SunFounder Kepler Kit for Raspberry Pi Pico W

リリース *1.0*

www.sunfounder.com

2024 年 01 月 15 日

目次

SunFounder Kepler Kit をお選びいただき、ありがとうございます。

注釈: このドキュメントは以下の言語で利用可能です。

-
-
-

ご希望の言語でドキュメントにアクセスするために、それぞれのリンクをクリックしてください。

SunFounder Kepler Kit をお選びいただき、ありがとうございます。

このキットは Raspberry Pi Pico W を基盤にした学習キットです。

Raspberry Pi Pico W は、Infineon CYW4343 を使用してオンボードのシングルバンド 2.4GHz 無線インターフェース（802.11n）を追加し、Pico の形状を維持しながら基本的な GPIO 機能に加え、ネットワークにも接続できるため、IoT プロジェクトにも活用できます。例えば、セキュリティシステムのために IFTTT を使ったり、MQTT を使ってクラウドプレーヤーやクラウドサービスのベルシステムを構築することができます。

このキットには、ディスプレイ、音響、ドライバー、コントローラー、センサーなど、多様なコンポーネントが含まれており、電子機器に対する包括的な理解が得られます。

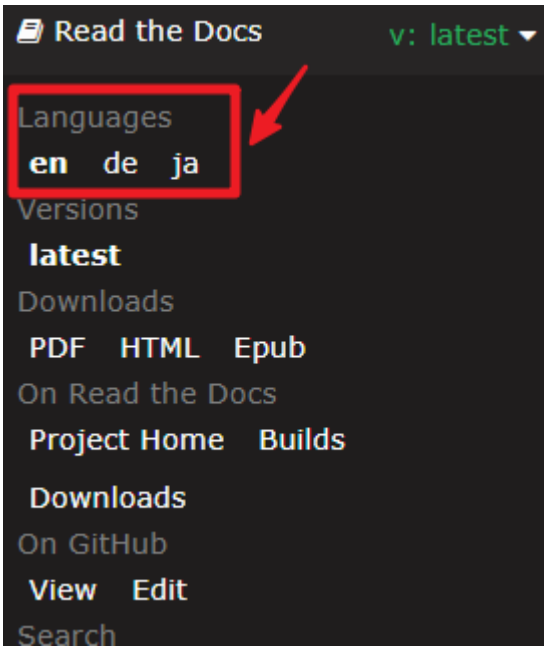
さらに、MicroPython、C/C++（Arduino）、Piper Make の 3 つのプログラミング言語を提供しています。各言語には、独自で興味深いプロジェクトが用意されているので、あなたのニーズに合わせて選ぶことができます。

当社がまだ提供していない他のプロジェクトに興味がある場合は、どうぞお気軽にメールでお知らせください。オンラインチュートリアルはできるだけ早く更新されます。

こちらがメールアドレスです：service@sunfounder.com。

表示言語について

このドキュメントは他の言語でもご利用いただけます。表示言語を切り替えるには、ページの左下にある **Read the Docs** アイコンをクリックしてください。



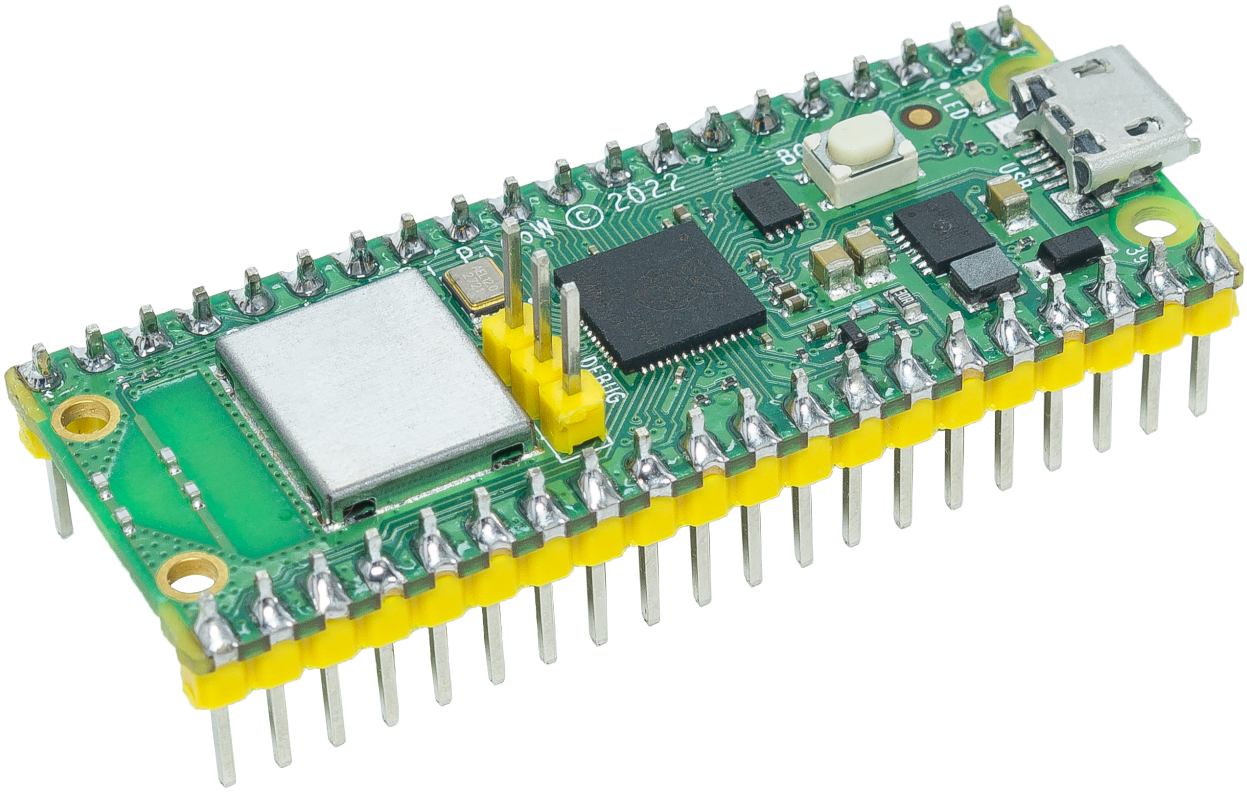
ソースコード

- SunFounder Kepler Kit
- または、Kepler Kit - GitHub でコードをチェックしてください。

コンテンツ

第 1 章

Raspberry Pi Pico W



Raspberry Pi Pico W は、ベストセラーである Raspberry Pi Pico 製品ラインに無線接続性をもたらします。私たちの RP2040 シリコンプラットフォームを基盤に、Pico 製品は高性能、低コスト、使いやすさといった私たちの特長をマイクロコントローラー領域に展開します。

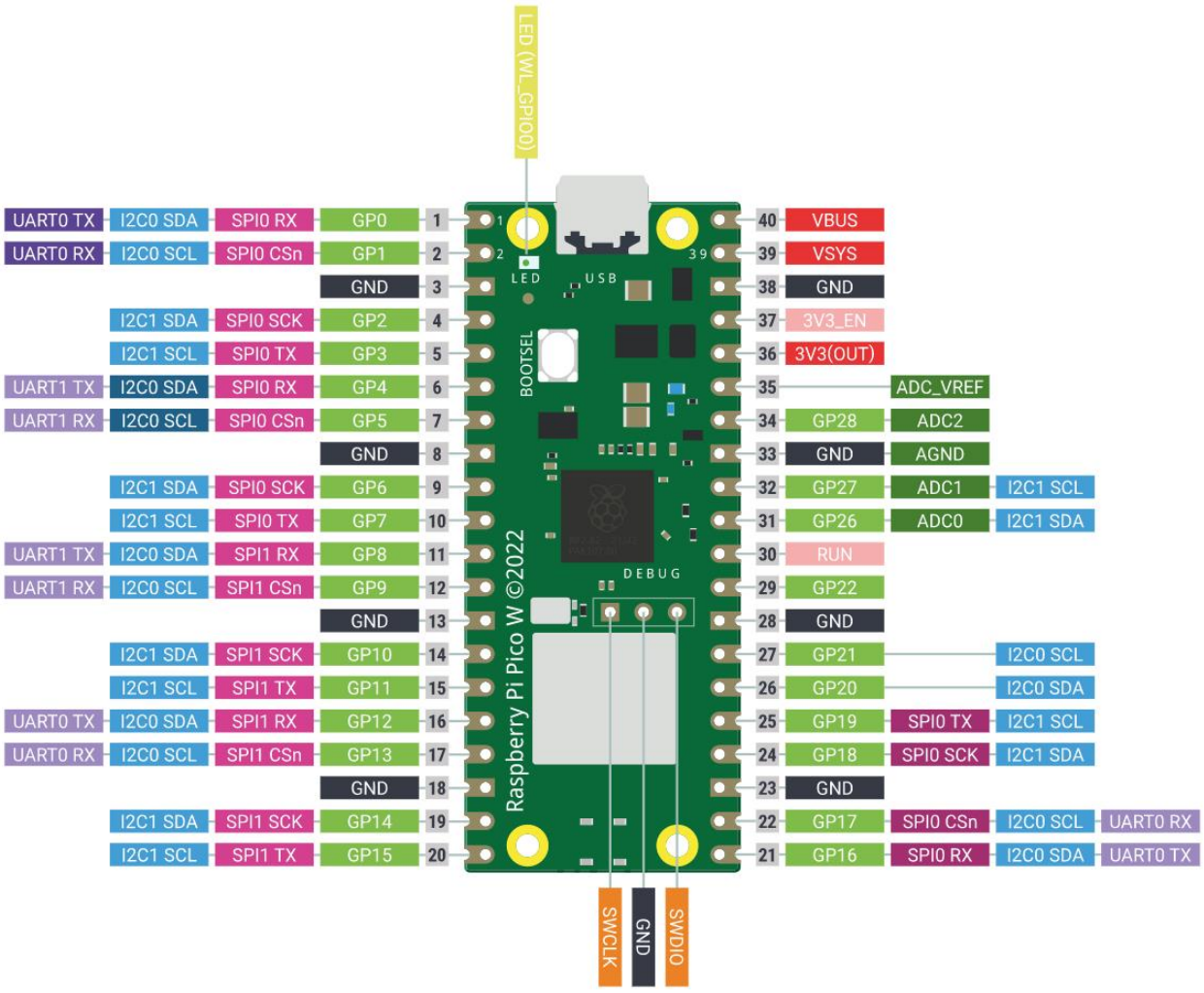
Raspberry Pi Pico W は、オンボードアンテナとモジュラーなコンプライアンス認証を備えた 2.4GHz 802.11 b/g/n の無線 LAN をサポートします。ステーションモードとアクセスポイントモードの両方で動作可能です。C 言語および MicroPython の開発者には、ネットワーク機能へのフルアクセスが可能です。RP2040 と 2MB のフラッシュメモリ、1.8–5.5V の入力電圧をサポートする電源チップを搭載。26 個の GPIO ピンを提供し、そのうち 3 つはアナログ入力として機能可能です。これらは 0.1 インチピッチのスルーホールパッドにキャステラーションエッジを

備えています。Raspberry Pi Pico W は、個々のユニットまたは 480 ユニットのリールで、自動組立て用に提供されています。

1.1 特長

- 21 mm x 51 mm のフォームファクタ
- Raspberry Pi によって英国で設計された RP2040 マイクロコントローラーチップ
- デュアルコア Arm Cortex-M0+ プロセッサ、最大 133MHz までの柔軟なクロック
- 264kB のオンチップ SRAM
- 2MB のオンボード QSPI フラッシュ
- 2.4GHz 802.11n 無線 LAN
- 26 個のマルチファンクション GPIO ピン、うち 3 つはアナログ入力
- 2 x UART、2 x SPI コントローラー、2 x I2C コントローラー、16 x PWM チャンネル
- 1 x USB 1.1 コントローラーおよび PHY、ホストとデバイスのサポートあり
- 8 x プログラマブル I/O（PIO）ステートマシン、カスタム周辺機器のサポート
- サポートする入力電力 1.8-5.5V DC
- 動作温度 -20 ° C から +70 ° C
- キャステラーションモジュールにより、キャリアボードへの直接はんだ付けが可能
- USB 経由のマスストレージを使ったドラッグアンドドロッププログラミング
- 低消費電力のスリープモードとドーマントモード
- 高精度なオンチップクロック
- 温度センサー
- オンチップでの整数および浮動小数点ライブラリの高速化

1.2 Pico のピン配置



名前	説明	機能
GP0-GP28	汎用入出力ピン	入力または出力として動作し、特定の用途はない
GND	0 ボルトのグラウンド	Pico W 周囲にいくつかの GND ピンがあり、配線が容易である。
RUN	Pico の有効/無効	別のマイクロコントローラーから Pico W を開始および停止。
GPxx_ADCx	汎用入出力またはアナログ入力	アナログ入力としても、デジタル入出力としても使用できるが、同時には使用できない。
ADC_VREF	アナログ-デジタル変換器（ADC）の電圧基準	任意のアナログ入力の基準電圧を設定する特別な入力ピン。
AGND	ADC 用の 0 ボルトのグラウンド	ADC_VREF ピンと一緒に使用するための特別なグラウンド接続。
3V3(O)	3.3 ボルト電源	Pico W が内部で動作するのと同じ 3.3V の電源供給。
3v3(E)	電源の有効/無効	3V3(O) 電源をオン/オフするとともに、Pico W もオフにすることができる。
VSYS	2-5 ボルト電源	Pico の内部電源供給に直接接続されたピン、Pico W をオフにしない限りオフにできない。
VBUS	5 ボルト電源	Pico のマイクロ USB ポートから取得した 5V の電源、3.3V 以上の電力が必要なハードウェアに使用。

Raspberry Pi Pico W を始めるために必要なすべては、こちら で見つかります。

また、以下のリンクも参考にしてください：

- Raspberry Pi Pico W 製品概要
- Raspberry Pi Pico W データシート
- Raspberry Pi Pico で始める：C/C++ 開発
- Raspberry Pi Pico C/C++ SDK
- Raspberry Pi Pico C/C++ SDK の API レベル Doxygen ドキュメンテーション
- Raspberry Pi Pico Python SDK
- Raspberry Pi RP2040 データシート
- RP2040 を用いたハードウェア設計
- Raspberry Pi Pico W 設計ファイル
- Raspberry Pi Pico W STEP ファイル

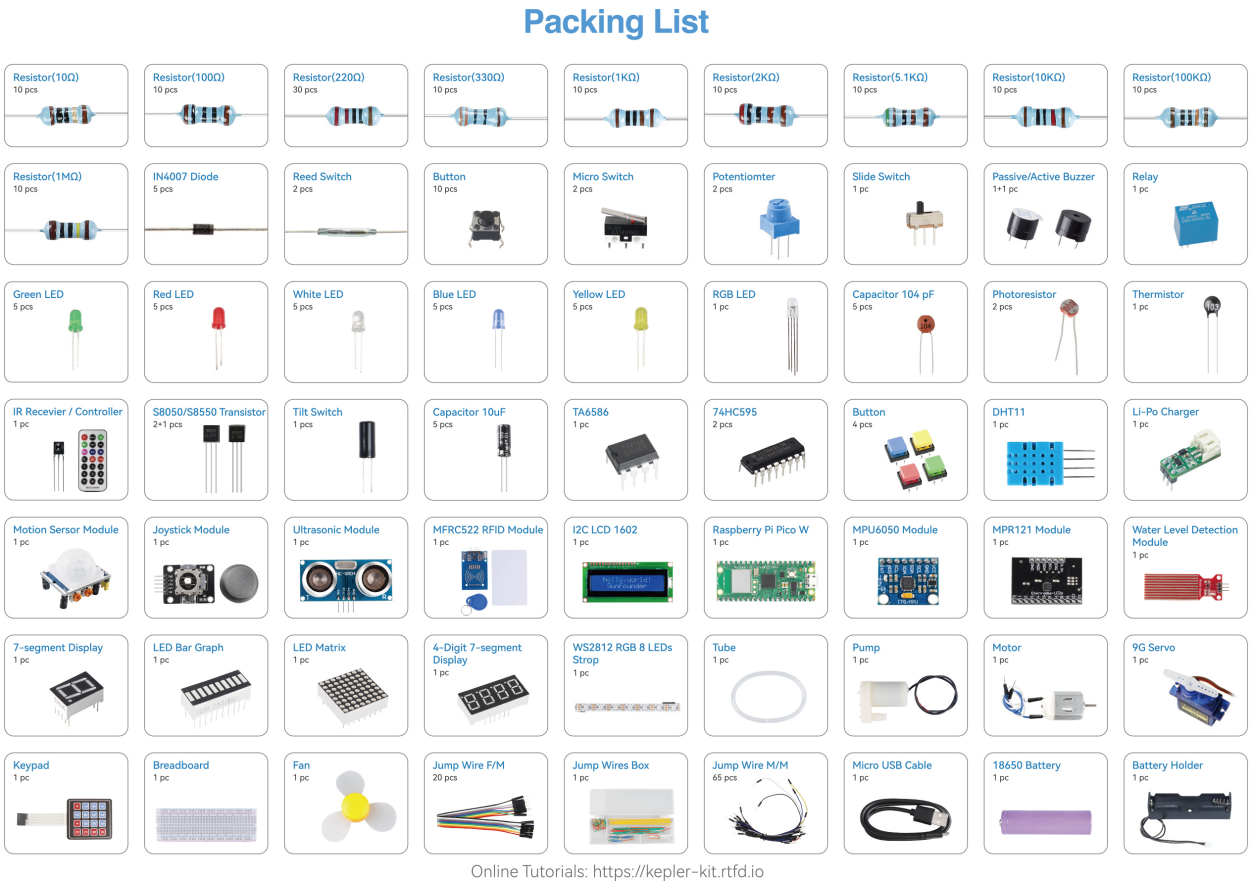
第 2 章

このキットに含まれるもの

以下は、このキットが届いた際に内容を確認できるような一覧です。

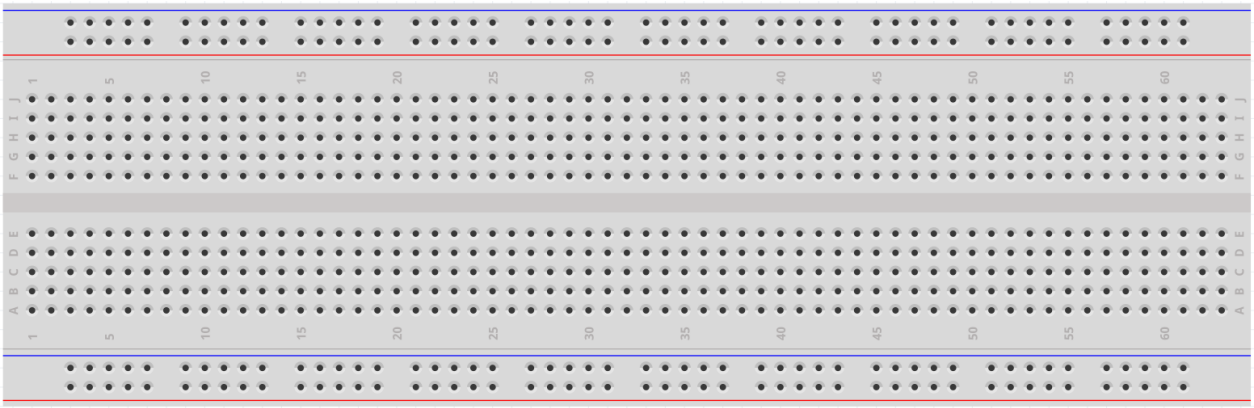
キットには非常に小さく、見た目が同じような部品がいくつか含まれているため、スタッフが梱包時に間違って送ることや見落とす可能性があります。不足しているか誤って送られた部品があれば、その名前をお知らせいただければ幸いです。

メールアドレスはこちらです：service@sunfounder.com。



基本部品

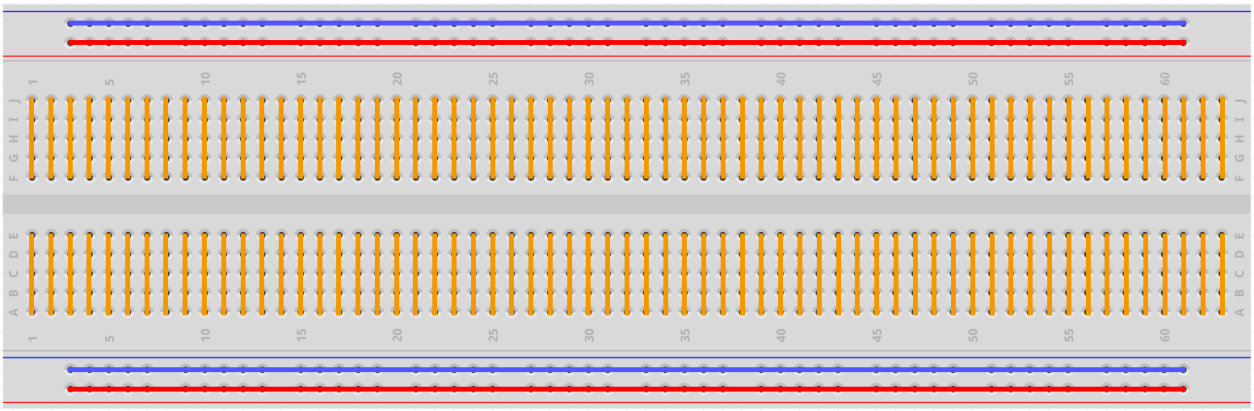
2.1 ブレッドボード



ブレッドボードは、電子回路のプロトタイピング用の構築ベースです。元々この言葉は、文字通りパンを切るために使われる磨かれた木片を指していました。1970 年代にはんだ不要のブレッドボード（別名プラグボード、ターミナルアレイボード）が登場し、現在では「ブレッドボード」という用語は一般的にこれらのものを指しています。

ブレッドボードは、最終的な回路設計を完了する前に、素早く回路を組み立ててテストするために使用されます。多くの穴があり、上記のようなコンポーネント（IC や抵抗器、ジャンパーワイヤなど）が挿入できます。ブレッドボードは、コンポーネントの簡単な挿入と取り外しが可能です。

写真はブレッドボードの内部構造を示しています。これらの穴は一見独立しているように見えますが、実際には内部で金属ストリップによって互いに接続されています。



ブレッドボードについてさらに知りたい場合は、以下を参照してください：How to Use a Breadboard - Science Buddies

例

- **はじめてのブレッドボード！**

2.2 ジャンパーワイヤー

二つの端子を接続するワイヤーをジャンパーワイヤーと呼びます。様々な種類のジャンパーワイヤーがありますが、ここではブレッドボードで使用されるものに焦点を当てます。特に、ブレッドボード上の任意の位置からマイクロコントローラーの入出力ピンへ電気信号を転送するために使用されます。

ジャンパーワイヤーは、「エンドコネクタ」をブレッドボードに用意されたスロットに挿入することで取り付けられます。ブレッドボードの表面の下には、エリアに応じて行または列のグループでスロットを接続する平行なプレートがいくつかセットされています。この「エンドコネクタ」は、特定のプロトタイプで接続が必要な特定のスロットに、はんだ付けせずに挿入されます。

ジャンパーワイヤーには三種類あります：Female-to-Female（メス-メス）、Male-to-Male（オス-オス）、そしてMale-to-Female（オス-メス）です。Male-to-Female と呼ぶ理由は、一方の端に突出した先端があり、もう一方の端が凹んだメス端子であるからです。Male-to-Male は両端がオスであり、Female-to-Female は両端がメスです。

注釈:

- プロジェクトによっては、複数のタイプのジャンパーワイヤーが使用されることがあります。
- ジャンパーワイヤーの色は異なる場合がありますが、その機能がそれに応じて異なるわけではありません。それは各回路間の接続をより容易に識別するために設計されています。

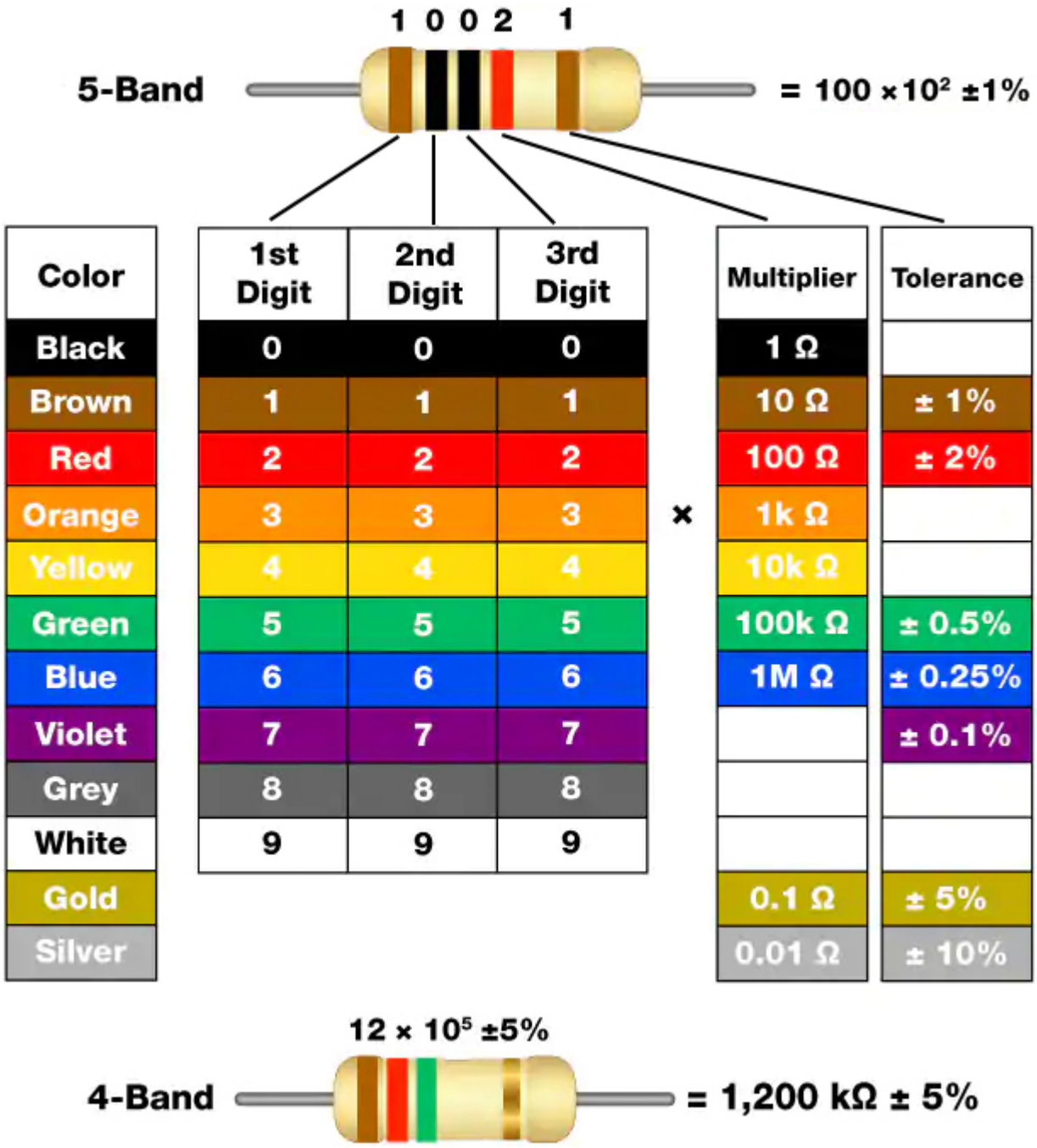
2.3 抵抗器

抵抗器は、分岐電流を制限することができる電子要素です。固定抵抗器は抵抗値を変更することができない種類の抵抗器であり、一方で可変抵抗器やポテンショメーターの抵抗値は調整することができます。

抵抗器の一般的に使われる回路記号は二つあります。通常、その抵抗値は記載されています。ですから、これらの記号が回路に見られた場合、それは抵抗器を表しています。

Ω は抵抗の単位であり、より大きな単位には K Ω、M Ωなどがあります。それらの関係は次のように示されます：1 M Ω = 1000 K Ω、1 K Ω = 1000 Ω。通常、抵抗値はその上に記載されています。

抵抗器を使用する際には、まずその抵抗値を知る必要があります。方法は二つあります：抵抗器のバンドを観察するか、マルチメーターで抵抗を測定します。便利で速いため、最初の方法が推奨されます。



このカードに示されているように、各色は数字に対応しています。

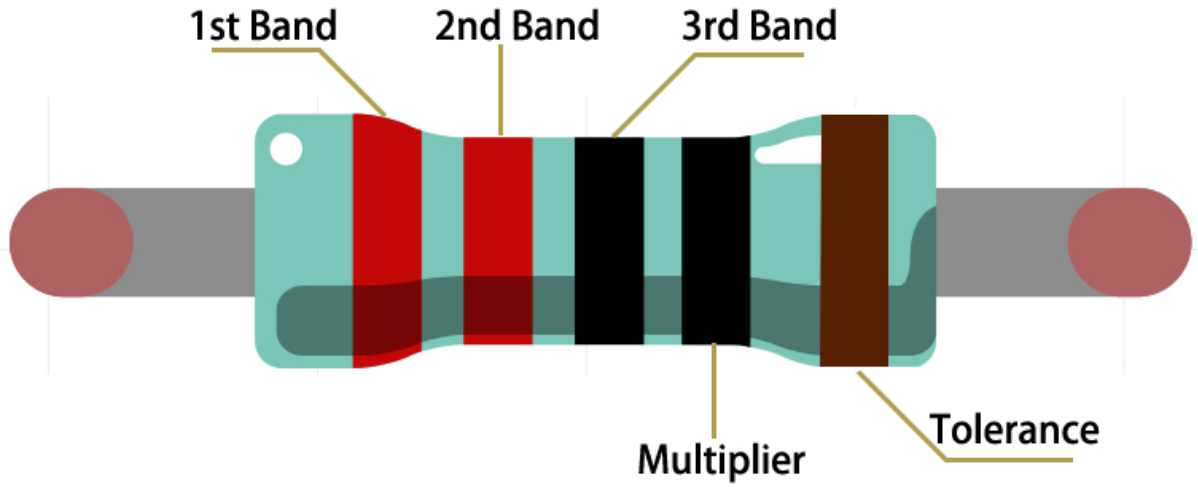
黒	茶	赤	オレンジ	黄	緑	青	紫	灰	白	金	銀
0	1	2	3	4	5	6	7	8	9	0.1	0.01

4 バンドおよび 5 バンドの抵抗器がよく使用されており、それぞれに 4 つまたは 5 つの彩色バンドがあります。

通常、抵抗器を手に入れたときには、どの端から色を読み取るべきかを決定するのが難しい場合があります。ヒントとして、4 番目と 5 番目のバンドの間の間隔が比較的大きいことです。

したがって、抵抗器の一方の端にある二つの彩色バンドの間隔を観察することができます。他のバンドの間隔よりも大きい場合、反対側から読み取ることができます。

以下に示すように、5 バンドの抵抗器の抵抗値をどのように読み取るか見てみましょう。



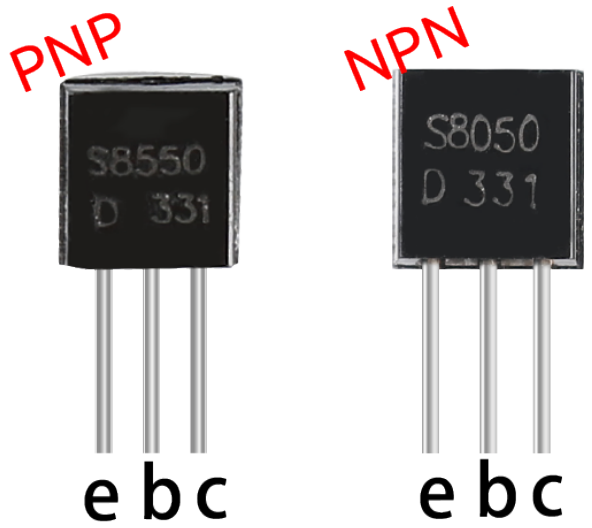
この抵抗器の場合、抵抗は左から右に読むべきです。値は次の形式であるべきです：1 番目のバンド 2 番目のバンド 3 番目のバンド x 10^乗数（Ω） そして許容誤差は ± 許容誤差 % です。したがって、この抵抗器の抵抗値は 2（赤）2（赤）0（黒）x 10^0（黒）Ω = 220 Ωであり、許容誤差は ±1%（茶）です。

表 1　一般的な抵抗器の色帯

抵抗器	色帯
10 Ω	茶黒黒銀茶
100 Ω	茶黒黒黒茶
220 Ω	赤赤黒黒茶
330 Ω	オレンジオレンジ黒黒茶
1k Ω	茶黒黒茶茶
2k Ω	赤黒黒茶茶
5.1k Ω	緑茶黒茶茶
10k Ω	茶黒黒赤茶
100k Ω	茶黒黒オレンジ茶
1M Ω	茶黒黒緑茶

抵抗器についての詳細は Wiki で学ぶことができます：抵抗器 - Wikipedia。

2.4 トランジスタ



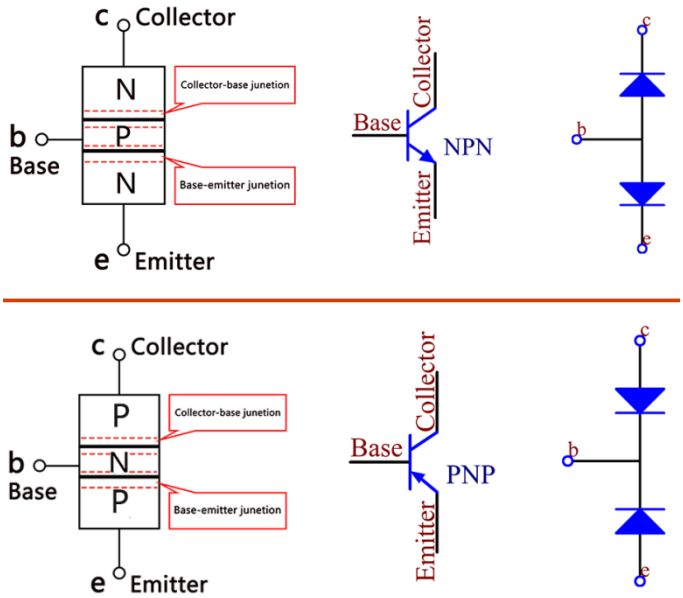
トランジスタは電流によって電流を制御する半導体デバイスです。弱い信号を大きな振幅の信号に増幅する機能を持ち、非接触スイッチとしても使用されます。

トランジスタは P 型と N 型の半導体で構成された 3 層の構造です。これにより内部には 3 つの領域が形成されます。中央の薄い部分がベース領域であり、他の 2 つはいずれも N 型か P 型です。多数キャリアが豊富な小さい領域がエミッタ領域であり、もう一方がコレクタ領域です。この構造によって、トランジスタは増幅器として機能します。これら 3 つの領域からそれぞれ 3 つの端子が生成され、これがベース (b)、エミッタ (e)、コレクタ (c) です。これらは 2 つの P-N 接合、すなわちエミッタ接合とコレクタ接合を形成します。トランジスタの回路記号の矢印の方向はエミッタ接合の方向を示しています。

- P–N 接合 - Wikipedia

半導体のタイプに基づいて、トランジスタは NPN と PNP の 2 つのグループに分けられます。略称から、前者は 2 つの N 型半導体と 1 つの P 型で作られ、後者はその逆です。下の図を参照してください。

注釈: s8550 は PNP トランジスタで、s8050 は NPN です。見た目は非常によく似ているため、ラベルを注意深く確認する必要があります。



NPN トランジスタにハイレベルの信号が通ると、エネルギーが供給されます。しかし、PNP トランジスタはローレベルの信号で制御されます。両方のタイプのトランジスタは非接触スイッチとして頻繁に使用されます。

- S8050 トランジスタのデータシート
- S8550 トランジスタのデータシート

ラベル面を自分たちに向け、ピンを下に向けます。左から右に、エミッタ (e)、ベース (b)、コレクタ (c) のピンがあります。

注釈:

- ベースは、より大きな電気供給のゲートコントローラーです。
- NPN トランジスタでは、コレクタが大きな電気供給であり、エミッタがその供給の出口です。PNP トラン

ジスタはその逆です。

例

- *2.15* **トランジスタの２種類** (MicroPython ユーザー向け)
- *2.16* **他の回路を制御する** (MicroPython ユーザー向け)
- *3.1* **ビープ音** (MicroPython ユーザー向け)
- *3.2* **カスタムトーン** (MicroPython ユーザー向け)
- *7.1* **光センサー・テルミン** (MicroPython ユーザー向け)
- *7.3* **警報サイレンランプ** (MicroPython ユーザー向け)
- *7.8 RFID* **音楽プレーヤー** (MicroPython ユーザー向け)
- *7.9* **フルーツピアノ** (MicroPython ユーザー向け)
- *7.10* **バックアップ支援** (MicroPython ユーザー向け)
- *3.1 -* **ビープ音** (Arduino ユーザー向け)
- *3.2 -* **カスタムトーン** (Arduino ユーザー向け)
- *2.15 -* **トランジスタの二種類** (Arduino ユーザー向け)
- *2.16 -* **別の回路を制御する** (Arduino ユーザー向け)
- *2.3* **サービスベル** (Piper Make ユーザー向け)
- *2.11* **逆転警報システム** (Piper Make ユーザー向け)
- *2.13* **リアクションゲーム** (Piper Make ユーザー向け)

2.5 コンデンサ

容量（キャパシタンス）とは、特定の電位差のもとで蓄積される電荷量を指し、記号は C、国際単位はファラッド（F）です。一般に、電場内で電荷は力の影響を受けて移動します。導体間に媒体が存在する場合、電荷の移動は妨げられ、導体上に電荷が蓄積されます。

この蓄積される電荷量を容量と呼びます。コンデンサは電子機器で最も広く使用される電子部品の一つであり、直流隔離、カップリング、バイパス、フィルタリング、調整回路、エネルギー変換、制御回路などに幅広く使用されています。コンデンサは、電解コンデンサ、固体コンデンサなどに分類されます。

材料特性によって、コンデンサはアルミニウム電解コンデンサ、フィルムコンデンサ、タンタルコンデンサ、セラミックコンデンサ、スーパーコンデンサなどに分類されます。

このキットでは、セラミックコンデンサと電解コンデンサが使用されています。

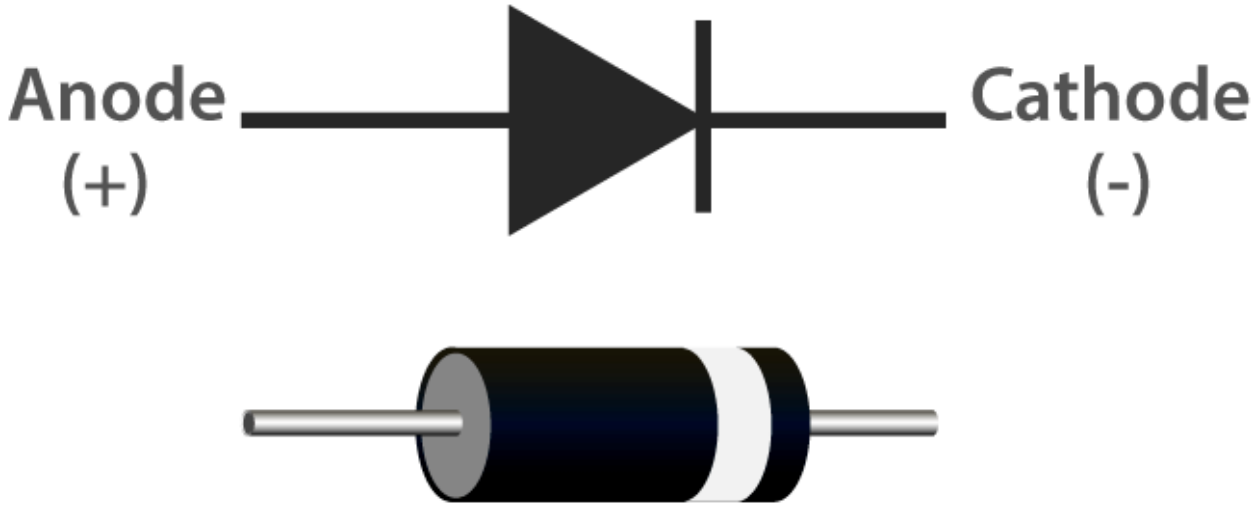
- Ceramic Capacitor - Wikipedia
- Electrolytic Capacitor - Wikipedia

セラミックコンデンサには 103 または 104 というラベルがあり、これは容量値を表しています。103=10x10^3pF、104=10x10^4pF です。

単位変換

1F=10^3mF=10^6uF=10^9nF=10^12pF

2.6 ダイオード



ダイオードは、二つの電極を持つ電子部品です。電流は一方向にしか流れないため、この機能はしばしば「整流」と呼ばれます。そのため、ダイオードは一種の電子版の逆止弁と考えることができます。

ダイオードの二つの端子は極性があり、正の端子をアノード、負の端子をカソードと呼びます。カソードは通常、銀製であるか、色の帯があります。電流の流れる方向を制御するのがダイオードの重要な特性であり、アノードからカソードに向かって電流が流れます。ダイオードの動作は逆止弁の動作に似ています。ダイオードの最も重要な特性の一つは、非線形の電流電圧特性です。アノードに高い電圧がかかると、アノードからカソードに電流が流れ、このプロセスは順方向バイアスと呼ばれます。しかし、カソードに高い電圧がかかると、ダイオードは電気を通さず、このプロセスは逆方向バイアスと呼ばれます。

その一方向の導電性のために、ダイオードは複雑な電子回路にほぼ必ず用いられます。ダイオードは最初に作成された半導体デバイスの一つで、その用途は広範です。

しかし、実際にはダイオードは完璧なオンオフ方向性を持っているわけではなく、むしろ特定のダイオード技術によって決まるより複雑な非線形電子特性を持っています。

ダイオードは p 型半導体と n 型半導体によって形成された p-n 接合であり、その界面の両側に空間電荷層が形成され、自己生成された電場があります。これは、p-n 接合の両側のキャリア濃度の差による拡散電流と、自己生成された電場によるドリフト電流が等しいため、外部電圧がかかっていないときに電気的に平衡状態にある。順方向バイアスが発生すると、外部電場と自己生成された電場との相互抑制により、キャリアの拡散電流が増加し、順方向電流（すなわち、導電性の理由）が発生します。逆方向バイアスが発生すると、外部電場と自己生成された電場がさらに強化され、逆方向バイアス電圧の値に依存しない一定の逆方向飽和電流 I0 が形成されます（これが非導電性の理由です）。逆方向バイアス電圧がある程度まで高くなると、p-n 接合の空間電荷層内の電場強度が臨界値に達し、キャリアの増倍プロセスが発生し、多数の電子-ホール対が生成され、逆方向のブレークダウン電流が大きくなります。これをダイオードのブレークダウン現象と呼びます。

1. 順方向特性

外部から順方向電圧がかかった場合、順方向特性の初めでは順方向電圧は非常に小さく、p-n 接合の電場の遮断効果を克服できません。順方向電流はほぼゼロで、この部分をデッドゾーンと呼びます。このダイオードが導電しない順方向電圧をデッドバンド電圧と呼びます。順方向電圧がデッドバンド電圧より大きくなると、p-n 接合の電場が克服され、ダイオードは順方向に導電します。電流は電圧とともに急増します。通常の使用電流範囲内で、ダイオードの端子電圧は導通中ほぼ一定であり、この電圧をダイオードの順方向電圧と呼びます。

2. 逆方向特性

逆方向電圧がかかり、一定の範囲を超えない場合、ダイオードを通る電流はわずかなキャリアが漂流運動によって形成される逆方向電流です。逆方向電流は非常に小さく、ダイオードは遮断状態にあります。この逆方向電流は逆方向飽和電流または漏れ電流とも呼ばれ、温度に大きく影響を受けます。

3. ブレークダウン

逆方向電圧が一定の値を超えると、逆方向電流は急激に増加し、これを電気的ブレークダウンと呼びます。電気的ブレークダウンを引き起こす臨界電圧を逆方向ブレークダウン電圧と呼びます。電気的ブレークダウンが発生すると、ダイオードは一方向の導電性を失います。したがって、逆方向電圧が高すぎる場合は、ダイオードの使用を避ける必要があります。

初期のダイオードは「キャットウィスカー」結晶と真空管（「熱電子管」とも呼ばれる）で構成されていました。今日の最も一般的なダイオードは、シリコンやゲルマニウムなどの半導体材料を使用しています。

- P–N 接合 - Wikipedia
- ダイオード - Wikipedia

2.7 Li-po 充電モジュール

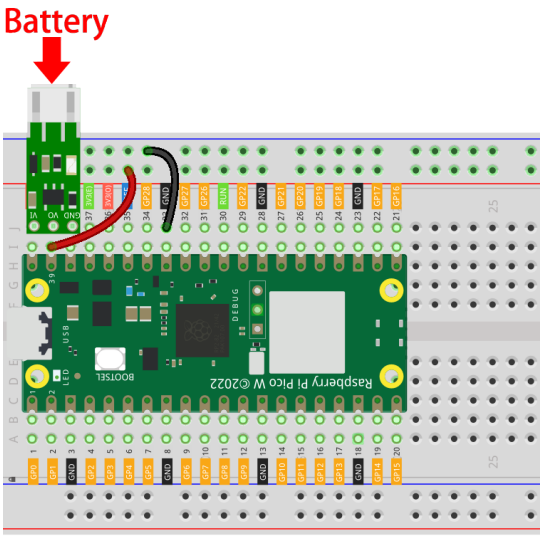
これは、Raspberry Pi Pico/Pico H/Pico W 用に設計された Li-po 充電モジュールです。以下に示すように、ブレッドボードにこのモジュールと Pico を接続し、他端にバッテリーを接続すれば、使用する準備が整います。

Pico W に USB ケーブルを接続し、コンピューターまたはソケットに挿入すると、Li-po 充電モジュールのインジケータライトが点灯し、バッテリーが同時に充電されます。USB ケーブルを抜くと、Pico W はバッテリーで駆動され、プロジェクトを継続できます。

注釈: 性能が低いコンピューターで、この充電モジュールを接続した状態で Pico W をコンピューターに接続すると、コンピューターが Pico W を認識しない場合があります。

その原因は、接続後にバッテリーを充電する際に、USB ポートの電圧が下がり、Pico W の電源が不足してコンピューターに認識されないためです。

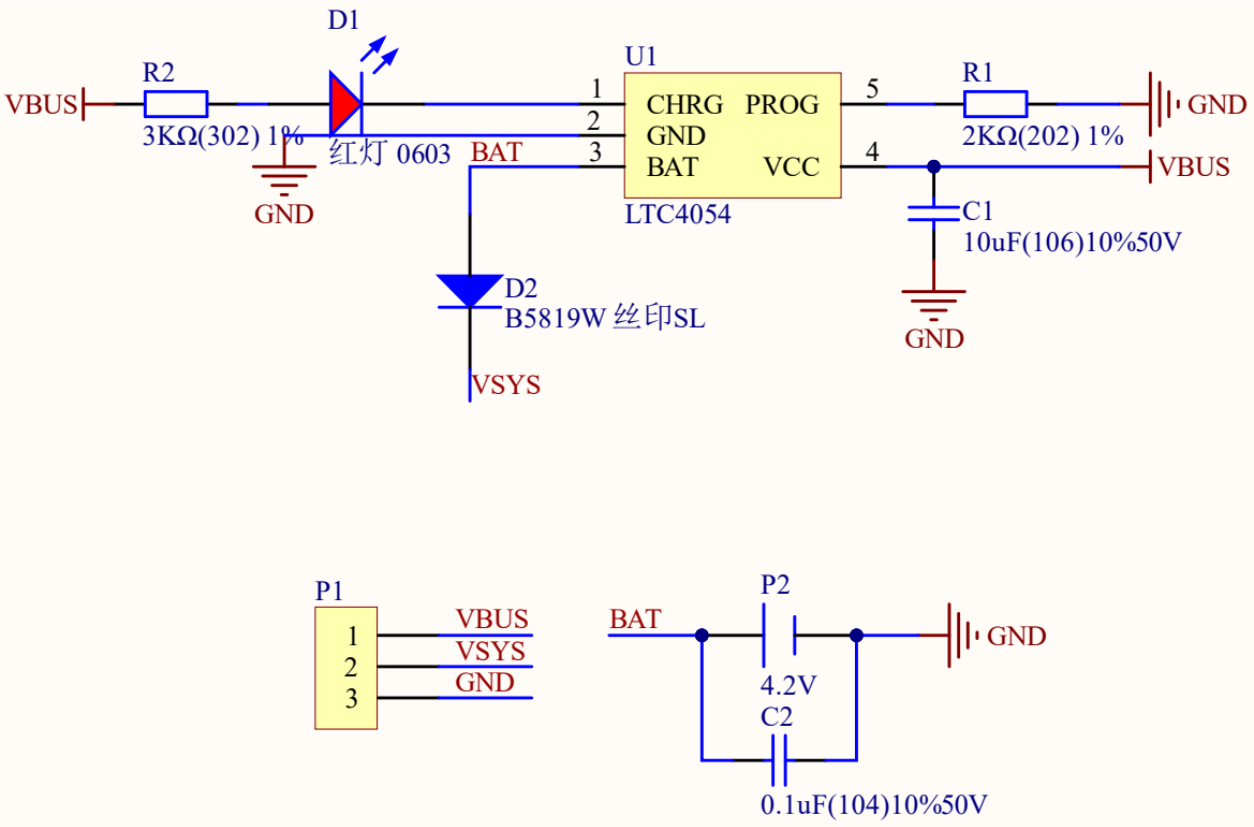
このような場合、Li-Po 充電モジュールを抜いてから、再度 Pico W を接続する必要があります。



特長

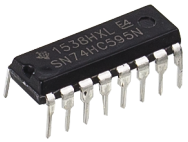
- 入力電圧: 5V
- 出力電圧: 3.3V
- サイズ: 20mmx7mm
- インターフェースモデル: PH2.0
- 1A バッテリーホルダーおよび 800mAh 18650 も一緒に使用できるものがあります。

回路図



チップ

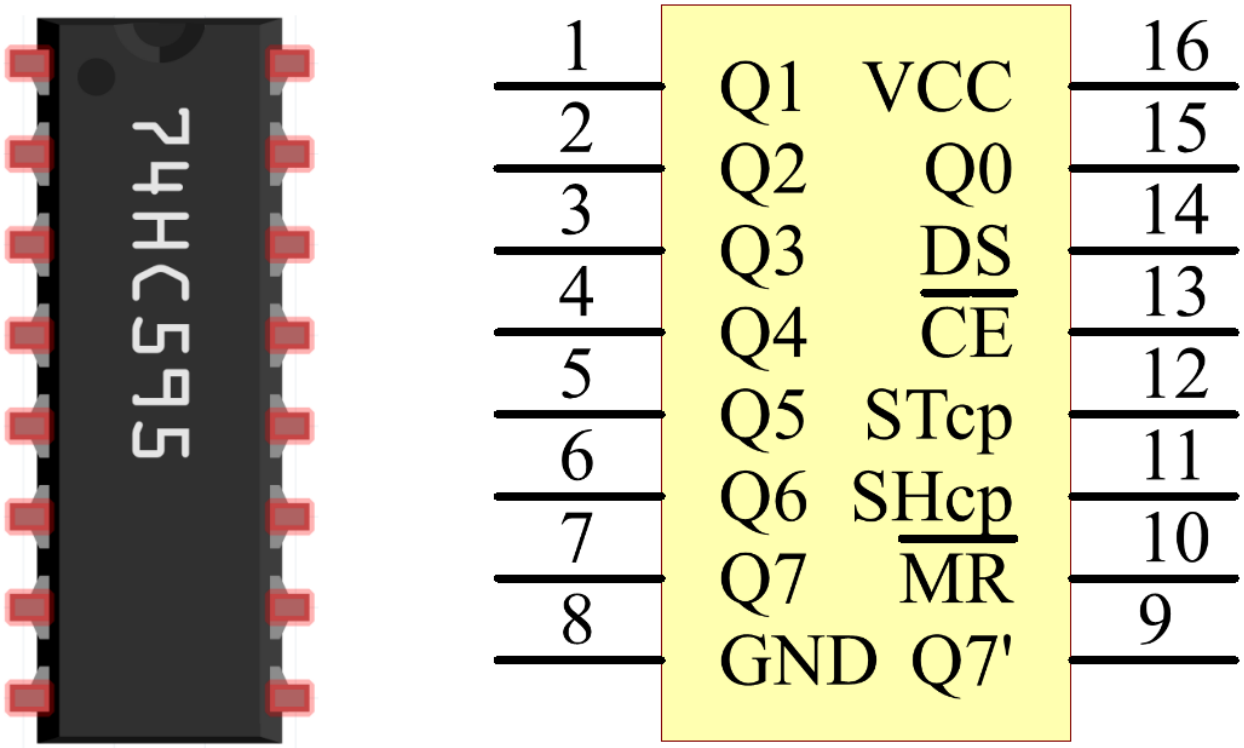
2.8 74HC595



74HC595 は、8 ビットのシフトレジスタと、3 状態の並列出力を持つストレージレジスタで構成されています。シリアル入力を並列出力に変換するため、MCU の IO ポートを節約することができます。

- MR（ピン 10）がハイレベルで、OE（ピン 13）がローレベルの場合、SHcp の立ち上がりエッジでデータが入力され、SHcp の立ち上がりエッジを介してメモリレジスタに移動します。
- 2 つのクロックが一緒に接続されている場合、シフトレジスタは常にメモリレジスタよりも 1 パルス早いです。
- メモリレジスタには、シリアルシフト入力ピン（Ds）、シリアル出力ピン（Q）、および非同期リセットボタン（ローレベル）があります。
- メモリレジスタは、3 つの状態で並列の 8 ビットバスを出力します。

- OE が有効（ローレベル）の場合、メモリレジスタのデータがバス（Q0〜Q7）に出力されます。
- 74HC595 データシート



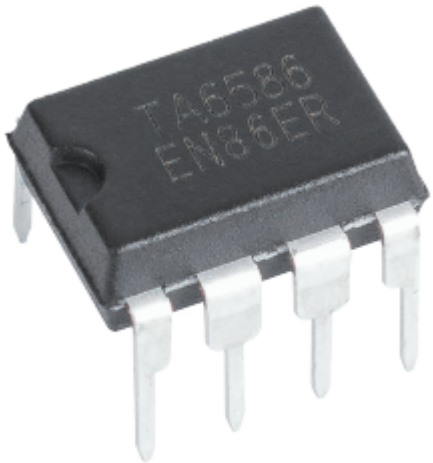
74HC595 のピンとその機能：

- **Q0-Q7**: 8 ビットの並列データ出力ピン。直接 8 つの LED または 7 セグメントディスプレイの 8 ピンを制御可能。
- **Q7'**: シリーズ出力ピン。別の 74HC595 の DS に接続して、複数の 74HC595 を直列に接続。
- **MR**: リセットピン。ローレベルでアクティブ。
- **SHcp**: シフトレジスタのタイムシーケンス入力。立ち上がりエッジで、シフトレジスタ内のデータが 1 ビットずつ順次移動します。
- **STcp**: ストレージレジスタのタイムシーケンス入力。立ち上がりエッジで、シフトレジスタ内のデータがメモリレジスタに移動。
- **CE**: 出力許可ピン。ローレベルでアクティブ。
- **DS**: シリアルデータ入力ピン。
- **VCC**: 正の電源電圧。
- **GND**: グラウンド。

例

- *5.1* **マイクロチップ** *- 74HC595* (MicroPython ユーザー向け)
- *5.2* **数字表示** (MicroPython ユーザー向け)
- *5.3* **時間カウンター** (MicroPython ユーザー向け)
- *5.4 8x8* **ピクセルグラフィックス** (MicroPython ユーザー向け)
- *7.4* **乗客カウンター** (MicroPython ユーザー向け)
- *7.5 GAME - 10* **秒ゲーム** (MicroPython ユーザー向け)
- *7.6* **交通信号機** (MicroPython ユーザー向け)
- *7.12* **デジタル水平器** (MicroPython ユーザー向け)
- *5.1* **マイクロチップ** *- 74HC595* (Arduino ユーザー向け)
- *5.2 -* **数字表示** (Arduino ユーザー向け)
- *5.3 -* **タイムカウンター** (Arduino ユーザー向け)
- *5.4 - 8x8* **ピクセルグラフィックス** (Arduino ユーザー向け)

2.9 TA6586 - モータードライバーチップ



TA6586 は、双方向の DC モーターを駆動するために設計されたモノリシック IC です。方向を制御するための論理入力ピンが 2 つあり、順方向と逆方向の制御が可能です。この回路は優れた抗干渉性能、小さいスタンバイ電流、そして低い出力飽和圧降を特徴としています。内蔵のクランプダイオードが誘導性負荷電流の放出に対する逆衝撃を逆転させるため、リレー、DC モーター、ステッピングモーターの駆動やスイッチング電源の使用が安全かつ信頼性があります。TA6586 は、玩具車、遠隔操作される航空機モータードライブ、自動バルブモーター、電磁ロック駆動、精密機器などの回路に適しています。

特長

- 低スタンバイ電流：≦ 2uA
- 広い供給電圧範囲
- 内蔵ブレーキ機能
- サーマルシャットダウン保護
- 過電流制限および短絡保護機能
- DIP8 Pb フリーパッケージ

ピン機能

Pin NO	Name	Function
1	BI	Backward input
2	FI	Forward input
3	GND	Ground
4	Vcc	Vcc
5, 6	FO	Forward output
7, 8	BO	Backward output

入力真理値表

2pin Finput	1pin Binput	5,6pin Foutput	7,8pin Boutput
H	L	H	L
L	H	L	H
H	H	L	L
L	L	Open	Open

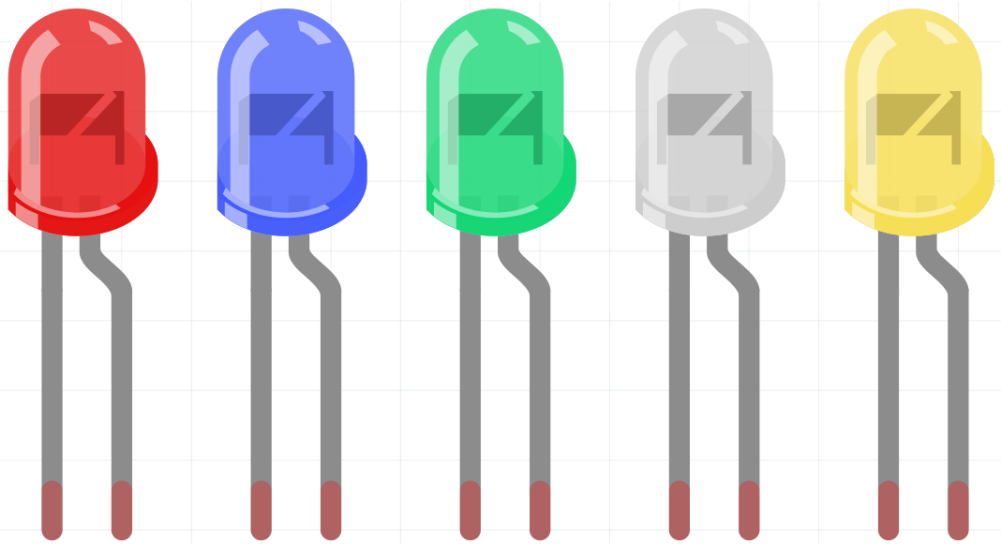
例

- *3.5* **スモールファン** (MicroPython ユーザー向け)
- *3.5 -* **小型ファン** (Arduino ユーザー向け)
- *3.6* **ポンピング** (MicroPython ユーザー向け)
- *3.6 -* **ポンピング** (Arduino ユーザー向け)

- *2.12* **スマートファン** (Piper Make ユーザー向け)

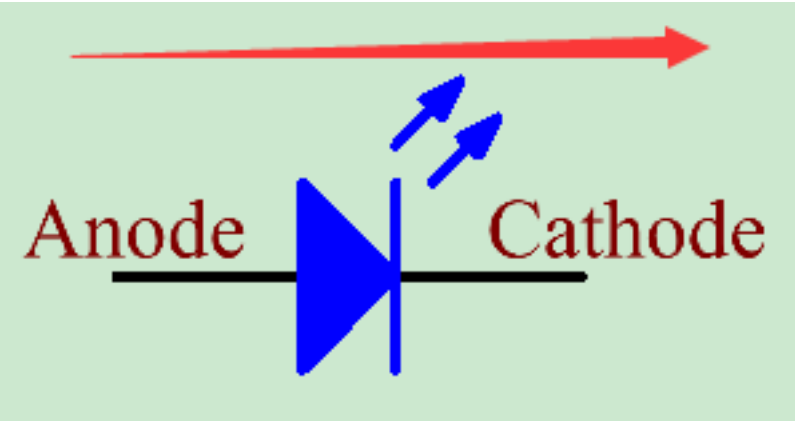
ディスプレイ

2.10 LED



半導体発光ダイオードは、PN 接合を通じて電気エネルギーを光エネルギーに変換する部品です。波長によっては、レーザーダイオード、赤外発光ダイオード、および一般的には発光ダイオード（LED）として知られている可視光発光ダイオードに分類できます。

ダイオードは単方向の導電性があり、回路記号の矢印が示すように電流が流れます。アノードに正の電源、カソードに負の電源を供給することで、LED は点灯します。



LED には二つのピンがあります。長い方がアノードで、短い方がカソードです。逆に接続しないように注意してください。LED には固定された順方向電圧降下があり、この降下を上回る供給電圧があると LED が焼けるため、回路に直接接続することはできません。赤、黄、緑の LED の順方向電圧は 1.8V、白の LED は 2.6V です。ほとんどの LED は最大電流 20mA まで耐えられるため、直列に電流制限抵抗を接続する必要があります。

抵抗値の計算式は以下の通りです：

R = (Vsupply – VD)/I

R は電流制限抵抗の抵抗値、 **Vsupply** は供給電圧、 **VD** は電圧降下、 **I** は LED の動作電流を表します。

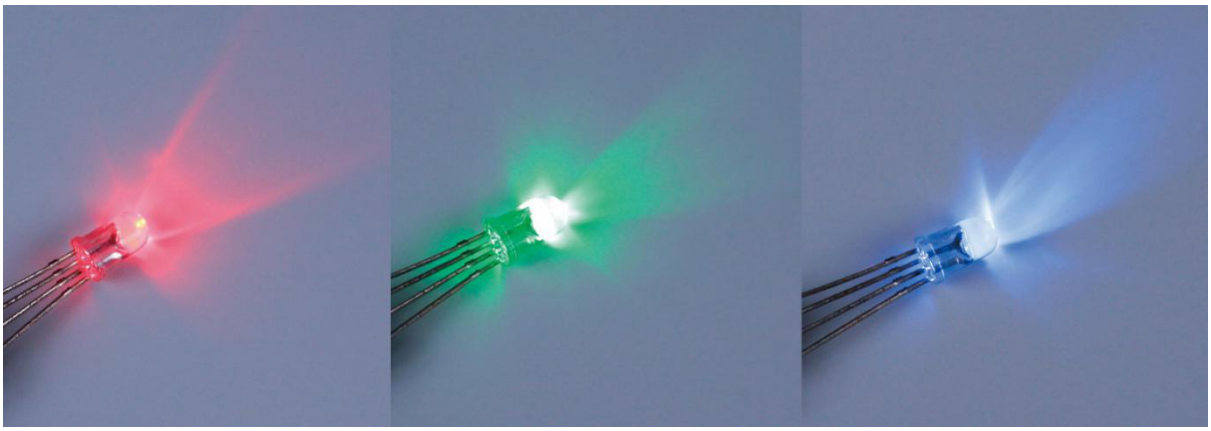
詳しい LED の紹介はこちら： LED - Wikipedia。

例

- *2.1* **こんにちは、*LED*！** （MicroPython ユーザー向け）
- *2.3 LED* **のフェード** （MicroPython ユーザー向け）
- *7.3* **警報サイレンランプ** （MicroPython ユーザー向け）
- *7.6* **交通信号機** （MicroPython ユーザー向け）
- *7.10* **バックアップ支援** （MicroPython ユーザー向け）
- *2.1 -* **こんにちは、*LED*！** （Arduino ユーザー向け）
- *2.3 - Fading LED* （Arduino ユーザー向け）
- *2.1 LED* **の点滅** （Piper Make ユーザー向け）
- *2.2* **ボタン** （Piper Make ユーザー向け）
- *2.3* **サービスベル** （Piper Make ユーザー向け）
- *2.11* **逆転警報システム** （Piper Make ユーザー向け）
- *2.13* **リアクションゲーム** （Piper Make ユーザー向け）

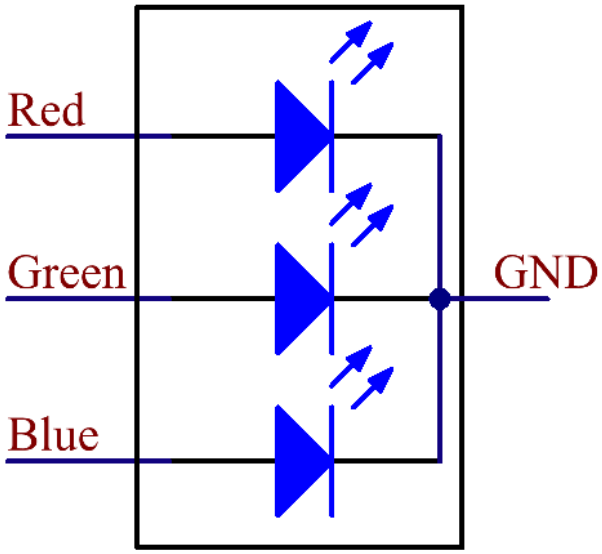
2.11 RGB LED

RGB LED は、さまざまな色の光を放出します。RGB LED は、赤、緑、青の 3 つの LED を透明または半透明のプラスチックケースにパッケージングしています。3 つのピンの入力電圧を変更してそれらを重ね合わせることで、様々な色を表示できます。統計によれば、16,777,216 種類の異なる色を生成することができます。

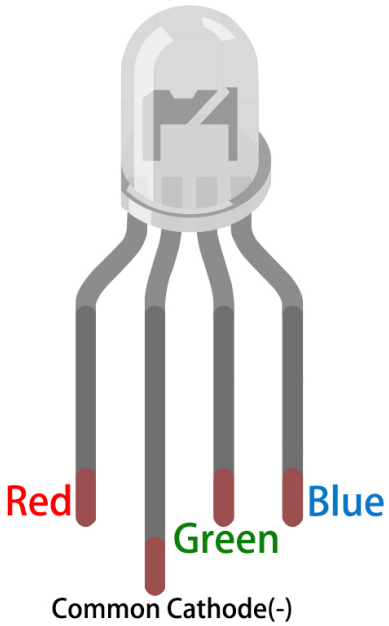


RGB LED には、共通アノードと共通カソードという 2 つのカテゴリーがあります。このキットでは後者が使用されています。 **共通カソード（Common Cathode：CC）** とは、3 つの LED のカソードを接続することを意味します。GND に接続して 3 つのピンを差し込むと、LED は対応する色で点滅します。

回路記号は次の図のようになっています。



RGB LED には 4 つのピンがあります：最も長いピンが共通カソードピンで、通常は GND に接続されます。最も長いピンの隣の左側のピンが赤で、右側の 2 つのピンは緑と青です。



特長

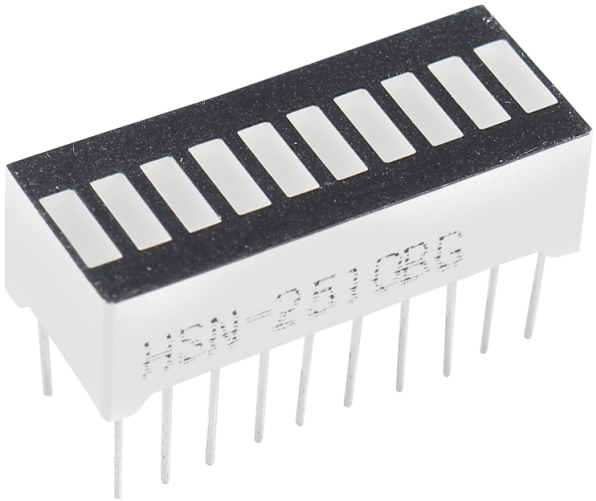
- 色：トリカラー（赤/緑/青）
- 共通カソード
- 5mm クリアラウンドレンズ
- 順方向電圧：赤：DC 2.0 - 2.2V; 青＆緑：DC 3.0 - 3.2V（IF=20mA）
- 0.06 ワット DIP RGB LED

- 輝度が最大 +20% 明るい
- 視野角：30 °

例

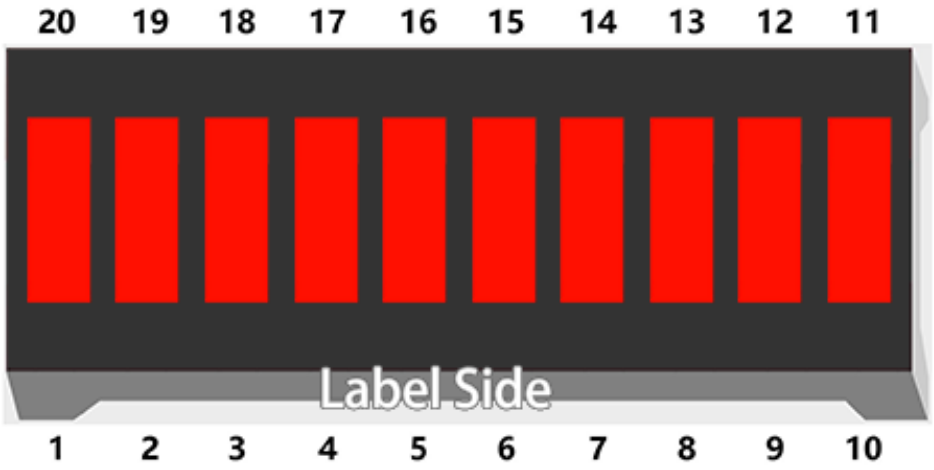
- *2.4* **カラフルな光** （MicroPython ユーザー向け）
- *7.9* **フルーツピアノ** （MicroPython ユーザー向け）
- *2.4 -* **カラフルな光** （Arduino ユーザー向け）
- *2.4* **レインボーライト** （Piper Make ユーザー向け）

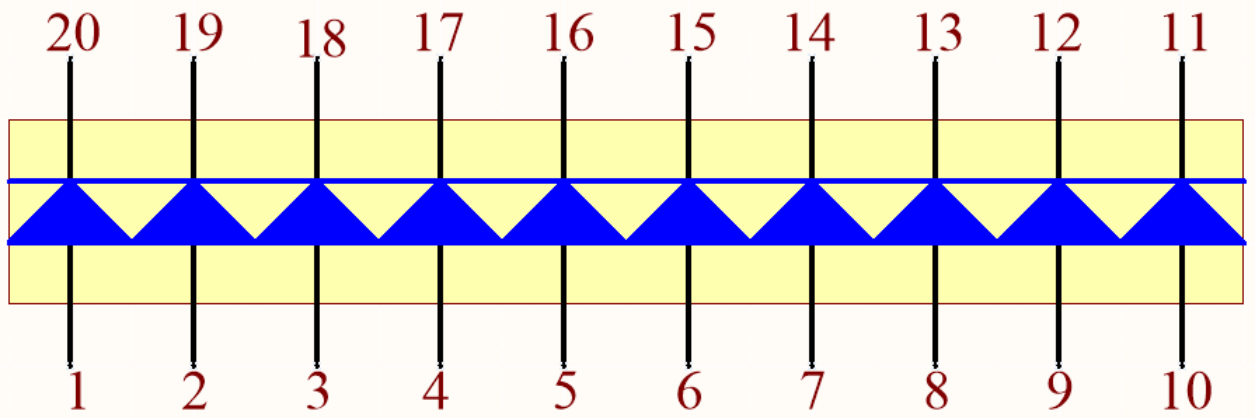
2.12 LED バーグラフ



LED バーグラフは、電子回路やマイクロコントローラに接続するための LED 配列です。基本的に、LED バーグラフを回路に接続するのは、10 個の個々の LED を 10 個の出力ピンに接続するのと同じように簡単です。一般的に、LED バーグラフはバッテリーレベルインジケータ、オーディオ機器、産業用制御パネルなどとして使用できます。他にも多くの用途があります。

以下は、LED バーグラフの内部回路図です。一般的には、ラベルがついている側がアノードで、反対側がカソードです。

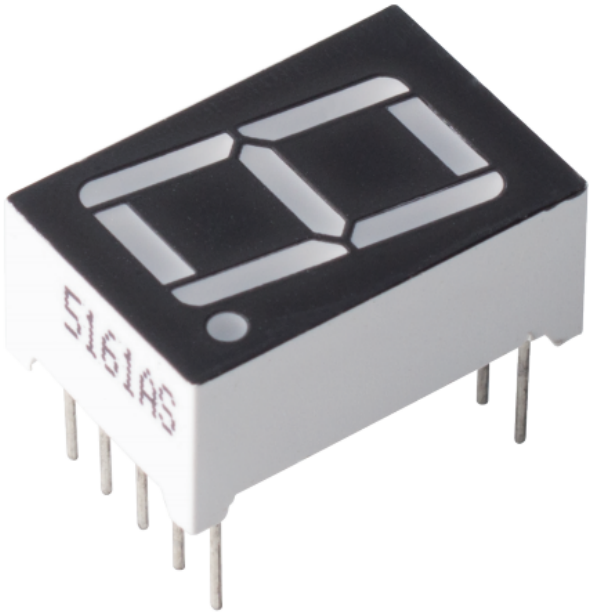




例

- *2.2* **レベルを表示** （MicroPython ユーザー向け）
- *2.2* - **レベル表示** （Arduino ユーザー向け）
- *2.8* **光強度表示装置** （Piper Make ユーザー向け）

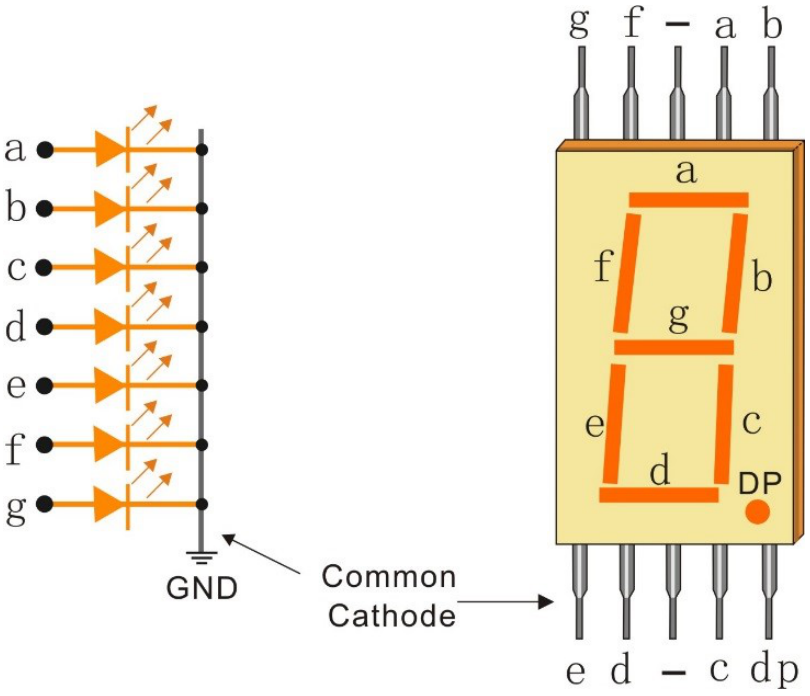
2.13 7 セグメントディスプレイ



7 セグメントディスプレイは、7 つの LED をパッケージした 8 の形状の部品です。各 LED はセグメントと呼ばれ、通電すると表示する数字の一部となります。

ピンの接続方式には、共通カソード（CC）と共通アノード（CA）の 2 種類があります。名前が示すように、CC ディスプレイは 7 つの LED のカソードが接続され、CA ディスプレイは 7 つのセグメントのアノードが接続されます。

このキットでは、共通カソードの 7 セグメントディスプレイを使用しています。以下がその電子記号です。



ディスプレイ内の各 LED は、位置的なセグメントが与えられ、その接続ピンの一つが長方形のプラスチックパッケージから引き出されます。これらの LED ピンは「a」から「g」までのラベルが付けられ、それぞれの個々の LED を表します。他の LED ピンは一緒に接続され、共通のピンを形成します。したがって、LED セグメントの適切なピンに順方向バイアスをかけると、一部のセグメントが明るく、他は暗くなるため、ディスプレイに対応する文字が表示されます。

- セブンセグメントディスプレイ - ウィキペディア

表示コード

7 セグメントディスプレイ（共通カソード）がどのように数字を表示するかを理解するために、以下の表を作成しました。数字は 7 セグメントディスプレイに表示される 0-F の数字であり、(DP) GFEDCBA は対応する LED が 0 または 1 に設定されています。例えば、00111111 は DP と G が 0 に設定され、他は 1 に設定されていることを意味します。そのため、7 セグメントディスプレイには数字の 0 が表示され、HEX コードは 16 進数に対応します。

表 2　グリフコード

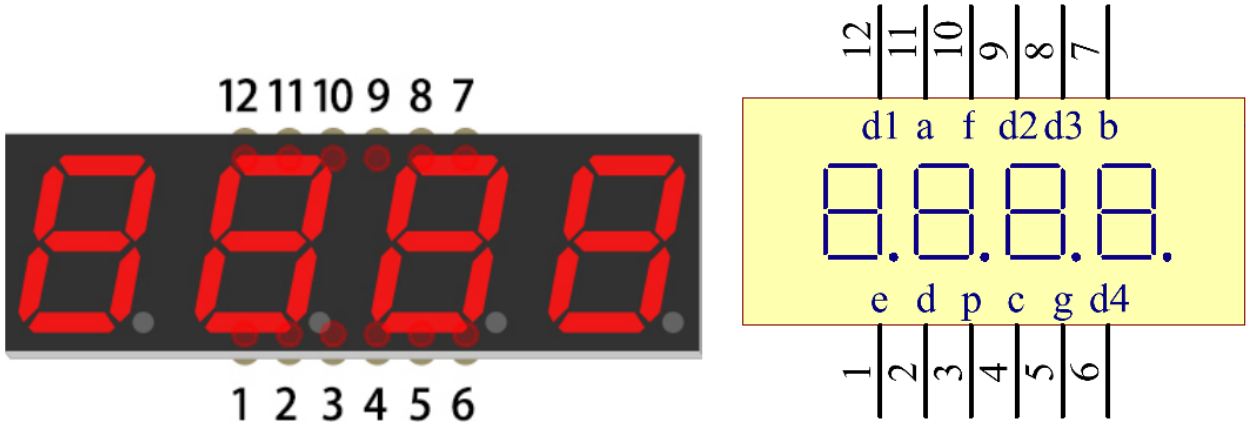
数字	二進数コード	16 進数コード
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f
A	01110111	0x77
B	01111100	0x7c
C	00111001	0x39
D	01011110	0x5e
E	01111001	0x79
F	01110001	0x71

例

- *5.2* **数字表示** (MicroPython ユーザー向け)
- *5.2 -* **数字表示** (Arduino ユーザー向け)

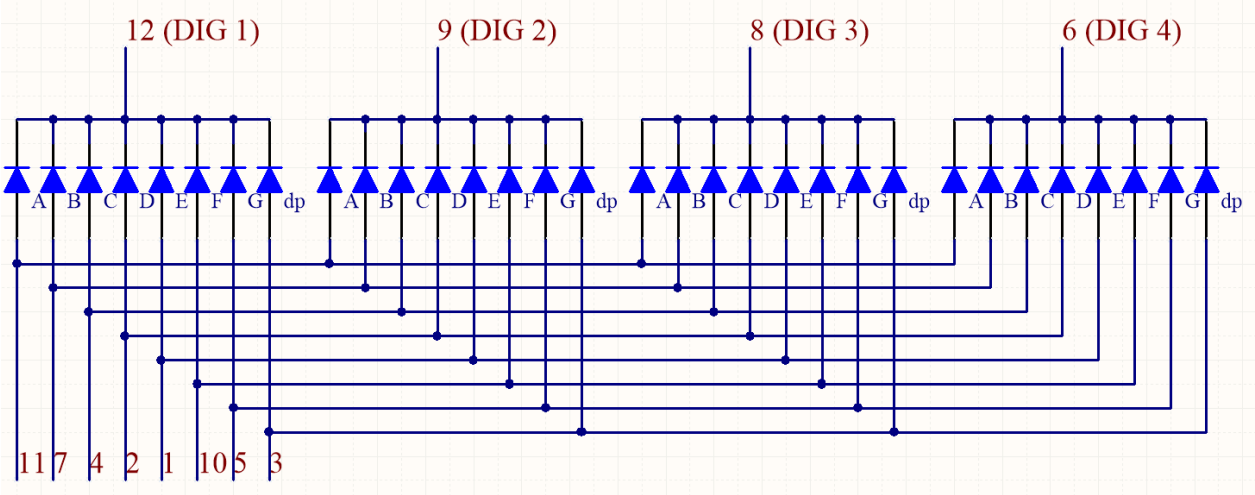
2.14 4 桁の 7 セグメントディスプレイ

4 桁の 7 セグメントディスプレイは、4 つの 7 セグメントディスプレイが連動して動作します。



この 4 桁の 7 セグメントディスプレイは独立して動作します。人間の視覚の残像の原理を利用して、各 7 セグメントの文字を高速でループ表示し、連続する文字列を形成します。

例えば、"1234"がディスプレイに表示される場合、最初の 7 セグメントには"1"が、"234"は表示されません。一定時間後、2 番目の 7 セグメントに"2"が表示され、1 番目、3 番目、4 番目の 7 セグメントは表示されません。このようにして、4 つのディジタルディスプレイが順番に表示されます。このプロセスは非常に短い（通常 5ms）ため、光の残光効果と視覚残留の原理により、4 つの文字が同時に見えます。



表示コード

7 セグメントディスプレイ（共通カソード）がどのように数字を表示するかを理解するために、以下の表を作成しました。数字は 7 セグメントディスプレイに表示される 0-F の数字であり、(DP) GFEDCBA は対応する LED が 0 または 1 に設定されています。例えば、00111111 は DP と G が 0 に設定され、他は 1 に設定されていることを意味します。そのため、7 セグメントディスプレイには数字の 0 が表示され、HEX コードは 16 進数に対応します。

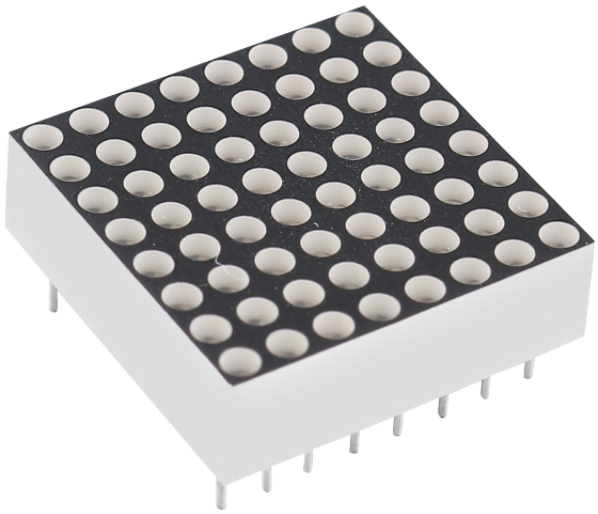
表 3　グリフコード

数字	二進数コード	16 進数コード
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f
A	01110111	0x77
B	01111100	0x7c
C	00111001	0x39
D	01011110	0x5e
E	01111001	0x79
F	01110001	0x71

例

- *5.3* **時間カウンター** (MicroPython ユーザー向け)
- *7.4* **乗客カウンター** (MicroPython ユーザー向け)
- *7.5 GAME - 10* **秒ゲーム** (MicroPython ユーザー向け)
- *7.6* **交通信号機** (MicroPython ユーザー向け)
- *5.3 -* **タイムカウンター** (Arduino ユーザー向け)

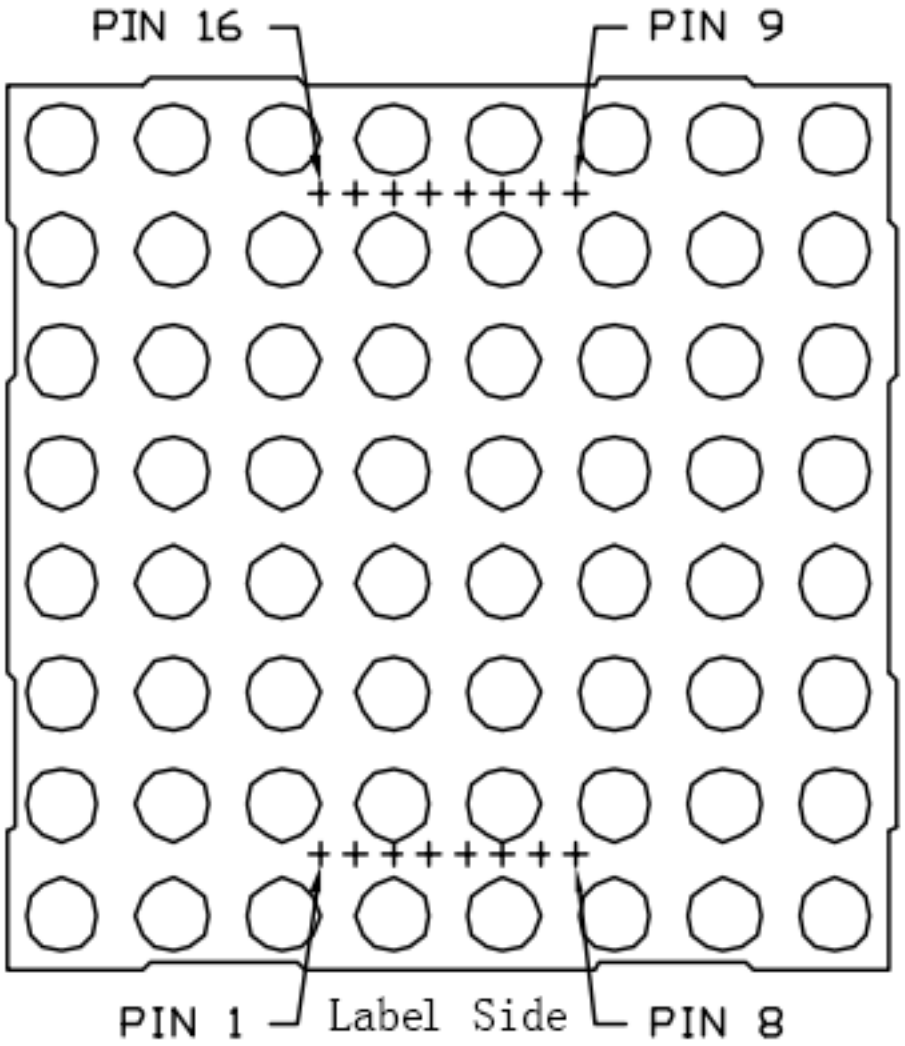
2.15 LED ドットマトリクス



一般的に、LED ドットマトリクスは共通カソード（CC）と共通アノード（CA）の 2 種類に分類されます。外観は非常に似ていますが、内部の構造が異なります。テストで確認できます。このキットでは CA タイプが使用されています。側面には 788BS とラベルが付いています。

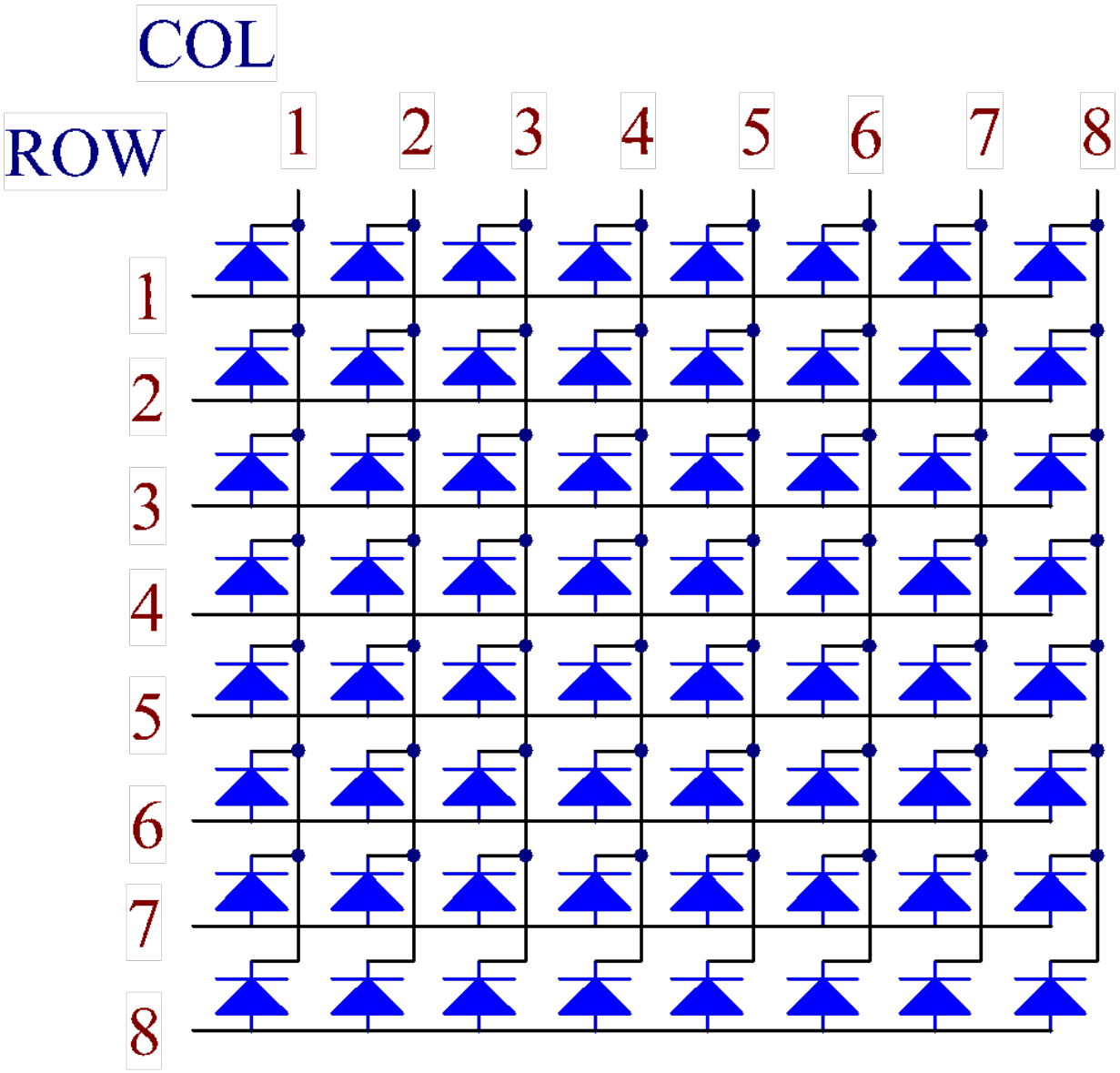
下の図を参照してください。ピンは背面の両端に配置されています。ラベル側を基準にすると、この端のピンは 1-8 番、他方の端は 9-16 番です。

外観：



以下の図は、その内部構造を示しています。CA の LED ドットマトリクスでは、ROW は LED のアノードを、COL はカソードを表します。CC の場合は逆です。共通の点は、どちらのタイプでも、ピン 13, 3, 4, 10, 6, 11, 15, 16 がすべて COL で、ピン 9, 14, 8, 12, 1, 7, 2, 5 がすべて ROW であることです。左上の最初の LED を点灯させたい場合、CA の LED ドットマトリクスでは、ピン 9 を High に、ピン 13 を Low に設定します。CC の場合は、ピン 13 を High に、ピン 9 を Low に設定します。全体の最初の列を点灯させたい場合は、CA では、ピン 13 を Low に、ROW の 9, 14, 8, 12, 1, 7, 2, 5 を High に設定します。CC では、ピン 13 を High に、ROW の 9, 14, 8, 12, 1, 7, 2, 5 を Low に設定します。より理解を深めるために、以下の図を参照してください。

内部ビュー：



上記の行と列に対応するピン番号：

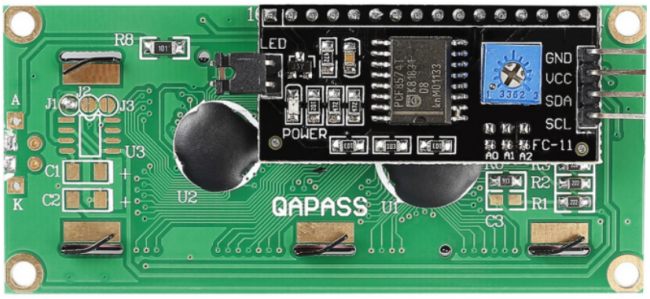
COL	1	2	3	4	5	6	7	8
Pin No.	13	3	4	10	6	11	15	16
ROW	1	2	3	4	5	6	7	8
Pin No.	9	14	8	12	1	7	2	5

さらに、ここでは 2 つの 74HC595 チップが使用されています。1 つは LED ドットマトリクスの行を制御し、もう 1 つは列を制御します。

例

- *5.4 8x8* **ピクセルグラフィックス** (MicroPython ユーザー向け)
- *7.12* **デジタル水平器** (MicroPython ユーザー向け)
- *5.4 - 8x8* **ピクセルグラフィックス** (Arduino ユーザー向け)

2.16 I2C LCD1602



- **GND**: グラウンド
- **VCC**: 電源供給、5V
- **SDA**: シリアルデータライン。プルアップ抵抗を通して VCC に接続。
- **SCL**: シリアルクロックライン。プルアップ抵抗を通して VCC に接続。

LCD やその他のディスプレイは、人間と機械の対話を豊かにするものの、一般的には欠点があります。それは、コントローラに接続すると多くの IO ポートを占有し、その結果、コントローラの他の機能が制限されてしまうことです。

この問題を解決するために、I2C モジュールを備えた LCD1602 が開発されました。I2C モジュールは、内蔵された PCF8574 I2C チップを使用して、I2C シリアルデータを LCD ディスプレイの並列データに変換します。

- PCF8574 データシート

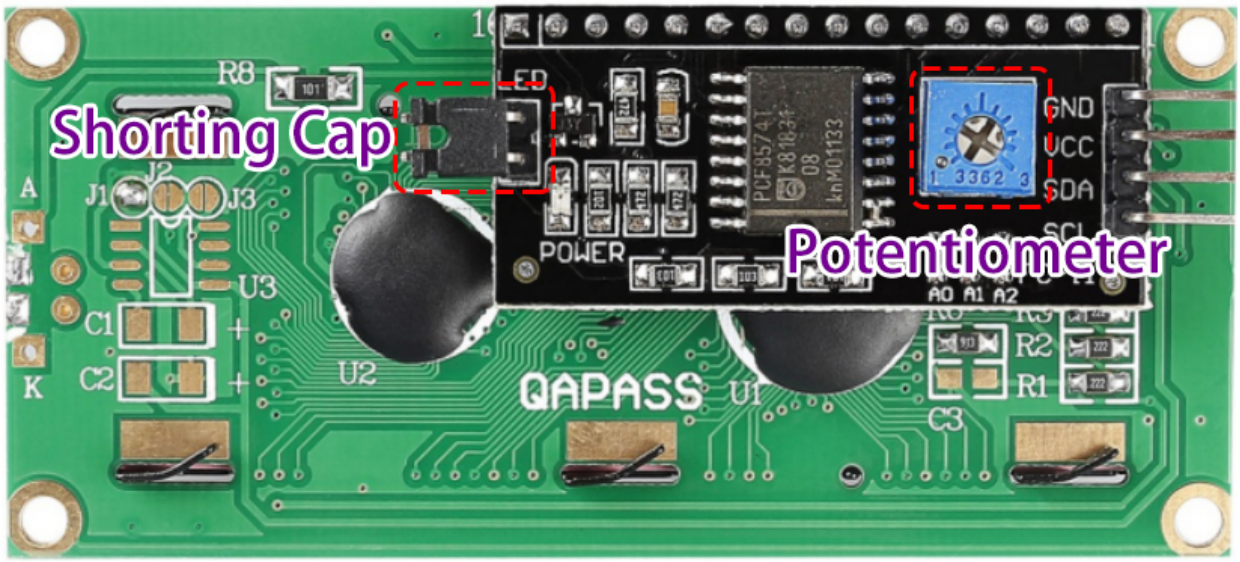
I2C アドレス

デフォルトのアドレスは基本的に 0x27 で、稀に 0x3F である場合もあります。

デフォルトのアドレスが 0x27 の場合を例にすると、A0/A1/A2 パッドを短絡することでデバイスのアドレスを変更できます。デフォルト状態では A0/A1/A2 は 1 で、パッドが短絡されると、A0/A1/A2 は 0 になります。

バックライト/コントラスト

バックライトはジャンパーキャップで有効にでき、ジャンパーキャップを外すとバックライトを無効にできます。背面の青いポテンショメータは、コントラスト（最も明るい白と最も暗い黒の明るさの比率）を調整するために使用されます。

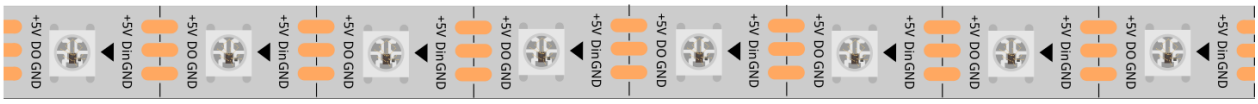


- **ショーティングキャップ**: このキャップでバックライトを有効にでき、キャップを外すとバックライトを無効にできます。
- **ポテンショメータ**: これはコントラスト（表示されるテキストの明瞭度）を調整するために使用され、時計回りに回すと増加し、反時計回りに回すと減少します。

例

- *3.4* **液晶ディスプレイ** （MicroPython ユーザー向け）
- *7.2* **室温計** （MicroPython ユーザー向け）
- *7.7* **数字当てゲーム** （MicroPython ユーザー向け）
- *3.4 -* **液晶ディスプレイ** （Arduino ユーザー向け）

2.17 WS2812 RGB 8 LED ストリップ



WS2812 RGB 8 LED ストリップは、8 個の RGB LED で構成されています。全ての LED を制御するために必要なのは一つのピンだけです。各 RGB LED には独立して制御できる WS2812 チップが搭載されています。256 階調の明るさ表示と、16,777,216 色の完全なトゥルーカラー表示が可能です。さらに、ピクセルには、インテリジェントなデジタルインターフェースデータラッチ信号成形アンプ駆動回路が組み込まれており、ピクセルポイントの色高さが一貫していることを効果的に確保するための信号成形回路が内蔵されています。

柔軟であり、自由にドッキング、曲げ、カットが可能です。裏面には粘着テープが装備されており、凹凸のある面にも自由に固定でき、狭いスペースにも取り付け可能です。

特長

- 動作電圧: DC5V
- IC: 一つの IC で一つの RGB LED を駆動
- 消費電力: 1 つの LED あたり 0.3w
- 動作温度: -15-50 ℃
- 色: フルカラー RGB
- RGB タイプ: 5050RGB（内蔵 IC WS2812B）
- ストリップの厚み: 2mm
- 各 LED は個別に制御可能

WS2812B の紹介

- WS2812B データシート

WS2812B は、制御回路と RGB チップが 5050 コンポーネントのパッケージ内に統合されたインテリジェント制御 LED 光源です。内部には、インテリジェントなデジタルポートデータラッチと信号再成形増幅駆動回路が含まれています。また、高精度な内部オシレータと 12V 電圧プログラム可能な定電流制御部分も内蔵しており、ピクセルポイントの光の色高さが一貫していることを効果的に確保します。

データ転送プロトコルは、単一の NZR 通信モードを使用します。ピクセルが電源オンリセット後、DIN ポートはコントローラからデータを受信し、最初のピクセルは初期の 24 ビットデータを収集して内部データラッチに送信します。その他のデータは、内部信号再成形増幅回路によって再成形され、DO ポートを通じて次のカスケードピクセルに送信されます。各ピクセルの伝送後、信号は 24 ビット減少します。ピクセルは、オートリシェイピング伝送テクノロジーを採用しており、ピクセルのカスケード数は信号転送の速度にのみ依存しています。

低駆動電圧の LED は、環境保護とエネルギー節約、高輝度、大きな散乱角、良好な一貫性、低電力、長寿命などの利点があります。制御チップが LED 上に統合されているため、回路がよりシンプルになり、容積が小さく、取り付けが便利です。

例

- *3.3 RGB LED* **ストリップ** （MicroPython ユーザー用）
- *7.8 RFID* **音楽プレーヤー** （MicroPython ユーザー用）
- *3.3 WS2812 RGB* **ストリップ** （Arduino ユーザー用）
- *2.10* **流れる** *LED* （Piper Make ユーザー用）

サウンド

2.18 ブザー

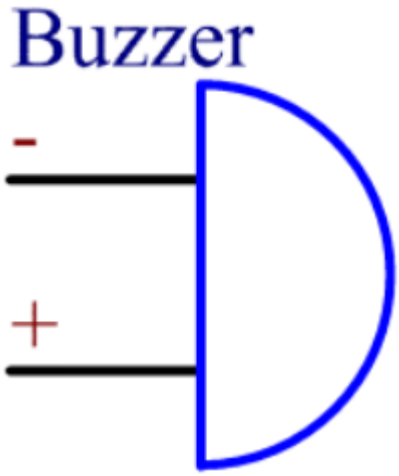
DC 電源で駆動されるブザーは、一体型の電子ブザーとして、コンピュータ、プリンタ、コピー機、アラーム、電子玩具、自動車用電子機器、電話、タイマーなど、さまざまな電子製品や音声機器で広く使用されています。

ブザーは、アクティブ型とパッシブ型に分類されます（以下の画像を参照）。ブザーをピンが上を向くように回転させると、緑色の基板を持つブザーはパッシブ型で、黒いテープで覆われたものはアクティブ型です。

アクティブ型とパッシブ型のブザーの違い：

アクティブ型ブザーは内蔵の振動源を持っているため、通電すると音を出します。一方で、パッシブ型ブザーはそのような振動源を持っていないので、DC 信号を使用しても音は出ません。代わりに、2K から 5K の周波数の矩形波を使用して駆動する必要があります。アクティブ型ブザーは、多くの内蔵振動回路があるため、パッシブ型よりも高価なことがよくあります。

以下はブザーの電気記号です。プラスとマイナスの極性を持つ 2 つのピンがあります。表面に + が表示されている方が陽極で、もう一方は陰極です。



ブザーのピンを確認すると、長い方が陽極で短い方が陰極です。接続する際にそれらを混同しないでください。そうしないと、ブザーは音を出さなくなります。

Buzzer - Wikipedia

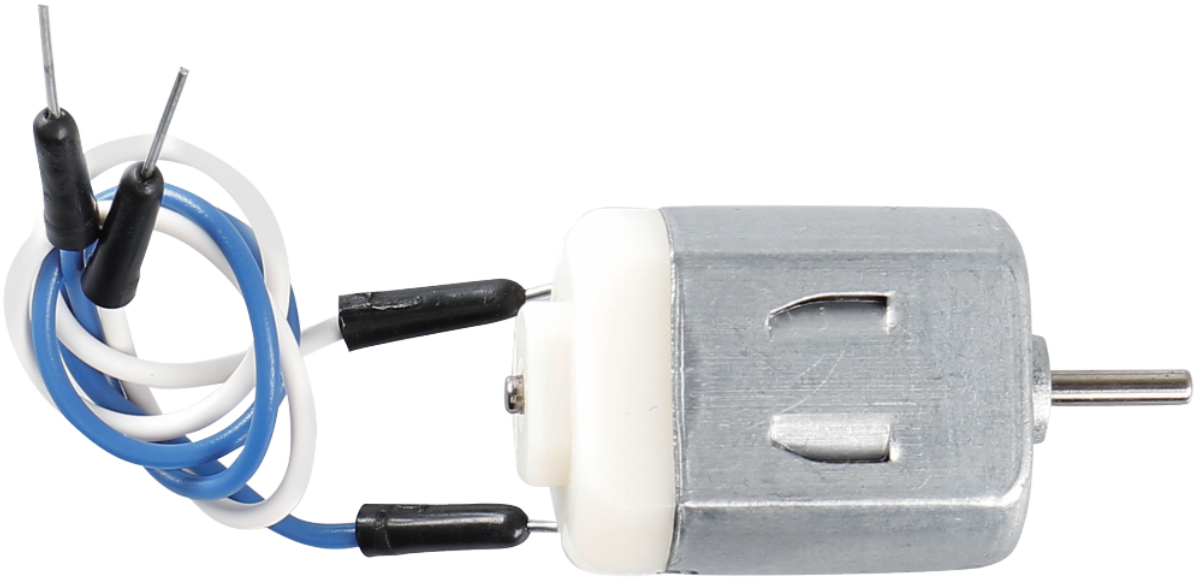
例

- *3.1* **ビープ音** (MicroPython ユーザー向け)
- *3.2* **カスタムトーン** (MicroPython ユーザー向け)
- *7.1* **光センサー・テルミン** (MicroPython ユーザー向け)

- *7.3* **警報サイレンランプ** (MicroPython ユーザー向け)
- *7.8 RFID* **音楽プレーヤー** (MicroPython ユーザー向け)
- *7.9* **フルーツピアノ** (MicroPython ユーザー向け)
- *7.10* **バックアップ支援** (MicroPython ユーザー向け)
- *3.1 -* **ビープ音** (Arduino ユーザー向け)
- *3.2 -* **カスタムトーン** (Arduino ユーザー向け)
- *2.3* **サービスベル** (Piper Make ユーザー向け)
- *2.11* **逆転警報システム** (Piper Make ユーザー向け)
- *2.13* **リアクションゲーム** (Piper Make ユーザー向け)

アクチュエータ

2.19 DC モーター



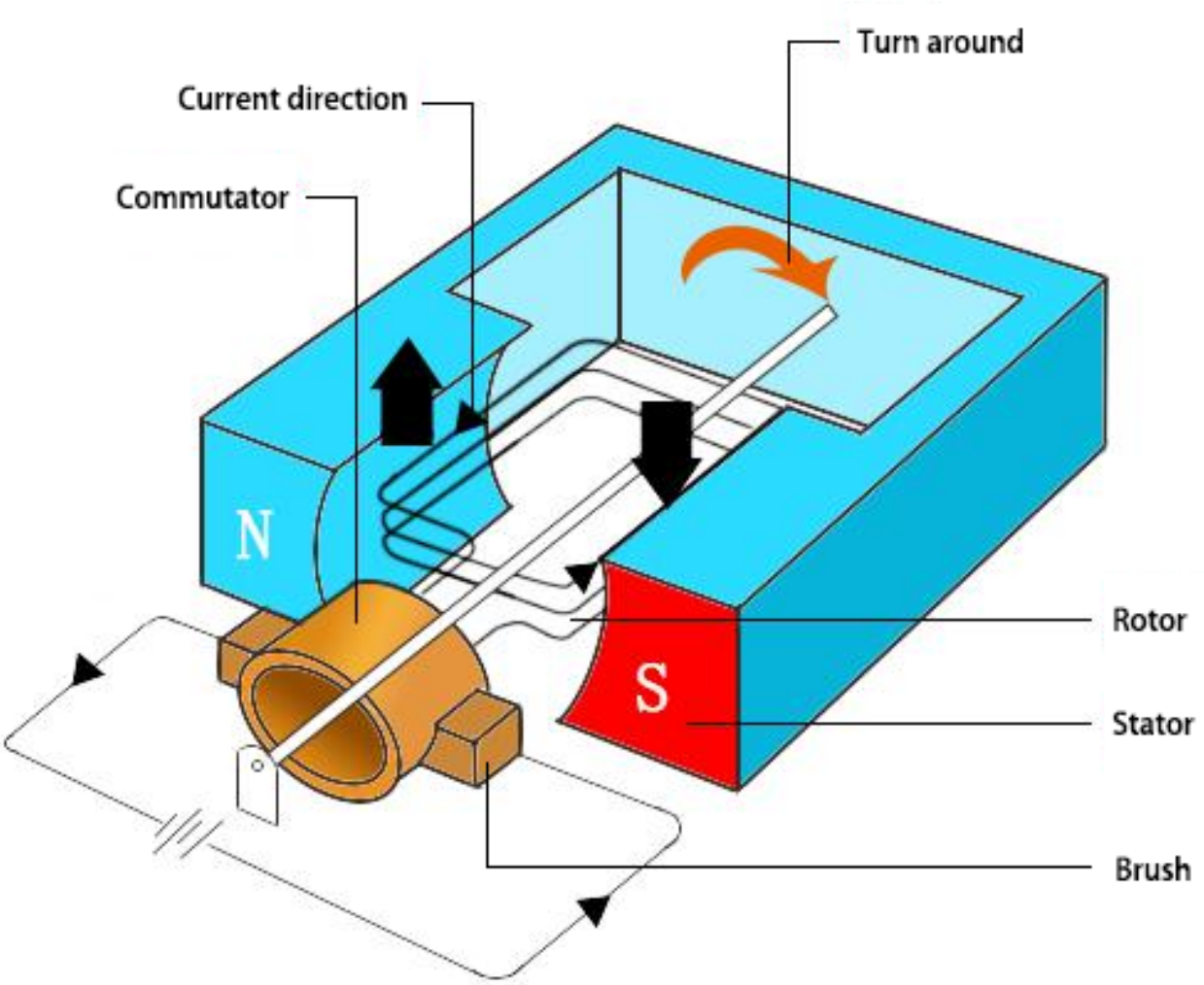
これは 3V の DC モーターです。2 つの端子のうち、一方に高レベル、もう一方に低レベルを与えると回転します。

- **サイズ**：25*20*15MM
- **動作電圧**：1-6V
- **無負荷電流**（3V）：70mA
- **無負荷回転数**（3V）：13000RPM

- **ストール電流**（3V）：800mA
- **シャフト径**：2mm

直流（DC）モーターは、電気エネルギーを機械エネルギーに変換する連続アクチュエーターです。DC モーターは、連続した角度回転を生成することで、回転ポンプ、ファン、圧縮機、インペラーなどの装置を動作させます。

DC モーターは、モーターの固定部である **ステーター** と、動作を生成するために回転するモーターの内部部分である **ローター**（または DC モーターの **アーマチュア**）の 2 つの部分で構成されています。動作を生成する鍵は、アーマチュアを永久磁石の磁場内（北極から南極にかけての磁場）に配置することです。磁場と移動する帯電粒子（電流を帯びたワイヤーが磁場を生成）との相互作用により、アーマチュアを回転させるトルクが生成されます。



電流は、バッテリーの正極から回路を通って銅製のブラシに流れ、コミュテーターを通ってアーマチュアに到達します。しかし、コミュテーターには 2 つの隙間があるため、この流れは 1 回の完全な回転の途中で逆転します。この連続した逆転により、バッテリーからの DC 電力は実質的に AC に変換され、アーマチュアは適切な方向とタイミングでトルクを経験し、回転が維持されます。

- DC Motor - MagLab

- Fleming's left-hand rule for motors - Wikipedia

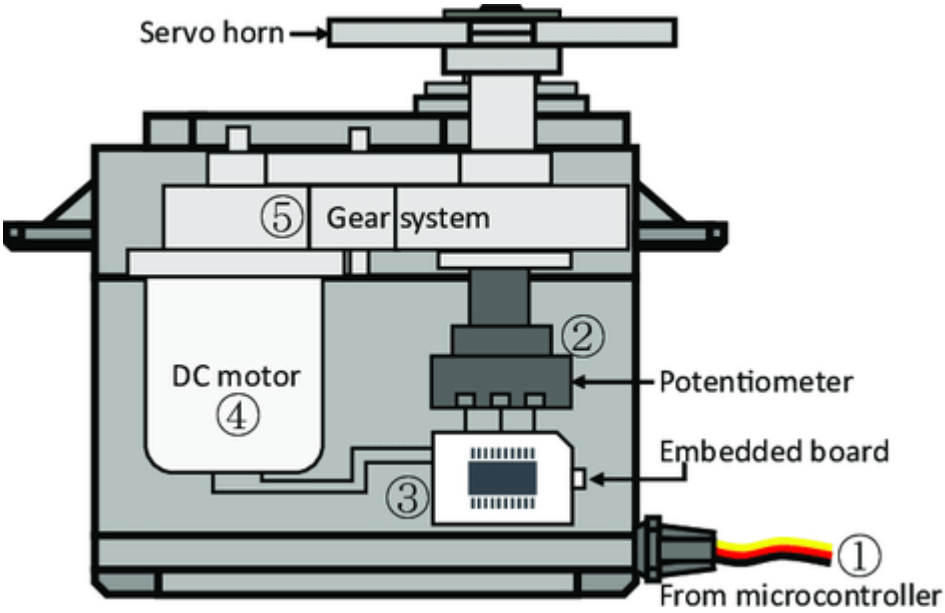
例

- *3.5* **スモールファン**（MicroPython ユーザー向け）
- *3.5 -* **小型ファン**（Arduino ユーザー向け）
- *2.12* **スマートファン**（Piper Make ユーザー向け）

2.20 サーボ

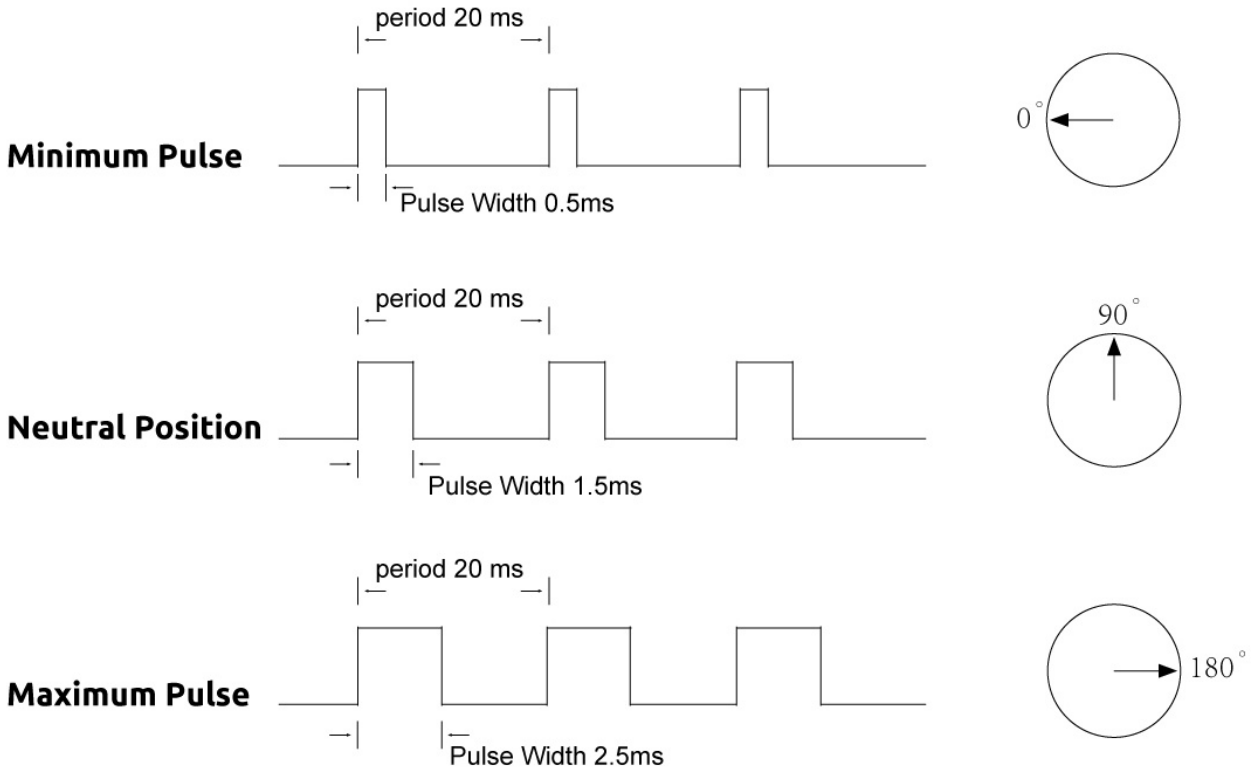
一般的なサーボは、ケース、軸、歯車システム、ポテンショメーター、直流モーター、および組み込みボードで構成されています。

動作原理は次の通りです：マイクロコントローラが PWM 信号をサーボに送信し、サーボ内の組み込みボードが信号ピンを介してこれらの信号を受け取り、内部のモーターを制御して回転させます。その結果、モーターは歯車システムを駆動し、減速後に軸を動かします。サーボの軸とポテンショメーターは接続されています。軸が回転すると、ポテンショメーターも駆動され、組み込みボードに電圧信号を出力します。ボードは現在の位置に基づいて回転の方向と速度を決定し、定義された正確な位置で停止してその位置を保ちます。



角度は、制御ワイヤーに適用されるパルスの持続時間によって決定されます。これをパルス幅変調（PWM）と呼

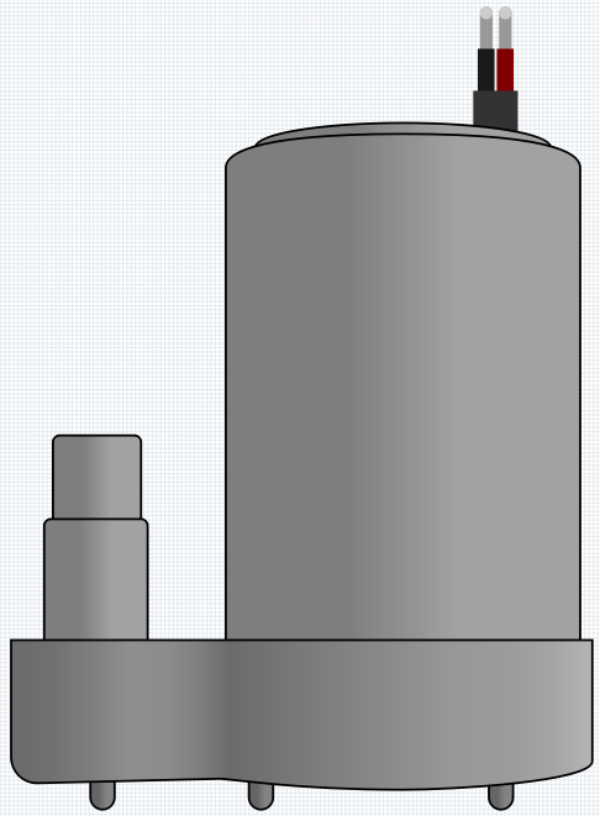
びます。サーボは 20ms ごとにパルスを見ることを期待しています。パルスの長さがモーターがどれだけ回転するかを決定します。例えば、1.5ms のパルスは、モーターを 90 度の位置（中立位置）に回転させます。1.5ms より短いパルスがサーボに送信されると、サーボは位置に回転し、出力軸を中立点から反時計回りに数度保持します。パルスが 1.5ms よりも長い場合は、逆の現象が発生します。サーボが回転するためのパルスの最小幅と最大幅は、各サーボの機能に依存します。一般的には、最小パルス幅は約 0.5ms、最大パルス幅は 2.5ms になります。



例

- *3.7* **サーボの振動** （MicroPython ユーザー向け）
- *7.11* **体感コントローラー** （MicroPython ユーザー向け）
- *3.7 -* **サーボの揺れ動き** （Arduino ユーザー向け）
- *2.6* **スマートウォータータンク** （Piper Make ユーザー向け）
- *2.7* **スイングサーボ** （Piper Make ユーザー向け）
- *2.9* **招き猫プロジェクト** （Piper Make ユーザー向け）

2.21 DC ウォーターポンプ



このポンプは基本的には DC モーターとして動作し、動作電圧は 3V、電流は 100mA です。電源を接続すると、ポンプはプラスチックケーシングの底から水を吸い上げ、出口パイプから水を排出します。このポンプは、正常に動作するためには常に水に浸漬させておく必要があります。極性を逆にしても、それによって吸水装置にはなりません。逆に、水はただ排出されるだけです！

この潜水ポンプは非常に使い勝手が良いため、初心者にとっては噴水や植物の水やりプロジェクトを作成するのに非常に適しています。

特長

- **動作電圧範囲**: DC 3 ~ 4.5V
- **動作電流**: 120 ~ 180mA
- **消費電力**: 0.36 ~ 0.91W
- **最大揚水高**: 0.35 ~ 0.55M
- **最大流量**: 80 ~ 100 L/H
- **連続作動寿命**: 100 時間
- **防水等級**: IP68

- **駆動方式**: DC, 磁気駆動
- **素材**: エンジニアリングプラスチック
- **出口外径**: 7.8 mm
- **出口内径**: 6.5 mm
- このポンプは潜水ポンプであり、そのように使用すべきです。水中でない状態で動作させると、過熱する危険性が高まります。
- 25cm のオス型ワイヤーが付属しているため、ブレッドボードに簡単に挿入できます。

例

- *3.6* **ポンピング**（MicroPython ユーザー向け）
- *3.6 -* **ポンピング**（Arduino ユーザー向け）

2.22 リレー

リレーは入力信号に応じて 2 つ以上のポイントやデバイス間の接続を提供する装置です。言い換えれば、リレーはデバイスが AC および DC の両方で動作する可能性があるため、コントローラとデバイスとの間に絶縁を提供します。一方、マイクロコントローラからの信号は DC であるため、このギャップを埋めるためにリレーが必要です。リレーは、小さな電気信号で大量の電流や電圧を制御する必要がある場合に非常に便利です。

リレーには 5 つの部分があります：

電磁石 - 鉄の芯にコイルの線が巻かれています。電気が通ると磁気になります。したがって、これを電磁石と呼びます。

アーマチュア - 可動式の磁気ストリップはアーマチュアとして知られています。電流が流れると、コイルがそれによってエネルギーを持ち、磁場が生成されます。これを用いて通常開（N/O）または通常閉（N/C）の接点を作成または解除します。アーマチュアは、直流（DC）および交流（AC）で動作させることができます。

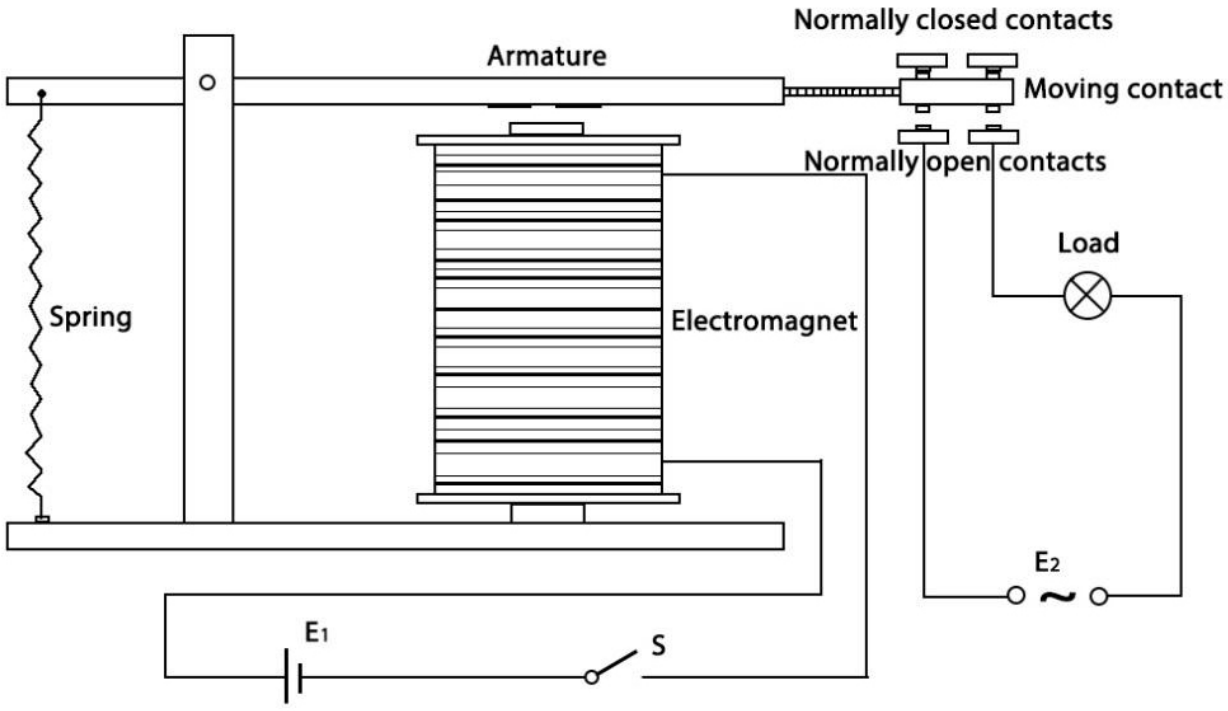
バネ - 電磁石のコイルに電流が流れていないとき、バネはアーマチュアを引っ張って回路を完成させません。

接点セット - 2 つの接点ポイントがあります：

- 通常開 - リレーが作動しているときに接続され、非作動時には切断されます。
- 通常閉 - リレーが作動しているときには接続されず、非作動時には接続されます。

成形フレーム - リレーは保護のためにプラスチックで覆われています。

リレーの動作原理は単純です。リレーに電源が供給されると、制御コイルを通じて電流が流れ始めます。その結果、電磁石がエネルギーを持ち始めます。次に、アーマチュアはコイルに引きつけられ、通常開の接点と接触するように動く接触部が下がります。その結果、負荷回路がエネルギーを持ちます。回路を切る場合も同様で、動く接触部はバネの力で通常閉の接点に引き上げられます。このようにして、リレーのオンとオフが負荷回路の状態を制御します。



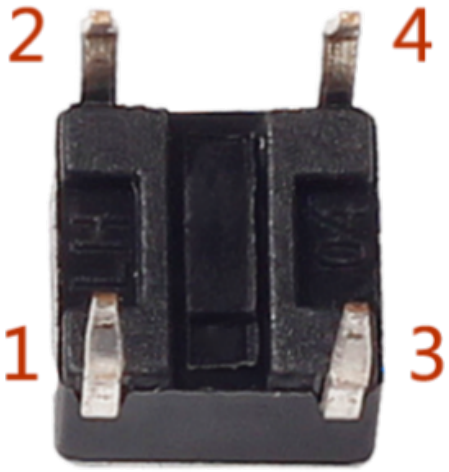
- リレー - Wikipedia

例

- *2.16* **他の回路を制御する** （MicroPython ユーザー向け）
- *2.16* - **別の回路を制御する** （Arduino ユーザー向け）

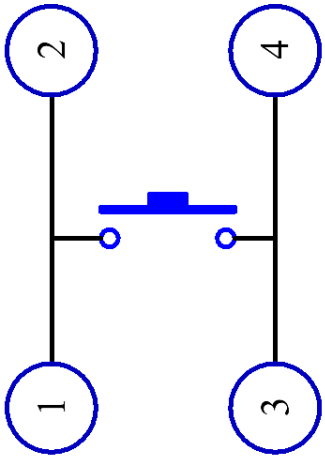
コントローラー

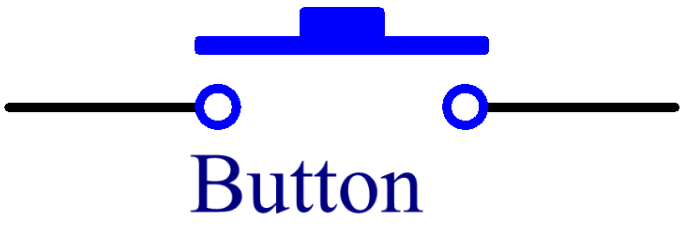
2.23 ボタン



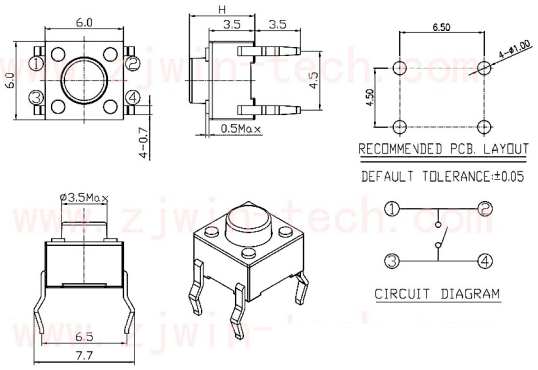
ボタンは、電子機器を制御するためによく使用される一般的な部品です。通常は、回路をつなぐまたは切るためのスイッチとして使用されます。ボタンにはさまざまなサイズや形状がありますが、ここで使用されるのは次の写真に示すような 6mm ミニボタンです。ピン 1 はピン 2 に、ピン 3 はピン 4 に接続されています。したがって、ピン 1 とピン 2 のいずれかをピン 3 またはピン 4 に接続するだけでよいです。

以下はボタンの内部構造です。右下の記号は、通常、回路内でボタンを表すために使用されます。





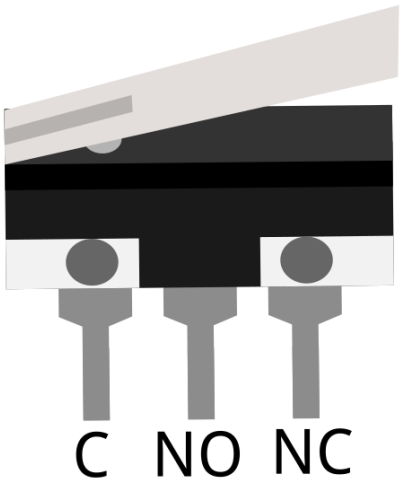
ピン 1 がピン 2 に、ピン 3 がピン 4 に接続されているため、ボタンが押されると、4 つのピンが接続され、回路が閉じます。



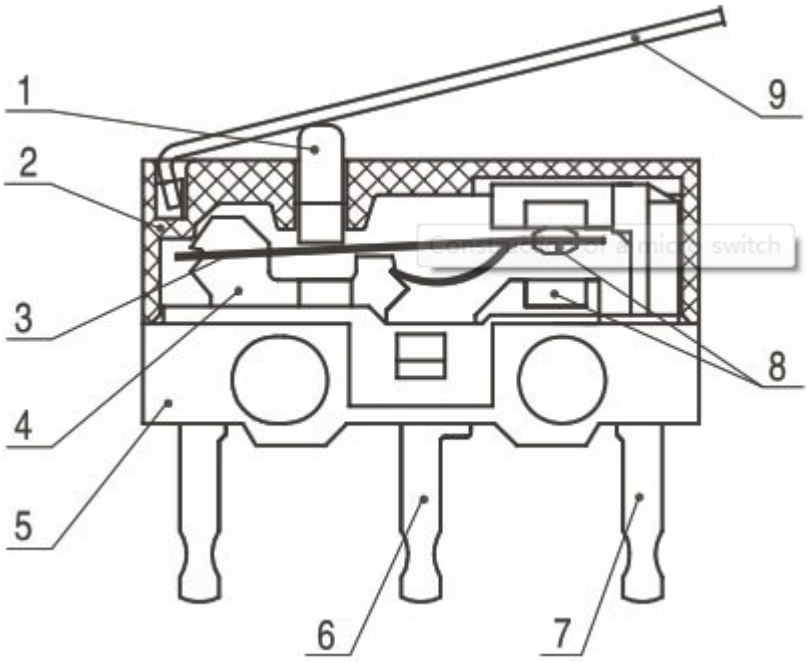
例

- *2.5* **ボタンの値を読み取る** (MicroPython ユーザー向け)
- *2.5 -* **ボタン値の読み取り** (Arduino ユーザー向け)
- *2.2* **ボタン** (Piper Make ユーザー向け)
- *2.4* **レインボーライト** (Piper Make ユーザー向け)
- *2.5* **ドラムキット** (Piper Make ユーザー向け)
- *2.13* **リアクションゲーム** (Piper Make ユーザー向け)

2.24 マイクロスイッチ



マイクロスイッチの構造は非常にシンプルです。主要な部品は以下の通りです。



- 1. プランジャー（アクチュエータ）
- 2. カバー
- 3. 可動部品
- 4. サポート
- 5. ケース
- 6. NO 端子：通常開放
- 7. NC 端子：通常閉鎖
- 8. コンタクト
- 9. 可動アーム

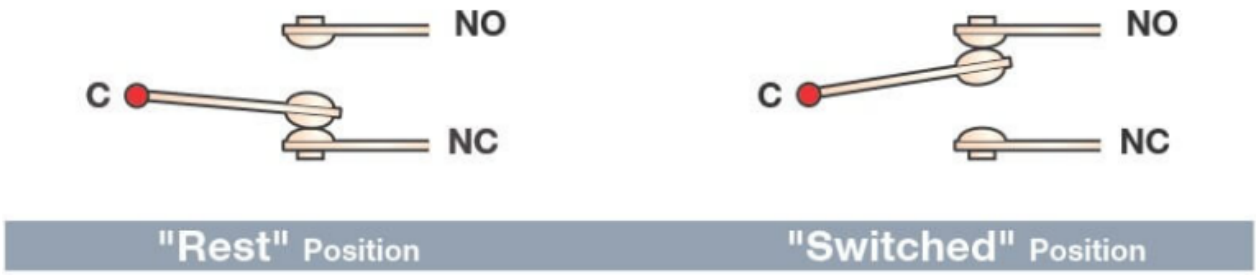
マイクロスイッチが物体と物理的に接触すると、その接点が位置を変えます。基本的な動作原理は以下の通りです。

プランジャーが解放されているか、または静止している場合。

- 通常閉鎖回路は電流を流すことができます。
- 通常開放回路は電気的に絶縁されています。

プランジャーが押されたり、切り替えられたりすると。

- 通常閉鎖回路は開いています。
- 通常開放回路は閉じています。



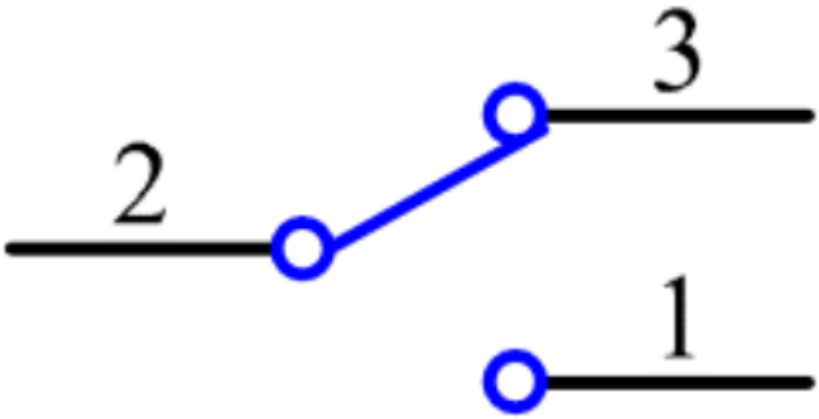
例

- *2.8* **やさしく押して**（MicroPython ユーザー向け）
- *2.8 -* **ソフトに押してください**（Arduino ユーザー向け）
- *2.3* **サービスベル**（Piper Make ユーザー向け）

2.25 スライドスイッチ

スライドスイッチとは、その名の通り、スイッチのバーをスライドさせて回路を接続または遮断し、さらに回路を切り替えるためのものです。一般的に使用されるタイプには SPDT、SPTT、DPDT、DPTT などがあります。スライドスイッチは低電圧回路でよく使用されます。柔軟性と安定性があり、電子機器や電子玩具に幅広く応用されています。動作原理：中央のピンを固定ピンとして設定します。スライドを左に引くと、左側の 2 つのピンが接続されます。スライドを右に引くと、右側の 2 つのピンが接続されます。したがって、回路を接続または遮断するスイッチとして機能します。下図を参照してください：

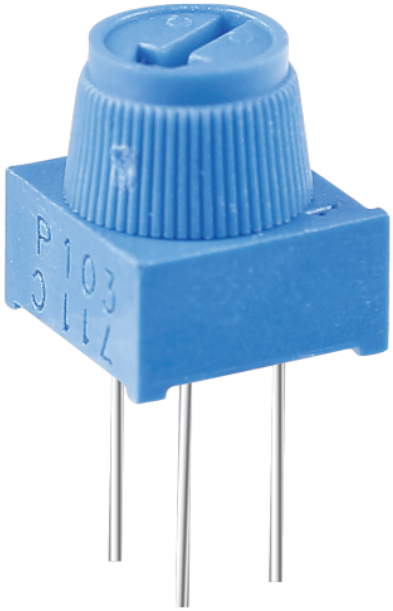
スライドスイッチの回路記号は下のようになっています。図中の pin2 は、中央のピンを指しています。



例

- *2.7* **左右切り替え**（MicroPython ユーザー向け）
- *7.3* **警報サイレンランプ**（MicroPython ユーザー向け）
- *2.7 -* **左右トグルスイッチ**（Arduino ユーザー向け）
- *2.5* **ドラムキット**（Piper Make ユーザー向け）

2.26 ポテンショメーター



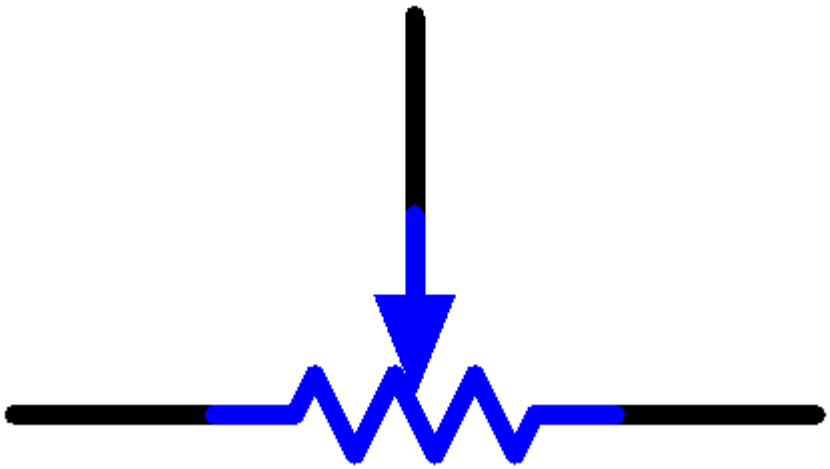
ポテンショメーターは、3 つの端子を持つ可変抵抗器です。抵抗値は一定の規則に従って調整することができます。

ポテンショメーターにはさまざまな形状、サイズ、値がありますが、以下の共通点があります。

- 3 つの端子（または接続点）があります。

- 抵抗値を変えるために動かすことができるノブ、ねじ、またはスライダーがあります。
- 中央の端子と外側のどちらか一方の端子との間の抵抗値は、ノブ、ねじ、またはスライダーが動かされると 0 Ωからポテンショメーターの最大抵抗値まで変わります。

以下はポテンショメーターの回路記号です。



ポテンショメーターが回路内で果たす機能は以下の通りです：

1. 電圧分割器として機能

   ポテンショメーターは連続的に調整可能な抵抗器です。ポテンショメーターの軸やスライドハンドルを調整すると、可動接点が抵抗素子上を滑ります。この時点で、ポテンショメーターに印加された電圧と、可動アームが回転した角度または移動した距離に応じて電圧が出力されます。

2. リオスタットとして機能

   ポテンショメーターをリオスタットとして使用する場合、中央のピンと他の 2 つのピンのうちの 1 つを回路に接続します。これにより、可動接点の移動範囲内で、抵抗値が滑らかかつ連続的に変化します。

3. 電流コントローラーとして機能

   ポテンショメーターが電流コントローラーとして機能する場合、スライド接触端子は出力端子の 1 つとして接続されなければなりません。

ポテンショメーターについてもっと知りたい場合は、参照してください： ポテンショメーター - Wikipedia

例

- *2.11* **ノブを回してみよう** （MicroPython ユーザー向け）
- *2.11 -* **ノブを回す** （Arduino ユーザー向け）
- *2.7* **スイングサーボ** （Piper Make ユーザー向け）

2.27 赤外線レシーバー

2.27.1 IR レシーバー

- S: シグナル出力
- +: VCC
- -: GND

赤外線レシーバーは、赤外線信号を受信し、TTL レベルと互換性のある信号を独立して出力できるコンポーネントです。サイズは通常のプラスチックパッケージのトランジスタと同じで、あらゆる種類の赤外線リモコンや赤外線伝送に適しています。

赤外線（IR）通信は、低コストで使いやすい無線通信技術として広く用いられています。赤外線は可視光線よりもわずかに波長が長いため、人間の目には見えません—無線通信に最適です。赤外線通信で一般的に用いられる変調方式は、38KHz 変調です。

- HX1838 IR レシーバーセンサー採用、高感度
- リモコンとして使用可能

- 電源供給: 3.3~5V
- インターフェース: デジタル
- 変調周波数: 38Khz

2.27.2 リモートコントロール

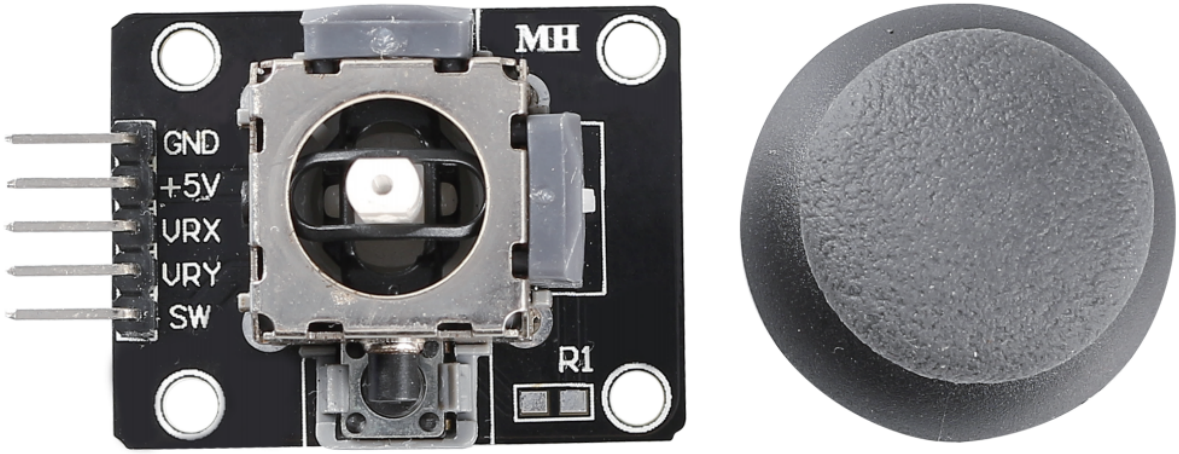
これは、21 の機能ボタンと最大 8 メートルの送信距離を持つミニ薄型赤外線無線リモコンです。子供部屋で幅広いデバイスを操作するのに適しています。

- サイズ: 85x39x6mm
- リモコン範囲: 8-10m
- 電池: 3V ボタン型リチウムマンガン電池
- 赤外線キャリア周波数: 38KHz
- 表面貼り付け材: 0.125mm PET
- 有効寿命: 20,000 回以上

例

- *6.4 IR* **リモートコントロール**（MicroPython ユーザー向け）
- *6.4 - IR* **リモートコントロール**（Arduino ユーザー向け）

2.28 ジョイスティックモジュール

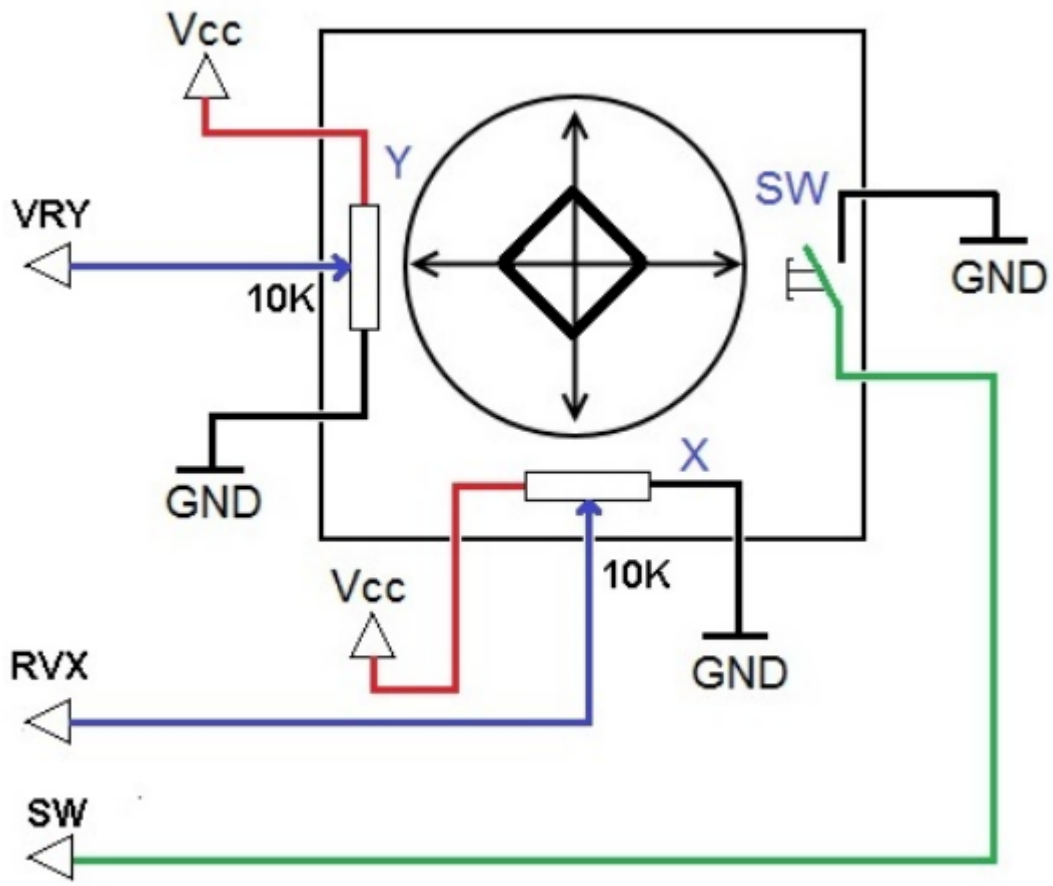


ジョイスティックの基本的なアイデアは、スティックの動きをコンピュータが処理できる電子情報に変換することです。

コンピュータに全範囲の動きを伝えるために、ジョイスティックはスティックの位置を二つの軸で測定する必要があります。それは X 軸（左から右へ）と Y 軸（上から下へ）です。基本的な幾何学と同様に、X-Y 座標はスティックの位置を正確に特定します。

スティックの位置を特定するために、ジョイスティック制御システムは各軸の位置を単純に監視します。従来のアナログジョイスティック設計は、これを二つの可変抵抗器、すなわちポテンショメータで行います。

ジョイスティックには、ジョイスティックが押されたときに作動するデジタル入力もあります。



- Joystick - Wikipedia

例

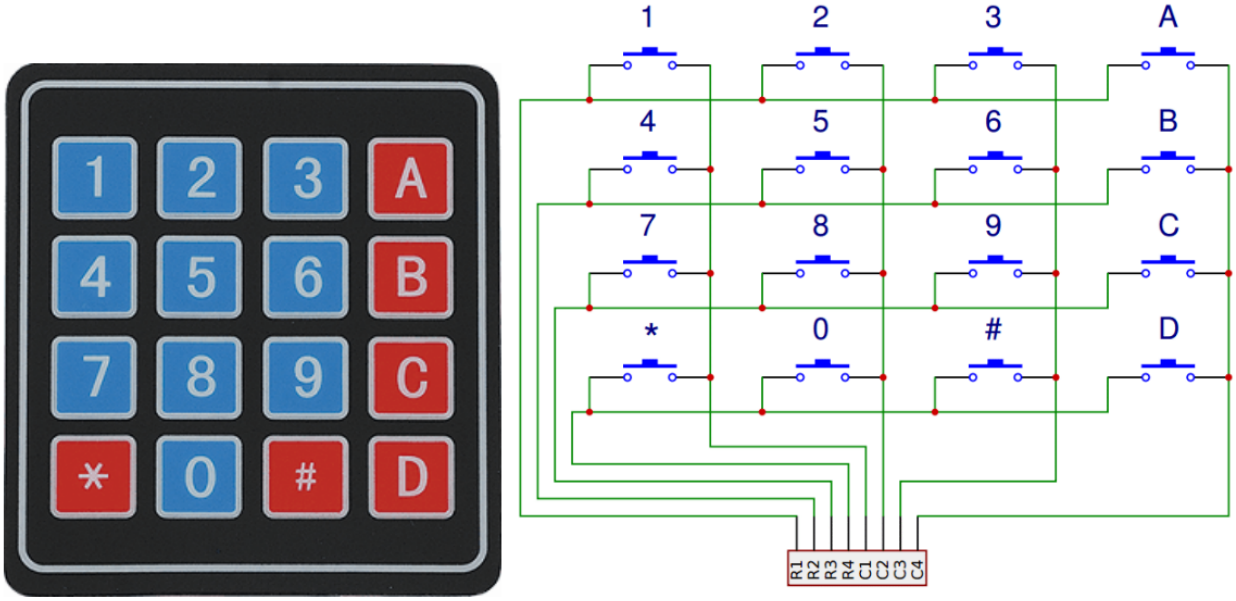
- *4.1* **ジョイスティックの切り替え**（MicroPython ユーザー向け）
- *4.1* - **ジョイスティックの切り替え**（Arduino ユーザー向け）

2.29 4x4 キーパッド

マイクロコントローラーシステムでは、電子コードロックや電話のキーパッドなど、多くのキーを使用する場合があります。通常、最低でも 12 から 16 のキーがあり、これは一般的にマトリックスキーパッドとして使用されます。

マトリックスキーパッドは、行キーパッドとも呼ばれ、4 つの I/O ラインを行として、4 つの I/O ラインを列として持っています。行と列の交点ごとに 1 つのキーが設定されます。そのため、キーボード上のキーの数は 4*4 となります。この行と列のキーボード構造は、マイクロコントローラーシステムでの I/O ポートの利用率を効果的に向上させることができます。

キーの接触部は、リボンケーブルでの接続やプリント基板への挿入に適したヘッダー経由でアクセスされます。一部のキーパッドでは、各ボタンがヘッダー内の別々の接触部と接続している一方、全てのボタンが共通の接地を共有しています。



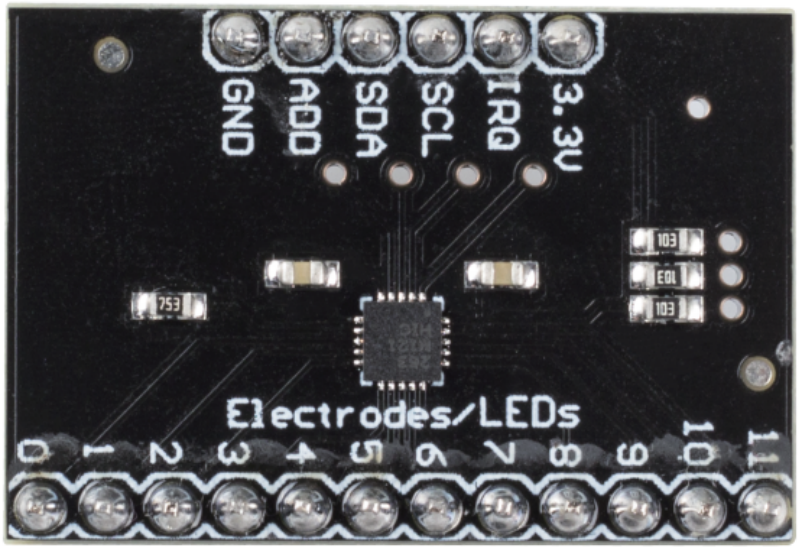
より一般的には、ボタンはマトリックスエンコードされており、各ボタンがマトリックス内の一意の導体ペアをブリッジしています。この構成は、マイクロコントローラによるポーリングに適しています。マイクロコントローラは、4 つの水平線のそれぞれに順番に出力パルスを送るようにプログラムされています。各パルス中に、残りの 4 つの垂直線を順番にチェックし、どれが信号を運んでいるかを判断します。信号が存在しないときにマイクロコントローラの入力が不安定にならないように、入力線にはプルアップまたはプルダウン抵抗器を追加する必要があります。

- キーパッド - ウィキペディア

例

- *4.2 4x4* **キーパッド**（MicroPython ユーザー向け）
- *7.7* **数字当てゲーム**（MicroPython ユーザー向け）
- *4.2 4x4* **キーパッド**（Arduino ユーザー向け）

2.30 MPR121 モジュール



- **3.3V**: 電源供給
- **IRQ**: アクティブローのオープンコレクタ割り込み出力ピン
- **SCL**: I2C クロック
- **SDA**: I2C データ
- **ADD**: I2C アドレス選択入力ピン。ADDR ピンを VSS、VDD、SDA、SCL ラインに接続すると、それぞれの I2C アドレスは 0x5A、0x5B、0x5C、0x5D になります。
- **GND**: グラウンド
- **0~11**: 電極 0~11、電極はタッチセンサーです。通常、電極は金属片やワイヤーで構成されています。しかし、ワイヤーの長さや電極が乗っている材料によっては、センサーのトリガーが難しくなる場合があります。そのため、MPR121 では電極のトリガーおよびアントリガーに必要な設定を行うことができます。

MPR121 概要

MPR121 は、初代 MPR03x シリーズデバイスのリリース後の第二世代の静電容量式タッチセンサーコントローラーです。MPR121 には、内部のインテリジェンスが強化され、主な追加点としては電極数の増加、ハードウェアで設定可能な I2C アドレス、デバウンスを備えた拡張フィルタリングシステム、および自動構成が組み込まれた完全に独立した電極があります。このデバイスには、多重化されたセンシング入力を使用して近接検出を行うための 13 番目のシミュレーションされたセンシングチャネルも備えています。

- MPR121 データシート

特長

低消費電力動作

- 1.71V から 3.6V までの供給動作
- 16ms のサンプリング間隔での 29 µ A の供給電流
- 3 µ A のストップモード電流

12 の静電容量センシング入力

- 8 つの入力は LED ドライバーおよび GPIO として多機能

完全なタッチ検出

- 各センシング入力に対する自動構成
- 各センシング入力に対する自動キャリブレーション
- タッチ検出のためのタッチ/リリース閾値およびデバウンス

- I2C インターフェース、割り込み出力付き
- 3mm x 3mm x 0.65mm 20 リード QFN パッケージ
- -40 ° C から +85 ° C の動作温度範囲

例

- *4.3* **電極キーボード** (MicroPython ユーザー向け)
- *7.9* **フルーツピアノ** (MicroPython ユーザー向け)
- *4.3 -* **電極キーボード** (Arduino ユーザー向け)

2.31 MFRC522 モジュール

MFRC522 は一種の統合型読み取りおよび書き込みカードチップであり、主に 13.56MHz の周波数で無線通信に用いられます。NXP 社によって導入されたこのチップは、低電圧、低コスト、小型の非接触カードチップであり、インテリジェント機器や携帯型ハンドヘルドデバイスに最適です。

MF RC522 は、13.56MHz のすべての種類の受動型非接触通信手法とプロトコルにおいて、先進的な変調および復調概念を完全に採用しています。さらに、MIFARE 製品を検証するための高速 CRYPTO1 暗号化アルゴリズムもサポートしています。MFRC522 は、MIFARE シリーズの高速非接触通信もサポートしており、双方向のデータ転送レートは最大 424kbit/s に達します。13.56MHz の高度に統合されたリーダーカードシリーズの新メンバーとして、MF RC522 は既存の MF RC500 や MF RC530 と多くの点で類似していますが、大きな違いも存在します。ホストマシンとの通信はシリアル方式で行われ、配線が少なくて済みます。SPI、I2C、シリアル UART モード（RS232 に似ている）のいずれかを選択することができ、これにより接続が削減され、PCB ボードのスペースが節約され（小型化）、コストが削減されます。

- MFRC522 データシート

例

- *6.5* **無線周波識別（*RFID*）** （MicroPython ユーザー向け）
- *7.8 RFID* **音楽プレーヤー** （MicroPython ユーザー向け）
- *6.5 -* **無線周波数識別（*RFID*）** （Arduino ユーザー向け）

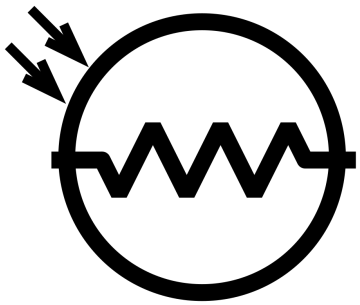
センサー

2.32 フォトレジスタ

フォトレジスタまたはフォトセルは、光によって制御される可変抵抗器です。光の強度が増えると、フォトレジスタの抵抗は減少します。言い換えれば、光導電性を有しています。

フォトレジスタは、光感応検出回路や光活性および暗活性のスイッチング回路で、抵抗性の半導体として使用することができます。暗闇の中では、フォトレジスタの抵抗は数メガオーム（M Ω）に達することがありますが、明るい環境では数百オームまで低下することがあります。

以下は、フォトレジスタの電子記号です。



- フォトレジスタ - Wikipedia

例

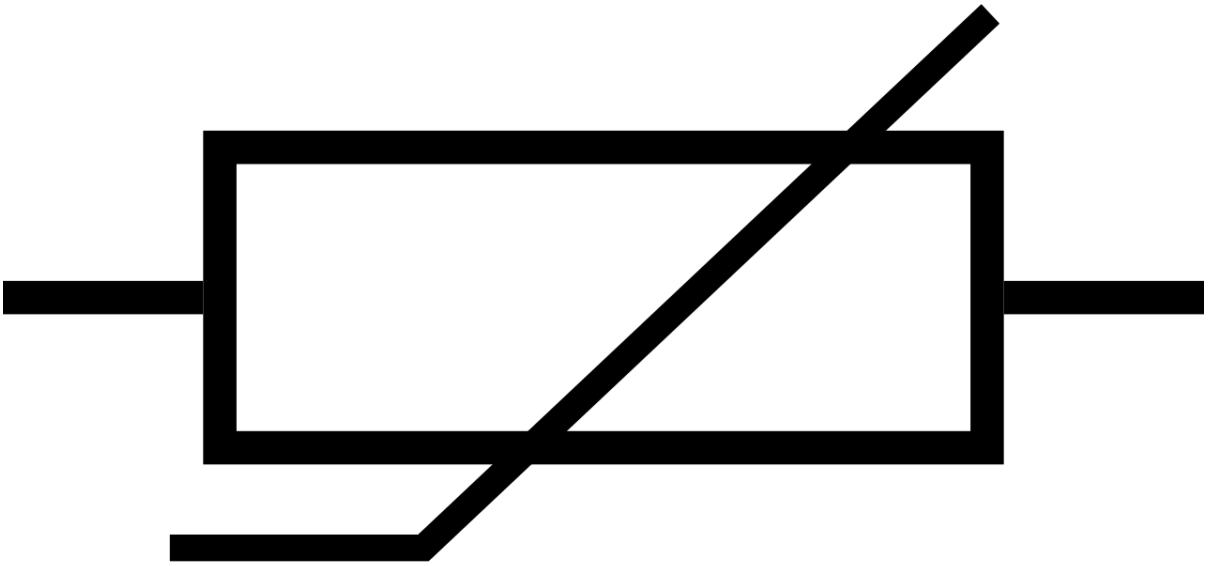
- *2.12* **光を感じる**（MicroPython ユーザー向け）
- *7.1* **光センサー・テルミン**（MicroPython ユーザー向け）
- *2.12 -* **光を感じる**（Arduino ユーザー向け）
- *2.8* **光強度表示装置**（Piper Make ユーザー向け）

2.33 サーミスター

サーミスターは、標準の抵抗器よりも温度に強く依存する抵抗を持つタイプの抵抗器です。この言葉は「thermal（熱）」と「resistor（抵抗器）」を組み合わせたものです。サーミスターは、電流制限器、温度センサー（通常は負温度係数または NTC タイプ）、自己リセット型過電流保護器、および自己調整加熱素子（通常は正温度係数または PTC タイプ）として広く使用されています。

- サーミスター - Wikipedia

以下は、サーミスターの電子記号です。



サーミスターには基本的に 2 つの逆のタイプがあります：

- NTC サーミスターでは、温度が上昇すると抵抗が減少します。これは通常、価電子帯から熱振動によって打ち上げられた伝導電子の増加によるものです。NTC は、温度センサーまたはインラッシュ電流制限器として回路と直列に接続されて一般的に使用されます。
- PTC サーミスターでは、温度が上昇すると抵抗が増加します。これは通常、特に不純物や欠陥の熱格子振動が増加した結果です。PTC サーミスターは、一般的に回路と直列に接続され、過電流条件に対する保護として、またはリセット可能なヒューズとして使用されます。

このキットでは NTC タイプを使用しています。各サーミスターには通常の抵抗値があります。ここではそれは 10k オームで、これは 25 度 Celsius で測定されたものです。

抵抗と温度の関係は以下の通りです：

RT = RN * expB(1/TK - 1/TN)

- **RT** は、温度が TK のときの NTC サーミスターの抵抗です。
- **RN** は、定格温度 TN 下での NTC サーミスターの抵抗です。ここでは、RN の数値は 10k です。
- **TK** はケルビン温度で、単位は K です。ここでは、TK の数値は 273.15 + 温度 Celsius です。
- **TN** も定格ケルビン温度であり、単位も K です。ここでは、TN の数値は 273.15+25 です。
- そして、**B（ベータ）** は NTC サーミスターの材料定数であり、熱感度指数とも呼ばれ、数値は 3950 です。
- **exp** は指数の略で、基数 e は自然数であり、おおよそ 2.7 と等しいです。

この式 TK=1/(ln(RT/RN)/B+1/TN) を変換して、ケルビン温度から 273.15 を引くと Celsius 温度になります。

この関係は実験式であり、温度と抵抗が有効範囲内である場合にのみ正確です。

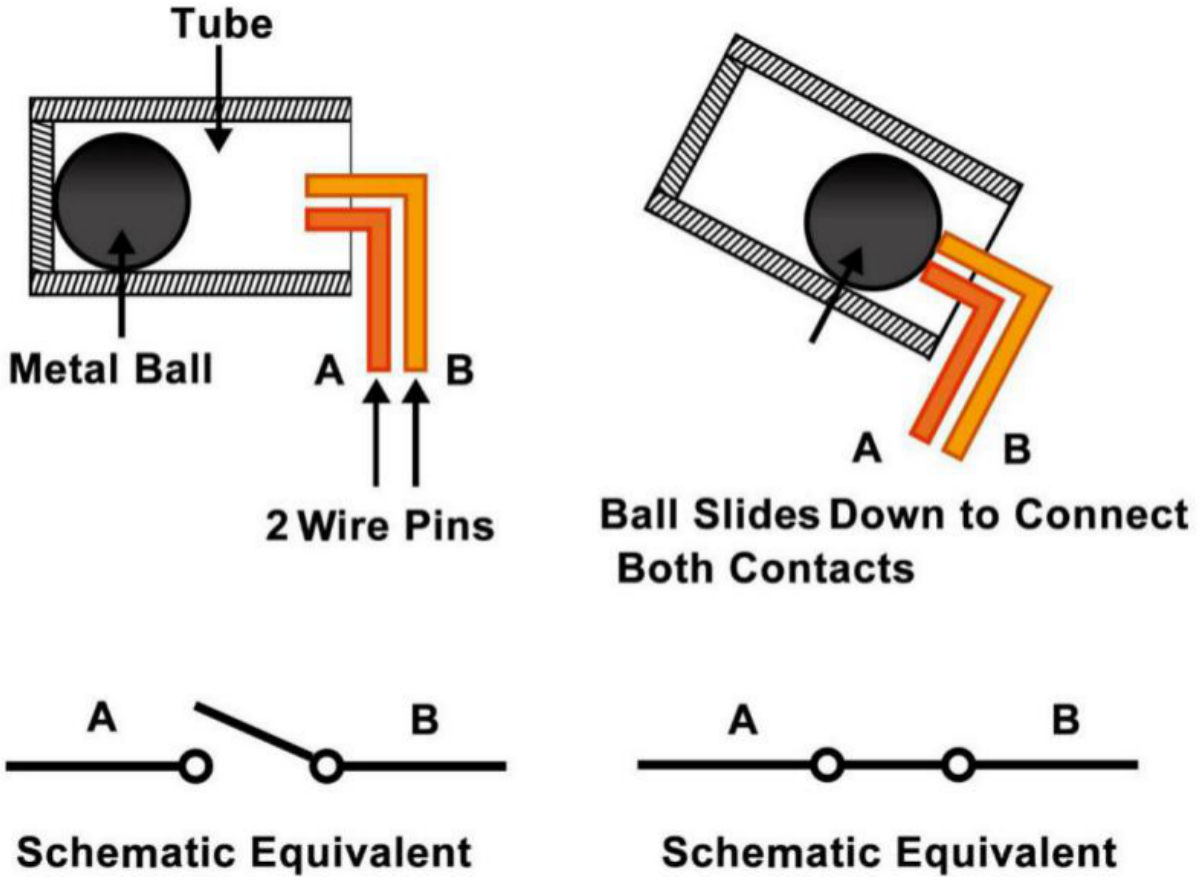
例

- *2.13* **温度計** (MicroPython ユーザー向け)
- *7.2* **室温計** (MicroPython ユーザー向け)
- *2.13 -* **温度計** (Arduino ユーザー向け)

2.34 傾斜スイッチ

ここで使用されている傾斜スイッチは、内部に金属製のボールが入っているタイプです。これは小角度の傾斜を検出するために使用されます。

その原理は非常にシンプルです。スイッチが特定の角度で傾けられると、内部のボールが転がって外部のピンに接続された 2 つの接点に触れ、回路を作動させます。それ以外の場合、ボールは接点から離れ、回路を遮断します。



- SW520D 傾斜スイッチのデータシート

例

- *2.6* **傾けてみよう！** (MicroPython ユーザー向け)
- *7.5 GAME - 10* **秒ゲーム** (MicroPython ユーザー向け)
- *2.6 -* **傾けてみよう！** (Arduino ユーザー向け)
- *2.10* **流れる** *LED* (Piper Make ユーザー向け)

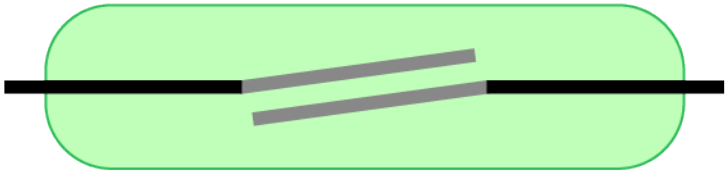
2.35 リードスイッチ

リードスイッチは、外部からの磁場を感知して動作する電気スイッチです。このスイッチは 1936 年にベル電話研究所のウォルター・B・エルウッドによって発明され、1940 年 6 月 27 日に特許番号 2264746 でアメリカ合衆国で特許を取得しました。

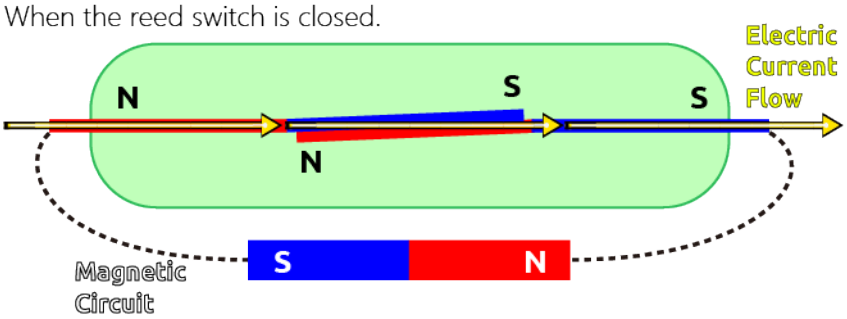
リードスイッチの動作原理は非常にシンプルです。二つのリード（通常は鉄とニッケルの二つの金属で作られています）が端点で重なるようにガラス管に封印されています。この二つのリードは小さな隙間（数マイクロメートル程度）で離れて重なっています。ガラス管内は高純度の不活性ガス（例えば窒素）で満たされており、高電圧性能を高めるために真空状態にされているリードスイッチもあります。

リードは磁束の導体として機能します。二つのリードは動作前には接触していませんが、永久磁石や電磁コイルによって生成された磁場を通過すると、二つのリードの端点付近で異なる極性が生じます。この磁力がリード自体のバネ力を超えると、二つのリードは引き寄せられて回路が閉じます。磁場が弱くなるか消えると、リードは自身の弾性によって解放され、接触面が開いて回路が開きます。

When the reed switch is open.



When the reed switch is closed.



- リードスイッチ - Wikipedia

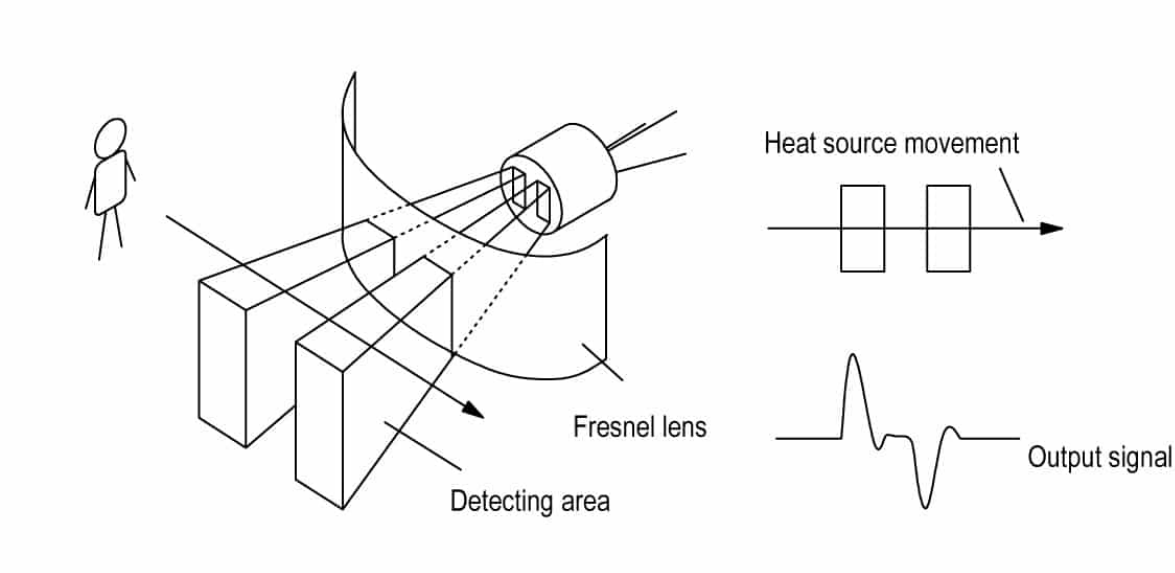
例

- *2.9* **磁気を感じる**（MicroPython ユーザー向け）
- *2.9* - **磁気を感じる**（Arduino ユーザー向け）

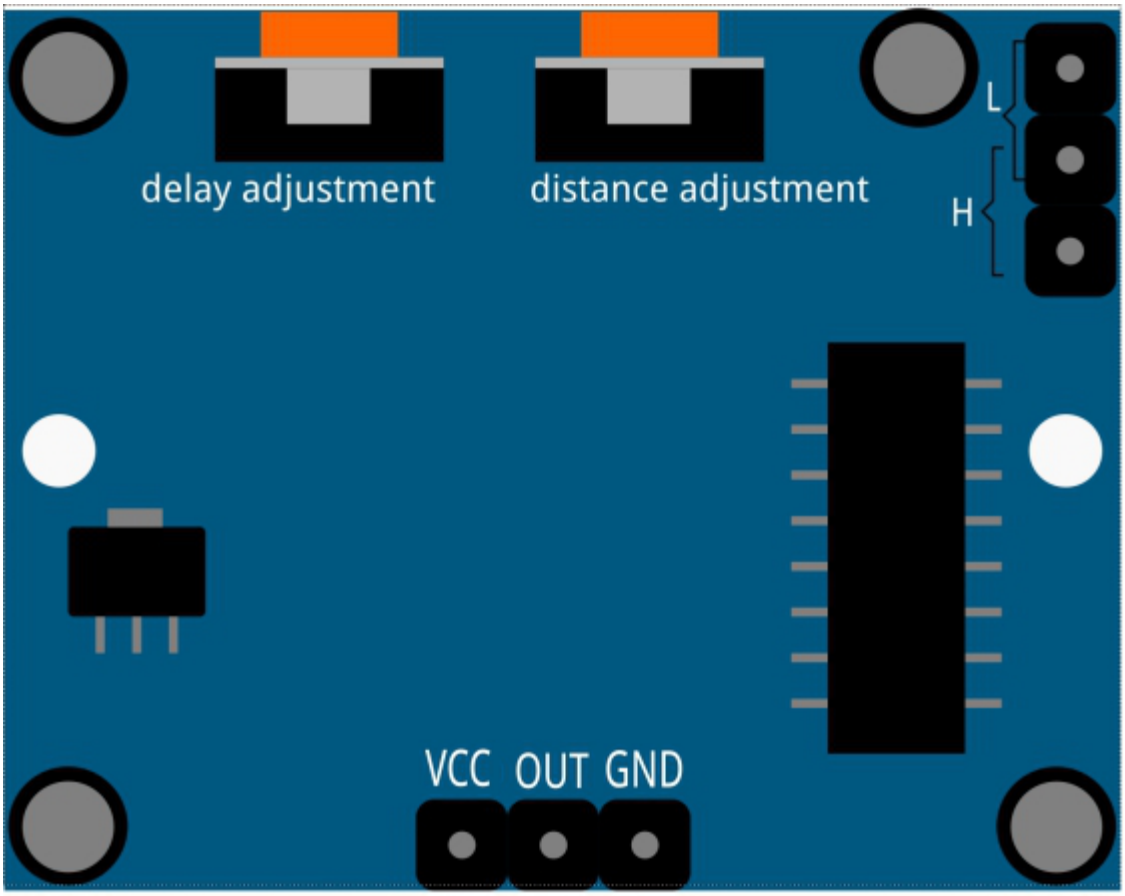
2.36 PIR モーションセンサーモジュール

PIR センサーは、赤外線熱放射を検出するセンサーです。これを用いて赤外線熱放射を発生する生物の存在を感知できます。

PIR センサーは二つのスロットに分かれており、それぞれが差動増幅器に接続されています。固定された物体がセンサーの前にある場合、両方のスロットが同じ量の放射線を受け取り、出力はゼロになります。動く物体がセンサーの前にある場合、一方のスロットがもう一方よりも多くの放射線を受け取り、出力が高いまたは低いに変動します。この出力電圧の変化が、動きの検出となります。



センシングモジュールが配線された後、1 分間の初期化があります。初期化中は、モジュールが 0〜3 回の間隔で出力します。その後、モジュールはスタンバイモードになります。モジュールの表面から光源や他の干渉源を遠ざけて、誤動作を防ぐようにしてください。風もセンサーに干渉する可能性があるため、風の影響を受けない場所で使用することが望ましいです。



距離調整

距離調整用のポテンショメーターのつまみを時計回りに回すと、センシング距離の範囲が広がります。最大センシング距離は約 0~7 メートルです。反時計回りに回すと、センシング距離の範囲が狭まり、最小センシング距離は約 0~3 メートルです。

遅延調整

遅延調整用のポテンショメーターのつまみを時計回りに回すと、センシング遅延が長くなります。最大で 300 秒に達することがあります。反対に、反時計回りに回すと、最小で 5 秒の遅延に短縮できます。

二つのトリガーモード

ジャンパーキャップを用いて異なるモードを選択できます。

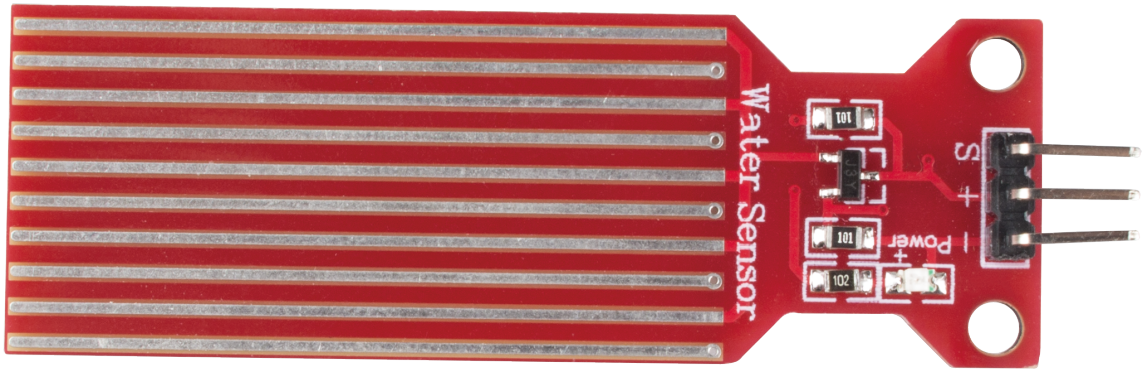
- **H**: 反復可能なトリガーモード。人体を感知した後、モジュールは高レベルを出力します。その後の遅延期間中に誰かが感知範囲に入ると、出力は高レベルのままです。
- **L**: 非反復トリガーモード。人体を感知すると高レベルを出力し、遅延後に自動的に低レベルに戻ります。

例

- *2.10* **人間の動きを検出** （MicroPython ユーザー向け）

- *7.4* **乗客カウンター** （MicroPython ユーザー向け）
- *2.10 -* **人の動きを検出する** （Arduino ユーザー向け）
- *2.9* **招き猫プロジェクト** （Piper Make ユーザー向け）

2.37 水位センサーモジュール



この水位センサーは、感知した水位信号をコントローラーに送信します。コントローラー内のコンピュータは、測定された水位信号と設定された信号を比較して偏差を導き出し、その偏差の性質に応じて給水電動弁に「オン」と「オフ」のコマンドを発行します。これにより、容器が設定された水位に達することが確保されます。

水位センサーには十本の露出した銅のトレースがあり、そのうち五本は電源トレース、残りの五本はセンサートレースです。これらは水位が上がると交差してブリッジされます。基板には、基板に電力が供給されたときに点灯する電源 LED があります。

これらのトレースの組み合わせは、可変抵抗器のように作用し、水位に応じて抵抗値が変わります。具体的には、センサーが浸かる水が多いほど、導電性が良く抵抗が低くなります。逆に、導電性が低いほど抵抗は高くなります。次に、このセンサーは出力信号電圧を処理し、それをマイクロコントローラーに送信します。これにより、水位を判定する際の補助となります。

> **警告:** このセンサーは水中に完全に没入させることはできません。十本のトレースがある部分のみを水と接触させてください。また、湿った環境でセンサーに電力を供給すると、プローブの腐食が速まり、センサーの寿命が短くなる可能性があるため、測定を行う際のみ電力を供給することをお勧めします。

例

- *2.14* **水位を感知する** （MicroPython ユーザー用）
- *2.14 -* **水位を感じる** （Arduino ユーザー用）

- *2.6* **スマートウォータータンク** （Piper Make ユーザー用）

2.38 超音波モジュール

- **TRIG**: トリガーパルス入力
- **ECHO**: エコーパルス出力
- **GND**: 接地
- **VCC**: 5V 電源

これは HC-SR04 超音波距離センサーで、非接触測定が 2cm から 400cm まで、精度は最大 3mm まで可能です。このモジュールには超音波送信機、受信機、および制御回路が搭載されています。

使用するためには 4 つのピンを接続するだけ：VCC（電源）、Trig（トリガー）、Echo（受信）、GND（接地）です。

特長

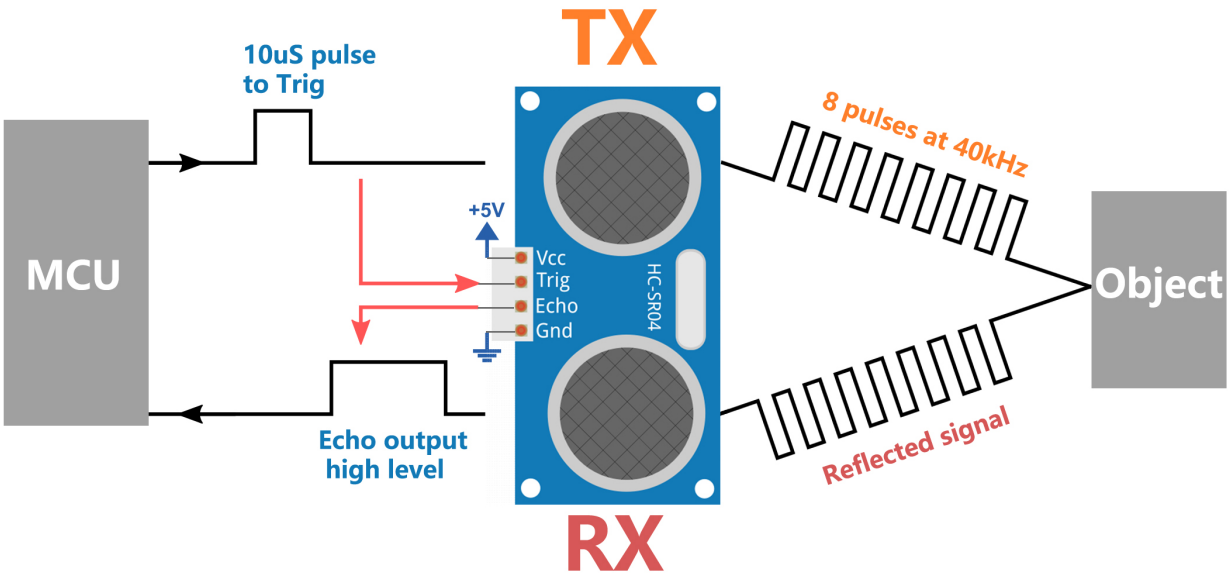
- 動作電圧: DC5V
- 動作電流: 16mA
- 動作周波数: 40Hz
- 最大測定範囲: 500cm
- 最小測定範囲: 2cm
- トリガー入力信号: 10uS TTL パルス
- エコー出力信号: 入力 TTL レベル信号と距離に比例
- コネクタ: XH2.54-4P
- 寸法: 46x20.5x15 mm

原理

基本的な原則は以下の通りです：

- 最低でも 10us の高レベル信号で IO トリガーを使用。

- モジュールは 40kHz で 8 サイクルの超音波バーストを送信し、パルス信号が受信されるかどうかを検出します。
- 信号が返されると、エコーは高レベルを出力します。この高レベルの持続時間は、放射から返信までの時間です。
- 距離 = （高レベル時間 x 音速（340M/S））/ 2



公式：

- us / 58 = センチメートル距離
- us / 148 = インチ距離
- distance = 高レベル時間 x 音速（340M/S）/ 2

注釈: このモジュールは電源が入っている状態で接続しないでください。必要な場合は、モジュールの GND を最初に接続してください。そうしないと、モジュールの動作に影響を与えます。

測定対象のオブジェクトの面積は少なくとも 0.5 平方メートルで、できるだけ平らである必要があります。そうでないと、結果に影響を与えます。

例

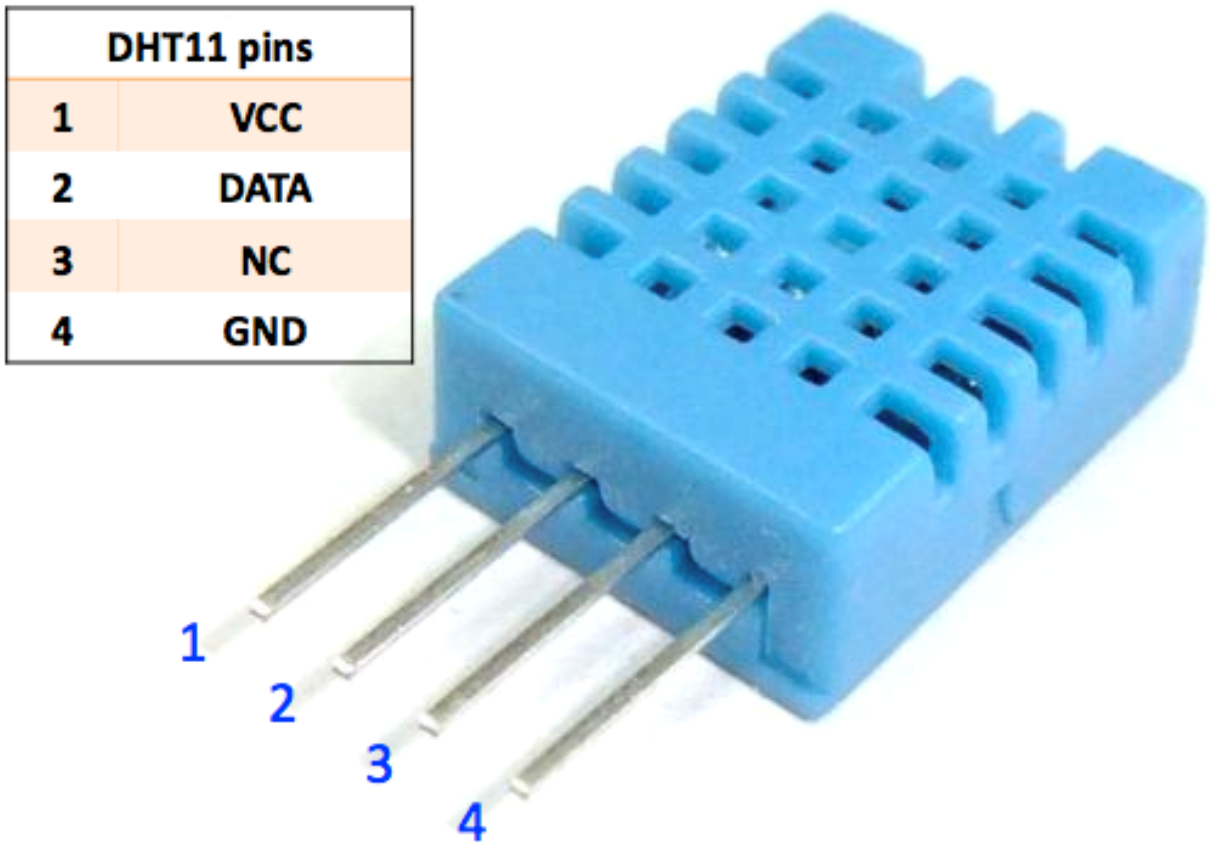
- *6.1* **距離の測定** （MicroPython ユーザー用）
- *7.10* **バックアップ支援** （MicroPython ユーザー用）
- *6.1 -* **距離の測定** （Arduino ユーザー用）
- *2.11* **逆転警報システム** （Piper Make ユーザー用）

2.39 DHT11 湿温度センサー

DHT11 デジタル温湿度センサーは、温度と湿度のキャリブレートされたデジタル信号出力を持つ複合センサーです。専用のデジタルモジュールの収集技術と温湿度センシング技術が組み合わされており、製品は高い信頼性と長期的な安定性を保証しています。

このセンサーには、抵抗式の湿度感知コンポーネントと NTC 温度測定装置が含まれており、高性能の 8 ビットマイクロコントローラーに接続されています。

使用できるピンは 3 つだけです：VCC、GND、および DATA。通信プロセスは、DATA ラインが DHT11 にスタート信号を送り、DHT11 がその信号を受け取って応答信号を返すところから始まります。その後、ホストが応答信号を受け取り、40 ビットの湿温度データ（8 ビット湿度整数 + 8 ビット湿度小数 + 8 ビット温度整数 + 8 ビット温度小数 + 8 ビットチェックサム）を受け取り始めます。



特長

1. 湿度測定範囲：20 - 90%RH
2. 温度測定範囲：0 - 60 ℃
3. 温度と湿度を示すデジタル信号出力

4. 動作電圧：DC 5V; PCB サイズ：2.0 x 2.0 cm
5. 湿度測定精度：±5%RH
6. 温度測定精度：±2 ℃

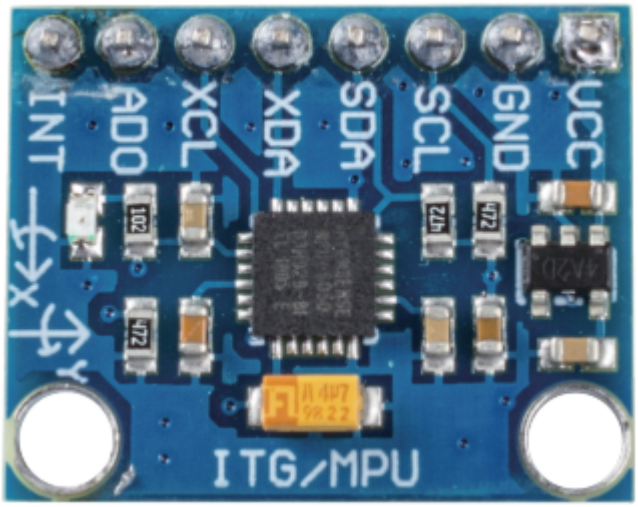
- DHT11 データシート

例

- *6.2* **温度・湿度センサー** （MicroPython ユーザー向け）
- *6.2 -* **温度・湿度** （Arduino ユーザー向け）

2.40 MPU6050 モジュール

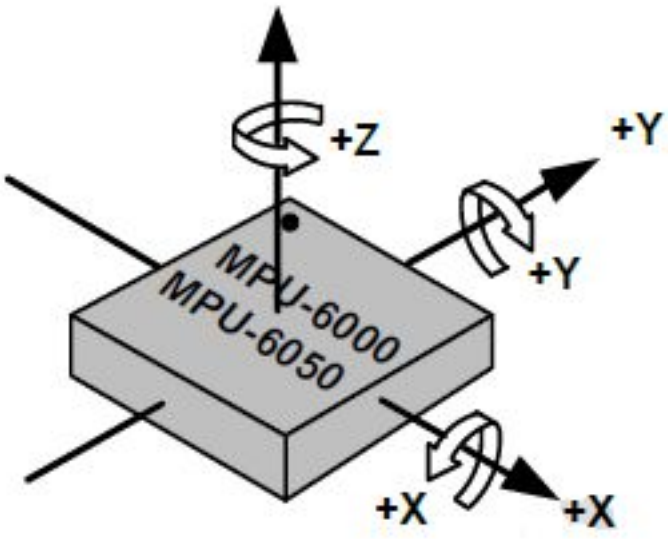
MPU6050



MPU-6050 は、3 軸ジャイロスコープと 3 軸加速度計を組み合わせた 6 軸の運動追跡デバイスです。

このチップの座標系は以下のように定義されています：

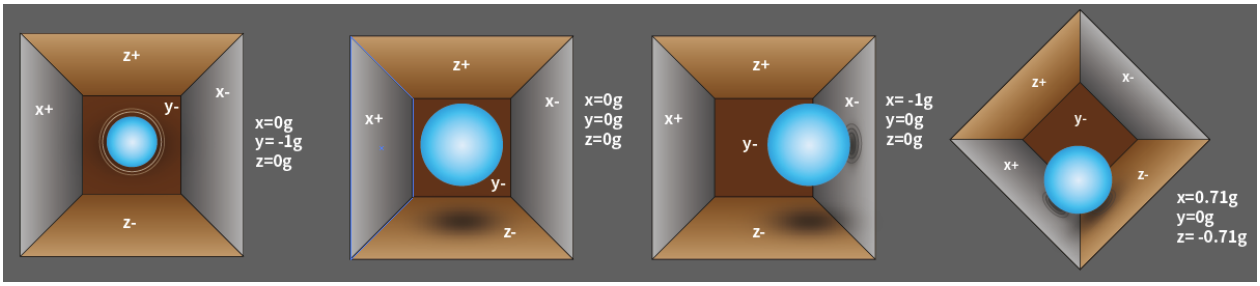
MPU6050 をテーブルに平らに置き、ラベル面が上になるようにして、その面の左上にドットがあることを確認します。この状態で、チップの Z 軸は上方向、X 軸は左から右方向、Y 軸は後ろから前方向と定義されます。



3 軸加速度計

加速度計は圧電効果の原理に基づいています。この効果は、機械的なストレスを受けた際に特定の材料が電荷を発生させる能力です。

ここで、立方体の箱に小さなボールが入っている状態を想像してください。この箱の壁は圧電結晶でできています。箱を傾けると、ボールは重力の影響で傾斜方向に移動します。ボールが衝突する壁は、微小な圧電電流を生じます。立方体には、対向する 3 組の壁があり、それぞれ X、Y、Z 軸に対応しています。壁から発生する電流によって、傾斜方向とその大きさを判断できます。



MPU6050 を使って、各座標軸に沿った加速度を検出することができます（静止状態での Z 軸加速度は 1G、X 軸と Y 軸は 0 です）。傾けられたり、無重力/過重状態にある場合、対応する読み取り値が変わります。

計測範囲はプログラムで選択でき、+/-2g、+/-4g、+/-8g、+/-16g（デフォルトは 2g）があります。値の範囲は-32768 から 32767 です。

加速度計の読み取り値は、次の式で加速度値に変換されます：

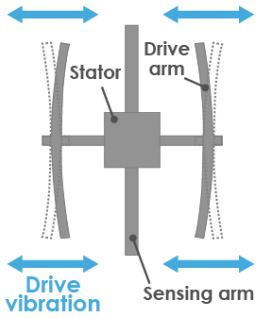
Acceleration = (Accelerometer axis raw data / 65536 * full scale Acceleration range) g

例えば、加速度計の X 軸の生データが 16384 で、範囲が +/-2g の場合：

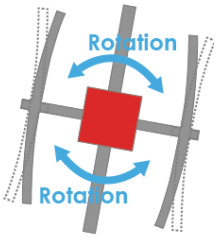
Acceleration along the X axis = (16384 / 65536 * 4) g =1g

3 軸ジャイロスコープ

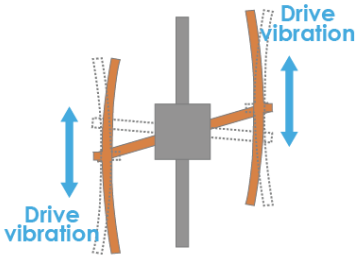
ジャイロスコープは、コリオリ加速度の原理に基づいて動作します。ここで、一定の前後運動を持つフォークのような構造があると想像してください。この構造は、圧電結晶で固定されています。この配置を傾けようとすると、結晶は傾斜方向に力を感じます。これは、動いているフォークの慣性の結果です。結晶は、圧電効果によって電流を生成し、この電流が増幅されます。



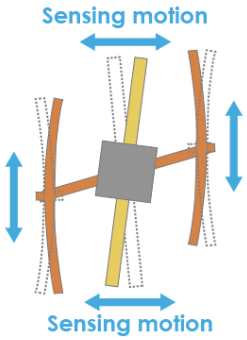
1.Normally, a drive arm vibrates in a certain direction.



