

Raspberry Pi Pico の使い方 (基本編)

(Pico2W 若干対応 2026 年 7 月版)

①Raspberry Pi Pico 使ってみよう(準備編)

必要なもの

- ・Raspberry Pi Pico2W (以下、Pico) ※PICOW でも可 (動作しない個所があります)
- ・USB ケーブル Micro-B (データ通信対応)
- ・ThonnyIDE <https://thonny.org/> ←からダウンロード 下の赤枠をダウンロード

Official downloads for Windows

Installer with Python 3.14 for Intel or AMD computers, requires Windows 10 or 11
[thonny-5.0.0-x64.exe \(24 MB\)](#)

Installer with Python 3.14 for Arm (e.g. Snapdragon) computers
[thonny-5.0.0-arm64.exe \(23 MB\)](#)

Portable variant with Python 3.14 for Intel or AMD computers, also usable on Arm
[thonny-py38-5.0.0-windows-portable-x64.zip \(36 MB\)](#)

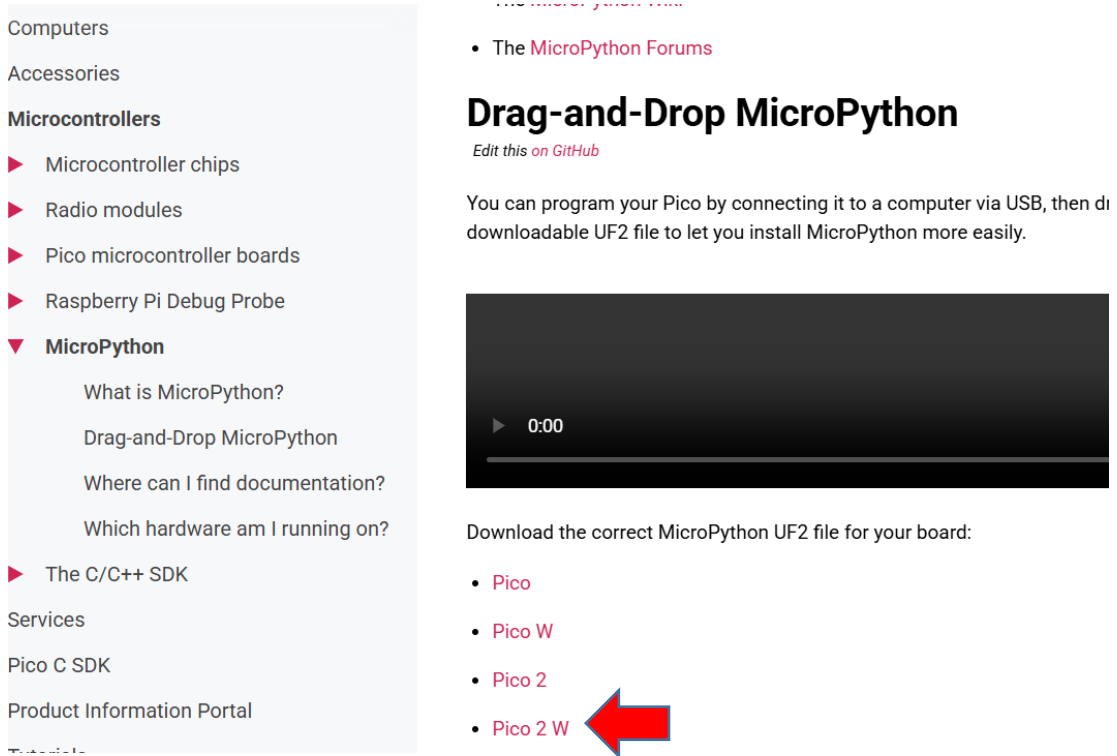
Re-using an existing Python installation (for advanced users)
`pip install thonny`

その都度、プログラム動作必要なものがありますので、それに合わせて購入してください。

(1) 事前に micropython を使用するための uf2 ファイルをダウンロードしておきます。

※もし PICO W を使う人は「PicoW」を選択。

<https://www.raspberrypi.com/documentation/microcontrollers/micropython.html>



Computers

Accessories

Microcontrollers

- ▶ Microcontroller chips
- ▶ Radio modules
- ▶ Pico microcontroller boards
- ▶ Raspberry Pi Debug Probe
- ▼ **MicroPython**
 - What is MicroPython?
 - Drag-and-Drop MicroPython
 - Where can I find documentation?
 - Which hardware am I running on?
- ▶ The C/C++ SDK

Services

Pico C SDK

Product Information Portal

• The **MicroPython Forums**

Drag-and-Drop MicroPython

Edit this on [GitHub](#)

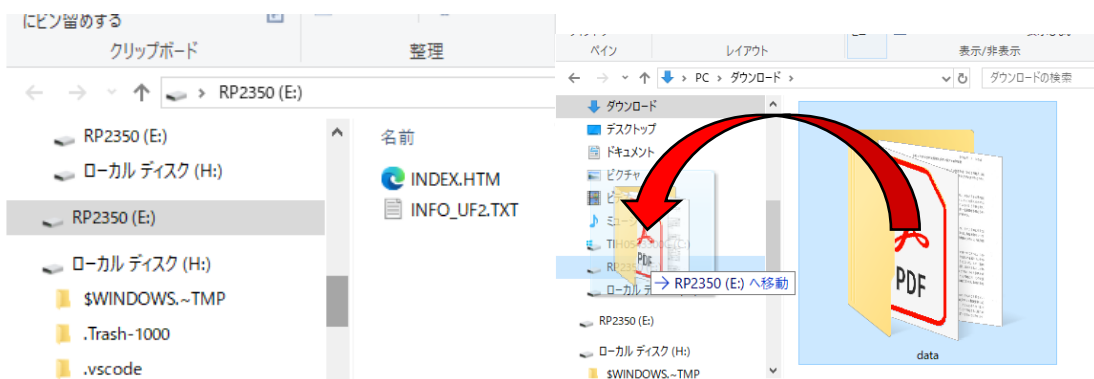
You can program your Pico by connecting it to a computer via USB, then download a downloadable UF2 file to let you install MicroPython more easily.

