ"
KP = 1.0              # 比例ゲイン(まず 0.5〜2.0 で調整)

# 出力の安全範囲
MIN_DUTY = 0
MAX_DUTY = 65535

while True:
    y = cds.read_u16()                  # 現在値(センサ)
    error = TARGET - y
    u = int(KP * error)                 # 比例出力
    # duty を 0..65535 にクリップ
    u =
(MIN_DUTY, min(MAX_DUTY, u))
```

```
    led.duty_u16(u)

    print("cds:", y, "err:", error, "duty:", u)
    time.sleep(0.05)
```

【課題 2-5】実際にセンサに手をかざしたり外したりして、LED の光り方がどう変わるかを観察しよう. スマホのライトを CdS に向けてみよう.

【課題 2-6】比例ゲイン KpK_pKp を変えてみよう.

・小さすぎるとどうなる？

・大きすぎるとどうなる？

Error の値が 0 に近づいているかから考察しよう

【課題 2-7】目標値をいくつか変えてみよう.

・低めに設定したとき、高めに設定したときの違いを確認しよう.

今回は LED でしたが、例えばエアコンは温度センサの値に応じて熱い空気を送るか冷たい空気を送るか制御しています. 自動運転車も車道に対してまっすぐになるようにセンサの値からタイヤの向きを堰御しています. P 制御は制御の基本なので、ぜひ覚えて、家の家電でセンサは何で出力は何でどうやって制御しているか考えてみてください.

2.4.加速度センサを使おう(I²C 通信)

2.2 節のように「電圧を出力する」ものとは違い、センサの中にはアナログ値を電圧値ではなく数値データとして出力するものがあります. 代表的なのが、MPU6050（加速度・ジャイロセンサ）です.

このようなセンサは I²C 通信 という仕組みを使い、デジタルデータを Pico W に送ってきます. この節では I²C 通信を使って加速度を読み取る方法を学びます.

1) I²C 通信とは

I²C 通信は、電子部品どうしがやり取りするための共通言語のようなものです. 2 本の信号線（SCL＝クロック、SDA＝データ）で通信します. 各センサは「アドレス」を持っており、同じ線に複数のセンサをつなぐことができます. Pico W が「アドレス 0x68 のセンサに、レジスタ 0x3B を読ませて」と命令すると、センサがそこに入っている数値を返してくれる仕組みです.

2) 回路を組もう

下記図を参考に回路を組んでください.

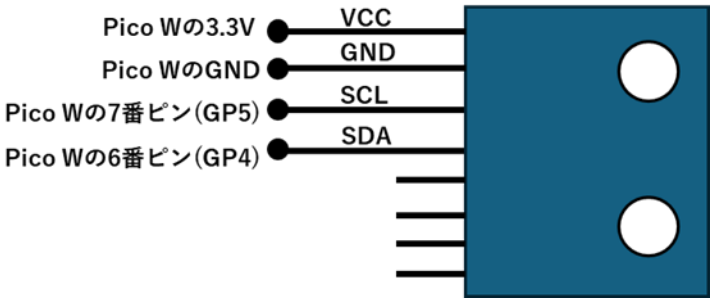
・Pico W GP4 (6 番ピン) → SDA

・Pico W GP5 (7 番ピン) → SCL

・Pico W 3.3V → VCC

・Pico W GND → GND

MPU6050



3) プログラムを入力する

Thonny に以下のコードを入力し、実行します.

```
from machine import Pin, I2C
i2c = I2C(0, scl=Pin(5), sda=Pin(4))
print("I2C devices:", i2c.scan())
```

→ [104](= 0x68)が表示されれば成功です. 表示を確認したら加速度データを取得してみましょう.

このプログラムは、最初に時間を扱うための time と I²C 通信用の machine モジュールを読み込み、I²C の配線(SCL が 5 番ピン、SDA が 4 番ピン)と加速度センサのアドレス 0x68 を設定し、電源管理レジスタに 0 を書き込むことでセンサのスリープ状態を解除します.

to_int16 という関数で「2 バイトのデータを符号付き 16 ビット整数に読み替える」仕組みを作ります(2 バイトを上位・下位でつなげ、先頭のビットが 1 ならマイナス扱いに直します). バイトを整数に変える操作は複雑なのでここでは、そんなものなんだなあと思ってください.

その後は、センサの加速度データが並んでいる場所(0x3B から 6 バイト)を一気に読み取り、最初の 2 バイトを X、次の 2 バイトを Y、最後の 2 バイトを Z として取り出し、それぞれ to_int16 で「マイナスも表現できる数」に直します. 直した生の値を SCALE で割ることで X・Y・Z の加速度を g の単位に変換し、小数点以下 3 桁で見やすく整えて 1 行に表示します. 最後に 0.2 秒だけ待機して、同じ読み取りと表示を繰り返し、リアルタイムに加速度がどのくらいかを連続で確認できるようにしています.

```
import time
from machine import Pin, I2C

i2c = I2C(0, scl=Pin(5), sda=Pin(4))
addr = 0x68

# スリープ解除
i2c.writeto_mem(addr, 0x6B, b'¥x00')

SCALE = 16384.0

def to_int16(b2):
    v = (b2[0] << 8) | b2[1]
    return v - 65536 if v & 0x8000 else v

while True:
    # X,Y,Z(各 2 バイト、合計 6 バイト)
    data = i2c.readfrom_mem(addr, 0x3B, 6)
```

```
    x_raw = to_int16(data[0:2])
    y_raw = to_int16(data[2:4])
    z  raw = to  int16(data[4:6])
    x = x_raw / SCALE
    y = y_raw / SCALE
    z = z_raw / SCALE
    print(f"X:{x:7.3f} g    Y:{y:7.3f} g    Z:{z:7.3f} g")
    time.sleep(0.2)
```

Tips:print を使う際に「””」のなかに「{変数}」を入れると変数の値をそのまま出力することができます．これは文字列でもできます．

【課題 2-8】加速度の大きさを計算してみよう

- X, Y, Z の値を使って「合成加速度（$\sqrt{(x^2+y^2+z^2)}$）」を求めよう．
- 静止状態ではおおよそ 1 g になるはずです．机に置いたり傾けたりして、値がどう変化するか観察してください．

【課題 2-9】傾きに応じて LED を光らせよう

- Pico W のオンボード LED を使って、センサの傾きを視覚化してみましょう．
- 例：
  - X 軸が +0.5g 以上なら LED 点灯
  - それ以外のときは LED 消灯

2.5.DC モータ制御（PWM 応用）

モータは、ロボットやドローンなど、モノを動かすために不可欠な部品です．LED の明るさ制御で使った PWM は、モータの回転速度をコントロールするためにも使えます．デューティ比を上げるほどモータに流れる平均電力が大きくなり、回転は速くなります．

ただし、モータは LED よりも大きな電流を必要とするため、マイコンのピンに直接接続するとマイコンが壊れてしまう危険があります．そのため、トランジスタなどを使って、外部の電源からモータに電力を供給する「ドライバ回路」を組むのが一般的です．つまり、マイコンからの出力に応じた電流を電池から引っ張ってくる回路なわけです．

トランジスタでの基本イメージとしては、NPN トランジスタを考えるとわかりやすいです．マイコンからベースを ON すると、コレクタ（電池＋モータ側）からエミッタ（GND）へ電流が流れます．この時流れる電流は電池からの電流なのでマイコンはほんの少しの電流で大きな電流を制御できます．これに PWM を与えれば、モータの平均電圧が変わり、回転数も変わります．

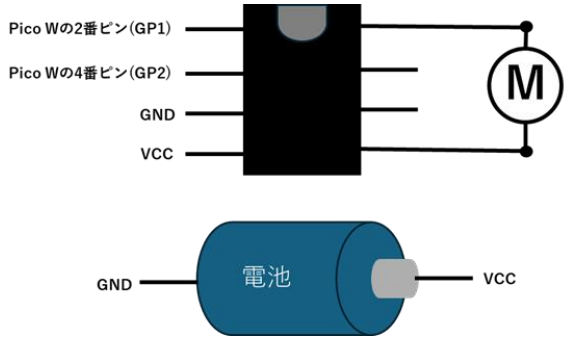
1）回路を組む

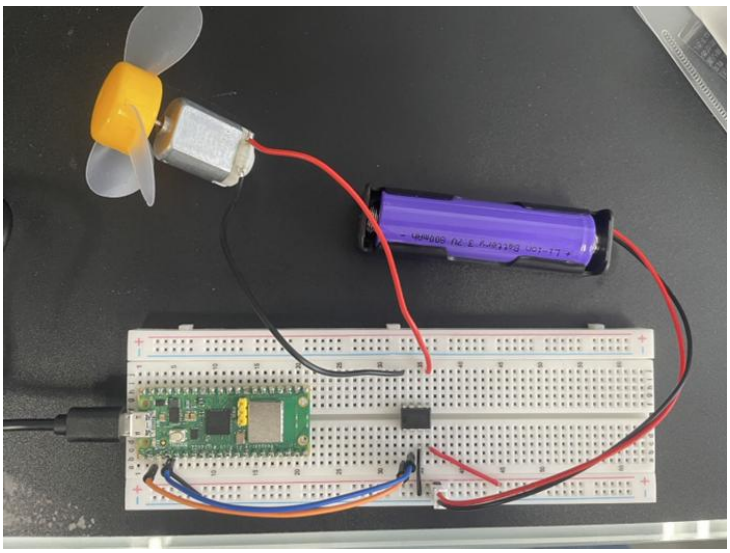
それでは、実際に回路を組んでいきましょう．今回使う素子は TA6586 です．一般的にモータドライバーと呼びます．これは、トランジスタ 1 個では正転しか制御できないため、トランジスタが H 型にブリッジして組まれた回路が入っています．

ここで、TN6586 には信号線が 2 本あります．この 2 本に下記のように電圧を加えることによって、モータを制御します．

IN1	IN2	出力（モータ両端の状態）	動作
0	0	両端ともGND	ブレーキ停止
1	0	正転方向に電圧	正回転
0	1	逆転方向に電圧	逆回転
1	1	両端ともVcc	ブレーキ停止

TA6586





2) モータを制御する

それでは実際にモータを制御するプログラムを書いていきましょう．下記のコードは、5 秒おきに正転と逆転が入れ替わるコードです．Duty 比を変えることにより回転の強さが変わっていることもわかります．

```
from machine import Pin, PWM
import time

duty = 10000

#  物理ピン: IN1=GP1, IN2=GP2  とする
IN1 = PWM(Pin(1))
IN2 = PWM(Pin(2))
IN1.freq(100)
IN2.freq(100)
```

```
while True:
        duty = 10000
        IN1.duty  u16(duty)
        IN2.duty_u16(0) #   IN2 を 0 にすることで正転

        time.sleep(5)

        duty = 30000
        IN1.duty_u16(0)
        IN2.duty_u16(duty) #   IN2 を 0 にすることで正転

        time.sleep(5)
```

【課題 2-10】duty 比がマイナスだったら|duty|の強さで逆転するように if 文で分岐を作ろう. また、duty の大きさに下限と上限を設けよう.

【課題 2-11】IN1 と IN2 をどちらも 0 にして、モータに触ってみよう

IN1=65536, IN2=65536 と比べてみると？

【課題 2-12】加速度センサの値に応じてモータを動かしてみよう(疑似コントローラ)

3.ネットワークと Web 連動

本章では、web サイトの作成方法と、作成した web サイトから遠隔で Pico W を操作する方法を学びます．外部監視システムや、外から操作することのできる便利な IoT 作品を作成するために必要な知識を得られます．

3.1.Wi-Fi 接続 & IP アドレスとは？

本項では WiFi に Pico W を接続する方法と、IP アドレスを指定することによってメッセージを送る簡単なチャットアプリを作成する方法を学びます．

1) Pico W をインターネットに接続する

① 自宅の WiFi の ssid と password を確認する

SSID は WiFi の電波の「名前」であり、スマホやパソコンで wifi に接続する際に選択する WiFi 固有の名前です．5GHz は対応していないので注意すること．

② Thonny のエディタに、以下のコードを入力

赤枠の部分を自宅の WiFi の ssid と password に書き換えてください．接続に成功すると「接続完了」と表示され、その WiFi の IP アドレスが表示されます．IP アドレスはすべてのネットワークにつながる機器が持っているインターネット世界の住所である．

例）192.168.1.23 IP アドレス＝ネットワーク上の“住所”．同じ家の中（同じ Wi-Fi）にあるスマホや PC は、この住所を宛先にして Pico W へアクセスできます．