2.Direction of rotation



3.When the gyro is rotated,the Coriolis force acts on the drive arms, producing vertical vibration.



4.The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

ジャイロスコープも 4 種類の測定範囲を持っています：+/- 250、+/- 500、+/- 1000、+/- 2000。計算方法は基本的に加速度計と同じです。

角速度を読み取り値に変換する式は次のとおりです：

Angular velocity = (Gyroscope axis raw data / 65536 * full scale Gyroscope range) ° /s

X 軸の例では、加速度計の X 軸の生データが 16384 で、範囲が +/- 250 ° /s の場合：

Angular velocity along the X axis = (16384 / 65536 * 500) ° /s =125 ° /s

例

- *6.3 6* **軸モーショントラッキング** (MicroPython ユーザー向け)
- *7.11* **体感コントローラー** (MicroPython ユーザー向け)
- *7.12* **デジタル水平器** (MicroPython ユーザー向け)
- *6.3 - 6* **軸モーショントラッキング** (Arduino ユーザー向け)

第 3 章

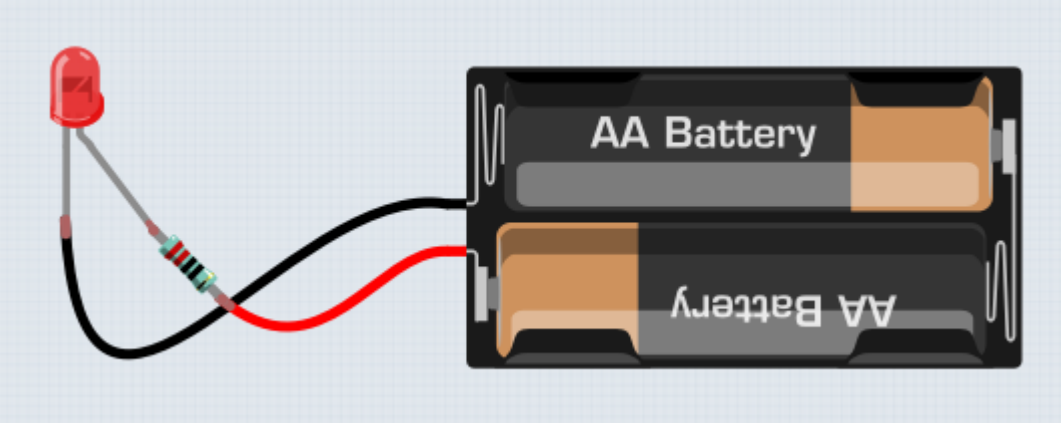
電子回路

毎日使用する多くのもの、例えば自宅の照明やこの文章を読んでいるコンピューターなど、電気で動いています。

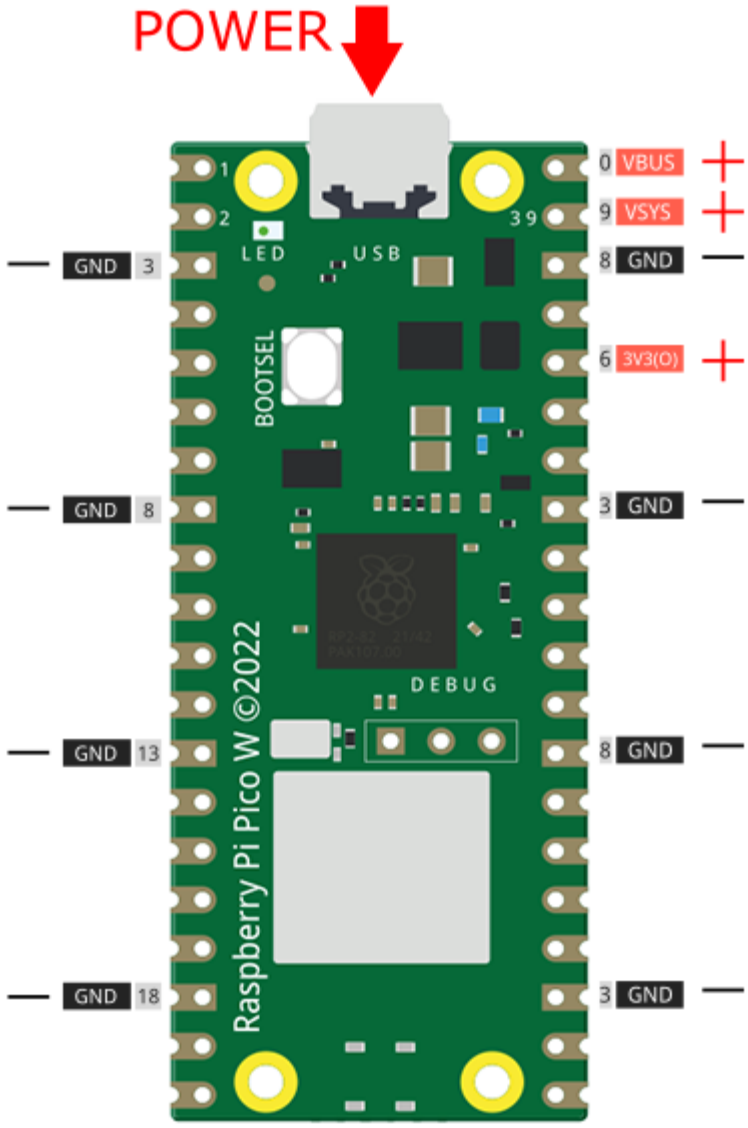
電気を使用するには、電気回路を作成する必要があります。電気回路は、金属ワイヤーと電気および電子部品で構成されています。

回路はどこかから電力を必要とします。家庭での大半の電化製品（例：テレビ、照明）は壁のコンセントから電力を供給されますが、多くの小型で携帯可能な回路（例：電子玩具、携帯電話）はバッテリーで動いています。バッテリーには二つの端子があり、プラス記号（+）でマークされたものが正極、マイナス記号（-）で示されるものが負極です。

電流が流れるためには、負極と正極を接続する導電性の経路が必要で、これを閉回路と呼びます（接続がない場合は開回路と呼びます）。電流は、ランプなどの機器を動作させるために流れます（例：点灯）。

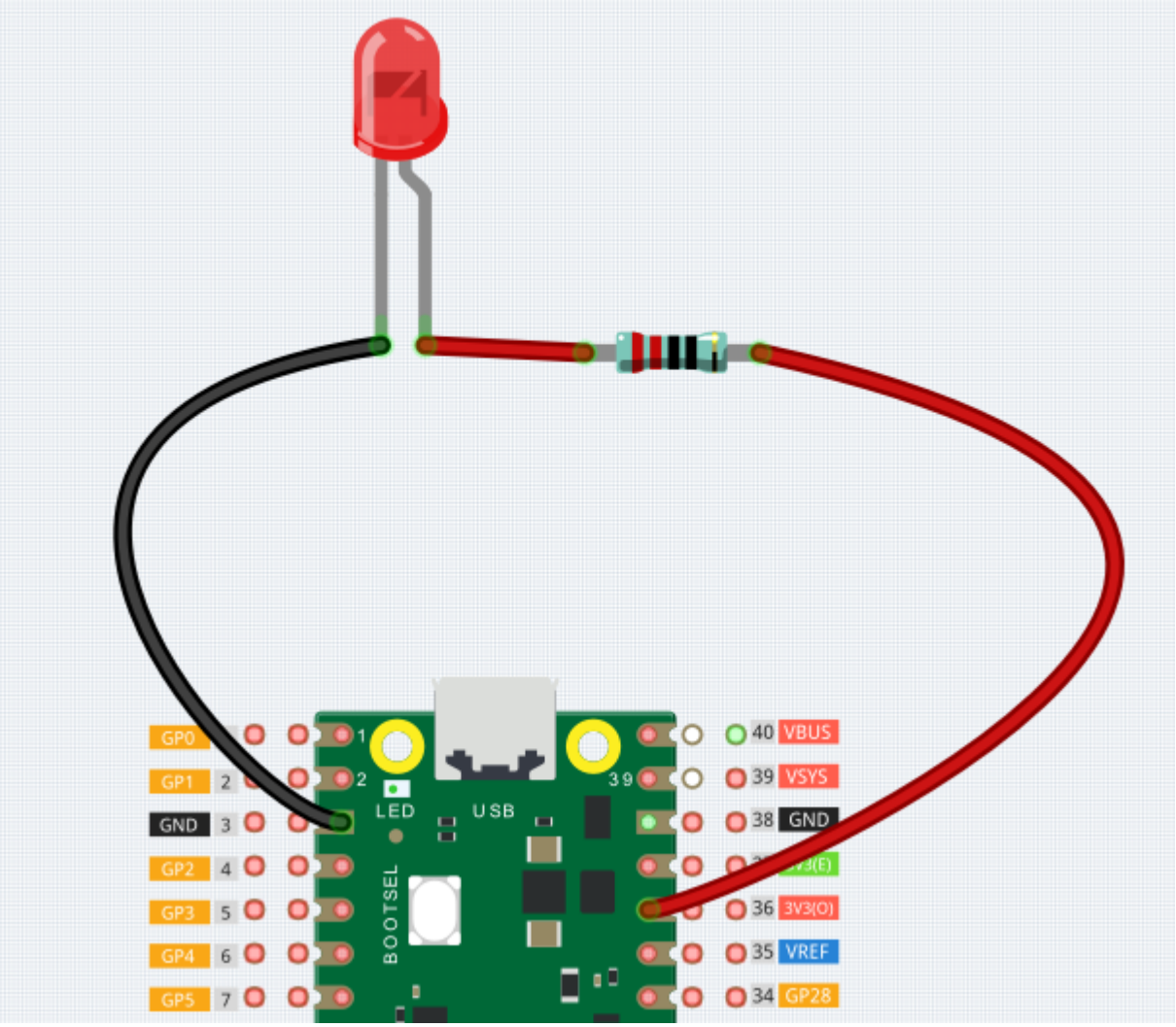


Pico W にはいくつかの電源出力ピン（正）とグラウンドピン（負）があります。Pico W を電源に接続することで、これらのピンを電源の正極と負極として使用することができます。



電気を使って、光や音、動きのある作品を作成できます。LED を点灯させるには、長いピンを正極に、短いピンを負極に接続します。この状態では LED はすぐに壊れてしまうため、回路内に 220 Ωの抵抗器を追加して保護する必要があります。

下にその回路を示します。



この時点で疑問が出てくるかもしれません：この回路をどのように組み立てるのか？ワイヤーを手で持っているだけなのか、それともピンとワイヤーをテープで固定するのか？

このような状況では、はんだ付け不要のブレッドボードが非常に便利です。

3.1 はじめてのブレッドボード！

ブレッドボードは、多数の小さな穴が開いた長方形のプラスチック板です。これらの穴を通して、電子部品を容易に挿入し、電子回路を組むことができます。ブレッドボードは電子部品を恒久的に固定しないので、何か問題が発生した場合には簡単に回路を修理またはやり直すことができます。

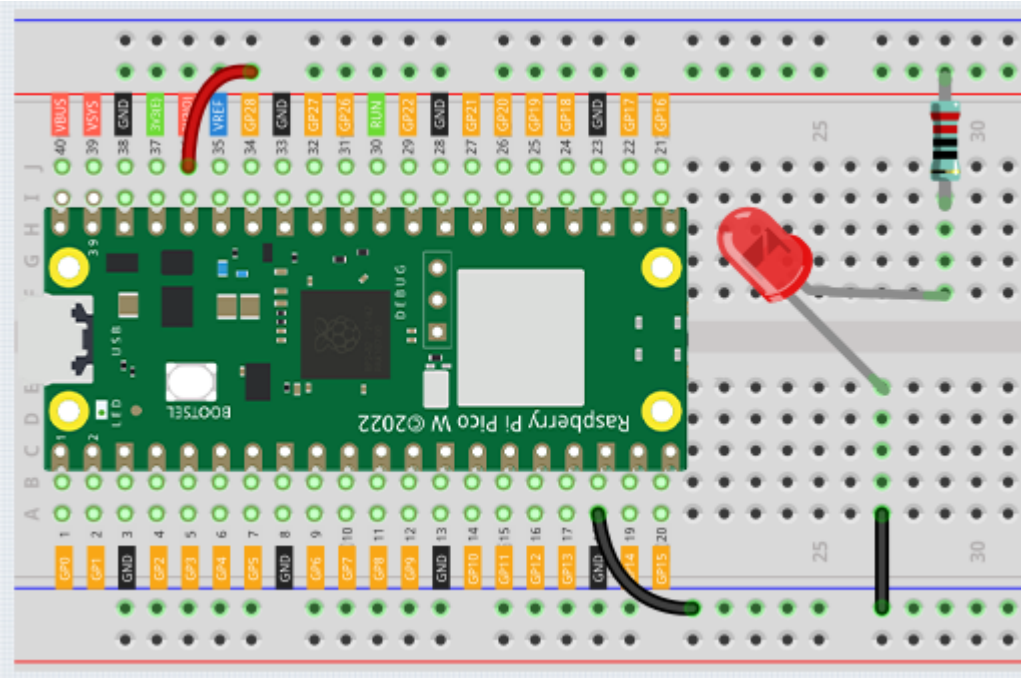
注釈: ブレッドボードを使用するための特別な工具は必要ありません。ただし、多くの電子部品は非常に小さく、ピンセットが小さな部品を取り扱う際に役立ちます。

インターネット上には、ブレッドボードに関する多くの情報があります。

- How to Use a Breadboard - Science Buddies
- What is a BREADBOARD? - Makezine

以下は、ブレッドボードについて知っておくべきいくつかのポイントです。

1. 各半行グループ（例えば、行 1 の列 A-E や行 3 の列 F-J など）は接続されています。したがって、電気信号が A1 から流れてきた場合、B1、C1、D1、E1 から流れ出ることができますが、F1 や A2 からは流れ出せません。
2. ほとんどの場合、ブレッドボードの両側は電源バスとして使用され、各列の穴（約 50 穴）が接続されています。一般的に、正の電源は赤いワイヤー近くの穴に、負の電源は青いワイヤー近くの穴に接続されます。
3. 回路内では、電流は負極から正極へ負荷を通過した後に流れます。この場合、短絡（ショート）が発生する可能性があります。



電流の方向に沿って、回路を組み立ててみましょう！

1. この回路では、Pico W ボードの 3V3 ピンを使用して LED に電力を供給します。M2M（オス-オス）ジャンパーワイヤーを使用して、それを赤い電源バスに接続します。
2. LED を保護するために、電流は 220 オームの抵抗器を通過する必要があります。抵抗器の一方の端（どちらの端でも可）を赤い電源バスに、もう一方の端をブレッドボードのフリーロー（私の回路では行 24）に接続します。

注釈: 220 オームの抵抗器のカラーリングは、赤、赤、黒、黒、茶です。

3. LED を取り上げると、一方のリードがもう一方よりも長いことがわかります。長いリードを抵抗器と同じ行に、短いリードをブレッドボードの中央のギャップを挟んで同じ行に接続します。

   注釈: 長いリードはアノードで、回路の正側を表します。短いリードはカソードで、回路の負側を表します。

   アノードは、抵抗器を介して GPIO ピンに接続する必要があります。カソードは GND ピンに接続する必要があります。

4. M2M（オス-オス）ジャンパーワイヤーを使用して、LED の短いピンをブレッドボードの負の電源バスに接続します。

5. ジャンパーを使用して、Pico W の GND ピンを負の電源バスに接続します。

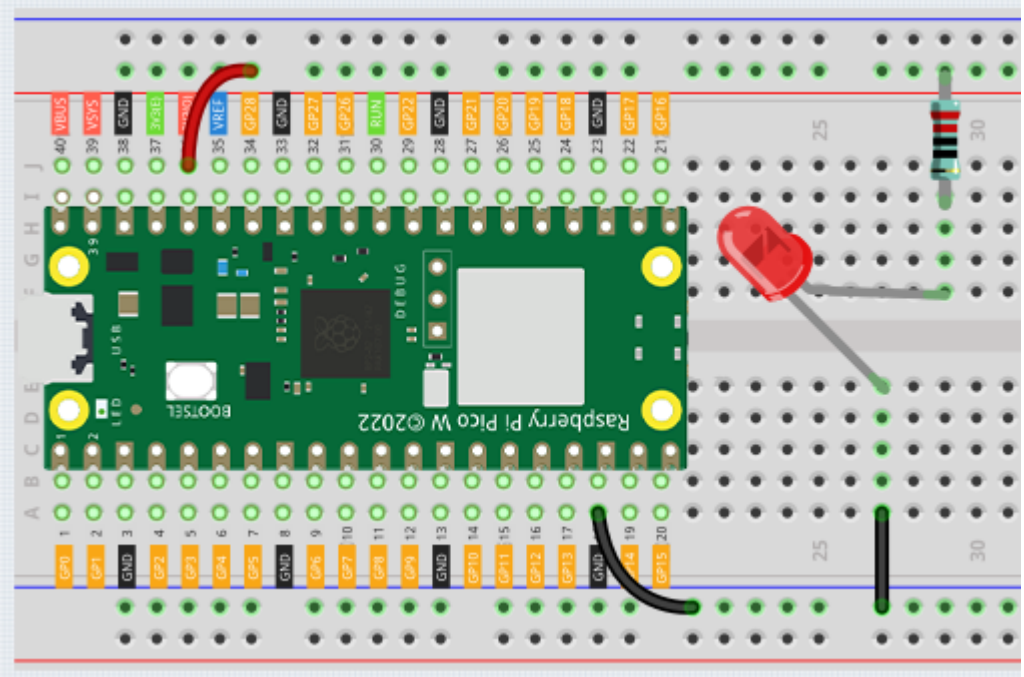
3.2 短絡に注意

短絡は、接続すべきでない二つの部品が「偶然」接続されたときに発生します。このキットには、互いにぶつかって短絡を引き起こす可能性のある、長い金属ピンを持つ抵抗器、トランジスタ、キャパシタ、LED などが含まれています。短絡が発生すると、一部の回路は正常に動作しなくなるだけでなく、場合によっては部品が恒久的に損傷することもあります。特に、電源バスと接地バスの間で短絡が発生すると、回路が非常に高温になり、ブレッドボードのプラスチックが溶けたり、部品が焼けたりする可能性があります。

したがって、ブレッドボード上のすべての電子部品のピンが互いに触れていないように常に注意してください。

3.3 回路の方向性

回路には方向性があり、その方向性は特定の電子部品において重要な役割を果たします。極性を持つデバイスがあり、それらは正極と負極に基づいて正確に接続される必要があります。方向が間違っていると、回路は正常に動作しません。



先ほど組み立てたこの単純な回路で LED の向きを逆にすると、LED はもう動作しなくなります。

対照的に、この回路の抵抗器のように方向を持たないデバイスもありますので、それらを反転させても LED の正常な動作には影響しません。

「+」、「-」、「GND」、「VCC」などのラベルが付いている部品や、ピンの長さが異なる部品は、特定の方法で回路に接続する必要があります。

3.4 回路の保護

電流とは、完全な電気回路の一点を通過する電子の流れの速度です。最も基本的には、電流 = 流れです。アンペア（AM-pir）またはアンプは、電流を測定するための国際単位です。これは、回路の一点を通過する電子（時々「電気荷」と呼ばれる）の量を、所定の時間に表します。

電流の流れの背後にある駆動力（電圧）は、ボルト（V）で測定されます。

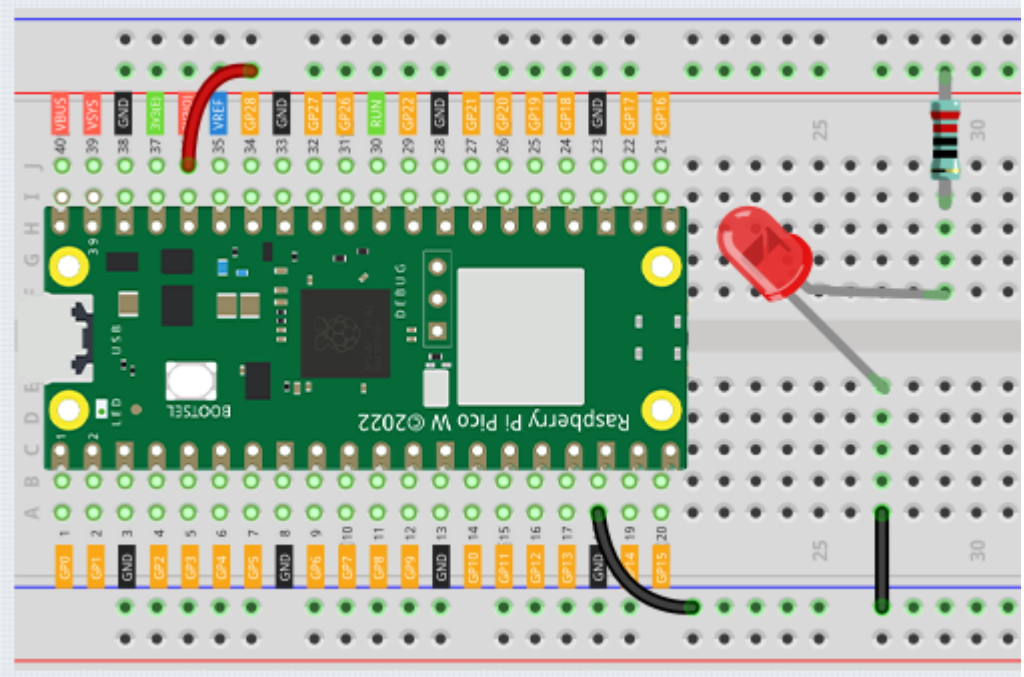
抵抗（R）は、電流の流れを制限する物質の性質であり、オーム（Ω）で測定されます。

オームの法則によれば（温度が一定である限り）、電流、電圧、抵抗は比例します。回路の電流は、その電圧に比例し、その抵抗に反比例します。

したがって、電流（I）= 電圧（V）/ 抵抗（R）です。

- オームの法則 - Wikipedia

オームの法則については、簡単な実験を行うことができます。



3V3 を 5V（すなわち、VBUS、Pico W の 40 番ピン）に接続するワイヤーを変更すると、LED がより明るくなります。抵抗器を 220 オームから 1000 オーム（カラーリング：茶、黒、黒、茶、茶）に変更すると、LED は以前よりも暗くなることに気付くでしょう。抵抗器が大きいほど、LED は暗くなります。

注釈: 抵抗器の紹介と抵抗値の計算方法については、 **抵抗器** を参照してください。

ほとんどのパッケージ化されたモジュールは、適切な電圧（通常は 3.3V または 5V）にアクセスするだけで済みます。例えば、超音波モジュールなどです。

しかし、自作の回路においては、供給電圧と電子デバイスの抵抗器使用に注意する必要があります。

例として、LED は通常、20mA の電流を消費し、その電圧降下は約 1.8V です。オームの法則に基づいて、5V の電源を使用する場合、LED を焼き付かせないように最低 160 オーム（(5-1.8) /20mA）の抵抗器を接続する必要があります。

第 4 章

MicroPython ユーザーのために

このセクションでは、MicroPython の歴史、Pico W に MicroPython をインストールする方法、基本的な文法、そして MicroPython を素早く理解するための数々の実用的で面白いプロジェクトについて学びます。

章は順番に読むことをお勧めします。

1. はじめに

4.1 1.1 MicroPython の紹介

MicroPython は、C 言語で書かれた、Python 3 とほぼ互換性のあるプログラミング言語のソフトウェア実装です。これは、マイクロコントローラ上で動作するように最適化されています。

MicroPython は、Python のバイトコードへのコンパイラと、そのバイトコードのランタイムインタプリターで構成されています。ユーザーには、対話型のプロンプト（REPL）が提供され、サポートされているコマンドを即座に実行できます。コア Python ライブラリの一部が含まれており、MicroPython には、プログラマーが低レベルハードウェアにアクセスできるモジュールも含まれています。

- 参照： MicroPython - Wikipedia

4.1.1 物語はここから始まります

2013 年、ダミアン・ジョージがクラウドファンディング（Kickstarter）を立ち上げたことで、状況が変わりました。

ダミアンはケンブリッジ大学の学部生で、ロボティクスプログラミングに熱心でした。彼は Python の世界をギガバイトマシンからキロバイトへと縮小したいと考えていました。Kickstarter キャンペーンは、彼が概念実証を完成した実装に変えるための開発をサポートするものでした。

MicroPython は、プロジェクトの成功に熱心な多様な Pythonista コミュニティによってサポートされています。

開発者たちは、コードベースのテストとサポートだけでなく、チュートリアル、コードライブラリ、ハードウェアのポーティングも提供しているため、ダミアンはプロジェクトの他の側面に集中することができました。

- 参照： realpython

4.1.2 なぜ MicroPython なのか？

オリジナルの Kickstarter キャンペーンでは、MicroPython は STM32F4 を搭載した開発ボード「pyboard」としてリリースされましたが、多くの ARM ベースの製品アーキテクチャをサポートしています。主にサポートされているポートには、ARM Cortex-M（多くの STM32 ボード、TI CC3200/WiPy、Teensy ボード、Nordic nRF シリーズ、SAMD21 および SAMD51）、ESP8266、ESP32、16 ビット PIC、Unix、Windows、Zephyr、JavaScript があります。また、MicroPython は高速なフィードバックを可能にします。これは、REPL を使用して対話的にコマンドを入力し、レスポンスを得ることができるためです。コードを微調整してすぐに実行することもできます。これにより、コード-コンパイル-アップロード-実行のサイクルを経なくても済みます。

Python にも同様の利点がありますが、Raspberry Pi Pico のような一部のマイクロコントローラボードは、小さくて単純で、Python 言語を全体的に実行するためのメモリが非常に少ないです。それが、MicroPython が主要な Python の機能を維持しつつ、これらのマイクロコントローラボードで動作する新しい機能を多数追加して進化した理由です。

次に、Raspberry Pi Pico に MicroPython をインストールする方法を学びます。

- 参照： MicroPython - Wikipedia
- 参照： realpython

4.2 1.2 Thonny IDE をインストール

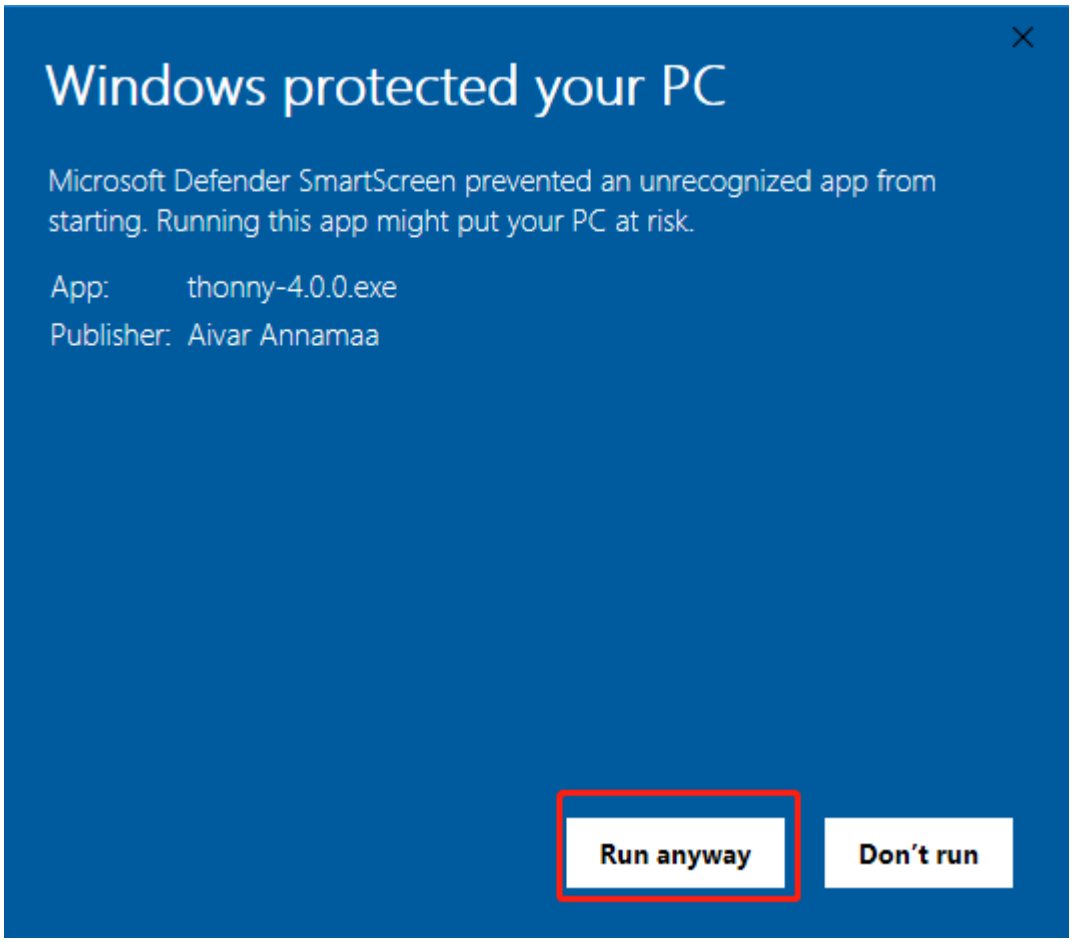
Raspberry Pi Pico で MicroPython プログラミングを始める前に、統合開発環境（IDE）が必要です。ここでは、Thonny をお勧めします。Thonny には Python 3.7 が内蔵されており、簡単なインストーラーを使えばすぐにプログラミングの学習を開始できます。

注釈: Raspberry Pi Pico のインタープリタは Thonny バージョン 3.3.3 以降でしか動作しないため、既にそれを持っている場合はこの章をスキップしても構いません。それ以外の場合は、更新またはインストールをお願いします。

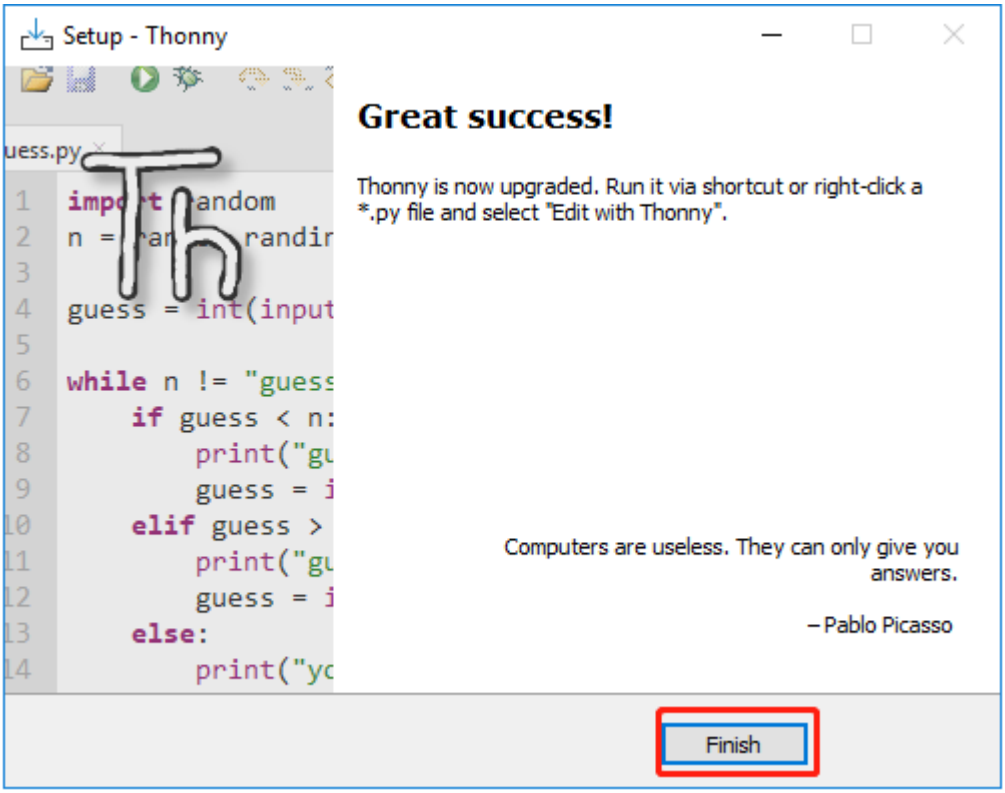
---

1. のウェブサイトにアクセスしてダウンロードします。ページを開いたら、右上隅に薄灰色のボックスが表示されるので、ご使用のオペレーティングシステムに適用されるリンクをクリックしてください。

2. インストーラは、まだ評判を築いていない新しい証明書で署名されています。ブラウザの警告（例：Chrome で「破棄」の代わりに「保持」を選択）や Windows Defender の警告（ **詳細情報** ⇒ **実行** ）をクリックして進む場合があります。



3. 次に、 **次へ** と **インストール** をクリックして、Thonny のインストールを完了します。



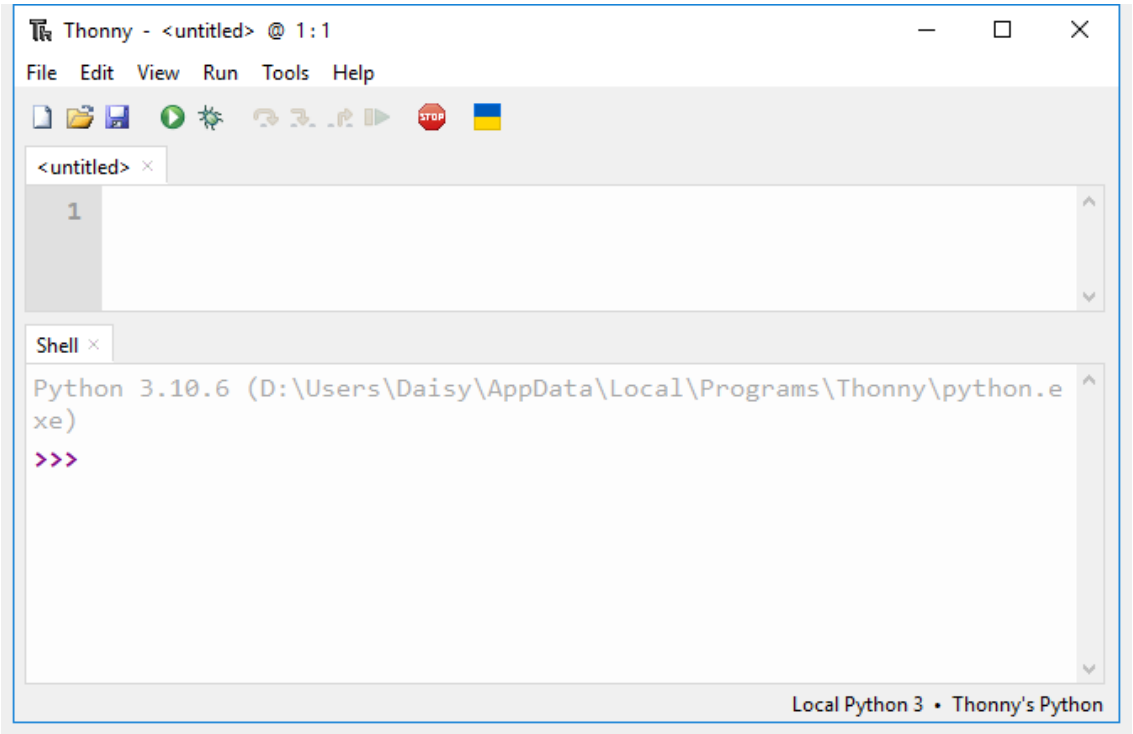
4.3 1.3 Raspberry Pi Pico に MicroPython をインストール

Thonny IDE を使って、Raspberry Pi Pico に MicroPython を簡単に一回でインストールしましょう。

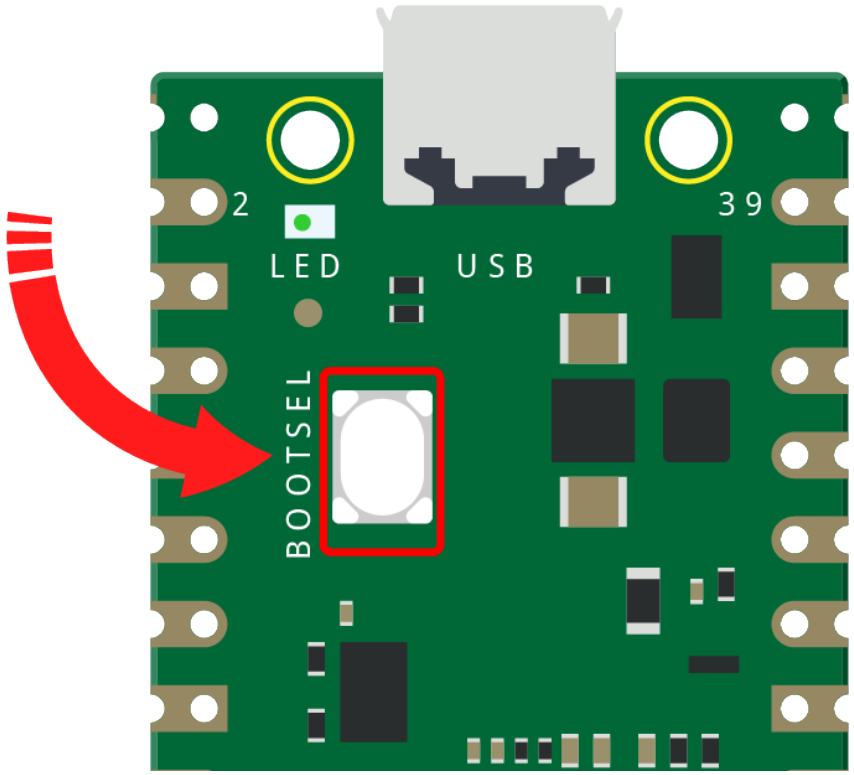
注釈: Thonny をアップグレードしたくない場合、Raspberry Pi 公式の を使って、 `rp2_pico_xxxx.uf2` ファイルを Raspberry Pi Pico にドラッグアンドドロップすることもできます。

---

1. Thonny IDE を開きます。

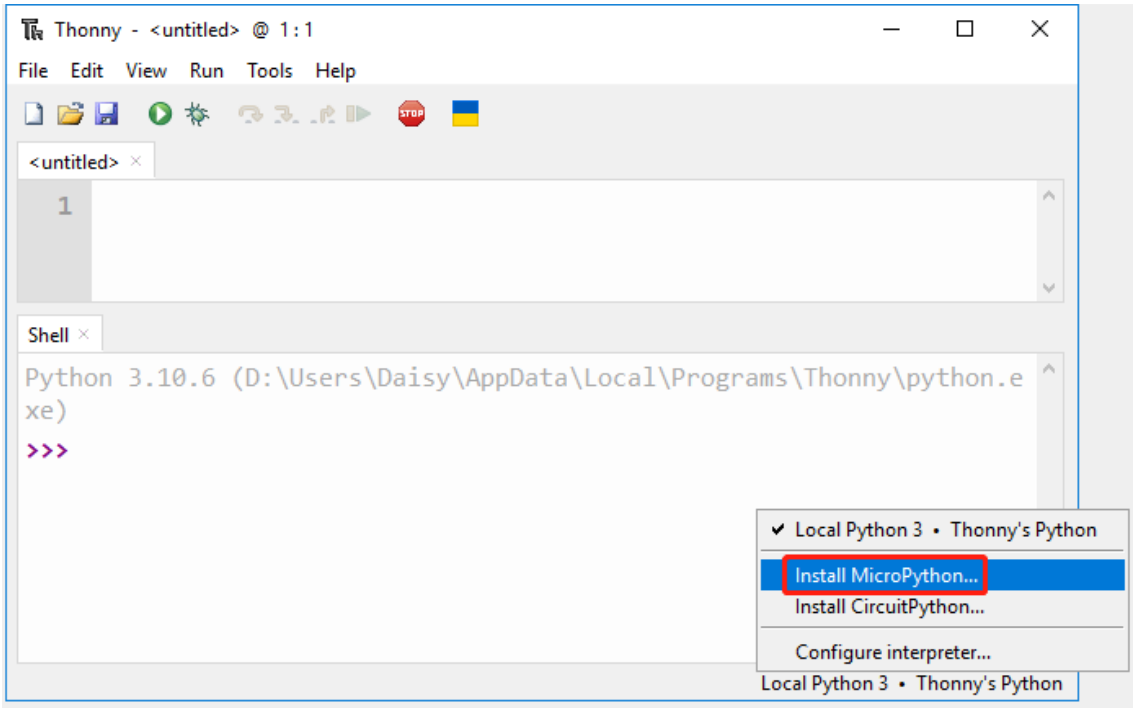


2. **BOOTSEL** ボタンを押しながら、Micro USB ケーブルで Pico をコンピューターに接続します。Pico が **RPI-RP2** という名前の大容量ストレージデバイスとしてマウントされたら、 **BOOTSEL** ボタンを離します。

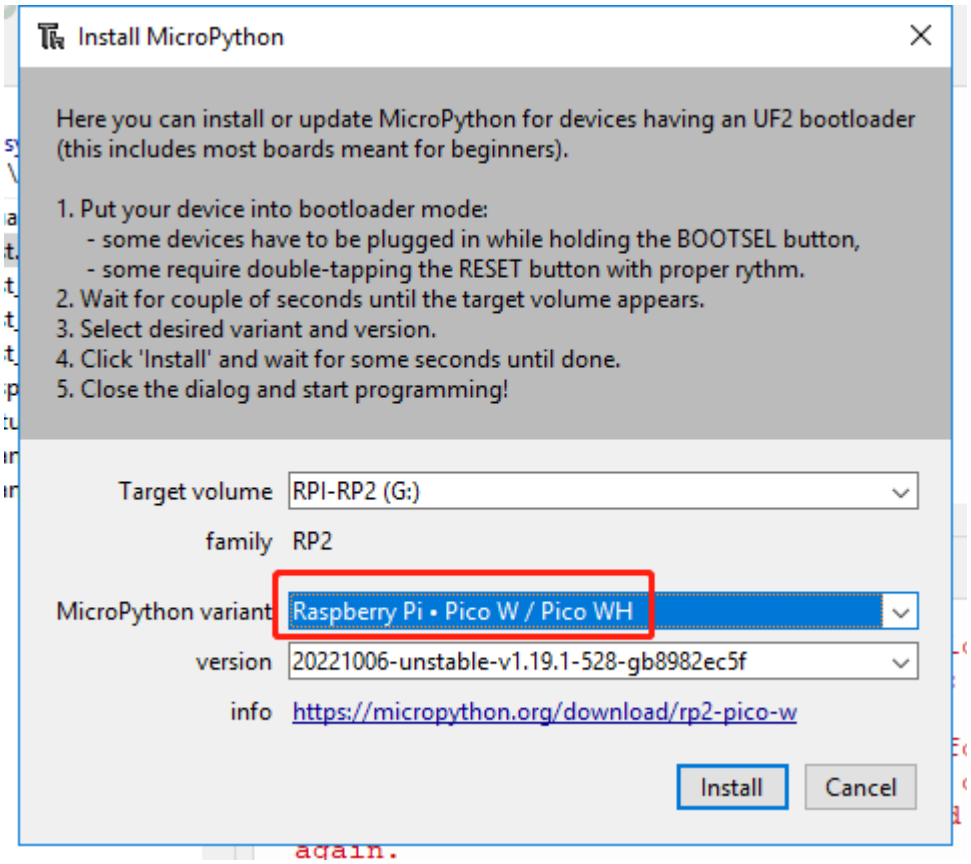


3. 右下隅で、インタープリタ選択ボタンをクリックし、 **MicroPython をインストール** を選択します。

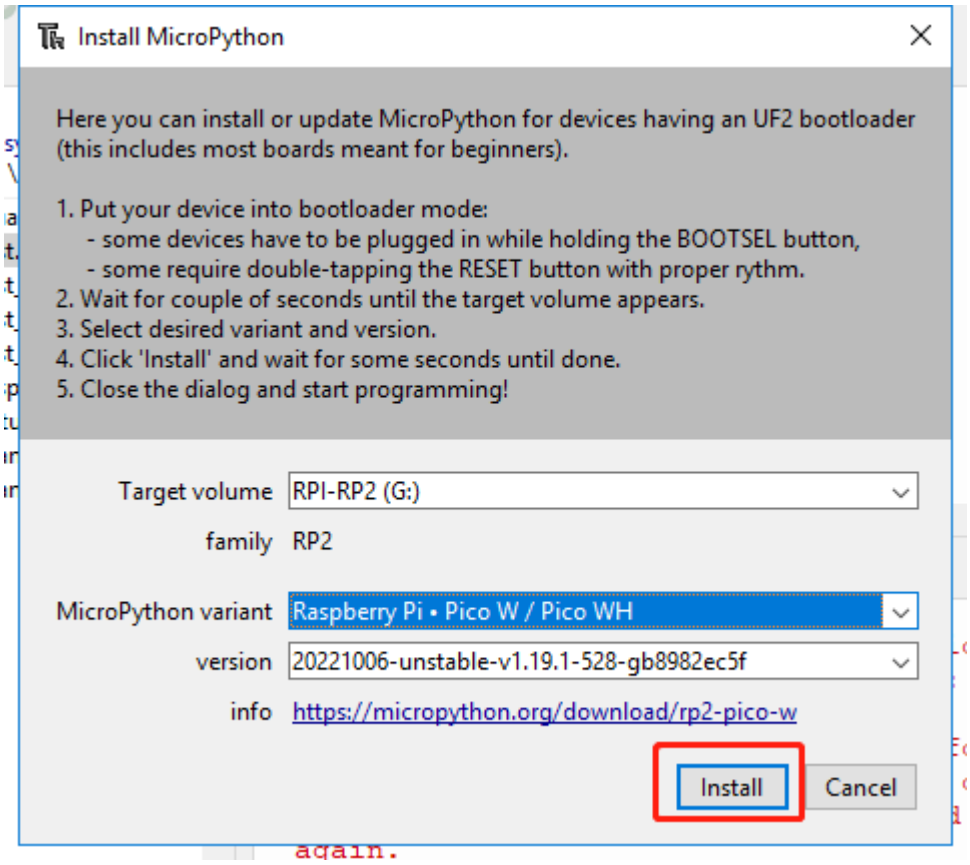
注釈: もし Thonny にこのオプションがない場合、最新バージョンに更新してください。



4. **対象のボリューム** で、先ほど挿入した Pico のボリュームが自動的に表示されます。 **MicroPython のバリアント** で、 **Raspberry Pi.Pico/Pico H** を選択します。



5. **インストール** ボタンをクリックし、インストールが完了するまで待ってからこのページを閉じます。



おめでとうございます、あなたの Raspberry Pi Pico は使用準備が整いました。

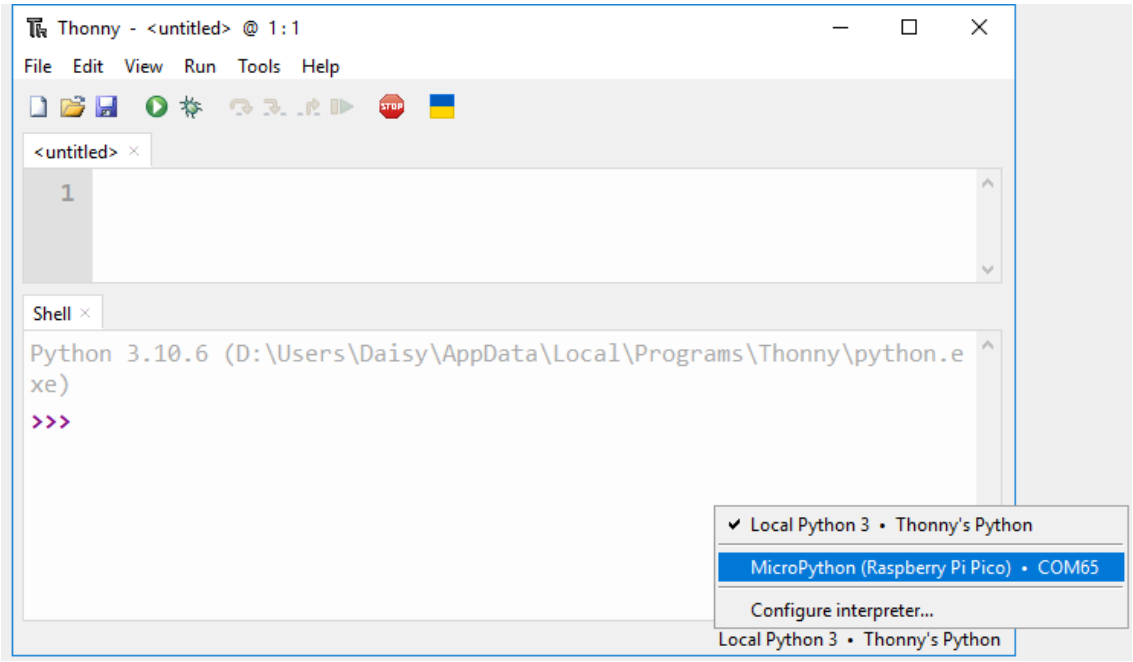
4.4 1.4 Pico にライブラリをアップロード

いくつかのプロジェクトでは、追加のライブラリが必要になることがあります。そこで、最初にこれらのライブラリを Raspberry Pi Pico W にアップロードし、後でコードを直接実行できるように設定します。

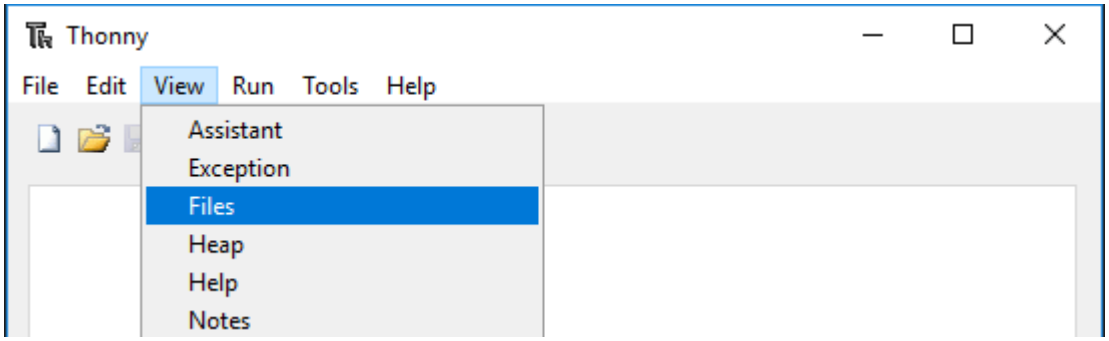
1. 下記のリンクから関連するコードをダウンロードします。

    - `SunFounder Kepler Kit`

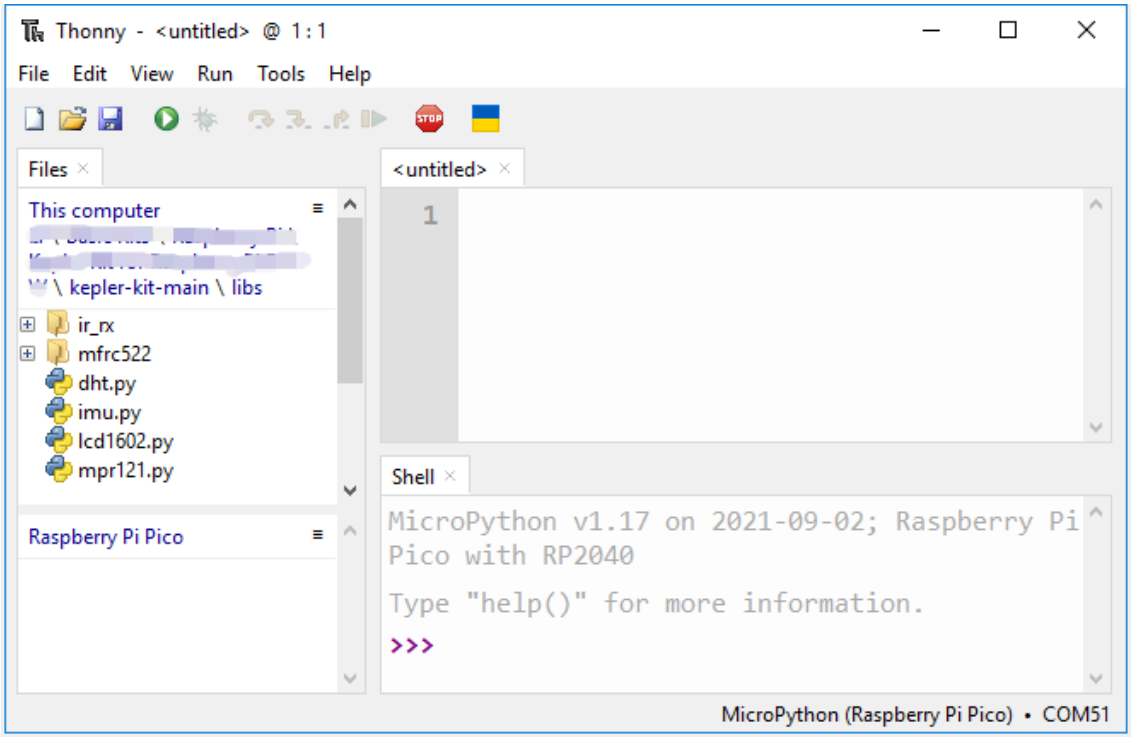
2. Thonny IDE を開き、マイクロ USB ケーブルで Pico をコンピュータに接続します。そして、右下隅にある「MicroPython（Raspberry Pi Pico）.COMXX」インタプリターをクリックします。



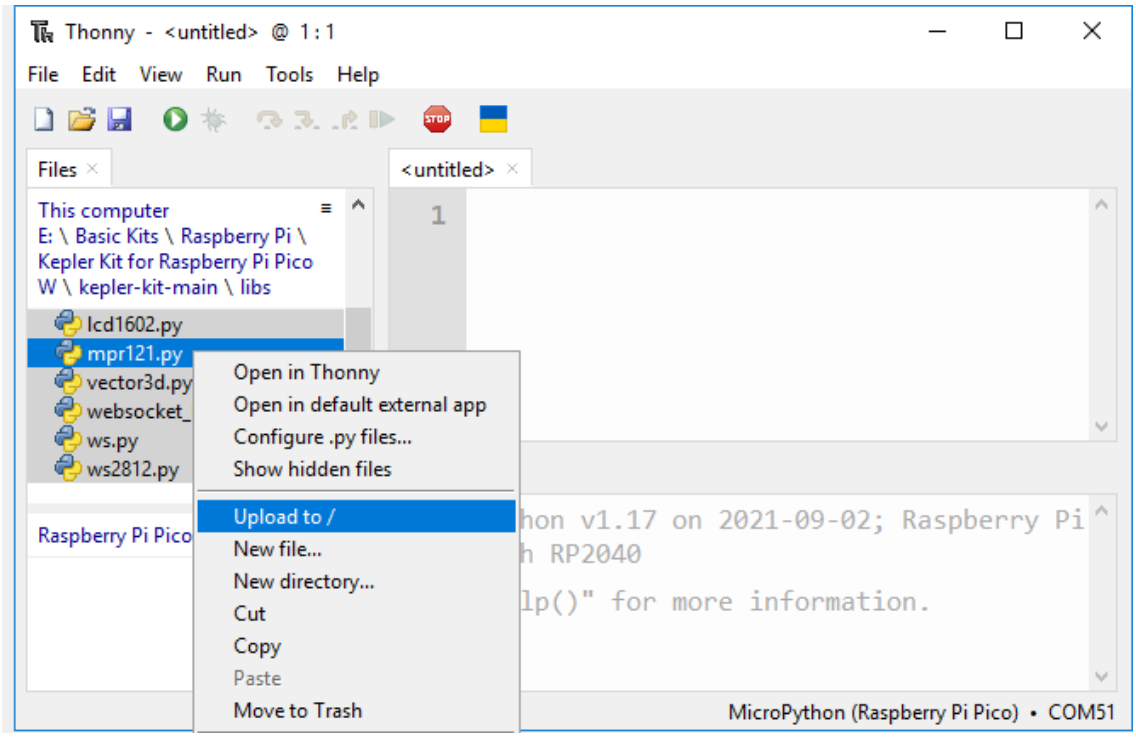
3. 上部ナビゲーションバーで、 **表示** -> **ファイル** を選択します。



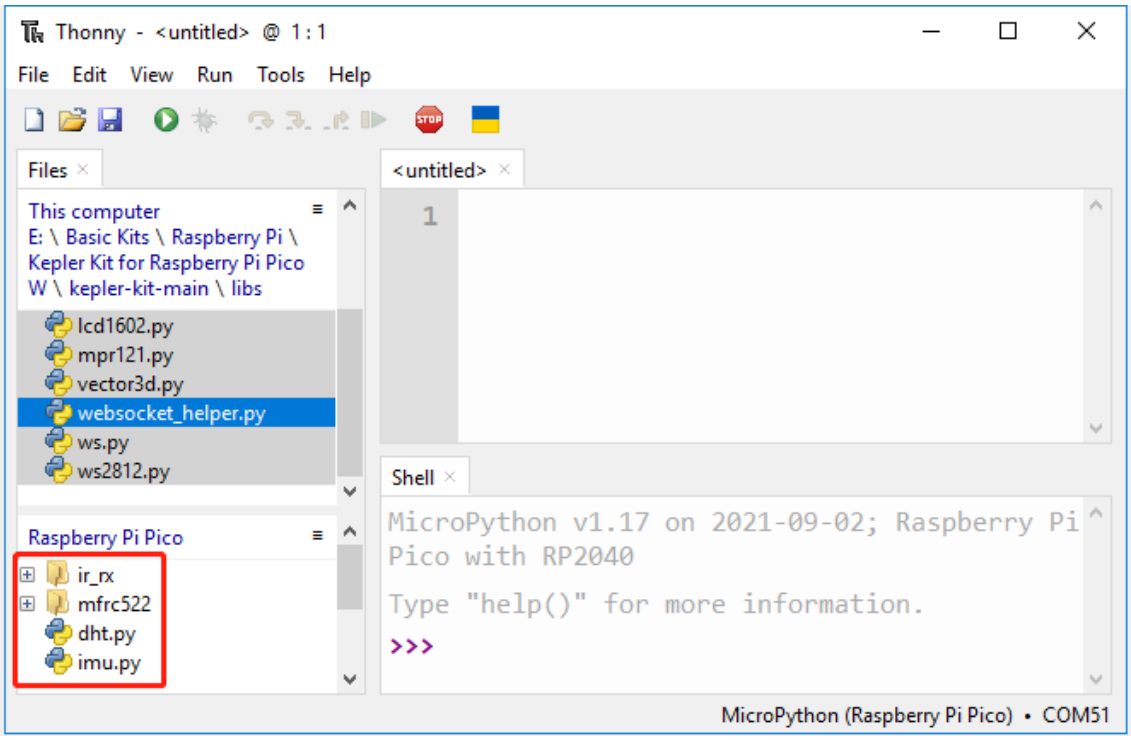
4. 以前ダウンロードした コードパッケージ が保存されているフォルダへとパスを切り替え、`kepler-kit-main/libs` フォルダに進みます。



5. `libs/` フォルダ内の全てのファイルまたはフォルダーを選択し、右クリックして **アップロード先** を選択します。アップロードには少し時間がかかります。



6. これで、アップロードしたばかりのファイルがドライブ内の「Raspberry Pi Pico」で確認できるようになります。



4.5 1.5 Thonny のクイックガイド

4.5.1 コードを直接開いて実行する

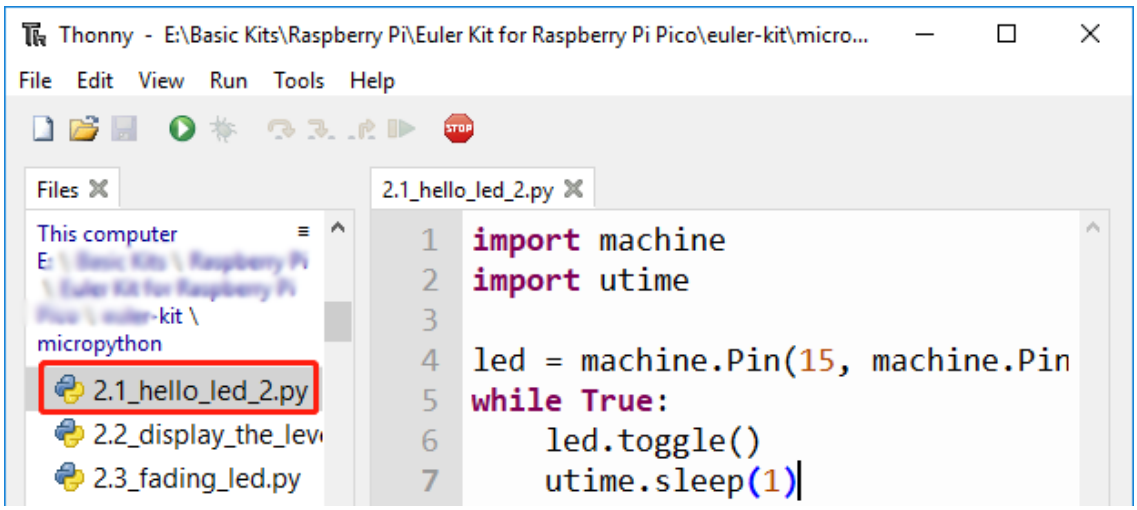
プロジェクトのコードセクションでは、どのコードが使用されているかが正確に示されています。したがって、`kepler-kit-main/micropython/` パス内のシリアル番号がついた `.py` ファイルをダブルクリックして開きます。

ただし、最初に *1.4 Pico* **にライブラリをアップロード** で説明されているように、パッケージをダウンロードしてライブラリをアップロードする必要があります。

1. コードを開く。

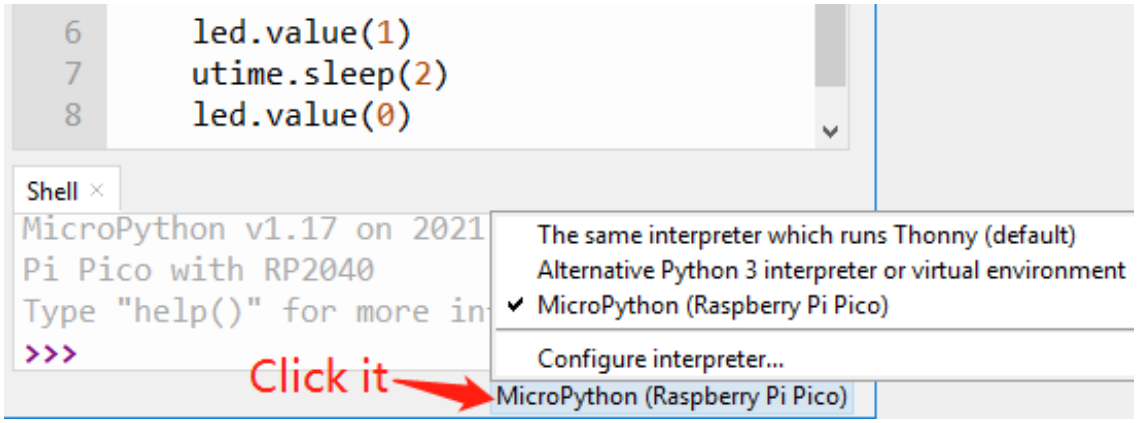
   例： `2.1_hello_led.py`

   ダブルクリックすると、右側に新しいウィンドウが開きます。同時に複数のコードを開くことができます。



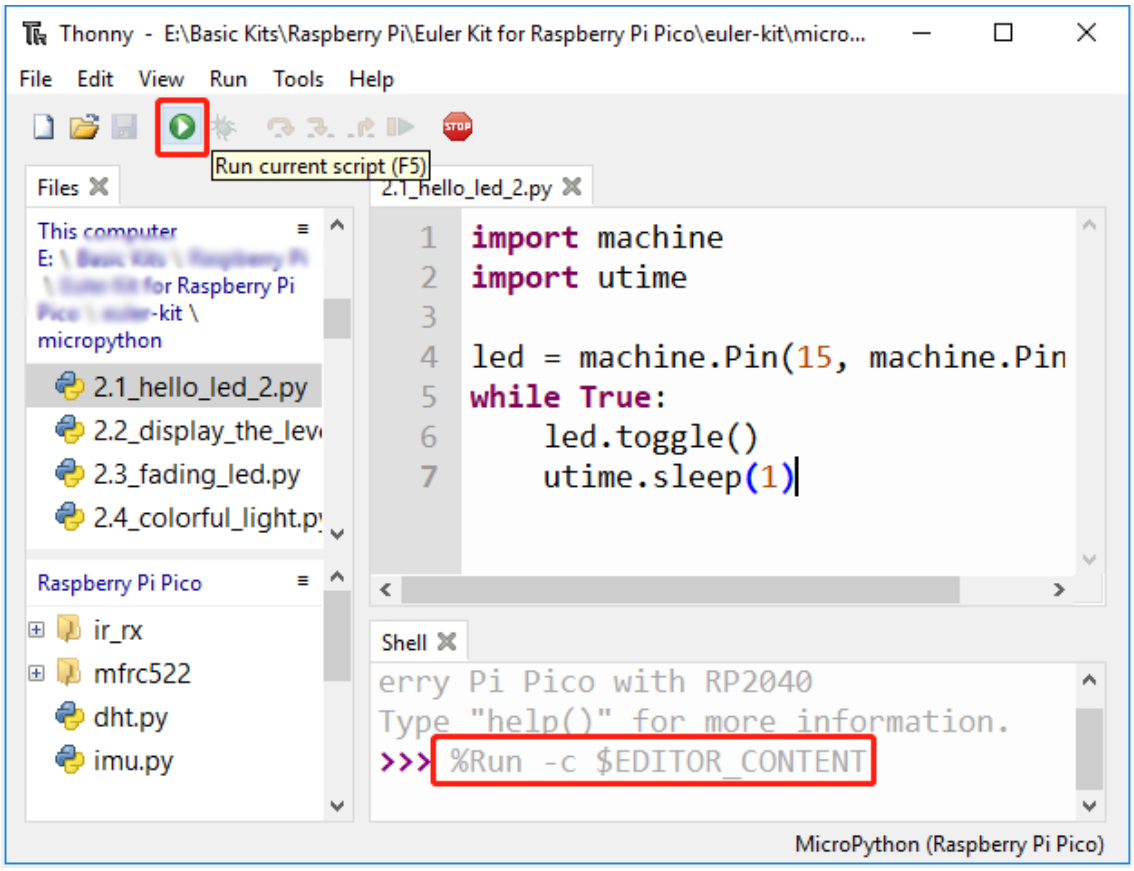
2. 正しいインタプリターを選択

マイクロ USB ケーブルを使用して、Pico W をコンピューターに接続し、「MicroPython（Raspberry Pi Pico）」インタプリターを選択します。



3. コードを実行する

スクリプトを実行するには、 **現在のスクリプトを実行** ボタンをクリックするか、F5 を押します。



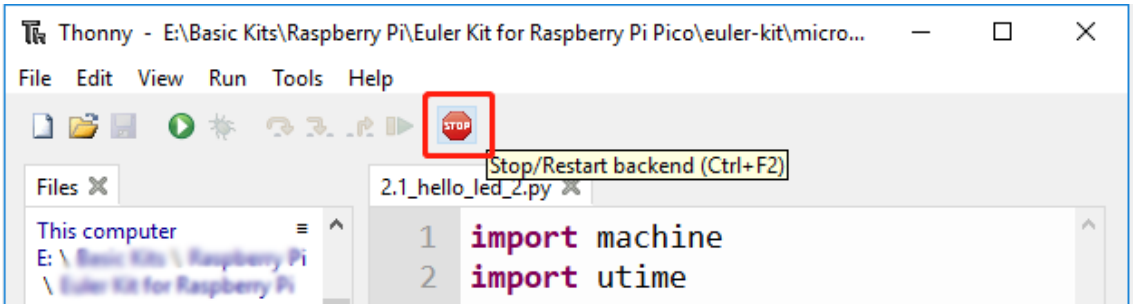
コードに出力する必要のある情報が含まれている場合、それはシェルに表示されます。そうでなければ、以下の情報のみが表示されます。

Thonny でシェルウィンドウが表示されない場合は、 **表示** -> **編集** をクリックしてシェルウィンドウを開きます。

```
MicroPython vx.xx on xxxx-xx-xx; Raspberry Pi Pico W  With RP2040

Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

- 最初の行は MicroPython のバージョン、日付、およびデバイス情報を示しています。
- 2 行目は「help()」を入力してヘルプを取得するように促しています。
- 3 行目は、Thonny からのコマンドで、Pico W 上の MicroPython インタプリターにスクリプトエリアの内容（「EDITOR_CONTENT」）を実行するように指示しています。
- 3 行目以降にメッセージがある場合、通常は MicroPython に出力するように指示したメッセージか、コードのエラーメッセージです。

4. 実行を停止する

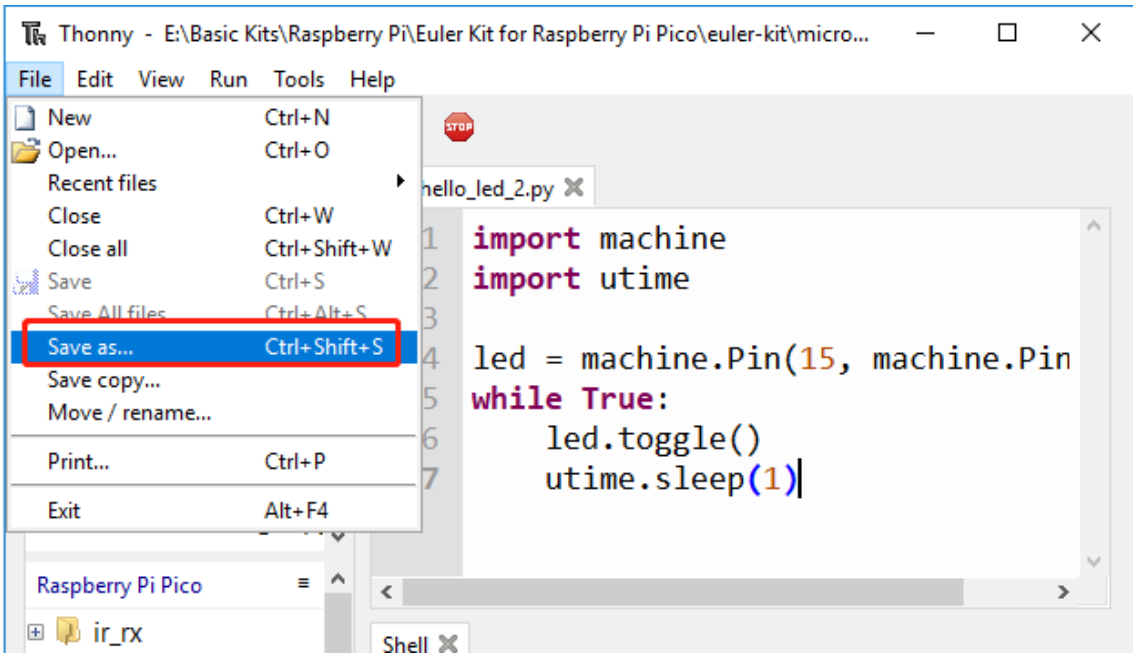


実行中のコードを停止するには、 **Stop/Restart バックエンド** ボタンをクリックします。 **%RUN -c $EDITOR_CONTENT** コマンドは停止後に消えます。

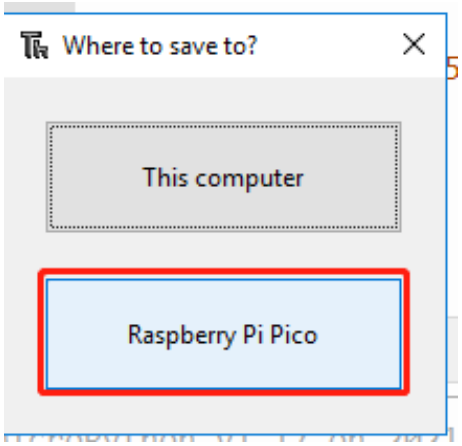
5. 保存または名前を付けて保存する

開いている例に行った変更を保存するには、 **Ctrl + S** を押すか、Thonny の **保存** ボタンをクリックします。

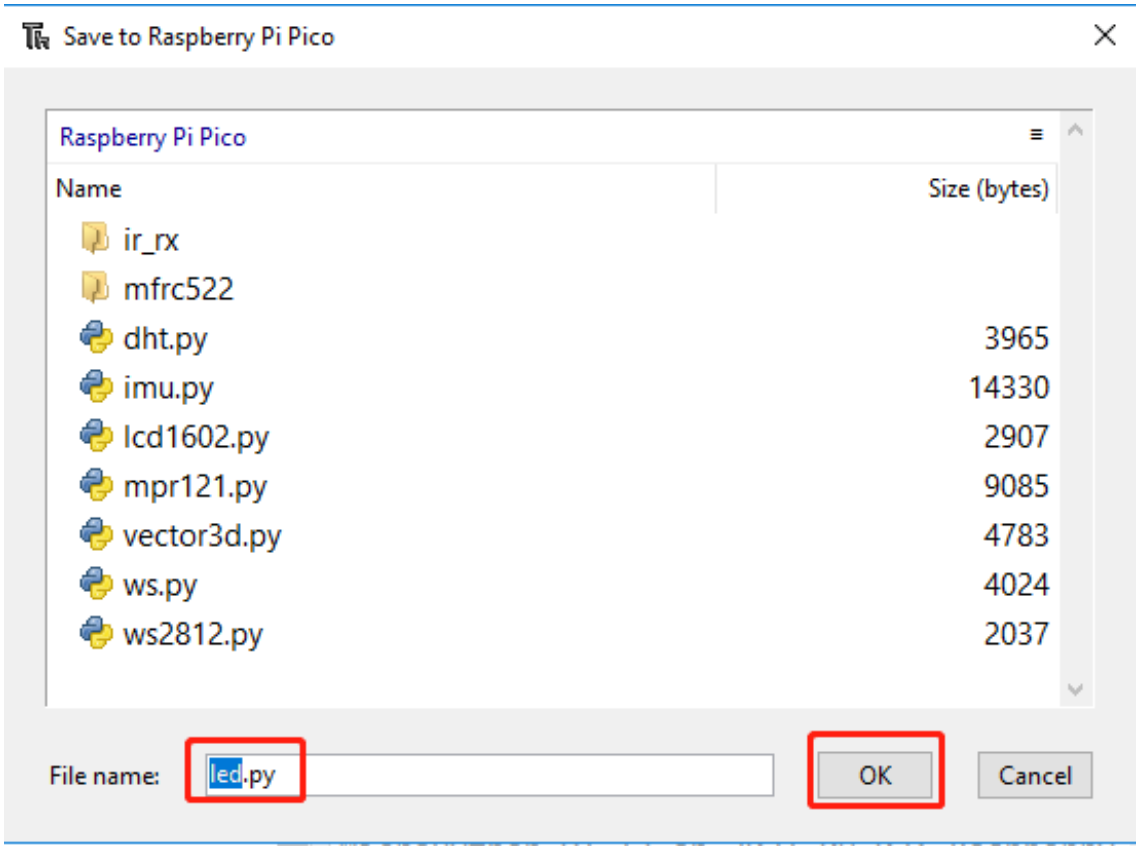
コードは、 **ファイル -> 名前を付けて保存** をクリックすることで、Raspberry Pi Pico W 内の別のファイルとして保存できます。



Raspberry Pi Pico を選択します。



ファイル名と拡張子 **.py** を入力した後、 **OK** をクリックします。Raspberry Pi Pico W ドライブに、保存したファイルが表示されます。



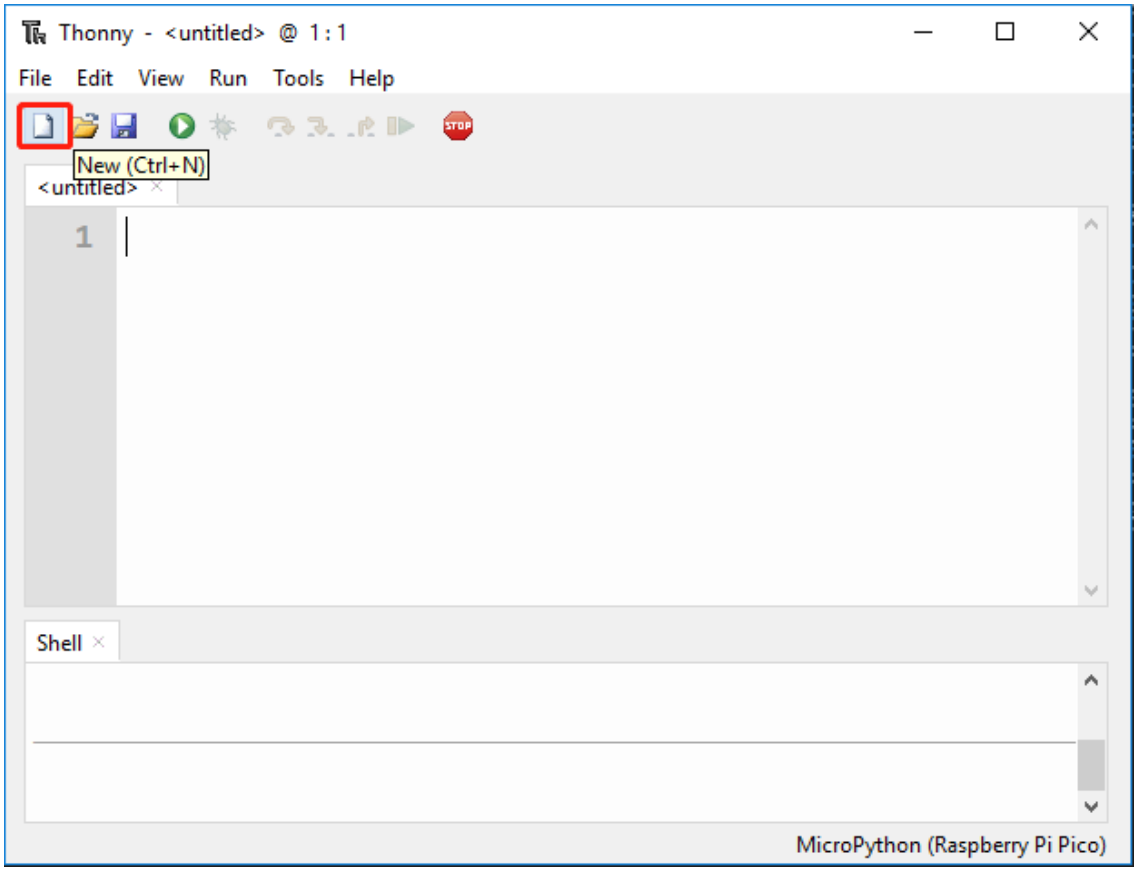
注釈: コードにどのような名前を付けるかに関わらず、それが何のコードであるかを説明する名前が最良です。意味のない名前（例： `abc.py` ）を付けないでください。コードを `main.py` として保存すると、電源を入れたときに自動的に実行されます。

4.5.2 新しいファイルを作成して実行する

コードはコードセクションに直接表示されます。これを Thonny にコピーして次のように実行できます。

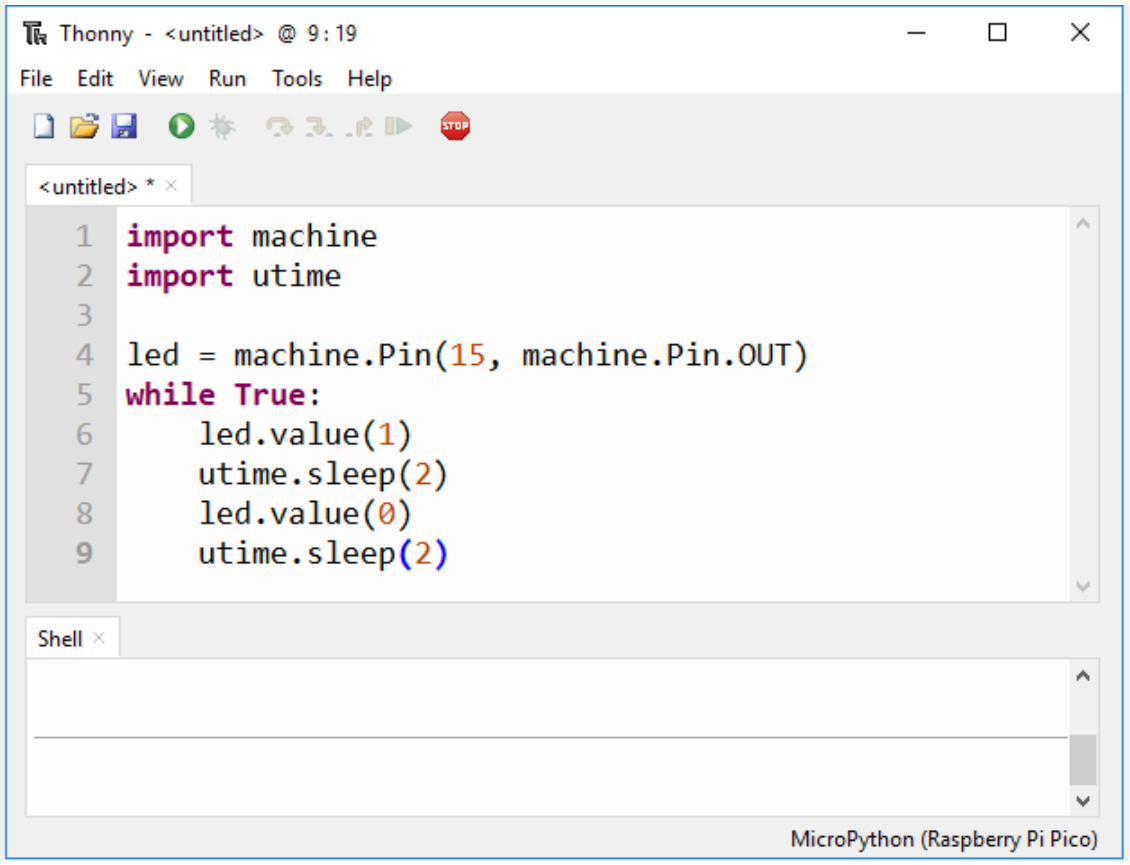
1. 新しいファイルを作成する

   Thonny IDE を開き、 **新規作成** ボタンをクリックして新しい空のファイルを作成します。



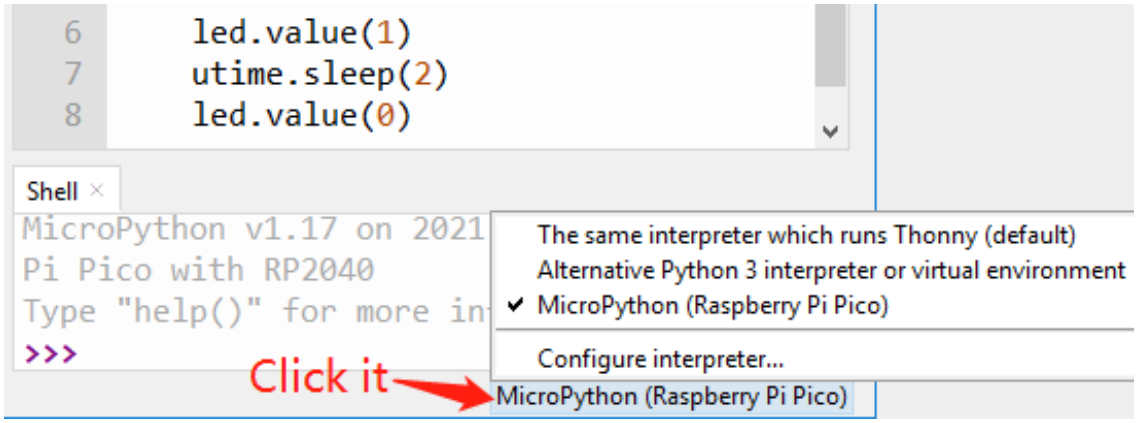
2. コードをコピーする

   プロジェクトから Thonny IDE にコードをコピーします。



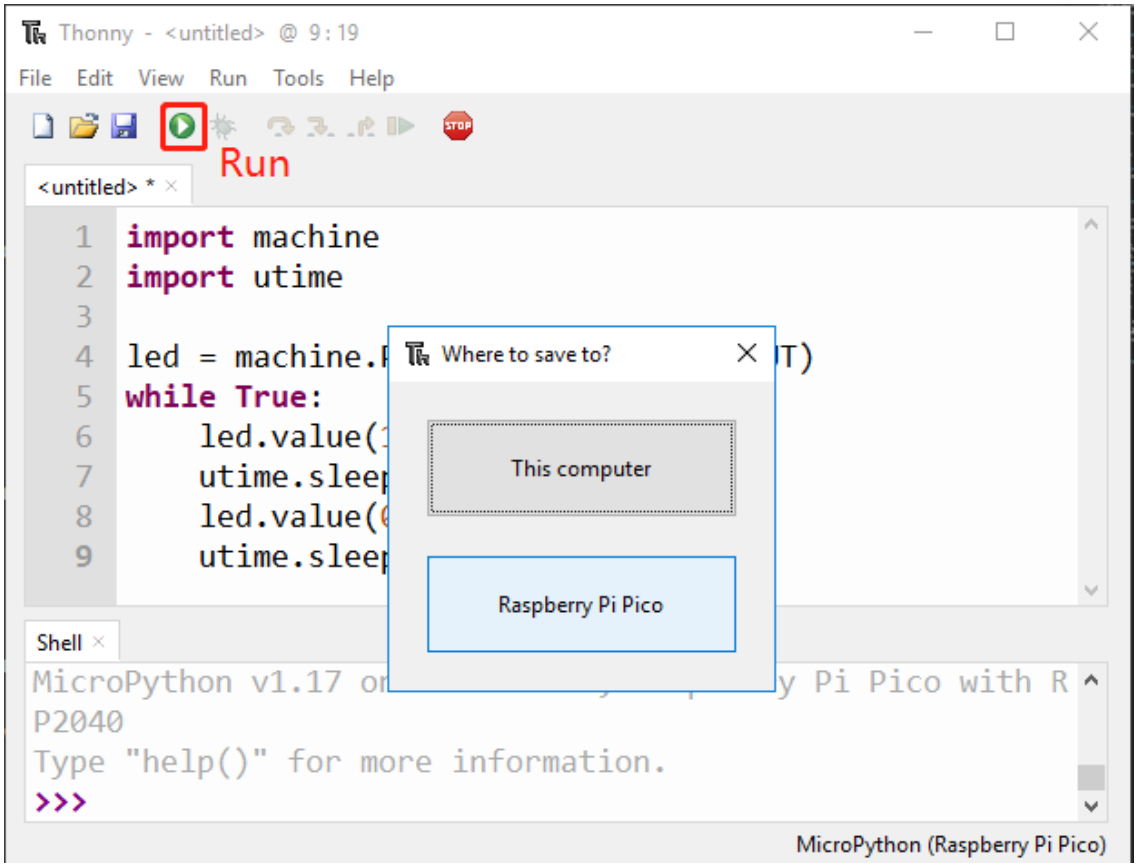
3. 正しいインタプリターを選択する

   マイクロ USB ケーブルで Pico W をコンピューターに接続し、右下隅で「MicroPython（Raspberry Pi Pico）」インタプリターを選択します。



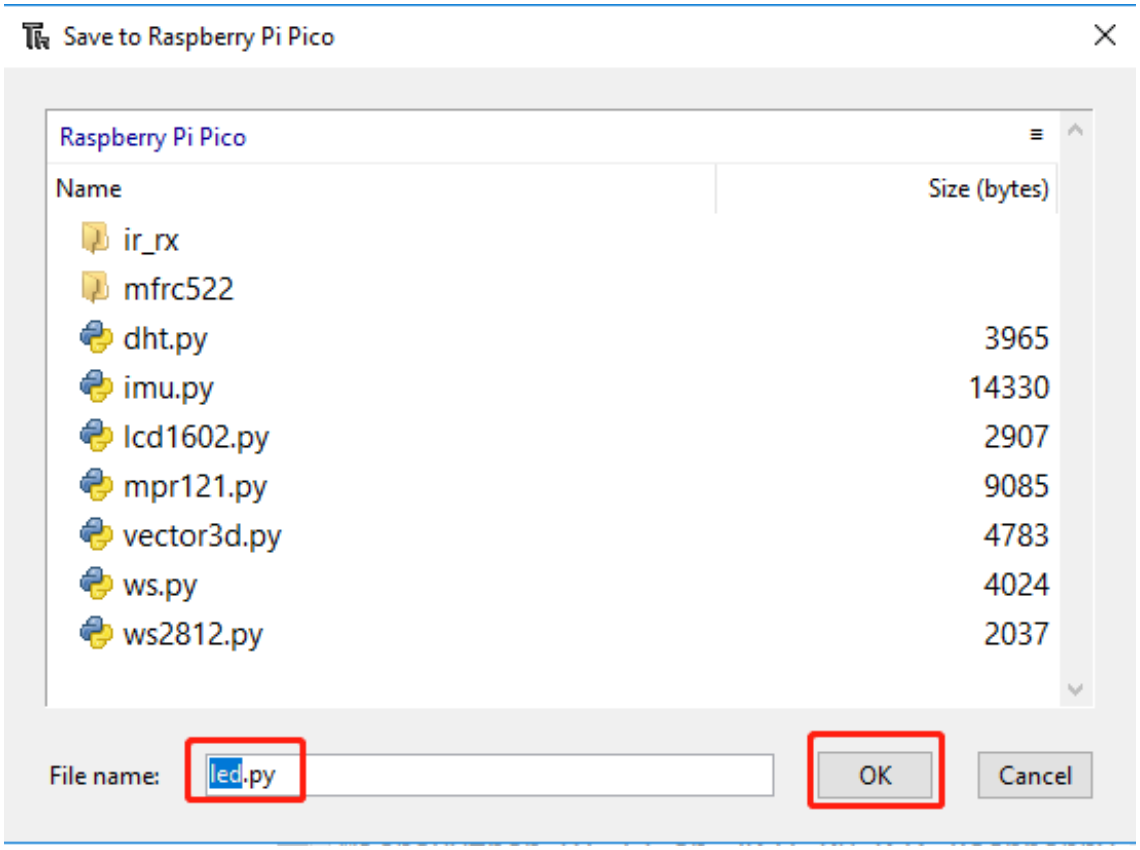
4. コードを実行して保存する

   実行するには、 **現在のスクリプトを実行** をクリックするか、F5 を押してください。コードが保存されていない場合、保存先として **このコンピューター** または **Raspberry Pi Pico** を選ぶポップアップウィンドウが表示されます。



注釈: Thonny は、指示した場合に Raspberry Pi Pico W にプログラムを保存します。したがって、Pico W を抜いて他の人のコンピューターに接続しても、プログラムはそのままです。

保存場所を選択し、ファイルに名前を付けて拡張子 **.py** を追加した後、OK をクリックします。



注釈: コードにどのような名前を付けるかに関わらず、それが何のコードであるかを説明する名前が最良です。意味のない名前（例： `abc.py` ）を付けないでください。コードを `main.py` として保存すると、電源を入れたときに自動的に実行されます。

プログラムが保存されると、自動的に実行され、シェルエリアに以下の情報が表示されます。

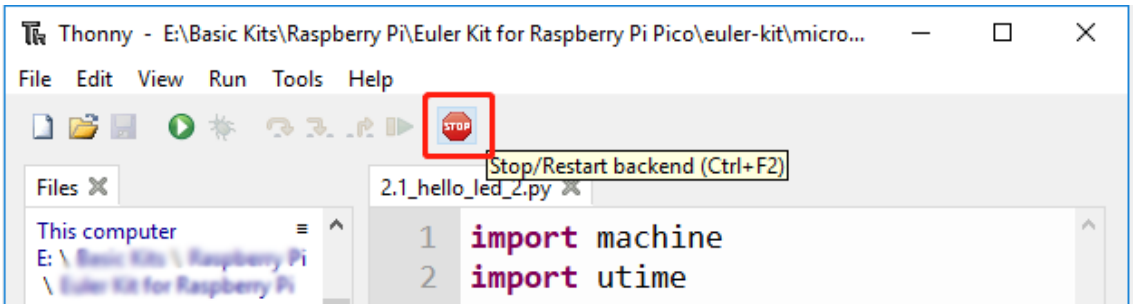
Thonny でシェルウィンドウが表示されない場合は、 **表示** -> **編集** をクリックしてシェルウィンドウを開きます。

```
MicroPython vx.xx.x on xxxx-xx-xx; Raspberry Pi Pico W With RP2040

Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

- 最初の行は MicroPython のバージョン、日付、およびデバイス情報を示しています。
- 2 行目は「help()」を入力してヘルプを取得するように促しています。
- 3 行目は、Thonny からのコマンドで、Pico W 上の MicroPython インタプリターにスクリプトエリアの内容（「EDITOR_CONTENT」）を実行するように指示しています。

- 3 行目以降にメッセージがある場合、通常は MicroPython に出力するように指示したメッセージか、コードのエラーメッセージです。

5. 実行を停止する



実行中のコードを停止するには、 **Stop/Restart バックエンド** ボタンをクリックします。 **%RUN -c $EDITOR_CONTENT** コマンドは停止後に消えます。

6. ファイルを開く

保存したコードファイルを開くには、2 つの方法があります。

- 第一の方法は、Thonny のツールバーにある開くアイコンをクリックすることです。プログラムを保存するときと同様に、それを **このコンピューター** から開くか **Raspberry Pi Pico** から開くかを尋ねられます。例えば、 **Raspberry Pi Pico** をクリックすると、Pico W に保存したすべてのプログラムの一覧が表示されます。
- 第二の方法は、 **表示** -> **ファイル** -> をクリックして、対応する `.py` ファイルをダブルクリックして直接ファイルプレビューを開くことです。

4.6 1.6 （オプション）MicroPython の基本構文

4.6.1 インデント

インデントとは、コード行の先頭にあるスペースのことです。標準的な Python プログラムと同様に、MicroPython プログラムも通常は上から下に実行されます：それは一行ずつ解釈され、インタープリタで実行され、次の行に進むのです。まるでシェルで一行ずつ入力するかのように。ただし、命令リストを一行ずつ見ていくだけのプログラムはあまり賢くありません。そのため、MicroPython も Python と同様に、プログラムの実行順序を制御する独自の方法があります：それがインデントです。

print() の前には少なくとも 1 つのスペースを入れなければならず、そうでないと「Invalid syntax（無効な構文）」というエラーメッセージが表示されます。通常は、Tab キーを一様に押すことでスペースを標準化することが推奨されます。

```
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スペースの数を揃えなければ、Python はエラーを出します。

```
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
```

4.6.2 コメント

コード内のコメントは、コードの理解を助け、全体のコードをより読みやすくするため、また、テスト中にコードの一部をコメントアウトして、その部分のコードが実行されないようにするためにあります。

単一行コメント

MicroPython における単一行のコメントは#で始まり、その後のテキストは行の末までコメントとみなされます。コメントはコードの前または後に配置することができます。

```
print("hello world")  #これは注釈です
```

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

コメントは、コードを説明するために使用されるテキストでなくても構いません。コードの一部をコメントアウトして、micropython がそのコードを実行しないようにすることもできます。

```
#print("これは実行されません！")
print("hello world") #これは注釈です
```

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

複数行コメント

複数行にわたるコメントを書きたい場合は、複数の#記号を使用できます。

```
#これはコメントです
#複数行に
#渡って書かれています
print("Hello, World!")
```

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

または、予想される代わりに複数行の文字列を使用することもできます。

MicroPython は、変数に割り当てられていない文字列リテラルを無視するため、コードに複数行の文字列（トリプルクォート）を追加し、それらにコメントを入れることができます。

```
"""
これはコメントです
複数行に
渡って書かれています
"""
print("Hello, World!")
```

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

文字列が変数に割り当てられていない限り、MicroPython はコードを読み取った後にそれを無視し、複数行のコメントを作成したかのように扱います。

4.6.3 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表示します。メッセージは文字列でも、その他のオブジェクトでも構いません。オブジェクトは、画面に書き出される前に文字列に変換されます。

複数のオブジェクトを出力する：

```
print("ようこそ！", "楽しんでください！")
```

```
>>> %Run -c $EDITOR_CONTENT
ようこそ！楽しんでください！
```

タプルを出力する：

```
x = ("梨", "リンゴ", "ぶどう")
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
('梨', 'リンゴ', 'ぶどう')
```

二つのメッセージを出力し、セパレーターを指定する：

```
print("こんにちは", "お元気ですか?", sep="---")
```

```
>>> %Run -c $EDITOR_CONTENT
こんにちは---お元気ですか?
```

4.6.4 変数

変数はデータ値を格納するためのコンテナです。

変数を作成するのは非常に簡単です。名前を付けて値を割り当てるだけです。変数は参照であり、割り当てを通じて異なるデータ型のオブジェクトにアクセスするので、データ型を指定する必要はありません。

変数の命名には次のルールに従う必要があります：

- 変数名は数字、文字、アンダースコアしか含めることができません
- 変数名の最初の文字は文字またはアンダースコアでなければなりません
- 変数名は大文字と小文字を区別します

変数の作成

MicroPython には変数を宣言するコマンドはありません。初めて値を割り当てたときに変数が作成されます。特定の型宣言を使用する必要はありませんし、変数を設定した後に型を変更することもできます。

```
x = 8       # x は int 型です
x = "lily"  # x は現在 str 型です
print(x)
```

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

型変換（キャスティング）

変数のデータ型を明示的に指定したい場合は、キャスティングを使用できます。

```
x = int(5)    # y は 5 になります
y = str(5)    # x は'5' になります
z = float(5)  # z は 5.0 になります
print(x, y, z)
```

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

型の取得

type() 関数を使用して変数のデータ型を取得できます。

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

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

シングルクォートかダブルクォートか？

MicroPython では、文字列変数を定義するためにシングルクォートまたはダブルクォートを使用できます。

```
x = "hello"
# は
x = 'hello'
# と同じです
```

大文字・小文字の区別

変数名は大文字と小文字を区別します。

```
a = 5
A = "lily"
# A は a を上書きしません
print(a, A)
```

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

4.6.5 If Else

特定の条件が満たされた場合にのみコードを実行したいときに、判断（デシジョンメイキング）が必要です。

if

```
if テスト式:
    ステートメント
```

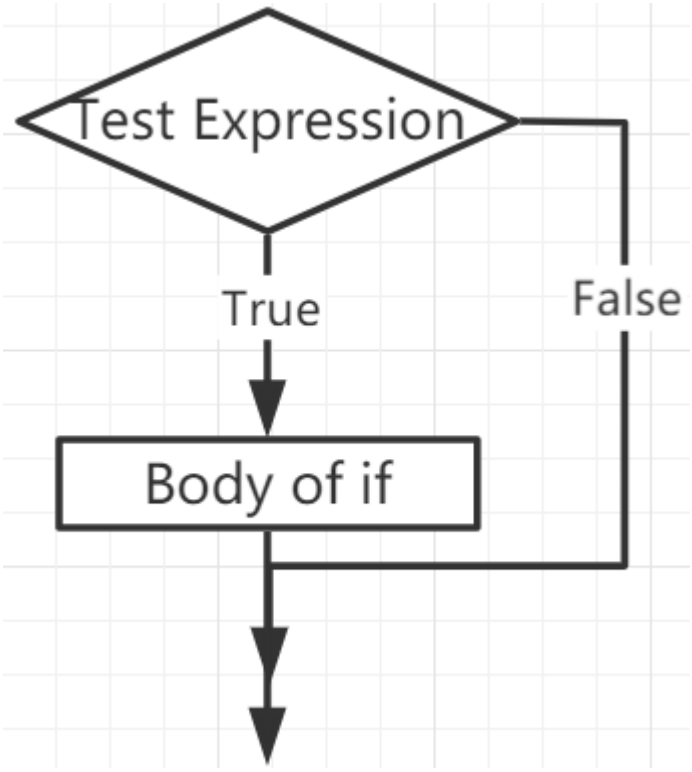
この場合、プログラムは **テスト式** を評価し、 **テスト式** が True である場合にのみ **ステートメント** を実行します。

テスト式 が False の場合、 **ステートメント** は実行されません。

MicroPython では、インデントは `if` ステートメントの本体を意味します。本体はインデントで始まり、最初の非インデントされた行で終わります。

Python は非ゼロの値を「True」と解釈します。None と 0 は「False」と解釈されます。

if 文のフローチャート



例

```
num = 8
if num > 0:
    print(num, "は正の数です。")
print("この行で終わり")
```

```
>>> %Run -c $EDITOR_CONTENT
8 は正の数です。
この行で終わり
```

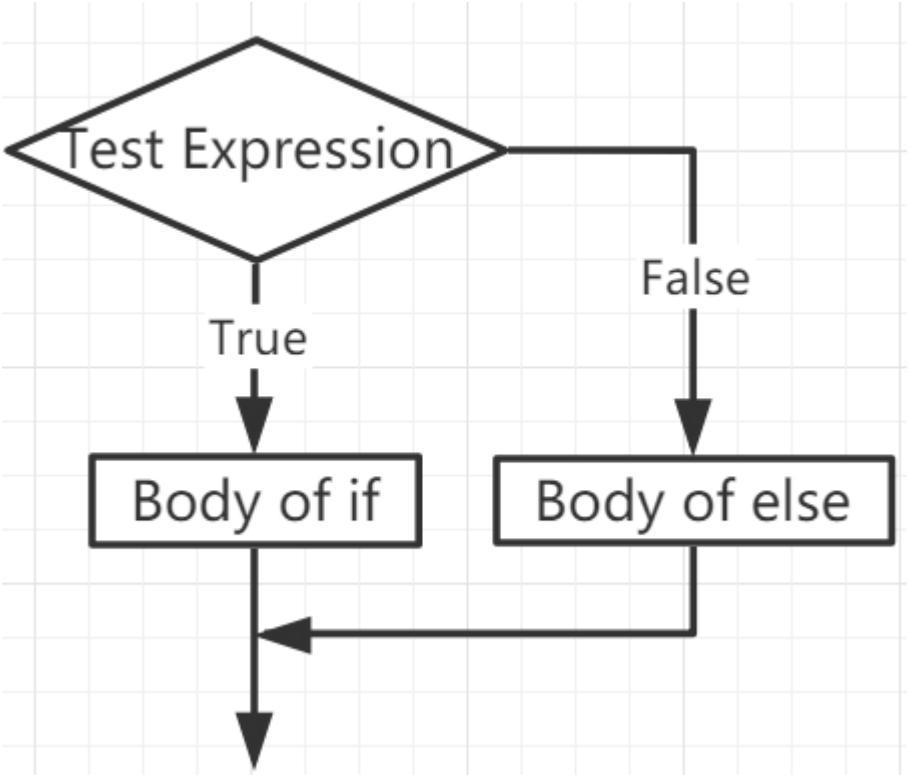
if...else

```
if テスト式:
    if の本体
else:
    else の本体
```

if..else ステートメントは **テスト式** を評価し、テスト条件が True の場合にのみ if の本体を実行します。

条件が False の場合、 else の本体が実行されます。インデントはブロックを区別するために使用されます。

if...else 文のフローチャート



例

```
num = -8
if num > 0:
    print(num, "は正の数です。")
else:
    print(num, "は負の数です。")
```

```
>>> %Run -c $EDITOR_CONTENT
-8 は負の数です。
```

if...elif...else

```
if テスト式:
    if の本体
elif テスト式:
    elif の本体
else:
    else の本体
```

Elif は else if の省略形です。複数の式をチェックすることができます。

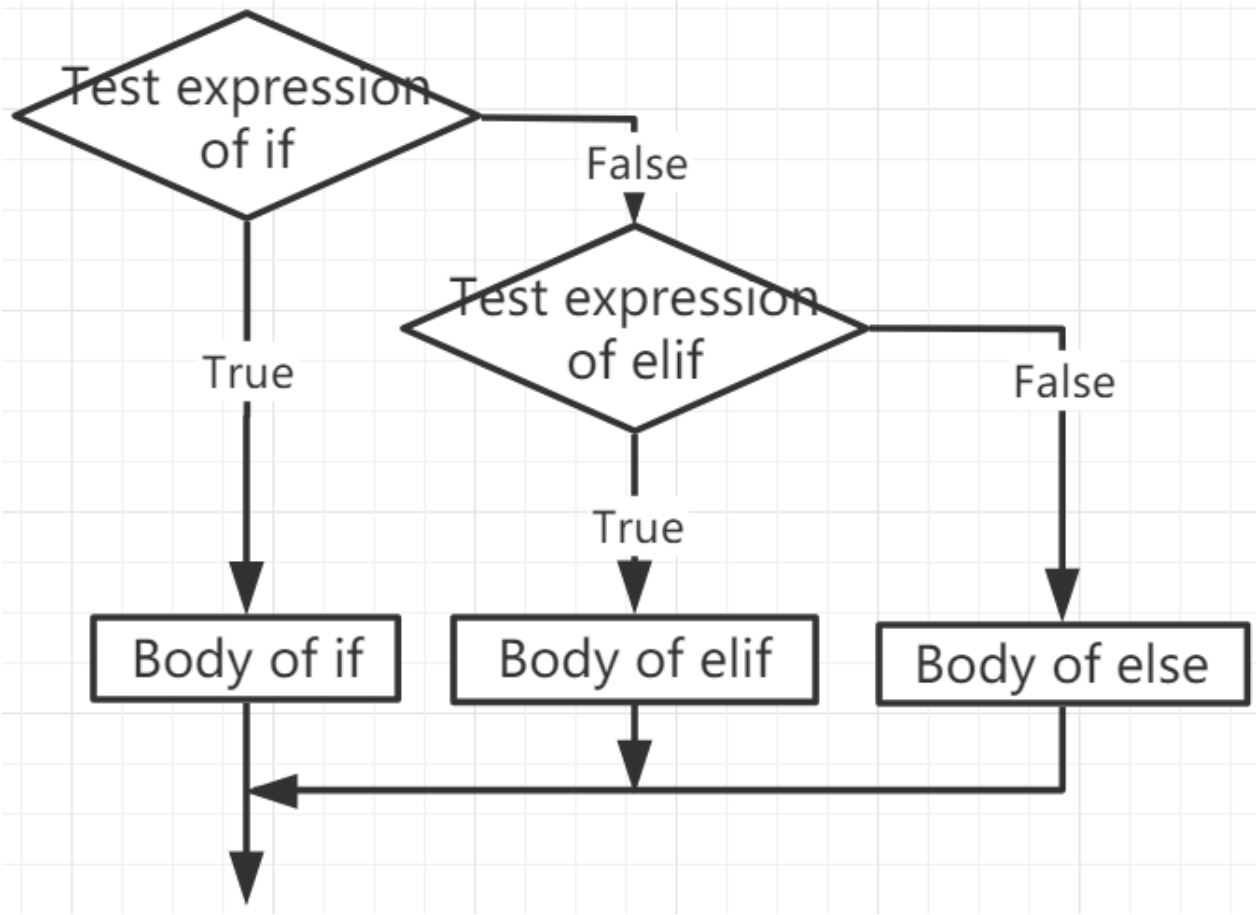
if の条件が False の場合、次の elif ブロックの条件がチェックされ、以降続きます。

すべての条件が False の場合、 else の本体が実行されます。

いくつかの if...elif...else ブロックのうち、条件に従って一つだけが実行されます。

if ブロックは一つの else ブロックしか持つことができませんが、複数の elif ブロックを持つことができます。

if...elif...else 文のフローチャート



例

```
x = 10
y = 9

if x > y:
    print("x は y より大きい")
elif x == y:
    print("x と y は等しい")
```

(次のページに続く)

(前のページからの続き)

```
else:
    print("x は y より大きい")
```

```
>>> %Run -c $EDITOR_CONTENT
x は y より大きい
```

ネストした if

if 文を別の if 文に埋め込むことができ、それをネストした if 文と呼びます。

例

```
x = 67

if x > 10:
    print("10 より上で、")
    if x > 20:
        print("さらに 20 よりも上です！")
    else:
        print("しかし 20 より上ではありません。")
```

```
>>> %Run -c $EDITOR_CONTENT
10 より上で、
さらに 20 よりも上です！
```

4.6.6 While ループ

`while` 文は、特定の条件下で同じタスクを繰り返し処理するためにプログラムをループで実行するために使用されます。

基本的な形式は以下の通りです：

```
while テスト式:
    while の本体
```

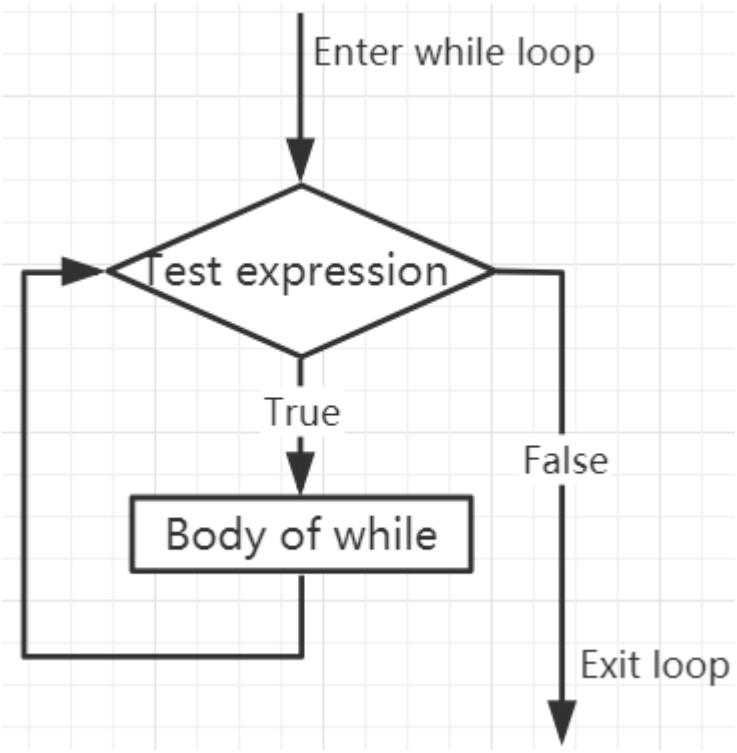
`while` ループでは、まず **テスト式** をチェックします。 **テスト式** が `True` と評価された場合のみ、while の本体に入ります。一回の反復後、再び **テスト式** をチェックします。このプロセスは、 **テスト式** が `False` と評価されるまで続きます。

MicroPython では、 `while` ループの本体はインデントによって決定されます。

本体はインデントで始まり、最初の非インデント行で終わります。

Python は、非ゼロの値を `True` と解釈します。None と 0 は `False` と解釈されます。

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
```

Break 文

break 文を使って、while 条件が true であってもループを停止することができます。

```
x = 10

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

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

Else を持つ While ループ

if ループと同様に、 while ループにもオプションで else ブロックを持たせることができます。

while ループの条件が False と評価された場合、 else 部分が実行されます。

```
x = 10

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

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

(次のページに続く)

(前のページからの続き)

```
3
2
1
Game Over
```

4.6.7 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走査することができます。

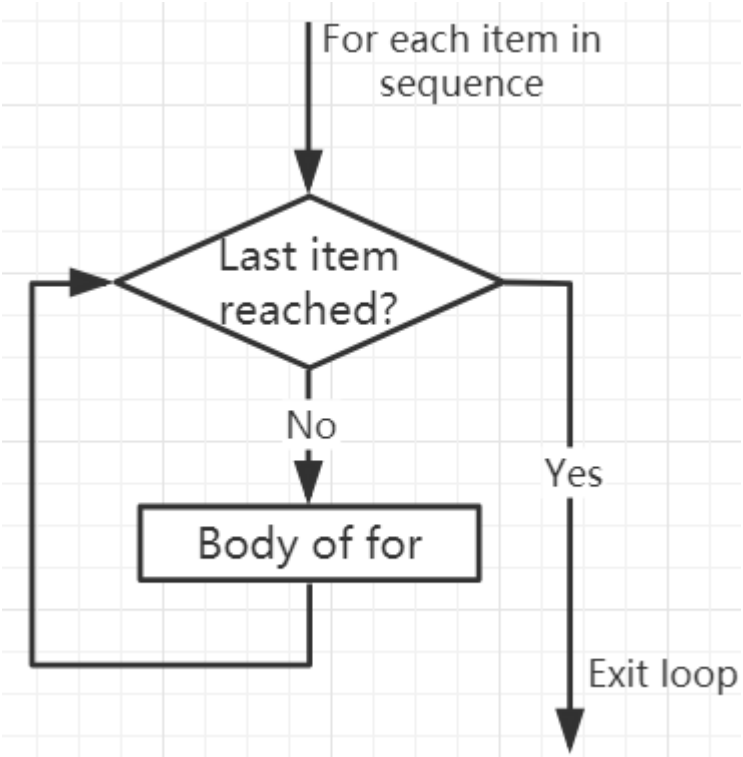
for ループの文法形式は以下の通りです：

```
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す。

ループは、シーケンス内の最後の項目に到達するまで続きます。 for ループの本体と残りのコードを区別するためにインデントを使用します。

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
```

break 文

break 文を使用すると、すべての項目をループ処理する前にループを停止することができます。

```
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
```

continue 文

continue 文を使用すると、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ます。

```
numbers = [1, 2, 3, 4]

for val in numbers:
    if val == 3:
        continue
    print(val)
```

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

range() 関数

range() 関数を使用して数値のシーケンスを生成することができます。range(6) は、0 から 5 まで（6 つの数値）を生成します。

また、開始値、終了値、ステップサイズを range(start, stop, step_size) として定義することもできます。step_size が指定されていない場合、デフォルトは 1 になります。

range オブジェクトは「遅延評価される」（lazy）と言えます。オブジェクトを生成した時点で、その中に「含まれる」すべての数値は生成されていません。ただし、これはイテレータではありません。なぜなら、in、len、そして __getitem__ 操作がサポートされているからです。

この関数はすべての値をメモリに保持していません。そうすると非効率的になります。したがって、開始値、終了値、ステップサイズを覚えておき、進行中に次の数値を生成します。

この関数がすべての項目を出力するように強制するには、list() 関数を使用することができます。

```
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]
```

for ループの中で range() を使用して数値のシーケンスを反復処理することができます。これは 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走査することもできます。

```
fruits = ['pear', 'apple', 'grape']

```

(次のページに続く)

(前のページからの続き)

```
for i in range(len(fruits)):
    print("I like", fruits[i])
```

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

For ループにおける Else

for ループには、オプションで else ブロックも追加することができます。ループで使用されるシーケンスの項目がすべて使い切られた場合、 else 部分が実行されます。

break キーワードを使用して for ループを停止することができます。この場合、 else 部分は無視されます。

したがって、何らかの中断が発生しない場合、 for ループの else 部分が実行されます。

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

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

break 文によってループが停止された場合、else ブロックは実行されません。

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

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

(次のページに続く)

(前のページからの続き)

```
0
1
```

4.6.8 関数

MicroPython において、関数（function）は特定のタスクを実行するための関連するステートメント（命令）のグループです。

関数はプログラムをより小さなモジュラーなブロックに分割するのに役立ちます。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たものにし、管理しやすくします。

さらに、コードの重複を避け、再利用可能にします。

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ます。
- 関数名は、関数を一意に識別するためのものです。関数名の命名規則は変数のそれと同じで、以下のルールに従います。
  - 数字、文字、アンダースコアのみを含めることができます。
  - 最初の文字は文字またはアンダースコアでなければなりません。
  - 大文字と小文字は区別されます。
- パラメータ（arguments）は、関数に値を渡すためのものです。これはオプションです。
- コロン（:）は、関数ヘッダの終わりを示します。
- オプションの docstring（ドキュメント文字列）は、関数の機能を説明するために使用されます。通常は三重引用符を使用して、docstring を複数行に拡張できるようにします。
- 関数の本体を構成する一つ以上の有効な MicroPython のステートメント。ステートメントは同じインデントレベル（通常は 4 つのスペース）を持つ必要があります。
- 各関数には少なくとも一つのステートメントが必要ですが、何らかの理由でステートメントを含まない関数がある場合は、エラーを避けるために pass 文を入れてください。
- 関数から値を返すためのオプションの `return` ステートメント。

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します。

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

my_function()
```

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

return ステートメント

return ステートメントは、関数を終了し、それが呼び出された場所に戻るために使用されます。

return の構文

```
return [expression_list]
```

このステートメントには、評価されて値が返される式を含めることができます。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に存在しない場合、関数は None オブジェクトを返します。

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

print(my_function())
```

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

ここでは、 None が返り値です。これは、 return ステートメントが使用されていないからです。

引数

情報は引数として関数に渡すことができます。

引数は、関数名の後の括弧内で指定します。必要な数だけ引数を追加でき、それらをコンマで区切ります。

```
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!
```

引数の数

デフォルトでは、関数は正確な数の引数で呼び出される必要があります。つまり、関数が 2 つのパラメータを期待している場合、関数は 2 つの引数で呼び出さなければならず、それ以上でもそれ以下でもありません。

```
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、welcome() 関数には 2 つのパラメータがあります。

この関数を二つの引数で呼び出しているので、関数はスムーズに動作し、エラーは発生しません。

異なる数の引数で呼び出された場合、インタープリタはエラーメッセージを表示します。

以下は、この関数への呼び出しで、一つとゼロの引数を含んでいる場合と、それぞれのエラーメッセージです。

```
welcome("Lily") #引数が一つだけ
```

```
>>> %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() #引数がない
```

```
>>> %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
```

デフォルト引数

MicroPython では、パラメータにデフォルト値を設定するために代入演算子（=）を使用できます。

引数なしで関数を呼び出した場合、デフォルト値が使用されます。

```
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 China!
```

この関数では、パラメータ name にはデフォルト値が設定されておらず、呼び出し時に必須（必須）です。

一方で、パラメータ msg のデフォルト値は "Welcome to China!" です。したがって、呼び出し時にはオプションです。値が提供された場合、デフォルト値は上書きされます。

関数内の任意の数の引数にデフォルト値を設定できます。ただし、一度デフォルト引数が設定されると、その右側のすべての引数にもデフォルト値が必要です。

これは、デフォルト引数に続く非デフォルト引数は許可されていないことを意味します。

例えば、上記の関数ヘッダを以下のように定義した場合：

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

次のエラーメッセージが表示されます：

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

キーワード引数

関数を特定の値で呼び出した場合、これらの値は位置に基づいて引数に割り当てられます。

例えば、上記の関数 welcome() で、それを welcome("Lily", "Welcome to China") として呼び出した場合、値 "Lily" は name に、同様に "Welcome to China" はパラメータ msg に割り当てられます。

MicroPython では、キーワード引数を使用して関数を呼び出すことができます。この方法で関数を呼び出すと、引数の順序（位置）を変更できます。

```
# 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!")
```

ここでわかるように、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ます。ただし、キーワード引数は位置引数の後に来る必要があります。

キーワード引数の後に位置引数があるとエラーが発生します。

例えば、関数の呼び出しが以下のような場合：

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

次のエラーが発生します：

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

任意の引数

事前に関数に渡される引数の数を知らない場合があります。

関数定義で、パラメータ名の前にアスタリスク（*）を追加できます。

```
def welcome(*names):
    """This function welcomes all the person
```

(次のページに続く)

(前のページからの続き)

```
    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
```

ここでは、関数を複数の引数で呼び出しています。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ます。

関数内部で、すべての引数を取得するために for ループを使用しています。

再帰

Python では、関数が他の関数を呼び出すことができることはよく知られています。さらに、関数が自分自身を呼び出すことも可能です。このような構造を再帰関数と呼びます。

この特性の利点は、データをループ処理して結果に到達できることです。

開発者は再帰に非常に慎重である必要があります。無限ループに陥ったり、過度なメモリやプロセッサのリソースを消費する関数を書いてしまう可能性があります。しかし、正確に書かれた再帰は、非常に効率的で数学的に優雅なプログラミング手法となることもあります。

```
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
```

(次のページに続く)

(前のページからの続き)

```
6
10
15
21
```

この例では、rec_func() は自分自身を呼び出す（「再帰」）関数として定義しています。 `i` 変数をデータとして使用し、再帰するたびにデクリメント（-1）します。条件が 0 より大きくない場合（つまり、0 の場合)、再帰は終了します。

新しい開発者にとっては、その動作を理解するのに時間がかかることがあります。最良のテスト方法は、テストして修正することです。

再帰の利点

- 再帰関数はコードをクリーンでエレガントに見せます。
- 再帰を使用すると、複雑なタスクをより単純なサブプロブレムに分解できます。
- ネストされた反復を使用するよりも、再帰を使用した方がシーケンス生成が容易です。

再帰の欠点

- 再帰の背後にあるロジックは、理解が難しい場合があります。
- 再帰呼び出しはメモリと時間を大量に消費するため、効率が悪いです。
- 再帰関数はデバッグが困難です。

4.6.9 データ型

組み込みデータ型

MicroPython には次のようなデータ型があります：

- 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
- ブーリアン型: bool
- バイナリ型: bytes, bytearray, memoryview

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ます。

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

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

データ型の設定

MicroPython では、変数に値を割り当てる際にデータ型が自動的に決定されるため、特に設定する必要はありません。

```
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'>
>>>
```

特定のデータ型の設定

データ型を明示的に指定したい場合は、以下のコンストラクタ関数を使用できます。

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

結果を確認するために、いくつかを出力してみましょう。

```
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
>>>
```

型変換

int(), float(), complex() メソッドを使用して、一つの型から別の型に変換できます。Python におけるキャスティングは、コンストラクタ関数を使用して行われます：

- int() - 整数リテラル、浮動小数点リテラル（小数部分をすべて削除）、文字列リテラル（文字列が整数を表している場合）から整数を生成
- float() - 整数リテラル、浮動小数点リテラル、または文字列リテラル（文字列が浮動小数点数または整数を表す場合）から浮動小数点数を生成
- str() - 文字列、整数リテラル、浮動小数点リテラルを含む多様なデータ型から文字列を生成

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

print(a)
print(b)
print(c)
```

注：複素数を別の数値型に変換することはできません。

4.6.10 演算子

演算子は、変数や値に対して操作を行うために使用されます。

- **算術演算子**
- **代入演算子**
- **比較演算子**
- **論理演算子**
- **同一性演算子**
- **メンバーシップ演算子**
- **ビット演算子**

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ます。

演算子	名前
+	加算
-	減算
*	乗算
/	除算
%	剰余
**	累乗
//	切り捨て除算

```
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
1.666667
2
125
```

(次のページに続く)

(前のページからの続き)

```
1
8
2
15
>>>
```

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されます。

演算子	例	同じ意味
=	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
>>>
```

比較演算子

比較演算子は、二つの値を比較するために使用されます。

演算子	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 を返します。なぜなら、 **a** は **b** より小さいからです。

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ます。

演算子	説明
and	両方の文が真なら True を返す
or	どちらかの文が真なら True を返す
not	結果を反転させ、結果が真なら False を返す

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

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

同一性演算子

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

演算子	説明
is	両方の変数が同じオブジェクトなら True を返す
is not	両方の変数が同じオブジェクトでないなら True を返す

```
a = ["hello", "welcome"]
b = ["hello", "welcome"]
c = a

print(a is c)
print(a is b)
print(a == b)
```

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

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に存在するかどうかをテストするために使用されます。

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

```
a = ["hello", "welcome", "Goodmorning"]

print("welcome" in a)
```

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

ビット演算子

ビット演算子は、（バイナリ）数字を比較するために使用されます。

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 に設定する
|	OR	二つのビットのうちの一つが 1 の場合、各ビットを 1 に設定する
^	XOR	二つのビットのうちの一つだけが 1 の場合、各ビットを 1 に設定する
~	NOT	すべてのビットを反転させる
<<	ゼロ埋め左シフト	右からゼロをプッシュして左端のビットを落とすことで左にシフトする
>>	符号付き右シフト	左端のビットのコピーを左からプッシュし、右端のビットを落とすことで右にシフトする

```
num = 2

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

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

4.6.11 リスト

リストは複数のアイテムを一つ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ます。

```
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、重複する値を許容します。リストのアイテムはインデックス付きで、最初のアイテムがインデックス [0]、次のアイテムがインデックス [1]、と続きます。

```
C_list = ["Red", "Blue", "Green", "Blue"]
print(C_list)             # 重複を許容
```

(次のページに続く)

(前のページからの続き)

```
print(C_list[0])
print(C_list[1])          # 順序付き
C_list[2] = "Purple"      # 変更可能
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ます：

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

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

リストの長さ

リストにいくつのアイテムが含まれているかを判定するには、len() 関数を使用します。

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

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

リストアイテムの確認

リストの二番目のアイテムを出力します：

```
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します：

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

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

二番目と三番目のアイテムを出力します：

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

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

リストアイテムの変更

二番目と三番目のアイテムを変更します：

```
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]
```

二番目の値を 2 つの値で置き換えます：

```
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]
```

リストアイテムの追加

append() メソッドを使用してアイテムを追加します：

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

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

二番目の位置にアイテムを挿入します：

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

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

リストアイテムの削除

remove() メソッドは指定したアイテムを削除します。

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

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

pop() メソッドは指定したインデックスのアイテムを削除します。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します。

```
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 キーワードも指定したインデックスのアイテムを削除します：

```
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します。リスト自体は残りますが、内容はありません。

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

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

2. 出力と入力

4.7 2.1 こんにちは、LED！

"Hello, world!"を出力することがプログラミング学習の第一歩であるように、LED をプログラムで制御することは、物理的なプログラミングを学ぶ際の伝統的な導入となっています。

- *LED*

必要な部品

このプロジェクトでは、以下の部品が必要です。

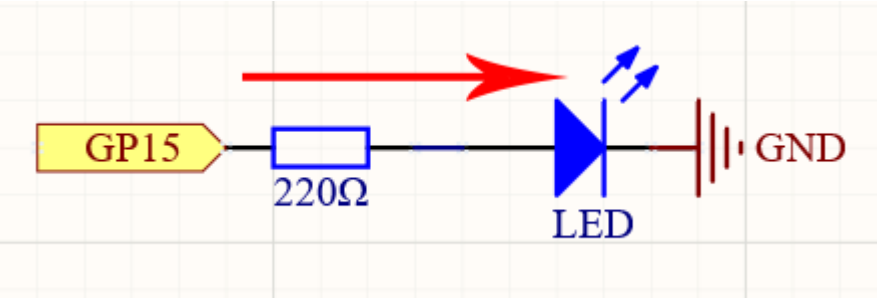
キット全体を購入することは非常に便利です。リンクは以下になります：

名前	このキットのアイテム	リンク
Kepler Kit	450 以上	

以下のリンクから個別に購入することもできます。

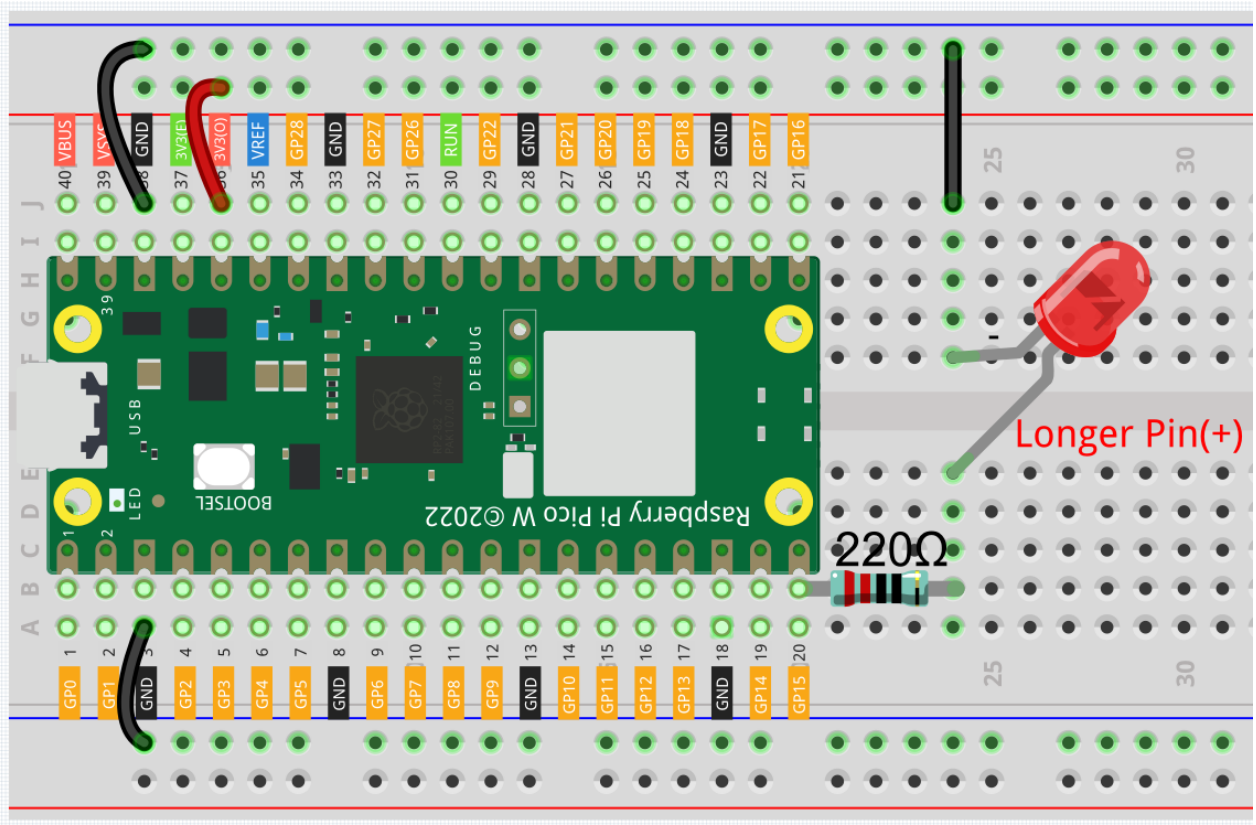
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	

回路図



この回路は単純な原理で動作し、図に示されているように電流の方向が示されています。GP15 が高レベル（3.3v）を出力すると、220ohm の電流制限抵抗を経て LED が点灯します。GP15 が低レベル（0v）を出力すると、LED は消灯します。

配線



回路を組む際は、電流の方向に従いましょう！

1. Pico W ボードの GP15 ピンで LED が供給され、ここから回路が始まります。
2. LED を保護するために、電流は 220 オームの抵抗器を通過しなければなりません。抵抗器の一方の端子は Pico W GP15 ピンと同じ行（私の回路では行 20）に挿入し、他方の端子はブレッドボードの空いている行（行 24）に挿入してください。

   注釈: 220 オームの抵抗器のカラーリングは赤、赤、黒、黒、茶です。

3. LED を取り上げると、一方のリードが他方よりも長いことがわかります。長いリードを抵抗器と同じ行に、短いリードをブレッドボードの中央の隙間を挟んで同じ行に接続してください。

   注釈: 長いリードは陽極であり、回路の正側を表します。短いリードは陰極であり、回路の負側を表します。

   陽極は GPIO ピンに抵抗器を介して接続する必要があり、陰極は GND ピンに接続する必要があります。

4. オス-オス（M2M）ジャンパーワイヤーを使用して、LED の短いピンをブレッドボードの負電源バスに接続します。

5. ジャンパーを使用して、Pico W の GND ピンを負の電源バスに接続します。

コード

注釈:

- kepler-kit-main/micropython のパスの下で 2.1_hello_led.py ファイルを開くか、このコードを Thonny にコピーしてから、"Run Current Script"をクリックするか、単に F5 を押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタプリターをクリックすることを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)
while True:
    led.value(1)
    utime.sleep(2)
    led.value(0)
    utime.sleep(2)
```

コードが実行された後、LED が点滅するのが見えるでしょう。

動作の仕組みは？

GPIO を使用するには、 machine ライブラリが必要です。

```
import machine
```

このライブラリには、MicroPython と Pico W との間で通信するために必要なすべての命令が含まれています。このコード行がない場合、GPIO を制御することはできません。

次に注目するべき行は以下のとおりです。

```
led = machine.Pin(15, machine.Pin.OUT)
```

ここでオブジェクト led が定義されています。技術的には、x、y、banana、Michael_Jackson、または任意の文字など、任意の名前にすることができます。プログラムを読みやすくするためには、目的を説明する名前を使用するのが最善です。

この行の第二部分（等号の後ろの部分）では、machine ライブラリ内の Pin 関数を呼び出しています。これは Pico の GPIO ピンに何をすべきかを指示するために使用されます。Pin 関数には 2 つのパラメーターがあります：最初

の 1 つ（15）は設定するピンを表し、第二のパラメーター（machine.Pin.OUT）は、ピンが入力ではなく出力であるべきことを指定します。

上記のコードではピンが「設定」されていますが、LED を点灯させるわけではありません。これを行うためには、ピンを「使用」する必要もあります。

```
led.value(1)
```

GP15 ピンは以前に設定され、 `led` と名付けられました。この文の機能は、 `led` の値を 1 に設定して LED を点灯させることです。

全体として、GPIO を使用するには、以下のステップが必要です：

- **machine ライブラリをインポートする**：これは必須であり、一度だけ実行されます。
- **GPIO を設定する**：使用する前に、各ピンを設定する必要があります。
- **使用する**：ピンに値を割り当てることで、ピンの動作状態を変更します。

上記のステップに従って例を書くと、次のようなコードになります：

```
import machine
led = machine.Pin(15, machine.Pin.OUT)
led.value(1)
```

これを実行すると、LED を点灯させることができます。

次に、"消灯"文を追加してみましょう：

```
import machine
led = machine.Pin(15, machine.Pin.OUT)
led.value(1)
led.value(0)
```

このコードに基づいて、このプログラムは最初に LED を点灯させ、次に消灯させます。しかし、実際に使用すると、このようにはなりません。LED から光が出ていないのは、2 行の間の実行速度が非常に速いためであり、人間の目が反応するよりもはるかに速いからです。LED が点灯すると、私たちは即座に光を感じません。これはプログラムを遅くすることで修正できます。

プログラムの第二行には、以下の文が含まれるべきです：

```
import utime
```

`machine` と同様に、ここでは `utime` ライブラリがインポートされており、時間に関連するすべてのことを処理します。必要な遅延はこれに含まれています。 `led.value(1)` と `led.value(0)` の間に遅延文を追加し、それらを 2 秒間隔で分けます。

```
utime.sleep(2)
```

これでコードは次のようになります。実行すると、LED が最初に点灯し、次に消灯するのがわかります：

```
import machine
import utime
led = machine.Pin(15, machine.Pin.OUT)
led.value(1)
utime.sleep(2)
led.value(0)
```

最後に、LED を点滅させるようにしましょう。ループを作成し、プログラムを書き直すと、この章の始めに見たものになります。

```
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)
while True:
    led.value(1)
    utime.sleep(2)
    led.value(0)
    utime.sleep(2)
```

- *While* **ループ**

さらに詳しく

通常、ライブラリには API（Application Programming Interface）ファイルが関連付けられています。このファイルには、このライブラリを使用するために必要なすべての情報が含まれています。これには、関数、クラス、戻り値のタイプ、パラメータのタイプなどの詳細な説明もあります。

この記事では、MicroPython の `machine` と `utime` ライブラリを使用しましたが、それらを使用するさまざまな方法は以下で見つけることができます。

- machine.Pin
- utime

LED を点滅させるこの例を理解するためには、API ファイルを読むことをお勧めします！

注釈:

- kepler-kit-main/micropython のパスの下にある 2.1_hello_led_2.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 を押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)
while True:
    led.toggle()
    utime.sleep(1)
```

4.8 2.2 レベルを表示

最初のプロジェクトは LED を点滅させるだけのシンプルなものです。このプロジェクトでは、一般的に電力やボリュームレベルを表示するために使用される、プラスチックケースに 10 個の LED を含む LED バーグラフを使用します。

- *LED* **バーグラフ**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

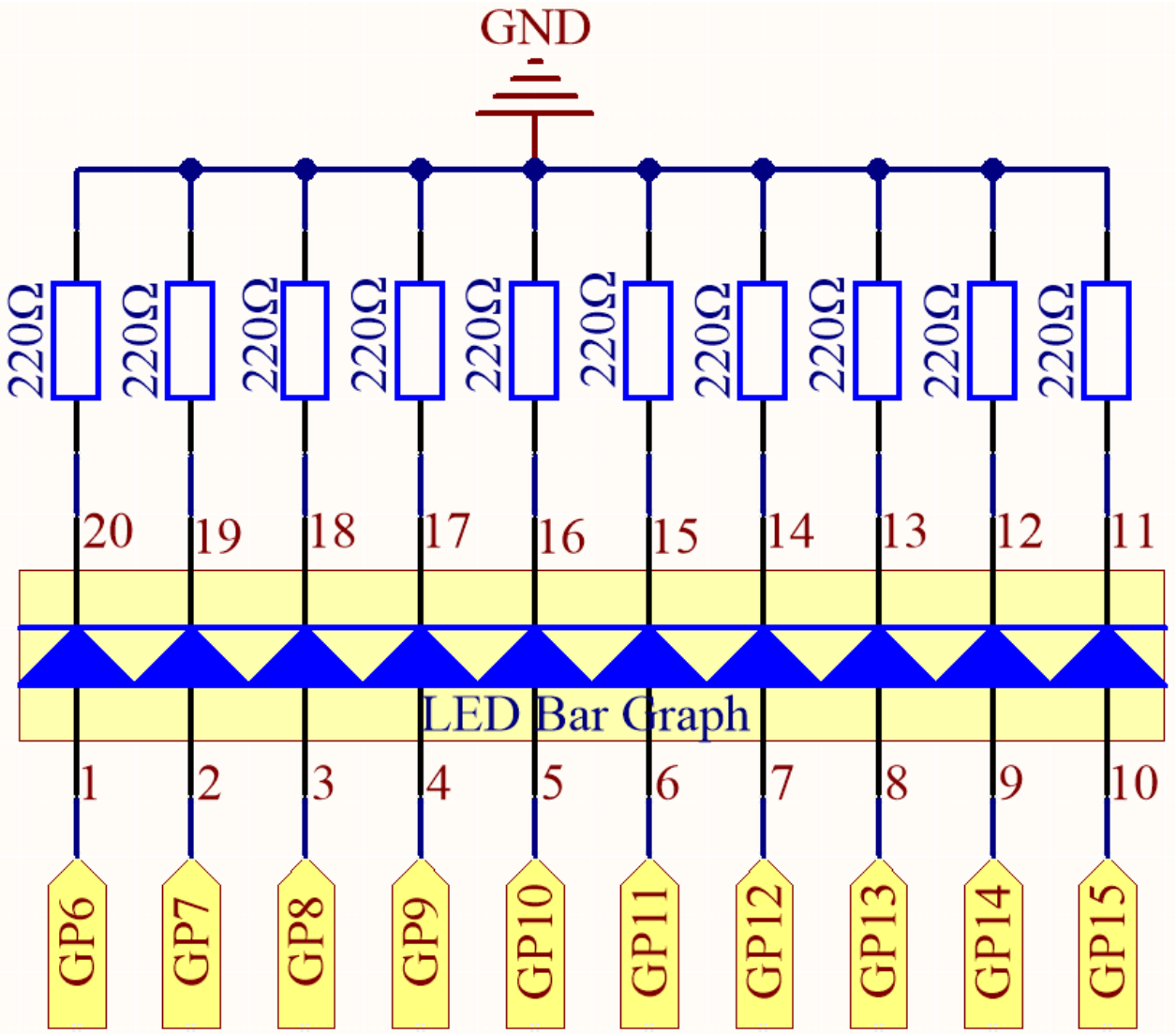
全体のキットを購入する方が便利です、リンクは以下の通りです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入できます。

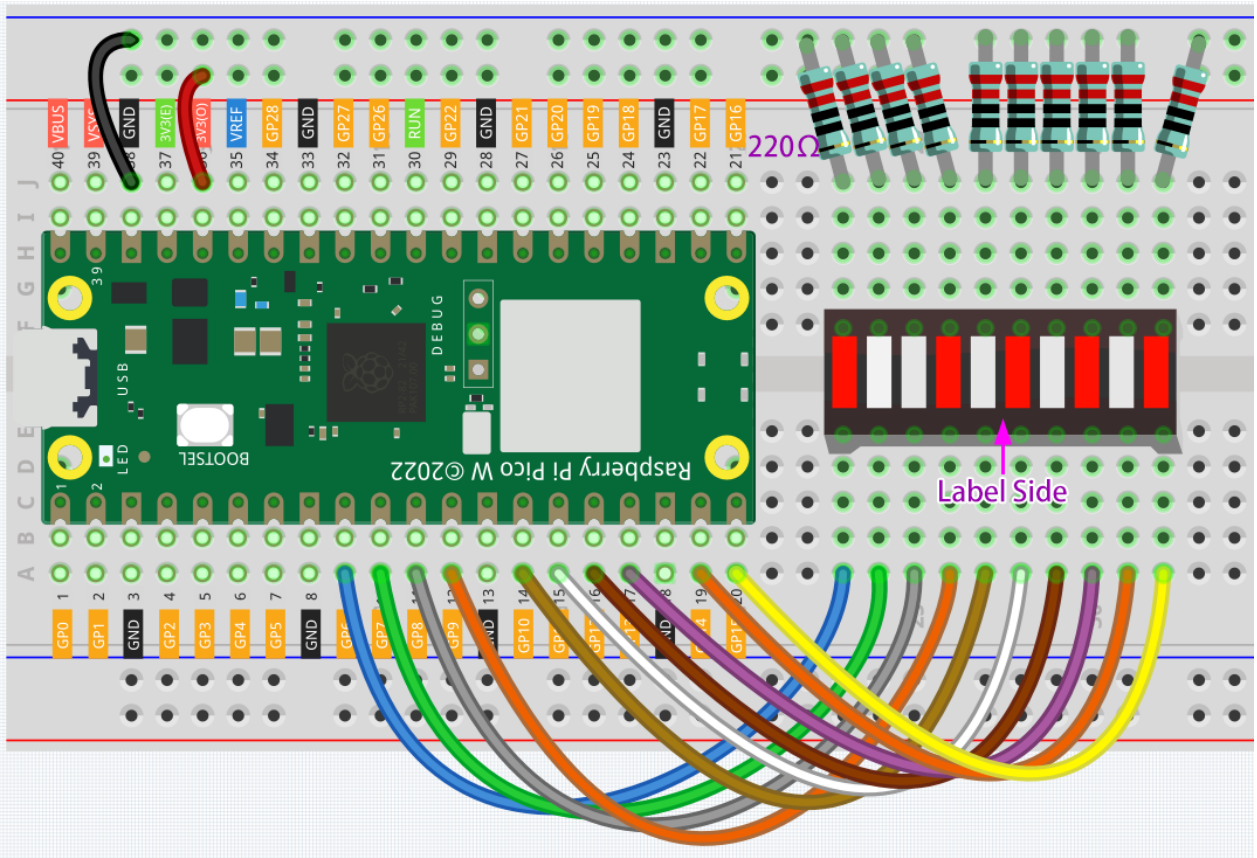
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	複数	
5	**抵抗器**	10(220 Ω)	
6	*LED* **バーグラフ**	1	

回路図



LED バーグラフには 10 個の LED があり、それぞれが個別に制御できます。各 LED のアノードは GP6～GP15 に接続され、カソードは 220 オームの抵抗を介して GND に接続されています。

配線



コード

注釈:

- kepler-kit-main/micropython のパス内の 2.2_display_the_level.py ファイルを開くか、このコードを Thonny にコピーしてから「Run Current Script」をクリック、または単に F5 キーを押して実行してください。
- 右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

pin = [6,7,8,9,10,11,12,13,14,15]
led= []
for i in range(10):
    led.append(None)
```

(次のページに続く)

(前のページからの続き)

```
    led[i] = machine.Pin(pin[i], machine.Pin.OUT)

while True:
    for i in range(10):
        led[i].toggle()
        utime.sleep(0.2)
```

プログラムが実行されていると、LED バーグラフ上の LED が順番に点灯し、その後消えます。

動作の仕組みは？

LED バーは、10 本のピンによって制御される 10 個の LED で構成されています。つまり、これらのピンを定義する必要があります。一つひとつ定義するのは煩雑な作業なので、ここでは **Lists（リスト）** を使用しています。

注釈: Python のリストは、一度に複数の要素を扱うことができる非常に多機能なデータ型であり、カンマで区切られた要素を角括弧 [] 内に配置することで作成されます。

```
pin = [6,7,8,9,10,11,12,13,14,15]
```

このコード行によって pin というリストが定義され、10 個の要素 6,7,8,9,10,11,12,13,14,15 が含まれます。インデックス演算子 [] を使用して、リスト内の項目にアクセスすることができます。Python では、インデックスは 0 から始まります。したがって、10 個の要素を持つリストは、0 から 9 までのインデックスを持ちます。このリストを例にすると、 pin[0] は 6 であり、 pin[4] は 10 です。

次に、10 個の LED オブジェクトを定義するために使用される空のリスト led を宣言します。

```
led = []
```

リストの長さが 0 であるため、配列に対する直接的な操作、たとえば led[0] を出力するなど、は機能しません。新しい項目を追加する必要があります。

```
led.append(None)
```

この append() メソッドの結果として、リスト led には最初の項目が追加され、長さが 1 になり、led[0] が None（null を意味する）という現在の値にもかかわらず有効な要素になります。

次のステップは、ピン 6 に接続されている led[0] を、最初の LED オブジェクトとして定義することです。

```
led[0] = machine.Pin(6, machine.Pin.OUT)
```

最初の LED オブジェクトが定義されました。

以上から、10 個のピン番号をリスト **pin** として作成しました。これにより、まとめて操作を行いやすくなります。

```
led[0] = machine.Pin(pin[0], machine.Pin.OUT)
```

for 文を使用して、10 本のピンすべてが上記の文を実行するようにします。

```
import machine

pin = [6,7,8,9,10,11,12,13,14,15]
led= []
for i in range(10):
    led.append(None)
    led[i] = machine.Pin(pin[i], machine.Pin.OUT)
```

- **リスト**
- *For* **ループ**

もう一つの for ループを使用して、LED バーの 10 個の LED が順番に状態を切り替えるようにします。

```
for i in range(10):
    led[i].toggle()
    utime.sleep(0.2)
```

このコード片を while ループ内に配置することで、コードの完成です。

```
import machine
import utime

pin = [6,7,8,9,10,11,12,13,14,15]
led= []
for i in range(10):
    led.append(None)
    led[i] = machine.Pin(pin[i], machine.Pin.OUT)

while True:
    for i in range(10):
        led[i].toggle()
        utime.sleep(0.2)
```

4.9 2.3 LED のフェード

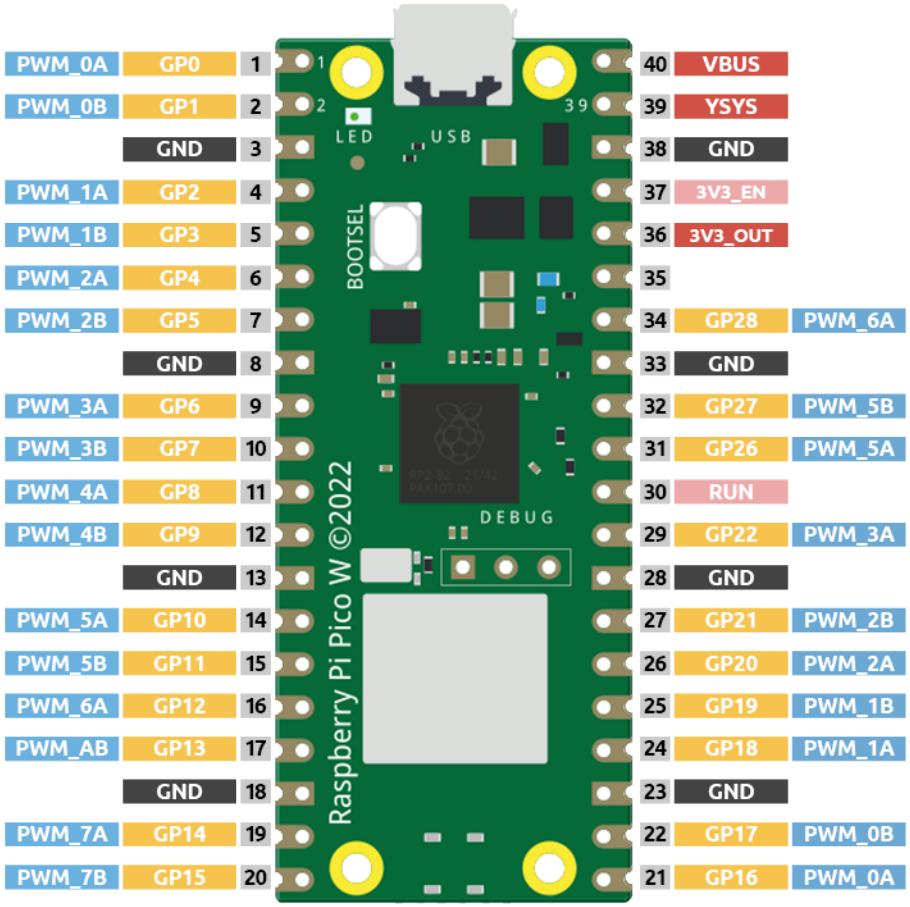
今まで、デジタル出力として高レベルと低レベル（ON と OFF）の二つの出力信号しか使用していませんでした。しかし、現実の利用シーンでは、多くのデバイスが単純に ON/OFF だけで動作するわけではありません。例えば、モーターの速度を調整したり、デスクランプの明るさを調整したりといった場面があります。以前は、これを達成するために抵抗を調整するスライダーが用いられましたが、それは信頼性に欠け、非効率的でした。そのため、パルス幅変調（PWM）がこのような複雑な問題に対する実用的な解決策として登場しました。

パルスは高レベルと低レベルを含むデジタル出力です。これらのピンのパルス幅は、ON/OFF の速度を変更することで調整できます。

20ms（ほとんどの人が視覚的に保持する期間）という短い時間内で、LED を点灯させ、消灯させ、再び点灯させると、消灯していたことに気づかず、光の明るさがわずかに弱くなるだけです。この期間中に LED が点いている時間が多いほど、明るくなります。言い換えれば、サイクル内でパルスが広いほど、マイクロコントローラーから出力される「電気信号の強度」が大きくなります。これが PWM が LED の明るさ（またはモーターの速度）を制御する仕組みです。

- パルス幅変調 - Wikipedia

Pico W が PWM を使用する際に注意すべき点がいくつかあります。この図を見てみましょう。



Pico W は各 GPIO ピンで PWM をサポートしていますが、実際には独立した PWM 出力が 16 個（30 個ではない）あり、これらは左側の GP0 から GP15 までに分散されています。右側の GPIO の PWM 出力は左側と同一です。

プログラミング中に同じ PWM チャンネルを異なる目的で設定しないように注意が必要です。例えば、GP0 と GP16 は両方とも PWM_0A です。

この知識を理解した上で、LED のフェード効果を実現してみましょう。

- *LED*

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

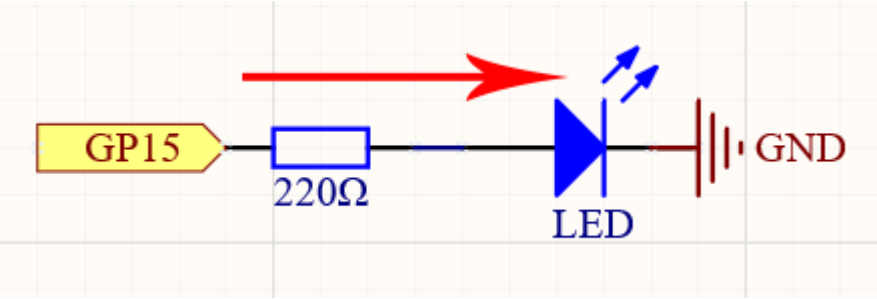
全体のキットを購入する方が確実に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

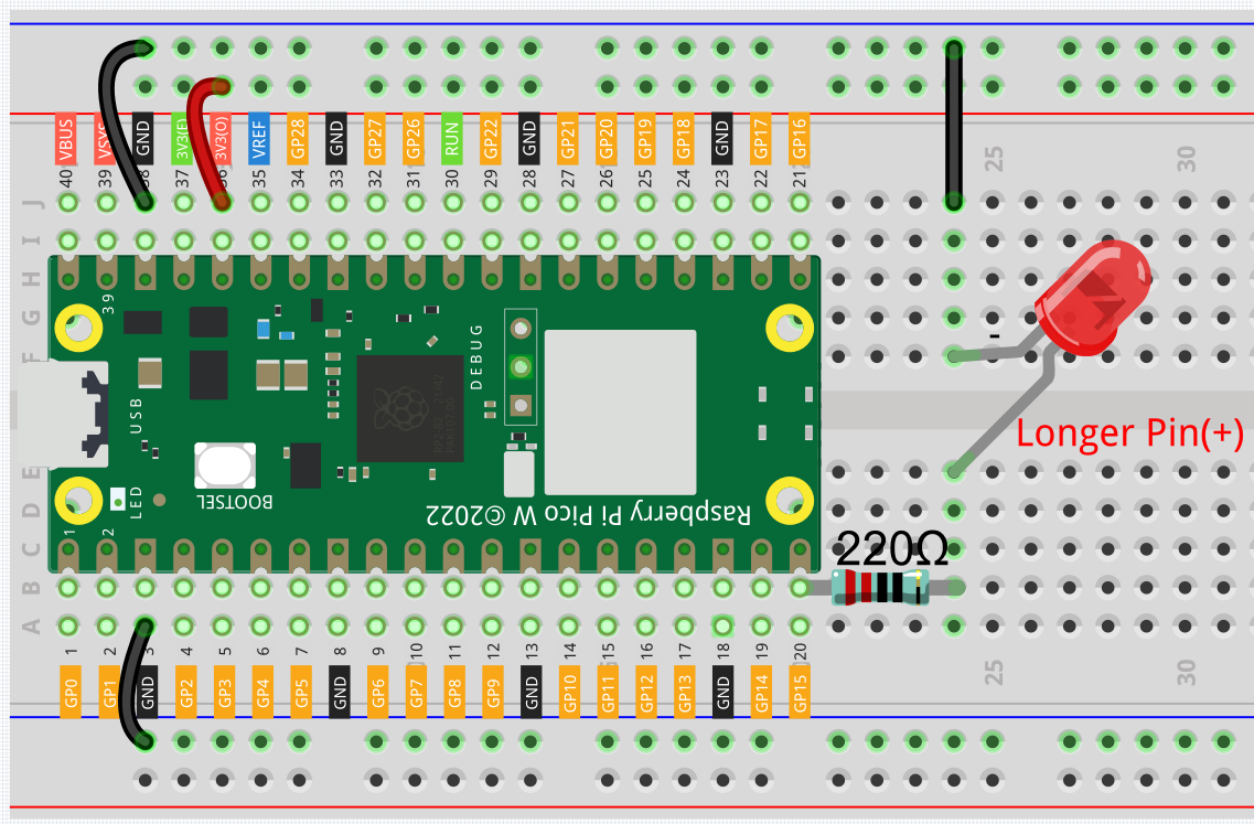
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	いくつか	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	

回路図



このプロジェクトは、最初のプロジェクト *2.1* **こんにちは、*LED*！** と同じ回路ですが、信号のタイプが異なります。最初のプロジェクトでは GP15 から直接デジタルの高レベルと低レベル（0&1）を出力して LED を点灯または消灯させていましたが、このプロジェクトでは GP15 から PWM 信号を出力して LED の明るさを制御します。

配線



コード

注釈:

- `kepler-kit-main/micropython` のディレクトリにある `2.3_fading_led.py` ファイルを開いたり、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 を押して実行します。
- 右下隅にある「MicroPython（Raspberry Pi Pico）」のインタプリタを選択するのを忘れないでください。

- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

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

for brightness in range(0,65535,50):
    led.duty_u16(brightness)
    utime.sleep_ms(10)
led.duty_u16(0)
```

このコードが実行されると、LED の明るさが徐々に増していきます。

どのように動作するのか？

ここでは、GP15 の PWM 出力のデューティサイクルを変更することで、LED の明るさを変更しています。以下の行に注目してください。

```
import machine
import utime

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

for brightness in range(0,65535,50):
    led.duty_u16(brightness)
    utime.sleep_ms(10)
led.duty_u16(0)
```

- `led = machine.PWM(machine.Pin(15))` は、GP15 ピンを PWM 出力として設定します。
- `led.freq(1000)` は PWM の周波数を設定するために使用され、ここでは 1000Hz に設定されています。つまり、1ms（1/1000）が 1 サイクルです。
- `led.duty_u16()` はデューティサイクルを設定するために使用され、これは 16 ビットの整数（2^16=65536）です。0 は 0% のデューティサイクルを示し、各サイクルで高レベルを出力する時間が 0%、すなわち、全てのパルスがオフになります。値 65535 は、デューティサイクルが 100% であることを示し、パルス全体がオンになり、結果は「1」になります。値が 32768 の場合、パルスを半分オンにするので、LED は全開時の半分の明るさになります。

4.10 2.4 カラフルな光

我々が知っているように、光は重ね合わせることができます。例えば、青い光と緑の光を混ぜるとシアン色の光が生まれ、赤い光と緑の光を混ぜると黄色の光が生まれます。これを「加法混色」と呼びます。

- 加法混色 - ウィキペディア

この方法に基づいて、三原色を用いて、異なる比重に応じて任意の色の可視光を混合することができます。例えば、オレンジ色は赤色を多く、緑色を少なくして作ることができます。

この章では、RGB LED を用いて加法混色の神秘を探求します！

RGB LED は、赤い LED、緑の LED、青い LED を一つのランプキャップの下に封入し、三つの LED は一つのカソードピンを共有しています。各アノードピンに電気信号が供給されるため、対応する色の光が表示されます。各アノードの電気信号強度を変更することで、さまざまな色を生成することができます。

- *RGB LED*

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

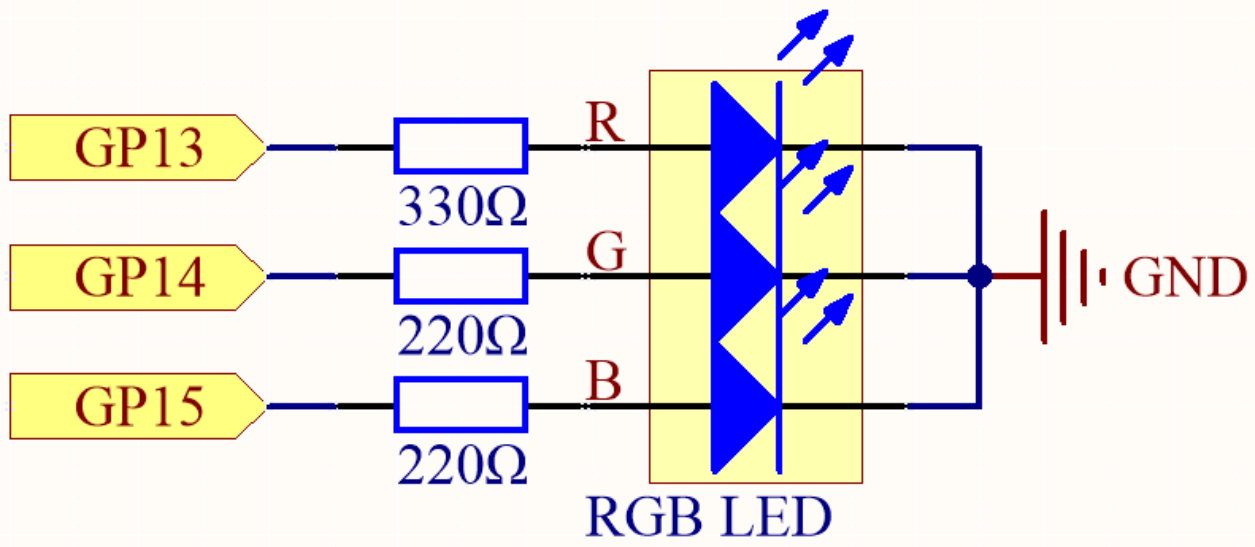
一式をまとめて購入するのは非常に便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することも可能です。

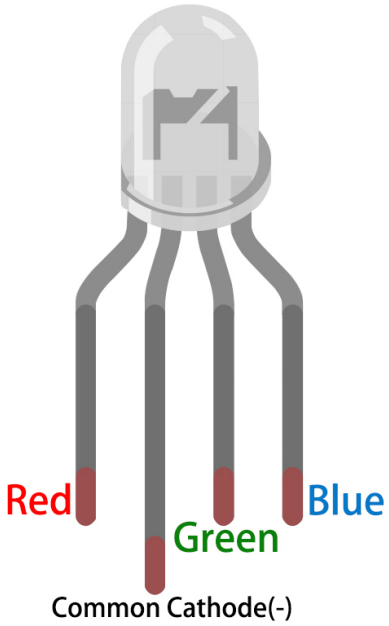
SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	3(1-330 Ω, 2-220 Ω)	
6	*RGB LED*	1	

回路図

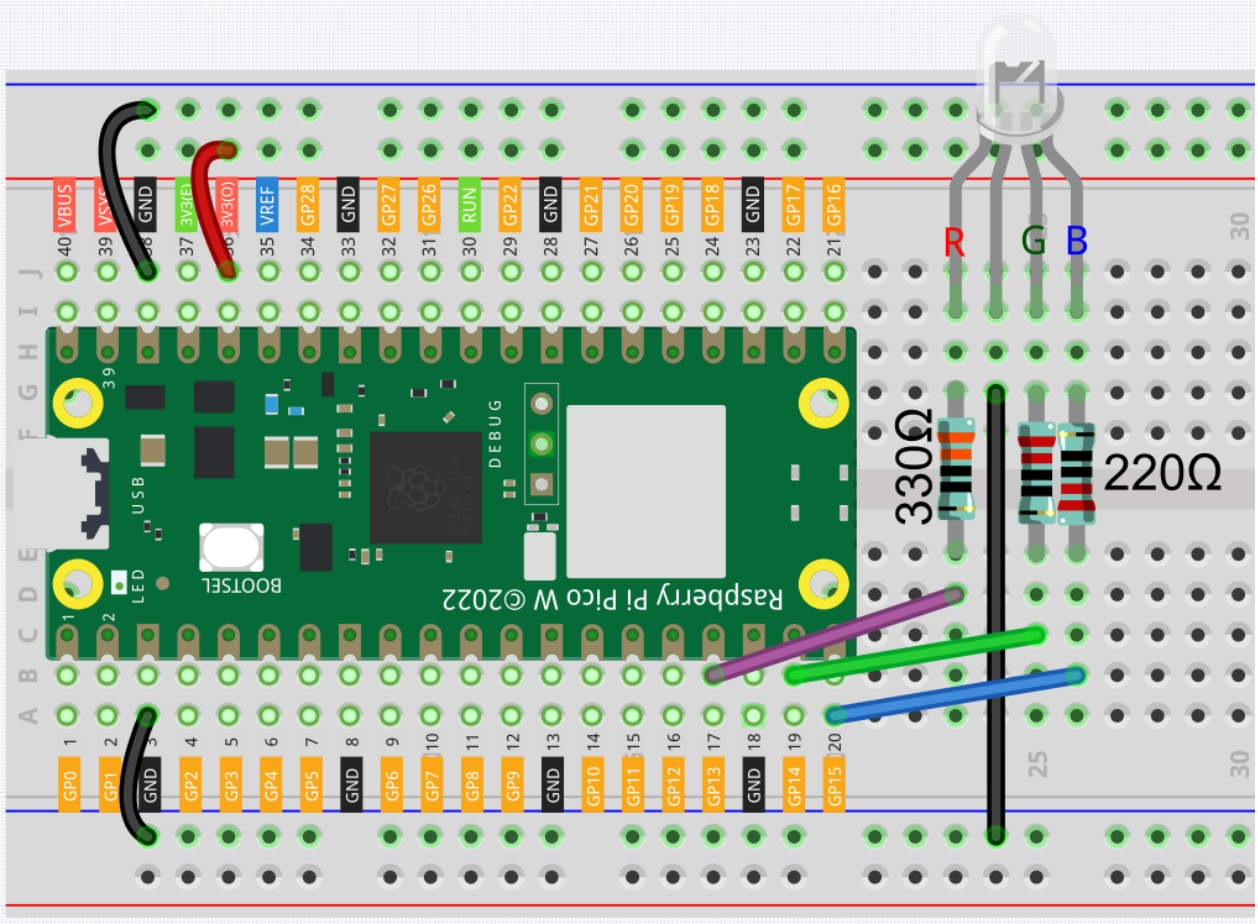


PWM ピンの GP13、GP14、および GP15 は、RGB LED の赤、緑、青のピンをそれぞれ制御します。共通のカソードピンは GND に接続されます。これにより、RGB LED は異なる PWM 値でこれらのピンに光を加算することで、特定の色を表示できます。

配線



RGB LED には 4 つのピンがあります：一番長いピンは共通のカソードピンで、通常は GND に接続されます。この長いピンの隣にある左側のピンが赤で、右側にある 2 つのピンは緑と青です。



コード

注釈:

- kepler-kit-main/micropython ディレクトリ内の 2.4_colorful_light.py ファイルを開くか、このコードを Thonny にコピペして、「Run Current Script」をクリック、または F5 キーを押して実行します。
- 右下角にある「MicroPython（Raspberry Pi Pico）」インタープリターをクリックして選択してください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

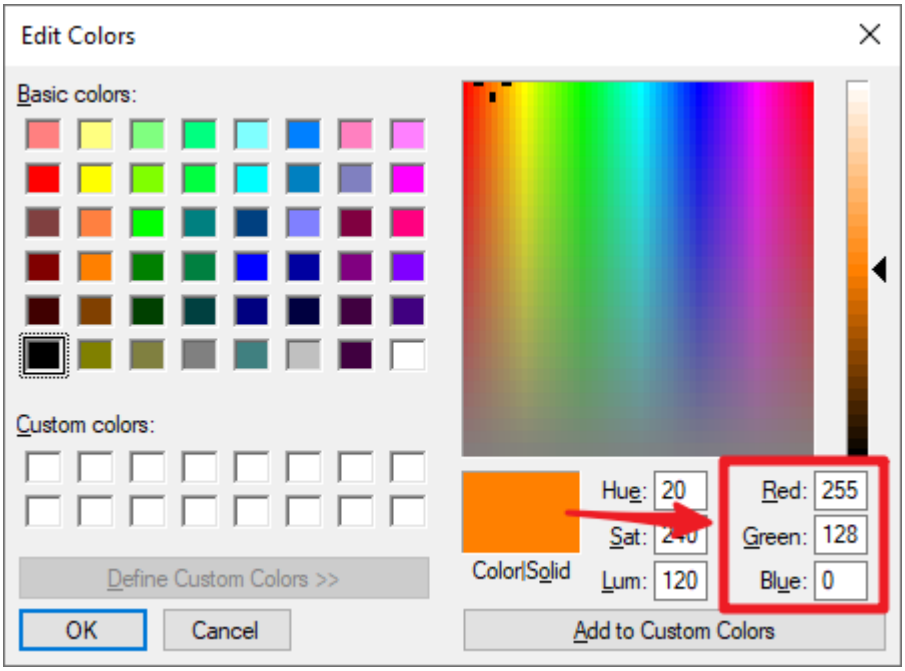
```
import machine
import utime

red = machine.PWM(machine.Pin(13))
green = machine.PWM(machine.Pin(14))
blue = machine.PWM(machine.Pin(15))
red.freq(1000)
```

(次のページに続く)

(前のページからの続き)

```
green.freq(1000)
blue.freq(1000)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def color_to_duty(rgb_value):
    rgb_value = int(interval_mapping(rgb_value, 0, 255, 0, 65535))
    return rgb_value

def color_set(red_value, green_value, blue_value):
    red.duty_u16(color_to_duty(red_value))
    green.duty_u16(color_to_duty(green_value))
    blue.duty_u16(color_to_duty(blue_value))

color_set(255, 128, 0)
```

こちらでは、描画ソフト（例：ペイント）で好みの色を選び、RGB LED でその色を表示できます。



`color_set()` 関数に RGB 値を入力すると、選択した色で RGB LED が点灯します。

仕組みについて

三原色を統合して機能するように、 `color_set()` 関数を定義しています。

現在、コンピュータのハードウェアピクセルは通常 24 ビットで表現されます。各基本色は 8 ビットに分けられ、色値は 0 から 255 までです。0 を含めて各基本色に 256 の可能な組み合わせがあります。よって、256 x 256 x 256 = 16,777,216 色が可能です。`color_set()` 関数も 24 ビット表記を使用しているため、色の選択が容易です。

そして、 `duty_u16()` の値域が 0〜65535 であるため、PWM を通じて RGB LED に信号を出力する際には、`color_to_duty()` と `interval_mapping()` 関数を用いて色値を duty 値にマッピングしています。

4.11 2.5 ボタンの値を読み取る

これらのピンは、その名前が示すように GPIO（汎用入出力）として、入力と出力の両方の機能を持っています。以前は出力機能を使用しましたが、この章では入力機能を使用してボタンの値を入力します。

- **ボタン**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

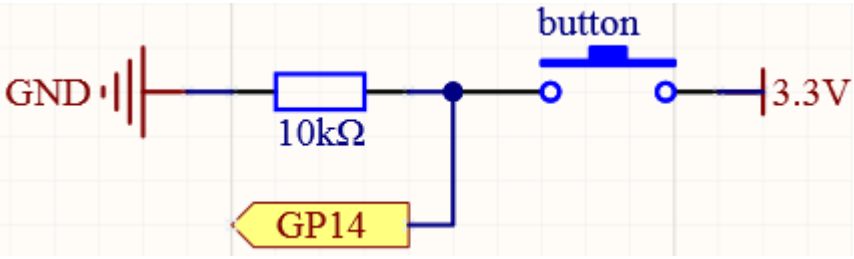
まとめてキットを購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入できます。

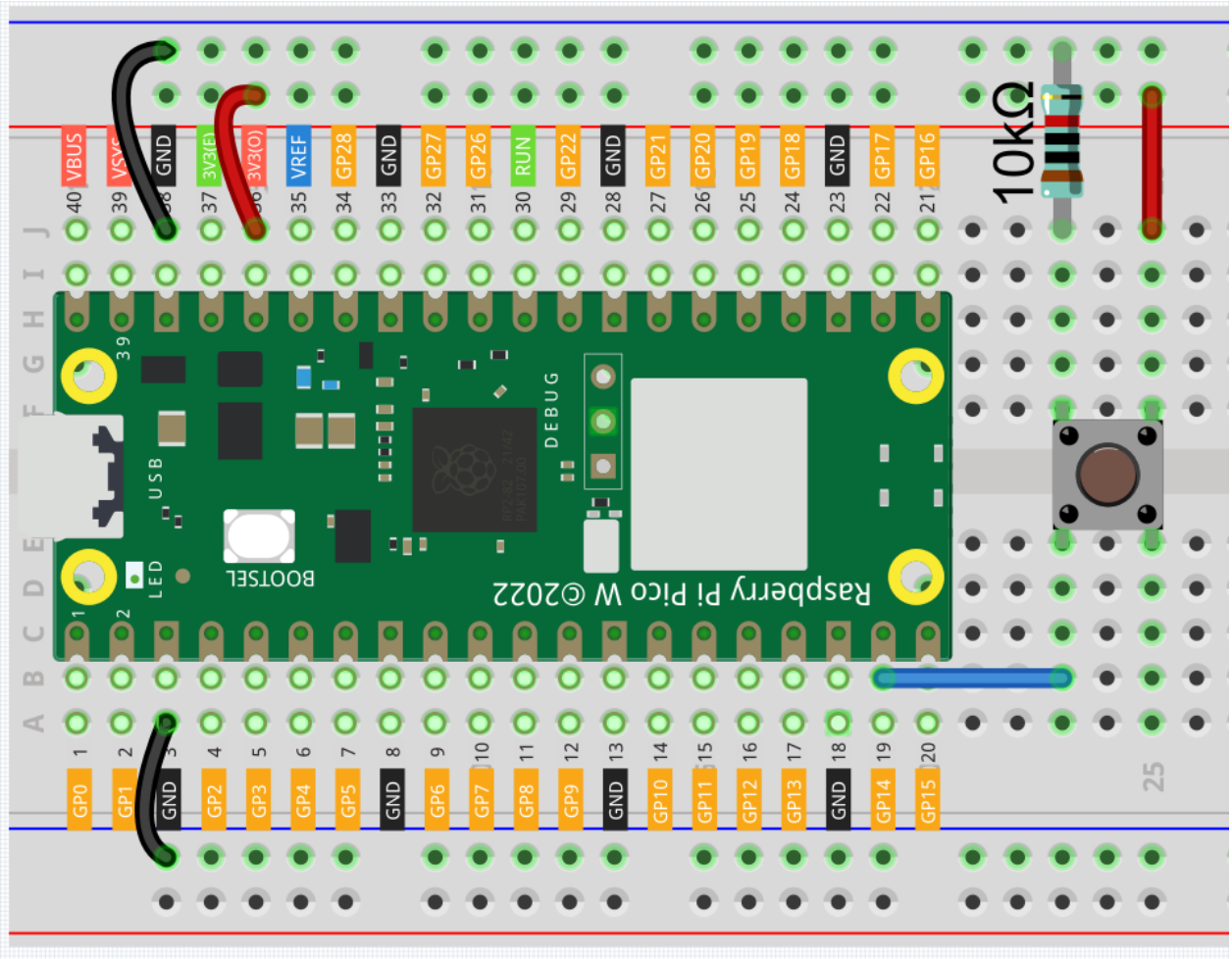
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	いくつか	
5	**抵抗器**	1(10KΩ)	
6	**ボタン**	1	

回路図



ボタンの片側ピンが 3.3v に、もう一方のピンが GP14 に接続されていれば、ボタンが押された状態で GP14 がハイレベルになります。しかし、ボタンが押されていない場合、GP14 は未定義状態となり、ハイかローかが不明です。ボタンが押されていない時に安定したローレベルを得るためには、10K のプルダウン抵抗を介して GP14 を GND に再接続する必要があります。

配線



注釈: 4 ピンのボタンは H 型になっています。左の 2 ピンか右の 2 ピンが接続されており、中央のギャップを越えると、同じ行番号の 2 つの半行が接続されます。(例えば、私の回路では、E23 と F23 が接続されていますし、E25 と F25 も接続されています)。

ボタンが押されるまで、左右のピンは互いに独立しており、一方から他方への電流の流れはありません。

コード

注釈:

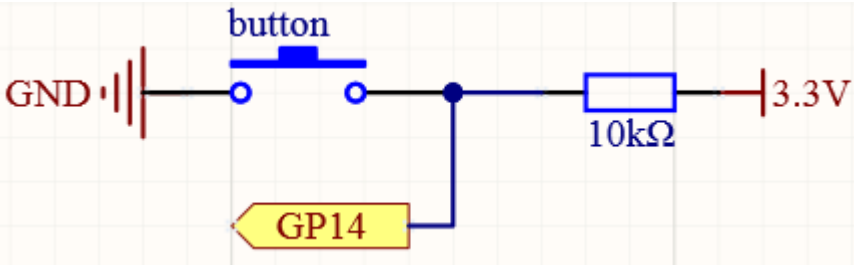
- kepler-kit-main/micropython パス下の 2.5_read_button_value.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、F5 キーを押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタを選択することを忘れずに。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

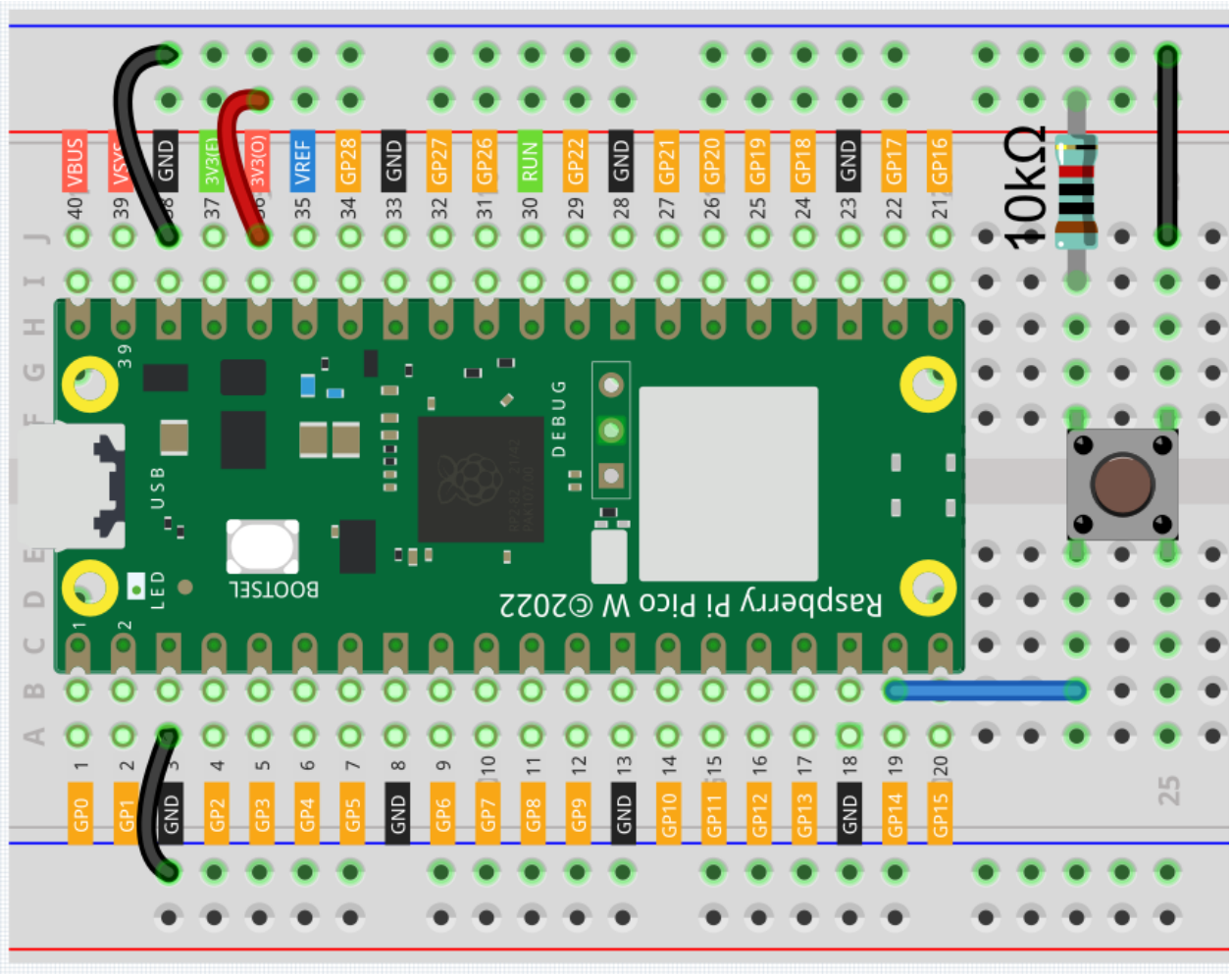
```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 1:
        print("You pressed the button!")
        utime.sleep(1)
```

コードが実行されると、シェルに「You pressed the button!」と表示されます。

プルアップ動作モード

次は、プルアップモードでボタンを使用する場合の配線とコードです。





プルダウンモードとの唯一の違いは、10K の抵抗が 3.3V に接続され、ボタンは GND に接続されているため、ボタンが押されると GP14 がローレベルになることです。これはプルダウンモードで得られる値とは逆です。したがって、このコードを `if button.value() == 0:` に変更するだけです。

参考資料もご覧ください：

- machine.Pin

4.12 2.6 傾けてみよう！

この傾きスイッチは、中央に金属の玉がある 2 ピンのデバイスです。スイッチが垂直の場合、2 つのピンが接続されています。それが傾いた場合、2 つのピンが切断されます。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

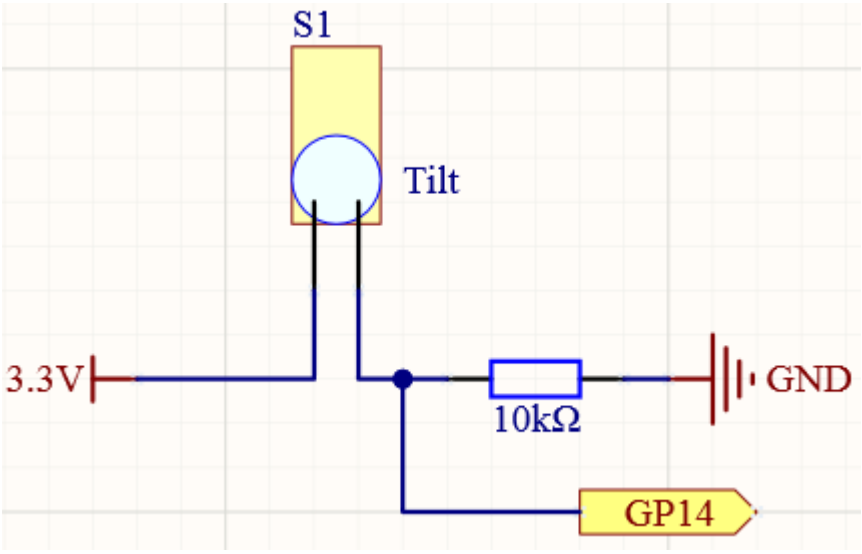
便宜上、全体のキットを購入することもできます。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**傾斜スイッチ**	1	

回路図

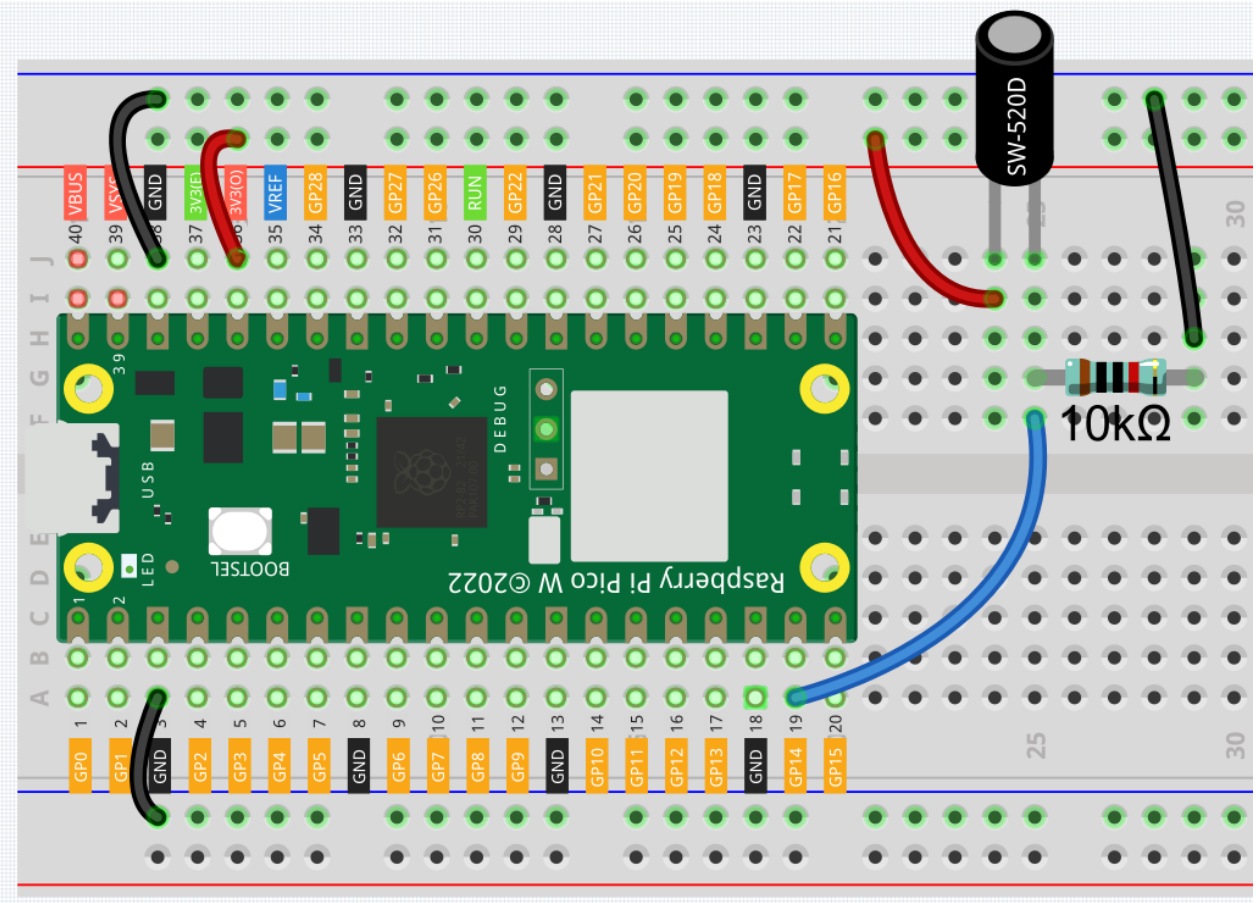


垂直に置いた場合、GP14 はハイレベルになります。傾けた後、GP14 はローレベルになります。

10K Ωの抵抗器の目的は、傾きスイッチが傾いた状態のときに GP14 を安定したローレベル状態に保つことです。

- **傾斜スイッチ**

配線



コード

注釈:

- kepler-kit-main/micropython のパスにある 2.6_tilt_switch.py ファイルを開くか、このコードを Thonny にコピーしてから、"Run Current Script"をクリックするか F5 を押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 0:
        print("The switch works!")
        utime.sleep(1)
```