Download the correct MicroPython UF2 file for your board:

- **Pico**
- **Pico W**
- **Pico 2**
- **Pico 2 W**

(2) Pico の『BOOTSEL』を押しながら USB ケーブルとパソコンと Pico をつなぎます。

(3) リムーバブルディスクとして表示されるのでダウンロードした uf2 ファイルをドラッグして中に入れます。



にピン留めする

クリップボード

整理

RP2350 (E:)

名前

INDEX.HTM

INFO_UF2.TXT

ダウンロード

デスクトップ

ドキュメント

ピクチャ

ビデオ

ミュージック

RP2350 (E:)

ローカル ディスク (H:)

ローカル ディスク (H:)

\$WINDOWS.~TMP

.Trash-1000

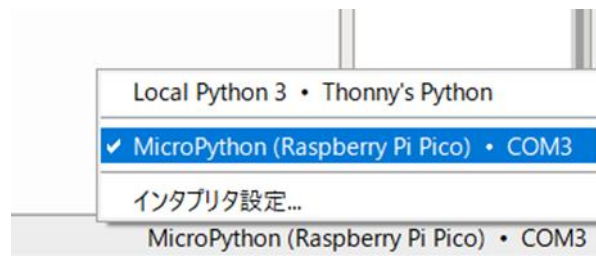
.vscode

データ

(4) 一度、リムーバブルディスクは消えますが、これで準備完了です。

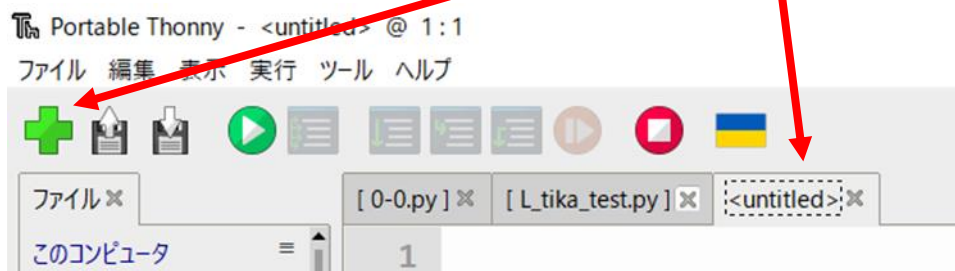
②Thonny IDE の使い方

- (1) パソコンに USB ケーブルを使って Pico を接続します。
- (2) Thonny IDE を立ち上げ右下のデバイスを「MicroPython」に変更します。パソコンによって COM○の番号が違いますが同じようなものを選んでください。



③プログラムを作成しよう

新しくプログラムを作るときは緑色のプラスマークを押すと新しいタブが出ます。



④練習問題1_内蔵 LED を点滅させよう。

最初から埋め込まれている LED を点滅させてみます。

```
from machine import Pin
import utime

t=0.5
L=25

for k in range(10):
    LNO = Pin("LED", Pin.OUT)
    #PicoW の場合、"LED"をL変更
    LNO.value(1)
    utime.sleep(t)
    LNO.value(0)
    utime.sleep(t)
```