```
import time
import network

# Wi-Fi の SSID とパスワードを入力
ssid = “ここに SSID を入力”　#5GHz は対応していません
password = “ここにパスワードを入力”

# Wi-Fi 設定
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid. password)

max  wait = 10

while max_wait > 0:
      if wlan.status() < 0 or wlan.status() >= 3:
            break:
      max_wait -= 1
      print(“接続待ち．．．”
      time.sleep(1)
if wlan.status() != 3:
      raise RuntimeError(“ネットワーク接続失敗”)
else:
      print(“接続完了”)
```

```
    status = wlan.ifconfig()
    print("IP アドレス＝" + status[0])
```

```
シェル
接続待ち...
接続待ち...
接続待ち...
接続待ち...
接続待ち...
接続完了
IPアドレス = 192.168.11.20

>>>
```

友達の Pico W にメッセージを送ってみよう（同一 wifi 上でなければいけません）

2) ソケット通信って何?

ソケットは通信するための窓口のことです．IP アドレスのみでは同一 IP アドレスの住所に対して、誰に渡すかを指定できません．これは同じパソコン上では同じ IP アドレスですが、同じパソコンでも、複数にアプリを使用しているときにどのアプリと通信するか決められないということです．そこでポート番号を合わせて使用します．IP アドレスとポート番号を合わせることで、きちんと相手に情報を届けることができます．IP アドレスがマンションの住所、ポート番号が部屋番号と思うとわかりやすいかもしれません．

ここで、最も簡単なソケット通信の方法の一つが UDP です．UDP はハガキをポストに投函するイメージを持ってください．UDP は一方的にハガキをその住所の部屋番号に届けるので、相手がその手紙を受け取ったか、また順番通りに受け取ったかはわかりません．しかしながら、その分、高速でデータを送信することができます．イメージとしてはオンラインゲームや動画配信のように速さ重視で情報を届ける際に使えます．

対して、相手からの応答を確認して通信する方式が TCP です．こちらは、ハガキに対して電話のイメージです．今回は扱わないので必要に応じて調べて実装してみてください．

3) プログラムを書き込む

```
# ===== ここから送信 =====
import socket

TARGET_IP = "192.168.1.50"     # ← 受信側 Pico W の固定/取得した IP に置き換え
TARGET_PORT = 5005                  #  任意のポート(受信側と合わせる)
MESSAGE = "Hello from Pico W!"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)    # UDP
sock.settimeout(2)

try:
    sock.sendto(MESSAGE.encode("utf-8"), (TARGET_IP, TARGET_PORT))
    print("送信完了 →", TARGET_IP, TARGET_PORT, MESSAGE)
finally:
    sock.close()

# ===== ここから受信 =====
import socket
PORT = 5005    #  送信側と合わせる

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)    # UDP
```

```
sock.bind(("0.0.0.0", PORT))
print("待受開始  (UDP port", PORT, ")")

while True:
    data, addr = sock.recvfrom(1024)
    try:
        text = data.decode("utf-8")
    except UnicodeError:
        text = repr(data)
    print("受信:", addr, "→", text)
```

上記のプログラムのうち、送信したい方は送信側、受信したい側は受信側のプログラムを Pico W をインターネットに接続するプログラムの下に挿入してください.

この際、相手の受信側の IP アドレスは、相手側の IP アドレスに合わせてください. MESSAGE の中身を編集して実行することで、任意のメッセージを相手に送ることができます.

成功すると、受信側の Thonny シェルに送信側の IP アドレスとメッセージが表示されます.

【課題 3-1】クラスメイトにメッセージを送ってみよう.

【課題 3-2】メッセージを送る際に入力を受け付けてみよう.

＜ヒント＞

1) Python では下記のコードを使うことで入力を受け付けることができます. 実行してみましょう.

```
MESSAGE = input("メッセージを入力してください: ")
```

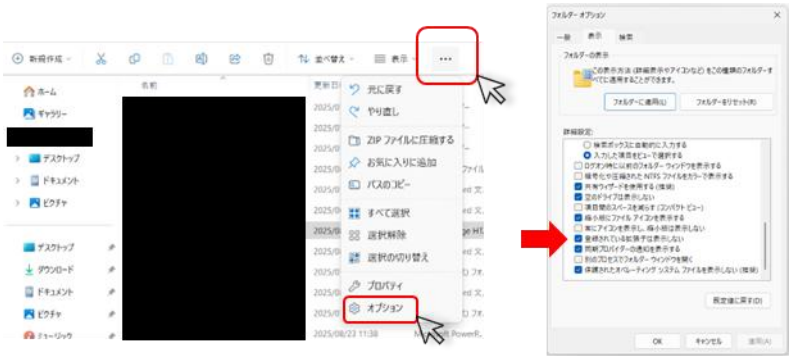
2) Input 関数は日本語入力を受け付けないので注意

3.2.ウェブサイトの基礎を学ぼう

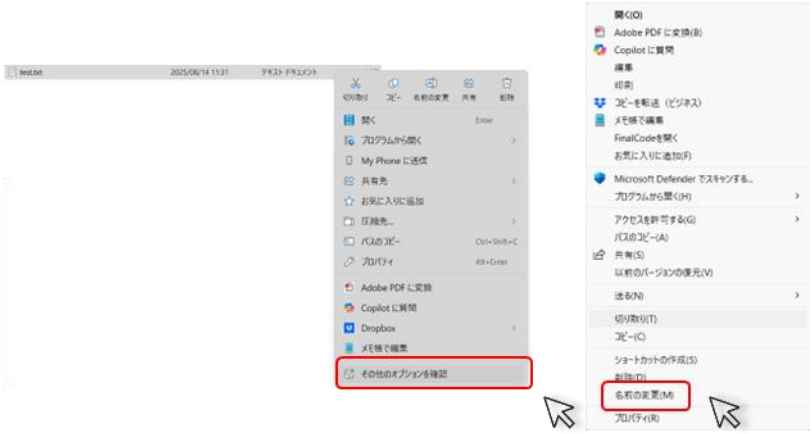
1) HTML って何?

HTML とは Web ページの骨組みを作るためのプログラミング言語であり、<html>, <head>, <body> で構造を作ります.

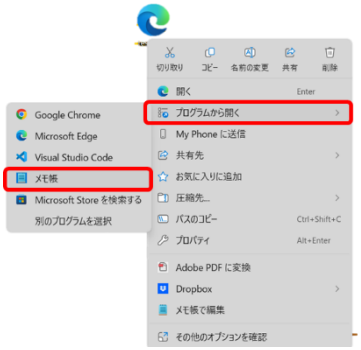
実際に web ページを製作してみよう. 特定のフォルダに test.txt というテキストファイルを作成し、.txt という拡張子を.html に変えてください. ここで、PC 上の既定の設定では、拡張子は表示されておらず test とのみ表示されているかもしれません. 理系の方は拡張子を変える作業をすることがあるかもしれないので、この拡張子を変ることができるように変更を加えます.



まず、フォルダアプリの「…ボタン」にカーソルを合わせます．するとオプションが表示されるので、オプションをクリックしてください．するとフォルダーオプションが表示されるので、「登録されている拡張子は表示しない」のチェックを外します．その後「OK ボタン」をクリックしてください．これで拡張子を変えられるようになりました．



名前の変更機能で text.txt の名前を test.html に変えてください．名前の変更は test.txt を右クリックし、「その他のオプションを確認」をクリックし、名前の変更を選ぶことで変えることができます．



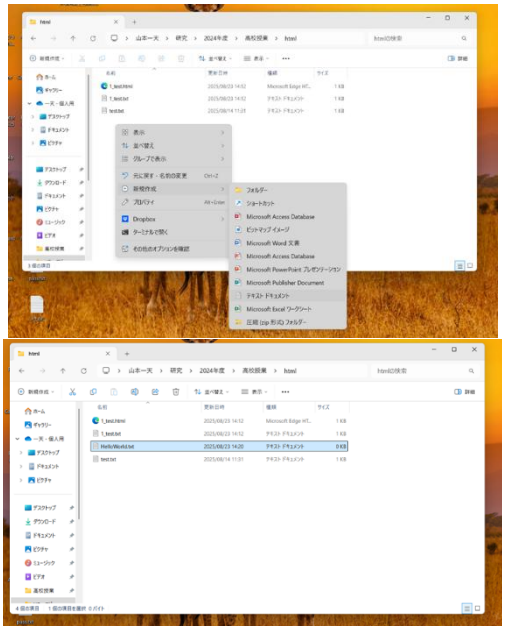
メモ帳で開いて中身を書き換えます．メモ帳で開くためには test.html を右クリックし、プログラムから開くから、メモ帳をクリックしてください．

その後、test.html をダブルクリックすることで、ローカル上で web サイトを開くことができます．実際の web サイトではドメインと呼ばれるものを取得し、サーバにプログラムをアップすることで世界中の人がアクセスできるようにしています．Pico W では、サーバにアップしなくても、同一 wifi 上であればパソコンから IP アドレスを利用することでこの web サイトにアクセスすることができます．

2) Hello World を表示してみよう

実際に「Hello World！」という表示がされるページを作ってみましょう.

新しく「HelloWorld.txt」というファイルを作って、ファイルを開いてください.

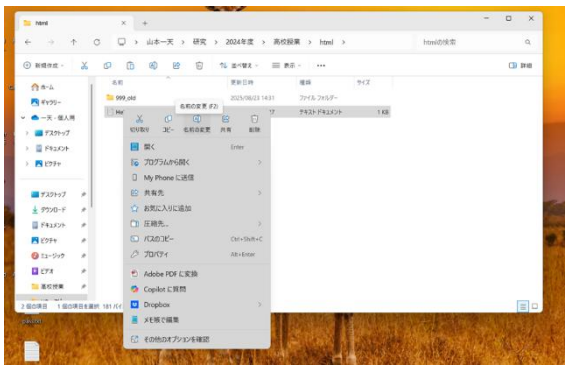


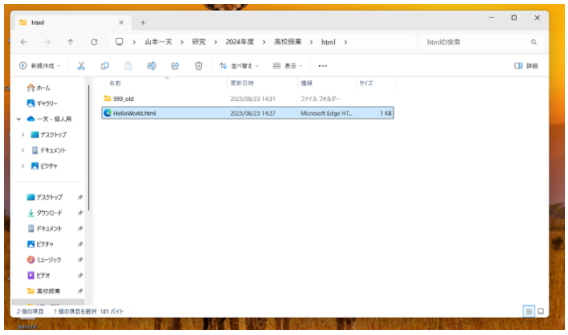
こうして作ったテキストファイルに次のコードを書いてください.

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>ページのタイトル</title>
    </head>

    <body>
        <h1>Hello World!!!</h1>
        <p></p>
    </body>
</html>
```

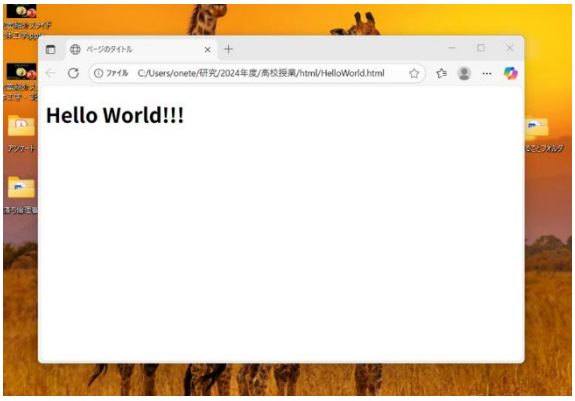
これが html の基本的な形で、普段から使っているページにも使われています. 実際に今回書いたコードのページを見てみましょう.

