

リスト 1 Pico に MicroPython をインストールする方法

Raspberry Pi Pico に MicroPython をインストールする方法を教えてください。
<p>Raspberry Pi Pico は、Raspberry Pi Foundation が開発した低コストのマイクロコントローラーボードです。Pico は RP2040 という独自のマイクロコントローラーを搭載しており、MicroPython をサポートしています。</p> <p>Raspberry Pi Pico に MicroPython をインストールする手順は以下の通りです：</p> <ol style="list-style-type: none">1. ファームウェアのダウンロード：<ul style="list-style-type: none">・最初に、公式の Raspberry Pi Pico のページ(またはその他の信頼できるリソース)から MicroPython のバイナリファイル(.uf2 形式)をダウンロードします。・ 公 式 の MicroPython Pico ペ ー ジ (https://www.raspberrypi.org/documentation/rp2040/getting-started/)で最新のファームウェアを見つけることができます。2. Pico をブートモードで接続：<ul style="list-style-type: none">・Raspberry Pi Pico を PC に接続する前に、`BOOTSEL`ボタンを押したまま、Pico を USB ケーブルで PC に接続します。・正しく接続されると、Pico は USB マスストレージデバイスとしてマウントされ、ファイルをコピーすることができるようになります。3. ファームウェアのコピー：<ul style="list-style-type: none">・ダウンロードした.uf2 ファイルを、Pico のマスストレージデバイスにドラッグ&ドロップします。・ファイルが転送されると、Pico は自動的にリセットされ、MicroPython が起動します。4. MicroPython の確認：<ul style="list-style-type: none">・Pico をシリアル接続ツール(例:PuTTY、screen、minicom など)で開き、115200 ボーの通信速度で接続します。・成功すれば、MicroPython のプロンプト`>>>`が表示されます。これで MicroPython コードを直接入力したり、スクリプトを実行したりすることができます。 <p>以上で、Raspberry Pi Pico に MicroPython をインストールする手順は完了です。必要な外部ライブラリやモジュールがある場合は、`upip`ツールや直接ファイルをコピーすることで追加できます。</p>

リスト 2 ESP32 に MicroPython をインストールする方法

ESP32-DevKitC-に MicroPython をインストールする方法を教えてください。

ESP32-DevKitC は、Espressif Systems が開発した人気のあるマイクロコントローラーボードで、Wi-Fi や Bluetooth などの通信機能を持っています。MicroPython は ESP32 に対応しており、以下の手順でインストールできます。

1. 必要なツールのインストール:

- esptool という Python ツールを使って、ESP32 にファームウェアを書き込むことができます。このツールは pip を使ってインストールできます。

```
pip install esptool
```

2. MicroPython ファームウェアのダウンロード:

- MicroPython の公式ダウンロードページ(<https://micropython.org/download/esp32/>)から、ESP32 向けの最新のファームウェアをダウンロードします。

3. ESP32 を書き込みモードにする:

- ESP32-DevKitC ボードには、モードボタン(通常は「BOOT」ボタンと表示されている)とリセットボタンがあります。
- 書き込みモードにするために、まず「BOOT」ボタンを押しながら「RESET」ボタンを一瞬押します。これで、ESP32 は書き込みモードに入ります。

4. ファームウェアの書き込み:

- 以下のコマンドを使用して、ダウンロードしたファームウェアを ESP32 に書き込みます。`/path/to/downloaded/firmware.bin` はダウンロードしたファイルのパスに置き換えてください。また、COMPORT はお使いのコンピューターの ESP32 の COM ポートまたはシリアルデバイスの名前に置き換えてください (Linux の場合は `"/dev/ttyUSB0"`、Mac の場合は `"/dev/tty.SLAB_USBtoUART"` など)。

```
esptool.py --chip esp32 --port COMPORT erase_flash
esptool.py --chip esp32 --port COMPORT write_flash -z 0x1000
/path/to/downloaded/firmware.bin
```

5. 再起動:

- 上記の手順が完了したら、ESP32 をリセットします。

6. MicroPython の確認:

- シリアル接続ツール(例: PuTTY、screen、minicom など)を使用して、ESP32 に接続します。

通信速度は 115200 ボーで接続します。

• 成功すれば、MicroPython のプロンプト`>>>`が表示され、MicroPython コードを直接入力したり、スクリプトを実行したりすることができます。

以上で、ESP32-DevKitC に MicroPython をインストールする手順は完了です。

リスト 3 Pico で LED を点滅させるプログラム

Raspberry Pi Pico でオンボード LED を点滅させるプログラムを書いてください。ただし以下の条件に従ってください。

- * 点灯状態は 0.5 秒
- * 消灯状態は 0.1 秒

Raspberry Pi Pico 上の MicroPython を使用してオンボード LED (通常は GPIO 25 に接続されている) を点滅させるプログラムを以下に示します。

```
from machine import Pin, Timer
import time
```

```
# GPIO 25 を出力モードで初期化
led = Pin(25, Pin.OUT)
```

```
while True:
    led.value(1) # LED を ON にする
    time.sleep(0.5) # 0.5 秒待つ

    led.value(0) # LED を OFF にする
    time.sleep(0.1) # 0.1 秒待つ
```

このコードを Raspberry Pi Pico 上の MicroPython にコピーして実行することで、指定された条件 (点灯 0.5 秒、消灯 0.1 秒) でオンボード LED が点滅します。

リスト 4 市販されているキャラクタ・ディスプレイの知識

秋月電子通商の有機ELキャラクタディスプレイモジュール(P-08277)について教えてください。

申し訳ございませんが、私の最後のトレーニングデータのカットオフは 2021 年 9 月であり、その時点での秋月電子通商の P-08277 という製品番号の「有機 EL キャラクタディスプレイモジュール」についての詳細情報は取得しておりません。

通常、秋月電子通商の商品に関する情報は、公式ウェブサイト上で提供されるデータシートや製品説明から得ることができます。このデータシートには、寸法、接続方法、電氣的仕様、使用するコントローラ IC の情報など、製品の詳細な仕様が記載されています。

もしご必要であれば、秋月電子通商の公式ウェブサイトを直接参照するか、該当の製品ページや関連するフォーラム・コミュニティを調査することをお勧めします。

リスト 5 SO1602A の制御プログラム(その 1)

ディスプレイ・コントローラ SO1602A を制御するプログラムを以下の条件で作ってください。

- MicroPython で記述
- I2C アドレスは 0x3C
- "Hello, SO1602A"という文字列を表示する

SO1602A は OLED キャラクタディスプレイモジュールの一つで、多くの場合 SSD131X シリーズのドライバ IC を使用しています。この IC は I2C を介して制御することができます。