プログラム入力後は保存して  を押してみましよう。

点滅したら、次のとおりに仕様を変更しましょう。

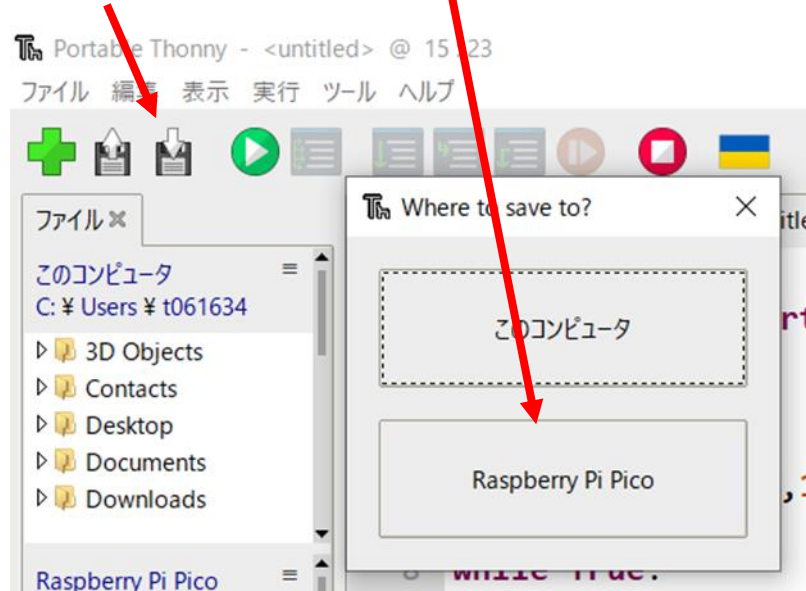
A点滅の速さを遅くしたり早くしたりしましょう。

B点滅する回数を 15 回に変更しましょう。

C点滅後は LED が点灯のままにしましょう。

⑤保存しよう

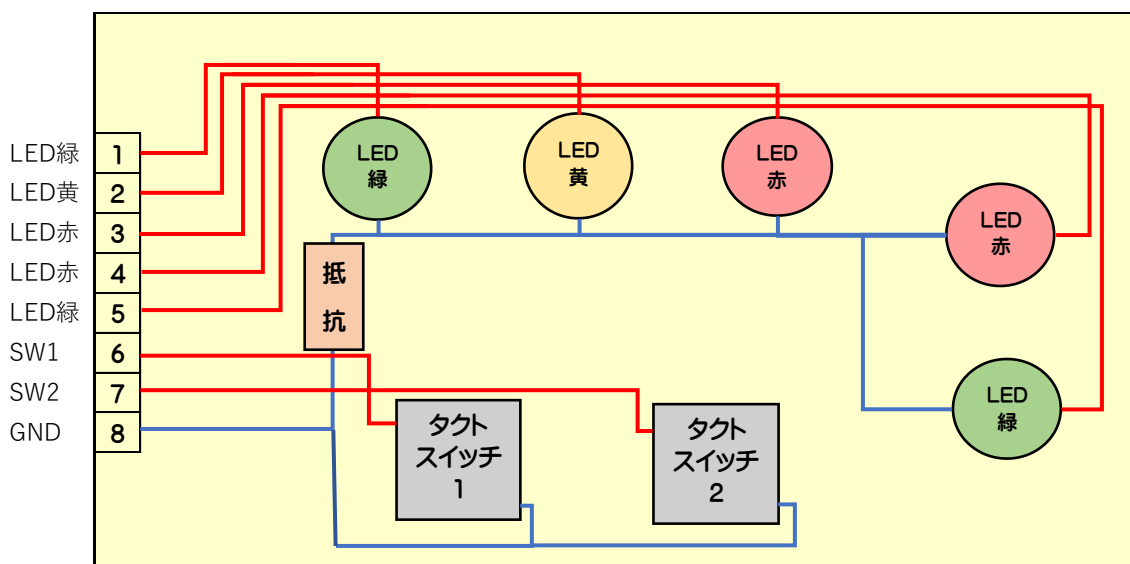
フロッピーマークを押した後、Pico 内部に保存する



保存の際は最後の「.py」
をつけるのを忘れずに！

⑥練習問題2_LED を制御させてみよう

機材があれば以下の内容でユニバーサル基板にはんだ付けして LED を制御する基盤を作ってみましょう。



ラズパイ pico2W はプルダウン接続に大きなバグが発生するので、プルアップ接続で使用する。

PICO2W GPIOの配列

UART0_TX	I2C0_SDA	SPI0_RX	GP0	1		40	VBUS	
UART0_RX	I2C0_SCL	SPI0_CS _n	GP1	2		39	VSYS	
			GND	3		38	GND	
	I2C1_SDA	SPI0_SCK	GP2	4		37	3V3_EN	
	I2C1_SCL	SPI0_TX	GP3	5		36	3V3(OUT)	
UART1_TX	I2C0_SDA	SPI0_RX	GP4	6		35		ADC_VREF
UART1_RX	I2C0_SCL	SPI0_CS _n	GP5	7		34	GP28	ADC2
			GND	8		33	GND	AGND
	I2C1_SDA	SPI0_SCK	GP6	9		32	GP27	ADC1
	I2C1_SCL	SPI0_TX	GP7	10		31	GP26	ADC0
UART1_TX	I2C0_SDA	SPI1_RX	GP8	11		30	RUN	
UART1_RX	I2C0_SCL	SPI1_CS _n	GP9	12		29	GP22	
			GND	13		28	GND	
	I2C1_SDA	SPI1_SCK	GP10	14		27	GP21	
	I2C1_SCL	SPI1_TX	GP11	15		26	GP20	
UART0_TX	I2C0_SDA	SPI1_RX	GP12	16		25	GP19	SPI0_TX
UART0_RX	I2C0_SCL	SPI1_CS _n	GP13	17		24	GP18	SPI0_SCK
			GND	18		23	GND	I2C1_SCL
	I2C1_SDA	SPI1_SCK	GP14	19		22	GP17	SPI0_CS _n
	I2C1_SCL	SPI1_TX	GP15	20		21	GP16	I2C0_SCL
								UART0_RX
								UART0_TX

自作したボードを1~9につなぎましょう。LED:GP0~GP4 sw:GP5~GP6

先ほどのプログラムを変更してLED が点滅するかチェックしてみましょう。

どの番号がどのLEDにつながっていたか理解できましたか？

⑦LED を連続で点滅させてみよう

LED が点灯することが理解出来たら、自分で考えて LED を1つずつ点灯させるプログラムを作ってみよう。

⑧リストを覚えよう

リストを覚えるとプログラムが長くない他に、条件が違くと動作が変化するプログラムにも対応できるので覚えよう。

```

2 from machine import Pin
3 import utime
4
5 t=0.5
6 L=[0,1,2,3,4]
7 ※この形が[リスト]
8
9 for k in range(5):
10     LNO = Pin(L[k], Pin.OUT)

```

```

from machine import Pin
import utime

t=0.5
L=[0,1,2,3,4]

for k in range(5):
    LNO = Pin(L[k], Pin.OUT)

    LNO.value(1)
    utime.sleep(t)
    LNO.value(0)
    utime.sleep(t)

```

⑨スイッチで制御しよう

```
sw1=Pin(sw[0],Pin.IN,Pin.PULL_UP)
sw2=Pin(sw[1],Pin.IN,Pin.PULL_UP)
                                ※スイッチの設定
while True:

    if sw1.value()==0: ※スイッチ1が押されたとき
        LNO = Pin(L[0], Pin.OUT)
        LNO.value(1)
        print("sw1")
```