「HelloWorld.txt」の拡張子「.txt」を html の拡張子「.html」に変更して、「HelloWorld.html」をダブルクリックしてください.





すると、このようなページが表示されます.



3) ボタンを配置

次に HelloWorld の下にボタンを配置します.

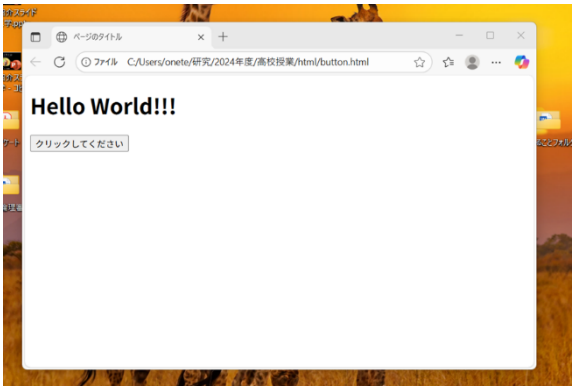
新しく「button.txt」というファイルを作って、次のコードを書いてください.

先ほど書いたコードから変わっているのは 1 行だけです.

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="UTF-8">
        <title>ページのタイトル</title>
    </head>

    <body>
        <h1>Hello World!!!</h1>
        <p></p>
```

```
    <button>クリックしてください</button>
  </body>
</html>
```

このコードを書くことができたら、先ほどと同様に拡張子を「.txt」から「.html」に変えて開いてください.



「クリックしてください」というボタンが表示されていることを確認してください.

実際にボタンを押しても何も起こらないはずです. ボタンを押したときに何かするためには、書かないといけないことが足りていません.

今は何も起こらないでいいので、ボタンについては以上です.

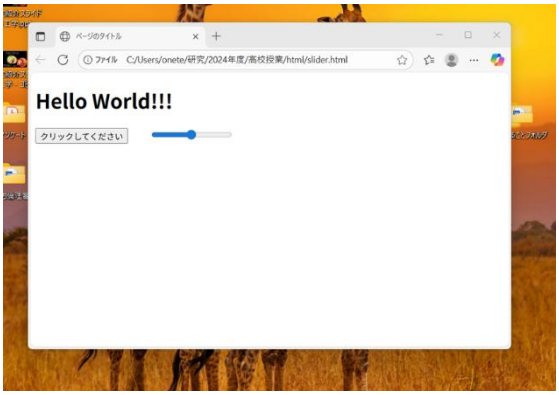
4) スライダを配置してみよう

次にスライダを配置してみましょう.

同じように、「slider.txt」というファイルを作って、次のコードを書いてください. もしコードが書けたら拡張子を変えてページを見てみましょう.

スライダが表示されたら、ボタンをクリックしながら動かしてみましょう.

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8">
    <title>ページのタイトル</title>
  </head>

  <body>
    <h1>Hello World!!!</h1>

    <button>クリックしてください</button>
  <input type="range" min="0" max="10">
  </body>
</html>
```



次の章では、今回作ったページからボタンを押すとマイコンの LED が光るようにつなげてみましょう.

【課題 3-1】文字の大きさや色を変えてみよう

＜ヒント＞

<h1 style="color: red; font-size: 50px;">Hello World!!!</h1>

フォントサイズやカラーを設定することで文字の大きさや色が変わるよ.

【課題 3-2】リンクを貼ってみよう

＜ヒント＞

Google へ移動

a タグを使うことで、リンクを貼ることができるよ

【課題 3-3】自分の好きなサイトを紹介するホームページを作ってみよう. 色や大きさ配置を工夫してね.

3.3.Pico W で Web サーバ経由の LED を制御をしてみよう

1) Pico W で Web サーバを立てよう

まずは、3.1 節を参考にして、Pico W を wifi に接続してください. また、この時 IP アドレスを記録することを忘れないでください.

それでは、Pico W で配信している web サイトにアクセスする方法を具体的に学びましょう. Web サーバはふつう ポート 80（HTTP）で待ち受けます.

Pico W をつないだ際にわかる IP アドレスがあれば、同じ Wi-Fi 上の PC やスマホからその IP アドレスにアクセスできます. Web サーバは通常ポート 80（HTTP）で待ち受けるので、特別な指定がなければ URL は http://192.168.1.23 という風になります.

2) Pico W をサーバとして、web ページを配信し、アクセスする

それでは、Pico W 上に web ページを作成してみましょう. 3.3 節で書いたプログラムの下に html という変数としてページを作成することにより、その web ページにアクセスすることができるようになります. 今回は下記を参考に、LED のオン、オフボタンを作成しましょう.

```
# HTML ページ(LED の ON/OFF ボタン)
Html = “””<!DOCTYPE html>
<html>
    <head><title>Pico W LED</title></head>
    <body>
        <h1>LED Control</h1>
        <form action”/led/on” method=”get”>
            <button type=”submit”>LED ON</button>
        </form>
        <form action=”/led/off” method=”get”>
            <button type=”submit”>LED OFF</button>
        </form>
    </body>
</html>
“””
```

ここで、3.2 節で説明を省いたボタンに役割を持たせる方法を説明します．3.2 節まではボタンは何の役割もないただの飾りでした．ここで、form タグはユーザーの操作をサーバに伝える仕組みであり、ボタンを押したときにどの処理を呼び出すかを指定することができます．

action 属性には送信先の URL を指定し、例えば action="/led/on"と書くとボタンを押したときにブラウザがサーバへ「/led/on」へのアクセスを行います．同様に action="/led/off"と書けば「/led/off」にアクセスが送られます．

method 属性はリクエストの送信方法を表し、ここでは get が使われています．GET メソッドは URL の形でリクエストを送る方式であり、今回のように特別な入力データがなく単にページ遷移や命令を伝えるときに簡単に利用できます．つまり、LED ON ボタンを押すとブラウザから Pico W に対して「http://PicoW の IP アドレス/led/on」というアクセスが送られ、LED OFF ボタンなら「http://PicoW の IP アドレス/led/off」にアクセスが送られます．サーバ側ではどの URL にアクセスが来たかを判定して、その内容に応じて LED を点灯または消灯する処理を実行することになります．

```
<form action="/led/on" method="get">
   <button type="submit">LED ON</button>
</form>
```

次に、Pico W が外部からのアクセスを受け付けるためのソケットを作成します．ソケットは通信の入口にあたる仕組みで、ここに接続が来るのを待ち受けます．80 と書いてあるのはポート番号が 80 番だからです．server = socket.socket() は新しいソケットを作成する命令です．次に server.bind(addr) でそのソケットにアドレス（IP とポート番号の組）を結びつけ、server.listen(1) で接続を受け付ける状態にします．

```
import socket
# ---  ソケット作成  ---
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
server = socket.socket()
server.bind(addr)
server.listen(1)

print("Listening on", addr)
```

アクセスを待つ → リクエストを読む → LED を操作 → HTML を返す → 接続を切る

このメインループでは、まず while True: によって無限ループを作り、Pico W が常にブラウザからのアクセスを待ち続ける状態にしています.

外部からアクセスがあると client, addr = server.accept() の部分で処理が止まり、接続が確立すると client には通信用のソケット、addr には接続元の IP アドレスが代入されます. ここで初めて「誰かがアクセスしてきた」とわかるわけです.

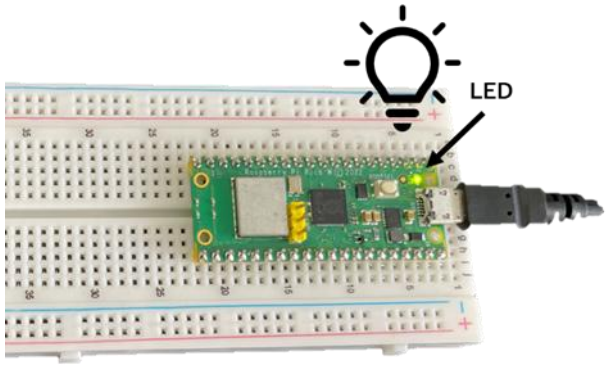
次に request = client.recv(1024).decode() によって接続してきた相手から送られてきた HTTP リクエストを受け取り、バイト列を文字列に変換することで GET /led/on HTTP/1.1 のような内容を確認できるようになります.

そのリクエストの内容を調べ、もし文字列に/led/on が含まれていれば led.value(1) を実行して LED を点灯し、逆に/led/off が含まれていれば led.value(0) を実行して LED を消灯します. こうしてボタンの押し分けによって Pico W が LED を操作できる仕組みになっています. LED の処理が終わると、次はブラウザに対して結果を返します.

最初に client.send('HTTP/1.1 200 OK¥r¥nContent-type: text/html¥r¥n¥r¥n') を送ることでレスポンスのヘッダーを返し、ここで「リクエストを正しく処理しました」という意味の HTTP/1.1 200 OK と「これから送る内容は HTML です」という Content-type: text/html を伝え、最後の ¥r¥n¥r¥n によってヘッダーと本文を区切ります. その後に client.send(html) であらかじめ用意しておいた LED ON/OFF ボタン付きの HTML ページを送り返します. そして最後に client.close() を実行して接続を終了します. HTTP は基本的に 1 回のリクエストごとに接続を閉じる仕組みなので、この手順を繰り返すことで次のアクセスを正しく受け付けられるようになります.

```
# --- メインループ ---
while True:
    client, addr = server.accept()
    print("Client connected from", addr)
    request = client.recv(1024).decode()
    print("Request:", request)


    # リクエストに応じて LED を操作
    if '/led/on' in request:
        led.value(1)
    elif '/led/off' in request:
        led.value(0)

    # HTTP レスポンスを返す
    client.send('HTTP/1.1 200 OK¥r¥nContent-type: text/html¥r¥n¥r¥n')
    client.send(html)
    client.close()
```

ここまで出来たら Pico W にプログラムを書き込み、ブラウザで IP アドレスにアクセスしてみてください. これによって、ON/OFF ボタンが表示されます. ボタンを押して、実際に LED が点灯・消灯することを確認しましょう.