プログラムが実行された後、ブレッドボード（傾きスイッチ）を傾けると、シェルに「The switch works!」と表示されます。

4.13 2.7 左右切り替え

スライドスイッチは 3 ピンのデバイスで、ピン 2（中央）が共通ピンです。スイッチを左に切り替えると、左側の 2 つのピンが接続され、右に切り替えると、右側の 2 つのピンが接続されます。

必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

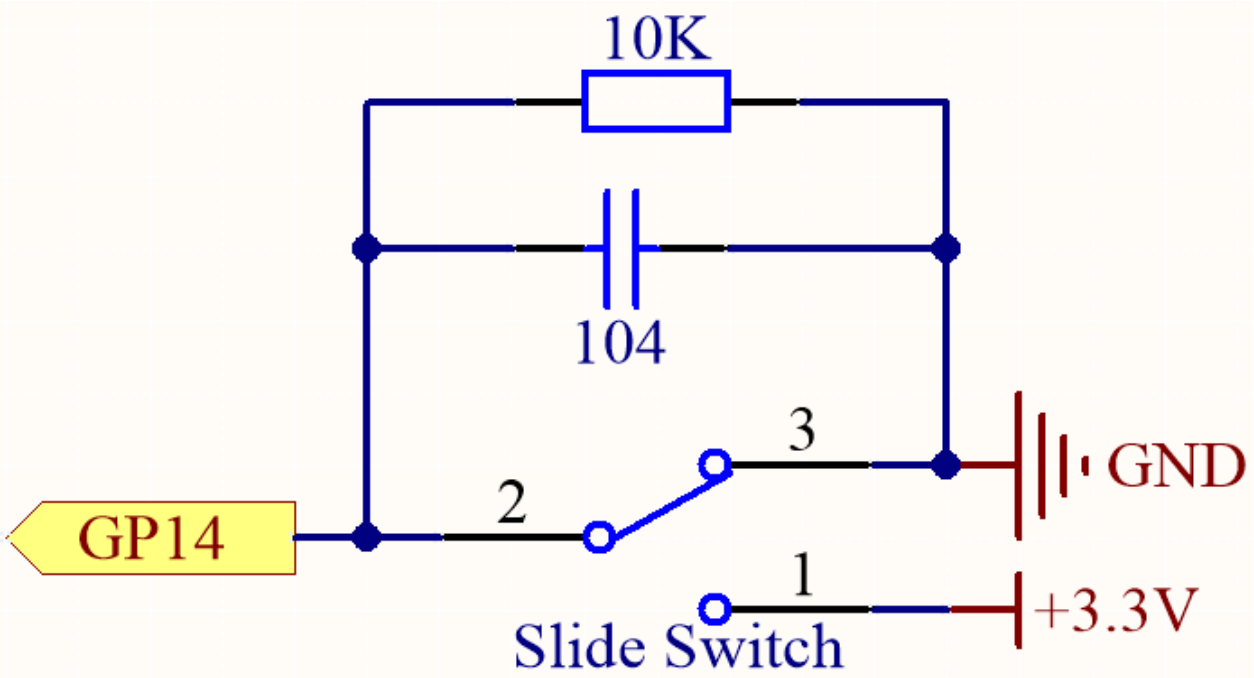
一式をまとめて購入するのは非常に便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**コンデンサ**	1(104)	
7	**スライドスイッチ**	1	

回路図



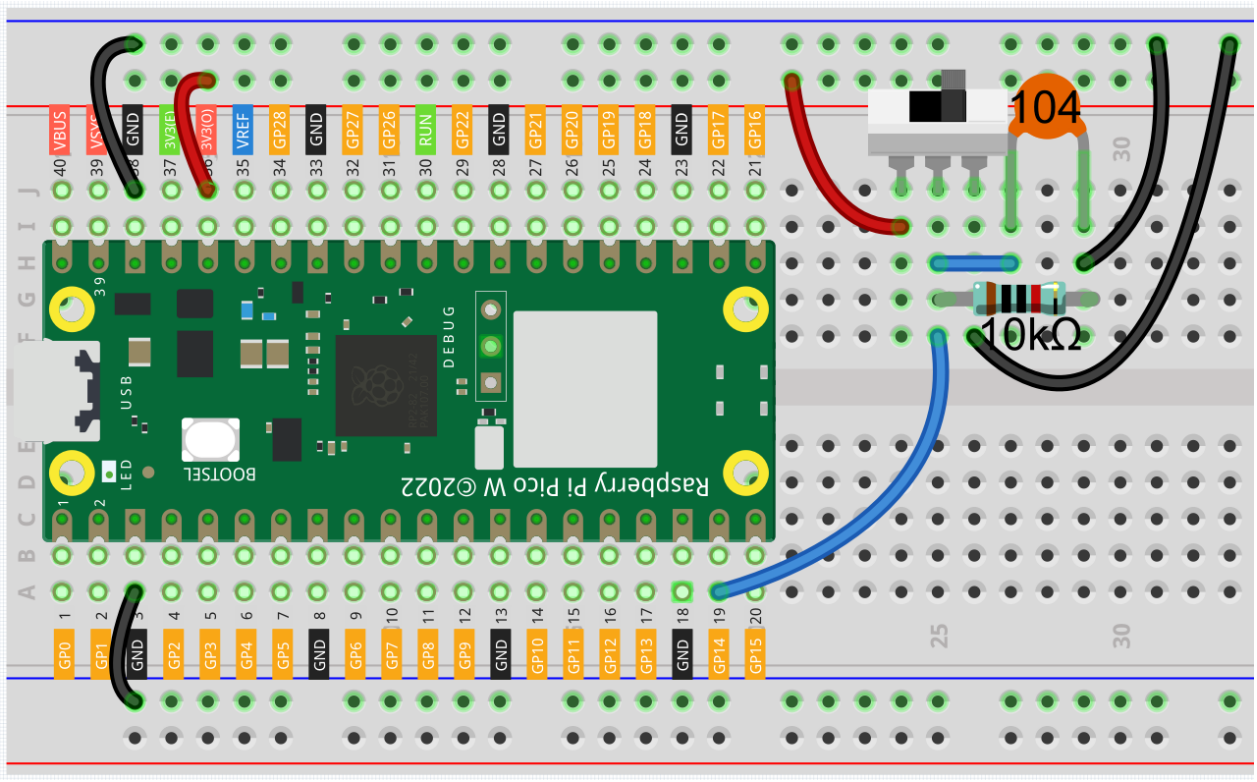
スライドスイッチを左右に切り替えると、GP14 のレベルが変わります。

10K の抵抗器の目的は、切り替え中に GP14 を低く保つことです（左端には切り替えず、右端にも切り替えない）。

104 のセラミックコンデンサは、ジッタを除去するために使用されています。

- **スライドスイッチ**
- **コンデンサ**

配線



コード

注釈:

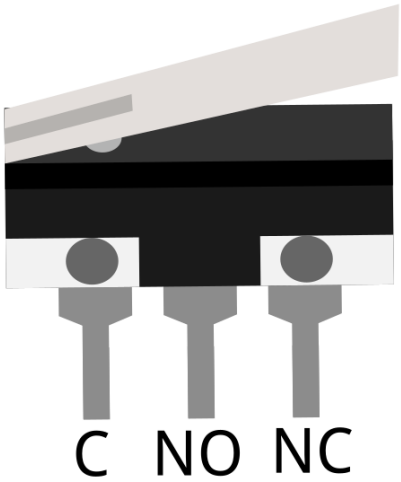
- kepler-kit-main/micropython のパス下にある 2.7_slide_switch.py ファイルを開くか、このコードを Thonny にコピペして、「Run Current Script」をクリックまたは F5 キーを押して実行します。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリターをクリックして選択してください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 0:
        print("The switch works!")
        utime.sleep(1)
```

プログラムが実行された後、スライドスイッチを右に切り替えると、シェルに「The switch works!」と表示され

ます。

4.14 2.8 やさしく押して



マイクロスイッチもまた 3 ピンのデバイスで、この 3 ピンの順序は C（共通ピン）、NO（通常開）および NC（通常閉）です。

マイクロスイッチが押されていない場合、1（C）と 3（NC）が接続され、押された場合は、1（C）と 2（NO）が接続されます。

- **マイクロスイッチ**

必要な部品

このプロジェクトでは、以下の部品が必要です。

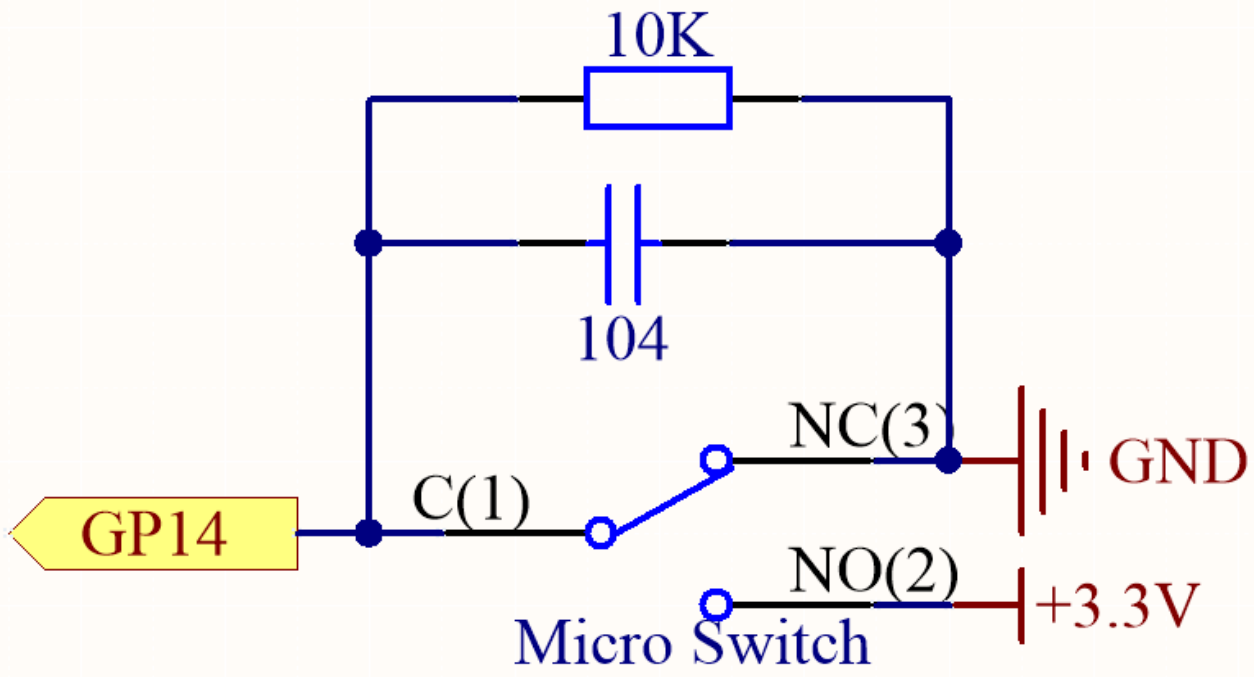
全ての部品が含まれるキットを購入するのは確かに便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから部品を個別に購入することもできます。

項番	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**コンデンサ**	1(104)	
7	**マイクロスイッチ**	1	

回路図

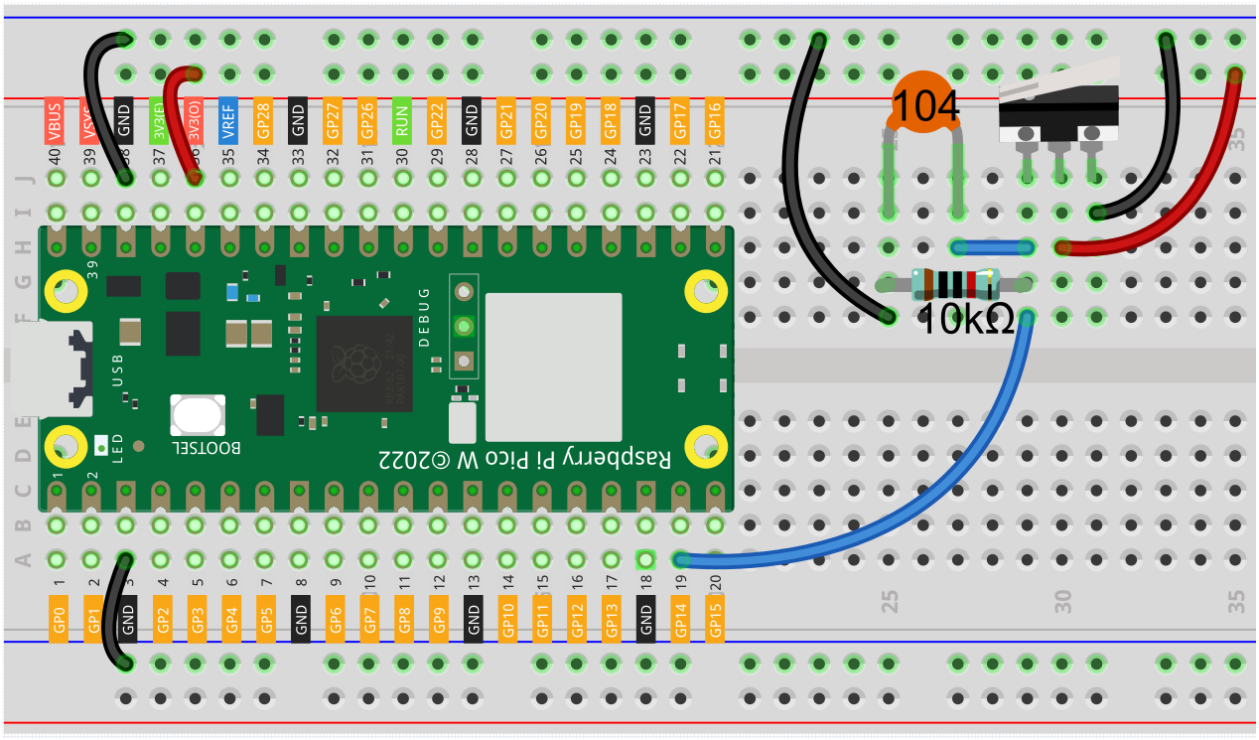


デフォルトでは、GP14 はローで、押されると GP14 はハイになります。

10K の抵抗器の目的は、押している間に GP14 を低く保つことです。

104 セラミックキャパシターは、ジッターを除去するためにここで使用されます。

配線



コード

注釈:

- kepler-kit-main/micropython パス下の 2.8_micro_switch.py ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある "MicroPython (Raspberry Pi Pico)" インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 1:
        print("The switch works!")
        utime.sleep(1)
```

プログラムが実行された後、スライドスイッチを右に切り替えると、シェルに「The switch works!」と表示されます。

4.15 2.9 磁気を感じる

最も一般的なタイプのリードスイッチは、スイッチが開いているときに小さな隙間で分離された、磁化可能で柔軟な金属製のリードの一対を含んでいます。

電磁石または永久磁石からの磁場が、リード同士を引き寄せることで電気回路を完成させます。磁場が消失すると、リードのバネ力によってそれらは分離し、回路が開きます。

リードスイッチの一般的な用途の一例は、セキュリティアラーム用にドアや窓が開いたことを検出することです。

- **リードスイッチ**

必要な部品

このプロジェクトでは、以下の部品が必要です。

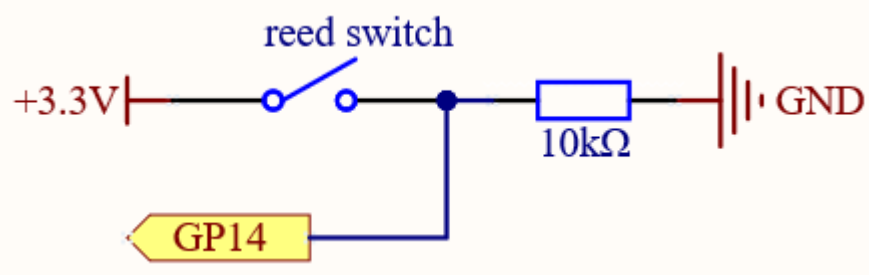
一式で購入する方が確実に便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**リードスイッチ**	1	

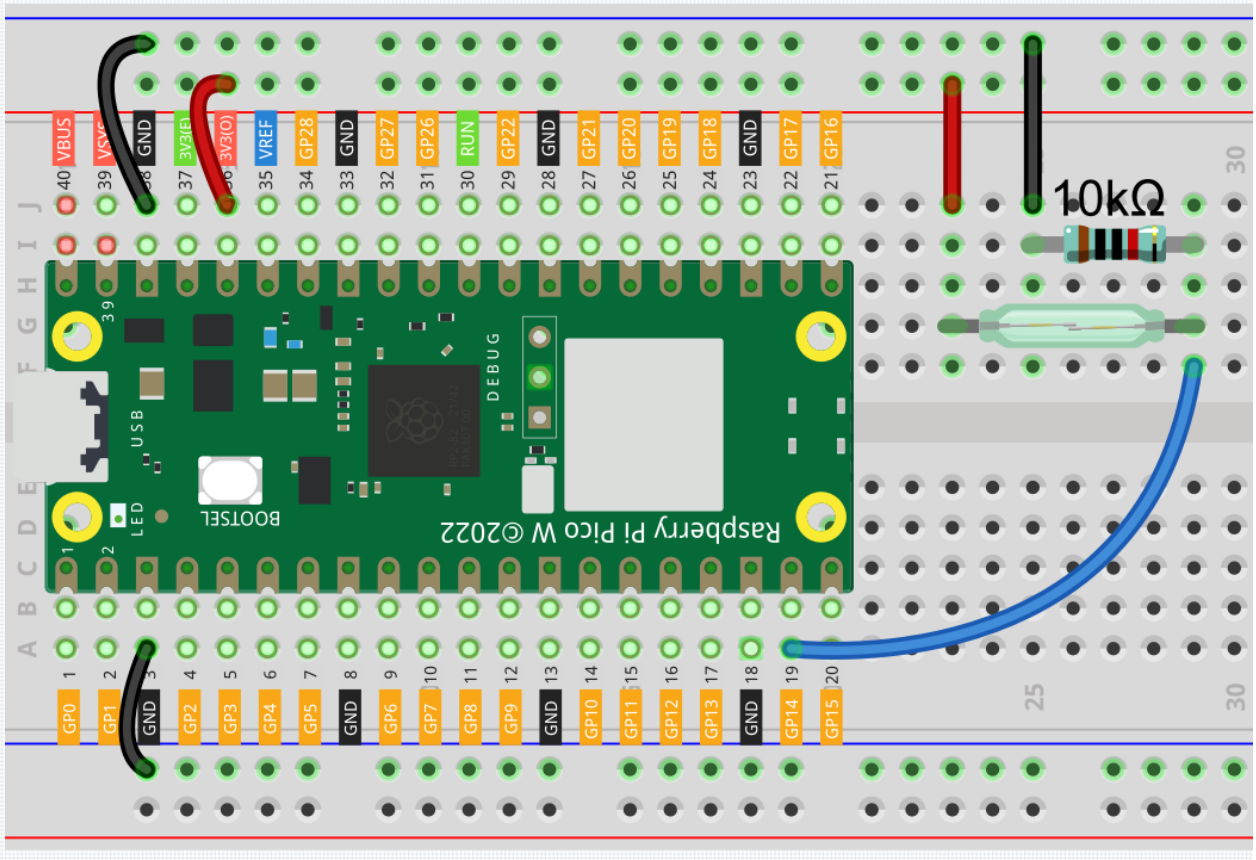
回路図



デフォルトでは、GP14 は低く、磁石がリードスイッチに近づくと高くなります。

10K Ωの抵抗の目的は、磁石が近くにないときに GP14 を安定した低レベルに保つことです。

配線



コード

注釈:

- `kepler-kit-main/micropython` のパスの下にある `2.9_feel_the_magnetism.py` ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 キーを押して実行してください。

- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
reed = machine.Pin(14, machine.Pin.IN)
while True:
    if reed.value() == 1:
        print("There are magnets here!!")
        utime.sleep(1)
```

コードが実行されると、リードスイッチに磁石が近づくと GP14 が高くなり、そうでない場合は低くなります。*2.5* **ボタンの値を読み取る** チャプターのボタンと同様です。

もっと詳しく

今回は、スイッチの柔軟な使い方を試してみました：割り込み要求、または IRQ（Interrupt Requests）。

例えば、あなたがプログラムがスレッドを実行しているかのように、ページごとに本を読んでいるとします。このとき、誰かが質問をしにきて、あなたの読書を中断しました。その人が割り込み要求を実行しています：あなたがやっていることをやめて、彼の質問に答え、その後で読書に戻らせます。

MicroPython の割り込み要求も同じように動作します。それは、特定の操作がメインプログラムを中断できるようにします。

注釈:

- `kepler-kit-main/micropython` のパスの下にある `2.9_feel_the_magnetism_irq.py` ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 キーを押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

reed_switch = machine.Pin(14, machine.Pin.IN)

def detected(pin):
```

(次のページに続く)

(前のページからの続き)

```
    print("Magnet!")


reed_switch.irq(trigger=machine.Pin.IRQ_RISING, handler=detected)
```

ここではまず、コールバック関数 `detected(pin)` が定義されています。これを割り込みハンドラーと呼びます。割り込み要求がトリガーされたときに実行されます。次に、メインプログラムで割り込み要求が設定されています。これには二つの部分が含まれています： `trigger` と `handler` 。

このプログラムで `trigger` は `IRQ_RISING` です。これは、ピンの値が低から高に変わること（つまり、ボタンの押下）を示します。

`handler` は、前に定義したコールバック関数 `detected` です。

- machine.Pin.irq - Micropython Docs

4.16 2.10 人間の動きを検出

受動型赤外線センサー（PIR センサー）は、視野内の物体が放出する赤外線（IR）光を測定できる一般的なセンサーです。簡単に言えば、体から放出される赤外線を受け取り、人や他の動物の動きを検出します。具体的には、誰かが部屋に入ったとメインコントロールボードに通知します。

PIR モーションセンサーモジュール

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下のとおりです。

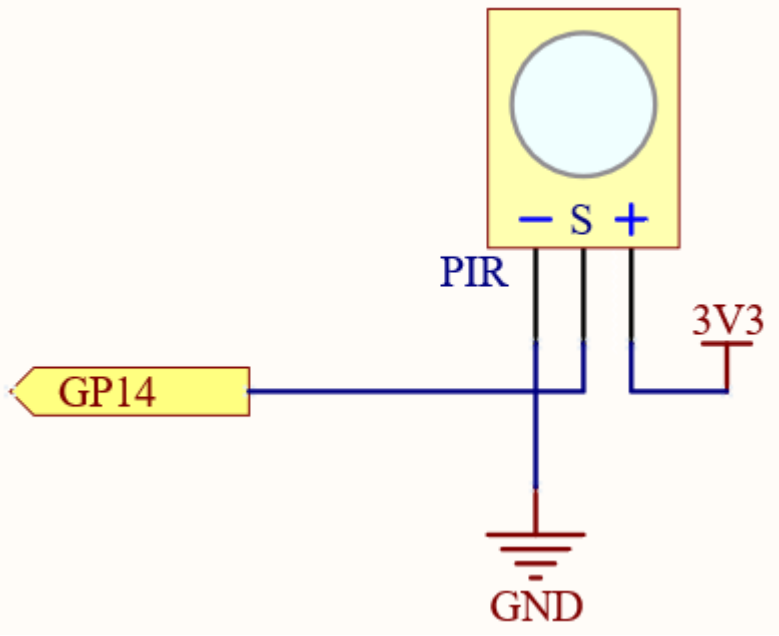
全体のキットを購入する方が確実に便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個々に購入することもできます。

S/N	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*PIR* **モーションセンサーモジュール**	1	

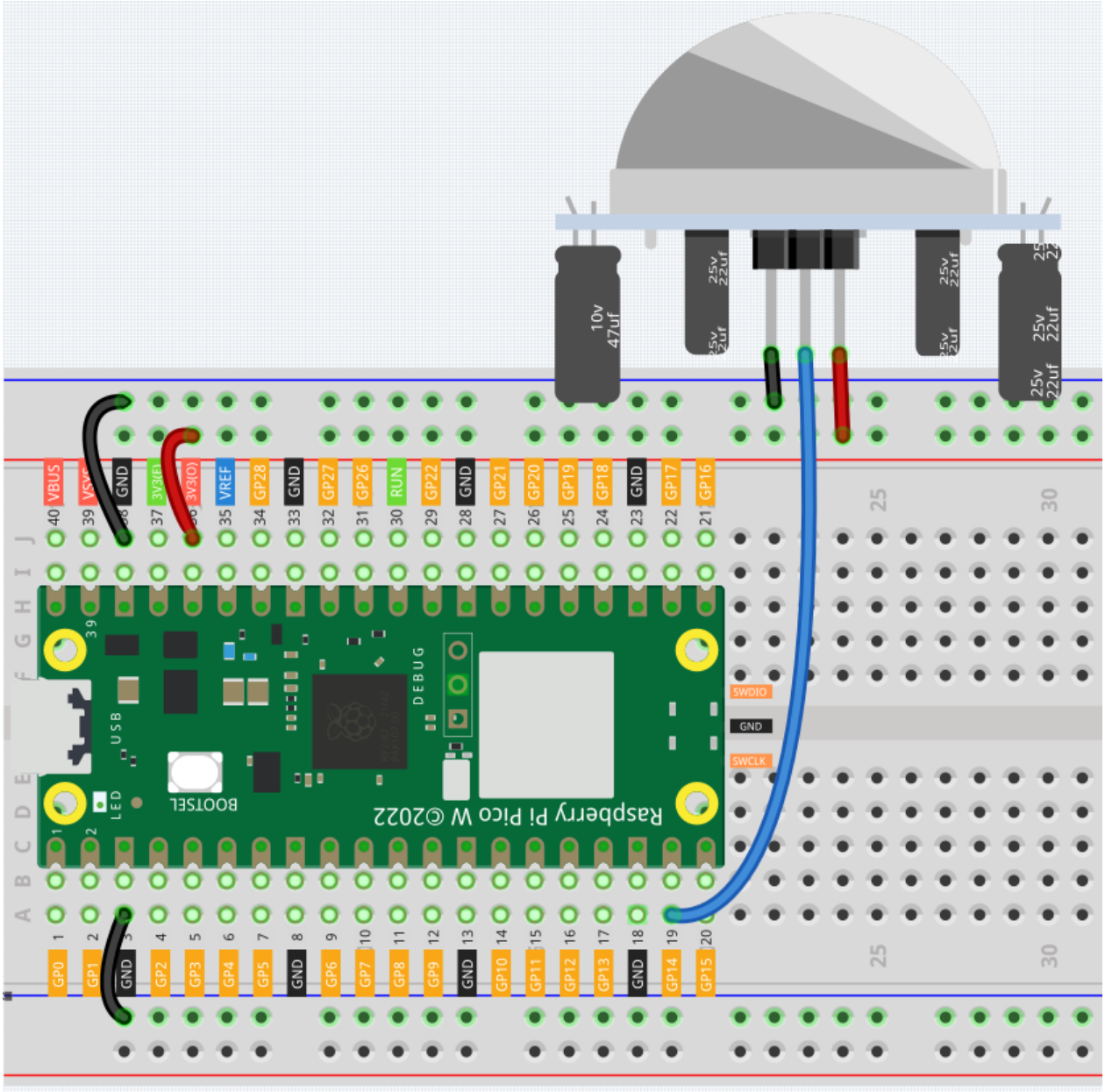
回路図



PIR モジュールが通過する人を検出すると、GP14 は高くなり、それ以外の場合は低くなります。

注釈: PIR モジュールには二つの可変抵抗があります：一つは感度を調整し、もう一つは検出距離を調整します。PIR モジュールをより効果的に動作させるには、両方を反時計回りに最後まで回してください。

配線



コード

注釈:

- `kepler-kit-main/micropython` のパス下の `2.10_detect_human_movement.py` ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 を押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタプリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

pir_sensor = machine.Pin(14, machine.Pin.IN)

def motion_detected(pin):
    print("Somebody here!")

pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=motion_detected)
```

プログラムが実行された後、PIR モジュールが近くに誰かを検出すると、シェルに「Somebody here!」と表示されます。

もっと詳しく

PIR は非常に敏感なセンサーです。使用環境に適応させるために調整が必要です。2 つの可変抵抗がある側を向けて、両方の可変抵抗を反時計回りに最後まで回し、L と中央のピンにジャンパーキャップを挿入してください。

注釈:

- kepler-kit-main/micropython のパス下の 2.10_pir_adjustment.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 を押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタプリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

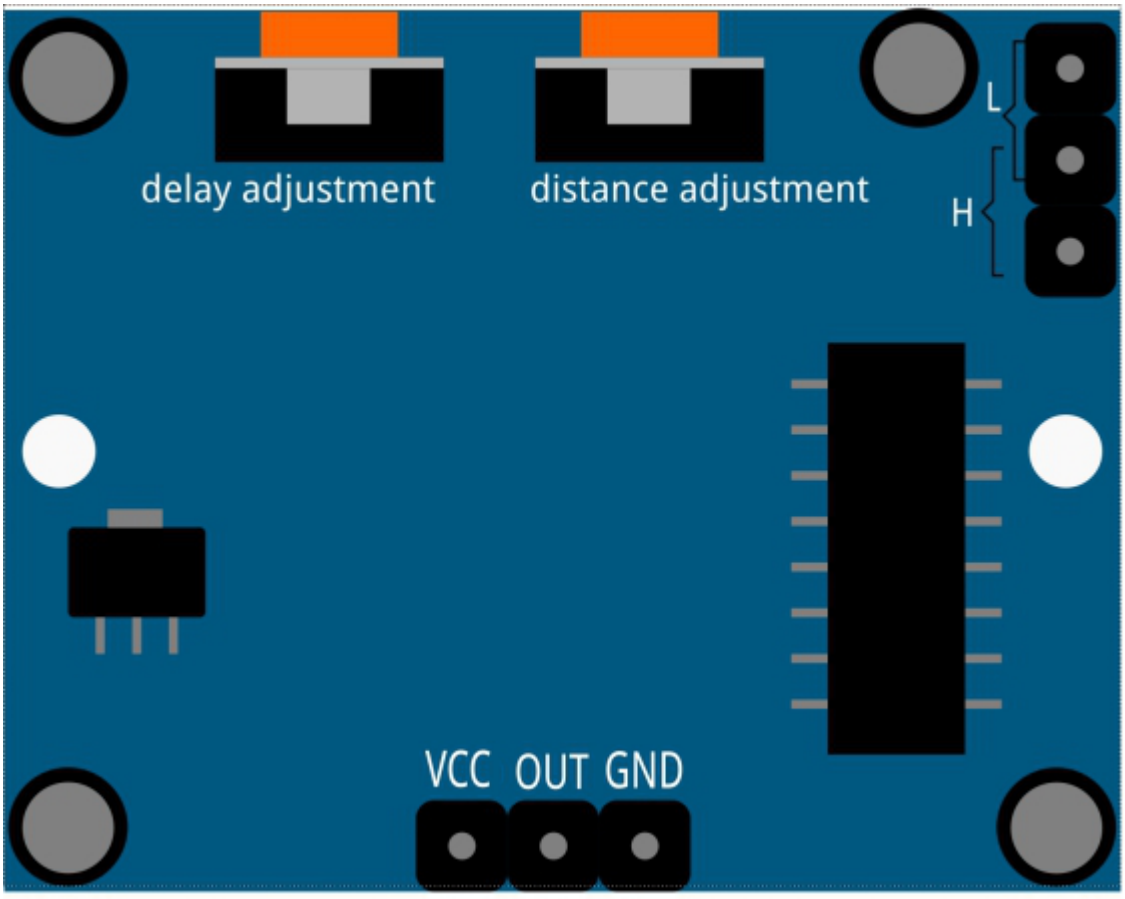
```
import machine
import utime

pir_sensor = machine.Pin(14, machine.Pin.IN)

global timer_delay
timer_delay = utime.ticks_ms()
print("start")

def pir_in_high_level(pin):
    global timer_delay
    pir_sensor.irq(trigger=machine.Pin.IRQ_FALLING, handler=pir_in_low_level)
    intervals = utime.ticks_diff(utime.ticks_ms(), timer_delay)
    timer_delay = utime.ticks_ms()
```

(次のページに続く)

(前のページからの続き)

```
    print("the dormancy duration is " + str(intervals) + "ms")

def pir_in_low_level(pin):
    global timer_delay
    pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_in_high_level)
    intervals2 = utime.ticks_diff(utime.ticks_ms(), timer_delay)
    timer_delay = utime.ticks_ms()
    print("the duration of work is " + str(intervals2) + "ms")

pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_in_high_level)
```

調整方法と実験結果を一緒に解析しましょう。



1. トリガーモード

   コーナーのジャンパーキャップがあるピンを見てみましょう。それによって PIR は、リピート可能なトリガーモードまたは非リピート可能なトリガーモードに入ります。

   現在、ジャンパーキャップは中央のピンと L ピンを接続しており、PIR は非リピート可能なトリガーモード

になっています。このモードでは、PIR が生体の動きを検出すると、約 2.8 秒間メインコントロールボードに高レベルの信号を送信します。印刷されたデータで見ると、作業の継続時間は常に約 2800ms 前後になります。

次に、下のジャンパーキャップの位置を変更して、中央のピンと H ピンを接続し、PIR をリピート可能なトリガーモードにします。このモードでは、PIR が生体の動きを検出する（センサーの前で静止しているのではなく、動いていることに注意）と、生体が検出範囲内で動き続ける限り、PIR はメインコントロールボードに高レベルの信号を送り続けます。印刷されたデータで見ると、作業の継続時間は不確かな値になります。

2. 遅延調整

   左側の可変抵抗は、二つの作業の間隔を調整するために使用されます。

   現在、反時計回りに最後まで回してありますので、PIR は高レベルの作業を送信し終えた後、約 5 秒のスリープ時間が必要です。この期間中、PIR は目標エリアでの赤外線放射を検出しません。印刷されたデータで見ると、休眠期間は常に 5000ms 以上になっています。

   可変抵抗を時計回りに回すと、スリープ時間も増加します。それを時計回りに最後まで回すと、スリープ時間は最大で 300 秒になります。

3. 距離調整

   中央の可変抵抗は、PIR の感知距離範囲を調整するために使用されます。

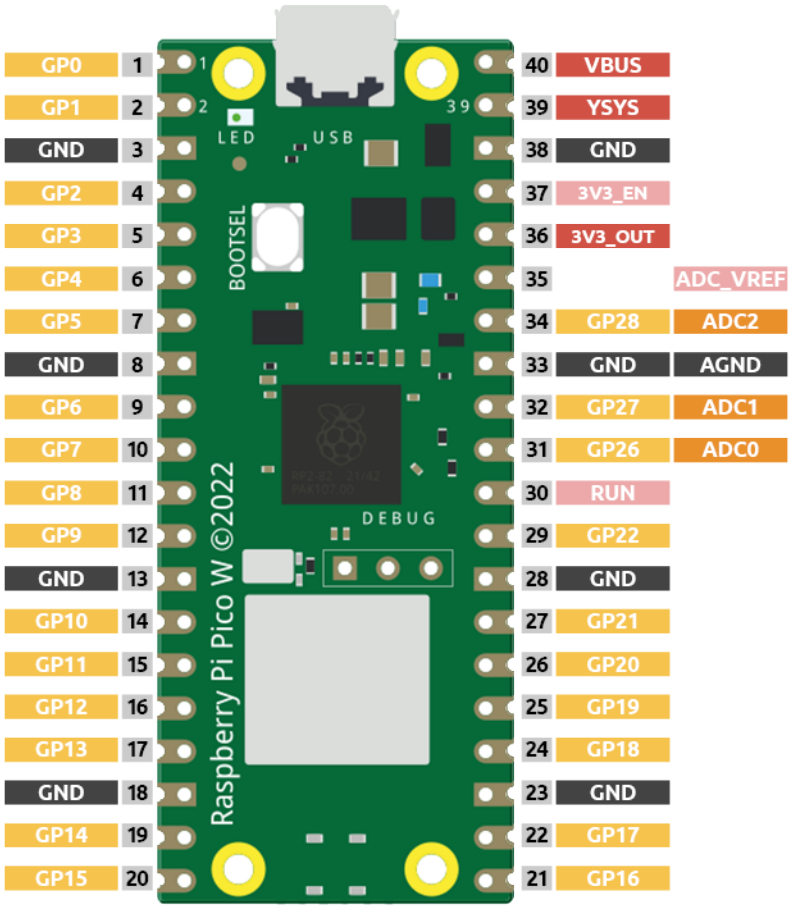
   距離調整の可変抵抗のノブを **時計回り** に回すと、感知距離範囲が増加し、最大感知距離範囲は約 0-7 メートルです。**反時計回り** に回すと、感知距離範囲が減少し、最小感知距離範囲は約 0-3 メートルです。

4.17 2.11 ノブを回してみよう

以前のプロジェクトでは、Pico W のデジタル入力を使用していました。たとえば、ボタンでピンのレベルを低（オフ）から高（オン）に変えることができます。これはバイナリの動作状態です。

しかし、Pico W は別のタイプの入力信号、すなわちアナログ入力も受け取ることができます。完全に閉じている状態から完全に開いている状態まで、様々な値をとることができます。アナログ入力により、マイクロコントローラは物理世界の光強度、音強度、温度、湿度などを感知することができます。

通常、マイクロコントローラにはアナログ入力を実装するための追加ハードウェア、すなわちアナログ-デジタルコンバータ（ADC）が必要です。しかし、Pico W 自体には ADC が組み込まれているので、直接使用することができます。



Pico W にはアナログ入力が使用できる GPIO ピンが 3 つあります：GP26、GP27、GP28。つまり、アナログチャンネル 0、1、2 です。さらに、内蔵温度センサに接続された第 4 のアナログチャンネルもありますが、ここでは紹介しません。

このプロジェクトでは、ポテンショメータのアナログ値を読み取ることに挑戦します。

- **ポテンショメーター**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

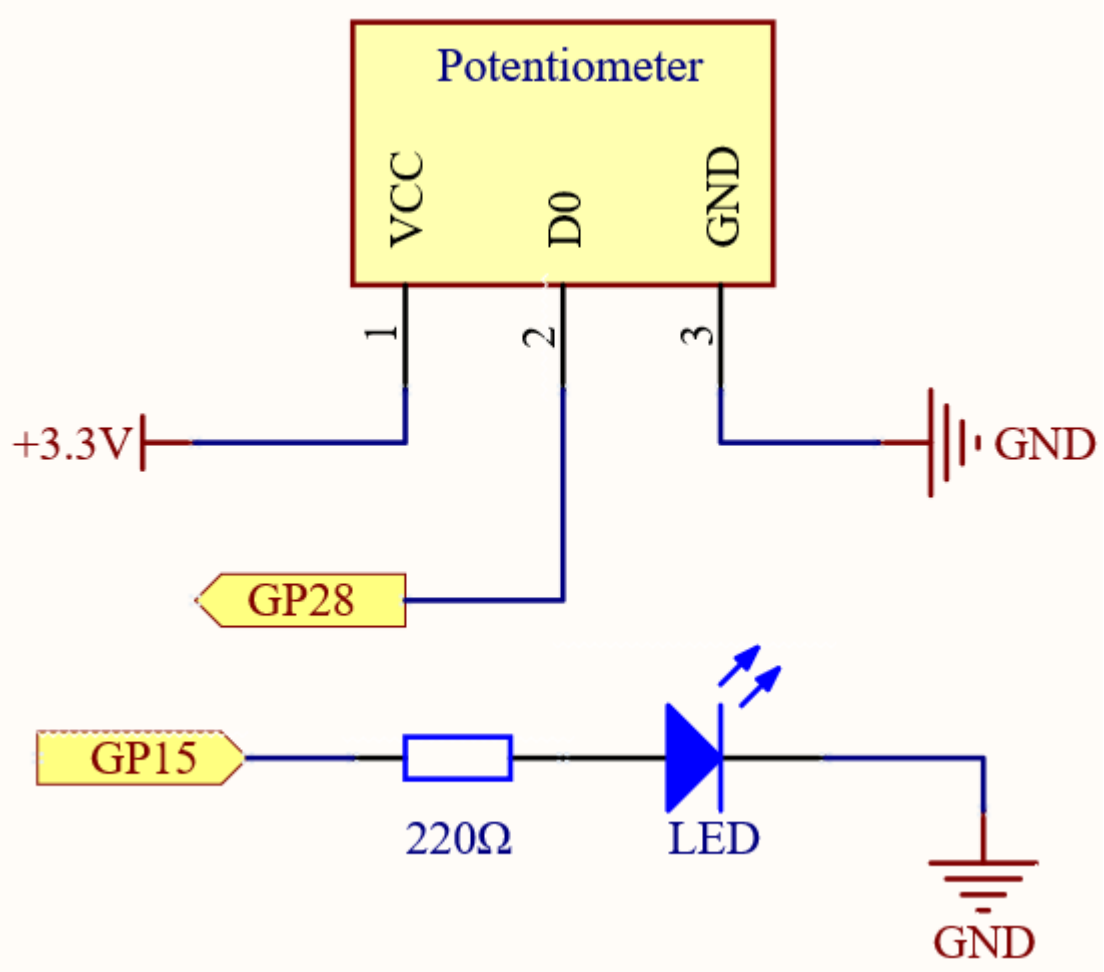
全体のキットを購入することは確かに便利です、以下がそのリンクです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

下記のリンクから別々にも購入することができます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	
7	**ポテンショメーター**	1	

回路図



ポテンショメータはアナログデバイスであり、2 つの異なる方向に回転させることができます。

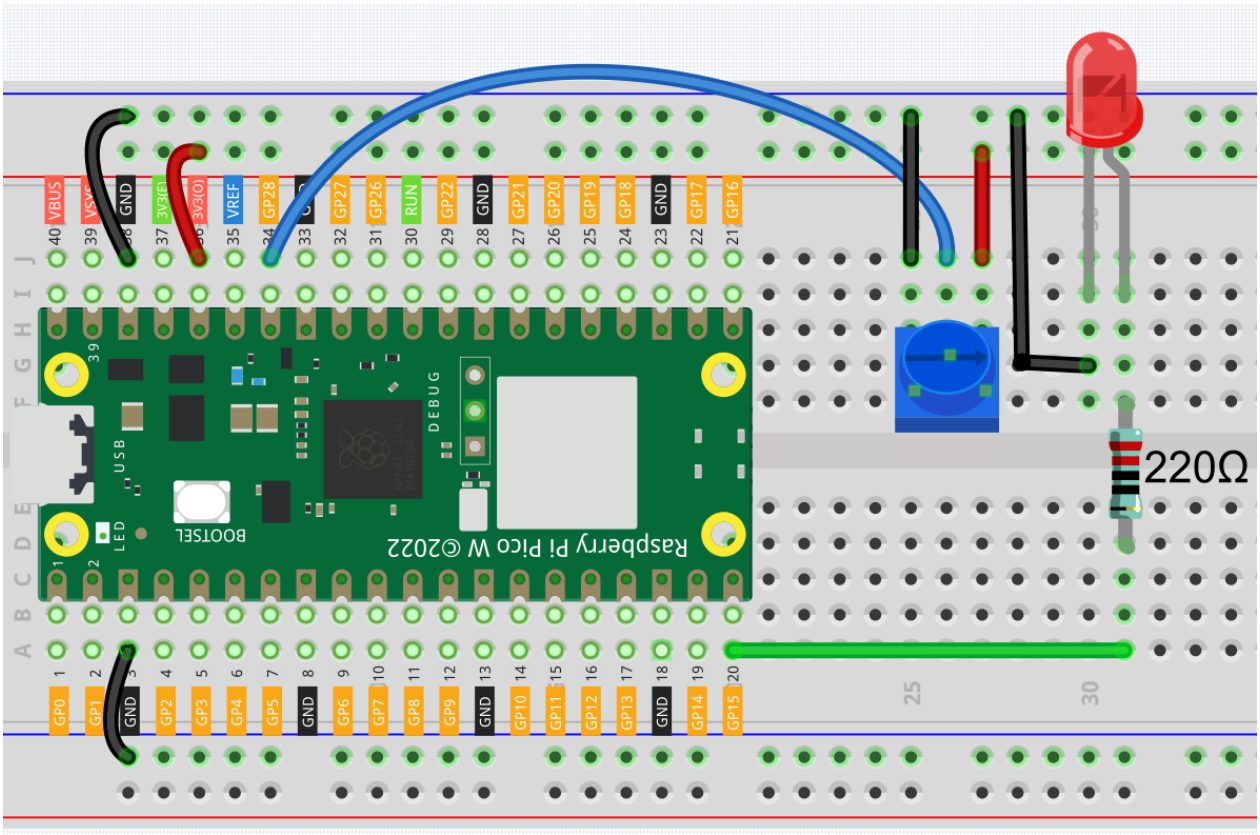
ポテンショメータの中央のピンをアナログピン GP28 に接続します。Raspberry Pi Pico W は、マルチチャネル、16 ビットのアナログ-デジタルコンバータを搭載しています。これにより、入力電圧が 0 から動作電圧（3.3V）の間で 0 から 65535 の整数値にマッピングされます。したがって、GP28 の値の範囲は 0 から 65535 です。

計算式は以下の通りです。

(Vp/3.3V) x 65535 = Ap

次に、GP28（ポテンショメータ）の値を GP15（LED）の PWM 値としてプログラムします。これにより、ポテンショメータを回転させると、LED の明るさも同時に変化することがわかります。

配線



コード

注釈:

- kepler-kit-main/micropython のパスの下にある 2.11_turn_the_knob.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 キーを押して実行します。
- 右下隅にある"MicroPython（Raspberry Pi Pico）"インタプリタをクリックするのを忘れないでください。

- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

potentiometer = machine.ADC(28)
led = machine.PWM(machine.Pin(15))
led.freq(1000)

while True:
    value = potentiometer.read_u16()
    print(value)
    led.duty_u16(value)
    utime.sleep_ms(200)
```

プログラムが動作しているとき、シェルで GP28 ピンが現在読み取っているアナログ値を確認できます。ノブを回すと、その値は 0 から 65535 に変化します。同時に、アナログ値が増加するにつれて、LED の明るさも増加します。

動作原理は？

```
potentiometer = machine.ADC(28)
```

この例では、id によって識別されたソースに関連付けられた ADC にアクセスします。この場合、それは GP28 です。

```
potentiometer.read_u16()
```

アナログ読み取りを行い、0〜65535 の範囲の整数を返します。返り値は、ADC によって取られた生の読み取り値を表し、最小値が 0 で最大値が 65535 になるようにスケーリングされています。

- machine.ADC - MicroPython Docs

4.18 2.12 光を感じる

フォトレジスタは典型的なアナログ入力デバイスであり、ポテンショメータと非常に類似した方法で使用されます。その抵抗値は光の強度に依存し、照射される光が強いほど抵抗値が小さくなり、逆に、光が弱いと抵抗値が増加します。

- **フォトレジスタ**

必要な部品

このプロジェクトでは、以下の部品が必要です。

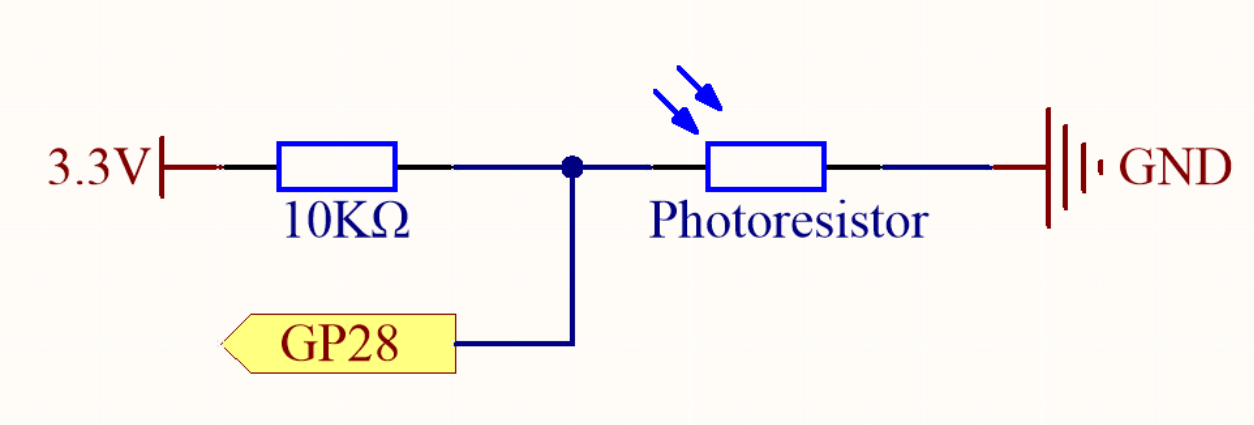
全体のキットを購入することは非常に便利です、以下がそのリンクです：

名前	このキットのアイテム	リンク
ケプラーキット	450 以上	

下記のリンクから個々の部品も購入可能です。

S/N	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	複数	
5	**抵抗器**	1(10K Ω)	
6	**フォトレジスタ**	1	

回路図



この回路では、10K Ωの抵抗とフォトレジスタが直列に接続され、両方を流れる電流は同じです。10K Ωの抵抗は保護として機能し、GP28 はフォトレジスタの電圧変換後の値を読み取ります。

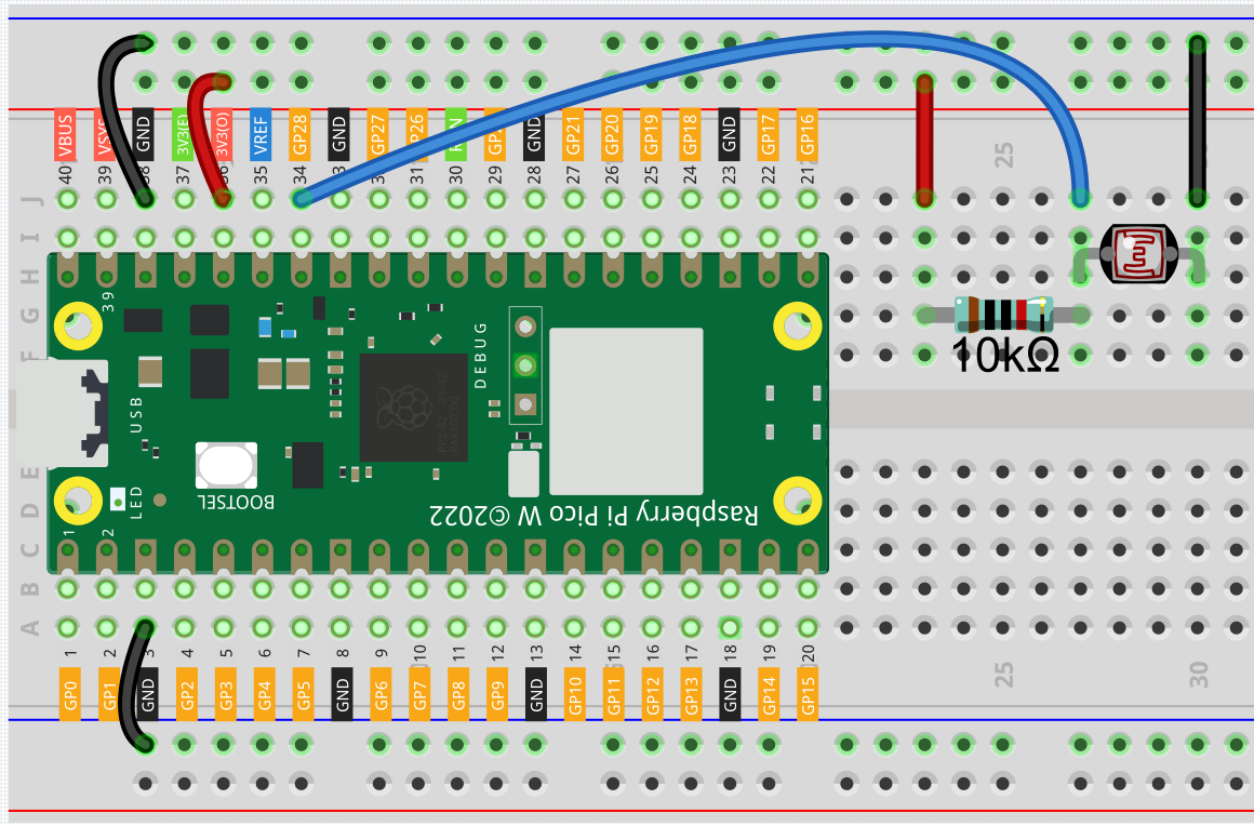
光が強くなると、フォトレジスタの抵抗が減少し、その電圧も減少します。その結果、GP28 からの値も減少します。光が十分に強い場合、フォトレジスタの抵抗はほぼ 0 になり、GP28 の値もほぼ 0 になります。この時、10K Ωの抵抗が保護役割を果たし、3.3V と GND が短絡するのを防ぎます。

フォトレジスタを暗い状況に置くと、GP28 の値が増加します。十分に暗い状況では、フォトレジスタの抵抗は無限大になり、その電圧はほぼ 3.3V（10K Ω の抵抗は無視できる）になり、GP28 の値は最大値 65535 に近づきます。

計算式は以下の通りです。

(Vp/3.3V) x 65535 = Ap

配線



コード

注釈:

- kepler-kit-main/micropython のパスの下にある 2.12_feel_the_light.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、F5 を押して実行してください。
- 右下隅の"MicroPython (Raspberry Pi Pico)" インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
```

(次のページに続く)

(前のページからの続き)

```
photoresistor = machine.ADC(28)

while True:
    light_value  = photoresistor.read_u16()
    print(light_value)
    utime.sleep_ms(10)
```

プログラムが実行された後、Shell にはフォトレジスタの値が出力されます。懐中電灯で照らすか、手で覆って値がどのように変わるかを確認できます。

4.19 2.13 温度計

温度計は、温度または温度勾配（物体の熱さや冷たさの度合い）を測定する装置です。温度計には 2 つの重要な要素があります：(1) 温度センサー（例えば、水銀温度計の球根や赤外線温度計の焦電センサー）で、これには温度の変化に伴い何らかの変化が生じます；そして（2）この変化を数値に変換する何らかの手段（例えば、水銀温度計にマークされた目盛りまたは赤外線モデルのデジタル表示)。温度計は、テクノロジーや産業でプロセスを監視するため、気象学、医学、科学研究で広く使用されています。

サーミスターは、温度に強く依存する抵抗値を持つ温度センサーの一種で、2 つのタイプがあります：負温度係数（NTC）と正温度係数（PTC)、別名 NTC および PTC です。PTC サーミスターの抵抗は温度とともに増加し、NTC の状態はそれと逆です。

この実験では、 **NTC サーミスター** を用いて温度計を作成します。

- **サーミスター**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

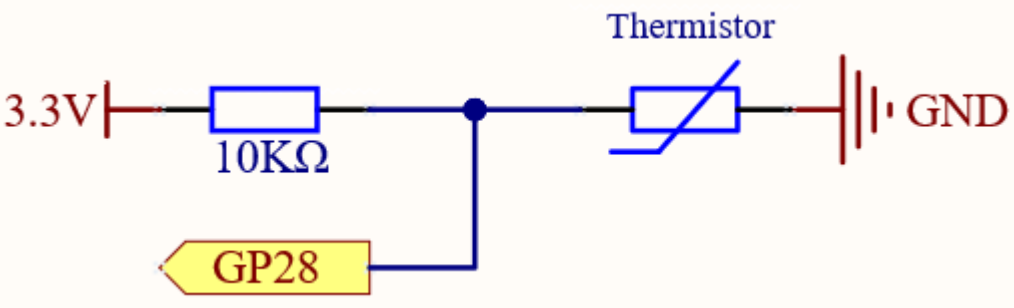
一式をまとめて購入するのは便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**サーミスター**	1	

回路図



この回路では、10K の抵抗とサーミスターが直列に接続されており、両方を通る電流は同じです。10K の抵抗は保護として機能し、GP28 はサーミスターの電圧変換後の値を読み取ります。

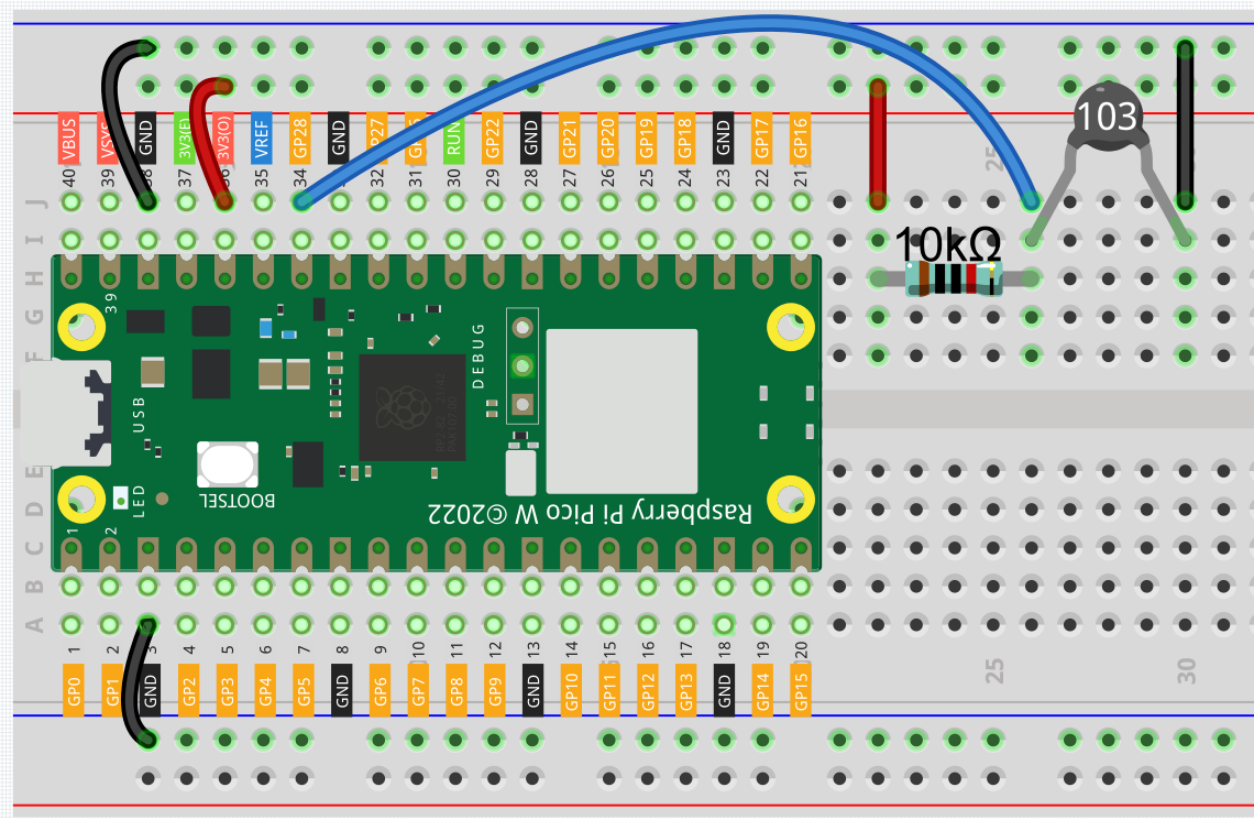
温度が上昇すると、NTC サーミスターの抵抗値は減少し、その電圧も減少するため、GP28 からの値も減少します。温度が十分に高い場合、サーミスターの抵抗はほぼ 0 に近く、GP28 の値もほぼ 0 に近くなります。このとき、10K の抵抗は保護として働き、3.3V と GND が直結し短絡することを防ぎます。

温度が下がると、GP28 の値は増加します。温度が十分に低い場合、サーミスターの抵抗は無限大になり、その電圧はほぼ 3.3V に近くなります（10K の抵抗は無視できます）、GP28 の値は最大値の 65535 に近くなります。

計算式は以下の通りです。

(Vp/3.3V) x 65535 = Ap

配線



注釈:

- サーミスターは黒く、103 とマークされています。
- 10K オームの抵抗器のカラーリングは赤、黒、黒、赤、茶です。

コード

注釈:

- `kepler-kit-main/micropython` のパスにある `2.13_thermometer.py` ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 キーを押して実行します。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターを忘れずにクリックしてください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
import math
```

(次のページに続く)

(前のページからの続き)

```
thermistor = machine.ADC(28)

while True:
    temperature_value = thermistor.read_u16()
    Vr = 3.3 * float(temperature_value) / 65535
    Rt = 10000 * Vr / (3.3 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
    utime.sleep_ms(200)
```

プログラムが実行された後、シェルは摂氏と華氏の温度を出力します。

仕組みは？

各サーミスターには通常の抵抗値があります。ここでは、それは 10k オームであり、25 度摂氏で測定されます。

温度が高くなると、サーミスターの抵抗が減少します。その後、A/D アダプターによって電圧データがデジタル量に変換されます。

プログラミングを介して摂氏または華氏での温度が出力されます。

```
import math
```

これは、一般的な数学的演算と変換を計算する一連の関数を宣言する数値ライブラリです。

- math

```
temperature_value = thermistor.read_u16()
```

この関数は、サーミスターの値を読み取るために使用されます。

```
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
Cel = temp - 273.15
Fah = Cel * 1.8 + 32
print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
utime.sleep_ms(200)
```

これらの計算は、サーミスターの値を摂氏度と華氏度に変換します。

```
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
```

上記の 2 行のコードでは、まず読み取ったアナログ値を使用して電圧を計算し、次に Rt（サーミスターの抵抗）を取得します。

```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

注釈: ここでは、抵抗と温度の関係が以下のようになっています：

RT = RN expB(1/TK – 1/TN)

- RT は、温度が TK のときの NTC サーミスターの抵抗です。
- RN は、定格温度 TN 下での NTC サーミスターの抵抗です。ここでは、RN の数値値は 10k です。
- TK はケルビン温度で、単位は K です。ここでは、TK の数値値は 273.15 + 摂氏度です。
- TN も定格ケルビン温度であり、単位も K です。ここでは、TN の数値値は 273.15+25 です。
- B（ベータ）は NTC サーミスターの材料定数であり、熱感度指数とも呼ばれ、数値値は 3950 です。
- exp は指数の略であり、基数 e は自然数で、約 2.7 に等しいです。

この関係は、実用的な公式です。温度と抵抗が有効範囲内にある場合にのみ正確です。

このコードは、ケルビン温度を取得するために、Rt を式 TK=1/(ln(RT/RN)/B+1/TN) に代入しています。

```
temp = temp - 273.15
```

ケルビン温度を摂氏度に変換します。

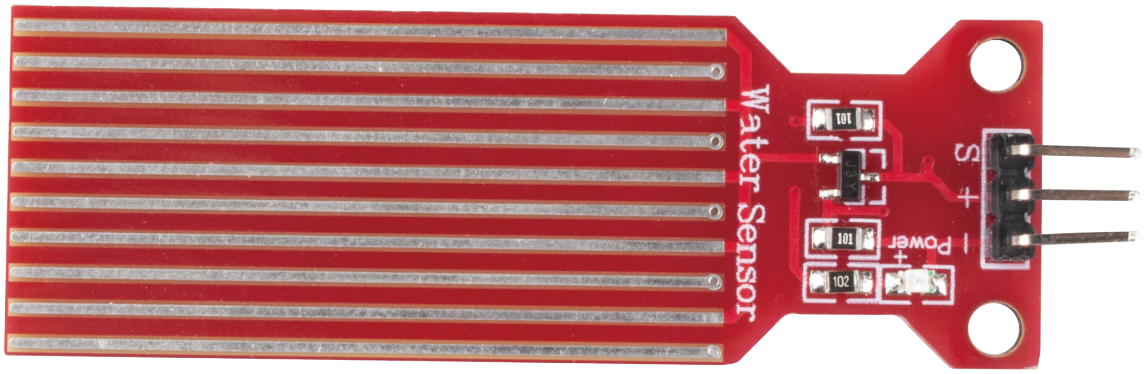
```
Fah = Cel * 1.8 + 32
```

摂氏度を華氏度に変換します。

```
print ('Celsius: %.2f ° C Fahrenheit: %.2f ^^e2^^84^^89' % (Cel, Fah))
```

シェルに摂氏度、華氏度、およびそれらの単位を出力します。

4.20 2.14 水位を感知する



この水センサーは、雨量、水位、さらには液体漏れを検知するために設計されています。

このセンサーは、水滴/水量の大きさを測定するために一連の露出した平行なワイヤートレースを使用して水位を測定します。水量は簡単にアナログ信号に変換され、出力アナログ値は、水位警報効果を実現するためにメインコントロールボードで直接読み取ることができます。

> **警告:** センサーは完全に水に浸けることはできません。十本のトレースがある部分だけを水に触れさせてください。また、湿度の高い環境でセンサーに電力を供給すると、プローブの腐食が加速し、センサーの寿命が短くなる可能性があります。したがって、読み取りを行う際にのみ電力を供給することをお勧めします。

- **水位センサーモジュール**

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

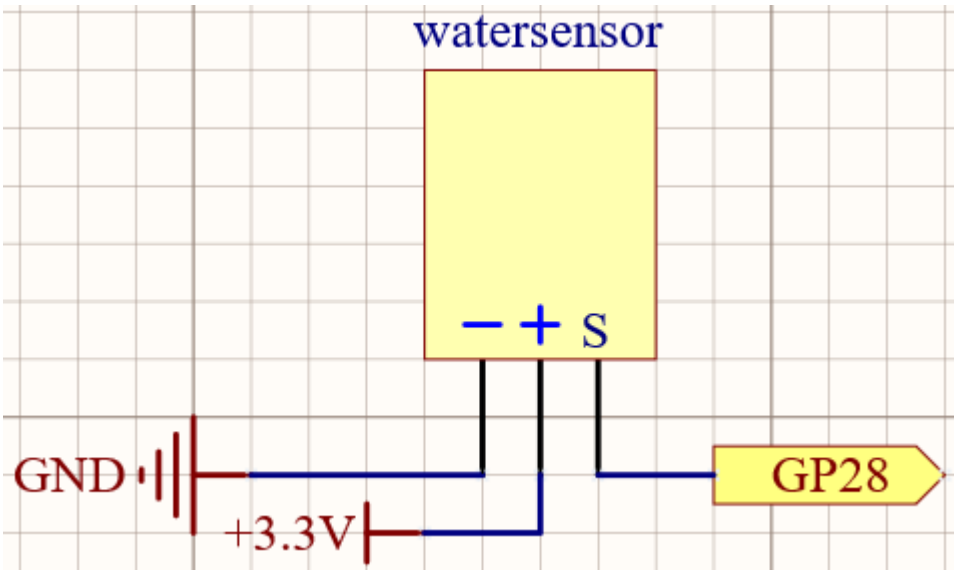
以下のリンクからキット全体を購入することが便利です：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

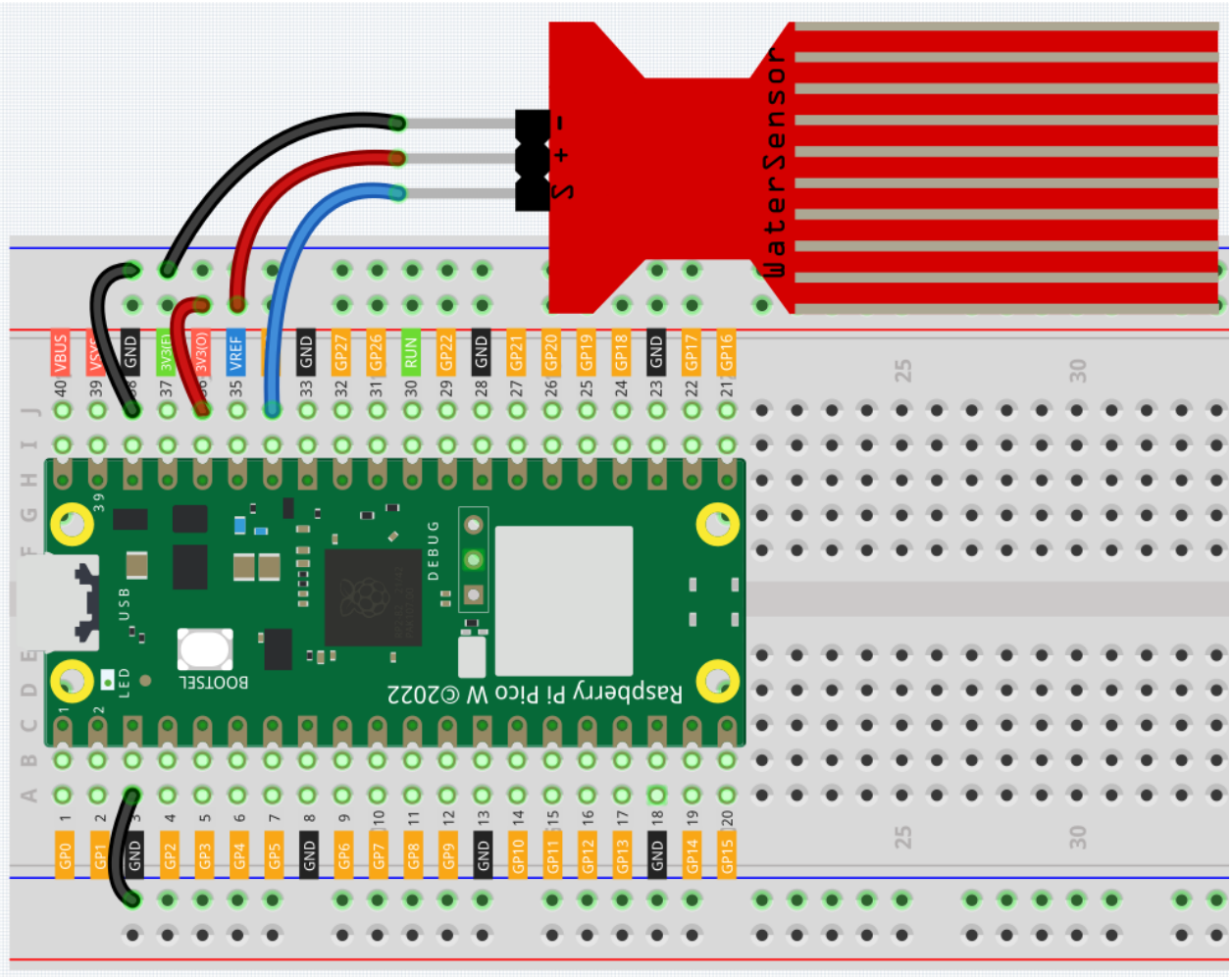
下記のリンクから個々に購入することもできます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**水位センサーモジュール**	1	

回路図



配線



コード

注釈:

- kepler-kit-main/micropython のパスにある 2.14_feel_the_water_level.py ファイルを開いて、このコードを Thonny にコピーします。その後、「Run Current Script」をクリックするか F5 キーを押して実行します。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime
```

(次のページに続く)

(前のページからの続き)

```
sensor = machine.ADC(28)

while True:
    value = sensor.read_u16()
    print(value)
    utime.sleep_ms(200)
```

プログラムを実行した後、水センサーモジュールをゆっくりと水に浸けます。深さが増すと、シェルはより大きな値を出力します。

詳しく学ぶ

アナログ入力モジュールをデジタルモジュールとして使用する方法があります。

まず、水センサーの乾燥した環境での読み取り値を記録し、それを閾値として使用します。次に、プログラミングを完了し、水センサーの読み取り値を再度読み取ります。水センサーの読み取り値が乾燥した環境での読み取り値と大きくずれている場合、液体に触れています。つまり、このデバイスを水道管の近くに置くと、水道管が漏れているかどうかを検出できます。

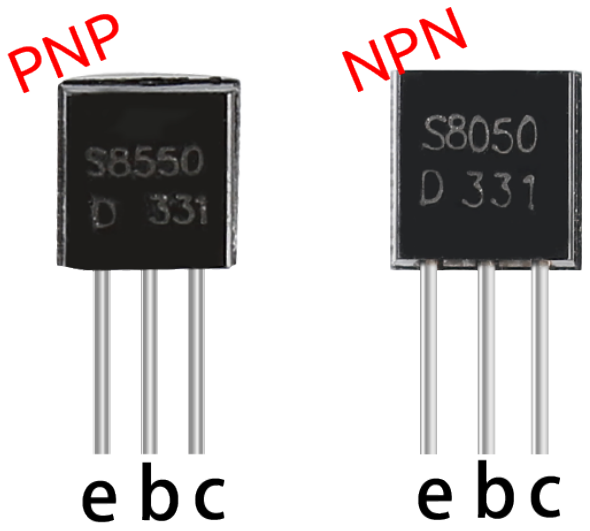
注釈:

- kepler-kit-main/micropython のパスにある 2.14_water_level_threshold.py ファイルを開いて、このコードを Thonny にコピーします。その後、「Run Current Script」をクリックするか F5 キーを押して実行します。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

sensor = machine.ADC(28)
threshold = 30000  # この値は環境に応じて修正する必要があります。

while True:
    value = sensor.read_u16()
    if value > threshold:
        print("Liquid leakage!")
    utime.sleep_ms(200)
```

4.21 2.15 トランジスタの 2 種類

このキットには、S8550 と S8050 という 2 種類のトランジスタが付属しています。前者は PNP で、後者は NPN です。見た目は非常によく似ているため、ラベルをよく確認する必要があります。NPN トランジスタにハイレベルの信号が通ると、それが励起されます。しかし、PNP トランジスタはローレベルの信号で制御されます。どちらのタイプのトランジスタも、この実験のように、非接触スイッチに頻繁に使用されます。



トランジスタの使い方を理解するために、LED とボタンを使ってみましょう！

トランジスタ

必要なコンポーネント

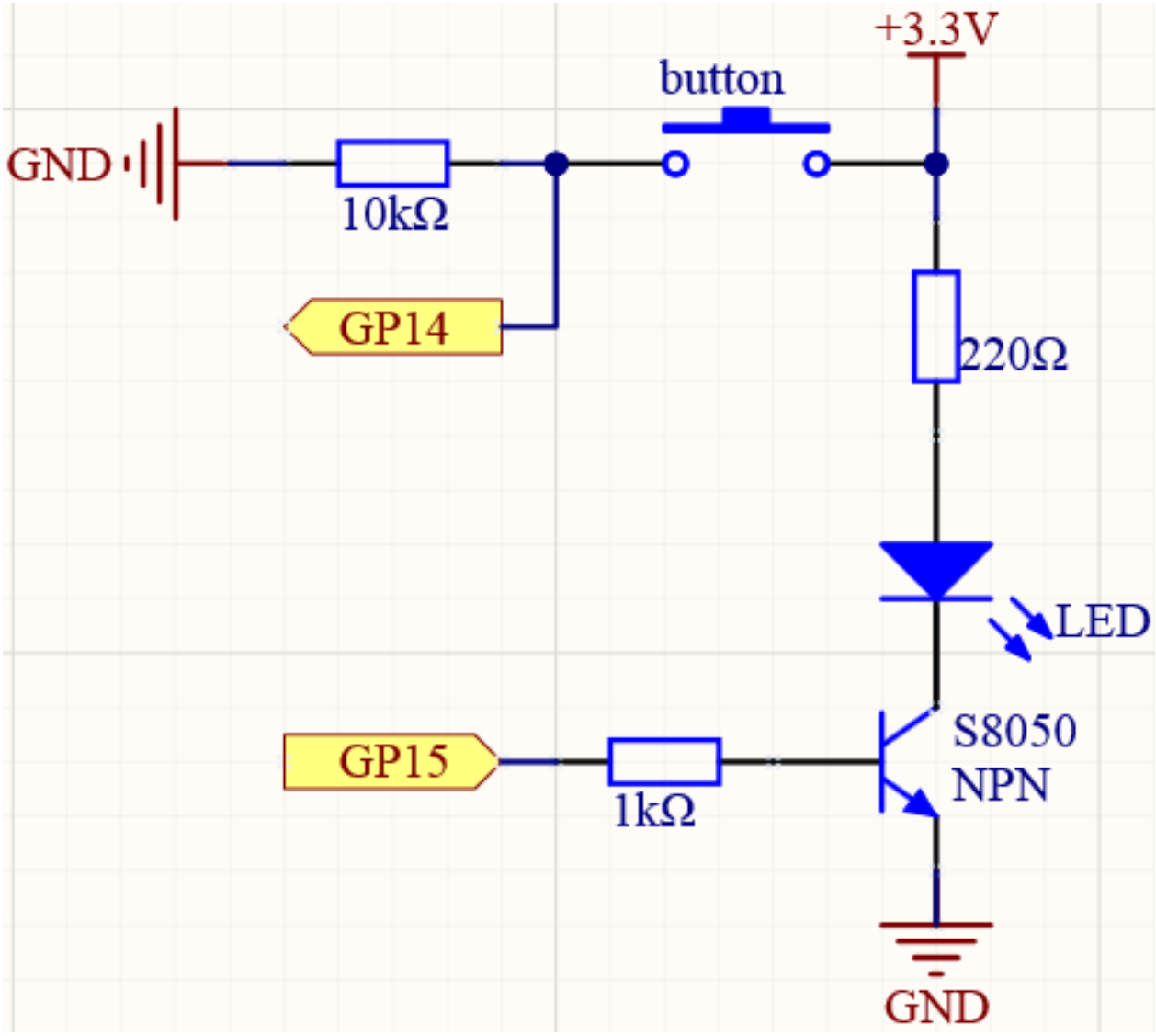
このプロジェクトでは、以下のコンポーネントが必要です。

名前	このキットに含まれるアイテム	リンク
Kepler キット	450 以上	

以下のリンクから個別に購入することも可能です。

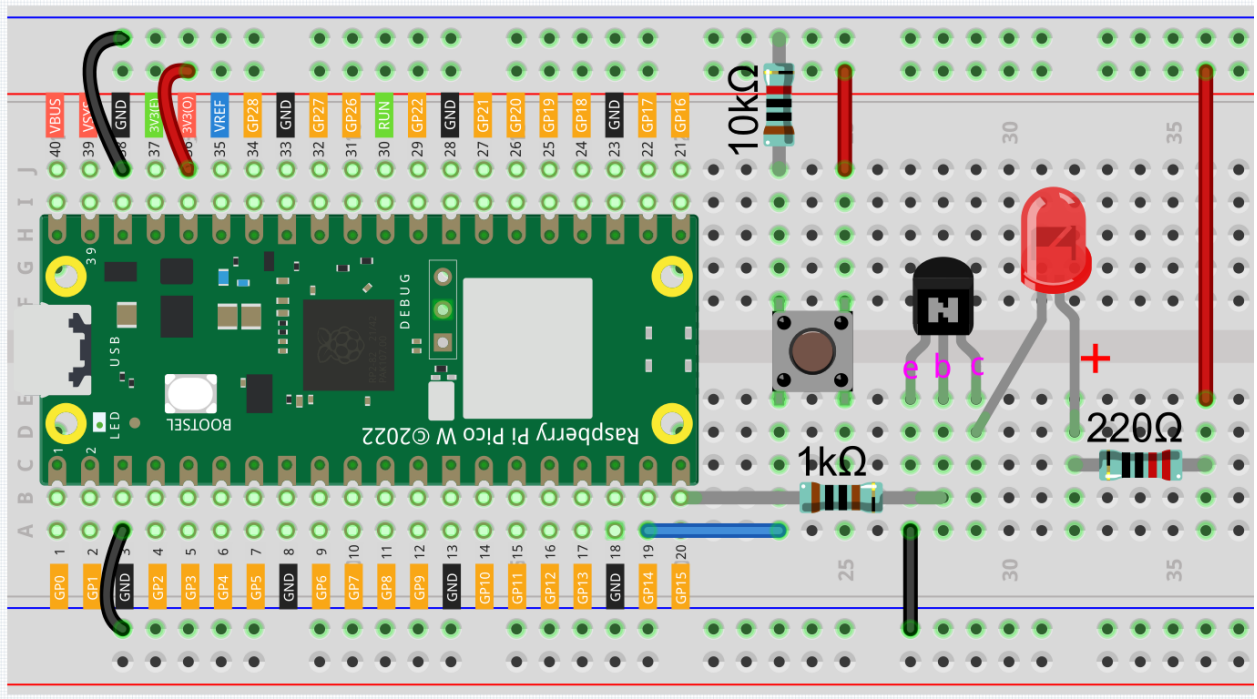
S/N	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	3(220 Ω， 1K Ω，10K Ω)	
6	*LED*	1	
7	**ボタン**	1	
8	**トランジスタ**	1(S8050/S8550)	

NPN（S8050）トランジスタの接続方法

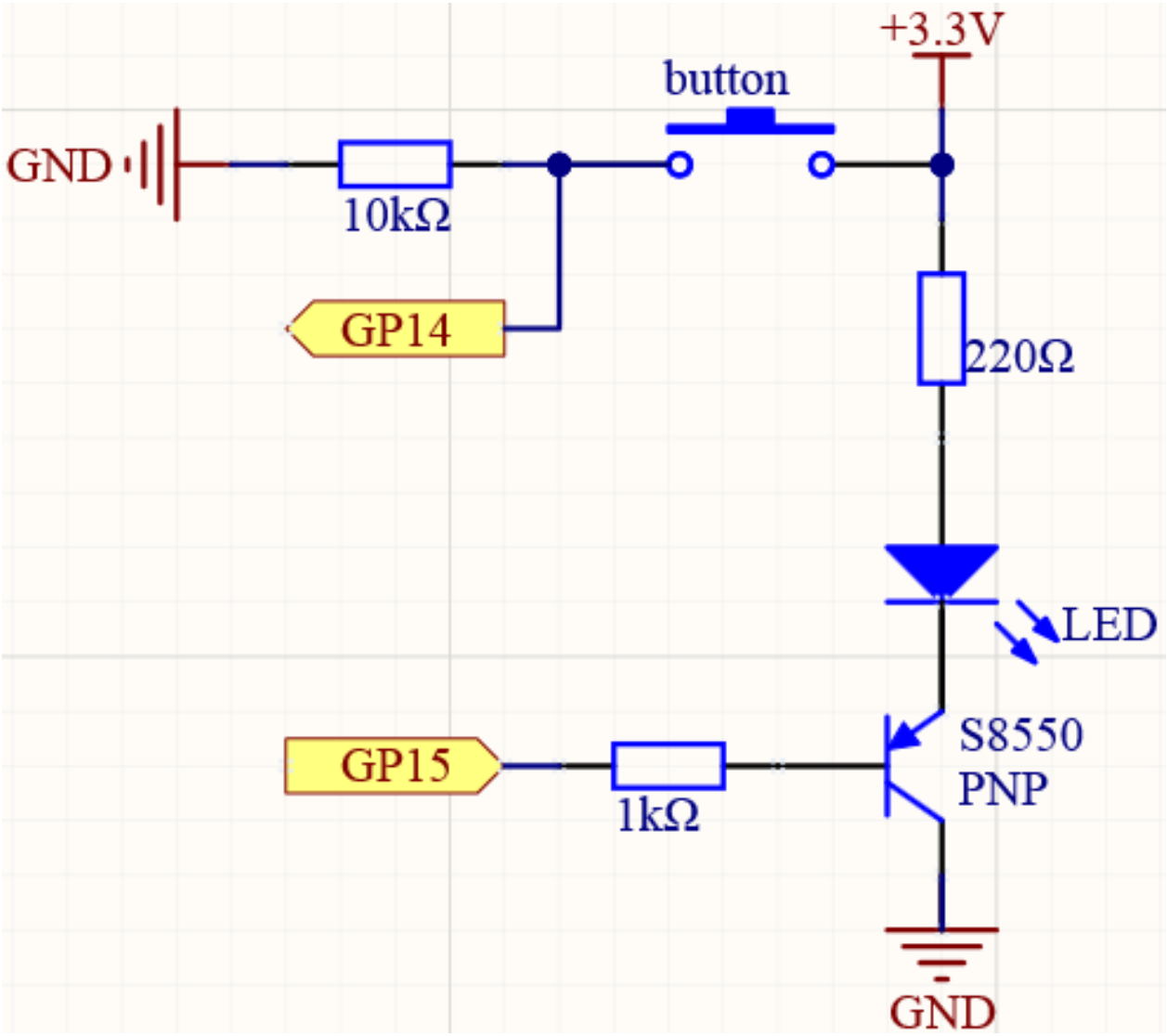


この回路では、ボタンが押されると、GP14 はハイになります。

プログラミングによって GP15 をハイ出力に設定すると、1k の電流制限抵抗を介して（トランジスタを保護するために）、S8050（NPN トランジスタ）は導通が許可され、それによって LED が点灯します。



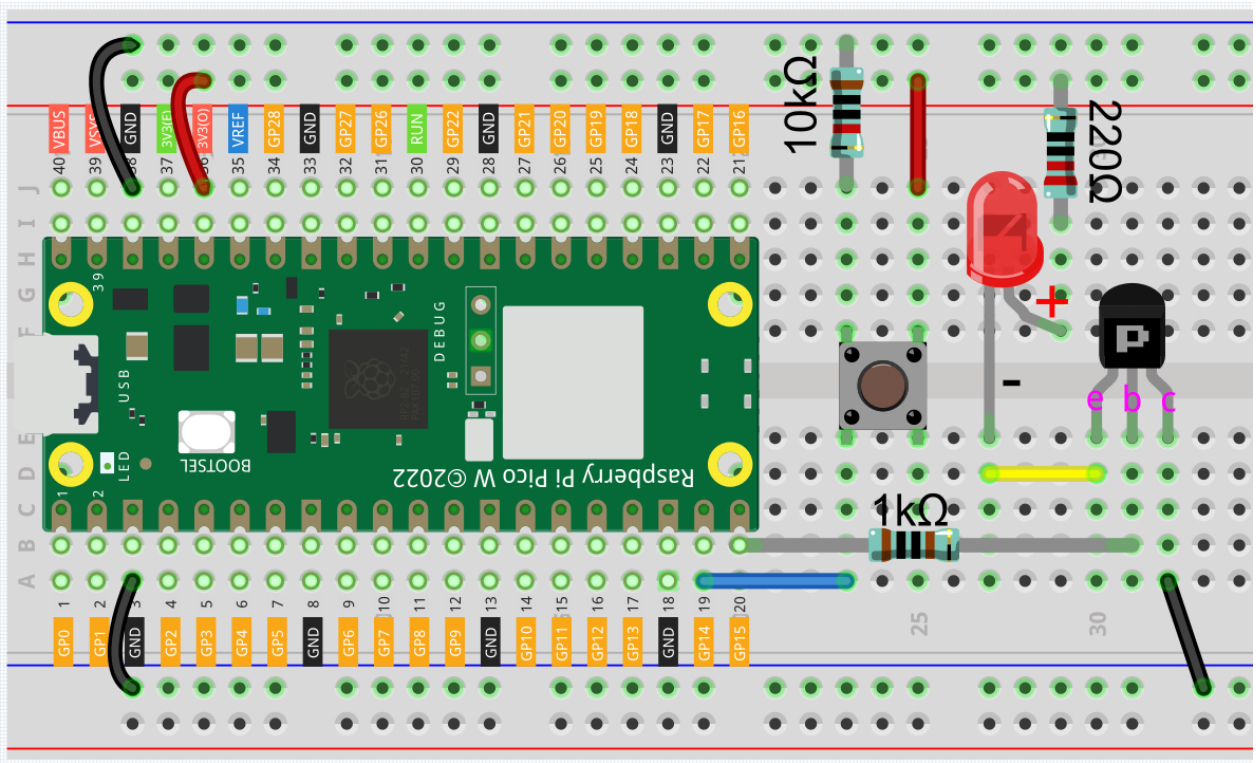
PNP（S8550）トランジスタの接続方法



この回路では、GP14 はデフォルトで低く、ボタンが押されると高くなります。

GP15 を **低出力** にプログラムすると、1k の電流制限抵抗（トランジスタを保護するため）の後、S8550（PNP トランジスタ）が導通することが許可され、それによって LED が点灯します。

この回路と前の回路との唯一の違いは、前の回路では LED のカソードが **S8050（NPN トランジスタ）** の **コレクタ** に接続されているのに対し、この回路では **S8550（PNP トランジスタ）** の **エミッタ** に接続されている点です。



コード

注釈:

- kepler-kit-main/micropython のパス下の 2.15_transistor.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」のインタープリタをクリックすることを忘れずに。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
button = machine.Pin(14, machine.Pin.IN)
signal = machine.Pin(15, machine.Pin.OUT)

while True:
    button_status = button.value()
    if button_status== 1:
        signal.value(1)
    elif button_status == 0:
        signal.value(0)
```

同じコードで 2 種類のトランジスタを制御できます。ボタンを押すと、Pico W はトランジスタにハイレベルの信号を送り、それを解放するとローレベルの信号を送ります。この 2 つの回路で真逆の現象が起きていることがわかります。

- S8050（NPN トランジスタ）を使用した回路は、ボタンが押されると点灯します。これはハイレベルの導通回路であるという意味です。
- S8550（PNP トランジスタ）を使用した回路は、解放されると点灯します。これはローレベルの導通回路であるという意味です。

4.22 2.16 他の回路を制御する

日常生活で、スイッチを押して照明を点けたり消したりしますが、Pico W を使って 10 分後に自動的に照明を消したい場合はどうでしょうか？

そのようなアイデアを実現するためには、リレーが役立ちます。

リレーは、一方の回路（通常は低電圧回路）で制御され、他方の回路（通常は高電圧回路）を制御する特別な種類のスイッチです。これにより、家庭用電化製品をプログラムで制御可能なスマートデバイスに改造したり、インターネットに接続したりすることが現実的になります。

> **警告:** 電化製品の改造は大きな危険性がありますので、専門家の指導の下で行ってください。

- リレー

今回は、ブレッドボードの電源モジュールで駆動される簡単な回路を例に、リレーを使用してどのように制御するかを説明します。

- cpn_power_module

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

一式をまとめて購入する方が便利です。以下がリンクです：

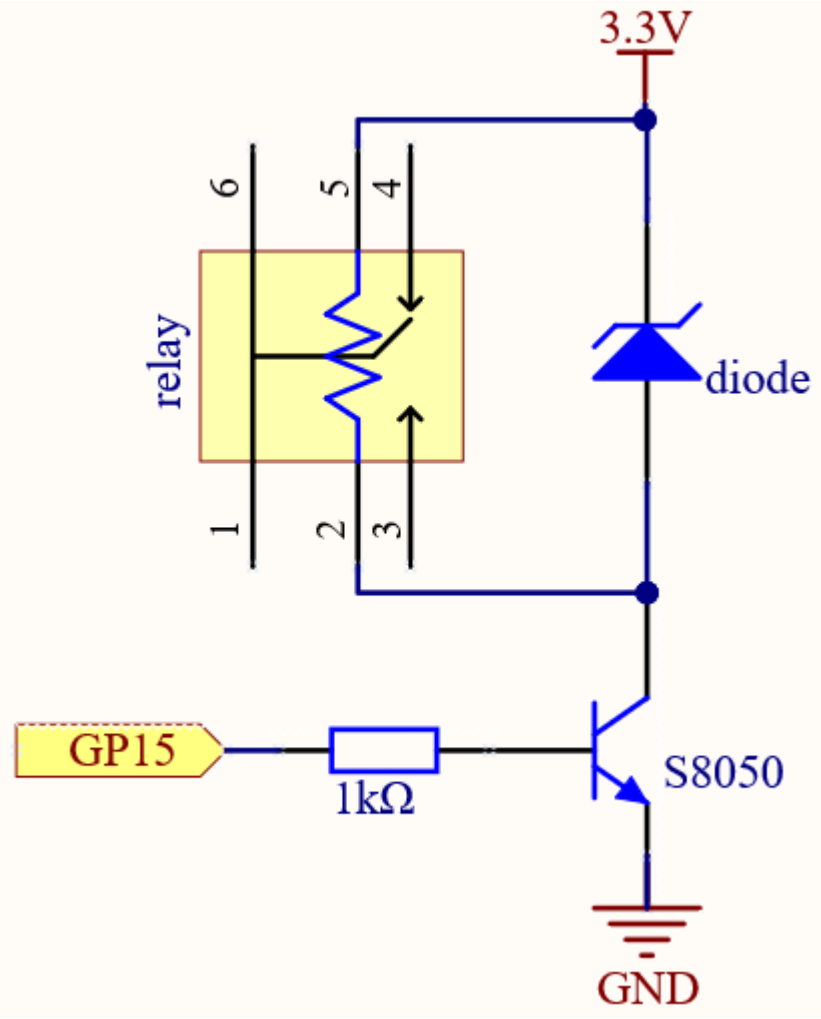
名前	キットの内容	リンク
ケプラーキット	450 以上	

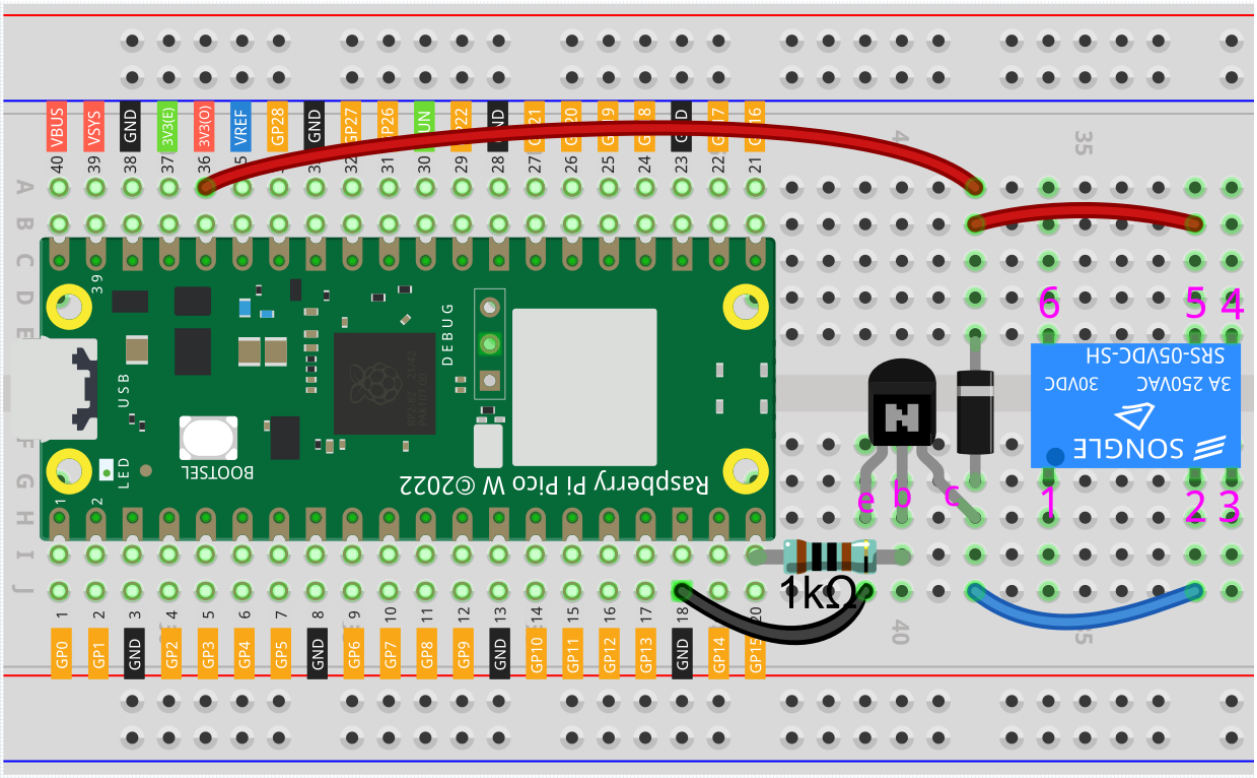
下のリンクからも個別に購入することができます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1 （S8050）	
6	**ダイオード**	1	
7	**リレー**	1	

配線

まず、リレーを制御するための低電圧回路を作ります。リレーを駆動するには高電流が必要なため、トランジスタが必要です。今回は S8050 を使用します。


3.3V
6
5
4
relay
diode
1
2
3
GP15
1kΩ
S8050
GND



ここではダイオード（フリーホイールダイオード）が回路を保護する役割を果たしています。カソード（マイナス側）は電源に接続された銀色のリボンの端であり、アノード（プラス側）はトランジスタに接続されています。

電圧入力が High（5V）から Low（0V）に変わると、トランジスタは飽和状態（増幅、飽和、カットオフ）からカットオフ状態に変わり、コイルを通る電流が突然途切れます。

この時点でフリーホイールダイオードが存在しない場合、コイルは供給電圧よりも何倍も高い自己誘導起電力を両端に発生させ、この電圧とトランジスタの電源からの電圧が加わって、トランジスタを焼き切ってしまいます。

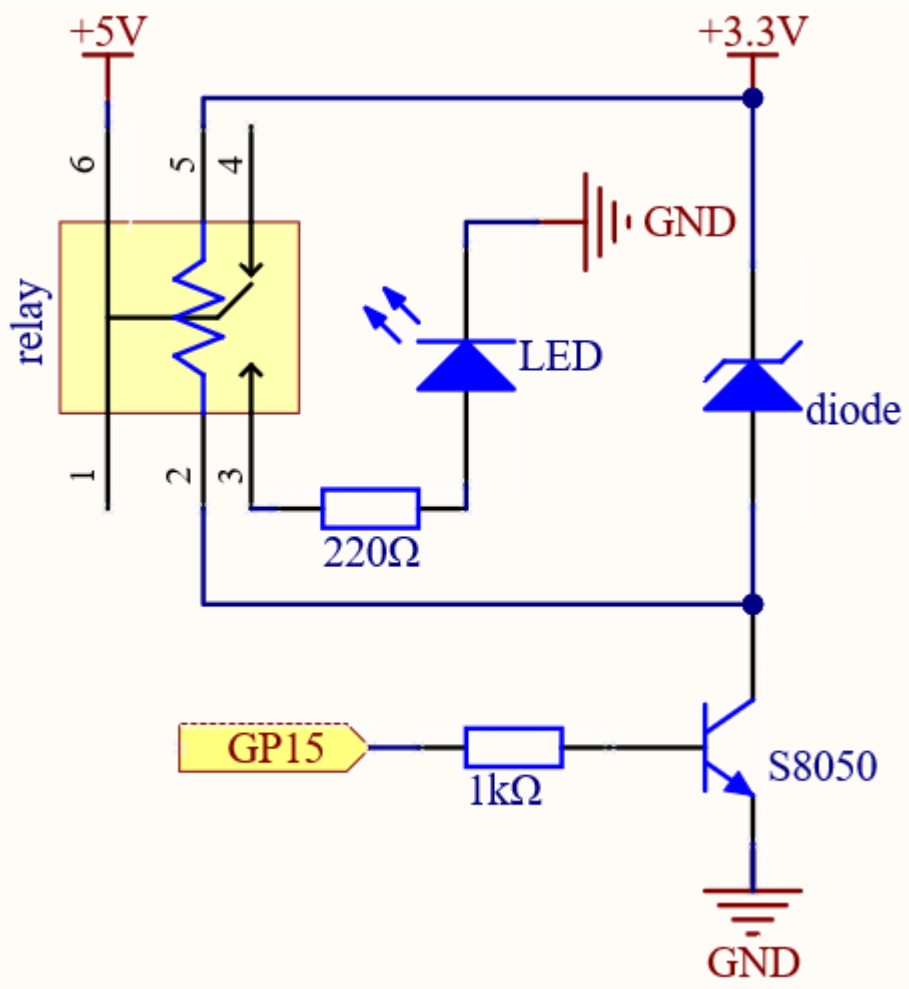
ダイオードを追加すると、コイルとダイオードは瞬時に新しい回路を形成し、コイルに蓄えられたエネルギーで放電し、回路上のトランジスタなどのデバイスが過度な電圧でダメージを受けるのを防ぎます。

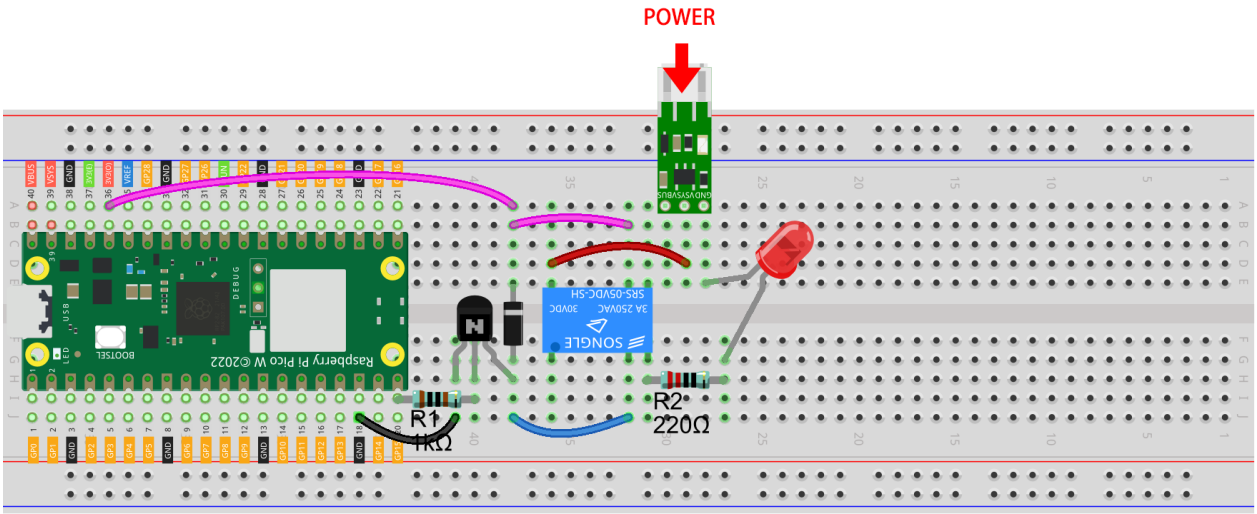
- **ダイオード**
- Flyback Diode - Wikipedia

この時点でプログラムは実行の準備ができており、実行後に「チクタク」という音が聞こえるでしょう。これはリレー内の接触器コイルが吸引して破断する音です。

次に、負荷回路の両端をそれぞれリレーのピン 3 とピン 6 に接続します。

..(前の記事で説明したブレッドボードの電源モジュールで駆動される簡単な回路を例に取ります。)





この時点で、リレーは負荷回路のオンとオフを制御できるようになります。

コード

注釈:

- kepler-kit-main/micropython のパスの下にある 2.16_control_another_circuit.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 キーを押して実行してください。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

relay = machine.Pin(15, machine.Pin.OUT)
while True:
    relay.value(1)
    utime.sleep(2)
    relay.value(0)
    utime.sleep(2)
```

コードを実行すると、リレーは制御される回路の動作状態を 2 秒ごとに切り替えます。リレー回路と負荷回路の対応関係をさらに明確にするため、手動で一方の行をコメントアウトすることができます。

詳細を学ぶ

リレーのピン 3 は通常開いており、接触器コイルが動作するとオンになります。ピン 4 は通常閉じており、接触器コイルが通電するとオンになります。ピン 1 はピン 6 に接続され、負荷回路の共通端子です。

負荷回路の一端をピン 3 からピン 4 に切り替えることで、まったく逆の動作状態を得ることができます。

3. 音と表示と動き

4.23 3.1 ビープ音

アクティブブザーは、LED を点灯させるのと同じくらい使いやすい典型的なデジタル出力デバイスです！

- **ブザー**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

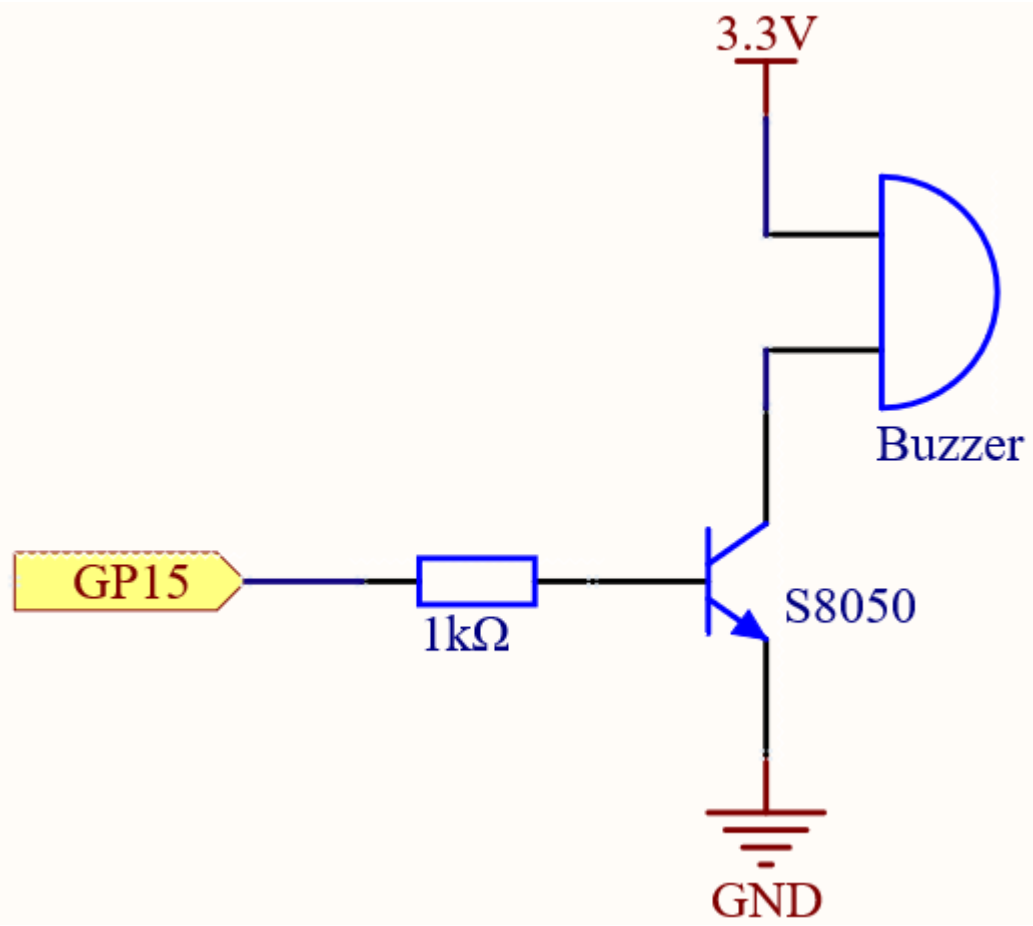
全てのキットをまとめて購入するのが便利です。リンクはこちら：

名前	キット内容	リンク
Kepler キット	450+ 点	

以下のリンクから個別にも購入できます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	アクティブ **ブザー**	1	

回路図



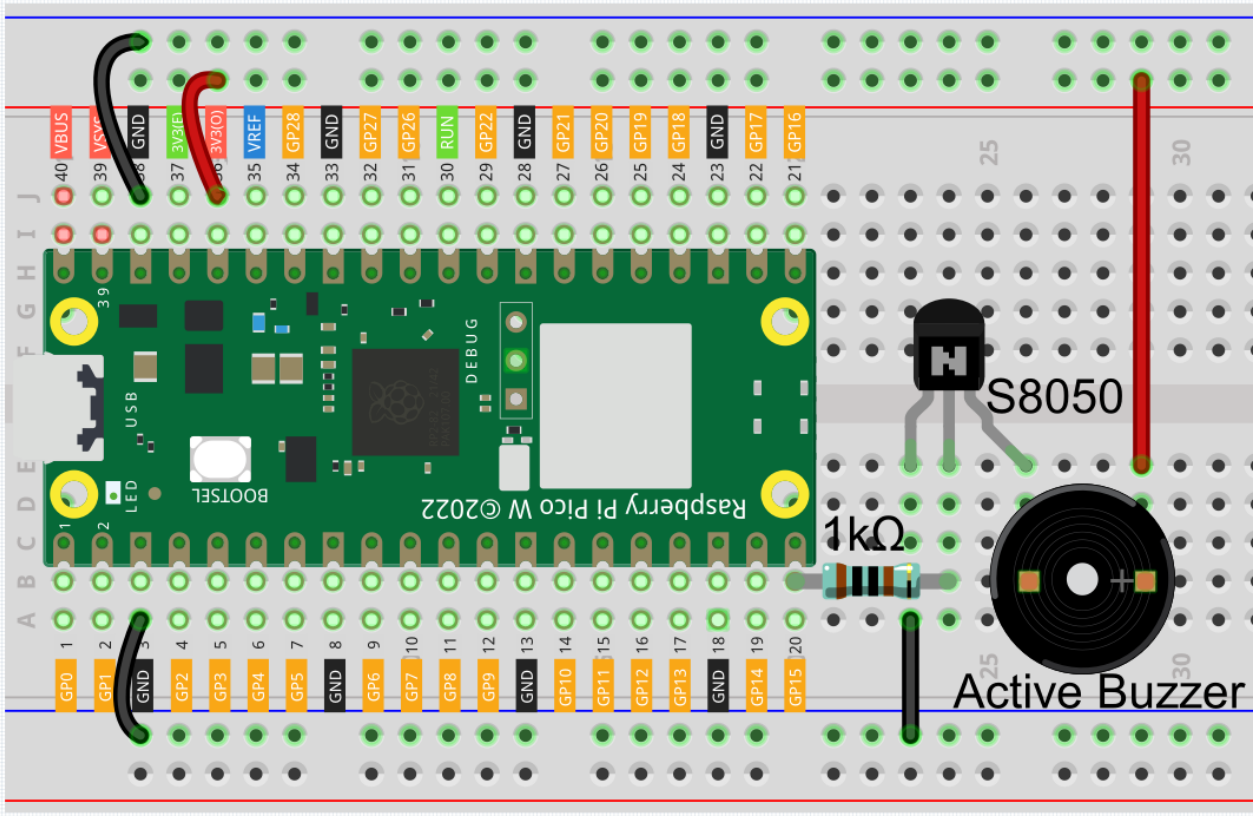
GP15 の出力が高い場合、1K の電流制限抵抗を経て（トランジスタを保護するため）、S8050（NPN トランジスタ）が導通し、ブザーが鳴ります。

S8050（NPN トランジスタ）の役割は、電流を増幅してブザーの音を大きくすることです。実際には、GP15 に直接ブザーを接続しても、ブザーの音は小さいことに気付くでしょう。

配線

キットには 2 種類のブザーが含まれています。アクティブブザーを使用する必要があります。裏返して、封じられた裏側（露出した PCB ではない方）が必要なものです。

ブザーは動作時にトランジスタが必要で、ここでは S8050（NPN トランジスタ）を使用しています。



コード

注釈:

- `kepler-kit-main/micropython` のパスの下で `3.1_beep.py` ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、F5 キーを押すだけで実行できます。
- 右下の"MicroPython (Raspberry Pi Pico)"インタプリタをクリックするのを忘れないでください。

- 詳しいチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

buzzer = machine.Pin(15, machine.Pin.OUT)
while True:
    for i in range(4):
        buzzer.value(1)
        utime.sleep(0.3)
        buzzer.value(0)
        utime.sleep(0.3)
    utime.sleep(1)
```

コードを実行すると、毎秒ビープ音が聞こえます。

4.24 3.2 カスタムトーン

前回のプロジェクトではアクティブブザーを使用しましたが、今回はパッシブブザーを使用します。

アクティブブザーと同様に、パッシブブザーも電磁誘導の現象を利用して動作します。違いは、パッシブブザーには振動源がないため、直流信号を使っても音を出さない点です。しかし、これによりパッシブブザーは自分自身の振動周波数を調整し、"ド、レ、ミ、ファ、ソ、ラ、シ"などの異なる音を出すことができます。

パッシブブザーでメロディを鳴らしましょう！

- **ブザー**

必要な部品

このプロジェクトで必要な部品は以下の通りです。

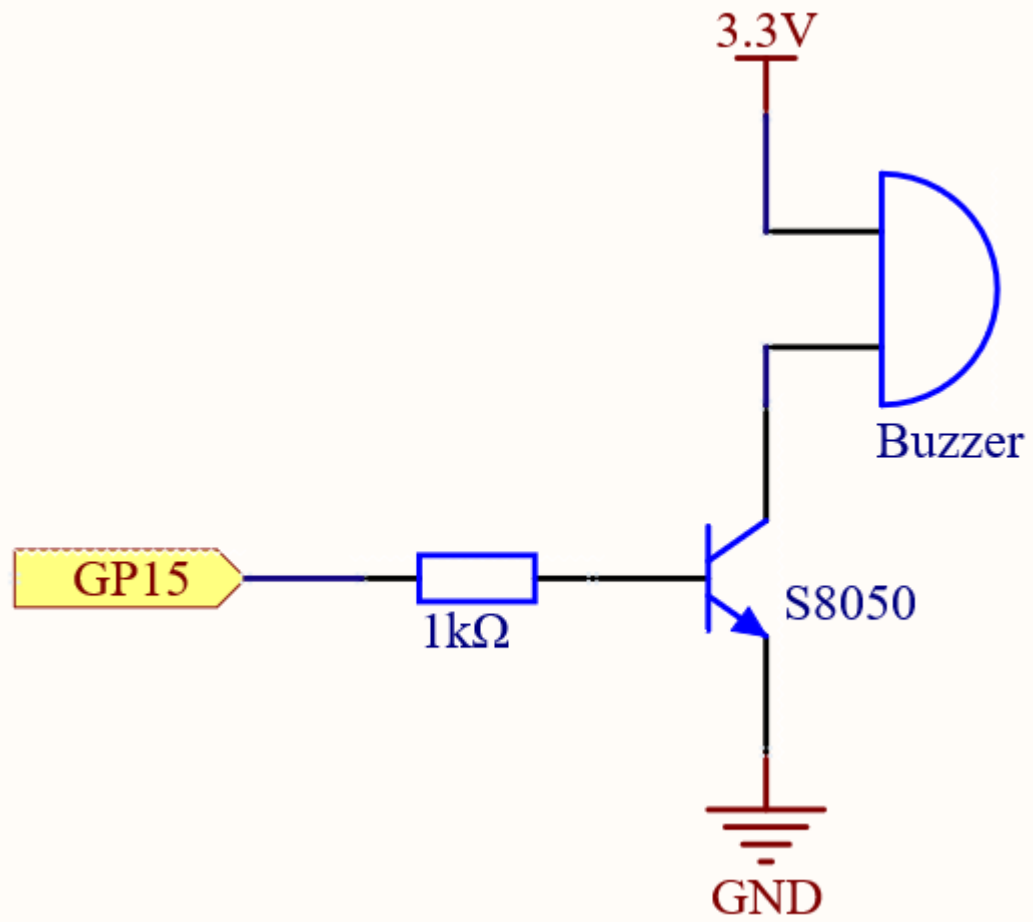
全体のキットを購入すると非常に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	パッシブ **ブザー**	1	

回路図



GP15 出力が高い場合、1K の電流制限抵抗を経て（トランジスタを保護するため）、S8050（NPN トランジスタ）

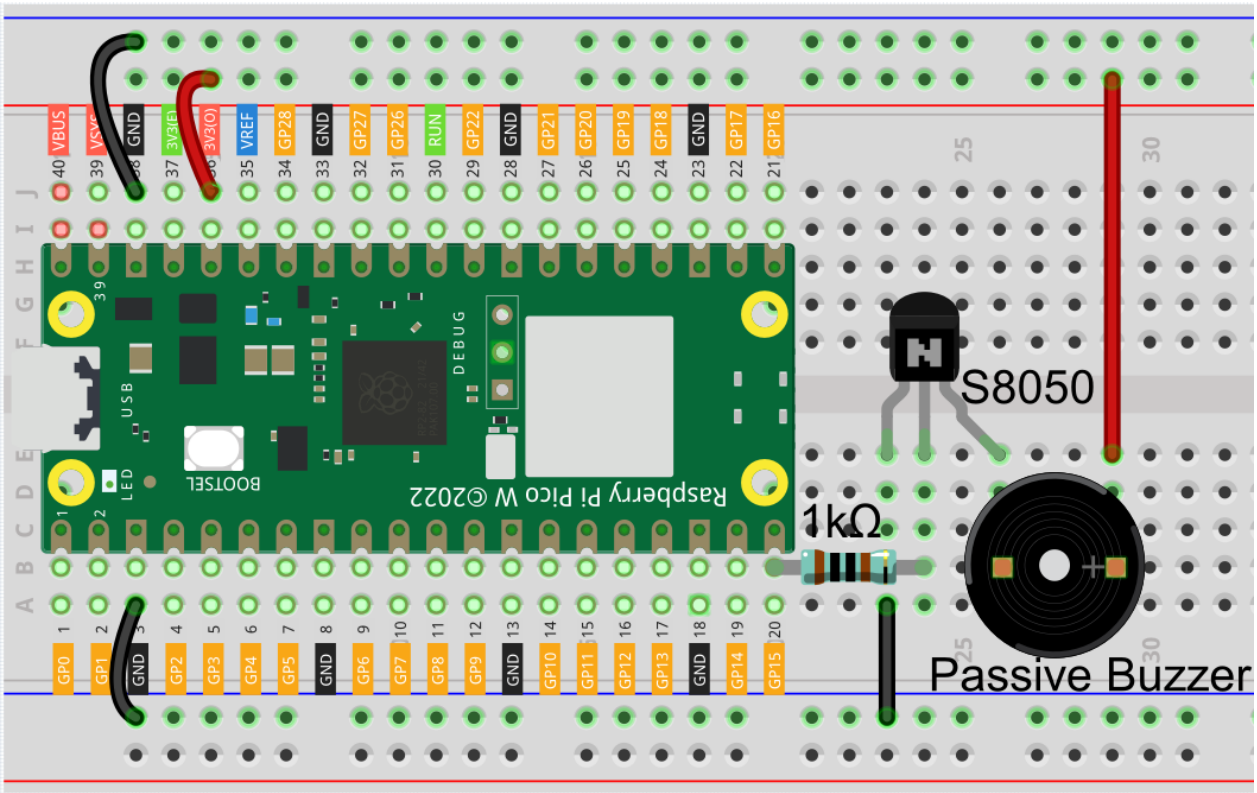
が導通し、ブザーが鳴ります。

S8050（NPN トランジスタ）の役割は、電流を増幅してブザーの音を大きくすることです。実際、GP15 に直接ブザーを接続することもできますが、ブザーの音が小さくなることに気づくでしょう。

配線

キットには 2 つのブザーが含まれていますが、ここではパッシブブザー（背面に露出した PCB があるもの）を使用します。

ブザーはトランジスタが必要なため、ここでは S8050 を使用します。



コード

注釈:

- kepler-kit-main/micropython のパス下にある 3.2_custom_tone.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、単に F5 キーを押して実行します。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

buzzer = machine.PWM(machine.Pin(15))

def tone(pin, frequency, duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

tone(buzzer, 440, 250)
utime.sleep_ms(500)
tone(buzzer, 494, 250)
utime.sleep_ms(500)
tone(buzzer, 523, 250)
```

動作原理

パッシブブザーにデジタル信号が与えられると、振動板を押し続けるだけで音は発生しません。

したがって、 tone() 関数を使用して、PWM 信号を生成し、パッシブブザーに音を出させます。

この関数には 3 つのパラメーターがあります：

- **pin** 、ブザーを制御する GPIO ピン。
- **frequency** 、ブザーの音程は周波数で決まり、周波数が高いほど音程も高くなります。
- **duration** 、音の持続時間。

duty_u16() 関数を使用して、デューティーサイクルを 30000（約 50%）に設定します。他の数値でも構いませんが、振動させるためには不連続な電気信号を生成する必要があります。

詳細

ピアノの基本周波数に従って特定の音をシミュレートし、完全な楽曲を演奏することができます。

- Piano key frequencies - Wikipedia

注釈:

- kepler-kit-main/micropython のパス下にある 3.2_custom_tone_2.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、単に F5 キーを押して実行します。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

NOTE_C4 = 262
NOTE_G3 = 196
NOTE_A3 = 220
NOTE_B3 = 247

melody = [NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, NOTE_B3, NOTE_C4]

buzzer = machine.PWM(machine.Pin(15))

def tone(pin, frequency, duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

for note in melody:
    tone(buzzer, note, 250)
    utime.sleep_ms(150)
```

4.25 3.3 RGB LED ストリップ

WS2812 は、制御回路と RGB チップが 5050 コンポーネントのパッケージ内に統合されたインテリジェントな LED 光源です。内部には、インテリジェントなデジタルポートデータラッチと信号再整形増幅駆動回路が含まれています。さらに、高精度な内部オシレーターとプログラム可能な定電流制御部も内蔵しており、各ピクセルの色の一貫性が高く確保されています。

データ転送プロトコルは、単一の NZR 通信モードを使用します。ピクセルが電源投入リセット後、DIN ポートはコントローラからデータを受信し、最初のピクセルは初期の 24 ビットデータを収集して内部データラッチに送ります。その後、内部信号再整形増幅回路によって整形された他のデータは、DO ポートを通じて次のカスケードピクセルに送られます。各ピクセルの送信後、信号は 24 ビット削減されます。ピクセルは自動再整形送信技術を採用しており、ピクセルのカスケード数は信号転送速度にのみ依存します。

- *WS2812 RGB 8 LED* **ストリップ**

必要な部品

このプロジェクトには、以下の部品が必要です。

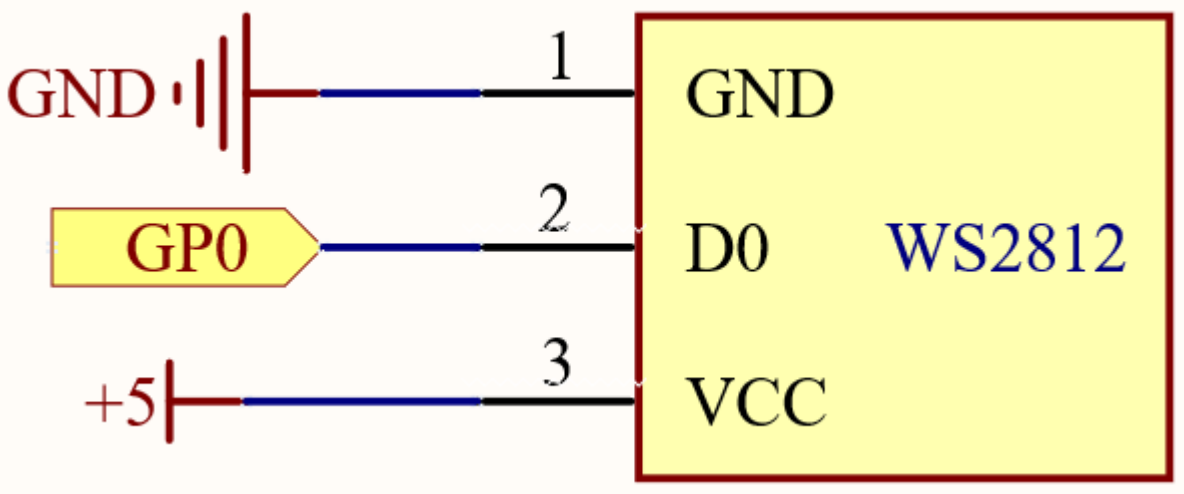
一式をまとめて購入するのが便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

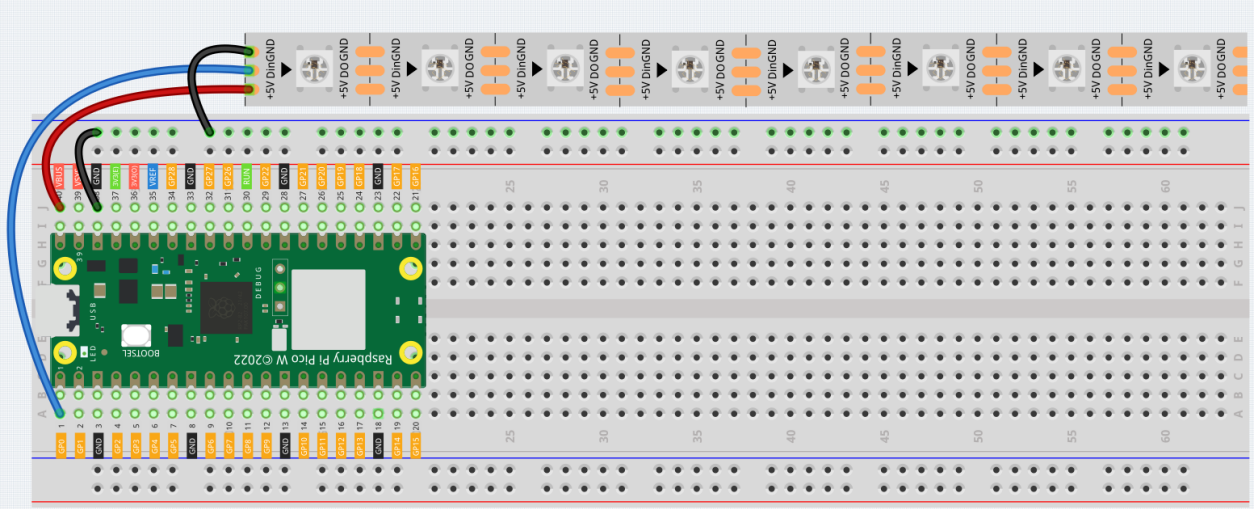
以下のリンクから個々の部品も購入できます。

項番	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*WS2812 RGB 8 LED* **ストリップ**	1	

回路図



配線



警告: 注意すべき点は電流です。

Pico W で任意の数の LED を使用することができますが、その VBUS ピンの電力は制限されています。ここでは、安全な 8 つの LED を使用します。ただし、より多くの LED を使用したい場合は、別途電源を追加する必要があります。

コード

注釈:

- `kepler-kit-main/micropython` のパスの下にある `3.3_rgb_led_strip.py` ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 キーを押して実行してください。

- 画面の右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。
- ここでは ws2812.py というライブラリを使用する必要があります。Pico W にアップロードされているかどうか確認してください。詳細なチュートリアルについては、 *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
import machine
from ws2812 import WS2812

ws = WS2812(machine.Pin(0), 8)

ws[0] = [64, 154, 227]
ws[1] = [128, 0, 128]
ws[2] = [50, 150, 50]
ws[3] = [255, 30, 30]
ws[4] = [0, 128, 255]
ws[5] = [99, 199, 0]
ws[6] = [128, 128, 128]
ws[7] = [255, 100, 0]
ws.write()
```

お気に入りの色をいくつか選んで、RGB LED ストリップに表示しましょう！

仕組み

ws2812 ライブラリでは、関連する関数を WS2812 クラスに統合しています。

次の文を使用して、RGB LED ストリップを操作できます。

```
from ws2812 import WS2812
```

WS2812 型のオブジェクトを「ws」という名前で宣言し、それが「pin」に接続されており、WS2812 ストリップには「number」個の RGB LED があります。

```
ws = WS2812(pin, number)
```

ws は配列オブジェクトであり、各要素は WS2812 ストリップ上の 1 つの RGB LED に対応しています。例えば、ws[0] は最初のもので、ws[7] は 8 番目です。

各 RGB LED に色値を割り当てることができます。これらの値は、24 ビットカラー（6 桁の 16 進数で表される）

または 3 つの 8 ビット RGB のリストでなければなりません。

例えば、赤の値は "0xFF0000" または "[255,0,0]" です。

```
ws[i] = color value
```

次に、この文を使用して LED ストリップの色を書き込み、点灯させます。

```
ws.write()
```

すべての LED を同じ色で点灯させるには、次の文を直接使用することもできます。

```
ws.write_all(color value)
```

さらに学ぶ

ランダムに色を生成し、カラフルな流れる光を作成することができます。

注釈:

- kepler-kit-main/micropython のパスの下にある 3.3_rgb_led_strip_2.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 キーを押して実行してください。
- 画面の右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
from ws2812 import WS2812
import utime
import urandom

ws = WS2812(machine.Pin(0),8)

def flowing_light():
    for i in range(7, 0, -1):
        ws[i] = ws[i - 1]
    ws[0] = int(urandom.uniform(0, 0xFFFFFF))
    ws.write()
    utime.sleep_ms(80)
```

(次のページに続く)

(前のページからの続き)

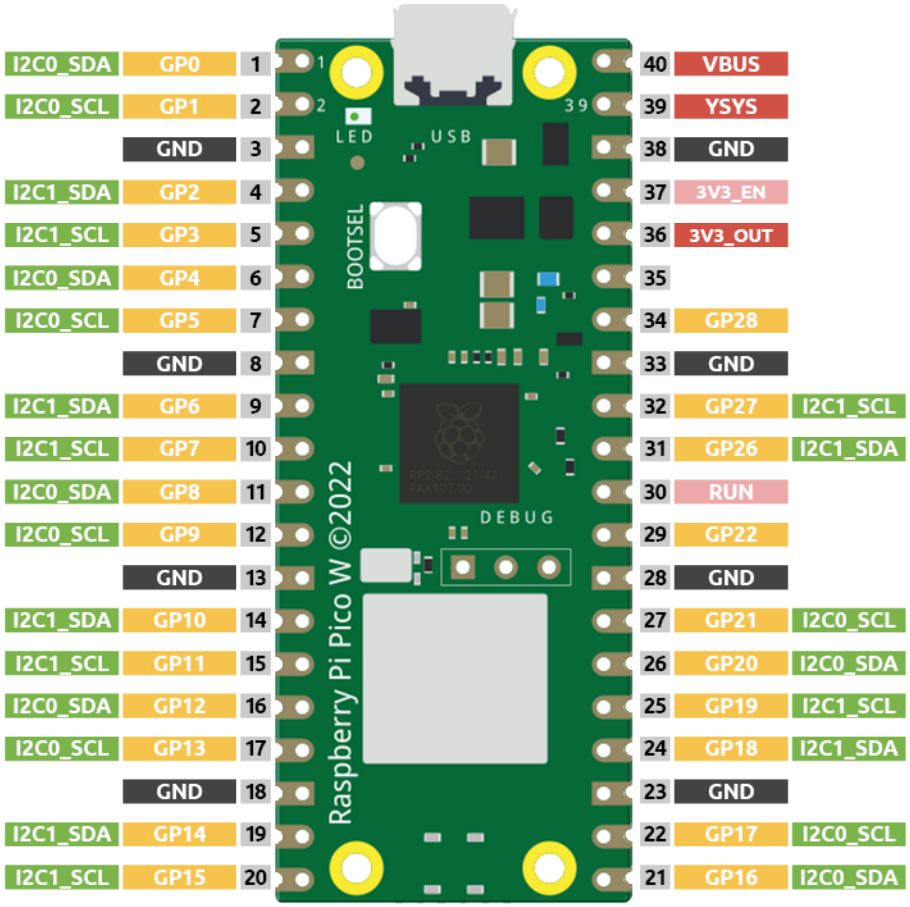
```
while True:
    flowing_light()
    print(ws[0])
```

4.26 3.4 液晶ディスプレイ

LCD1602 は、キャラクタ型の液晶ディスプレイで、同時に 32（16 × 2）文字を表示することができます。

ご存知のように、LCD やその他のディスプレイは人間とマシンの対話を大いに豊かにしていますが、一つの弱点があります。それは、コントローラに接続すると、多くの IO ポートが占有されてしまうことです。特に、外部ポートの少ないコントローラではこの問題は顕著です。そのため、この問題を解決するために I2C バスを備えた LCD1602 が開発されました。

- *I2C LCD1602*
- Inter-Integrated Circuit - Wikipedia



このセクションでは、I2C0 インターフェースを使用して LCD1602 を制御し、テキストを表示します。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

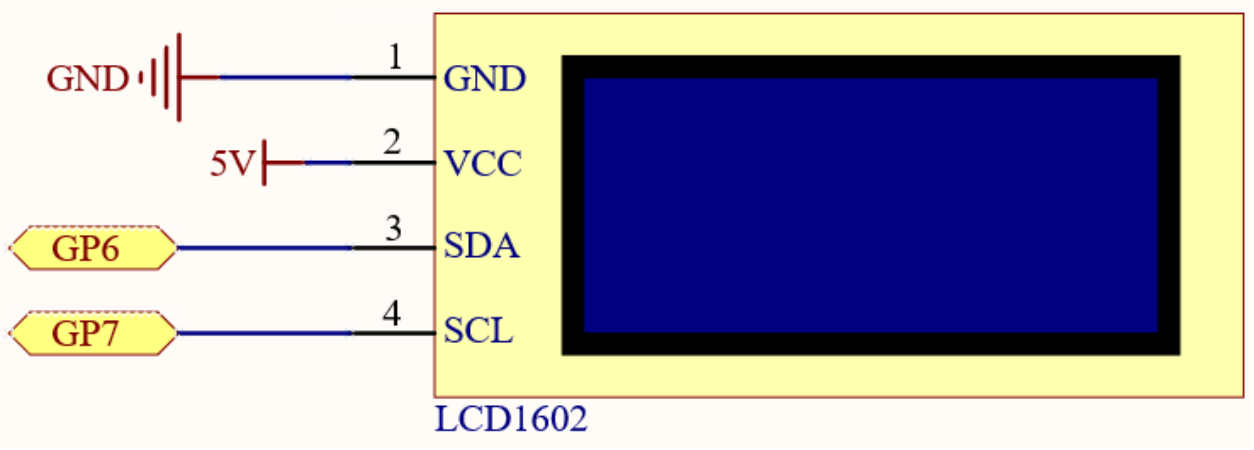
一式を購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

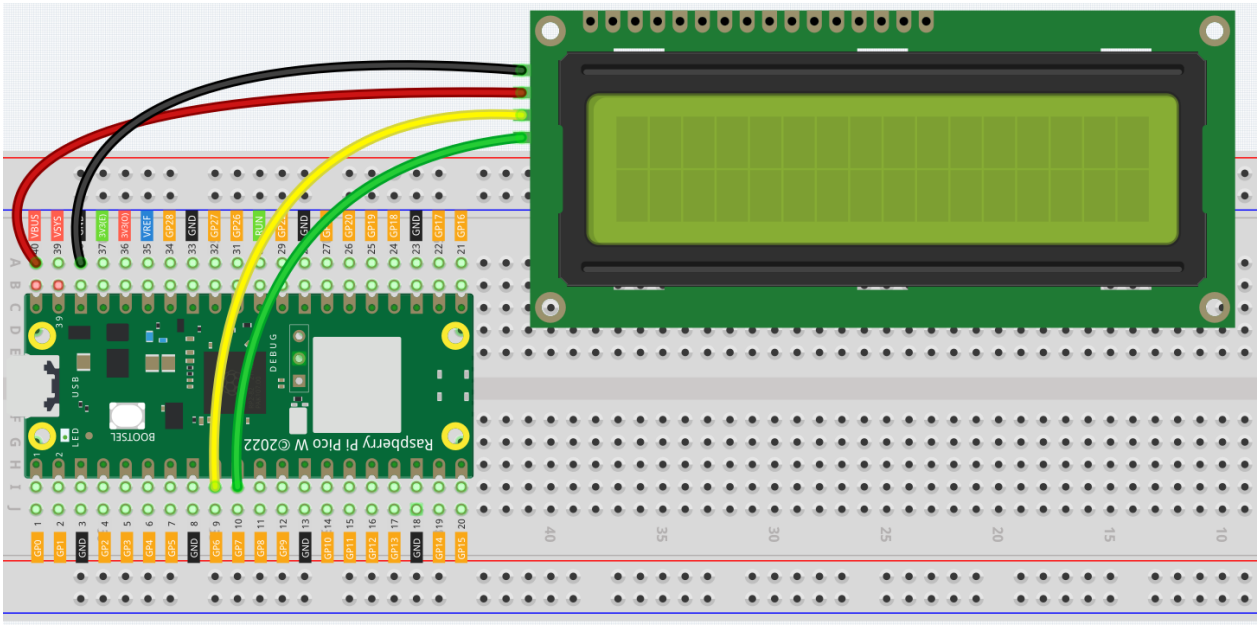
以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*I2C LCD1602*	1	

回路図



配線



コード

注釈:

- kepler-kit-main/micropython フォルダ内の 3.4_liquid_crystal_display.py ファイルを開くか、このコードを Thonny にコピーしてから「Run Current Script」をクリック、または単に F5 キーを押して実行してください。
- 右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは **コードを直接開いて実行する** を参照してください。
- ここでは lcd1602.py というライブラリが必要です。Pico W にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from lcd1602 import LCD
import utime

lcd = LCD()
string = " Hello!\n"
lcd.message(string)
utime.sleep(2)
string = "    Sunfounder!"
lcd.message(string)
utime.sleep(2)
lcd.clear()
```

プログラムを実行すると、LCD には順番に 2 行のテキストが表示され、その後消えます。

注釈: コードが実行されているときに画面が真っ白な場合、背面のポテンショメータを回してコントラストを調整できます。

動作原理は？

lcd1602 ライブラリでは、lcd1602 に関連する機能を LCD クラスに統合しています。

lcd1602 ライブラリをインポート

```
from lcd1602 import LCD
```

LCD クラスのオブジェクトを宣言し、それに lcd という名前を付けます。

```
lcd = LCD()
```

このステートメントは LCD にテキストを表示します。引数は文字列型でなければならない点に注意が必要です。整数や浮動小数点数を渡したい場合は、強制的に変換する `str()` を使用する必要があります。

```
lcd.message(string)
```

このステートメントを複数回呼び出すと、lcd はテキストを重ねて表示します。そのため、次のステートメントを使用して表示をクリアする必要があります。

```
lcd.clear()
```

4.27 3.5 スモールファン

今回は、TA6586 を用いて DC モーターを時計回りと反時計回りに回転させます。DC モーターは比較的大きな電流を必要とするため、安全上の理由からここでは電源モジュールを用いてモーターに電力を供給します。

- *DC* **モーター**
- *TA6586 -* **モータードライバーチップ**
- cpn_power_module

必要な部品

このプロジェクトでは、以下の部品が必要です。

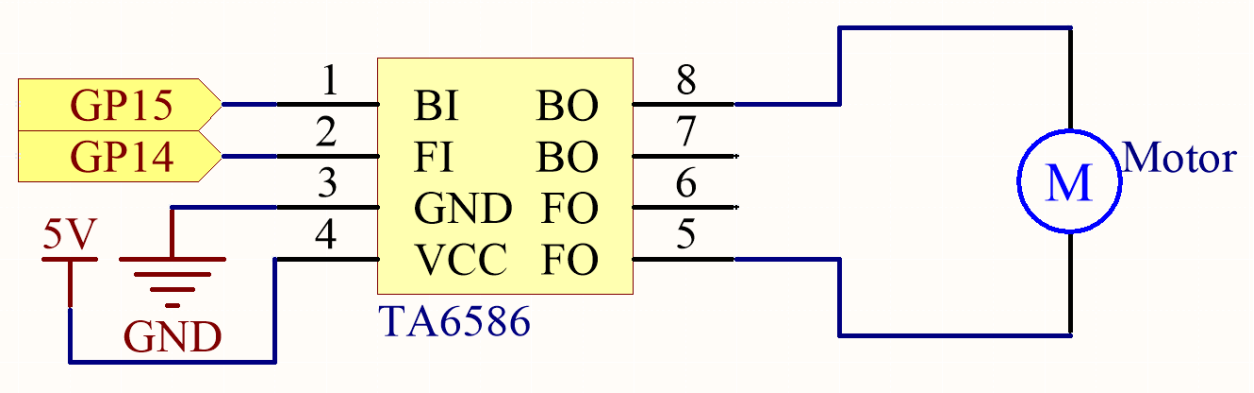
全ての部品が含まれるキットを購入するのは確かに便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから部品を個別に購入することもできます。

項番	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*TA6586* - **モータードライバーチップ**	1	
6	*DC* **モーター**	1	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	

回路図

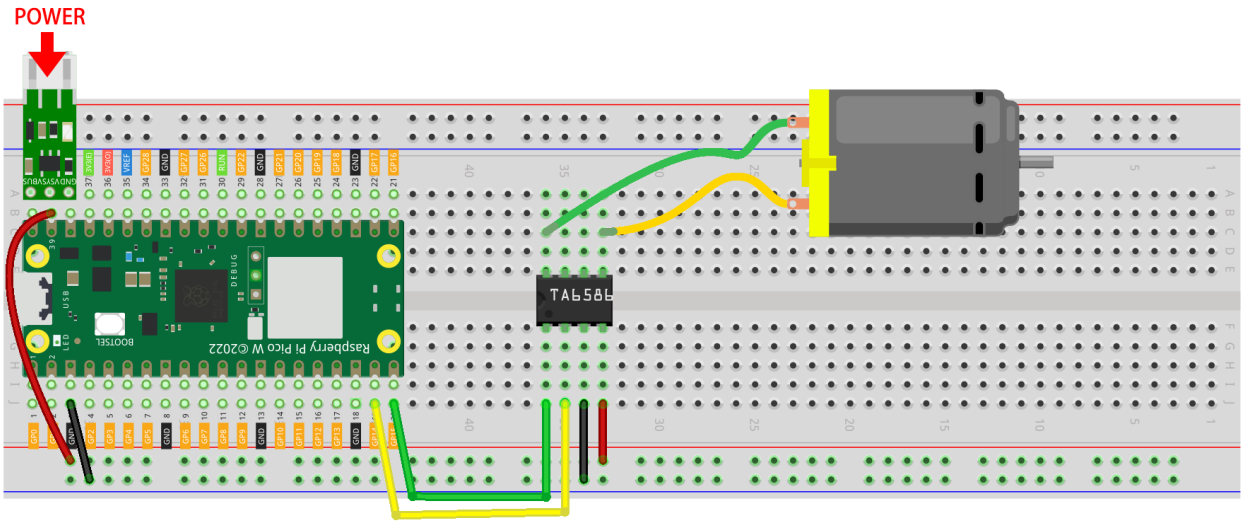


配線

注釈:

- DC モーターは高電流が必要なため、安全のためここでは Li-po チャージャーモジュールを用いてモーターに電力を供給します。

- ダイアグラムに示されているように Li-po チャージャーモジュールが接続されていることを確認してください。そうしないと、バッテリーと回路が短絡し、ダメージを受ける可能性があります。



コード

注釈:

- kepler-kit-main/micropython パス下の 3.5_small_fan.py ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある "MicroPython (Raspberry Pi Pico)" インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

motor1A = machine.Pin(14, machine.Pin.OUT)
motor2A = machine.Pin(15, machine.Pin.OUT)

def clockwise():
    motor1A.high()
    motor2A.low()

def anticlockwise():
```

(次のページに続く)

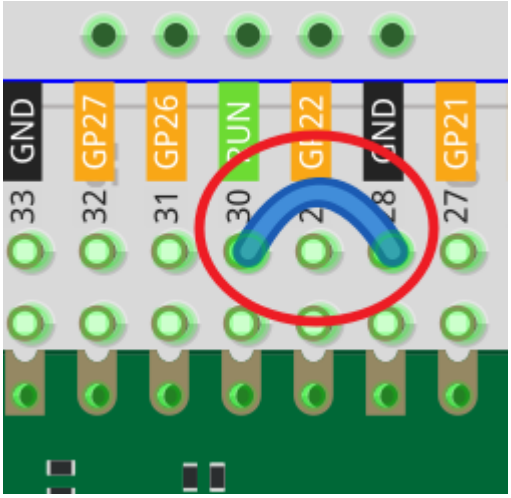
(前のページからの続き)

```
    motor1A.low()
    motor2A.high()

def stopMotor():
    motor1A.low()
    motor2A.low()

while True:
    clockwise()
    utime.sleep(1)
    stopMotor()
    utime.sleep(1)
    anticlockwise()
    utime.sleep(1)
    stopMotor()
    utime.sleep(1)
```

プログラムが動作すると、モーターは一定のパターンで前後に回転します。

注釈:

- ストップボタンをクリックした後もモーターが回転し続ける場合は、この時点で GND にワイヤーで Pico W の **RUN** ピンをリセットする必要があります。その後、このワイヤーを抜いて再度コードを実行してください。
- これはモーターが大量の電流を使用しているため、Pico W がコンピュータから切断される可能性があるためです。



4.28 3.6 ポンピング

小型の遠心ポンプは、自動植物水やりプロジェクトに適しています。また、ちょっとしたスマートな水の仕掛けを作るためにも使用できます。

その動力源は電動モーターで、通常のモーターと全く同じ方法で駆動されます。

- *DC* **ウォーターポンプ**
- *DC* **モーター**
- *TA6586 -* **モータードライバーチップ**
- cpn_power_module

注釈:

1. チューブをモーターの出口に接続し、ポンプを水に浸し、電源を投入します。
2. 水位が常にモーターよりも高いことを確認する必要があります。アイドリングは、発熱と騒音を発生させ、モーターを破損する可能性があります。
3. 植物に水をやる場合、土が吸い込まれるのを防ぐ必要があります。これはポンプを詰まらせる原因となる可能性があります。
4. チューブから水が出ない場合、チューブ内に残留水があり、エアフローを妨げている可能性があり、まずは排水が必要です。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

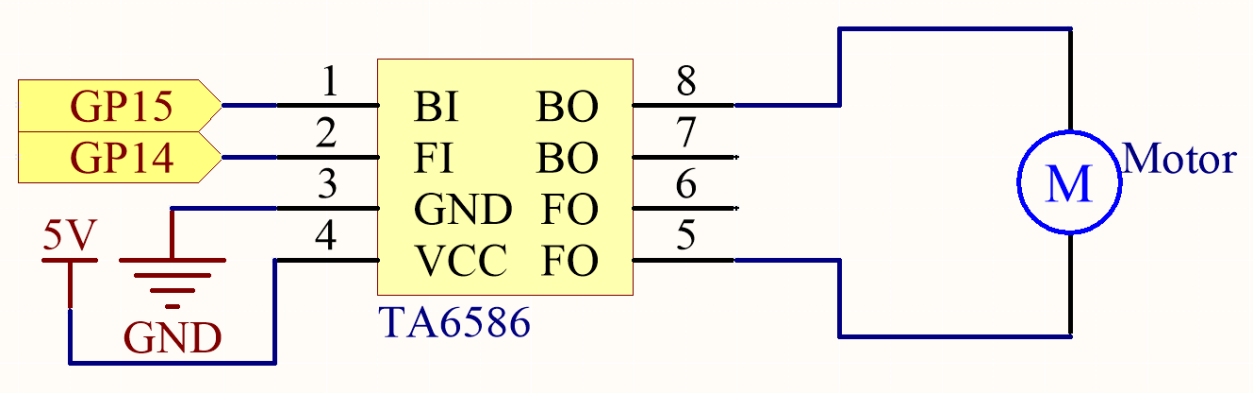
すべてを一つのキットで購入するのが便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入できます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*TA6586* - **モータードライバーチップ**	1	
6	*Li-po* **充電モジュール**	1	
7	18650 バッテリー	1	
8	バッテリーホルダー	1	
9	*DC* **ウォーターポンプ**	1	

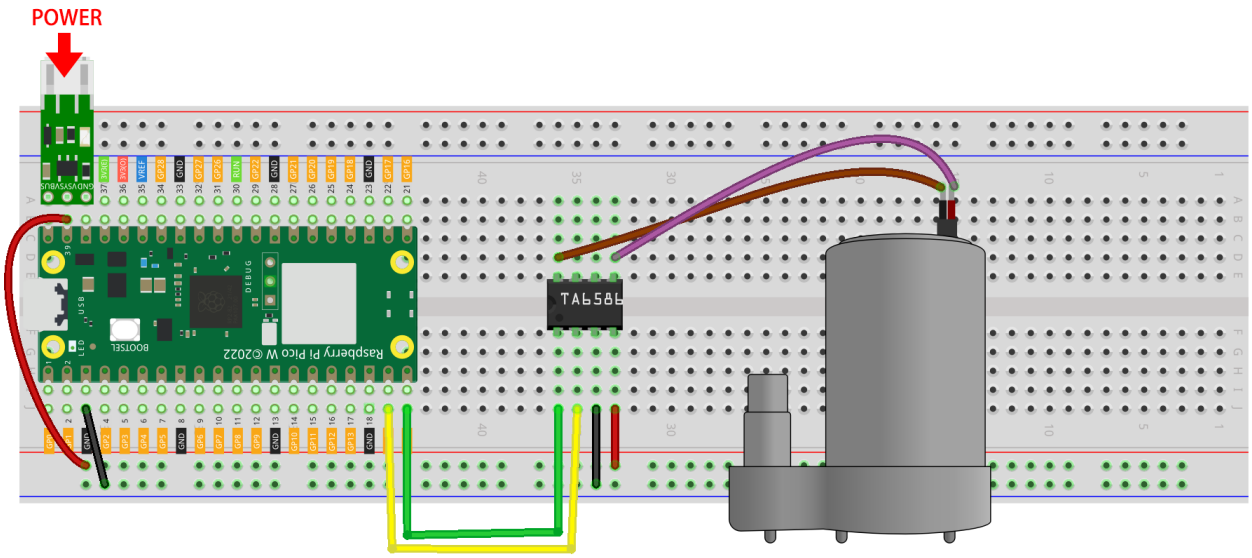
回路図



配線

注釈:

- ポンプは高電流を必要とするため、安全性を考慮してここでは Li-po チャージャーモジュールを使用してモーターに電力を供給します。
- Li-po チャージャーモジュールが図に示されているように接続されていることを確認してください。そうでないと、短絡が発生し、バッテリーや回路が損傷する可能性が高くなります。



コード

注釈:

- kepler-kit-main/micropython のパスの下で 3.6_pumping.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 を押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

motor1A = machine.Pin(14, machine.Pin.OUT)
motor2A = machine.Pin(15, machine.Pin.OUT)

while True:
    motor1A.high()
    motor2A.low()
```

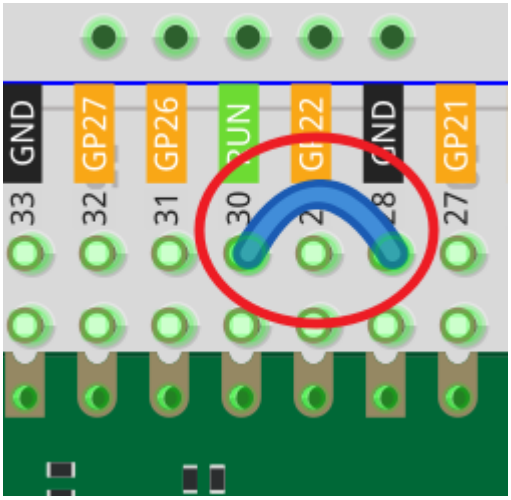
コードが実行された後、ポンプが動作を開始し、同時にチューブから水が流れ出るのを確認できます。

注釈:

- 「停止」ボタンをクリックした後もモーターが回転し続けている場合は、この時点で GND へのワイヤーで Pico W の **RUN** ピンをリセットする必要があります。その後、このワイヤーを抜いてコードを再度実行し

てください。

- これは、モーターが過度な電流で動作しているため、Pico W がコンピュータから切断される可能性があるためです。



4.29 3.7 サーボの振動

このキットには、LED やパッシブブザーに加えて、PWM 信号で制御されるデバイス、サーボも含まれています。

サーボは、位置（角度）制御が可能なデバイスであり、一定の角度変更が必要な制御システムに適しています。高級リモコン玩具、例えば飛行機、潜水艦モデル、リモコンロボットなどで広く利用されています。

それでは、サーボを振動させてみましょう！

- **サーボ**

必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

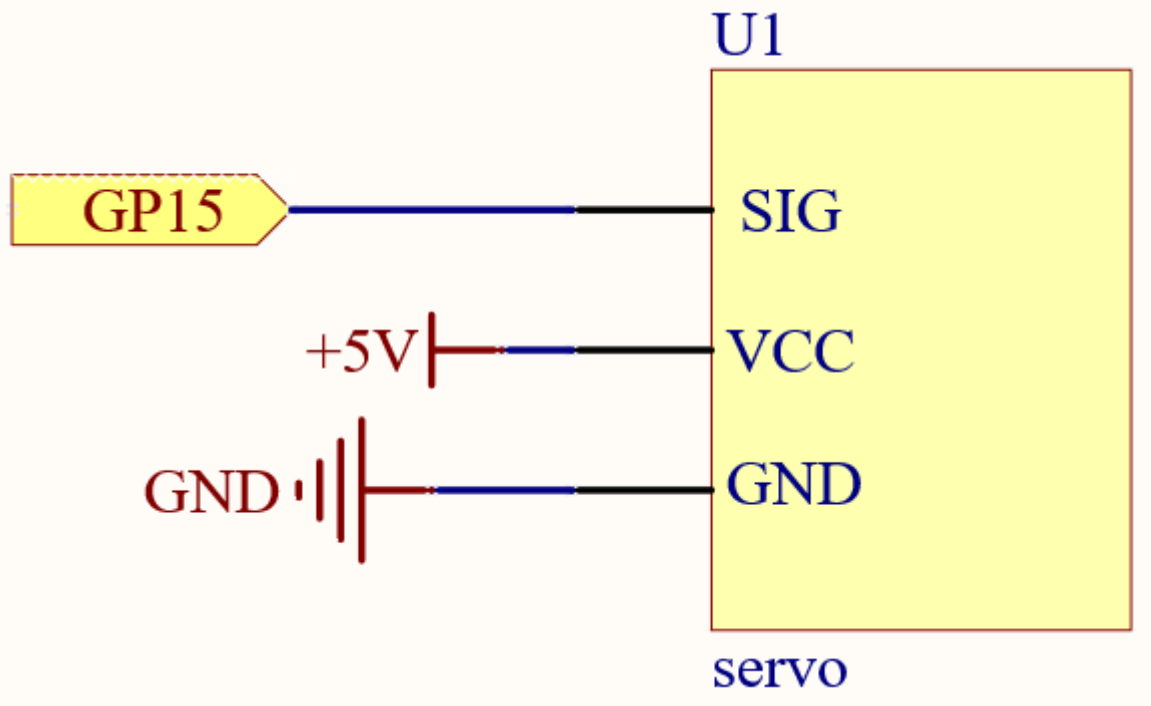
一式をまとめて購入するのは非常に便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

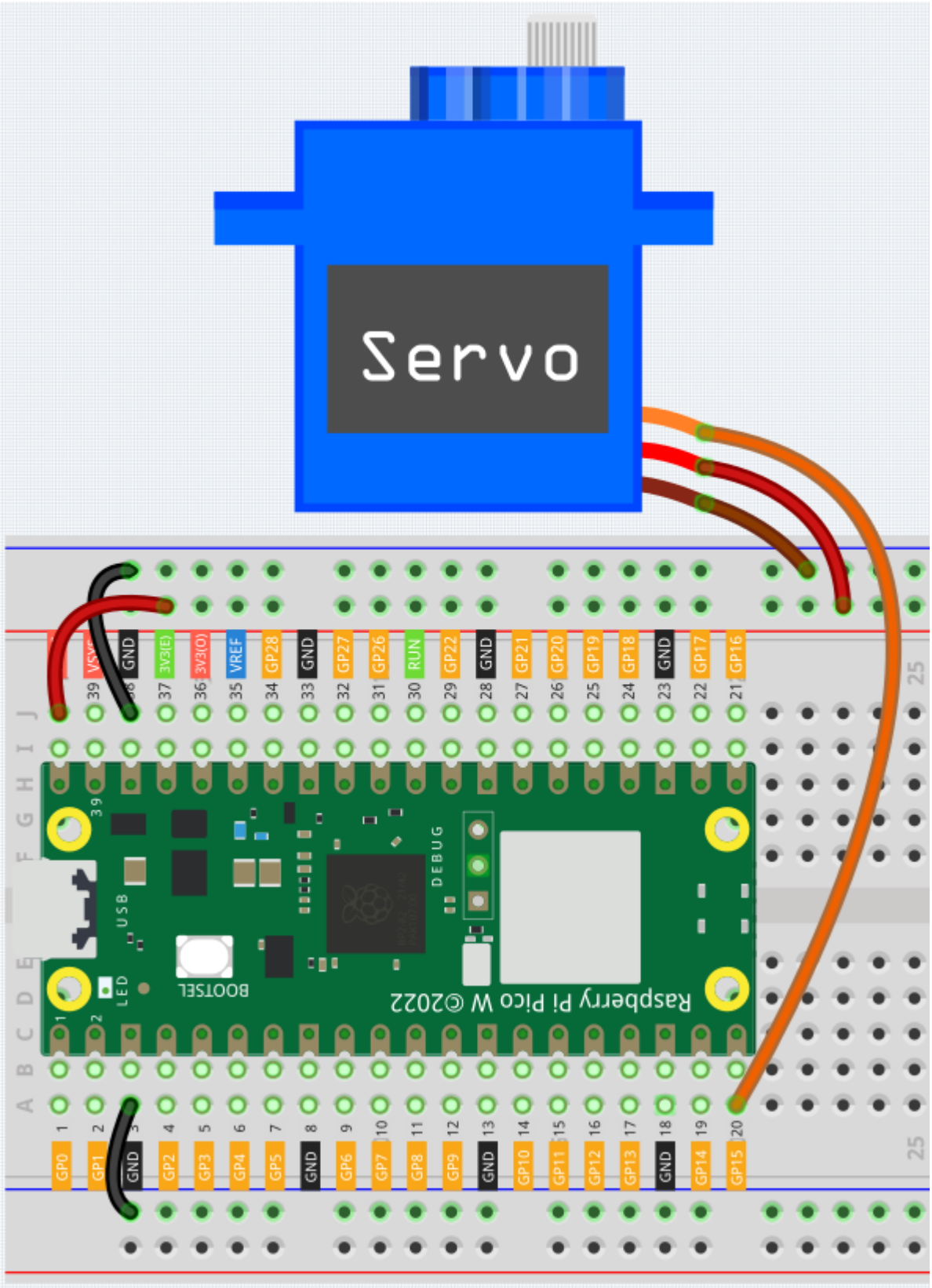
以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	

回路図



配線



- オレンジ色のワイヤーは信号で、GP15 に接続されています。

- 赤色のワイヤーは VCC で、VBUS(5V) に接続されています。
- 茶色のワイヤーは GND で、GND に接続されています。

コード

注釈:

- kepler-kit-main/micropython のパス下にある 3.7_swinging_servo.py ファイルを開くか、このコードを Thonny にコピペして、「Run Current Script」をクリックまたは F5 キーを押して実行します。
- 右下角にある「MicroPython（Raspberry Pi Pico）」インタープリターをクリックして選択してください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

servo = machine.PWM(machine.Pin(15))
servo.freq(50)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def servo_write(pin, angle):
    pulse_width = interval_mapping(angle, 0, 180, 0.5, 2.5)
    duty = int(interval_mapping(pulse_width, 0, 20, 0, 65535))
    pin.duty_u16(duty)

while True:
    for angle in range(180):
        servo_write(servo, angle)
        utime.sleep_ms(20)
    for angle in range(180, -1, -1):
        servo_write(servo, angle)
        utime.sleep_ms(20)
```

プログラムが実行中のとき、サーボアームが 0 ° から 180 ° まで前後に振動するのが見えます。

while True ループによってプログラムは絶えず動作していますので、プログラムを終了するには Stop ボタンを押す必要があります。

動作原理は？

サーボを動かすために servo_write() 関数を定義しました。

この関数には二つのパラメーターがあります：

- pin 、サーボを制御する GPIO ピン。
- Angle 、軸の出力角度。

この関数内で、 interval_mapping() が呼び出され、角度範囲 0~180 をパルス幅範囲 0.5~2.5ms にマッピングします。

```
pulse_width = interval_mapping(angle, 0, 180, 0.5, 2.5)
```

なぜ 0.5~2.5 なのか？これはサーボの動作モードによって決定されます。

- **サーボ**

次に、パルス幅を周期からデューティに変換します。 duty_u16() は小数点を持つことができない（値は浮動小数点型であってはならない）ので、 int() を用いてデューティを整数型に強制変換します。

```
duty = int(interval_mapping(pulse_width, 0, 20, 0, 65535))
```

最後に、デューティ値を duty_u16() に書き込みます。

4. コントローラ

4.30 4.1 ジョイスティックの切り替え

ビデオゲームをよくプレイするなら、ジョイスティックには非常に馴染みがあるでしょう。このデバイスは主にキャラクターの移動や画面の回転などに使用されます。

ジョイスティックがコンピュータに対して私たちのアクションを読み取る仕組みは非常にシンプルです。これは、直交する 2 つのポテンショメータから成り立っていると考えることができます。これらのポテンショメータは、ジョイスティックの垂直および水平なアナログ値を測定し、平面直角座標系での値（x,y）を出力します。

このキットのジョイスティックには、ジョイスティックが押されたときに活性化するデジタル入力もあります。

- **ジョイスティックモジュール**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

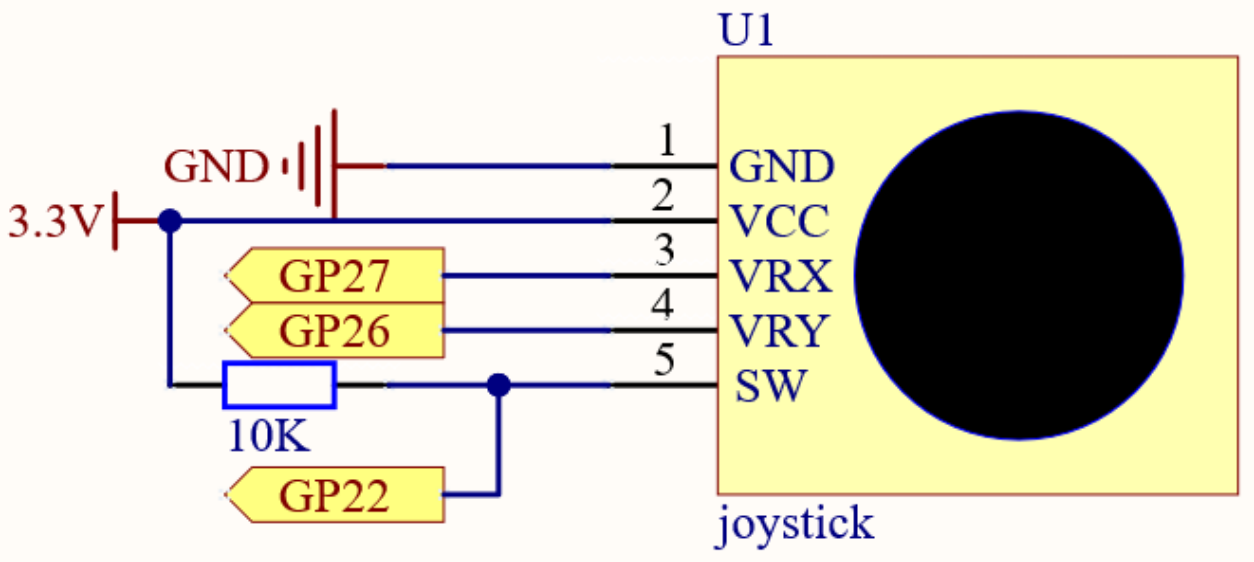
一式を購入するのは確かに便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

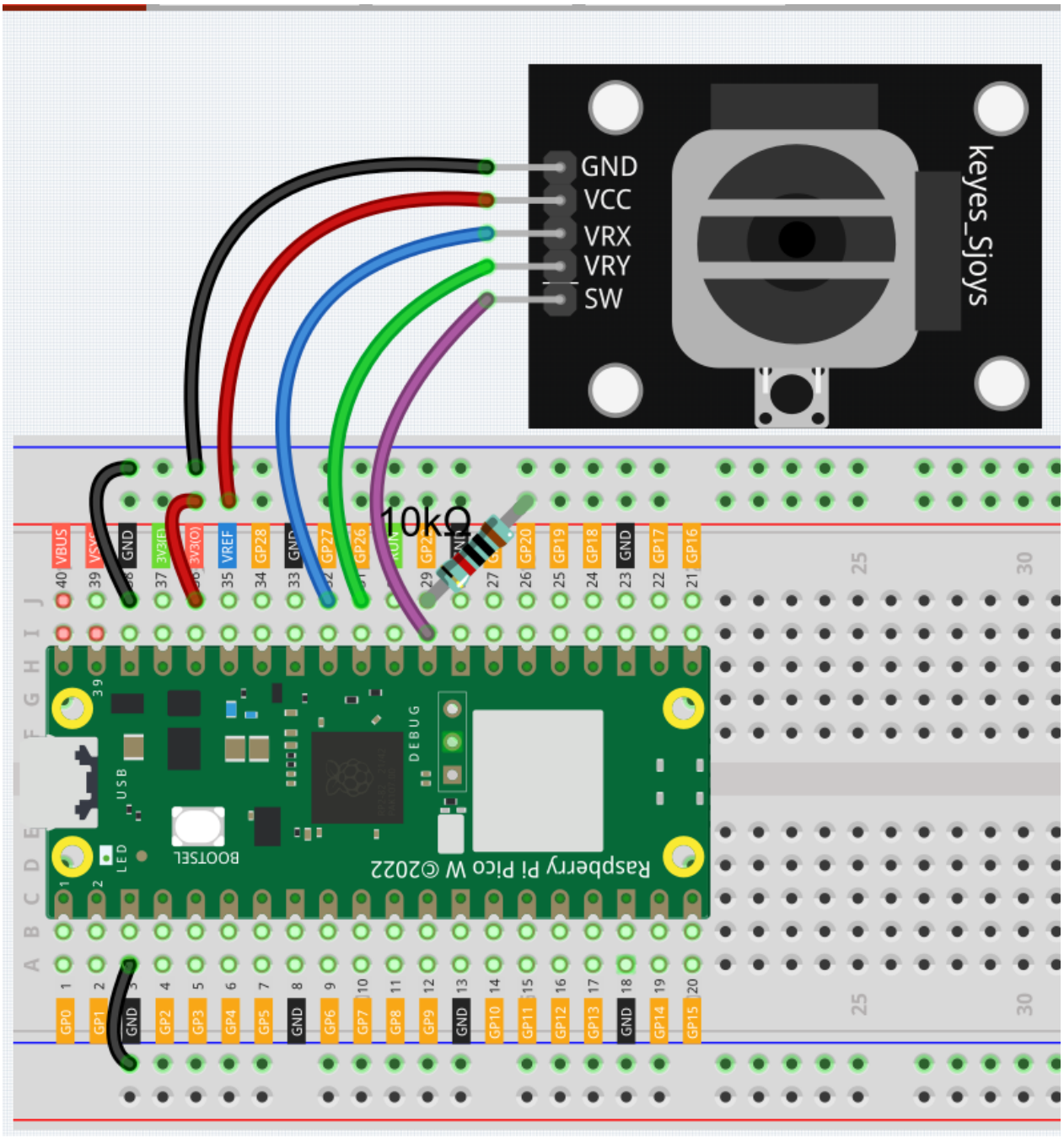
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**ジョイスティックモジュール**	1	

回路図



SW ピンは 10K のプルアップ抵抗に接続されています。その理由は、ジョイスティックが押されていないときに SW ピン（Z 軸）で安定した高レベルを得るためです。そうでないと、SW はサスペンド状態になり、出力値は 0/1 の間で変動する可能性があります。

配線



コード

注釈:

- `kepler-kit-main/micropython` のパスの下にある `4.1_toggle_the_joystick.py` ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。
- 右下隅にある「MicroPython（Raspberry Pi Pico）」のインタープリタをクリックするのを忘れないでくだ

さい。

- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

x_joystick = machine.ADC(27)
y_joystick = machine.ADC(26)
z_switch = machine.Pin(22,machine.Pin.IN)

while True:
    x_value = x_joystick.read_u16()
    y_value = y_joystick.read_u16()
    z_value = z_switch.value()
    print(x_value,y_value,z_value)
    utime.sleep_ms(200)
```

プログラムを実行した後、Shell はジョイスティックの x, y, z の値を出力します。

- x 軸と y 軸の値は、0 から 65535 までのアナログ値です。
- Z 軸は、状態が 1 または 0 のデジタル値です。

4.31 4.2 4x4 キーパッド

4x4 キーボード、別名マトリックスキーボードは、単一のパネルに 16 キーが配置されたマトリックスです。

このキーパッドは、主にデジタル入力が必要なデバイスに見られます。例えば、電卓、テレビのリモートコントロール、押しボタン式の電話、自動販売機、ATM、ダイヤル錠、デジタルドアロックなどです。

このプロジェクトでは、押されたキーを特定し、関連するキーの値を取得する方法を学びます。

- *4x4* **キーパッド**
- E.161 - Wikipedia

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

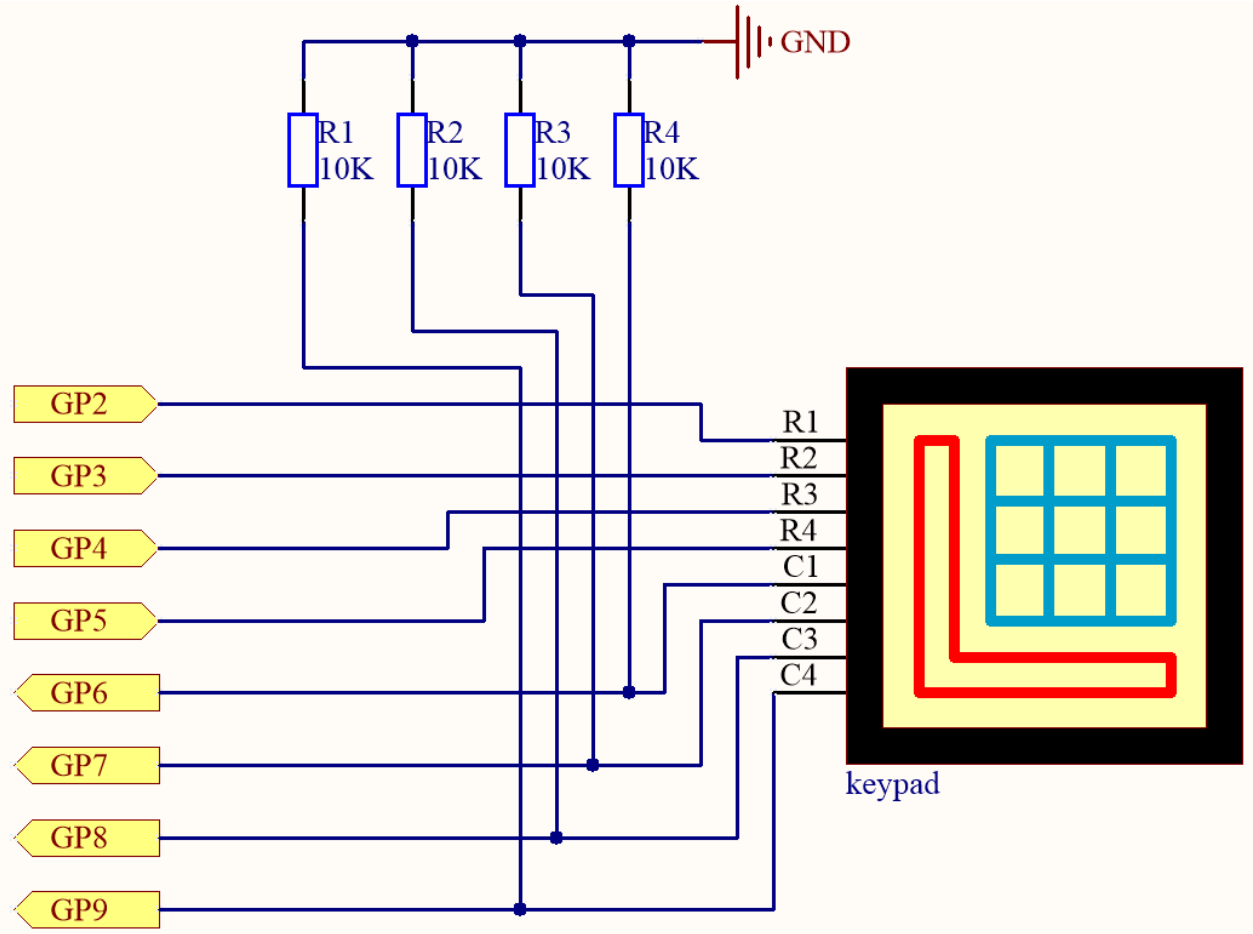
一式を購入するのは確かに便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(10K Ω)	
6	*4x4* **キーパッド**	1	

回路図

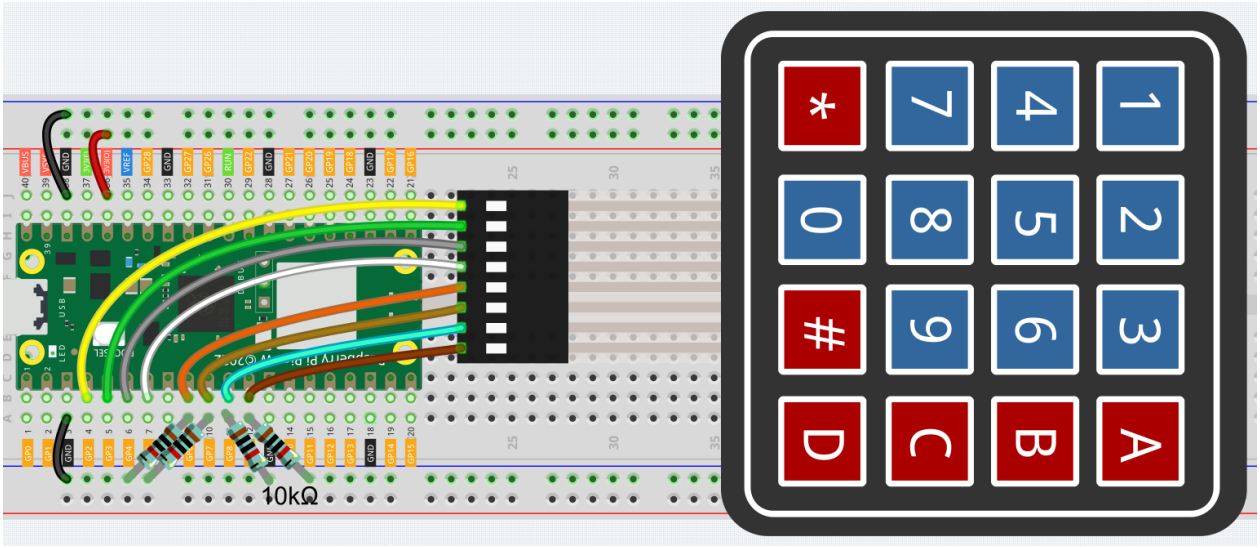


マトリックスキーボードの各列には 4 つのプルダウン抵抗が接続されており、キーが押されていない場合に G6〜G9 が安定した低レベルになるようにしています。

キーボードの行（G2〜G5）は高レベルに設定されています。もし G6〜G9 のいずれかが高レベルで読み取られた場合、どのキーが押されたかがわかります。

例えば、G6 が高レベルで読み取られた場合、数字のキー 1 が押されています。これは、数字のキー 1 の制御ピンが G2 と G6 であり、数字のキー 1 が押されたときに G2 と G6 が接続され、G6 も高レベルになるからです。

配線



配線を容易にするため、上記の図では、マトリックスキーボードの列と 10K 抵抗が同時に G6〜G9 の位置にある穴に挿入されています。

コード

注釈:

- kepler-kit-main/micropython のパスにある 4.2_4x4_keypad.py ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。
- 右下隅にある「MicroPython（Raspberry Pi Pico）」のインタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

characters = [["1","2","3","A"],["4","5","6","B"],["7","8","9","C"],["*","0","#","D"]]

pin = [2,3,4,5]
row = []
for i in range(4):
    row.append(None)
    row[i] = machine.Pin(pin[i], machine.Pin.OUT)

pin = [6,7,8,9]
```

(次のページに続く)

(前のページからの続き)

```
col = []
for i in range(4):
    col.append(None)
    col[i] = machine.Pin(pin[i], machine.Pin.IN)

def readKey():
    key = []
    for i in range(4):
        row[i].high()
        for j in range(4):
            if(col[j].value() == 1):
                key.append(characters[i][j])
        row[i].low()
    if key == [] :
        return None
    else:
        return key

last_key = None
while True:
    current_key = readKey()
    if current_key == last_key:
        continue
    last_key = current_key
    if current_key != None:
        print(current_key)
    time.sleep(0.1)
```

プログラムを実行すると、Shell がキーパッドで押したキーを出力します。

仕組み

```
import machine
import time

characters = [["1","2","3","A"],["4","5","6","B"],["7","8","9","C"],["*","0","#","D"]]

pin = [2,3,4,5]
row = []
for i in range(4):
```

(次のページに続く)

(前のページからの続き)

```
    row.append(None)
    row[i] = machine.Pin(pin[i], machine.Pin.OUT)

pin = [6,7,8,9]
col = []
for i in range(4):
    col.append(None)
    col[i] = machine.Pin(pin[i], machine.Pin.IN)
```

マトリックスキーボードの各キーを配列 characters[] に宣言し、各行と列のピンを定義します。

```
last_key = None
while True:
    current_key = readKey()
    if current_key == last_key:
        continue
    last_key = current_key
    if current_key != None:
        print(current_key)
    time.sleep(0.1)
```

これは、ボタンの値を読み取り、出力するメイン関数の一部です。

関数 readKey() は、各ボタンの状態を読み取ります。

if current_key != None および if current_key == last_key の文は、キーが押されているかどうかと、押されたボタンの状態を判断するために使用されます。(例えば、'1' を押しているときに'3' を押すと、判断が成立します。)

条件が成立すると、現在押されているキーの値を出力します。

last_key = current_key の文は、各判断の状態を配列 last_key に割り当て、次の条件判断に備えます。

```
def readKey():
    key = []
    for i in range(4):
        row[i].high()
        for j in range(4):
            if(col[j].value() == 1):
                key.append(characters[i][j])
        row[i].low()
```

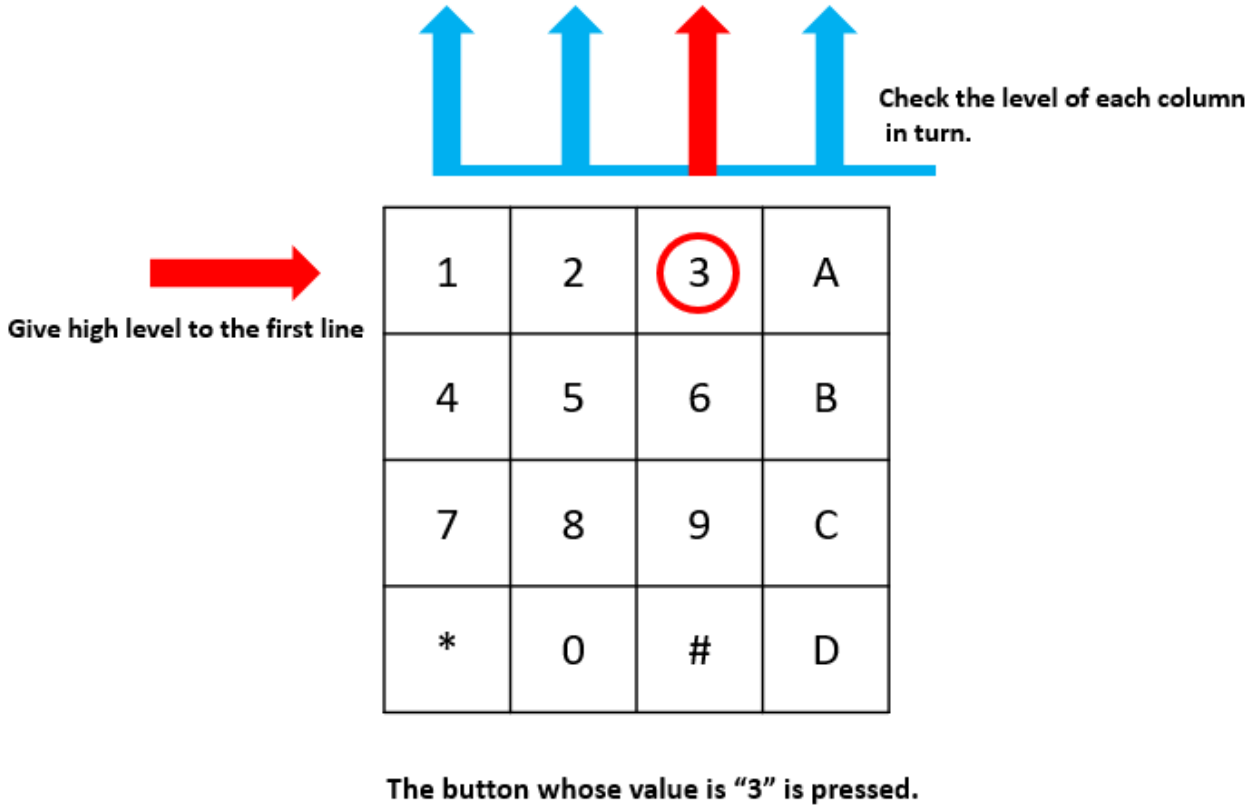
(次のページに続く)

(前のページからの続き)

```
    if key == [] :
        return None
    else:
        return key
```

この関数は、各行に順番に高レベルを割り当てます。ボタンが押されると、キーの位置する列が高レベルになります。二層のループの判断後、状態が 1 のボタンの値が配列 key に格納されます。

キー'3' を押す場合：



row[0] が高レベルに書き込まれ、 col[2] が高レベルになります。

col[0] 、 col[1] 、 col[3] は低レベルになります。

四つの状態があります：0、0、1、0；そして、'3' を pressed_keys に書き込みます。

row[1] 、 row[2] 、 row[3] が高レベルに書き込まれると、col[0] ~ col[4] は低レベルになります。

ループが停止し、key = '3' が返されます。

ボタン'1' と'3' を押すと、key = ['1','3'] が返されます。

4.32 4.3 電極キーボード

プロジェクトに多数のタッチスイッチを追加したい場合、MPR121 は良い選択です。このモジュールには導体で拡張できる電極があります。例えば、電極をバナナに接続すると、そのバナナをタッチスイッチに変えることができます。

- *MPR121* **モジュール**

必要な部品

このプロジェクトには、以下の部品が必要です。

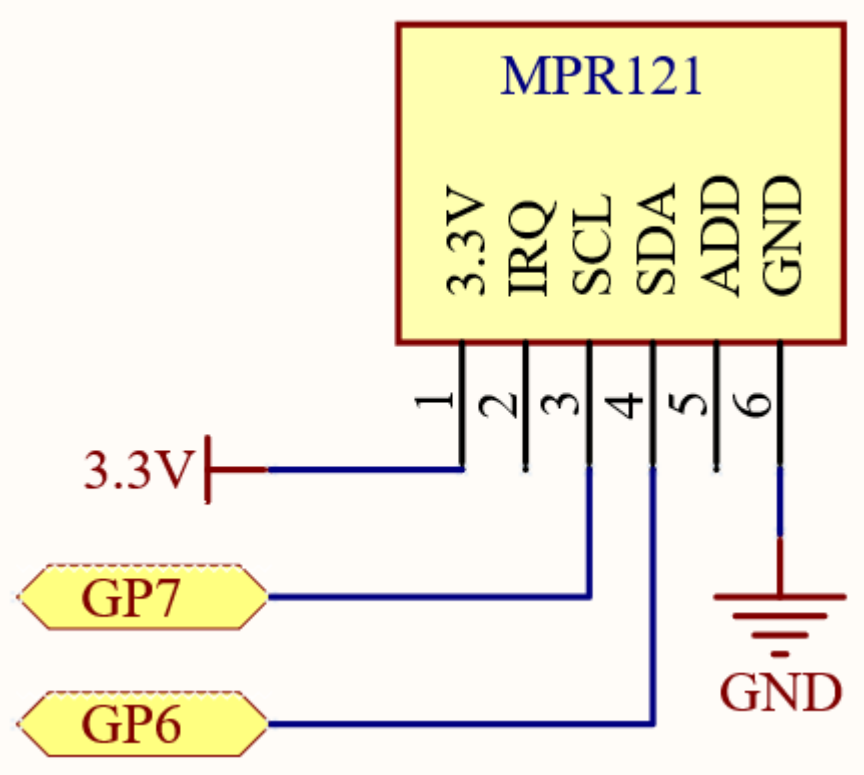
全てを一つのキットで購入するのも便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