このプログラムができたなら次の仕様に変更しよう

A sw1を押したら右から1つずつLEDが点灯する

B sw2を押したら左から1つずつLEDが点灯する

※間隔は任意で構いません。

```
from machine import Pin
import utime

L=[0,1,2,3,4]
sw=[5,6]
sw1=Pin(sw[0],Pin.IN,Pin.PULL_UP)
sw2=Pin(sw[1],Pin.IN,Pin.PULL_UP)

while True:

    if sw1.value()==0:
        LNO = Pin(L[0], Pin.OUT)
        LNO.value(1)
        print("sw1")
    elif sw2.value()==0:
        LNO = Pin(L[1], Pin.OUT)
        LNO.value(1)
        print("sw2")
    else:
        for x in range(5):
            LNO = Pin(L[x], Pin.OUT)
            LNO.value(0)
            print("OFF")

        utime.sleep_ms(20)
```

※PICO2Wはプルダウンのときに動作にバグが発生するためプルアップで制御する。

課題:if sw1.value()==0: なぜ「0」のときなのか調べてみよう。

⑩PWM 制御を理解しよう

```
1 from machine import Pin,PWM
2 import utime
3
4 L=[0,1,2,3,4]
5 sw=[5,6]
6 t=0.5
7
8 n=0
9 LEP=PWM(Pin(L[n]))
10 LEP.freq(1000)
11
12 pwd=65535/10
13
14 for x in range(10):
15     for ud in range(10):
```

※PWMの設定

```
from machine import Pin,PWM
import utime

L=[0,1,2,3,4]
sw=[5,6]
t=0.5

n=0
LEP=PWM(Pin(L[n]))
LEP.freq(1000)

pwd=65535/10

for x in range(10):
    for ud in range(10):
        LEP.duty_u16(int(pwd*ud))
        utime.sleep(0.05)
    for dd in reversed(range(10)):
        LEP.duty_u16(int(pwd*dd))
        utime.sleep(0.05)
```

PWM制御の基本（デューティ比で電力を調整する）

PWM（Pulse Width Modulation）とは、半導体を使って電力を制御する方式の1つです。
 オン（ON）とオフ（OFF）を高速で繰り返し、ONの時間の長さ（パルス幅）を変えることで、出力される電力（平均電圧や明るさ、モータの回転速度など）を調整します。

ポイント

- 周期（T）は一定に保ち、ON時間（パルス幅 Ton）を変える
- デューティ比が大きいほど、出力は強くなる（明るくなる、速くなる）
- PicoのPWMは16bit（0～65535）の分解能を持つ

① PWMの波形イメージ（周期は一定 20ms:50Hz）

デューティ比：10%（弱い・暗い）

ON時間：2ms OFF時間：18ms

デューティ比：50%（中くらい・普通）

ON時間：10ms OFF時間：10ms

デューティ比：80%（強い・明るい）

ON時間：16ms OFF時間：4ms

デューティ比	パルス幅 Ton (周期20msのとき)	イメージ (例：LEDの明るさ)
10%	約2ms	
50%	約10ms	
80%	約16ms	

② Pico（MicroPython）でのPWM設定：duty_u16の計算

PicoのPWMは16bit（0～65535）の分解能を持ちます。デューティ比（%）を0～65535の値に変換して設定します。

計算式

$$\text{duty_u16} = \frac{\text{デューティ比}(\%) \times 65535}{100}$$

または、パルス幅（ms）から直接計算することもできます。

$$\text{duty_u16} = \frac{\text{パルス幅}(\text{ms}) \times 65535}{\text{周期}(\text{ms}) (= 20\text{ms})}$$

デューティ比	パルス幅 Ton (ms)	OFF時間 Toff (ms)	duty_u16 の値 (約)
10%	2ms	18ms	6553
50%	10ms	10ms	32768
80%	16ms	4ms	52428
100%	20ms	0ms	65535
0%	0ms	20ms	0

※値は小数点以下を四捨五入しています。周速により微調整してください。

③ MicroPythonの例（PicoでLEDを明るさ調整）

```
from machine import Pin, PWM
import utime

led = PWM(Pin(0)) # GPIOを使用
led.freq(1000) # 周波数を1kHzに設定 (例)

# 10% → 50% → 80% を繰り返す
while True:
    led.duty_u16(6553) # 10% (約2ms)
    utime.sleep(1)
    led.duty_u16(32768) # 50% (約10ms)
    utime.sleep(1)
    led.duty_u16(52428) # 80% (約16ms)
    utime.sleep(1)
```

計算例（デューティ比 50% の場合）

デューティ比 = 50%

$$\text{duty_u16} = 50 \times 65535 \div 100 = 32767.5 \approx 32768$$

または、パルス幅から計算

$$\text{duty_u16} = 10 \times 65535 \div 20 = 32767.5 \approx 32768$$

まとめ

- PWMはONとOFFの繰り返しで電力を制御する方式です。
- デューティ比（ON時間の割合）を変えることで、出力の強さ（明るさや速度など）を調整できます。
- PicoのPWMは16bit（0～65535）の高分解能で設定します。

よく使う周波数の例

- LED：500Hz～2kHz程度
- モータ／ローボ：50Hz（サーボ）、数百Hz～数千Hz（モータ）

このプログラムができたなら次の仕様変更をしよう
 タクトスイッチ1を押すと 0.5 秒間隔で点滅し、タクトスイッチ2を押すと PWM で点滅するプログラム

- A sw1 を押したら緑 LED が点滅する
 B sw2を押したら黄LED 制御が PWM 制御で点滅する
 C sw1 を押したら PWM 制御で点滅を早くする
 D sw2 を押したら PWM 制御で点滅を遅くする
 E デューティー比を 100%から徐々に下げて消灯させる
 F sw1 を押すと右から順に LED を PWM 制御で点灯させる、sw2 を押すと左から順に LED を PWM 制御で点灯させる。