3.4.Pico W でセンサの状態を web ページから見れるようにしよう

この節では、Pico W の内部温度センサ（ADC4）を読み取り、その値を Web サーバ経由で配信します．3.3 節では「LED を Web ページのボタンで ON/OFF」しましたが、今回は逆にセンサ → Web の方向です．最終的に、ブラウザでページを開くと現在の温度が 1 秒ごとに表示されます．それでは実際のプログラムを書いていきましょう．

1）温度センサの値を取得する

Wifi と接続するコードを 3.1 節を参考に書いてください．その後ろにまずは、温度センサの値を取得するコードを書きます．ここで、見慣れない def というものが出てきました．これは、関数というものです．こうすることで read_temp()は temp と等しいものとして使うことができます．例えば t = read_temp()というように書くと def ないが実行され、t には計算後の temp の値が入ります．何度も同じコードを書かなくて済むようにするために関数は使われます．今まで使ってきた print()なども関数の一種です．

```
sensor_temp = ADC(4)
conversion_factor = 3.3 / 65535

def read_temp():
    adc_voltage = sensor_temp.read_u16() * conversion_factor
    temp = 27 - (adc_voltage - 0.706) / 0.001721
    return round(temp, 1)
```

2）温度センサの値を HTML に挿入する

赤く強調表示した個所を見てください．{}の中に変数が書かれています．ほかの章でも説明しましたが、このように書くことで temp は文字ではなく変数として扱われ、temp の代わりにそこに格納された数値が書かれます．

```
html_template = """<!DOCTYPE html>
<html>
<head><title>Pico W Temperature Monitor</title></head>
<body>
    <h1>Pico W Temperature Monitor</h1>
    <p>Current temperature: <b>{temp}</b></p>
</body>
</html>
"""
```

3) サーバを立てて配信する

ポート番号 80 番でサーバを立てて配信します．html_template はあらかじめ作っておいたひな形です． Current temperature: {temp} .format(temp=temp) は、{temp} の部分に 変数 temp の中身を入れて文字列を作る仕組みです．例えば、temp が 28.3 なら Current temperature: 28.3 というHTML 文字列に変換されます．

```
# サーバを立てる
addr = socket.getaddrinfo("0.0.0.0", 80)[0][-1]
server = socket.socket()
server.bind(addr)
server.listen(1)
print("Serving on", addr)

while True:
    client, addr = server.accept()
    request = client.recv(1024)  # リクエスト読み込み
    temp = read_temp()
    html = html_template.format(temp=temp)  # 温度を埋め込む

    # HTTP レスポンス送信
    client.send('HTTP/1.1 200 OK¥r¥nContent-Type: text/html¥r¥n¥r¥n')
    client.send(html)
    client.close()
```

最後にブラウザで http://IP アドレスにアクセスしてください．

【課題 3-4】温度を大きな文字で強調表示してみよう

【課題 3-5】しきい値を超えたら警告をブラウザに表示しよう

＜ヒント＞ html = html_template.format(temp=temp, msg=message)←2 つ以上の引数を持たせるにはこのように書きます．

【課題 3-6】単位を切り替えてみよう

温度を「℃」と「°F」で切り替えられるようにボタンを追加する．°F = ℃ × 9/5 + 32

「°F」が選択されたら表示する数値を変える．

3.5.スライダで光の強さを変えよう

これまで、LED を Web ページから ON/OFF することはできました．これはあくまでつけるか消すかの二択でした．ここでは、スライダを使って、LED の明るさを調整できるようにします．

まずは、LED の回路を組み GP14 につないでください．その後 wifi に接続するコードを書きます．そして、LED の設定を行うコードを書いてください．そこまで出来たら下記に進みます．

1) スライダをサイトに追加する

3.2 節でスライダの追加方法は学びました．ここではスライダに役割を持たせる方法を学びます．まずは、

0~65536 の値のスライダを作成します.

```
<input type = “range” min = “0” max = “65535” value = “0” name = “val”>
```

スライダから値を送信するには<form action="/led" method="get"> にして送信すると、

http://<Pico の IP>/led?val=12345 のようにアクセスされます. サーバ側では URL に含まれる val=12345 を取り出し、その値を PWM の duty に設定します. そして、set ボタンを押すと、その値が送られて、光の強さが変わります. スライダだけでは、アクセスを更新することができないためこのように submit メソッドを用いてアクセスを更新する必要があります.

これを踏まえて HTML は下記のようになります.

```
html_template = """<!DOCTYPE html>
<html>
<head><title>Pico W LED Control</title></head>
<body>
    <h1>LED Brightness Control</h1>
    <form action="/led" method="get">
        <input type="range" min="0" max="65535" value="{val}" name="val">
        <input type="submit" value="Set">
    </form>
    <p>Current value: <b>{val}</b></p>
</body>
</html>
"""
```

2) 光の強さを変更する

まず、ポート番号 80 番でサーバを立てて配信するコードを書いてください.

ブラウザから送られてきたリクエストの文字列の中に「GET /led?val=」という形が含まれているかをまず確認し、もし含まれていればその後ろに書かれている数値を取り出して変数 val に代入し、さらにその数値を整数に変換して LED の PWM のデューティ比として設定することで、ブラウザのスライダで選ばれた値に応じて LED の明るさを変えているという流れになっています.

.split("GET /led?val=")は"GET /led?val="←この文字列の前後で文字列を分割します. [1]を取り出すと"GET /led?val="の後にある文字つまり val が取り出されます.

```
val = 0
while True:
    client, addr = server.accept()
    request = client.recv(1024).decode()

    # URL に  val  が含まれていれば取り出す
    if "GET /led?val=" in request:
        try:
            part = request.split("GET /led?val=")[1]
            val_str = part.split(" ")[0]        #  空白までが数値
            val = int(val_str)
            led.duty_u16(val)
        except:
            pass
```

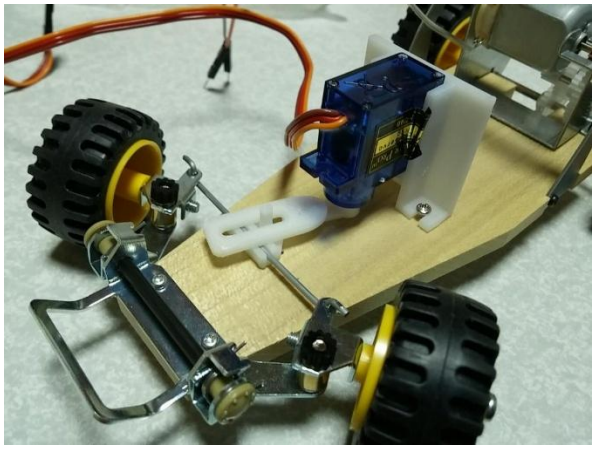
```
    html = html_template.format(val=val)

    # レスポンス送信
    client.send("HTTP/1.1 200 OK¥r¥nContent-Type: text/html¥r¥n¥r¥n")
    client.send(html)
    client.close()
```

【課題 3-7】複数の LED を接続して、それぞれを独立したスライダで制御できるようにしよう.

【課題 3-8】LED の近くに CdS センサを設置し、CdS の値をサイトに表示しよう.

4.ロボティック・システムを作ろう

ここまで、よく耐え抜きました．それでは今までの知識を使って、ロボティック・システムを作ってみましょう．ここではロボティック・システムでよく使われるアクチュエータとして、モータとサーボモータを両方動かすことを試します．想定しているのは、下図のようなステアリングをサーボモータで、前後移動をモータとギアボックスで担うようなロボット・カーです．

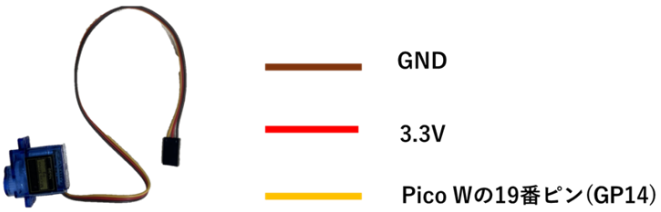


どのようなロボティック・システムを作るのかは皆さんのアイデア次第です．

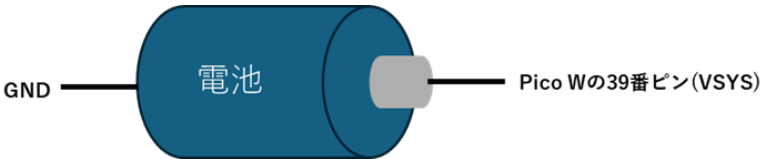
1) 回路を作ろう

まずは、2.5 節を参考にモータを動かすための回路を組んでください．

次に、サーボモータの回路を組みます．サーボモータは、モータに制御回路と位置センサが組み込まれた部品です．通常の DC モータは「速さ・回転方向」を制御しますが、サーボモータは「角度」を直接制御できます．信号線は 3 本で、Vcc（電源）、GND（グラウンド）、信号線（PWM 入力）となっています．



そして、もう一つ、実は、Pico W はパソコンから給電しなくても、電池で動かすことができます．モータを動かすために使っている電池から下記のように配線してください．ただし、電池は抜いておいてください．また、パソコンに電池を入れたまま USB をささないでください．パソコンが壊れる可能性があります．



2) プログラムを作ろう

サーボを動かすには下記のコードを参考にしてください．90 度にすると正面をモータが向いて固まります．サ

ーボの腕を押してみてください.

```
from machine import Pin, PWM

servo = PWM(Pin(14), freq=50)    # 50Hz 固定

def set_angle(deg):
    deg = max(0, min(180, int(deg)))
    us = 600 + (2400 - 600) * deg // 180
    servo.duty_u16(us * 65535 // 20000)

#  例:90 度にするだけ
set  angle(90)
```

それでは、いよいよモータとサーボモータを動かすプログラムを書いていきます. 3.3 節を参考に、まず wifi につないでください. その後、上記のサーボのコードを書いてください.

また、モータの設定コードを書いてください.

```
#  物理ピン: IN1=GP1, IN2=GP2  とする
IN1 = PWM(Pin(1))
IN2 = PWM(Pin(2))
IN1.freq(100)
IN2.freq(100)
```

ここから、WEB でコントロールするための html を書きます. html には Go, Right, Left の 3 つのボタンを設置し、それぞれ、押すと"/car/go", "/car/right", "/car/left"が送られるようにしました. また、Go ボタンを押すと、stop_html が読み込まれて表示されることで、ボタンを切り替えられるようにしました.

```
# ---- HTML（状態で出し分け）----
go_html = """<!DOCTYPE html>
<html>
<head><title>Radio Control</title></head>
<body>
  <h1>Servo Control</h1>
  <form action="/car/go" method="get">
    <button type="submit">Go</button>
  </form>
  <form action="/car/right" method="get">
    <button type="submit">Right</button>
  </form>
  <form action="/car/left" method="get">
    <button type="submit">Left</button>
  </form>
</body>
</html>
"""

stop_html = """<!DOCTYPE html>
<html>
<head><title>Radio Control</title></head>
<body>
  <h1>Servo Control</h1>
  <form action="/car/stop" method="get">
    <button type="submit">Stop</button>
```

```
    </form>
    <form action="/car/right" method="get">
        <button type="submit">Right</button>
    </form>
    <form action="/car/left" method="get">
        <button type="submit">Left</button>
    </form>
</body>
</html>
"""
```

この後にソケット通信を開始するコードを書いてください．メインループでは受け取ったメッセージに応じて動作が切り替わるようにしました．

```
# --- メインループ ---
while True:
    client, addr = server.accept()
    print("Client connected from", addr)
    request = client.recv(1024).decode()
    print("Request:", request)

    if '/car/go' in request:
        IN1.duty_u16(65536)
        IN2.duty_u16(0)
        motor_on = True

    elif '/car/stop' in request:
        IN1.duty_u16(0)
        IN2.duty_u16(0)
        motor_on = False

    elif '/car/right' in request:
        servo_angle = 60
        set_angle(servo_angle)

    elif '/car/left' in request:
        servo_angle = 120
        set_angle(servo_angle)

    # 現在状態に応じて HTML を返す
    html = stop_html if motor_on else go_html

    client.send('HTTP/1.1 200 OK¥r¥nContent-Type: text/html¥r¥nConnection: close¥r¥n¥r¥n')
    client.sendall(html)
    client.close()
```

これで、モータとサーボモータを同時に動かすことができます．

【課題 4-1】 他人のモータをハッキングしよう

http://IP アドレスに命令を加えて例えば http://IP アドレス/car/stop を書き込むことで無理やり他人のモータを停止させてみましょう．IP アドレスをうまく聞き出して下さい．

このように IoT 機器はセキュリティが甘いと他人から勝手に動かされてしまう可能性があります．

【課題 4-2】 自分なりのロボティック・システムを開発しよう

今まで習ったことを生かして便利な機器を作ってください.