以下は MicroPython を使用して SO1602A を制御する基本的なプログラムの例です。このコードは Raspberry Pi Pico やその他の MicroPython 対応のマイクロコントローラで動作することを前提としています。

```
from machine import I2C, Pin
import time

# OLED display constants
ADDR = 0x3C
CMD = 0x00
DATA = 0x40

# Init I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000) # SCL: GPIO9, SDA: GPIO8

def oled_write_cmd(cmd):
    i2c.writeto(ADDR, bytes([CMD, cmd]))

def oled_write_data(data):
    i2c.writeto(ADDR, bytes([DATA, data]))

def oled_init():
    oled_write_cmd(0xAE) # Display off
    oled_write_cmd(0xA8) # Set Multiplex
    oled_write_cmd(0x3F)
    oled_write_cmd(0xD3) # Set Display Offset
    oled_write_cmd(0x00)
    oled_write_cmd(0x40) # Set Display Start Line
    oled_write_cmd(0xA1) # Segment Re-map
    oled_write_cmd(0xC8) # COM Output Scan Direction
    oled_write_cmd(0xDA) # COM Pins hardware config
    oled_write_cmd(0x12)
    oled_write_cmd(0x81) # Contrast control
    oled_write_cmd(0xCF)
    oled_write_cmd(0xA4) # Disable Entire Display On
    oled_write_cmd(0xA6) # Set Normal Display
```

```

oled_write_cmd(0xAF) # Display on

def oled_set_pos(col, page):
    oled_write_cmd(0xB0 + page) # Set Page Address
    oled_write_cmd(0x10 + (col >> 4)) # Set Higher Column Start Address
    oled_write_cmd(0x00 + (col & 0x0F)) # Set Lower Column Start Address

def oled_clear():
    for page in range(8):
        oled_set_pos(0, page)
        for col in range(128):
            oled_write_data(0x00)

def oled_display_text(col, page, text):
    oled_set_pos(col, page)
    for char in text:
        for byte in font[char]:
            oled_write_data(byte)

# Assuming a basic 8x8 font named 'font'
# The font dictionary can map characters to their byte representations
font = {
    'H': [0x00, 0x7E, 0x11, 0x11, 0x11, 0x7E, 0x00, 0x00],
    # ... add other character representations ...
    'A': [...], 'e': [...], 'l': [...], etc.
}

oled_init()
oled_clear()
oled_display_text(0, 0, "Hello, SO1602A")

```

注意: 上記のコードは、`font`辞書に基づいて文字を表示しますが、完全なフォントセットは省略されています。使用するフォントやキャラクタセットに基づいて辞書を補完する必要があります。

リスト 6 SO1602A の制御プログラム(その 2)

oled_init 関数の内容が仕様に仕上がっていません。プログラムを修正してください。

申し訳ございません、誤りました。SO1602A は確かに SSD131X シリーズとは異なります。以下は、SO1602A を制御するための MicroPython のプログラムを修正したものです。SO1602A は通常、標準的なキャラクタディスプレイのコマンドセットを使用していると想定しています。

```

from machine import I2C, Pin

# OLED display constants
ADDR = 0x3C

# Init I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000) # SCL: GPIO9, SDA: GPIO8

def oled_cmd(cmd):

```

```

i2c.writeto(ADDR, bytes([0x00, cmd]))

def oled_init():
    oled_cmd(0x38) # Function set: 8-bit, 2-line, 5x8 dots
    oled_cmd(0x0C) # Display ON; Cursor off
    oled_cmd(0x01) # Clear display
    oled_cmd(0x06) # Entry mode: Move right, no shift

def oled_clear():
    oled_cmd(0x01) # Clear display

def oled_set_pos(col, row):
    if row == 0:
        oled_cmd(0x80 + col)
    else:
        oled_cmd(0xC0 + col)

def oled_display_text(col, row, text):
    oled_set_pos(col, row)
    i2c.writeto(ADDR, bytes([0x40] + [ord(c) for c in text]))

oled_init()
oled_clear()
oled_display_text(0, 0, "Hello, SO1602A")

```

注意: このコードは、SO1602A の標準的なキャラクタディスプレイコマンドセットを基にしています。もし特定のコントローラ IC やカスタム仕様を使用している場合は、データシートを参照し、必要に応じてコードを適宜調整してください。

リスト 7 UART の結線

Raspberry Pi Pico と ESP32-DevKitC を UART で通信するための結線方法を教えて。ただし以下の条件を満たすようにしてください。

- * Raspberry Pi Pico は UART0 を使う。
- * ESP32-DevKitC は UART2 を使う。

Raspberry Pi Pico と ESP32-DevKitC を UART で通信させるための結線はシンプルです。以下のステップに従ってください。

Raspberry Pi Pico のピン配置:

- ・ GP0: UART0 の TX (Transmit)
- ・ GP1: UART0 の RX (Receive)
- ・ GND: Ground

ESP32-DevKitC のピン配置:

- ・ GPIO17: UART2 の TX

- ・ GPIO16: UART2 の RX
- ・ GND: Ground

結線方法:

1. **TX to RX:** Raspberry Pi Pico の GP0 (TX)を ESP32-DevKitC の GPIO16 (RX)に接続します。
2. **RX to TX:** Raspberry Pi Pico の GP1 (RX)を ESP32-DevKitC の GPIO17 (TX)に接続します。
3. **Ground to Ground:** Raspberry Pi Pico の GND ピンを ESP32-DevKitC の GND ピンに接続します。

この結線方法で、Raspberry Pi Pico と ESP32-DevKitC の間で UART 通信が可能になります。ただし、以下の点を注意してください:

- ・両デバイスの動作電圧を確認し、それらが互換性があることを確認してください。Raspberry Pi Pico は 3.3V で動作しますが、ESP32-DevKitC も 3.3V で動作するため、この結線は安全です。
- ・UART 通信を開始する前に、両デバイスのボーレートや通信設定が一致していることを確認してください。

これで、Raspberry Pi Pico と ESP32-DevKitC を UART で接続し、通信する準備が整いました。

リスト 8 UART の送受信プログラム

Raspberry Pi Pico と ESP32-DevKitC を UART で接続するとします。Raspberry Pi Pico と ESP32-DevKitC との間で UART による通信を行うサンプルプログラムを教えてください。ただし以下の条件を満たすようにしてください。

- * Raspberry Pi Pico と ESP32-DevKitC の両方で MicroPython を使う。
- * Raspberry Pi Pico が送信側、ESP32-DevKitC が受信側とする。
- * 通信するデータはアスキー文字列とする。

了解しました。以下に、Raspberry Pi Pico (送信側)と ESP32-DevKitC (受信側)の UART 通信のための MicroPython サンプルプログラムを示します。

Raspberry Pi Pico (送信側):

```
from machine import UART, Pin
import time

# UART0 初期化
uart0 = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

while True:
    # 文字列送信
```

```
uart0.write('Hello, ESP32!\n')
time.sleep(1) # 1 秒待つ
```

ESP32-DevKitC (受信側):

```
from machine import UART, Pin
```

```
# UART2 初期化
```

```
uart2 = UART(2, baudrate=9600, tx=17, rx=16)
```

```
while True:
```

```
    # データが利用可能である場合に読み取ります。
```

```
    if uart2.any():
```

```
        data = uart2.readline() # 改行までのデータを読み取る
```

```
        print("Received:", data.decode().strip()) # アスキー文字列としてデコードして表示
```

上記のサンプルプログラムでは、Raspberry Pi Pico が毎秒"Hello, ESP32!\n"という文字列を ESP32-DevKitC に送信します。受信側の ESP32-DevKitC は、データが利用可能な場合に読み取り、コンソールに受信した文字列を表示します。