※点滅や 1 つずつ点灯させる時間の間隔は任意。

```
from machine import Pin,PWM
import utime

L=[0,1,2,3,4]
sw=[5,6]
t=0.5

n=0
LEP=PWM(Pin(L[n]))
LEP.freq(1000)

pwd=65535/10

for x in range(10):
    for ud in range(10):
        LEP.duty_u16(int(pwd*ud))
        utime.sleep(0.05)
    for dd in reversed(range(10)):
        LEP.duty_u16(int(pwd*dd))
        utime.sleep(0.05)
```

⑪サーボモータを動かしてみよう

サーボモータのPWM制御 (一般的な180度サーボの例)

サーボモータは、1周期 (約20ms) の中で「ONになっている時間 (パルス幅)」によってサーボホーンの角度が決まります。周波数は一般的に 50Hz (周期20ms) です。

0° (左端)

パルス幅: **1.0ms**

デューティ比: 約5% (1.0ms ÷ 20ms)

90° (中央)

パルス幅: **1.5ms**

デューティ比: 約7.5% (1.5ms ÷ 20ms)

180° (右端)

パルス幅: **2.0ms**

デューティ比: 約10% (2.5ms ÷ 20ms)

ポイント

- ・周期は約20ms (50Hz)
- ・パルス幅が短いほど角度が小さい
- ・パルス幅が長いほど角度が大きい (機種により多少の違いがあります)
- ・一般的な範囲: 1.0ms ~ 2.0ms

参考: デューティ比と明るさのイメージ

パルス幅	デューティ比	イメージ
1.0ms	約5%	
1.5ms	約7.5%	
2.0ms	約10%	

Raspberry Pi Pico (MicroPython) のPWM設定方法 (duty_u16の計算)

PicoのPWMは16bit分解能 (0 ~ 65535) です。デューティ比(%)を 0 ~ 65535 の値に変換して duty_u16() に設定します。

$$\text{duty_u16} = \frac{\text{デューティ比}(\%) \times 65535}{100}$$

または、パルス幅 (ms) から直接計算することもできます。

$$\text{duty_u16} = \frac{\text{パルス幅}(\text{ms}) \times 65535}{\text{周期}(\text{ms}) (= 20\text{ms})}$$

サーボ角度と duty_u16 の値 (周期=20ms)

角度	パルス幅 (ms)	デューティ比 (%)	duty_u16 の値 (約)
0° (左端)	1.0	5.0	3277
90° (中央)	1.5	7.5	4915
180° (右端)	2.0	10.0	6554

※機種や個体差により、最適な値は多少異なる場合があります。微調整しながら使用してください。

設定例 (MicroPython)

```
from machine import Pin, PWM
import utime

servo = PWM(Pin(15)) # 例: GP15を使用
servo.freq(50) # 周波数 50Hz (周期20ms)

# 角度ごとの duty_u16 値 (上の表の参考値)
ANG_0 = 3277 # 0° (約1.0ms)
ANG_90 = 4915 # 90° (約1.5ms)
ANG_180 = 6554 # 180° (約2.0ms)

while True:
    servo.duty_u16(ANG_0) # 0°
    utime.sleep(1)
    servo.duty_u16(ANG_90) # 90°
    utime.sleep(1)
    servo.duty_u16(ANG_180) # 180°
    utime.sleep(1)
```

計算例 (1.5msの場合)

$$\text{デューティ比}(\%) = 1.5 \div 20 \times 100 = 7.5\%$$

$$\text{duty_u16} = 7.5 \times 65535 \div 100 \approx 4915$$

または、直接計算

$$1.5 \times 65535 \div 20 \approx 4915$$

ポイント

- ・サーボは ON 時間 (パルス幅) で角度を制御
- ・PicoのPWMは 16bit (0 ~ 65535)
- ・周波数は 50Hz (周期20ms) に設定する

次のとおりプログラムを変更してみよう。

デューティ比を角度で設定する。[0,90,180]

※計算式を考えてみよう。

プログラムを変えて次の動作をさせてみよう。

A サーボをボタンでコントロールしよう。

ボタンを押さなければサーボホーンを真ん中に (90°)

タクトスイッチ1を押すとサーボホーンを左 (0°)

タクトスイッチ2を押すとサーボホーンを右 (180°)

B サーボホーンを左右に動かすとき、PWM 制御を使ってゆっくり動かしてみよう。

※【もしできたら】もし、ボタンを押し続けた場合、サーボホーンを右あるいは左に固定する

C 戻るときもゆっくり戻してみよう。

```
from machine import Pin, PWM
import utime
```

```
servo1 = PWM(Pin(15))
servo1.freq(50)
```

```
du = [1638, 4751, 7864]
# デューティ比に 65535 をかけた値
for x in range(3):
    servo1.duty_u16(int(du[x]))
    utime.sleep(2)
```

⑫距離センサを使ってみよう(AD 変換)

シャープ距離センサ (GP2Y0A21YK0F) を使ってみよう (アナログ入力)

① 距離センサのしくみ

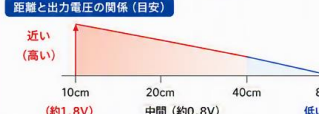


GP2Y0A21YK0Fとは

- 赤外線を利用して距離を測定するアナログセンサ
- 測定範囲: 約10cm~80cm
- 距離に応じて出力電圧 (Vo) が変化
- Pico の ADC 機能で電圧を読み取る

ケーブル色	接続先
赤 (Vcc)	→ VBUS (5V)
黒 (GND)	→ GND
黄 (Vo)	→ ADC入力 (GP26)

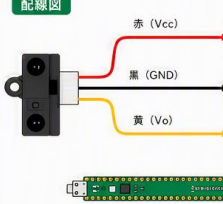
距離と出力電圧の関係 (目安)