注）レポート、プレゼン資料を作成すること.

資料　MicroPython の基本文法[4]

下図のように、灰色塗りつぶし枠内は**作成するプログラム**を、薄緑塗りつぶし枠内は**表示結果**を示す.

```
作成するプログラムコード
```

```
表示結果
```

1. インデント

インデント（字下げ）はコード行の先頭の空白を指す. 下の 2 つの例では、print()の前には少なくとも 1 つのスペースを入れる必要がある. さもないと"Invalid syntax"（無効な構文）というエラーメッセージが表示される. スペースキーの代わりに Tab キーを用いてもよいが、スペースと Tab の混在はエラーを招く危険性がある.

```
if 8 > 5:
print("Eight is greater than Five!")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

同じブロックのコード内で同じ数のスペースを使用しなければ、Python はエラーを出す.

```
if 8 > 5:
print("Eight is greater than Five!")
        print("Eight is greater than Five")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

if 8 > 5 の時に print("Eight is greater than Five!") したいのであれば、下記のように記す必要がある.

```
if 8 > 5:
    print("Eight is greater than Five!")
```

2. コメント

コード内のコメントは、コードを理解するのに役立ち、全体のコードの可読性を高め、テスト中にコードの一部をコメントアウト（実行されないようにコメント化すること）して、その部分のコードが実行されないようにできる.

2.1. 単一行コメント

MicroPython における単一行コメントは # で始まり、その後のテキストは行の終わりまでコメントと見なされる. コメントはコードの前後に配置することができる.

```
print("hello world")    #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

---

[4] https://docs.sunfounder.com/projects/esp32-starter-kit/ja/latest/micropython/python_start/syntax/micropython_basic_syntax.html を改編

コメントはコードを説明するためのテキストに限らない. コードの一部をコメントアウトして、micropython がコードを実行しないようにすることもできる. これは、プログラム・エラーをデバック（バグを取り除く）でよく利用される.

```
#print("Can't run it ！")
print("hello world") #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

2.2. 複数行コメント

複数行にわたってコメントする場合は、複数の # を使用できる.

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

または、複数行の文字列（トリプルクオート、クオート記号”を 3 つ）をコードに追加し、その中にコメントを入れることができる. これは、プログラムの表題などに利用される.

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

3. print( )

print() 関数は、指定されたメッセージを画面や他の標準出力デバイスに印刷（表示）する. メッセージは文字列でも、他のオブジェクトでも構わない. オブジェクトは画面に表示する前に文字列に変換される.

・複数のオブジェクトを印刷する

```
print("Welcome!", "Enjoy yourself!")
```

```
>>> %Run -c $EDITOR_CONTENT
Welcome! Enjoy yourself!
```

・タプル（複数の値をひとまとめにしたもの）を印刷する:

```
x = ("pear", "apple", "grape")
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
('pear', 'apple', 'grape')
```

4. 変数

変数は値を格納するために使われる. 割り当てる際に変数のデータタイプを指定する必要はない. 変数名を命名する際は、以下のルールに従う必要がある.

- 変数名には半角の数字、文字、アンダースコア（“_”）のみを含めることができる
- 変数名の最初は文字またはアンダースコアでなければならない

・ 変数名は大文字と小文字を区別する

なお、MicroPython の変数の型には、整数、浮動小数点、文字列、ブール値（True、False）などが使える．整数型は任意精度整数で範囲に制限はなく（メモリの制約内）、浮動小数点型は通常 IEEE754 の倍精度（64bit）で約$\pm 1.8\times 10^{308}$ まで扱える．

4.1. 変数を作成する

変数は初めて値を割り当てたときに作成される．特定の型宣言を使用する必要はなく、変数を設定した後で型を変更することもできる．

```
x = 8           # x is of type int
x = "lily" # x is now of type str
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
lily
```

4.2. キャスト

変数にデータタイプ（整数型、実数型、文字列など）を指定したい場合は、キャスト（型変換）によって行うことができる．

```
x = int(5)       # y will be 5
y = str(5)       # x will be '5'
z = float(5)     # z will be 5.0
print(x,y,z)
```

```
>>> %Run -c $EDITOR_CONTENT
5 5 5.0
```

4.3. タイプを取得する

type() 関数を使用して変数のデータタイプを取得することができる．

```
x = 5
y = "hello"
z = 5.0
print(type(x),type(y),type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'int'> <class 'str'> <class 'float'>
```

4.4. シングルクォートまたはダブルクォート?

MicroPython では、シングルクォート（’）またはダブルクォート（”）を使用して文字列変数を定義できる．

```
x = "hello"
# is the same as
x = 'hello'
```

4.5. 大文字・小文字の区別

変数名は大文字と小文字を区別する．

```
a = 5
A = "lily"
#A will not overwrite a
print(a, A)
```

```
>>> %Run -c $EDITOR_CONTENT
5 lily
```

5. 条件分岐

条件によってコードを実行したい場合には、条件分岐が必要となる.

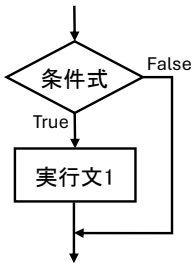
5.1. if

```
if test expression:
    statement(s)
```

ここでは、プログラムが`条件式`を評価し、条件式の結果が True（真）の場合のみ実行文 1 を実行する. 条件式の結果が False（偽）であれば、実行文 1 は実行されない.

MicroPython では、同じインデントされた実行文が *if* ステートメントが真である場合の実行本体を意味する. 本体はインデントで始まり、最初の非同一インデント行の前で終わる.

Python はゼロでない値を「True」と解釈する. None と 0 は共に「False」と解釈される.



if ステートメントのフローチャート

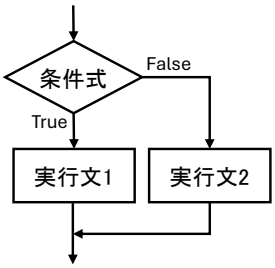
```
num = 8
if num > 0:
    print(num, "is a positive number.")
print("End with this line")
```

```
>>> %Run -c $EDITOR_CONTENT
8 is a positive number.
End with this line
```

5.2. if...else

```
if test expression:
    Body of if
else:
    Body of else
```

if..else ステートメントは条件式を評価し、テスト条件が *True* の場合に実行文 1 を実行する. 条件が *False* の場合、 *else* の実行文 2 が実行される.



if...else ステートメントのフローチャート

```
num = -8
if num > 0:
    print(num, "is a positive number.")
else:
    print(num, "is a negative number.")
```

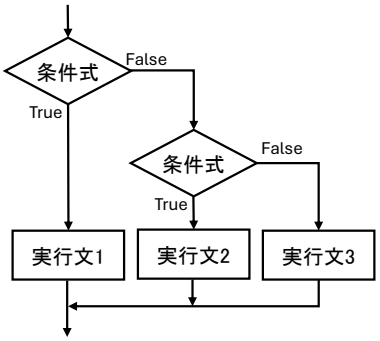
```
>>> %Run -c $EDITOR_CONTENT
-8 is a negative number.
```

5.3. if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

Elif は *else if* の略であり、複数の表現をチェックすることができる.

if の条件が False の場合、次の elif ブロックの条件がチェックされ、以下同様に続く. すべての条件が *False* であれば、 *else* の本体が実行される. 複数ある *if...elif...else* ブロックのうち、条件に応じて実行されるのは 1 つだけである. *if* ブロックは 1 つの *else* ブロックを持つことができるが、複数の *elif* ブロックを持つことができる.



if...elif...else ステートメントのフローチャート

```
x = 10
y = 9
if x > y:
    print("x is greater than y")
elif x == y:
```

```
    print("x and y are equal")
else:
    print("x is greater than y")
```

```
>>> %Run -c $EDITOR_CONTENT
x is greater than y
```

5.4. Nested if

if ステートメントを別の if ステートメントに埋め込むことができ、これをネストされた if ステートメントと呼ぶ.

```
x = 67
if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
>>> %Run -c $EDITOR_CONTENT
Above ten,
and also above 20!
```

5.5. try

MicroPython の try は、プログラム中で起こる「エラー（例外）」を安全に処理するための仕組みである. センサ・通信・ネットワーク・ファイル操作など「失敗しやすい処理」を守るために使われる.

基本構造は、

```
try:
    # エラーが起きるかもしれない処理
except:
    # エラーが起きたときの処理
```

である.

ここで、MicroPython の主な実行時エラーをまとめる. ”raise” は エラーを自分で発生させるための命令である.

表 5.1. MicroPython の主な実行時エラー

種類	エラー	説明
データ・型・値	TypeError	型が合わない操作をしたとき
	ValueError	存在しないピン番号、範囲外の値など、値が不正なとき
	IndexError	リストやタプルの範囲外アクセス
	KeyError	辞書に存在しないキー
	ZeroDivisionError	0 で割ったとき
	UnicodeError	文字列のエンコード・デコードに失敗
OS・ハードウェア	OSError	OS レベルの失敗
	MemoryError	メモリ不足

	RuntimeError	実行中の一般的な失敗
モジュール・通信	ImportError	モジュールが見つからない
	AssertionError	通信プロトコルの内部チェックが失敗
	EOFError	入力が予期せず終わった

6. while ループ

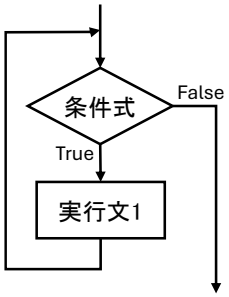
while 文はループ内でプログラムを実行するために使われる. すなわち、特定の条件下で繰り返し処理する必要があるタスクを処理するために、ループ内でプログラムを実行する.

基本的な形式は以下の通りである.

```
while test expression:
    Body of while
```

while ループでは、まず条件式をチェックする. 条件式が True と評価された場合のみ、while の本体に入る. 一回のイテレーション（同じ処理の繰り返し）の後、再び条件式をチェックする. このプロセスは条件式が False と評価されるまで続く.

MicroPython では、 while ループの本体はインデントによって決定される. 本体はインデントで始まり、最初の非同一インデント行で終わる. Python はゼロでない任意の値を True と解釈する. None と 0 は False と解釈される.



while ループのフローチャート

```
x = 10
while x > 0:
    print(x)
    x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
```

6.1. break 文

break 文を使えば、while 条件が真であってもループを停止させることができる.

```
x = 10
while x > 0:
        print(x)
        if x == 6:
                break
        x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
```

6.2. else を持つ while ループ

if ループのように、 *while* ループにもオプショナルの *else* ブロックを持たせることができる. *while* ループの条件が *False* と評価された場合、 *else* 部分が実行される.

```
x = 10
while x > 0:
        print(x)
        x -= 1
else:
        print("Game Over")
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
Game Over
```

7. for ループ

for ループは、リストや文字列など、任意の項目のシーケンスをトラバースすることができる. for ループの構文フォーマットは以下の通りである.

```
for val in sequence:
        Body of for
```

ここで、 *val* は各イテレーションでシーケンス内の項目の値を取得する変数である. ループはシーケンスの最後の項目に到達するまで続く. 本体をコードの残りの部分から分離するためにインデントを使用する.