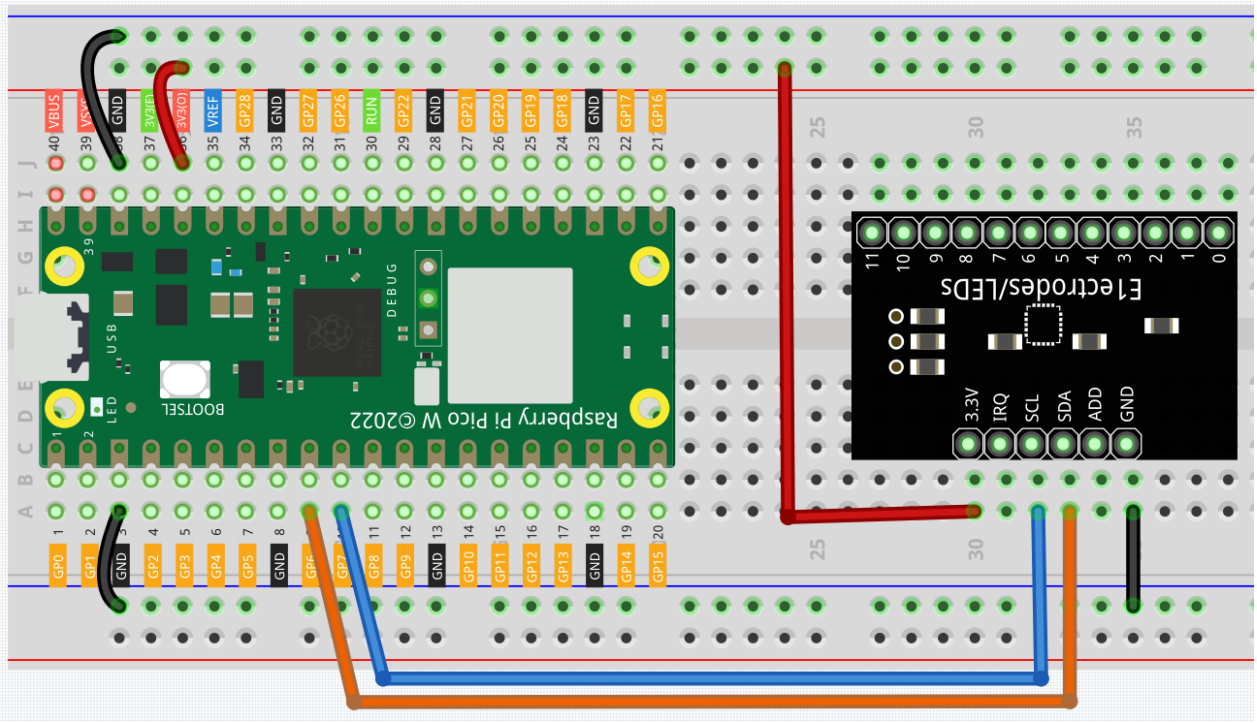
もちろん、以下のリンクから個々の部品を購入することもできます。

項番	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MPR121* **モジュール**	1	

回路図


MPR121
3.3V
IRQ
SCL
SDA
ADD
GND
1
2
3
4
5
6
3.3V
GP7
GP6
GND

配線


Raspberry Pi Pico W ©2022
BOOTSEL
DEBUG
USB
LED
Electrodes/LEDs
3.3V
IRQ
SCL
SDA
ADD
GND

コード

注釈:

- `kepler-kit-main/micropython` パス下の `4.3_electrode_keyboard.py` ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある "MicroPython (Raspberry Pi Pico)" インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- このプロジェクトでは `mpr121.py` というライブラリが必要です。Pico W にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from mpr121 import MPR121
from machine import Pin, I2C
import time

i2c = I2C(1, sda=Pin(6), scl=Pin(7))
mpr = MPR121(i2c)

# すべてのキーを確認
while True:
    value = mpr.get_all_states()
    if len(value) != 0:
        print(value)
    time.sleep_ms(100)
```

プログラムが動作すると、MPR121 の 12 個の電極に手を触れると、触れた電極が表示されます。

電極を他の導体、例えばフルーツやワイヤー、箔などに拡張して接続することで、これらの電極をトリガーするさまざまな方法が増えます。

仕組みは？

mpr121 ライブラリには、 `MPR121` クラスに機能が統合されています。

```
from mpr121 import MPR121
```

MPR121 は I2C モジュールであり、 `MPR121` オブジェクトを初期化するために I2C ピンのセットを定義する必要があります。この時点で、モジュールの電極の状態が初期値として記録されます。電極が拡張されている場合、初期値をリセットするために例を再実行する必要があります。

```
from machine import Pin, I2C
i2c = I2C(1, sda=Pin(6), scl=Pin(7))
mpr = MPR121(i2c)
```

- Inter-Integrated Circuit - Wikipedia

その後、 `mpr.get_all_states()` を使用して電極がトリガーされたかどうかを読み取ります。もし電極 2 と 3 がトリガーされた場合、値 `[2, 3]` が生成されます。

```
while True:
    value = mpr.get_all_states()
    if len(value) != 0:
        print(value)
    time.sleep_ms(100)
```

特定の電極を検出するために `mpr.is_touched(electrode)` も使用できます。トリガーされた場合、 `True` を返し、そうでない場合は `False` を返します。

```
while True:
    value = mpr.is_touched(0)
    print(value)
    time.sleep_ms(100)
```

5. マイクロチップ

4.33 5.1 マイクロチップ - 74HC595

集積回路（Integrated Circuit、IC）は、電子回路で"IC"として表されるミニチュア電子デバイスまたはコンポーネントの一種です。

特定のプロセスを用いて、トランジスタ、抵抗器、コンデンサ、インダクタなど、回路に必要なコンポーネントと配線を小型またはいくつかの小型の半導体ウェハーや誘電体基板上に作成し、それをパッケージに収めることで、必要な回路機能を持つマイクロ構造になります。全てのコンポーネントは一体となって構造化され、電子部品は微小化、低消費電力、高信頼性、および知能化に大きく前進しています。

集積回路の発明者は、ジャック・キルビー（ゲルマニウム（Ge）を基にした集積回路）とロバート・ノートン・ノイス（シリコン（Si）を基にした集積回路）です。

このキットには、GPIO ピンの使用を大幅に節約できる IC、74HC595 が装備されています。具体的には、8 ビットの 2 進数を書き込むことで、デジタル信号出力のための 8 ピンを置き換えることができます。

- 2 進数 - ウィキペディア

- *74HC595*

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

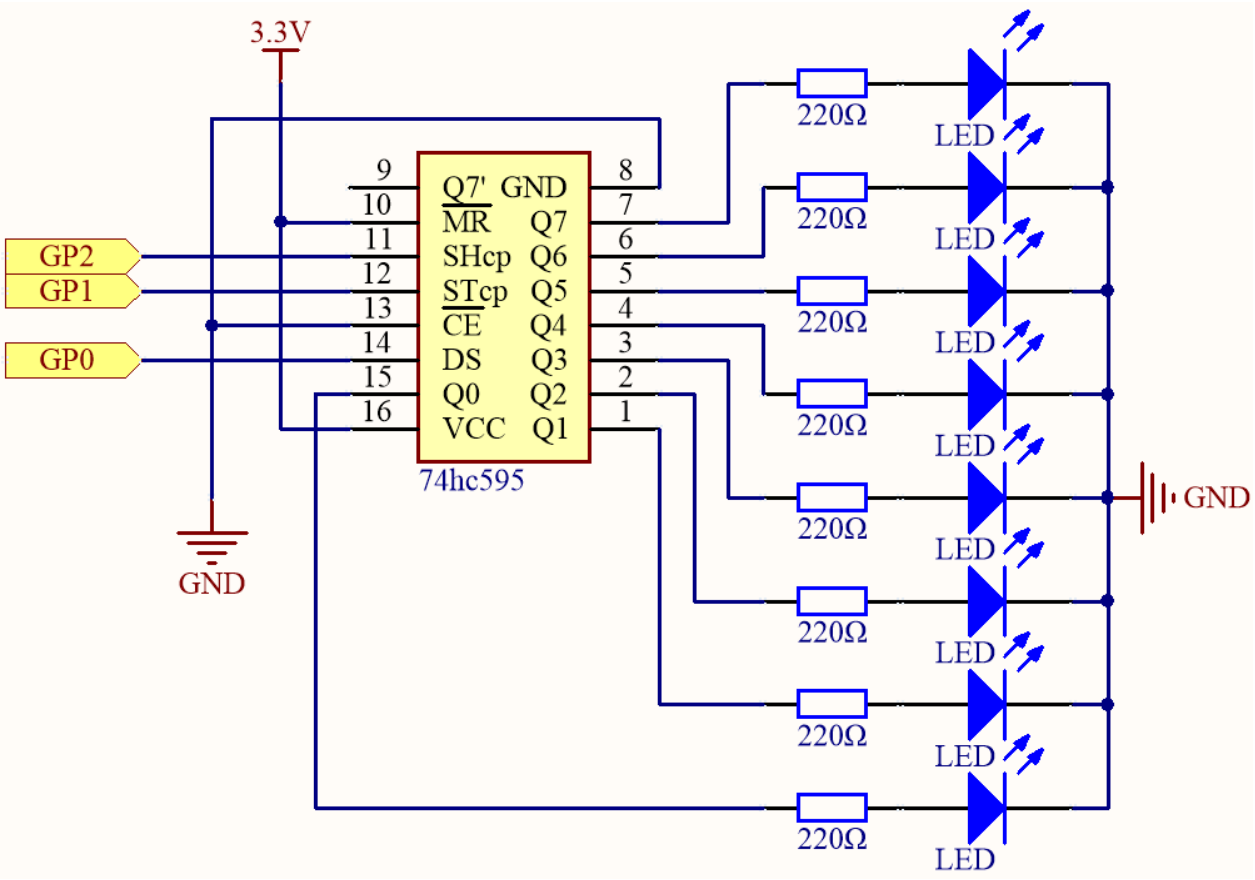
全体のキットを購入するのが確実に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入できます。

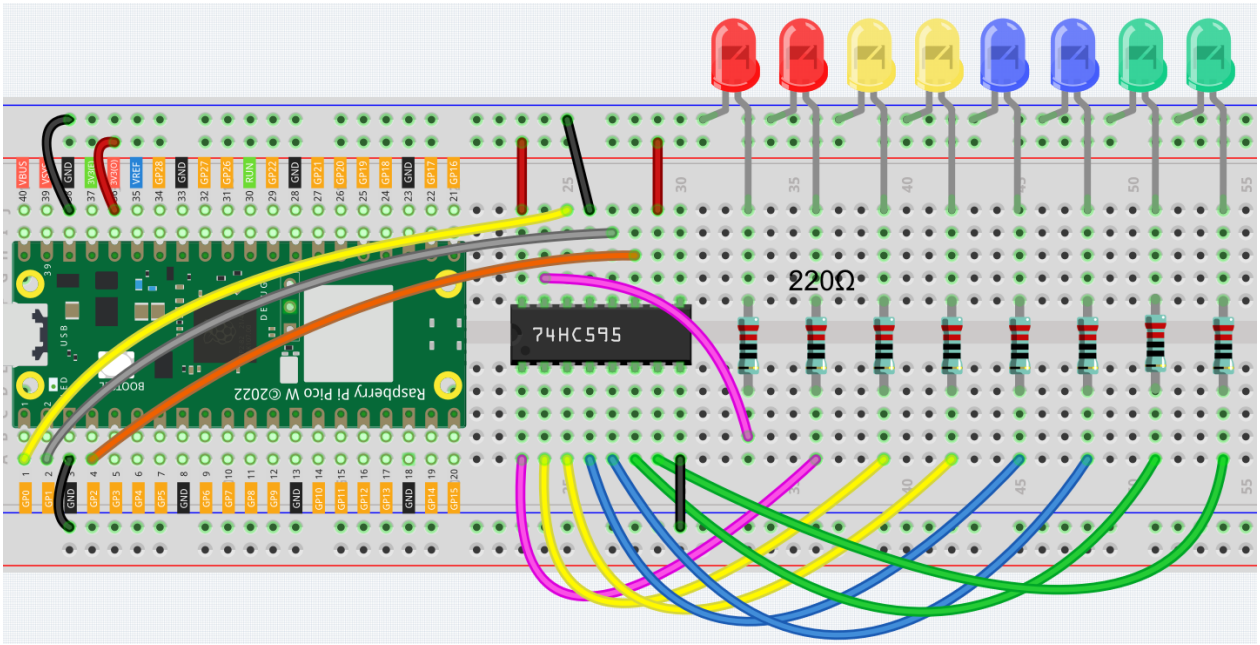
品番	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	8（220 Ω）	
6	*LED*	8	
7	*74HC595*	1	

回路図



- MR（ピン 10）がハイレベルで、OE（ピン 13）がローレベルの場合、SHcp の立ち上がりエッジでデータが入力され、SHcp の立ち上がりエッジを通じてメモリレジスタに移動します。
- 2 つのクロックが一緒に接続されている場合、シフトレジスタは常にメモリレジスタよりも 1 パルス早いです。
- メモリレジスタには、シリアルシフト入力ピン（Ds）、シリアル出力ピン（Q）、および非同期リセットボタン（ローレベル）があります。
- メモリレジスタは、3 つの状態で並列 8 ビットのバスを出力します。
- OE が有効化（ローレベル）された場合、メモリレジスタ内のデータがバス（Q0～Q7）に出力されます。

配線



コード

注釈:

- kepler-kit-main/micropython のパスの下にある 5.1_microchip_74hc595.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単純に F5 を押して実行します。
- 画面右下の角にある「MicroPython（Raspberry Pi Pico）」のインタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

sdi = machine.Pin(0,machine.Pin.OUT)
rclk = machine.Pin(1,machine.Pin.OUT)
srclk = machine.Pin(2,machine.Pin.OUT)

def hc595_shift(dat):
    rclk.low()
    time.sleep_ms(5)
    for bit in range(7, -1, -1):
```

(次のページに続く)

(前のページからの続き)

```
        srclk.low()
        time.sleep_ms(5)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_ms(5)
        srclk.high()
        time.sleep_ms(5)
    time.sleep_ms(5)
    rclk.high()
    time.sleep_ms(5)

num = 0

for i in range(16):
    if i < 8:
        num = (num<<1) + 1
    elif i>=8:
        num = (num & 0b01111111)<<1
    hc595_shift(num)
    print("{:0>8b}".format(num))
    time.sleep_ms(200)
```

プログラムが動作しているとき、 num は 8 ビットの 2 進数として 74HC595 チップに書き込まれ、8 つの LED のオンオフを制御します。シェルで num の現在の値を確認できます。

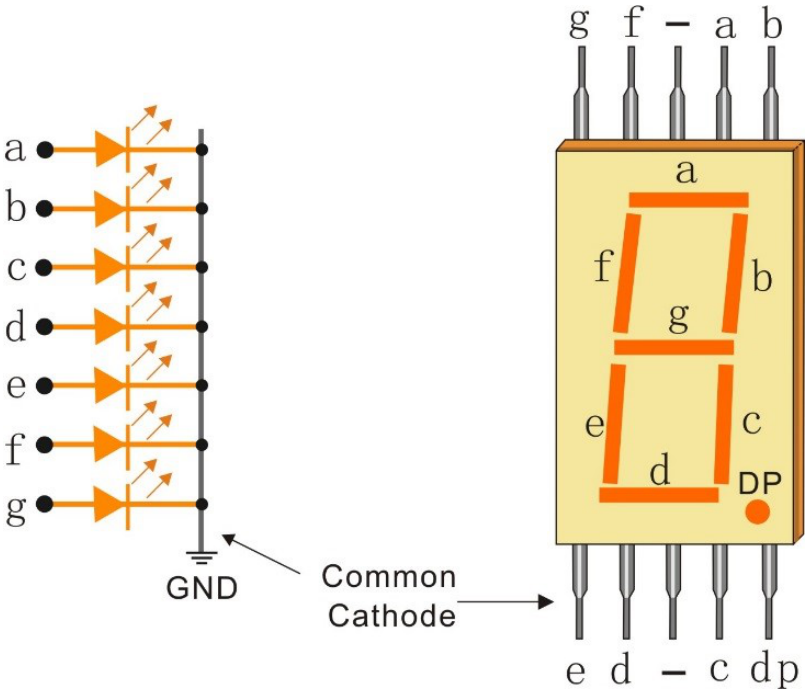
仕組み

hc595_shift() は、74HC595 に 8 つのデジタル信号を出力させます。それは 2 進数の最後のビットを Q0 に、最初のビットを Q7 に出力します。言い換えれば、2 進数「00000001」を書き込むと、Q0 はハイレベルを出力し、Q1〜Q7 はローレベルを出力します。

4.34 5.2 数字表示

7 セグメントディスプレイは日常生活で至るところで見かけます。例えば、エアコンでは温度を表示するために使われ、交通信号機ではタイマーを表示するために用いられます。

7 セグメントディスプレイは基本的に 8 つの LED で構成されています。このうち 7 つの帯状の LED が「8」の形を形成し、小さな点状の LED が小数点として機能します。これらの LED は a、b、c、d、e、f、g、および dp とマークされています。各 LED は独自のアノードピンを持ち、カソードは共有されています。ピンの位置は以下の図で示されています。



これは、7 セグメントディスプレイが完全に動作するためには同時に 8 つのデジタル信号で制御する必要があることを意味します。74HC595 がこれを可能にします。

- *7* セグメントディスプレイ

必要な部品

このプロジェクトでは以下の部品が必要です。

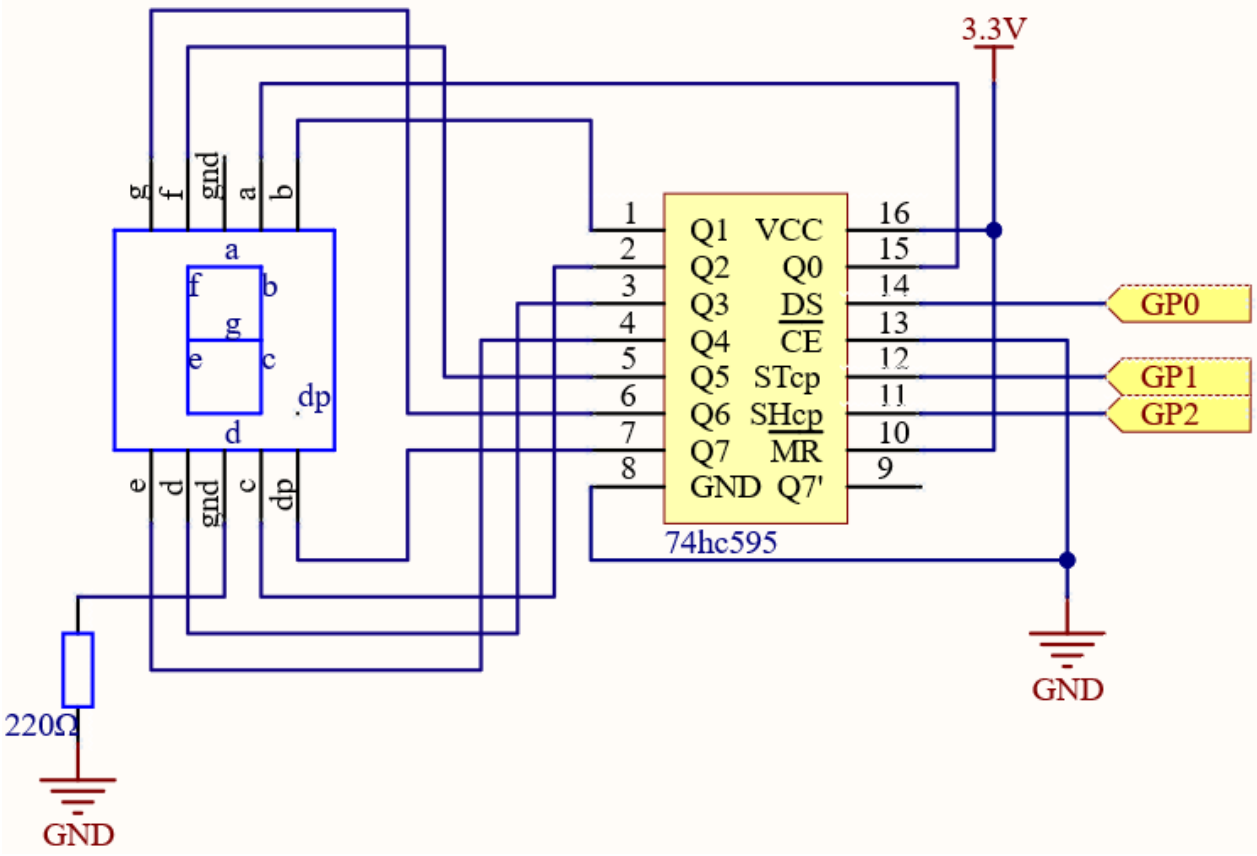
一式で購入すると便利です、そのリンクはこちらです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個々の部品も購入することができます。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*7* **セグメントディスプレイ**	1	
7	*74HC595*	1	

回路図

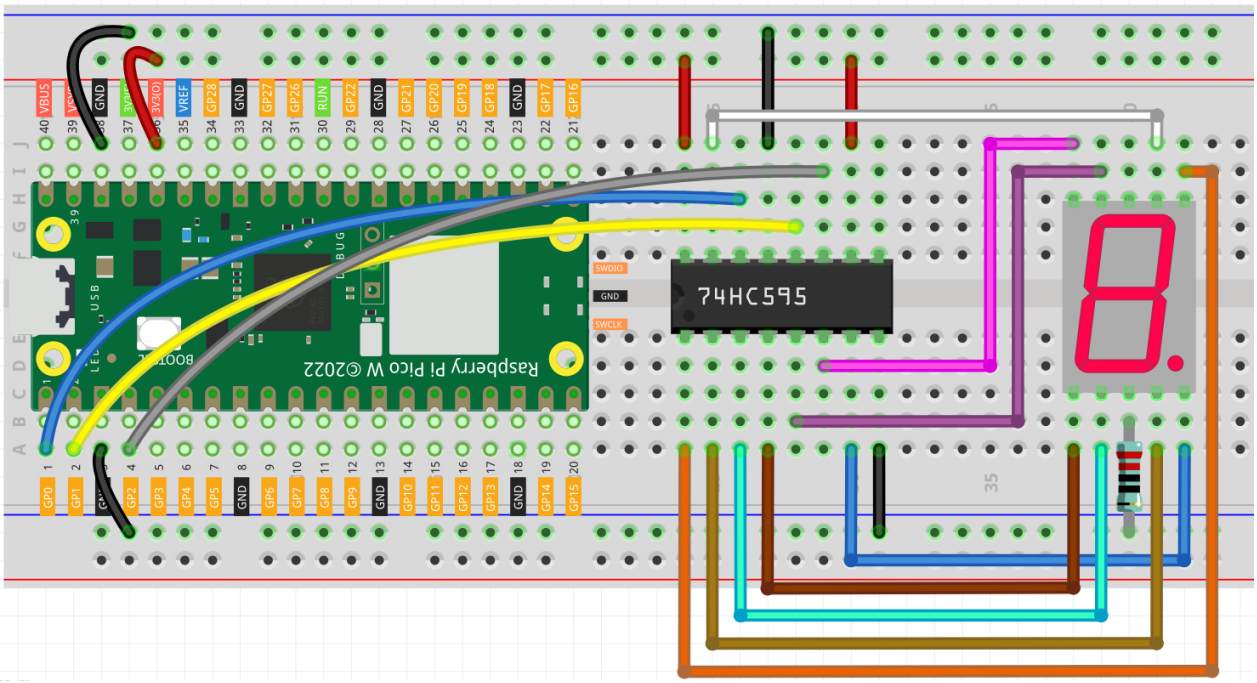


こちらの配線原理は基本的に *5.1* **マイクロチップ** *- 74HC595* と同じで、唯一の違いは Q0~Q7 が 7 セグメントディスプレイの a~g ピンに接続されている点です。

表 1　配線

74HC595	*LED* セグメントディスプレイ
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

配線図



コード

注釈:

- `kepler-kit-main/micropython` のパス内にある `5.2_number_display.py` ファイルを開くか、このコードを Thonny にコピーしてから「Run Current Script」をクリックするか、F5 を押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタプリタをクリックすることを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

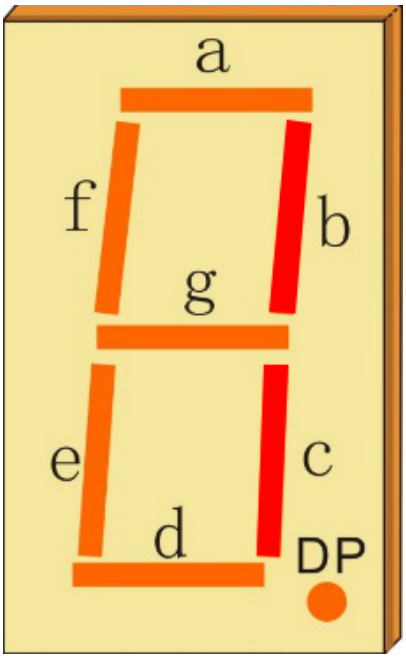
```
import machine
import time

SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(0,machine.Pin.OUT)
rclk = machine.Pin(1,machine.Pin.OUT)
srclk = machine.Pin(2,machine.Pin.OUT)

def hc595_shift(dat):
    rclk.low()
    time.sleep_ms(5)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_ms(5)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_ms(5)
        srclk.high()
        time.sleep_ms(5)
    time.sleep_ms(5)
    rclk.high()
    time.sleep_ms(5)

while True:
    for num in range(10):
        hc595_shift(SEGCODE[num])
        time.sleep_ms(500)
```

プログラムが実行されていると、LED セグメントディスプレイが 0〜9 を順番に表示するのが確認できます。

動作原理

hc595_shift() 関数によって 74HC595 は 8 つのデジタル信号を出力します。この関数は、2 進数の最後のビットを Q0 に、最初のビットを Q7 に出力します。つまり、2 進数「00000001」を書き込むと、Q0 は高レベルを、Q1〜Q7 は低レベルを出力します。

7 セグメントディスプレイが「1」と表示する場合、b と c に高レベルを書き込み、a、d、e、f、g、および dg に低レベルを書き込む必要があります。



すなわち、2 進数「00000110」を書き込む必要があります。可読性のため、16 進数表記「0x06」を使用します。

- 16 進数
- BinaryHex 変換器

同様にして、LED セグメントディスプレイに他の数字を表示させることもできます。以下の表は、それぞれの数字に対応するコードを示しています。

表 2　グリフコード

数字	2 進数コード	16 進数コード
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

これらのコードを `hc595_shift()` 関数に書き込むことで、LED セグメントディスプレイが対応する数字を表示するようにできます。

4.35 5.3 時間カウンター

4 桁の 7 セグメントディスプレイは、4 つの 7 セグメントディスプレイが連動して動作するものです。

この 4 桁の 7 セグメントディスプレイは独立して動作します。それは、人間の視覚の残像効果の原理を使って、各 7 セグメントの文字を高速でループ表示し、連続した文字列を形成します。

例えば、「1234」と表示された場合、最初の 7 セグメントには「1」が表示され、それ以降「234」は表示されません。一定の時間が経過すると、次の 7 セグメントに「2」が表示され、1 つ目、3 つ目、4 つ目の 7 セグメントは何も表示されない、といった具体です。このプロセスは非常に短い（一般に 5ms 程度）であり、視覚の残像効果と残像の原理により、我々は同時に 4 つの文字を見ることができます。

必要な部品

このプロジェクトでは、以下のコンポーネントが必要です。

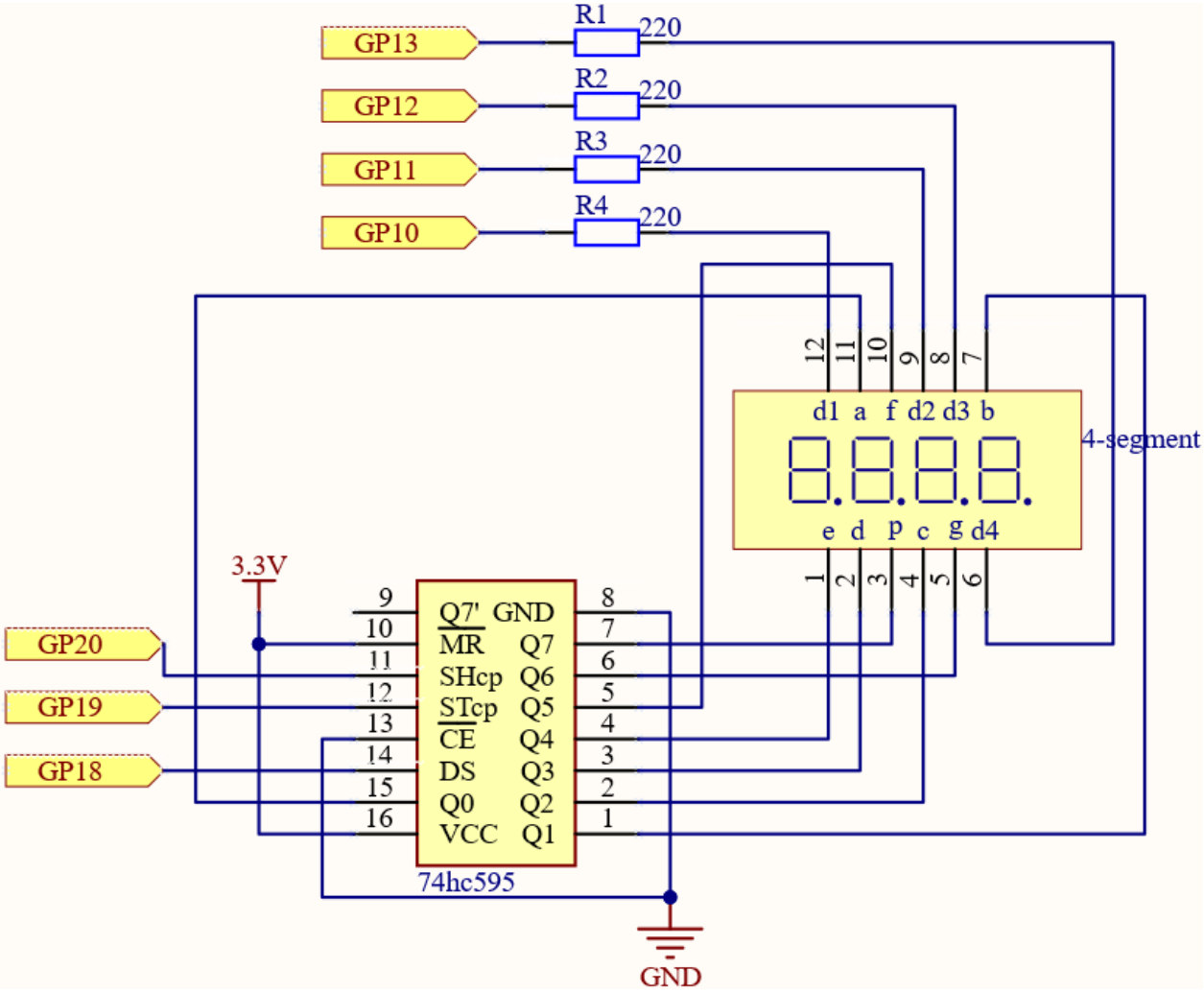
一式をまとめて購入すると確実に便利です、リンクはこちらです：

名前	キット内容	リンク
Kepler キット	450+	

以下のリンクからも個別に購入することができます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(220 Ω)	
6	*4* **桁の** *7* **セグメントディスプレイ**	1	
7	*74HC595*	1	

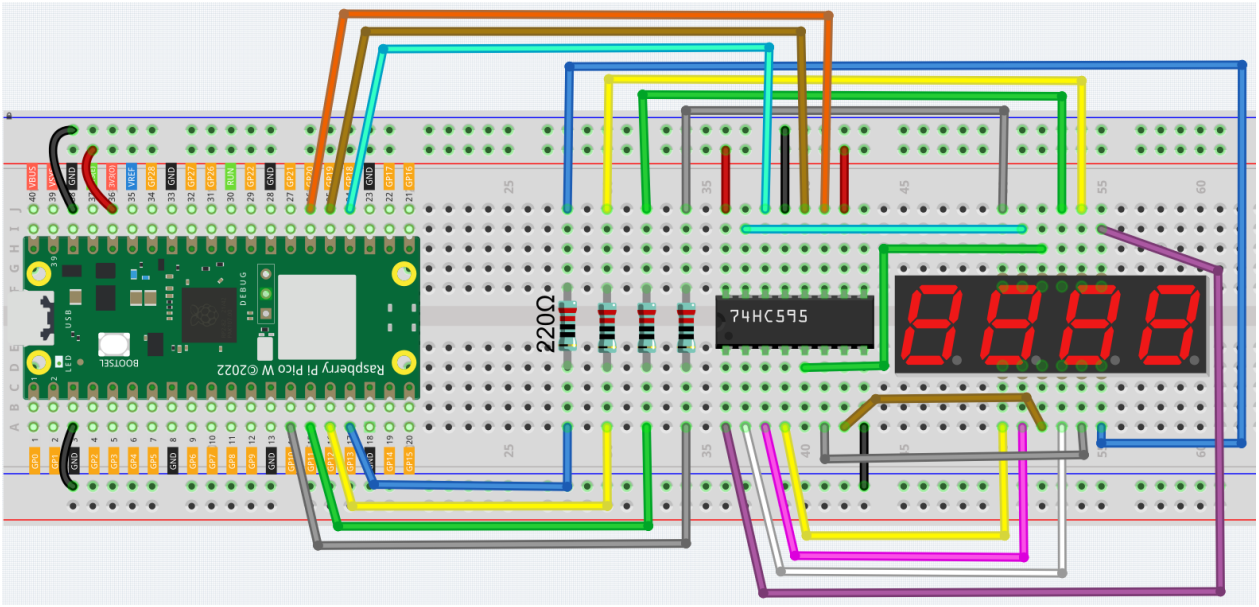
回路図



ここでの配線の原理は、*5.1* **マイクロチップ** *- 74HC595* と基本的に同じで、唯一の違いは、Q0～Q7 が 4 桁の 7 セグメントディスプレイの a～g ピンに接続されている点です。

そして、G10～G13 がどの 7 セグメントディスプレイが動作するかを選択します。

配線



コード

注釈:

- kepler-kit-main/micropython パスの下で 5.3_time_counter.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 キーを押して実行します。
- 忘れずに右下の「MicroPython（Raspberry Pi Pico）」インタプリタをクリックしてください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
srclk = machine.Pin(20,machine.Pin.OUT)

placePin = []
pin = [10,13,12,11]
for i in range(4):
    placePin.append(None)
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)
```

(次のページに続く)

(前のページからの続き)

```
timerStart=time.ticks_ms()

def timer1():
    return int((time.ticks_ms()-timerStart)/1000)

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

def clearDisplay():
    hc595_shift(0x00)

def hc595_shift(dat):
    rclk.low()
    time.sleep_us(200)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_us(200)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_us(200)
        srclk.high()
        time.sleep_us(200)
    time.sleep_us(200)
    rclk.high()
    time.sleep_us(200)

while True:
    count = timer1()
    #print(count)

    pickDigit(0)
    hc595_shift(SEGCODE[count%10])

    pickDigit(1)
    hc595_shift(SEGCODE[count%100//10])
```

(次のページに続く)

(前のページからの続き)

```
    pickDigit(2)
    hc595_shift(SEGCODE[count%1000//100])

    pickDigit(3)
    hc595_shift(SEGCODE[count%10000//1000])
```

プログラムが実行されると、4 桁の 7 セグメントディスプレイがカウンターになり、数字が 1 秒ごとに 1 増えます。

どのように動作するのか？

各 7 セグメントディスプレイに信号を書き込む処理は、 *5.2* **数字表示** と同様に、 `hc595_shift()` 関数を使用しています。4 桁の 7 セグメントディスプレイの要点は、各 7 セグメントディスプレイを選択的に活性化することです。この関連するコードは以下の通りです。

```
placePin = []
pin = [13, 12, 11, 10]
for i in range(4):
    placePin.append(None)
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

while True:
    hc595_shift(SEGCODE[count % 10])
    pickDigit(0)

    hc595_shift(SEGCODE[count % 100 // 10])
    pickDigit(1)

    hc595_shift(SEGCODE[count % 1000 // 100])
    pickDigit(2)

    hc595_shift(SEGCODE[count % 10000 // 1000])
    pickDigit(3)
```

ここでは、4 つのピン（GP10、GP11、GP12、GP13）が 4 桁の 7 セグメントディスプレイの各ビットを個々に制御するために使用されています。これらのピンの状態が `0` であれば、対応する 7 セグメントディスプレイは活性化されます。状態が 1 であれば、その逆です。

pickDigit(digit) 関数は、すべての桁を無効化した後、特定の桁だけを個別に有効にするために使用されます。その後、 hc595_shift() 関数で、7 セグメントディスプレイに対応する 8 ビットのコードが書き込まれます。

4 桁の 7 セグメントディスプレイは、連続的に交互に活性化する必要があり、それによって 4 つの数字が同時に表示されるように見えます。しかし、この例ではタイミング機能も追加する必要があります。 sleep(1) を追加すると、それが一目瞭然になります。そのため、 time.ticks_ms() 関数を使用することが、この問題に対する優れた解決策です。

```
import time

timerStart=time.ticks_ms()

def timer1():
    return int((time.ticks_ms()-timerStart)/1000)

while True:
    count = timer1()
```

time.ticks_ms() 関数で取得する時間は（非明示的な）もので、最初に取得した時間値を timerStart として記録します。その後、時間が必要な場合には、再度 time.ticks_ms() 関数を呼び出し、その値から timerStart を引いて、プログラムがどれくらい動いているか（ミリ秒単位で）を計算します。

最後に、この時間値を 4 桁の 7 セグメントディスプレイに変換して出力し、完成です。

- Time - MicroPython Docs

4.36 5.4 8x8 ピクセルグラフィックス

LED マトリクスは、低解像度のドットマトリクスディスプレイです。これは、模様表示のためにピクセルとして発光ダイオードの配列を使用しています。

これらは、屋外の日光でも見えるほど明るく、一部の店舗、広告板、サイン、および可変メッセージディスプレイ（公共交通機関の車両など）で見ることができます。

このキットで使用されているのは、16 ピンを持つ 8x8 ドットマトリクスです。アノードは行に、カソードは列に接続されています（回路レベルで）、これにより、これら 64 個の LED がまとめて制御されます。

最初の LED を点灯させるには、Row1 に高レベルを、Col1 に低レベルを供給する必要があります。2 番目の LED を点灯させるには、Row1 に高レベル、Col2 に低レベルを供給する必要があります。それ以降も同様です。各行と列のペアを通る電流を制御することで、各 LED を個々に制御して文字や画像を表示できます。

- *LED* **ドットマトリクス**
- *74HC595*

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

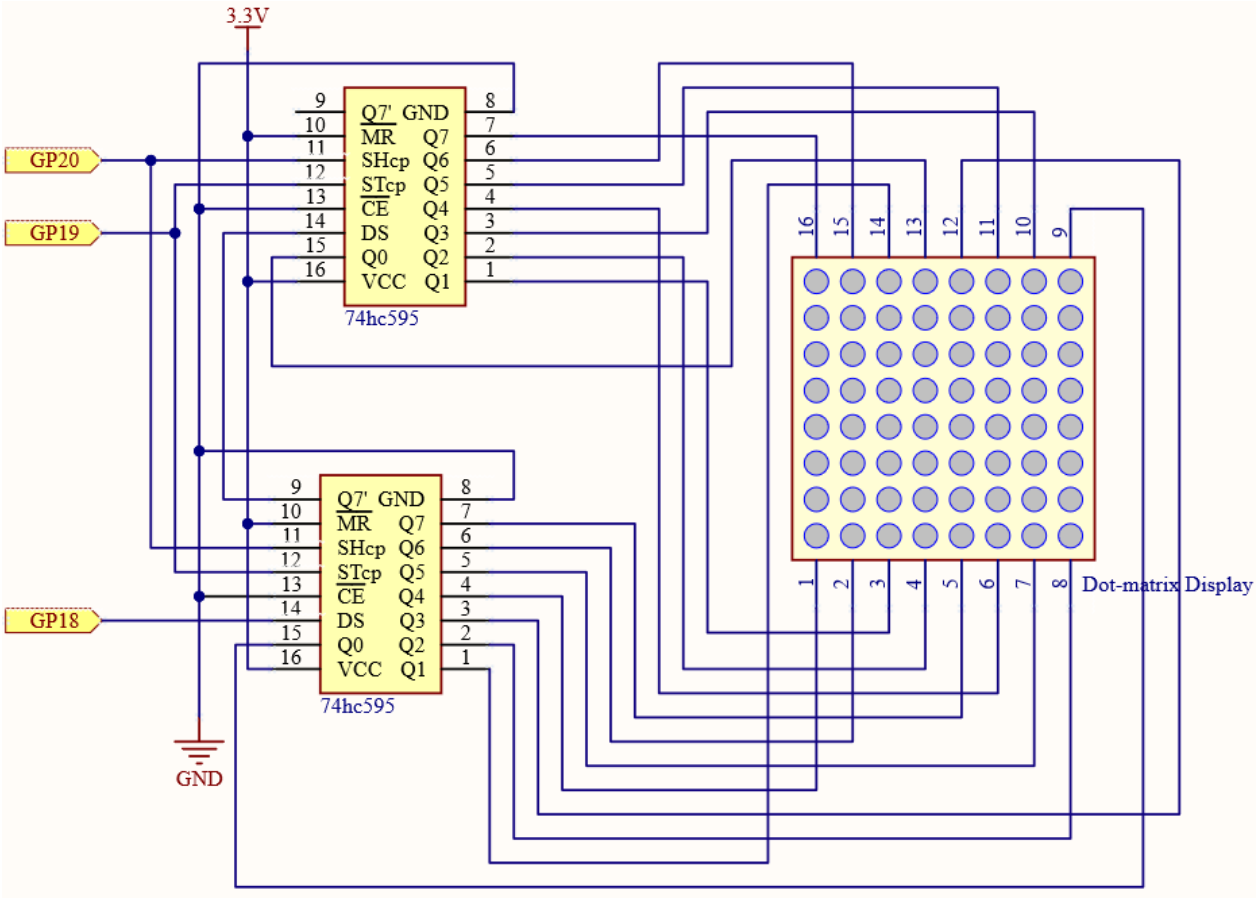
全体のキットを購入するのが便利です。リンクは以下です：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*LED* **ドットマトリクス**	1	
6	*74HC595*	2	

回路図



8x8 ドットマトリクスは、2 つの 74HC595 チップによって制御されています。一方が行を制御し、もう一方が列を制御しています。これら 2 つのチップは G18〜G20 を共有しており、これにより Pico W ボードの I/O ポートを大幅に節約できます。

Pico W は一度に 16 ビットのバイナリ数を出力する必要があります。最初の 8 ビットは行を制御する 74HC595 に与えられ、残りの 8 ビットは列を制御する 75HC595 に与えられます。これにより、ドットマトリクスが特定のパターンを表示できます。

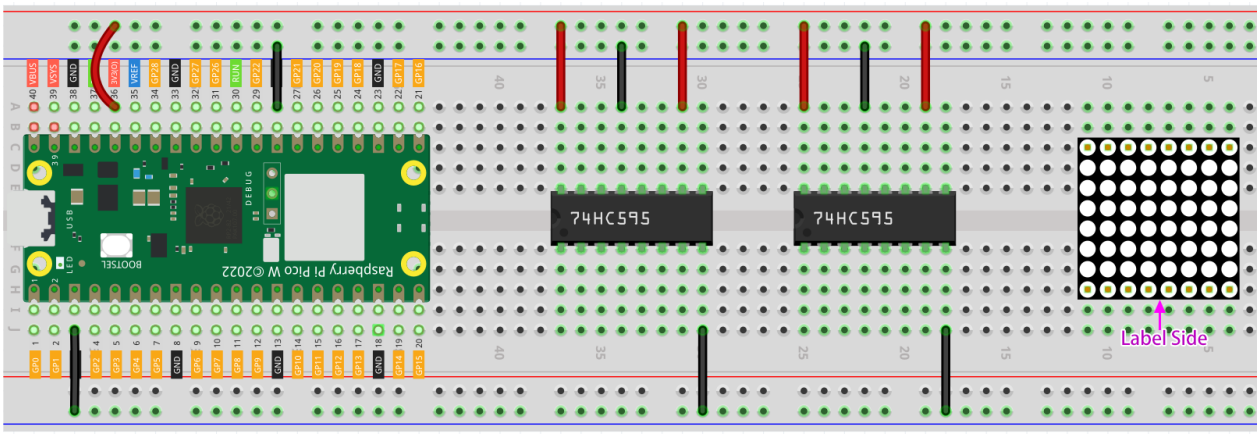
Q7': 一連の出力ピンで、複数の 74HC595 を一列に接続するために、別の 74HC595 の DS に接続されます。

配線

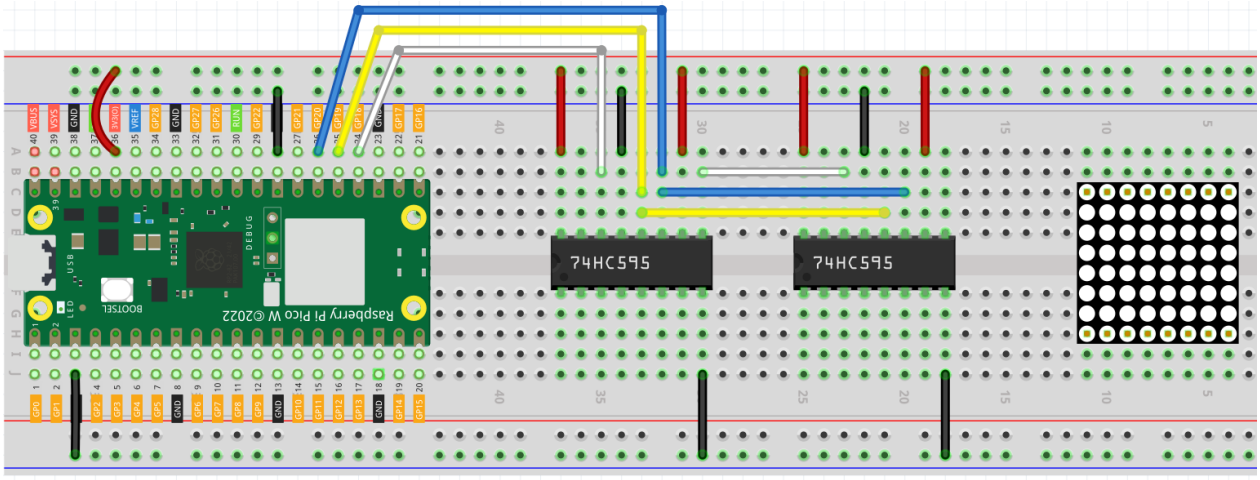
回路を組み立てます。配線が複雑なので、ステップバイステップで進めましょう。

ステップ 1: まず、ブレッドボードに Pico W、LED ドットマトリクス、および 2 つの 74HC595 チップを挿入します。Pico W の 3.3V と GND をボードの両側の穴に接続し、次に、2 つの 74HC595 チップのピン 16 と 10 を VCC に、ピン 13 とピン 8 を GND に接続します。

注釈: 上記の Fritzing 画像では、ラベルがある側が下になっています。

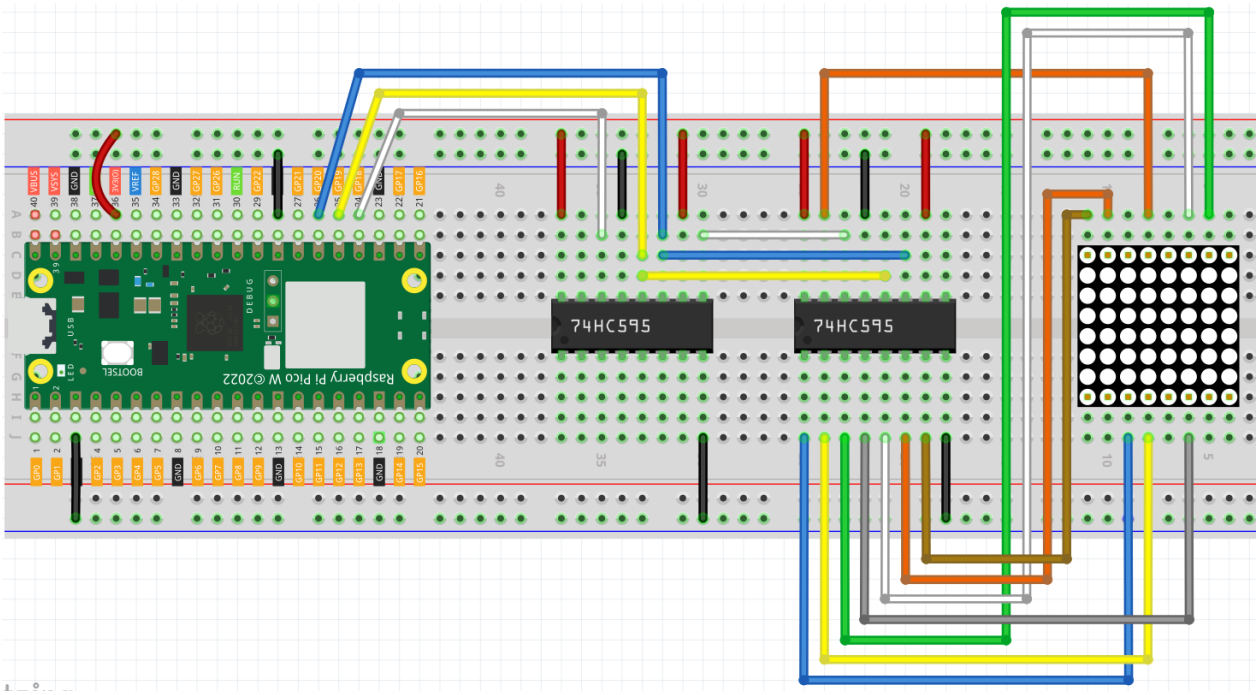


ステップ 2: 2 つの 74HC595 のピン 11 を一緒に接続し、次に GP20 に接続します。次に、2 つのチップのピン 12 を GP19 に、次に、左側の 74HC595 のピン 14 を GP18 に、ピン 9 を 2 番目の 74HC595 のピン 14 に接続します。



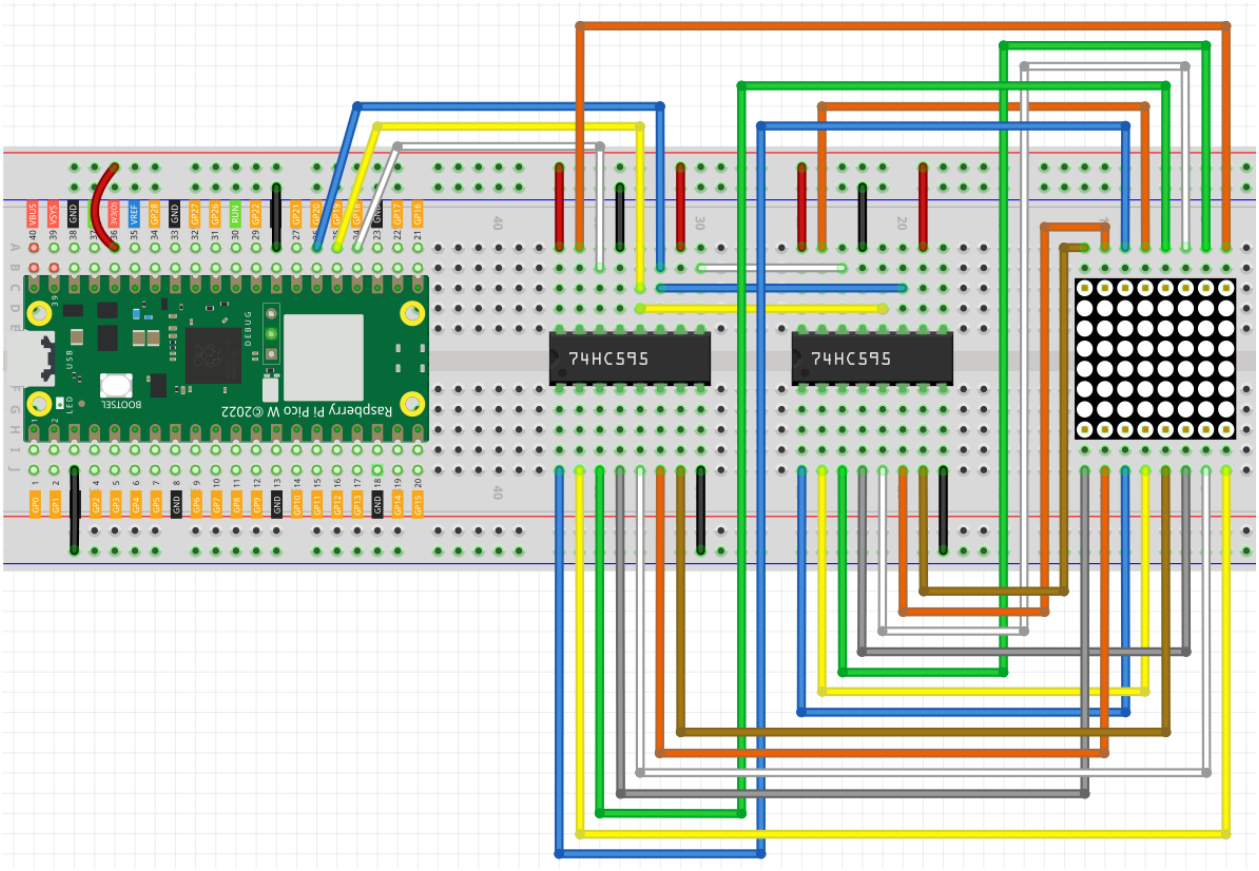
ステップ 3: 右側の 74HC595 は、LED ドットマトリクスの列を制御するためのものです。以下の表でマッピングを参照してください。したがって、74HC595 の Q0-Q7 ピンは、それぞれピン 13、3、4、10、6、11、15、および 16 にマッピングされています。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



ステップ 4: それでは、LED ドットマトリクスの ROW を接続しましょう。左側の 74HC595 は、LED ドットマトリクスの ROW を制御します。以下の表でマッピングを参照してください。見ての通り、左側の 74HC595 の Q0-Q7 は、それぞれピン 9、14、8、12、1、7、2、および 5 にマッピングされています。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



コード

注釈:

- kepler-kit-main/micropython のパスの下で 5.4_8x8_pixel_graphics.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、F5 キーを押すだけで実行できます。
- 右下の"MicroPython (Raspberry Pi Pico)"インタプリタをクリックするのを忘れないでください。
- 詳しいチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
srclk = machine.Pin(20,machine.Pin.OUT)
```

(次のページに続く)

(前のページからの続き)

```
glyph = [0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF]

# 74HC595 にデータをシフト
def hc595_in(dat):
    for bit in range(7,-1, -1):
        srclk.low()
        time.sleep_us(30)
        sdi.value(1 & (dat >> bit))
        time.sleep_us(30)
        srclk.high()

def hc595_out():
    rclk.high()
    time.sleep_us(200)
    rclk.low()

while True:
    for i in range(0,8):
        hc595_in(glyph[i])
        hc595_in(0x80>>i)
        hc595_out()
```

プログラムを実行すると、8x8 ドットマトリクスに **x** グラフィックが表示されます。

動作原理は？

ここでは、ドットマトリクスの行と列の信号を提供するために、2 つの 74HC595 を使用しています。信号を供給する方法は、前の章の `hc595_shift(dat)` と同じですが、違いはここでは一度に 16 ビットのバイナリ数を書き込む必要があることです。したがって、`hc595_shift(dat)` を二つの関数、`hc595_in(dat)` と `hc595_out()` に分割しました。

```
def hc595_in(dat):
    for bit in range(7,-1, -1):
        srclk.low()
        time.sleep_us(30)
        sdi.value(1 & (dat >> bit))
        time.sleep_us(30)
        srclk.high()

def hc595_out():
```

(次のページに続く)

(前のページからの続き)

```
    rclk.high()
    time.sleep_us(200)
    rclk.low()
```

次に、メインループで hc595_in(dat) を二回呼び出し、二つの 8 ビットのバイナリ数を書き込み、その後 hc595_out() を呼び出して、ドットマトリクスに特定のパターンを表示します。

ただし、ドットマトリクスの LED は共通の極を使用しているため、同時に複数の行/複数の列を制御すると、お互いに干渉します（例えば、(1,1) と (2,2) を同時に点灯すると、(1,2) と (2,1) が必然的に一緒に点灯します）。したがって、一度に一つの列（または一つの行）を活性化し、8 回サイクルさせ、残像の原理を使用して、人の目で 8 つのパターンを統合し、8x8 の情報量を含む一対のパターンを得る必要があります。

```
while True:
    for i in range(0,8):
        hc595_in(glyph[i])
        hc595_in(0x80>>i)
        hc595_out()
```

この例では、メイン関数は for ループをネストしており、 i が 1 のとき、最初の行だけが活性化される（制御線のチップが値 0x80 を取得する）と、最初の行の画像が書き込まれる。i が 2 のとき、2 行目が活性化され（制御線のチップが値 0x40 を取得する）と、2 行目の画像が書き込まれる。そして、8 つの出力を完了します。

ちなみに、4 桁の 7 セグメントディスプレイのように、人の目によるちらつきを防ぐために、更新レートを維持する必要があります。そのため、メインループでの余分な sleep() はできるだけ避けるべきです。

もっと学ぶ

glyph を以下の配列に置き換えて、何が表示されるか確認してみてください！

```
glyph1 = [0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF]
glyph2 = [0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF]
glyph3 = [0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF]
glyph4 = [0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF]
glyph5 = [0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF]
glyph6 = [0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF]
```

もしくは、独自のグラフィックを描いてみてください。

6. 上級編

4.37 6.1 距離の測定

超音波センサーモジュールは、物体までの距離を決定するために、ソナーおよびレーダーシステムの原理に基づいて動作します。

- **超音波モジュール**

必要なコンポーネント

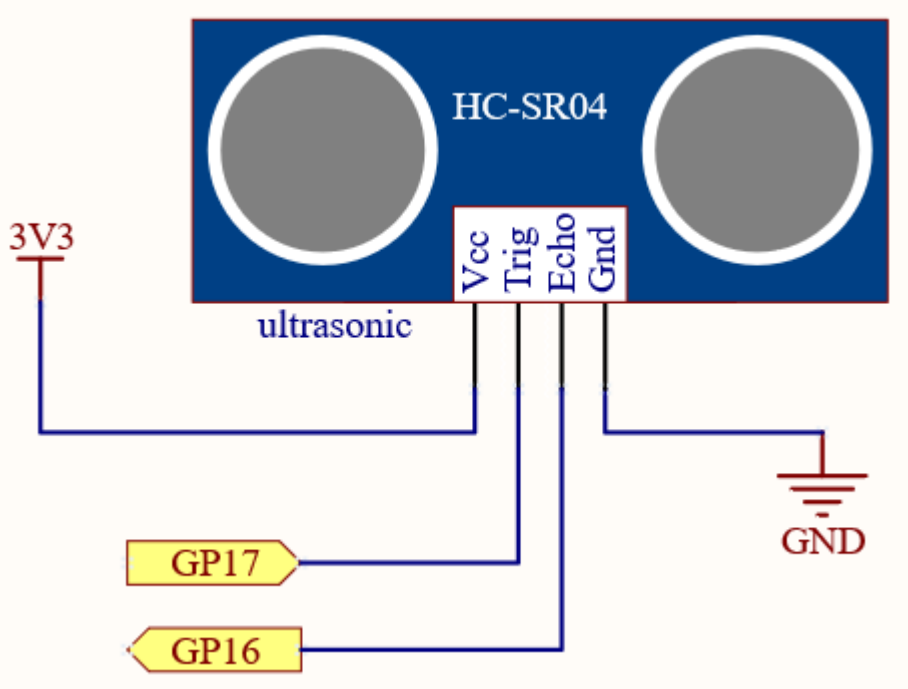
このプロジェクトには、以下のコンポーネントが必要です。

名前	このキットに含まれるアイテム	リンク
Kepler キット	450 以上	

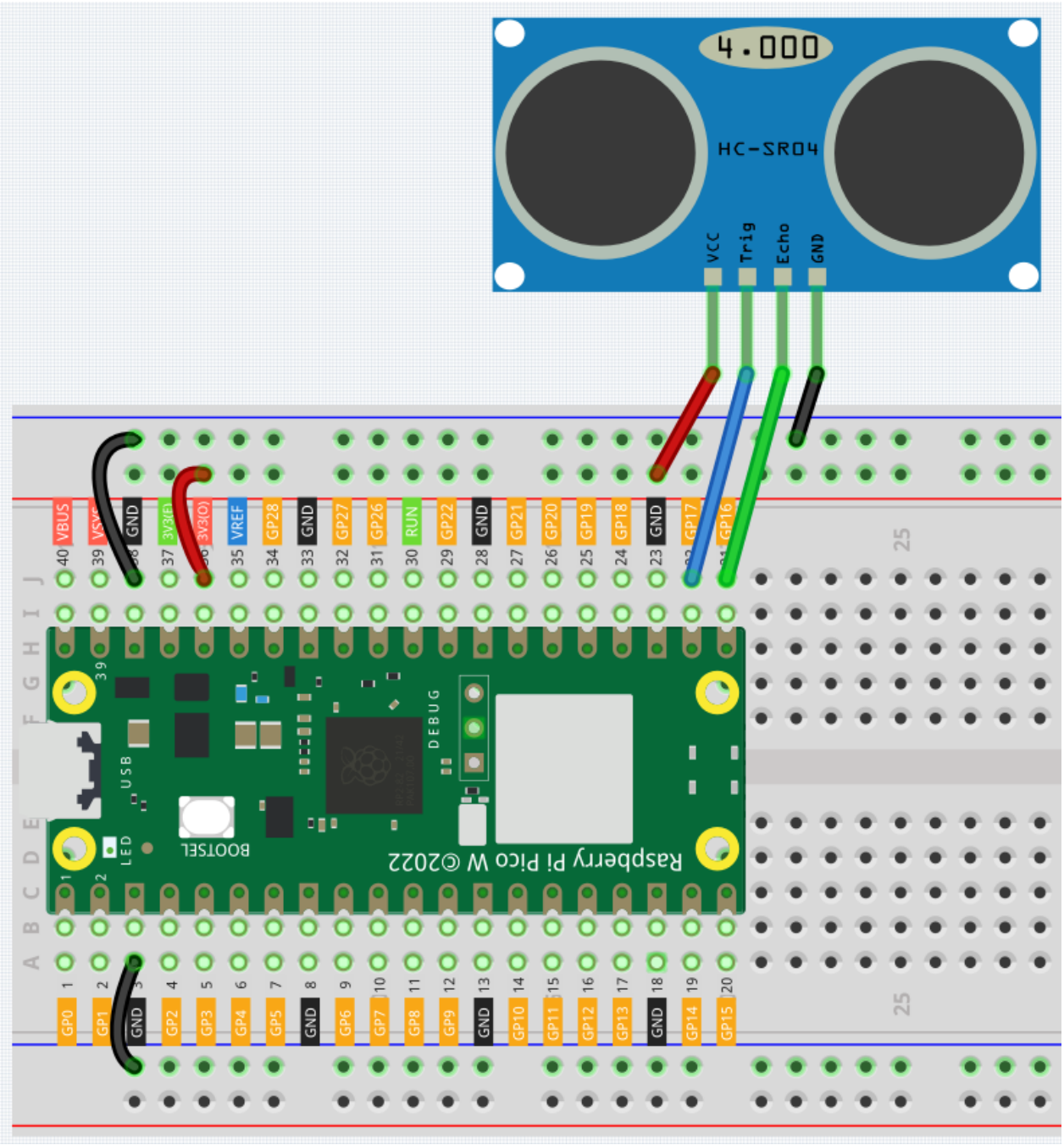
以下のリンクから個別にも購入可能です。

S/N	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**超音波モジュール**	1	

回路図



配線



コード

注釈:

- kepler-kit-main/micropython のパスの下にある 6.1_measuring_distance.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面右下隅の「MicroPython（Raspberry Pi Pico）」インタプリタをクリックするのを忘れないでください。

- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

TRIG = machine.Pin(17, machine.Pin.OUT)
ECHO = machine.Pin(16, machine.Pin.IN)

def distance():
    TRIG.low()
    time.sleep_us(2)
    TRIG.high()
    time.sleep_us(10)
    TRIG.low()
    while not ECHO.value():
        pass
    time1 = time.ticks_us()
    while ECHO.value():
        pass
    time2 = time.ticks_us()
    during = time.ticks_diff(time2, time1)
    return during * 340 / 2 / 10000

while True:
    dis = distance()
    print('Distance: %.2f' % dis)
    time.sleep_ms(300)
```

プログラムが動作すると、シェルは前方の障害物からの超音波センサーの距離を出力します。

動作原理は？

超音波センサーは、送信プローブによって発生された高周波の音波（超音波）を生成します。この超音波が物体に衝突すると、エコーとして反射され、受信プローブによって検出されます。送信から受信までの時間を計算することで、距離を求めることができます。この原理に基づいて、 `distance()` 関数が導出されます。

```
def distance():
    TRIG.low()
    time.sleep_us(2)
    TRIG.high()
```

(次のページに続く)

(前のページからの続き)

```
    time.sleep_us(10)
    TRIG.low()
    while not ECHO.value():
        pass
    time1 = time.ticks_us()
    while ECHO.value():
        pass
    time2 = time.ticks_us()
    during = time.ticks_diff(time2, time1)
    return during * 340 / 2 / 10000
```

- その中で、最初の数行は 10us の超音波を送信するために使用されます。

```
TRIG.low()
time.sleep_us(2)
TRIG.high()
time.sleep_us(10)
TRIG.low()
```

- 次に、超音波が発射された瞬間にプログラムが一時停止し、現在の時間が記録されます。

```
while not ECHO.value():
    pass
time1 = time.ticks_us()
```

- その後、プログラムは再び一時停止します。エコーが受信された後、再度現在の時間が記録されます。

```
while ECHO.value():
    pass
time2 = time.ticks_us()
```

- 最後に、2 つの記録の時間差に音速（340m/s）を掛けて、超音波モジュールと障害物の間の距離（すなわち、モジュールから障害物までの超音波の往復）を 2 倍にします。単位をセンチメートルに変換すると、必要な戻り値が得られます。

```
during = time.ticks_diff(time2, time1)
return during * 340 / 2 / 10000
```

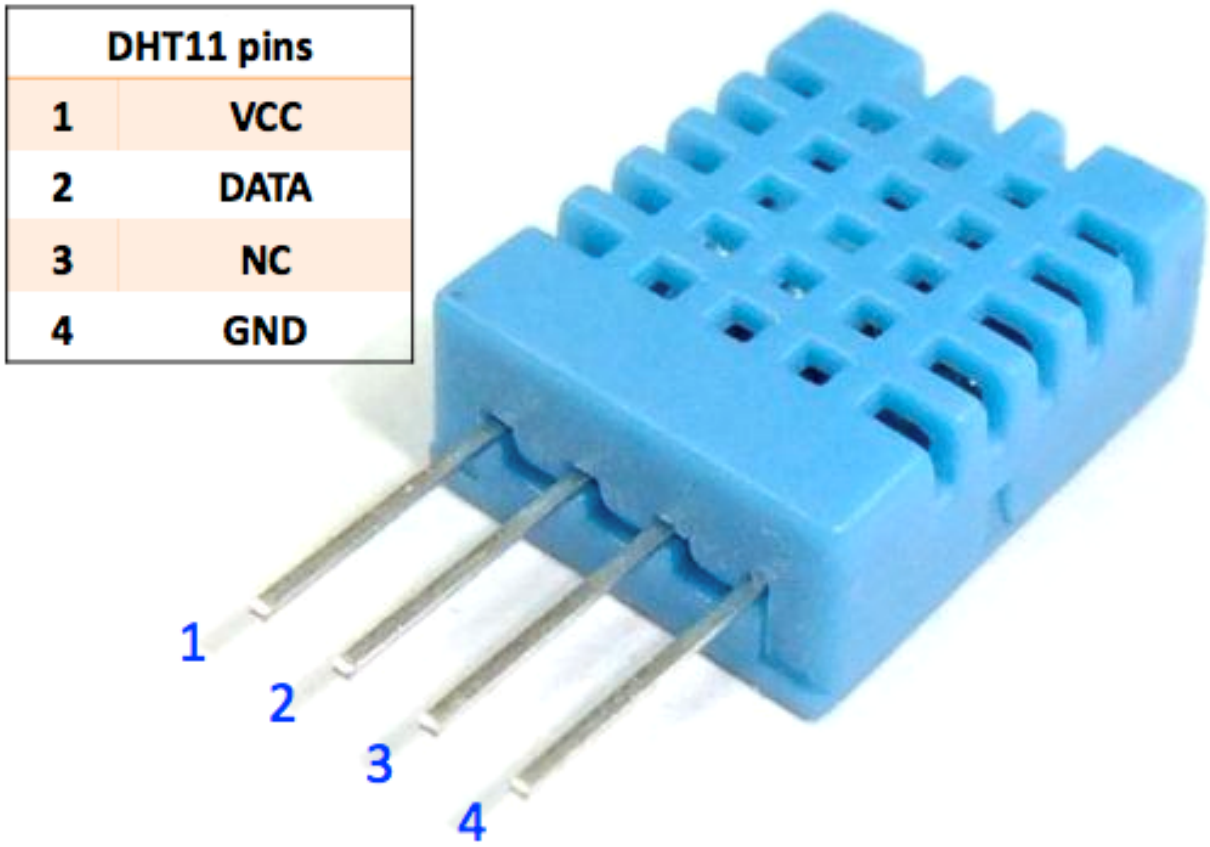
注意：超音波センサーが動作している間、プログラムは一時停止するため、複雑なプロジェクトを作成する際には遅延が発生する可能性があります。

4.38 6.2 温度・湿度センサー

湿度と温度は、物理量自体から日常生活まで密接に関連しています。人間の環境の温度と湿度は、体温調節機能や熱伝達の効果に直接影響を与えます。さらに、これは思考活動や精神状態にも影響を与え、学習や仕事の効率にも影響を与えます。

温度は、国際単位系（SI）での 7 つの基本的な物理量の一つであり、物体の熱さや寒さを測るために使用されます。摂氏度は、世界でよく使用される温度の尺度の一つで、"℃"という記号で表されます。

湿度は、空気中に存在する水蒸気の濃度です。一般的には相対湿度が使用され、%RH で表されます。相対湿度は温度に密接に関連しています。密閉された一定量のガスに対して、温度が高いほど相対湿度は低く、温度が低いほど相対湿度は高くなります。



このキットには基本的なデジタル温度・湿度センサー、 **DHT11** が付属しています。このセンサーは、周囲の空気の温度と湿度を測るために、容量性湿度センサーとサーミスターを使用し、データピンでデジタル信号を出力します（アナログ入力ピンは不要です）。

- *DHT11* **湿温度センサー**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

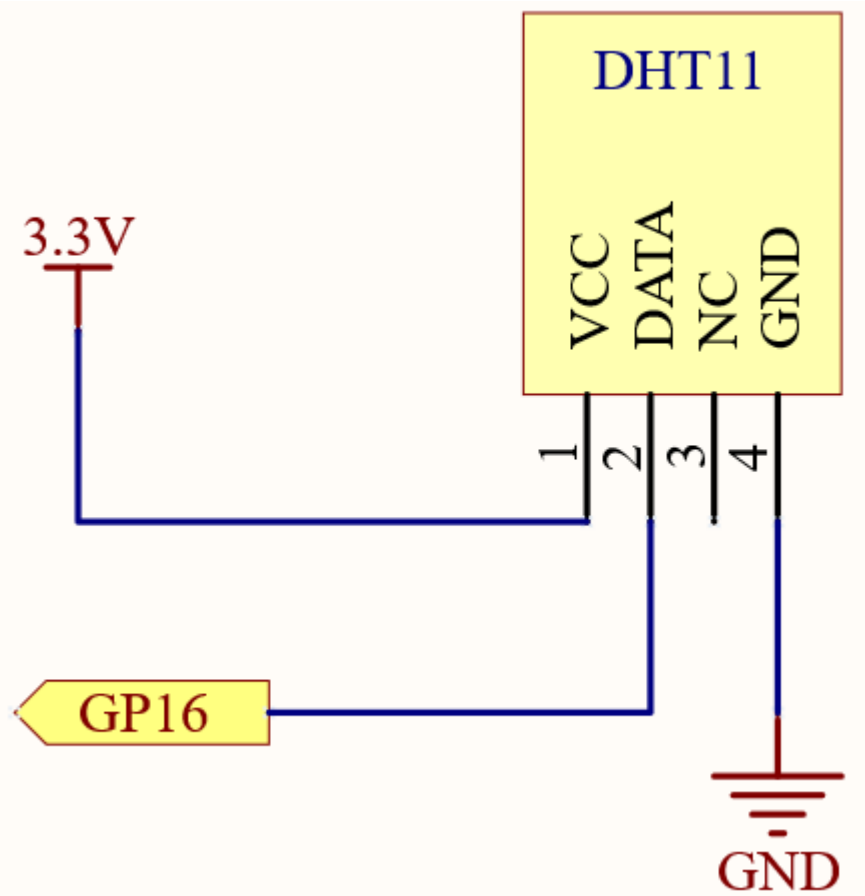
一式をまとめて購入すると便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

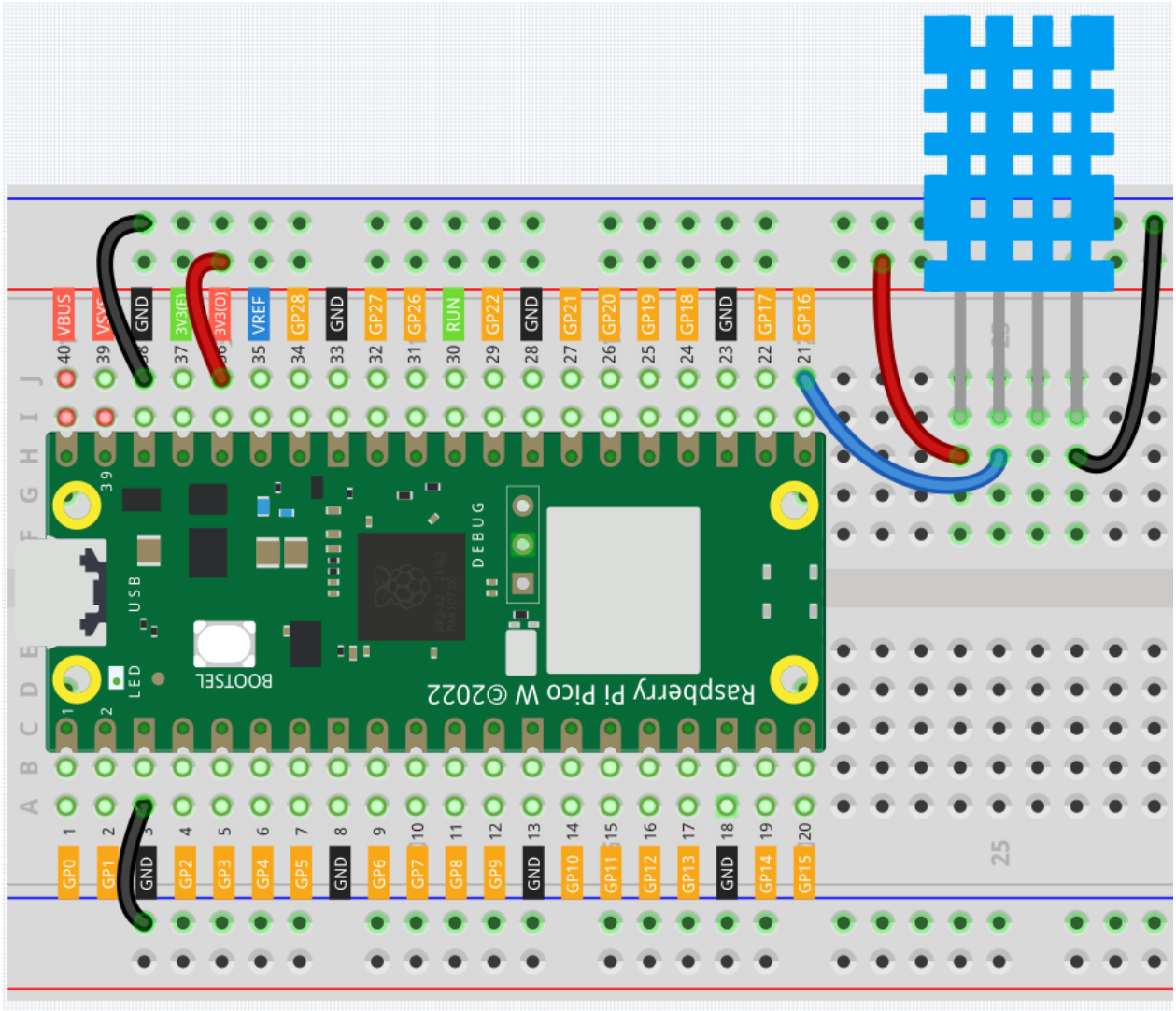
以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	いくつか	
5	*DHT11* **湿温度センサー**	1	

回路図



配線



コード

注釈:

- kepler-kit-main/micropython フォルダ内の 6.2_temperature_humidity.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、F5 キーを押して実行してください。
- 右下の角にある「MicroPython（Raspberry Pi Pico）」のインタープリターを選択することを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- ここでは dht.py というライブラリを使用する必要があります。Pico W にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from machine import Pin, I2C
import utime as time
from dht import DHT11, InvalidPulseCount

pin = Pin(16, Pin.IN, Pin.PULL_UP)
sensor = DHT11(pin)
time.sleep(5)  # 初期遅延

while True:
    try:
        sensor.measure()
        string = "Temperature:{}\nHumidity: {}".format(sensor.temperature, sensor.
↪humidity)
        print(string)
        time.sleep(4)

    except InvalidPulseCount as e:
        print('Bad pulse count - retrying ...')
```

コードを実行すると、シェルには温度と湿度が継続して表示され、プログラムが安定して動作するにつれて、これらの値はますます正確になります。

仕組み

dht ライブラリでは、関連する機能を DHT11 クラスに統合しています。

```
from dht import DHT11, InvalidPulseCount
```

DHT11 オブジェクトを初期化します。このデバイスは、デジタル入力だけで使用できます。

```
pin = Pin(16, Pin.IN, Pin.PULL_UP)
sensor = DHT11(pin)
```

sensor.measure() を使用して現在の温度と湿度を読み取り、 sensor.temperature 、 sensor.humidity に保存されます。それらはその後、出力されます。最後に、DHT11 のサンプリングレートは 1HZ なので、ループ内で time.sleep(1) が必要です。

```
while True:
    try:
        sensor.measure()
        string = "Temperature:{}\nHumidity: {}".format(sensor.temperature, sensor.
```

(次のページに続く)

(前のページからの続き)

```
↪humidity)
        print(string)
        time.sleep(4)

    except InvalidPulseCount as e:
        print('Bad pulse count - retrying ...')
```

4.39 6.3 6 軸モーショントラッキング

MPU-6050 は 6 軸（3 軸ジャイロスコープ、3 軸加速度計）のモーショントラッキングデバイスです。

加速度計は、適切な加速度を測定するツールです。例えば、地球上で静止している加速度計は、地球の重力による上向きの加速度 [3]（定義上）を約 9.81 m/s^2 として測定します。

加速度計は、産業や科学で多くの用途があります。例としては、航空機やミサイルの慣性航法システム、タブレットやデジタルカメラの画像を縦に保つためなどがあります。

ジャイロスコープは、デバイスまたは機器の方向と角速度を測定するために使用されます。ジャイロスコープの応用例としては、自動車の反転防止とエアバッグシステム、スマートデバイスのモーションセンシングシステム、ドローンの姿勢安定化システムなどがあります。

- *MPU6050* **モジュール**

必要な部品

このプロジェクトには、以下の部品が必要です。

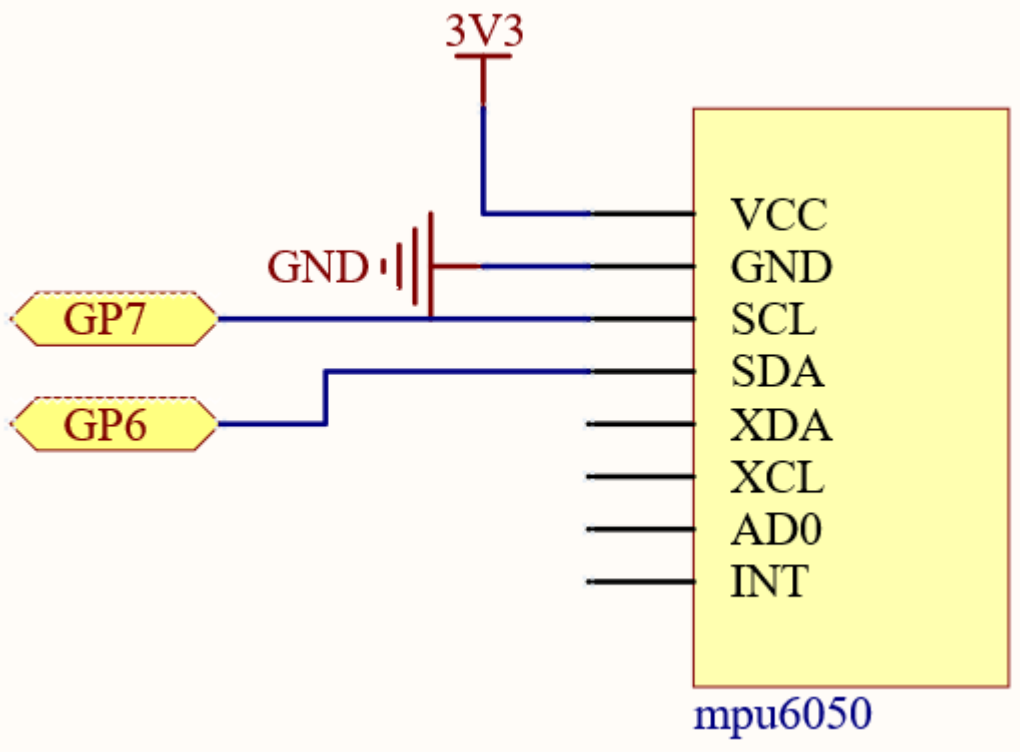
全てを一つのキットで購入するのも便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

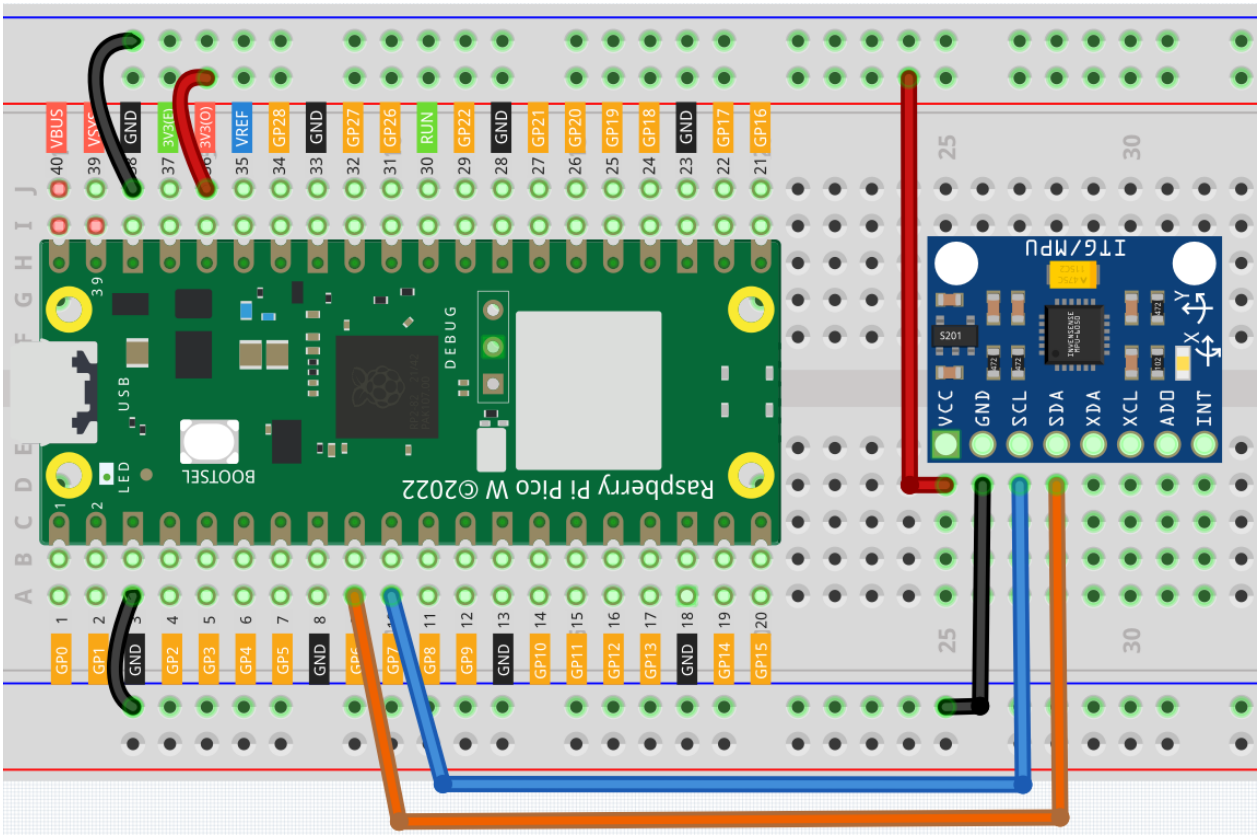
もちろん、以下のリンクから個々の部品を購入することもできます。

項番	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MPU6050* **モジュール**	1	

回路図



配線



コード

注釈:

- `kepler-kit-main/micropython` パス下の `6.3_6axis_motion_tracking.py` ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある "MicroPython (Raspberry Pi Pico)" インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- このプロジェクトでは `imu.py` と `vector3d.py` が必要です。Pico W にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from imu import MPU6050
from machine import I2C, Pin
import time
```

(次のページに続く)

(前のページからの続き)

```
i2c = I2C(1, sda=Pin(6), scl=Pin(7), freq=400000)
mpu = MPU6050(i2c)

while True:
    print("x: %s, y: %s, z: %s"%(mpu.accel.x, mpu.accel.y, mpu.accel.z))
    time.sleep(0.1)
    print("A: %s, B: %s, Y: %s"%(mpu.gyro.x, mpu.gyro.y, mpu.gyro.z))
    time.sleep(0.1)
```

プログラムを実行すると、3 軸加速度計の値と 3 軸ジャイロスコープの値が出力で循環します。この時点で MPU6050 を自由に回転させると、これらの値もそれに応じて変わるでしょう。変更を容易に確認するために、print 文の一つをコメントアウトして、他のデータセットに集中することもできます。

加速度計の値の単位は「m/s^2」、ジャイロスコープの値の単位は「°/s」です。

仕組みは？

imu ライブラリでは、関連する関数を `MPU6050` クラスに統合しています。MPU6050 は I2C モジュールであり、初期化のために I2C ピンのセットを定義する必要があります。

```
from imu import MPU6050
from machine import I2C, Pin

i2c = I2C(1, sda=Pin(6), scl=Pin(7), freq=400000)
mpu = MPU6050(i2c)
```

その後、 `mpu.accel.x` 、 `mpu.accel.y` 、 `mpu.accel.z` 、 `mpu.gyro.x` 、 `mpu.gyro.y` 、 `mpu.gyro.z` でリアルタイムの加速度と角速度の値を取得できます。

```
while True:
    print("x: %s, y: %s, z: %s"%(mpu.accel.x, mpu.accel.y, mpu.accel.z))
    time.sleep(0.1)
    print("A: %s, B: %s, Y: %s"%(mpu.gyro.x, mpu.gyro.y, mpu.gyro.z))
    time.sleep(0.1)
```

4.40 6.4 IR リモートコントロール

消費者向け電子機器では、テレビや DVD プレーヤーなどの機器を操作するためにリモートコントロールが使用されます。場合によっては、リモートコントロールによって手の届かない場所にある機器、例えばセントラルエアコンを操作することもあります。

IR レシーバーは、赤外線を受信するように調整されたフォトセルを搭載したコンポーネントです。このタイプのレシーバーはほぼ常にリモートコントロールの検出に使用されています。すべてのテレビや DVD プレーヤーの前面には、クリッカーからの IR 信号を受信するためのものがあります。リモートコントロールの内部には、テレビをオン/オフにしたりチャンネルを変更するための IR パルスを発生する対応する IR LED があります。

- **赤外線レシーバー**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

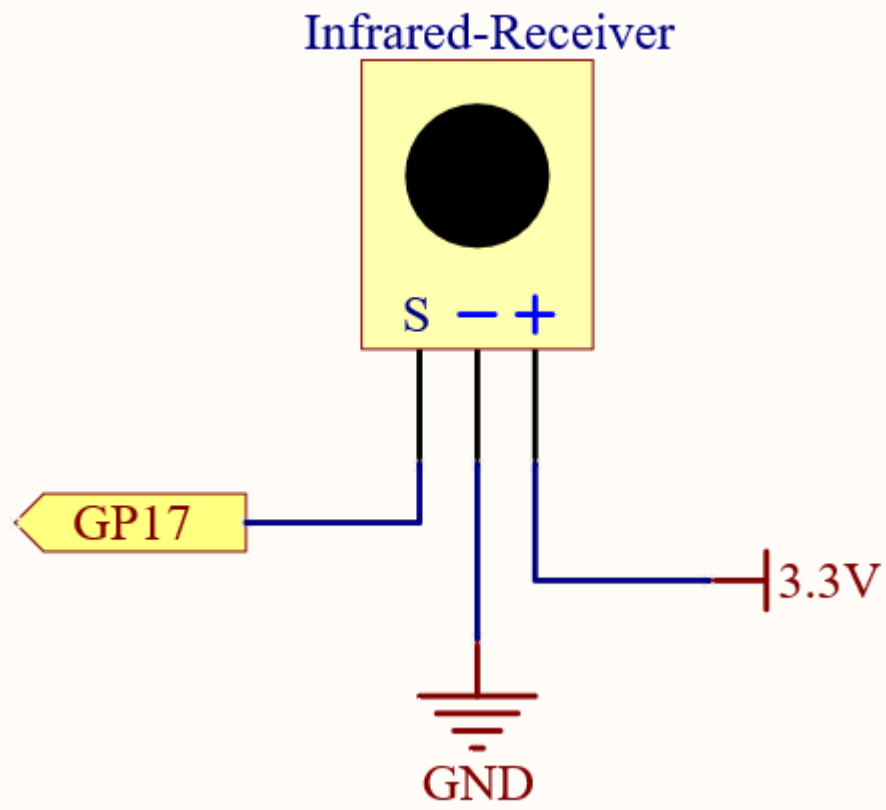
便利なのは、全体のキットを購入することです。リンクはこちら：

名前	キット内容	リンク
ケプラーキット	450 以上	

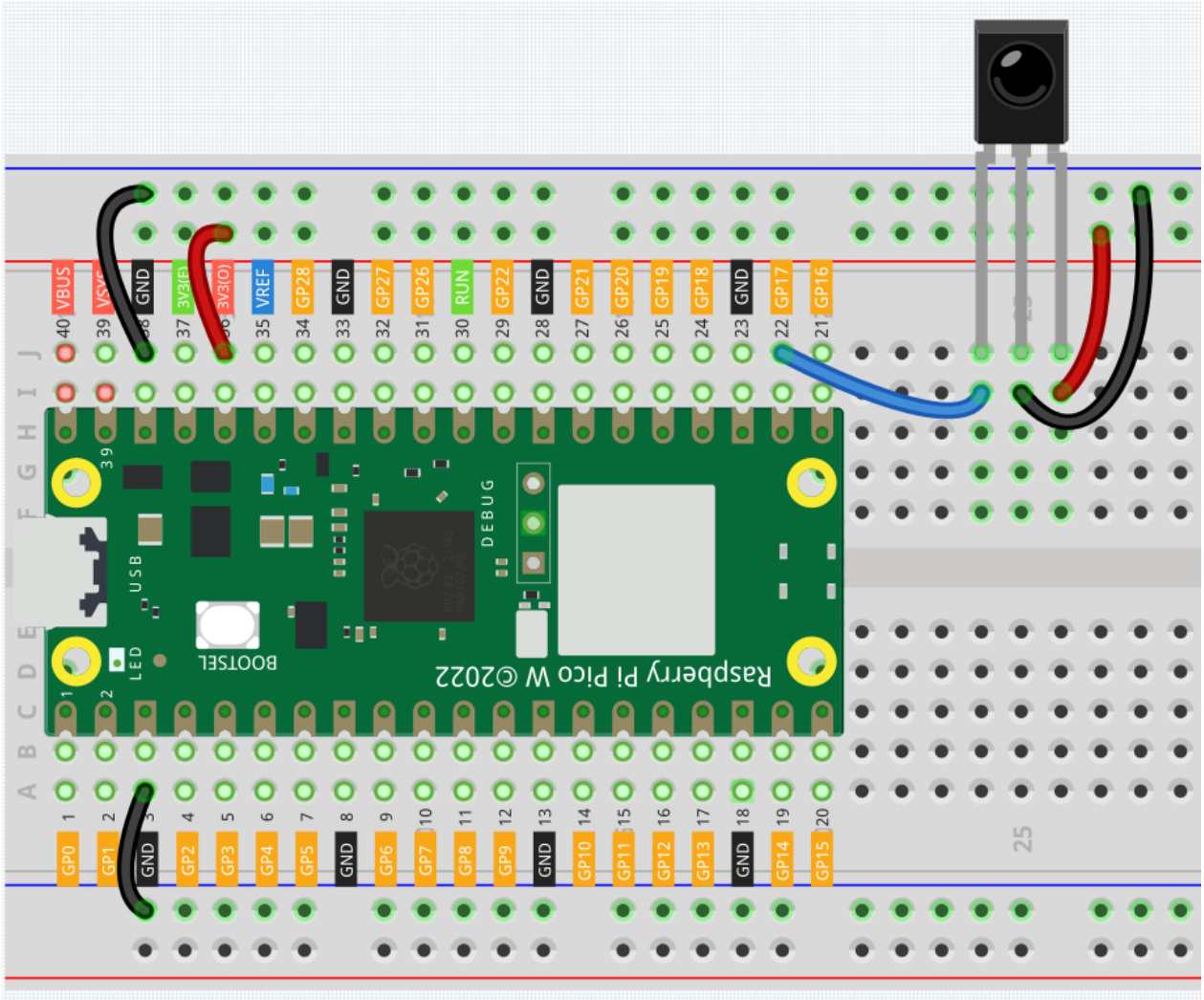
以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**赤外線レシーバー**	1	

回路図



配線



コード

注釈:

- `kepler-kit-main/micropython` のパス内の `6.4_ir_remote_control.py` ファイルを開くか、このコードを Thonny にコピペして、"Run Current Script"をクリックするか、単に F5 キーを押して実行してください。
- 右下隅の "MicroPython (Raspberry Pi Pico)" インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。
- ここでは `ir_rx` フォルダ内のライブラリが必要です。Pico にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
import time
from machine import Pin, freq
from ir_rx.print_error import print_error
from ir_rx.nec import NEC_8

pin_ir = Pin(17, Pin.IN)

def decodeKeyValue(data):
    if data == 0x16:
        return "0"
    if data == 0x0C:
        return "1"
    if data == 0x18:
        return "2"
    if data == 0x5E:
        return "3"
    if data == 0x08:
        return "4"
    if data == 0x1C:
        return "5"
    if data == 0x5A:
        return "6"
    if data == 0x42:
        return "7"
    if data == 0x52:
        return "8"
    if data == 0x4A:
        return "9"
    if data == 0x09:
        return "+"
    if data == 0x15:
        return "-"
    if data == 0x7:
        return "EQ"
    if data == 0x0D:
        return "U/SD"
    if data == 0x19:
        return "CYCLE"
    if data == 0x44:
```

(次のページに続く)

(前のページからの続き)

```
        return "PLAY/PAUSE"
    if data == 0x43:
        return "FORWARD"
    if data == 0x40:
        return "BACKWARD"
    if data == 0x45:
        return "POWER"
    if data == 0x47:
        return "MUTE"
    if data == 0x46:
        return "MODE"
    return "ERROR"

# User callback
def callback(data, addr, ctrl):
    if data < 0:  # NEC protocol sends repeat codes.
        pass
    else:
        print(decodeKeyValue(data))

ir = NEC_8(pin_ir, callback)  # Instantiate receiver
ir.error_function(print_error)  # Show debug information

try:
    while True:
        pass
except KeyboardInterrupt:
    ir.close()
```

この新しいリモートコントロールには、バッテリーを隔離するためのプラスチック片が最後にあります。それを使用する際には、このプラスチック片を引き抜いてリモートを起動する必要があります。プログラムが実行されているとき、リモートコントロールのボタンを押すと、Shell に押したキーが表示されます。

仕組みは？

このプログラムは少し複雑に見えますが、実際は IR レシーバーの基本機能を数行で実装しています。

```
import time
from machine import Pin, freq
from ir_rx.nec import NEC_8
```

(次のページに続く)

(前のページからの続き)

```
pin_ir = Pin(17, Pin.IN)

# User callback
def callback(data, addr, ctrl):
    if data < 0:  # NEC protocol sends repeat codes.
        pass
    else:
        print(decodeKeyValue(data))

ir = NEC_8(pin_ir, callback)  # Instantiate receiver
```

ここでは ir オブジェクトがインスタンス化され、IR レシーバーによって取得された信号を常に読み取ります。

結果はコールバック関数の data に記録されます。

- コールバック関数 - ウィキペディア

IR レシーバーが重複値（例：キーを押して離さない場合）を受け取ると、data < 0 となり、このデータはフィルタリングする必要があります。

そうでなければ、data は使用可能な値であり、 decodeKeyValue(data) 関数がそれをデコードするために使用されます。

```
def decodeKeyValue(data):
    if data == 0x16:
        return "0"
    if data == 0x0C:
        return "1"
    if data == 0x18:
        return "2"
    if data == 0x5E:
        return "3"
    if data == 0x08:
        return "4"
    if data == 0x1C:
        return "5"
    if data == 0x5A:
        return "6"
    if data == 0x42:
        return "7"
```

(次のページに続く)

(前のページからの続き)

```
    if data == 0x52:
        return "8"
    if data == 0x4A:
        return "9"
    if data == 0x09:
        return "+"
    if data == 0x15:
        return "-"
    if data == 0x7:
        return "EQ"
    if data == 0x0D:
        return "U/SD"
    if data == 0x19:
        return "CYCLE"
    if data == 0x44:
        return "PLAY/PAUSE"
    if data == 0x43:
        return "FORWARD"
    if data == 0x40:
        return "BACKWARD"
    if data == 0x45:
        return "POWER"
    if data == 0x47:
        return "MUTE"
    if data == 0x46:
        return "MODE"
    return "ERROR"
```

1 キーを押すと、IR レシーバーは **0x0C** のような値を出力する必要があり、それを特定のキーに対応させる必要があります。

次に、いくつかのデバッグ関数があります。これらは重要ですが、私たちが達成したい効果には関係ないため、プログラムに含めています。

```
from ir_rx.print_error import print_error

ir.error_function(print_error) # Show debug information
```

最後に、主要なプログラムとして空のループを使用します。そして、try-except を使用してプログラムが ir オブジェクトを終了させるようにします。

```
try:
    while True:
        pass
except KeyboardInterrupt:
    ir.close()
```

- try 文 - Python Docs

4.41 6.5 無線周波識別（RFID）

無線周波識別（RFID）は、オブジェクト（またはタグ）と問い合わせ装置（またはリーダー）との間で無線通信を使用して、それを追跡および識別する技術です。タグの送信範囲は数メートルに限られています。リーダーとタグは、必ずしも視界を必要としません。

ほとんどのタグには、通常、集積回路（IC）とアンテナが搭載されています。このマイクロチップは、無線周波（RF）を介してリーダーとの通信を管理するだけでなく、情報も保存します。パッシブタグには独立したエネルギー源がなく、リーダーからの外部電磁信号に電力供給を依存しています。アクティブタグは、バッテリーなどの独立したエネルギー源で動作し、その結果、処理、送信、範囲の面でより高性能かもしれません。

- *MFRC522* **モジュール**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

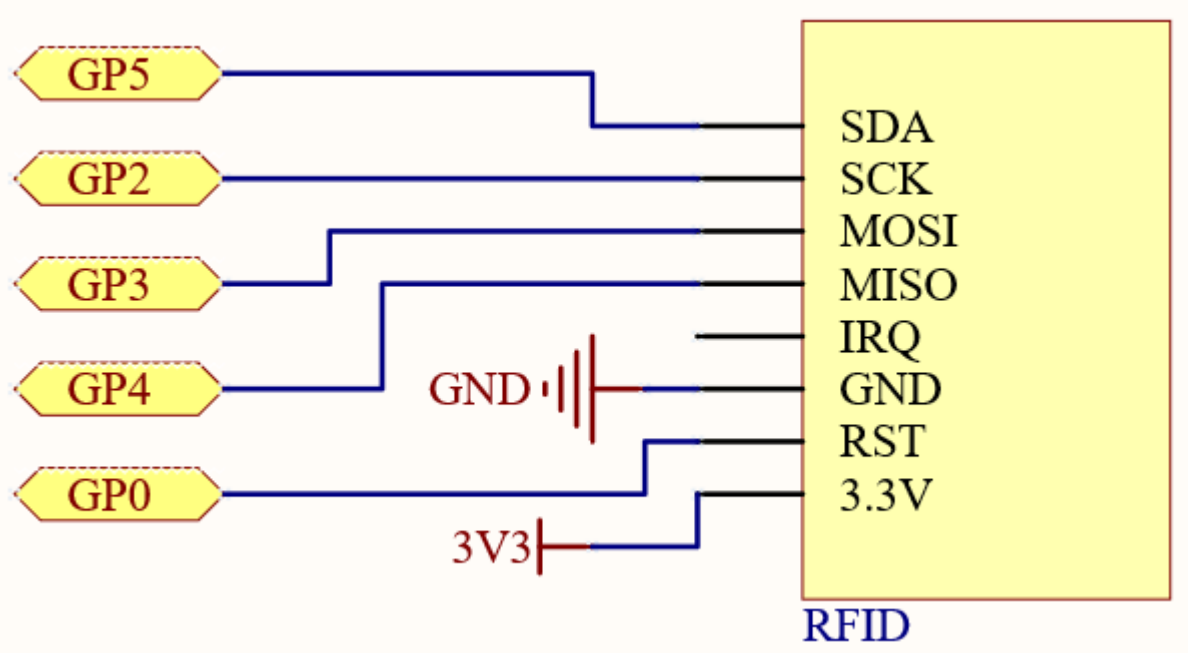
全体のキットを購入すると非常に便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

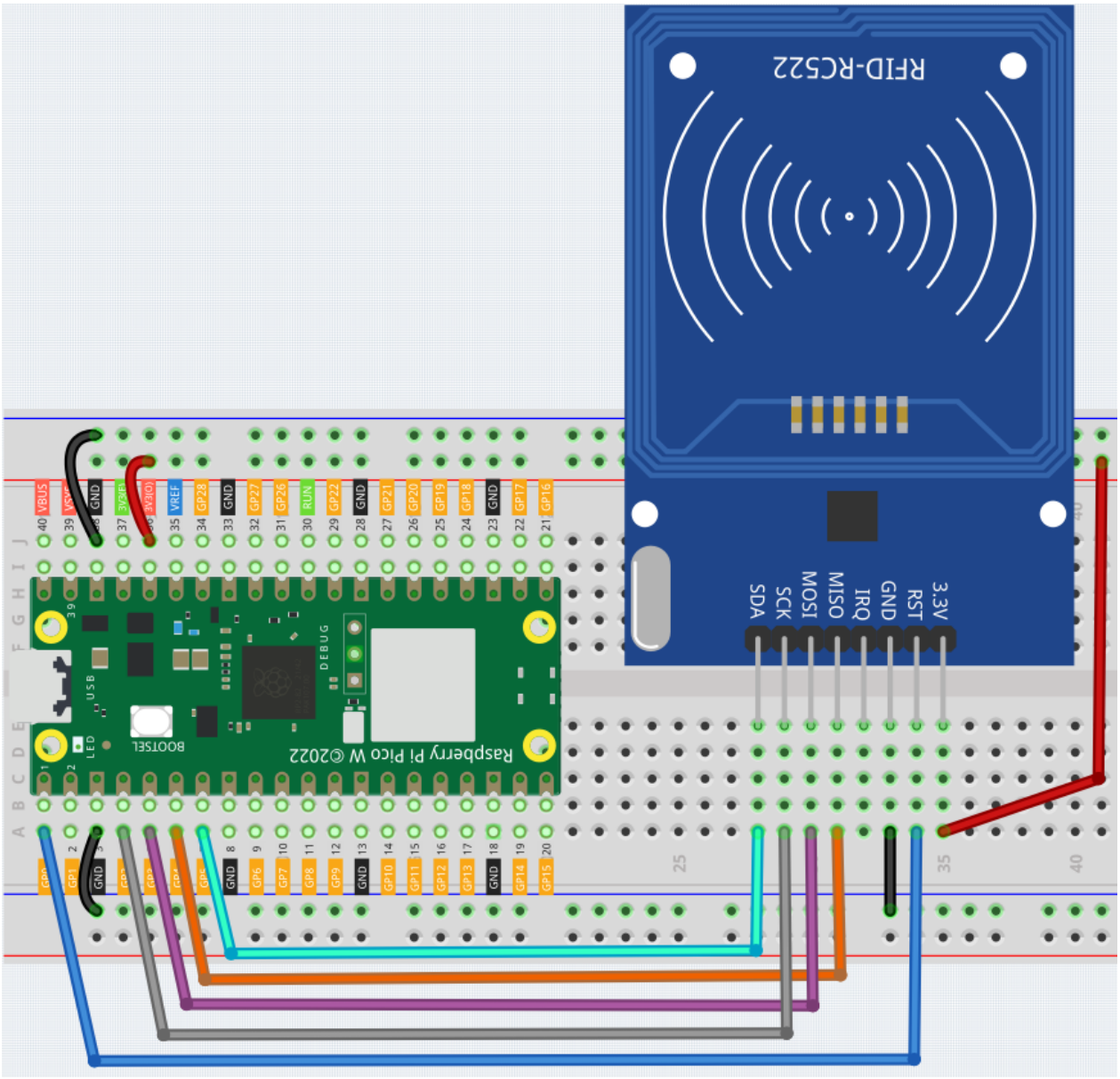
以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MFRC522* **モジュール**	1	

回路図



配線



コード

ここでは、 `mfrc522` フォルダ内のライブラリを使用する必要があります。Pico W にアップロードされているかどうか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

主要な関数は二つに分かれています：

- `6.5_rfid_write.py` ：カード（またはキー）に情報を書き込むために使用されます。
- `6.5_rfid_read.py` ：カード（またはキー）に格納されている情報を読み取るために使用されます。

`kepler-kit-main/micropython` のパスの下で `6.5_rfid_write.py` ファイルを開くか、このコードを Thonny にコピーしてから、「現在のスクリプトを実行」をクリックするか、単に F5 を押して実行します。

実行後、シェルでメッセージを入力し、その後 MFRC522 モジュールに近づけてメッセージを書き込むことができます。

```
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)

def write():
    to_write = input("Please enter the message: ")
    print("Writing...Please place the card...")
    id, text = reader.write(to_write)
    print("ID: %s\nText: %s" % (id,text))

write()
```

kepler-kit-main/micropython のパスの下で 6.5_rfid_read.py ファイルを開くか、このコードを Thonny にコピーしてから、「現在のスクリプトを実行」をクリックするか、単に F5 を押して実行します。

実行後、カード（またはキー）に格納されたメッセージを読み取ることができます。

```
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)

def read():
    print("Reading...Please place the card...")
    id, text = reader.read()
    print("ID: %s\nText: %s" % (id,text))

read()
```

どのように動作するか？

```
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)
```

SimpleMFRC522() クラスをインスタンス化します。

```
id, text = reader.read()
```

この関数はカードデータを読み取るために使用されます。読み取りが成功すると、id と text が返されます。

```
id, text = reader.write("text")
```

この関数は、カードに情報を書き込むために使用されます。 **Enter** キーを押して書き込みを完了します。`text` はカードに書き込む情報です。

7. 面白いプロジェクト

4.42 7.1 光センサー・テルミン

テルミンは物理的な接触を必要としない電子楽器です。プレイヤーの手の位置に応じて、異なる音階を出力します。

通常、制御部分は、テルミニスト（テルミン奏者）の手の位置を感知し、一方の手で発振器を、もう一方の手で音量を制御する二つの金属アンテナで構成されています。テルミンからの電気信号は増幅され、スピーカーに送られます。

Pico W を使って同じ楽器を再現することはできませんが、フォトレジスタとパッシブブザーを使用して、類似したゲームプレイを実現できます。

- Theremin - Wikipedia

必要な部品

このプロジェクトでは、以下の部品が必要です。

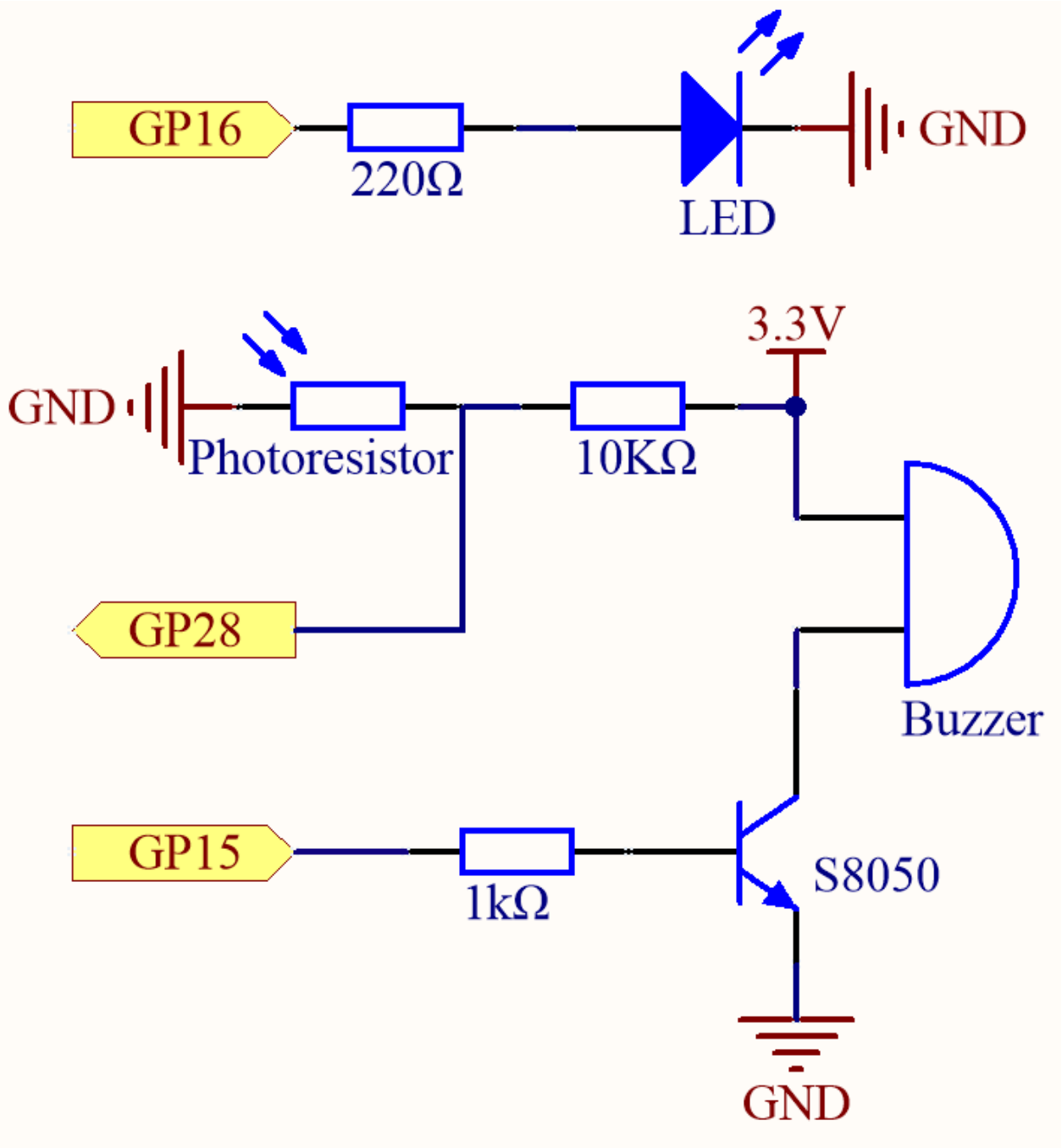
一式を購入することは非常に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個々の部品も購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*LED*	1	
6	**トランジスタ**	1(S8050)	
7	**抵抗器**	3(1K Ω, 220 Ω, 10K Ω)	
8	アクティブ **ブザー**	1	
9	**フォトレジスタ**	1	

回路図

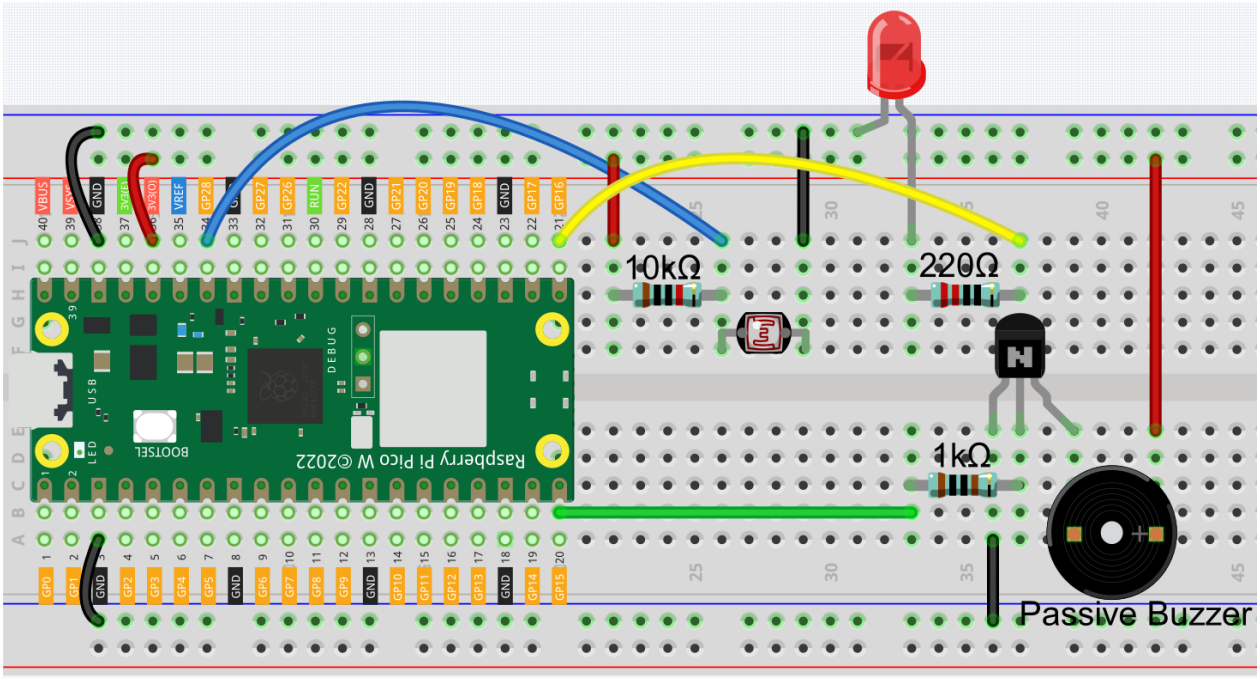


プロジェクトを開始する前に、光の強度の範囲を調整するために、フォトレジスタの上で手を上下に振ってください。GP16 に接続された LED は、デバッグ時間を示すために使用され、LED が点灯しているとデバッグが開始され、消灯しているとデバッグが終了したことを示します。

GP15 が高レベルを出力すると、S8050（NPN トランジスタ）が導通し、パッシブブザーが鳴り始めます。

光が強いと、GP28 の値は小さくなります。逆に、光が弱いと、値は大きくなります。フォトレジスタの値をプログラムしてパッシブブザーの周波数に影響を与えることで、感光デバイスをシミュレートできます。

配線



コード

注釈:

- kepler-kit-main/micropython パス下の 7.1_light_theremin.py ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある "MicroPython (Raspberry Pi Pico)" インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import utime

led = machine.Pin(16, machine.Pin.OUT)
photoresistor = machine.ADC(28)
buzzer = machine.PWM(machine.Pin(15))

light_low = 65535
light_high = 0
```

(次のページに続く)

(前のページからの続き)

```
def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def tone(pin, frequency, duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

# フォトレジスタの最大・最小値をキャリブレーションする。
timer_init_start = utime.ticks_ms()
led.value(1)
while utime.ticks_diff(utime.ticks_ms(), timer_init_start) < 5000:
    light_value = photoresistor.read_u16()
    if light_value > light_high:
        light_high = light_value
    if light_value < light_low:
        light_low = light_value
led.value(0)

# プレイ
while True:
    light_value = photoresistor.read_u16()
    pitch = int(interval_mapping(light_value, light_low, light_high, 50, 6000))
    if pitch > 50:
        tone(buzzer, pitch, 20)
    utime.sleep_ms(10)
```

プログラムが実行されると、LED が点灯し、フォトレジスタの検出範囲をキャリブレーションするために 5 秒間の時間が与えられます。

これは、使用する際に違う光環境（例えば、正午と夕暮れでの光の強度が異なる）や、フォトレジスタ上で手を動かす高さが異なるためです。楽器を演奏する際に手をどれだけ高く持ち上げるか、つまり手の最大・最小の高さを設定する必要があります。

5 秒後、LED は消灯し、その時点でフォトレジスタ上で手を振って演奏できます。

4.43 7.2 室温計

サーミスターと I2C LCD1602 を使って、室温計を作成できます。

このプロジェクトは非常にシンプルで、 *2.13* **温度計** に基づき I2C LCD1602 で温度を表示します。

必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

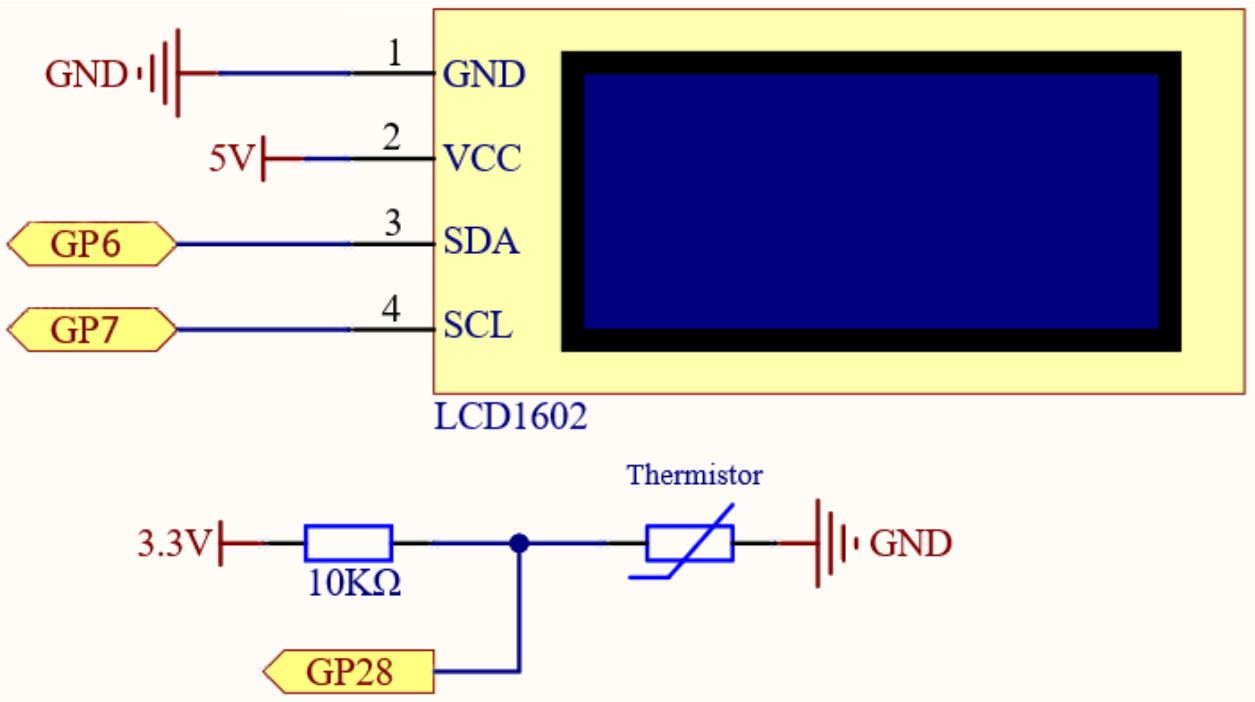
一式をまとめて購入するのは非常に便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

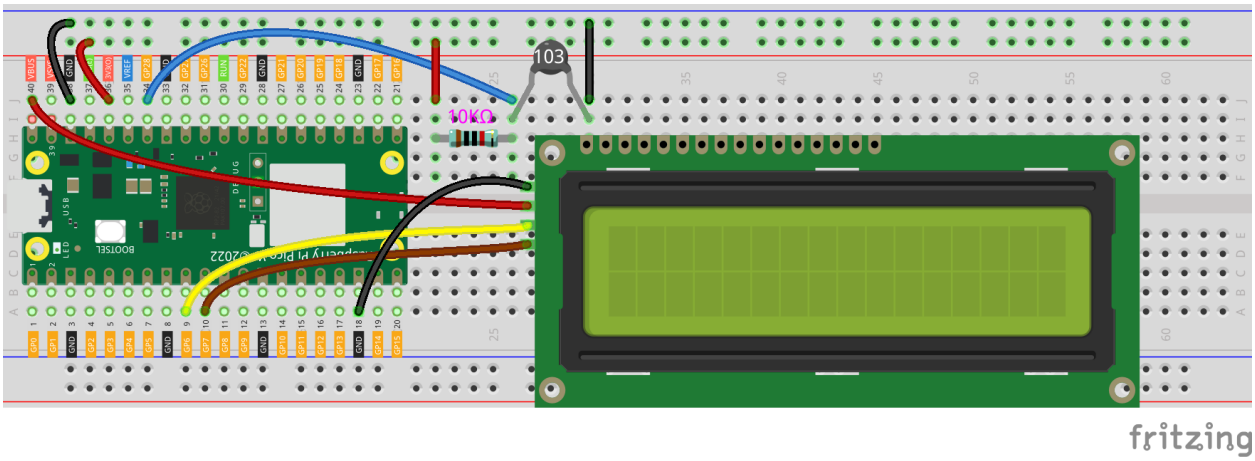
以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**サーミスター**	1	
7	*I2C LCD1602*	1	

回路図



配線



コード

注釈:

- `kepler-kit-main/micropython` のパス下にある `7.2_room_temperature_meter.py` ファイルを開くか、このコードを Thonny にコピペして、「Run Current Script」をクリックまたは F5 キーを押して実行します。
- 右下角にある「MicroPython（Raspberry Pi Pico）」インタープリターをクリックして選択してください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
from lcd1602 import LCD
import machine
import utime
import math

thermistor = machine.ADC(28)
lcd = LCD()

while True:
    temperature_value = thermistor.read_u16()
    Vr = 3.3 * float(temperature_value) / 65535
    Rt = 10000 * Vr / (3.3 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    #Fah = Cel * 1.8 + 32
    #print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
    #utime.sleep_ms(200)

    string = " Temperature is \n    " + str('{:.2f}'.format(Cel)) + " C"
    lcd.message(string)
    utime.sleep(1)
    lcd.clear()
```

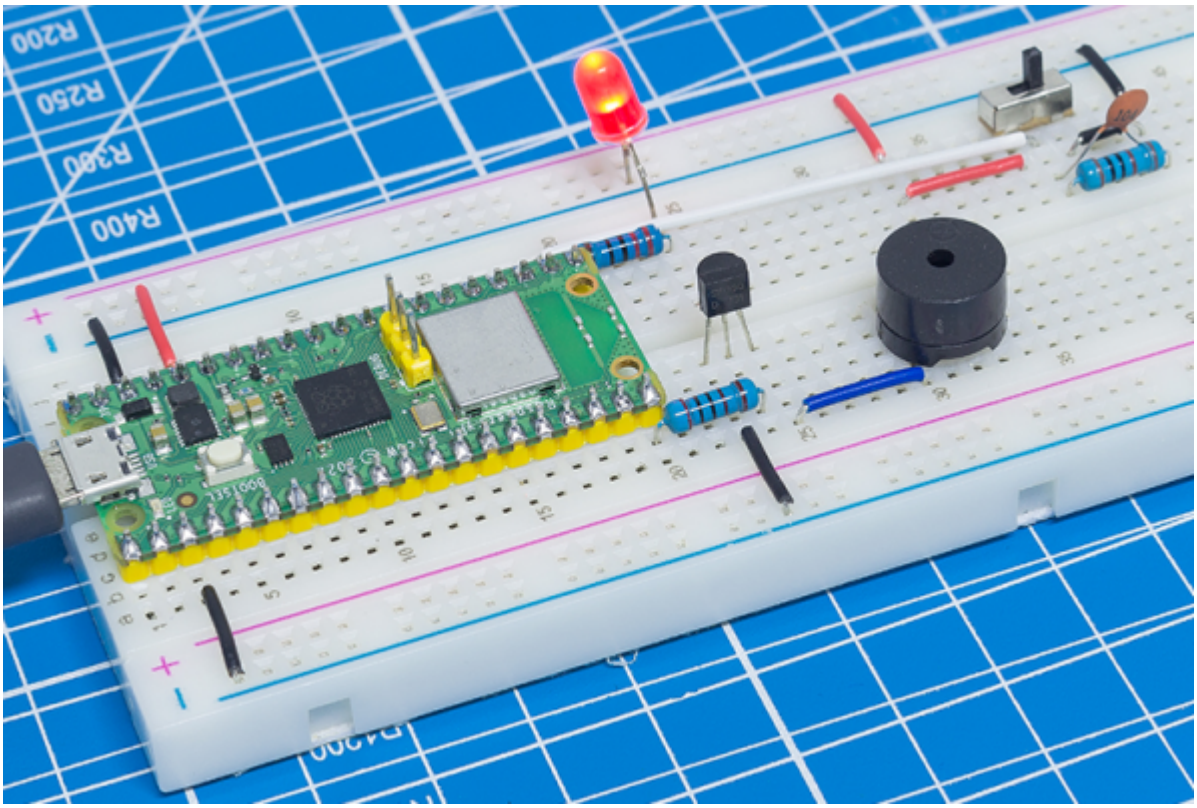
プログラムが実行された後、LCD には現在の環境の温度値が表示されます。

注釈: コードと配線が正しいのにもかかわらず、LCD が何も表示しない場合は、裏面のポテンショメーターを回してコントラストを調整できます。

4.44 7.3 警報サイレンランプ

警察の灯りは、実際の生活（または映画）でよく見かけます。通常、交通を整理するため、警告装置として、そして警察官、緊急車両、消防車、作業車の重要な安全装置として使用されます。その灯りや音を見聞きした場合、注意が必要です。それは、あなた（または周囲の人々）が危険にさらされている可能性があるからです。

ここでは、LED とブザーを用いて小型の警告灯を作成し、スライドスイッチで起動します。



必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

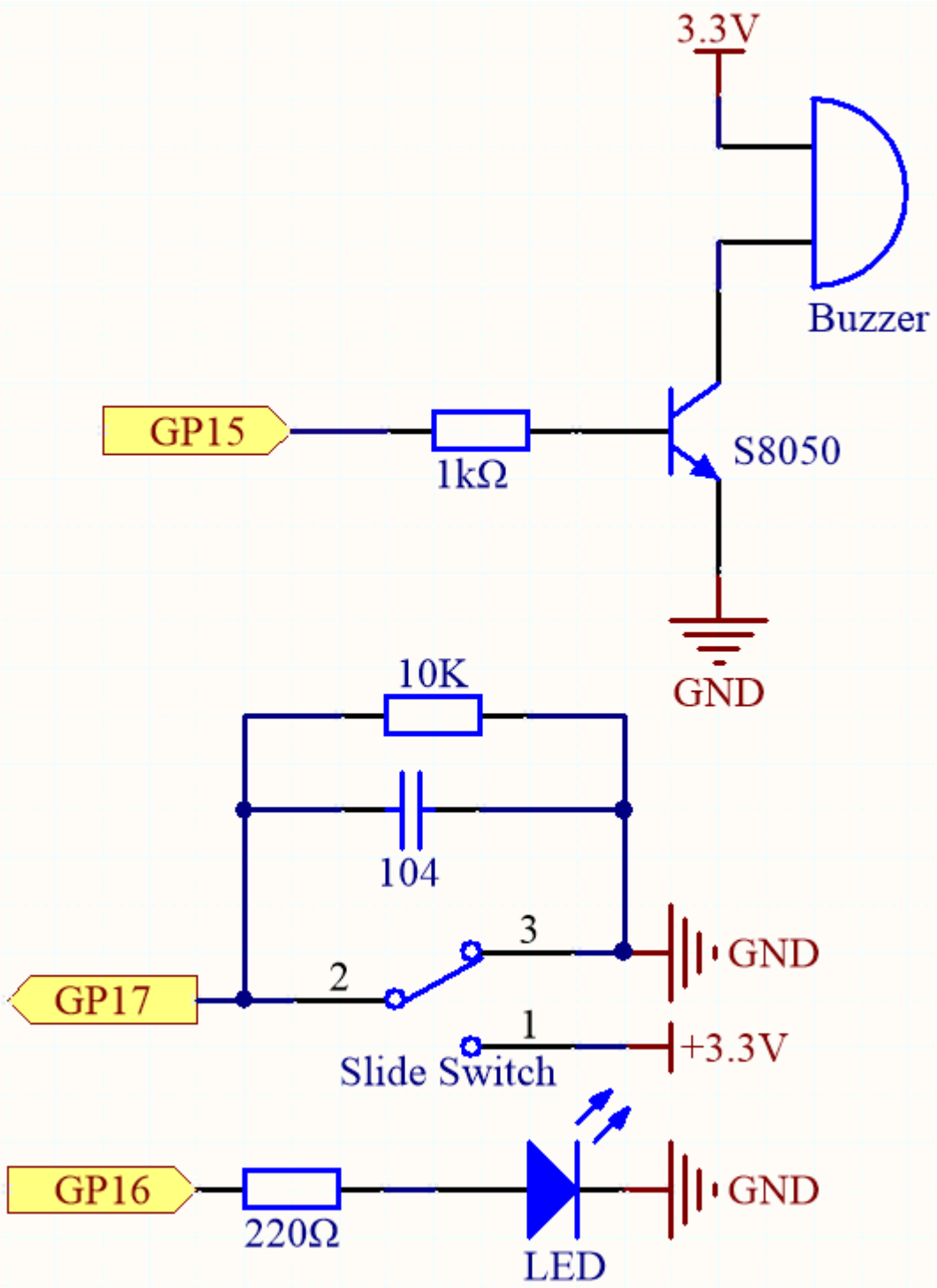
全てを一つのキットで購入すると便利です。リンクはこちら：

名前	キットの内容	リンク
ケプラーキット	450+ 点	

以下のリンクから個別にも購入できます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*LED*	1	
6	**トランジスタ**	1(S8050)	
7	**抵抗器**	3(1K Ω, 220 Ω, 10K Ω)	
8	パッシブ **ブザー**	1	
9	**コンデンサ**	1(104)	
10	**スライドスイッチ**	1	

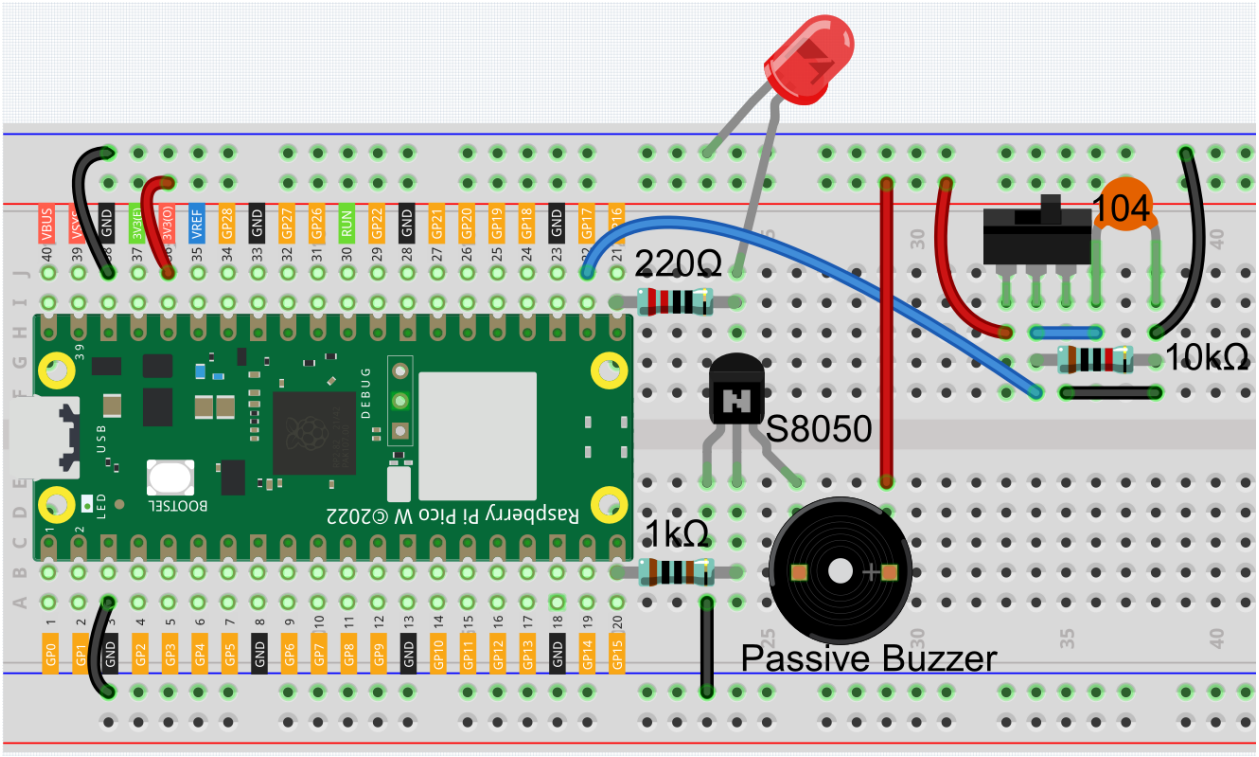
回路図



- GP17 はスライダーの中央ピンに接続されており、10K の抵抗と一緒にキャパシタ（フィルタ）が GND に並列接続されています。これにより、スライダーは左右に切り替えられたときに安定した高いまたは低いレベルを出力します。
- GP15 が高いとすぐに、NPN トランジスタが導通し、パッシブブザーが鳴り始めます。このパッシブブザーは、サイレン音を出すように周波数が徐々に増加するようにプログラムされています。

- LED は GP16 に接続されており、サイレンを模倣するように定期的に明るさを変えるようにプログラムされています。

配線



コード

注釈:

- kepler-kit-main/micropython フォルダ下の 7.3_alarm_siren_lamp.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、F5 キーを押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタを選択することを忘れずに。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

buzzer = machine.PWM(machine.Pin(15))
led = machine.PWM(machine.Pin(16))
led.freq(1000)
```

(次のページに続く)

(前のページからの続き)

```
switch = machine.Pin(17, machine.Pin.IN)

def noTone(pin):
    pin.duty_u16(0)

def tone(pin, frequency):
    pin.freq(frequency)
    pin.duty_u16(30000)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def toggle(pin):
    global bell_flag
    bell_flag = not bell_flag
    print(bell_flag)
    if bell_flag:
        switch.irq(trigger=machine.Pin.IRQ_FALLING, handler=toggle)
    else:
        switch.irq(trigger=machine.Pin.IRQ_RISING, handler=toggle)

bell_flag = False
switch.irq(trigger=machine.Pin.IRQ_RISING, handler=toggle)

while True:
    if bell_flag == True:
        for i in range(0, 100, 2):
            led.duty_u16(int(interval_mapping(i, 0, 100, 0, 65535)))
            tone(buzzer, int(interval_mapping(i, 0, 100, 130, 800)))
            time.sleep_ms(10)
    else:
        noTone(buzzer)
        led.duty_u16(0)
```

プログラムが動作している状態で、スライドスイッチを左に切り替えると（あなたのスライドスイッチの配線によっては右かもしれません）、ブザーが段階的に警告音を発し、LED の明るさもそれに応じて変わります。スライドスイッチを右に切り替えると、ブザーと LED は動作を停止します。

4.45 7.4 乗客カウンター

大型ショッピングモール、ショッピングセンター、チェーンストア、空港、駅、美術館、展示会場などの公共の場所で、乗客の流れは欠かせないデータです。

例えば、空港や駅では、人々の数を厳密に制御して、安全と円滑な流れを確保する必要があります。また、ショッピングセンターやチェーンストアでは、より多くの訪問者がいる時間帯や、各ユーザーが生成できる注文数などを把握することが可能です。その結果、人々の消費習慣を分析し、売上を向上させることができます。

乗客カウンターは、これらの公共の場所の運営を理解し、効率的に運営を整えるのに役立ちます。

このプロジェクトでは、PIR センサーと 4 桁の 7 セグメントディスプレイを使用して、シンプルな乗客カウンターを作成します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

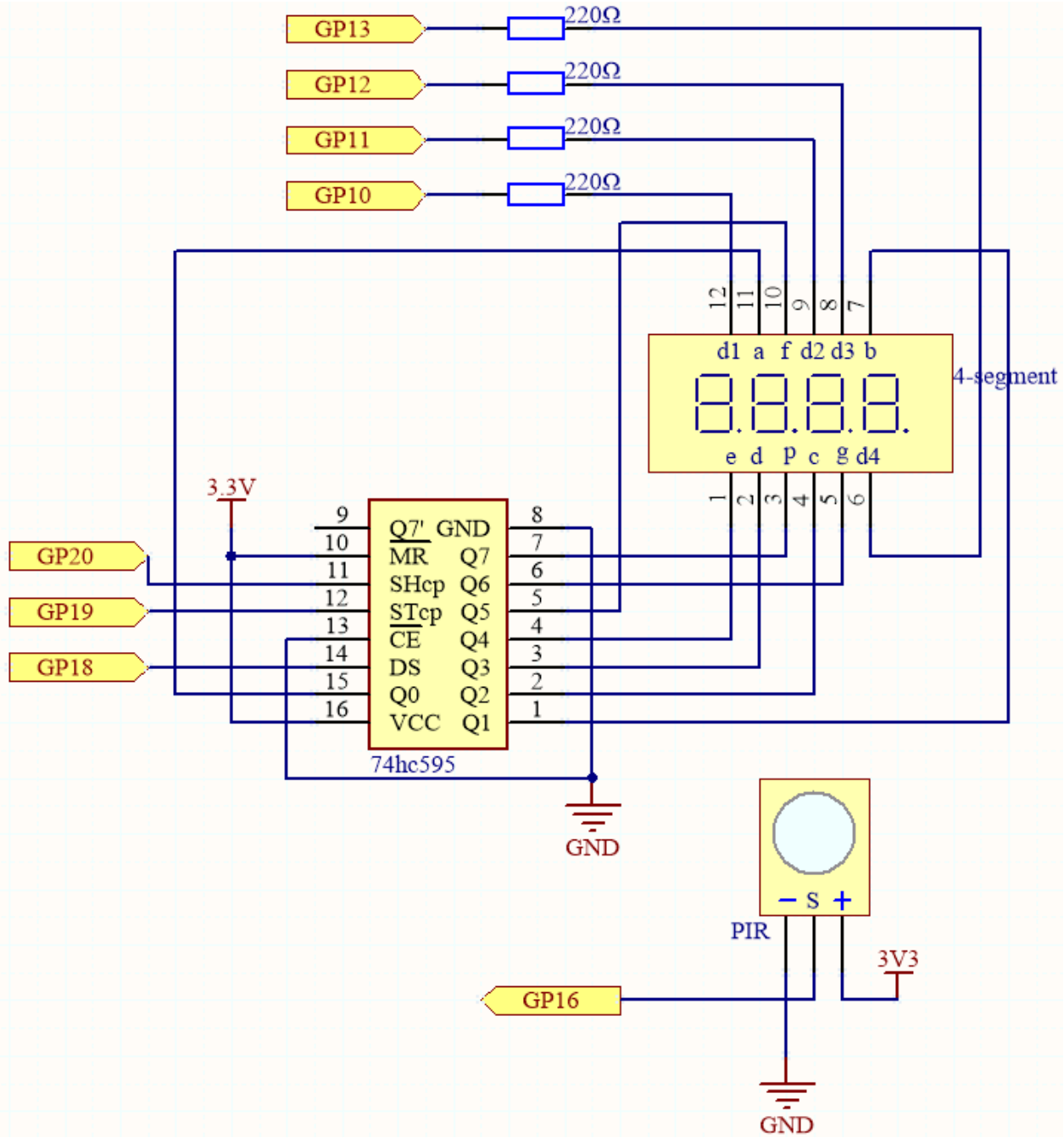
全体のキットを購入するのが確かに便利です、リンクはこちらです：

名前	キット内容	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

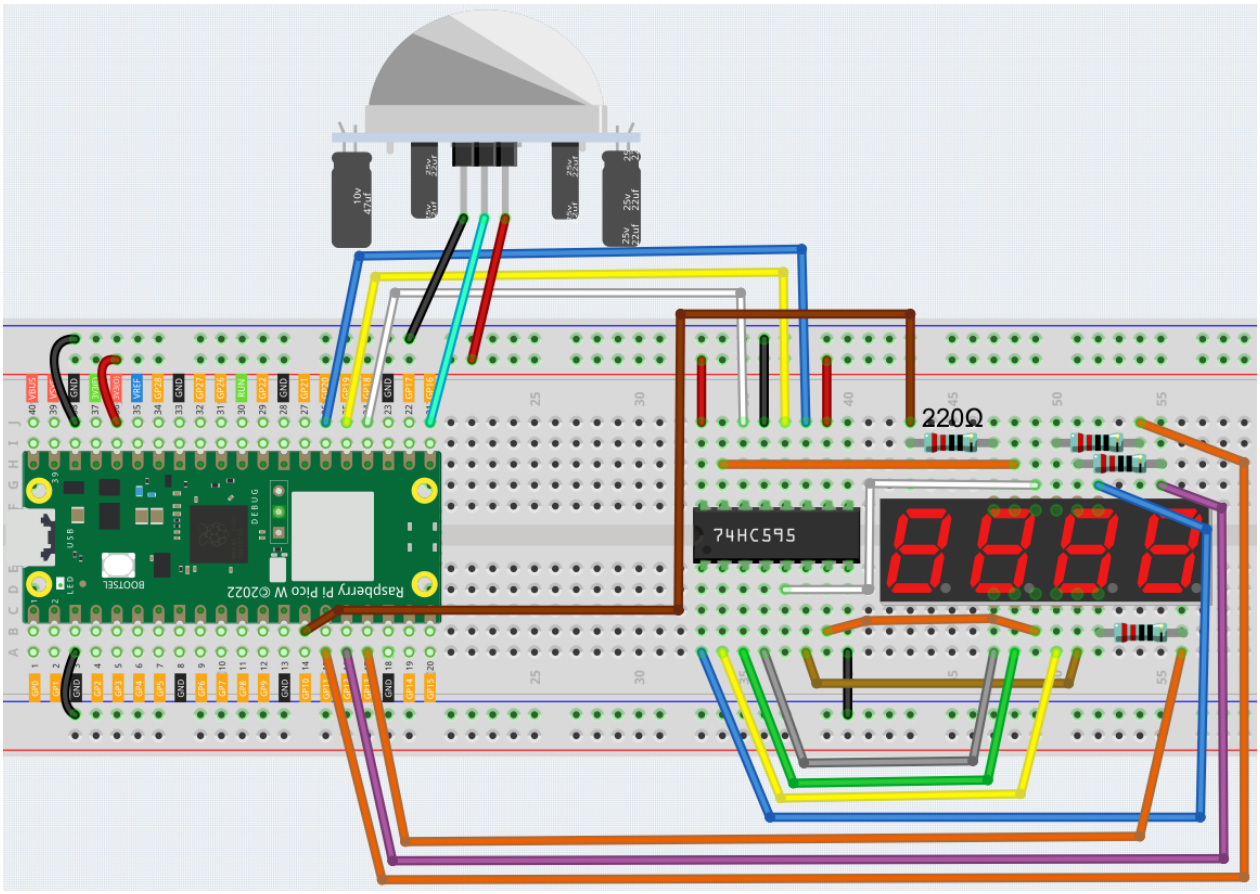
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(220 Ω)	
6	*4* **桁の** *7* **セグメントディスプレイ**	1	
7	*74HC595*	1	
8	*PIR* **モーションセンサーモジュール**	1	

回路図



- この回路は、 *5.3* **時間カウンター** を基にして、PIR モジュールが追加されています。
- 誰かが通過すると、PIR は約 2.8 秒間の高い信号を送ります。
- PIR モジュールには二つの可変抵抗があります：一つは感度を調整し、もう一つは検出距離を調整します。PIR モジュールをより効果的に動作させるためには、これらの両方を反時計回りに最後まで回す必要があります。

配線



コード

注釈:

- kepler-kit-main/micropython パスの下にある 7.4_passenger_counter.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 キーを押して実行してください。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」インタープリターをクリックすることを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

pir_sensor = machine.Pin(16, machine.Pin.IN)

SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
srclk = machine.Pin(20,machine.Pin.OUT)

placePin = []
pin = [10,13,12,11]
for i in range(4):
    placePin.append(None)
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)

count = 0

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

def clearDisplay():
    hc595_shift(0x00)

def hc595_shift(dat):
    rclk.low()
```

(次のページに続く)

(前のページからの続き)

```
    time.sleep_us(200)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_us(200)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_us(200)
        srclk.high()
        time.sleep_us(200)
    time.sleep_us(200)
    rclk.high()

def motion_detected(pin):
    global count
    count = count+1

pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=motion_detected)

while True:
    #print(count)

    pickDigit(0)
    hc595_shift(SEGCODE[count%10])

    pickDigit(1)
    hc595_shift(SEGCODE[count%100//10])

    pickDigit(2)
    hc595_shift(SEGCODE[count%1000//100])

    pickDigit(3)
    hc595_shift(SEGCODE[count%10000//1000])
```

コードが実行されたとき、PIR モジュールの前を誰かが通過すると、4 桁の 7 セグメントディスプレイ上の数字が 1 つ増加します。

4.46 7.5 GAME - 10 秒ゲーム

集中力を試すために、次に私に続いてゲームデバイスを作りましょう。傾きスイッチと棒を接続して魔法の杖を作成します。この杖を振ると、4 桁のセグメントディスプレイがカウントを始め、再度振るとカウントが停止します。勝つためには、表示されるカウントを **10.00** に保つ必要があります。友達とこのゲームで時間の魔法使いが誰かを見つけることができます。

必要な部品

このプロジェクトで必要な部品は以下のとおりです。

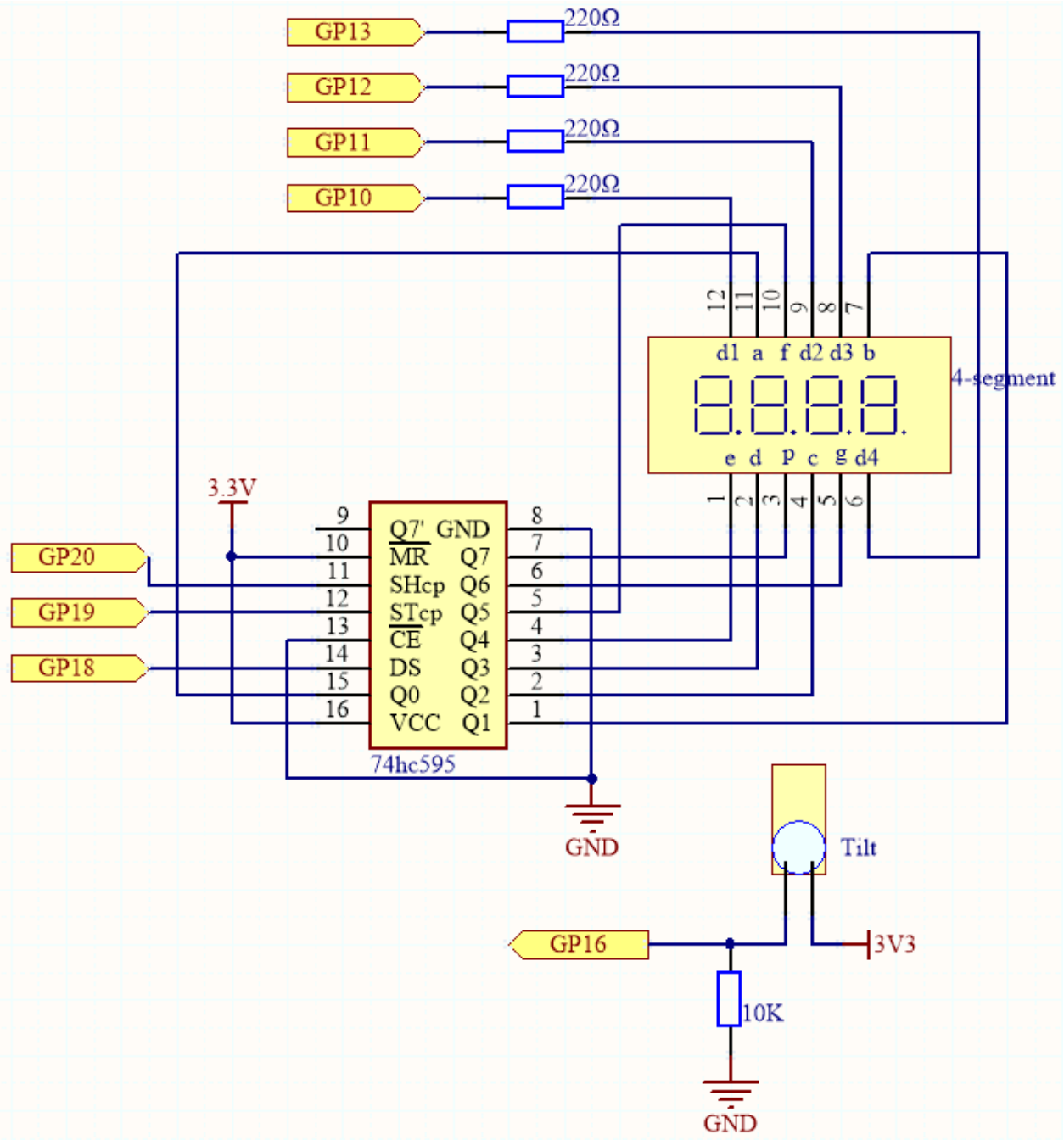
一式をまとめて購入する方が確実に便利です、リンクはこちら：

名前	このキットのアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個々に購入することもできます。

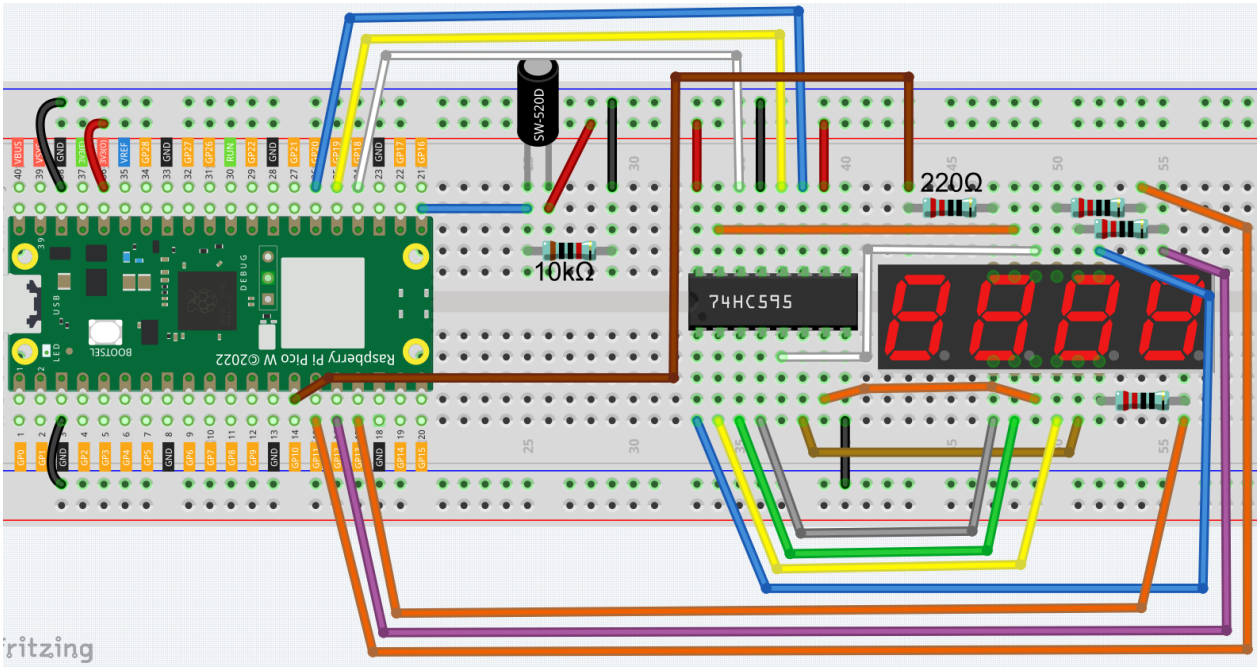
SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	5（4-220 Ω、1-10K Ω）	
6	*4* **桁の** *7* **セグメントディスプレイ**	1	
7	*74HC595*	1	
8	**傾斜スイッチ**	1	

回路図



- この回路は、 *5.3* **時間カウンター** に傾きスイッチを追加したものです。
- GP16 は、傾きスイッチが垂直のときに高く、傾いたときに低くなります。

配線図



コード

注釈:

- kepler-kit-main/micropython のパス下にある 7.5_game_10_second.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか F5 キーを押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターをクリックすることを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time

SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
srclk = machine.Pin(20,machine.Pin.OUT)

placePin = []
pin = [10,13,12,11]
for i in range(4):
    placePin.append(None)
```

(次のページに続く)

(前のページからの続き)

```
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

def clearDisplay():
    hc595_shift(0x00)

def hc595_shift(dat):
    rclk.low()
    time.sleep_us(200)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_us(200)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_us(200)
        srclk.high()
        time.sleep_us(200)
    time.sleep_us(200)
    rclk.high()
    #time.sleep_us(200)

def display(num):

    pickDigit(0)
    hc595_shift(SEGCODE[num%10])

    pickDigit(1)
    hc595_shift(SEGCODE[num%100//10])

    pickDigit(2)
    hc595_shift(SEGCODE[num%1000//100]+0x80)

    pickDigit(3)
    hc595_shift(SEGCODE[num%10000//1000])
```

(次のページに続く)

(前のページからの続き)

```
tilt_switch = machine.Pin(16,machine.Pin.IN)

count_flag = False

def shake(pin):
    global timeStart,count_flag
    count_flag = not count_flag
    if count_flag == True:
        timeStart = time.ticks_ms()

tilt_switch.irq(trigger=machine.Pin.IRQ_RISING, handler=shake)

count = 0
while True:
    if count_flag == True:
        count = int((time.ticks_ms()-timeStart)/10)
    display(count)
```

魔法の杖を振ると、4 桁の 7 セグメントディスプレイがカウントを開始し、再度振るとカウントが停止します。表示されたカウントが 10.00 になった場合、あなたの勝ちです。もう一度振るとゲームが続きます。

4.47 7.6 交通信号機

交通信号機 は、道路交差点や横断歩道、その他の場所で交通の流れを制御するための信号装置です。

交通信号は、 ウィーン道路標識および信号に関する条約 によって標準化されています。三つの標準色の LED を交互に点灯させて、交通の優先権を与えます。

- **赤信号** : 点滅する赤い光を見たら、停止標識と同等として停止すべきです。
- **黄信号** : 赤に変わる前の警告信号です。黄信号の解釈は国や地域によって異なります。
- **緑信号** : 指示された方向への交通を許可します。

このプロジェクトでは、交通信号機の変化を実装するために 3 色の LED と、各交通状態の時間を表示するための 4 桁 7 セグメントディスプレイを使用します。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

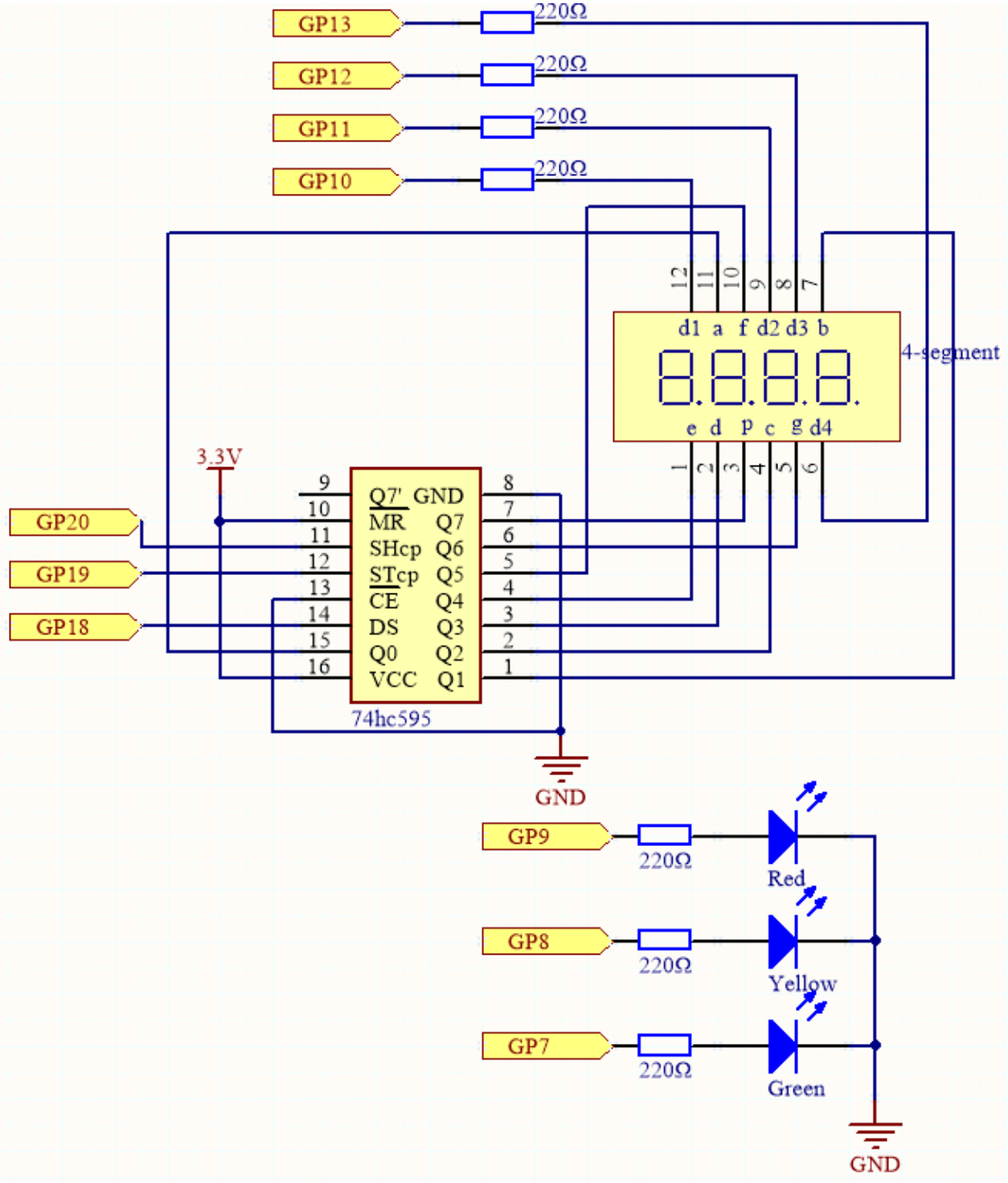
便宜上、全体のキットを購入することもできます。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別に購入することもできます。

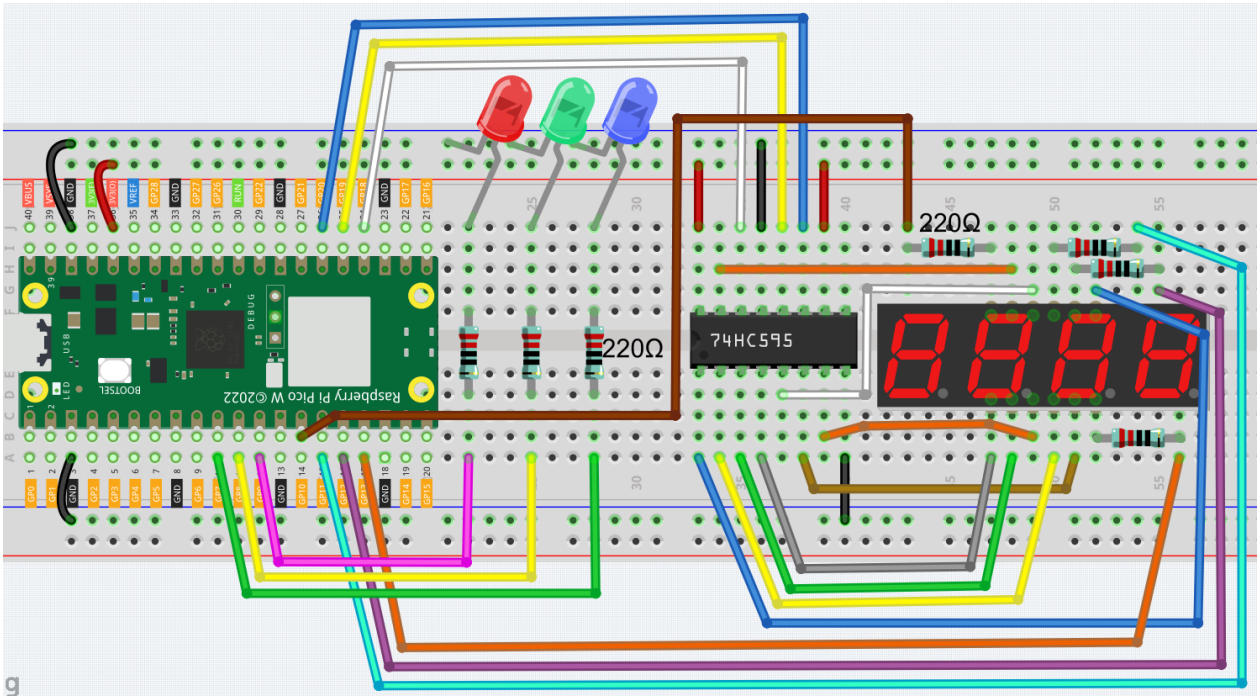
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	7(220 Ω)	
6	*4* **桁の** *7* **セグメントディスプレイ**	1	
7	*74HC595*	1	
8	*LED*	1	

回路図



- この回路は、 *5.3* **時間カウンター** を基に、3 つの LED が追加されています。
- 3 つの赤、黄、緑の LED はそれぞれ GP7~GP9 に接続されています。

配線図



コード

注釈:

- kepler-kit-main/micropython のパス下の 7.6_traffic_light.py ファイルを開くか、このコードを Thonny にコピーしてから、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。
- 画面の右下隅にある「MicroPython（Raspberry Pi Pico）」のインタープリタをクリックすることを忘れずに。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time
from machine import Timer

# [Green, Yellow, Red]
lightTime=[30, 5, 30]

# display
SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
```

(次のページに続く)

(前のページからの続き)

```
srclk = machine.Pin(20,machine.Pin.OUT)

placePin = []
pin = [10,13,12,11]
for i in range(4):
    placePin.append(None)
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

def clearDisplay():
    hc595_shift(0x00)

def hc595_shift(dat):
    rclk.low()
    time.sleep_us(200)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_us(200)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_us(200)
        srclk.high()
        time.sleep_us(200)
    time.sleep_us(200)
    rclk.high()

def display(num):

    pickDigit(0)
    hc595_shift(SEGCODE[num%10])

    pickDigit(1)
    hc595_shift(SEGCODE[num%100//10])

    pickDigit(2)
```

(次のページに続く)

(前のページからの続き)

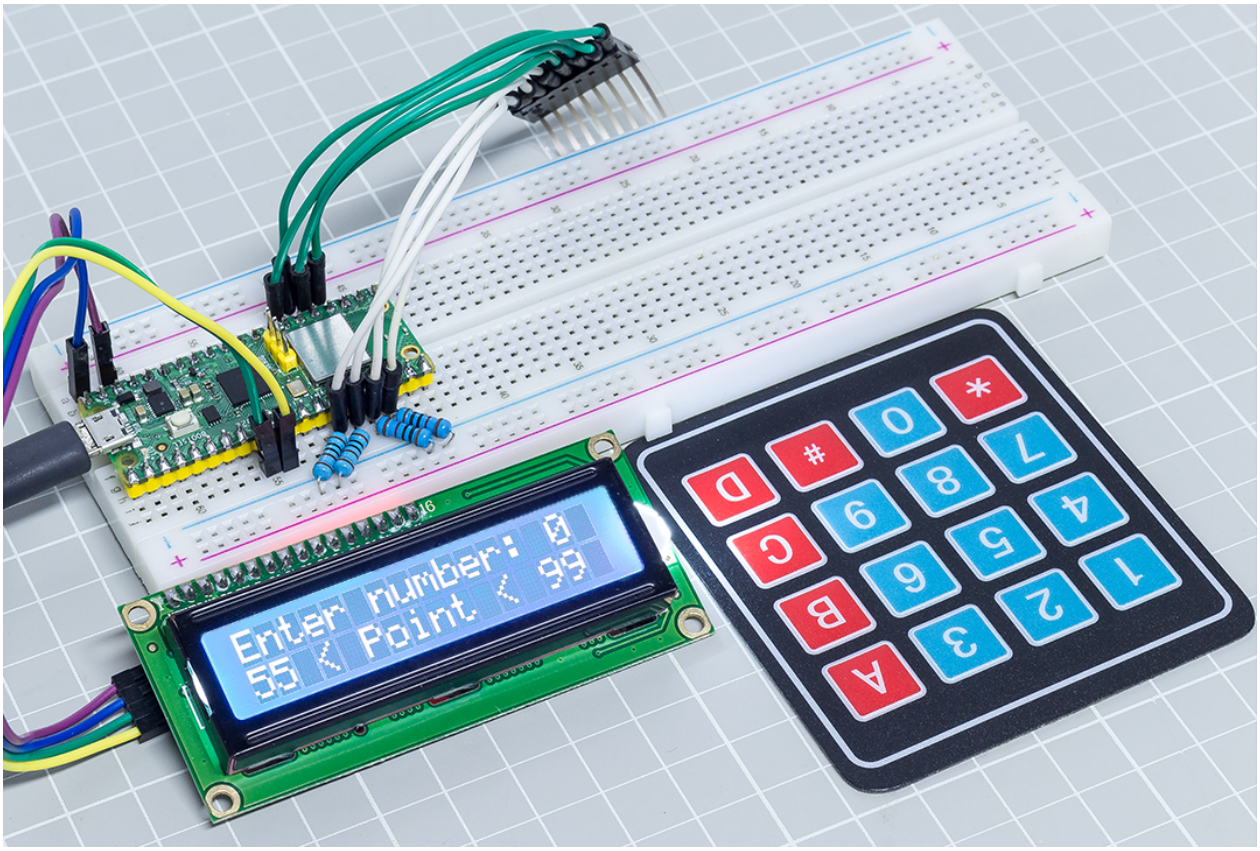
```
    hc595_shift(SEGCODE[num%1000//100])

    pickDigit(3)
    hc595_shift(SEGCODE[num%10000//1000])

# led
# 9Red, 8Yellow,7Green
pin = [7,8,9]
led=[]
for i in range(3):
    led.append(None)
    led[i] = machine.Pin(pin[i], machine.Pin.OUT)

def lightup(state):
    for i in range(3):
        led[i].value(0)
    led[state].value(1)

# timer
counter = 0
color_state= 0

def time_count(ev):
    global counter, color_state
    counter -= 1
    if counter <= 0:
        color_state = (color_state+1) % 3
        counter = lightTime[color_state]

tim = Timer(period=1000, mode=Timer.PERIODIC, callback=time_count)


while True:
    display(counter)
    lightup(color_state)
```

コードが実行されると、緑の LED が 30 秒間点灯し、黄色の LED が 5 秒間点灯し、赤の LED が 30 秒間点灯します。

4.48 7.7 数字当てゲーム

数字当ては楽しいパーティーゲームで、友達と一緒に 0〜99 の数字を入力します。各プレイヤーが数字を入力する度に、範囲が狭まり、誰かが正解するとそのプレイヤーは敗北し、罰を受けます。

例えば、運の良い数字が 51 で、プレイヤーはそれを見ることができない場合、プレイヤー 1 が 50 を入力すると、プロンプトは 50 - 99 に変わります。プレイヤー 2 が 70 を入力すると、範囲は 50 - 70 に変わります。プレイヤー 3 が 51 を入力した場合、そのプレイヤーは不運です。このケースでは、数字はキーパッドを通じて入力され、結果は LCD スクリーンに表示されます。



必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

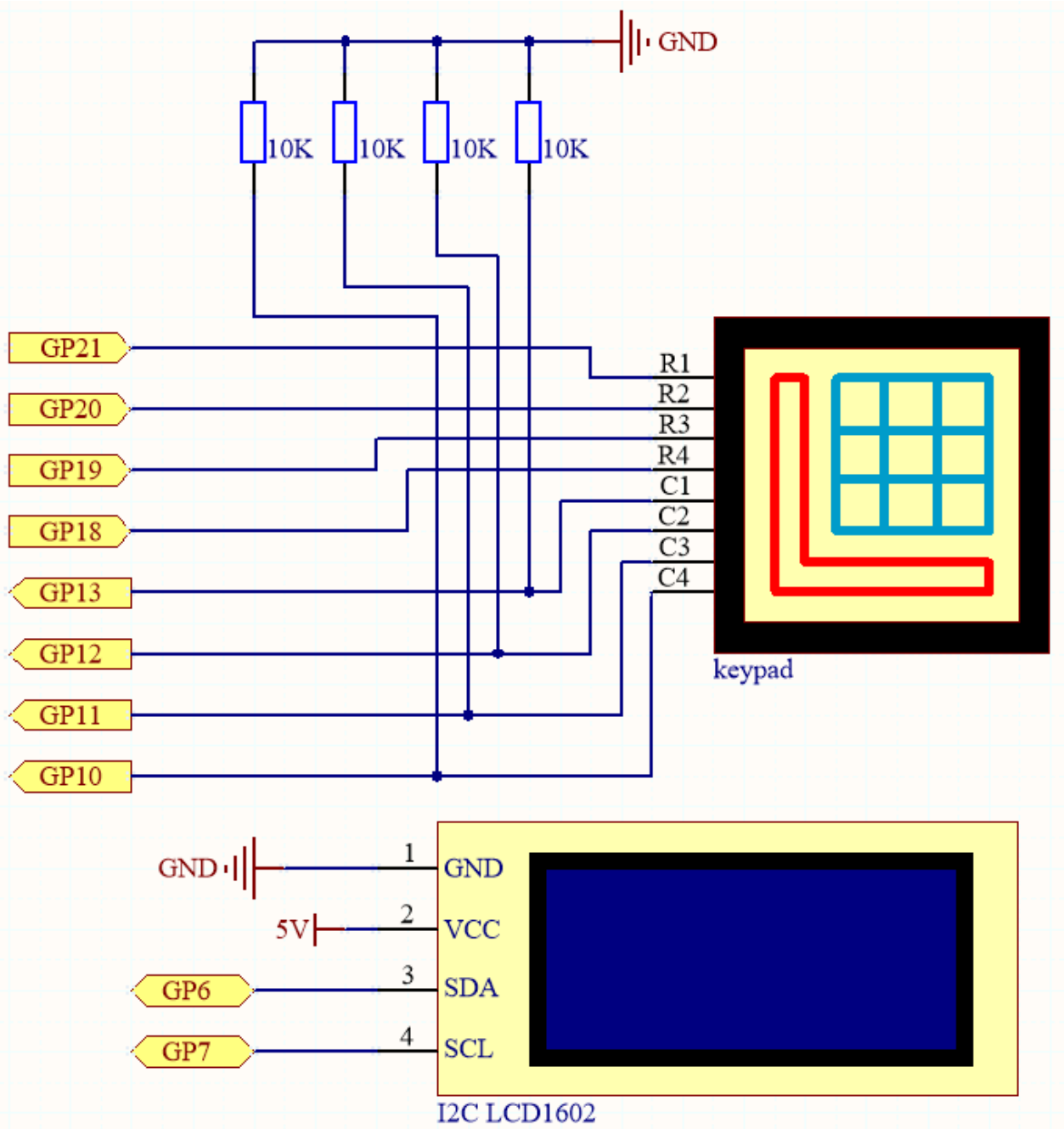
一式を購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

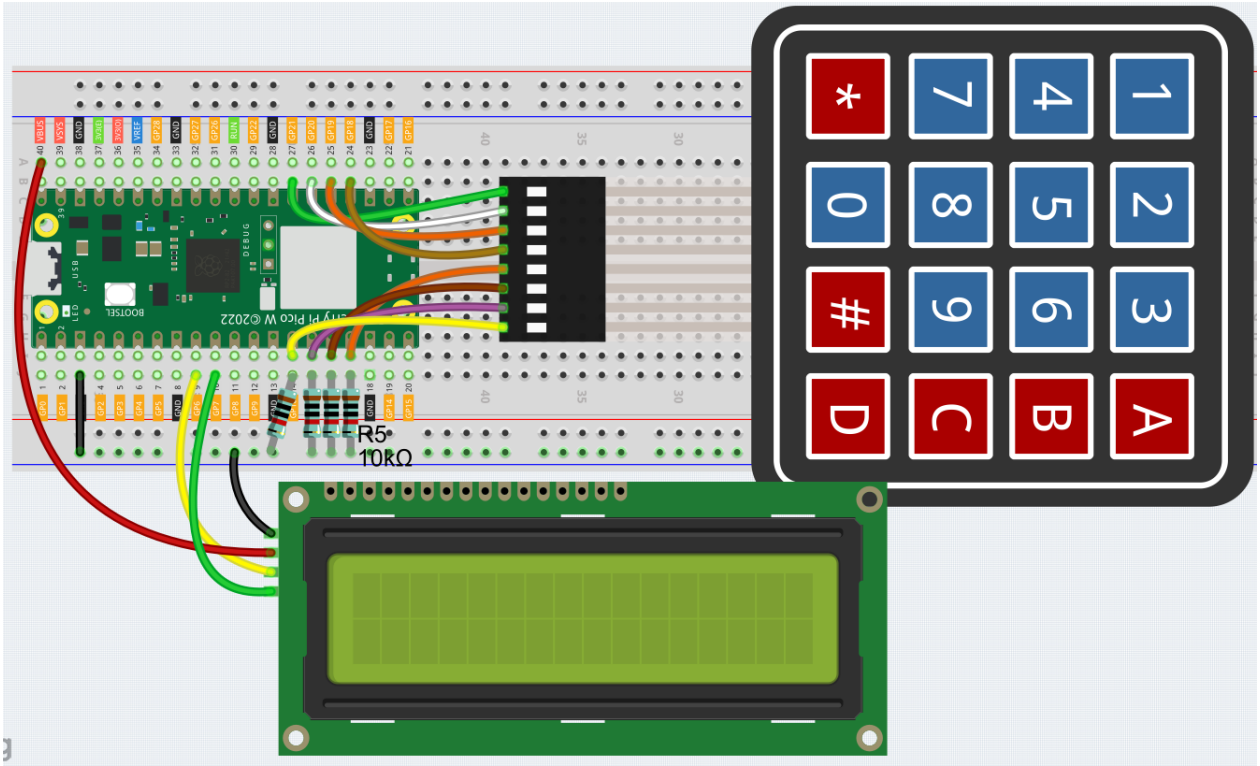
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(10K Ω)	
6	*4x4* **キーパッド**	1	
7	*I2C LCD1602*	1	

回路図



この回路は、 *4.2 4x4* **キーパッド** を基にしており、押されたキーを表示するための I2C LCD1602 が追加されています。

配線



配線を簡単にするために、上記の図では、マトリックスキーボードの列行と 10K の抵抗器が同時に G10～G13 の穴に挿入されています。

コード

注釈:

- kepler-kit-main/micropython のパスの下にある 7.7_game_guess_number.py ファイルを開くか、このコードを Thonny にコピーしてから「Run Current Script」をクリックするか、単純に F5 キーを押して実行します。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタープリタをクリックするのを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。

```
from lcd1602 import LCD
import machine
import time
import urandom


# keypad function
```

(次のページに続く)

(前のページからの続き)

```
characters = [["1","2","3","A"],["4","5","6","B"],["7","8","9","C"],["*","0","#","D"]]

pin = [21,20,19,18]
row = []
for i in range(4):
    row.append(None)
    row[i] = machine.Pin(pin[i], machine.Pin.OUT)

pin = [13,12,11,10]
col = []
for i in range(4):
    col.append(None)
    col[i] = machine.Pin(pin[i], machine.Pin.IN)

def readKey():
    key = []
    for i in range(4):
        row[i].high()
        for j in range(4):
            if(col[j].value() == 1):
                key.append(characters[i][j])
        row[i].low()
    if key == [] :
        return None
    else:
        return key

# init/reset number
# reset the result as False for lcd show
def init_new_value():
global pointValue,upper,count,lower
pointValue = int(urandom.uniform(0, 99))
print(pointValue)
upper = 99
lower = 0
count = 0
return False
```

(次のページに続く)

(前のページからの続き)

```
# lcd show message
# If target, show game over.
# If not target, or not detected, show guess number.
def lcd_show(result):
    lcd.clear()
    if result == True:
        string ="GAME OVER!\n"
        string +="Point is "+ str(pointValue)
    else :
        string ="Enter number: " + str(count) +"\n"
        string += str(lower)+ " < Point < " + str(upper)
    lcd.message(string)
    return

# detect number & reflesh show message
# if not target, reflesh number (upper or lower) and return False
# if target, return True
def number_processing():
    global upper,count,lower
    if count > pointValue:
        if count < upper:
            upper = count
    elif count < pointValue:
        if count > lower:
            lower = count
    elif count == pointValue:
        return True
    count = 0
    return False

## start
lcd = LCD()
string = "Welcome!\n"
string = "Press A to Start!"
lcd.message(string)
result=init_new_value()

# read key & display
last_key = None
```

(次のページに続く)

(前のページからの続き)

```
while True:
    current_key = readKey()
    if current_key == last_key:
        continue
    last_key = current_key
    if current_key != None:
        # print(current_key)
        if current_key ==["A"]: # reset number
            result=init_new_value()
        elif current_key==["D"]: # check
            result=number_processing()
        elif current_key[0] in list(["1","2","3","4","5","6","7","8","9","0"]) and count
↪< 10: #check validity & limit digits
            count = count * 10 + int(current_key[0])
        lcd_show(result) # show
    time.sleep(0.1)
```

- コードが実行された後、 A を押してゲームを開始します。ランダムな数字 point が生成されますが、LCD には表示されません。あなたがするべきことは、その数字を推測することです。
- 最終計算が終わるまで、最初の行の末尾に入力した数字が表示されます（比較を開始するには D を押します）。
- point の数字の範囲が 2 行目に表示されます。範囲内の数字を入力する必要があります。
- 数字を入力すると、範囲が狭まります。もし幸運な数字（または不運な数字）を当てた場合は、GAME OVER! が表示されます。

注釈: コードと配線が問題ないが、LCD がまだ内容を表示しない場合は、裏側のポテンショメータを回してコントラストを上げることができます。

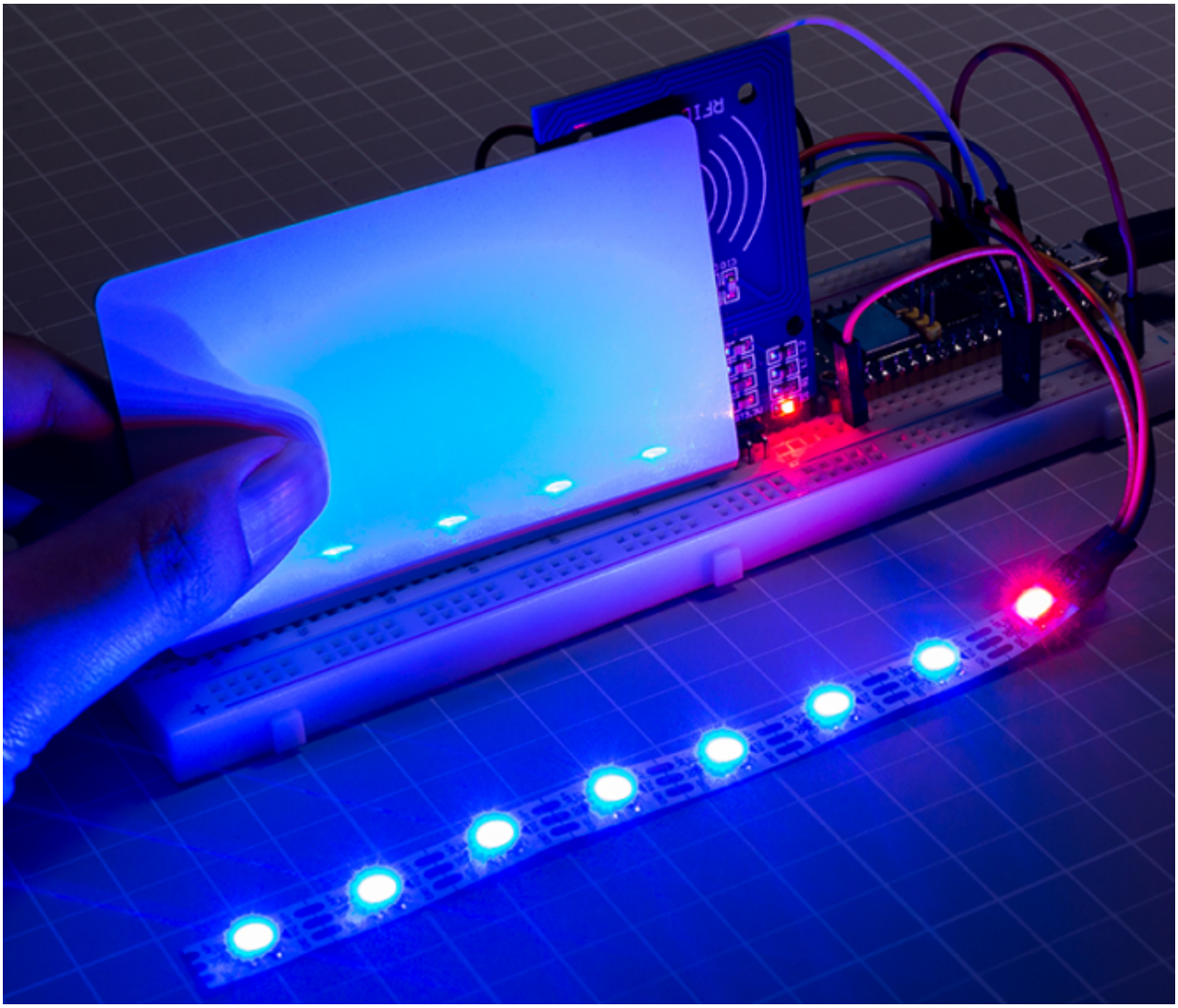
4.49 7.8 RFID 音楽プレーヤー

前回のプロジェクト、 で、MFRC522 モジュールを使ってカード（またはキー）に最大 48 文字の情報を書き込むことができることを学びました。この情報には、キーと ID 情報の他にも、楽譜が含まれます。

例えば、 EEFGGFEDCCDEEDD EEFGGFEDCCDEDCC と書き込んだ場合、カード（またはキー）を再度読み取ったときにブザーが音楽を再生します。WS2812 も装備することで、素晴らしいエフェクトを表示することもできます。

インターネットでさまざまな楽譜を見つけたり、自分自身で音楽を作曲して、それをカード（またはキー）に保存

し、友達と共有することもできます！



必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

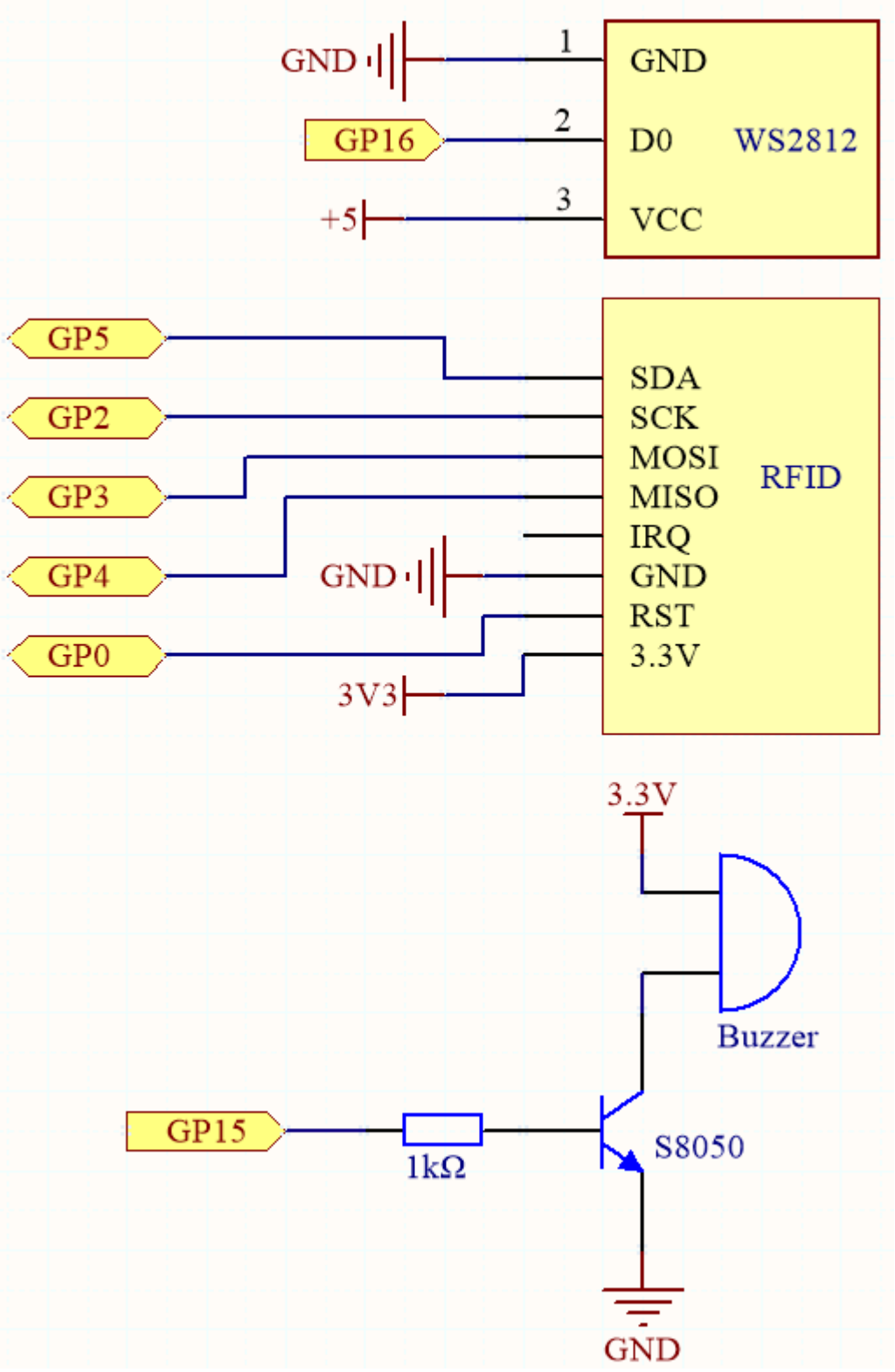
一式をまとめて購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