② Pico との接続

センサ (ケーブル色)	Pico 側 (ピン番号)
Vcc (赤)	VBUS (40番・5V)
GND (黒)	GND (33番または38番)
Vo (黄)	GP26 (31番: ADC0)

配線図



③ Pico の ADC 端子

ピン番号	Pico 側	ADC
40	VBUS (5V入力)	—
39	VSYS	—
38	GND	—
37	3V3_EN	—
36	3V3(OUT)	—
35	ADC_VREF	基準電圧
34	GP28	ADC2
33	GND (AGND)	—
32	GP27	ADC1
31	GP26	ADC0 ← 今回使用
30	RUN	—

※ ADC は 16bit (0~65535) の分解能を持つ

④ テストプログラム (電圧と距離を表示)

```
from machine import ADC
import utime

sensor = ADC(0) # ADC0 (GP26) を使用

while True:
    voltRaw = sensor.read_u16() # 0~65535の値を取得
    volt = voltRaw * 3.3 / 65535 # 電圧に変換 (V)
    dis = 27.149 * volt ** -1.185 # 距離 (cm) に変換

    print("電圧 = {:.2f} V".format(volt))
    print("距離 = {:.1f} cm".format(dis))
    utime.sleep(0.1)
```

プログラムの説明

- sensor.read_u16() で ADC 値 (0~65535) を取得する。
- ADC 値を実際の電圧 (0~3.3V) に変換する。
- 電圧から距離 (cm) を計算する。
計算式: $dis = 27.149 \times volt^{-1.185}$
- 0.1 秒ごとに表示を更新する。

実行結果の例

電圧	距離
1.35 V	21.5 cm
0.82 V	41.2 cm
0.45 V	74.1 cm

※ センサの特性や周囲環境により値は変化します。実際に確認しましょう。

⑤ 距離の目安 (参考値)

距離 (cm)	電圧 (V) (おおよそ)	ADC 値 (目安)
10	約1.8	約35700
20	約1.3	約25800
30	約1.0	約19800
40	約0.8	約15900
50	約0.6	約11900
60	約0.5	約9900
70	約0.45	約8900
80	約0.4	約7900

※ 値は参考です。センサの個体差や測定条件により変わります。

⑥ いろいろな応用例

- LED で距離を表示する
例) 20cm未満で点灯
- ブザーで知らせる
例) 10cm未満でブザーON
- 自動走行車の障害物検知
例) 20cm未満で停止
- サーボを使った物体追跡
距離に応じて角度を調整

まとめ

- ✓ GP2Y0A21YK0F はアナログ距離センサ
- ✓ VBUS (5V) でセンサを動作させる
- ✓ GP26 (ADC0) で電圧を読み取る
- ✓ ADC 値 (0~65535) を電圧に変換する
- ✓ 電圧を距離 (cm) に変換する
- ✓ 物体が近いほど出力電圧が高くなる
- ✓ 自動走行車や障害物検知に活用できる

注意

- ▲ 測定範囲は約 10cm ~ 80cm です。
- ▲ 黒線 (GND) は Pico の GND と必ず共通にしてください。
- ▲ 鏡面走光物体は誤差が大きくなる場合があります。
- ▲ 直射日光や強い光の下では正確に測れない場合があります。

```
from machine import ADC
import utime

sensor = ADC(0)

while True:
    adc_raw = sensor.read_u16()
    voltage = adc_raw * 3.3 / 65535
    print("ADC =", adc_raw)
    print("V =", round(voltage, 2))

    utime.sleep(0.5)
```

A プログラムを変えて次の動作をさせてみよう。

※センサに誤差があるので実際に距離がどのくらいの adc_raw かを調べてみよう

30cm 以上なら緑の LED が点灯する

20cm 以上なら黄色の LED が点灯する。

10cm 以下なら赤色の LED が点灯する。

⑬ i2c 通信でセンサーを使ってみよう

i2c 通信を使ってみよう (DHT20を使った温度・湿度の取得)

① i2c とは？

i2c (アイ・スクエア・シー)

正式名称
Inter-Integrated Circuit
(インター・インテグレートド・サーキット)

SDA (データ線) と SCL (クロック線) の
2本の信号線で、複数の機器と通信できる
方式です。電源 (VCC) と GND も必要です。

信号線の役割	
信号	役割
VCC	電源 (センサーの動作電圧)
GND	基準電位 (グラウンド)
SDA	データ通信線
SCL	クロック信号線

③ i2c0 と i2c1

Pico2Wには i2c の回路が 2 系統あります。使う番号を選んで使います。

i2c0 (SDA0 / SCL0)	i2c1 (SDA1 / SCL1)
GP0 SDA0	GP2 SDA1
GP1 SCL0	GP3 SCL1
GP4 SDA0	GP6 SDA1
GP5 SCL0	GP7 SCL1
GP8 SDA0	GP10 SDA1
GP9 SCL0	GP11 SCL1

今回使用するピン

プログラムでは
i2c0 を使います。

i2c = I2C(0)

GP8 → SDA0 (データ線)

GP9 → SCL0 (クロック線)

なぜ GP8・GP9 を使うの？

GP0~GP6 は LED やスイッチに
使用しているため、空いている
GP8・GP9 を使います。

② DHT20 (Grove) と Pico2W の配線

DHT20 (Grove) のコネクタ



配線表	
DHT20 (Grove)	Pico2W
VCC (赤)	VBUS (40番・5V)
GND (黒)	GND (33番または38番)
SDA (白)	GP8 (SDA0)
SCL (黄)	GP9 (SCL0)

配線図	
VCC (赤)	VBUS (40番・5V)
GND (黒)	GND (33番または38番)
SDA (白)	GP8 (SDA0)
SCL (黄)	GP9 (SCL0)

④ i2c アドレスとは？

i2c 機器には“住所 (アドレス)”があります。
これにより、同じ配線に複数の機器を接続できます。

機器の例	アドレス
DHT20 (温湿度センサー)	0x38 (56)
SSD1306 (OLED)	0x3C (60)
DS1307 (RTC)	0x68 (104)
...	...