注意点として、両方のデバイスのボーレート(ここでは 9600)が一致していること、および先に説明した通りにデバイスを物理的に接続していることを確認してください。

リスト 9 Pico で温度測定をするプログラムを作成

Raspberry Pi Pico で以下のプログラムを書いてください。

- * MicroPython で記述する。

- * 温度センサーMCP9700A を Raspberry Pi Pico の ADC0 に接続し、1 秒間隔で温度を測定する。

- * 温度センサーで取得した温度を SO1062A に"Temp: 温度"の形式で表示する。

- * SO1602A に温度を表示する際には、文字列表示の直前に画面をクリアする。

- * SO1602A の I2C アドレスは 0x3c とする。

- * SO1602A は I2C0 に接続し、SDA は GPIO8、SCL は GPIO9 に接続する。

- * SO1602A の初期化コマンドは、0x38、0x0c、0x01、0x06 の順とする。

- * 温度センサーで取得した値を文字列として Raspberry Pi Pico の UART0 に送信する。

- * UART0 に送信する文字列は数値を文字列に変換しただけのものとし、他の情報は付加しない。

指定された条件に基づき、以下に Raspberry Pi Pico で使用する MicroPython プログラムを示します。

```
from machine import Pin, I2C, ADC, UART
import utime
```



```

# SO1602A の初期化コマンド
INIT_COMMANDS = [0x38, 0x0c, 0x01, 0x06]

# SO1602A の I2C セットアップ
i2c = I2C(0, sda=Pin(8), scl=Pin(9))
oled_addr = 0x3c

# MCP9700A の ADC セットアップ
adc = ADC(Pin(26)) # ADC0

# UART のセットアップ
uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

# SO1602A の初期化関数
def oled_init():
    for cmd in INIT_COMMANDS:
        i2c.writeto(oled_addr, bytes([0x00, cmd]))
        utime.sleep_ms(5)

# 温度を取得する関数
def get_temperature():
    voltage = (adc.read_u16() * 3.3) / 65535 # ADC の読み取り値を電圧に変換
    temp_c = (voltage - 0.5) * 100 # MCP9700A の特性を使用して温度に変換
    return temp_c

# SO1602A に文字列を表示する関数
def display_on_oled(string):
    i2c.writeto(oled_addr, bytes([0x00, 0x01])) # クリアコマンド
    utime.sleep_ms(2)
    i2c.writeto(oled_addr, bytes([0x40]) + string.encode()) # テキスト表示コマンド

# メインループ
oled_init()

while True:
    temperature = get_temperature()

    # OLED に温度を表示
    display_string = "Temp: {:.2f}".format(temperature)
    display_on_oled(display_string)

    # UART に温度を送信
    uart.write(str(temperature))

    utime.sleep(1)

```

このプログラムは、Raspberry Pi Pico に接続された MCP9700A から温度を読み取り、その値を SO1602A に表示し、UART0 に送信します。SO1602A の表示前には画面をクリアします。

リスト 10 ESP32 で MQTT ブローカにメッセージを送信するプログラム

ESP32-DevKitC で以下の条件を満たすプログラムを書いてください。

- * MicroPython で記述する。
- * ESP32-DevKitC は WiFi アクセスポイント(SSID: XXXXX, パスワード: YYYYYY)に接続する。
- * WiFi 接続時に自分自身の IP アドレスを表示する
- * UART2 で受信したデータを MQTT ブローカーに以下の JSON 形式で送信する。
...
{
 "temperature": 温度
}
...
* MQTT ブローカーは test.mosquitto.org を使用する。
* MQTT トピックは"home/temperature"とする。

指定された条件に基づき、以下に ESP32-DevKitC で使用する MicroPython プログラムを示します。

```
import network
import ujson
import time
import machine
from umqtt.simple import MQTTClient

# WiFi の設定
WIFI_SSID = 'XXXXXX'
WIFI_PASSWORD = 'YYYYYY'

# MQTT の設定
MQTT_BROKER = 'test.mosquitto.org'
MQTT_TOPIC = 'home/temperature'

# WiFi 接続
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(WIFI_SSID, WIFI_PASSWORD)

# WiFi に接続するまで待機
while not wlan.isconnected():
    time.sleep(1)

print('Connected to WiFi, IP address:', wlan.ifconfig()[0])

# MQTT クライアントの設定
client = MQTTClient("umqtt_client", MQTT_BROKER)
client.connect()

# UART2 の設定
```

```
uart = machine.UART(2, baudrate=9600, tx=17, rx=16)

while True:
    if uart.any():
        data = uart.readline().decode('utf-8').strip() # 受信データの読み取りとデコード

        # JSON データの作成
        payload = ujson.dumps({"temperature": float(data)})

        # MQTT ブローカーにデータを送信
        client.publish(MQTT_TOPIC, payload)
```

このプログラムは、ESP32-DevKitC が指定された WiFi ネットワークに接続し、自身の IP アドレスを表示します。次に、UART2 から受信したデータを MQTT ブローカー (test.mosquitto.org) に送信します。受信データは `home/temperature` というトピックの下で JSON 形式で送信されます。

注意点として、使用する前に `umqtt` の MicroPython モジュールをインストールする必要があります。