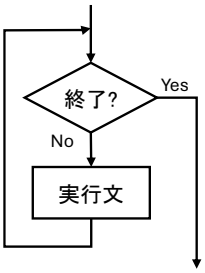


図 7.1. for ループのフローチャート

```
numbers = [1, 2, 3, 4]
sum = 0
for val in numbers:
        sum = sum+val
print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 10
```

7.1. break 文

break 文を使えば、全ての項目をループする前にループを停止させることができる.

```
numbers = [1, 2, 3, 4]
sum = 0
for val in numbers:
        sum = sum+val
        if sum == 6:
                break
print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 6
```

7.2. continue 文

continue 文を使えば、ループの現在のイテレーションを停止し、次のイテレーションで続行することができる.

```
numbers = [1, 2, 3, 4]
for val in numbers:
        if val == 3:
                continue
        print(val)
```

```
>>> %Run -c $EDITOR_CONTENT
1
2
4
```

7.3. range()関数

range()関数を使って一連の数値を生成することができる. range(6)は 0 から 5 の間の数値（6 つの数値）を生成する.

また、range(start, stop, step_size)として開始、停止、ステップサイズを定義することもできる. 指定されていない場合、step_size はデフォルトで 1 になる.

range の意味では、オブジェクトは「怠惰」です. なぜなら、オブジェクトを作成しても、それが「含む」すべての数値を生成しないからである. しかし、これは in、len、_getitem_操作をサポートしているため、イテレータではない. この関数はすべての値をメモリに格納しない. したがって、開始、停止、ステップサイズを覚えておいて、進行中に次の数値を生成する.

この関数にすべての項目を出力させるには、list()関数を使用することができる.

```
print(range(6))
print(list(range(6)))
print(list(range(2, 6)))
print(list(range(2, 10, 2)))
```

```
>>> %Run -c $EDITOR_CONTENT
range(0, 6)
[0, 1, 2, 3, 4, 5]
[2, 3, 4, 5]
[2, 4, 6, 8]
```

range() を *for* ループ内で使用して数値のシーケンスを繰り返すことができる. len()関数と組み合わせてインデックスを使用してシーケンスをトラバースすることもできる.

```
fruits = ['pear', 'apple', 'grape']
for i in range(len(fruits)):
    print("I like", fruits[i])
```

```
>>> %Run -c $EDITOR_CONTENT
I like pear
I like apple
I like grape
```

7.4. for ループでの else

for ループにはオプショナルな *else* ブロックも持たせることができる. ループに使用されるシーケンスの項目が使い果たされた場合、 *else* 部分が実行される.

break キーワードを使用して *for* ループを停止させることができる. この場合、 *else* 部分は無視される. したがって、中断が発生しなければ、 *for* ループの *else* 部分が実行される.

```
for val in range(5):
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
2
3
4
Finished
```

ループが break 文で停止された場合、else ブロックは実行されない.

```
for val in range(5):
    if val = = 2: break
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
```

8. 関数

MicroPython において、関数は特定のタスクを実行する関連するステートメントのグループである. 関数はプログラムをより小さなモジュールブロックに分割するのに役立つ. プランが大きくなるにつれて、関数はそれをより整理され、管理しやすくする. また、重複を避け、コードを再利用可能にする.

8.1. 関数の作成

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

- 関数は def キーワードを使用して定義される.
- 関数を一意に識別する関数名. 関数の命名は変数の命名と同じで、以下のルールに従う.
  - 数字、文字、アンダースコアのみを含むことができる.
  - 最初の文字は文字またはアンダースコアでなければならない.
  - 大文字と小文字を区別する.
- パラメータ(引数)は、値を関数に渡すために使用する.
- コロン(:)は関数ヘッダの終わりを示す.
- オプショナルなドキュメント文字列は、関数の機能を記述するために使用される. 通常、ドキュメント文字列を複数行に拡張できるようにトリプルクオートを使用する.
- 関数の本体を構成する 1 つ以上の有効な Micropython ステートメント. ステートメントは同じインデントレベル(通常は 4 スペース)でなければならない.
- どんな理由があっても、ステートメントを含まない関数がある場合、エラーを避けるために pass ステートメントを入れること.
- 関数から値を返すオプショナルな return ステートメント.

8.2. 関数の呼び出し

関数を呼び出すには、関数名の後に括弧を追加する.

```
def my_function():
    print("Your first function")
my_function()
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
```

8.3. return ステートメント

return ステートメントは関数から出て、それが呼び出された場所に戻るために使用される.

```
return [expression_list]
```

このステートメントには、評価されて値を返す式を含むことができる. ステートメントに式がない場合、または return ステートメント自体が関数内に存在しない場合、関数は None オブジェクトを返す.

```
def my_function():
            print("Your first function")
print(my_ function())
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
None
```

ここで、 None は戻り値である. なぜなら return ステートメントが使用されていないからである.

8.4. 引数

情報を引数として関数に渡すことができる. 関数名の後の括弧内に引数を指定する. 必要なだけ多くの引数を追加でき、コンマで区切る.

```
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
welcome("Lily", "Welcome to China!")
```

```
>>> %Run -c $EDITOR_ CONTENT
Hello Lily, Welcome to China!
```

8.5. 引数の数

デフォルトでは、関数は正しい数の引数で呼び出されなければならない. すなわち、関数が 2 つのパラメータを期待している場合、関数を 2 つの引数で呼び出す必要がある.

```
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
welcome("Lily", "Welcome to China!")
```

ここでは、関数 welcome()は 2 つのパラメータを持っている. この関数を 2 つの引数で呼び出すと、エラーなくスムーズに機能する. 引数の数が異なる場合、インタープリタはエラーメッセージを表示する. 以下は、この関数を一つの引数と引数なしで呼び出した例と、それぞれのエラーメッセージである.

```
welcome("Lily")  #Only one argument
```

```
>>> %Run -c $EDITOR_ CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 1 were given
```

```
welcome()#No arguments
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 0 were given
```

8.6. デフォルト引数

MicroPython では、代入演算子(=)を使用してパラメータにデフォルト値を提供することができる. 引数なしで関数を呼び出すと、デフォルト値が使用される.

```
def welcome(name, msg = "Welcome to China!"):
```

```
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
welcome("Lily")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello Lily, Welcome to Japan!
```

この関数では、パラメータ name にはデフォルト値がなく、呼び出し時に必須である. 一方で、パラメータ msg のデフォルト値は「日本へようこそ！」である. したがって、呼び出し時にはオプショナルである. 値が提供された場合、デフォルト値は上書きされる.

関数の任意の数の引数にデフォルト値を持たせることができる. しかし、デフォルト引数がある場合、右側のすべての引数にもデフォルト値が必要である. これは、デフォルト引数の後に非デフォルト引数を続けることはできないことを意味する. 例えば、上記の関数ヘッダを以下のように定義した場合、エラーメッセージを受け取る.

```
def welcome(name = "Lily", msg)
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: non-default argument follows default argument
```

8.7. キーワード引数

特定の値で関数を呼び出すと、これらの値は位置に基づいて引数に割り当てられる. 例えば、上記の関数 welcome()で、welcome("Lily", "中国へようこそ")と呼び出したとき、"Lily"の値が name に、同様に"中国へようこそ"がパラメータ msg に割り当てられる.

MicroPython ではキーワード引数を使って関数を呼び出すことができる. この方法で関数を呼び出すと、引数の順序（位置）を変更することができる.

```
# keyword arguments
welcome(name = "Lily",msg = "Welcome to China!")
```

```
# keyword arguments (out of order)
welcome(msg = "Welcome to China ！",name = "Lily")
```

```
#1 positional, 1 keyword argument
welcome("Lily", msg = "Welcome to China!")
```

関数呼び出し時に位置引数とキーワード引数を混在させることができることがわかる. しかし、キーワード引数は位置引数の後に来なければならないことを覚えておく必要がある. キーワード引数の後に位置引数があるとエラーになる. 例えば、関数の呼び出しが以下のようであれば、エラーが発生する.

```
welcome(name="Lily","Welcome to China!")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
SyntaxError: non-keyword arg after keyword arg
```

8.8. 任意の引数

関数に渡される引数の数が事前にわからない場合がある. 関数定義で、パラメータ名の前にアスタリスク(*)を追加することができる.

```
def welcome(*names):
    """This function welcomes all the person
    in the name tuple"""
    #names is a tuple with arguments
    for name in names:
        print("Welcome to China!", name)
welcome("Lily","John","Wendy")
```

```
>>> %Run -c $EDITOR  CONTENT
Welcome to China! Lily
Welcome to China! John
Welcome to China! Wendy
```

ここで、複数の引数で関数を呼び出した. これらの引数は関数に渡される前にタプルにパックされる. 関数の中では、for ループを使用してすべての引数を取り出す.

8.9. 再帰

Python では、関数が他の関数を呼び出すことができる. 関数が自分自身を呼び出すことも可能である. このタイプの構造は再帰関数と呼ばれる. これはデータをループして結果に到達するという意味で有利である.

開発者は再帰に注意する必要がある. 簡単に終了しない関数を書いたり、過剰なメモリやプロセッサパワーを使用したりすることがある. しかし、正しく書かれた再帰は、非常に効率的で数学的にエレガントなプログラミングのアプローチになることがある.

```
def rec_func(i):
    if(i > 0):
        result = i + rec_func(i - 1)
        print(result)
    else:
        result = 0
    return result
rec_func(6)
```

```
>>> %Run -c $EDITOR  CONTENT
1
3
6
10
15
21
```

この例では、rec_func()は自分自身を呼び出すように定義された再帰関数である. i 変数をデータとして使用し、再帰するたびに(-1)減少する. 条件が 0 より大きくない場合(つまり 0 の場合)、再帰は終了する.

初心者にとっては、どのように動作するかを把握するまでに時間がかかるかもしれないが、テストして修正することが最善の方法である.

再帰の利点は、

- 再帰関数はコードをクリーンでエレガントにする.
- 複雑なタスクを再帰を使ってよりシンプルなサブ問題に分割することができる.
- いくつかのネストされたイテレーションを使用するよりも、再帰を使ったシーケンス生成が容易である.

再帰の欠点は、

- 再帰の背後にある論理を追うのが難しい.

- 再帰呼び出しは高コスト（非効率的）で、多くのメモリと時間を取る.
- 再帰関数はデバッグが難しい.

8.10. MicroPython の組み込み関数とメソッド

表 8.1 は MicroPython に組み込まれている主な関数である.