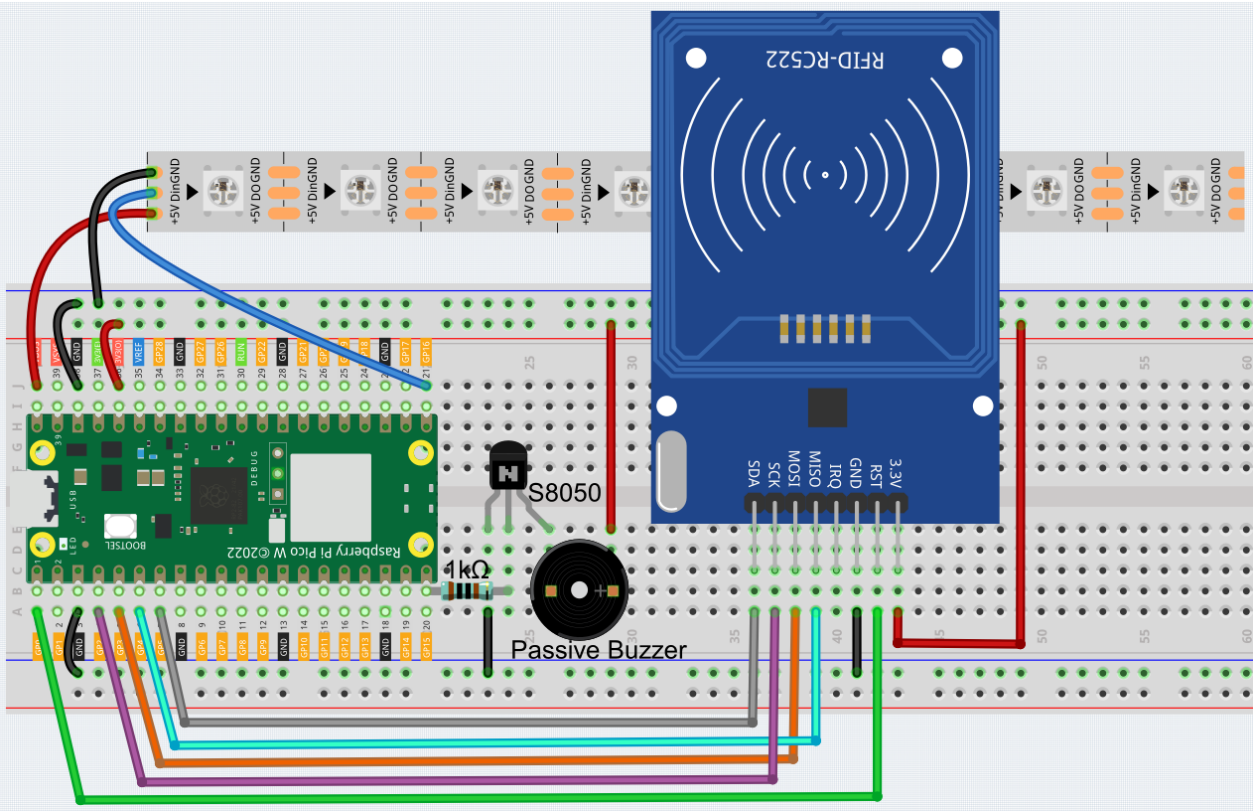
以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	複数	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	パッシブ **ブザー**	1	
8	*MFRC522* **モジュール**	1	
9	*WS2812 RGB 8 LED* **ストリップ**	1	

回路図


GND
1
GND
GP16
2
D0
WS2812
+5
3
VCC
GP5
GP2
GP3
GP4
GP0
SDA
SCK
MOSI
MISO
IRQ
GND
RST
3.3V
RFID
GND
3V3
3.3V
Buzzer
GP15
1kΩ
S8050
GND

配線図



コード

1. kepler-kit-main/micropython フォルダ内の 6.5_rfid_write.py ファイルを開いて、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。

2. 実行後、シェルに EEFGGFEDCCDEEDD EEFGGFEDCCDEDCC と入力し、カード（またはキー）を MFRC522 モジュールに近づけます。これで、歓喜の歌の楽譜が保存されます。

3. kepler-kit-main/micropython フォルダ内の 7.8_rfid_music_player.py ファイルを開くか、このコードを Thonny にコピーして、「Run Current Script」をクリックするか、単に F5 キーを押して実行します。

```
from mfrc522 import SimpleMFRC522
import machine
import time
from ws2812 import WS2812
import urandom

# ws2812
ws = WS2812(machine.Pin(16), 8)

# mfrc522
reader = SimpleMFRC522(spi_id=0, sck=2, miso=4, mosi=3, cs=5, rst=0)
```

(次のページに続く)

(前のページからの続き)

```
# ブザー
NOTE_C4 = 262
NOTE_D4 = 294
NOTE_E4 = 330
NOTE_F4 = 349
NOTE_G4 = 392
NOTE_A4 = 440
NOTE_B4 = 494
NOTE_C5 = 523

buzzer = machine.PWM(machine.Pin(15))
note = [NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4, NOTE_G4, NOTE_A4, NOTE_B4, NOTE_C5]

def tone(pin, frequency, duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    time.sleep_ms(duration)
    pin.duty_u16(0)

# 明るくする
def lumi(index):
    for i in range(8):
        ws[i] = 0x0000FF
    ws[index] = 0xFF0000  # int(urandom.uniform(0, 0xFFFFFF))
    ws.write()

# テキストをインデックスにエンコード
words = ["C", "D", "E", "F", "G", "A", "B", "N"]
def take_text(text):
    string = text.replace(' ', '').upper()
    while len(string) > 0:
        index = words.index(string[0])
        tone(buzzer, note[index], 250)
        lumi(index)
        new_str = ""
        for i in range(0, len(string)):
            if i != 0:
                new_str = new_str + string[i]
```

(次のページに続く)

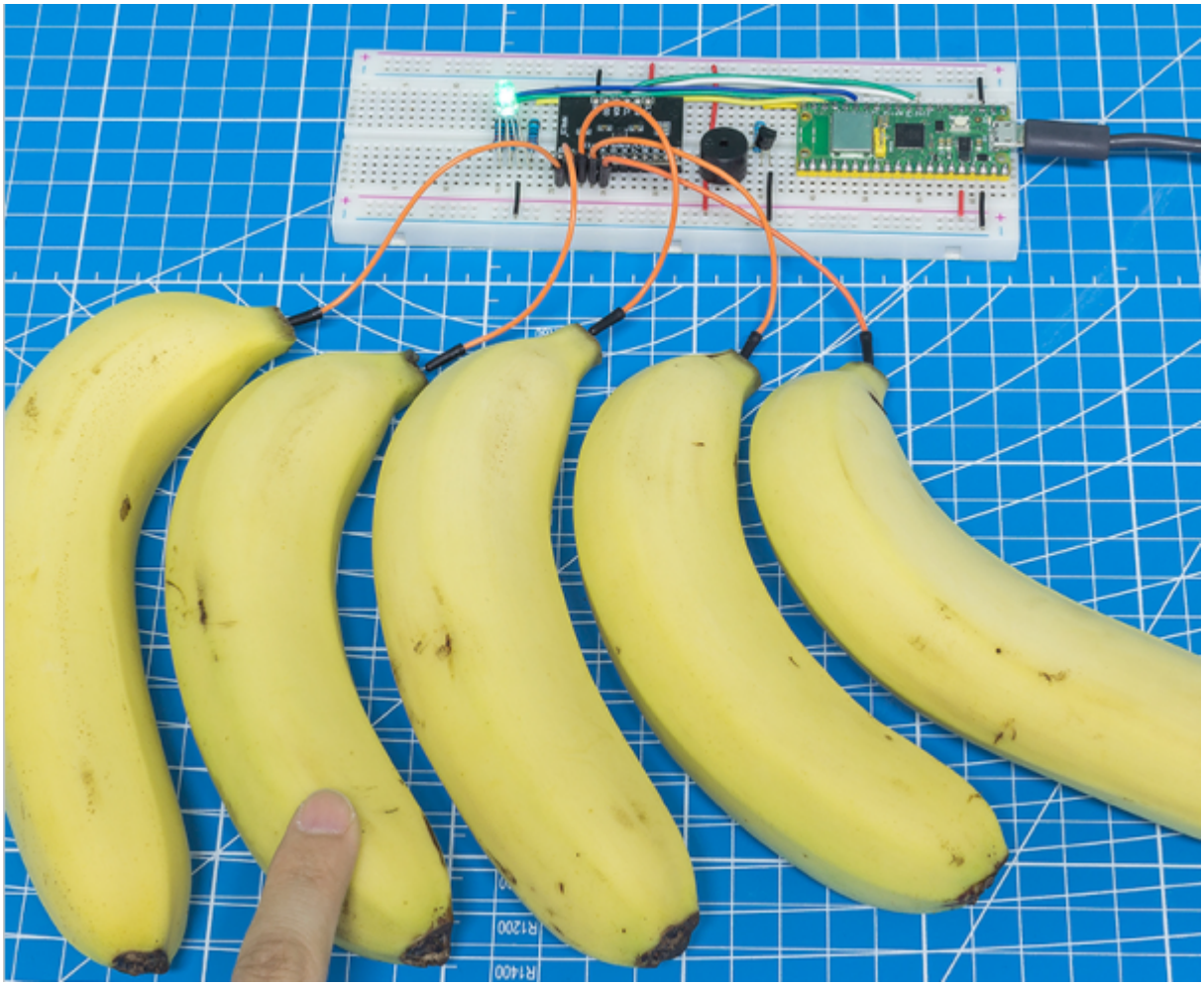
(前のページからの続き)

```
        string = new_str

# カードを読む
def read():
    print("Reading...Please place the card...")
    id, text = reader.read()
    print("ID: %s\nText: %s" % (id,text))
    take_text(text)

read()
```

4. カード（またはキー）を再度 MFRC522 モジュールに近づけると、ブザーがカード（またはキー）に保存された音楽を再生し、RGB ストリップがランダムな色で点灯します。

4.50 7.9 フルーツピアノ

電気伝導性は多くの金属物体や人体、さらには果物にも存在します。この性質を使って、ちょっとした楽しいプロジェクト、すなわちフルーツピアノを作成することができます。言い換えれば、私たちは触れるだけで音楽を演奏できるキーボードに果物を変えます。



必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

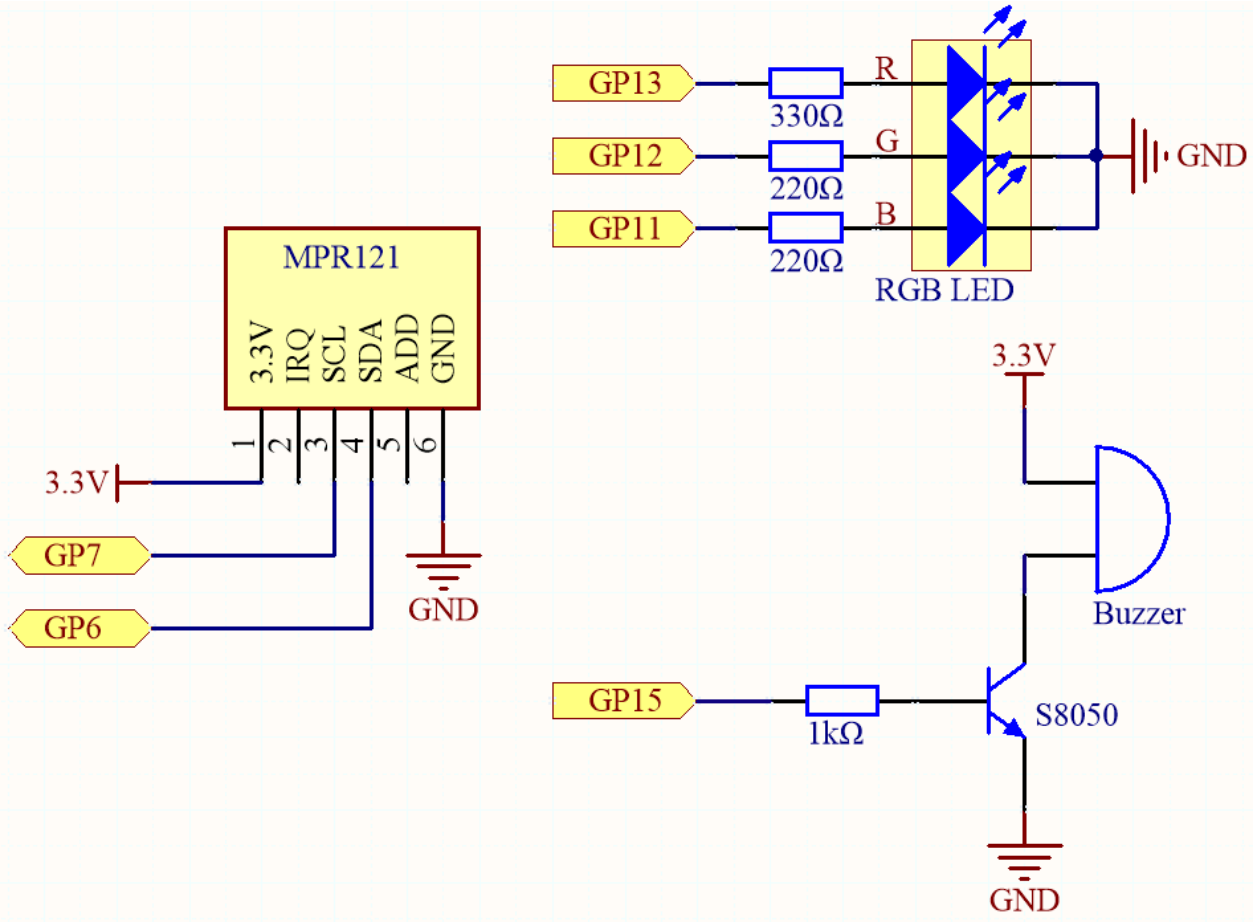
一式をまとめて買う方が便利です、こちらがそのリンクです：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

以下のリンクから個別にも購入できます。

品番	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	4(1-1KΩ, 1-330 Ω, 2-220 Ω)	
7	パッシブ **ブザー**	1	
8	*RGB LED*	1	
9	*MPR121* **モジュール**	1	

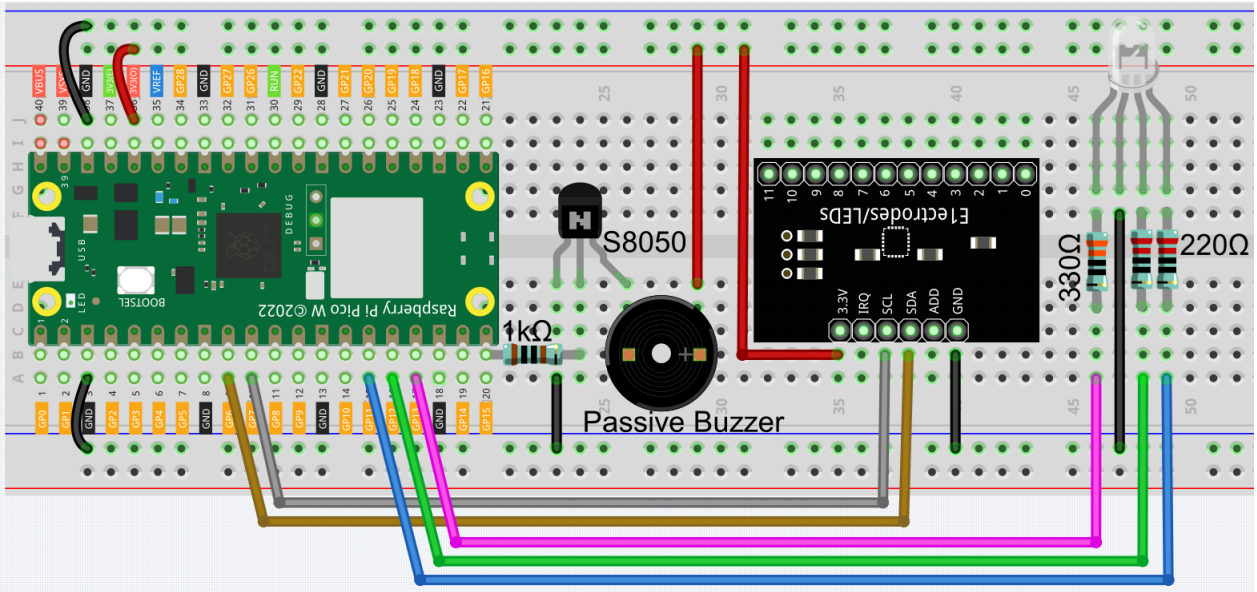
回路図



フルーツをピアノのキーに変えるには、MPR121 上の電極をフルーツ（例：バナナのハンドル）に接続する必要があります。

最初に、MPR121 は初期化され、各電極は現在の電荷に基づいて値を取得します。導体（人体など）が電極に触れると、電荷が移動して再調整されます。その結果、電極の値は初期値とは異なり、メインコントロールボードに触れられたことを知らせます。このプロセス中に、各電極の配線が安定していることを確認し、初期化時にその電荷がバランスするようにしてください。

配線図



コード

注釈:

- kepler-kit-main/micropython のパスの下で 7.9_fruit_piano.py ファイルを開くか、Thonny にこのコードをコピーして、「Run Current Script」をクリックするか、F5 キーを押して実行してください。
- 右下隅の「MicroPython（Raspberry Pi Pico）」インタプリタをクリックするのを忘れないでください。
- 詳しいチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- ここでは、 mpr121.py というライブラリを使用する必要があります。Pico W にアップロードされたかどうか確認してください。詳細なチュートリアルは、 *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from mpr121 import MPR121
from machine import Pin, I2C
import time
import urandom

# mpr121
i2c = I2C(1, sda=Pin(6), scl=Pin(7))
mpr = MPR121(i2c)

# ブザー
NOTE_A3 = 220
```

(次のページに続く)

(前のページからの続き)

```
NOTE_B3 = 247
NOTE_C4 = 262
NOTE_D4 = 294
NOTE_E4 = 330
NOTE_F4 = 349
NOTE_G4 = 392
NOTE_A4 = 440
NOTE_B4 = 494
NOTE_C5 = 523
NOTE_D5 = 587
NOTE_E5 = 659

buzzer = machine.PWM(machine.Pin(15))
note = [NOTE_A3, NOTE_B3, NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4, NOTE_G4, NOTE_A4, NOTE_B4,␣
↪NOTE_C5, NOTE_D5, NOTE_E5]

def tone(pin, frequency):
    pin.freq(frequency)
    pin.duty_u16(30000)

def noTone(pin):
    pin.duty_u16(0)

# RGB LED
red = machine.PWM(machine.Pin(13))
green = machine.PWM(machine.Pin(12))
blue = machine.PWM(machine.Pin(11))
red.freq(1000)
green.freq(1000)
blue.freq(1000)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def lightup():
    red.duty_u16(int(urandom.uniform(0, 65535)))
    green.duty_u16(int(urandom.uniform(0, 65535)))
    blue.duty_u16(int(urandom.uniform(0, 65535)))
```

(次のページに続く)

(前のページからの続き)

```
def dark():
    red.duty_u16(0)
    green.duty_u16(0)
    blue.duty_u16(0)

# メインプロジェクト
lastState = mpr.get_all_states()
touchMills = time.ticks_ms()
beat = 500

while True:
    currentState = mpr.get_all_states()
    if currentState != lastState:
        for i in range(12):
            if i in list(currentState) and not i in list(lastState):
                tone(buzzer, note[i])
                lightup()
                touchMills = time.ticks_ms()
    if time.ticks_diff(time.ticks_ms(), touchMills) >= beat or len(currentState) == 0:
        noTone(buzzer)
        dark()
    lastState = currentState
```

プログラムが動作する前に果物に触れないでください。初期化中に正確でない参照値を取得する可能性があります。プログラムが動作した後、果物に優しく触れると、ブザーが対応する音を鳴らし、RGB ライトがランダムに一回点滅します。

4.51 7.10 バックアップ支援

このプロジェクトでは、LED、ブザー、および超音波モジュールを使用して、バックアップ支援システムを作成します。これをリモートコントロールカーに取り付けて、ガレージに車をバックして入れる実際のプロセスをシミュレーションすることができます。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

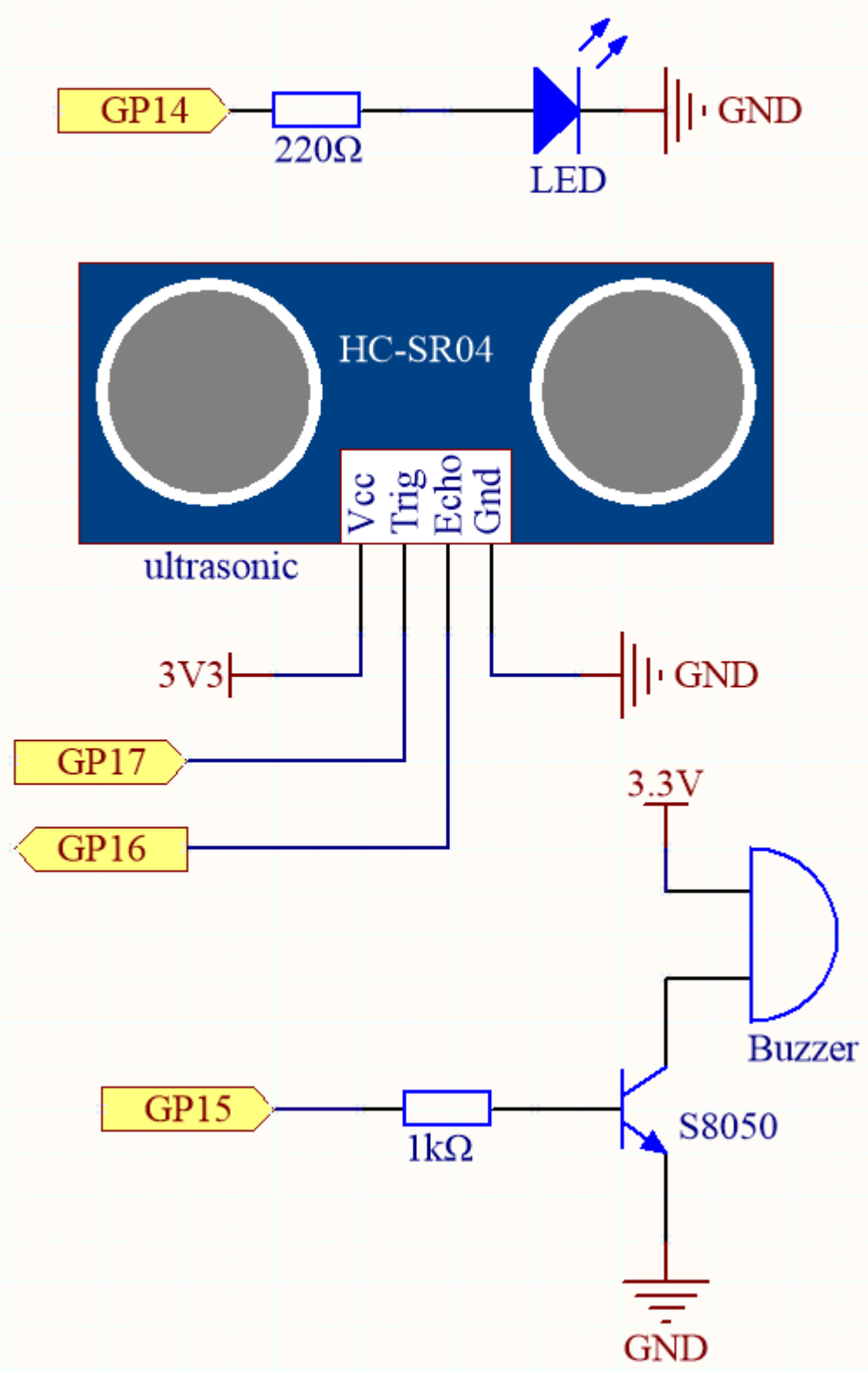
便利なのは、一式をまとめて購入することです。リンクはこちらです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

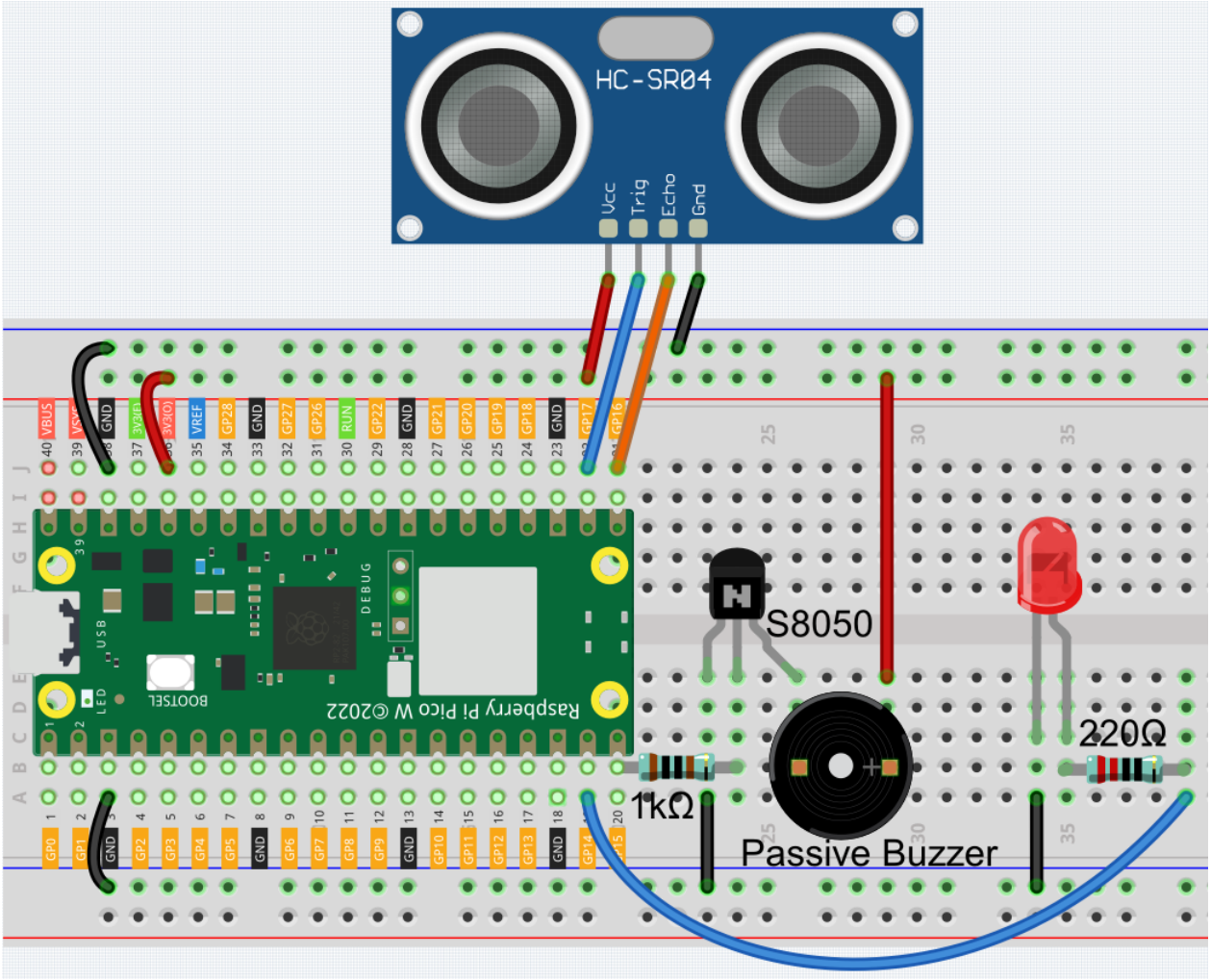
以下のリンクからそれぞれ個別に購入することも可能です。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	2(1K Ω、220 Ω)	
7	パッシブ **ブザー**	1	
8	*LED*	1	
9	**超音波モジュール**	1	

回路図


GP14
220Ω
LED
GND
HC-SR04
Vcc
Trig
Echo
Gnd
ultrasonic
3V3
GND
GP17
GP16
3.3V
Buzzer
GP15
1kΩ
S8050
GND

配線



コード

注釈:

- kepler-kit-main/micropython のパスの下で 7.10_reversing_aid.py ファイルを開くか、このコードを Thonny にコピーして、"Run Current Script"をクリックするか、単に F5 キーを押して実行してください。
- 右下隅の"MicroPython（Raspberry Pi Pico）"インタープリターをクリックするのを忘れないでください。
- 詳細なチュートリアルについては、 **コードを直接開いて実行する** を参照してください。

```
import machine
import time
import _thread

buzzer = machine.Pin(15, machine.Pin.OUT)
```

(次のページに続く)

(前のページからの続き)

```
led = machine.Pin(14, machine.Pin.OUT)

TRIG = machine.Pin(17, machine.Pin.OUT)
ECHO = machine.Pin(16, machine.Pin.IN)

dis = 100

def distance():
    timeout = 10000 * 5 / 340
    TRIG.low()
    time.sleep_us(2)
    TRIG.high()
    time.sleep_us(10)
    TRIG.low()
    timeout_start = time.ticks_ms()
    while not ECHO.value():
        waiting_time = time.ticks_ms()
        if waiting_time - timeout_start > timeout:
            return -1
    time1 = time.ticks_us()
    while ECHO.value():
        waiting_time = time.ticks_ms()
        if waiting_time - timeout_start > timeout:
            return -1
    time2 = time.ticks_us()
    during = time.ticks_diff(time2, time1)
    return during * 340 / 2 / 10000

def ultrasonic_thread():
    global dis
    while True:
        dis = distance()

_thread.start_new_thread(ultrasonic_thread, ())

def beep():
    buzzer.value(1)
    led.value(1)
    time.sleep(0.1)
```

(次のページに続く)

(前のページからの続き)

```
    buzzer.value(0)
    led.value(0)
    time.sleep(0.1)

intervals = 10000000
previousMills = time.ticks_ms()
time.sleep(1)

while True:
    if dis < 0:
        pass
    elif dis <= 10:
        intervals = 300
    elif dis <= 20:
        intervals = 500
    elif dis <= 50:
        intervals = 1000
    else:
        intervals = 2000
    if dis != -1:
        print('Distance: %.2f' % dis)
        time.sleep_ms(100)

    currentMills = time.ticks_ms()

    if time.ticks_diff(currentMills, previousMills) >= intervals:
        beep()
        previousMills = currentMills
```

- プログラムが動作するとすぐに、超音波センサーは前方の障害物までの距離を連続して読み取ります。シェル上で正確な距離値を確認できます。
- LED とブザーは、距離値に応じて点滅とビープの頻度が変わり、障害物が近づいていることを示します。
- *6.1* **距離の測定** の記事で、超音波センサーが動作すると、プログラムが一時停止すると言及されています。
- この例で LED やブザーのタイミングに干渉しないように、測定用に別のスレッドを作成しました。

4.52 7.11 体感コントローラー

ロボット映画をよく観ているなら、このような光景を見たことがあるでしょう。主人公が手首をひねると、巨大なロボットがそれに応じて動き、主人公が拳を振ると、ロボットもそれに続く。非常にクールです。

この技術の使用は、すでに大学や研究機関で一般的であり、5G の到来によってその応用範囲は大いに拡大しています。「外科ロボットダ・ヴィンチ」の遠隔手術は典型的な例です。

この種のロボットシステムは、通常、人間の動きをキャプチャするモジュールとロボットアームを駆動するモジュール（一部の応用シナリオにはデータ通信モジュールも含まれる）の 2 つのモジュールで構成されています。

ここでは、MPU6050 を用いて人間の動きをキャプチャ（グローブに取り付ける）し、サーボを用いてロボットアームの動きを表現しています。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

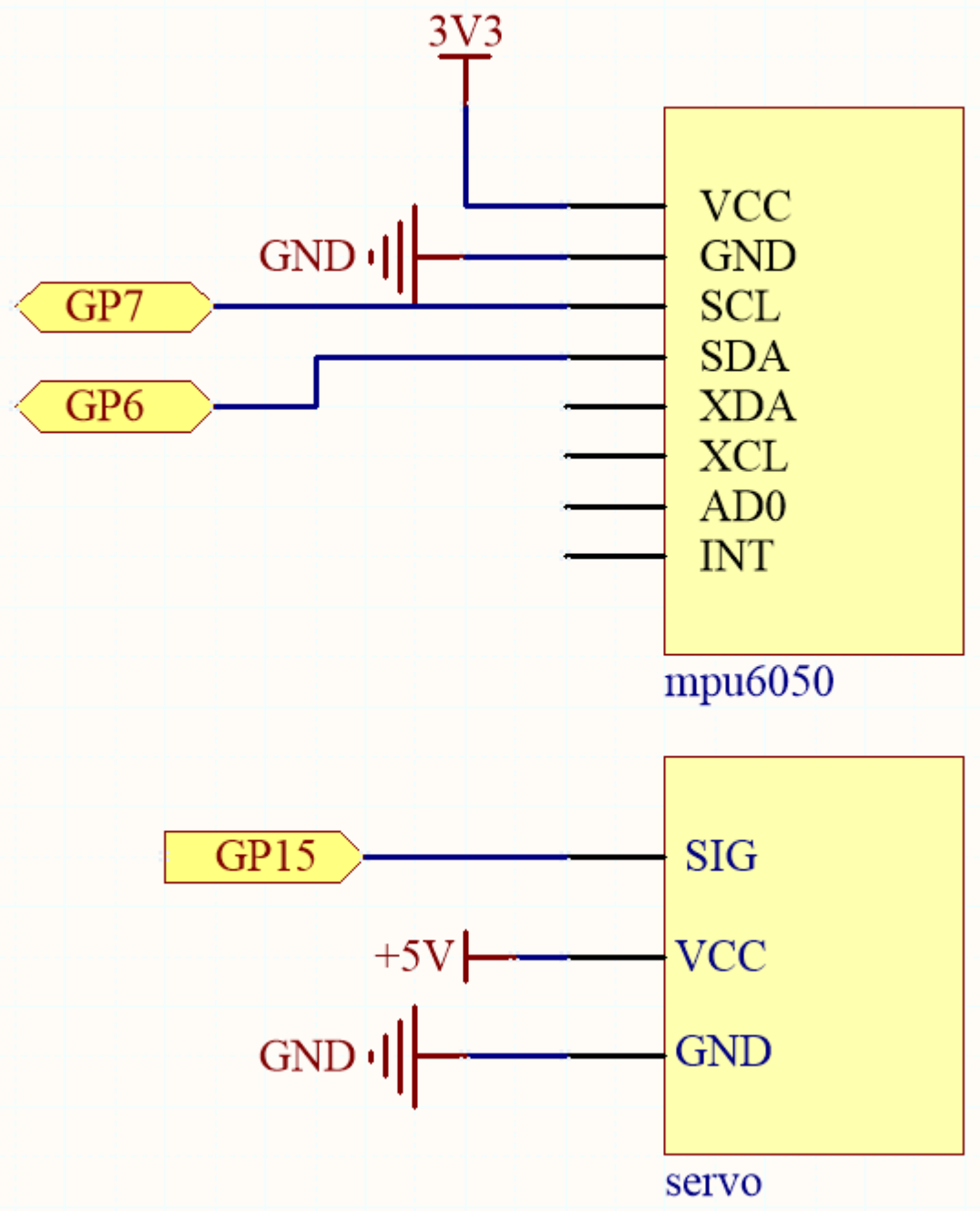
一式をまとめて購入するのは便利です。リンクはこちら：

名称	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MPU6050* **モジュール**	1	
6	**サーボ**	1	

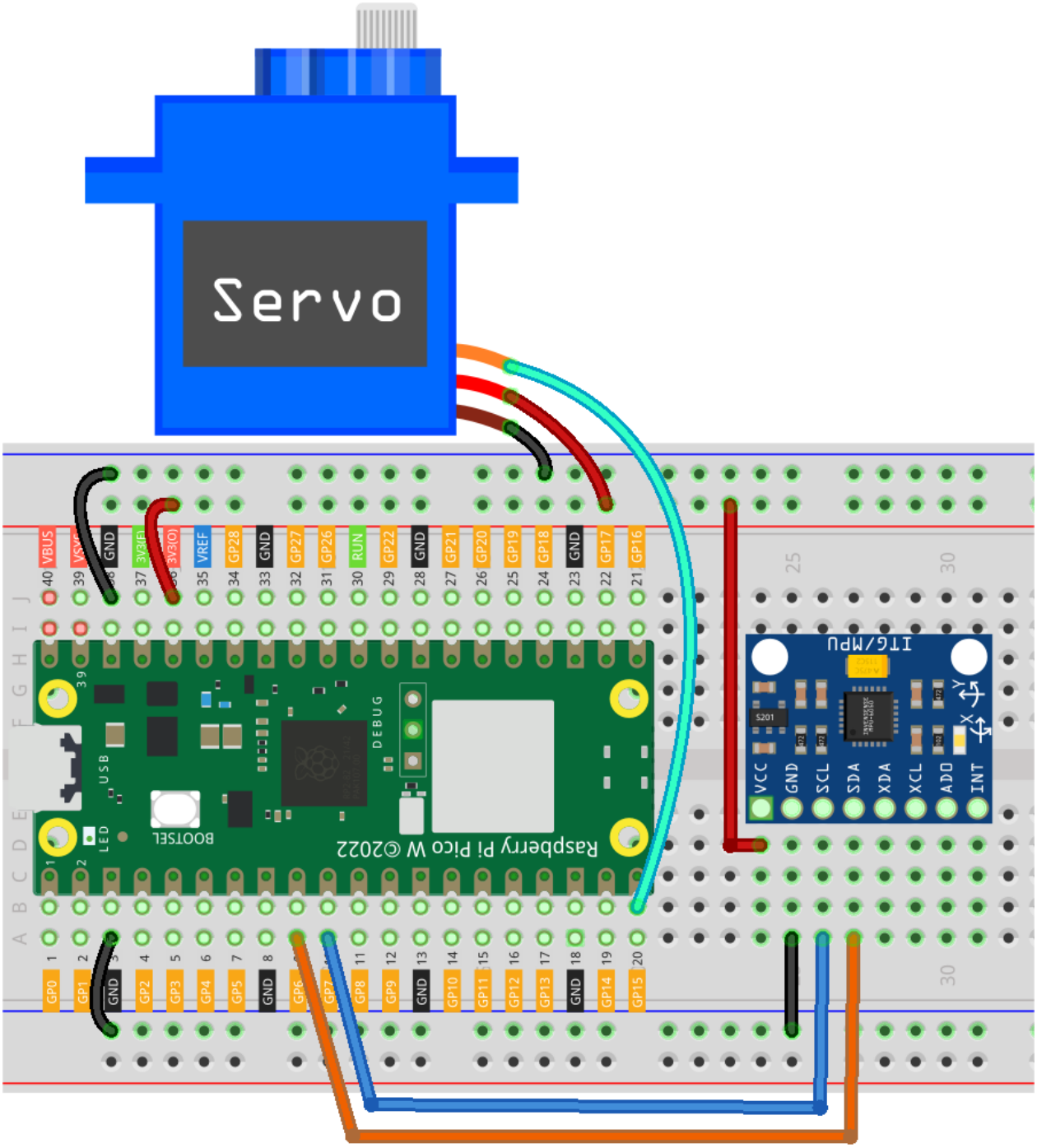
回路図



MPU6050 は各方向の加速度値に基づいて姿勢角を計算します。

プログラムは、姿勢角が変わるにつれて、サーボを対応する偏角で制御します。

配線



コード

注釈:

- `kepler-kit-main/micropython` のパス下にある `7.11_somatosensory_controller.py` ファイルを開くか、このコードを Thonny にコピペして、"Run Current Script"をクリックまたは F5 キーを押して実行します。

- 右下の「MicroPython（Raspberry Pi Pico）」インタープリターをクリックして選択してください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- こちらでは `imu.py` と `vector3d.py` が必要です。Pico W にアップロードされているか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
from imu import MPU6050
from machine import I2C, Pin
import time
import math

# mpu6050
i2c = I2C(1, sda=Pin(6), scl=Pin(7), freq=400000)
mpu = MPU6050(i2c)

# servo
servo = machine.PWM(machine.Pin(15))
servo.freq(50)


def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min



# get rotary angle
def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

# servo work
def servo_write(pin,angle):
```

(次のページに続く)

(前のページからの続き)

```
    pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
    duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
    pin.duty_u16(duty)

times=25
while True:
    total=0
    for i in range(times):
        angle=get_y_rotation(mpu.accel.x, mpu.accel.y, mpu.accel.z) #get rotation value
        total+=angle
    average_angle=int(total/times) # make the value smooth
    servo_write(servo,interval_mapping(average_angle,-90,90,0,180))
```

プログラムが動作すると、MPU6050 を傾ける（またはグローブに取り付けた場合は手首を回す）と、サーボが左右に回転します。

4.53 7.12 デジタル水平器

水平器 は、面が水平（レベル）か垂直（垂直）かを示すための計測器具です。大工、石工、レンガ職人、他の建築関連の作業者、測量士、精密機械工、そして一部の写真やビデオ作業にも使用されるさまざまな種類の水平器があります。

ここでは、MPU6050 と 8x8 LED マトリックスを使用してデジタル水平器を作成します。MPU6050 を傾けると、LED マトリックス上のバブルも傾きます。

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

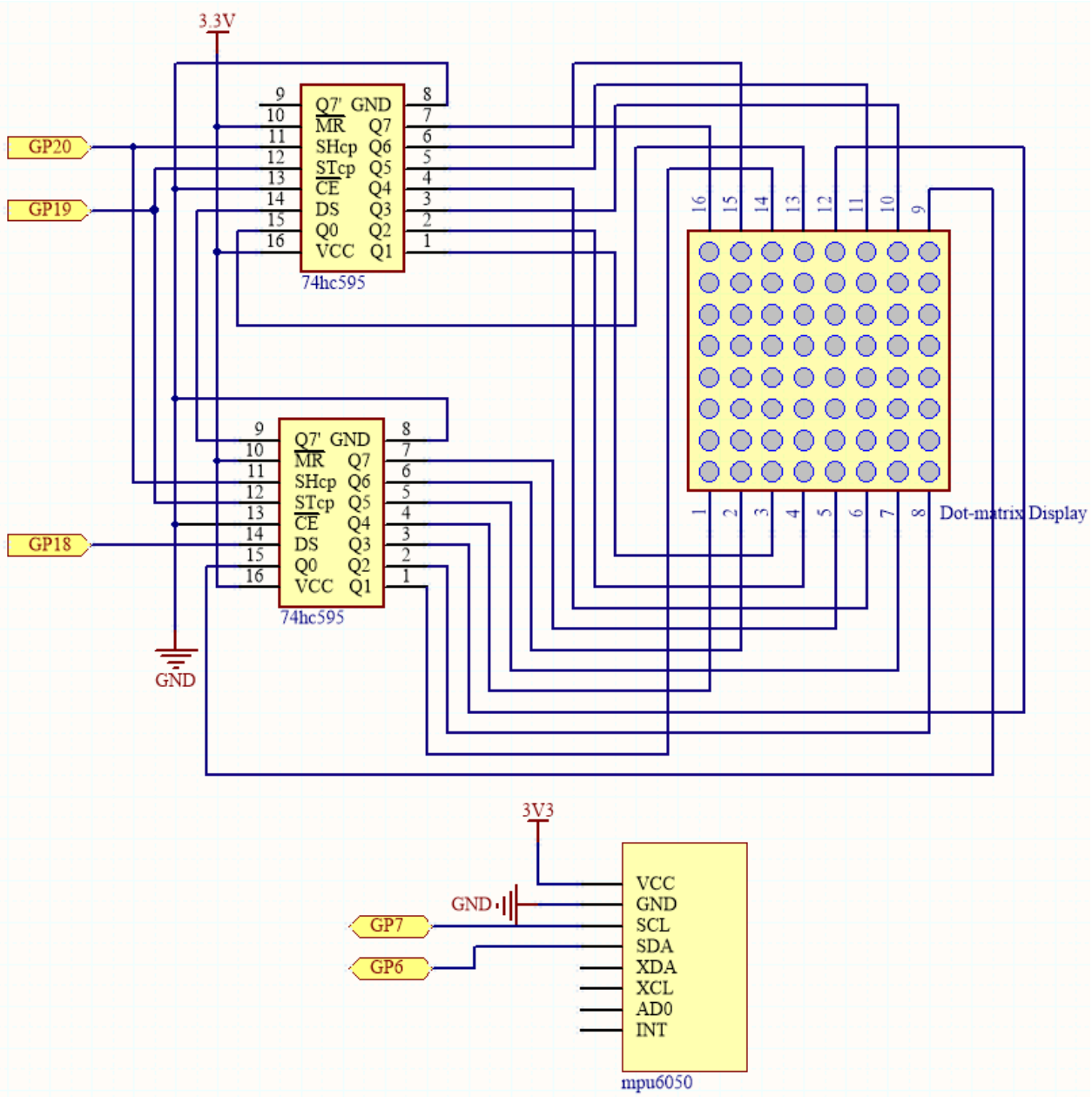
一式をまとめて購入する方が便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

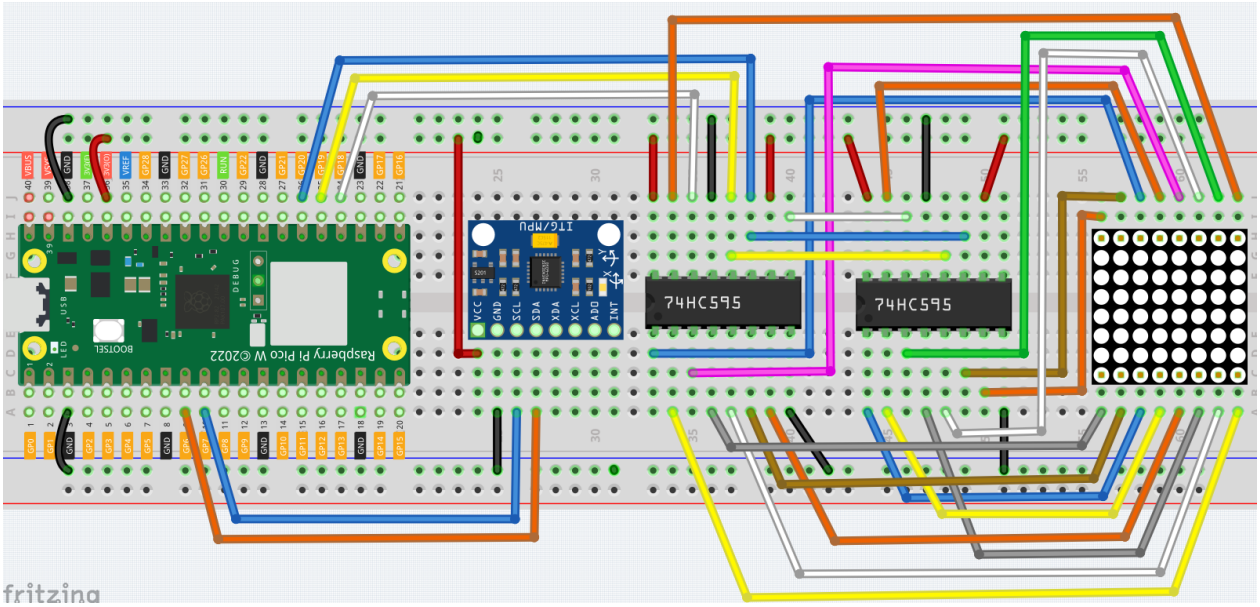
SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	いくつか	
5	*LED* **ドットマトリクス**	1	
6	*74HC595*	2	
7	*MPU6050* **モジュール**	1	

回路図



MPU6050 は、各方向の加速度値を取得し、姿勢角を計算します。その結果、プログラムは、2 つの 74HC595 チップからのデータに基づいて、ドットマトリックス上に 2x2 のドットを描画します。姿勢角が変わると、プログラムは 74HC595 チップに異なるデータを送信し、ドットの位置が変わり、バブル効果が生まれます。

配線



コード

注釈:

- kepler-kit-main/micropython フォルダの 7.12_digital_bubble_level.py ファイルを開いて実行するか、このコードを Thonny にコピーして「Run Current Script」をクリック、または F5 キーを押して実行してください。
- 右下の角にある「MicroPython（Raspberry Pi Pico）」のインタープリターを選択することを忘れないでください。
- 詳細なチュートリアルは、 **コードを直接開いて実行する** を参照してください。
- ここでは imu.py と vector3d.py が必要です。Pico W にアップロードされているかどうか確認してください。詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。

```
import machine
from machine import I2C, Pin
import time
import math
from imu import MPU6050


### mpu6050
i2c = I2C(1, sda=Pin(6), scl=Pin(7), freq=400000)
mpu = MPU6050(i2c)
```

(次のページに続く)

(前のページからの続き)

```
# get rotary angle
def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

def get_angle():
    y_angle=get_y_rotation(mpu.accel.x, mpu.accel.y, mpu.accel.z)
    x_angle=get_x_rotation(mpu.accel.x, mpu.accel.y, mpu.accel.z)
    return x_angle,y_angle

### led matrix display
sdi = machine.Pin(18,machine.Pin.OUT)
rclk = machine.Pin(19,machine.Pin.OUT)
srclk = machine.Pin(20,machine.Pin.OUT)

def hc595_in(dat):
    for bit in range(7,-1, -1):
        srclk.low()
        time.sleep_us(30)
        sdi.value(1 & (dat >> bit))
        time.sleep_us(30)
        srclk.high()

def hc595_out():
    rclk.high()
    time.sleep_us(200)
    rclk.low()

def display(glyph):
    for i in range(0,8):
        hc595_in(glyph[i])
```

(次のページに続く)

(前のページからの続き)

```
        hc595_in(0x80>>i)
        hc595_out()

# data transformation
def matrix_2_glyph(matrix):
    glyph= [0 for i in range(8)] # glyph code for display()
    for i in range(8):
        for j in range(8):
            glyph[i]+=matrix[i][j]<<j
    return glyph

def clamp_number(val, min, max):
    return min if val < min else max if val > max else val

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

# Calculate the position of the bubble
sensitivity=4           # The higher the number, the more sensitive
matrix_range=7          # The size of the matrix is 8, so the coordinate range is 0~7
point_range=matrix_range-1      # The x, y value of the bubble's marker point (upper left␣
↪point) should be between 0-6
def bubble_position():
    x,y=get_angle()
    x=int(clamp_number(interval_mapping(x,-90,90,0-sensitivity,point_range+sensitivity),
↪0,point_range))
    y=int(clamp_number(interval_mapping(y,-90,90,point_range+sensitivity,0-sensitivity),
↪0,point_range))
    return [x,y]

# Drop the bubble into empty matrix
def drop_bubble(matrix,bubble):
    matrix[bubble[0]][bubble[1]]=0
    matrix[bubble[0]+1][bubble[1]]=0
    matrix[bubble[0]][bubble[1]+1]=0
    matrix[bubble[0]+1][bubble[1]+1]=0
    return matrix

while True:
```

(次のページに続く)

(前のページからの続き)

```
matrix= [[1 for i in range(8)] for j in range(8)]  # empty matrix
bubble=bubble_position() # bubble coordinate
matrix=drop_bubble(matrix,bubble) # drop the bubble into empty matrix
display(matrix_2_glyph(matrix)) # show matrix
```

プログラムを実行した後、ブレッドボードを水平な面に置いてください。LED マトリックスの中央にドットが表示されます（中央にない場合は、MPU6050 が水平でない可能性があります)。ブレッドボードを傾けると、ドットも傾けた方向に動きます。

第 5 章

IoT プロジェクト

このセクションでは、Pico W をネットワークに接続し、いくつかの興味深い IoT プロジェクトを完成させる方法を解説します。

ただし、このセクションに進む前の最初のステップは、 *MicroPython* **ユーザーのために** 内の **1. はじめに** を完了し、Thonny IDE のインストール、Raspberry Pi Pico W 用の Micropython ファームウェアのインストール、およびライブラリのアップロードを行うことです。

5.1 1. ネットワークへのアクセス

Raspberry Pi Pico W は、Raspberry Pi Pico と非常に似ており、GPIO、microUSB ポート、サイズが同じです。唯一の違いは、Infineon からの CYW43439 2.4-GHz Wi-Fi チップが追加されている点です。それでは、どのようにして私たちの Wi-Fi ネットワークに接続するのか見てみましょう。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

こちらが一式を購入する便利なリンクです：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	

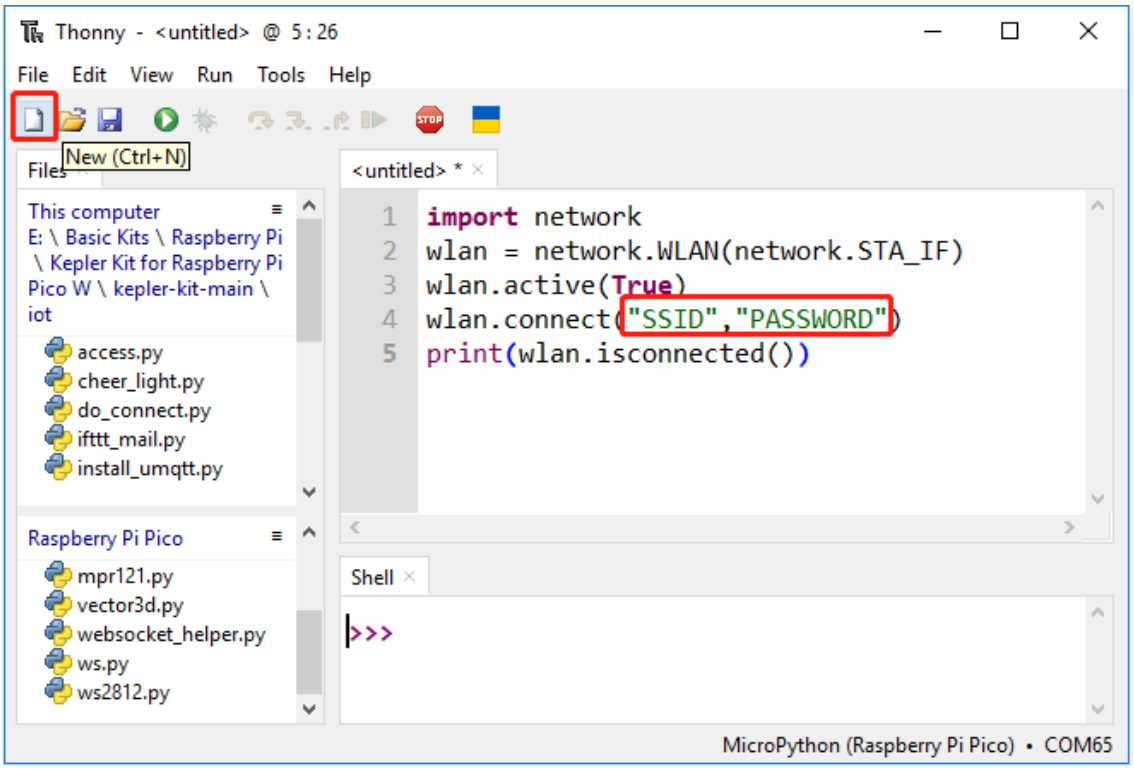
5.1.1 1. インターネットに接続

わずか 5 行の MicroPython で、Raspberry Pi Pico W はインターネットに接続されます。

これら 5 行のコードは、Shell から直接実行することができ、入力後に Enter キーを押します。または、以下の方法に従って新しい .py ファイルを作成して実行します。

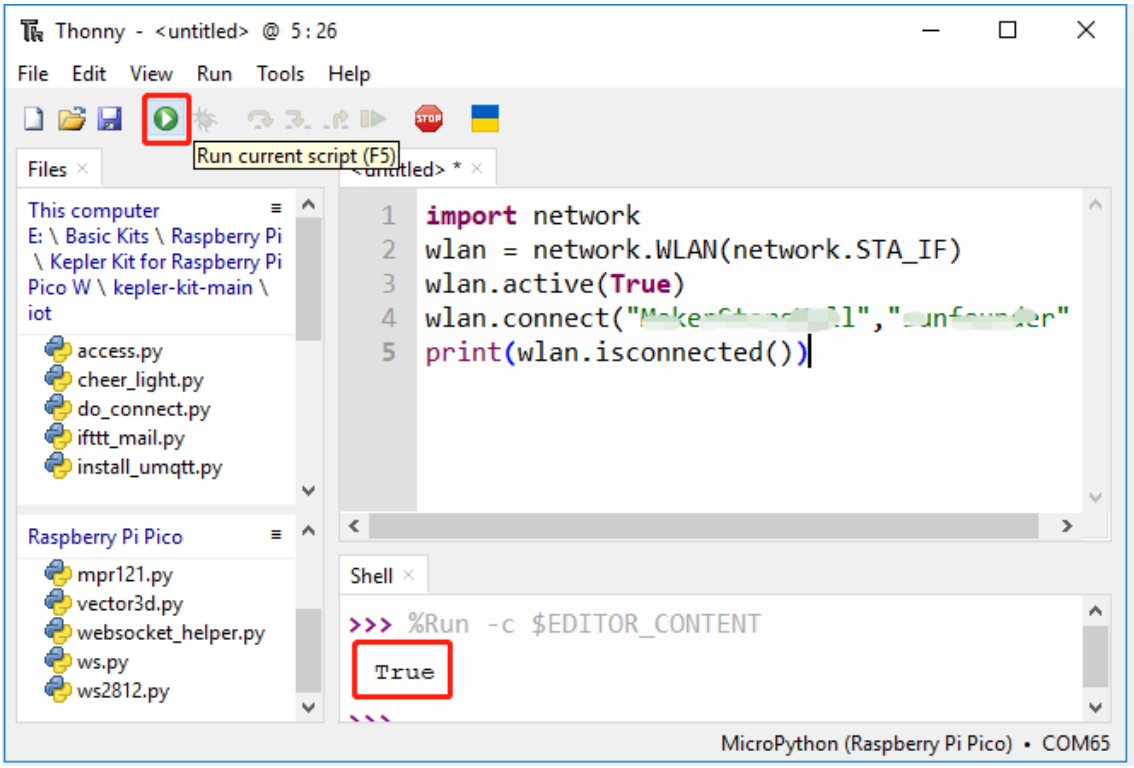
```
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect("SSID","PASSWORD")
print(wlan.isconnected())
```

1. Thonny の **新規作成** ボタンをクリックして新しいスクリプトを作成し、上記のコードをコピーして貼り付け、 SSID と PASSWORD を自分のものに変更します。



2. スクリプトを実行するには、 **現在のスクリプトを実行** ボタンをクリックするか、F5 キーを押します。接続が成功すると、 true が表示されます。

注釈: Raspberry Pi Pico W が USB ケーブルでコンピュータに接続されていることを確認し、右下のコーナーをクリックして MicroPython（Raspberry Pi Pico）.COMXxx をインタープリタとして選択します。



5.1.2 2. タイムアウト判断と IP の表示

ネットワーク状況が良くない場合を考慮して、コードにタイムアウトの判断を追加しましょう。

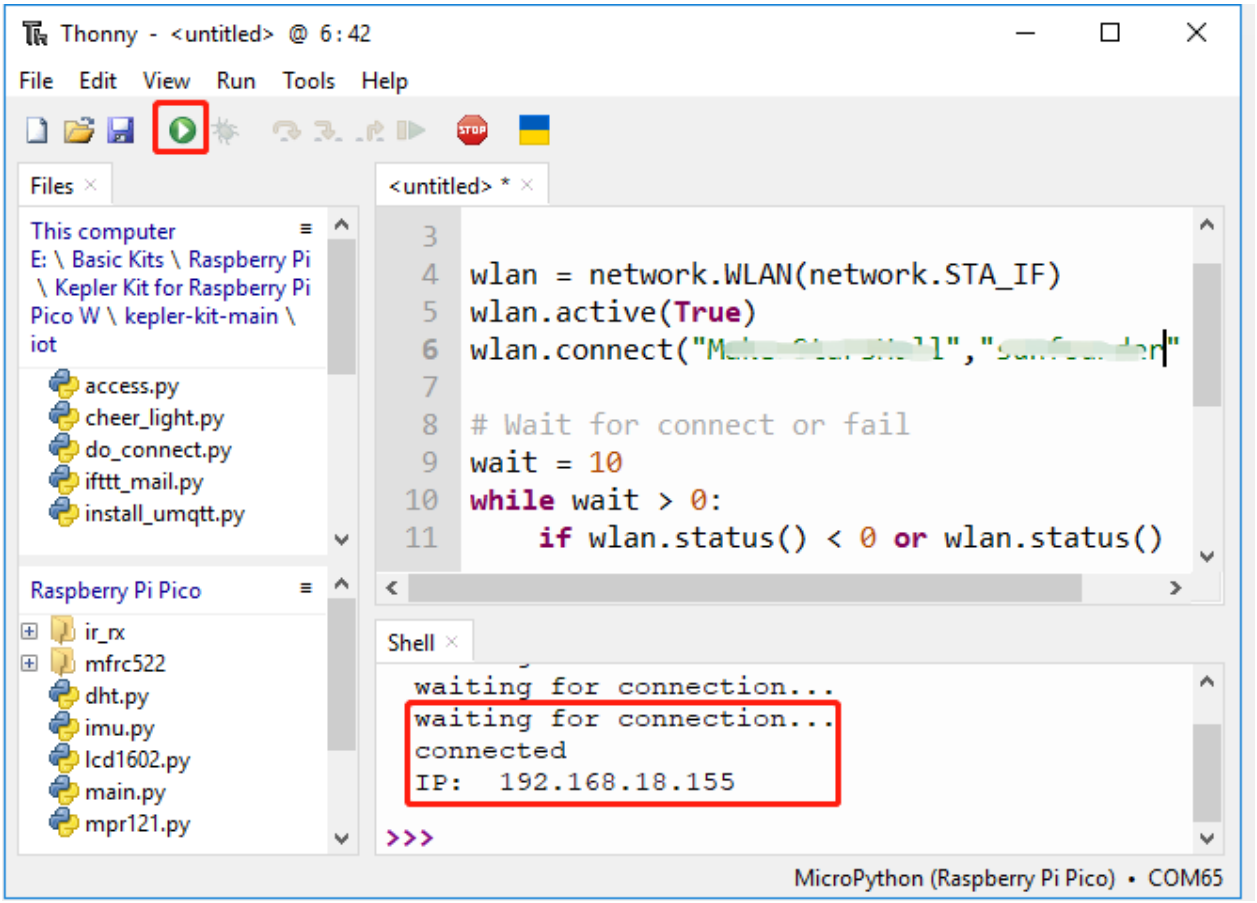
接続に成功すると、スクリプトをコピーして実行した後に、Pico W の IP が表示されます。

```
import network
import time

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect("SSID","PASSWORD")

# Wait for connect or fail
wait = 10
while wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    wait -= 1
    print('waiting for connection...')
    time.sleep(1)
```

(次のページに続く)

(前のページからの続き)

```
# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    print('IP: ', wlan.ifconfig()[0])
```



- `wlan.status()` 関数：無線接続の現在のステータスを返します、戻り値は以下の表に示されています。

ステータス	値	説明
STAT_IDLE	0	接続も活動もなし
STAT_CONNECTING	1	接続中
STAT_WRONG_PASSWORD	-3	パスワードが不正なため失敗
STAT_NO_AP_FOUND	-2	アクセスポイントが応答しないため失敗
STAT_CONNECT_FAIL	-1	その他の問題による失敗
STAT_GOT_IP	3	接続成功

- `wlan.ifconfig()` 関数：IP アドレス、サブネットマスク、ゲートウェイ、DNS サーバーを取得します。このメソッドは、直接呼び出された場合、上記の情報を含む 4 タプルを返します。この場合、IP アドレスのみを表示します。

- class WLAN – MicroPython Docs

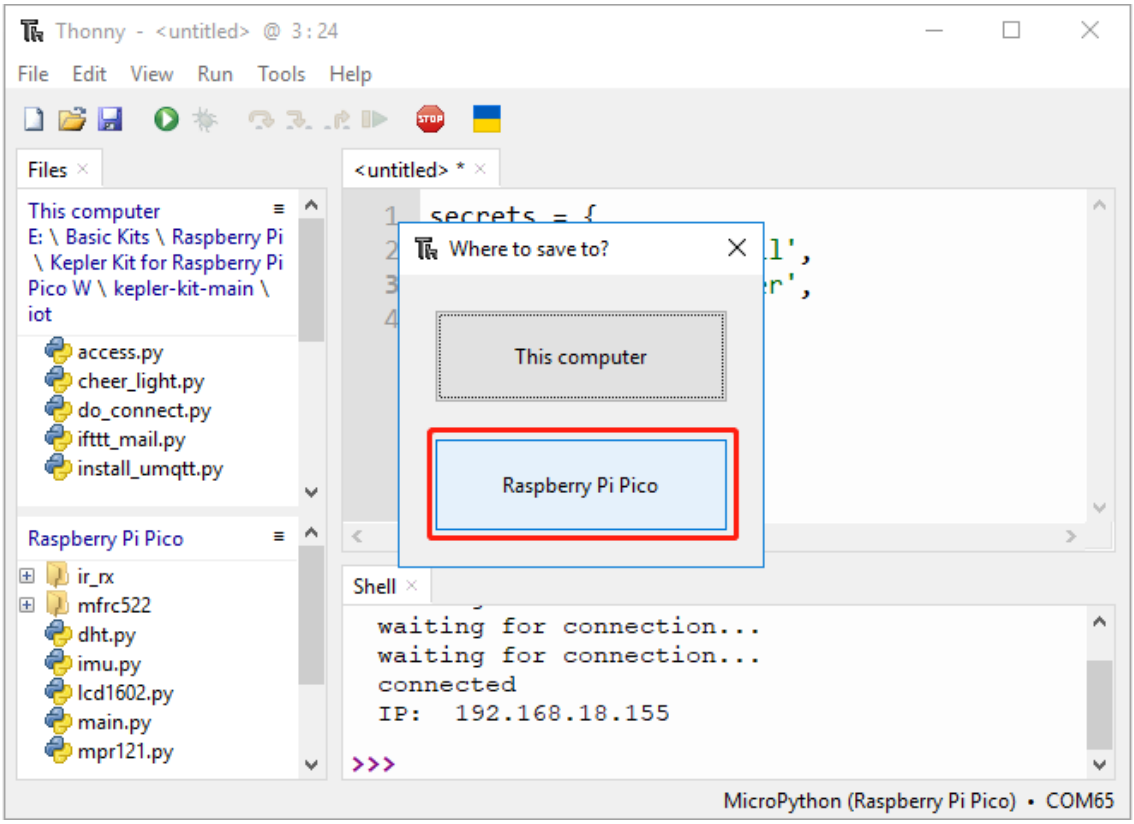
5.1.3 3. secrets.py にプライベート情報を保存

Pico W プロジェクトを共有する際、Wi-Fi のパスワードや API キーを他人に見られたくないでしょう。より高いセキュリティを確保するために、 `secrets.py` ファイルを作成してプライベート情報を保存できます。

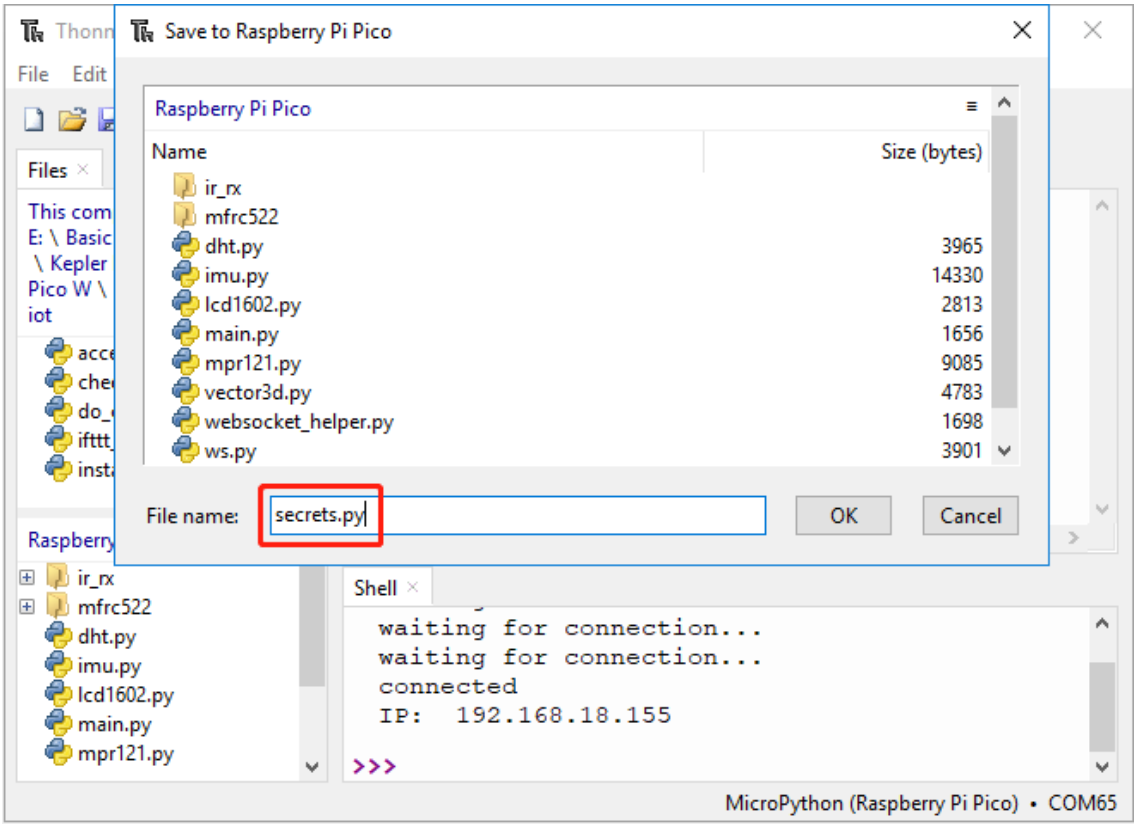
1. 以下のコードを Thonny で新しいスクリプトファイルにコピーします。 `SSID` と `PASSWORD` は自分のものに変更してください。

```
secrets = {
'ssid': 'SSID',
'password': 'PASSWORD',
}
```

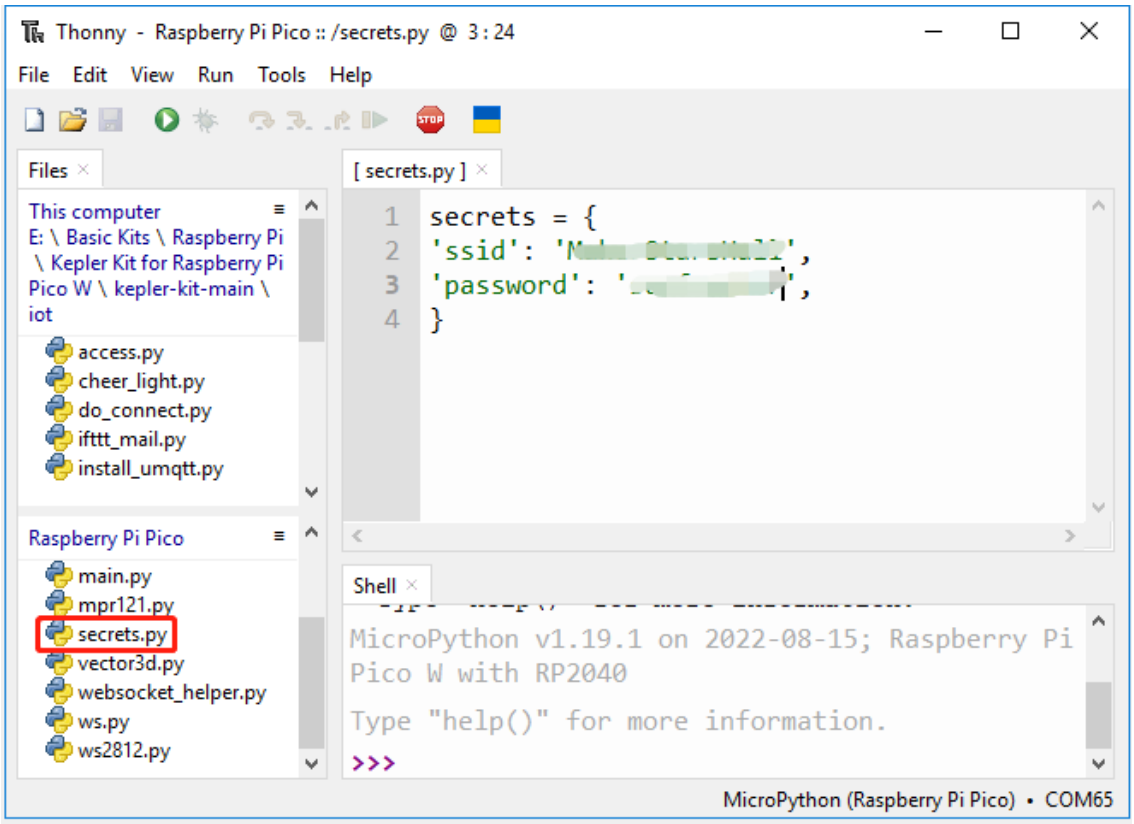
2. 「保存」ボタンをクリックするか、 `Ctrl+S` を押すと表示されるポップアップウィンドウで Raspberry Pi Pico を選択します。



3. 名前を `secrets.py` に設定します。



4. これで、このスクリプトは Raspberry Pi Pico W で見ることができます。



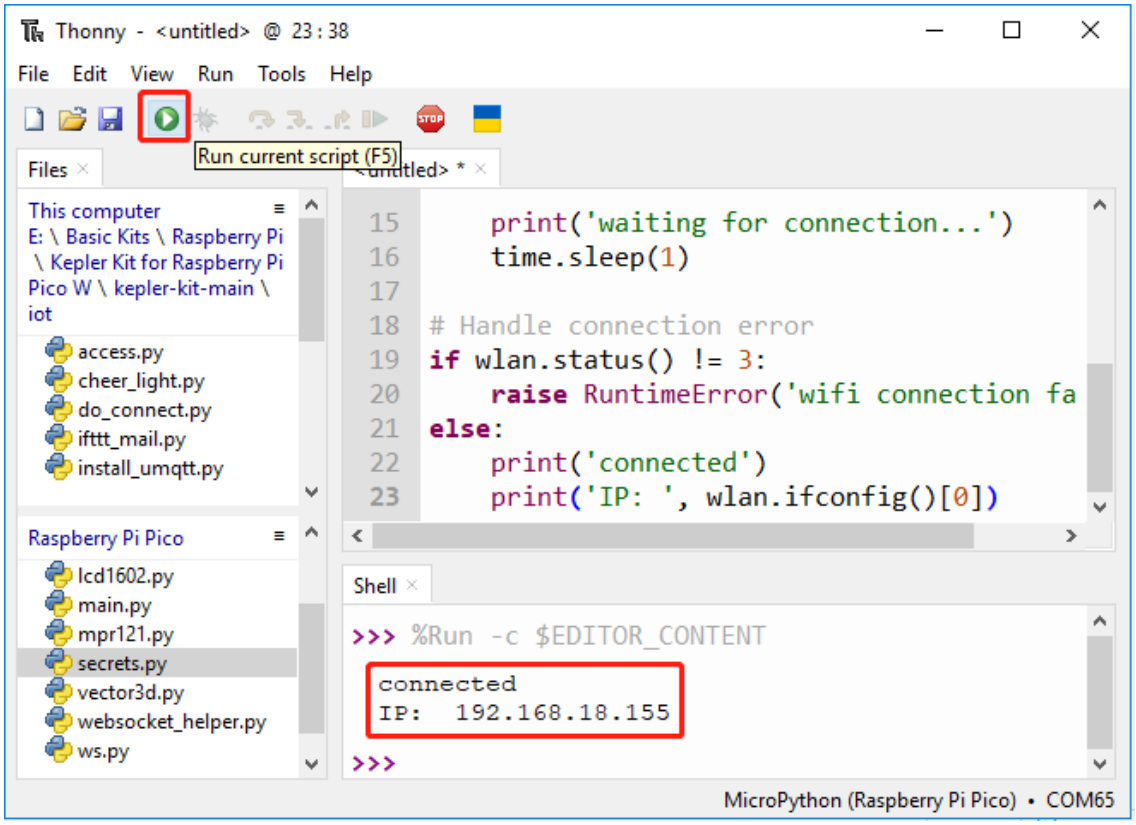
5. 他のスクリプトで次のように呼び出すことができます。実行すると、Wi-Fi 接続が成功することが確認できます。 secrets.py ファイルはライブラリとしてインポートされるので、情報の漏洩を心配する必要はありません。

```
import network
import time
from secrets import secrets

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(secrets['ssid'], secrets['password'])

# Wait for connect or fail
wait = 10
while wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    wait -= 1
    print('waiting for connection...')
    time.sleep(1)
```

(次のページに続く)

(前のページからの続き)

```
# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    print('IP: ', wlan.ifconfig()[0])
```



5.1.4 4. do_connect.py でインターネットに接続

次々と行うプロジェクトでネットワーク接続が必要となるため、新しい do_connect.py ファイルを作成して関連する関数をそこに記述し、再利用するのはいかがでしょうか。これにより、複雑なプロジェクトのコードを大幅にシンプルにすることができます。

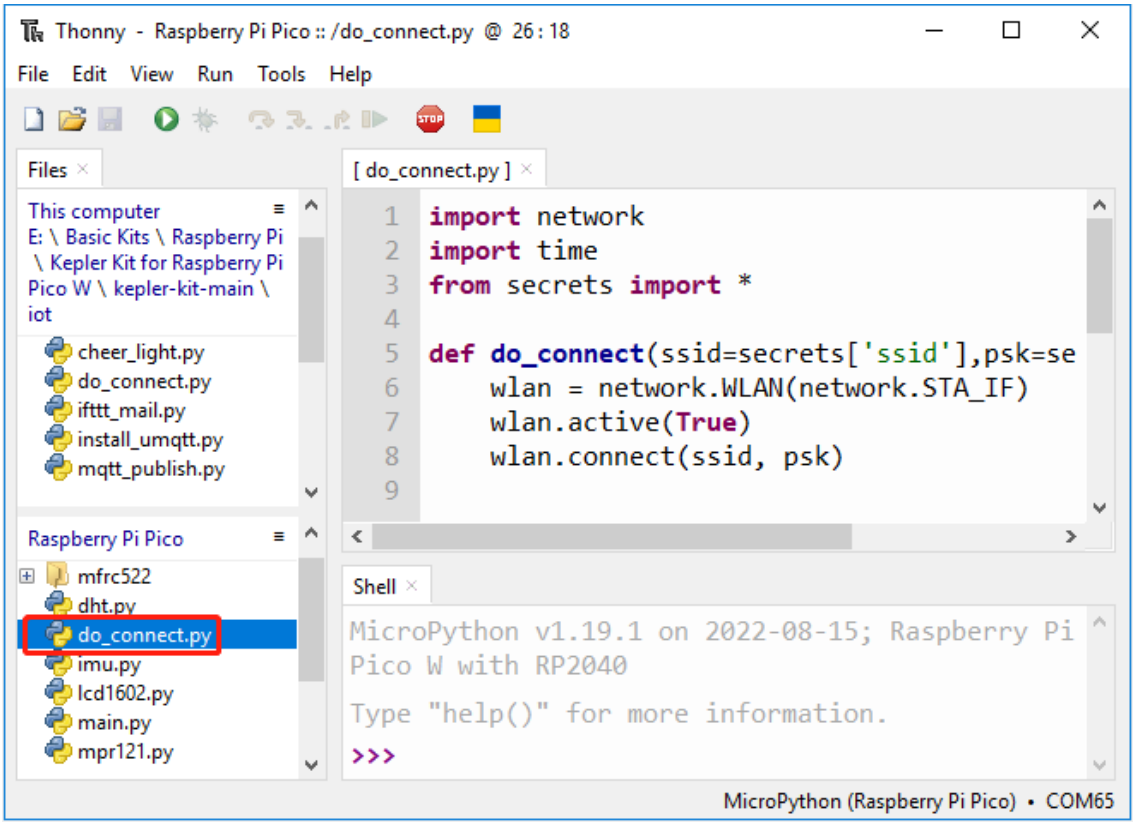
1. 以下のコードを新しいスクリプトファイルにコピーし、Raspberry Pi Pico に do_connect.py として保存します。

```
import network
import time
```

(次のページに続く)

(前のページからの続き)

```
from secrets import *

def do_connect(ssid=secrets['ssid'],psk=secrets['password']):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, psk)

    # Wait for connect or fail
    wait = 10
    while wait > 0:
        if wlan.status() < 0 or wlan.status() >= 3:
            break
        wait -= 1
        print('waiting for connection...')
        time.sleep(1)

    # Handle connection error
    if wlan.status() != 3:
        raise RuntimeError('wifi connection failed')
    else:
        print('connected')
        ip=wlan.ifconfig()[0]
        print('network config: ', ip)
        return ip
```



2. 以下のように他のスクリプトで呼び出すと、Raspberry Pi Pico W はネットワークに接続されます。

```
from do_connect import *
do_connect()
```

5.2 2. @CheerLights に参加する

これはロマンチックなプロジェクトで、 LED のカラーチェンジングコミュニティに参加するものです。このコミュニティでは、世界中の LED が同時に色を変えることができます。

オフィスの隅に置いて、「自分は一人ではない」と感じるためのリマインダーとしても使用できます。

@cheerlights にツイートを送り、そのツイート内に色の名前を含めると、世界中の LED が指定した色に変わります。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

一式をまとめて購入するのが便利ですが、その場合のリンクはこちらです。

名前	このキットの内容	リンク
ケプラーキット	450 以上	

下記のリンクから個々に購入することも可能です。

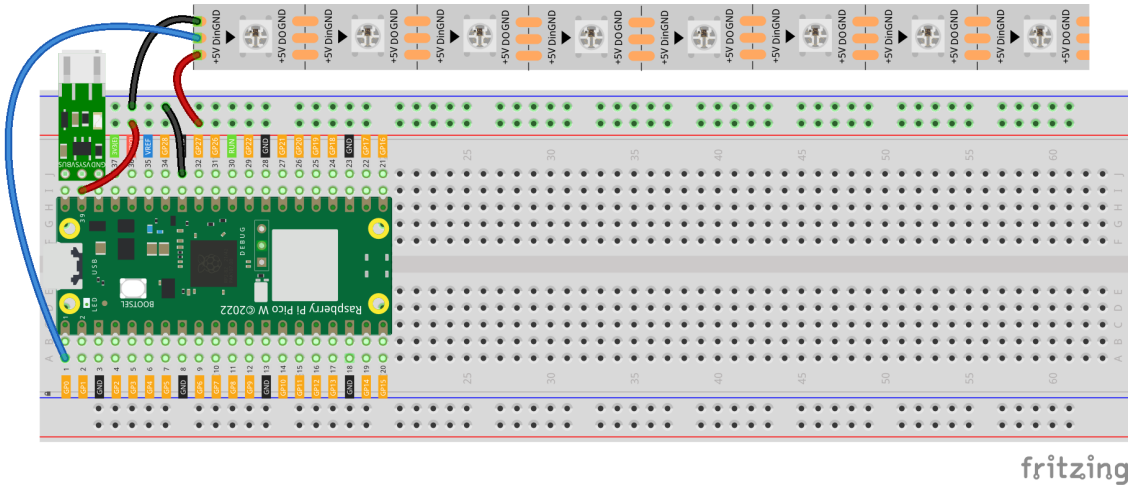
SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*WS2812 RGB 8 LED* **ストリップ**	1	
6	*Li-po* **充電モジュール**	1	
7	18650 バッテリー	1	
8	バッテリーホルダー	1	

手順

1. 回路を組み立てます。

   ここで使用する Li-po Charger Module（リチウムポリマー充電モジュール）は、USB ケーブルを切断して、プロジェクトを他の場所で遊ぶために回路に電力を供給します。

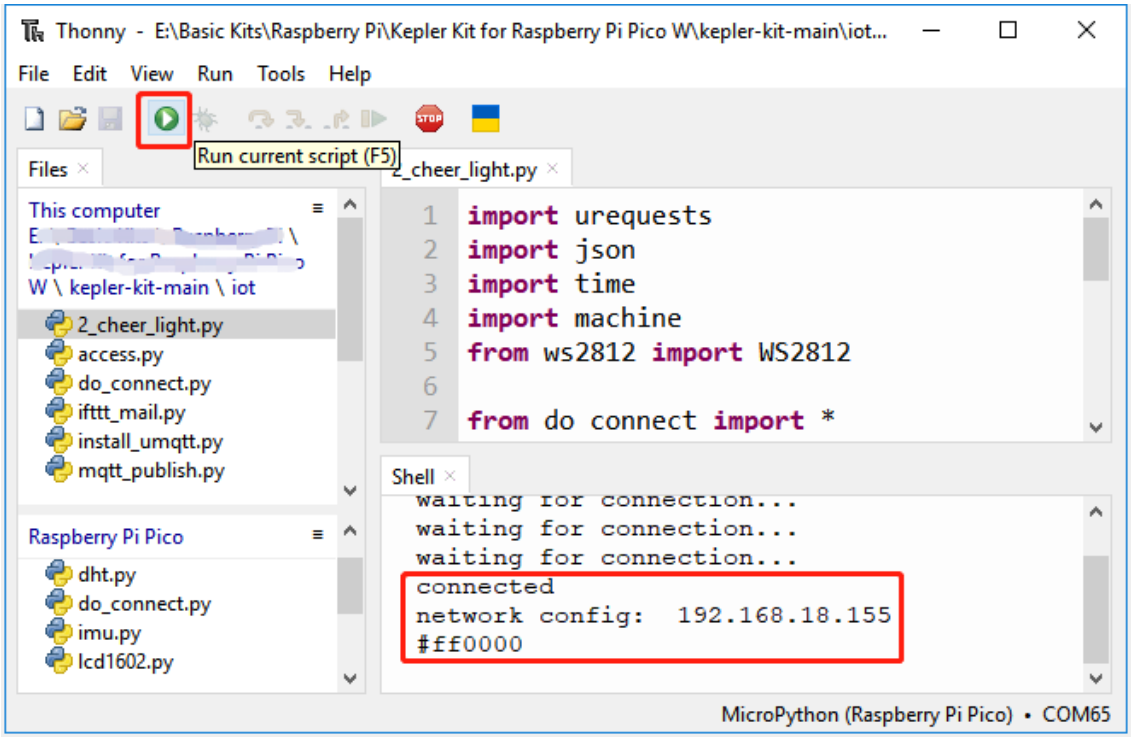
   警告: Li-po Charger Module が図に示されているように接続されていることを確認してください。それ以外の場合、ショートする可能性があり、バッテリーと回路が損傷する可能性があります。



2. 以前に コードパッケージ をダウンロードしたフォルダにパスを切り替え、「kepler-kit-main/iot」のパスの下で `2_cheer_light.py` ファイルを開きます。

3. スクリプトを実行するには、 **Run current script（現在のスクリプトを実行）** ボタンをクリックするか、F5 キーを押します。その後、Shell に接続プロンプト、IP、色（0xff0000 は赤）が表示されます。

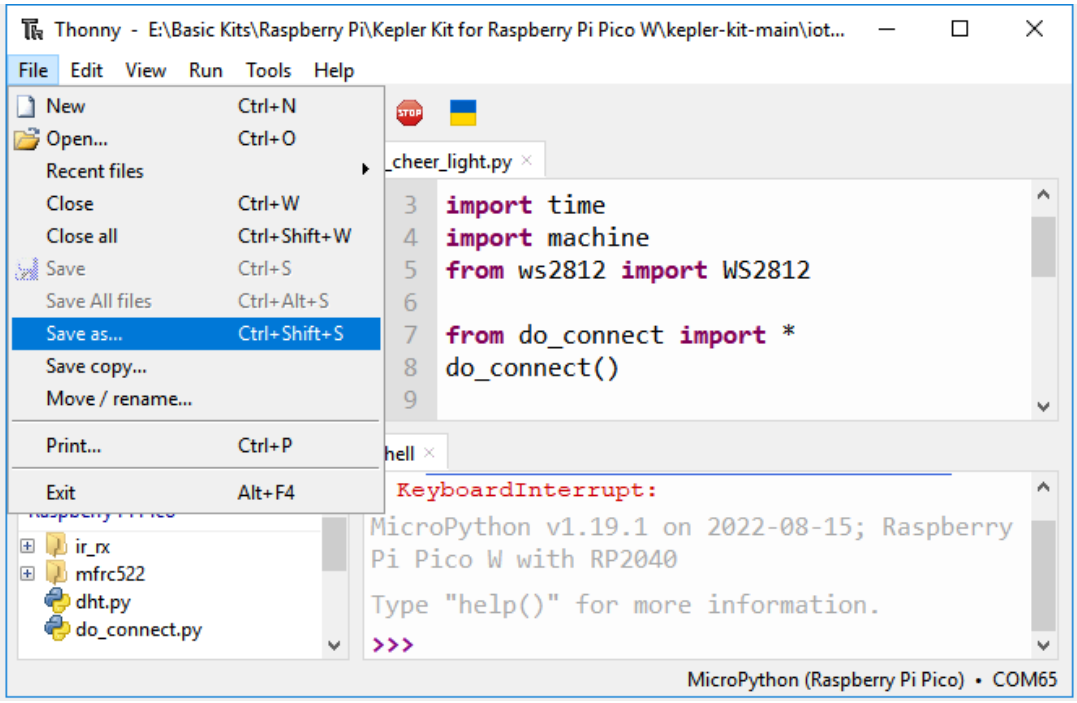
注釈: コードを実行する前に、Pico W に `do_connect.py` と `secrets.py` のスクリプトがあることを確認してください。もし無ければ、 *1.* **ネットワークへのアクセス** を参照してそれらを作成してください。



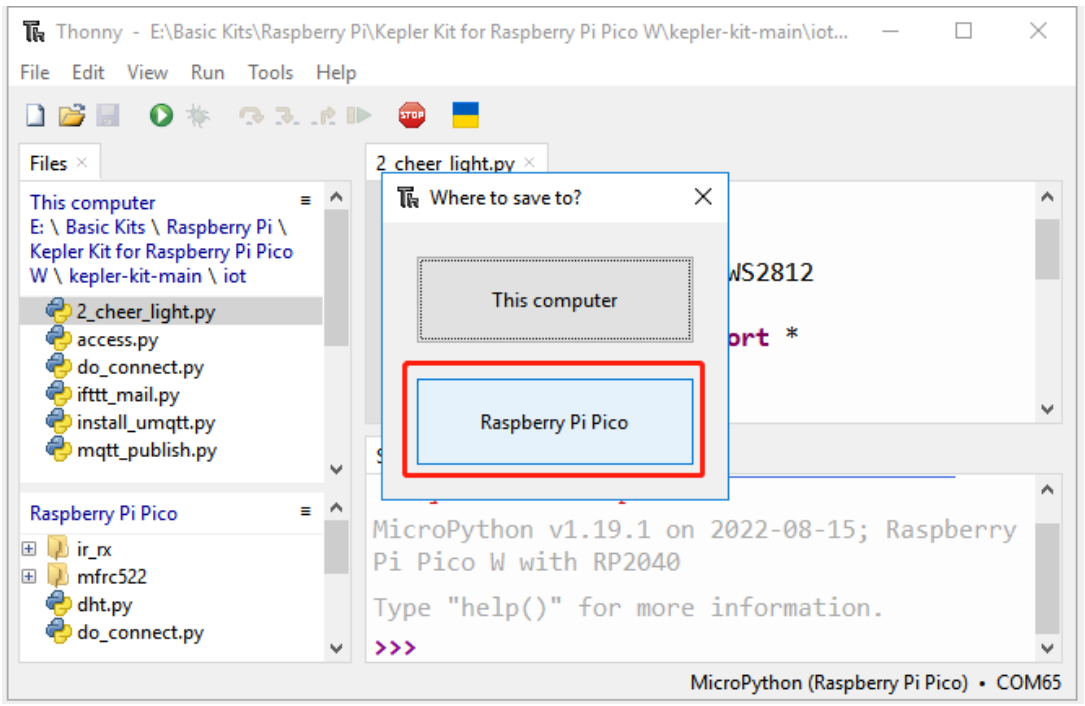
4. スクリプトが実行された後、WS2812 RGB ストリップは色を表示しますが、時抗その色は変わります。

5. 起動時にこのスクリプトを実行する場合、以下の手順に従って Raspberry Pi Pico W に `main.py` として保存する必要があります。

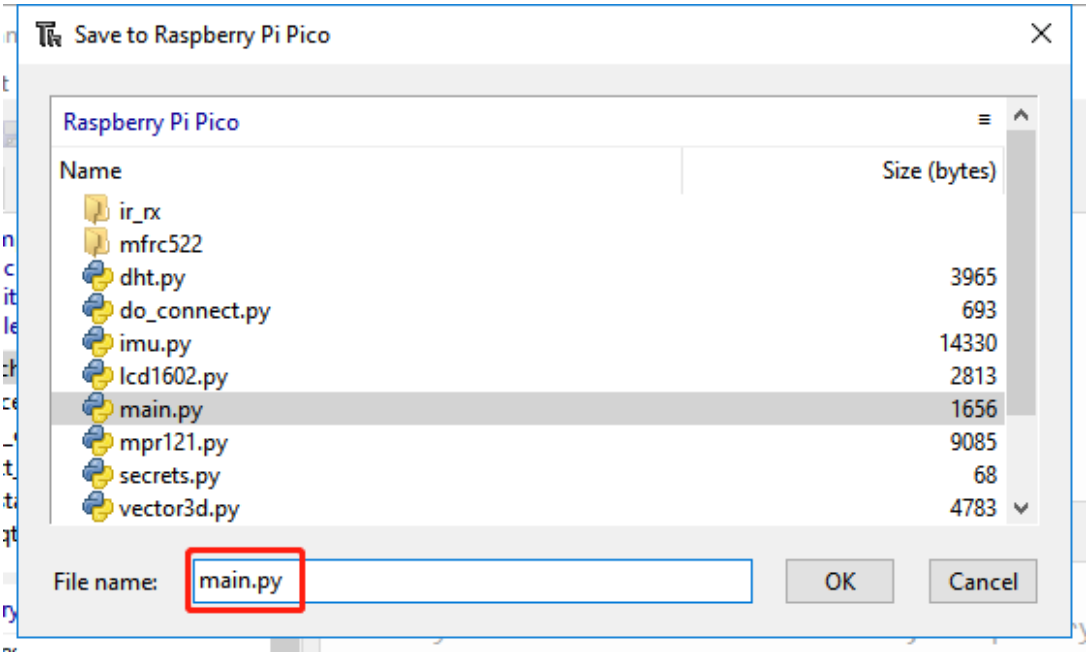
   - スクリプトの実行を停止し、 **File（ファイル）** -> **Save as（名前を付けて保存）** をクリックします。



   - ポップアップウィンドウで **Raspberry Pi Pico** を選択します。



- ファイル名を `main.py` に設定します。同じファイルがすでに Pico W に存在する場合は、プロンプトが表示されます。



- USB ケーブルを抜いて、Li-po Charger Module で Raspberry Pi Pico W に電力を供給することができます。角に置いておけば、自動的に動作します。

仕組み

Raspberry Pi Pico W は、 *1.* **ネットワークへのアクセス** で説明されているように、インターネットに接続する必要があります。このプロジェクトでは、そのまま使用します。

```
from do_connect import *
do_connect()
```

WS2812 RGB ストリップの設定については、 *3.3 RGB LED* **ストリップ** を参照してください。

```
import machine
from ws2812 import WS2812
ws = WS2812(machine.Pin(18), 8)
```

次に、@CheerLights の色を取得する方法が必要です。Twitter から色の変更を受け取るバックエンドシステムがあり、その情報を JSON 形式で URL http://api.thingspeak.com/channels/1417/field/2/last.json に投稿します。

この URL をブラウザで直接開くと、以下のようなものが表示されます。必要なのは `field2` データで、16 進数でエンコードされた色情報です。

```
{"created_at":"2022-08-16T06:12:44Z","entry_id":870488,"field2":"#ff00ff"}
```

このデータを取得するために urequests モジュールを使用し、 json モジュールを使用してこの文字を Python の辞書に変換します。次のコードは、URL から最新の@CheerLights の色を取得し、WS2812 で使用できる色値を返します。

```
def get_colour():
    url = "http://api.thingspeak.com/channels/1417/field/2/last.json"
    try:
        r = urequests.get(url)
        if r.status_code > 199 and r.status_code < 300:
            cheerlights = json.loads(r.content.decode('utf-8'))
            print(cheerlights['field2'])
            colour = int('0x'+cheerlights['field2'][1:7])#Convert from String to Integer
            r.close()
            return colour
        else:
            return None
    except Exception as e:
        print(e)
        return None
```

最後に、1 分ごとに ws2812 が動作するようにループを使用します。

```
while True:
    colour = get_colour()
    if colour is not None:
        ws.write_all(colour)
    time.sleep(60)
```

5.3 3. @IFTTT を使用したセキュリティシステム

このプロジェクトでは、侵入者や迷い込んだ動物が自宅に侵入した場合に検出する PIR センサーを使用したセキュリティデバイスを作成します。そのような場合には、メールで警告が届きます。

最も基本的なサービスとして Webhook が使用されます。Raspberry Pi Pico W から IFTTT のサービスに POST リクエストが送信されます。IFTTT を使用して、Webhook を傍受し、メールを送信する Applet を作成します。

1. 必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

便利なのは、キット全体を購入することです。リンクはこちら：

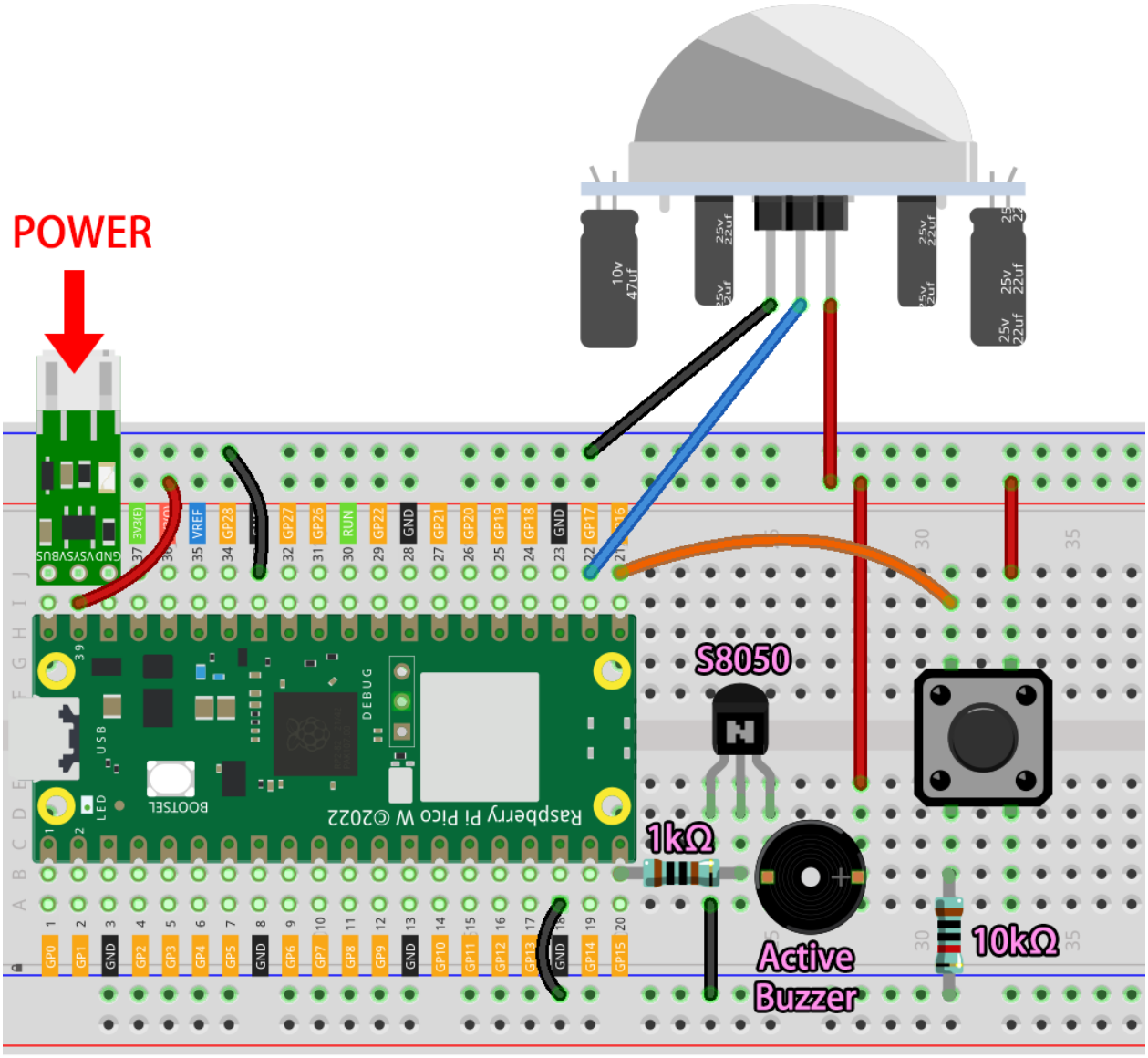
名前	キット内容	リンク
ケプラーキット	450+	

以下のリンクから個々に購入することも可能です。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	2(1K Ω, 10K Ω)	
7	**ボタン**	1	
8	アクティブ **ブザー**	1	
9	*Li-po* **充電モジュール**	1	
10	18650 バッテリー	1	
11	バッテリーホルダー	1	
12	*PIR* **モーションセンサーモジュール**	1	

2. 回路を作成する

> **警告:** Li-po Charger Module が図に示されているように接続されていることを確認してください。それ以外の場合は、ショートが発生し、バッテリーと回路が損傷する可能性があります。



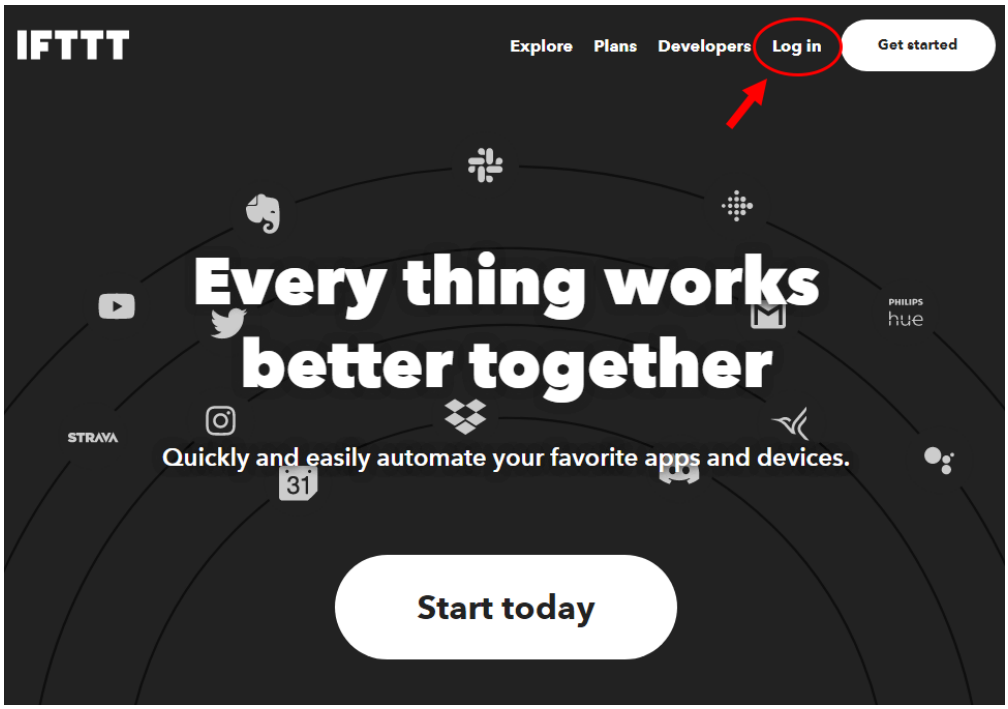
3. IFTTT の設定

IFTTT は、さまざまなデータサービスを連携する多様な方法を提供する無料のサービスです。

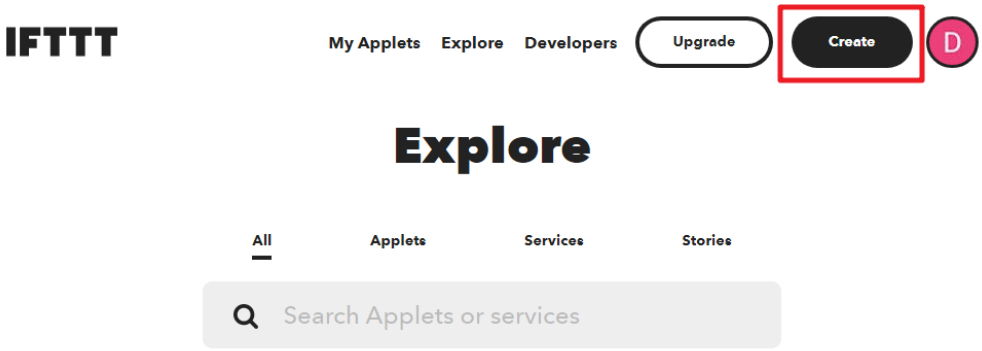
ここでは、IFTTT にデータを送信する Webhook（カスタム URL）に反応する Applet を作成します。その後、IFTTT が代わりにメールを送信します。

以下の手順で IFTTT で設定してください。

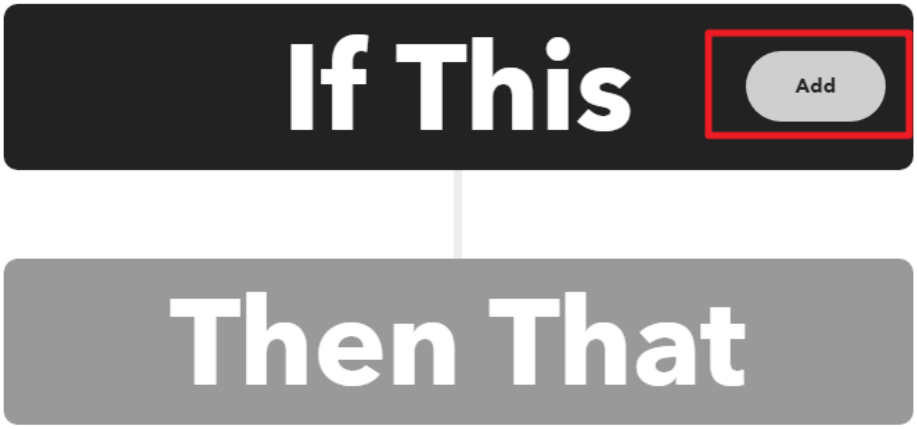
1. にアクセスして、ログインまたはアカウントを作成します。



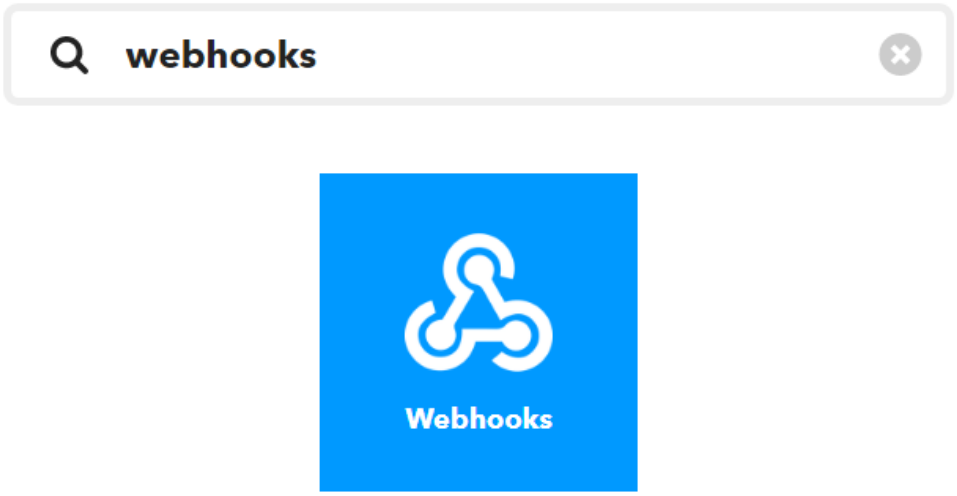
2. **Create（作成）** をクリックします。



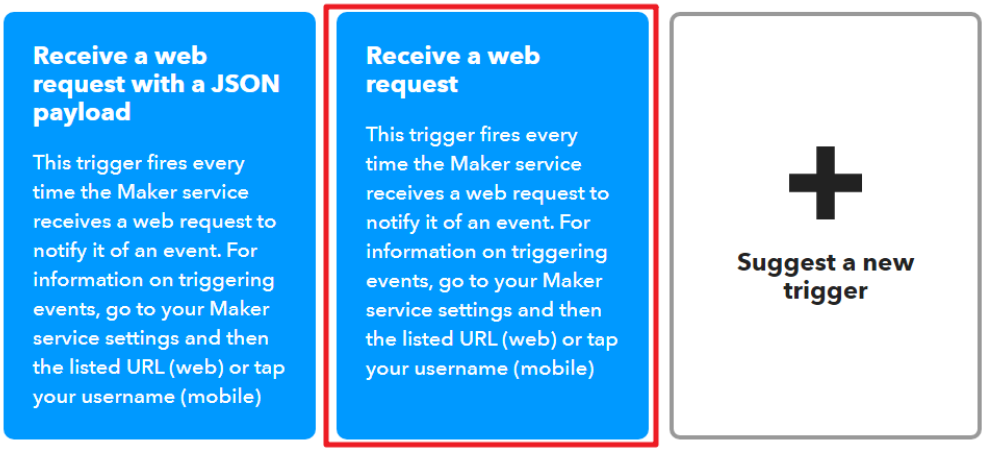
3. **If This（これが起きたら）** イベントを追加します。



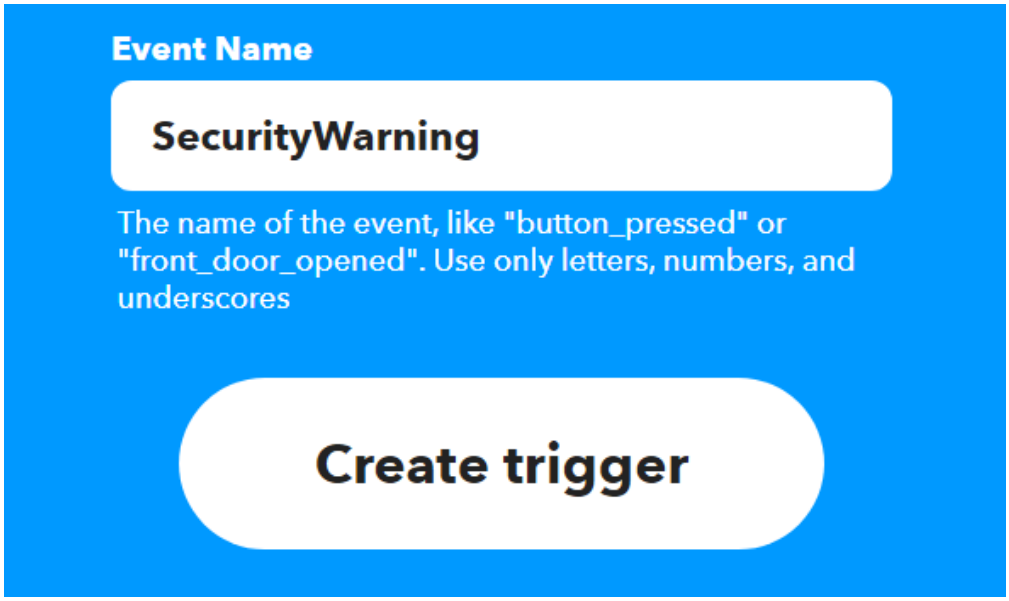
4. **Webhooks（ウェブフック）** を検索します。



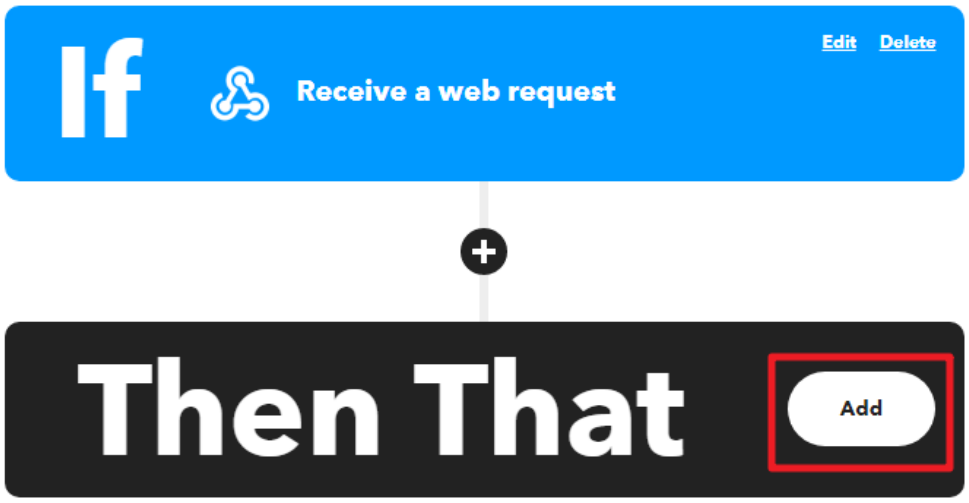
5. **Receive a web request（ウェブリクエストを受け取る）** をタップします。



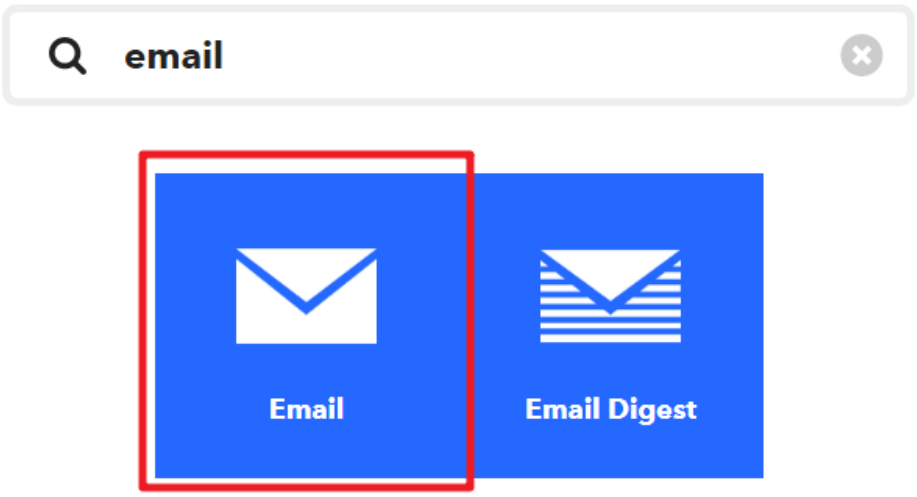
6. イベント名（例：SecurityWarning）を入力し、 **Create trigger（トリガーを作成）** をクリックします。



7. **Then That（それがあれば）** イベントを追加します。

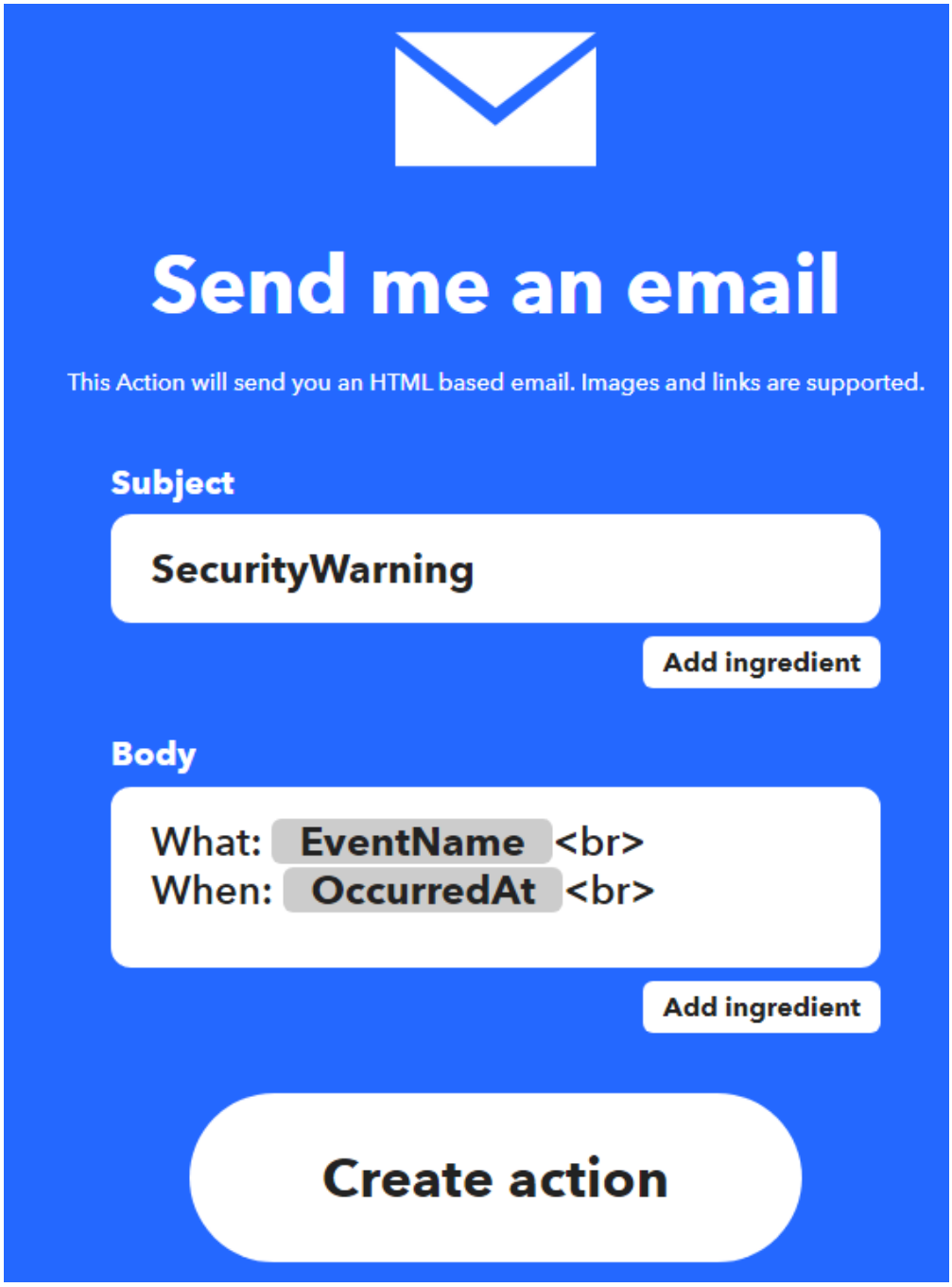


8. Email を検索します。



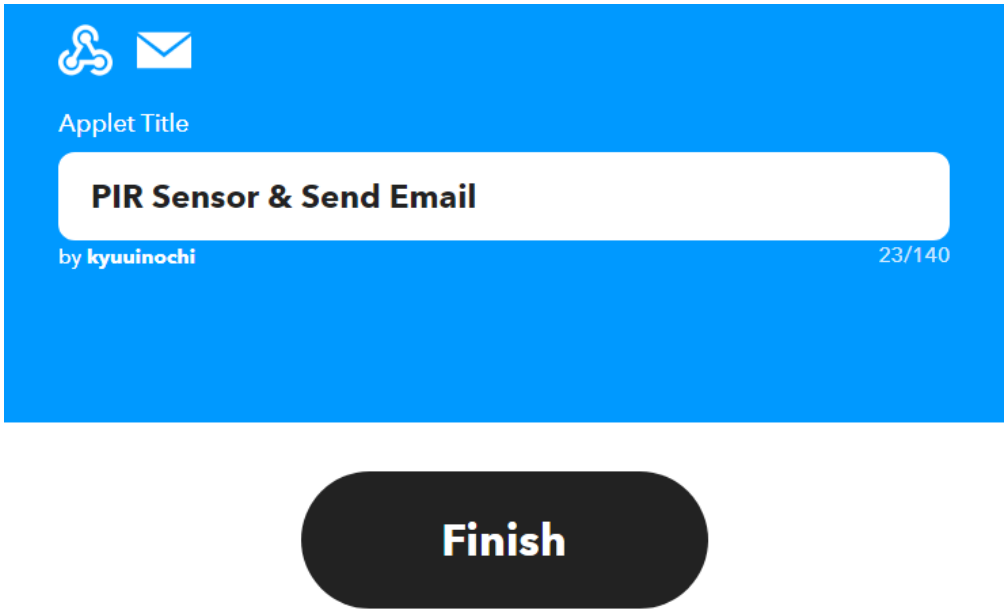
9. **Send me an email（私にメールを送る）** をクリックします。

10. **Subject（件名）** と **Body（本文）** を入力し、 **Create action（アクションを作成）** をクリックします。

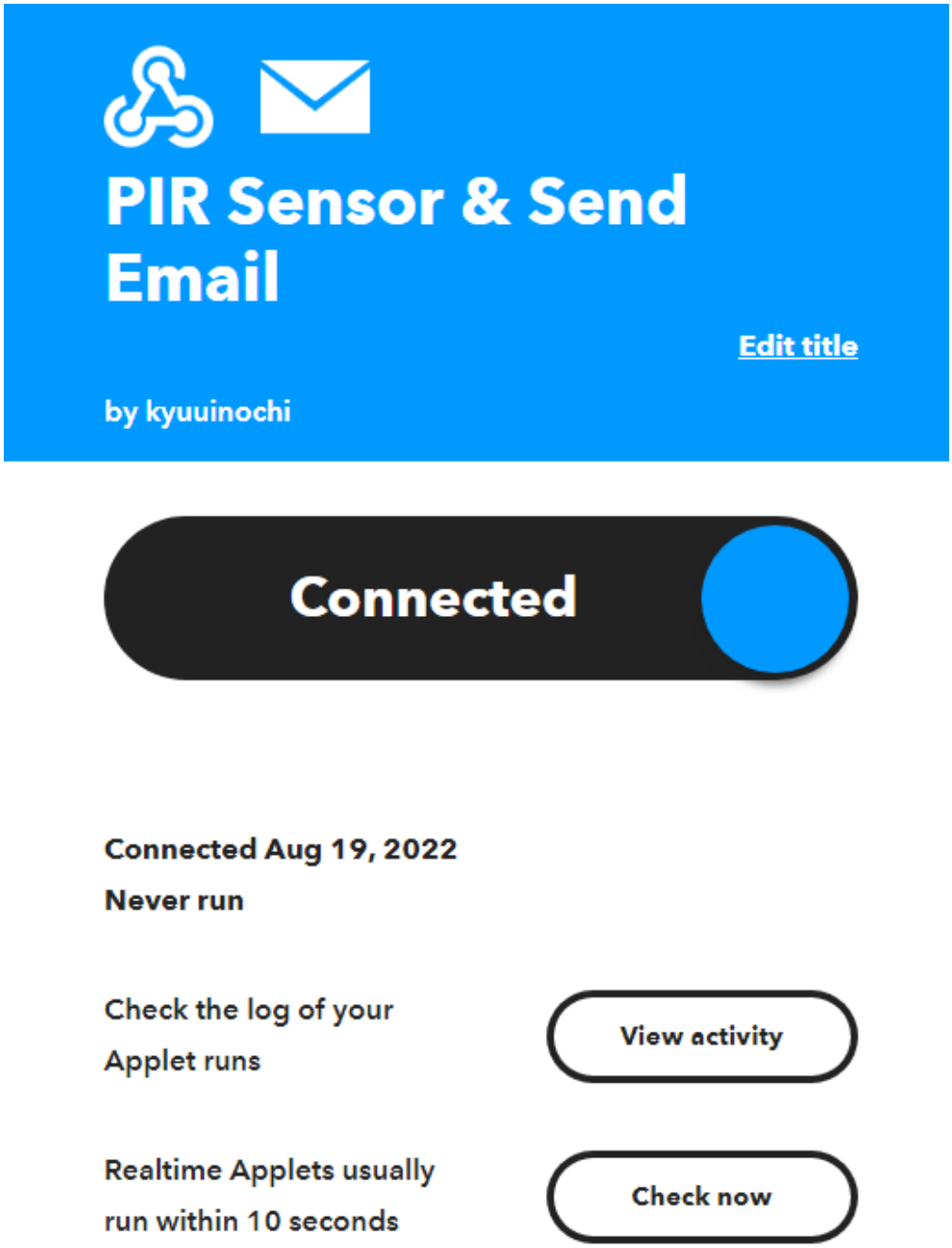


11. **Continue（続ける）** をクリックして、設定を完了します。

12. タイトル名を変更して、完了です。

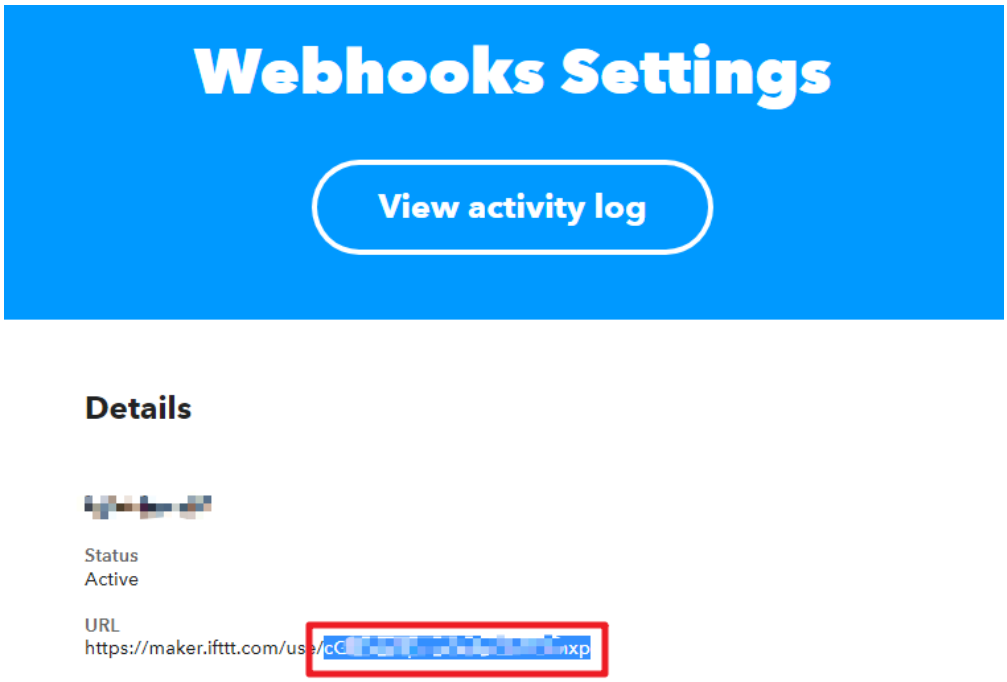


13. 自動的に Applet 詳細ページにリダイレクトされ、Applet が現在接続されていること、およびスイッチを切り替えて開始/終了することができることが確認できます。

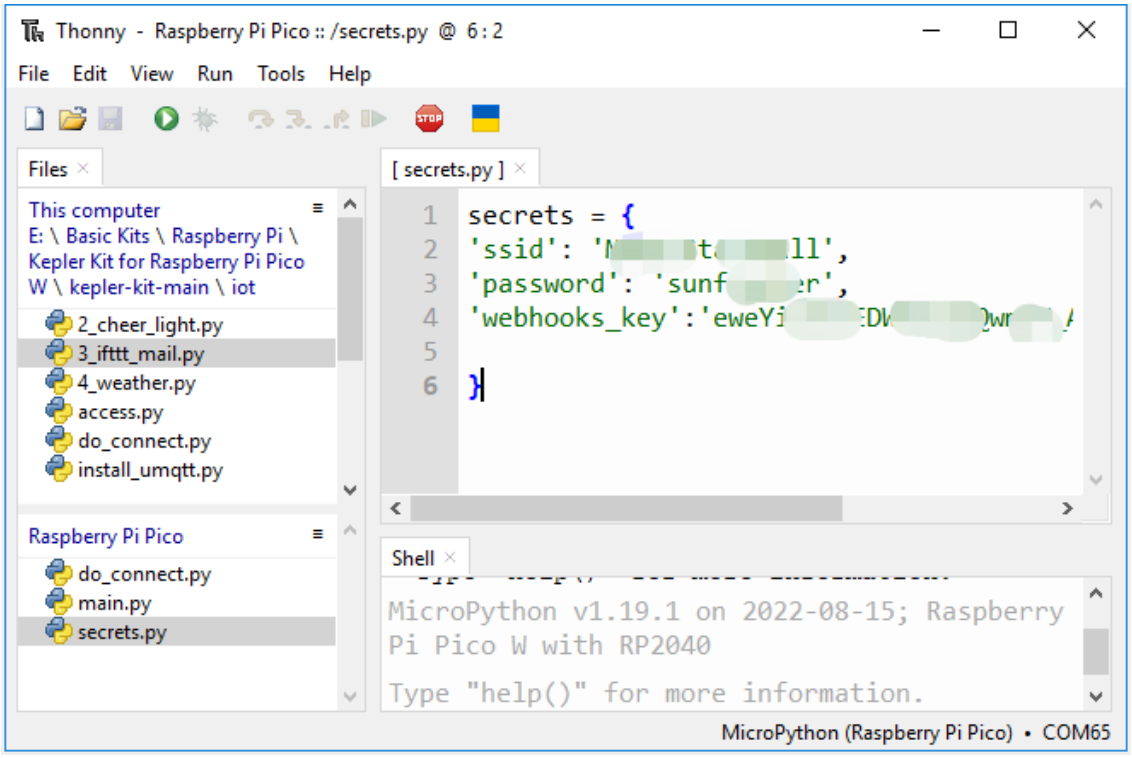


4. スクリプトの実行

1. すでに IFTTT の Applet を作成したわけですが、Pico W が IFTTT にアクセスするためには API キーも必要です。これは から取得できます。



2. それを Raspberry Pi Pico W 内の `secrets.py` スクリプトにコピーします。

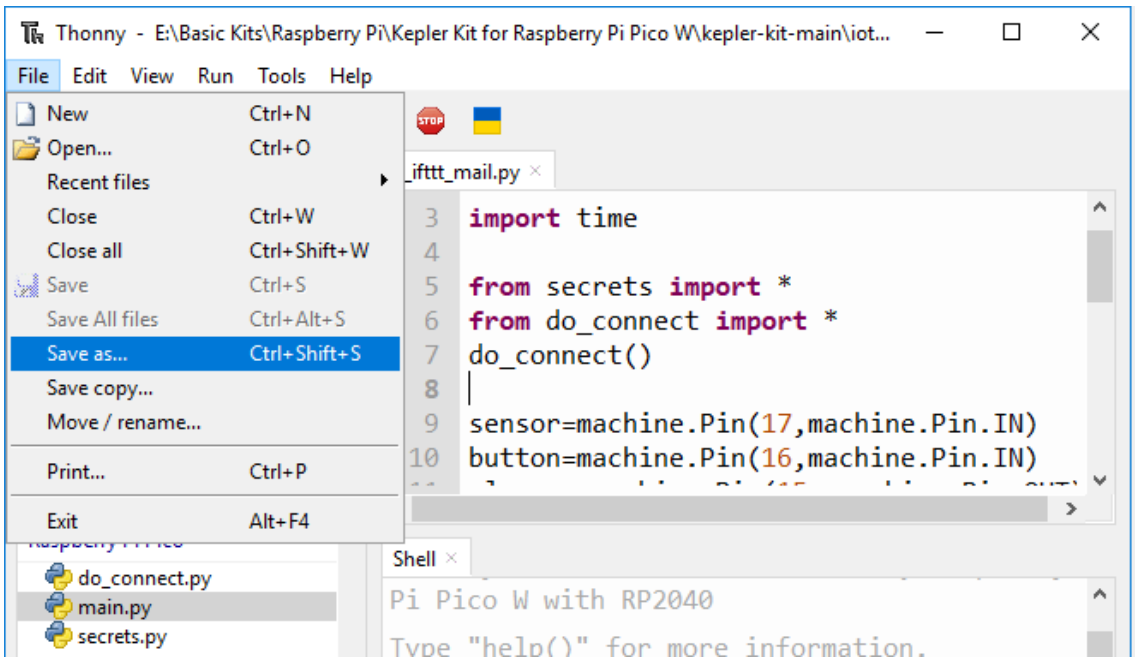


注釈: Pico W に `do_connect.py` と `secrets.py` スクリプトがない場合は、 *1.* **ネットワークへのアク**

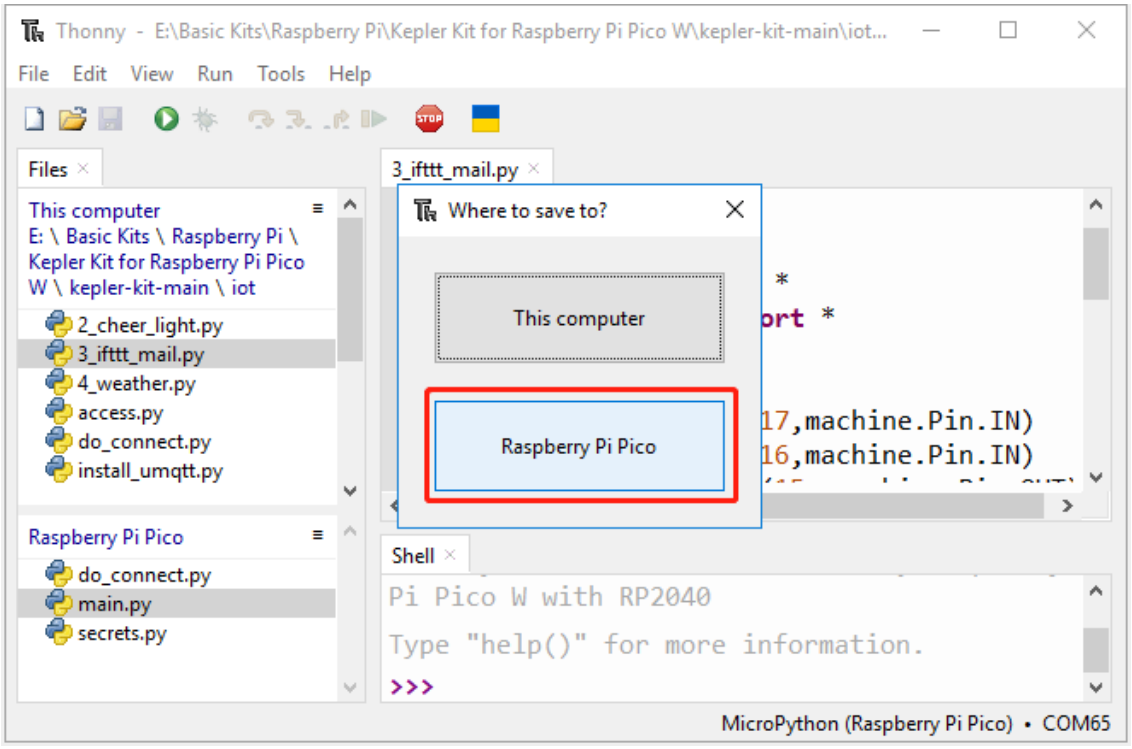
セス を参照してそれらを作成してください。

```
secrets = {
'ssid': 'SSID',
'password': 'PASSWORD',
'webhooks_key': 'WEBHOOKS_API_KEY'
}
```

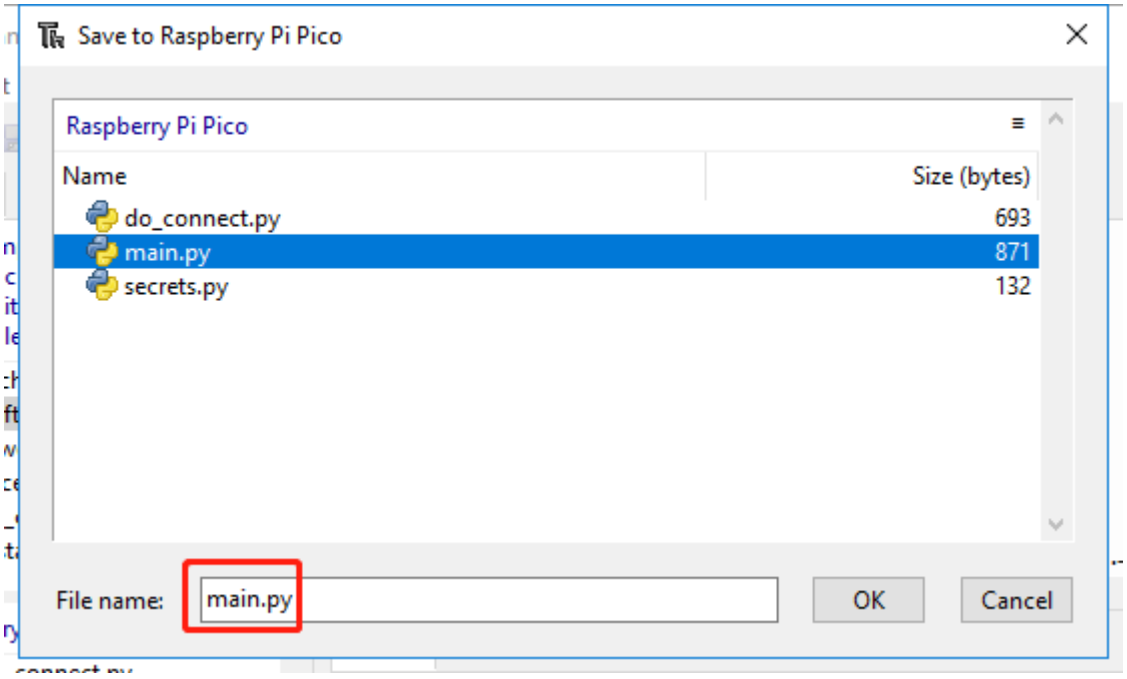
3. kepler-kit-main/iot ディレクトリ下の 3_ifttt_mail.py ファイルを開き、 **File（ファイル）** -> **Save as（名前を付けて保存）** をクリックするか、 Ctrl+Shift+S を押します。



4. ポップアップウィンドウで **Raspberry Pi Pico** を選択します。



5. ファイル名を `main.py` に設定します。同じファイルがすでに Pico W に存在する場合は、プロンプトが表示されます。



6. USB ケーブルを抜いて、Li-po チャージャーモジュールで Raspberry Pi Pico W に電源を供給できるようにします。スクリプトが実行されているときにブザーの音が聞こえます。PIR モジュールが通り過ぎる人物/生物を検出すると、ブザーが鳴り続け、警告のメールが送信されます。ボタンを押してスクリプトを再起動

できます。

仕組みについて

Raspberry Pi Pico W はインターネットに接続する必要があります。その詳細は *1.* **ネットワークへのアクセス** に記載されています。このプロジェクトでは、それをそのまま使用します。

```
from do_connect import *
do_connect()
```

PIR モジュールからのデータを読み取り、何かが通り過ぎると検出された場合には motion_detected() 関数を呼び出します（データは 0 から 1）。

```
import machine

sensor = machine.Pin(17, machine.Pin.IN)

sensor.irq(trigger = machine.Pin.IRQ_RISING, handler = motion_detected)
```

次に Pico W が IFTTT にデータを送信します。IFTTT に送信する message は URL の文字列であることがわかります。IFTTT は secrets['webhooks_key'] で送信者を識別し、 event でトリガーされるイベントを識別します。したがって、それらが正確であることを確認してください。

```
import urequests
from secrets import *

event = 'SecurityWarning'
message = f"https://maker.ifttt.com/trigger/{event}/with/key/{secrets['webhooks_key']}"

def motion_detected(pin):
    urequests.post(message)
    print(message)
    global warn_flag
    warn_flag = True
    sensor.irq(handler = None)
```

motion_detected() が呼び出されると、変数 warn_flag が True に設定され、ブザーが鳴り続けます。

```
while True:
    if warn_flag == True:
        alarm.toggle()
        time.sleep_ms(50)
```

ここでのボタンの目的は、スクリプトを再起動することです。

```
button = machine.Pin(16, machine.Pin.IN)

def reset_device(pin):
    machine.reset()

button.irq(trigger = machine.Pin.IRQ_RISING, handler = reset_device)
```

5.4 4. @OpenWeatherMap からのリアルタイム天気情報

このプロジェクトでは、LCD に時間と一緒に現在の天気を表示するスマートクロックを作成します。

1. 必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

一式をまとめて購入するのが便利です。リンクはこちら：

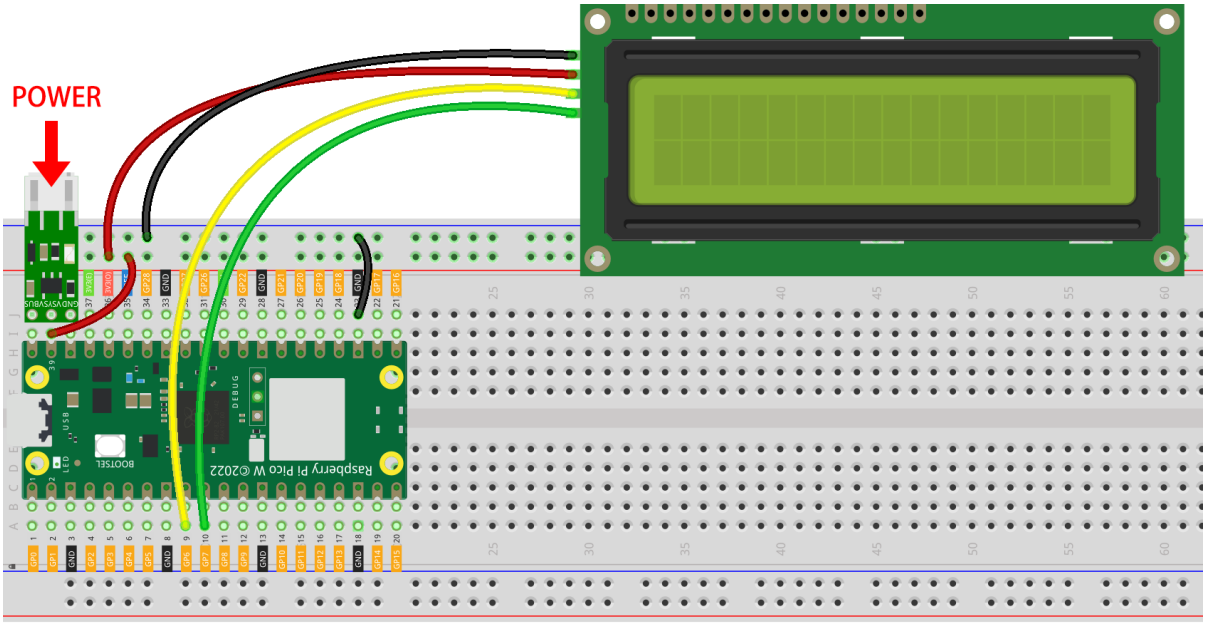
名称	このキットに含まれるアイテム数	リンク
ケプラーキット	450+	

下記のリンクから個別にも購入可能です。

SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*I2C LCD1602*	1	
6	*Li-po* **充電モジュール**	1	
7	18650 バッテリー	1	
8	バッテリーホルダー	1	

2. 回路を作成する

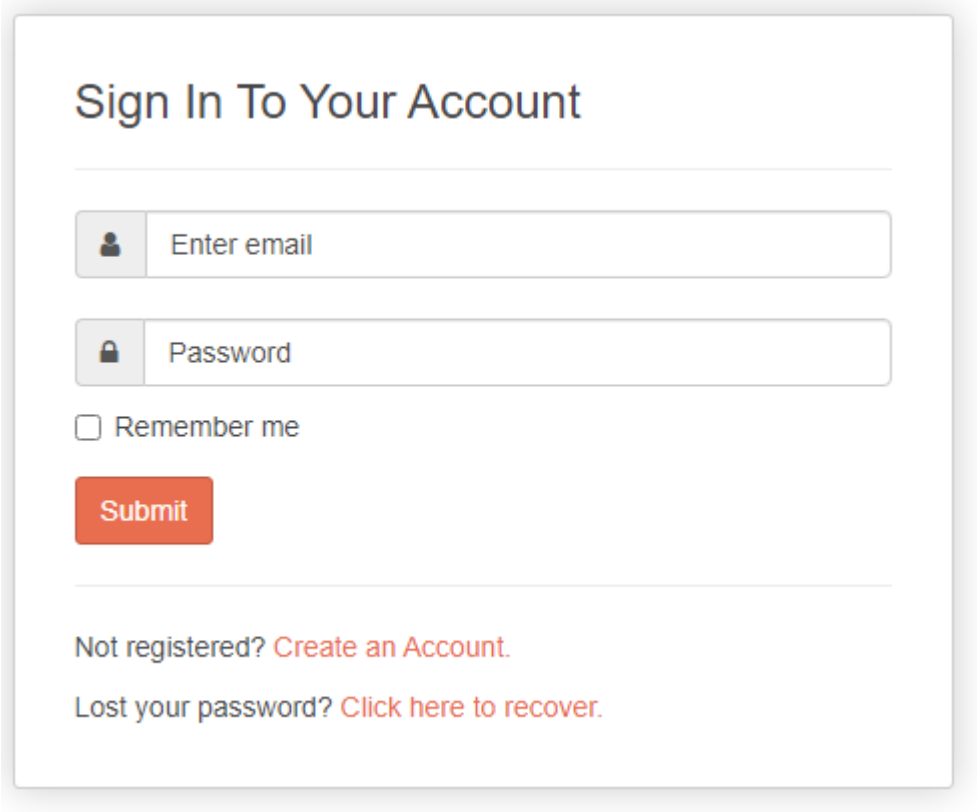
> **警告:** Li-po 充電モジュールが図の通りに接続されていることを確認してください。それ以外の場合、ショートが起きてバッテリーや回路が損傷する可能性があります。



3. OpenWeather の API キーを取得する

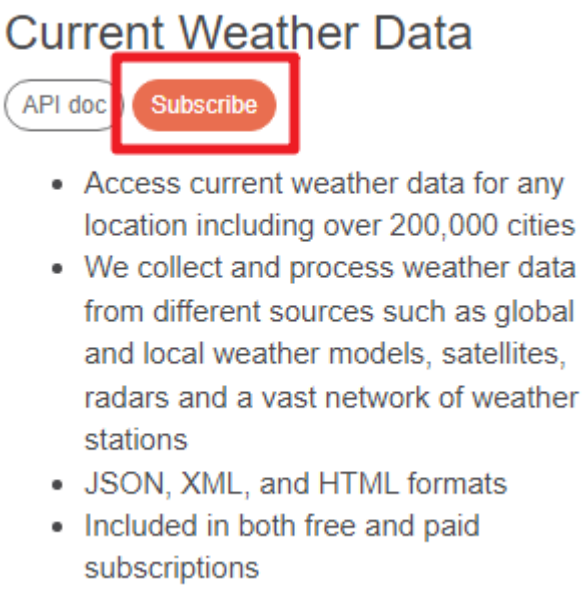
は、OpenWeather Ltd が運営するオンラインサービスで、API を介して全世界の天気データを提供しています。現在の天気、予報、短期予報、過去の天気データなど、地理的な位置に関係なく取得できます。

1. にアクセスしてログイン／アカウントを作成します。

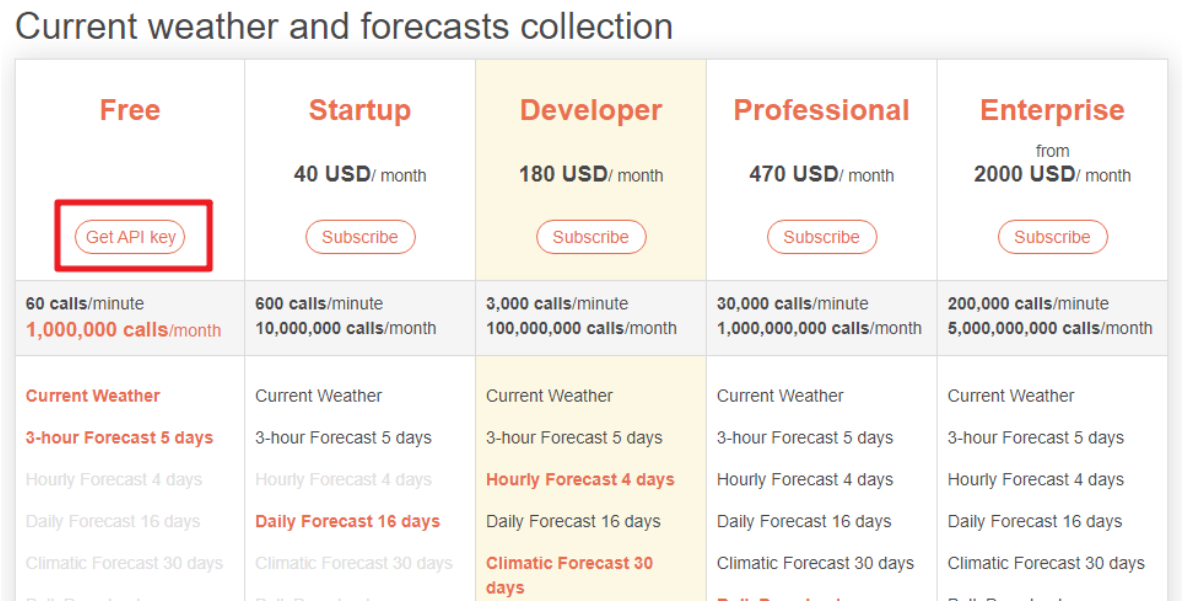


2. ナビゲーションバーから API ページに移動します。

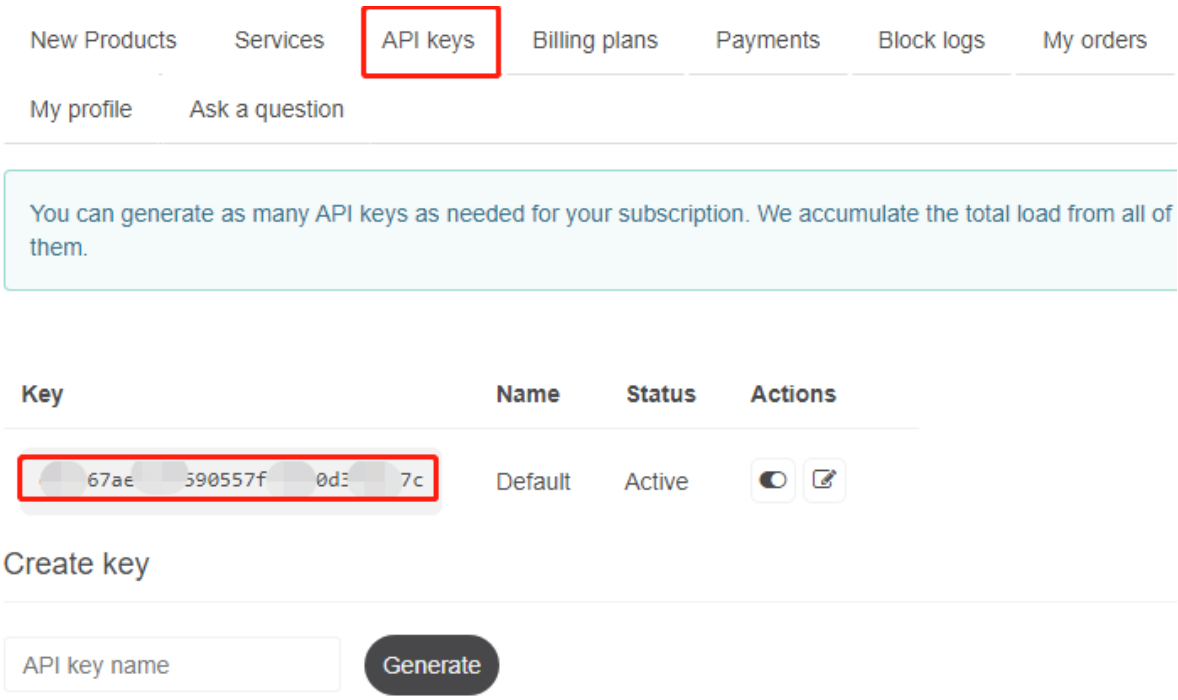
3. **現在の天気データ** を見つけて、サブスクライブをクリックします。



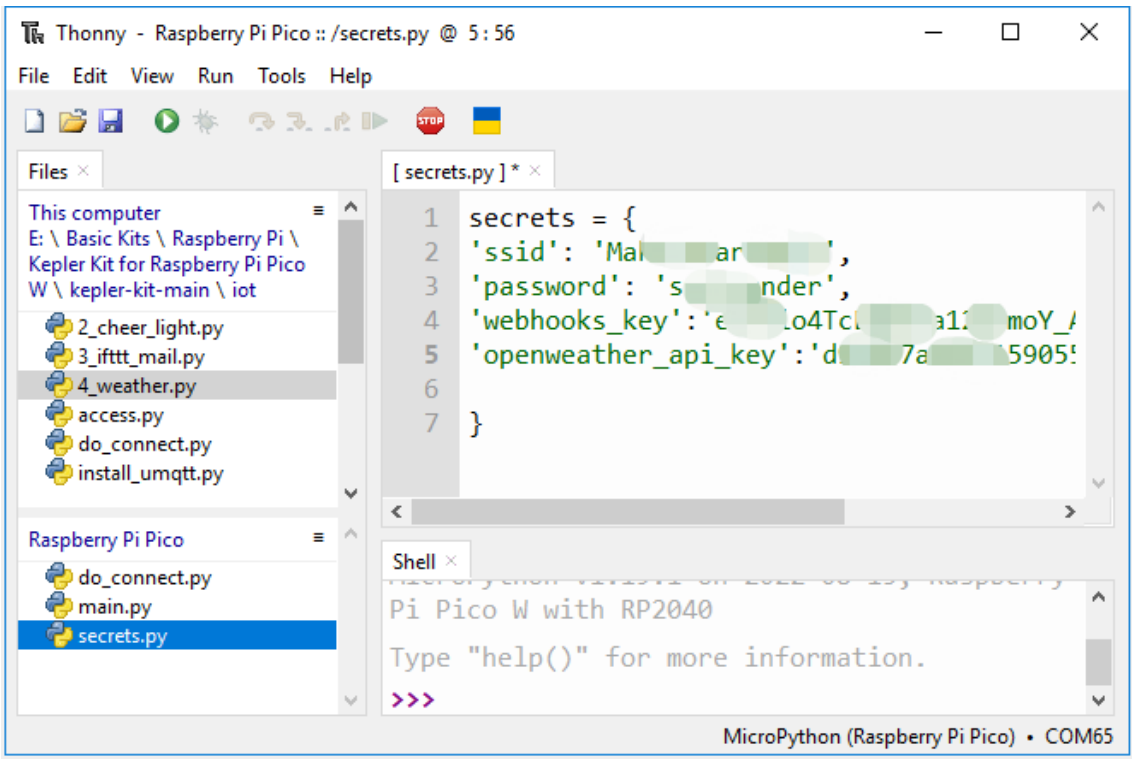
4. **現在の天気と予報コレクション** の下で、適切なサービスにサブスクライブします。このプロジェクトでは、無料プランで十分です。



5. **API キー** ページからキーをコピーします。



6. それを Raspberry Pi Pico W の `secrets.py` スクリプトにコピーします。



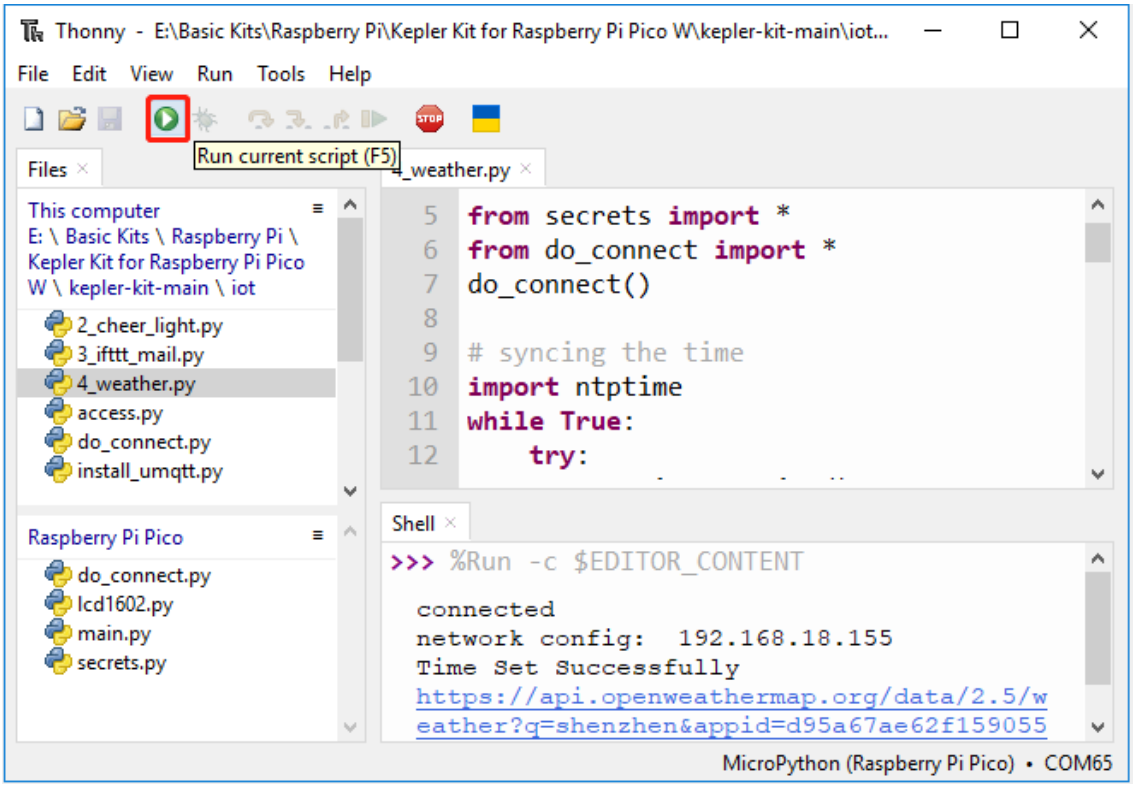
注釈: もし Pico W に `do_connect.py` および `secrets.py` スクリプトがない場合、 *1.* **ネットワーク**

へのアクセス を参照して作成してください。

```
secrets = {
'ssid': 'SSID',
'password': 'PASSWORD',
'webhooks_key':'WEBHOOKS_API_KEY',
'openweather_api_key':'OPENWEATHERMAP_API_KEY'
}
```

4. スクリプトを実行する

1. kepler-kit-main/iot パスの下にある 4_weather.py ファイルを開き、 **現在のスクリプトを実行** ボタンをクリックするか、F5 キーを押して実行します。



2. スクリプトが実行された後、I2C LCD1602 にあなたの地域の時間と天気情報が表示されます。

注釈: コードが実行中で画面が真っ白な場合は、モジュールの背面にあるポテンショメータを回してコントラストを調整できます。

3. このスクリプトを起動時に自動的に実行させたい場合は、それを Raspberry Pi Pico W に main.py として保存できます。

動作原理は？

Raspberry Pi Pico W は、 *1.* **ネットワークへのアクセス** で説明されているように、インターネットに接続する必要があります。このプロジェクトでは、そのまま使用します。

```
from do_connect import *
do_connect()
```

インターネットに接続した後、以下の数行のコードで Pico W の時刻をグリニッジ標準時に同期します。

```
import ntptime
while True:
   try:
      ntptime.settime()
      print('Time Set Successfully')
      break
   except OSError:
      print('Time Setting...')
      continue
```

LCD を初期化するには、 *3.4* **液晶ディスプレイ** を参照して、使用方法の詳細を確認してください。

```
from lcd1602 import LCD
lcd=LCD()
lcd.clear()
string = 'Loading...'
lcd.message(string)
```

天気データ（例：気温、風速）を取得する前に、単位を選択する必要があります。この場合、単位は `metric` です。

```
# Open Weather
TEMPERATURE_UNITS = {
   "standard": "K",
   "metric": "°C",
   "imperial": "°F",
}

SPEED_UNITS = {
   "standard": "m/s",
   "metric": "m/s",
   "imperial": "mph",
}
```

(次のページに続く)

(前のページからの続き)

```
units = "metric"
```

次に、この関数は openweathermap.org から天気データを取得します。都市名、API キー、設定された単位で URL メッセージを投稿します。結果として、天気データが含まれる JSON ファイルを受け取ります。

```
def get_weather(city, api_key, units='metric', lang='en'):
    '''
    Get weather data from openweathermap.org
        city: City name, state code and country code divided by comma, Please, refer to␣
↪ISO 3166 for the state codes or country codes. https://www.iso.org/obp/ui/#search
        api_key: Your unique API key (you can always find it on your openweather account␣
↪page under the "API key" tab https://home.openweathermap.org/api_keys)
        unit: Units of measurement. standard, metric and imperial units are available. If␣
↪you do not use the units parameter, standard units will be applied by default. More:␣
↪https://openweathermap.org/current#data
        lang: You can use this parameter to get the output in your language. More: https://
↪openweathermap.org/current#multi
    '''
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&
↪units={units}&lang={lang}"
    print(url)
    res = urequests.post(url)
    return res.json()
```

この一連の生データを出力すると、以下に示すような情報が表示されます。

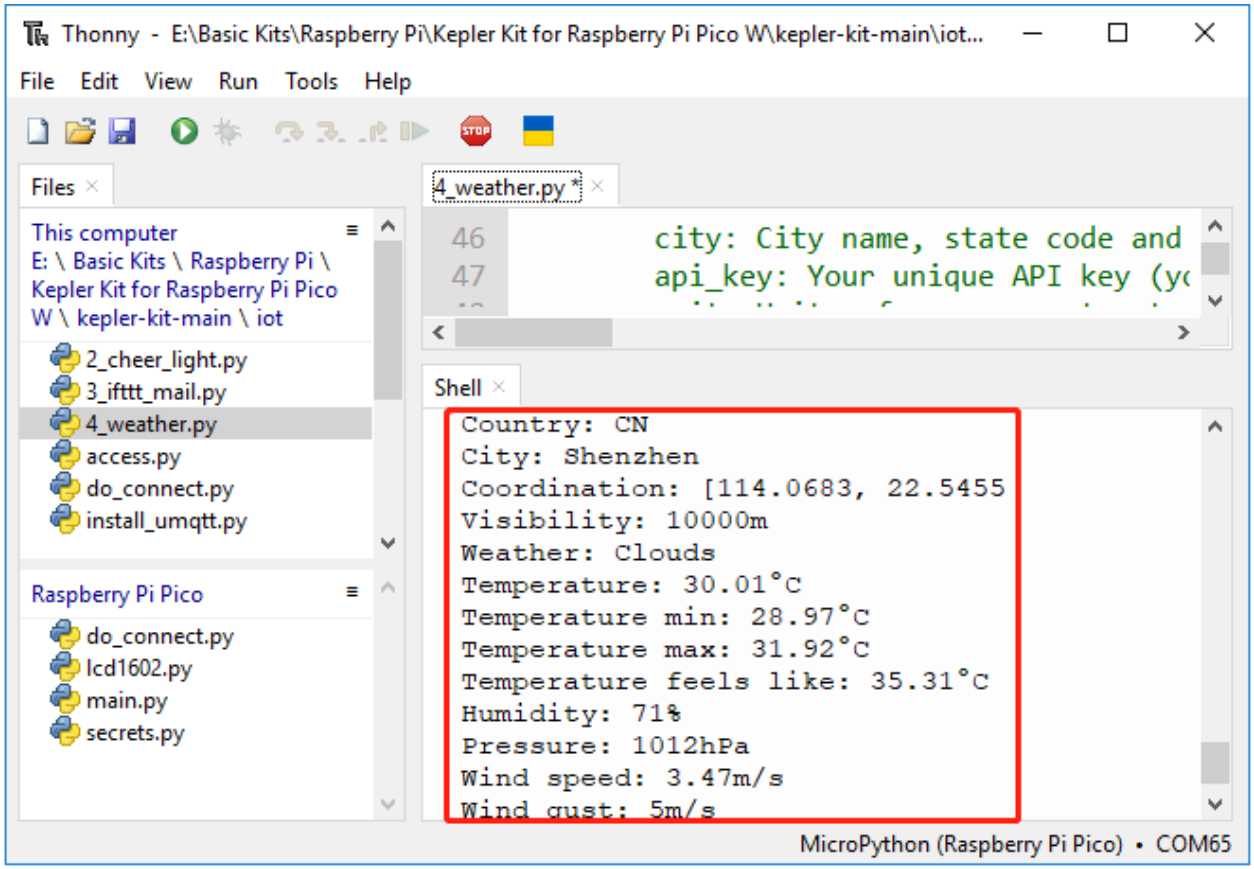
```
weather data example:
{
    'timezone': 28800,
    'sys': {
        'type': 2,
        'sunrise': 1659650200,
        'country': 'CN',
        'id': 2031340,
        'sunset': 1659697371
    },
    'base': 'stations',
    'main': {
```

(次のページに続く)

(前のページからの続き)

```
        'pressure': 1008,
        'feels_like': 304.73,
        'temp_max': 301.01,
        'temp': 300.4,
        'temp_min': 299.38,
        'humidity': 91,
        'sea_level': 1008,
        'grnd_level': 1006
    },
    'visibility': 10000,
    'id': 1795565,
    'clouds': {
        'all': 96
    },
    'coord': {
        'lon': 114.0683,
        'lat': 22.5455
    },
    'name': 'Shenzhen',
    'cod': 200,
    'weather':[{
        'id': 804,
        'icon': '04d',
        'main': 'Clouds',
        'description': 'overcast clouds'
    }],
    'dt': 1659663579,
    'wind': {
        'gust': 7.06,
        'speed': 3.69,
        'deg': 146
    }
}
```

`print_weather(weather_data)` 関数を使って、これらの生データを読みやすい形式に変換して出力します。この関数は呼び出されていませんが、 `while True` 内で必要に応じてこの行をコメント解除できます。




```
# シェル出力
print_weather(weather_data)
```

while True ループでは、最初に get_weather() 関数が呼び出され、このプロジェクトに必要な weather 、temperature 、 humidity 情報が取得されます。

```
weather_data = get_weather('shenzhen', secrets['openweather_api_key'], units=units)
weather=weather_data["weather"][0]["main"]
t=weather_data["main"]["temp"]
rh=weather_data["main"]["humidity"]
```

現地時間を取得します。ここでは time.localtime() 関数が呼び出され、タプル（年、月、日、時間、分、秒、曜日、年日）のセットが返されます。この中から hour と minute を取り出しています。

すでに Pico W はグリニッジ標準時に同期されているため、あなたの地域のタイムゾーンを加える必要があります。

```
# 時間の取得（+24 は西半球用）
# 負の場合は 24 を加える
# hours = time.localtime()[3] + int(weather_data["timezone"] / 3600) + 24  # 西半球のみ
```

(次のページに続く)

(前のページからの続き)

```
hours = time.localtime()[3] + int(weather_data["timezone"] / 3600)
mins = time.localtime()[4]
```

最終的に、天気情報と時刻は LCD1602 に表示されます。

```
lcd.clear()
time.sleep_ms(200)
string = f'{hours:02d}:{mins:02d} {weather}\n'
lcd.message(string)
string = f'{t}{TEMPERATURE_UNITS[units]} {rh}%rh'
lcd.message(string)
```

メインループが 30 秒ごとに実行されると、LCD1602 は 30 秒ごとにリフレッシュする時計になります。

5.5 5. @MQTT を使用したクラウド呼び出しシステム

Message Queuing Telemetry Transport（MQTT）はシンプルなメッセージングプロトコルです。また、IoT（Internet of Things）の最も一般的なメッセージングプロトコルでもあります。

MQTT プロトコルは、IoT デバイスがデータを転送する方法を定義します。これらはイベント駆動であり、Pub/Sub モデルを使用して相互接続されています。送信者（Publisher）と受信者（Subscriber）は、トピックを介して通信します。デバイスが特定のトピックでメッセージを公開すると、そのトピックに登録されたすべてのデバイスがメッセージを受け取ります。

このセクションでは、Pico W、HiveMQ（無料の公開 MQTT ブローカーサービス）、および 4 つのボタンを使用してサービスベルシステムを作成します。4 つのボタンはレストランの 4 つのテーブルを意味し、顧客がボタンを押すと HiveMQ でどのテーブルのゲストがサービスが必要かが表示されます。

1. 必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

全体のキットを購入するのが確かに便利です、リンクはこちら：

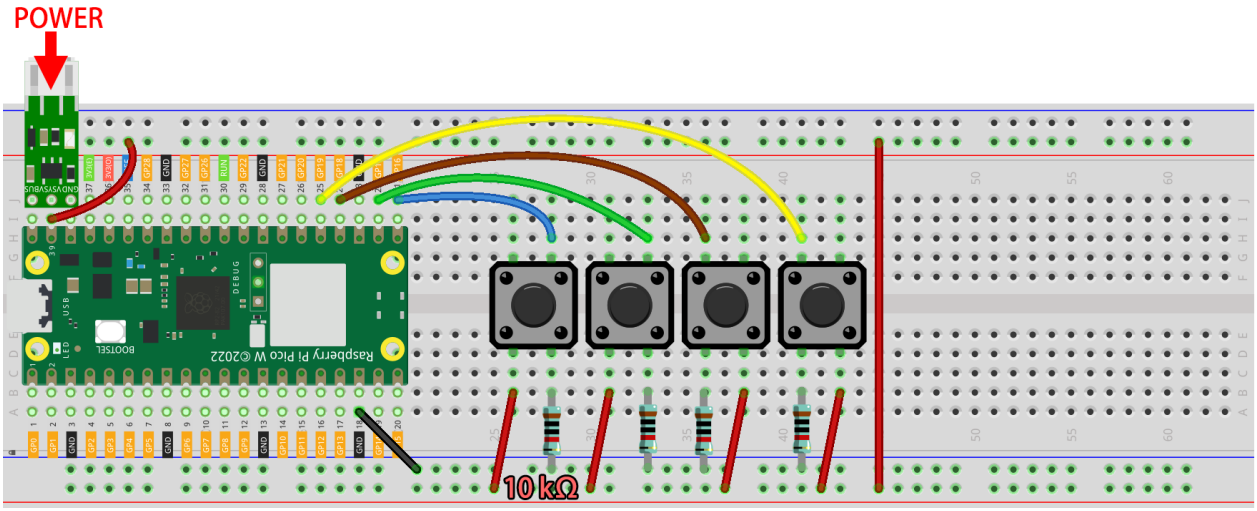
名前	このキットに含まれるアイテム	リンク
Kepler Kit	450 以上	

以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(10K Ω)	
6	**ボタン**	4	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	

2. 回路を組み立てる

> **警告:** 図に示されているように Li-po 充電モジュールが接続されていることを確認してください。それ以外の場合、短絡が原因でバッテリーや回路が壊れる可能性があります。

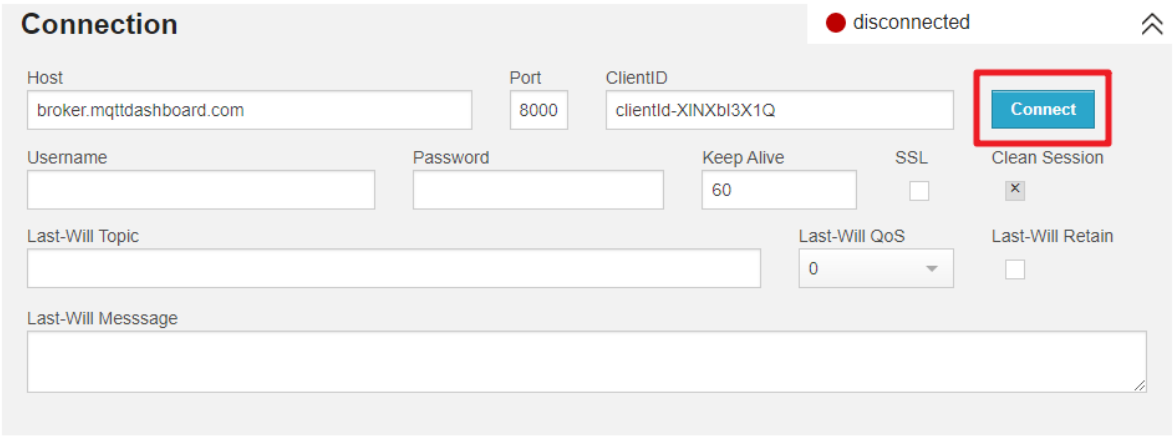


3. HiveMQ にアクセス

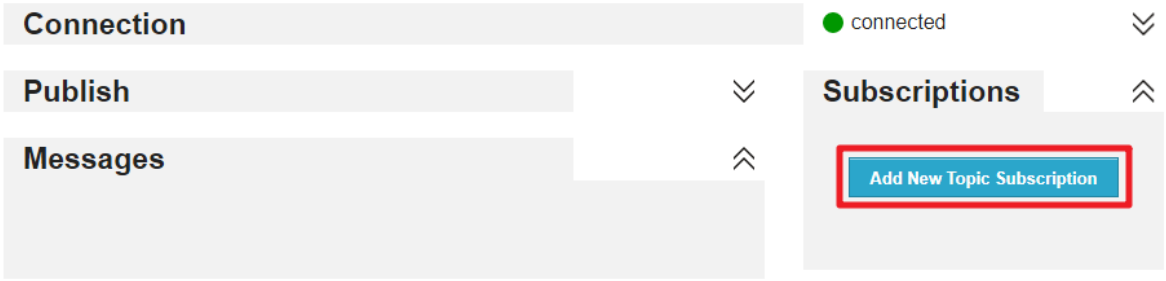
HiveMQ は、IoT デバイスへの迅速かつ効率的な信頼性の高いデータ転送を可能にする MQTT ブローカーおよびクライアントベースのメッセージングプラットフォームです。

1. ブラウザで を開きます。

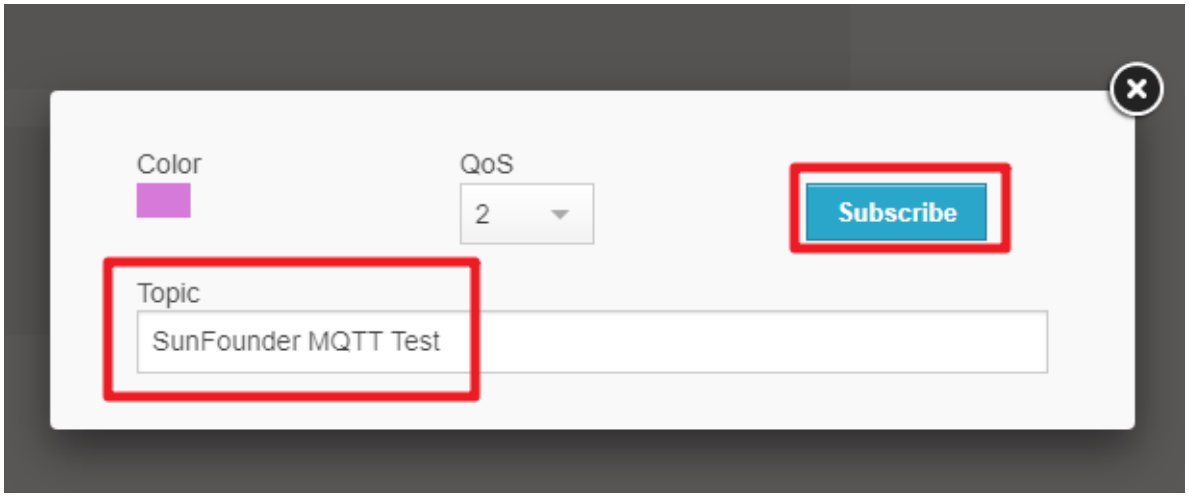
2. クライアントをデフォルトの公開プロキシに接続します。



3. **新しいトピックの購読を追加** をクリックします。



4. フォローしたいトピックを入力し、 **購読** をクリックします。ここで設定するトピックは、他のユーザーからメッセージを受け取らないように、より個別化されるべきです。また、大文字と小文字に注意してください。



4. MQTT モジュールのインストール

プロジェクトを開始する前に、Pico W 用の MQTT モジュールをインストールする必要があります。

1. 既に書いた do_connect() を Shell で実行してネットワークに接続します。

注釈:

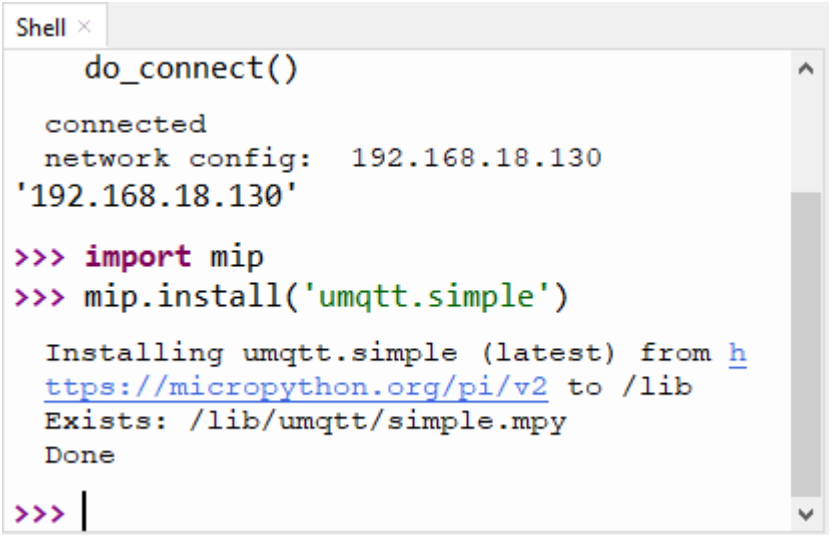
- 以下のコマンドを Shell に入力し、 Enter を押して実行します。
- もし Pico W に do_connect.py と secrets.py スクリプトがない場合は、 *1.* **ネットワークへのアクセス** を参照して作成してください。

```
from do_connect import *
do_connect()
```

2. 成功したネットワーク接続の後、シェルで mip モジュールをインポートし、MicroPython 用の簡易化された MQTT クライアントである umqtt.simple モジュールを mip でインストールします。

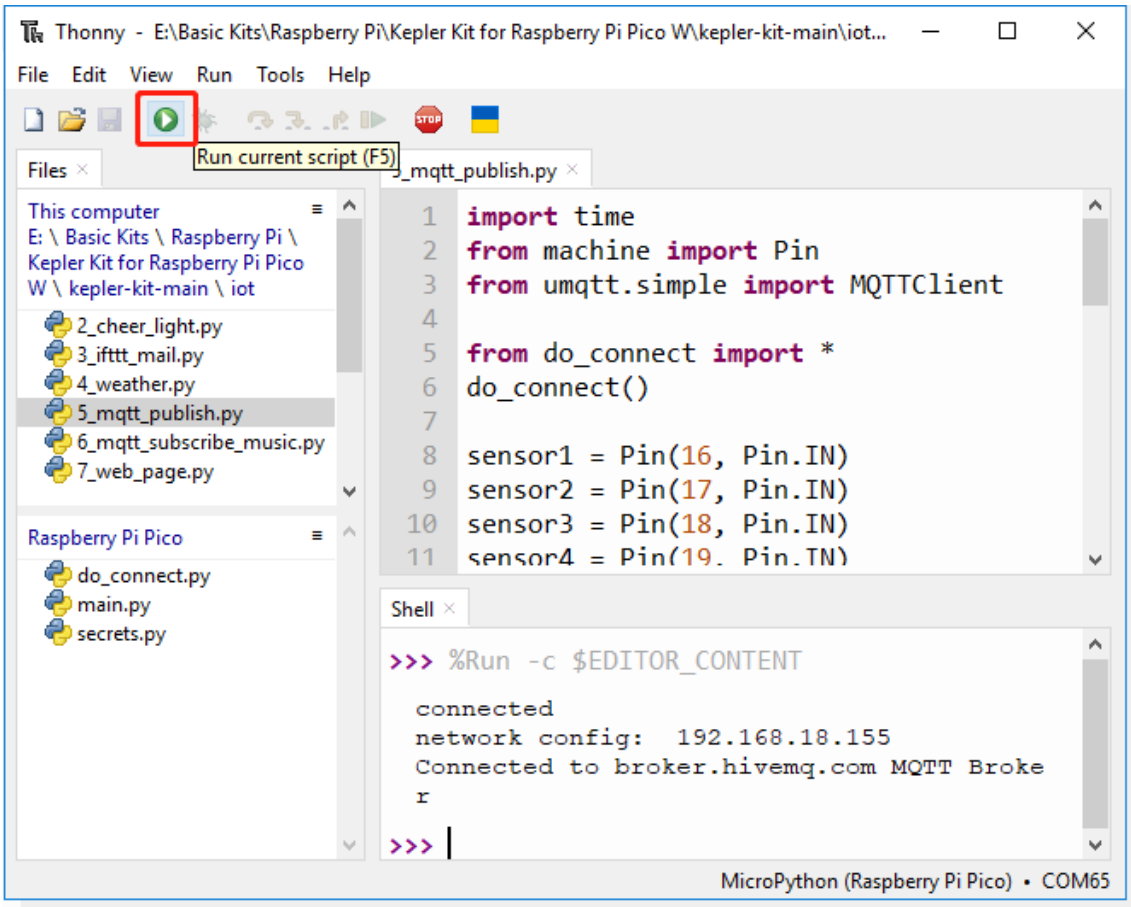
```
import mip
mip.install('umqtt.simple')
```

3. インストールが完了したら、 umqtt モジュールが Pico W の /lib/ パスにインストールされていることがわかります。

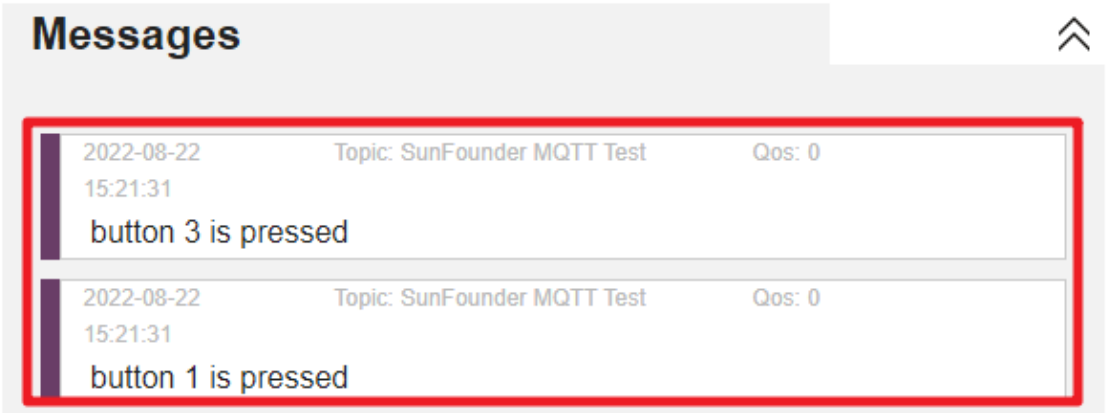


5. スクリプトを実行する

1. kepler-kit-main/iot のパス下で 5_mqtt_publish.py ファイルを開きます。
2. **現在のスクリプトを実行する** ボタンをクリックするか、F5 キーを押して実行します。



3. 再度 に戻って、ブレッドボード上のボタンのいずれかを押すと、HiveMQ 上でメッセージプロンプトが表示されます。



4. このスクリプトが起動できるようにしたい場合、それを Raspberry Pi Pico W に `main.py` として保存できます。

動作の仕組みは？

Raspberry Pi Pico W は、 *1.* **ネットワークへのアクセス** で説明されているように、インターネットに接続する必要があります。このプロジェクトではそのまま使用します。

```
from do_connect import *
do_connect()
```

4 つのボタンピンを初期化します。

```
sensor1 = Pin(16, Pin.IN)
sensor2 = Pin(17, Pin.IN)
sensor3 = Pin(18, Pin.IN)
sensor4 = Pin(19, Pin.IN)
```

MQTT ブローカーに接続するために使用する URL と client ID を保存するための 2 つの変数を作成します。公開ブローカーを使用しているので、 client ID は必須ではありません。

```
mqtt_server = 'broker.hivemq.com'
client_id = 'Jimmy'
```

MQTT エージェントに接続し、1 時間保持します。失敗した場合は、Pico W をリセットします。

```
try:
    client = MQTTClient(client_id, mqtt_server, keepalive=3600)
    client.connect()
    print('Connected to %s MQTT Broker'%(mqtt_server))
except OSError as e:
    print('Failed to connect to the MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset()
```

変数 topic を作成します。これは、購読者がフォローする必要があるトピックです。それは上記の **2. HiveMQ に訪れる** の **ステップ 4** で記入したトピックと同じでなければなりません。ちなみに、ここでの b は、MQTT がバイナリベースのプロトコルであるため、文字列をバイトに変換します。

```
topic = b'SunFounder MQTT Test'
```

各ボタンに割り込みを設定します。ボタンが押されたら、 topic の下にメッセージが投稿されます。

```
def press1(pin):
    message = b'button 1 is pressed'
    client.publish(topic, message)
    print(message)

sensor1.irq(trigger=machine.Pin.IRQ_RISING, handler=press1)
```

- UMQTT クライアント API

5.6 6. @MQTT を使ったクラウドプレイヤー

このプロジェクトを始める前に、 *5. @MQTT* **を使用したクラウド呼び出しシステム** プロジェクトを先に完了させて、いくつかのモジュールのインストールと HiveMQ プラットフォームの設定を完了することをお勧めします。

このプロジェクトでは、Pico W は購読者として動作し、トピックの下で曲名を受信します。曲名がコード内に既に存在する場合、Pico W はブザーでその曲を演奏します。

1. 必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

こちらのリンクで一式を購入すると便利です：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

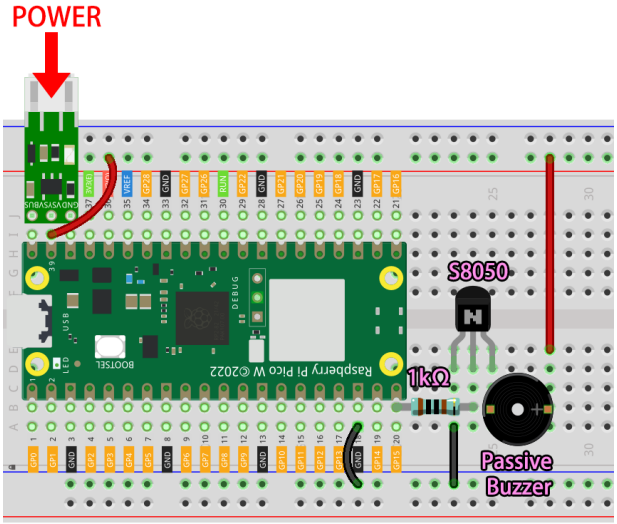
以下のリンクから個別に購入することもできます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	受動型 **ブザー**	1	
8	*Li-po* **充電モジュール**	1	
9	18650 バッテリー	1	
10	バッテリーホルダー	1	

2. 回路を作成する

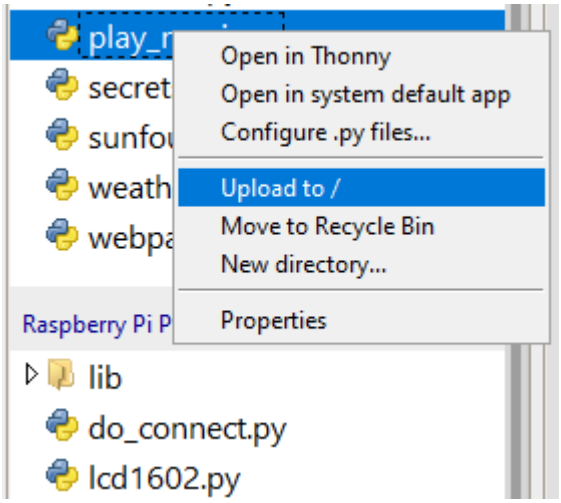
キットには 2 つのブザーが含まれていますが、ここでは受動ブザー（裏側に露出した PCB があるもの）を使用します。このブザーはトランジスタが必要です。ここでは S8050 を使用します。