アドレスの表し方

i2c.scan() の結果は
10進数で表示されます。

例 56

↑↑

0x38

どちらも同じ値です。

⑤ アドレス確認方法 (スキャン)

接続されている i2c 機器のアドレス調べることができます。

```
from machine import Pin, I2C

i2c = I2C(
    0,
    scl=Pin(9),
    sda=Pin(8),
    freq=1000000
)

print(i2c.scan())
```

実行結果の例

接続: DHT20 だけの場合

[56]

接続: DHT20 と OLED の場合

[56, 60]

56 = DHT20 (0x38)

60 = SSD1306 (0x3C)

⑥ DHT20 の温湿度を取得する (ライブラリなし)

DHT20 (AHT20互換) は i2c 通信で温度と湿度を読み取ります。

```
from machine import Pin, I2C
import utime

i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=1000000)

while True:
    # 0x38
    i2c.writeto(i2c.DEVICE_ADDRESS, bytes([0xAC, 0x33, 0x00]))
    utime.sleep(0.1)
    # 7bit 12.5℃
    data = i2c.readfrom(i2c.DEVICE_ADDRESS, 7)
    # 湿度
    temp_raw = (data[1]<<12) | (data[2]<<4) | (data[3]>>4)
    humidity = (data[4]<<16) | (data[5]<<8) | data[6]
    humidity = humidity * 100 / 1048576
    temperature = temp_raw * 200 / 1048576 - 50
    print("Temp: {:.1f} °C".format(temperature))
    print("Hum: {:.1f} %".format(humidity))
```

実行結果の例

Temp : 25.3 °C

Humi : 52.1 %

ポイント

・まずはアドレスをスキャンで確認

・i2c.scan() の結果は 10進数で表示されます。

・DHT20 のアドレスは 0x38 (56)

・複数の機器を同じ配線に接続できる

注意 VCC は VBUS (5V) を使います。
必ず GND を共通にしてください。

今回は DHT20 というセンサーを i2c 通信で接続し、気温を取得します。

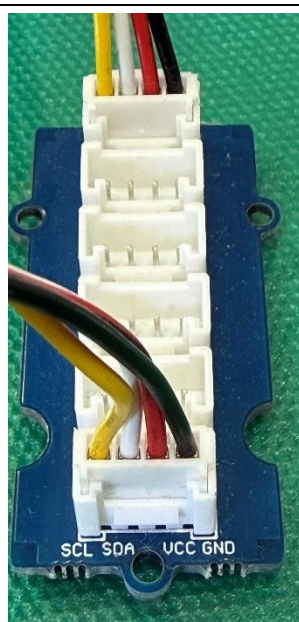
LED やスイッチの配線があるので次に使う GP8 と GP9 に接続します。

GPIO ピンは、i2c 通信の SDA や SCL としても利用できます。

i2c の赤い Vcc は 5V、GND はラズパイに GND に接続してください。



Grove DHT20



Grove I2C ハブ

		GND	8
	I2C1_SDA	GP6	9
	I2C1_SCL	GP7	10
TX	I2C0_SDA	GP8	11
PX	I2C0_SCL	GP9	12
	GND		13

VCC (赤) は 5V

GND (黒) は GND

SDA (白) は GP8 (SDA)

SCL (黄) は GP9 (SCL)

⑭ SSD1306(OLED)を使ってみよう (1/2)

① SSD1306とは？

- 有機EL(OLED)ディスプレイ
- 128×64ドット表示
- I²C通信で制御する
- 文字や図形を表示できる
- アドレスは通常 0x3C (60)

② 配線してみよう

● Grove OLED (SSD1306) と Pico2W の配線

SSD1306側	Pico2W側	説明
VCC (赤)	VBUS (40番 - 5V)	電源
GND (黒)	GND (38番または33番)	グランド
SDA (白)	GP8 (SDA0)	データ線
SCL (黄)	GP9 (SCL0)	クロック線

③ I²C0 と I²C1

Pico2W には I²C通信の回路が2つあります。0番か1番を選んで使います。

【I ² C0 のピン (SDA0 / SCL0)】			
GP0	SDA0	GP1	SCL0
GP4	SDA0	GP5	SCL0
GP8	SDA0	GP9	SCL0

【I ² C1 のピン (SDA1 / SCL1)】			
GP2	SDA1	GP3	SCL1
GP6	SDA1	GP7	SCL1
GP10	SDA1	GP11	SCL1

i2c = I2C(0) を使うとき
上の I²C0 のピンの組み合わせを使用します。

i2c = I2C(1) を使うとき
上の I²C1 のピンの組み合わせを使用します。

今回使用するピン
i2c = I2C(0)
GP8 → SDA0 (データ線)
GP9 → SCL0 (クロック線)

なぜ GP8・GP9 を使う？
GP0～GP6 は LED やスイッチに使用しているため、空いている GP8 と GP9 を使います。

④ I²Cアドレスとは？

I²C機器はそれぞれ住所 (アドレス) を持っています。同じ配線になってもアドレスで区別します。

機器名	アドレス(16進数)	アドレス(10進数)
DHT20 (温湿度センサ)	0x38	56
SSD1306 (OLED)	0x3C	60
DS1307 (RTC)	0x68	104

⑭ SSD1306(OLED)を使ってみよう (2/2)