表 8.1. 主な組み込み関数

分類	組み込み関数	機能
数値・型変換	int()	整数に変換
	float()	浮動小数点に変換
	str()	文字列に変換
	bool()	真偽値に変換
	abs()	絶対値
	round()	四捨五入
	divmod(a,b)	a÷b の商と余りの 2 つを返す
シーケンス操作	len()	リスト・タプル・文字列の長さ
	max()	最大値
	min()	最小値
	sum()	合計
	sorted()	ソートしたリストを返す
	enumerate()	インデックスと値を同時に取り出す
	range()	数列を生成
	repr()	“正確な表現”を文字列として返す
型チェック・属性取得	type()	型を返す
	isinstance(obj,type)	型チェック
	id()	オブジェクトの識別子
	dir()	オブジェクトの属性一覧
データ構造	list()	リストに変換
	tuple()	タプルに変換
	dict()	辞書に変換
	set()	集合に変換
計算・評価	eval()	文字列を式として評価
	pow(a,b)	a の b 乗
	hex()	16 進文字列に変換
	bin()	2 進文字列に変換
入出力	print()	出力
	input()	入力

その他	help()	ヘルプ
	globals()/locals()	名前空間を返す
	super()	親クラスの呼び出し

表 8.2 は MicroPython に組み込まれている主なメソッドである．関数とメソッドの違いは呼び出し方の違いにある．メソッドの呼び出し方は「オブジェクト．メソッド名()」であり、型（オブジェクト）に属している関数とすることができる．

表 8.2．主な組み込みメソッド

種別	メソッド	機能
文字列型	.encode()	文字列 → bytes に変換（通信で必須）
	.split()	区切り文字で分割
	.replace()	置換
	.format()	文字列フォーマット
	.join()	リストを結合
バイト列	.decode()	bytes → 文字列に変換（受信で必須）
	.find()	部分一致検索
	.hex()	16 進文字列に変換
リスト	.append()	末尾に追加
	.pop()	末尾を取り出す
	.remove()	指定値を削除
	.sort()	並べ替え
辞書	.get()	キーが無い時も安全に取得
	.keys()	キー一覧
	.values()	値一覧
	.items()	(キー,値) のペア一覧
タプル	.count()	値の出現回数
	.index()	値の位置

9. データ型

9.1. 組み込みデータ型

MicroPython には以下のデータ型がある．

- テキスト型: str
- 数値型: int, float, complex
- シーケンス型: list, tuple, range
- マッピング型: dict

- セット型: set, frozenset
- ブール型: bool
- バイナリ型: bytes, bytearray, memoryview

9.2. データ型の取得

type() 関数を使用することで、任意のオブジェクトのデータ型を取得できる.

```
a = 6.8
print(type(a))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'float'>
```

9.3. データ型の設定

MicroPython では、データ型を設定する必要はない. 変数に値を割り当てたときに決定される.

```
x = "welcome"
y = 45
z = ["apple", "banana", "cherry"]
print(type(x))
print(type(y))
print(type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'str'>
<class 'int'>
<class 'list'>
>>>
```

9.4. 特定のデータ型の設定

データ型を指定したい場合は、以下のコンストラクタ関数を使用できる.

表 9.1. データ型の例

例	データ型
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = str("Hello World")	str
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes

x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

いくつかを出力して結果を見ることができる.

```
a = float(20.5)
b = list(("apple", "banana", "cherry"))
c = bool(5)
print(a)
print(b)
print(c)
>>> %Run -c $EDITOR_CONTENT
20.5
['apple', 'banana', 'cherry']
True
>>>
```

9.5. 型変換

int()、float()、complex() を使用して、一つの型から別の型に変換できる. Python では、コンストラクタ関数を使用して型変換を行う.

- int() - 整数定数、実数定数（すべての小数を除去する）、または文字列定数（文字列が整数を表す場合）から整数を作り出す
- float() - 整数定数、実数定数、または文字列定数（文字列が実数または整数を表す場合）から実数を作り出す.
- str() - 文字列、整数定数、実数定数を含む幅広いデータ型から文字列を構築する.

```
a = float("5")
b = int(3.7)
c = str(6.0)
print(a)
print(b)
print(c)
```

注意: 複素数は他の数値型に変換することはできない.

10. 演算子

演算子は、変数と値に対して操作を行うために使用される.

10.1. 算術演算子

算術演算子を使用して、一般的な数学的操作を行うことができる.

表 10.1. 算術演算子

演算子	名前
+	たし算
-	ひき算
*	かけ算
/	わり算

%	剰余
**	累乗
//	割り算の余り

```
x = 5
y = 3
a = x + y
b = x - y
c = x * y
d = x / y
e = x % y
f = x ** y
g = x // y
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
>>> %Run -c $EDITOR  CONTENT
8
2
15
1.66667
2
125
1
8
2
15
>>>
```

10.2. 代入演算子

代入演算子は、変数に値を代入するために使用できる.

表 10.2. 代入演算子

オペレータ	例	同じ表現
=	a = 6	a =6
+=	a += 6	a = a + 6
-=	a -= 6	a = a - 6
*=	a *= 6	a = a * 6
/=	a /= 6	a = a / 6
%=	a %= 6	a = a % 6
**=	a **= 6	a = a ** 6
//=	a //= 6	a = a // 6

&=	a &= 6	a = a & 6
|=	a |= 6	a = a | 6
^=	a ^= 6	a = a ^ 6
>>=	a >>= 6	a = a >> 6
<<=	a <<= 6	a = a << 6

```
a = 6
a *= 6
print(a)
```

```
%Run test.py
36
```

10.3. 比較演算子

比較演算子は 2 つの値を比較するために使用される.

表 10.3. 比較演算子

オペレータ	名前
==	等しい
!=	等しくない
<	より小さい
>	より大きい
>=	以上
<=	以下

```
a = 6
b = 8
print(a>b)
```

```
%Run test.py
False
```

False を返す. なぜなら、 **a** は **b** より小さいからである.

10.4. 論理演算子

論理演算子は条件文を組み合わせるために使用される.

表 10.4. 論理演算子

演算子	説明
and	両方の文が真であれば True を返す
or	一方の文が真であれば True を返す
not	結果を反転させ、結果が真であれば False を返す

```
a = 6
print(a > 2 and a < 8)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

10.5. 同一性演算子

同一性演算子は、オブジェクトが等しいかではなく、実際に同じオブジェクトで、同じメモリ位置にあるかどうかを比較するために使用される.

表 10.5. 同一性演算子

演算子	説明
is	両方の変数が同じオブジェクトであれば True を返す
is not	両方の変数が同じオブジェクトでなければ True を返す

```
a = ["hello", "welcome"]
b = ["hello", "welcome"]
c = a
print(a is c)
# returns True because z is the same object as x
print(a is b)
# returns False because x is not the same object as y, even if they have the same content
print(a == b)
# returns True because x is equal to y
```

```
>>> %Run -c $EDITOR_CONTENT
True
False
True
>>>
```

10.6. メンバーシップ演算子

メンバーシップ演算子は、シーケンスがオブジェクト内に存在するかどうかをテストするために使用される.

表 10.6. メンバーシップ演算子

演算子	説明
in	指定された値のシーケンスがオブジェクトに存在する場合は True を返す
not in	指定された値のシーケンスがオブジェクトに存在しない場合は True を返す

```
a = ["hello", "welcome", "Goodmorning"]
print("welcome" in a)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

10.7. ビット演算子

ビット演算子は、2 真数の数値を比較するために使用される.

表 10.7. ビット演算子

演算子	名前	説明
&	AND	両方のビットが 1 の場合、各ビットを 1 に設定
|	OR	二つのビットのうち一方が 1 の場合、各ビットを 1 に設定
^	XOR	二つのビットのうち一方だけが 1 の場合、各ビットを 1 に設定
~	NOT	すべてのビットを反転
<<	ゼロ埋め左シフト	右からゼロを押し込んで左シフトし、最左ビットを落とす
>>	符号付き右シフト	最左ビットのコピーを左から押し込んで右シフトし、最右ビットを落とす

```
num = 2
print(num & 1)
print(num | 1)
print(num << 1)
```

```
>>> %Run -c $EDITOR_CONTENT
0
3
4
>>>
```

11.リスト

リストは、複数のアイテムを単一の変数で格納するために使用され、角括弧を使って作成される.

```
B_list = ["Blossom", "Bubbles","Buttercup"]
print(B_list)
```

リストのアイテムは変更可能で、順序があり、重複した値を許可する. リストアイテムはインデックス付けされ、最初のアイテムがインデックス[0]、2 番目のアイテムがインデックス[1]などである.

```
C_list = ["Red", "Blue", "Green", "Blue"]
print(C_list)                  # duplicate
print(C_list[0])
print(C_list[1])               # ordered
C_list[2] = "Purple"           # changeable
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Blue']
Red
Blue
['Red', 'Blue', 'Purple', 'Blue']
```

リストは異なるデータ型を含むことができる.

```
A_list = ["Banana", 255, False, 3.14]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', 255, False, 3.14]
```

11.1. リストの長さ

リスト内のアイテム数を確認するには、len()関数を使用する.

```
A_list = ["Banana", 255, False, 3.14]
print(len(A_list))
```

```
>>> %Run -c $EDITOR_CONTENT
4
```

11.2. リストアイテムの確認

リストの 2 番目のアイテムを出力する.

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[1])
```

```
>>> %Run -c $EDITOR_CONTENT
[255]
```

リストの最後のアイテムを出力する.

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[-1])
```

```
>>> %Run -c $EDITOR_CONTENT
[3.14]
```

2 番目と 3 番目のアイテムを出力する.

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[1:3])
```

```
>>> %Run -c $EDITOR_CONTENT
[255, False]
```

11.3. リストアイテムの変更

2 番目と 3 番目のアイテム（要素）を変更する.

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:3] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', 3.14]
```

2 番目の値を 2 つの値で置き換えて変更する.

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:2] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', False, 3.14]
```

11.4. リストアイテムの追加

append()メソッド（関数）を使用してアイテム（要素）を追加する.

```
C_list = ["Red", "Blue", "Green"]
C_list.append("Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Orange']
```

2 番目の位置にアイテム（要素）を挿入する.

```
C_list = ["Red", "Blue", "Green"]
C_list.insert(1, "Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Orange', 'Blue', 'Green']
```

11.5. リストアイテムの削除

remove() メソッド（関数）は指定されたアイテム（要素）を削除する.

```
C_list = ["Red", "Blue", "Green"]
C_list.remove("Blue")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

pop() メソッド（関数）は指定されたインデックスを削除する. インデックスを指定しない場合、pop()メソッドは最後のアイテムを削除する.

```
A_list = ["Banana", 255, False, 3.14, True,"Orange"]
A_list.pop(1)
print(A_list)
A_list.pop()
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
255
['Banana', False, 3.14, True, 'Orange']
'Orange'
['Banana', False, 3.14, True]
```

del キーワードも指定されたインデックスを削除する.

```
C_list = ["Red", "Blue", "Green"]
del C_list[1]
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

clear()メソッドはリストを空にする. リストは残るが、内容はない.

```
C_list = ["Red", "Blue", "Green"]
C_list.clear()
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
[]
```

12. ロボティック・システムのプログラミング

MicroPython には、Raspberry Pi Pico W に接続したセンサからデータを取り込む、あるいはアクチュエータを動かす専用のライブラリが用意されている.

12.1. モジュール

表 12.1 に示すように、ライブラリの中には機能ごとにモジュールが用意されており、下記の例に示すように、必要な機能を import する必要がある.

表 12.1. 代表的な MicroPython モジュール

モジュール名	主な機能

machine	GPIO、PWM、ADC、I2C などのハードウェア制御
time	時間の取得、待機処理
network	Wi-Fi や通信関連の制御
socket	ファイルシステムの操作
math	数学関数

各々のモジュールには、更に機能を担当するクラスが内蔵されている.

1) machine モジュール

machine モジュールは、表 12.2 に示すように、マイコンのハードウェア制御の中心的な役割を担うモジュールで、GPIO、I2C、SPI、PWM、ADC、タイマー、割り込み、リセットなど、低レベルのハードウェア機能に直接アクセスするための関数やクラスが含まれている.

表 12.2. machine モジュールの主な機能とクラス

機能カテゴリ	内容
GPIO 制御	Pin クラス：入力・出力ピンの制御
PWM 制御	PWM クラス：LED の明るさやモータ速度制御など
アナログ入力	ADC クラス：センサなどからのアナログ値読み取り
通信	I2C, SPI, UART クラス：外部デバイスとのデータ通信
タイマー	Timer クラス：定期的な処理の実行
電源管理	deepsleep(), lightsleep()：省電力モード
システム制御	reset(), freq(), unique_id()：リセット、CPU 周波数、ID 取得など
割り込み制御	enable_irq(), disable_irq()：割り込みの有効化・無効化

下記は、machine モジュールを使った LED を点滅させるプログラム例である.