> **警告:** ダイアグラムに示されているように、Li-po チャージャーモジュールを接続してください。そうしないと、ショートしてバッテリーや回路が損傷する可能性があります。

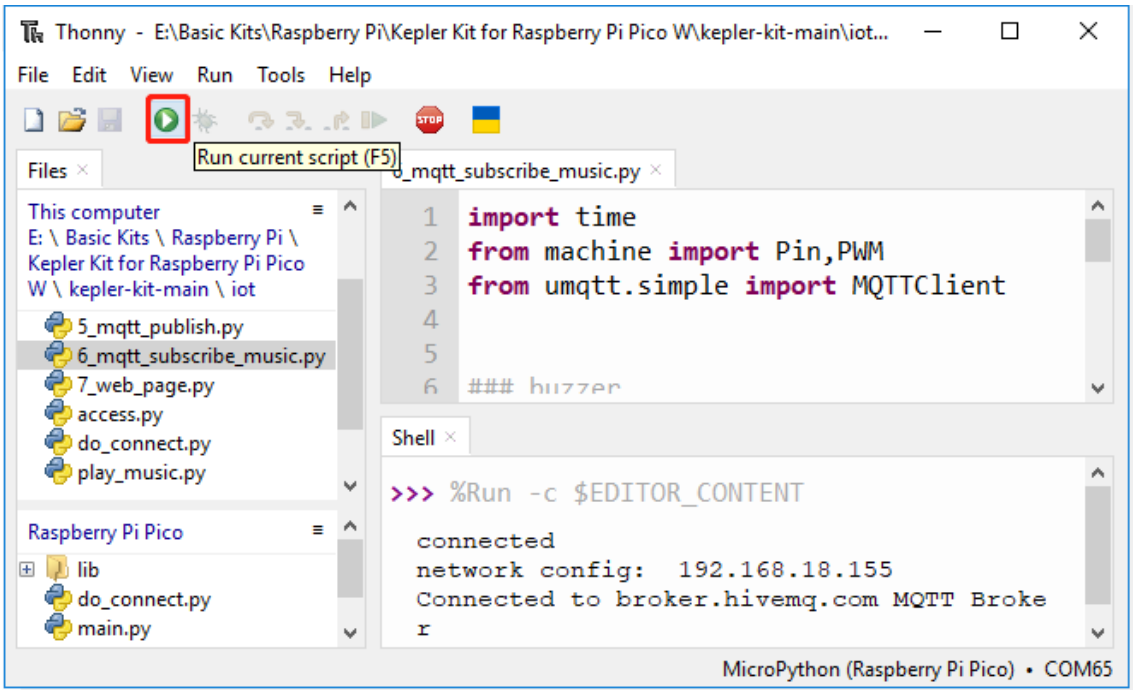


3. コードを実行する

1. `kepler-kit-main/iot` のパスで `play_music.py` ファイルを Raspberry Pi Pico W にアップロードします。



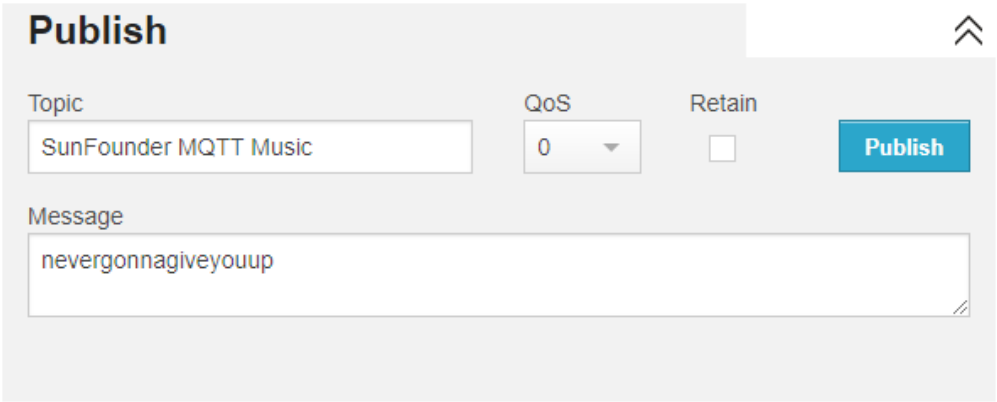
2. `kepler-kit-main/iot` のパスで `6_mqtt_subscribe_music.py` ファイルを開き、 **現在のスクリプトを実行** ボタンをクリックするか、F5 を押して実行します。



注釈: コードを実行する前に、Pico W に do_connect.py および secrets.py スクリプトが存在することを確認してください。ない場合は、 *1.* **ネットワークへのアクセス** を参照して作成してください。

3. ブラウザで を開き、「トピック」欄に SunFounder MQTT Music 、 **メッセージ** 欄に曲名を入力します。**Publish** ボタンをクリックすると、Pico W に接続されたブザーが対応する曲を演奏します。

注釈: play_music.py には nokia , starwars , nevergonnagiveyouup , gameofthrone , songofstorms , zeldatheme , harrypotter が含まれています。



4. このスクリプトを起動可能にしたい場合、Raspberry Pi Pico W に main.py として保存できます。

どうやって動作するのか？

理解しやすくするために、MQTT のコードは他の部分から分離されています。その結果、MQTT の購読に関する最も基本的な機能を 3 か所で実装する以下のコードが得られます。

```
import time
from umqtt.simple import MQTTClient

from do_connect import *
do_connect()

mqtt_server = 'broker.hivemq.com'
client_id = 'Jimmy'

# to subscribe the message
topic = b'SunFounder MQTT Music'

def callback(topic, message):
    print("New message on topic {}".format(topic.decode('utf-8')))
    message = message.decode('utf-8')
    print(message)

try:
    client = MQTTClient(client_id, mqtt_server, keepalive=60)
    client.set_callback(callback)
    client.connect()
    print('Connected to %s MQTT Broker'%(mqtt_server))
except OSError as e:
    print('Failed to connect to MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset()

while True:
    client.subscribe(topic)
    time.sleep(1)
```

MQTT ブローカーに接続する際、 `client.set_callback(callback)` 関数を呼び出して、受信した購読メッセージのコールバックとして機能させます。

```
try:
    client = MQTTClient(client_id, mqtt_server, keepalive=60)
    client.set_callback(callback)
```

(次のページに続く)

(前のページからの続き)

```
    client.connect()
    print('Connected to %s MQTT Broker'%(mqtt_server))
except OSError as e:
    print('Failed to connect to MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset()
```

次に、フェッチされたトピックからのメッセージを出力するコールバック関数です。MQTT はバイナリベースのプロトコルであり、制御要素もバイナリバイトです。したがって、これらのメッセージは `message.decode('utf-8')` を使用してデコードする必要があります。

```
def callback(topic, message):
    print("New message on topic {}".format(topic.decode('utf-8')))
    message = message.decode('utf-8')
    print(message)
```

While True ループを使用して、このトピックに定期的にメッセージを取得します。

```
while True:
    client.subscribe(topic)
    time.sleep(1)
```

次に、音楽が演奏されます。この関数は play_music.py スクリプトに配置され、主に 3 つの部分で構成されています。

- Tone : 基礎となる に基づいて特定の音をシミュレートします。

  ```
  NOTE_B0 =  31
  NOTE_C1 =  33
  ...
  NOTE_DS8 = 4978
  REST =      0
  ```

- Score : プログラムが使用できる形式に楽曲を編集します。これらの楽譜は Robson Couto の無料共有からです。

  ```
  song = {
      "nokia":[NOTE_E5, 8, NOTE_D5, 8, NOTE_FS4, 4, NOTE_GS4, 4, NOTE_CS5, 8, NOTE_
  ↪B4, 8, NOTE_D4, 4,
                  NOTE_E4, 4,NOTE_B4, 8, NOTE_A4, 8, NOTE_CS4, 4, NOTE_E4, 4, NOTE_
  ```

(次のページに続く)

(前のページからの続き)

```
↪A4, 2],
    "starwars":[,,,],
    "nevergonnagiveyouup":[,,,],
    "gameofthrone":[,,,],
    "songofstorms":[,,,],
    "zeldatheme":[,,,],
    "harrypotter":[,,,],
}
```

- Play : この部分は基本的に *3.2* **カスタムトーン** と同じですが、上記の楽譜に適合するようにわずかに最適化されています。

```
import time
import machine

tempo = 220

wholenote = (60000 * 4) / tempo

def tone(pin,frequency,duration):
    if frequency is 0:
        pass
    else:
        pin.freq(frequency)
        pin.duty_u16(30000)
    time.sleep_ms(duration)
    pin.duty_u16(0)

def noTone(pin):
    tone(pin,0,100)

def play(pin,melody):
    for thisNote in range(0,len(melody),2):
        divider = melody[thisNote+1]
        if divider > 0:
            noteDuration = wholenote/divider
        elif divider < 0:
            noteDuration = wholenote/-(divider)
            noteDuration *= 1.5
```

(次のページに続く)

(前のページからの続き)

```
        tone(pin,melody[thisNote],int(noteDuration*0.9))

        time.sleep_ms(int(noteDuration))

        noTone(pin)
```

メイン関数に戻って、MQTT が音楽の再生をトリガーするようにします。コールバック関数で、送信されたメッセージが含まれている曲の名前であるかどうかを判断します。そうであれば、曲名を変数 melody に割り当て、play_flag を True に設定します。

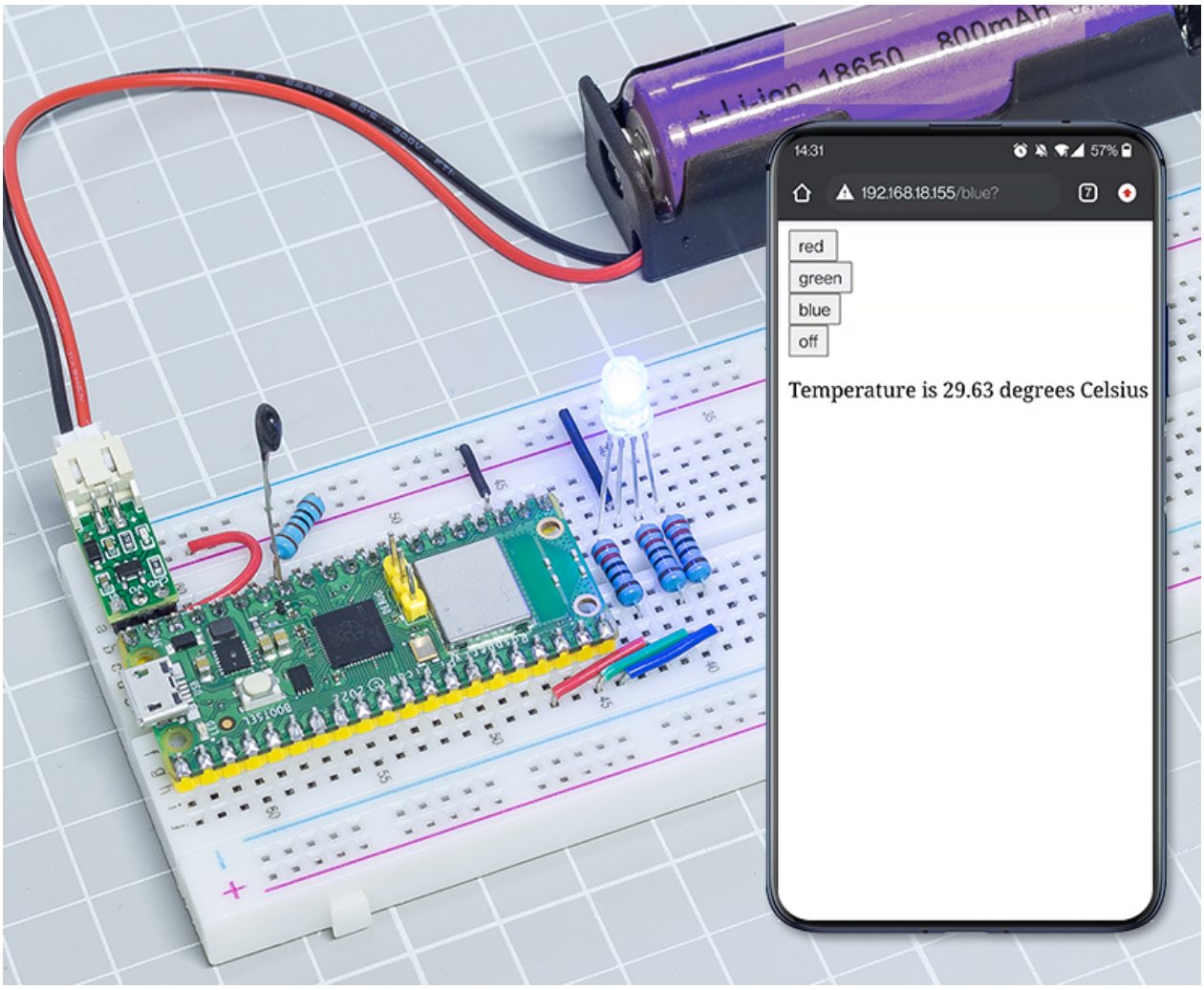
```
def callback(topic, message):
    print("New message on topic {}".format(topic.decode('utf-8')))
    message = message.decode('utf-8')
    print(message)
    if message in song.keys():
        global melody,play_flag
        melody = song[message]
        play_flag = True
```

メインループでは、 play_flag が True であれば、 melody を演奏します。

```
while True:
    client.subscribe(topic)
    time.sleep(1)
    if play_flag is True:
        play(buzzer,melody)
        play_flag = False
```

5.7 7. ウェブサーバーのセットアップ

この記事では、Pico W をブラウザから回路を操作したり、センサーからの読み取りを取得できるウェブサーバーにする方法を学びます。



1. 必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

一式を購入すると便利です。こちらがリンクです：

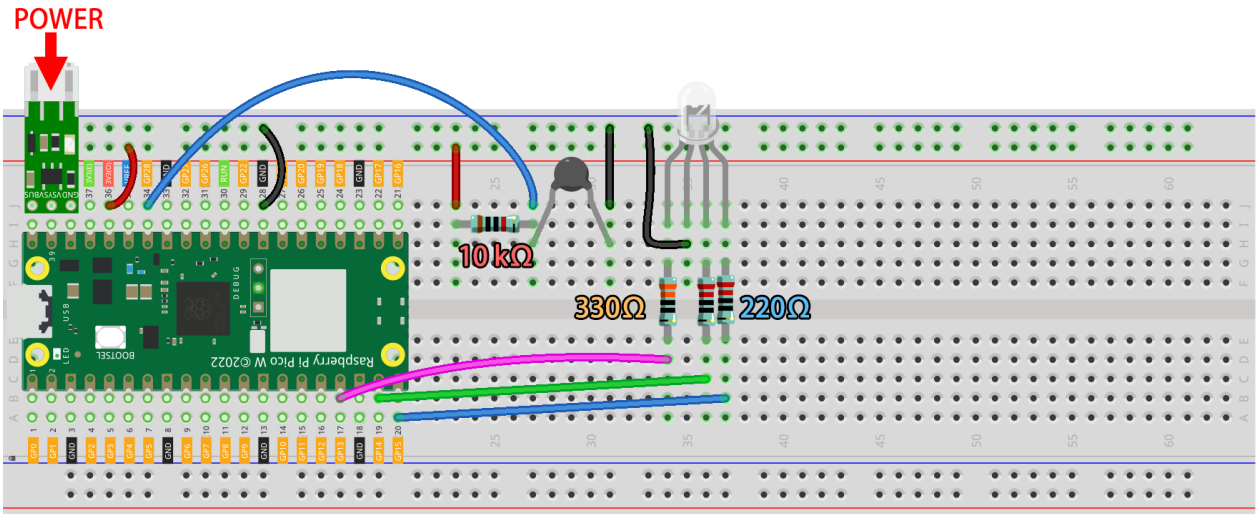
名前	このキットに含まれるアイテム	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(1-330 Ω, 2-220 Ω, 1-10K Ω)	
6	*RGB LED*	1	
7	**サーミスター**	1	
8	*Li-po* **充電モジュール**	1	
9	18650 バッテリー	1	
10	バッテリーホルダー	1	

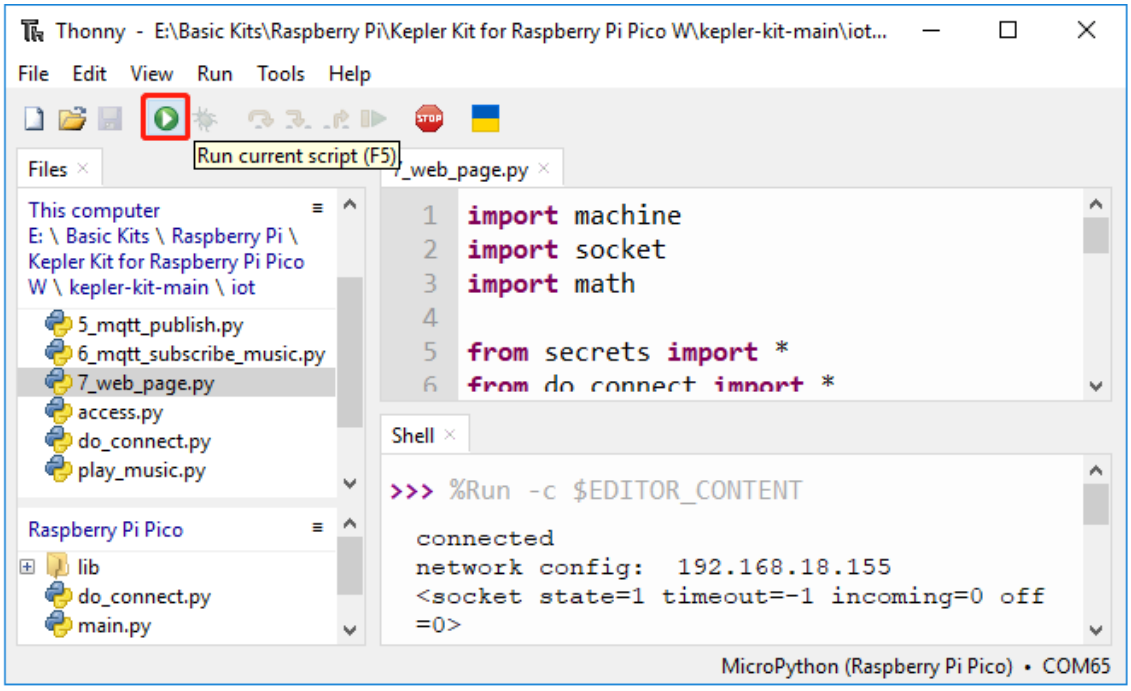
2. 回路を組み立てる

> **警告:** 図に示されている通り、Li-po チャージャーモジュールが接続されていることを確認してください。そうでないと、ショート回路が起きてバッテリーや回路が損傷する可能性があります。



3. コードを実行する

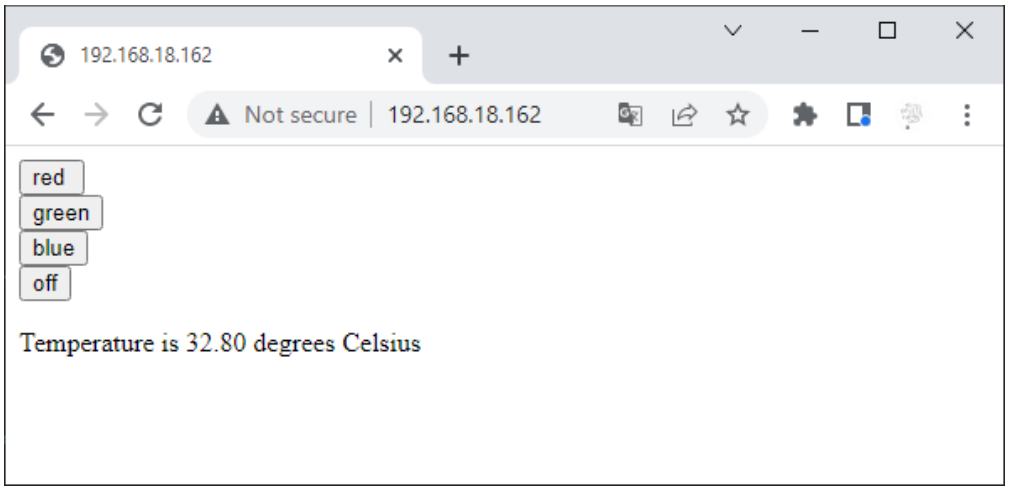
1. `kepler-kit-main/iot` のパスの下にある `7_web_page.py` ファイルを開きます。
2. **Run current script** ボタンをクリックするか、F5 キーを押して実行します。接続が成功すると、Pico W の IP が表示されます。



注釈: コードを実行する前に、Pico W に `do_connect.py` と `secrets.py` スクリプトがあることを確認してください。もしなければ、 *1.* **ネットワークへのアクセス** を参照して作成してください。

---

3. Pico W の IP アドレスをブラウザに入力して、このプロジェクトのために構築されたウェブページにアクセスします。任意のボタンをクリックして、RGB LED の色を変更し、温度と湿度を更新します。



4. このスクリプトを起動できるようにしたい場合は、Raspberry Pi Pico W に `main.py` として保存できます。

動作原理は？

アクセスしているウェブページは実際には何らかのサーバーでホストされており、そのサーバーのソケットが訪問時にウェブページを送信します。ソケットとは、サーバーが接続を希望するクライアントを待ち受ける方法です。

このプロジェクトでは、Pico W がサーバーであり、ブラウザを介して Pico W でホストされているウェブページにアクセスしています。

最初にソケットを作成します。これには IP アドレスと が必要です。ネットワーク接続と IP の取得方法は、 *1. ネットワークへのアクセス* で説明されています。ポートには 80 を使用します。ソケットの設定が完了したら、それを返して次のステップで使用します。

socket library - Python Docs

```
import socket

def open_socket(ip):
    # ソケットを開く
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)
    return(connection)
```

次に、以前に設定したソケットを使用してウェブサービスを設定します。以下のコードにより、Pico W はブラウザからのアクセス要求を受け取ることができます。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        client.close()
```

次に、訪問者に送信する html ページが必要です。この例では、変数 `html` に文字形式で単純な html ページを格納しています。

注釈: 自分で html を書きたい場合は、 でヘルプを得ることができます。

```
def webpage(value):
    html = f"""
            <!DOCTYPE html>
            <html>
            <body>
            <form action="./red">
            <input type="submit" value="red " />
            </form>
            <form action="./green">
            <input type="submit" value="green" />
            </form>
            <form action="./blue">
            <input type="submit" value="blue" />
            </form>
            <form action="./off">
            <input type="submit" value="off" />
            </form>
            <p>温度は{value}度です</p>
            </body>
            </html>
            """
    return html
```

訪問者に HTML ページを送信する。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        html = webpage(0)
        client.send(html)
        client.close()
```

上記の部分を組み合わせると、ブラウザでページにアクセスできます。効果を確認したい場合は、以下のコードを Thonny で実行してください。

```
import machine
import socket

from secrets import *
```

(次のページに続く)

(前のページからの続き)

```
from do_connect import *

def webpage(value):
    html = f"""
            <!DOCTYPE html>
            <html>
            <body>
            <form action="./red">
            <input type="submit" value="赤" />
            </form>
            <form action="./green">
            <input type="submit" value="緑" />
            </form>
            <form action="./blue">
            <input type="submit" value="青" />
            </form>
            <form action="./off">
            <input type="submit" value="オフ" />
            </form>
            <p>温度は{value}度です</p>
            </body>
            </html>
            """
    return html

def open_socket(ip):
    # ソケットを開く
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)
    return(connection)

def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        html = webpage(0)
```

(次のページに続く)

(前のページからの続き)

```
        client.send(html)
        client.close()

try:
    ip = do_connect()
    if ip is not None:
        connection = open_socket(ip)
        serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

上記のコードを実行すると、ウェブページのみが表示され、RGB LED の制御やセンサーの読み取りは許可されていないことがわかります。このウェブサービスはさらに洗練される必要があります。

最初に知るべきことは、ブラウザがウェブページにアクセスしたときにサーバーが受け取る情報です。それゆえに、 `serve()` をわずかに変更して `request` を出力します。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        print(request)
        html = webpage(0)
        client.send(html)
        client.close()
```

スクリプトを再実行すると、シェルはウェブページでキーを押すときに以下のメッセージを出力します。

```
b'GET /red? HTTP/1.1\r\nHost: 192.168.18.162\r\nConnection: keep-alive.......q=0.5\r\n\r\
↪n'
b'GET /favicon.ico HTTP/1.1\r\nHost: 192.168.18.162\r\nConnection: keep-alive.......q=0.
↪5\r\n\r\n'
b'GET /blue? HTTP/1.1\r\nHost: 192.168.18.162\r\nConnection: keep-alive.......q=0.5\r\n\
↪r\n'
b'GET /favicon.ico HTTP/1.1\r\nHost: 192.168.18.162\r\nConnection: keep-alive.......q=0.
↪5\r\n\r\n'
```

読むには長すぎます！

しかし、実際に必要なのは `/red?` や `/blue?` の前にある小さな情報だけです。これはどのボタンが押されたのか

を教えてくれます。それで、キーストロークの情報を抽出するために serve() を少し改良しました。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        print(request)
        html = webpage(0)
        client.send(html)
        client.close()
```

プログラムを再実行すると、ウェブページでキーを押すと、シェルは以下のようなメッセージを出力します。

```
/red?
/favicon.ico
/blue?
/favicon.ico
/off?
/favicon.ico
```

次に、 request の値に応じて RGB LED の色を変更するだけです。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass

        print(request)

        if request == '/off?':
            red.low()
```

(次のページに続く)

(前のページからの続き)

```
            green.low()
            blue.low()
        elif request == '/red?':
            red.high()
            green.low()
            blue.low()
        elif request == '/green?':
            red.low()
            green.high()
            blue.low()
        elif request == '/blue?':
            red.low()
            green.low()
            blue.high()

        html = webpage(0)
        client.send(html)
        client.close()
```

最後に、ウェブページにサーミスターの値を表示する必要があります（サーミスターの使用方法の詳細については、 *2.13* **温度計** を参照してください)。この部分は実際には HTML のテキストを修正することで実現されます。webpage(value) 関数でパラメータを設定し、入力パラメータを変更するだけでウェブページに表示される数字を変更します。

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass

        #print(request)

        if request == '/off?':
            red.low()
            green.low()
```

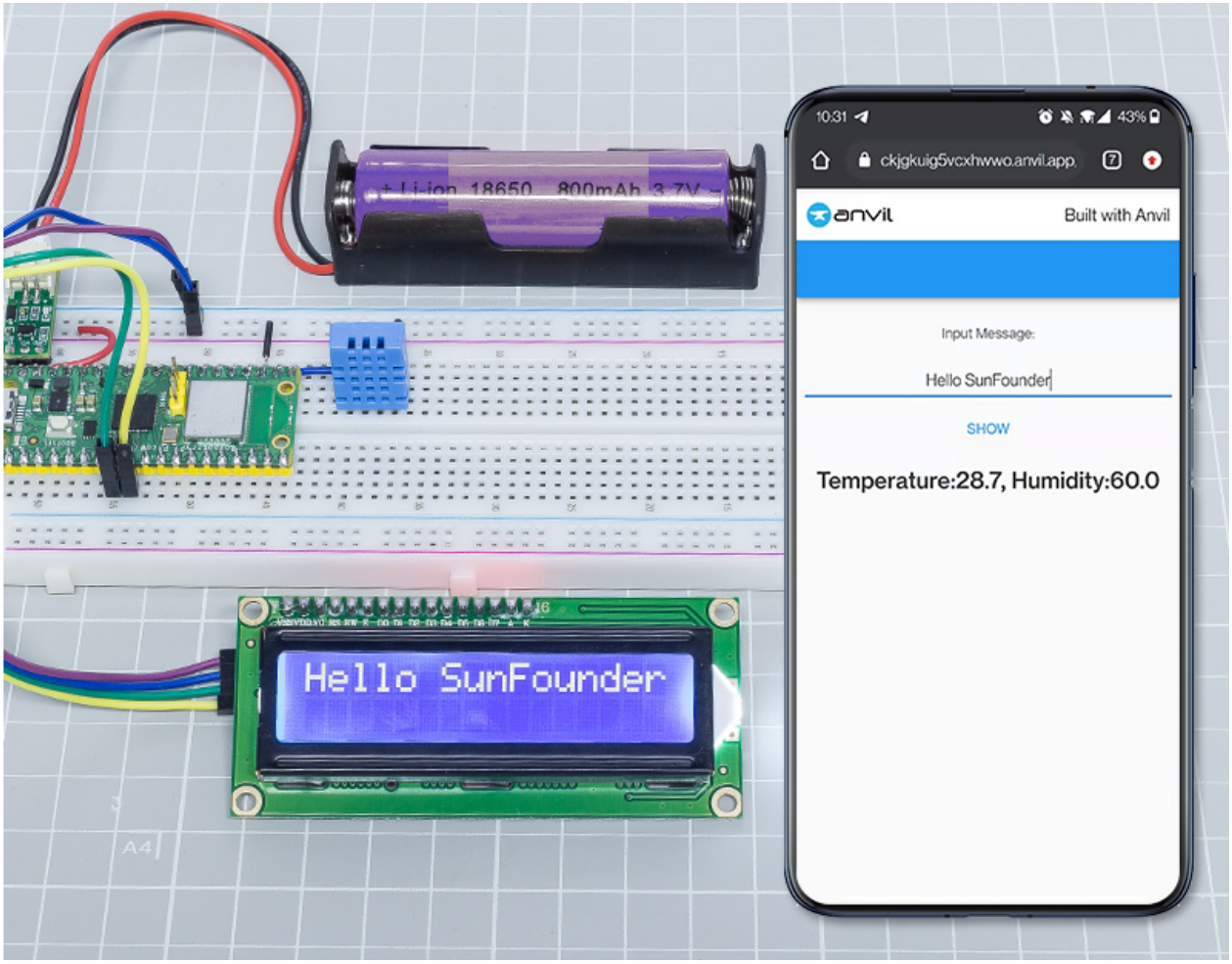
(次のページに続く)

(前のページからの続き)

```
            blue.low()
        elif request == '/red?':
            red.high()
            green.low()
            blue.low()
        elif request == '/green?':
            red.low()
            green.high()
            blue.low()
        elif request == '/blue?':
            red.low()
            green.low()
            blue.high()

        value = '%.2f' % temperature()
        html = webpage(value)
        client.send(html)
        client.close()
```

5.8 8. @Anvil を用いた Web アプリの構築

このプロジェクトでは、Raspberry Pi Pico W と Anvil のサーバーとの間で双方向通信を行います。Pico W から送信される温度と湿度は Anvil でリアルタイムに表示されます。さらに、Anvil でメッセージを入力すると、それが Pico W の I2C LCD1602 に表示されます。



1. 必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

全体のキットを購入する方が確実に便利です。リンクはこちら：

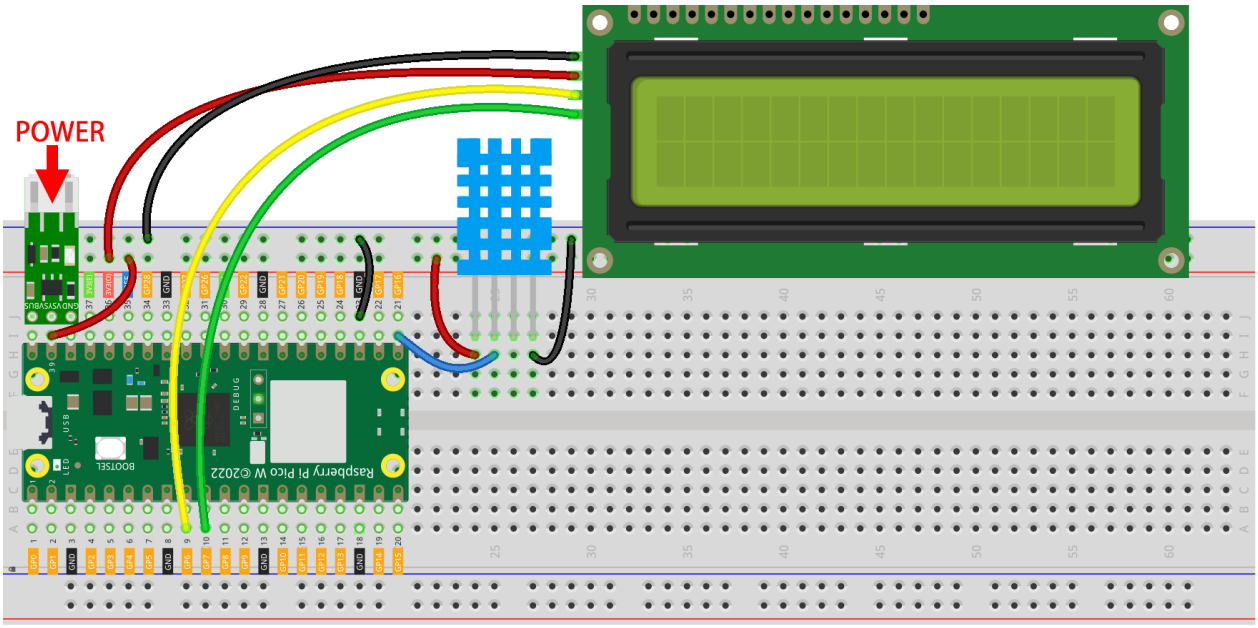
名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個々に購入することも可能です。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	複数	
5	*I2C LCD1602*	1	
6	*DHT11* **湿温度センサー**	1	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	

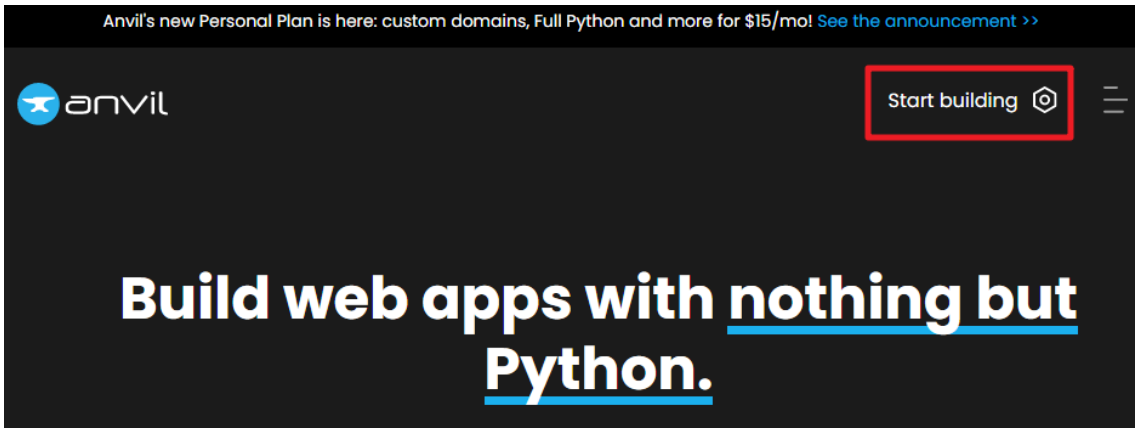
2. 回路を組む

> **警告:** 図に示されているように Li-po チャージャーモジュールが接続されていることを確認してください。そうでない場合、短絡が発生してバッテリーと回路が損傷する可能性があります。

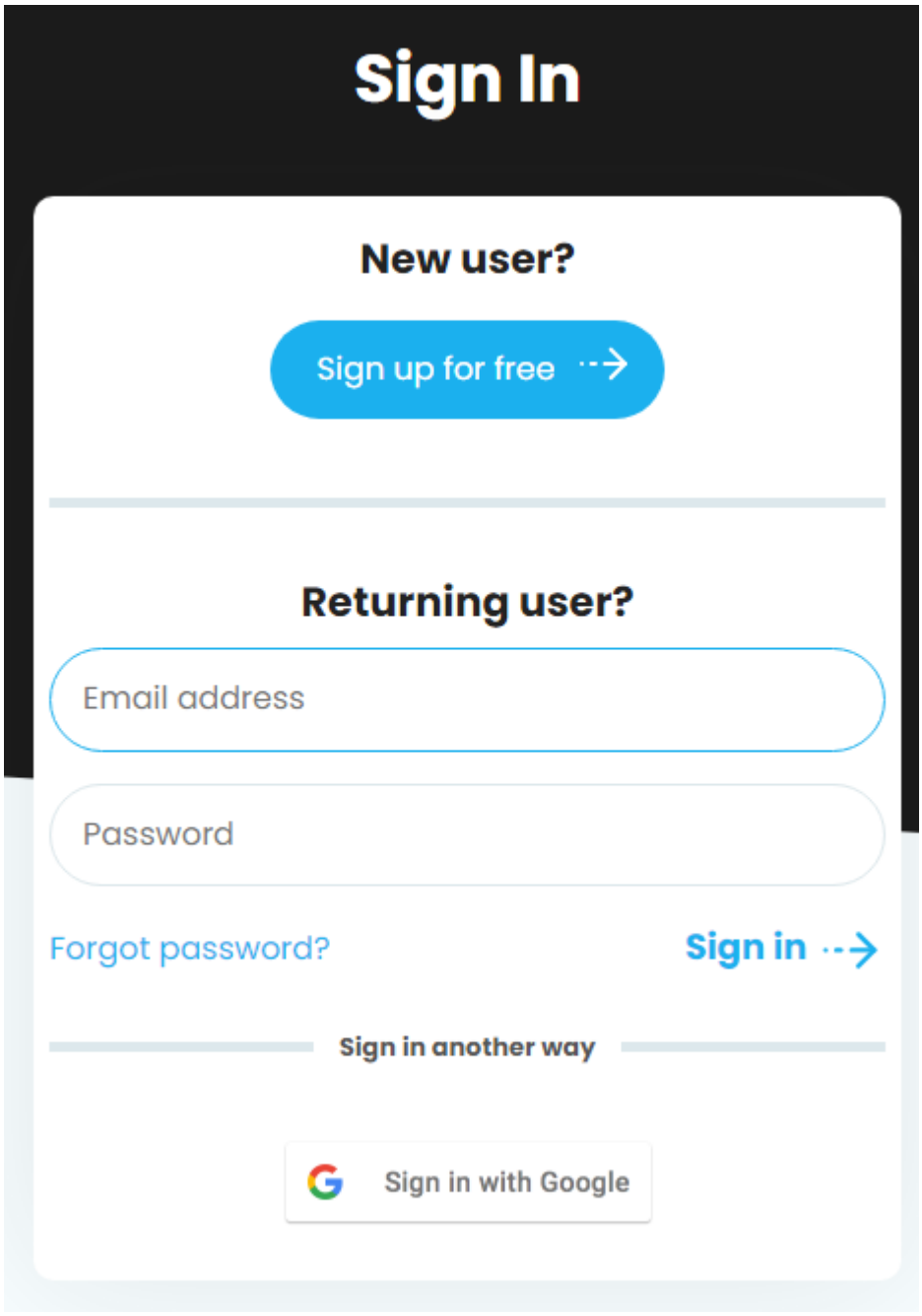


3. Anvil アプリを作成する

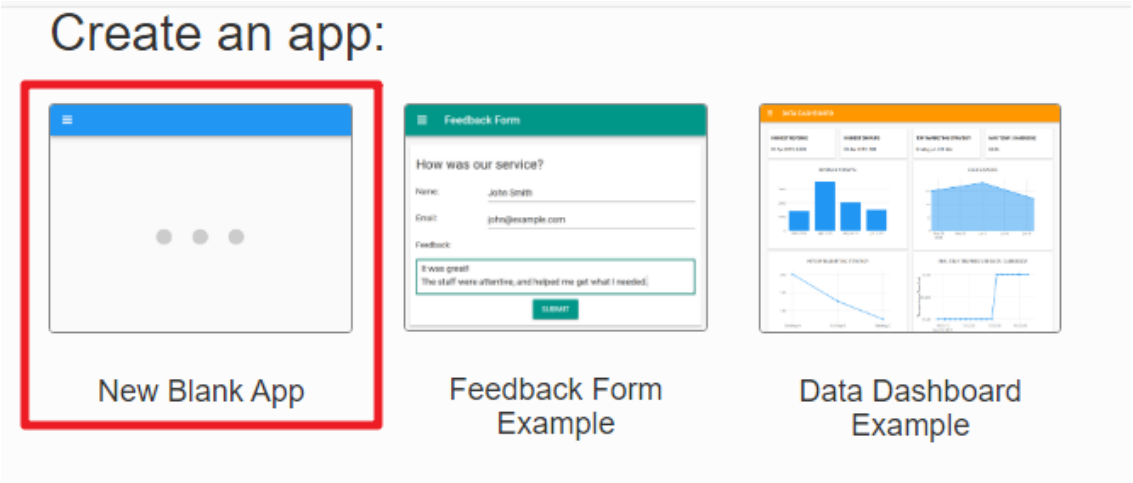
1. にアクセスして、 **Start building** をクリックします。



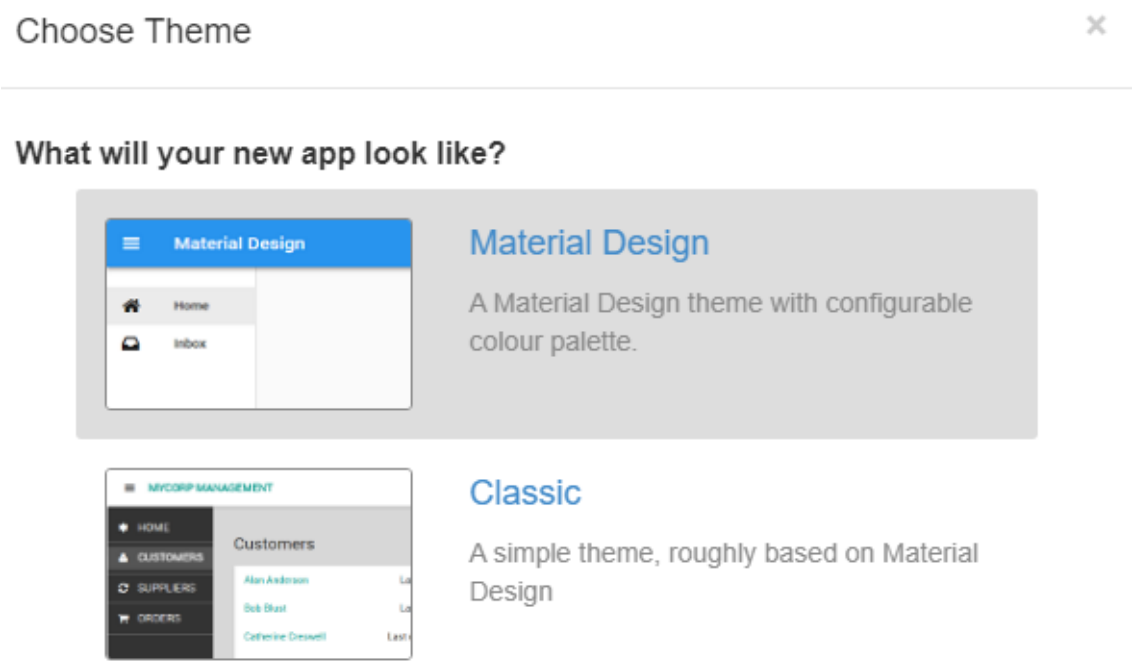
2. サインインまたはサインアップします。



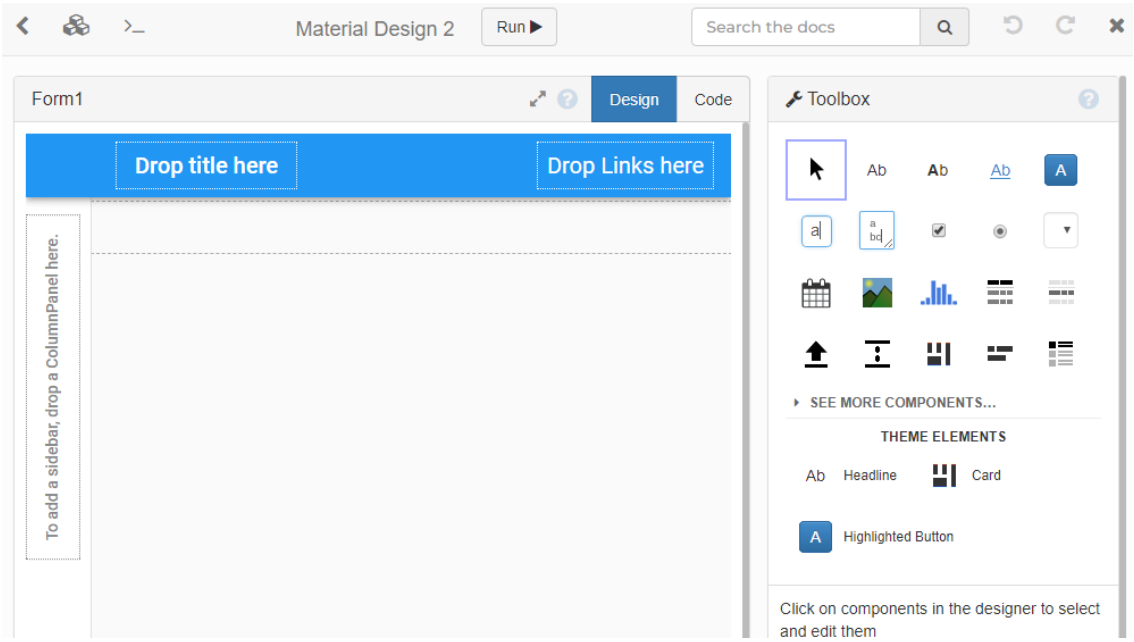
3. **新しいブランクアプリ** を作成します。



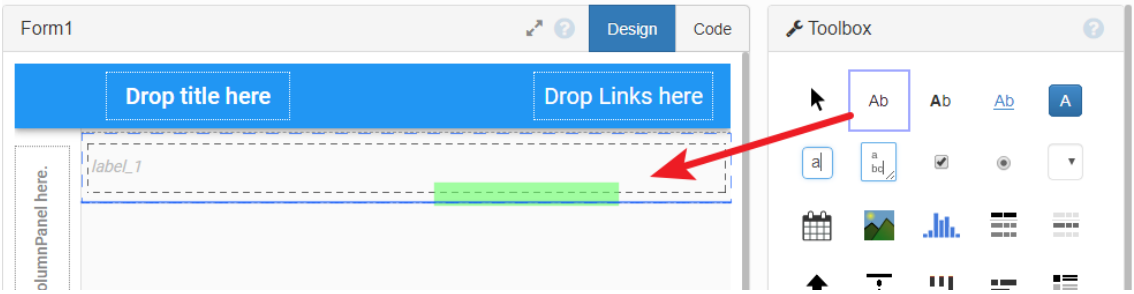
4. **マテリアルデザインテーマ** を選択します。



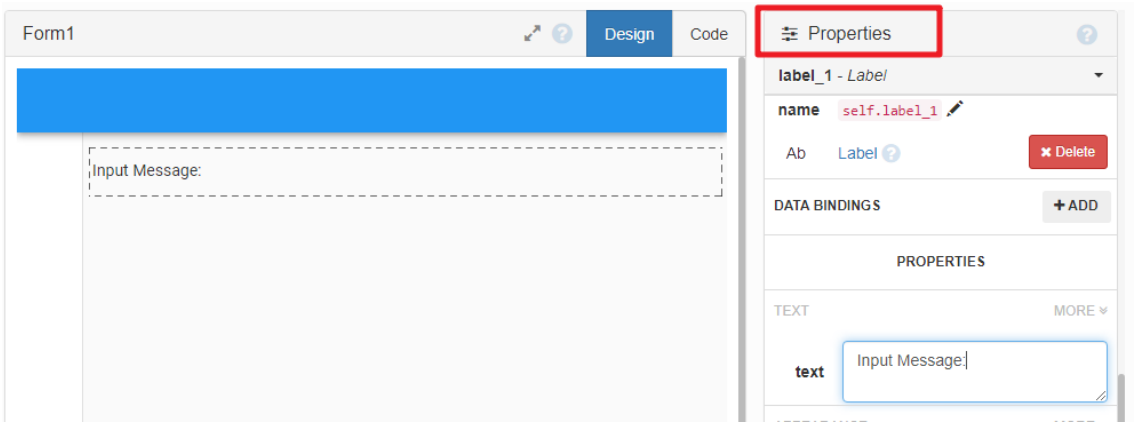
5. これでアプリの編集ページに移動します。



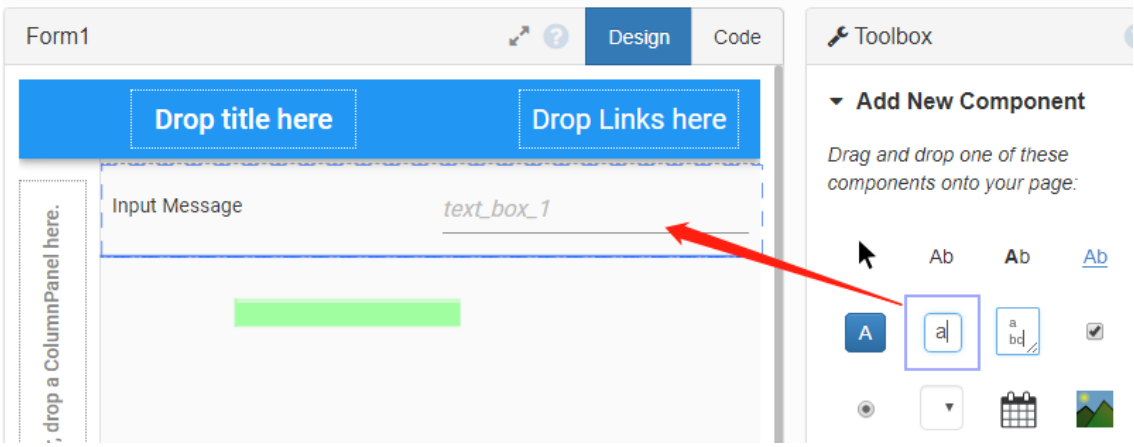
6. ツールボックスから **Label** ツールをドラッグして **Drop title here** に配置します。



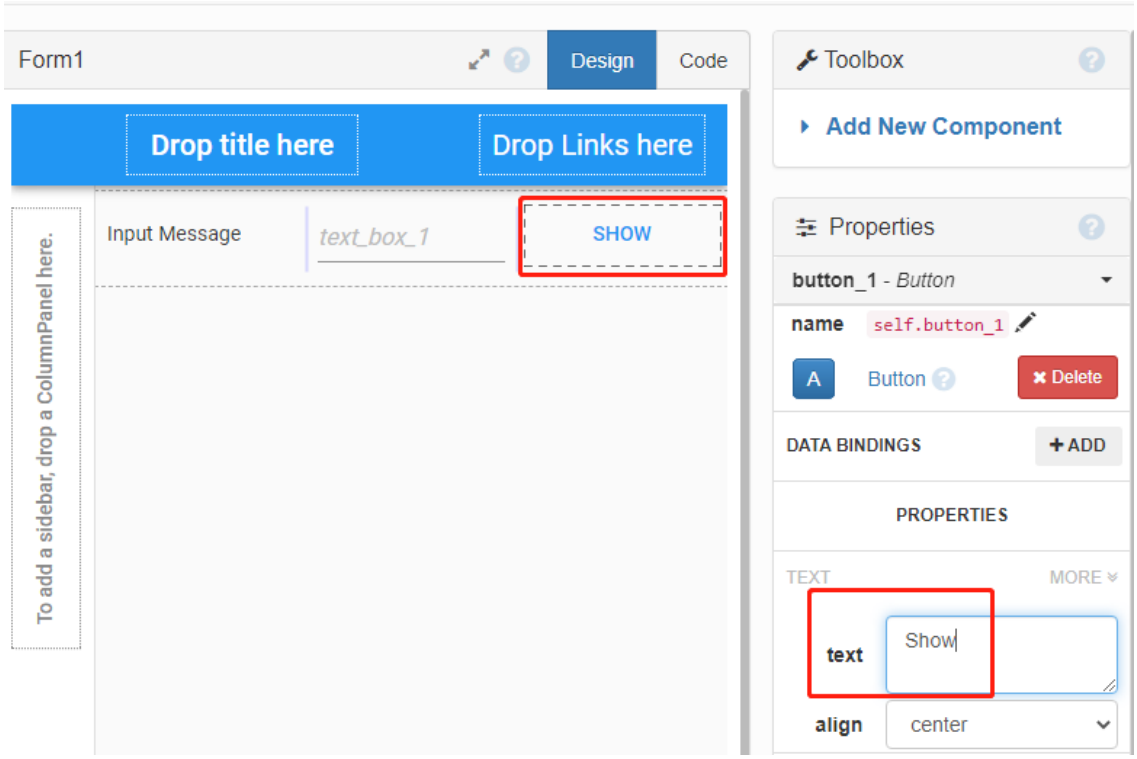
7. **プロパティ** メニューの下の **テキスト** フィールドでラベルのテキストを入力できます。



8. 同様に、右側に **TextBox** をドラッグします。



9. **ボタン** を右端にドラッグし、 **テキスト** フィールドを変更できます。このボタンは Raspberry Pi Pico W にメッセージを「送信」するために使用されます。

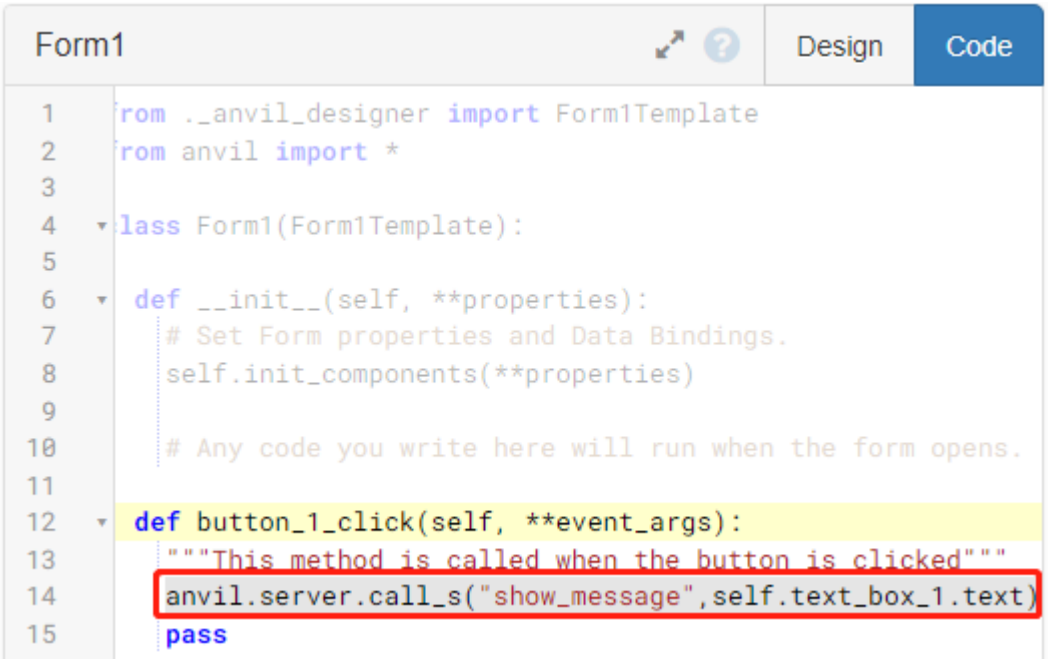


10. **SHOW** ボタンをダブルクリックすると、フォームはデザインページからコードページに切り替わり、その **ボタン** のコードが強調表示されます。次のコードを入力する必要があります。このコードは、サーバー（この場合、Pico W）内の関数を呼び出す機能があります。

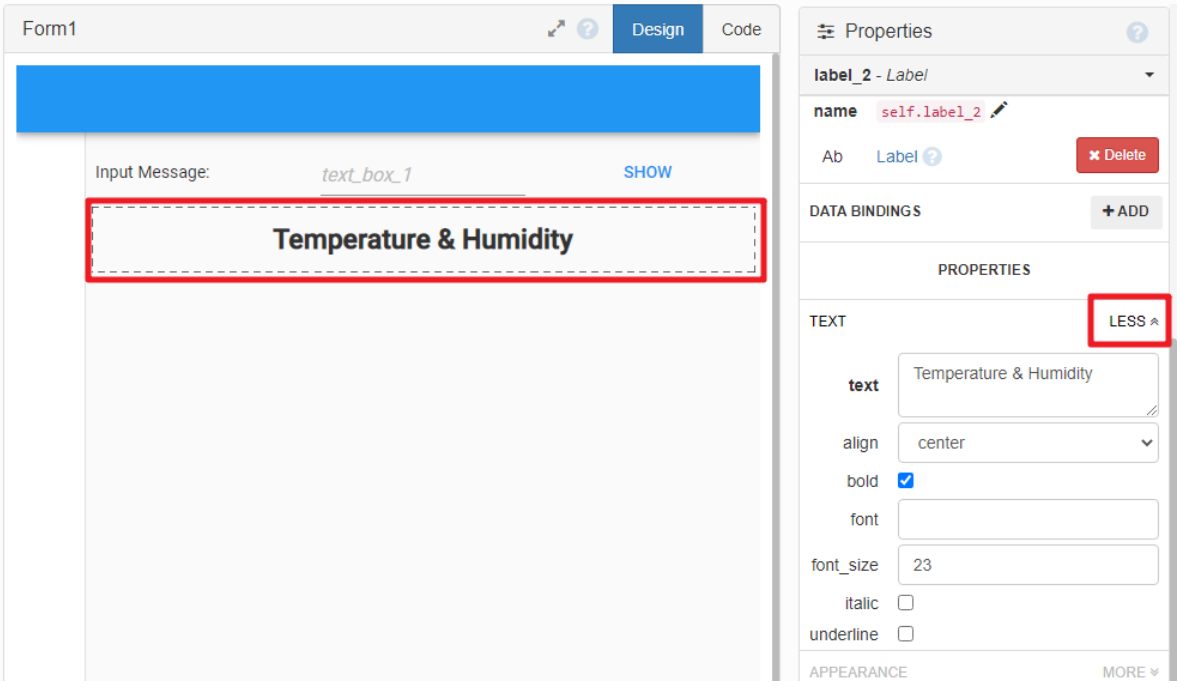
```
anvil.server.call_s("show_message",self.text_box_1.text)
```

- `show_message` は Pico W がプログラムされたときに定義される関数です。
- `self.text_box_1.text` はテキストボックスに入力するメッセージであり、 `show_message()` にパス

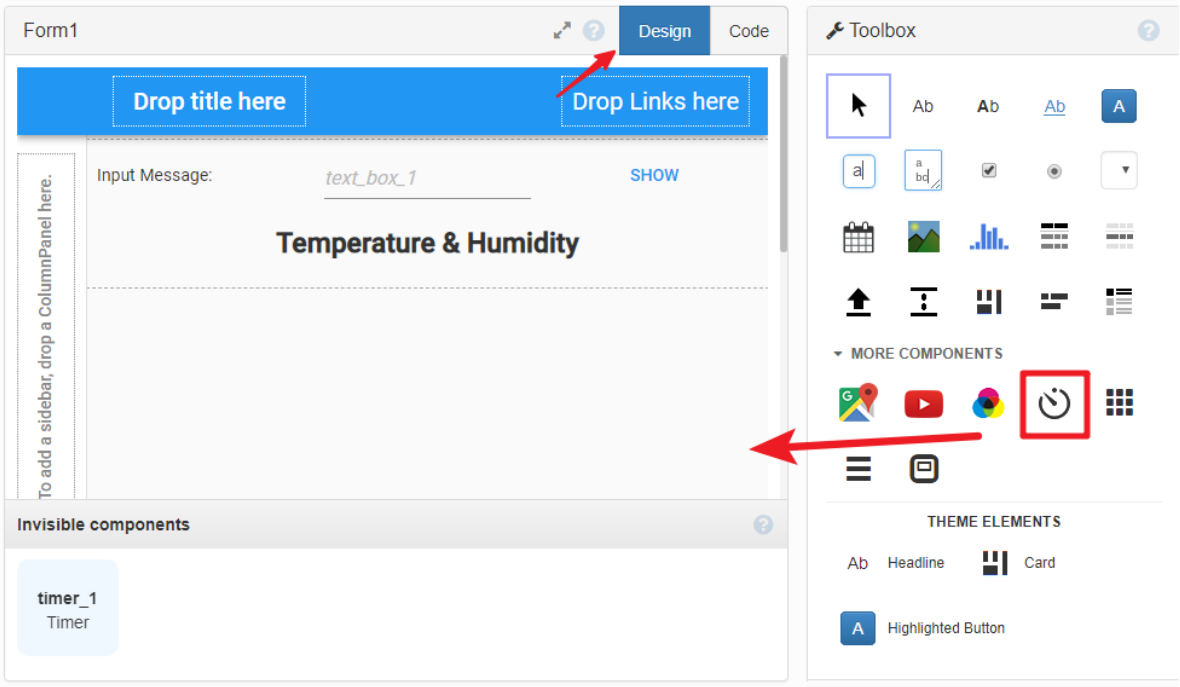
スルーとして送信されます。



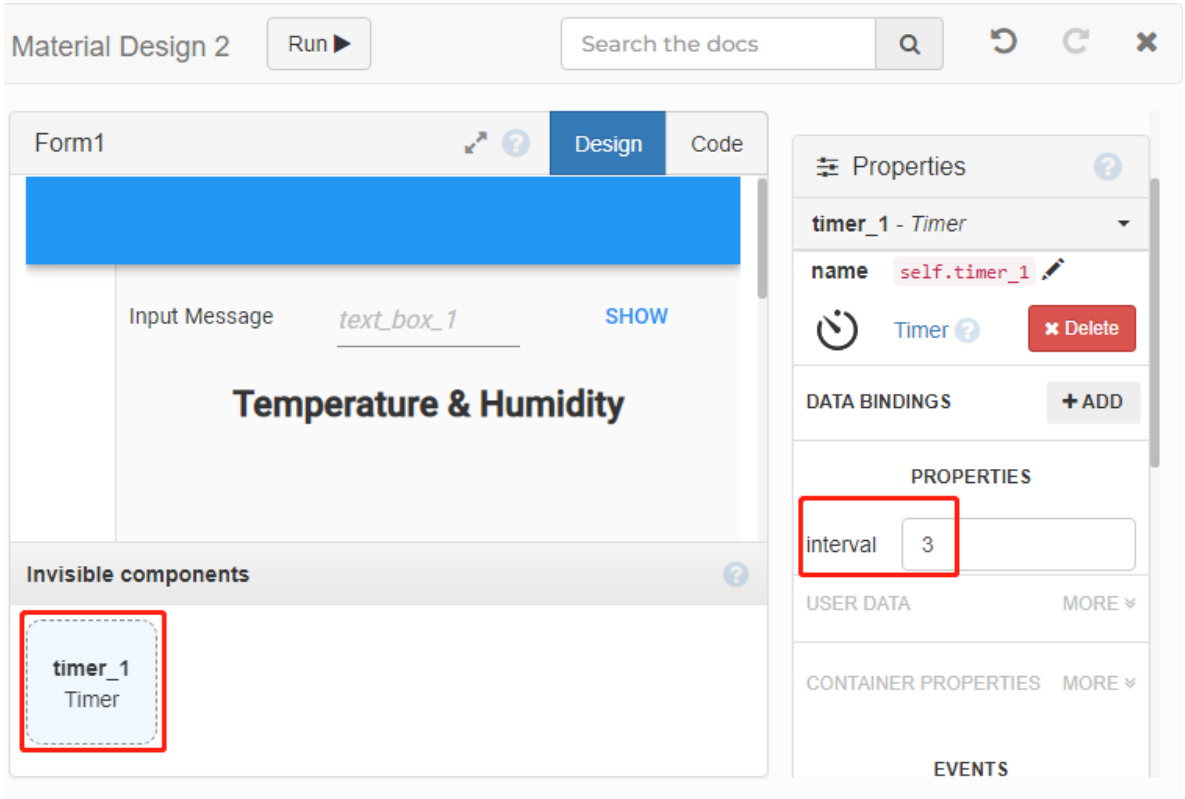
11. デザインページに戻って、別のラベルをドラッグして以前の要素の下に配置します。このラベルは Pico W からの DHT11 センサーデータを表示します。



12. **ツールボックス** で **More Components** をクリックし、 **Timer** をフォームにドラッグします。



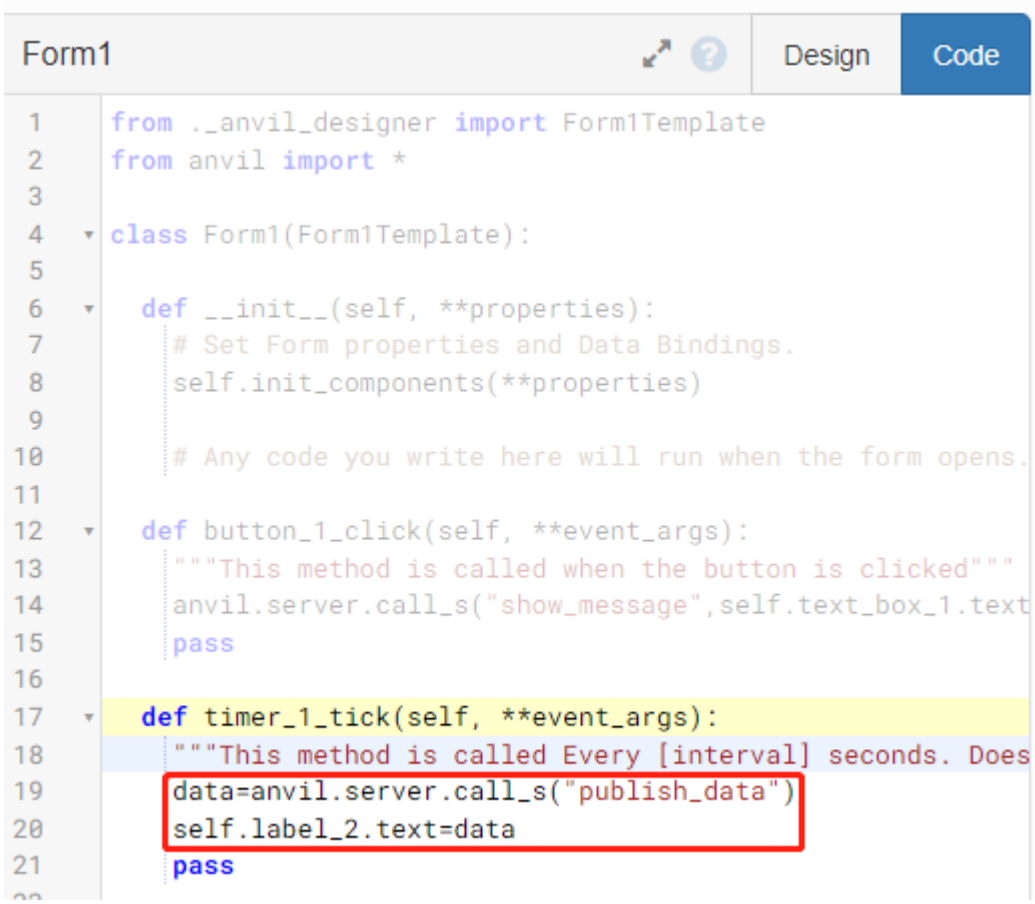
13. **プロパティ** を使用して、タイマーを 3 秒の間隔に設定します。この時間は、センサーデータの画面を更新するために使用されます。



14. **Timer** ツールをダブルクリックしてプログラムします。 `anvil.server.call_s()` 関数を使用して、Anvil アプリに表示する必要のあるメッセージをサーバーから取得するための `publish_data()` 関数を呼び出し、

それを self.label_2.text に割り当てて完了します。

```
data=anvil.server.call_s("publish_data")
self.label_2.text=data
```

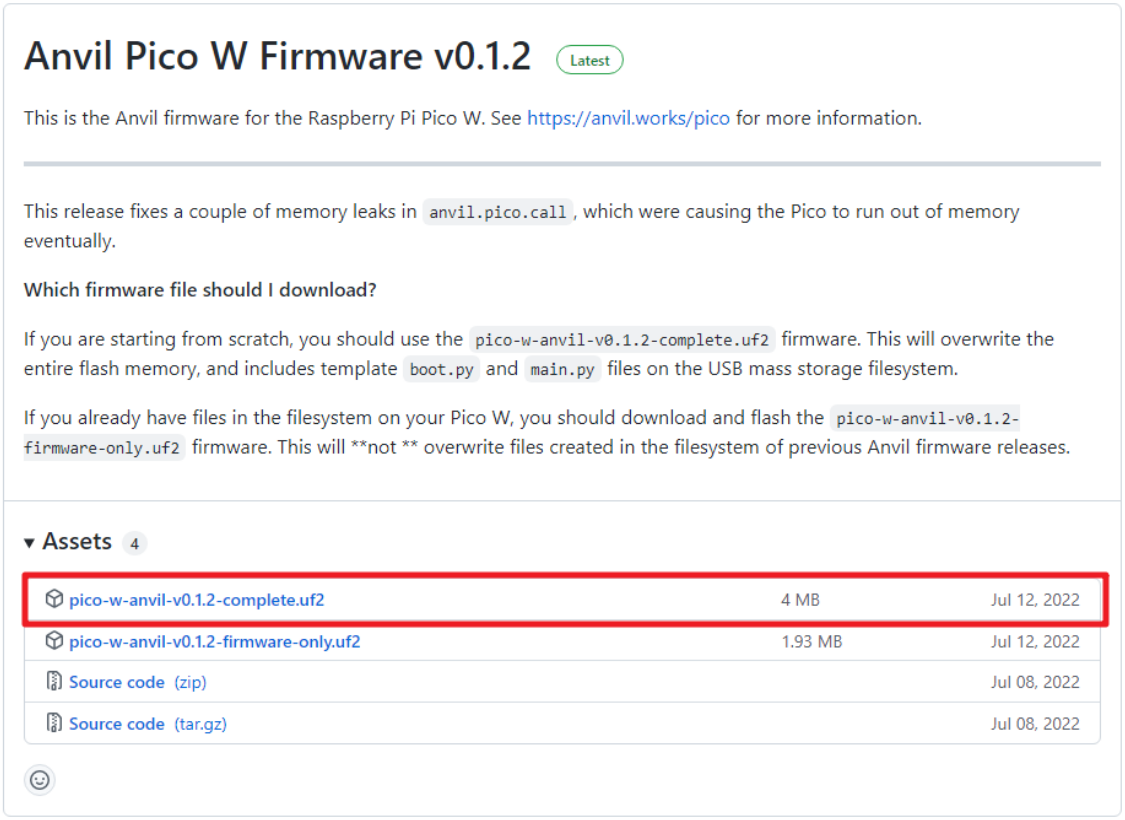


15. この時点で、Anvil でプログラムする必要のある部分は完了です。Anvil の使用に関する詳細は、 で確認できます。

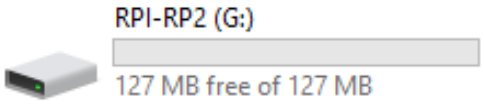
4. Pico W のセットアップ

Anvil サービスへの Raspberry Pi Pico W の接続を簡単にするため、Anvil はカスタムファームウェアイメージを使用します。Pico W のファームウェアは MicroPython で書かれており、USB ドライブとして（boot.py と main.py の 2 つのファイルを持つ形で）認識されます。コードを書き始める前に、Pico W にカスタムファームウェアをフラッシュし、Wi-Fi に接続する必要があります。

1. Raspberry Pi Pico W 用のカスタムファームウェア をダウンロードします。完全版のダウンロードが推奨されます。

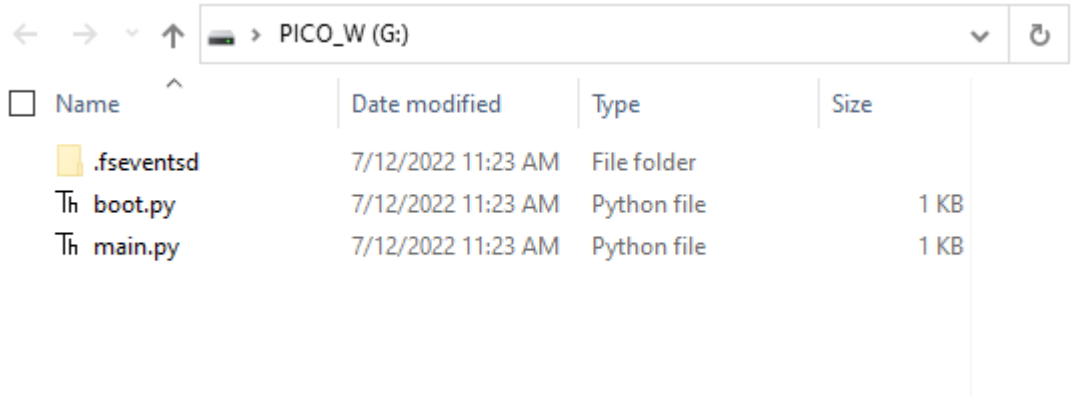


2. Pico W の **BOOTSEL** ボタンを押しながら、マイクロ USB ケーブルでコンピュータに接続します。ドライブ RPI-RP2 がコンピュータに表示されたら、BOOTSEL を離します。

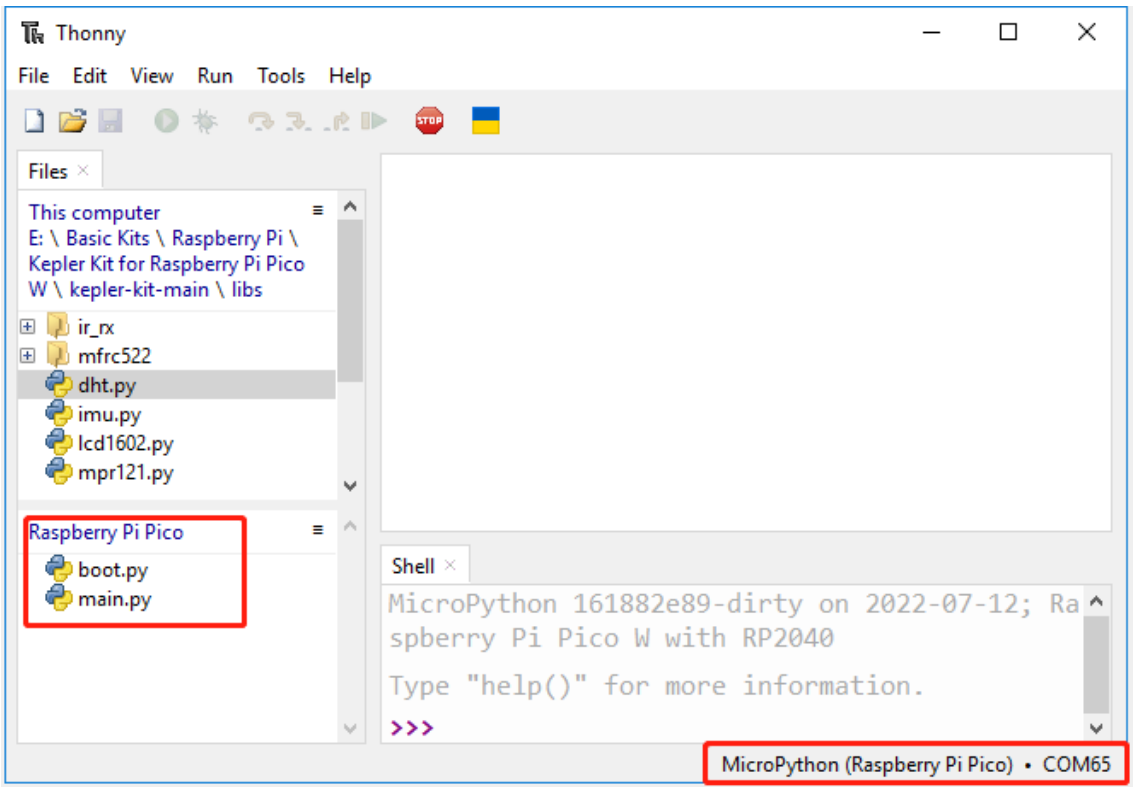


3. ダウンロードしたばかりの `.uf2` ファイルをドラッグ&ドロップします。この時点で Pico W はファームウェアをインストールします。完了すると、 `main.py` と `boot.py` ファイルが表示されます。

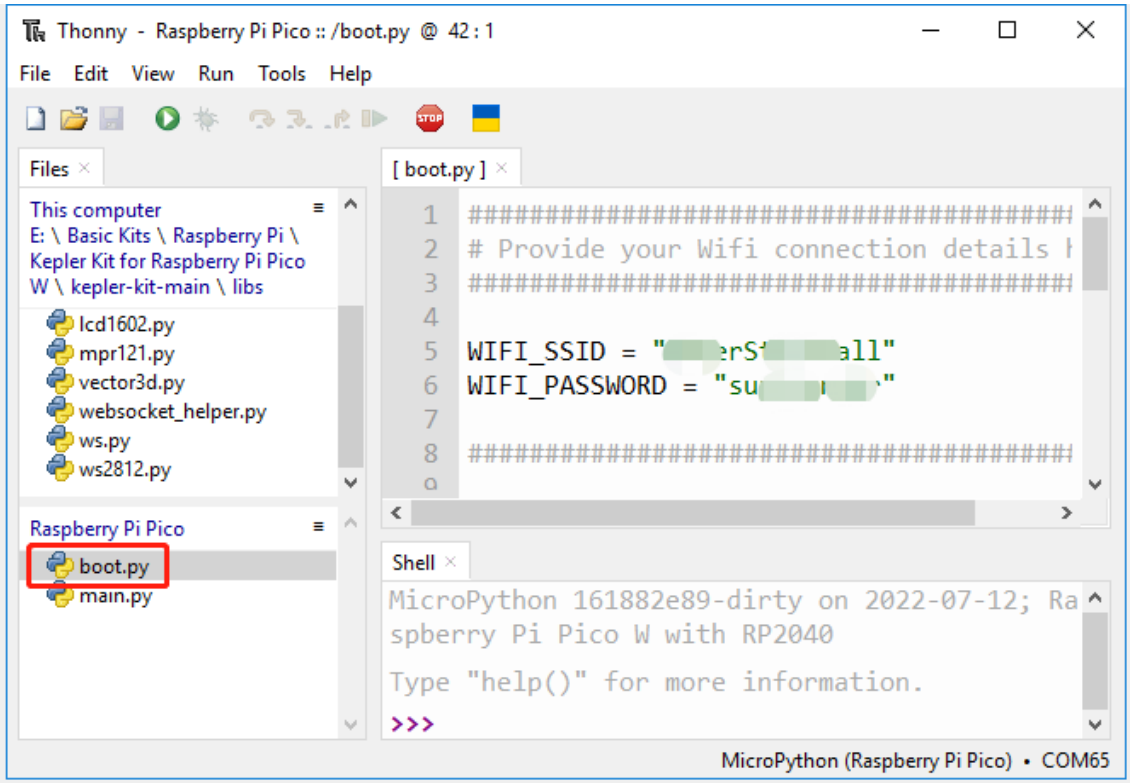
注釈: ファームウェアを再インストールする前に、Pico W に保存された重要なファイルのバックアップを取ってください。



4. Thonny IDE でインタプリタとして"MicroPython(Raspberry Pi Pico).COMXX"を選択します。 **View -> Files** をクリックすると、ローカルドライブと Raspberry Pi Pico のハードドライブが表示されます。

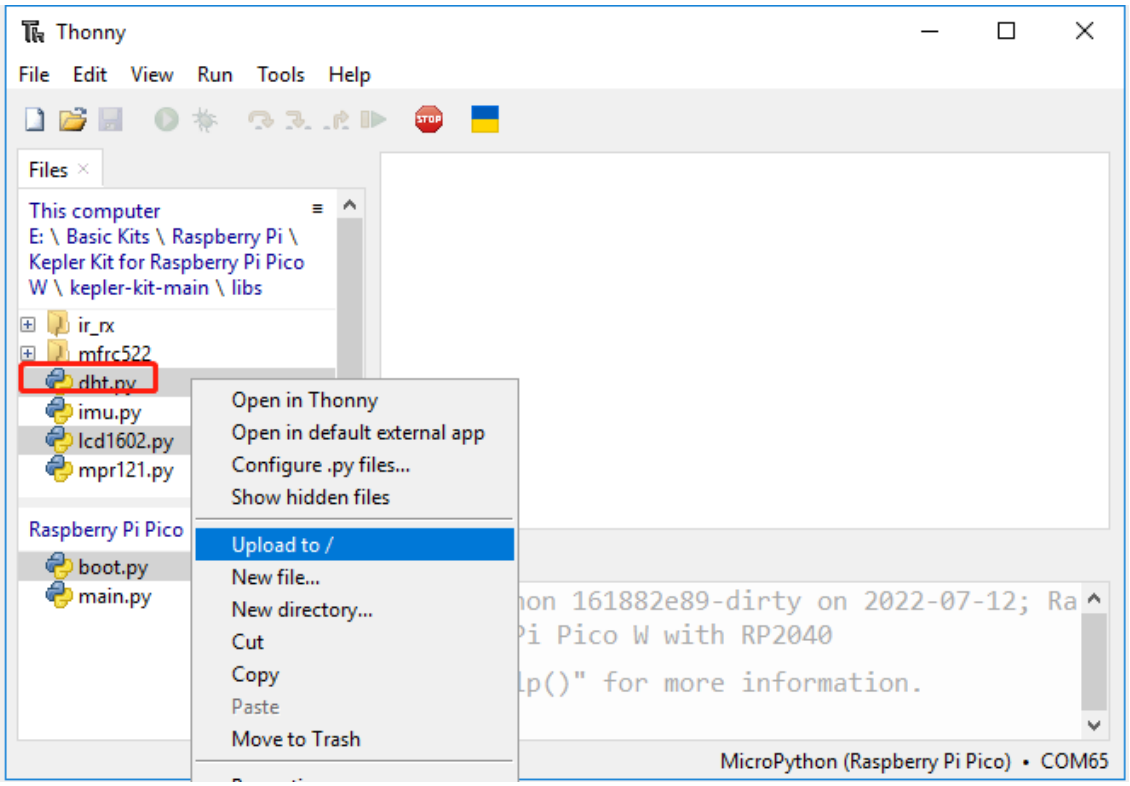


5. `boot.py` スクリプトをダブルクリックし、WiFi の `SSID` と `PASSWORD` を入力します。



5. コードの完成

1. `kepler-kit-main/libs` のパスから `dht.py` と `lcd1602.py` を Raspberry Pi Pico W にアップロードします。



2. main.py を開き、以下のコードで元のコードを置き換えます。

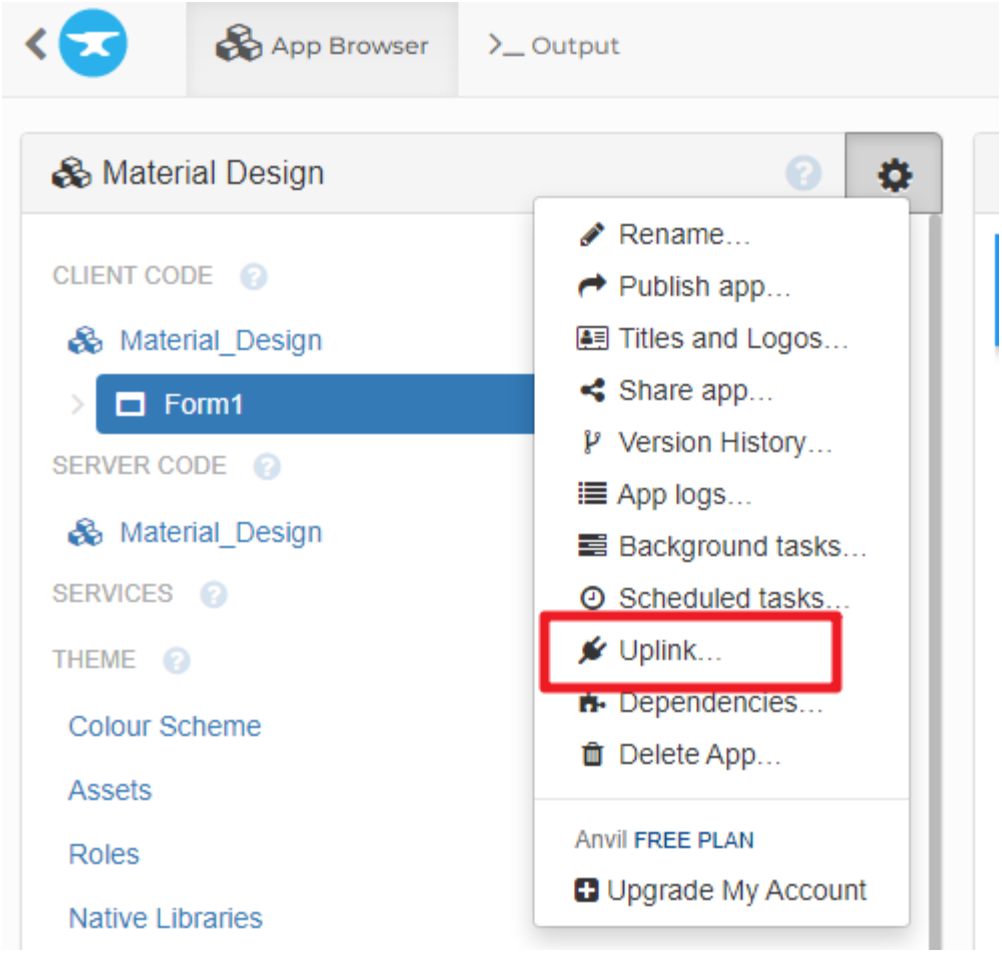
```
import anvil.pico
import uasyncio as a
from machine import Pin,I2C

from lcd1602 import LCD
lcd = LCD()

from dht import DHT11
sensor = DHT11(Pin(16, Pin.OUT, Pin.PULL_DOWN))

UPLINK_KEY = "<uplink_key_goes_here>"

@anvil.pico.callable(is_async=True)
async def show_message(text):
    print(f"show anvil's input message: {text}")
    lcd.clear()
    lcd.message(text)
    return
```

(次のページに続く)

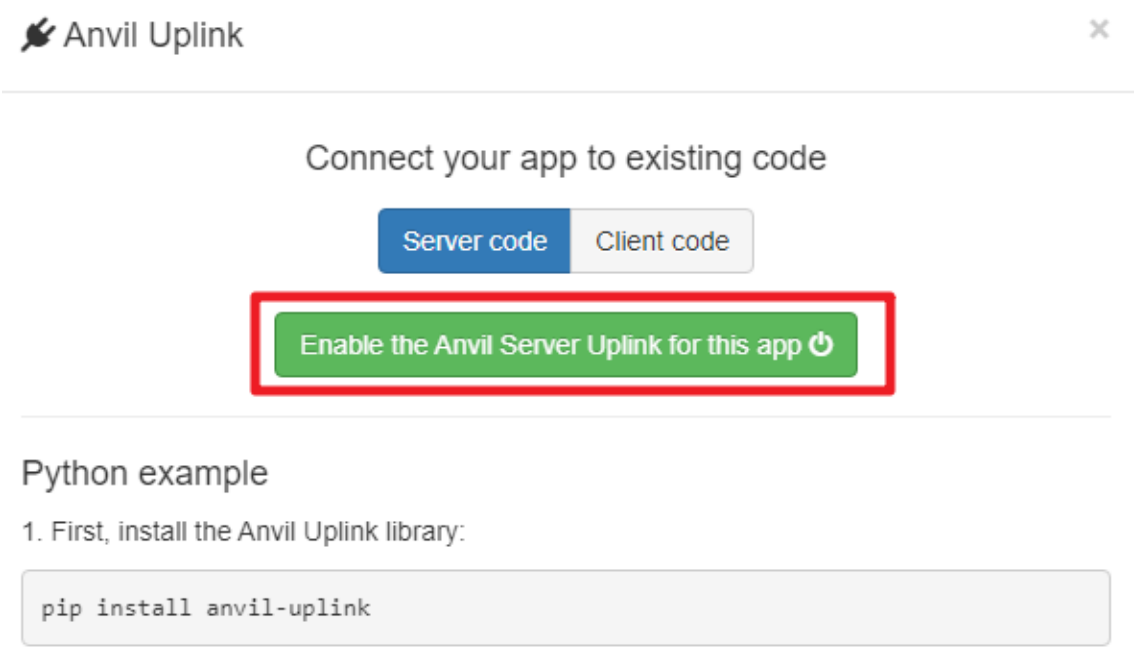
(前のページからの続き)

```
@anvil.pico.callable(is_async=True)
async def publish_data():
    sensor.measure()
    return "Temperature: {}, Humidity: {}".format(sensor.temperature, sensor.
↪humidity)

anvil.pico.connect(UPLINK_KEY)
```

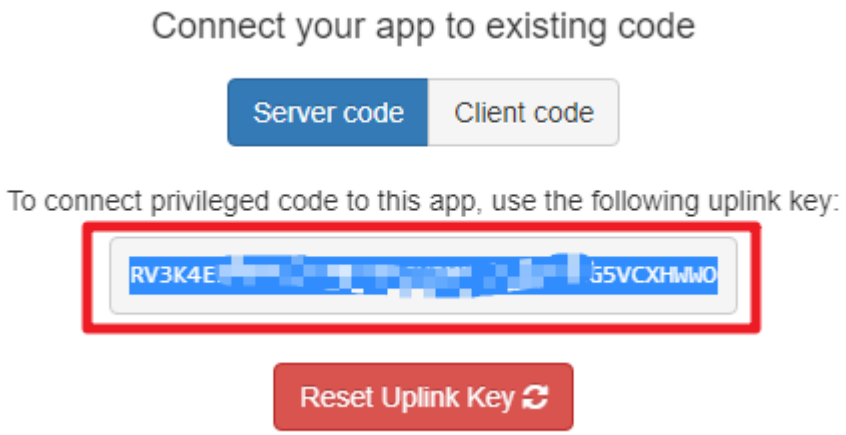
3. Anvil インターフェースに戻り、App Browser の設定で Uplink オプションをタップします。



4. **Enable the Anvil Server Uplink for this app** をクリックして、uplink キーを取得します。

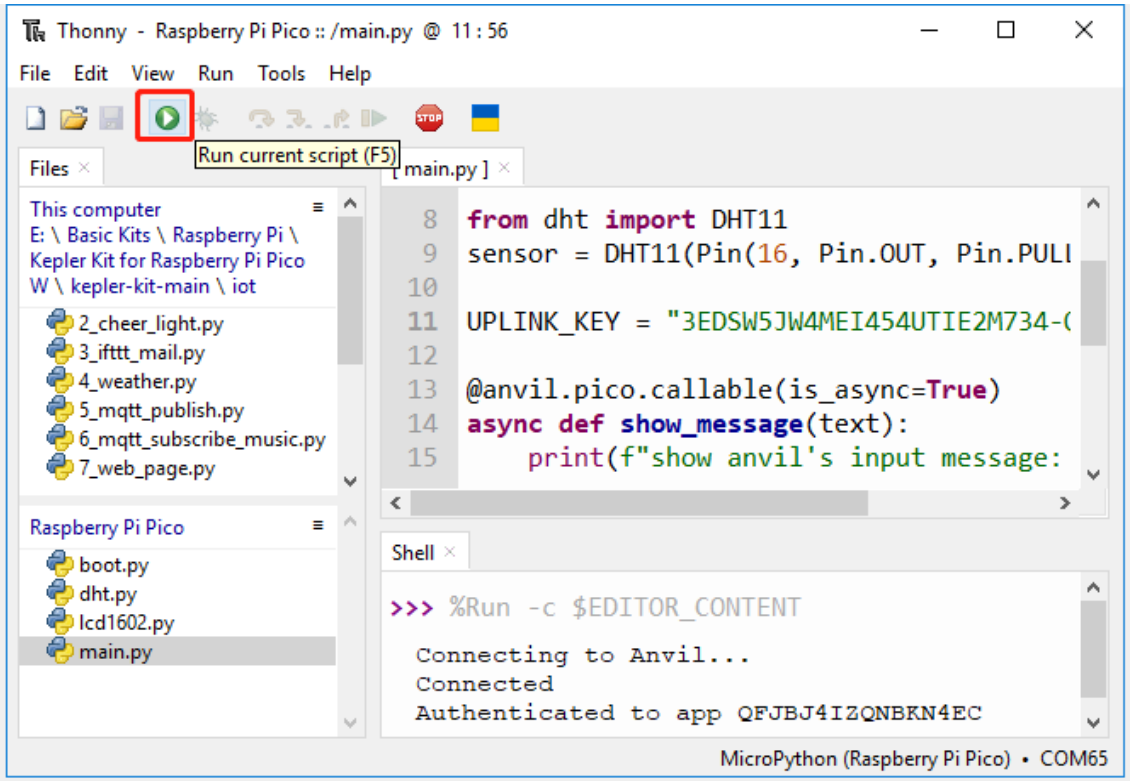


5. それをコピーし、 `main.py` の `<uplink_key_goes_here>` を置き換えます。これにより、作成した Anvil APP に Pico W が接続できるようになります。



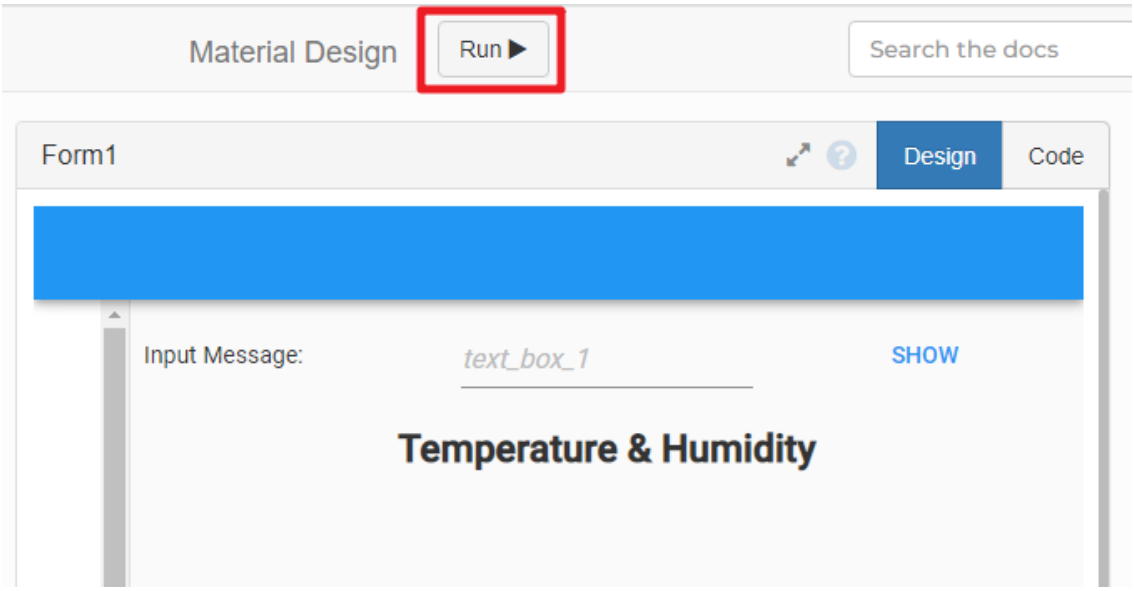
6. プロジェクトの実行

1. **Run current script** ボタンをクリックするか、F5 を押して実行します。接続が成功すると、Shell に接続成功のプロンプトが表示されます。



2. Anvil を実行します。これで、Anvil APP から温度と湿度が表示されるようになります。テキストボックスにメッセージを入力した後に **SHOW** ボタンをクリックすると、I2C LCD1602 に入力したメッセージが表示されます。

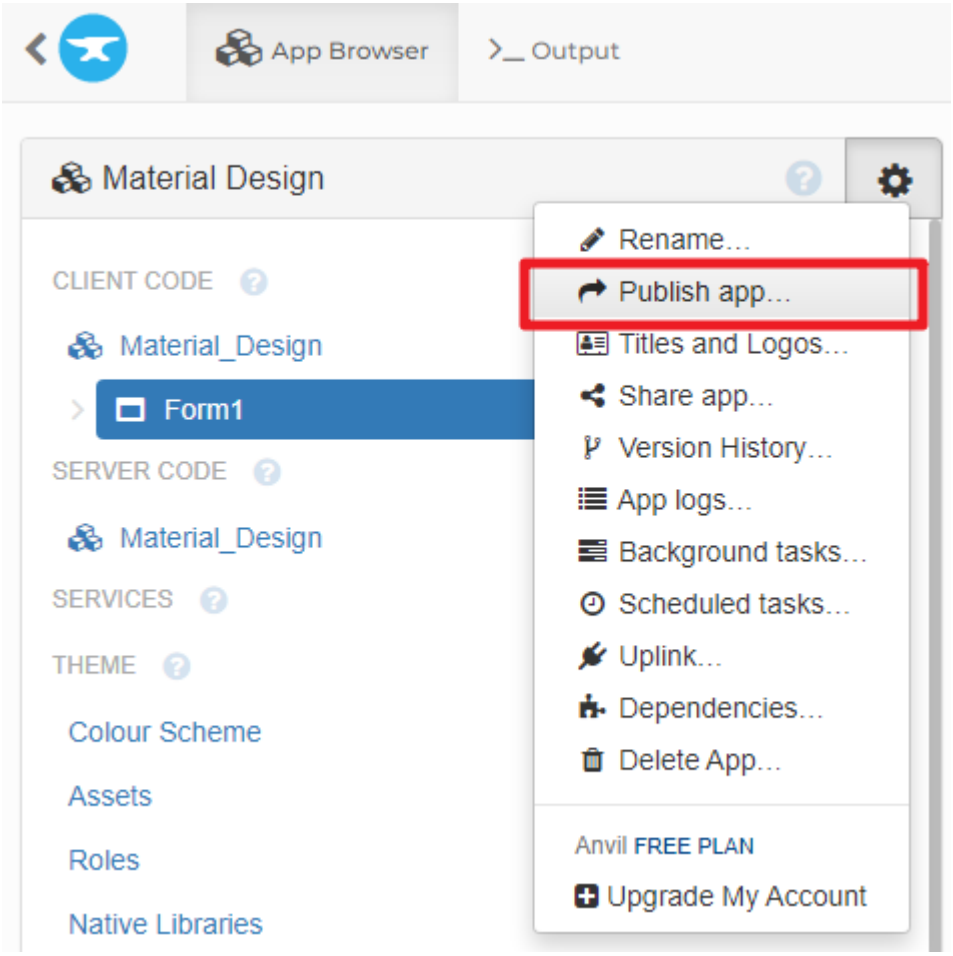
注釈: 入力した文字が I2C LCD1602 に表示されない場合は、モジュールの裏側にあるポテンショメータを回してコントラストを調整できます。



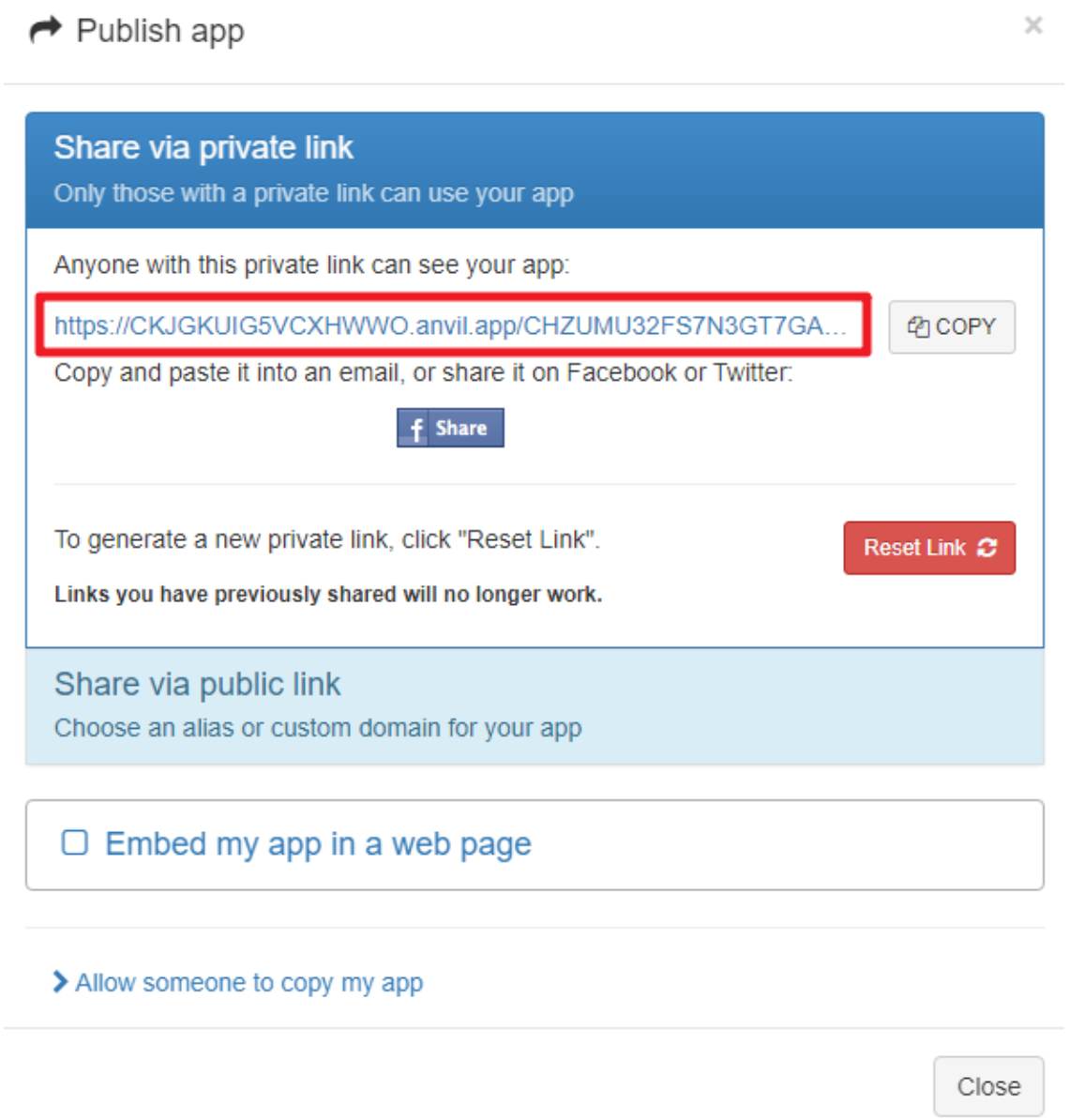
7. APP の公開

作成したアプリを他人と共有したい場合は、以下の方法で共有リンクを生成できます。

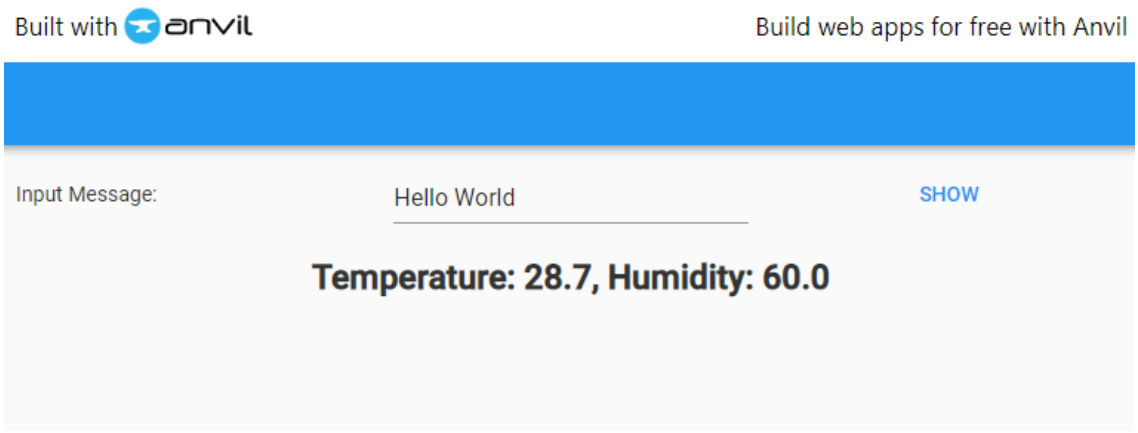
1. **Anvil** ページに戻って、 **App Browser settings** 内の **publish app** オプションをクリックします。



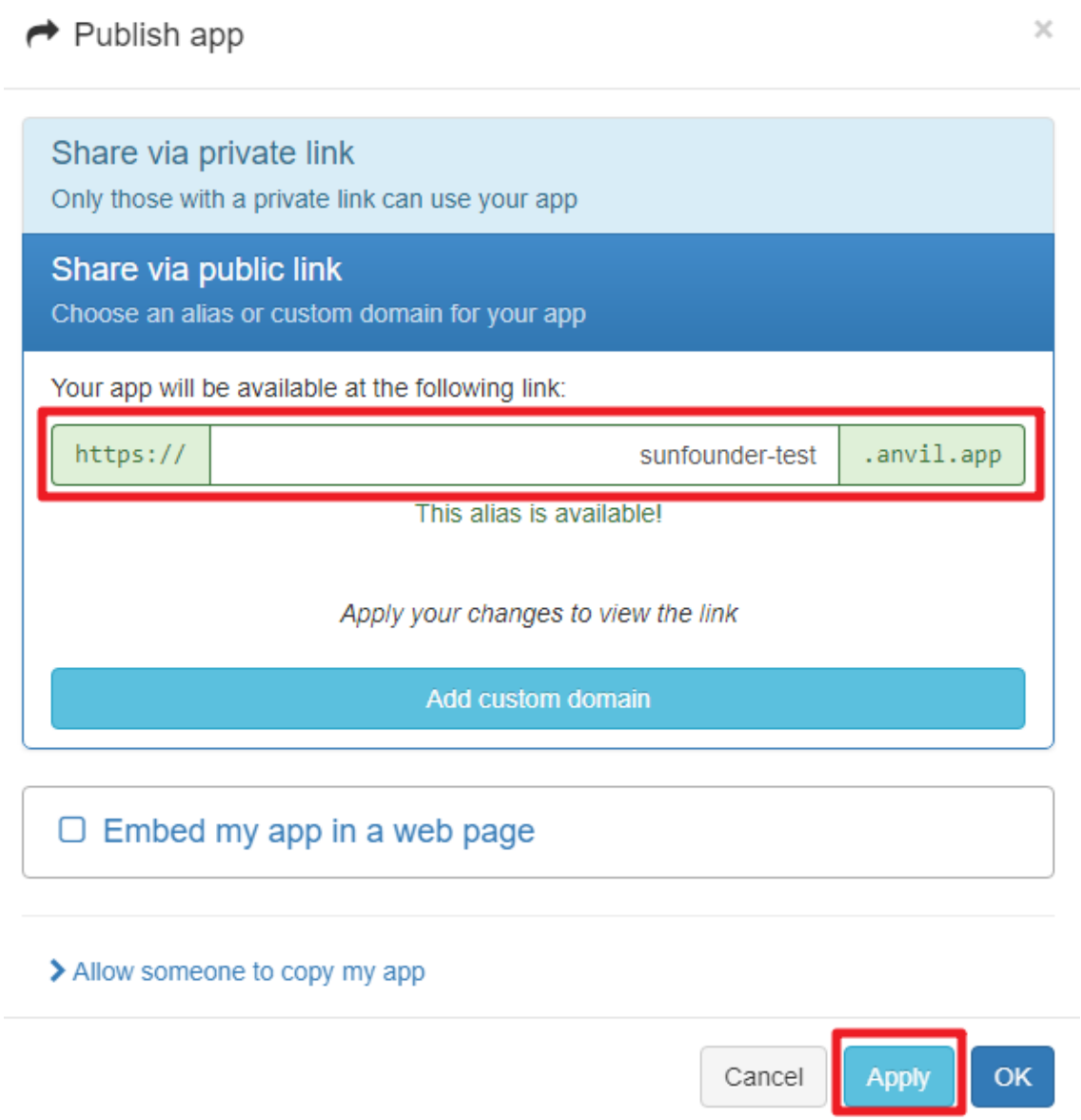
2. **Share via private link** タブにはリンクのリストが表示されます。このリンクを通じて誰でもアプリにアクセスできます。



3. リンクにアクセスすると、アプリは直接使用可能になります。



4. 公開リンクを通じてもアプリを共有できます。独自のドメイン名を入力し、下の **Apply** をクリックして有効にします。



仕組みは？

以下は、Pico W と Anvil APP の通信の基本となる `main.py` の基本フレームワークです。

```
import anvil.pico
import uasyncio as a

UPLINK_KEY = "<uplink_key_goes_here>"

anvil.pico.connect(UPLINK_KEY)
```

dht11 と lcd1602 のセットアップ。これら 2 つのコンポーネントの使用方法の詳細は、*6.2* **温度・湿度センサー** と *3.4* **液晶ディスプレイ** で確認できます。

```
from machine import Pin,I2C

from lcd1602 import LCD
lcd = LCD()

from dht import DHT11
sensor = DHT11(Pin(16, Pin.OUT, Pin.PULL_DOWN))
```

Anvil のコードでは、サーバー（Pico W）の 2 つの内部関数を呼び出しています。

最初は show_message() で、この関数は Anvil で入力されたメッセージを LCD に表示させる役割があります。デコレータ @anvil.pico.callable(is_async=True) は、この関数を Anvil から呼び出し可能にします。

```
@anvil.pico.callable(is_async=True)
async def show_message(text):
    print(f"show anvil's input message: {text}")
    lcd.clear()
    lcd.message(text)
    return
```

次は publish_data() で、これは dht11 の値を取得し、温度と湿度を Anvil に返す機能があります。これもデコレータ @anvil.pico.callable(is_async=True) を使用して、Anvil から呼び出し可能にします。

```
@anvil.pico.callable(is_async=True)
async def publish_data():
    sensor.measure()
    return "Temperature: {}, Humidity: {}".format(sensor.temperature, sensor.humidity)
```

5.9 9. @SunFounder コントローラーで遊ぶ

このプロジェクトでは、SunFounder Controller APP を使用してリモートプロジェクトを構築する方法を学びます。ローカルエリアネットワーク（LAN）環境で、スマートフォンやタブレットを使って Pico W 回路を制御できます。Pico W でシンプルなロボットを作成したい場合、このアプリは非常に便利です。

ここでは、APP のスライダーバーでサーボの角度を制御し、APP のゲージで超音波によって検出された距離を表示します。

1. 必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

一式を購入することは確かに便利です、リンクはこちらです：

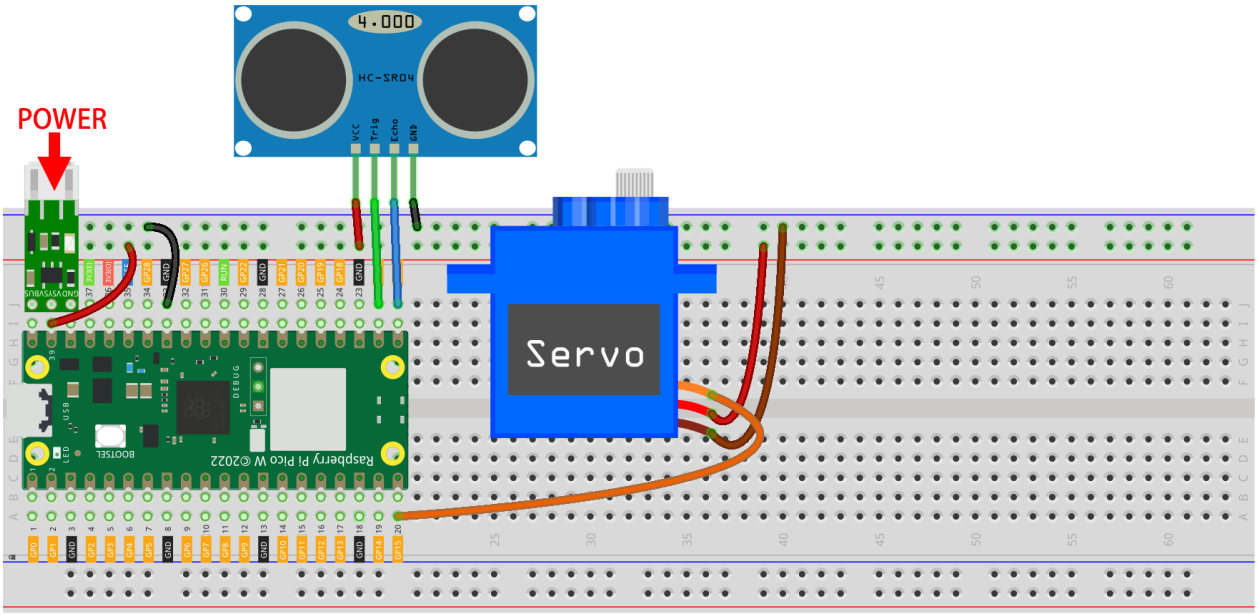
名前	このキットのアイテム	リンク
ケプラーキット	450 以上	

以下のリンクからそれぞれ個別に購入することもできます。

番号	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	
6	**超音波モジュール**	1	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	

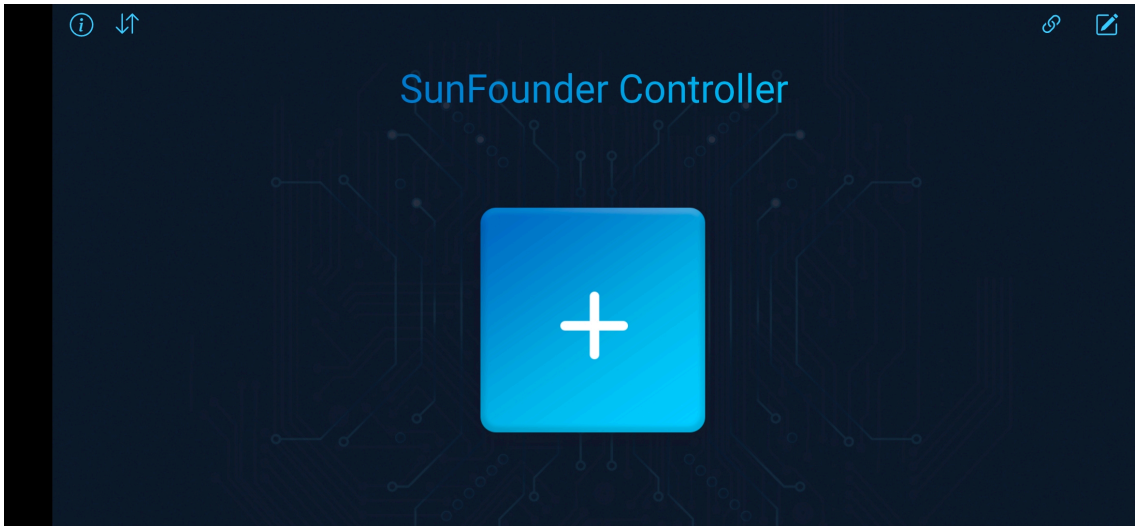
2. 回路の構築

警告: ダイヤグラムに示されているように、Li-po チャージャーモジュールが接続されていることを確認してください。そうでない場合、ショートが起きてバッテリーや回路が損傷する可能性があります。

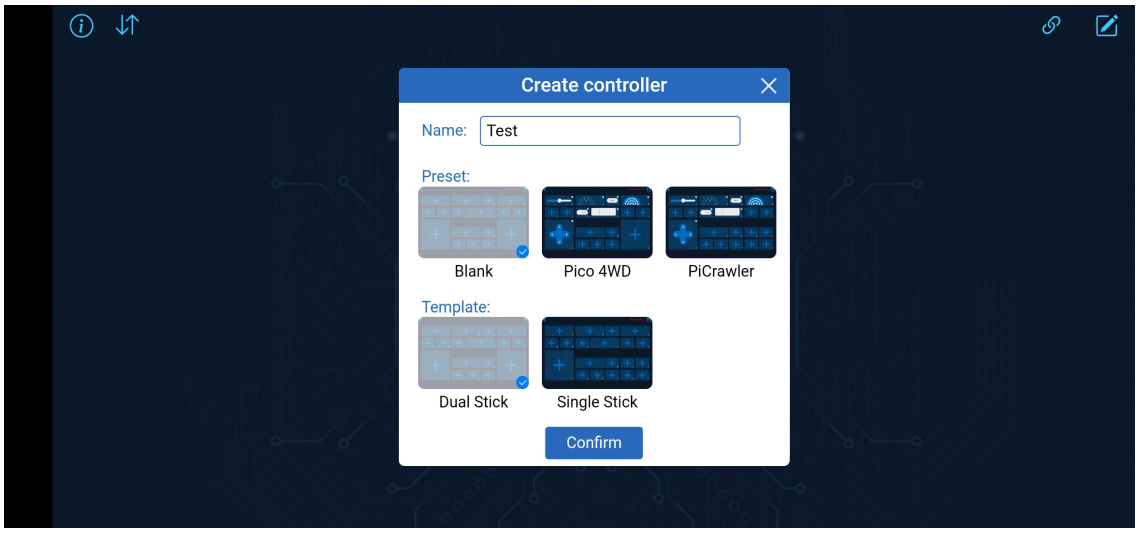


3. SunFounder コントローラーのセットアップ

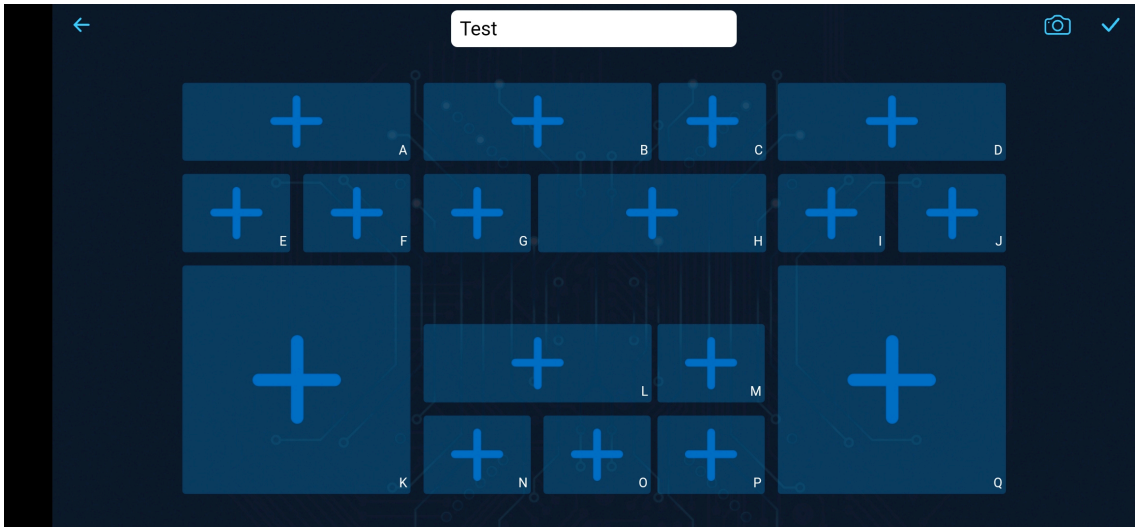
1. SunFounder Controller APP を **APP Store(iOS)** または **Google Play(Android)** からインストールします。



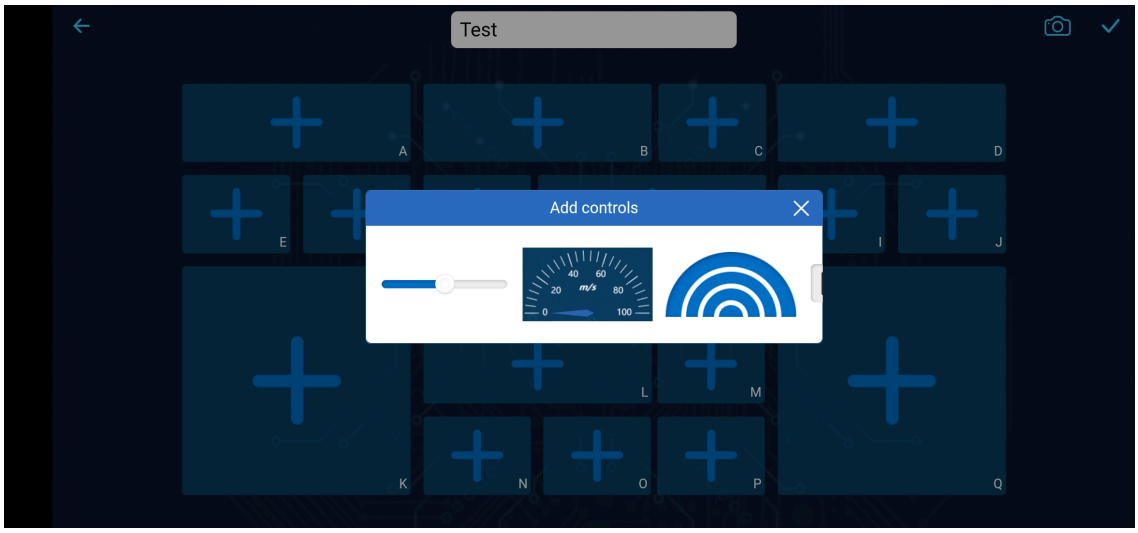
2. APP を開いて、ホームページの **+** ボタンをクリックしてコントローラーを作成します。



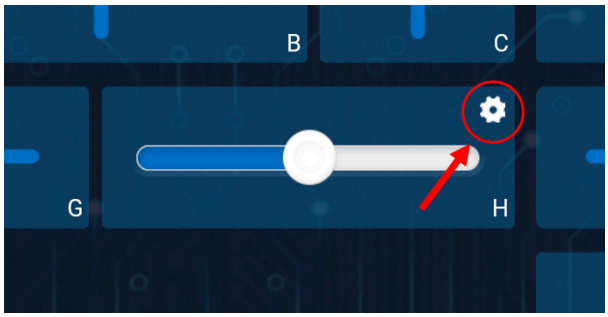
3. ここでは **Blank** と **Dual Stick** を選びます。



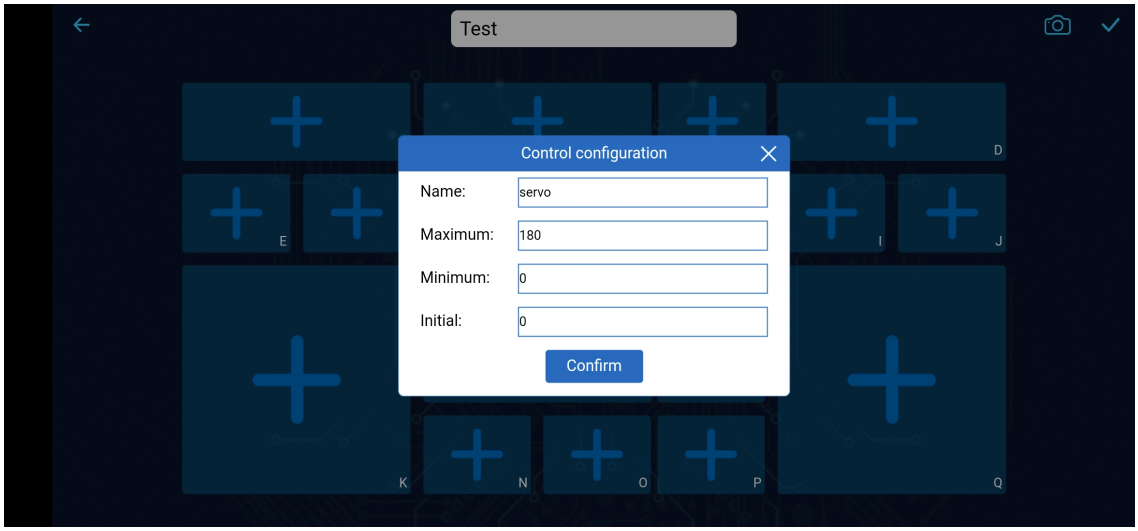
4. 空のコントローラーが表示されます。



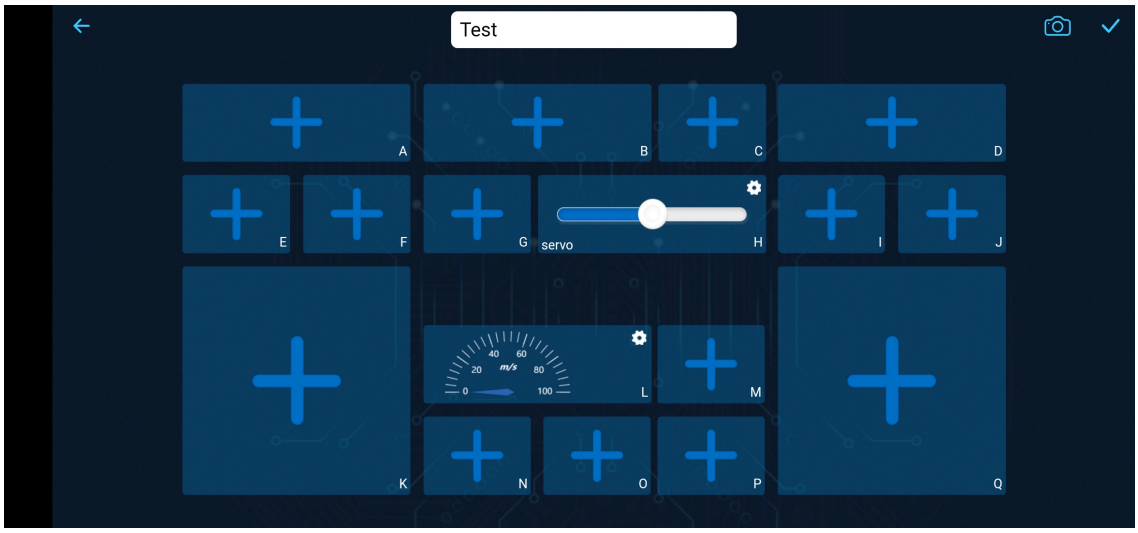
5. **H** エリアをクリックして、 **Slider** ウィジェットを追加します。



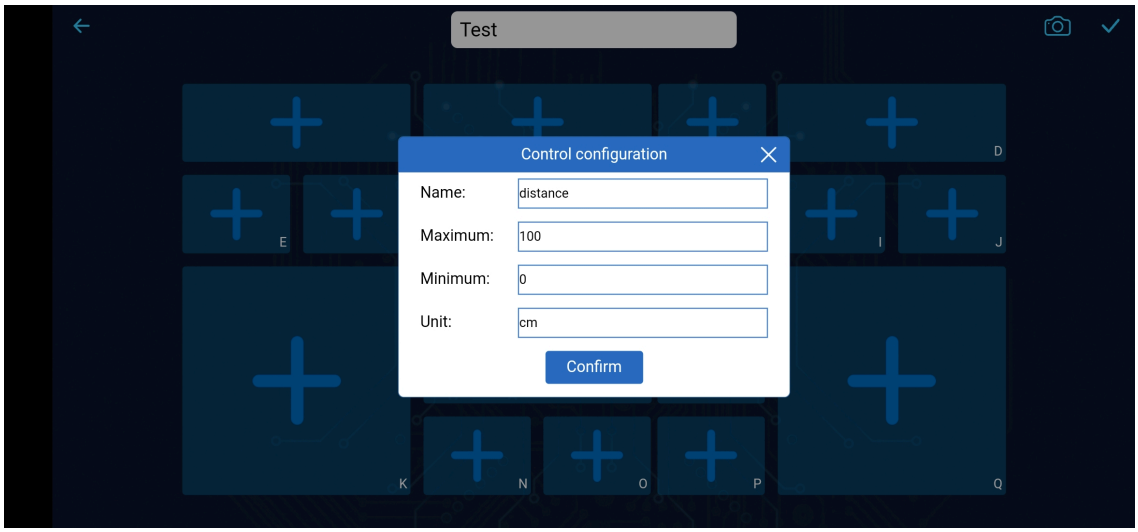
6. コントロールの歯車をクリックして設定ウィンドウを開きます。



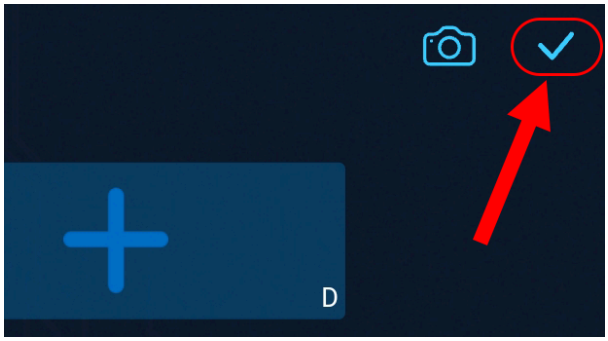
7. 最大値を 180、最小値を 0 に設定し、 **Confirm** をクリックします。



8. L エリアをクリックして、ゲージウィジェットを追加します。



9. ゲージの歯車をクリックして設定ウィンドウを開き、最大値を 100、最小値を 0、単位を cm に設定します。

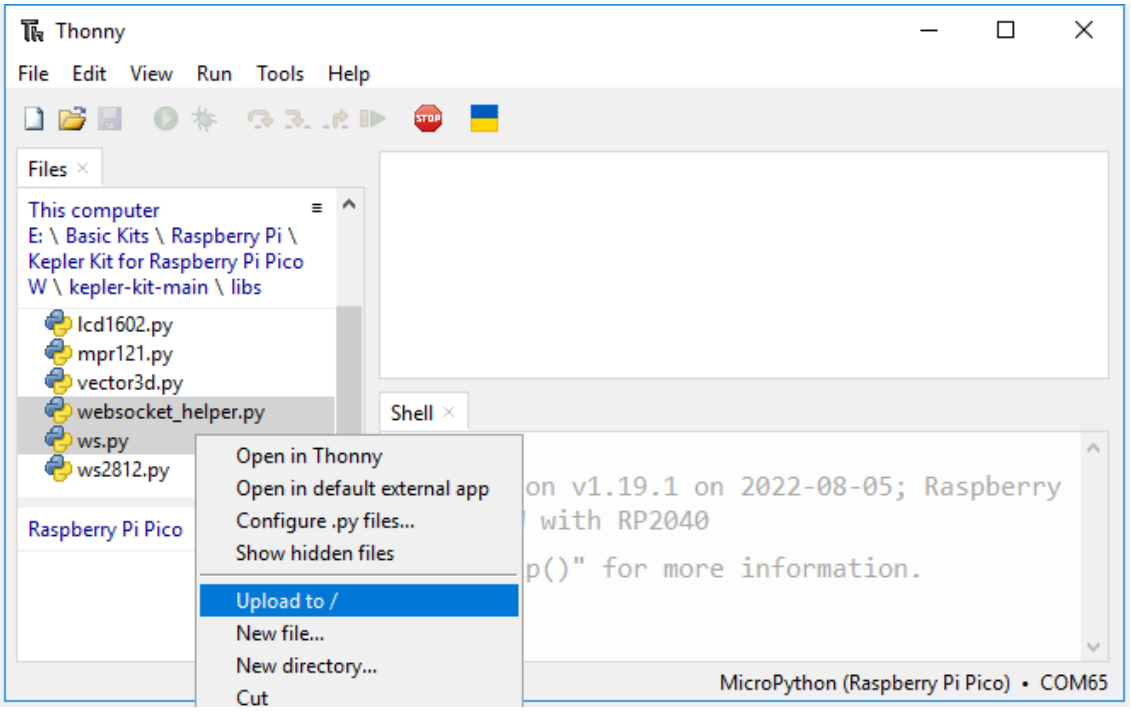


10. ウィジェットの設定が完了したら、保存をクリックします。

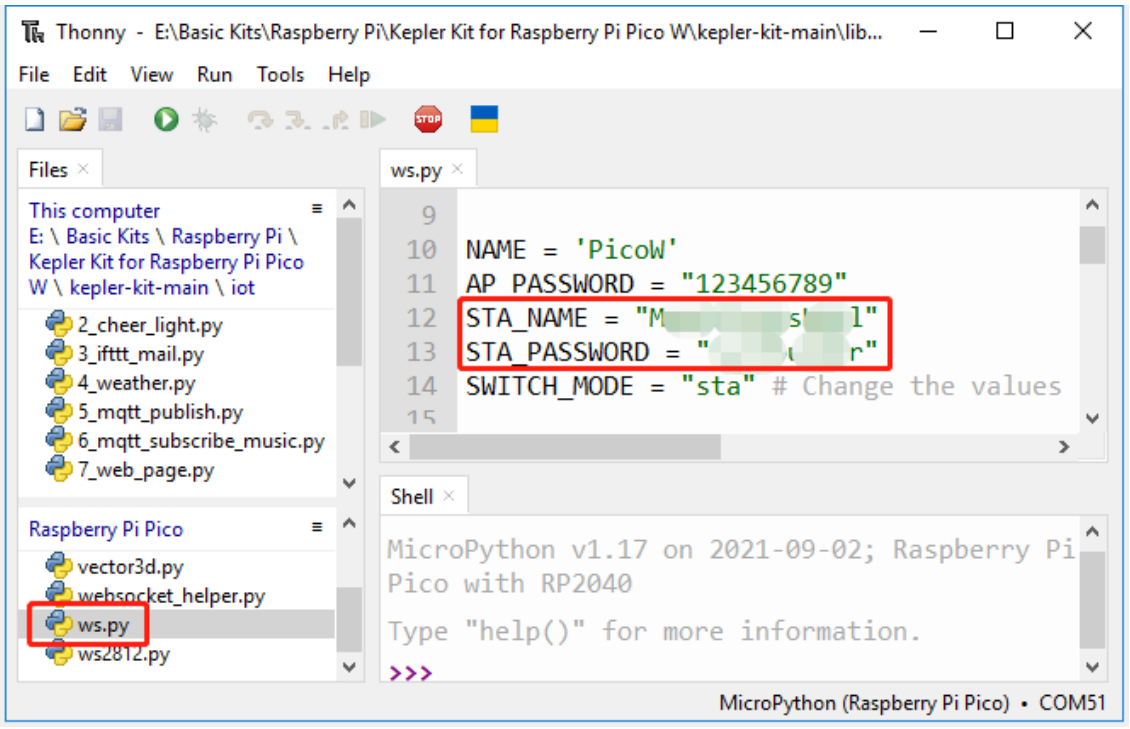
4. コードの実行

注釈: Pico W が現在 Anvil ファームウェアを使用している場合、 *1.3 Raspberry Pi Pico に MicroPython をインストール* が必要です。

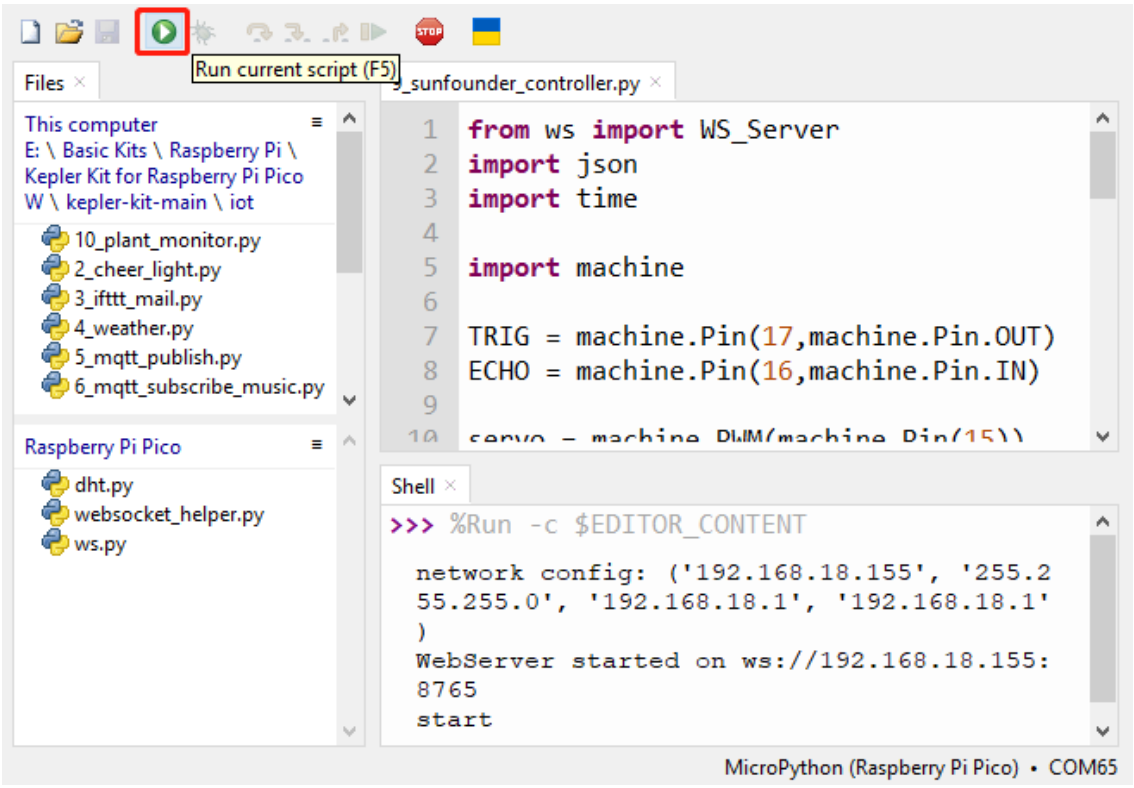
1. `kepler-kit-main/libs` のパスから `ws.py` と `websocket_helper.py` を Raspberry Pi Pico W にアップロードします。



2. `ws.py` スクリプトをダブルクリックして、WiFi の `SSID` と `PASSWORD` を入力します。

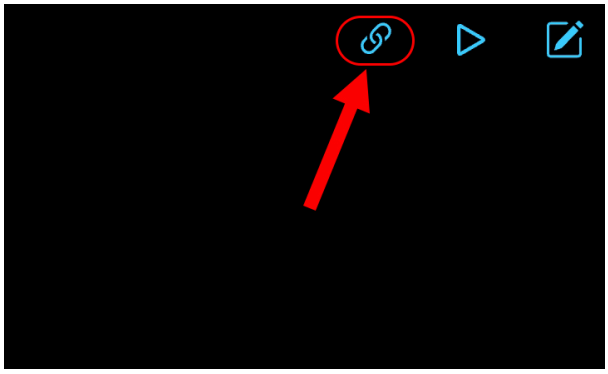


3. `kepler-kit-main/iot` のパス下の `9_sunfounder_controller.py` を開きます。 **Run current script** ボタンをクリックするか、F5 を押して実行します。接続に成功すると、Pico W の IP が表示されます。

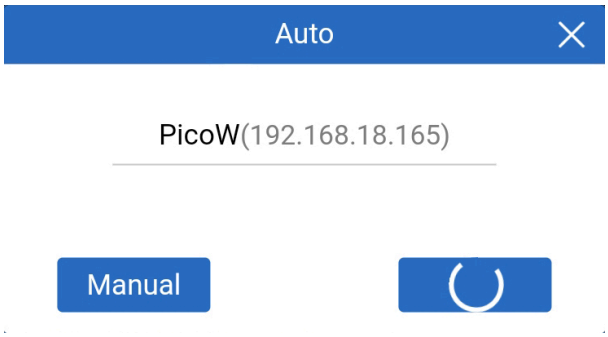


注釈: このスクリプトを起動できるようにするには、それを Raspberry Pi Pico W に `main.py` として保存できます。

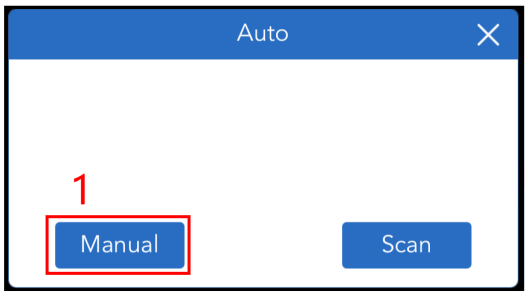
4. SunFounder Controller APP に戻り、 **Connect** ボタンをクリックします。

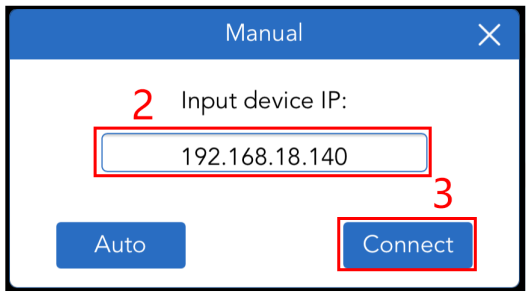


5. PicoW が検出された場合、それを直接タップして接続します。

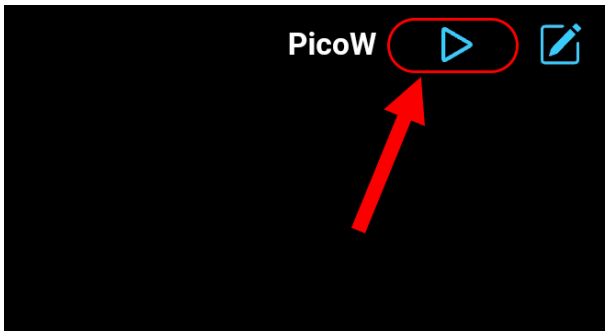


6. 自動的に検索しない場合、IP を手動で入力して接続することもできます。





7. Run ボタンをクリックした後、H エリアのスライダーバーをスライドすると、サーボが角度を調整します。L エリアのゲージは、手が超音波センサーから 100cm 以内であれば距離を表示します。



どのように動作するか？

ws.py ライブラリの WS_Server クラスは、APP との通信を実装しています。以下は、その基本機能を実装するためのフレームワークです。

```
from ws import WS_Server
import json
import time

ws = WS_Server(8765)  # WebSocket の初期化

def main():
    ws.start()
    while True:
        status, result = ws.transfer()
        time.sleep_ms(100)

try:
    main()
finally:
    ws.stop()
```

まず、 WS_Server オブジェクトを作成する必要があります。

```
ws = WS_Server(8765)
```

それを開始します。

```
ws.start()
```

次に、 while True ループを使用して、Pico W と SunFounder Controller APP 間でのデータ転送を実行します。

```
while True:
```

(次のページに続く)

(前のページからの続き)

```
    # WebSocket でデータを転送
    status, result = ws.transfer()

    # 転送データの状態
    print(status)

    # 受信したデータ
    print(result)

    # 送信するデータ
    print(ws.send_dict)

    time.sleep_ms(100)
```

status は、SunFounder Controller APP からデータを取得できなかった場合に False です。

そして、 result は、Pico W が SunFounder Controller APP からフェッチしたデータです。それを出力すると、以下のようなものが表示されます。これは、すべてのウィジェットエリアの値です。

```
{'C': None, 'B': None, 'M': None,,,,, 'A': None, 'R': None}
```

このケースでは、H エリアの値を別々に出力し、それを使用して回路を操作します。

```
status, result = ws.transfer()
#print(result)
if status == True:
    print(result['H'])
```

そして、ws.send_dict 辞書は、Pico W が SunFounder Controller APP に送信するデータです。これは WS_Server クラスで作成されています。 ws.transfer() が実行されたときに送信されます。

そのメッセージは以下の通りです。

```
{'Check': 'SunFounder Controller', 'Name': 'PicoW', 'Type': 'Blank'}
```

これは空のメッセージであり、SunFounder Controller APP のウィジェットにそれをコピーするには、辞書内の対応するエリアに値を割り当てる必要があります。例えば、L エリアに値 50 を割り当てます。

```
ws.send_dict['L'] = 50
```

表示されるデータは以下の通りです。

```
{'L': 50, 'Type': 'Blank', 'Name': 'PicoW', 'Check': 'SunFounder Controller'}
```

SunFounder Controller の詳しい使い方については、 SunFounder Controller APP をご参照ください。

5.10 10. @SunFounder Controller で植物モニター

このプロジェクトでは、Sunfounder Controller APP を使用して植物の水やりシステムを作成する方法を学びます。

APP 上で、環境の現在の温度と湿度、および鉢植えの植物の水位を確認できます。また、APP 上のボタンをクリックして植物に水をやることもできます。

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

一式をまとめて購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	リンク
Kepler キット	450+	

以下のリンクから個別にも購入できます。

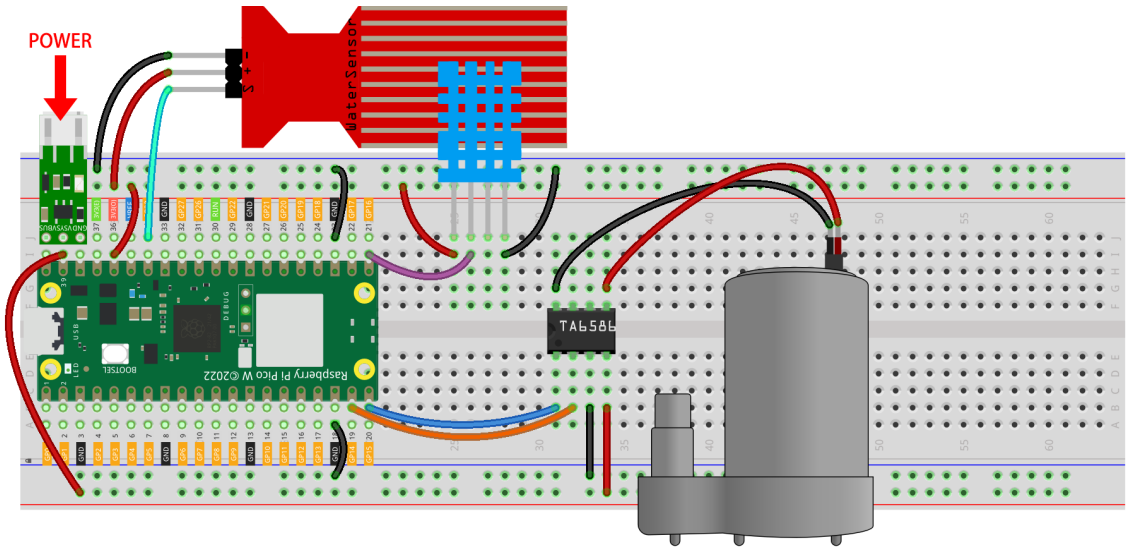
SN	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*DHT11* **湿温度センサー**	1	
6	**水位センサーモジュール**	1	
7	*TA6586* - **モータードライバーチップ**	1	
8	*Li-po* **充電モジュール**	1	
9	18650 バッテリー	1	
10	バッテリーホルダー	1	
11	*DC* **ウォーターポンプ**	1	

手順

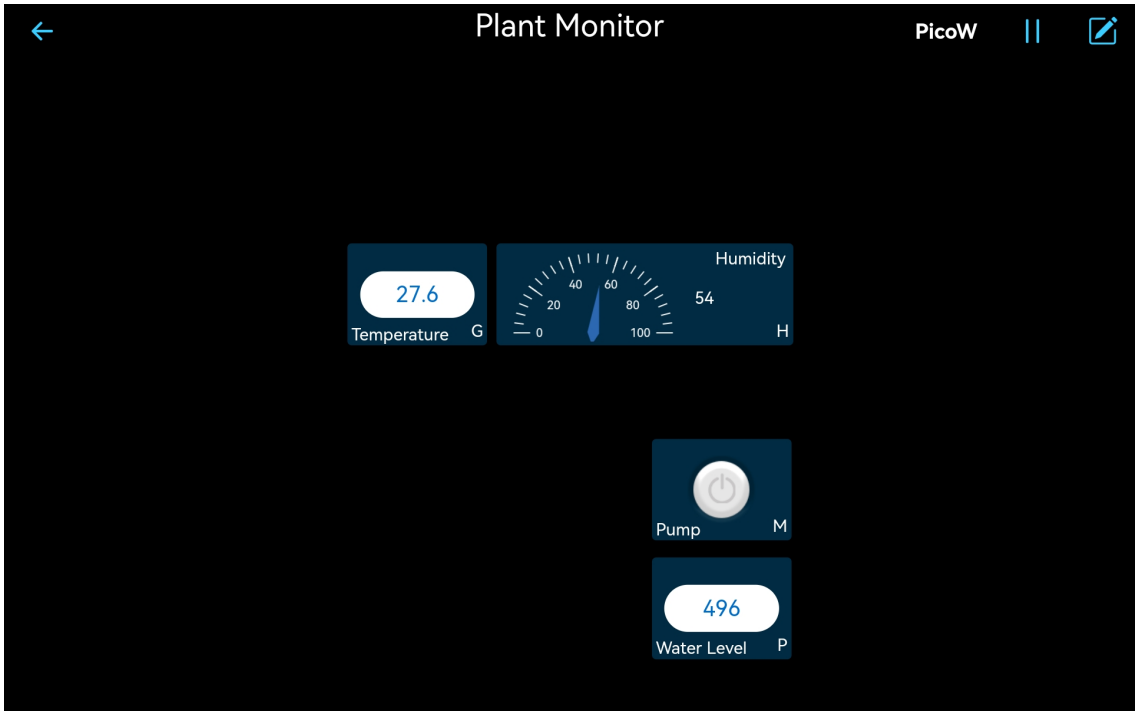
注釈: 前のプロジェクト *9. @SunFounder* **コントローラーで遊ぶ** を完了させることをお勧めします。それにより、SunFounder Controller の基本的な使い方を理解できます。

---

1. 回路を作成する。

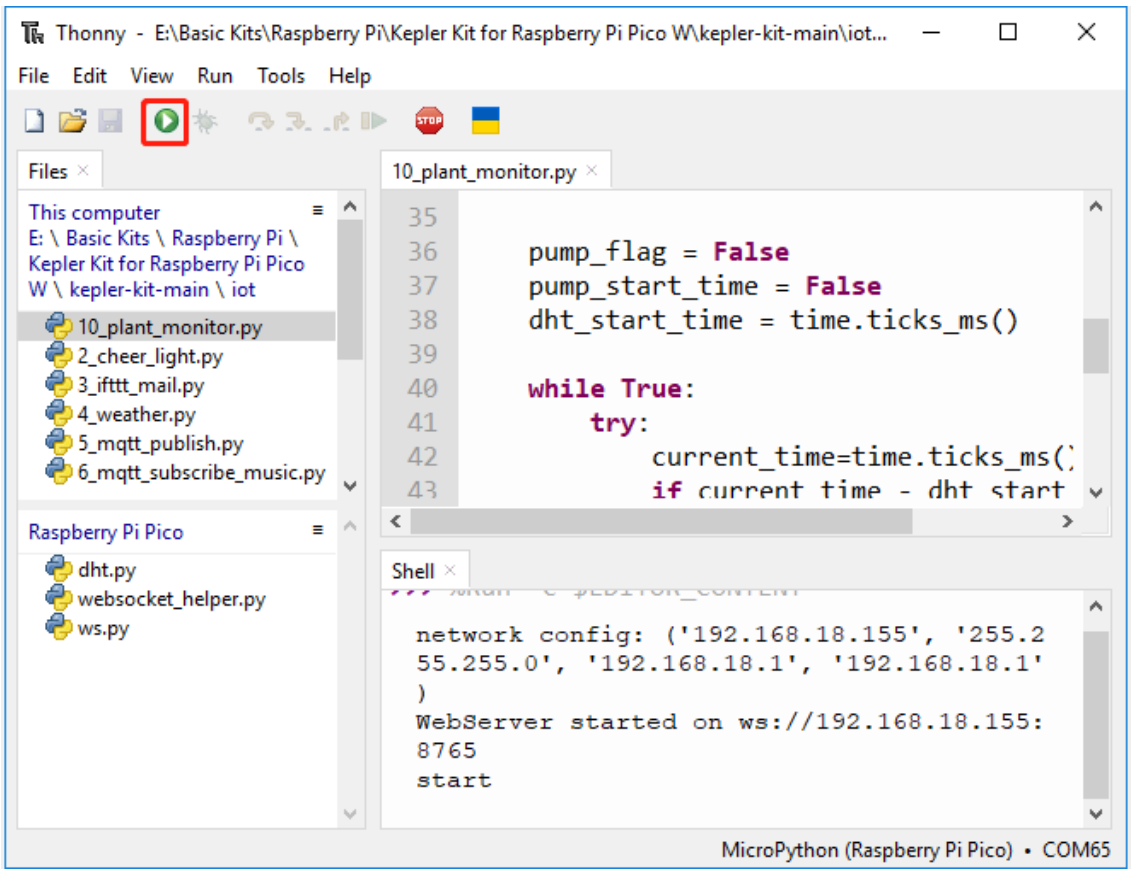


2. 新しいコントローラーを作成し、以下のウィジェットを追加して名前を変更する。

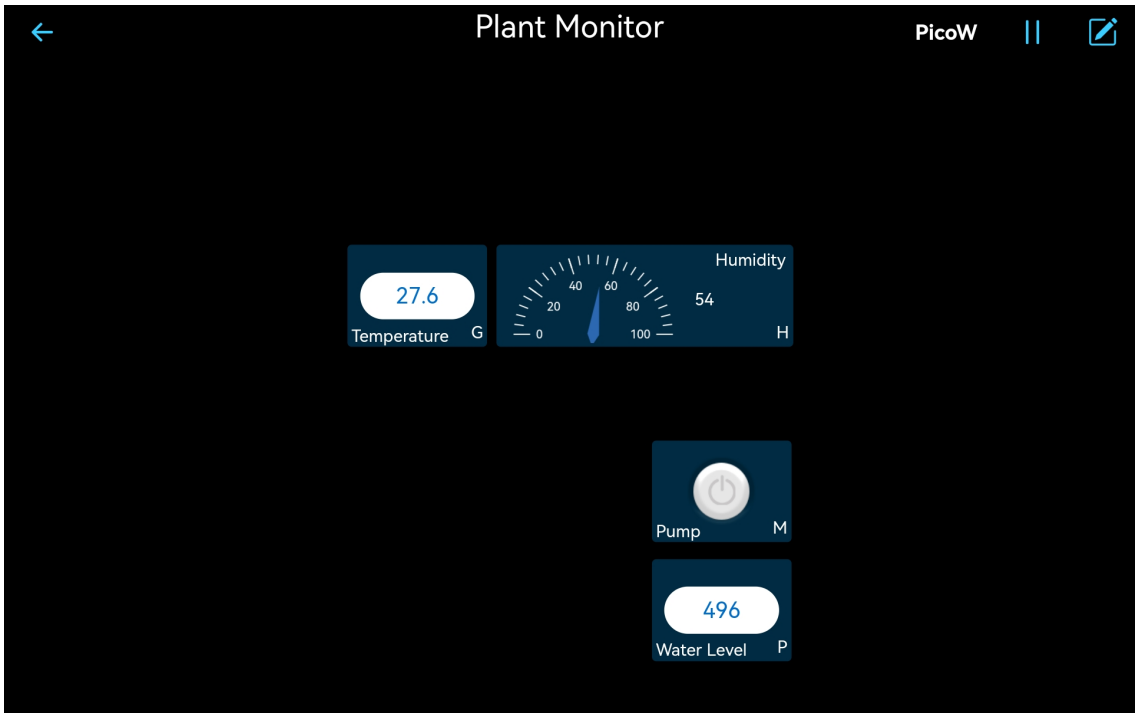


3. `kepler-kit-main/iot` のパス内で `10_plant_monitor.py` を開く。 **Run current script** ボタンをクリッ

クするか、F5 キーを押して実行する。成功した場合、Pico W の IP が表示されます。



4. SunFounder APP に戻り、PicoW に接続した後に Run をクリックする。APP 上では、環境の温度と湿度、および鉢植え植物の水位を確認できます。水が少ないと思った場合、ボタンをクリックして鉢植え植物に 5 秒間水をやることができます。



5. このスクリプトを起動可能にしたい場合、それを Raspberry Pi Pico W に `main.py` として保存できます。

動作原理は？

このプロジェクトは基本的に *9. @SunFounder* **コントローラーで遊ぶ** と同じように動作します。

さらに、このプロジェクトでは DHT11、ポンプ、および水位モジュールも使用されています。これらのコンポーネントの使用詳細は、 *6.2* **温度・湿度センサー** 、 *3.6* **ポンピング** 、 *2.14* **水位を感知する** で確認できます。

第 6 章

Arduino ユーザー向け

この章では、Arduino IDE のインストール、Arduino IDE を使った Raspberry Pi へのコードアップロード、そして Arduino のコードを素早く学べるようにするいくつかの興味深く実用的なプロジェクトを紹介しています。

章は順番に読むことをお勧めします。

1. はじめに

6.1 1.1 Arduino IDE のインストール（重要）

Arduino IDE（Arduino Integrated Development Environment、Arduino 統合開発環境）は、Arduino プロジェクトを完成させるために必要なソフトウェアサポートを全て提供します。これは Arduino チームによって提供される、Arduino 専用のプログラミングソフトウェアであり、プログラムの記述と Arduino ボードへのアップロードが可能です。

Arduino IDE 2.0 はオープンソースプロジェクトであり、堅牢な先代である Arduino IDE 1.x から大きく進化しています。リニューアルされた UI、強化されたボード&ライブラリマネージャー、デバッガー、オートコンプリート機能など多くの新機能が追加されています。

このチュートリアルでは、Windows、Mac、Linux コンピュータに Arduino IDE 2.0 をダウンロードしてインストールする方法を説明します。

6.1.1 必要条件

- Windows - Win 10 以降、64 ビット
- Linux - 64 ビット
- Mac OS X - バージョン 10.14「Mojave」以降、64 ビット

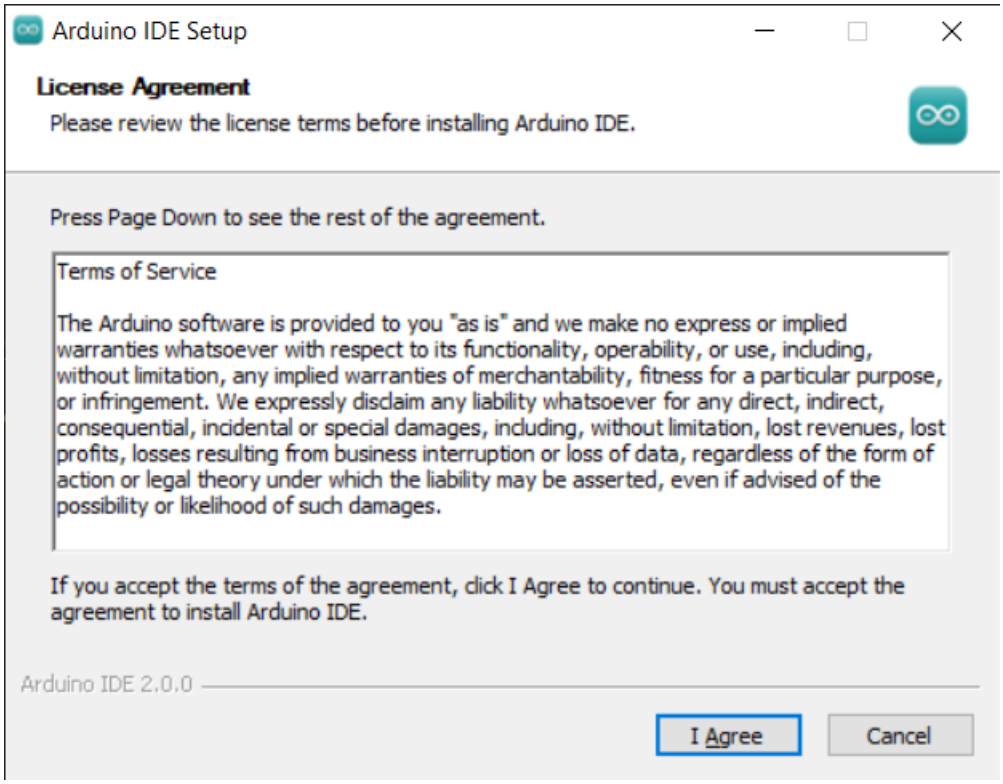
6.1.2 Arduino IDE 2.0 をダウンロード

1. ページを訪れてください。
2. お使いの OS バージョンに合わせた IDE をダウンロードしてください。

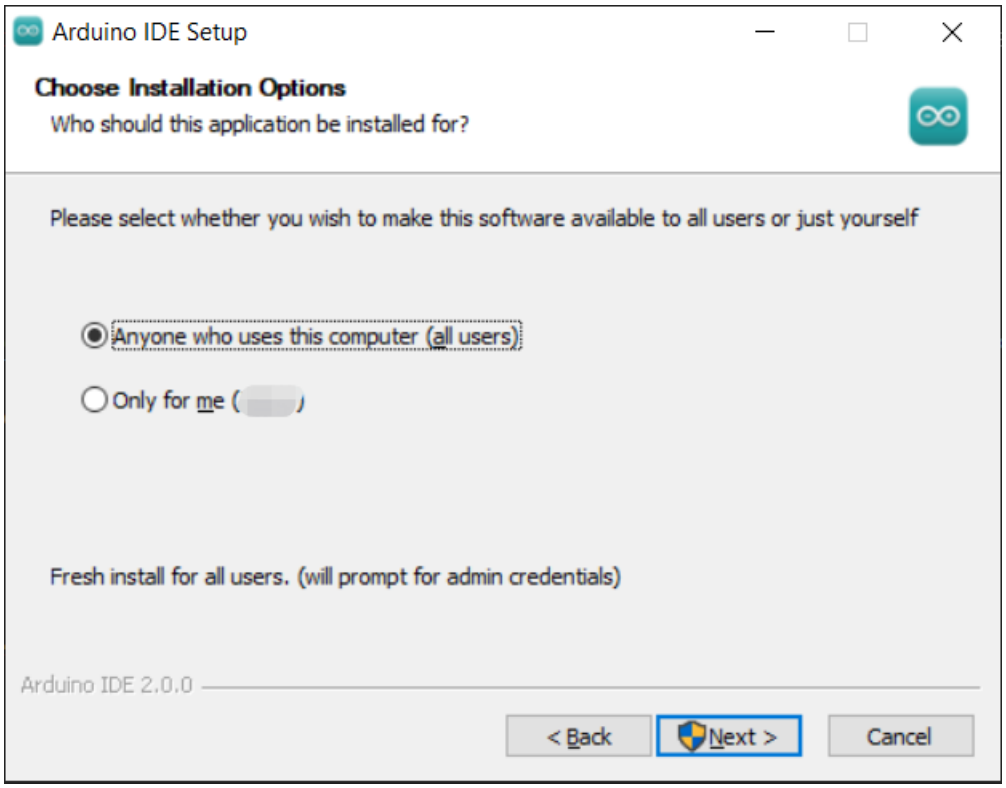
6.1.3 インストール

Windows

1. ダウンロードした `arduino-ide_xxxx.exe` ファイルをダブルクリックして実行します。
2. ライセンス契約を読み、同意してください。

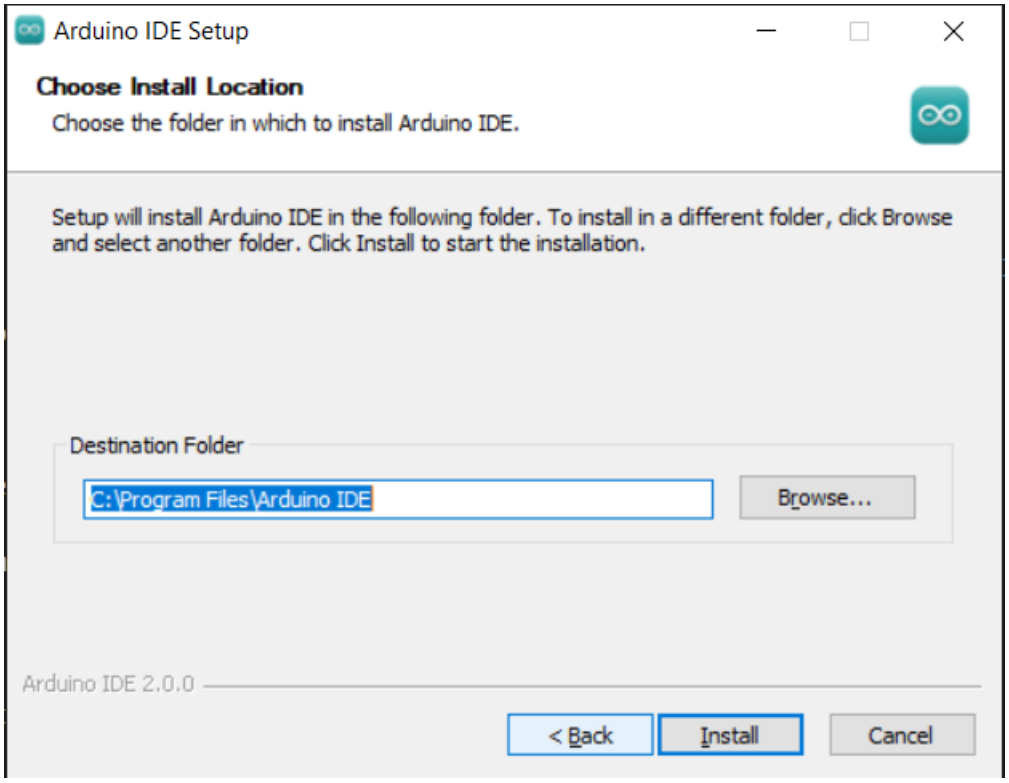


3. インストールオプションを選択します。

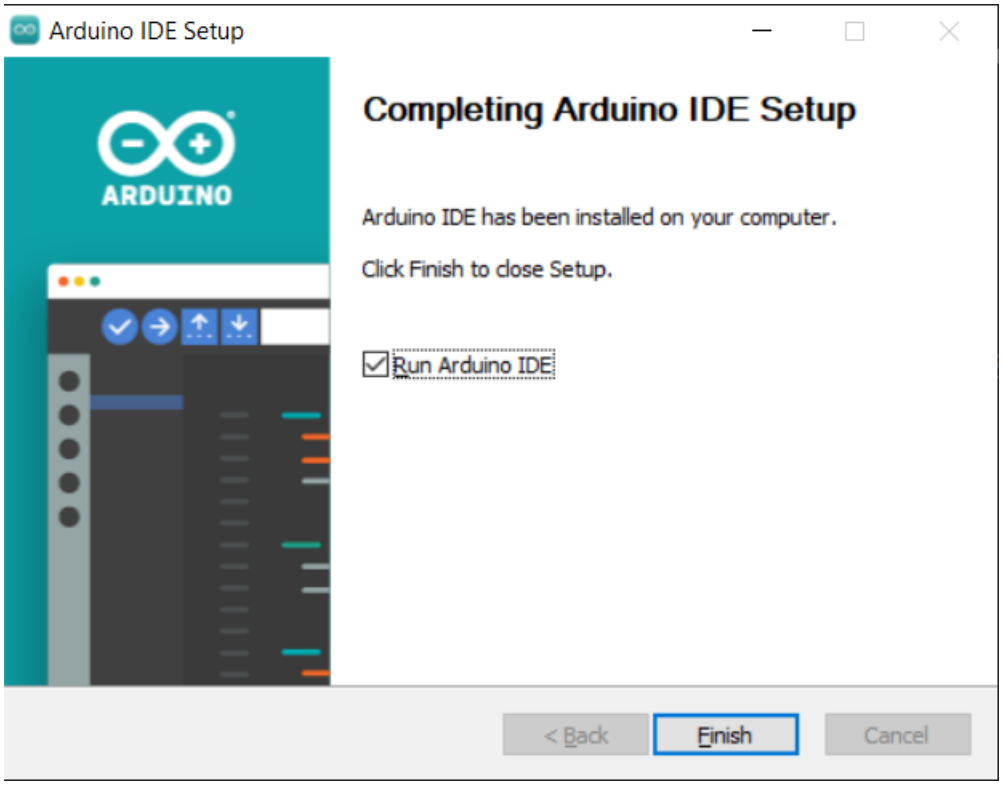


4. インストール先を選択します。システムドライブ以外にソフトウェアをインストールすることが推奨されて

います。

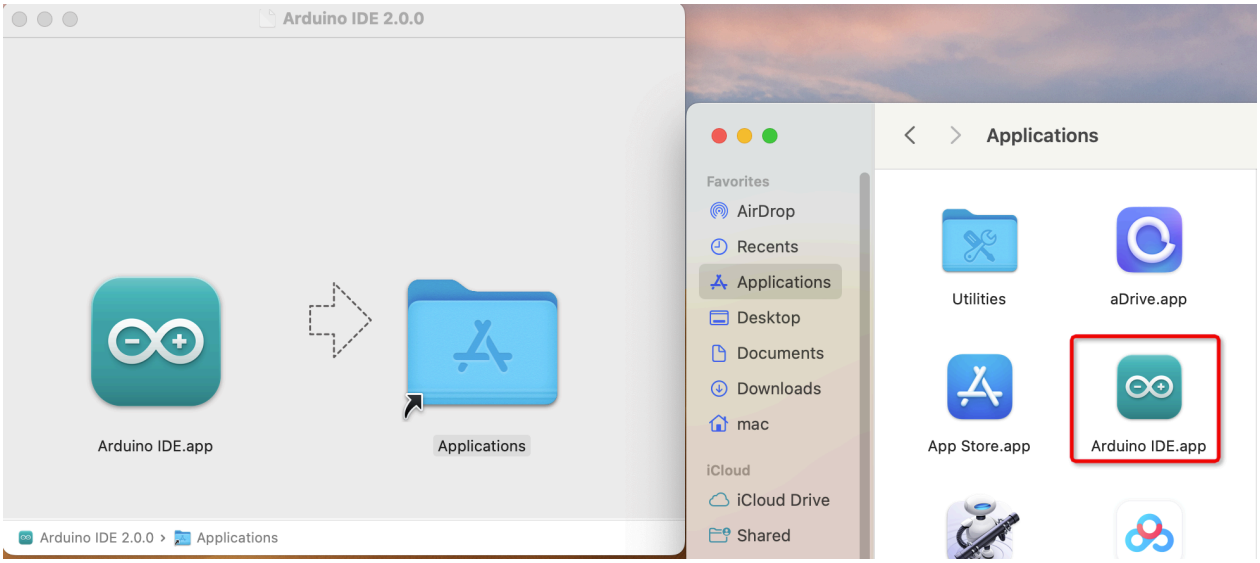


5. その後、完了します。



macOS

ダウンロードした `arduino_ide_xxxx.dmg` ファイルをダブルクリックし、指示に従って **Arduino IDE.app** を **Applications** フォルダにコピーします。数秒後に Arduino IDE が正常にインストールされたことが確認できます。

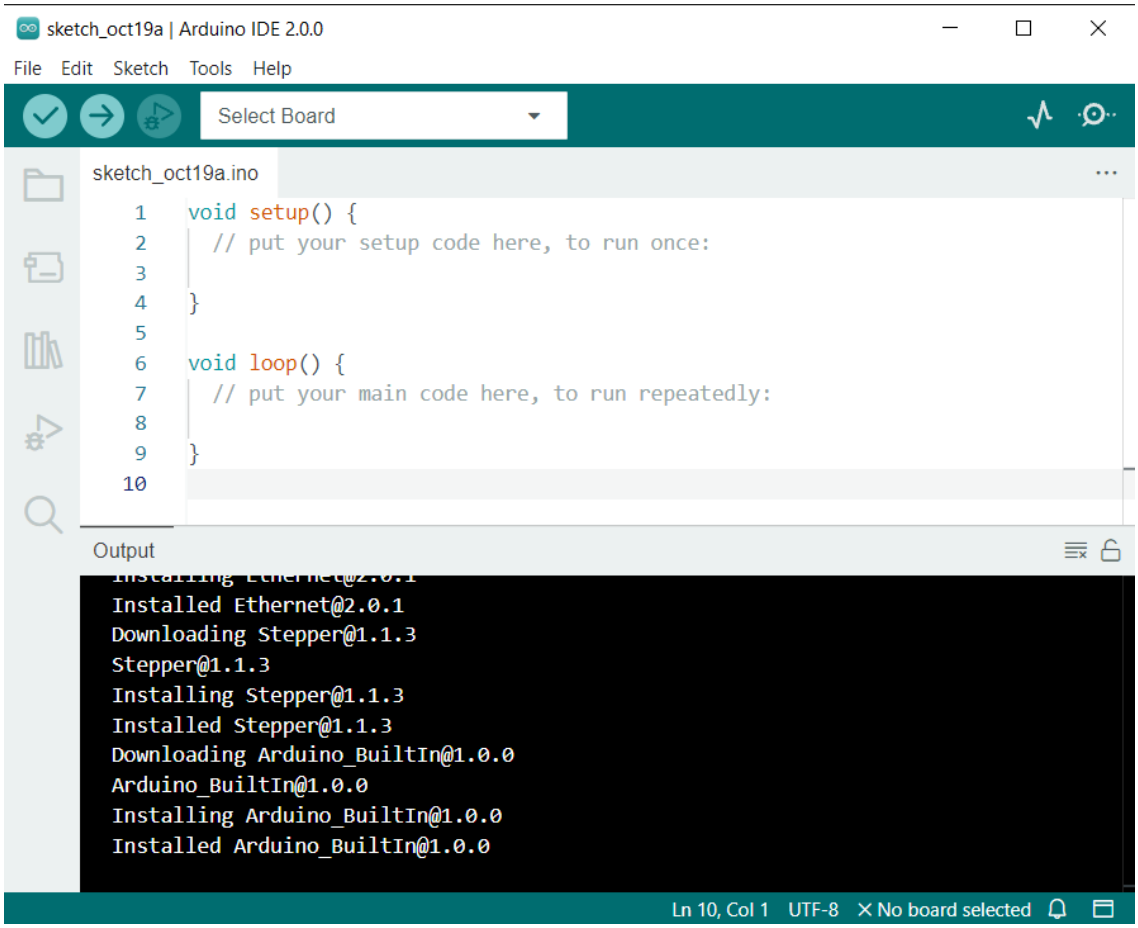


Linux

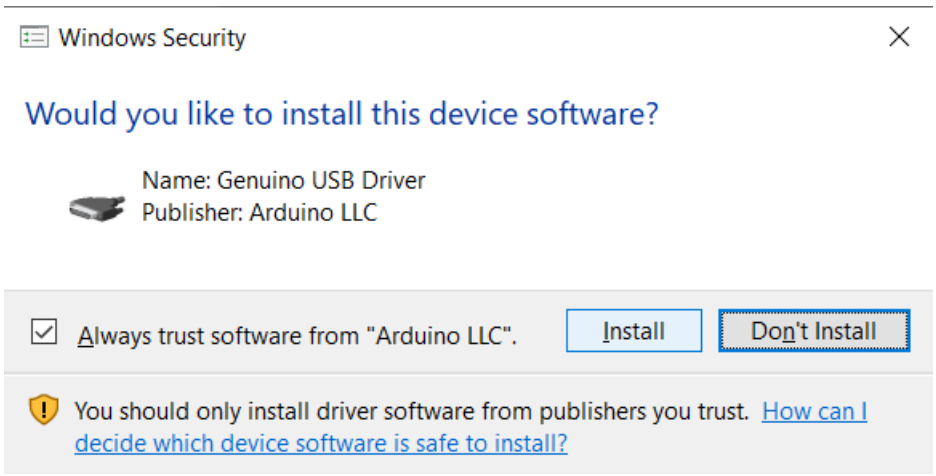
Linux システムで Arduino IDE 2.0 をインストールするチュートリアルは、以下のリンクを参照してください：
https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing#linux

6.1.4 IDE を開く

1. Arduino IDE 2.0 を初めて開くと、Arduino AVR Boards、組み込みライブラリ、その他必要なファイルが自動的にインストールされます。



2. また、ファイアウォールやセキュリティセンターがデバイスドライバーのインストールを求めるポップアップが数回表示される場合があります。全てインストールしてください。

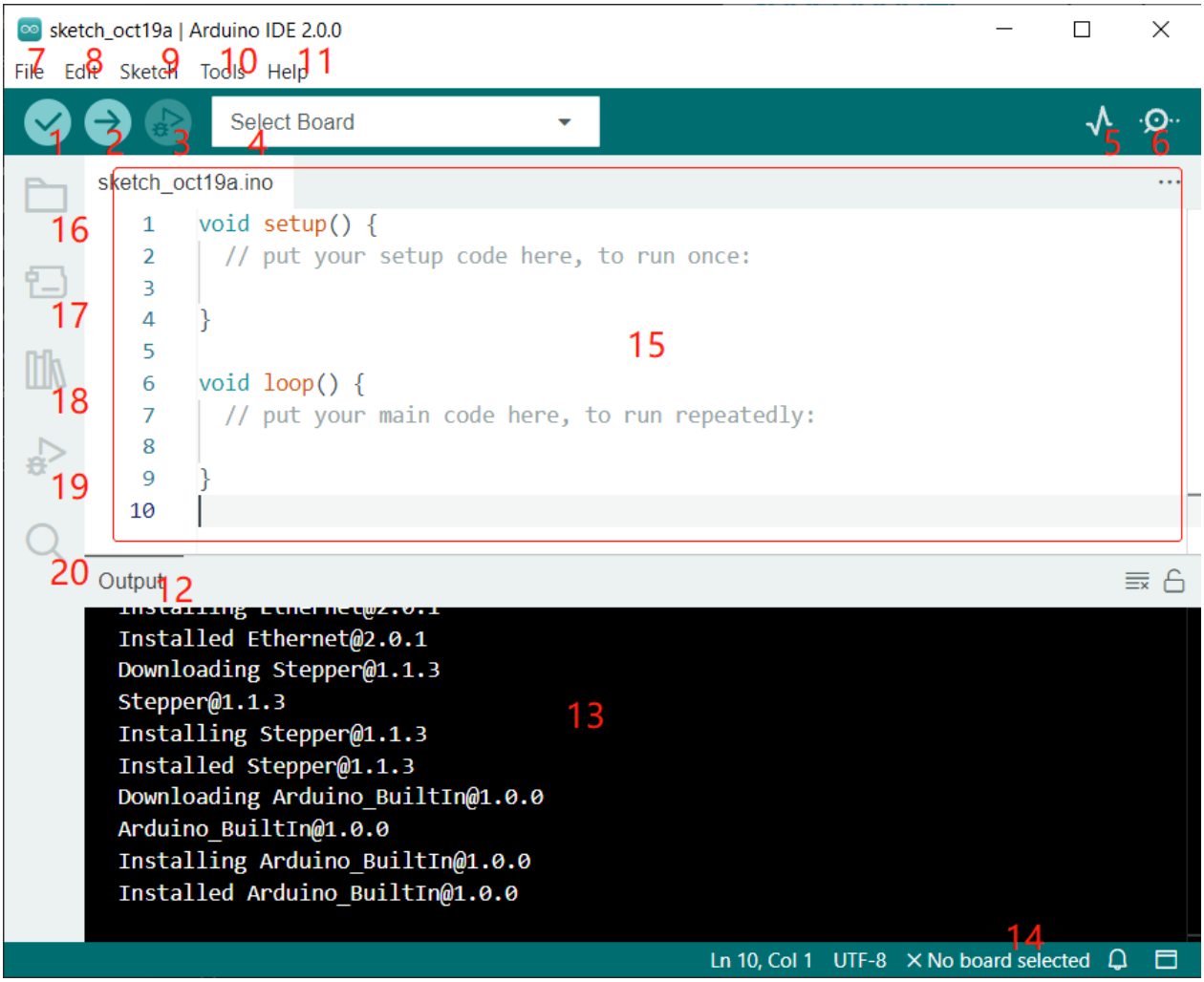


3. これで、Arduino IDE の準備が完了です！

注釈: ネットワーク問題やその他の理由で一部のインストールが失敗した場合、Arduino IDE を再度開くと残りのインストールが完了します。すべてのインストールが完了した後、Verify または Upload を

クリックしない限り、Output ウィンドウは自動的には開きません。

6.2 1.2 Arduino IDE の紹介



1. **検証（Verify）**：コードをコンパイルします。構文に問題があれば、エラーメッセージが表示されます。
2. **アップロード（Upload）**：ボードにコードをアップロードします。ボタンをクリックすると、ボード上の RX と TX の LED が高速で点滅し、アップロードが完了するまで止まりません。
3. **デバッグ（Debug）**：行ごとのエラーチェックが可能です。
4. **ボード選択（Select Board）**：ボードとポートのクイックセットアップ。
5. **シリアルプロッタ（Serial Plotter）**：読み取り値の変化を確認できます。
6. **シリアルモニタ（Serial Monitor）**：ボタンをクリックするとウィンドウが表示され、制御ボードから送信

されたデータを受信します。デバッグに非常に便利です。

7. **ファイル（File）**：メニューをクリックすると、ファイルの作成、開く、保存、閉じる、いくつかのパラメータ設定などが含まれるドロップダウンリストが表示されます。
8. **編集（Edit）**：メニューをクリックすると、**切り取り（Cut）**、**コピー（Copy）**、**貼り付け（Paste）**、**検索（Find）** など、対応するショートカットと共に編集操作がドロップダウンリストに表示されます。
9. **スケッチ（Sketch）**：**検証（Verify）**、**アップロード（Upload）**、**ファイルの追加（Add files）** などの操作が含まれます。より重要な機能は **ライブラリをインクルード（Include Library）** です、ここでライブラリを追加できます。
10. **ツール（Tool）**：頻繁に使用するボードとポートが含まれます。コードをアップロードするたびに、これらを選択または確認する必要があります。
11. **ヘルプ（Help）**：初心者の場合、このメニューの下のオプションを確認し、必要なヘルプを得ることができます。IDE の操作、紹介情報、トラブルシューティング、コードの説明などが含まれます。
12. **出力バー（Output Bar）**：ここで出力タブを切り替えます。
13. **出力ウィンドウ（Output Window）**：情報を出力します。
14. **ボードとポート（Board and Port）**：コードアップロード用に選択されたボードとポートをプレビューできます。何か誤りがあれば、**ツール（Tools）** -> **ボード（Board）** / **ポート（Port）** で再選択できます。
15. IDE の編集エリアです。ここでコードを書くことができます。
16. **スケッチブック（Sketchbook）**：スケッチファイルを管理します。
17. **ボードマネージャ（Board Manager）**：ボードドライバを管理します。
18. **ライブラリマネージャ（Library Manager）**：ライブラリファイルを管理します。
19. **デバッグ（Debug）**：コードのデバッグを支援します。
20. **検索（Search）**：自分のスケッチからコードを検索します。

6.3 1.3 Raspberry Pi Pico W のセットアップ（重要）

6.3.1 1. UF2 ファームウェアのインストール

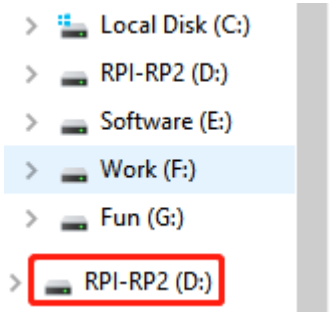
Raspberry Pi Pico W を初めて接続する際、または BOOTSEL ボタンを押しながら挿入すると、COM ポートが割り当てられずにドライブとして認識されます。これではコードのアップロードができません。

この問題を解決するには、UF2 ファームウェアをインストールする必要があります。このファームウェアは MicroPython をサポートしており、Arduino IDE とも互換性があります。

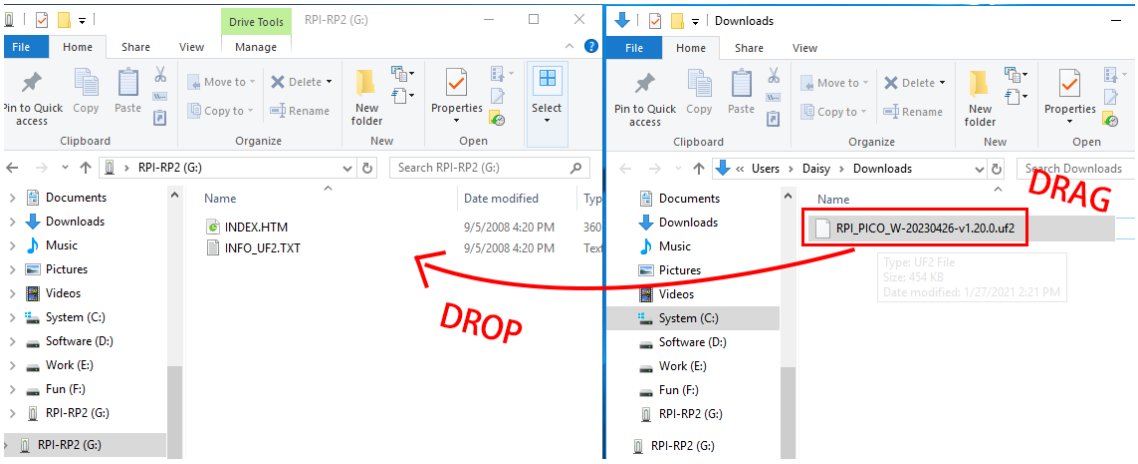
1. 以下のリンクから UF2 ファームウェアをダウンロードしてください。

    - `Raspberry Pi Pico W UF2` **ファームウェア**

2. Micro USB ケーブルを使用して、Raspberry Pi Pico W をコンピュータに接続します。Pico W は **RPI-RP2** という名前の大容量ストレージデバイスとしてマウントされます。



3. ダウンロードした UF2 ファームウェアを **RPI-RP2** ドライブにドラッグアンドドロップします。

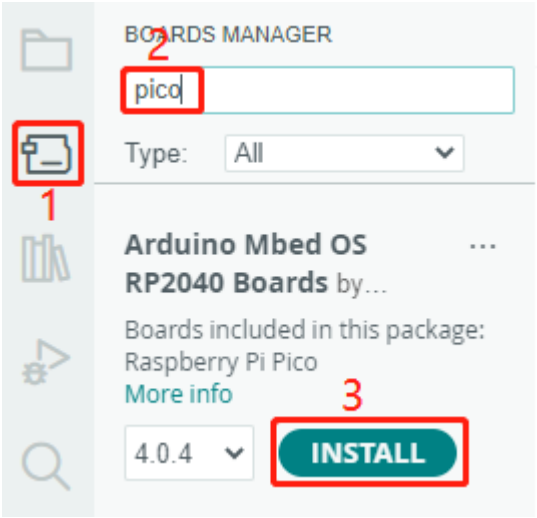


4. これにより、 **RPI-RP2** ドライブは消失し、次のステップに進むことができます。

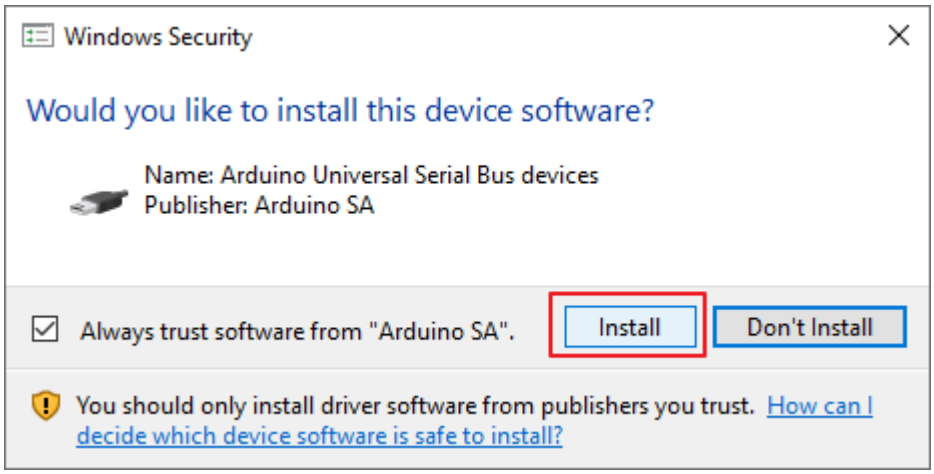
6.3.2 2. ボードパッケージのインストール

Raspberry Pi Pico W をプログラムするには、Arduino IDE に対応するパッケージをインストールする必要があります。手順は以下の通りです：

1. **ボードマネージャー** ウィンドウで **pico** と検索し、 **インストール** ボタンをクリックしてインストールを開始します。これにより、Raspberry Pi Pico W をサポートする **Arduino Mbed OS RP2040 Boards** パッケージがインストールされます。



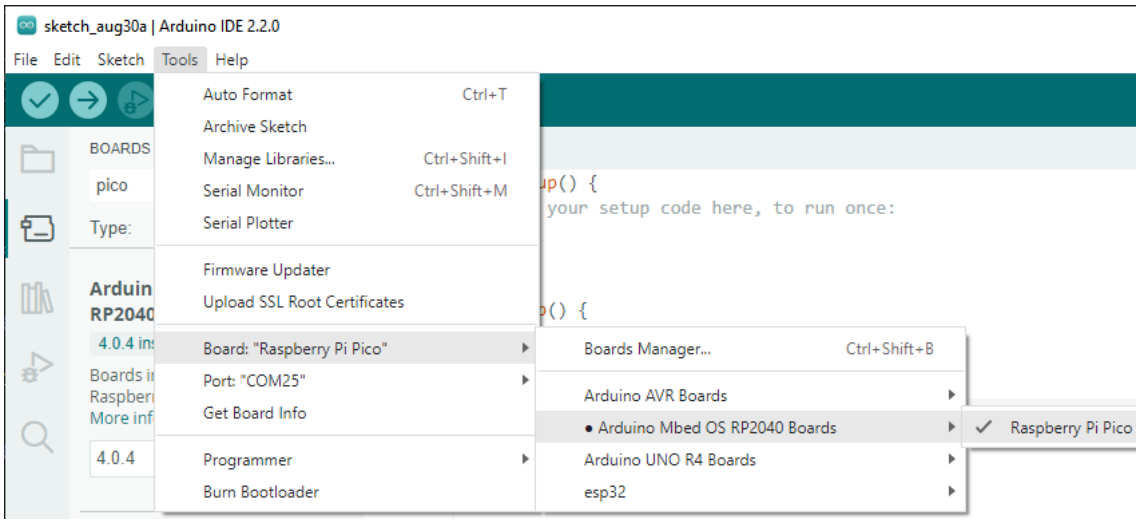
2. インストール過程で、特定のデバイスドライバーのインストールに関するポップアップが数回表示されます。 **「インストール」** を選択してください。



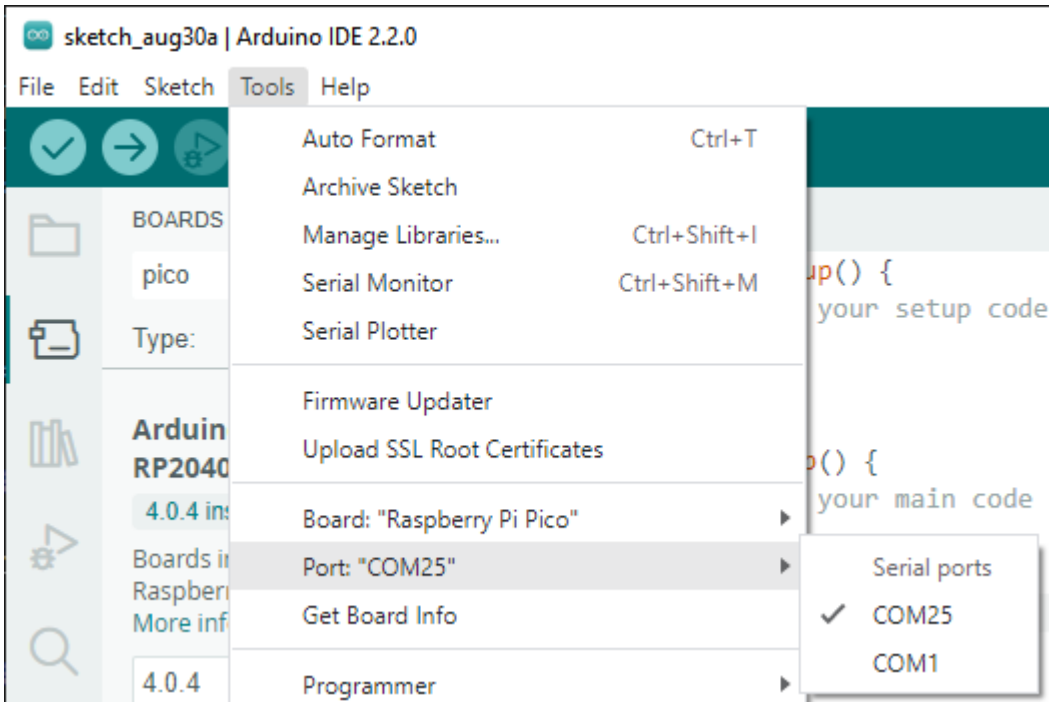
3. インストールが完了したという通知が表示されます。

6.3.3 3. ボードとポートの選択

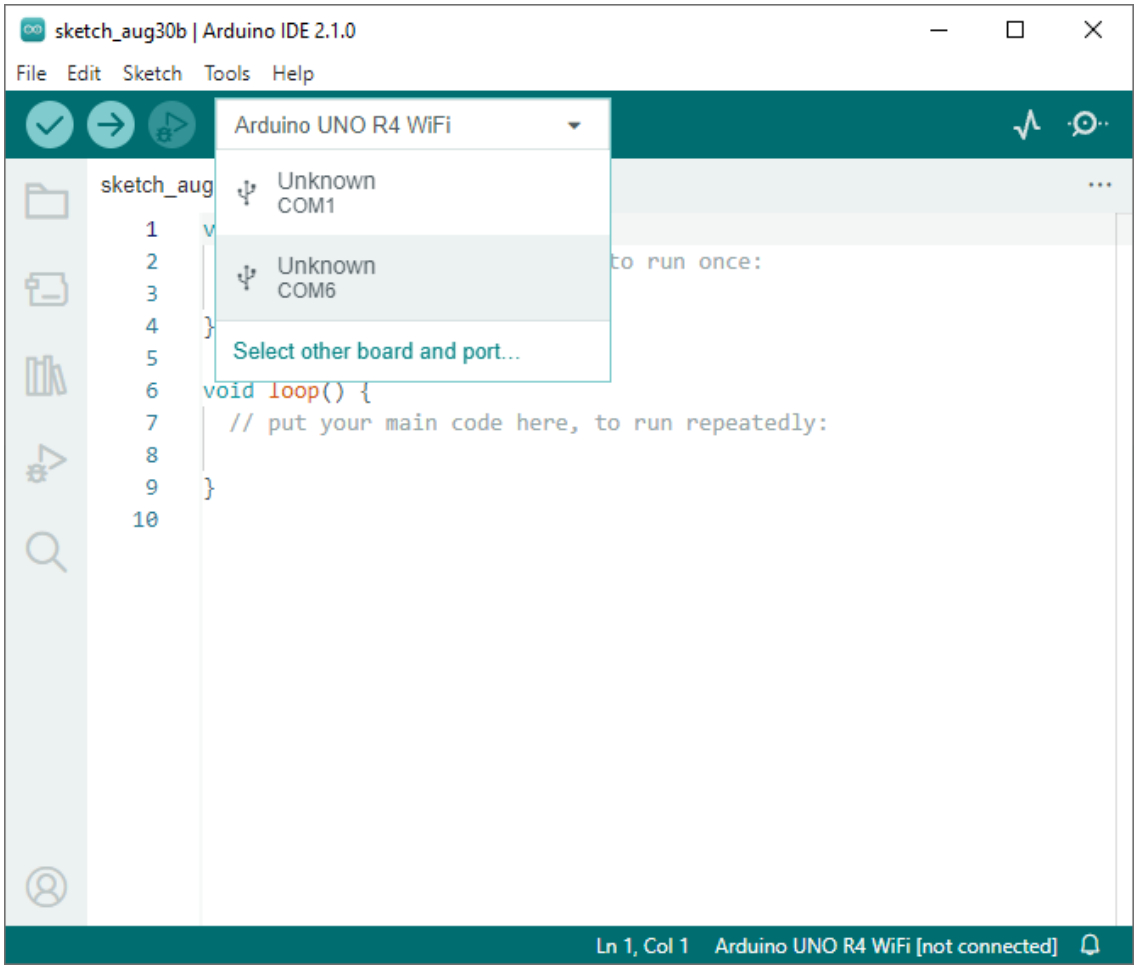
1. 適切なボードを選ぶには、 **ツール** -> **ボード** -> **Arduino Mbed OS RP2040 Boards** -> **Raspberry Pi Pico** に移動します。



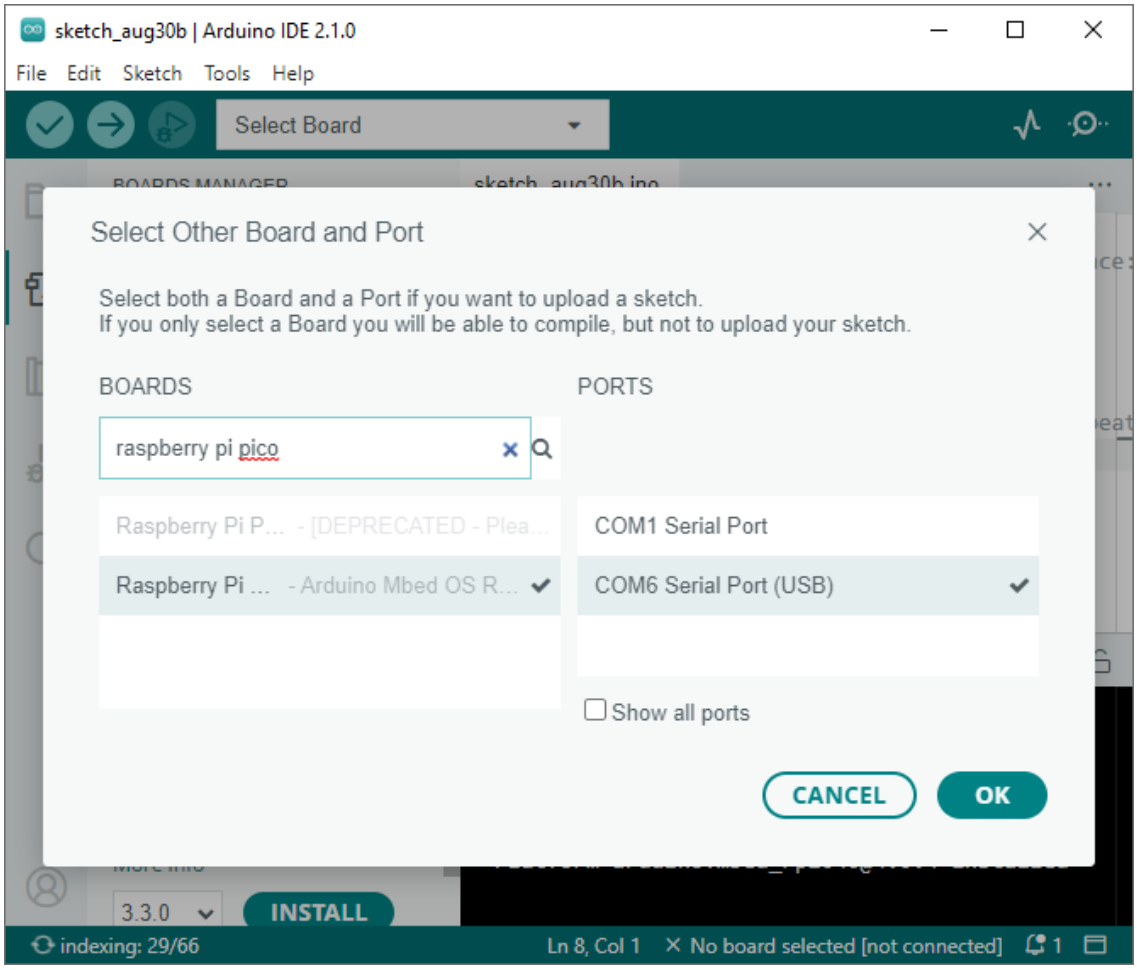
2. Raspberry Pi Pico W がコンピュータに接続されている場合、 **ツール** -> **ポート** で正しいポートを設定します。



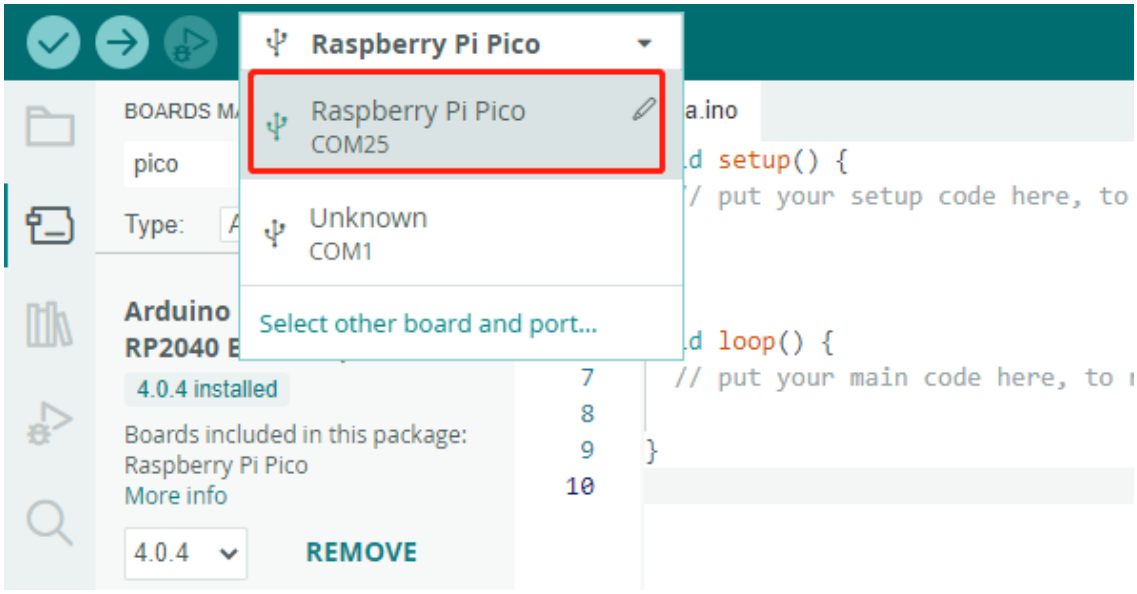
3. Arduino 2.0 は新しいクイックセレクト機能を提供しています。Raspberry Pi Pico W は通常自動認識されないため、 **その他のボードとポートを選択** をクリックします。



4. 検索バーに **Raspberry Pi Pico** と入力し、表示されたら選択し、適切なポートを選んで **OK** をクリックします。



5. このクイックアクセスウィンドウを通じて、後で簡単に再選択できます。



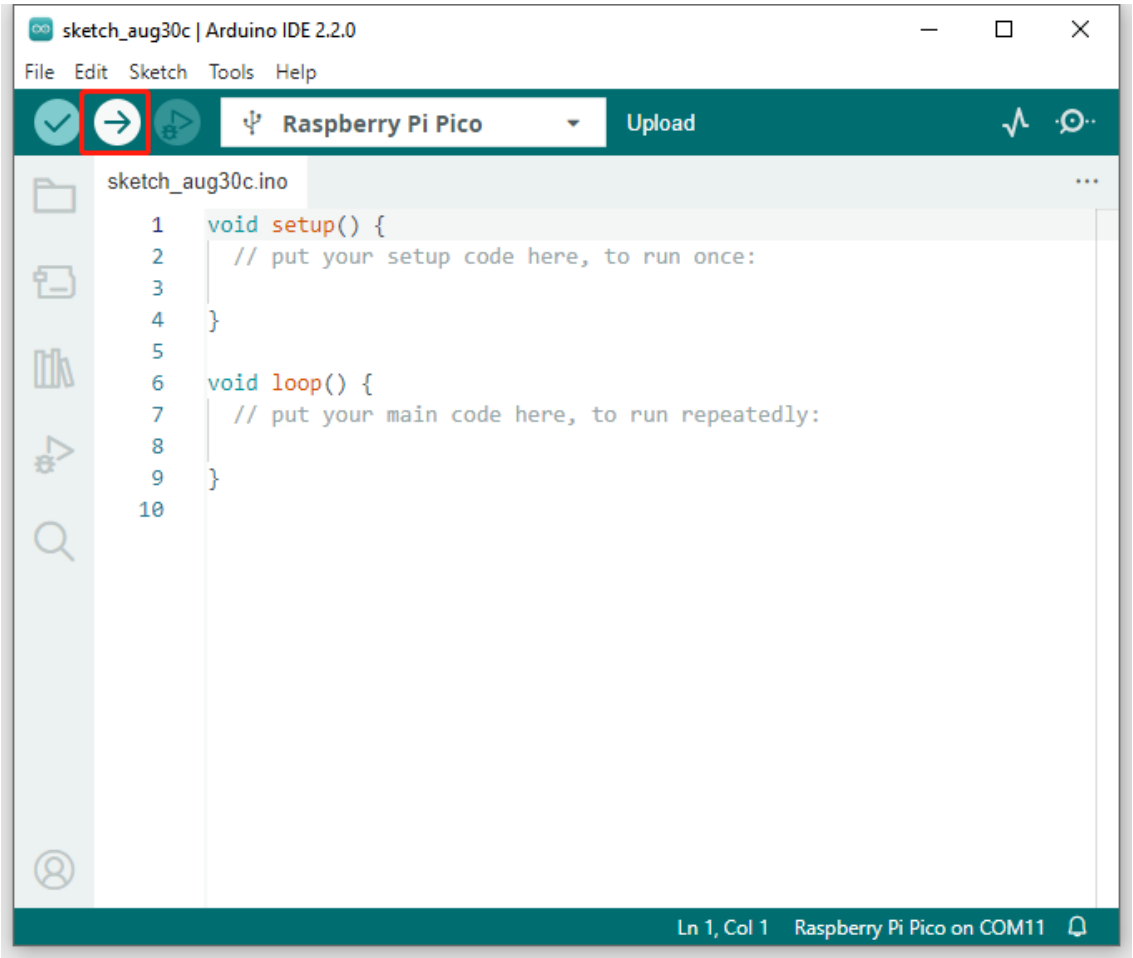
6. これらの手法のいずれかで、正確なボードとポートが設定されます。これで Raspberry Pi Pico W にコード

をアップロードする準備が整いました。

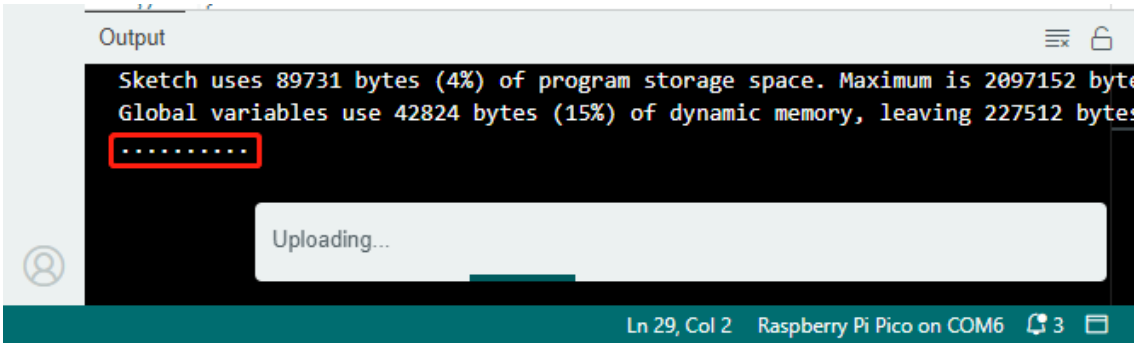
6.3.4 4. コードのアップロード

次に、Raspberry Pi Pico W にコードをアップロードする方法について説明します。

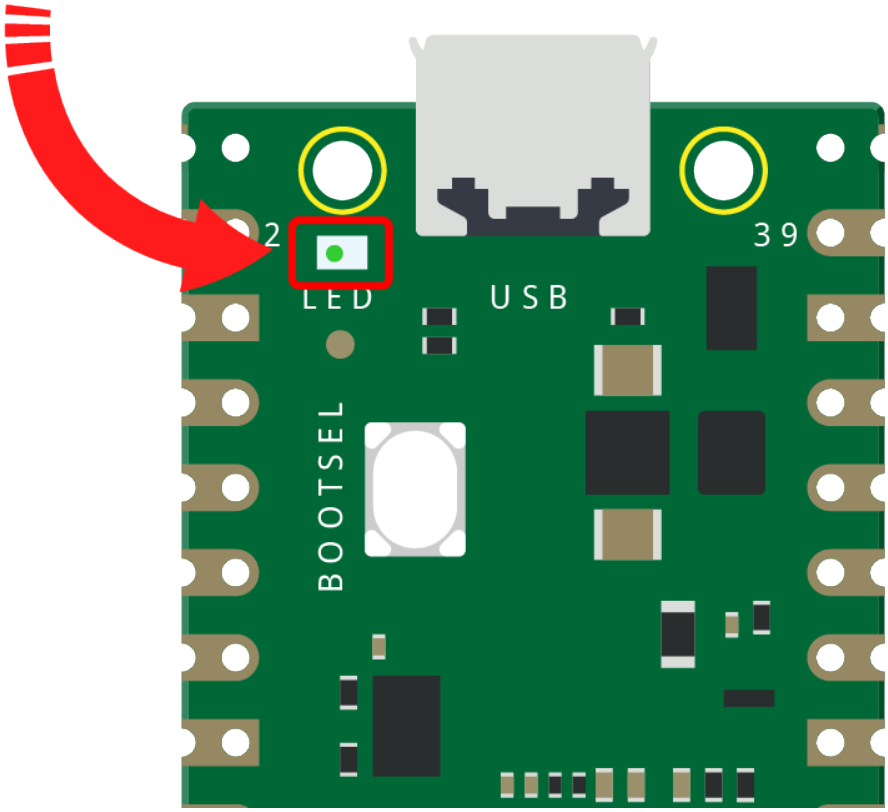
1. 任意の `.ino` ファイルを開くか、現在表示されている空のスケッチを使用します。その後、**アップロード** ボタンをクリックします。



2. アップロード中のメッセージが表示されるのを待ちます。



3. **BOOTSEL** ボタンを押しながら、Raspberry Pi Pico W の接続を素早く外し、再接続します。

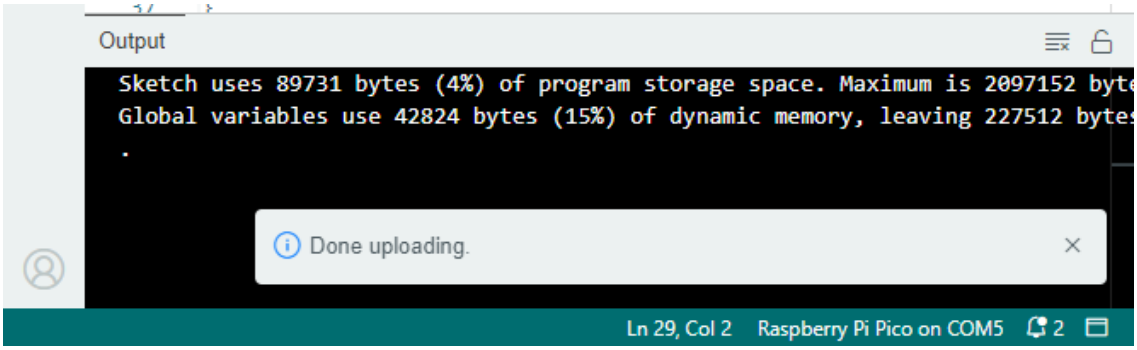


注釈:

- このステップは、特に Arduino IDE を初めて使用するユーザーにとって重要です。このステップをスキップすると、アップロードに失敗します。
- この回のコードアップロードが成功した場合、Pico W はコンピュータに認識されます。次回以降は、単純にコンピュータに接続するだけで良いです。

---

4. アップロード成功のプロンプトが表示されます。



6.4 1.4 ライブラリのインストール（重要）

コードをダウンロードする

以下のリンクから関連するコードをダウンロードしてください。

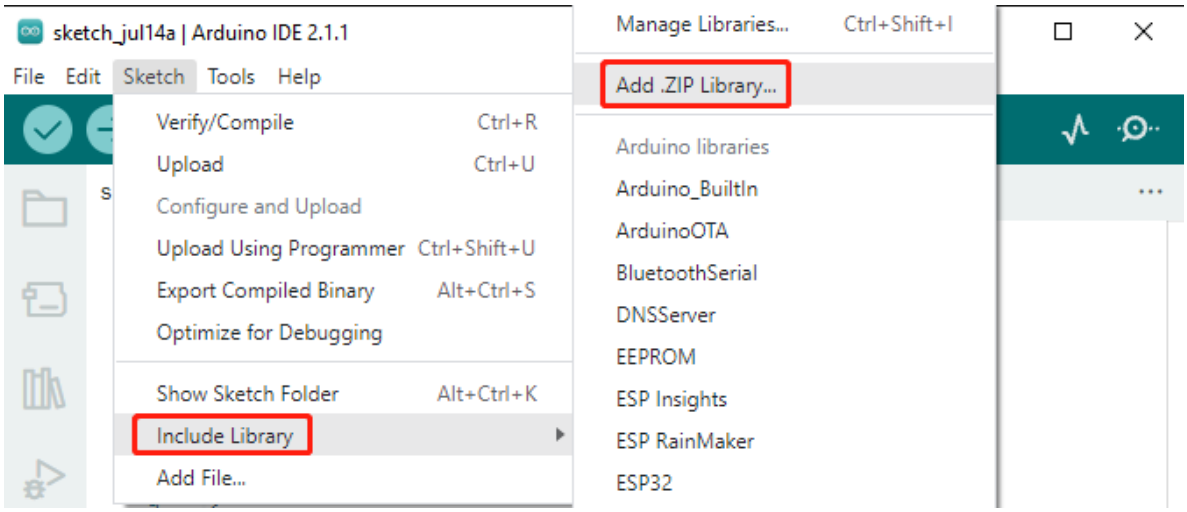
- **SunFounder Kepler Kit のサンプル**
- または、 Kepler Kit - GitHub でコードを確認できます。

6.4.1 ライブラリを追加

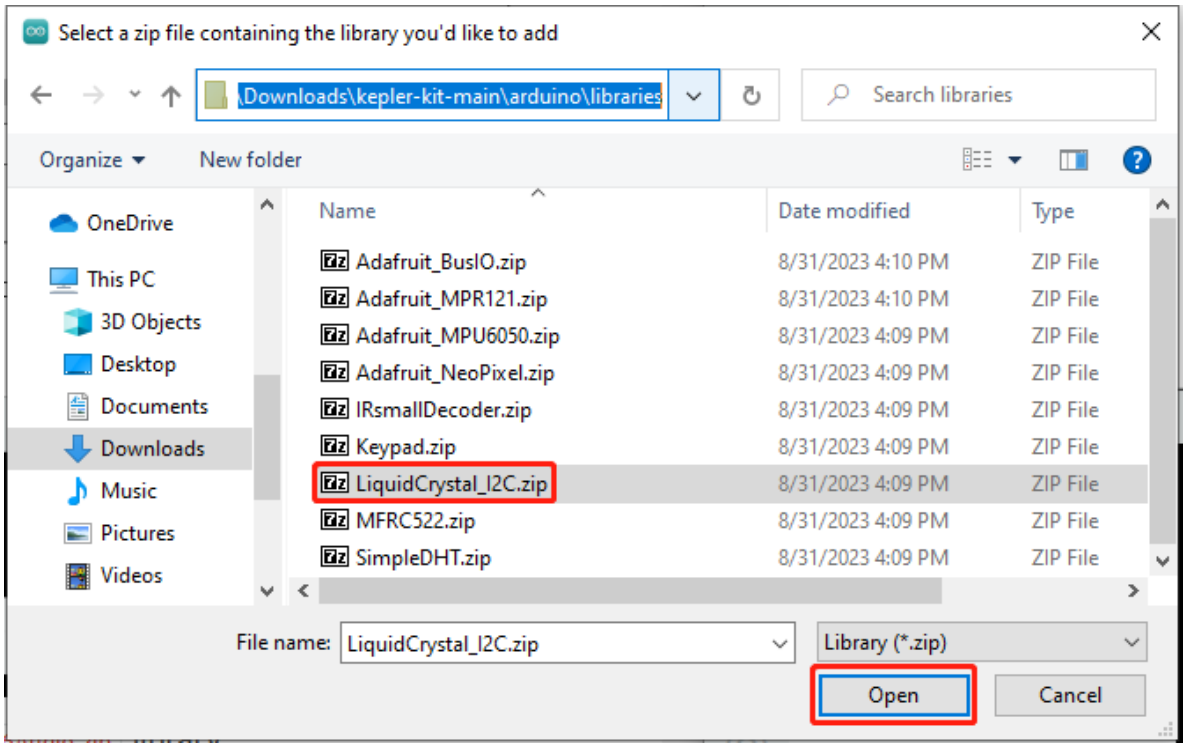
ライブラリは、いくつかの関数定義やヘッダーファイルを集めたもので、通常は .h（ヘッダーファイル、関数の宣言、マクロの定義、コンストラクタの定義など）と .cpp（実行ファイル、関数の実装、変数の定義など）の 2 つのファイルで構成されています。何らかのライブラリの関数を使用する必要がある場合は、ヘッダーファイル（例：#include <dht.h>）を追加し、その関数を呼び出すだけです。これにより、コードが簡潔になります。ライブラリを使用したくない場合は、その関数定義を直接書くこともできます。ただし、その結果としてコードが長くなり、読みにくくなる可能性があります。

Arduino IDE には既にいくつかのライブラリが組み込まれていますが、追加する必要があるものもあります。それでは、ライブラリを追加する方法を見ていきましょう。

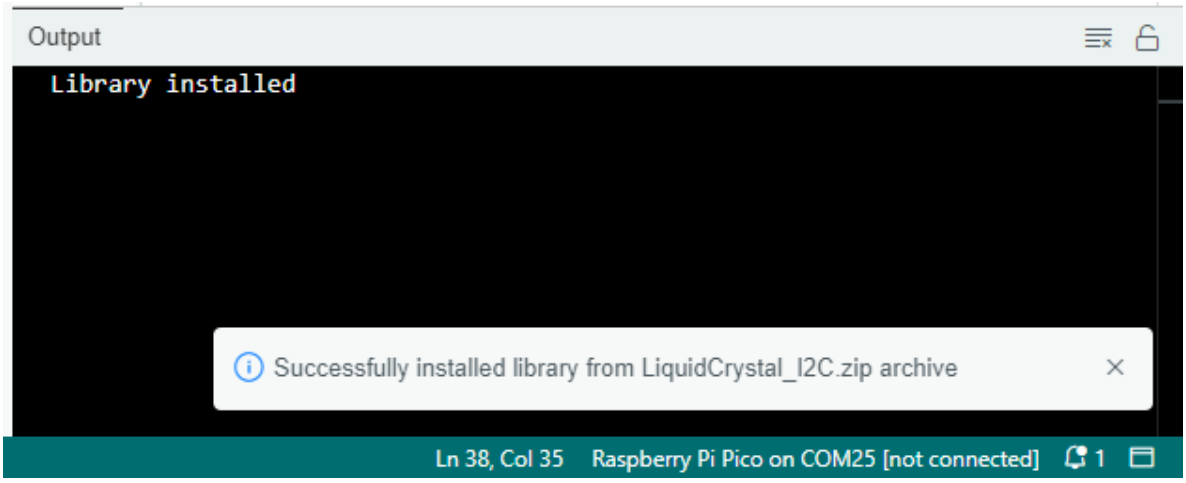
1. Arduino IDE を開き、 **スケッチ** -> **ライブラリをインクルード** -> **.ZIP ライブラリを追加** へ進んでください。



2. ライブラリファイルが保存されているディレクトリ（例： `kepler-kit-main\arduino\libraries` フォルダ）へ移動し、必要なライブラリファイル（例： `LiquidCrystal_I2C.zip` ）を選択して、 **開く** をクリックしてください。



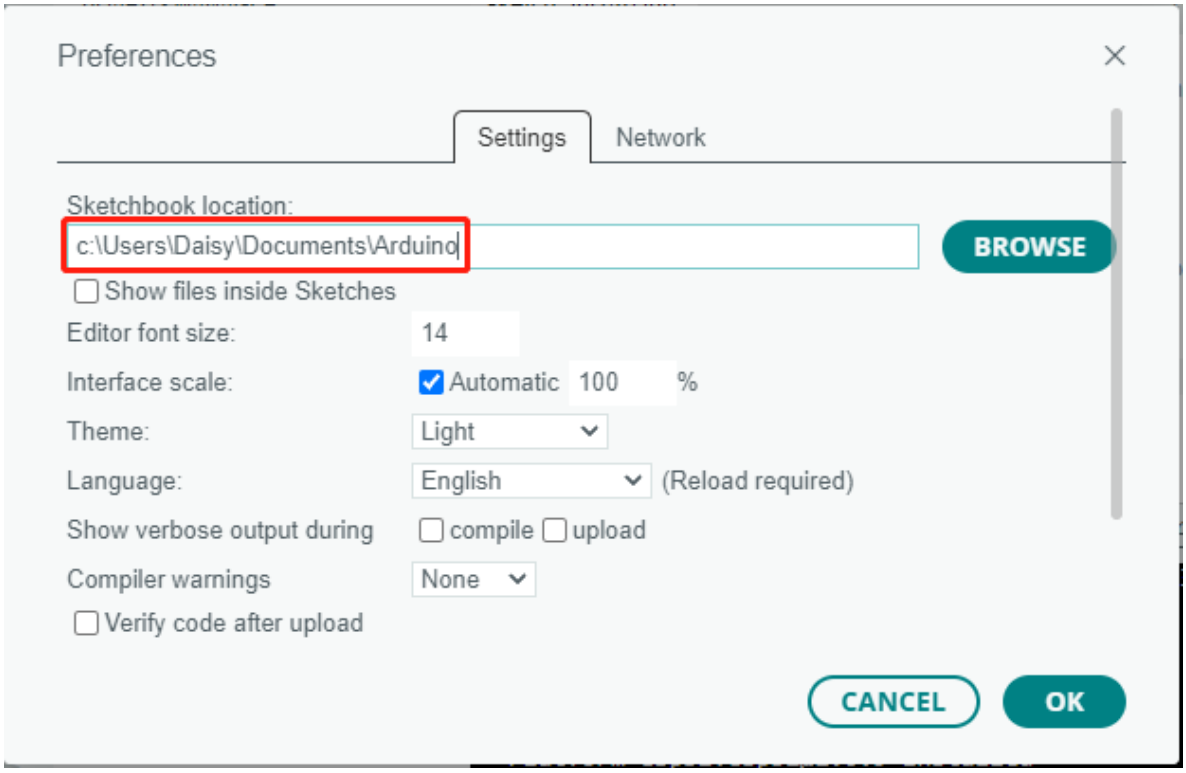
3. しばらくすると、インストールが成功したことを示す通知が表示されます。



4. 他のライブラリも同様の手順で追加してください。

注釈: インストールされたライブラリは Arduino IDE のデフォルトのライブラリディレクトリ、通常は `C:\Users\xxx\Documents\Arduino\libraries` に保存されます。

ライブラリディレクトリが異なる場合は、 **ファイル** -> **環境設定** で確認できます。



---

2. 出力 & 入力

6.5 2.1 - こんにちは、LED！

「Hello, world!」を出力することがプログラミング学習の第一歩であるように、LED を制御するプログラムを使用することは、物理的なプログラミングを学ぶ際の伝統的な入門です。

- *LED*

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

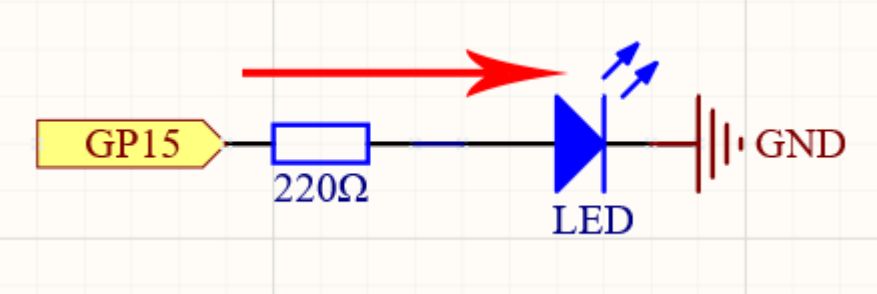
全体のキットを購入するのが便利です、以下がリンクです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

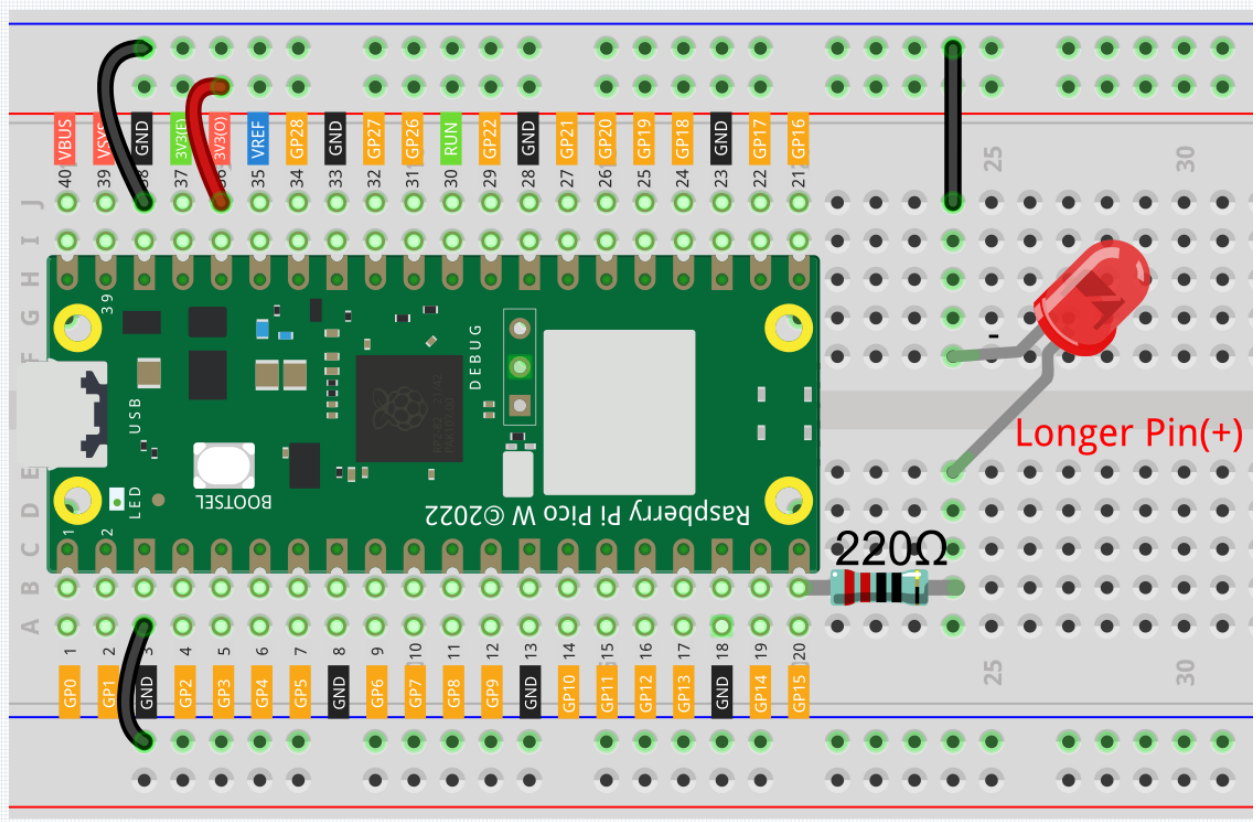
SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	

回路図



この回路の原理はシンプルであり、図には電流の方向が示されています。GP15 が高レベル（3.3V）を出力すると、220 Ωの電流制限抵抗を経て LED が点灯します。GP15 が低レベル（0V）を出力すると、LED は消灯します。

配線



電流の方向に沿って回路を組み立てましょう！

1. ここでは、Pico W ボードの GP15 ピンからの電気信号を使用して LED を動作させ、この回路はここから始まります。
2. 電流は 220 Ωの抵抗器（LED を保護するために使用）を通過する必要があります。抵抗器の一方の端（どちらでもよい）を Pico W GP15 ピンと同じ行（私の回路では行 20）に挿入し、もう一方の端をブレッドボードの空いている行（私の回路では行 24）に挿入します。
3. LED を取り上げると、一方のリードがもう一方よりも長いことに気づくでしょう。抵抗器の端と同じ行に長いリードを挿入し、短いリードをブレッドボードの中央のギャップを越えて同じ行に接続します。
4. 男性対男性（M2M）ジャンパーワイヤーを LED の短いピンと同じ行に挿入し、次にそれをブレッドボードの負の電源バスに接続します。
5. ジャンパーを使用して、負の電源バスを Pico W の GND ピンに接続します。

コード

注釈:

- ファイル 2.1_hello_led.ino は、パス kepler-kit-main/arduino/2.1_hello_led の下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

コードが実行された後、LED が点滅するのが見えます。

仕組みは？

ここでは、LED をデジタルピン 15 に接続していますので、プログラムの最初で「ledPin」という名前の int 変数を宣言し、その値として 15 を割り当てる必要があります。