⑤ I²Cアドレスを確認してみよう

Pico2W から接続されている I²C機器のアドレスを調べることができます。

```
from machine import Pin, I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8))
print(i2c.scan())
```

実行結果の例

[56, 60]

56 (10進数) = 0x38 → DHT20
60 (10進数) = 0x3C → SSD1306

ポイント
i2c.scan() の結果は 10進数で表示されます。0x38 や 0x3C のように 16進数で覚えましょう。

⑥ ライブラリを追加しよう

OLED を簡単に使うためにライブラリを使います。

- Pico2W の中に「lib」フォルダを作成
- lib フォルダの中に「ssd1306.py」ファイルを保存

ポイント
ライブラリを使うと、複雑な制御を簡単に行うことができます。

⑦ Hello を表示してみよう

まずは OLED に「Hello」と表示してみます。

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)
oled.fill(0)
oled.text("Hello", 30, 20) # 画面を消す
oled.show() # 文字を描く
oled.show() # 画面に反映
```

表示例

⑧ DHT20の温度・湿度を表示しよう

DHT20 で取得した、温度・湿度を表示します。

```
from machine import Pin, I2C
from dht20 import DHT20
from ssd1306 import SSD1306_I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8))
dht20 = DHT20(i2c)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3C)

while True:
    t = dht20.temperature # 温度 (°C)
    h = dht20.humidity # 湿度 (%)
    oled.fill(0)
    oled.text("Temp: {:.1f} °C".format(t), 0, 10)
    oled.text("Humi: {:.1f} %".format(h), 0, 30)
    time.sleep(1)
```

表示例

Temp: 25.3 °C
Humi: 52.1 %

ポイント
DHT20 と SSD1306 は同じ I²C のピン (2本) につなぐことができます。アドレスで機器を区別しています。

⑨ I²Cのメリット (2本の配線で複数の機器を接続できる)

まとめ

- SSD1306 は I²C 接続の OLED ディスプレイ
- アドレスは通常 0x3C (60)
- Pico2W の GP8 (SDA0)、GP9 (SCL0) を使用
- i2c.scan() でアドレスを確認できる
- DHT20 と同じ配線に接続できる

まずは接続が間違えていないか I2C に必要なアドレスを確認してみよう。

i2c = I2C(0, scl=Pin(9), sda=Pin(8))

I2C1_SDA
I2C1_SCL
I2C0_SDA
I2C0_SCL

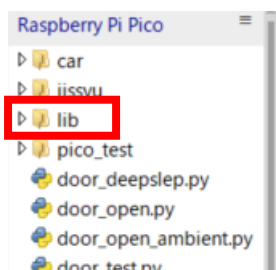
I2C のポートは 2 つあるので

I2C"0"の場合はプログラムにも"0"となります。I2C1 の場合は"1"

チェックした答えは[56]と出ますが、この値は 10 進数ですので、16 進数に直します。プログラムで使うのは 16 進数ですので[0x38]となります。

今後はスムーズにプログラムを作るためにライブラリというある機種を動かすためのプログラムを使って動作させます。

PICO2W の中に「lib」というフォルダを作ります。その中にdht20.py というプログラムを保存します。



```
from machine import Pin, I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8))
print(i2c.scan())
```

アドレスがわかったので次は温度が取得できるか右のプログラムを打ち込んで動作確認をしてみよう。

```
from machine import Pin, I2C
import time # タイムスリープを使うために追加
from dht20 import DHT20

i2c = I2C(0, scl=Pin(9), sda=Pin(8))
dht20 = DHT20(i2c)

while True:
    t = dht20.temperature
    h = dht20.humidity

    print("tem:", t)
    print("Hum:", h)
    print("-----")

    time.sleep(1) # 1 秒待って繰り返す
```

⑭ i2c 通信でディスプレイを使ってみよう

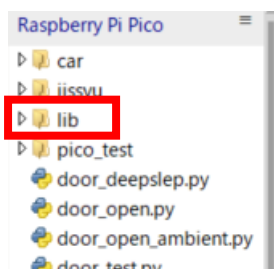
温度センサと同様にディスプレイも配線します。

配線後はもう一度 i2c チェックのプログラムを動作させると今度は [56, 60] と2つ出ます。

[60] は 10 進数ですので 16 進数になるので [0x3C] がディスプレイのアドレスです。

今後はスムーズにプログラムを作るためにライブラリというある機種を動かすためのプログラムを使って動作させます。

PICO2W の中に「lib」というフォルダを作ります。その中に ssd1306.py というプログラムを保存します。



右のプログラムを入力させ、実行すると”Hello”と出力することを確認します。

```
from machine import I2C, Pin
import time
from ssd1306 import SSD1306_I2C

# 接続設定 (ピン番号は環境に合わせてください)
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)

WIDTH = 128
HEIGHT = 64

oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)

oled.fill(0)
oled.text("Hello", 0, 0, 1)
oled.show()
```

A DHT20 で取得した気温や湿度を表示させてみましょう。

```
from machine import Pin, I2C
import time
from dht20 import DHT20
from ssd1306 import SSD1306_I2C

# 1. I2C の共通設定 (周波数は 400kHz でどちらも正常に動きます)
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)

# 2. ディスプレイとセンサーの初期化
WIDTH = 128
HEIGHT = 64
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
dht20 = DHT20(i2c)

print("温湿度表示プログラムを開始します。")

while True:
    t = dht20.temperature
    h = dht20.humidity

    oled.fill(0)

    str_t = "Temp: " + str(t) + " C"
    str_h = "Humi: " + str(h) + " %"

    oled.text(str_t, 0, 10, 1) # 上側に気温を表示
    oled.text(str_h, 0, 30, 1) # 下側に湿度を表示

    oled.show()

    print("T:", t, " H:", h)

    time.sleep(1)
```