```
from machine import Pin, Timer

led = Pin(25, Pin.OUT)    # Pico のオンボード LED
timer = Timer()

def blink(timer):
        led.toggle()

timer.init(freq=2, mode=Timer.PERIODIC, callback=blink)    # 2Hz で点滅
```

2) time モジュール

time モジュールは、時間に関する操作を行うための標準モジュールで、プログラムの制御やタイミング調整に欠かせない機能を提供する.

表 12.3. time モジュールの主な機能とクラス

関数名	説明

time.sleep(s)	秒単位で処理を停止（例：sleep(1) → 1 秒停止）
time.sleep_ms(ms)	ミリ秒単位で停止（例：sleep_ms(100) → 0.1 秒停止）
time.sleep_us(us)	マイクロ秒単位で停止（例：sleep_us(50) → 50μ 秒停止）
time.ticks_ms()	ミリ秒単位の経過時間（起動からの時間など）
time.ticks_us()	マイクロ秒単位の経過時間
time.ticks_cpu()	高精度な CPU クロック時間（一部のポートのみ）
time.ticks_diff(t1, t0)	2 つの tick 値の差分を計算（オーバーフロー対応）
time.ticks_add(t, delta)	tick 値に時間を加算（将来のイベントタイミング計算に便利）
time.localtime()	現在のローカル時間（RTC が有効な場合）
time.gmtime()	UTC 時間を取得
time.time()	UNIX 時間（秒）を取得

下記は time モジュールを用いて、LED を 1 秒ごとに点滅させるプログラム例である.

```
from machine import Pin
import time

led = Pin(25, Pin.OUT)

while True:
    led.toggle()
    time.sleep(1)    # 1 秒待つ
```

3) network モジュール

network モジュールは、Wi-Fi や Ethernet などのネットワーク接続を制御するためのモジュールである. Raspberry Pi Pico W のようなネットワーク対応マイコンでは、インターネット接続やローカル通信を行う際に不可欠である. Network クラスの目的は、Wi-Fi, Ethernet などのネットワークインターフェースの初期化、接続、設定であり、用途は、IoT デバイスのインターネット接続、Web サーバ構築、MQTT 通信などである.

表 12.4. network モジュールの主な機能とクラス

クラス・定数	説明
network.WLAN	Wi-Fi 接続用のクラス（Pico W ではこれを使用）
network.STA_IF	ステーションモード（既存の Wi-Fi に接続）
network.AP_IF	アクセスポイントモード（自分が Wi-Fi を提供）
active(True/False)	インターフェースの有効化・無効化
connect(ssid, password)	Wi-Fi ネットワークへの接続
status()	状態コードを返す
isconnected()	接続状態の確認
ifconfig()	IP アドレスなどのネットワーク情報取得. ifconfig() はタプル（4 つの値）を返すが、status[0]は IP アドレスである.

下記は、WiFi に接続するプログラムの例である.

```
import network
import time

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('SSID 名', 'パスワード')

while not wlan.isconnected():
    time.sleep(1)

print('接続完了:', wlan.ifconfig())
```

4) socket モジュール

socket モジュールは、ネットワーク通信の規格である BSD（Berkeley Software Distribution）ソケットインターフェースに基いている.

表 12.5. socket モジュールの主な機能とクラス

メソッド・定数	説明
socket()	ソケットオブジェクトの生成（例：socket(AF_INET, SOCK_STREAM)）
getaddrinfo（）	socket 通信で使う「名前解決（DNS）」のための関数
bind((host, port))	サーバ側で IP とポートを指定して待ち受け設定
listen()	接続待ち状態にする（TCP サーバ）
accept()	クライアントからの接続を受け入れる
connect((host, port))	クライアント側でサーバに接続
recv(bufsize)	データ受信（バイト単位、TCP 通信）
recvfrom(bufsize)	データ受信（バイト単位、UDP 通信）
send(data) / sendall()	データ送信（TCP 通信）
sendto(data,ip・port)	データ送信（UDP 通信）
close()	ソケットの切断
settimeout(seconds)	タイムアウト時間の設定

下記は、TCP サーバの例である.

```
import socket

s = socket.socket()
s.bind(('0.0.0.0', 80))
s.listen(1)

while True:
    conn, addr = s.accept()
    print('接続:', addr)
    request = conn.recv(1024)
    conn.send(b'HTTP/1.1 200 OK¥r¥nContent-Type: text/plain¥r¥n¥r¥nHello from Pico W!')
```

```
    conn.close()
```

5) math モジュール（数学関数）

MicroPython の math モジュールは、CPython の math に比べて軽量化されたサブセットであるが、三角関数、指数、対数など、マイコンでよく使う数学関数は揃っている.

主な関数は表 12.6 の通りである.

表 12.6. math モジュールの主な関数

種類	定数	説明
三角関数	math.sin(x)	サイン
	math.cos(x)	コサイン
	math.tan(x)	タンジェント
	math.asin(x)	逆サイン
	math.acos(x)	逆コサイン
	math.atan(x)	逆タンジェント
双曲線関数	math.sinh(x)	ハイパブリックサイン
	math.cosh(x)	ハイパブリックコサイン
	math.tanh(x)	ハイパブリックタンジェント
指数・対数	math.exp(x)	指数
	math.log(x)	自然対数
	math.log10(x)	常用対数
	math.sqrt(x)	平方根
丸め・符号	math.floor(x)	切り捨て
	math.ceil(x)	切り上げ
	math.trunc(x)	整数部分
	math.fabs(x)	絶対値
角度変換	math.radians(x)	度→ラジアン
	math.degrees(x)	ラジアン→度

定数には表 2.17 の 2 つがある.

表 12.7. math モジュールの定数

メソッド・定数	説明
math.pi	円周率 π
math.e	自然対数の底（ネイピア数）e

12.2. machine モジュールのクラス

最後に、ロボティック・システム制作実習で使う machine モジュール内の Pin クラス、PWM クラス、ADC クラス、I2C クラスの説明を記す.

1) Pin クラス

Pin クラスは、マイコンの GPIO（汎用入出力）ピンを制御するための基本クラスである. LED の点灯、スイッチの入力、センサとの接続など、物理的な入出力を扱う際に使われる.

Pin クラスの生成は、

```
Pin(pin_number, mode, pull=None, *, value=None, drive=None, alt=None)
```

であり、最小限の初期化は pin_number（ピン番号）と mode（モード）の 2 つだけである.

主な定数は表 12.8 の通りである.

表 12.8. Pin クラスの定数

定数	説明
Pin.IN	入力モード（スイッチやセンサ用）
Pin.OUT	出力モード（LED やモータ制御用）
Pin.OPEN_DRAIN	オープンドレイン出力（一部用途向け）
Pin.PULL_UP	プルアップ抵抗を有効にする
Pin.PULL_DOWN	プルダウン抵抗を有効にする

主なメソッド（関数）は下表の通りである.

表 12.9. Pin クラスのメソッド

メソッド	説明
value()	ピンの値を取得または設定（0/1）
on()	ピンを High（1）に設定
off()	ピンを Low（0）に設定
init()	ピンの再初期化（モードやプル設定を変更）
irq()	割り込みハンドラの設定（立ち上がり/立ち下がりなど）

下記は LED 点灯の例である.

```
from machine import Pin

led = Pin(25, Pin.OUT)    # Pico のオンボード LED
led.on()                  #  点灯
led.off()                 #  消灯
```

2) PWM クラス

PWM クラスは、GPIO ピンからパルス幅変調（PWM、Pulse Width Modulation）信号を出力するためのクラスである. PWM は、LED の明るさ調整、モータの速度制御、サーボの角度指定など、アナログ的な制御をデジタル信号で実現するために使われる.

PWM クラスの生成は、

```
PWM(Pin(pin_number))
```

であり、最小限必要なのは PWM を使いたい GPIO ピンの指定だけである.

主なメソッド（関数）は表 12.10 の通りである.

表 12.10. PWM クラスのメソッド

メソッド	説明
PWM(pin)	PWM オブジェクトを生成（ピンを指定）
freq(hz)	周波数を設定（例：1000Hz）
duty_u16(val)	デューティ比を 16 ビット値で設定（0〜65535）
duty_ns(ns)	デューティ比をナノ秒単位で設定（より精密な制御）
init(...)	周波数やデューティ比をまとめて再設定
deinit()	PWM 出力を停止し、ピンを通常の GPIO に戻す

下記は LED の明るさを変えるプログラムの例である.

```
from machine import Pin, PWM
import time

led = PWM(Pin(15))
led.freq(1000)

for duty in range(0, 65536, 4096):
    led.duty_u16(duty)
    time.sleep(0.1)
```

このコードでは、LED の明るさが徐々に変化する. デューティ比が高いほど、LED は明るくなる.

3) ADC クラス

ADC クラスは、アナログ信号をデジタル値に変換（A/D 変換）するためのクラスである. センサなどから得られる電圧値（アナログ）を、プログラムで扱える数値（デジタル）に変換する際に使用する. 光センサ、温度センサ、可変抵抗などの入力を扱う際に不可欠である.

ADC クラスの生成は、

ADC(pin_or_channel)

であり、GPIO ピン番号、または内部 ADC チャンネル番号を指定する.

主なメソッド（関数）は表 12.11 の通りである.

表 12.11. ADC クラスのメソッド

メソッド	説明
ADC(Pin)	指定ピンを ADC として初期化
read_u16()	16 ビットスケールでアナログ値を読み取る（0〜65535）

なお、GPIO26〜28 が ADC 対応である. また、実際の分解能は 12 ビット（0〜4095）で、内部的に 16 ビットにスケーリングされている. 更に、入力電圧は 最大 3.3V を超えないように注意する必要がある.

下記は入力電圧を表示するプログラム例である.

```
VREF = 3.3    # Pico の基準電圧
voltage = adc.read_u16() * VREF / 65535
print("電圧:", voltage, "V")
```

4) I2C クラス

I2C クラスは、I²C（Inter-Integrated Circuit）通信を行うためのクラスで、センサや LCD、RTC などの外部デバイスと 2 本の信号線（SDA と SCL）でデータをやりとりするために使われる. 複数のデバイスを 1 つのバスに接続できるため、IoT や組み込み開発でよく使用される.

I2C クラスの生成は、

```
I2C(id, scl=Pin(x), sda=Pin(y), freq=周波数)
```

であり、id には“0”（I2C0）または“1”（I2C0）を、scl、sda の Pin 番号 x、y には GPIO 番号を、freq には 40000（高速 400 kHz）が推奨されている.

主なメソッド（関数）は表 12.12 の通りである.

表 12.12. I2C クラスのメソッド

メソッド	説明
scan()	バス上のデバイスのアドレス一覧を取得（7 ビットアドレス）
readfrom(addr, nbytes)	指定アドレスのデバイスから nbytes バイト読み取る
writeto(addr, buf)	指定アドレスのデバイスにデータを書き込む
readfrom_mem(addr, memaddr, nbytes)	指定アドレスのデバイスのメモリから読み取り
writeto_mem(addr, memaddr, buf)	指定アドレスのデバイスのメモリに書き込み

下記は I2C デバイスの有無をスキャンするプログラム例である.

```
1.  from machine import Pin, I2C
2.  import time
3.
4.  # I2C インスタンスの作成（I2C0 を使用）
5.  i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=100000)
6.
7.  #  接続されている I2C デバイスのアドレスをスキャン
8.  devices = i2c.scan()
9.
10. if devices:
11.         print("I2C デバイスが見つかりました:")
12.         for device in devices:
13.                 print("アドレス: ", hex(device))
14. else:
15. print("I2C デバイスが見つかりませんでした")
```

ここに 4 行目の I2C クラスの初期化において、1 番目の引数は I2C バス番号（通常 0 または 1）、2 番目の引数はクロック線（SCL）に使うピンの指定、3 番目の引数はデータ線（SDA）に使うピンの指定、4 番目の引数は通信速度（Hz）である.