```
const int ledPin = 15;
```

次に、 setup() 関数内でピンを初期化します。ここでは、ピンを OUTPUT モードに初期化する必要があります。

```
void setup() {
    pinMode(ledPin, OUTPUT);
}
```

loop() 内で、 digitalWrite() を使用して ledPin に 3.3V の高レベル信号を供給します。これにより、LED のピン間に電圧差が生じ、LED が点灯します。

```
digitalWrite(ledPin, HIGH);
```

レベル信号が LOW に変更されると、ledPin の信号は 0V に戻り、LED が消灯します。

```
digitalWrite(ledPin, LOW);
```

点灯と消灯の間には間隔が必要ですので、 delay(1000) コードを使用して、コントローラーが 1000ms 何もしないようにします。

```
delay(1000);
```

6.6 2.2 - レベル表示

最初のプロジェクトは LED を点滅させる単純なものです。このプロジェクトでは、LED バーグラフを使用しましょう。これは、10 個の LED がプラスチックケースにパッケージされており、一般的には電力や音量レベルを表示するために使用されます。

- *LED* **バーグラフ**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

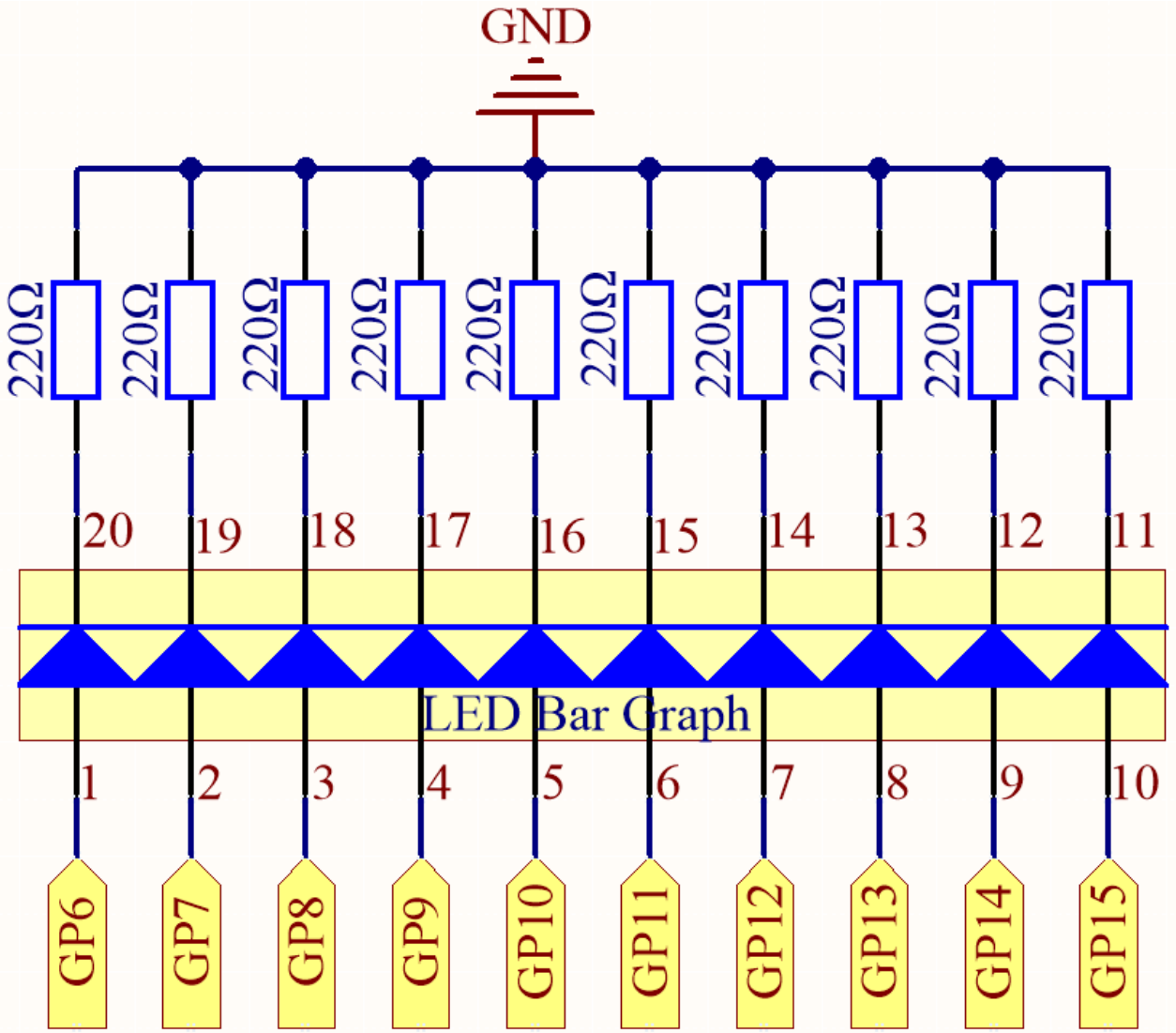
全体のキットを購入するのが便利です。以下がリンクです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

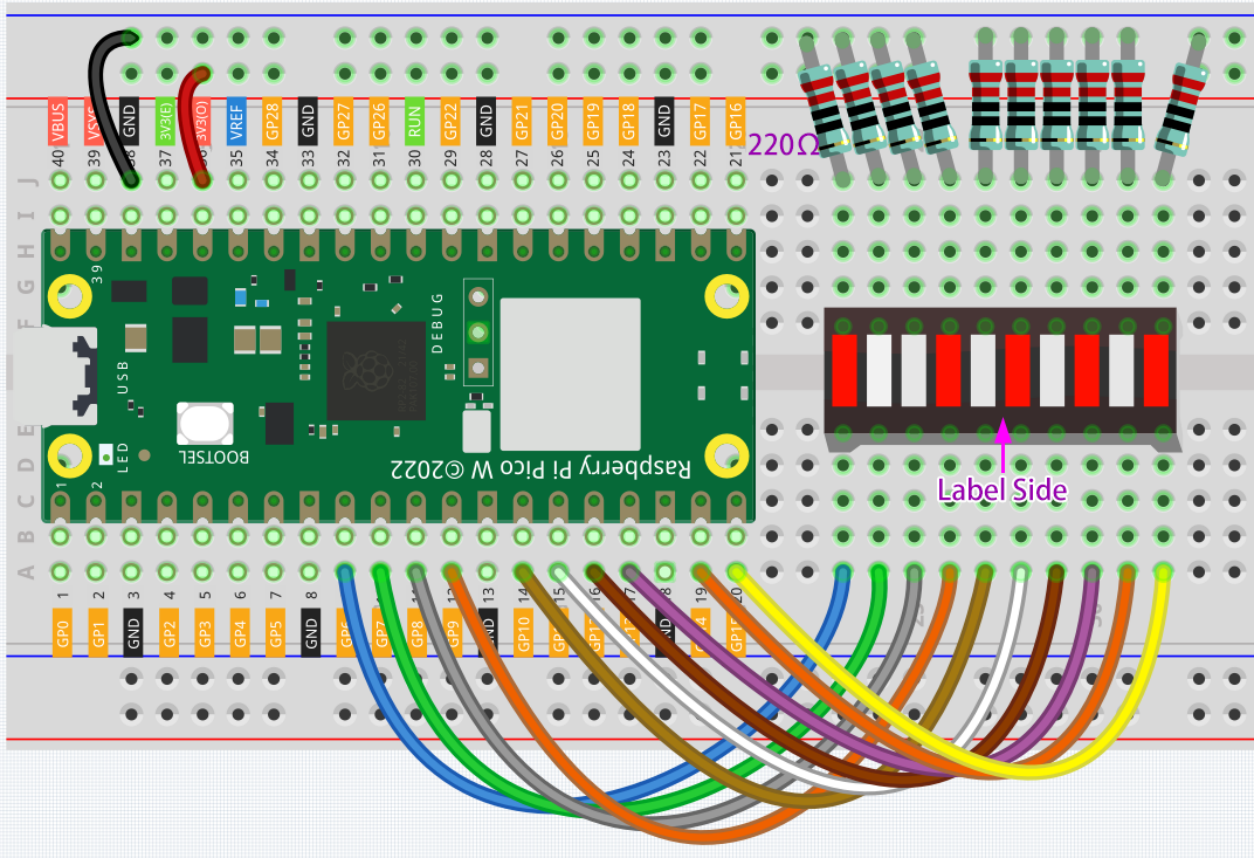
SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	10(220 Ω)	
6	*LED* **バーグラフ**	1	

回路図



LED バーグラフには 10 個の LED があり、それぞれが個別に制御可能です。ここでは、10 個の LED のアノードは GP6〜GP15 に接続され、カソードは 220 Ωの抵抗器を介して GND に接続されています。

配線



コード

注釈:

- ファイル `2.2_display_the_level.ino` は `kepler-kit-main/arduino/2.2_display_the_level` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。

プログラムが動作すると、LED バーグラフの LED が順番に点灯し、次に消灯します。

動作原理

LED バーの各 LED はピンで制御する必要があります。つまり、これらの 10 個のピンを定義する必要があります。

`setup()` 内のコードは for ループを使用して、順番にピン 6〜15 を出力モードに初期化します。

```
for(int i=6;i<=15;i++)
{
    pinMode(i,OUTPUT);
}
```

loop() 内で for ループを使用して、LED を順番に点滅させます（0.5 秒点灯、次に 0.5 秒消灯）。

```
for(int i=6;i<=15;i++)
{
    digitalWrite(i,HIGH);
    delay(500);
    digitalWrite(i,LOW);
    delay(500);
}
```

6.7 2.3 - Fading LED

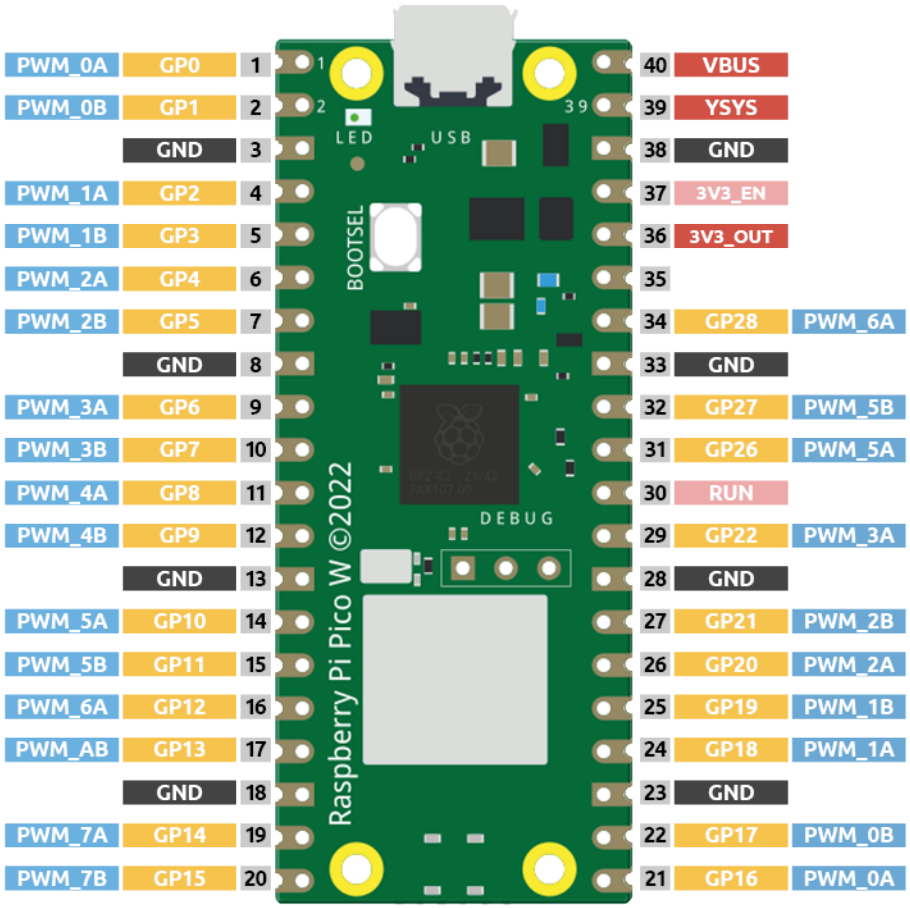
これまで、高レベルと低レベル（または 1 と 0、ON と OFF とも呼ばれる）の 2 つの出力信号のみを使用しており、これをデジタル出力と呼びます。しかし、実際の使用において、多くのデバイスは単純に ON/OFF して動作するわけではありません。例えば、モーターの速度調整やデスクランプの明るさ調整などです。以前は、抵抗を調整できるスライダーがこの目的を達成するために使用されていましたが、これは常に信頼性が低く非効率的です。そのため、パルス幅変調（PWM）がこのような複雑な問題に対する実用的な解決策として登場しました。

高レベルと低レベルで構成されるデジタル出力をパルスと呼びます。これらのピンのパルス幅は、ON/OFF の速度を変更することで調整することができます。

簡単に言えば、短い期間（例えば 20ms、ほとんどの人々の視覚保持時間）で LED を点灯、消灯、再点灯させると、消灯したことに気づかないでしょうが、光の明るさはわずかに弱くなります。この期間中に LED が点灯している時間が多いほど、LED の明るさが高くなります。言い換えれば、サイクル内でパルスが広いほど、マイクロコントローラーによって出力される"電気信号強度"が大きくなります。これが PWM が LED の明るさ（またはモーターの速度）を制御する方法です。

- パルス幅変調 - Wikipedia

Pico W が PWM を使用する際に注意すべきいくつかの点があります。この画像を見てみましょう。



Pico W の各 GPIO ピンは PWM をサポートしていますが、実際には合計 16 個の独立した PWM 出力（30 個ではない）があり、左側の GP0 から GP15 までに分散されています。右側の GPIO の PWM 出力は左側のコピーと同等です。

注意すべき点は、プログラミング中に異なる目的で同じ PWM チャネルを設定しないようにすることです。（例えば、GP0 と GP16 は両方とも PWM_0A です）

この知識を理解した後、Fading LED の効果を実現してみましょう。

- *LED*

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

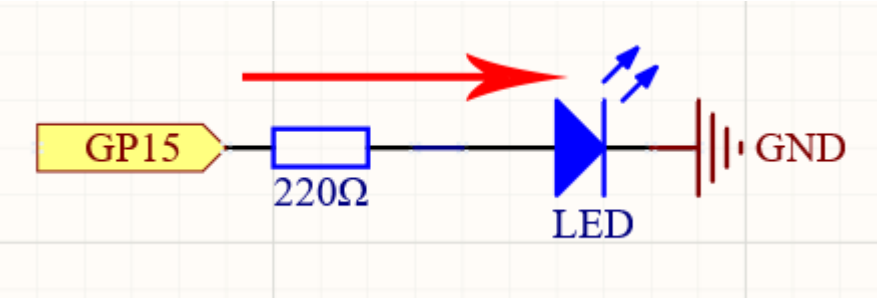
全体のキットを購入する方が確実に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

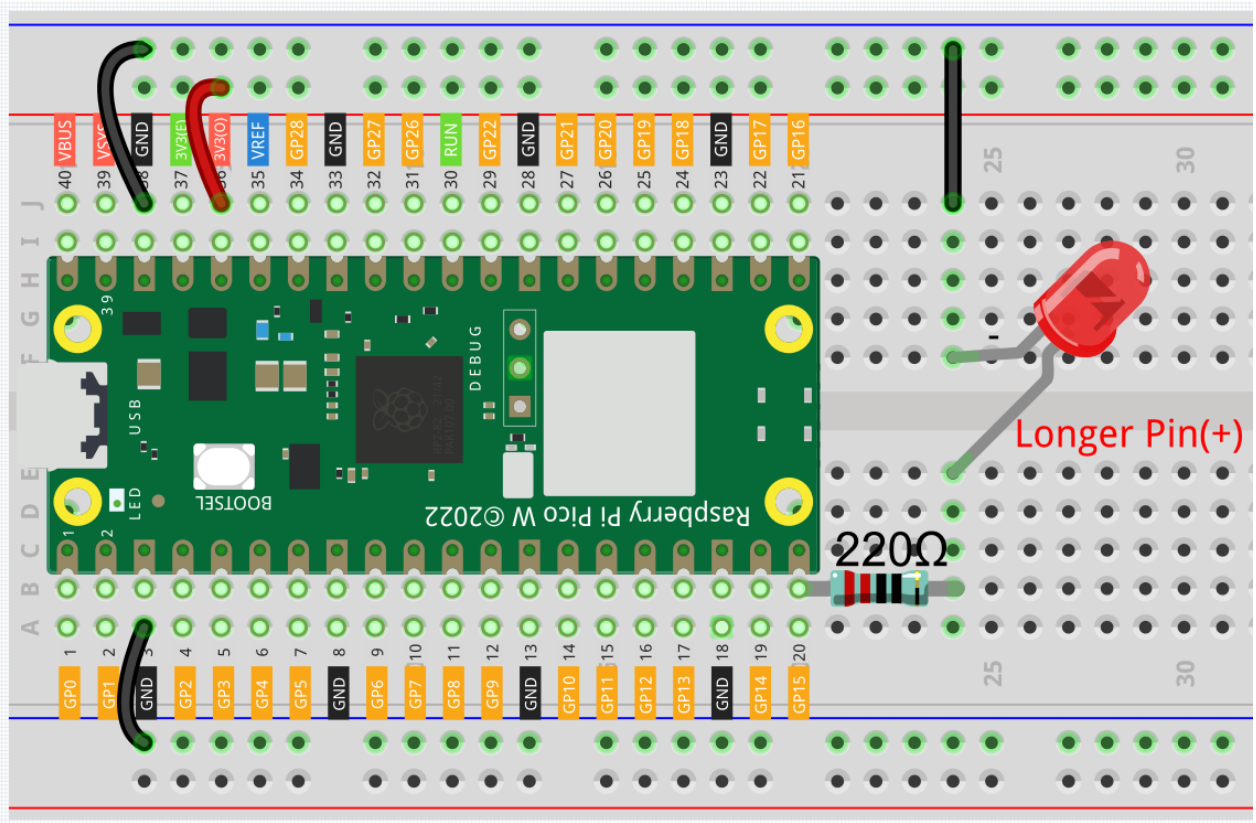
SN	コンポーネントの紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	

回路図



このプロジェクトは最初のプロジェクト *2.1 -* **こんにちは、***LED***！** と同じ回路ですが、信号タイプが異なります。最初のプロジェクトでは、GP15 から直接デジタルの高レベルと低レベル（0&1）を出力して LED を点灯または消灯させるのに対し、このプロジェクトでは GP15 から PWM 信号を出力して LED の明るさを制御します。

配線



コード

注釈:

- ファイル `2.3_fading_led.ino` は、パス `kepler-kit-main/arduino/2.3_fading_led` の下で開くことができます。
- またはこのコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択することを忘れないでください。

プログラムが実行されるにつれて、LED は徐々に明るくなります。

動作原理

ピン 15 を ledPin として宣言します。

```
const int ledPin = 15;
```

`loop()` 内の `analogWrite()` は、ledPin に 0 から 255 までのアナログ値（PWM 波）を割り当てて LED の明るさを変更します。

```
analogWrite(ledPin, value);
```

for ループを使用して、 `analogWrite()` の値を最小値（0）と最大値（255）の間で段階的に変更することができます。

```
for (int value = 0 ; value <= 255; value += 5) {
    analogWrite(ledPin, value);
}
```

実験現象を明確に観察するために、for サイクルに `delay(30)` を追加して、明るさの変更時間を制御する必要があります。

```
for (int value = 0 ; value <= 255; value += 5) {
    analogWrite(ledPin, value);
    delay(30);
}
```

6.8 2.4 - カラフルな光

私たちは知っているとおり、光は重ね合わせることができます。例えば、青い光と緑の光を混ぜるとシアン色の光ができ、赤い光と緑の光を混ぜると黄色の光ができます。このことを「加法的色の混合」と呼びます。

- 加法的色 - ウィキペディア

この方法に基づいて、異なる比重で三原色を混合することで、任意の色の可視光を作成することができます。例えば、赤を多く、緑を少なくすることでオレンジ色を作ることができます。

この章では、RGB LED を使用して加法的色の混合の神秘を探究します！

RGB LED は、赤、緑、青の LED を一つのランプキャップの下に封入し、三つの LED は共通のカソードピンを共有しています。各アノードピンに電気信号が供給されるため、対応する色の光が表示されます。各アノードの電気信号の強度を変更することで、さまざまな色を生成することができます。

- *RGB LED*

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

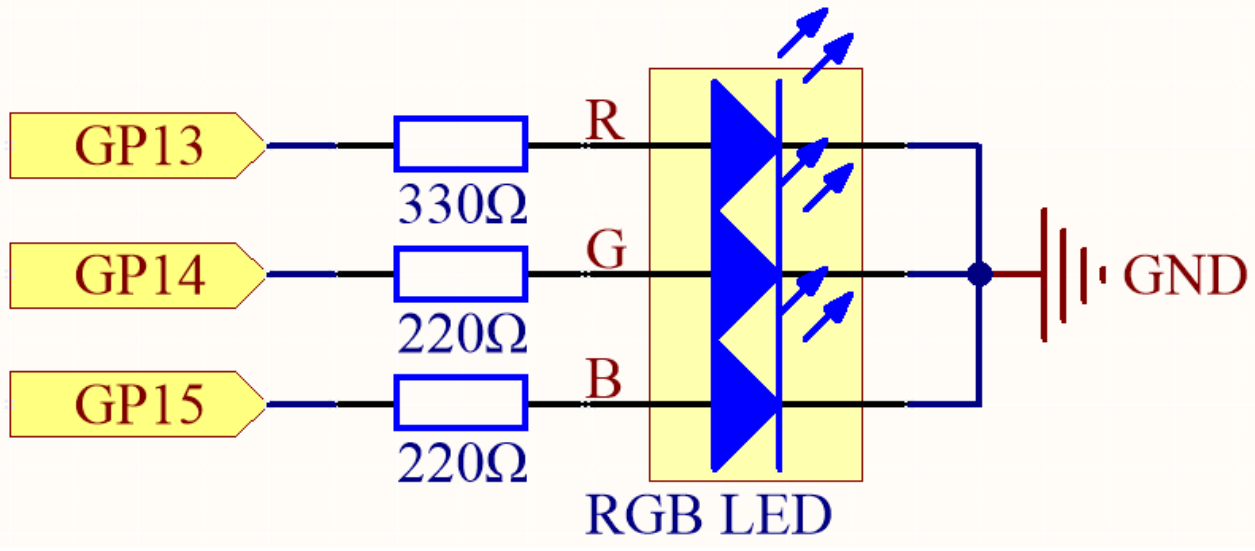
全体のキットを購入すると非常に便利です。リンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

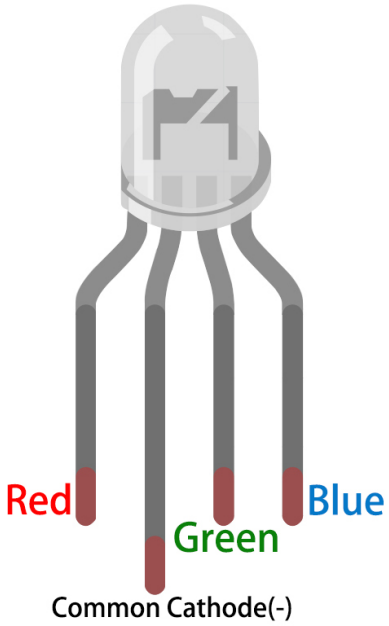
SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	3（1-330 Ω, 2-220 Ω）	
6	*RGB LED*	1	

回路図

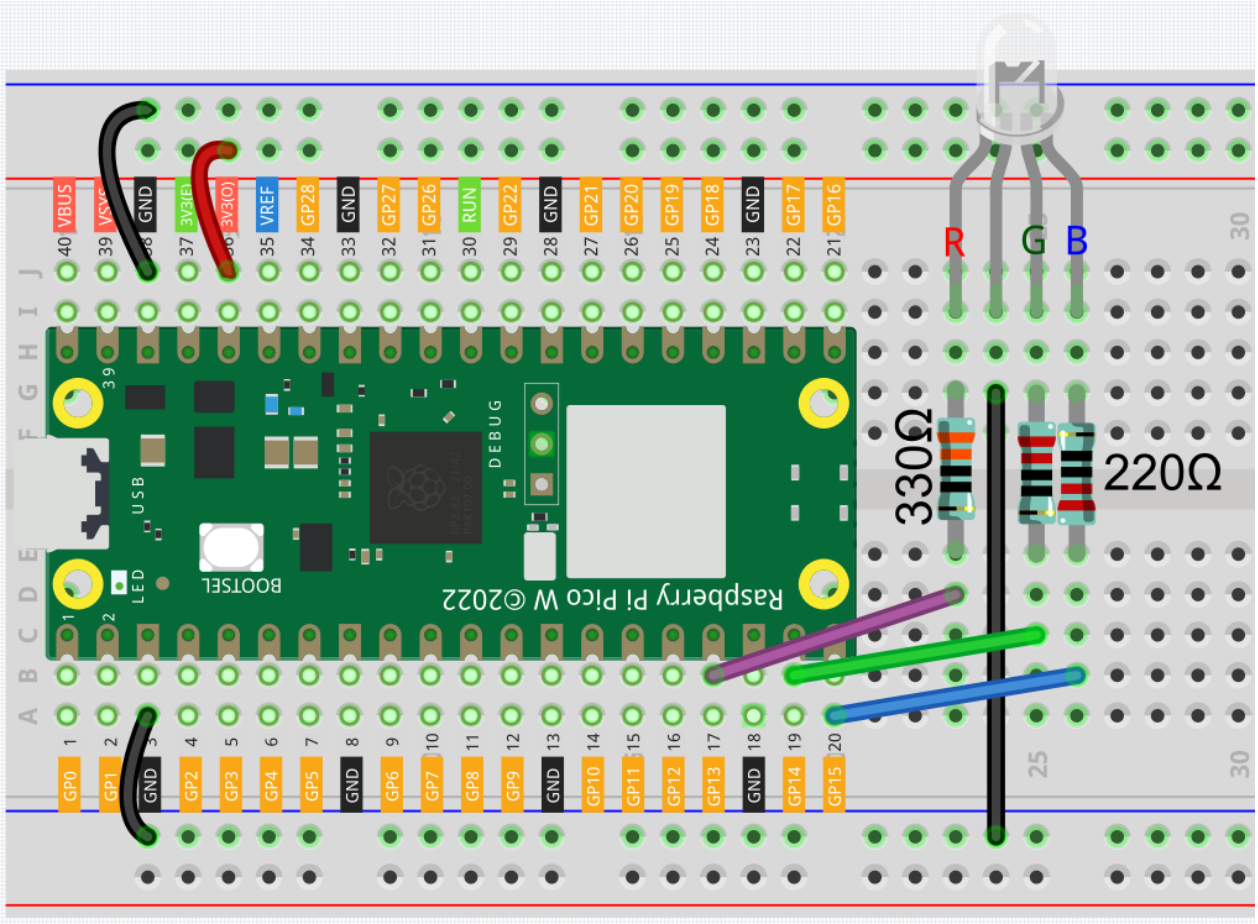


PWM ピン GP13、GP14、GP15 はそれぞれ RGB LED の赤、緑、青のピンを制御し、共通のカソードピンを GND に接続します。これにより、異なる PWM 値でこれらのピンに光を重ね合わせることで、RGB LED が特定の色を表示することができます。

配線



RGB LED には 4 本のピンがあります：最も長いピンが共通のカソードピンで、通常は GND に接続され、最も長いピンの隣の左側のピンが赤で、右側の 2 本のピンが緑と青です。

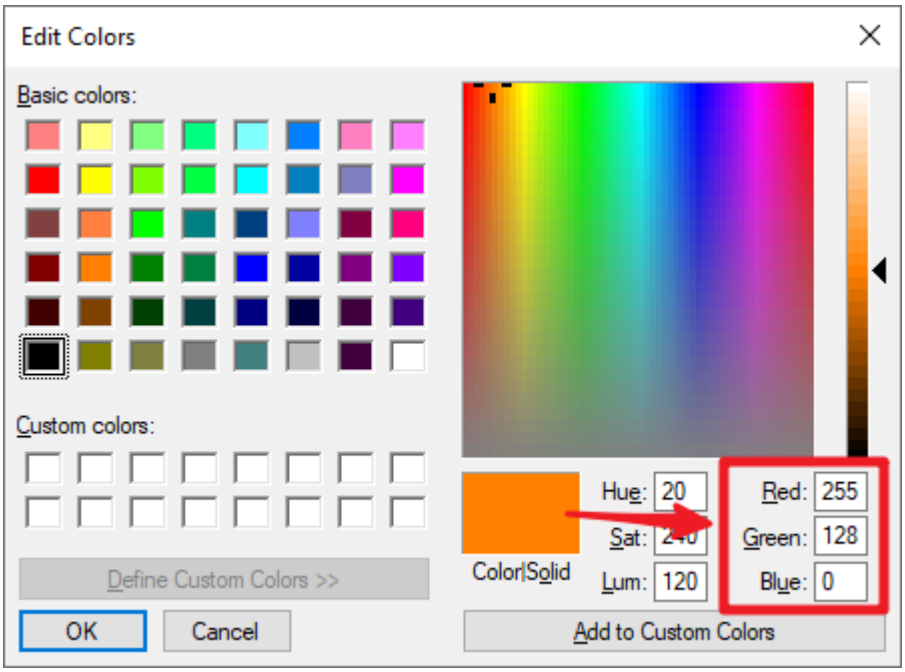


コード

ここで、お気に入りの色を描画ソフトウェア（例：ペイント）で選択し、RGB LED で表示させることができます。

注釈:

- ファイル `2.4_colorful_light.ino` は、パス `kepler-kit-main/arduino/2.4_colorful_light` で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。



color_set() に RGB 値を入力すると、RGB が希望する色に光るようになります。

どうやって動くのか？

この例では、RGB の三つのピンに値を割り当てるために使用される関数は、独立したサブ関数 color() にパッケージされています。

```
void color (unsigned char red, unsigned char green, unsigned char blue)
{
    analogWrite(redPin, red);
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
}
```

loop() 内では、RGB 値が入力引数として color() 関数を呼び出すことで、RGB が異なる色を発するようになります。

```
void loop()
{
    color(255, 0, 0); //  red
    delay(1000);
    color(0,255, 0); //  green
    delay(1000);
    color(0, 0, 255); //  blue
    delay(1000);
```

(次のページに続く)

(前のページからの続き)

```
}
```

6.9 2.5 - ボタン値の読み取り

GPIO（General-purpose input/output、汎用入出力）という名前からわかるように、これらのピンには入力と出力の両方の機能があります。前のレッスンでは出力機能を使用しましたが、この章では入力機能を使用してボタンの値を読み取ります。

- **ボタン**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

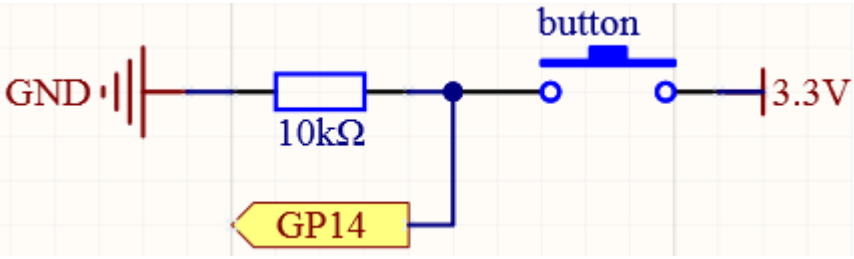
全体のキットを購入するのが便利です、リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個々にも購入できます。

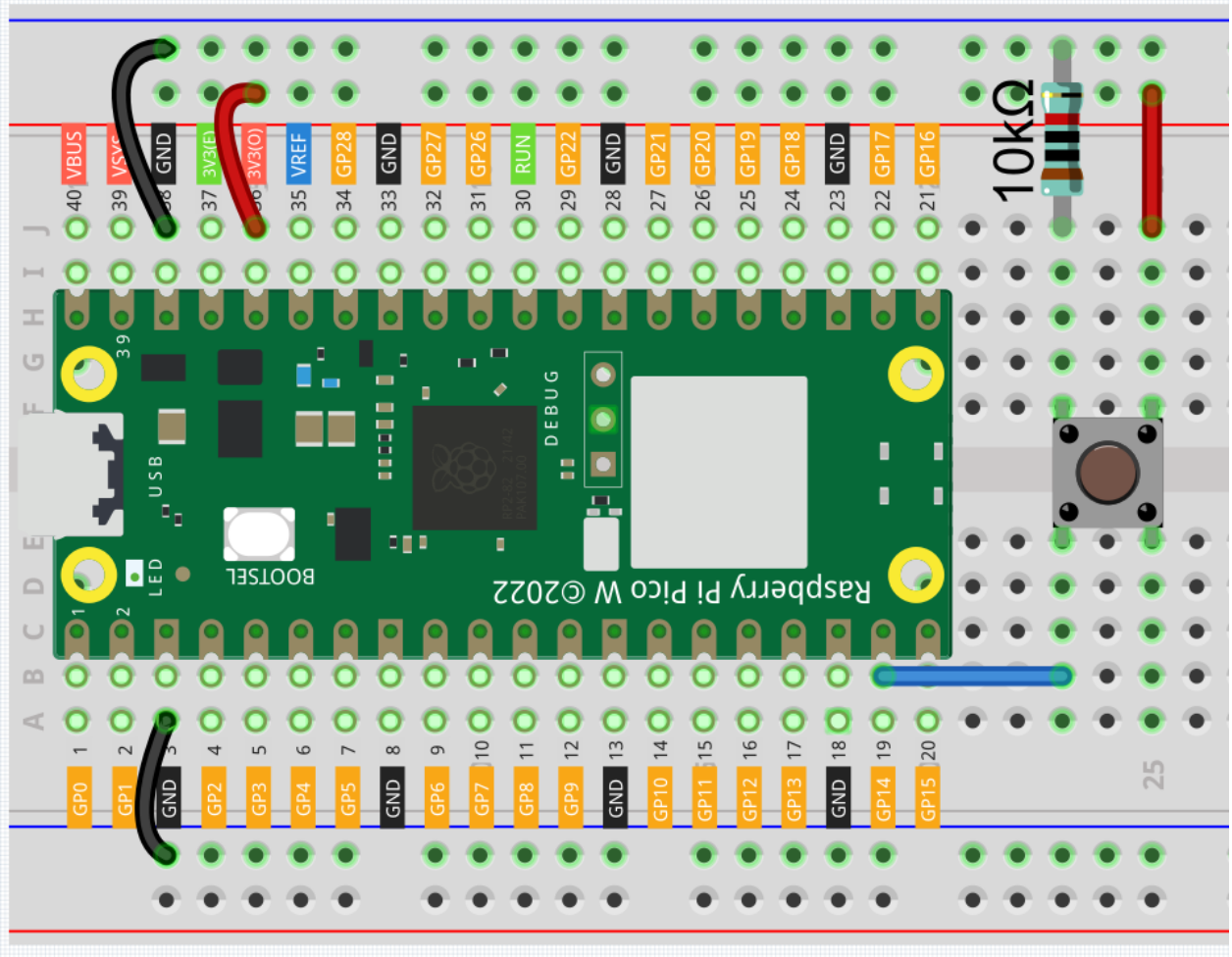
SN	コンポーネントの紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10KΩ)	
6	**ボタン**	1	

回路図



ボタンのピンの一方は 3.3v に接続され、もう一方のピンは GP14 に接続されているため、ボタンが押されると、GP14 は高くなります。しかし、ボタンが押されていないとき、GP14 は未定義の状態にあり、高いか低いかが不明です。ボタンが押されていないときに安定した低レベルを得るために、GP14 は 10K プルダウン抵抗を介して GND に再接続する必要があります。

配線



注釈: 4 本足のボタンは H 型のボタンと考えることができます。その左（右）の 2 本の足は接続されており、中央の仕切り線をまたぐと、同じ行番号の 2 つの半行が接続されます（例えば、私の回路では、E23 と F23 が接続されています。E25 と F25 も同様です）。

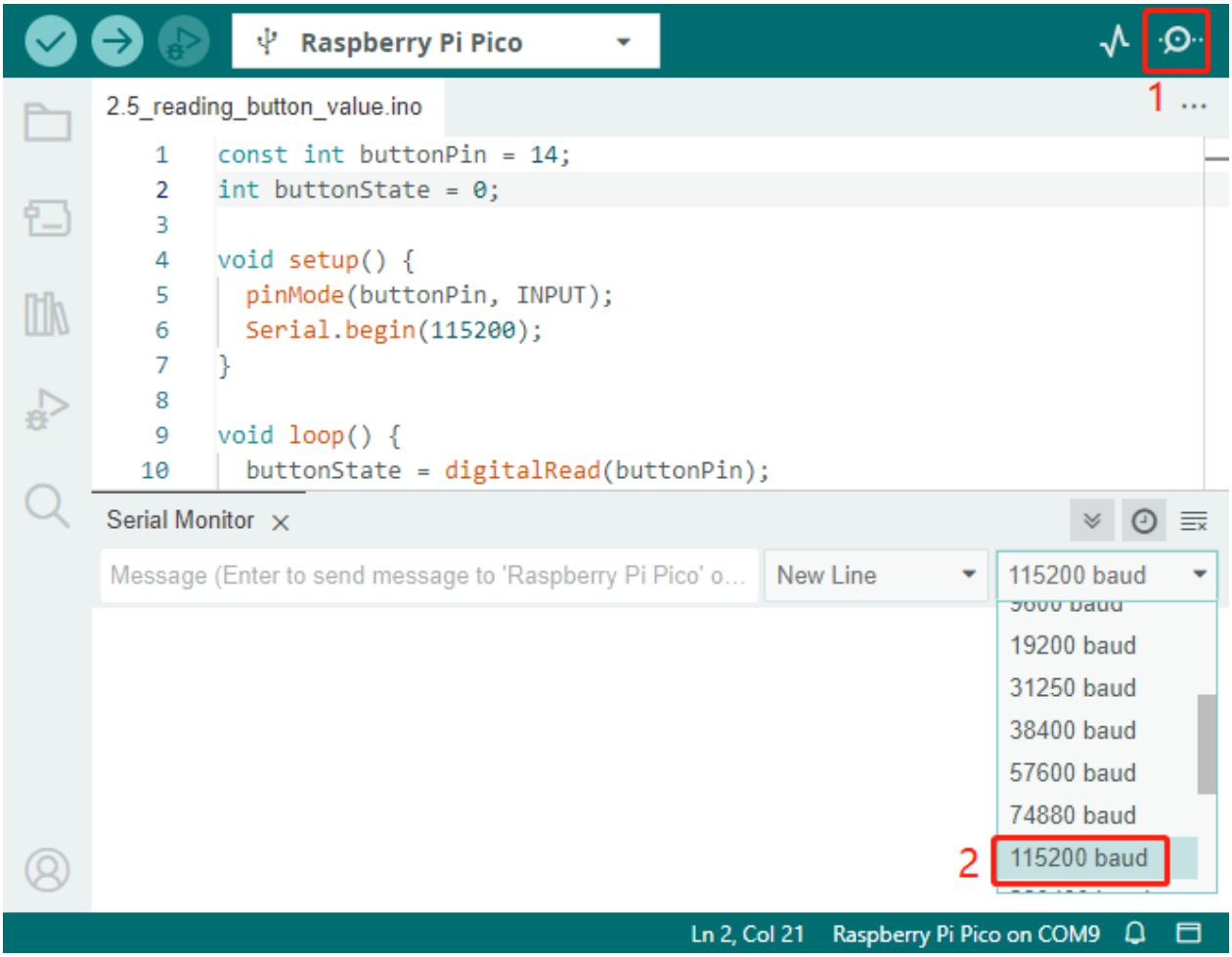
ボタンが押される前は、左右は互いに独立しており、一方から他方への電流は流れません。

コード

注釈:

- ファイル 2.5_reading_button_value.ino は、kepler-kit-main/arduino/2.5_reading_button_value のパスにあります。
- または、このコードを **Arduino IDE** にコピーしてください。
- アップロードボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

コードが実行された後、Arduino IDE の右上角にある虫眼鏡アイコン（シリアルモニタ）をクリックしてください。



これで、ボタンを押すと、シリアルモニタに「You pressed the button!」と表示されます。

動作原理は？

シリアルモニタを有効にするには、 `setup()` でシリアル通信を開始し、データレートを 9600 に設定する必要があります。

```
Serial.begin(115200);
```

- Serial

ボタンには、その値を取得できるようにモードを `INPUT` に設定する必要があります。

```
pinMode(buttonPin, INPUT);
```

`buttonPin` の状態を `loop()` で読み取り、変数 `buttonState` に割り当てます。

```
buttonState = digitalRead(buttonPin);
```

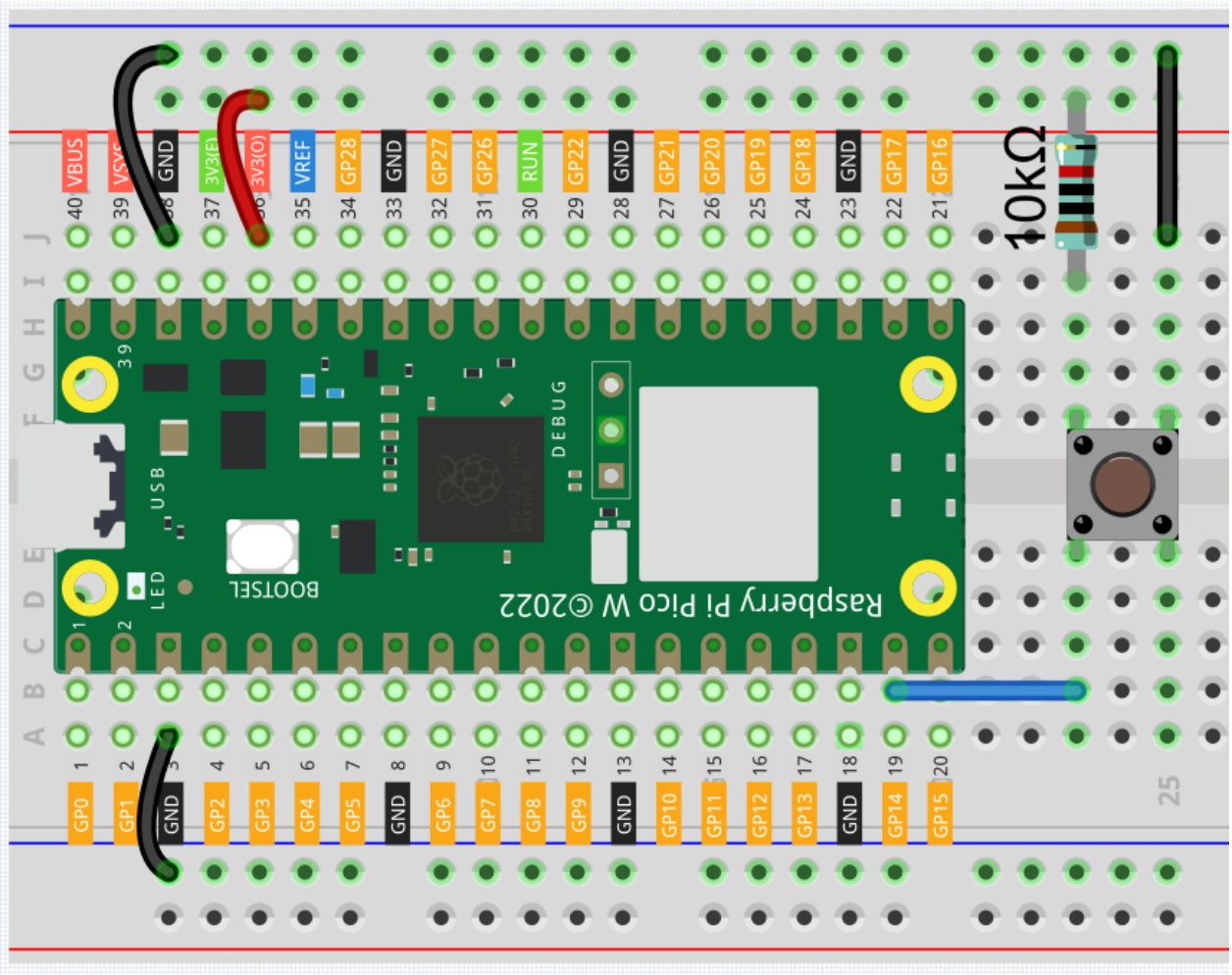
- digitalRead()

`buttonState` が HIGH であれば、LED が点滅し、シリアルモニタに「You pressed the button!」と表示されます。

```
if (buttonState == HIGH) {
    Serial.println("You pressed the button!");
}
```

プルアップ動作モード

次に、ボタンがプルアップ動作モードでの配線とコードです、試してみてください。



プルダウンモードとの唯一の違いは、10K の抵抗器が 3.3V に接続され、ボタンが GND に接続されているため、ボタンを押すと GP14 は低レベルになることです。これは、プルダウンモードで得られる値とは逆です。したがって、このコードを `if (buttonState == LOW)` に変更するだけです。

6.10 2.6 - 傾けてみよう！

この傾斜スイッチは中央に金属ボールが入っている 2 ピンのデバイスです。このスイッチを垂直に置くと、2 つのピンが接続されます。スイッチを傾けると、2 つのピンが切断されます。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

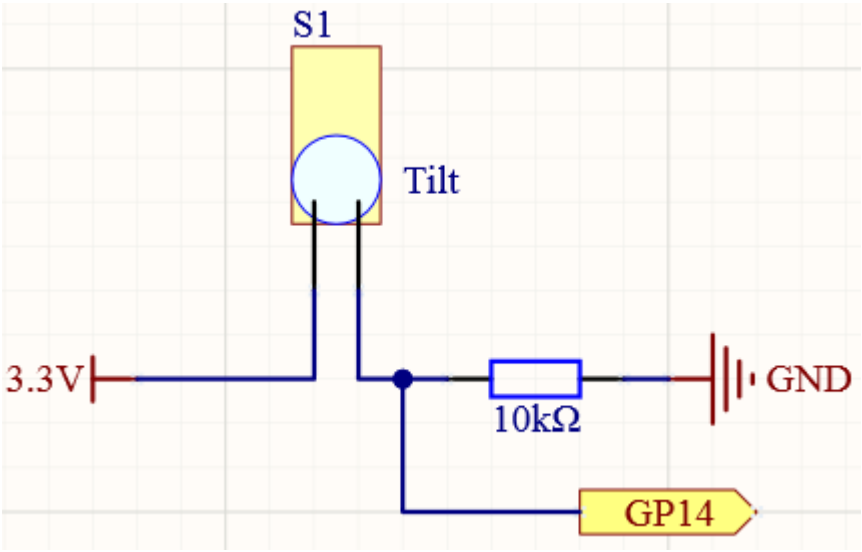
全体のキットを購入する方が間違いなく便利です。こちらがリンクです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1（10K Ω）	
6	**傾斜スイッチ**	1	

回路図

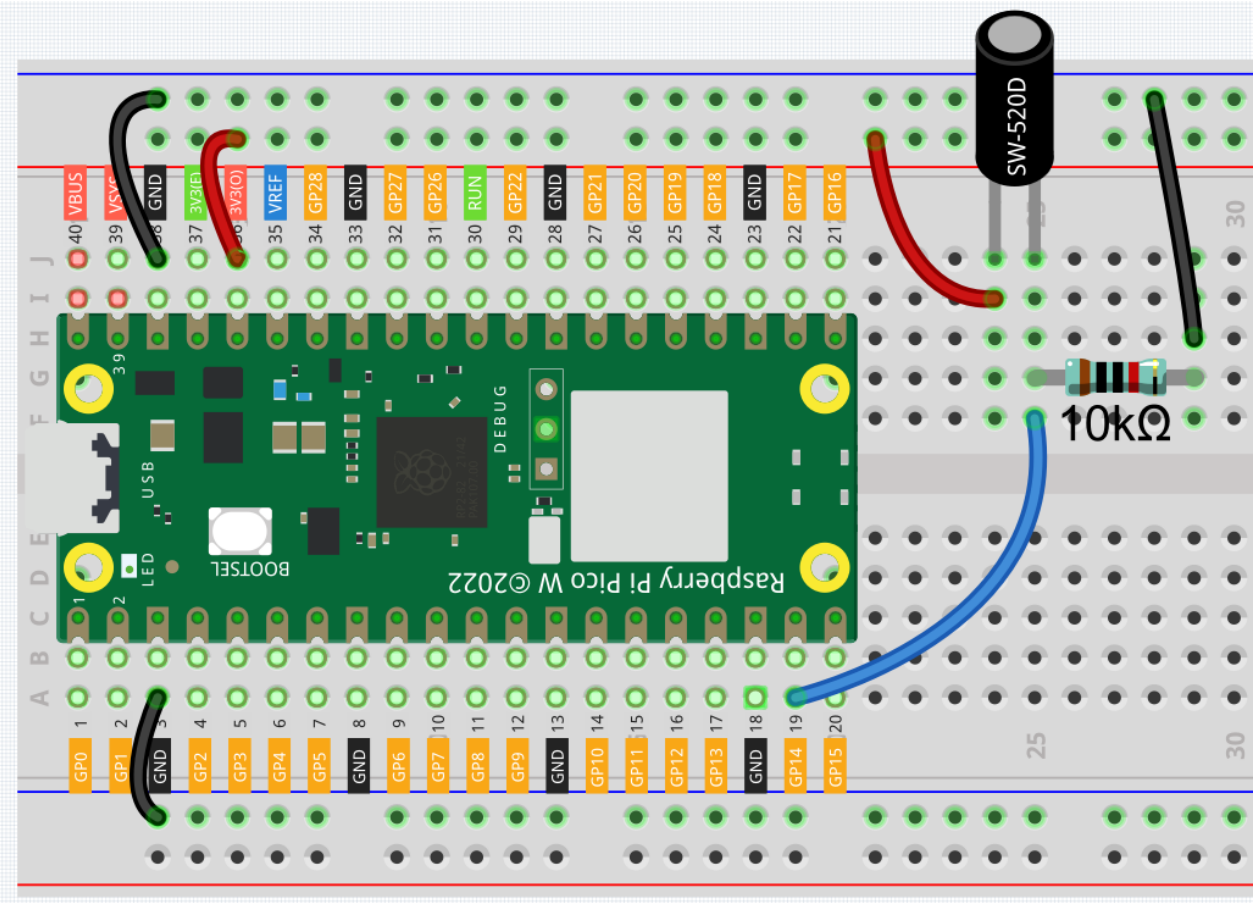


スイッチを垂直に置くと、GP14 はハイ状態になります。傾けた後、GP14 はロー状態になります。

10K Ωの抵抗の目的は、傾斜スイッチが傾いた状態のとき、GP14 を安定したロー状態に保つことです。

- **傾斜スイッチ**

配線



コード

注釈:

- ファイル `2.6_tilt_it.ino` は、パス `kepler-kit-main/arduino/2.4_colorful_light` の下にあります。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

プログラムが動作すると、ブレッドボード（傾斜スイッチ）を傾けると、シェルに「スイッチが作動します！」と表示されます。

6.11 2.7 - 左右トグルスイッチ

スライドスイッチは 3 ピンのデバイスで、ピン 2（中央）が共通ピンです。スイッチを左にトグルすると、左側の 2 ピンが接続され、右にトグルすると、右側の 2 ピンが接続されます。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

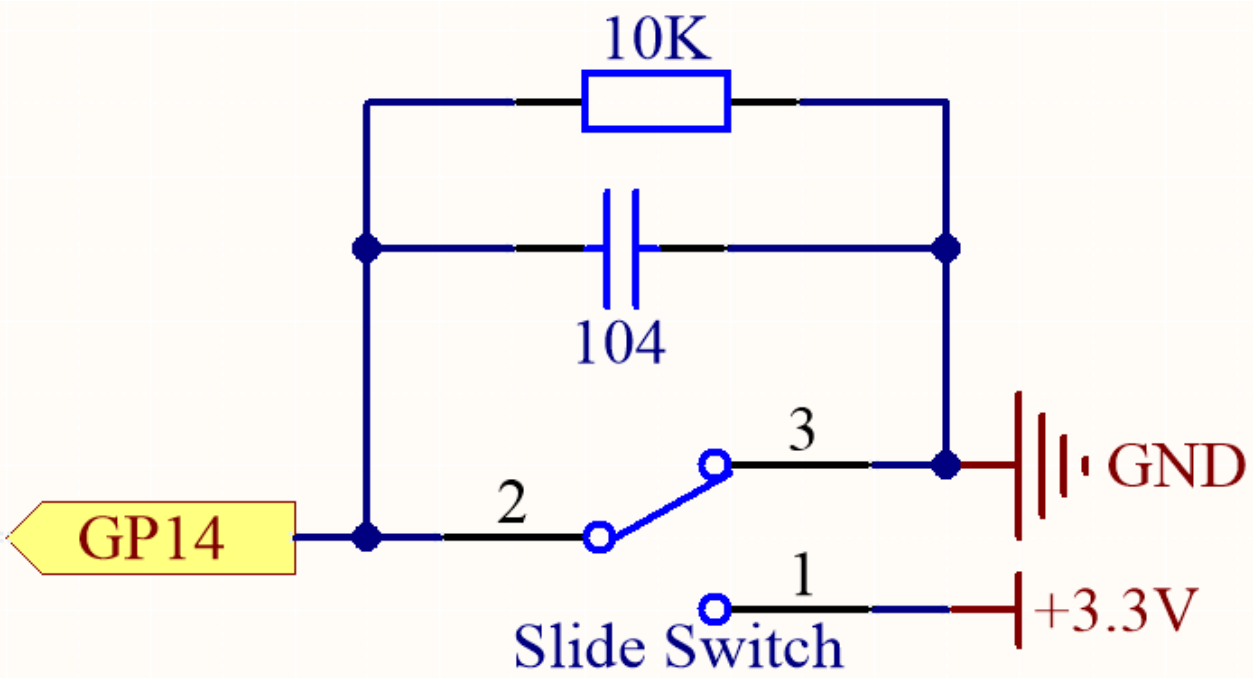
全体のキットを購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1（10K Ω）	
6	**コンデンサ**	1（104）	
7	**スライドスイッチ**	1	

回路図



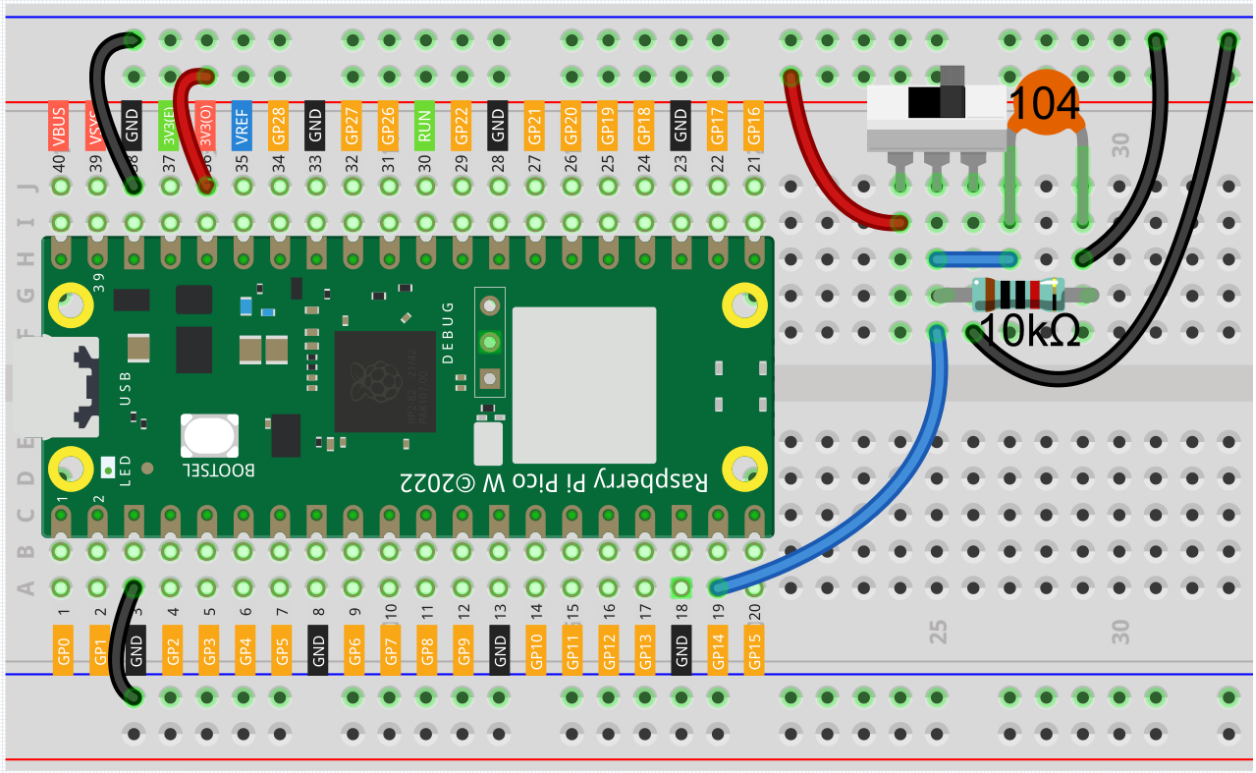
スライドスイッチを右か左にトグルすると、GP14 は異なるレベルになります。

10K Ωの抵抗器の目的は、トグル中（極端な左または右にトグルしていない状態で）GP14 を低く保つことです。

ここで使用される 104 セラミックコンデンサは、ジッタを排除するためです。

- **スライドスイッチ**
- **コンデンサ**

配線



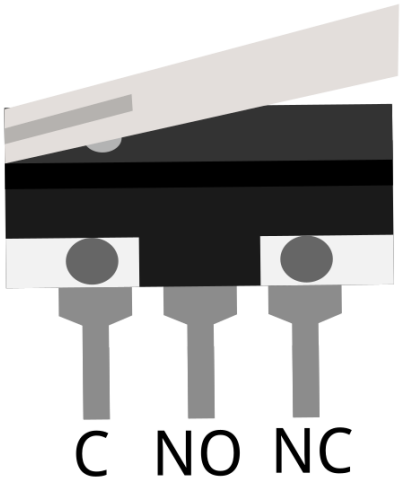
コード

注釈:

- ファイル `2.7_toggle_left_right.ino` は、パス `kepler-kit-main/arduino/2.7_toggle_left_right` で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。

プログラムが動作していると、シリアルモニターにはスイッチを左または右にトグルしたときに「ON」または「OFF」と表示されます。

6.12 2.8 - ソフトに押してください



マイクロスイッチは 3 ピンデバイスでもあり、3 つのピンの順番は C（共通ピン）、NO（通常開）および NC（通常閉）です。

マイクロスイッチが押されていないとき、1（C）と 3（NC）が接続されています。押されると、1（C）と 2（NO）が接続されます。

- **マイクロスイッチ**

必要な部品

このプロジェクトでは、以下のコンポーネントが必要です。

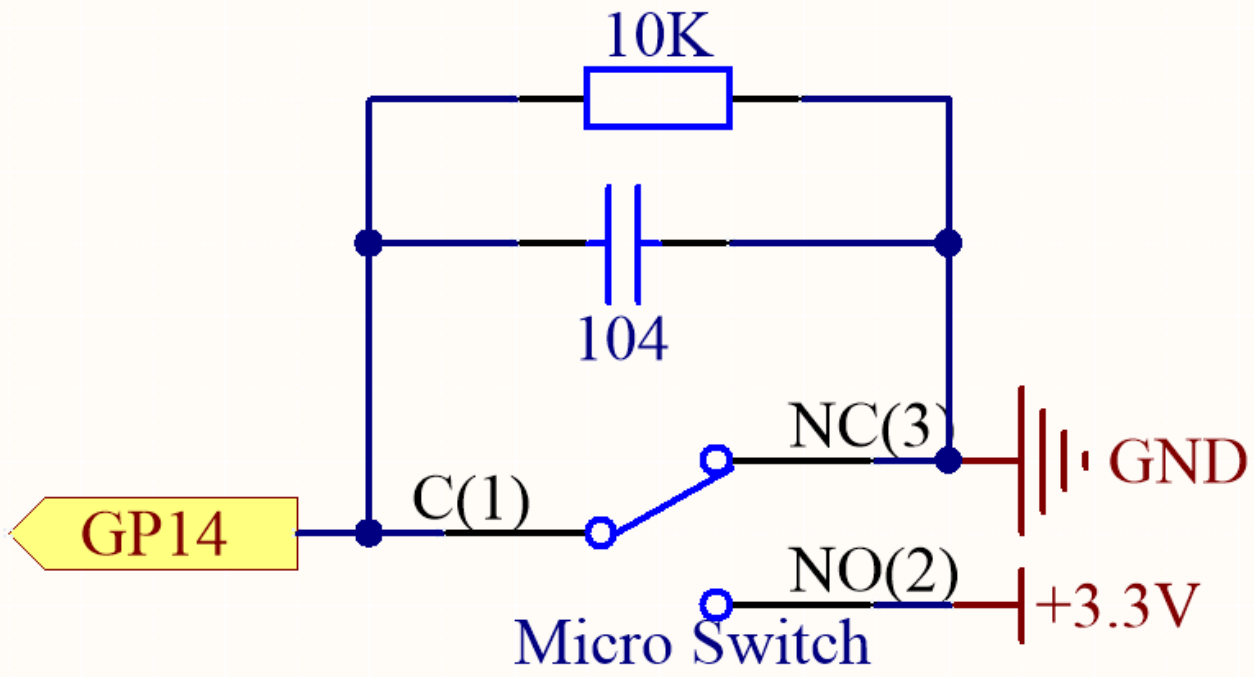
一式を購入することは非常に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450+	

以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**コンデンサ**	1(104)	
7	**マイクロスイッチ**	1	

回路図

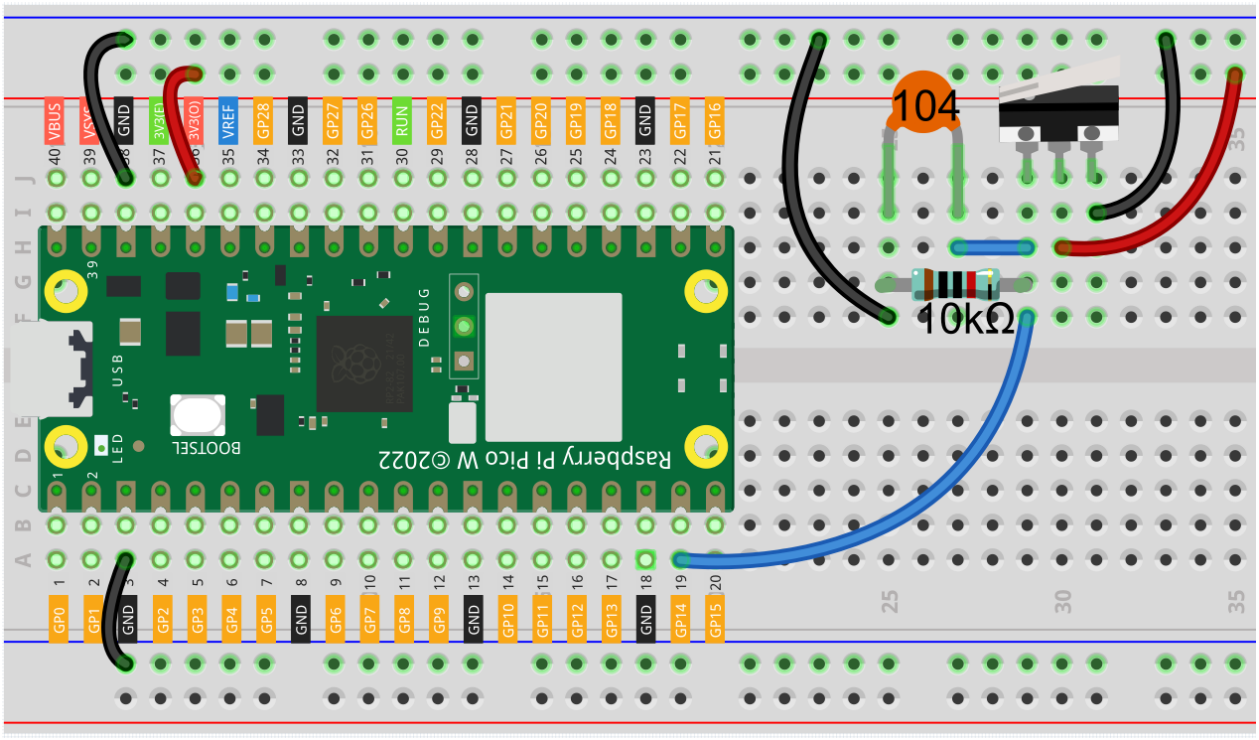


デフォルトでは、GP14 はロー状態ですが、押されると GP14 はハイ状態になります。

10K Ωの抵抗器の目的は、押下中に GP14 をロー状態に保つことです。

104 セラミックコンデンサは、ジッタを除去するためにここで使用されます。

配線



コード

注釈:

- ファイル `2.8_press_gently.ino` は、 `kepler-kit-main/arduino/2.8_press_gently` のパスの下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

プログラムが実行された後、スライドスイッチを右に切り替えると、「The switch works!」とシリアルモニターに表示されます。

6.13 2.9 - 磁気を感じる

最も一般的なタイプのリードスイッチには、スイッチが開いているときに小さなギャップで隔てられた、磁気化可能な柔軟な金属製のリードのペアが含まれています。

電磁石または永久磁石からの磁場がリードを引きつけることで、電気回路が完成します。磁場が消えると、リードのばね力によってそれらが離れ、回路が開きます。

リードスイッチの一般的な用途の一例は、セキュリティアラームのためのドアや窓の開閉を検出することです。

- **リードスイッチ**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

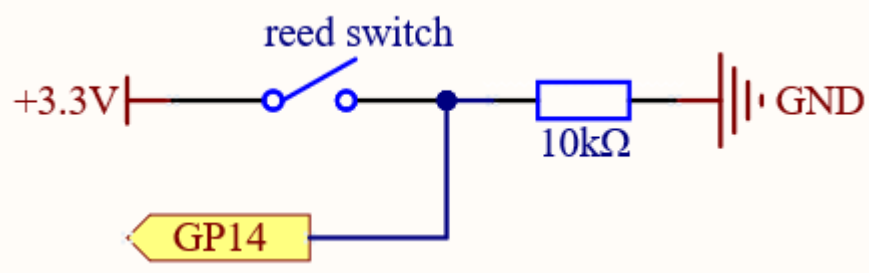
全体のキットを購入すると非常に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

番号	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1（10K Ω）	
6	**リードスイッチ**	1	

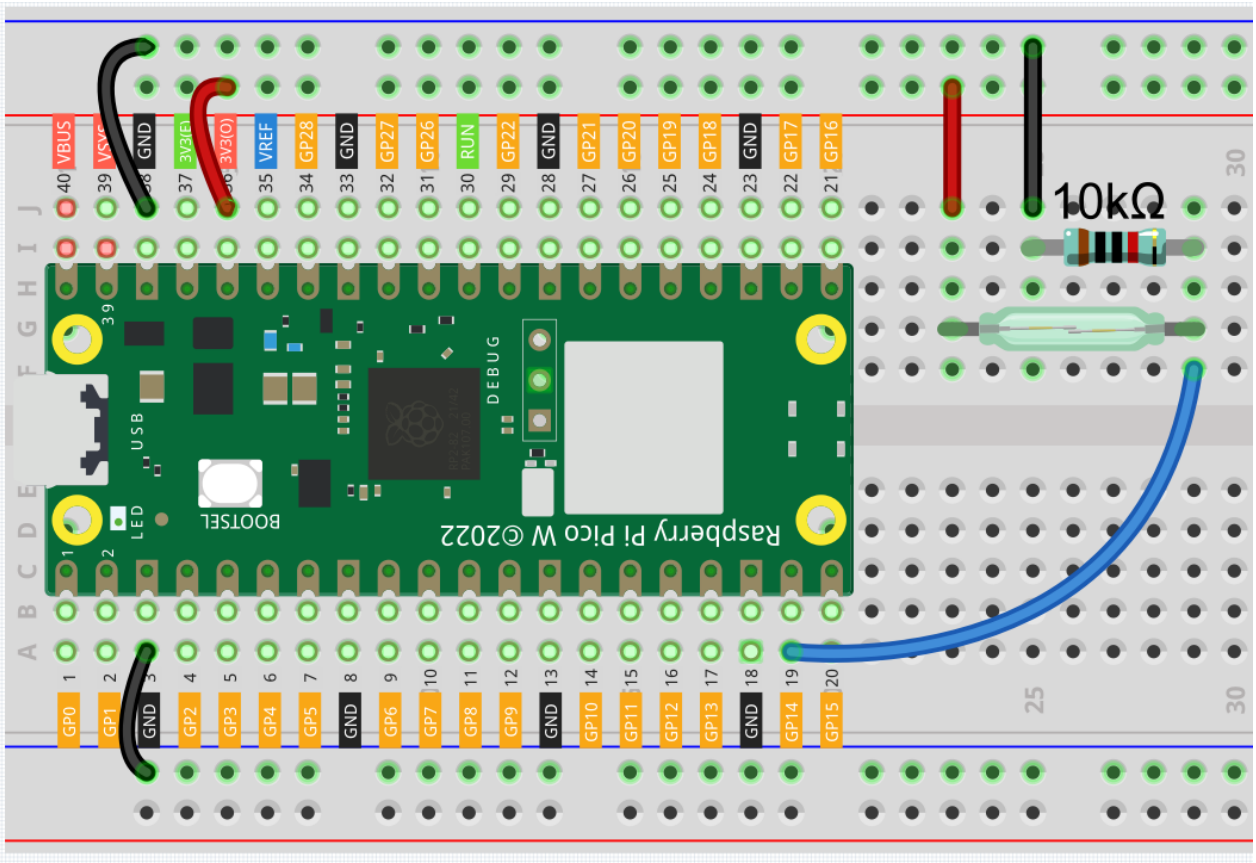
回路図



デフォルトでは、GP14 は低く、磁石がリードスイッチに近づくと高くなります。

10K Ωの抵抗器の目的は、磁石が近くにないときに GP14 を安定した低レベルに保つことです。

配線



コード

注釈:

- ファイル 2.9_feel_the_magnetism.ino は、パス kepler-kit-main/arduino/2.9_feel_the_magnetism で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **Upload** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

磁石が近づくと、回路が閉じます。 *2.5 -* **ボタン値の読み取り** 章のボタンと同じように動作します。

6.14 2.10 - 人の動きを検出する

受動型赤外線センサー（PIR センサー）は、視野内の物体が放出する赤外線（IR）を測定できる一般的なセンサーです。簡単に言えば、人体や他の動物から放出される赤外線を受け取り、それによって人々や他の動物の動きを検出します。具体的には、誰かがあなたの部屋に入ったことをメインコントロールボードに通知します。

PIR モーションセンサーモジュール

必要なコンポーネント

このプロジェクトに必要なコンポーネントは以下の通りです。

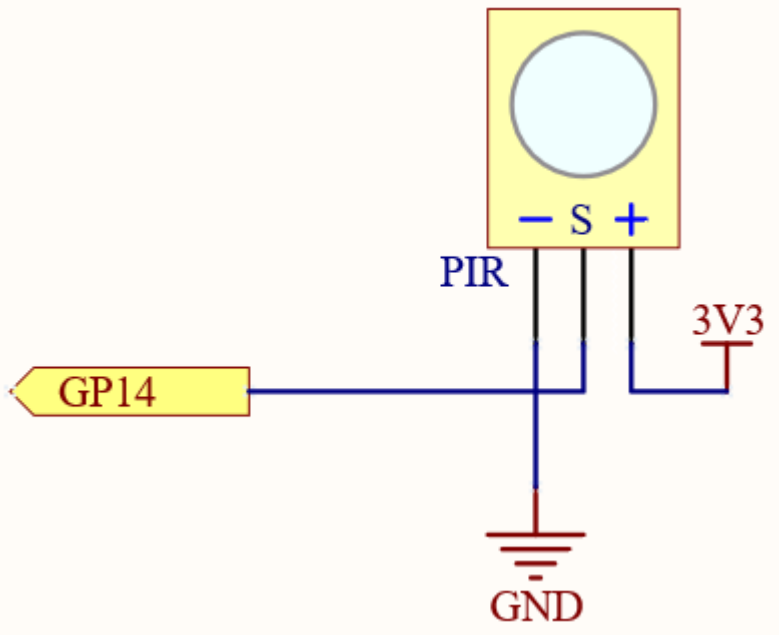
一式をまとめて購入すると便利です、そのためのリンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

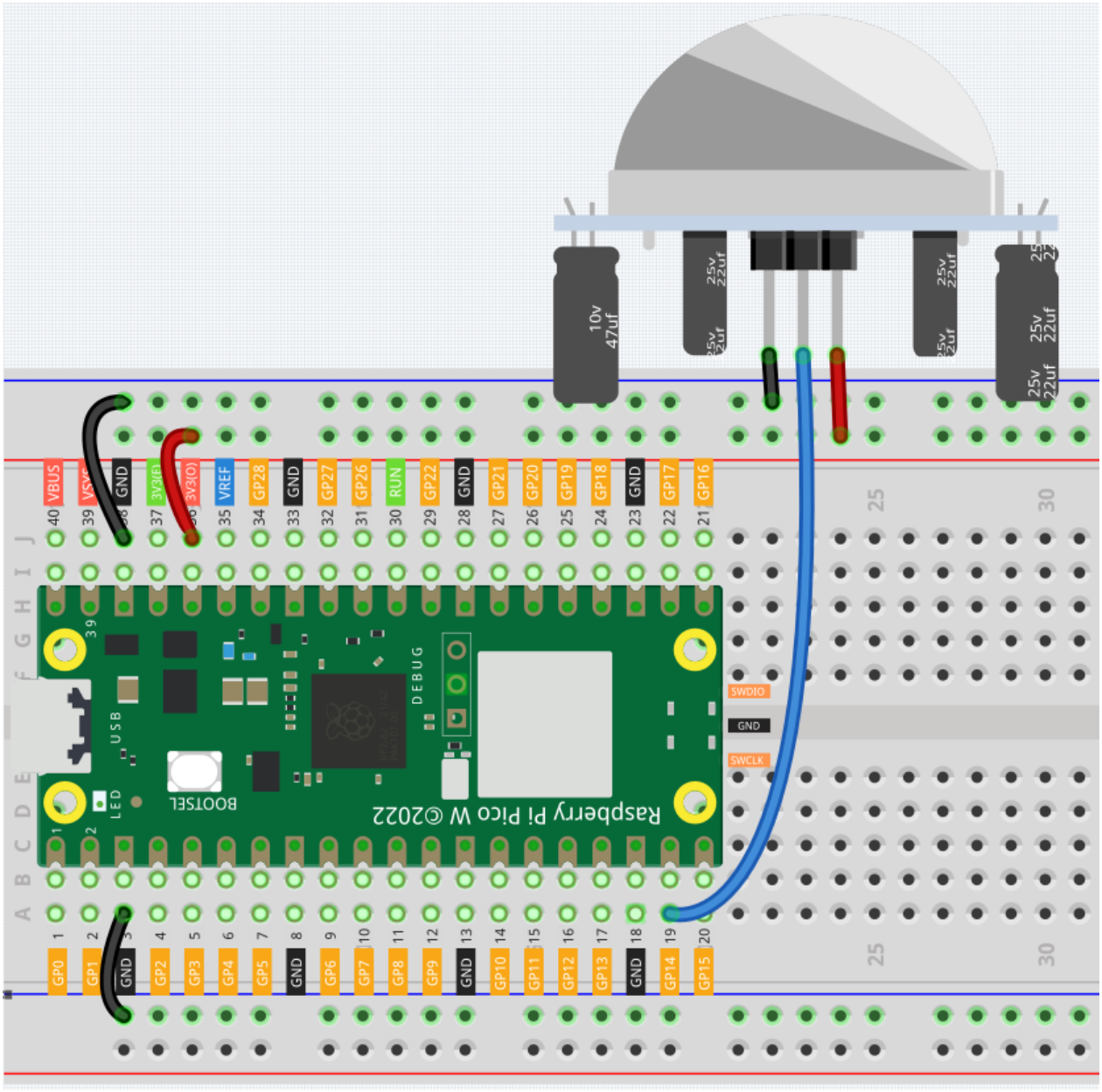
SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	ブレッドボード	1	
4	ジャンパーワイヤー	数本	
5	*PIR* モーションセンサーモジュール	1	

回路図



PIR モジュールが誰かが通り過ぎるのを検出すると、GP14 は高くなります。そうでなければ低いままです。

配線



コード

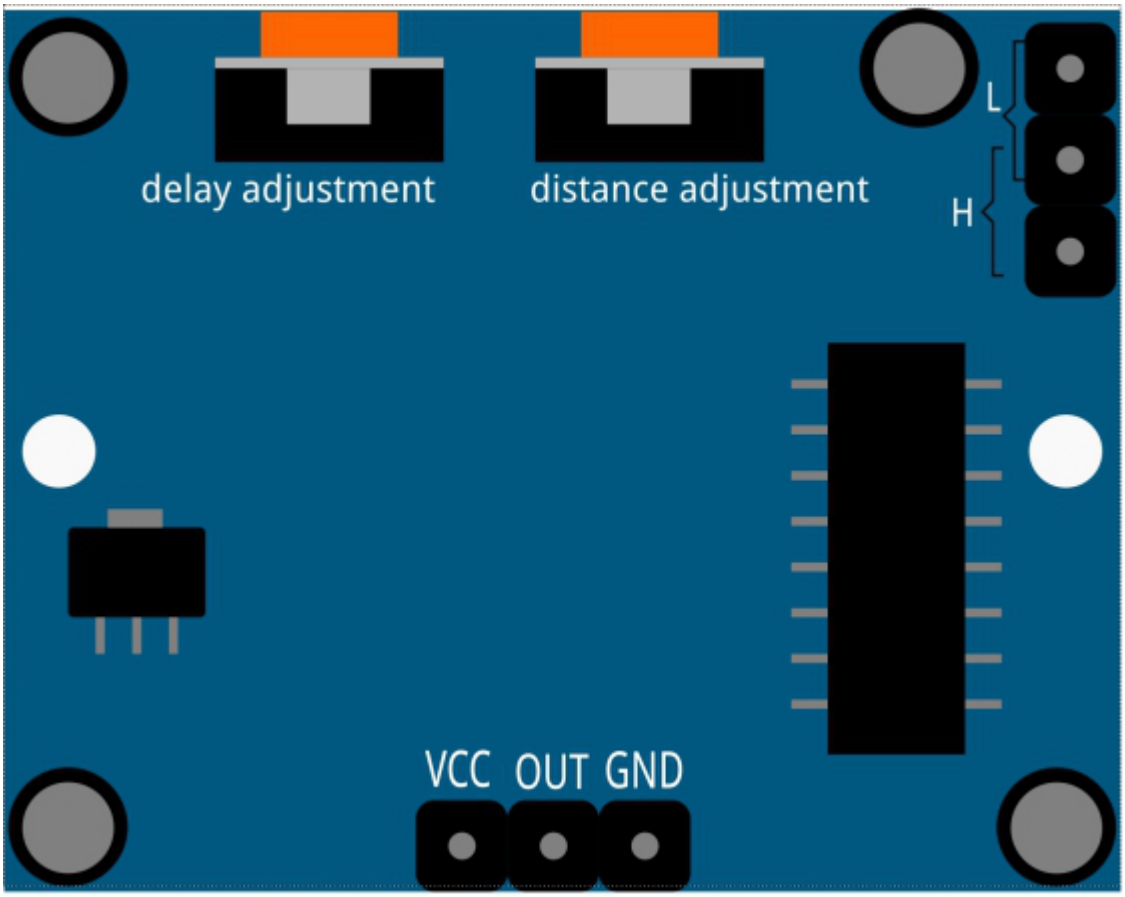
注釈:

- ファイル 2.10_detect_human_movement.ino は、kepler-kit-main/arduino/2.10_detect_human_movement のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

プログラムが実行された後、PIR モジュールが近くに誰かいることを検出すると、シリアルモニターに「Somebody here!」と表示されます。

詳しく知る

PIR は非常に敏感なセンサーです。使用環境に適応させるためには、調整が必要です。2 つのポテンショメータがある側を自分に向けて、両方のポテンショメータを反時計回りに最後まで回し、L と中央のピンにジャンパーキャップを挿入します。



1. トリガーモード

   まずは、角にジャンパーキャップがあるピンを見てみましょう。これによって、PIR は繰り返し可能なトリガーモードまたは非繰り返し可能なトリガーモードに入ることができます。

   現在、ジャンパーキャップは中央のピンと L ピンを接続しており、PIR は非繰り返し可能なトリガーモードになっています。このモードでは、PIR が生物の動きを検出すると、約 2.8 秒間、メインコントロールボードに高レベルの信号を送ります。.. 印刷されたデータで見ると、作動時間は常に 2800ms 前後です。

   次に、下のジャンパーキャップの位置を変更し、中央のピンと H ピンを接続して、PIR を繰り返し可能なトリガーモードにします。このモードでは、PIR が生物の動きを検出する（センサーの前で静止しているのではなく、動いていることに注意）と、生物が検出範囲内で動き続ける限り、PIR は高レベルの信号をメインコントロールボードに続けて送ります。.. 印刷されたデータで見ると、作動時間は不確定な値です。

2. 遅延調整

   左側のポテンショメータは、二つのジョブの間隔を調整するために使用されます。

   現在、それを反時計回りに最後まで回していますが、これによって PIR は高レベル作動が終了した後、約 5 秒間スリープ状態に入る必要があります。この期間中、PIR は対象エリアの赤外線放射を検出しません。.. 印刷されたデータで見ると、休眠期間は常に 5000ms 以上です。

   ポテンショメータを時計回りに回すと、スリープ時間も増加します。時計回りに最後まで回すと、スリープ時間は最大 300s になります。

3. 距離調整

   中央のポテンショメータは、PIR の感知距離範囲を調整するために使用されます。

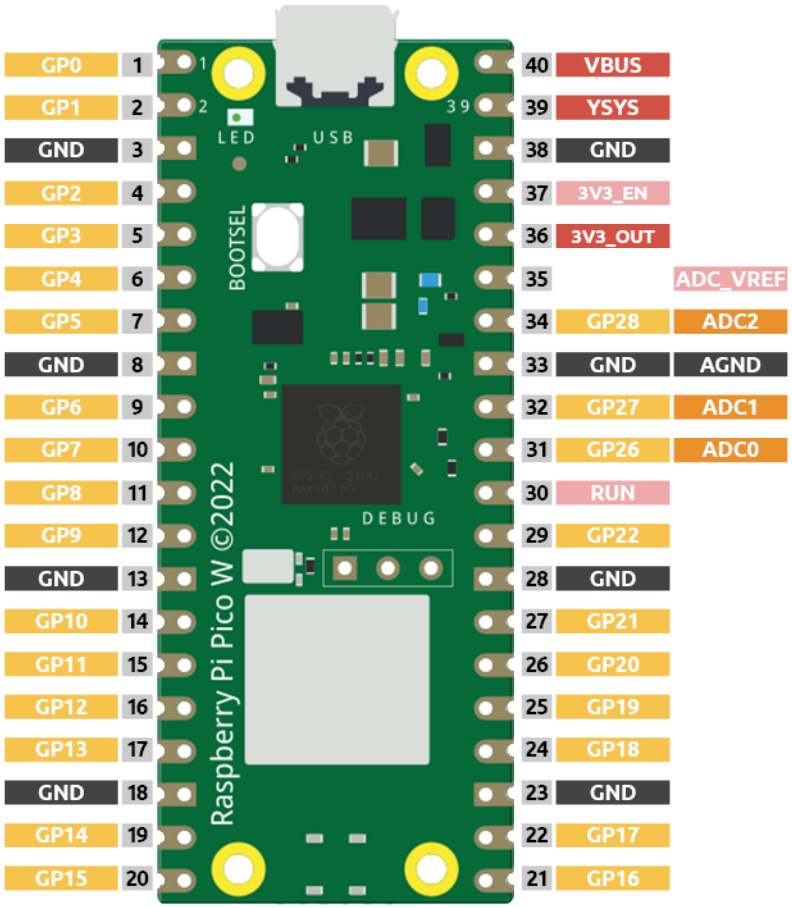
   距離調整のポテンショメータのつまみを **時計回り** に回すと、感知距離範囲が増加し、最大感知距離範囲は約 0-7 メートルです。**反時計回り** に回すと、感知距離範囲が減少し、最小感知距離範囲は約 0-3 メートルです。

6.15 2.11 - ノブを回す

前回のプロジェクトでは、Pico W のデジタル入力を使用しました。たとえば、ボタンはピンを低レベル（オフ）から高レベル（オン）に変更できます。これは 2 値の動作状態です。

しかし、Pico W は別のタイプの入力信号、つまりアナログ入力も受け取ることができます。完全に閉じた状態から完全に開いた状態まで、任意の状態になることができ、可能な値の範囲があります。アナログ入力によって、マイクロコントローラは物理世界の光強度、音強度、温度、湿度などを感知できます。

通常、マイクロコントローラにはアナログ入力を実装するための追加ハードウェア、すなわちアナログ-デジタル変換器（ADC）が必要です。しかし、Pico W 自体には直接使用できる内蔵 ADC があります。



Pico W には、アナログ入力を使用できる GPIO ピンが 3 つあります、GP26、GP27、GP28。すなわち、アナログチャンネル 0、1、2 です。さらに、内蔵の温度センサーに接続された第 4 のアナログチャンネルもありますが、ここでは紹介しません。

このプロジェクトでは、ポテンショメータのアナログ値を読み取ろうとしています。

- **ポテンショメーター**

必要なコンポーネント

このプロジェクトに必要なコンポーネントは以下の通りです。

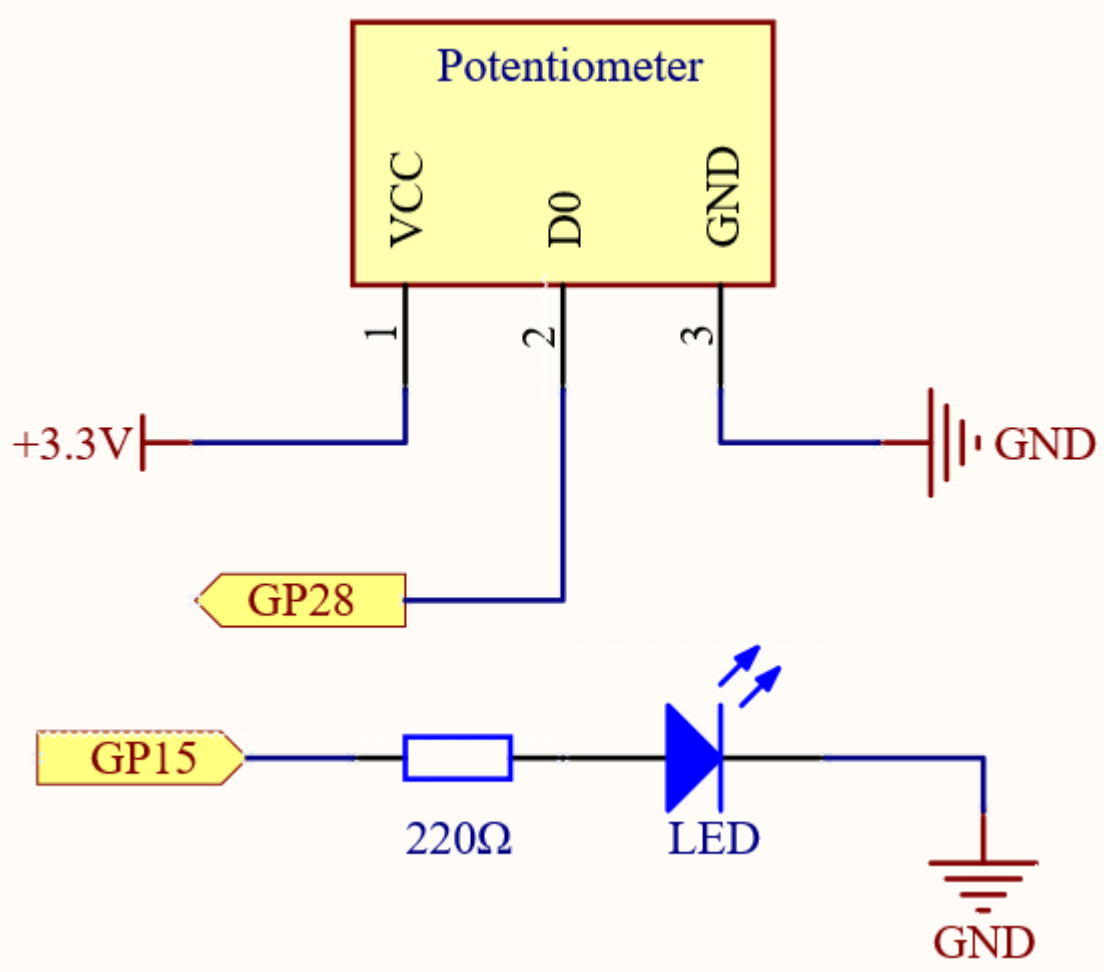
一式をまとめて購入すると便利です、そのためのリンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*LED*	1	
7	**ポテンショメーター**	1	

回路図



ポテンショメータはアナログデバイスであり、2 つの異なる方向に回すことができます。

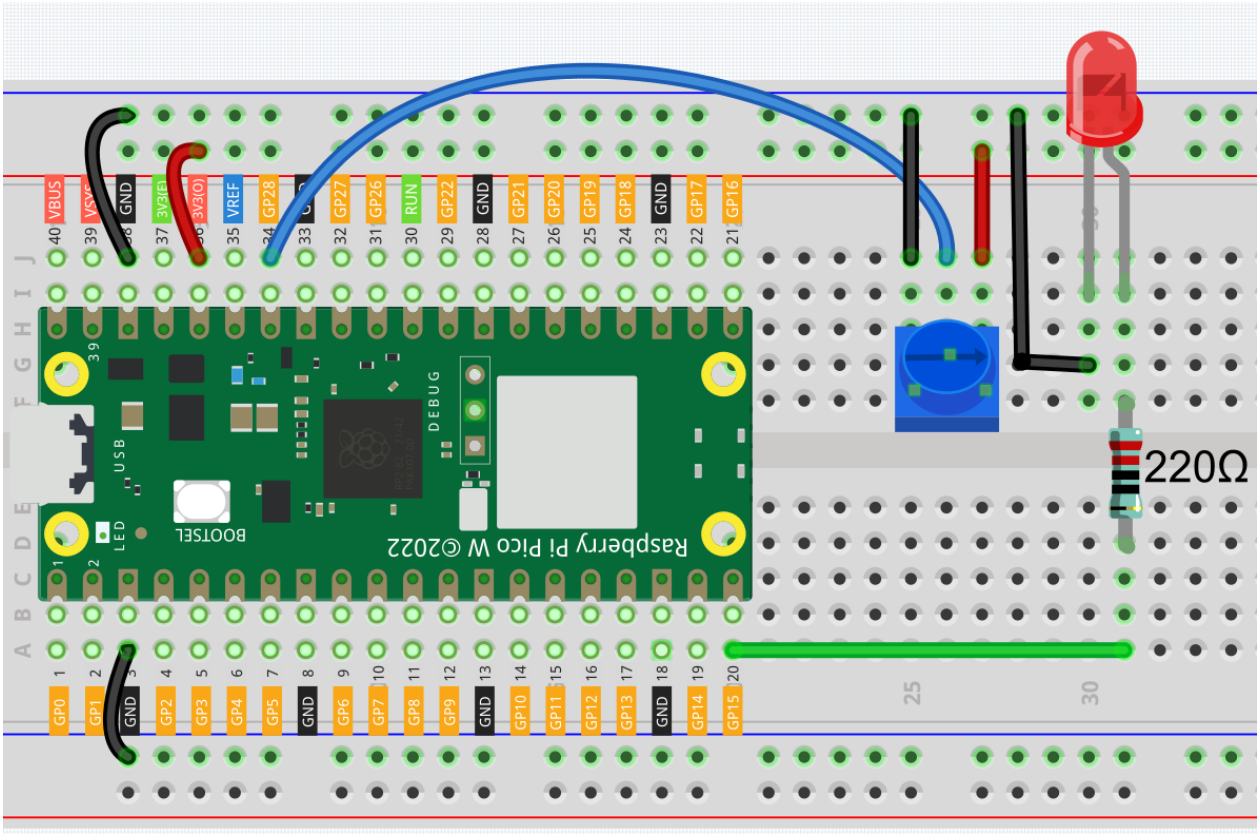
ポテンショメータの中央のピンをアナログピン GP28 に接続します。Raspberry Pi Pico W には、マルチチャンネル、16 ビットのアナログ-デジタル変換器が含まれています。これは、入力電圧を 0 から動作電圧（3.3V）の間で 0 から 65535 までの整数値にマッピングすることを意味します。したがって、GP28 の値は 0 から 65535 までの範囲です。

以下に計算式を示します。

(Vp/3.3V) x 65535 = Ap

次に、GP28（ポテンショメータ）の値を GP15（LED）の PWM 値としてプログラムします。このようにすると、ポテンショメータを回すことで、LED の明るさも同時に変わることがわかります。

配線



コード

注釈:

- ファイル `2.11_turn_the_knob.ino` は、パス `kepler-kit-main/arduino/2.11_turn_the_knob` の下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。

- **Upload** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

プログラムが実行されているとき、シリアルモニターで GP28 ピンによって現在読み取られているアナログ値を見ることができます。ノブを回すと、値は 0 から 1023 まで変わります。同時に、アナログ値が増加するにつれて、LED の明るさも増加します。

どのように動作するか？

シリアルモニターを有効にするには、 `setup()` でシリアル通信を開始し、データレートを 9600 に設定する必要があります。

```
void setup() {
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}
```

- Serial

ループ関数では、ポテンショメータの値を読み取り、その値を 0-1023 から 0-255 にマッピングし、最終的にマッピング後の値を使用して LED の明るさを制御します。

```
void loop() {
    int sensorValue = analogRead(sensorPin);
    Serial.println(sensorValue);
    int brightness = map(sensorValue, 0, 1023, 0, 255);
    analogWrite(ledPin, brightness);
}
```

- analogRead() は、sensorPin（ポテンショメータ）の値を読み取り、変数 `sensorValue` に割り当てるために使用されます。

```
int sensorValue = analogRead(sensorPin);
```

- シリアルモニターで SensorValue の値を表示します。

```
Serial.println(sensorValue);
```

- ここで、 map(value, fromLow, fromHigh, toLow, toHigh) 関数が必要です。ポテンショメータで読み取られる値は 0-1023 の範囲であり、PWM ピンの値は 0-255 の範囲です。この関数は、値を別の範囲に再マッピングするために使用されます。

```
int brightness = map(sensorValue, 0, 1023, 0, 255);
```

- これで、この値を使用して LED の明るさを制御できます。

```
analogWrite(ledPin, brightness);
```

6.16 2.12 - 光を感じる

フォトレジスタは、アナログ入力に典型的に使用されるデバイスであり、ポテンショメータと非常に似た方法で使用されます。その抵抗値は光の強度に依存し、照射される光が強ければ抵抗値は小さくなり、逆に、光が弱ければ抵抗値は増加します。

- **フォトレジスタ**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

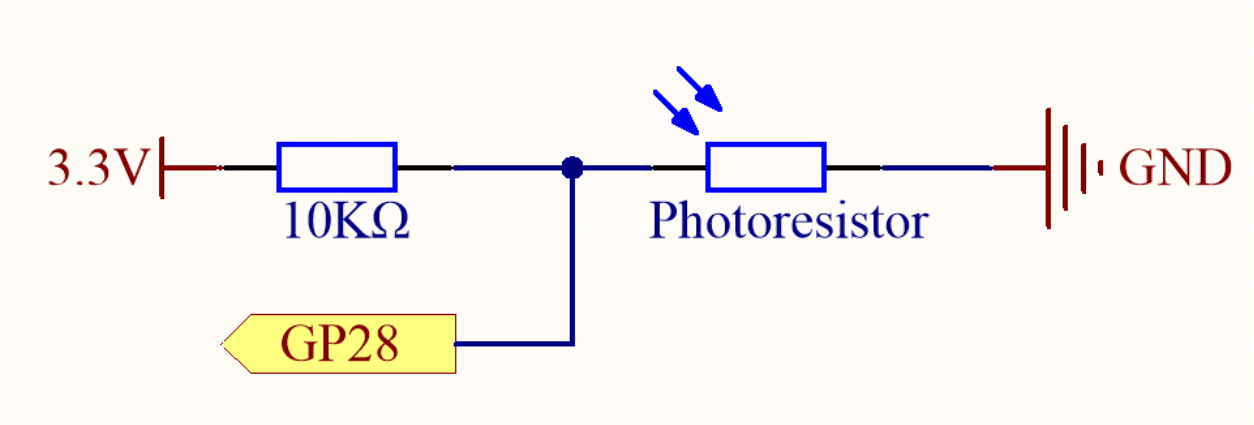
一式をまとめて購入すると便利です、そのためのリンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**フォトレジスタ**	1	

回路図



この回路では、10K Ωの抵抗器とフォトレジスタが直列に接続され、流れる電流は同じです。10K Ωの抵抗器は保護として機能し、GP28 はフォトレジスタの電圧変換後の値を読み取ります。

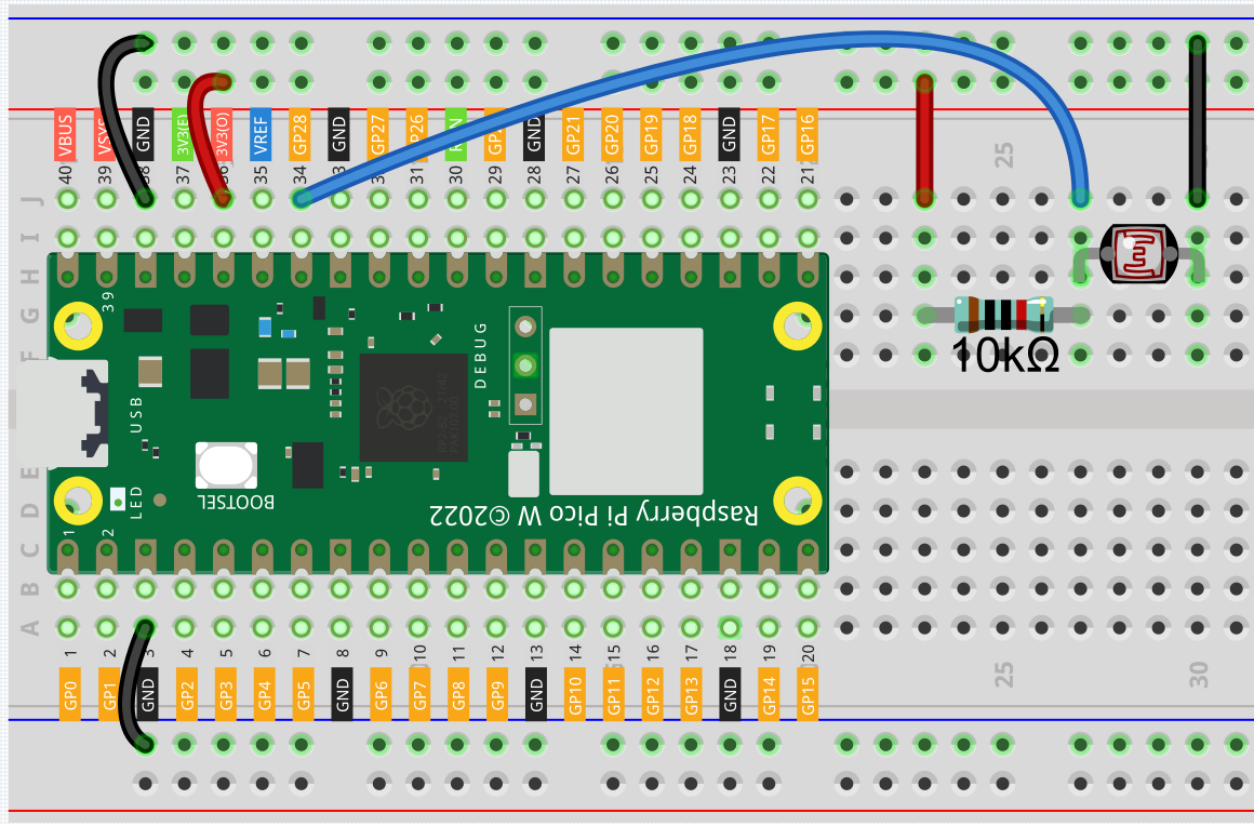
光が強くなると、フォトレジスタの抵抗が減少し、その結果、電圧が低下し、GP28 からの値も低下します。光が十分に強いと、フォトレジスタの抵抗はほぼ 0 に近く、GP28 の値もほぼ 0 に近くなります。このとき、10K Ωの抵抗器は、3.3V と GND が直接接続されて短絡するのを防ぐ保護役となります。

フォトレジスタを暗い状況に置くと、GP28 の値は上昇します。十分に暗い状況では、フォトレジスタの抵抗は無限大になり、その電圧はほぼ 3.3V（10K Ωの抵抗は無視できる）に近く、GP28 の値は最大値 65535 に近くなります。

計算式は以下の通りです。

(Vp/3.3V) x 65535 = Ap

配線



コード

注釈:

- ファイル 2.12_feel_the_light.ino は、 kepler-kit-main/arduino/2.12_feel_the_light のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

プログラムが実行された後、シリアルモニターはフォトレジスタの値を出力します。手で覆うか、フラッシュライトで照らして、値がどのように変わるかを確認できます。

6.17 2.13 - 温度計

温度計は、温度または温度勾配（物体の温度または冷度の程度）を測定する装置です。温度計には、2 つの重要な要素があります：(1) 温度に変化が生じる何らかの変化が起こる温度センサー（例：水銀温度計の球根や赤外線温度計の焦電センサー）；そして（2）この変化を数値で表示する手段（例：水銀温度計にマークされた可視スケールや赤外線モデルのデジタル表示)。温度計は、テクノロジーと産業でプロセスを監視するため、気象学、医学、科学研究で広く使用されています。

サーミスターは、温度に強く依存する抵抗を持つ温度センサーの一種で、2 つのタイプがあります：負の温度係数（NTC）と正の温度係数（PTC）とも呼ばれ、NTC と PTC です。PTC サーミスターの抵抗は温度とともに増加し、NTC の状態はそれに反対です。

この実験では、 **NTC サーミスター** を使用して温度計を作成します。

- **サーミスター**

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

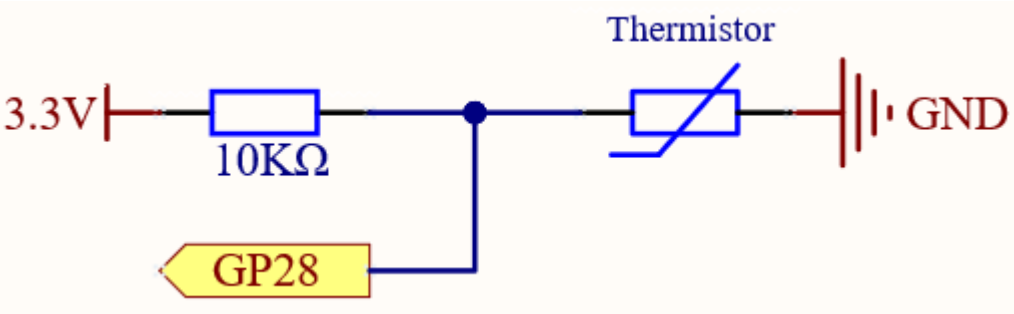
全体のキットを購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**サーミスター**	1	

回路図



この回路では、10K Ωの抵抗とサーミスタは直列に接続され、流れる電流は一定です。10K Ωの抵抗は保護役割を果たし、GP28 はサーミスタの電圧変換後の値を読み取ります。

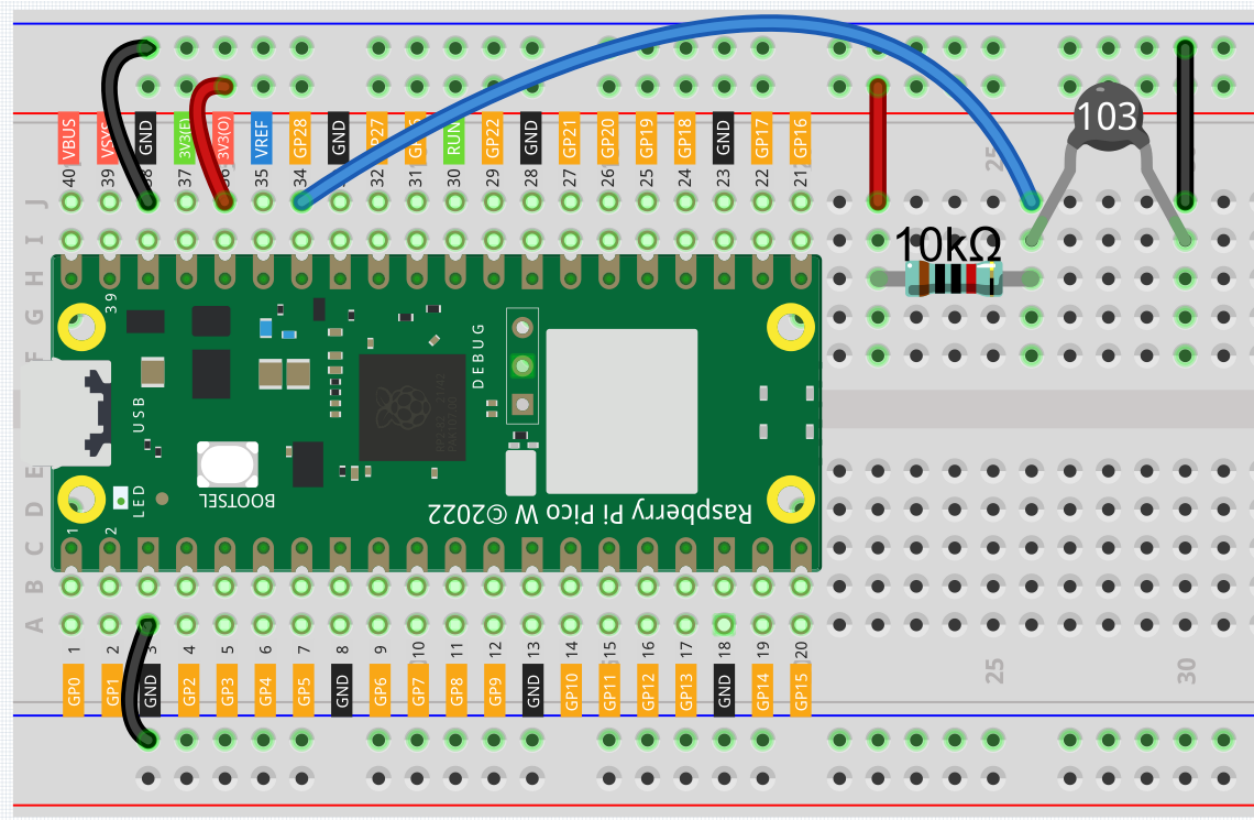
温度が上昇すると、NTC サーミスタの抵抗値は減少し、その結果、電圧も減少するため、GP28 からの値も減少します。温度が十分に高いと、サーミスタの抵抗はほぼ 0 に近づき、GP28 の値も 0 に近くなります。このとき、10K Ωの抵抗は保護役割を果たし、3.3V と GND が短絡するのを防ぎます。

温度が低下すると、GP28 の値が増加します。温度が十分に低い場合、サーミスタの抵抗は無限大になり、その電圧はほぼ 3.3V に近づきます（10K Ωの抵抗は無視できる）。

計算式は以下のとおりです。

(Vp/3.3V) x 65535 = Ap

配線



注釈:

- サーミスタは黒く、103 と表示されています。
- 10K Ω抵抗の色リングは赤、黒、黒、赤、茶色です。

コード

注釈:

- ファイル `2.13_thermometer.ino` を `kepler-kit-main/arduino/2.13_thermometer` パスから開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- ボード（Raspberry Pi Pico）と正確なポートを選択した後、 **アップロード** ボタンをクリックすることを忘れないでください。

プログラムが実行されると、シリアルモニターに摂氏と華氏の温度が表示されます。

動作原理

各サーミスタには基準となる抵抗があります。この場合、それは 10k Ωで、25 度摂氏で測定されます。

温度が上がると、サーミスタの抵抗が減少します。その後、A/D アダプターによって電圧データがデジタル量に変換されます。

プログラミングにより、摂氏または華氏での温度が出力されます。

```
long a = analogRead(analogPin);
```

この行は、サーミスタの値を読むために使用されます。

```
float tempC = beta / (log((1025.0 * 10 / a - 10) / 10) + beta / 298.0) - 273.0;
float tempF = 1.8 * tempC + 32.0;
```

これらの計算は、サーミスタの値を摂氏度と華氏度に変換します。

注釈: ここでは、抵抗と温度の関係は以下の通りです：

RT = RN expB(1/TK – 1/TN)

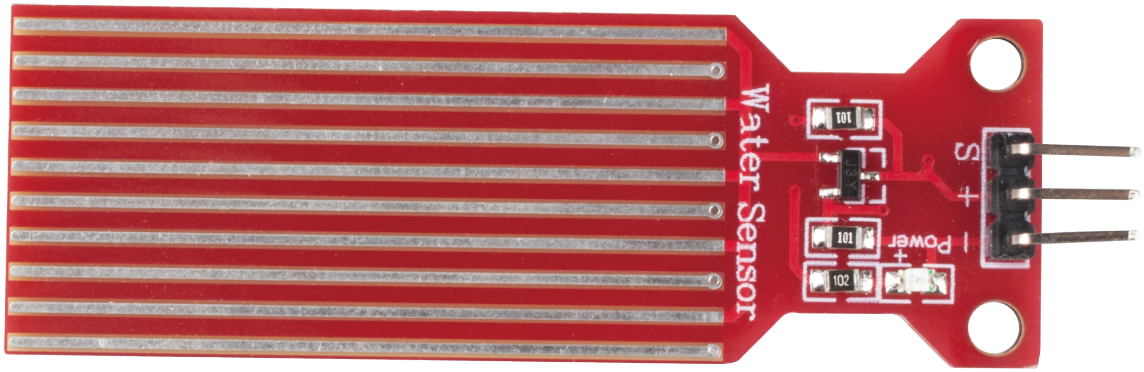
- RT は、温度が TK のときの NTC サーミスタの抵抗です。

- RN は、評価温度 TN での NTC サーミスタの抵抗です。この場合、RN の数値は 10k です。
- TK はケルビン温度で、単位は K です。この場合、TK の数値は 273.15 + 摂氏度です。
- TN は評価ケルビン温度で、単位も K です。この場合、TN の数値は 273.15+25 です。
- そして B（ベータ）は、NTC サーミスタの材料定数であり、熱感度指数とも呼ばれ、数値は 3950 です。
- exp は指数関数の略であり、基数 e は自然数で、おおよそ 2.7 と等しいです。

この関係式は、経験式です。温度と抵抗が有効範囲内の場合にのみ正確です。

このコードは、ケルビン温度を得るために、式 TK=1/(ln(RT/RN)/B+1/TN) に Rt を代入します。

6.18 2.14 - 水位を感じる



水センサーは、水検出のために設計されており、降雨、水位、さらには液体の漏れを感知する広範な用途に使用できます。

このセンサーは、水滴/水量のサイズを測定するために一連の露出した平行なワイヤートレースを用いて水位を測定します。水量は簡単にアナログ信号に変換され、出力されるアナログ値はメイン制御ボードで直接読み取ることができ、水位警報の効果を実現します。

警告: センサーは水中に完全に浸すことはできません。10 本のトレースがある部分だけを水と接触させてください。また、湿度の高い環境でセンサーに電力を供給すると、プローブの腐食が加速し、センサーの寿命が短くなる可能性があるため、測定を取る際にのみ電力を供給することをお勧めします。

- **水位センサーモジュール**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

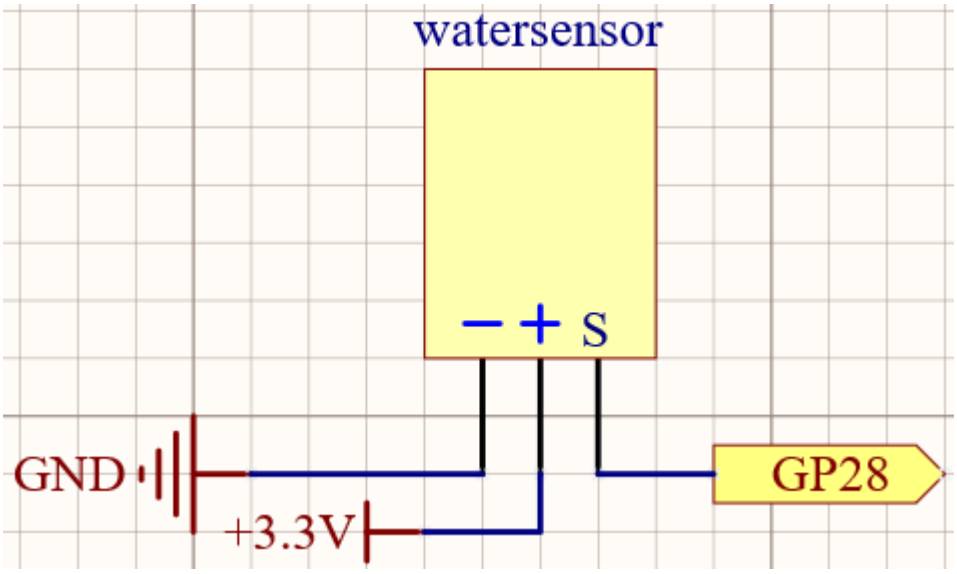
便利なのは一式を購入することです、リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

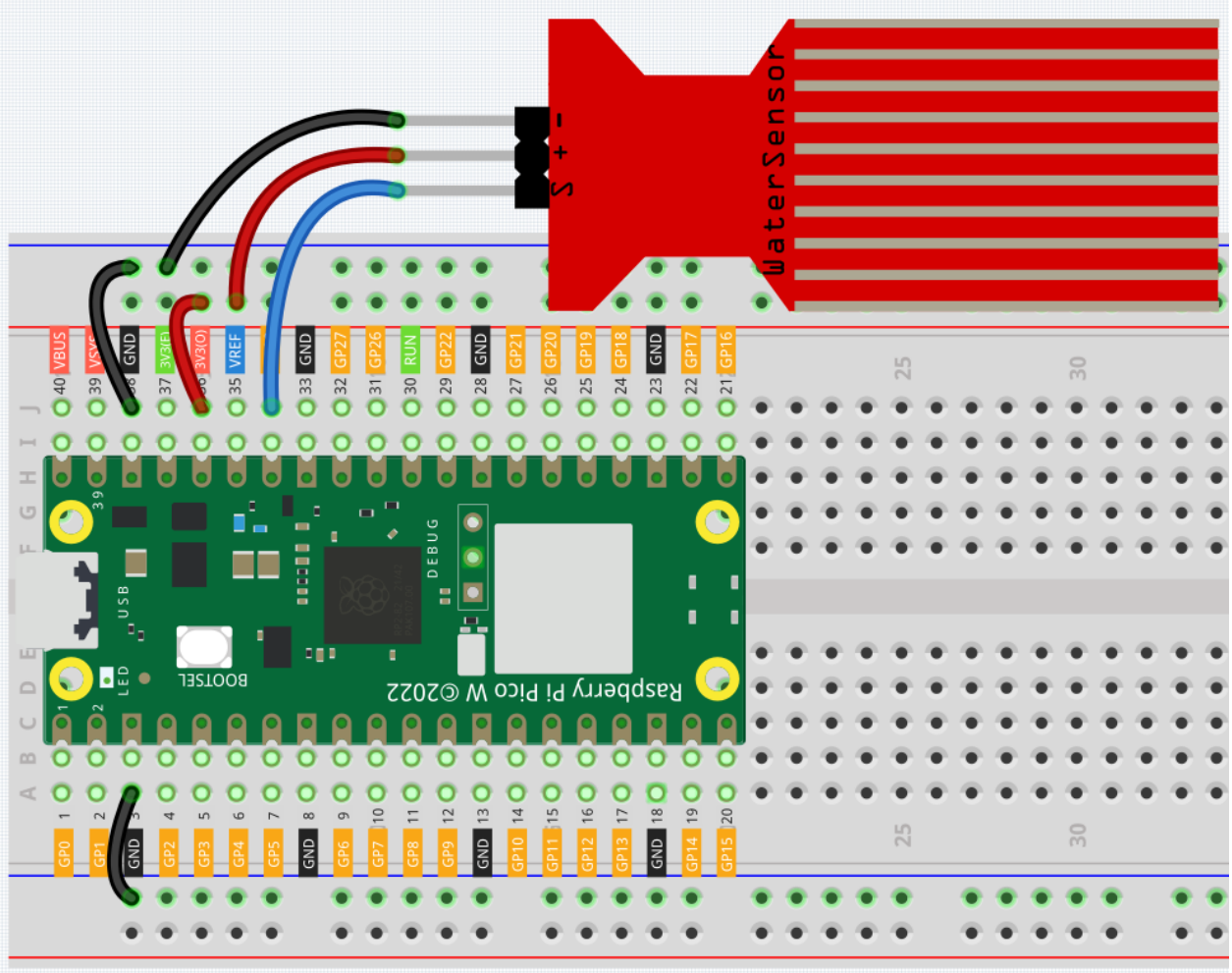
以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**水位センサーモジュール**	1	

回路図



配線



コード

注釈:

- ファイル 2.14_feel_the_water_level.ino は、kepler-kit-main/arduino/2.14_feel_the_water_level のパスにあります。
- または、このコードを **Arduino IDE** にコピーできます。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と適切なポートを選択してください。

プログラムが動作した後、水センサーモジュールをゆっくりと水に浸し、深さが増すにつれて、シェルにはより大きな値が表示されます。

詳細を学ぶ

アナログ入力モジュールをデジタルモジュールとして使用する方法があります。

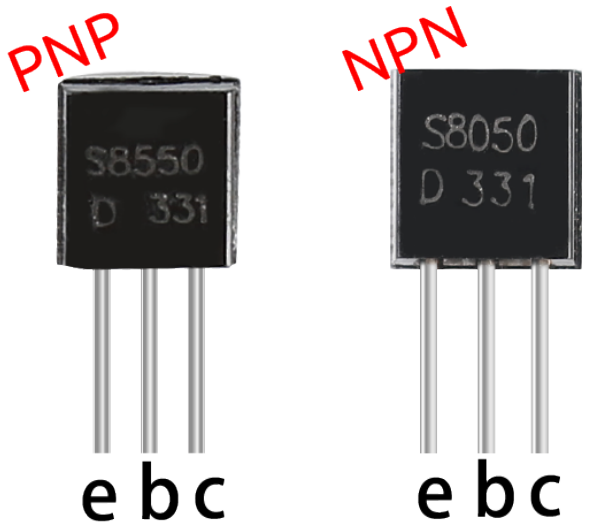
まず、乾燥した環境で水センサーの読み取りを行い、それを記録して閾値として使用します。次に、プログラミングを完了し、水センサーの読み取りを再度行います。水センサーの読み取りが乾燥した環境での読み取りと大きく逸脱すると、それは液体に触れています。つまり、このデバイスを水道管の近くに置くと、水道管が漏れているかどうかを検出できます。

注釈:

- ファイル `2.14_water_level_threshold.ino` は、`kepler-kit-main/arduino/2.14_water_level_threshold` のパスにあります。
- または、このコードを **Arduino IDE** にコピーできます。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と適切なポートを選択してください。

6.19 2.15 - トランジスタの二種類

このキットには、S8550 と S8050 という二種類のトランジスタが付属しています。前者は PNP 型、後者は NPN 型です。見た目は非常によく似ているため、ラベルをしっかりと確認する必要があります。NPN トランジスタに High レベルの信号が通ると、それはエネルギーを帯びます。しかし、PNP 型は Low レベルの信号で制御する必要があります。両方のトランジスタはこの実験で見られるように、非接触スイッチとしてよく使用されます。



LED とボタンを使って、トランジスタの使い方を理解しましょう！

トランジスタ

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

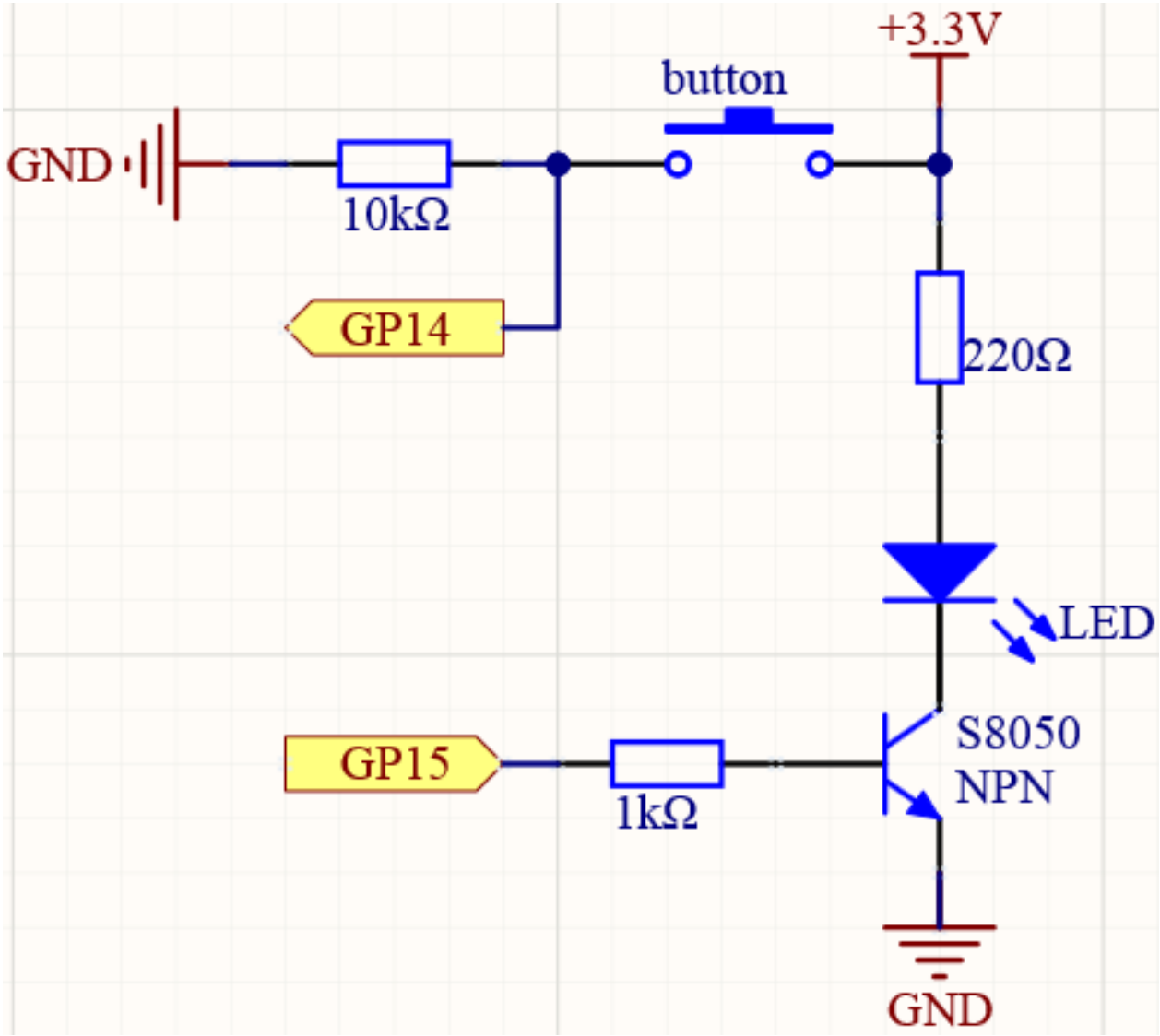
全体のキットを購入する方が確実に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入できます。

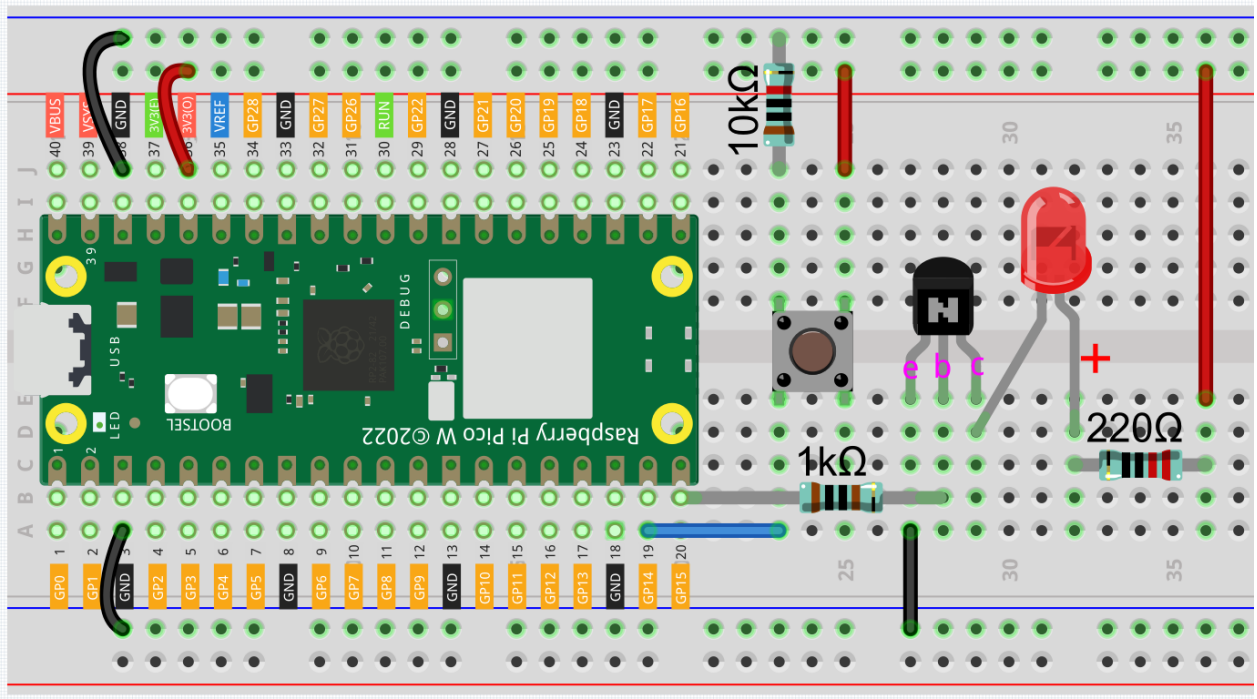
SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	3(220 Ω， 1K Ω，10K Ω)	
6	*LED*	1	
7	**ボタン**	1	
8	**トランジスタ**	1 （S8050/S8550）	

NPN（S8050）トランジスタの接続方法

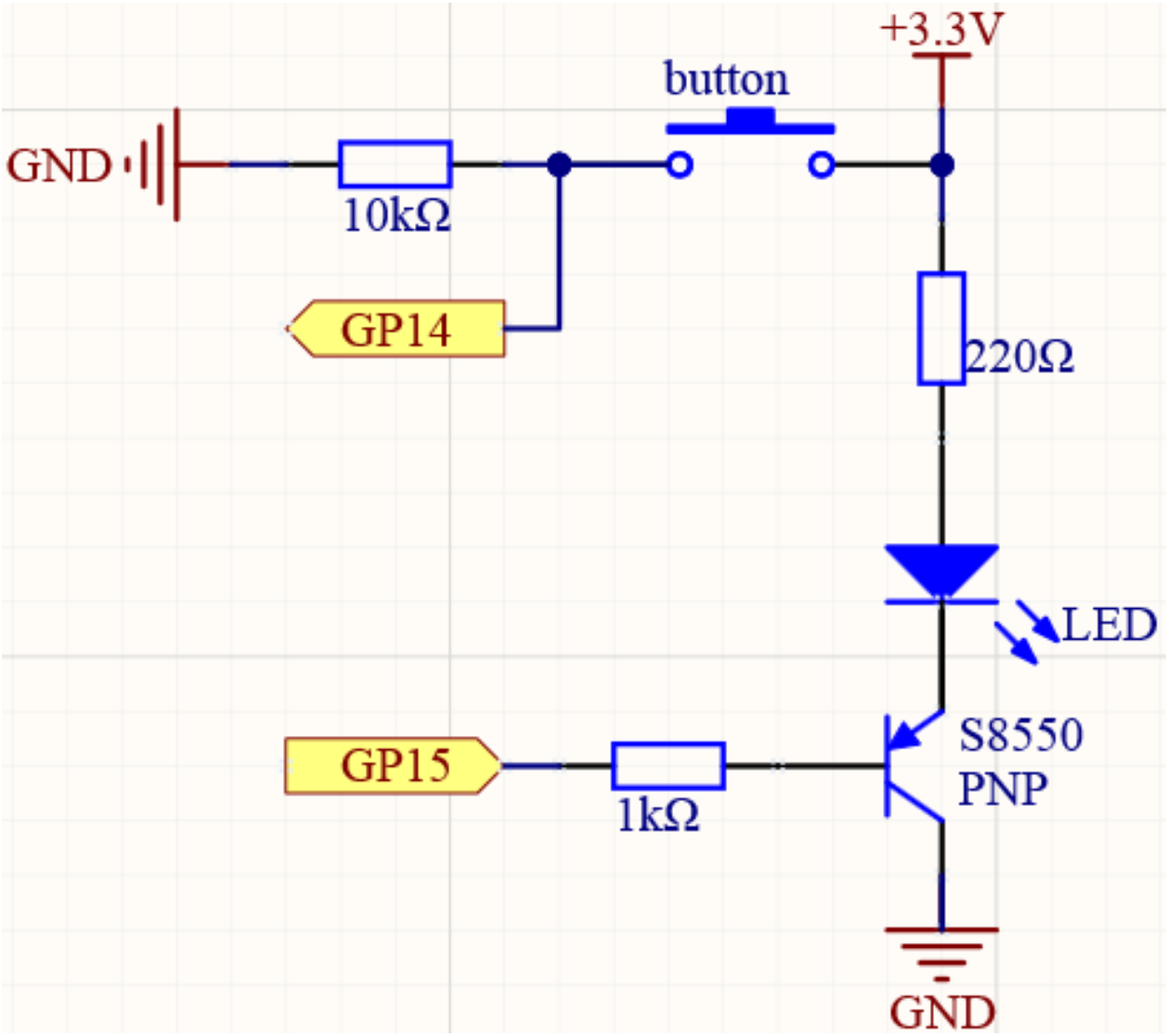


この回路では、ボタンが押されると、GP14 はハイになります。

GP15 をプログラミングしてハイ出力にすると、1k の電流制限抵抗（トランジスタを保護するため）を経て、S8050（NPN トランジスタ）が導通し、LED が点灯するようになります。



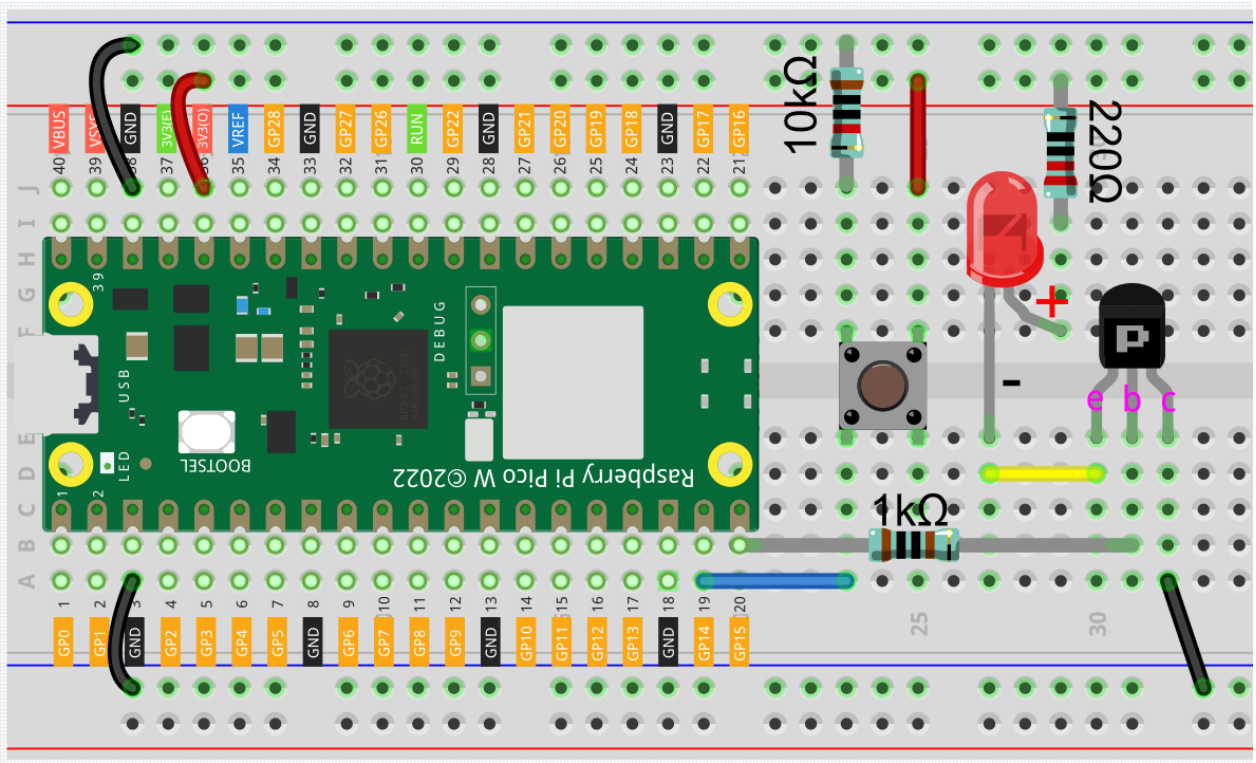
PNP（S8550）トランジスターの接続方法



この回路では、デフォルトで GP14 は低レベルとなっており、ボタンが押されると高レベルになります。

GP15 をプログラムして **低出力** に設定すると、1k の電流制限抵抗（トランジスターを保護するため）を経て、S8550（PNP トランジスター）が導通し、LED が点灯します。

この回路と前の回路との唯一の違いは、前の回路では LED の陰極が **S8050（NPN トランジスター）のコレクター** に接続されているのに対し、この回路では **S8550（PNP トランジスター）のエミッター** に接続されている点です。



コード

注釈:

- ファイル `2.15_transistor.ino` は、`kepler-kit-main/arduino/2.15_transistor` のパスにあります。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と適切なポートを選択してください。

同じコードで 2 種類のトランジスターを制御できます。ボタンを押すと、Pico W はトランジスターに高レベルの信号を送ります。ボタンを離すと、低レベルの信号を送ります。このように、2 つの回路で正反対の現象が発生していることが確認できます。

- S8050（NPN トランジスター）を使用した回路は、ボタンが押されたときに点灯するため、高レベルの導通回路を受け取っています。
- S8550（PNP トランジスター）を使用した回路は、ボタンが離されたときに点灯するため、低レベルの導通回路を受け取っています。

6.20 2.16 - 別の回路を制御する

日常生活で、スイッチを押すだけでランプを点灯または消灯できます。しかし、Pico W を使って、10 分後に自動的に消灯するようにランプを制御したい場合はどうでしょうか？

このアイデアを実現するためには、リレーが役立ちます。

リレーは、一方の回路（通常は低電圧回路）によって制御され、他方の回路（通常は高電圧回路）を制御する特殊な種類のスイッチです。これにより、プログラムで制御できるスマートデバイスになるように、またはインターネットにアクセスできるように、家庭用電器を改造するのが現実的になります。

> **警告:** 電気製品の改造は非常に危険ですので、専門家の指導のもとで行ってください。

- **リレー**

ここでは、ブレッドボード電源モジュールで駆動される簡単な回路を例に、リレーを使ってどのように制御するかを示します。

- cpn_power_module

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

全体のキットを購入することは非常に便利です。リンクは以下のとおりです：

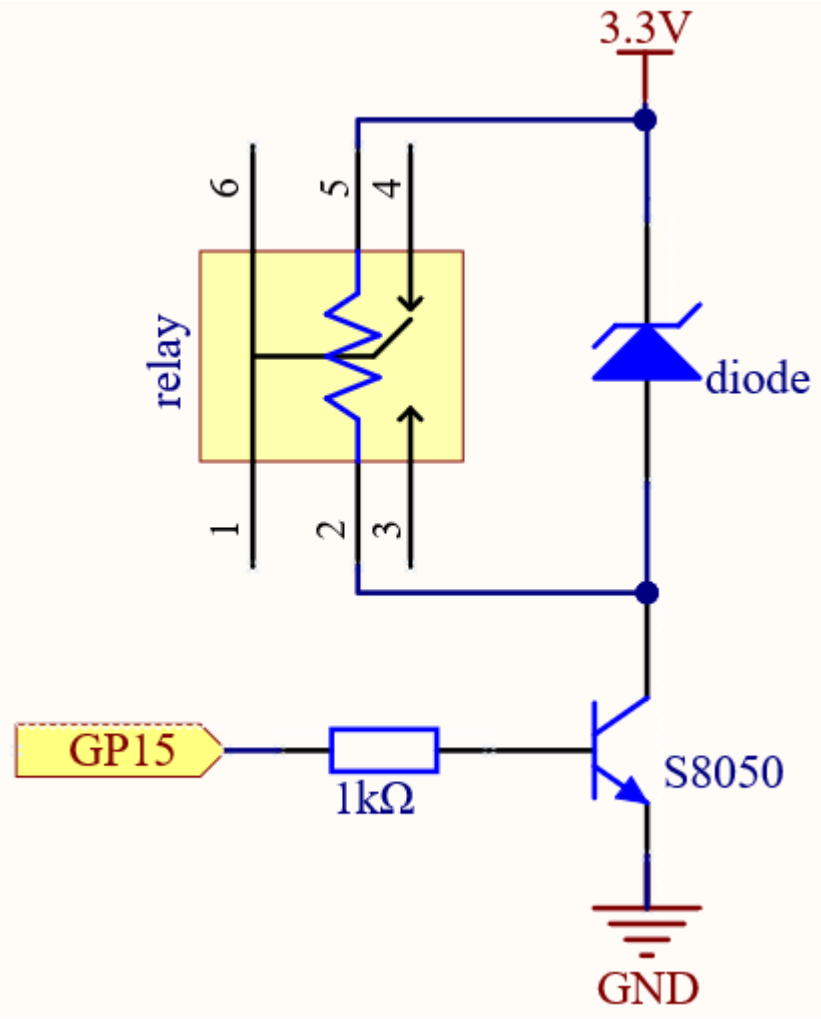
名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

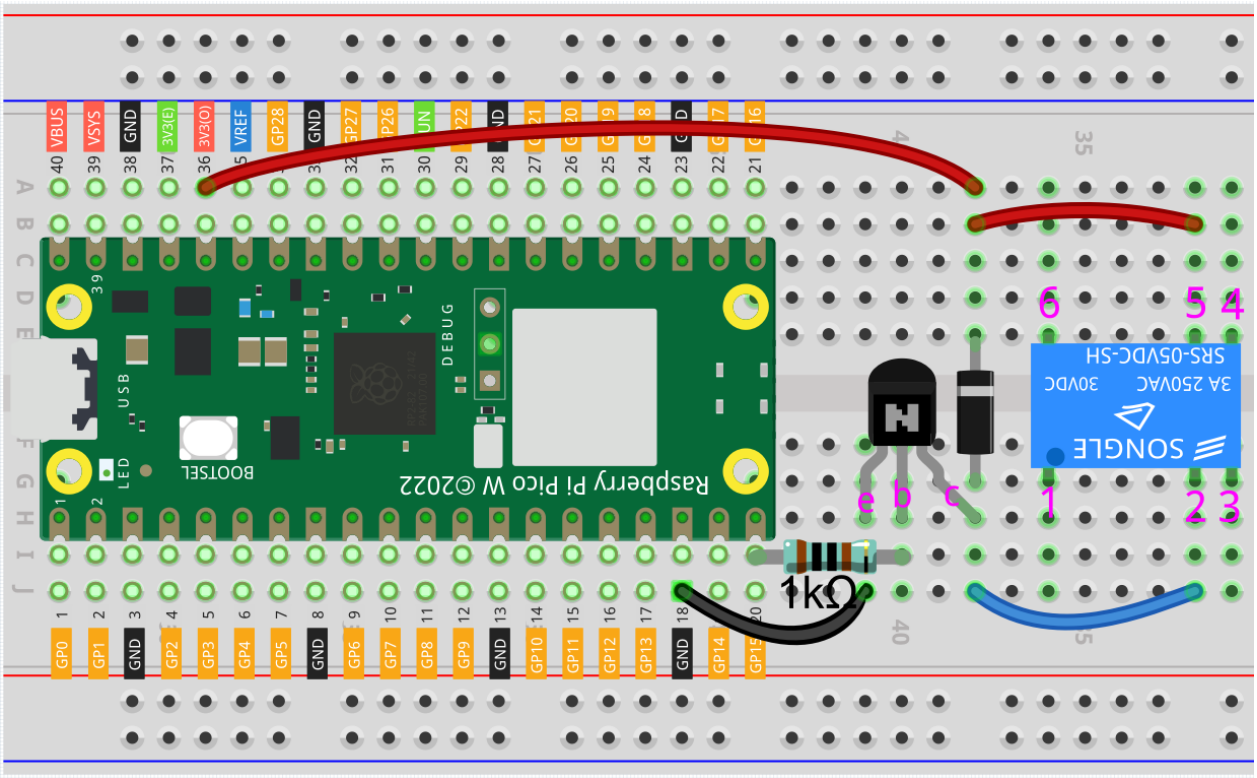
以下のリンクから個別に購入することもできます。

番号	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**ダイオード**	1	
7	**リレー**	1	

配線

まず、リレーを制御するための低電圧回路を作成します。リレーの駆動には高電流が必要なため、ここではトランジスタ S8050 を使用します。


3.3V
6
5
4
relay
diode
1
2
3
GP15
1kΩ
S8050
GND



ここでは、回路を保護するためにダイオード（フリーホイーリングダイオード）が使用されています。カソードは、電源に接続された銀色のリボンの端であり、アノードはトランジスタに接続されています。

電圧入力が High（5V）から Low（0V）に変わると、トランジスタは飽和（増幅、飽和、カットオフ）からカットオフに変わり、コイルを通る電流の道が突如なくなります。

この時点で、このフリーホイーリングダイオードが存在しない場合、コイルは供給電圧の数倍にもなる自己誘導電位を両端で生成します。この電圧プラストランジスタ電源の電圧は、それを焼き尽くすほどです。

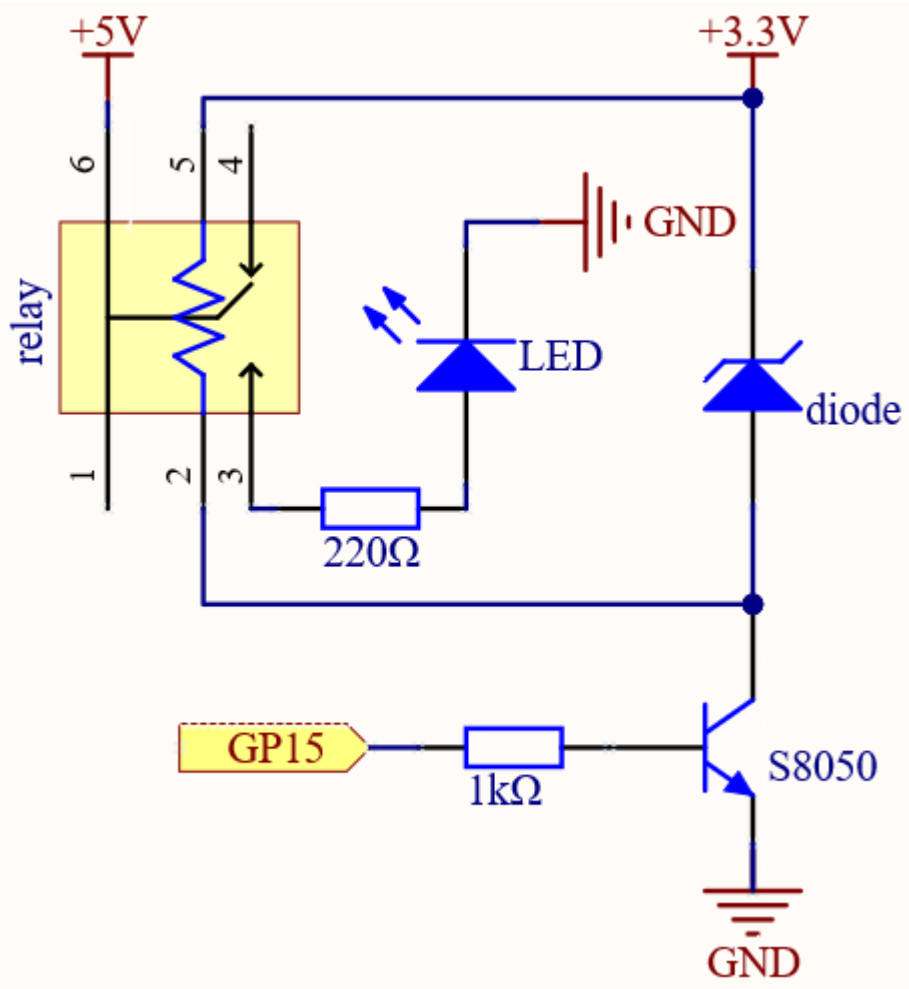
ダイオードを追加することで、コイルとダイオードは瞬時にコイルに蓄えられたエネルギーで駆動される新しい回路を形成し、放電します。これにより、回路上のトランジスタなどのデバイスが過度な電圧で損傷するのを防ぎます。

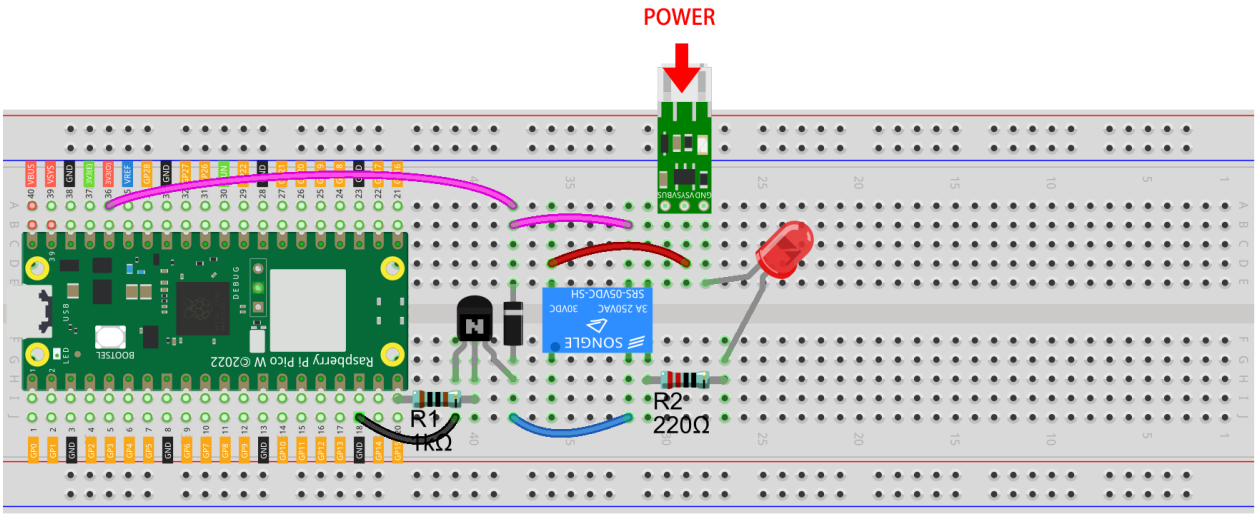
- **ダイオード**
- Flyback Diode - Wikipedia

この時点でプログラムは実行可能です。実行すると「チクタク」という音がしますが、これはリレー内のコンタクタコイルが吸い寄せられて破れる音です。

次に、負荷回路の両端をそれぞれリレーのピン 3 と 6 に接続します。

..（前述のブレッドボード電源モジュールで駆動される簡単な回路を例に取ります。）





この時点で、リレーは負荷回路のオンとオフを制御できるようになります。

コード

注釈:

- ファイル `2.16_relay.ino` は、 `kepler-kit-main/arduino/2.16_relay` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **Upload** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

コードが実行されると、リレーは制御された回路の動作状態を 2 秒ごとに切り替えます。リレー回路と負荷回路の対応関係をさらに明確にするために、1 行を手動でコメントアウトできます。

もっと詳しく

リレーのピン 3 は通常開いており、コンタクタコイルが動作しているときだけオンになります。ピン 4 は通常閉じており、コンタクタコイルが通電されたときにオンになります。ピン 1 はピン 6 に接続されており、負荷回路の共通端子です。

負荷回路の一端をピン 3 からピン 4 に切り替えると、正確に反対の動作状態が得られます。

3. 音 & 表示 & 動き

6.21 3.1 - ビープ音

アクティブ・ブザーは、LED を点灯させるのと同じくらい簡単に使える典型的なデジタル出力デバイスです！

- **ブザー**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

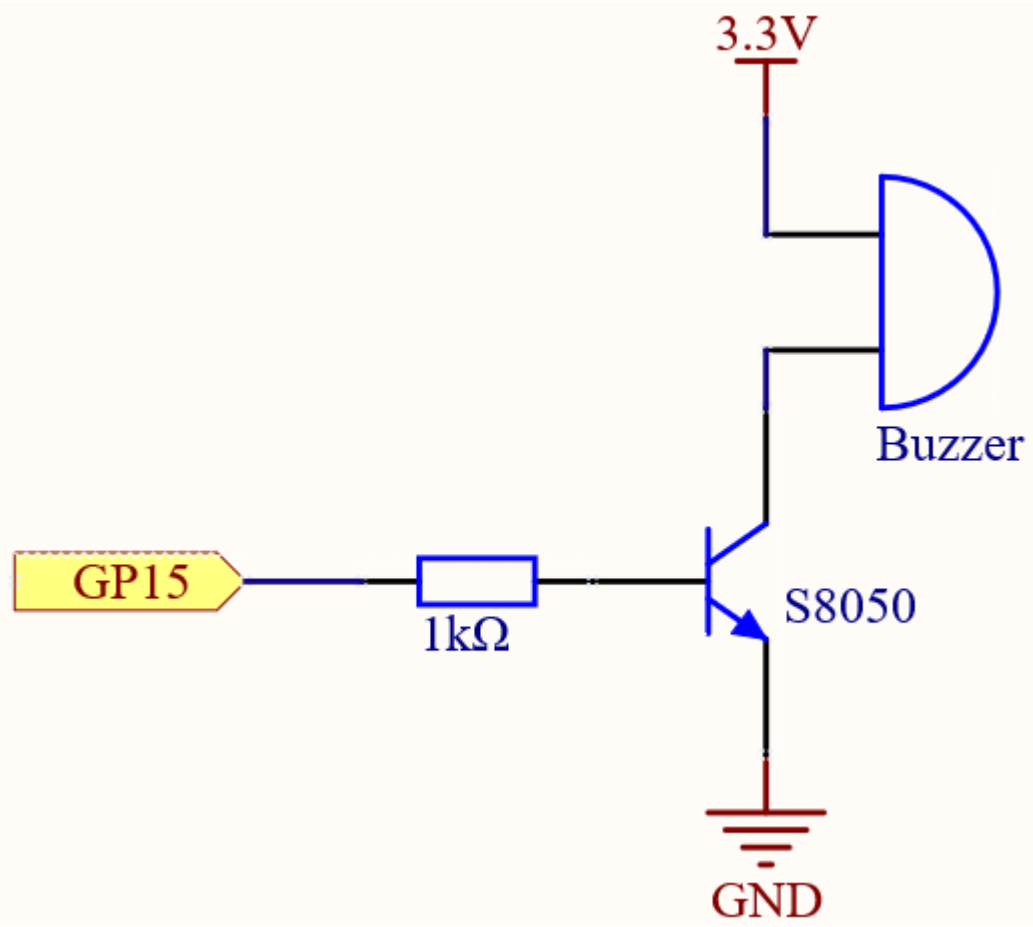
全体のキットを購入するのが確かに便利です、リンクはこちら：

名前	このキットのアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個々に購入することもできます。

SN	コンポーネントの紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	アクティブ **ブザー**	1	

回路図



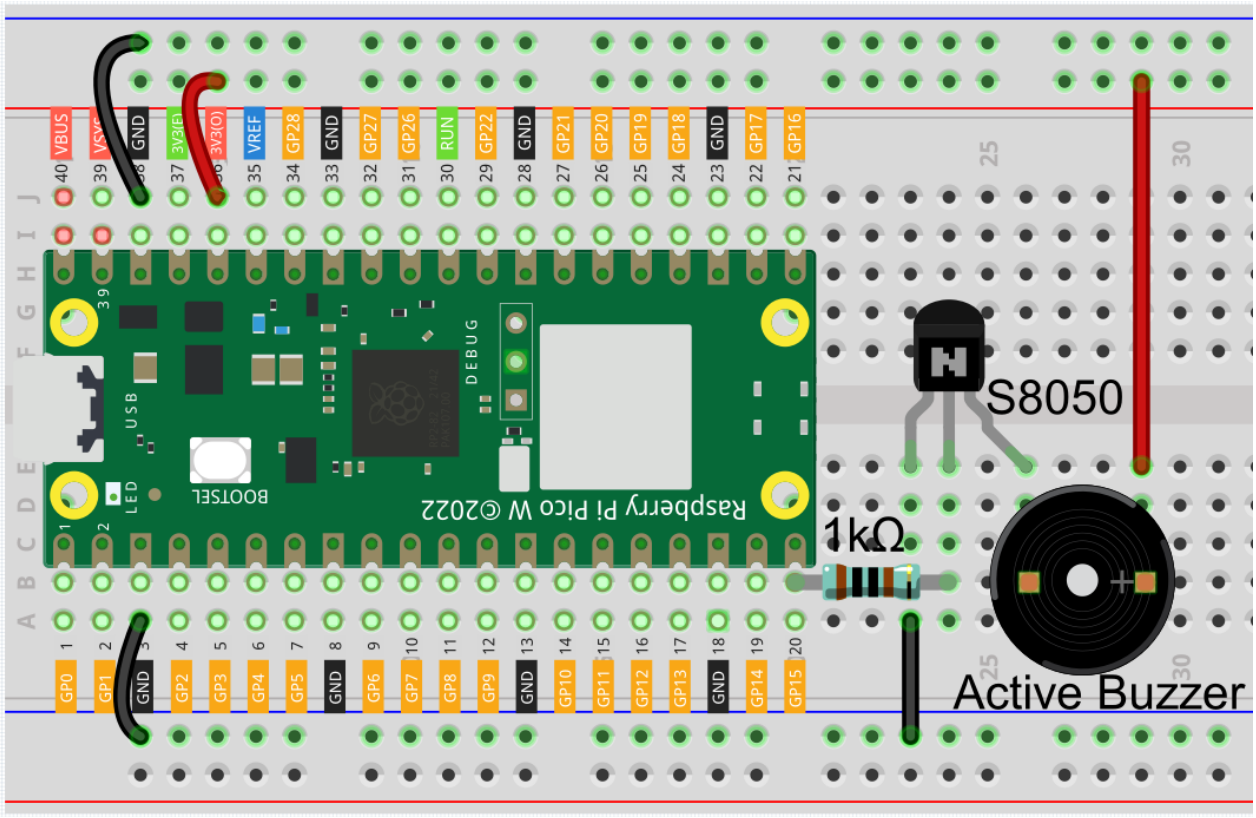
GP15 の出力が高い場合、1K の電流制限抵抗を経て（トランジスタを保護するため）、S8050（NPN トランジスタ）は導通し、ブザーが音を出します。

S8050（NPN トランジスタ）の役割は、電流を増幅してブザーの音を大きくすることです。実際、GP15 に直接ブザーを接続することもできますが、ブザーの音は小さくなるでしょう。

配線

キットには 2 種類のブザーが含まれています。アクティブブザーが必要です。裏返して、密封された背面（露出した PCB ではない）が必要なものです。

ブザーは動作時にトランジスタを使用する必要があります。ここでは、S8050（NPN トランジスタ）を使用します。



コード

注釈:

- ファイル `3.1_beep.ino` は、 `kepler-kit-main/arduino/3.1_beep` のパスの下で開くことができます。
- または、このコードを **Arduino IDE** にコピペしてください。
- ボード（Raspberry Pi Pico）と正確なポートを選択したら、 **アップロード** ボタンをクリックする前に設定を忘れないでください。

コードが実行された後、1 秒ごとにビープ音が聞こえます。

6.22 3.2 - カスタムトーン

前回のプロジェクトではアクティブブザーを使用しましたが、今回はパッシブブザーを使用します。

アクティブブザーと同様に、パッシブブザーも電磁誘導の現象を利用して動作します。違いは、パッシブブザーには振動源がないため、直流信号を使用しても音を出さない点です。しかし、この特性により、パッシブブザー自体の振動周波数を調整し、"ド、レ、ミ、ファ、ソ、ラ、シ"などの異なる音階を出すことができます。

さあ、パッシブブザーにメロディーを鳴らしてみましょう！

- **ブザー**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

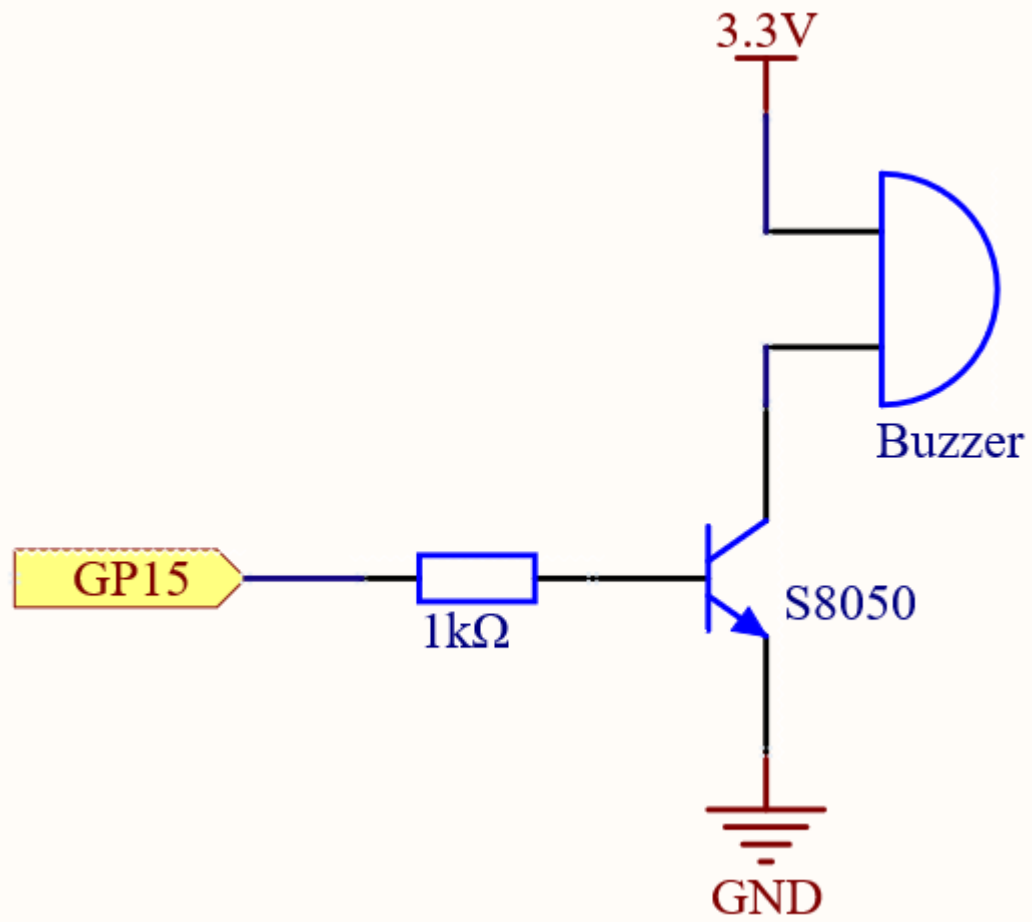
一式をまとめて購入するのが便利です、そのためのリンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

下記のリンクから個別にも購入できます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	パッシブ **ブザー**	1	

回路図



GP15 の出力が高いと、1K の電流制限抵抗を通過した後で、S8050（NPN トランジスター）が導通し、ブザーが鳴

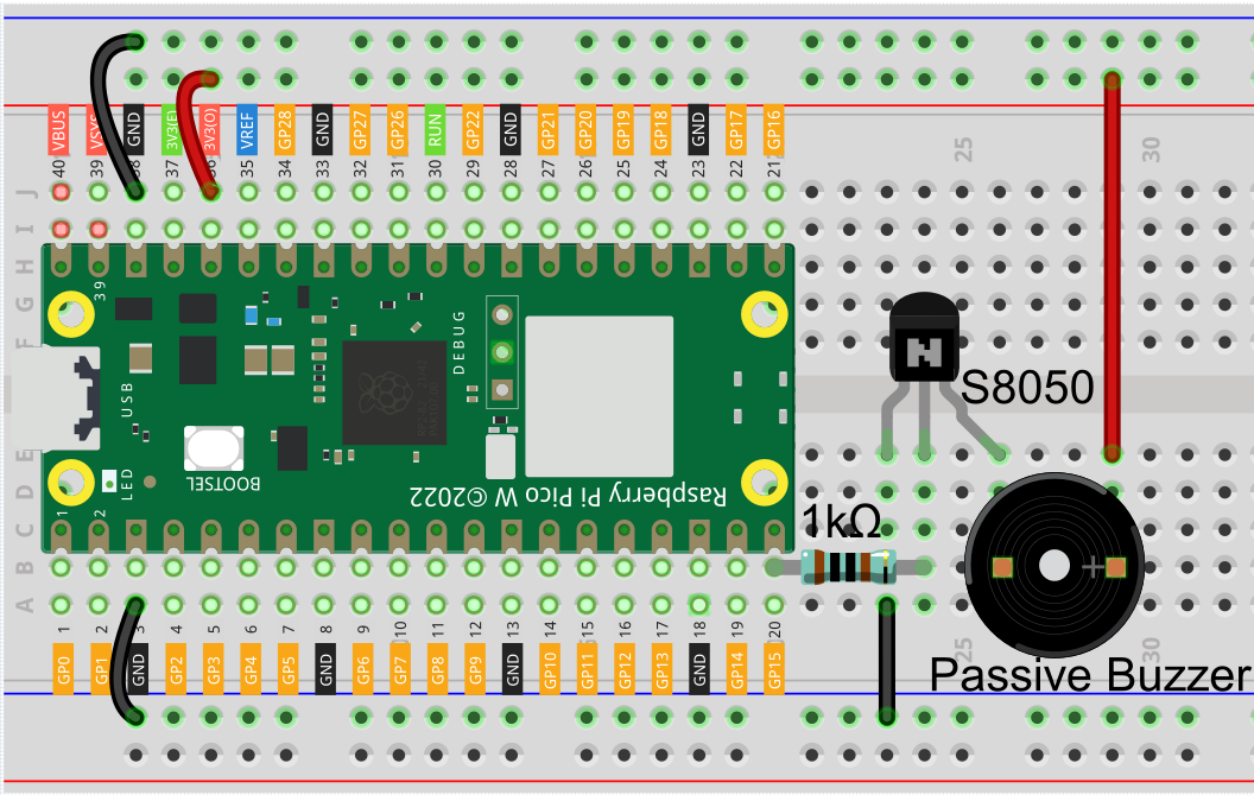
ります。

S8050（NPN トランジスター）の役割は、電流を増幅してブザーの音を大きくすることです。実際には、GP15 に直接ブザーを接続することもできますが、ブザーの音が小さいことに気付くでしょう。

配線

このキットには 2 つのブザーが含まれていますが、ここではパッシブブザー（背面に露出した PCB があるもの）を使用します。

ブザーは動作するためにトランジスタが必要です、ここでは S8050 を使用します。



コード

注釈:

- ファイル `3.2_custom_tone.ino` は、 `kepler-kit-main/arduino/3.2_custom_tone` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

動作の仕組み

パッシブブザーにデジタル信号を与えると、振動板を押し出すだけで音を出すことはありません。

そのため、 `tone()` 関数を使用して PWM 信号を生成し、パッシブブザーに音を出させます。

この関数には 3 つのパラメーターがあります：

- **pin** ：ブザーを制御する GPIO ピン。
- **frequency** ：ブザーの音程は周波数で決まります。周波数が高いほど音程も高くなります。
- **Duration** ：音の持続時間。
- tone

さらに学ぶ

ピアノの基本周波数に応じて特定の音を模倣し、完全な曲を演奏することができます。

- Piano key frequencies - Wikipedia

注釈:

- ファイル `3.2_custom_tone_2.ino` は、`kepler-kit-main/arduino/3.2_custom_tone_2` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

6.23 3.3 WS2812 RGB ストリップ

WS2812 は、制御回路と RGB チップが 5050 コンポーネントのパッケージ内に統合されているインテリジェントな LED 光源です。この中には、インテリジェントなデジタルポートデータラッチと信号整形増幅駆動回路が内蔵されています。さらに、高精度の内部オシレーターとプログラム可能な定電流制御部も含まれており、ピクセルポイントの光色の高度な一貫性を効果的に確保します。

データ転送プロトコルは、シングル NZR 通信モードを使用します。ピクセルが電源投入リセット後、DIN ポートはコントローラからデータを受信し、最初のピクセルが初期の 24 ビットデータを収集して内部データラッチに送ります。その他のデータは、内部信号整形増幅回路によって整形され、DO ポートを介して次のカスケードピクセルに送信されます。各ピクセルの伝送後、信号は 24 ビット減少します。ピクセルは、自動的に形状を整える伝送技術を採用しており、ピクセルのカスケード数は信号伝送の速度にのみ依存しています。

- *WS2812 RGB 8 LED* **ストリップ**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

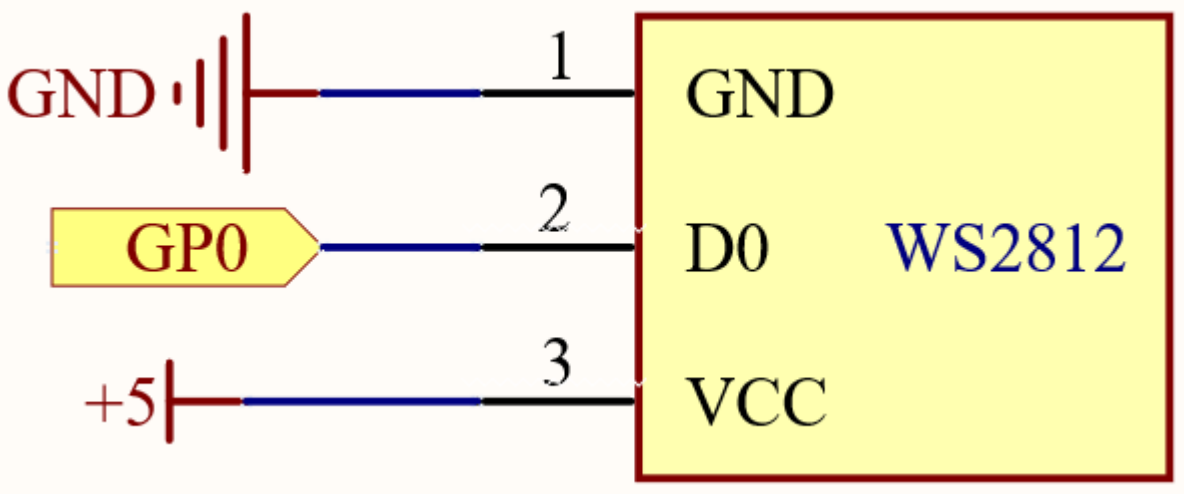
便利なのは、一式を購入することです。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

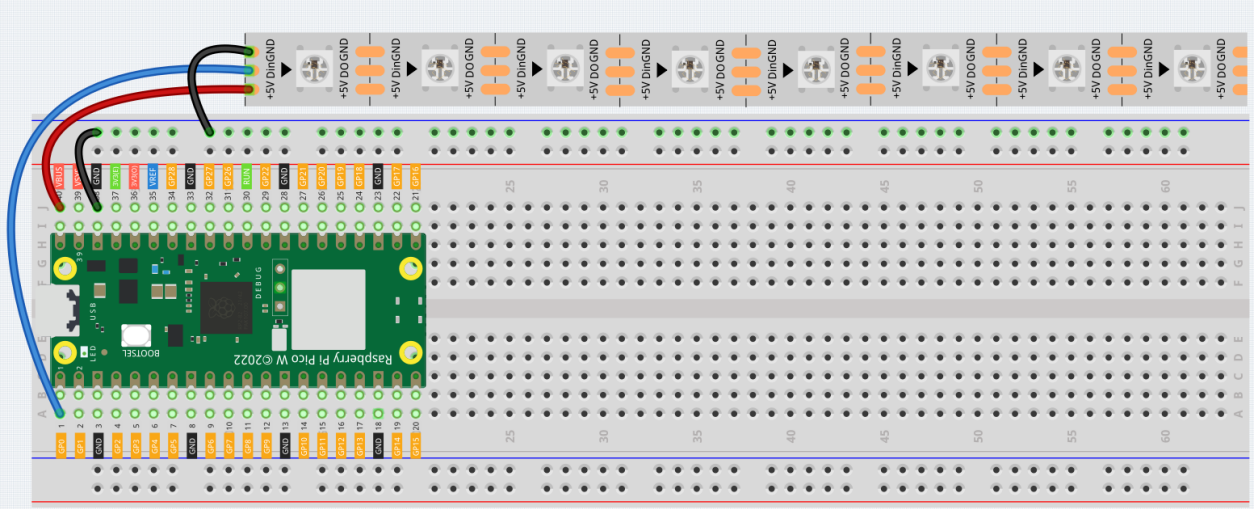
以下のリンクから個別に購入することもできます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*WS2812 RGB 8 LED* **ストリップ**	1	

回路図



配線



警告: 注目すべき一点は電流です。

Pico W で任意の数の LED を使用することは可能ですが、その VBUS ピンの電力は限られています。ここでは、安全な範囲内である 8 つの LED を使用します。ただし、より多くの LED を使用したい場合は、別の電源を追加する必要があります。

コード

注釈:

- ファイル 3.3_rgb_led_strip.ino を kepler-kit-main/arduino/3.3_rgb_led_strip のパスで開けます。

- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。
- ここではライブラリ `Adafruit_NeoPixel` を使用しています。Arduino IDE に追加する方法については、**ライブラリを追加** を参照してください。

お気に入りの色を選んで、RGB LED ストリップで表示してみましょう！

動作の仕組み

Adafruit_NeoPixel 型のオブジェクトを宣言し、 `PIXEL_PIN` に接続されています。ストリップには `PIXEL_COUNT` 個の RGB LED があります。

```
#define PIXEL_PIN    0
#define PIXEL_COUNT 8

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(PIXEL_COUNT, PIXEL_PIN, NEO_GRB + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
// Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
```

ストリップオブジェクトを初期化し、すべてのピクセルを「オフ」に設定します。

関数

- `strip.begin()`：NeoPixel ストリップオブジェクトを初期化（必須）。
- `strip.setPixelColor(index, color)`：ピクセルの色を設定（RAM 内）。 `color` は単一の' パックされた' 32 ビット値でなければなりません。
- `strip.Color(red, green, blue)`：単一の' パックされた' 32 ビット値としての色。
- `strip.show()`：新しい内容でストリップを更新。

さらに学ぶ

ランダムに色を生成し、カラフルな流れる光を作成することができます。

注釈:

- ファイル `3.3_rgb_led_strip_flowing.ino` を `kepler-kit-main/arduino/3.3_rgb_led_strip_flowing` のパスで開けます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

または、この WS2812 LED ストリップで色の輪（範囲 65535）をサイクルさせることができます。

注釈:

- ファイル `3.3_rgb_led_strip_rainbow.ino` を `kepler-kit-main/arduino/3.3_rgb_led_strip_rainbow` のパスで開けます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

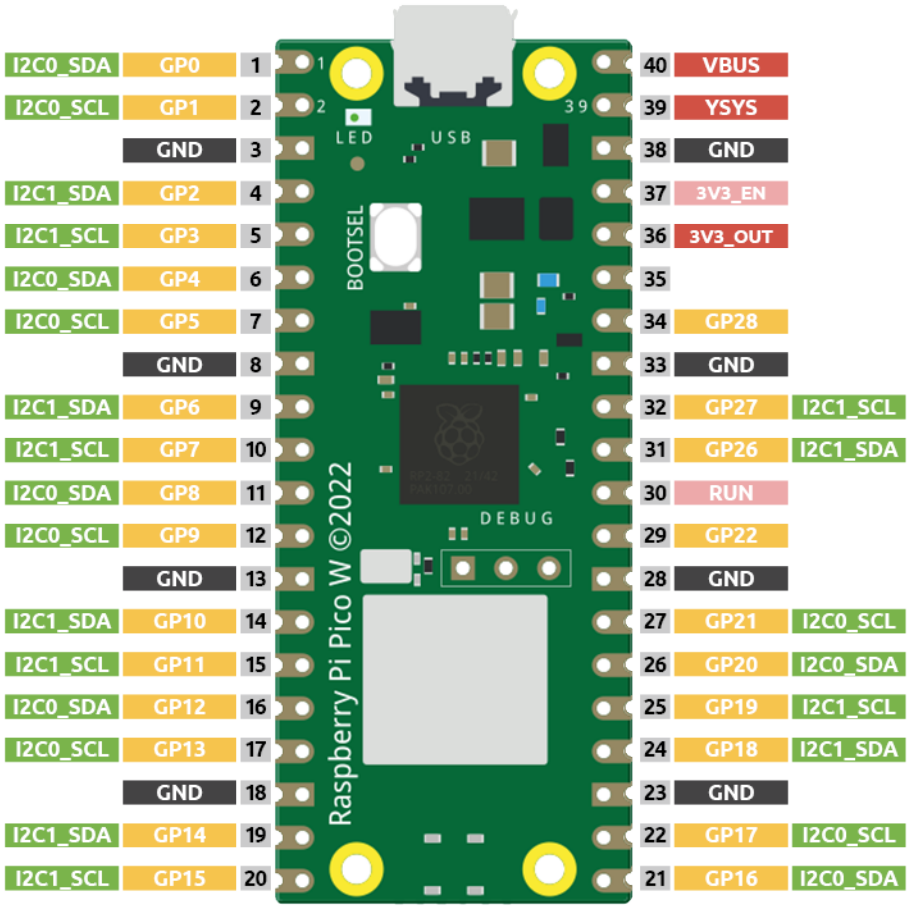
- `strip.getPixelColor(index)`：以前に設定されたピクセルの色をクエリします。
- `strip.ColorHSV(pixelHue)`：色相、彩度、明度を `setPixelColor()` または他の RGB 互換関数に渡すことができるパックされた 32 ビット RGB 色に変換します。
- `strip.gamma32()`：各ピクセルに割り当てる前に"より真実な"色を提供します。

6.24 3.4 - 液晶ディスプレイ

LCD1602 は、文字型の液晶ディスプレイで、同時に 32（16 × 2）文字を表示することができます。

ご存知のように、LCD やその他のディスプレイは人間と機械とのインタラクションを大いに豊かにしていますが、一つ共通の弱点があります。それは、コントローラーに接続すると、多数の I/O ポートを占有し、コントローラーの他の機能に制限をかけてしまうことです。そのため、この問題を解決するために I2C バスを備えた LCD1602 が開発されました。

- *I2C LCD1602*
- Inter-Integrated Circuit - Wikipedia



ここでは、I2C0 インターフェースを使用して LCD1602 を制御し、テキストを表示します。

必要な部品

このプロジェクトには、以下の部品が必要です。

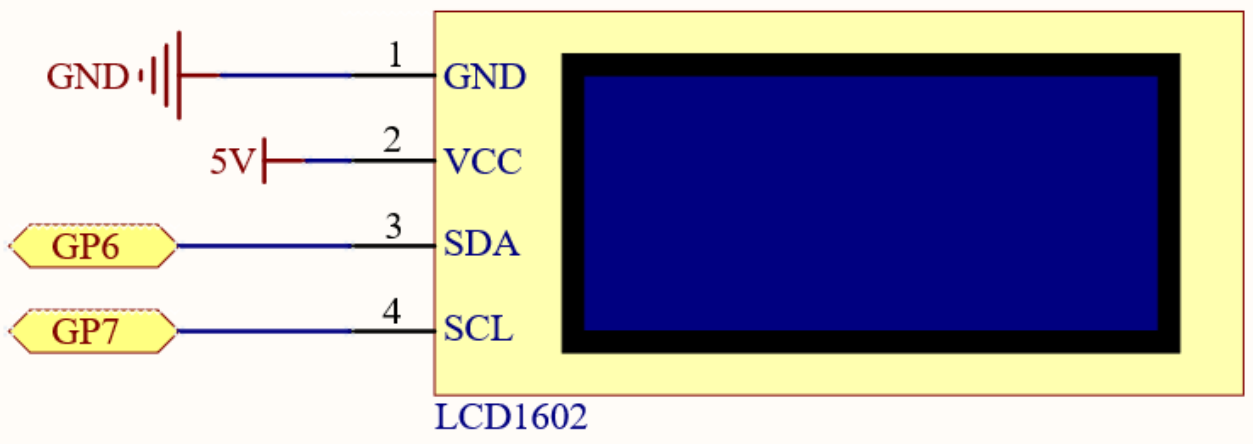
全体のキットを購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

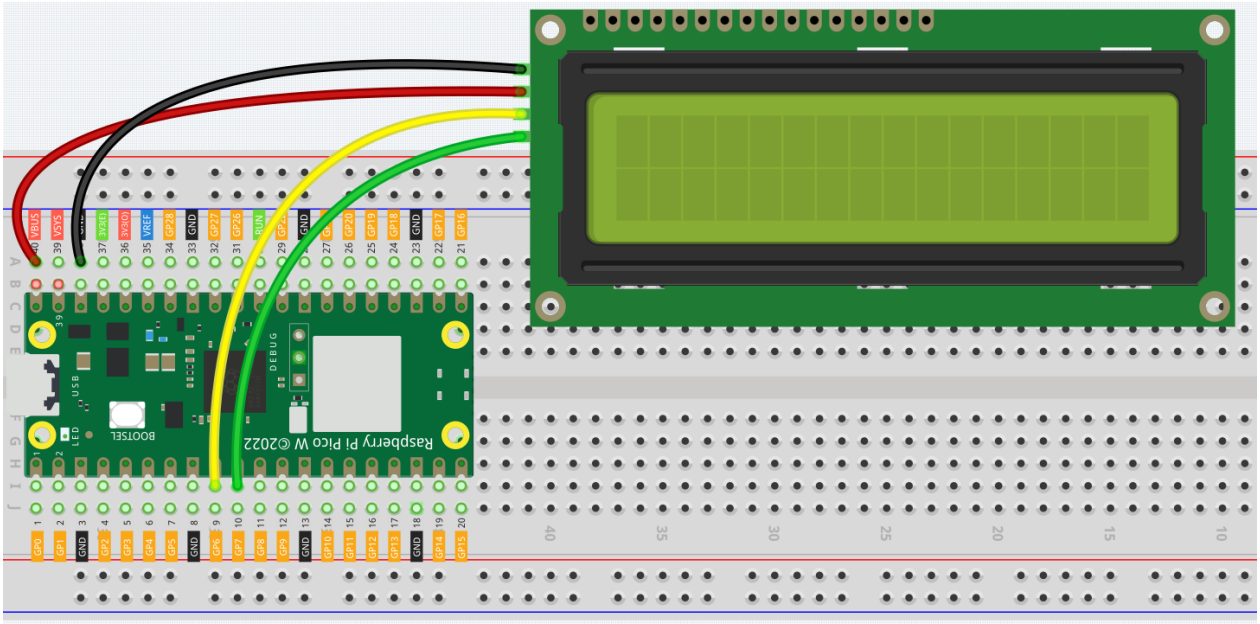
以下のリンクから個々にも購入できます。

SN	部品紹介	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*I2C LCD1602*	1	

回路図



配線



コード

注釈:

- ファイル 3.4_liquid_crystal_display.ino は、kepler-kit-main/arduino/3.4_liquid_crystal_display のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。
- ここで使用されているライブラリは LiquidCrystal_I2C です。それを Arduino IDE に追加する方法については、 **ライブラリを追加** を参照してください。

プログラムが実行された後、LCD に順番に 2 行のテキストが表示され、その後消えます。

注釈: コードと配線が正しくても、LCD が内容を表示しない場合は、背面のポテンショメータを回してコントラストを上げてみてください。

どのように動作するか？

ライブラリ LiquidCrystal_I2C.h を呼び出すことで、LCD を簡単に制御できます。

```
#include "LiquidCrystal_I2C.h"
```

ライブラリ関数

```
LiquidCrystal_I2C(uint8_t lcd_Addr,uint8_t lcd_cols,uint8_t lcd_rows)
```

Arduino ボードに接続された特定の LCD を表す LiquidCrystal_I2C クラスの新しいインスタンスを作成します。

- **lcd_Addr** : LCD のアドレスはデフォルトで 0x27 です。
- **lcd_cols** : LCD1602 は 16 列です。
- **lcd_rows** : LCD1602 は 2 行です。

```
void init()
```

LCD を初期化します。

```
void backlight()
```

（オプションの）バックライトをオンにします。

```
void nobacklight()
```

（オプションの）バックライトをオフにします。

```
void display()
```

LCD ディスプレイをオンにします。

```
void nodisplay()
```

LCD ディスプレイを素早くオフにします。

```
void clear()
```

ディスプレイをクリアし、カーソル位置をゼロに設定します。

```
void setCursor(uint8_t col,uint8_t row)
```

カーソル位置を col,row に設定します。

```
void print(data,BASE)
```

テキストを LCD に出力します。

- **data**: 出力するデータ（char、byte、int、long、または string）。
- **BASE（オプション）**：数値を出力する際の基数：BIN（2 進数）、DEC（10 進数）、OCT（8 進数）、HEX（16 進数）。

詳しくは

Pico W にコードをアップロードすると、シリアルモニターで入力した内容が LCD に表示されます。

注釈:

- ファイル `3.4_liquid_crystal_display_2.ino` は、`kepler-kit-main/arduino/3.4_liquid_crystal_display_2` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。

Pico W は、電子部品からのデータを読み取るだけでなく、シリアルポートモニターで入力されたデータも読み取れます。そのため、 `Serial.read()` を回路実験のコントローラーとして使用できます。

setup() でシリアル通信を実行し、データレートを 9600 に設定します。

```
Serial.begin(9600);
```

loop() でシリアルポートモニターの状態を判断し、データが受信された場合のみ情報処理が行われます。

```
if (Serial.available() > 0){}
```

画面をクリアします。

```
lcd.clear();
```

シリアルポートモニターで入力値を読み取り、それを変数 incomingByte に格納します。

```
char incomingByte = Serial.read();
```

各文字を LCD に表示し、改行文字はスキップします。

```
while (Serial.available() > 0) {
    char incomingByte=Serial.read();
    if(incomingByte==10){break;}// 改行文字をスキップ
    lcd.print(incomingByte);// 各文字を LCD に表示
}
```

- Serial Read

6.25 3.5 - 小型ファン

今回は TA6586 を用いて、DC モーターを時計回りと反時計回りに回転させます。DC モーターは比較的大きな電流を必要とするため、安全上の理由で、ここでは電源モジュールを使ってモーターに電力を供給します。

- *DC* モーター
- *TA6586* - モータードライバーチップ
- cpn_power_module

必要な部品

このプロジェクトでは、以下のコンポーネントが必要です。

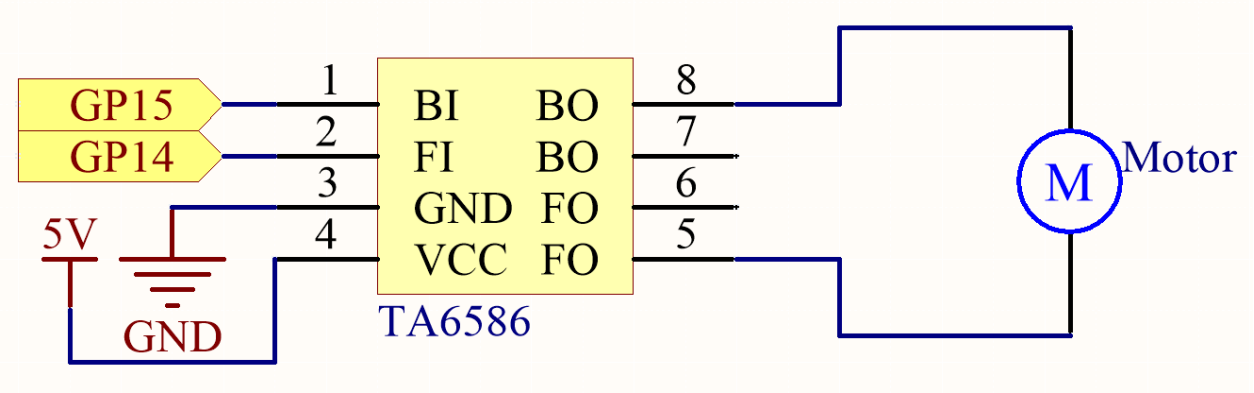
一式を購入することは非常に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450+	

以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*TA6586* - **モータードライバーチップ**	1	
6	*DC* **モーター**	1	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	

回路図

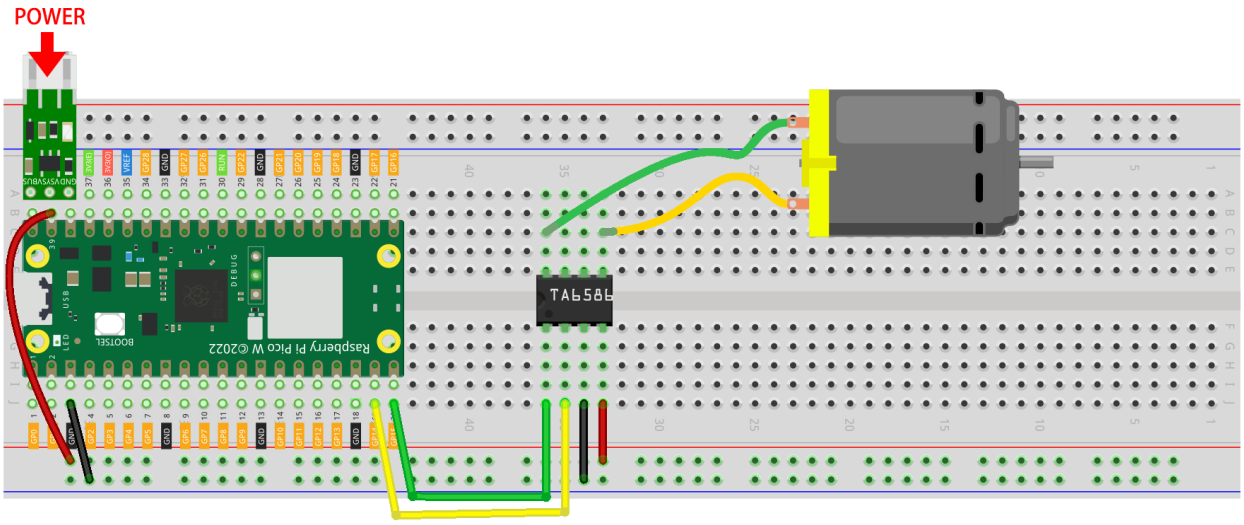


配線

注釈:

- DC モーターは高電流を必要とするため、安全のためにここでは Li-po チャージャーモジュールを使用してモーターに電力を供給します。

- 図に示されているように Li-po チャージャーモジュールが接続されていることを確認してください。そうでなければ、短絡が起きる可能性があり、バッテリーと回路が損傷する可能性があります。



コード

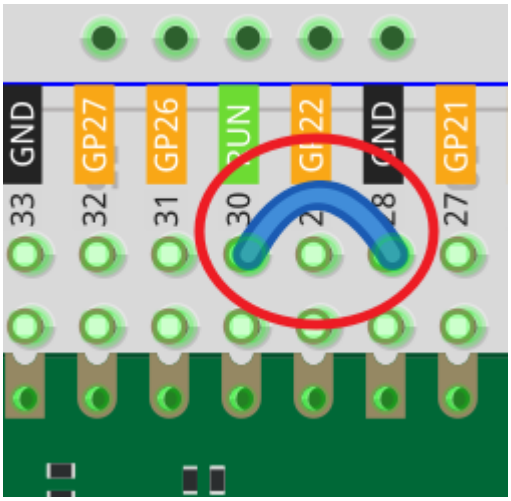
注釈:

- ファイル `3.5_small_fan.ino` は、 `kepler-kit-main/arduino/3.5_small_fan` のパスの下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

プログラムが実行されると、モーターは規則的なパターンで前後に回転します。

注釈:

- もしコードを再度アップロードできない場合は、今回は Pico W の **RUN** ピンを GND にワイヤで接続してリセットし、その後このワイヤを抜いてコードを再実行してください。
- これは、モーターが大量の電流を使用しているため、Pico W がコンピュータから切断される可能性があるからです。



6.26 3.6 - ポンピング

小型の遠心ポンプは、自動植物水やりのプロジェクトに適しています。また、小さなスマートな水の仕掛けを作るためにも使用できます。

動力源は電動モーターであり、通常のモーターとまったく同じ方法で駆動されます。

- *DC* **ウォーターポンプ**
- *DC* **モーター**
- *TA6586* - **モータードライバーチップ**
- cpn_power_module

注釈:

1. モーターの出口にチューブを接続し、ポンプを水に浸し、電源を入れます。
2. 水位が常にモーターより高いことを確認してください。アイドリングは、発熱と騒音を発生させ、モーターに損傷を与える可能性があります。
3. 植物に水をやる場合、土が吸い込まれないように注意する必要があります。それがポンプを詰まらせる可能性があります。
4. チューブから水が出ない場合、チューブ内に残留水があり、空気の流れを妨げている可能性があります。まずそれを排水する必要があります。

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

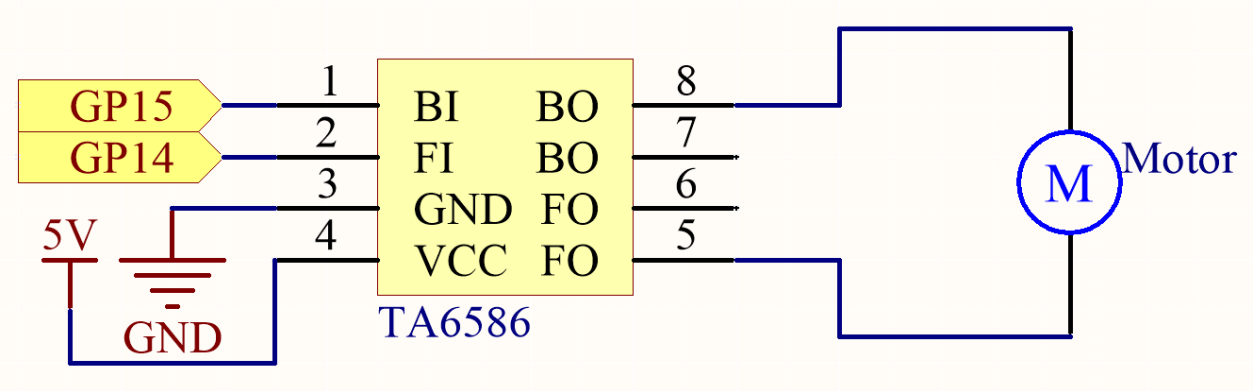
全体のキットを購入すると非常に便利です。以下にリンクを示します：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

番号	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*TA6586* - **モータードライバーチップ**	1	
6	*Li-po* **充電モジュール**	1	
7	18650 バッテリー	1	
8	バッテリーホルダー	1	
9	*DC* **ウォーターポンプ**	1	

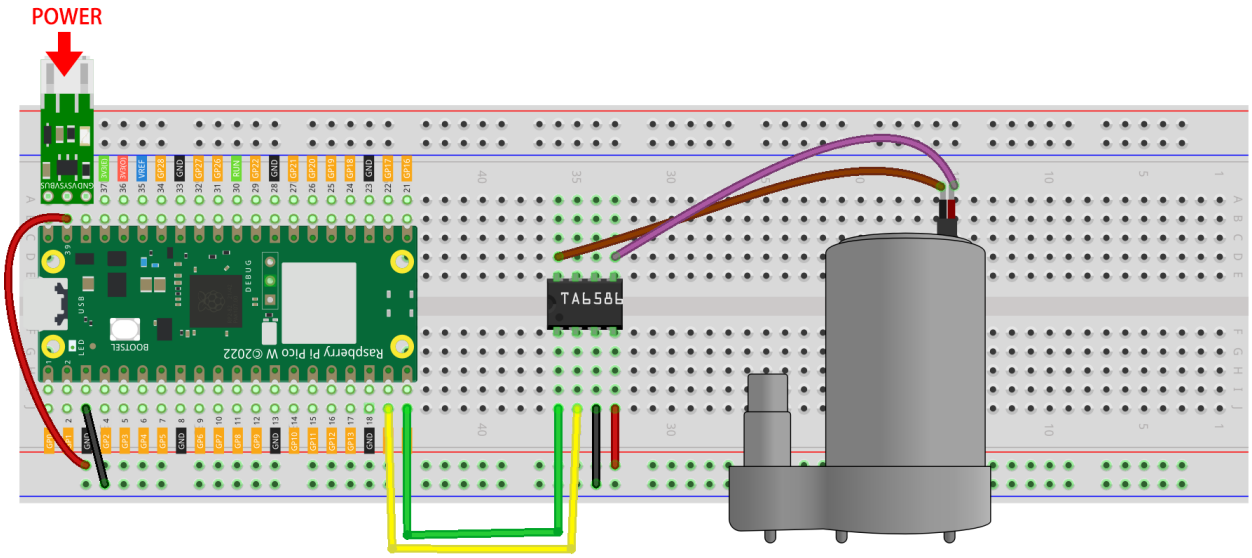
回路図



配線

注釈:

- ポンプは高電流を必要とするため、安全上の理由でここでは Li-po チャージャーモジュールを使用してモーターに電力を供給します。
- Li-po チャージャーモジュールが図に示されているように接続されていることを確認してください。そうでない場合、短絡が発生し、バッテリーと回路が損傷する可能性があります。



コード

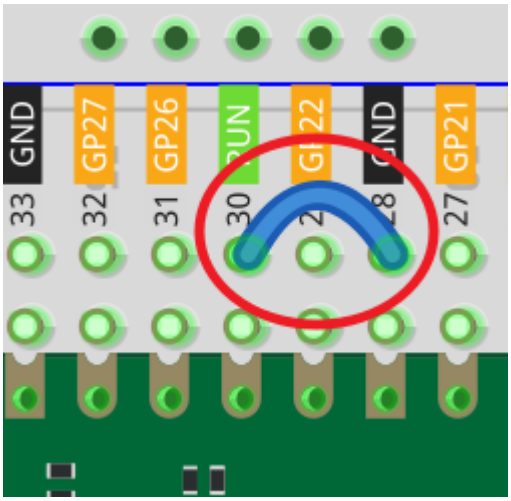
注釈:

- ファイル `3.6_pumping.ino` は、パス `kepler-kit-main/arduino/3.6_pumping` の下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **Upload** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。

コードが実行された後、ポンプが動作を開始し、同時にチューブから水が流れ出ます。

注釈:

- コードを再度アップロードできない場合、この時は Pico W の **RUN** ピンを GND にワイヤーで接続してリセットし、その後このワイヤーを抜いてコードを再度実行します。
- これは、モーターが過度な電流で動作しているため、Pico W がコンピュータから切断される可能性があるからです。



6.27 3.7 - サーボの揺れ動き

このキットには、LED やパッシブブザーに加えて、PWM 信号で制御されるデバイス、サーボも含まれています。

サーボは位置（角度）制御用のデバイスで、一定の角度の変更が必要な制御システムに適しています。飛行機、潜水艦の模型、リモコンロボットなどの高級リモコン玩具で広く使用されています。

さあ、サーボを揺らしてみましょう！

- **サーボ**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

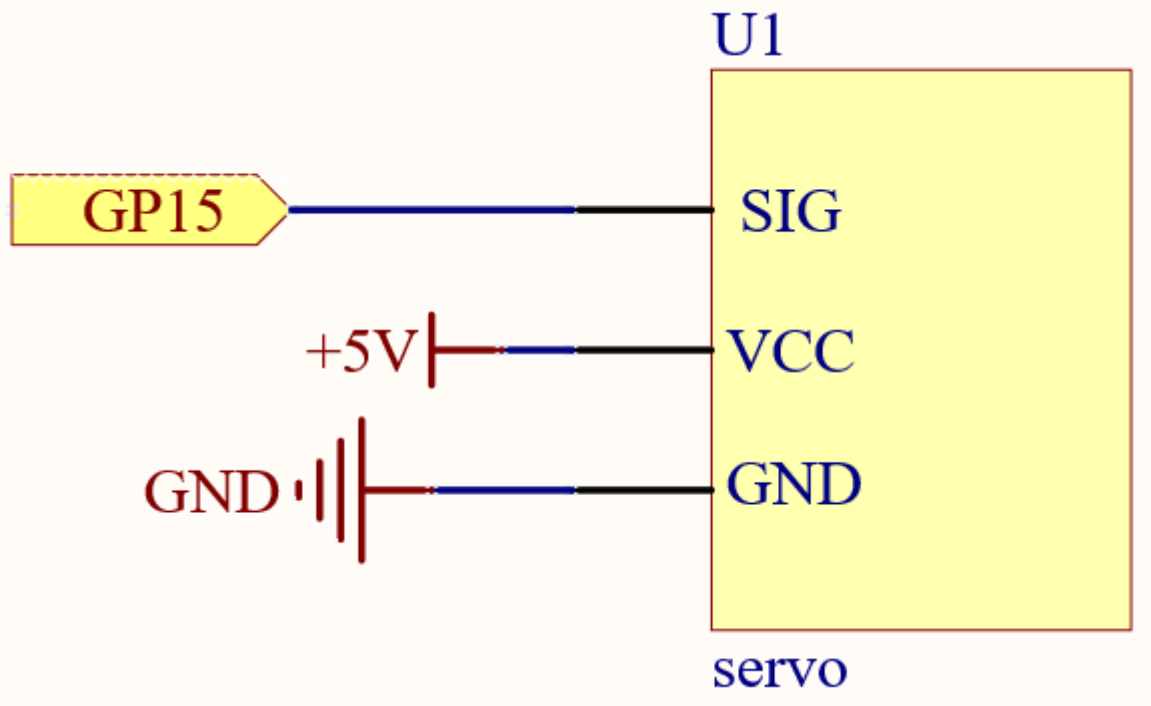
全体のキットを購入すると非常に便利です。リンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

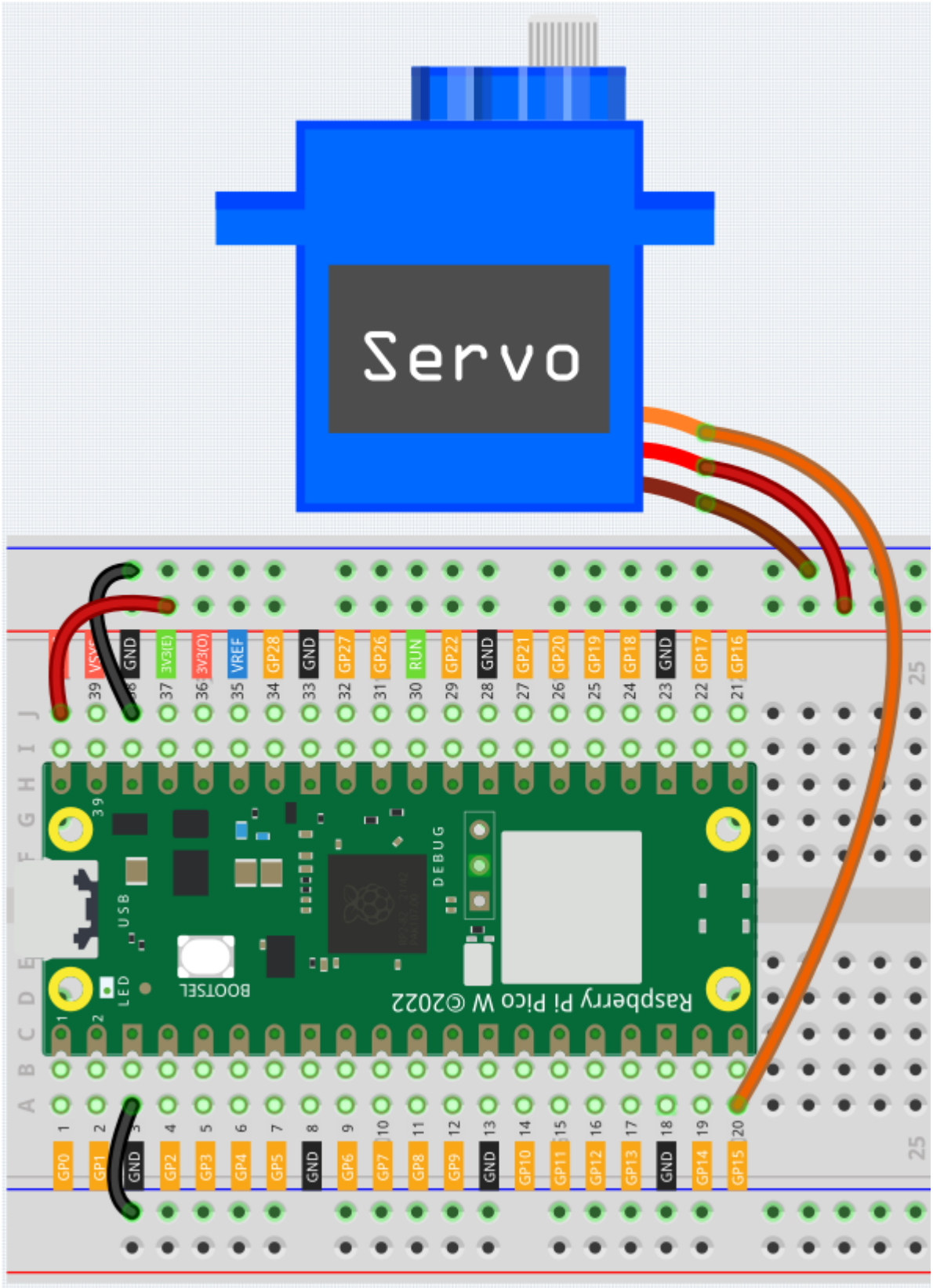
以下のリンクから個別に購入することもできます。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	

回路図



配線



- オレンジ色のワイヤーは信号で、GP15 に接続されています。

- 赤色のワイヤーは VCC で、VBUS(5V) に接続されています。
- 茶色のワイヤーは GND で、GND に接続されています。

コード

注釈:

- ファイル `3.7_swinging_servo.ino` は、パス `kepler-kit-main/arduino/3.7_swinging_servo` で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。

プログラムが実行されていると、サーボアームが 0 ° から 180 ° まで前後に揺れ動くのが見られます。

どうやって動くのか？

Servo.h ライブラリを呼び出すことで、簡単にサーボを制御できます。

```
#include <Servo.h>
```

ライブラリ関数

```
Servo
```

サーボを制御するための **Servo** オブジェクトを作成。

```
uint8_t attach(int pin);
```

ピンをサーボドライバーに変換。pinMode を呼び出す。失敗時は 0 を返す。

```
void detach();
```

サーボドライブからピンを解放。

```
void write(int value);
```

サーボの角度を度で設定、0 から 180。

```
int read();
```

最後の write() で設定した値を返す。

```
bool attached();
```

サーボが現在接続されている場合は 1 を返す。

4. コントローラー

6.28 4.1 - ジョイスティックの切り替え

ビデオゲームをよくプレイするなら、ジョイスティックに非常に慣れているはずです。通常、キャラクターを動かしたり、画面を回転させたりするために使用されます。

ジョイスティックがコンピュータに私たちの行動を読み取らせる仕組みは非常にシンプルです。これは、直交する 2 つのポテンショメーターで構成されていると考えることができます。これら 2 つのポテンショメーターは、ジョイスティックの垂直および水平のアナログ値を測定し、平面直角座標系での値（x,y）を生成します。

このキットのジョイスティックには、ジョイスティックが押されたときに作動するデジタル入力もあります。

- **ジョイスティックモジュール**

必要な部品

このプロジェクトでは、以下の部品が必要です。

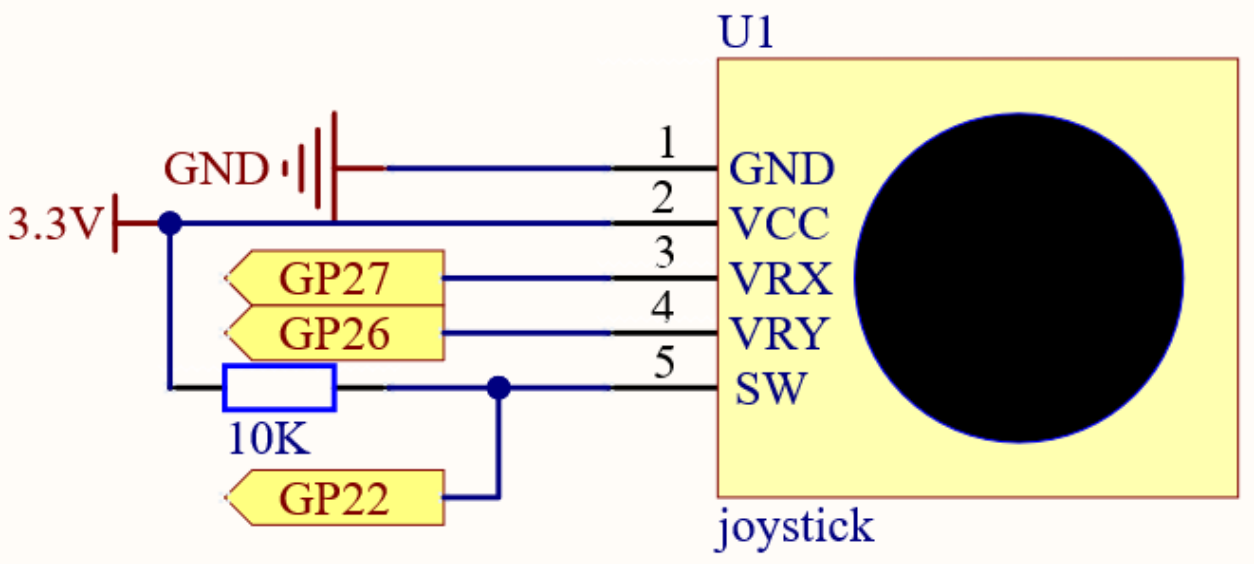
便利なのは、一式をまとめて購入することです。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

以下のリンクから個々にも購入できます。

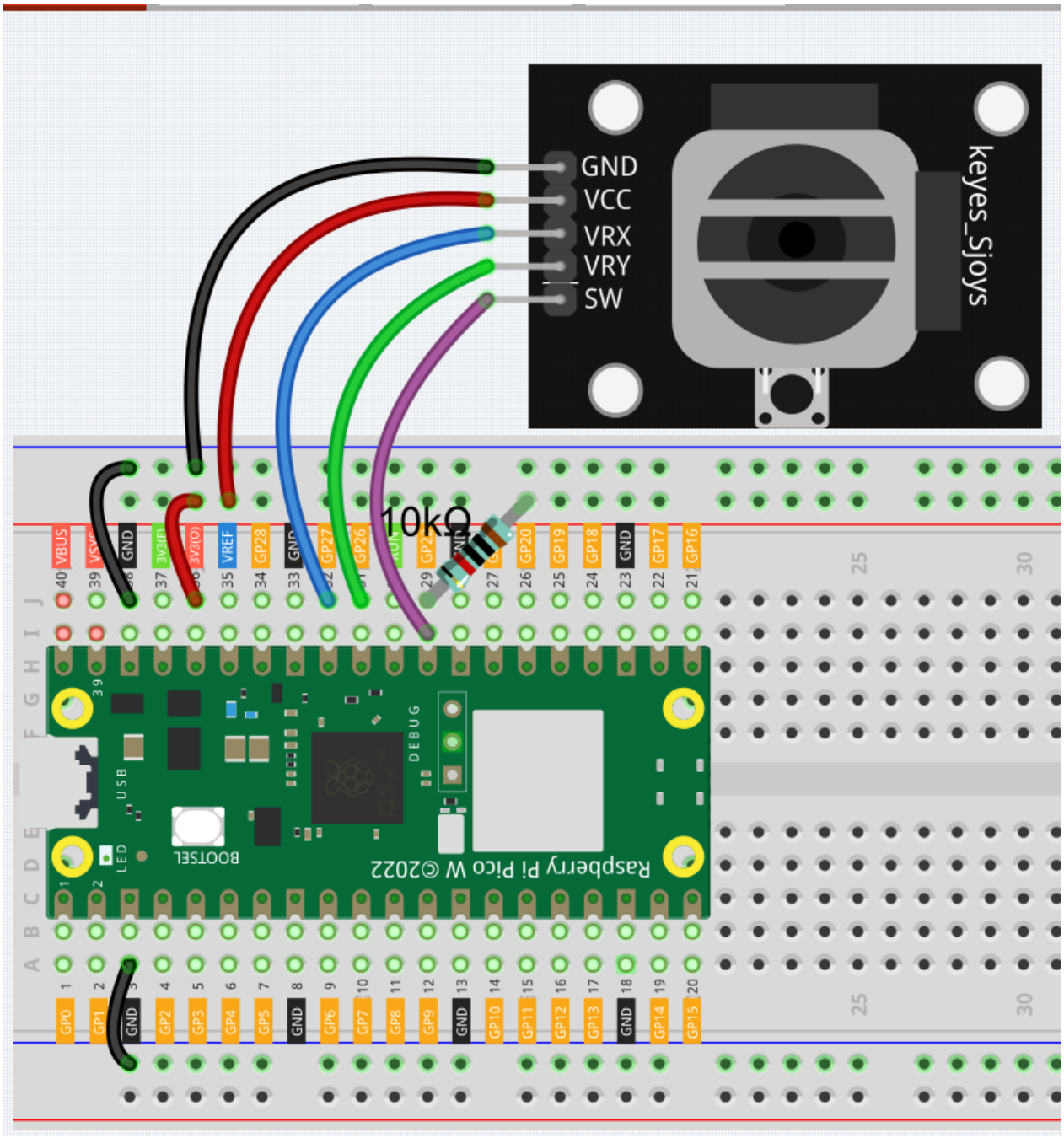
SN	部品紹介	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**ジョイスティックモジュール**	1	

回路図



SW ピンは 10K のプルアップ抵抗に接続されています。これは、ジョイスティックが押されていないときに SW ピン（Z 軸）で安定した高レベルを得るためです。それ以外の場合、SW は一時停止状態にあり、出力値は 0/1 の間で変動する可能性があります。

配線



コード

注釈:

- ファイル 4.1_toggle_the_joyostick.ino は、kepler-kit-main/arduino/4.1_toggle_the_joyostick のパスで開くことができます。
- またはこのコードを **Arduino IDE** にコピーペーストしてください。

- **アップロード** ボタンを押す前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。

プログラムが実行された後、シェルはジョイスティックの x,y,z 値を出力します。

- x 軸と y 軸の値は 0 から 65535 まで変動するアナログ値です。
- Z 軸は、状態が 1 または 0 のデジタル値です。

6.29 4.2 4x4 キーパッド

4x4 キーボード、またはマトリックスキーボードは、一つのパネル内で排除された 16 個のキーのマトリックスです。

キーパッドは、主にデジタル入力が必要なデバイス、例えば電卓、テレビのリモートコントロール、押しボタン式の電話、自動販売機、ATM、組み合わせロック、デジタルドアロックなどで見られます。

このプロジェクトでは、押されたキーを特定し、関連するキー値を取得する方法を学びます。

- *4x4* **キーパッド**
- E.161 - Wikipedia

必要な部品

このプロジェクトには、以下の部品が必要です。

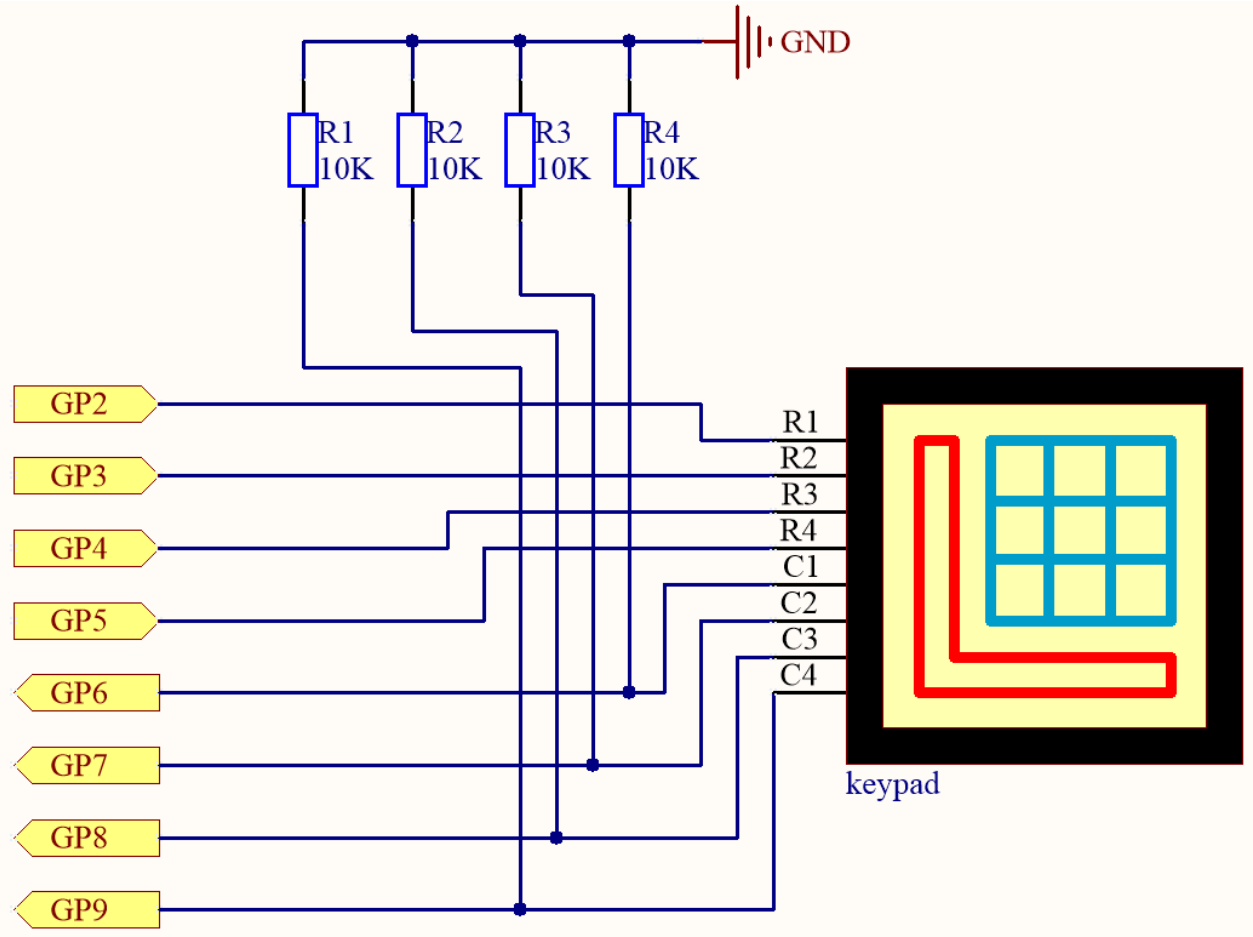
全体のキットを購入するのが便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

以下のリンクから個々にも購入できます。

SN	部品紹介	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(10K Ω)	
6	*4x4* **キーパッド**	1	

回路図

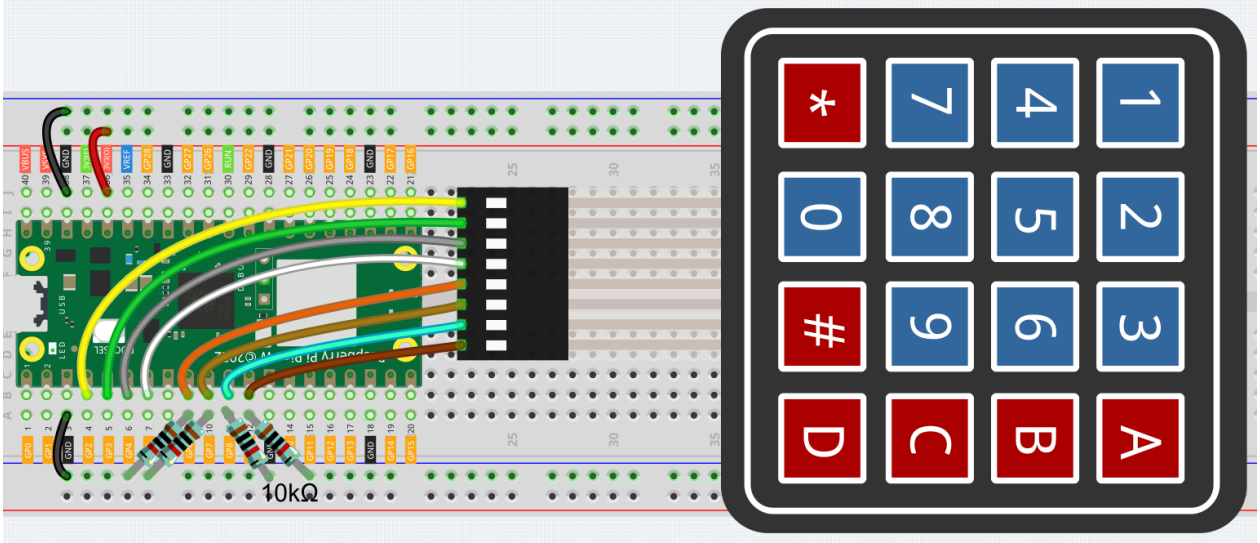


4 つのプルダウン抵抗がマトリックスキーボードの各列に接続されています。これにより、キーが押されていないときに G6～G9 が安定したローレベルを取得します。

キーボードの行（G2〜G5）は、高い状態にプログラムされています。G6〜G9 のうちの 1 つが高い状態で読み取られた場合、どのキーが押されたかを知ることができます。

例えば、G6 が高い状態で読み取られた場合、数字キー 1 が押されています。これは、数字キー 1 の制御ピンが G2 と G6 であり、数字キー 1 が押されたときに G2 と G6 が接続され、G6 も高い状態になるためです。

配線



配線を簡単にするために、上記の図では、マトリックスキーボードの列と 10K 抵抗が、同時に G6〜G9 の位置にある穴に挿入されています。

コード

注釈:

- ファイル `4.2_4x4_keypad.ino` は、 `kepler-kit-main/arduino/4.2_4x4_keypad` のパスで開くことができます。
- またはこのコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。
- ここで使われるライブラリは Keypad です。それを Arduino IDE に追加する方法については、 **ライブラリを追加** を参照してください。

プログラムが実行された後、シェルはキーパッドで押したキーを出力します。

仕組み

`Keypad.h` ライブラリを呼び出すことで、簡単にキーパッドを使用できます。

```
#include <Keypad.h>
```

ライブラリ関数：

```
Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols)
```

内部のキーマップを userKeymap と同じに初期化します。

userKeymap ：キーパッドのボタン上のシンボル。

row , col ：ピン設定。

numRows , numCols ：キーパッドのサイズ。

```
char getKey()
```

押されているキーを返します（あれば）。この関数は非ブロッキングです。

6.30 4.3 - 電極キーボード

多数のタッチスイッチをプロジェクトに追加したい場合、MPR121 は優れた選択です。このモジュールは、導体で拡張可能な電極を備えています。電極をバナナに接続すると、そのバナナをタッチスイッチに変えることができます。

- *MPR121* **モジュール**

必要な部品

このプロジェクトでは、以下のコンポーネントが必要です。

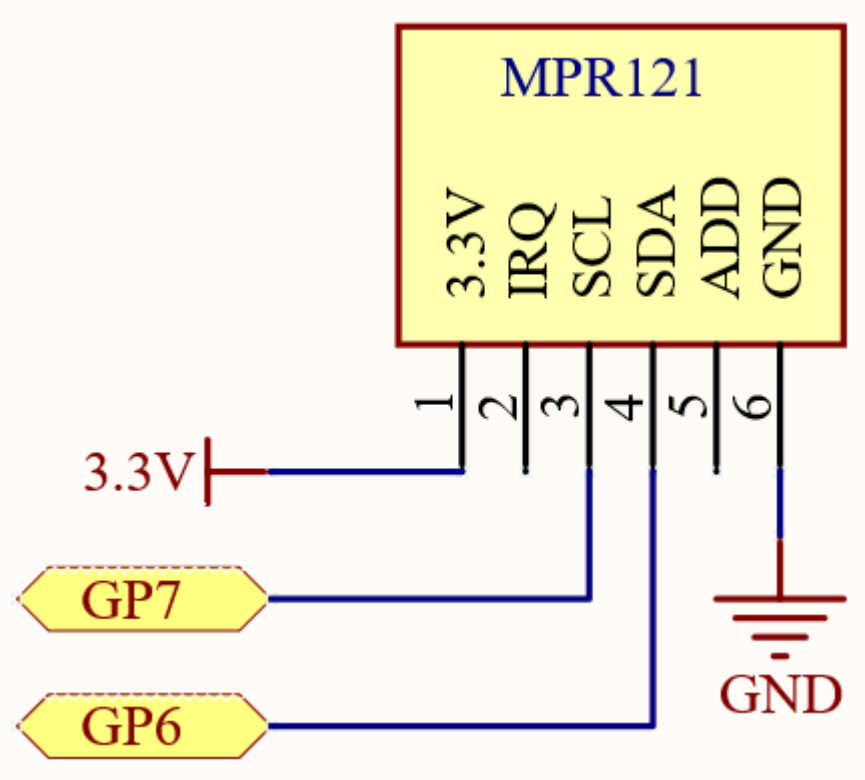
一式を購入することは非常に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450+	

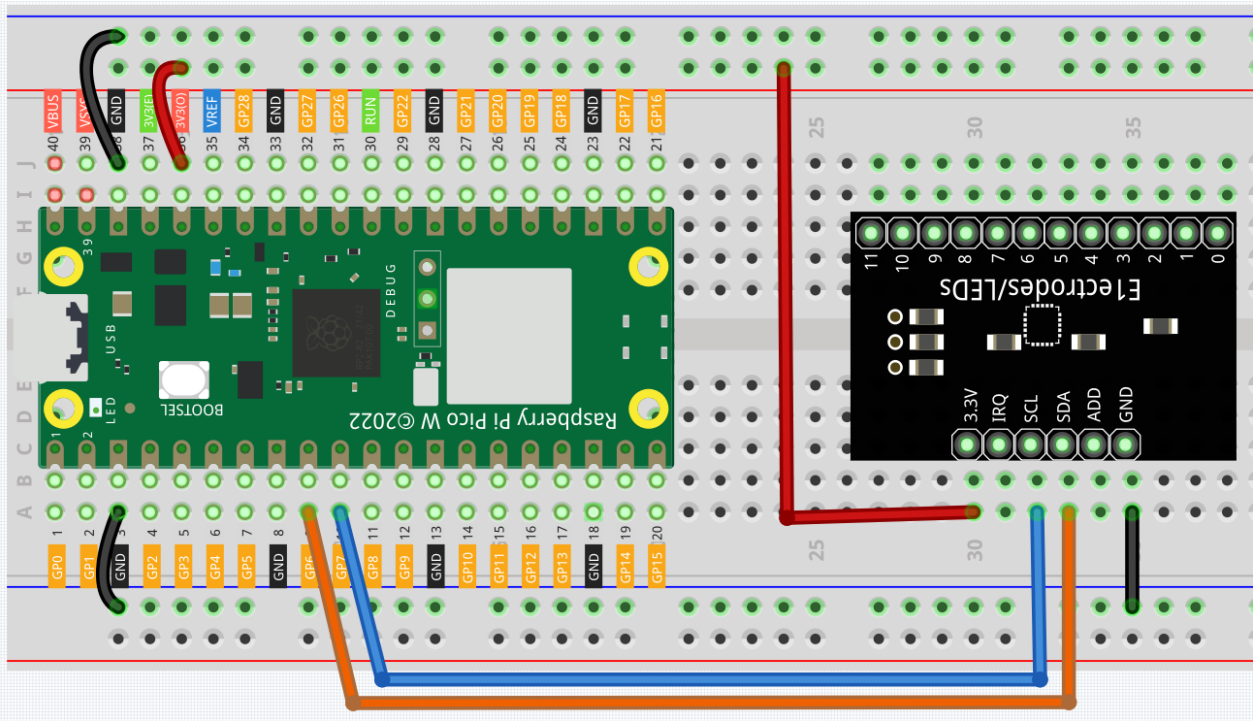
以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MPR121* **モジュール**	1	

回路図



配線



コード

注釈:

- ファイル 4.3_electrode_keyboard.ino は、パス kepler-kit-main/arduino/4.3_electrode_keyboard で開くことができます。
- または、このコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。
- ここで使用されている二つのライブラリは `Adafruit_MPR121` と `Adafruit_BusIO` です。Arduino IDE に追加する方法は、 **ライブラリを追加** を参照してください。

プログラムが動作すると、MPR121 モジュール上の 12 個の電極に手で触れ、それらの電極のタッチ状態が 12 ビットのブーリアン型配列に記録され、シリアルモニターに出力されます。最初と 11 番目の電極に触れると、`100000000010` が出力されます。

電極を果物、ワイヤー、箔などの他の導体に接続して拡張することができます。これにより、これらの電極をトリガーする多くの方法が得られます。

動作原理

`MPR121` オブジェクトを初期化します。この時点で、モジュールの電極の状態が初期値として記録されます。電極を拡張する場合は、初期値をリセットするために例を再実行する必要があります。

```
#include "Adafruit_MPR121.h"

Adafruit_MPR121 cap = Adafruit_MPR121();

void setup() {
    Serial.begin(9600);
    int check = cap.begin(0x5A);
    if (!check) {
        Serial.println("MPR121 not found, check wiring?");
        while (1);
    }
    Serial.println("MPR121 found!");
}
```

現在の電極の値を取得します。最初と 11 番目の電極に触れると、 100000000010 が取得されます。

```
// Get the currently touched pads
currtouched = cap.touched();
```

Determine if the electrode state has changed.

```
void loop() {
    currtouched = cap.touched();
    if (currtouched != lasttouched) {}

    // reset our state
    lasttouched = currtouched;
}
```

電極の状態に変更が検出された場合、 currtouched の値が touchStates[12] 配列にビットごとに格納されます。最後に、配列が出力されます。

```
if (currtouched != lasttouched) {
    for (int i = 0; i < 12; i++) {
        if (currtouched & (1 << i)) touchStates[i] = 1;
        else touchStates[i] = 0;
    }
    for (int i = 0; i < 12; i++){
        Serial.print(touchStates[i]);
    }
```

(次のページに続く)

(前のページからの続き)

```
    Serial.println();
}
```

5. マイクロチップ

6.31 5.1 マイクロチップ - 74HC595

集積回路（integrated circuit）は、回路内で「IC」という文字で表されるミニチュア電子デバイスまたはコンポーネントです。

トランジスタ、抵抗器、コンデンサ、インダクタなど、回路で必要なコンポーネントと配線を特定のプロセスで相互接続し、小型またはいくつかの小型半導体ウェハーまたは誘電体基板上に製造し、それをパッケージに封装することで、必要な回路機能を持つ微細構造になります。全てのコンポーネントは一体化されており、これにより電子部品は微細化、低消費電力、知能化、高信頼性に大きく前進しています。

集積回路の発明者は、Jack Kilby（ゲルマニウム（Ge）ベースの集積回路）と Robert Norton Noyce（シリコン（Si）ベースの集積回路）です。

このキットには、GPIO ピンの使用を大幅に節約できる IC、74HC595 が装備されています。具体的には、8 ビットの 2 進数を書き込むことで、デジタル信号出力のための 8 ピンを置き換えることができます。

- 2 進数 - Wikipedia
- *74HC595*

必要な部品

このプロジェクトでは、以下の部品が必要です。

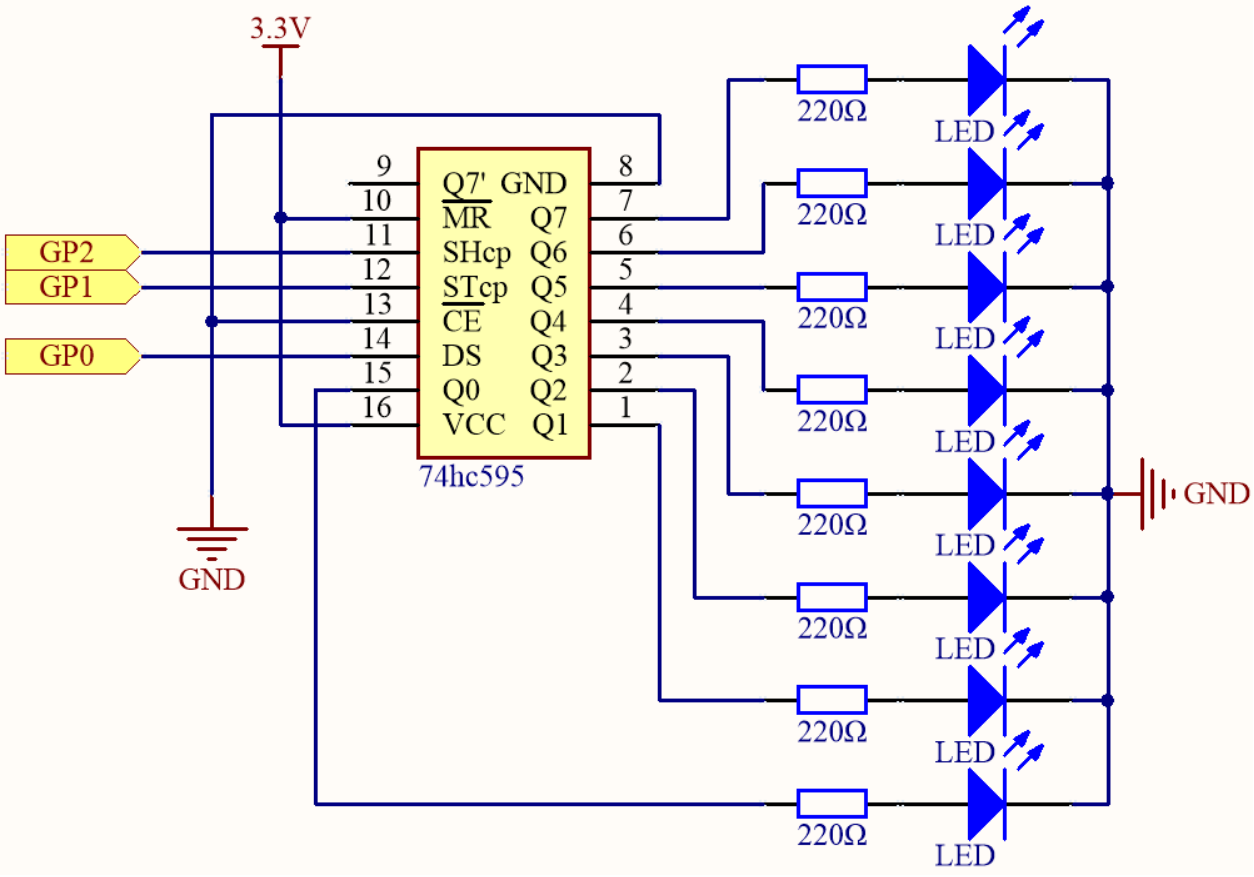
便利なのは、一式を購入することです。こちらがそのリンクです：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450 以上	

以下のリンクから個別に購入することも可能です。

SN	コンポーネントの説明	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	8（220 Ω）	
6	*LED*	8	
7	*74HC595*	1	

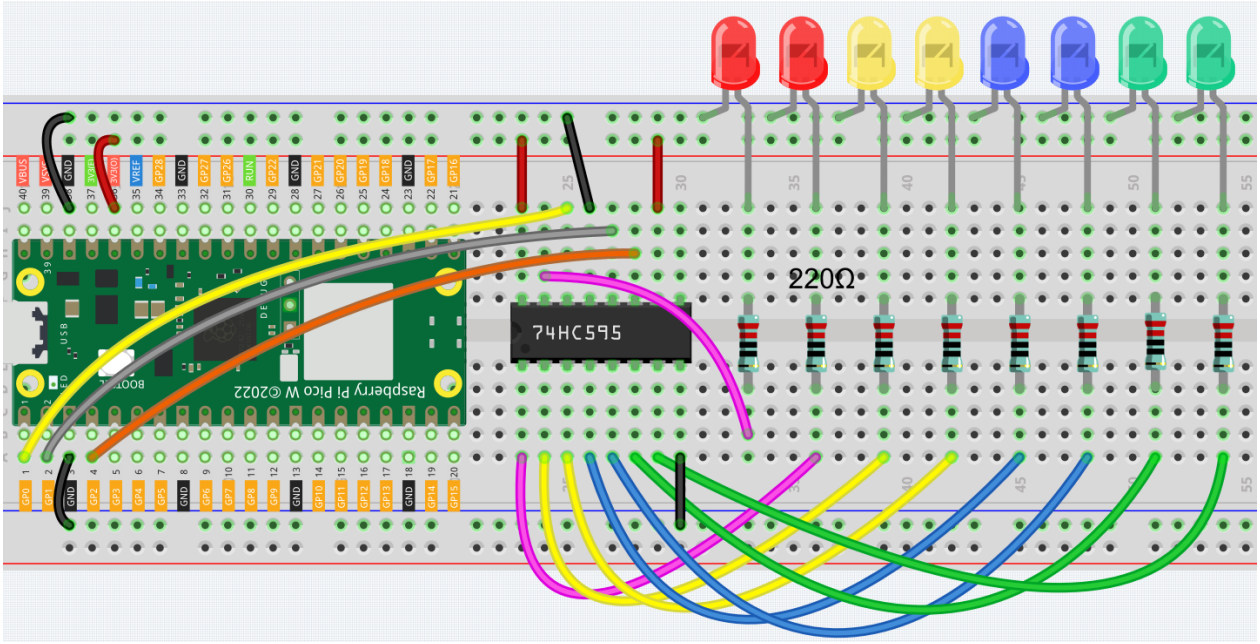
回路図



- MR（ピン 10）がハイレベルで、OE（ピン 13）がローレベルの場合、SHcp の立ち上がりエッジでデータが入力され、SHcp の立ち上がりエッジを介してメモリレジスタに移動します。

- 2 つのクロックが一緒に接続されている場合、シフトレジスタは常にメモリレジスタよりも 1 パルス早いです。
- メモリレジスタには、シリアルシフト入力ピン（Ds）、シリアル出力ピン（Q）、および非同期リセットボタン（ローレベル）があります。
- メモリレジスタは、3 つの状態で 8 ビットの並列バスを出力します。
- OE が有効（ローレベル）の場合、メモリレジスタ内のデータがバス（Q0～Q7）に出力されます。

配線



コード

注釈:

- `kepler-kit-main/arduino/5.1_microchip_74hc595` のパスの下で `5.1_microchip_74hc595.ino` ファイルを開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択することを忘れないでください。

プログラムが動作していると、LED が順番に点灯しているのが見えます。

仕組みは？

配列を宣言し、74HC595 で制御される 8 つの LED の作動状態を変更するために使用されるいくつかの 8 ビットの 2 進数を格納します。

```
int datArray[] = {0b00000000, 0b00000001, 0b00000011, 0b00000111, 0b00001111, 0b00011111,
↪ 0b00111111, 0b01111111, 0b11111111};
```

まず STcp をローレベルに設定し、次にハイレベルに設定します。これにより、 STcp の立ち上がりエッジパルスが生成されます。

```
digitalWrite(STcp,LOW);
```

shiftOut() は、1 ビットずつデータのバイトをシフトアウトするために使用されます。つまり、DS ピンを使って datArray[num] のデータバイトをシフトレジスタにシフトします。MSBFIRST は高ビットから動かすことを意味します。

```
shiftOut(DS,SHcp,MSBFIRST,datArray[num]);
```

digitalWrite(STcp,HIGH) が実行された後、STcp は立ち上がりエッジになります。この時点で、シフトレジスタ内のデータがメモリレジスタに移動します。

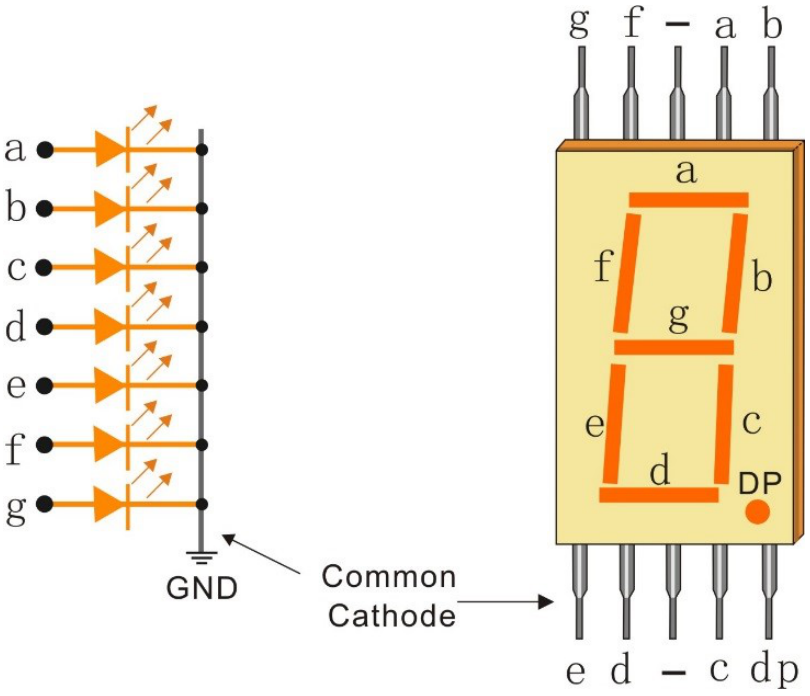
```
digitalWrite(STcp,HIGH);
```

8 回後に、1 バイトのデータがメモリレジスタに転送されます。その後、メモリレジスタのデータがバス（Q0-Q7）に出力されます。例えば、 B00000001 をシフトアウトすると、Q0 で制御される LED が点灯し、Q1～Q7 で制御される LED は消灯します。

6.32 5.2 - 数字表示

LED セグメントディスプレイは日常生活の至る所で見かけます。例えば、エアコンでは温度を表示するため、交通信号ではタイマーを表示するために使用されます。

LED セグメントディスプレイは、基本的に 8 つの LED で構成された装置です。そのうち、7 つのストリップ状の LED が「8」の形を形成し、もう一つは少し小さい点状の LED が小数点としてあります。これらの LED は a, b, c, d, e, f, g, および dp とマークされています。それぞれには独自のアノードピンがあり、カソードは共有されています。ピンの位置は以下の図で示されています。



これは、フル動作するために同時に 8 つのデジタル信号で制御する必要があることを意味し、74HC595 でこれを行うことができます。

- *7* セグメントディスプレイ

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

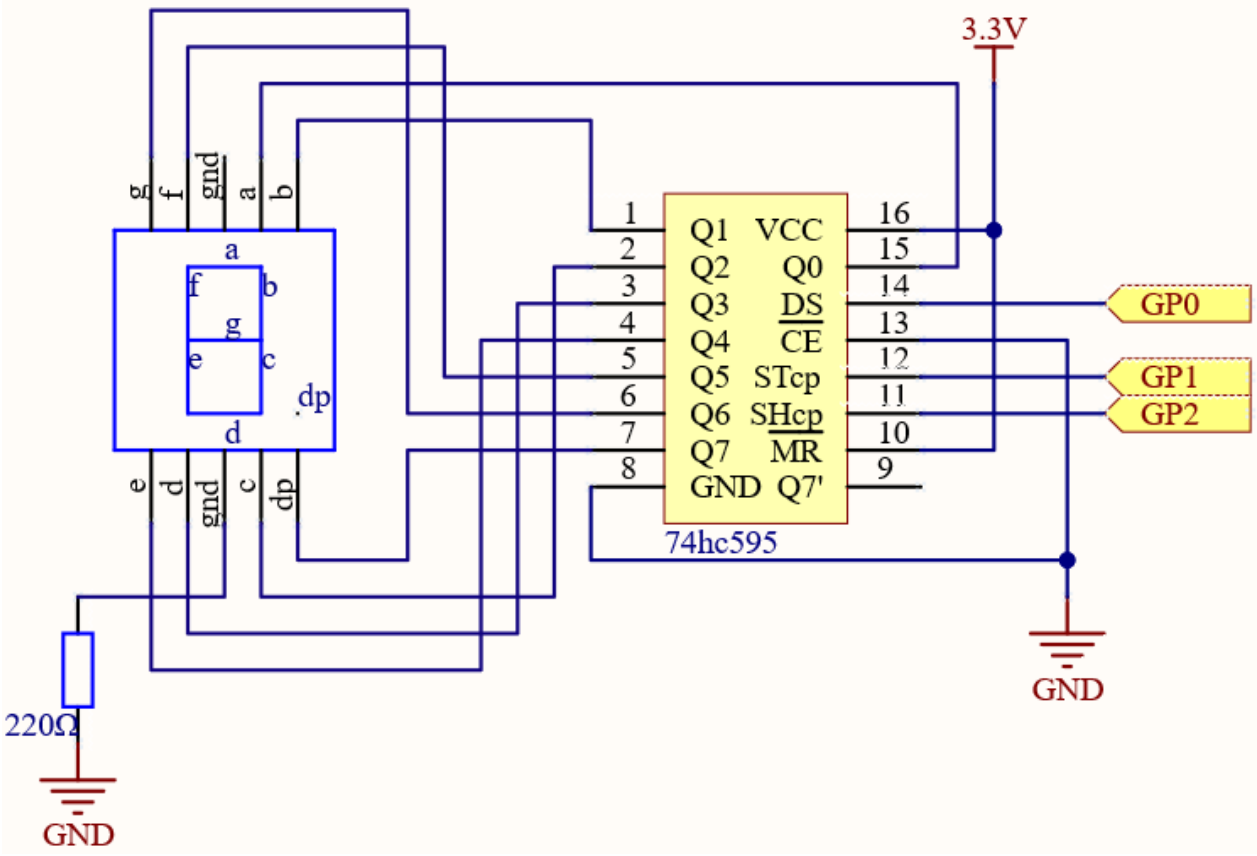
一式を購入するのが便利です、リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450+	

以下のリンクから個別に購入することもできます。

SN	コンポーネントの紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(220 Ω)	
6	*7* **セグメントディスプレイ**	1	
7	*74HC595*	1	

回路図



配線

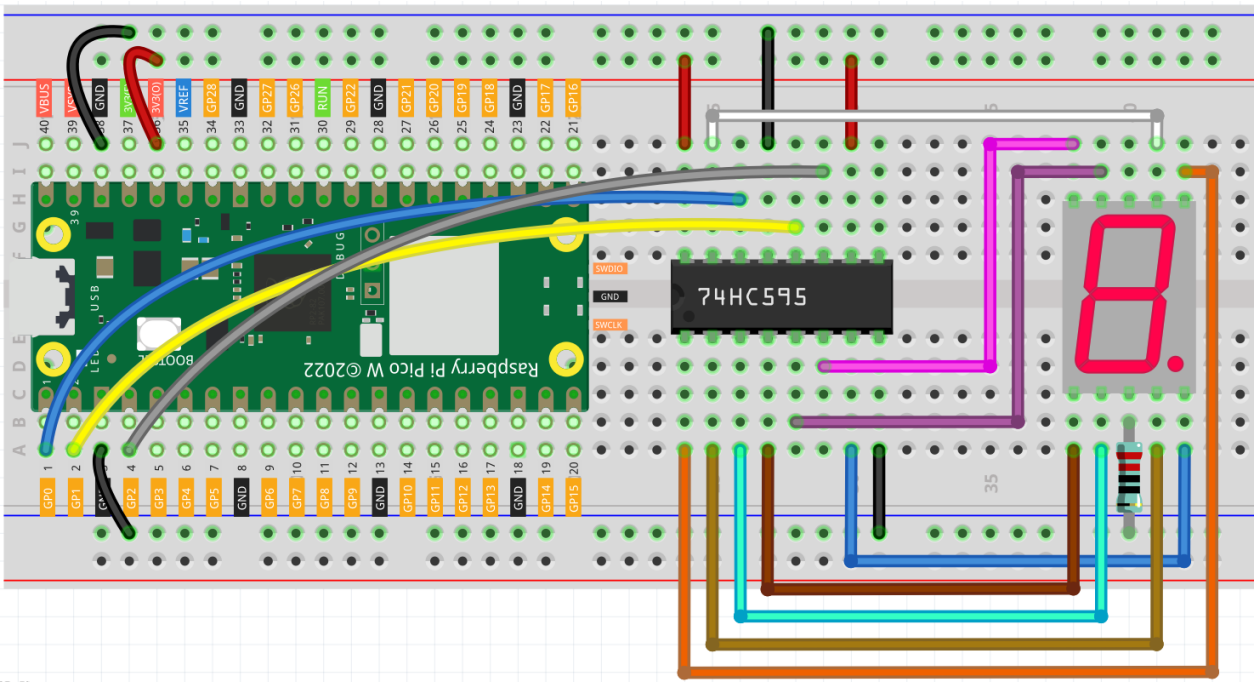


表 1　配線

74HC595	*LED* セグメントディスプレイ
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

コード

注釈:

- `kepler-kit-main/arduino/5.2_number_display` のパスの下で `5.2_number_display.ino` ファイルを開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択することを忘れないでください。

プログラムが動作していると、LED セグメントディスプレイが 0~9 までの数字を順番に表示するのが見えるでしょう。

仕組みは？

`shiftOut()` は 74HC595 に 8 つのデジタル信号を出力させます。最後のビットの二進数を Q0 に、最初のビットの出力を Q7 にします。言い換えると、二進数 "00000001" を書くと、Q0 はハイレベルを出力し、Q1~Q7 はローレベルを出力します。

7 セグメントディスプレイが数字 "1" を表示する場合、b、c にハイレベルを書き、a、d、e、f、g、および dg にローレベルを書きます。つまり、二進数 "00000110" を書く必要があります。可読性のため、16 進数表記 "0x06" を使用します。

- 16 進数
- BinaryHex 変換器

同様に、同じ方法で LED セグメントディスプレイに他の数字を表示させることもできます。以下の表は、これらの数字に対応するコードを示しています。

表 2　グリフコード

数字	二進数コード	16 進数コード
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

これらのコードを `shiftOut()` に書き込むと、LED セグメントディスプレイが対応する数字を表示します。

6.33 5.3 - タイムカウンター

4 桁の 7 セグメントディスプレイは、4 つの 7 セグメントディスプレイが連動して動作します。

この 4 桁の 7 セグメントディスプレイは独立して動作します。人間の視覚残留の原理を利用して、各 7 セグメントの文字をループで素早く表示し、連続した文字列を形成します。

例えば、ディスプレイに「1234」と表示された場合、最初の 7 セグメントには「1」が表示され、「234」は表示さ

れません。一定の時間が経過した後、2 番目の 7 セグメントが「2」を表示し、1 番目、3 番目、4 番目の 7 セグメントは何も表示しません。このように、4 つのディジタル表示が順番に表示されます。このプロセスは非常に短い（通常 5ms）ので、光の残光効果と視覚残留の原理により、4 つの文字を同時に見ることができます。

必要なコンポーネント

このプロジェクトには以下のコンポーネントが必要です。

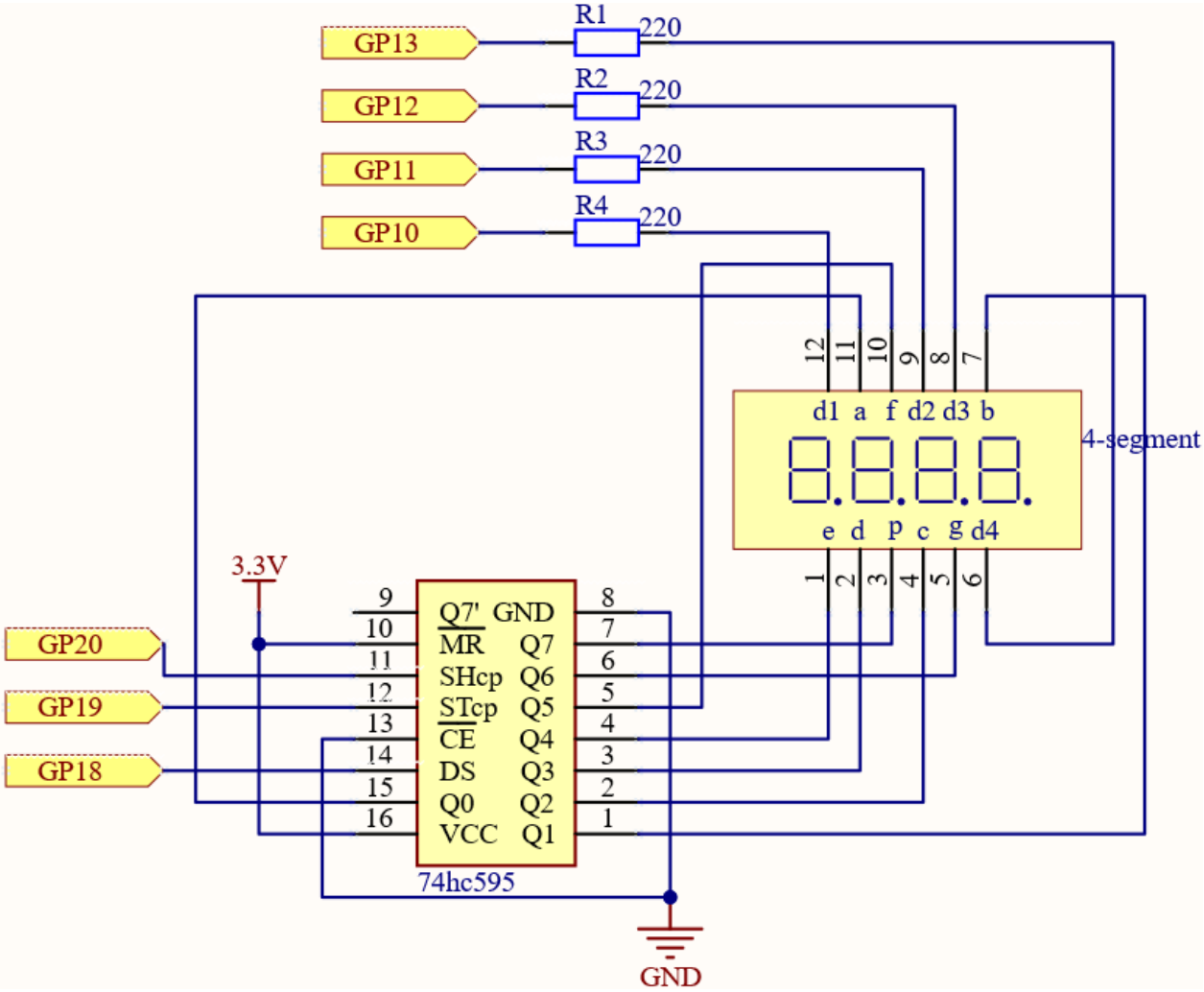
一式を購入すると便利です、リンクは以下です：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

以下のリンクから個別にも購入可能です。

SN	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(220 Ω)	
6	*4* **桁の** *7* **セグメントディスプレイ**	1	
7	*74HC595*	1	

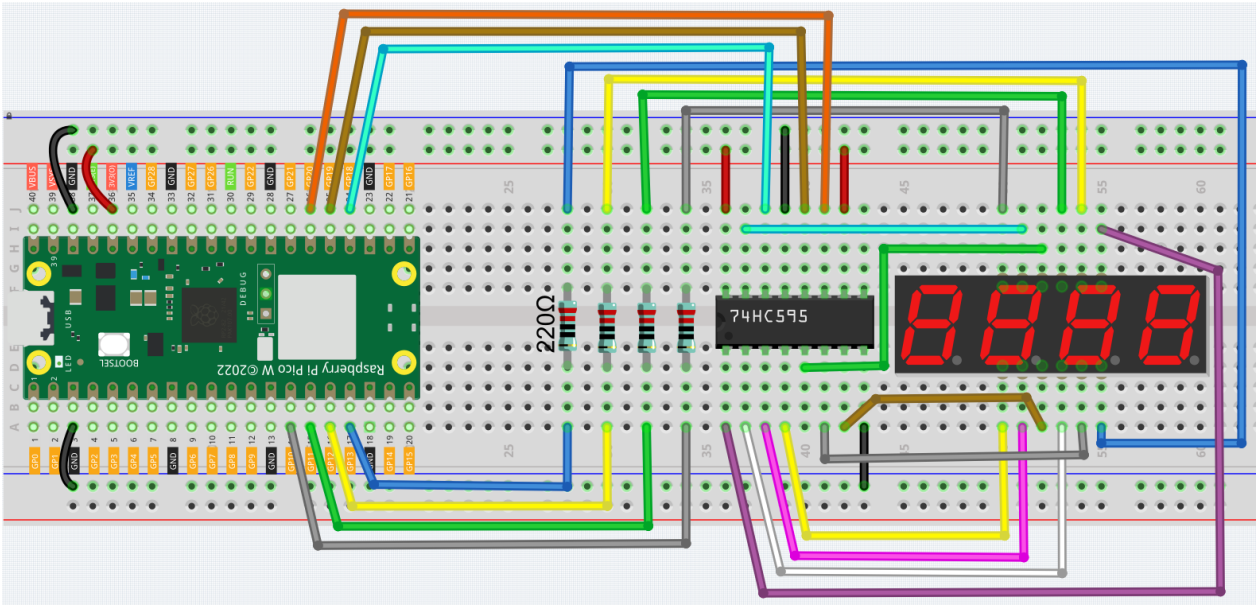
回路図



配線の原理は基本的に *5.1* **マイクロチップ** *- 74HC595* と同じで、唯一の違いは Q0～Q7 が 4 桁の 7 セグメントディスプレイの a～g ピンに接続されている点です。

次に、G10～G13 は動作させる 7 セグメントディスプレイを選択します。

配線



コード

注釈:

- パス kepler-kit-main/arduino/5.3_time_counter の下の 5.3_time_counter.ino ファイルを開くことができます。
- または、このコードを **Arduino IDE** にコピーアンドペーストしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

プログラムを実行すると、4 桁の 7 セグメントディスプレイがカウンターとして動作し、数字が 1 秒ごとに 1 ずつ増加します。

動作の仕組みは？

各 7 セグメントディスプレイへの信号の書き込みは、 *5.2 -* **数字表示** で使われる hc595_shift() 関数を用いて、同じように行われます。4 桁の 7 セグメントディスプレイの要点は、各 7 セグメントディスプレイを選択的に活性化することです。これに関連するコードは以下のとおりです。

```
const int placePin[4] = {13,12,11,10};

void setup ()
{
    for (int i = 0; i<4;i++){
        pinMode(placePin[i],OUTPUT);
    }
```

(次のページに続く)

(前のページからの続き)

```
}

void loop()
{
    pickDigit(0);
    hc595_shift(count%10/1);

    pickDigit(1);
    hc595_shift(count%100/10);

    pickDigit(2);
    hc595_shift(count%1000/100);

    pickDigit(3);
    hc595_shift(count%10000/1000);
}

void pickDigit(int digit){
    for(int i = 0; i < 4; i++){
        digitalWrite(placePin[i],HIGH);
    }
    digitalWrite(placePin[digit],LOW);
}
```

ここでは、4 つのピン（GP10、GP11、GP12、GP13）が 4 桁の 7 セグメントディスプレイの各ビットを個々に制御するために使用されています。これらのピンの状態が LOW の場合、対応する 7 セグメントディスプレイが活性化します。状態が HIGH の場合、7 セグメントディスプレイは動作しません。

ここで pickDigit(digit) 関数は、すべての 7 セグメントディスプレイを無効にし、特定の桁を個別に有効にするために使用されます。その後、 hc595_shift() が 7 セグメントディスプレイの対応する 8 ビットコードを書き込むために使用されます。

4 桁の 7 セグメントディスプレイは、私たちが 4 桁を表示していると感じられるように、連続して順番に活性化する必要があります。これは、主プログラムがタイミングに影響を与えるコードを簡単に追加できないことを意味します。

しかし、この例にタイミング関数を追加する必要があります。 delay(1000) を追加すると、4 つの 7 セグメントディスプレイが同時に動作しているという錯覚を検出することができます。

その後、 millis() 関数を使用することが優れた方法です。

```
void setup ()
{
    timerStart = millis();
}

void loop()
{
    unsigned int count = (millis()-timerStart)/1000;
}
```

millis() 関数は、現在のプログラムが開始されてからのミリ秒数を取得します。最初の時間値を timerStart として記録します。

その後、再度時間を取得する必要がある場合は、 millis() 関数を再度呼び出し、その値から timerStart を減算して、プログラムがどれだけ実行されているかを取得します。

最後に、この時間値を変換し、4 桁の 7 セグメントディスプレイに表示させます。

- millis()

6.34 5.4 - 8x8 ピクセルグラフィックス

ED マトリックスは低解像度のドットマトリックスディスプレイです。これは、パターン表示のためのピクセルとして発光ダイオード（LED）の配列を使用します。

これらは屋外の日光でも十分に明るく見え、店舗、広告看板、標識、そして公共交通機関の可変メッセージディスプレイ（例えばバスや電車など）でよく見られます。

このキットでは、16 ピンを持つ 8x8 ドットマトリックスが使用されています。アノードは行に、カソードは列に接続されており（回路レベルで）、これらの 64 個の LED をまとめて制御します。

最初の LED を点灯させるには、Row1 に高レベルを、Col1 に低レベルを供給する必要があります。2 つ目の LED を点灯させるには、Row1 に高レベル、Col2 に低レベルを供給すればよく、以降も同様です。行と列の各ペアに流れる電流を制御することで、各 LED は個々に文字や画像を表示するために制御できます。

- *LED* **ドットマトリクス**
- *74HC595*

必要な部品

このプロジェクトで必要な部品は以下の通りです。

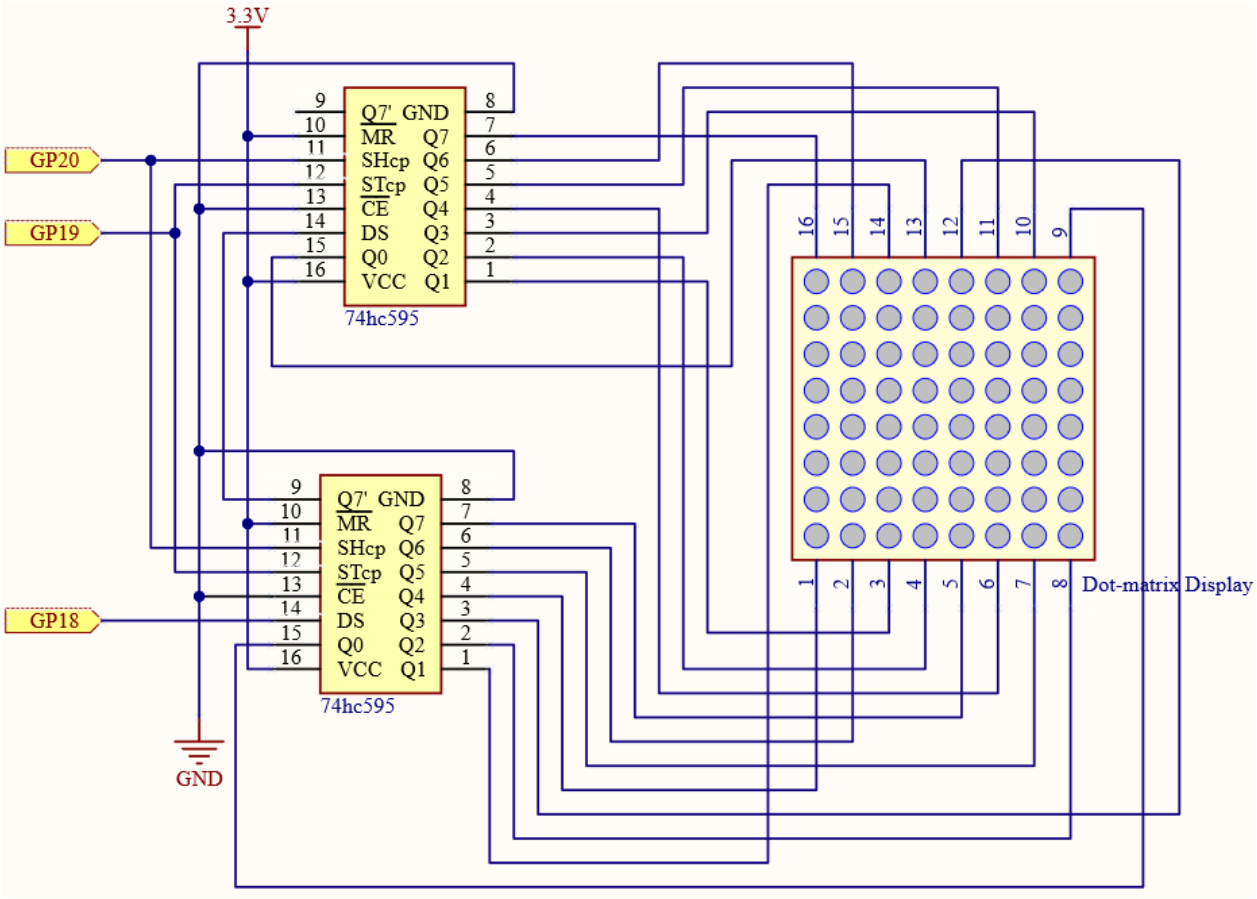
一式を購入するのが確実に便利です、そのリンクはこちらです：

名前	このキットに含まれるアイテム	購入リンク
Kepler Kit	450 以上	

以下のリンクから個々に購入することも可能です。

SN	コンポーネントの説明	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*LED* **ドットマトリクス**	1	
6	*74HC595*	2	

回路図



この 8x8 ドットマトリックスは、2 つの 74HC595 チップによって制御されています。1 つは行を、もう 1 つは列を制御しています。また、これら 2 つのチップは G18~G20 を共有しており、これにより Pico W ボードの I/O ポートを大幅に節約できます。

Pico W は一度に 16 ビットの 2 進数を出力する必要があります。最初の 8 ビットは行を制御する 74HC595 に、残りの 8 ビットは列を制御する 74HC595 に与えられ、このようにしてドットマトリックスは特定のパターンを表示できます。

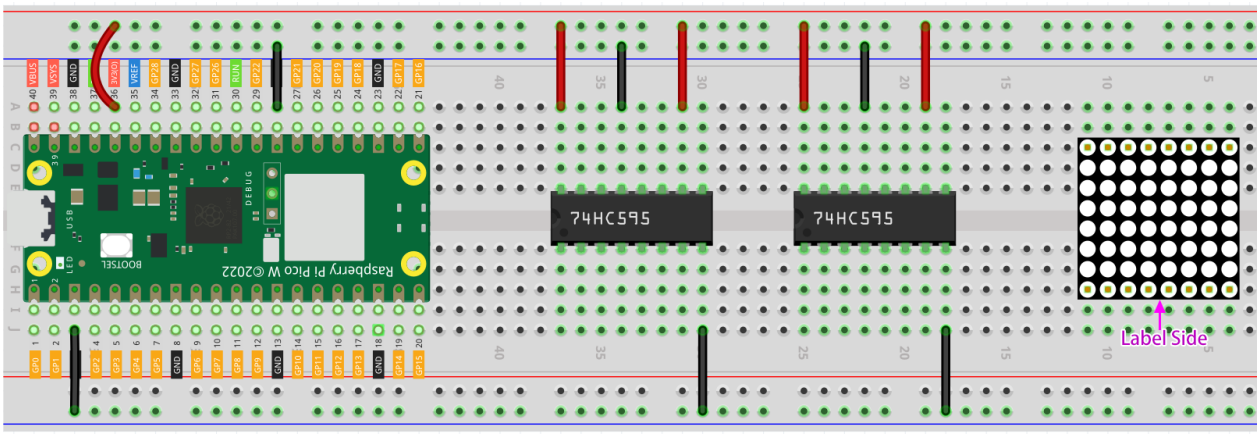
Q7': シリーズ出力ピンで、複数の 74HC595 をシリーズ接続するために別の 74HC595 の DS に接続されます。

配線

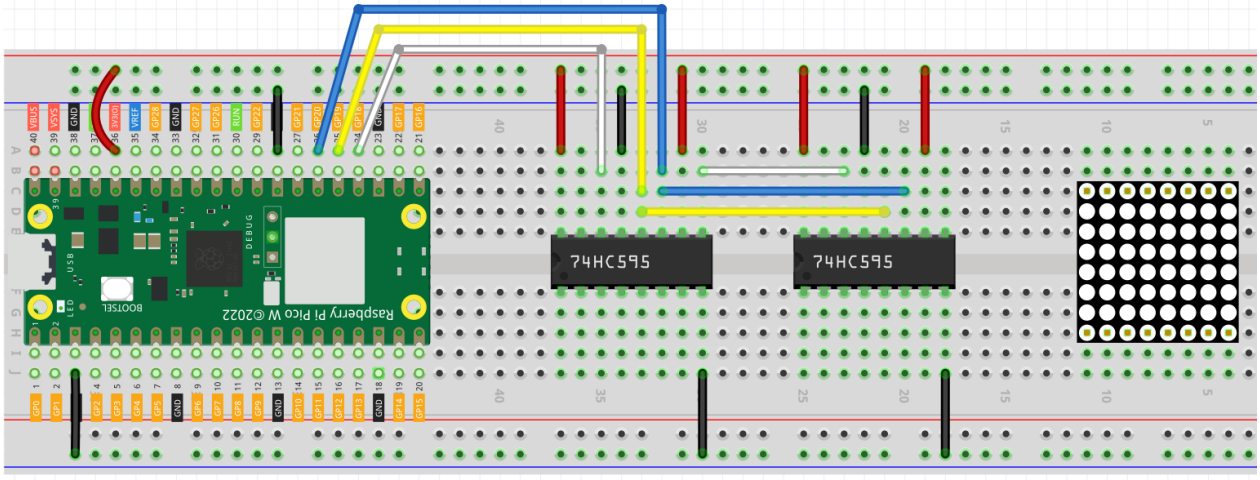
回路を組み立てましょう。配線が複雑なので、ステップバイステップで進めます。

ステップ 1: まず、Pico W、LED ドットマトリックス、および 2 つの 74HC595 チップをブレッドボードに挿入します。Pico W の 3.3V と GND をボードの両側の穴に接続し、2 つの 74HC595 チップの pin16 と pin10 を VCC に、pin13 と pin8 を GND に接続します。

注釈: 上の Fritzing 画像では、ラベルがある側は下です。

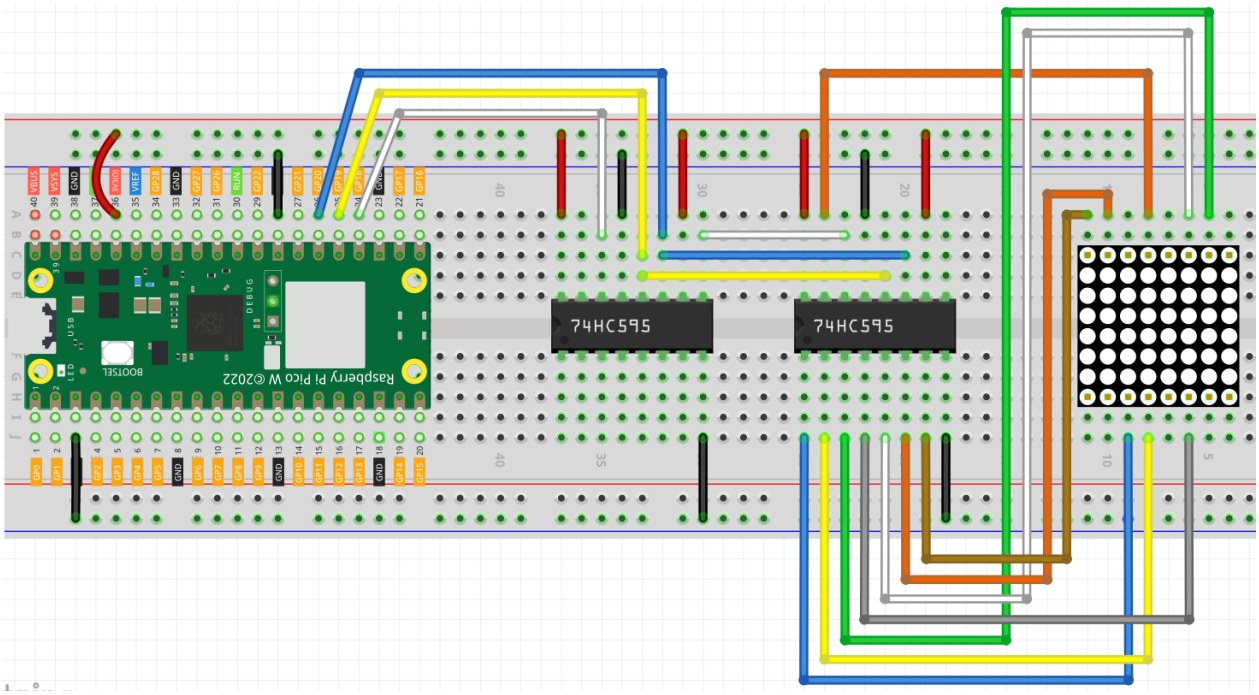


ステップ 2: 2 つの 74HC595 の pin11 を接続し、次に GP20 に接続します。次に、2 つのチップの pin12 を GP19 に接続します。次に、左側の 74HC595 の pin14 を GP18 に、pin9 を 2 つ目の 74HC595 の pin14 に接続します。



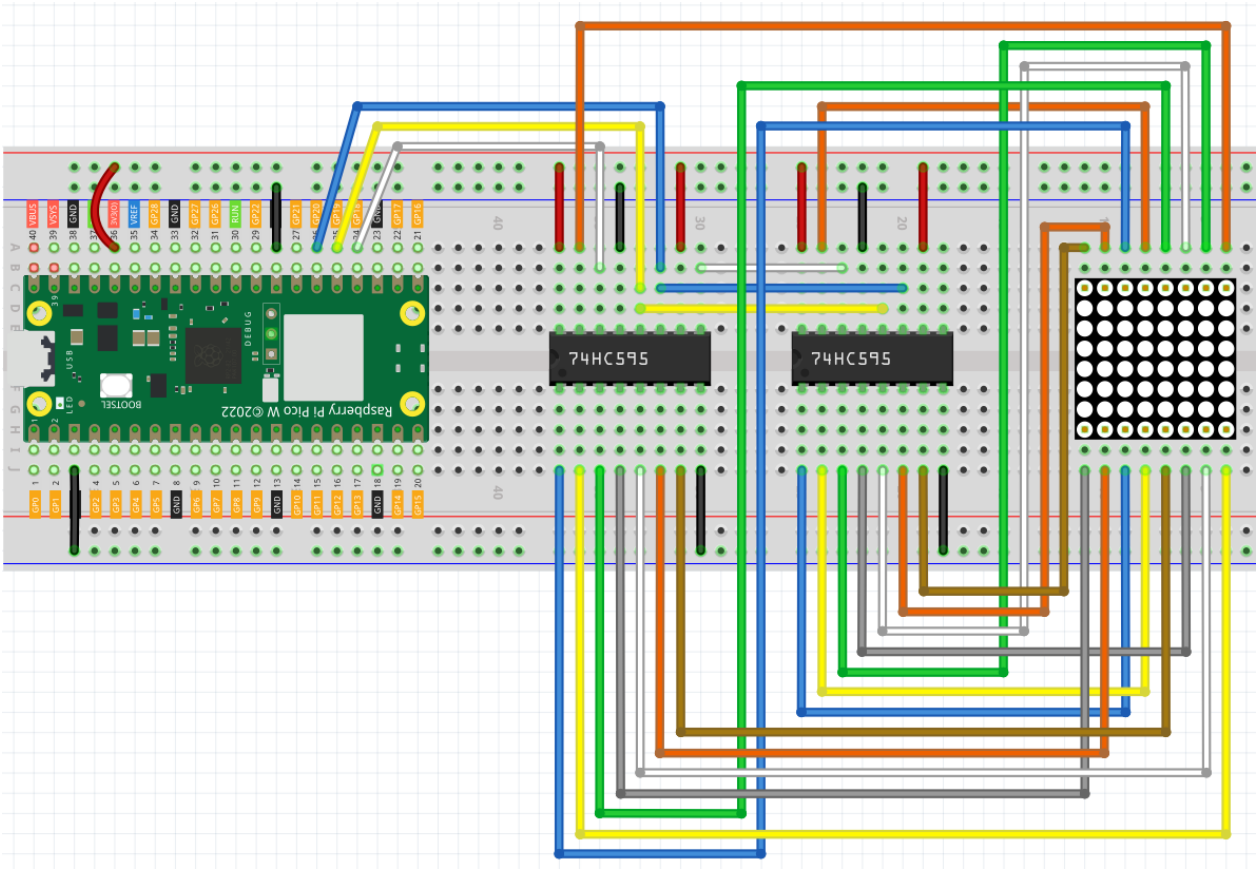
ステップ 3: 右側の 74HC595 は、LED ドットマトリックスの列を制御するためです。以下の表でマッピングを参照してください。したがって、74HC595 の Q0-Q7 ピンはそれぞれ、pin13、3、4、10、6、11、15、および 16 とマッピングされます。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



ステップ 4: 今度は LED ドットマトリックスの ROW を接続します。左側の 74HC595 が LED ドットマトリックスの ROW を制御します。以下の表でマッピングを参照してください。Q0-Q7 の 74HC595 はそれぞれ、pin9、14、8、12、1、7、2、5 とマッピングされます。

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



コード

注釈:

- ファイル `5.4_8x8_pixel_graphics.ino` を `kepler-kit-main/arduino/5.4_8x8_pixel_graphics` のパスで開くことができます。
- または、このコードを **Arduino IDE** にコピペしてください。
- ボード（Raspberry Pi Pico）と正確なポートを選択したら、 **Upload** ボタンをクリックする前に忘れずに設定してください。

プログラムが実行されると、8x8 ドットマトリックスに「X」グラフィックが表示されます。

動作原理

ここでは 2 つの 74HC595 を使用して、ドットマトリックスの行と列に信号を供給します。信号の供給方法は前の章の `shiftOut()` と同じですが、ここでは一度に 16 ビットの 2 進数を書き込む必要があります。

メインループは `shiftOut()` を 2 回呼び出し、2 つの 8 ビットの 2 進数を書き込んでバスに出力します。これにより、特定のパターンが表示されます。

ただし、ドットマトリックス内の LED は共通の極を使用しているため、複数の行/列を同時に制御すると互いに干渉します（例えば、(1,1) と (2,2) が同時に点灯すると、(1,2) と (2,1) も必然的に点灯します）。したがって、一度に 1 つの列（または 1 つの行）を活性化し、8 回のサイクルを行い、残像原理を使用して人間の目で 8 つのパターンをマージさせる必要があります。

```
for(int num = 0; num <=8; num++)
{
   digitalWrite(STcp,LOW); //ground ST_CP and hold low for as long as you are␣
↪transmitting
   shiftOut(DS,SHcp,MSBFIRST,datArray[num]);
   shiftOut(DS,SHcp,MSBFIRST,0x80>>num);
   //return the latch pin high to signal chip that it
   //no longer needs to listen for information
   digitalWrite(STcp,HIGH); //pull the ST_CPST_CP to save the data
}
```

この例では、メイン関数は for ループをネストしています。 i が 1 のとき、最初の行だけが活性化され（制御ラインのチップが 0x80 の値を取得し）、最初の行の画像が書き込まれます。i が 2 のとき、2 行目が活性化され（制御ラインのチップが 0x40 の値を取得し）、2 行目の画像が書き込まれます。これを 8 回繰り返して出力を完了します。

ちなみに、4 桁の 7 セグメントディスプレイと同様に、人間の目によるちらつきを防ぐためにリフレッシュレートを維持する必要があり、メインループ内の余分な sleep() はできるだけ避けるべきです。

もっと学ぶ

datArray を以下の配列に置き換えて、どのような画像が表示されるか試してみてください！

```
int datArray1[] = {0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF};
int datArray2[] = {0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF};
int datArray3[] = {0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF};
int datArray4[] = {0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF};
int datArray5[] = {0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF};
int datArray6[] = {0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF};
```

または、独自のグラフィックを描いてみてください。

6. 上級編

6.35 6.1 - 距離の測定

超音波センサーモジュールは、物体までの距離を決定するために、ソナーおよびレーダーシステムの原理に基づいて動作します。

- **超音波モジュール**

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

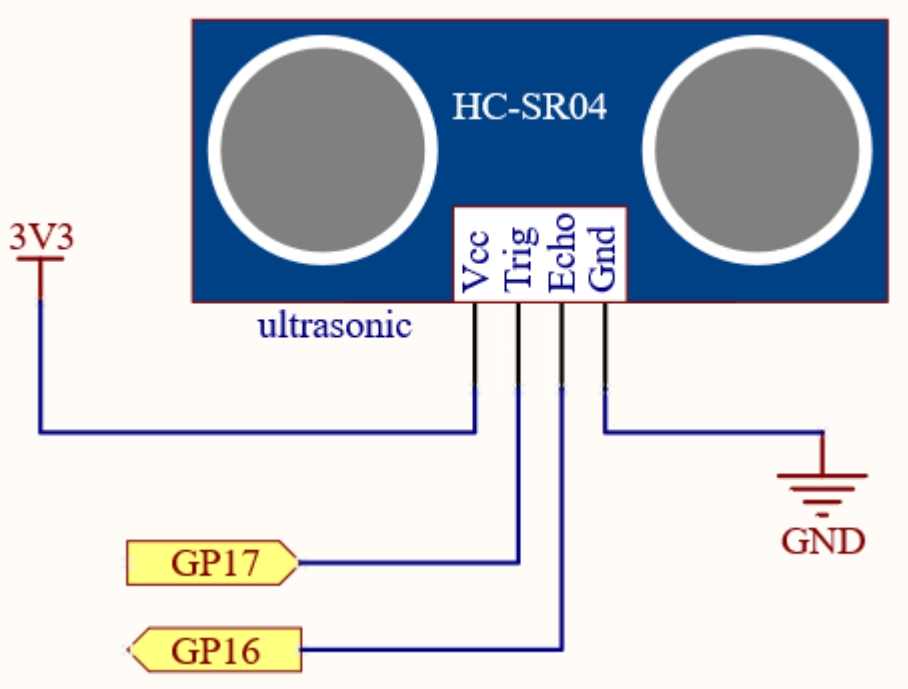
全体のキットを購入する方が確実に便利です。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

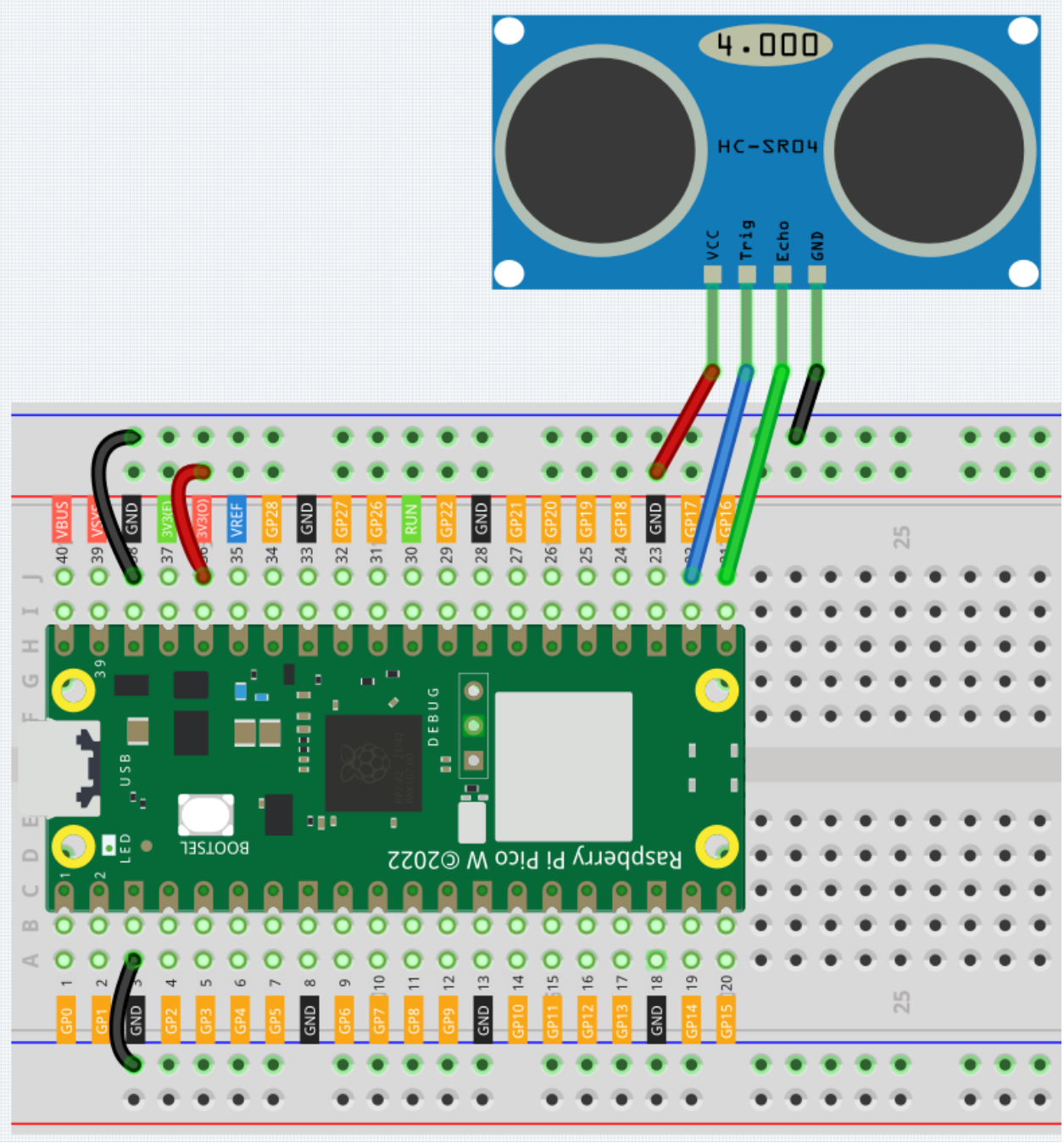
以下のリンクから個別にも購入できます。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**超音波モジュール**	1	

回路図



配線



コード

注釈:

- ファイル `6.1_ultrasonic.ino` は、パス `kepler-kit-main/arduino/6.1_ultrasonic` にあります。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と適切なポートを選択してください。

プログラムが動作していると、シリアルモニターには超音波センサーから先の障害物までの距離が表示されます。

動作原理

超音波センサーの適用については、サブ関数を直接確認できます。

```
float readSensorData(){// ...}
```

PING は、2 マイクロ秒以上の HIGH パルスでトリガーされます。（クリーンな HIGH パルスを確保するために、事前に短い LOW パルスを与えます。）

```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
```

エコーピンは、PING からの信号を読み取るために使用され、その期間は物体のエコーを受信するまでの時間（マイクロ秒単位）です。

```
microsecond=pulseIn(echoPin, HIGH);
```

音速は 340 m/s、または 1 センチメートル当たり 29 マイクロ秒です。

これは、ピンによって移動した距離、往復を指し、障害物までの距離を得るために 2 で割ります。

```
float distance = microsecond / 29.00 / 2;
```

超音波センサーが動作しているときにプログラムが一時停止することに注意してください。これは、複雑なプロジェクトを作成しているときに遅延を引き起こす可能性があります。

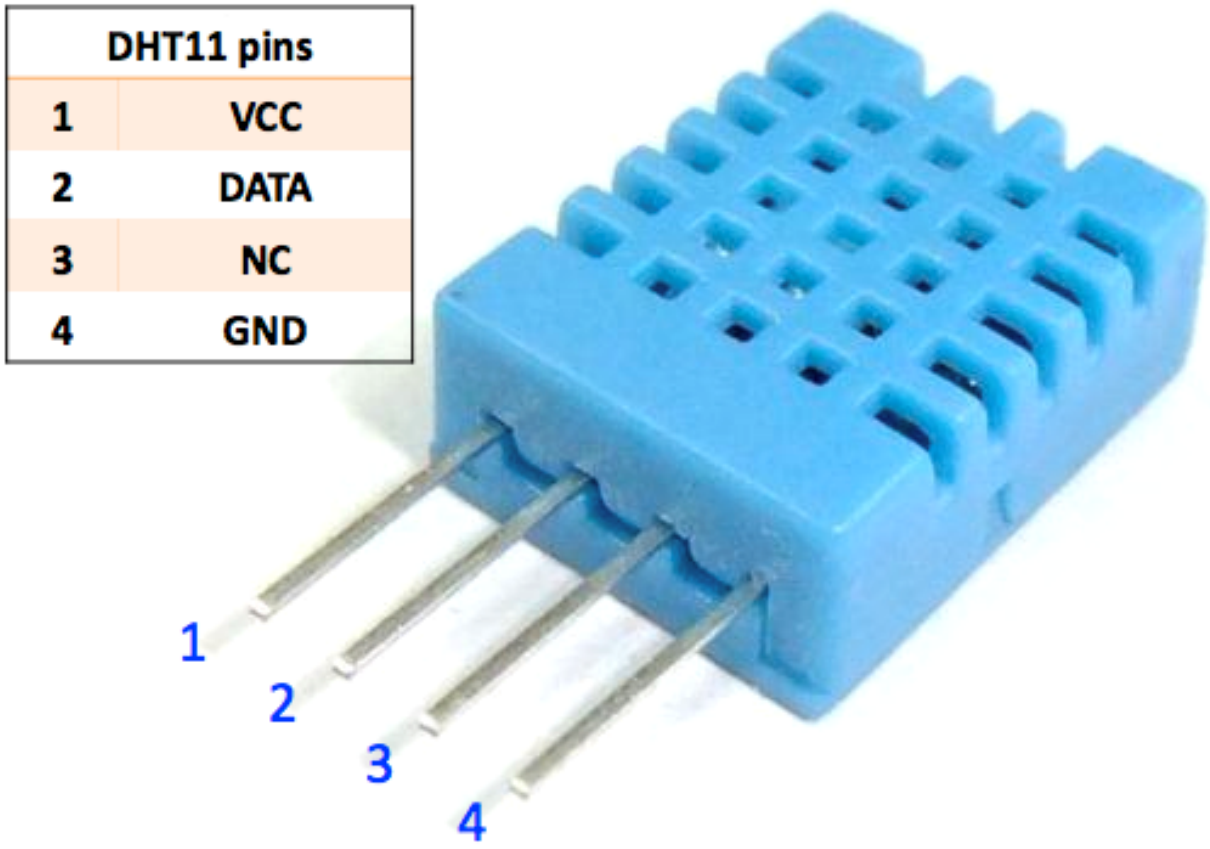
6.36 6.2 - 温度・湿度

湿度と温度は、物理量自体から日常生活に至るまで、密接に関連しています。人々が生活する環境の温度と湿度は、人体の体温調節機能や熱伝達効果に直接影響を与えます。さらには、思考活動や精神状態にも影響を与え、学習や仕事の効率にも関わってきます。

温度は国際単位系（SI）における 7 つの基本物理量の一つであり、物体の熱い・冷たい程度を測るために使用されます。摂氏は、世界で広く使用されている温度の尺度の一つであり、「℃」という記号で表されます。

湿度とは、空気中に存在する水蒸気の濃度です。相対湿度は、日常生活でよく使用されるものであり、%RH で表されます。相対湿度は温度と密接に関連しています。密閉された一定量のガスにおいて、温度が高いほど相対湿度

は低く、温度が低いほど相対湿度は高くなります。



このキットには基本的なデジタル温度・湿度センサー、 **DHT11** が含まれています。このセンサーは、周囲の空気の湿度と温度を測定するために、容量性湿度センサーとサーミスタを使用し、データピンでデジタル信号を出力します（アナログ入力ピンは不要です)。

- *DHT11* **湿温度センサー**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

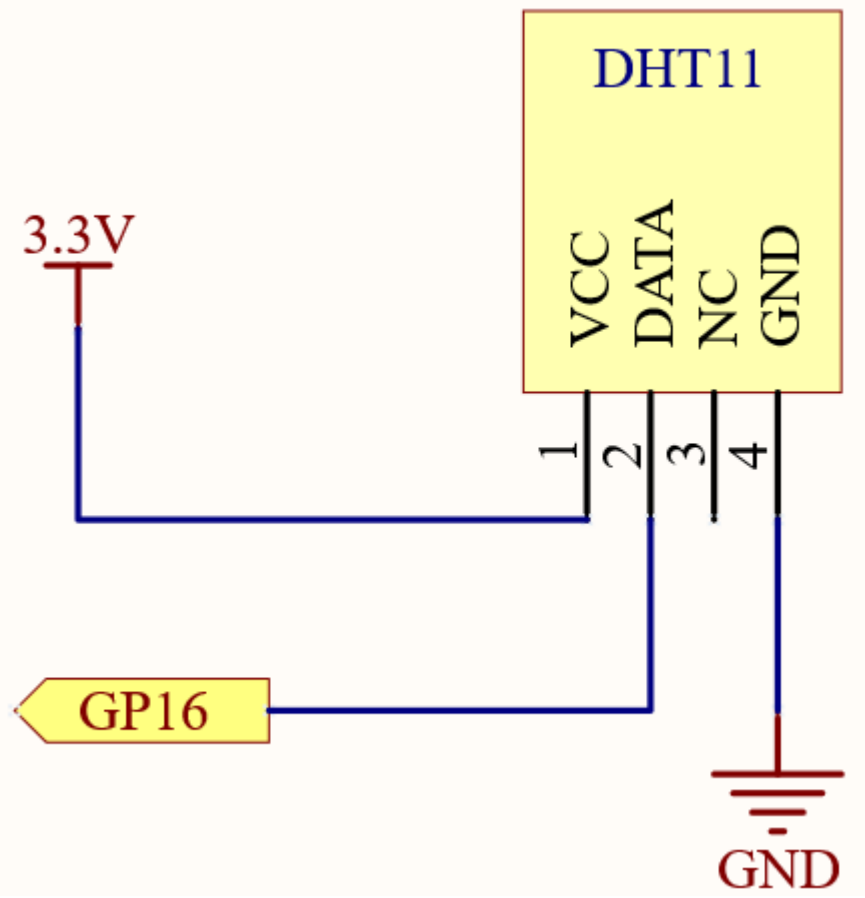
全体のキットを購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

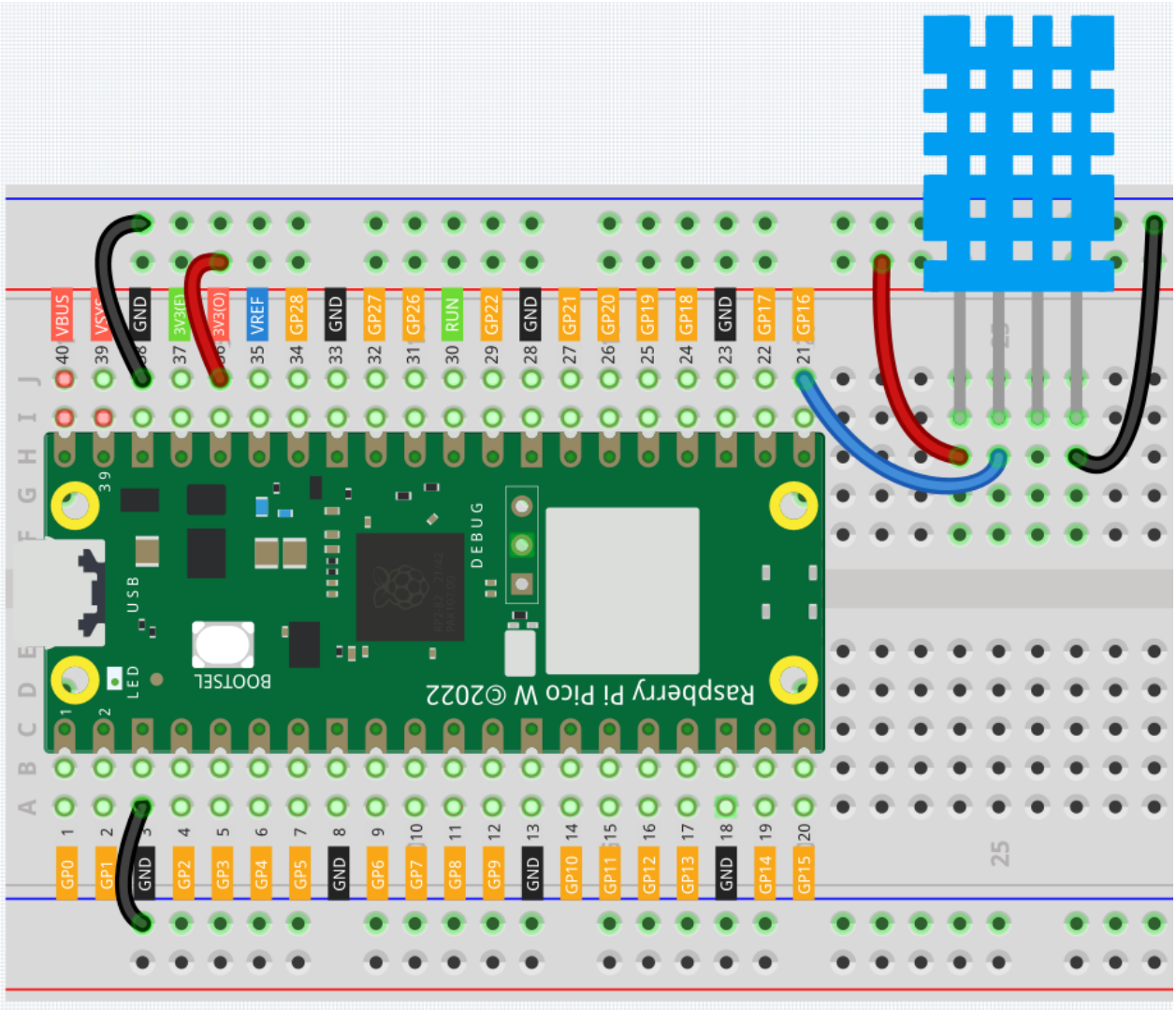
以下のリンクから個別にも購入可能です。

SN	コンポーネントの説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*DHT11* **湿温度センサー**	1	

回路図



配線



コード

注釈:

- ファイル `6.2_dht11.ino` は、パス `kepler-kit-main/arduino/6.2_dht11` の下で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。
- ここでは `SimpleDHT` ライブラリが使用されています。Arduino IDE に追加する方法については、 **ライブラリを追加** を参照してください。

コードが実行された後、シリアルモニターが連続して温度と湿度を出力するようになり、プログラムが安定して動

作するにつれて、これらの二つの値はより正確になります。

動作原理は？

DHT11 オブジェクトを初期化します。このデバイスは、デジタル入力だけで使用できます。

```
int pinDHT11 = 16;
SimpleDHT11 dht11(pinDHT11);
```

現在の温度と湿度を読み取り、それらは変数 temperature と humidity に保存されます。 err はデータの妥当性を判断するために使用されます。

```
byte temperature = 0;
byte humidity = 0;
int err = dht11.read(&temperature, &humidity, NULL);
```

無効なデータをフィルタリングします。

```
if (err != SimpleDHTErrSuccess) {
    Serial.print("Read DHT11 failed, err=");
    Serial.print(SimpleDHTErrCode(err));
    Serial.print(",");
    Serial.println(SimpleDHTErrDuration(err));
    delay(1000);
    return;
}
```

温度と湿度を出力します。

```
Serial.print((int)temperature);
Serial.print(" *C, ");
Serial.print((int)humidity);
Serial.println(" H");
```

最後に、DHT11 のサンプリングレートは 1HZ であり、ループ内で delay(1500) が必要です。

```
delay(1500);
```

6.37 6.3 - 6 軸モーショントラッキング

MPU-6050 は、3 軸ジャイロスコープと 3 軸加速度計を組み合わせた 6 軸モーショントラッキングデバイスです。

加速度計は、適切な加速度を測定するツールです。例えば、地球上で静止している加速度計は、地球の重力による加速度を直上方向に測定します。その値はおおよそ g 9.81 m/s2 です。

加速度計は産業や科学で多くの用途があります。例としては、航空機やミサイルの慣性航法システム、タブレットやデジタルカメラの画像を垂直に保つためなどがあります。

ジャイロスコープは、デバイスの方向や角速度を測定するために使用されます。ジャイロスコープの応用例としては、自動車の反転防止やエアバッグシステム、スマートデバイスのモーションセンシングシステム、ドローンの姿勢安定化システムなどがあります。

- *MPU6050* **モジュール**

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

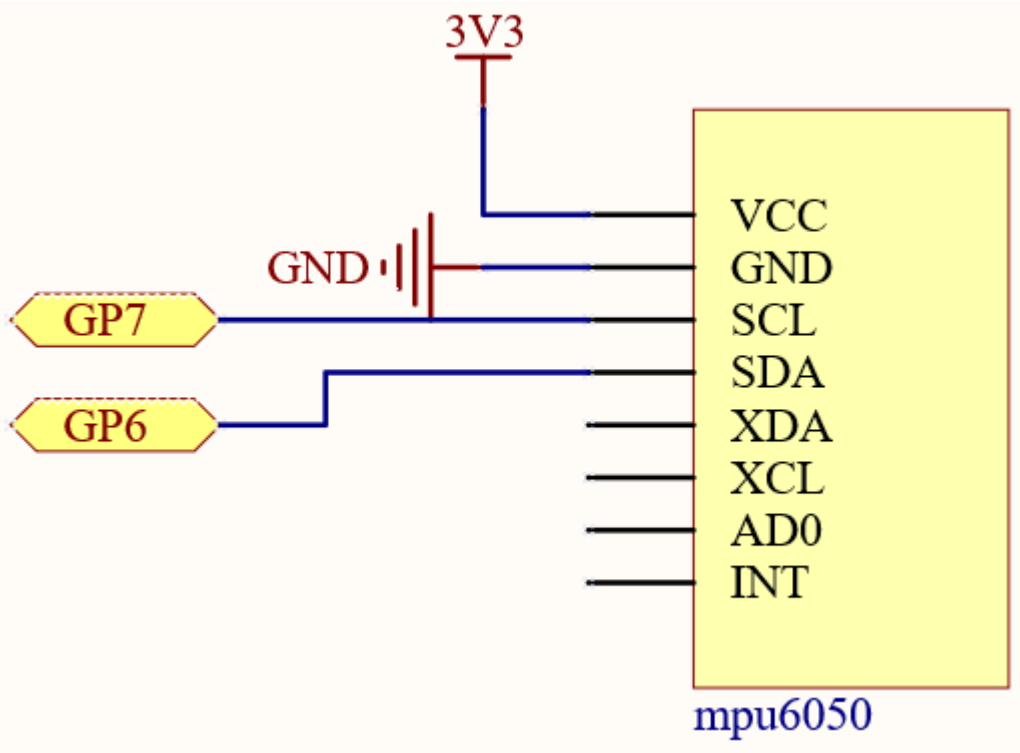
一式をまとめて購入するのが便利です。詳細は以下のリンクを参照してください：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450+	

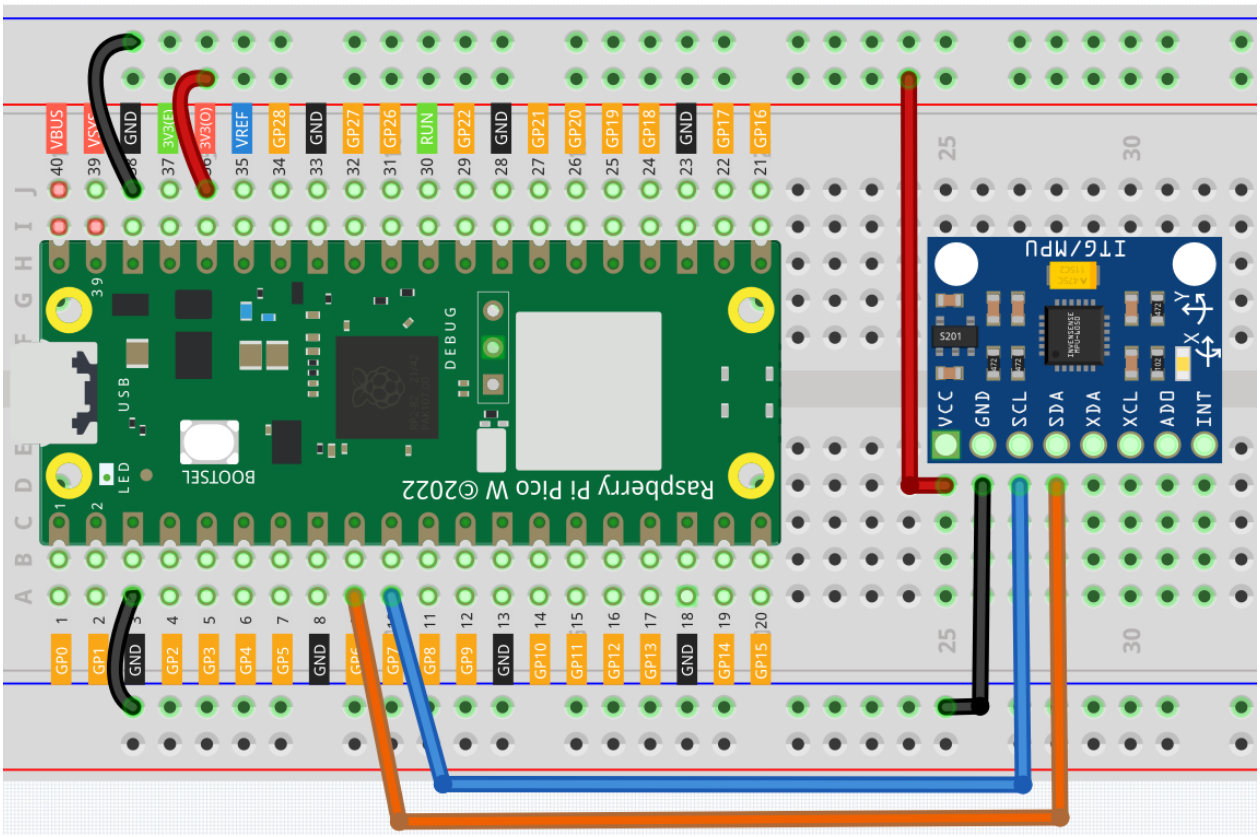
以下のリンクから個々に購入することもできます。

SN	コンポーネント説明	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MPU6050* **モジュール**	1	

回路図


3V3
GND
GP7
GP6
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
mpu6050

配線


VBUS
GND
VREF
GP28
GP27
GP26
RUN
GP22
GP21
GP20
GP19
GP18
GP17
GP16
GP0
GP1
GP2
GP3
GP4
GP5
GP8
GP9
GP10
GP11
GP12
GP13
GP14
GP15
DEBUG
USB
LED
BOOTSEL
Raspberry Pi Pico W ©2022
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT

コード

注釈:

- kepler-kit-main/arduino/6.3_6axis_motion_tracking のパスにある 6.3_6axis_motion_tracking.ino ファイルを開いてください。
- または、このコードを **Arduino IDE** にコピーアンドペーストしてください。
- **Upload** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正確なポートを選択することを忘れないでください。
- ここでは Adafruit_MPU6050 ライブラリを使用しています。Arduino IDE に追加する方法については、**ライブラリを追加** を参照してください。

プログラムを実行した後、3 軸加速度計の値と 3 軸ジャイロスコープの値が出力に順次表示されます。この時点で MPU6050 をランダムに回転させると、これらの値はそれに応じて変化するでしょう。変化を容易に確認するために、出力ラインの一つをコメントアウトして、別のデータセットに焦点を当てることができます。

動作原理

MPU6050 オブジェクトをインスタンス化します。

```
#include <Adafruit_MPU6050.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
```

MPU6050 を初期化し、その精度を設定します。

```
void setup(void) {
    Serial.begin(115200);
    while (!Serial)
        delay(10); // will pause Zero, Leonardo, etc until serial console opens

    Serial.println("Adafruit MPU6050 test!");

    // Try to initialize!
    if (!mpu.begin()) {
        Serial.println("Failed to find MPU6050 chip");
        while (1) {
        delay(10);
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
    Serial.println("MPU6050 Found!");

    // Set range
    mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
    mpu.setGyroRange(MPU6050_RANGE_500_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

    Serial.println("");
    delay(100);
}
```

新しいセンサーイベントとその読み取り値を取得します。

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
```

これにより、データ a.acceleration.x 、 a.acceleration.y 、 a.acceleration.z 、 g.gyro.x 、 g.gyro.y 、 g.gyro.z でリアルタイムの加速度と角速度の値を取得できます。

```
Serial.print("Acceleration X: ");
Serial.print(a.acceleration.x);
Serial.print(", Y: ");
Serial.print(a.acceleration.y);
Serial.print(", Z: ");
Serial.print(a.acceleration.z);
Serial.println(" m/s^2");

Serial.print("Rotation X: ");
Serial.print(g.gyro.x);
Serial.print(", Y: ");
Serial.print(g.gyro.y);
Serial.print(", Z: ");
Serial.print(g.gyro.z);
Serial.println(" rad/s");
```

6.38 6.4 - IR リモートコントロール

家庭用電子機器では、テレビや DVD プレーヤーなどのデバイスを操作するためにリモートコントロールが使用されます。場合によっては、リモートコントロールによって手の届かない場所にあるデバイス、例えばセントラルエアコンを操作することも可能です。

IR レシーバーは、赤外線を受信するように調整されたフォトセルを持つ部品です。ほとんどの場合、リモートコントロールの検出に使用されます。すべてのテレビや DVD プレーヤーの前面には、クリッカーからの IR 信号を受信するためのものがあります。リモートコントロールの内部には、テレビをオン、オフ、またはチャンネルを変更するように指示する IR LED があります。

- **赤外線レシーバー**

必要な部品

このプロジェクトでは、以下の部品が必要です。

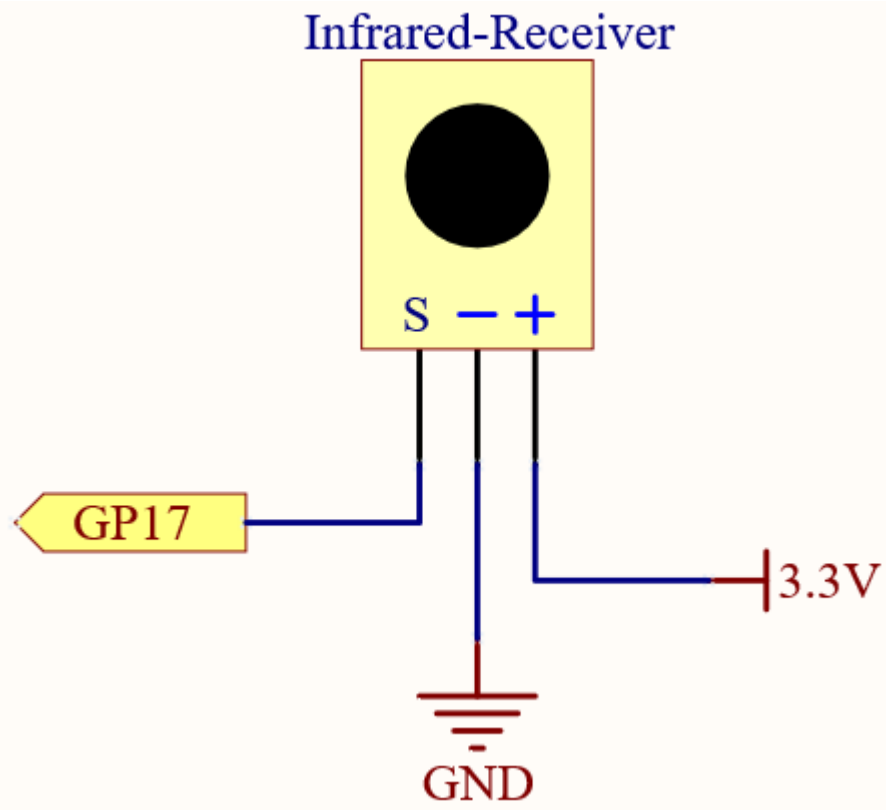
便利なのは、一式をまとめて購入することです。リンクはこちら：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

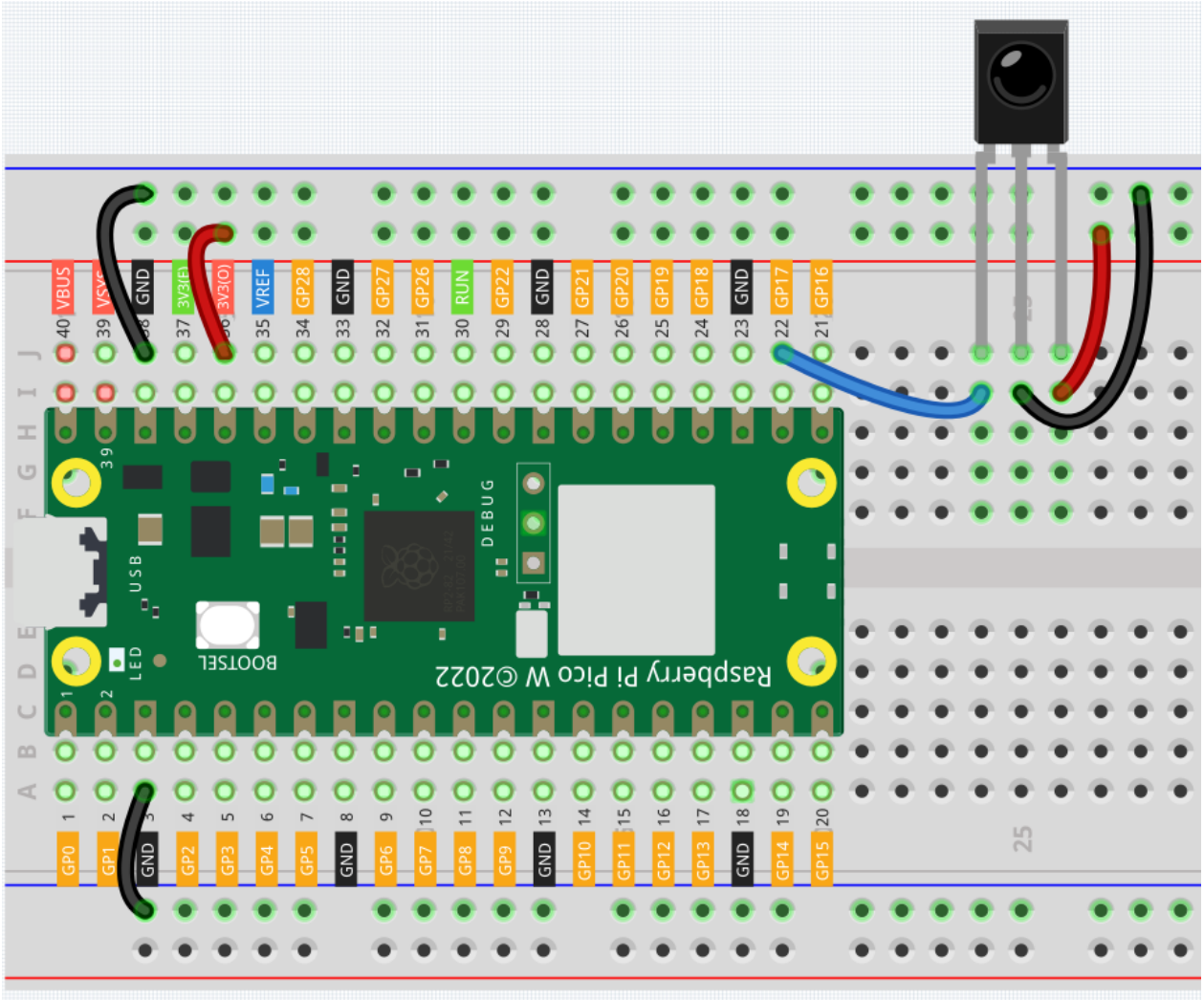
以下のリンクから個々にも購入できます。

SN	部品紹介	個数	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**赤外線レシーバー**	1	

回路図



配線



コード

注釈:

- ファイル `6.4_ir_remote_control.ino` は、 `kepler-kit-main/arduino/6.4_ir_remote_control` のパスで開くことができます。
- またはこのコードを **Arduino IDE** にコピーペーストしてください。
- **アップロード** ボタンを押す前に、ボード（Raspberry Pi Pico）と正確なポートを選択してください。
- ここでは `IRsmallDecoder` ライブラリが使用されています。 **ライブラリを追加** を参照して、Arduino IDE に追加してください。

新しいリモートコントロールには、内部のバッテリーを隔離するためのプラスチック片が端にあります。使用する際には、このプラスチック片を引き抜いてリモートコントロールに電源を供給してください。プログラムが動作し

ている間、リモートコントロールのボタンを押すと、シリアルモニターに押したキーが表示されます。

6.39 6.5 - 無線周波数識別（RFID）

無線周波数識別（RFID）は、オブジェクト（またはタグ）と照会装置（またはリーダー）との間で無線通信を使用して、そのようなオブジェクトを自動的に追跡・識別するテクノロジーを指します。タグの伝送範囲はリーダーから数メートルに限られます。リーダーとタグの間には必ずしも直線的な視界が必要ではありません。

ほとんどのタグには、少なくとも一つの集積回路（IC）とアンテナが含まれています。このマイクロチップは情報を格納し、リーダーとの無線周波数（RF）通信を管理しています。パッシブタグには独立したエネルギー源がなく、リーダーから提供される外部の電磁信号に依存して動作します。アクティブタグには独立したエネルギー源、例えばバッテリーが含まれています。したがって、これらは処理能力、伝送能力、および範囲が増加する可能性があります。

- *MFRC522* **モジュール**

必要なコンポーネント

このプロジェクトでは、以下のコンポーネントが必要です。

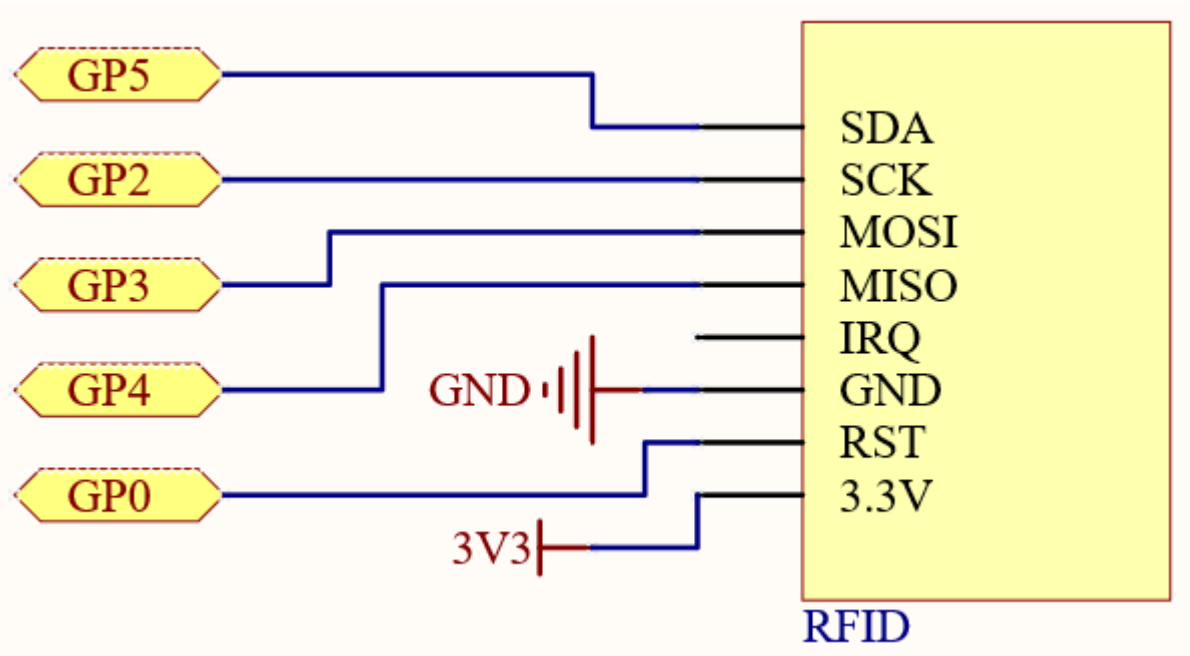
全体のキットを購入するのが便利です、リンクは以下の通りです：

名前	このキットに含まれるアイテム	購入リンク
ケプラーキット	450 以上	

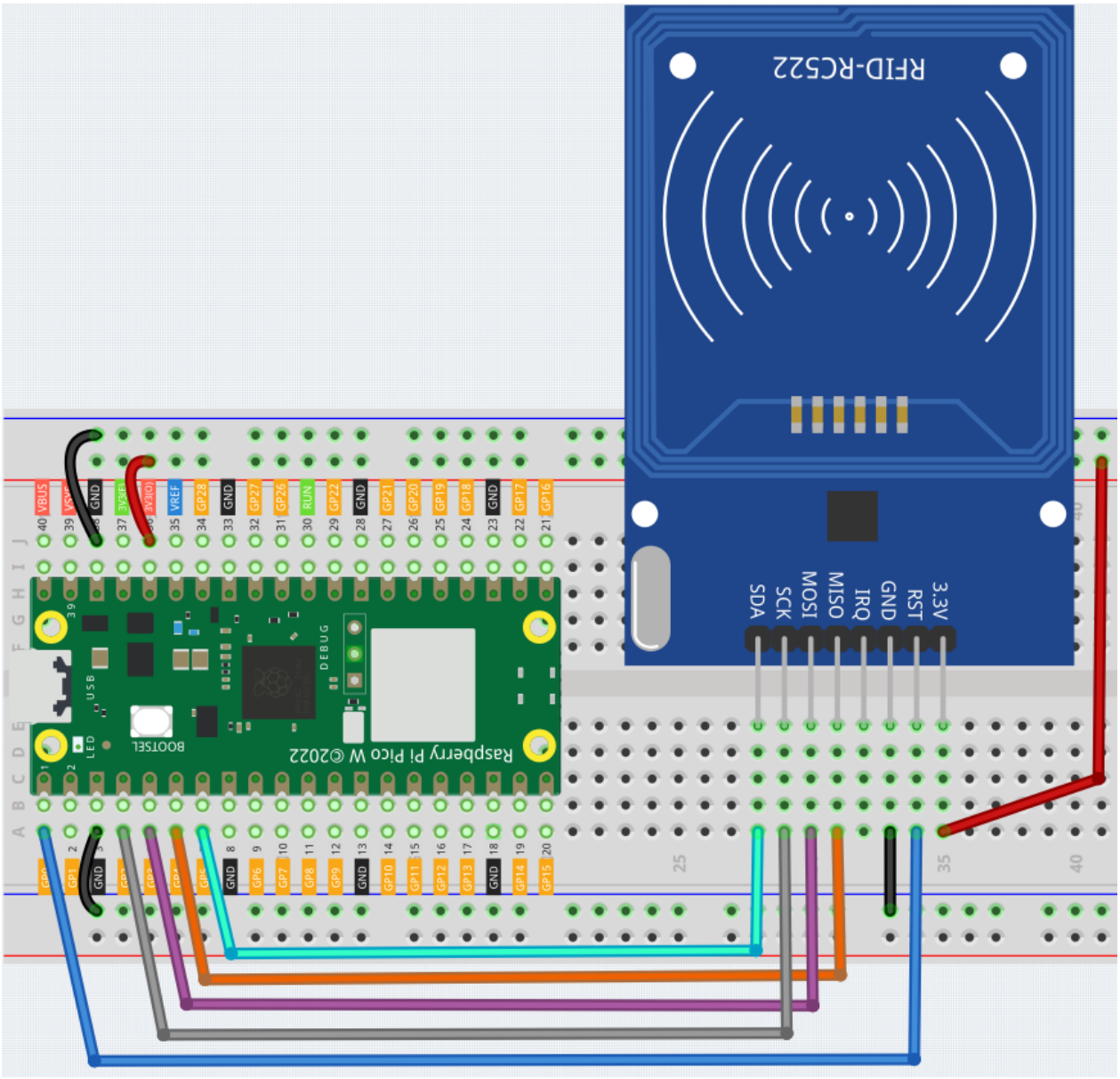
以下のリンクから個別に購入することもできます。

番号	コンポーネント紹介	数量	購入リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*MFRC522* **モジュール**	1	

回路図


GP5
GP2
GP3
GP4
GP0
GND
3V3
SDA
SCK
MOSI
MISO
IRQ
GND
RST
3.3V
RFID

配線



コード

注釈:

- ファイル `6.5_rfid_write.ino` は、パス `kepler-kit-main/arduino/6.5_rfid_write` で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。
- ここではライブラリ「MFRC522」が使用されています。Arduino IDE に追加する方法については、 **ライブラリを追加** を参照してください。

メインの関数は二つに分かれています：

- 6.5_rfid_write.ino ：カード（またはキー）に情報を書き込むために使用されます。
- 6.5_rfid_read.ino ：カード（またはキー）内の情報を読み取るために使用されます。

注釈:

- ファイル 6.5_rfid_write.ino は、パス kepler-kit-main/arduino/6.5_rfid_write で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

実行後、シリアルモニターでメッセージを入力して、 # で終了した後、MFRC522 モジュールに近づけることでカード（またはキー）にメッセージを書き込むことができます。

注釈:

- ファイル 6.5_rfid_read.ino は、パス kepler-kit-main/arduino/6.5_rfid_read で開くことができます。
- または、このコードを **Arduino IDE** にコピーしてください。
- **アップロード** ボタンをクリックする前に、ボード（Raspberry Pi Pico）と正しいポートを選択してください。

実行後、カード（またはキー）に保存されているメッセージを読み取ることができます。

どのように動作するのか？

```
#include <MFRC522.h>

#define RST_PIN         0
#define SS_PIN          5

MFRC522 mfrc522(SS_PIN, RST_PIN);
```

まず、 MFRC522() クラスをインスタンス化します。

使いやすさのために、 MFRC522 ライブラリは以下の関数でさらにカプセル化されています。

- void simple_mfrc522_init() : SPI 通信を開始し、mfrc522 モジュールを初期化します。

- `void simple_mfrc522_get_card()`：カード（またはキー）が検出されるまでプログラムを一時停止し、カードの UID と PICC タイプを表示します。
- `void simple_mfrc522_write(String text)`：カード（またはキー）に文字列を書き込みます。
- `void simple_mfrc522_write(byte* buffer)`：通常はシリアルポートから来る情報をカード（またはキー）に書き込みます。
- `void simple_mfrc522_write(byte section, String text)`：特定のセクターに文字列を書き込みます。 `section` が 0 の場合、セクター 1-2 に書き込みます; `section` が 1 の場合、セクター 3-4 に書き込みます。
- `void simple_mfrc522_write(byte section, byte* buffer)`：通常はシリアルポートから来る情報を特定のセクターに書き込みます。 `section` が 0 の場合、セクター 1-2 に書き込みます; `section` が 1 の場合、セクター 3-4 に書き込みます。
- `String simple_mfrc522_read()`：カード（またはキー）内の情報を読み取り、文字列を返します。
- `String simple_mfrc522_read(byte section)`：特定のセクター内の情報を読み取り、文字列を返します。 `section` が 0 の場合、セクター 1-2 を読み取ります; `section` が 1 の場合、セクター 3-4 を読み取ります。

`6.5_rfid_write.ino` の例では、一般的なシリアル入力方法として `Serial.readBytesUntil()` 関数が使用されています。

- Serial.readBytesUntil

第 7 章

Piper Make について

この章では、Piper Make の紹介、Piper Make で Pico W を接続およびプログラムする方法、そして Piper をすぐに使いこなすためのいくつかの面白いプロジェクトについて説明します。

この章は順番に読むことをお勧めします。

Piper Make は、Raspberry Pi Pico W を使ってプロジェクトを作成するための非常に簡単で楽しい方法です。Scratch のようなブロックを使用しているため、プログラミング経験は必要ありません。基本的な原理は、補助ライブラリとともに CircuitPython を使用することです。

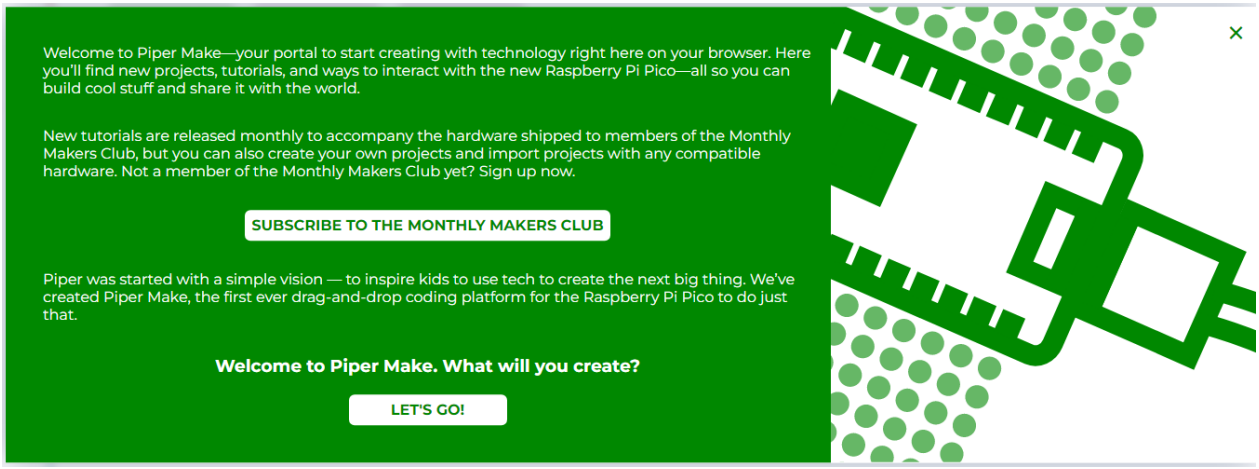
1. はじめに

7.1 1.1 Pico のセットアップ

まず、以下のリンクから Piper Make にアクセスしてください。

https://make.playpiper.com/

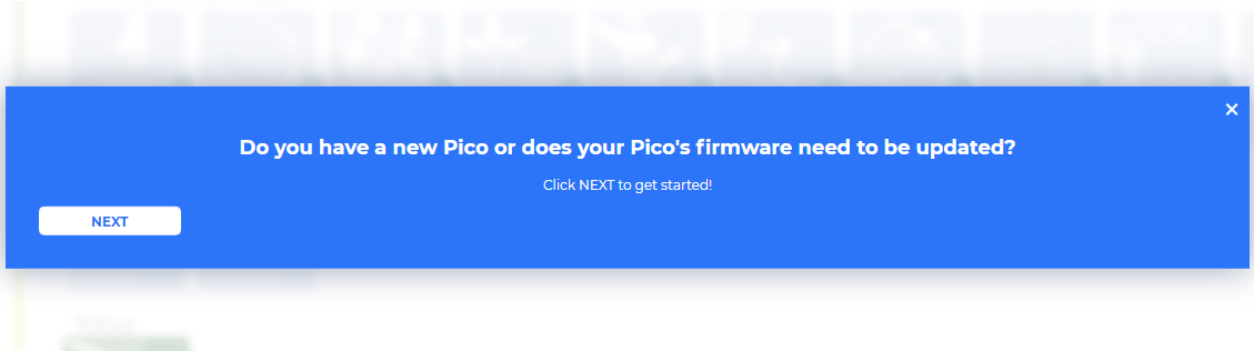
ポップアップページが表示された場合、追加のチュートリアルにサブスクライブする必要がない場合は、 **Let's Go!** または **x** ボタンをクリックしてください。



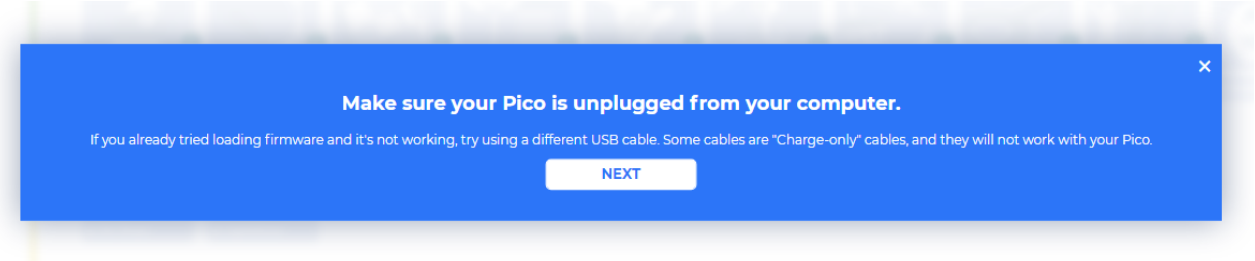
注釈: 異なるポップアップウィンドウが表示される場合は、お使いのブラウザのバージョンがサポートされていない可能性があります。ブラウザをアップデートして再試行してください。

SETUP MY PICO ボタンを見つけてクリックし、指示に従って設定を行ってください。

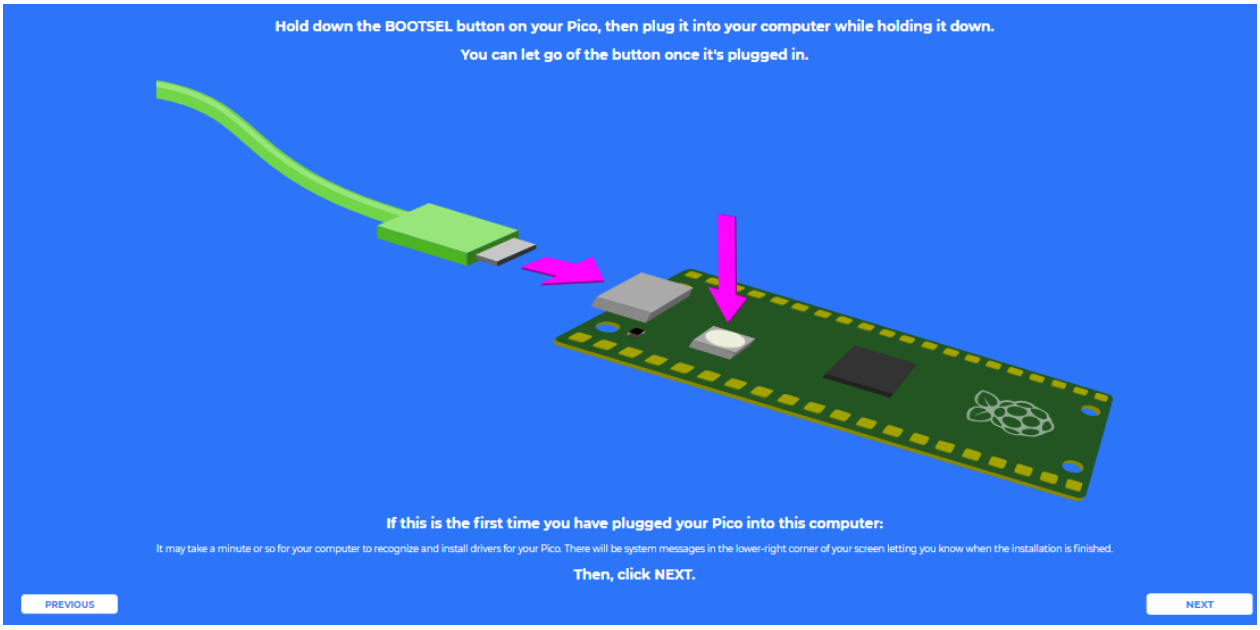
Next をクリックして、Pico W の設定を開始します。以前に設定したことがあっても、この手順は Pico W のファームウェアを更新する際にも使用します。



このステップでは、次のステップで特定の方法で接続する必要があるため、Pico W がコンピュータから切断されていることを確認する必要があります。ケーブルが電源とデータの両方を処理できることを確認してください。多くのマイクロ USB ケーブルは電源しか提供していません。

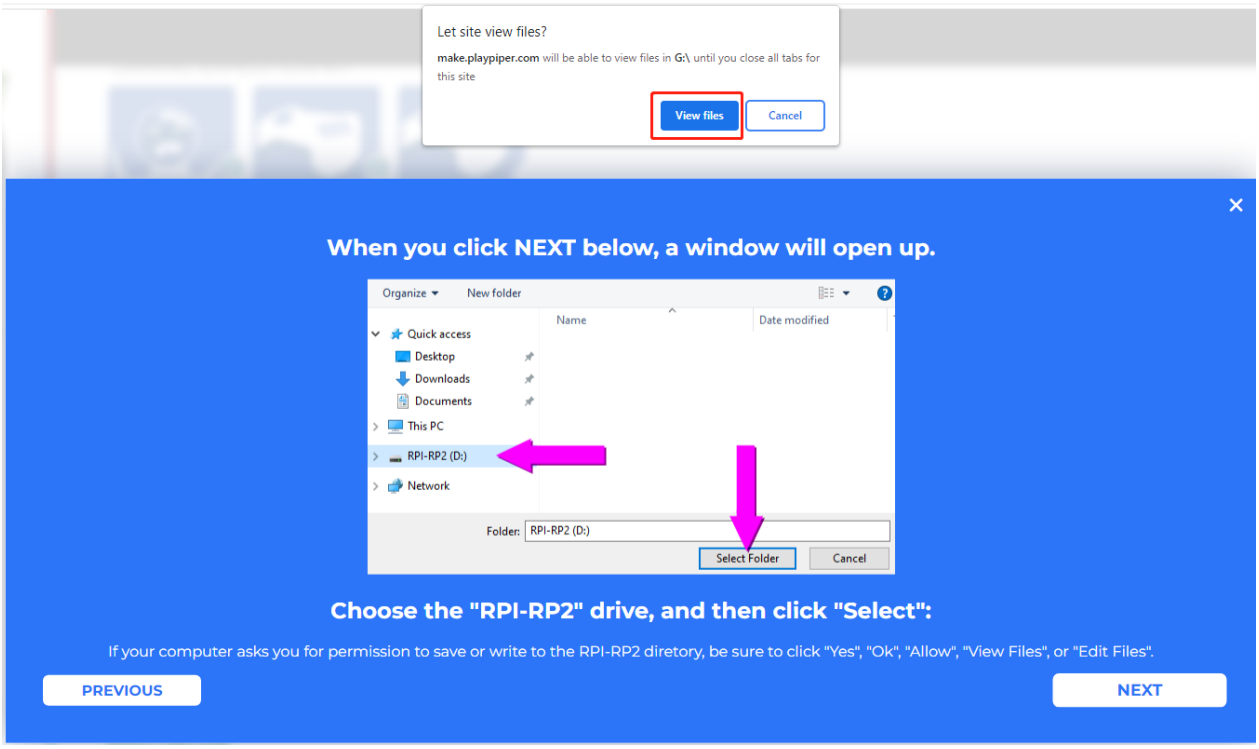


次に、Pico W の RST（白）ボタンを押しながら、Pico W をコンピュータに接続します。接続したら、ボタンを離してください。

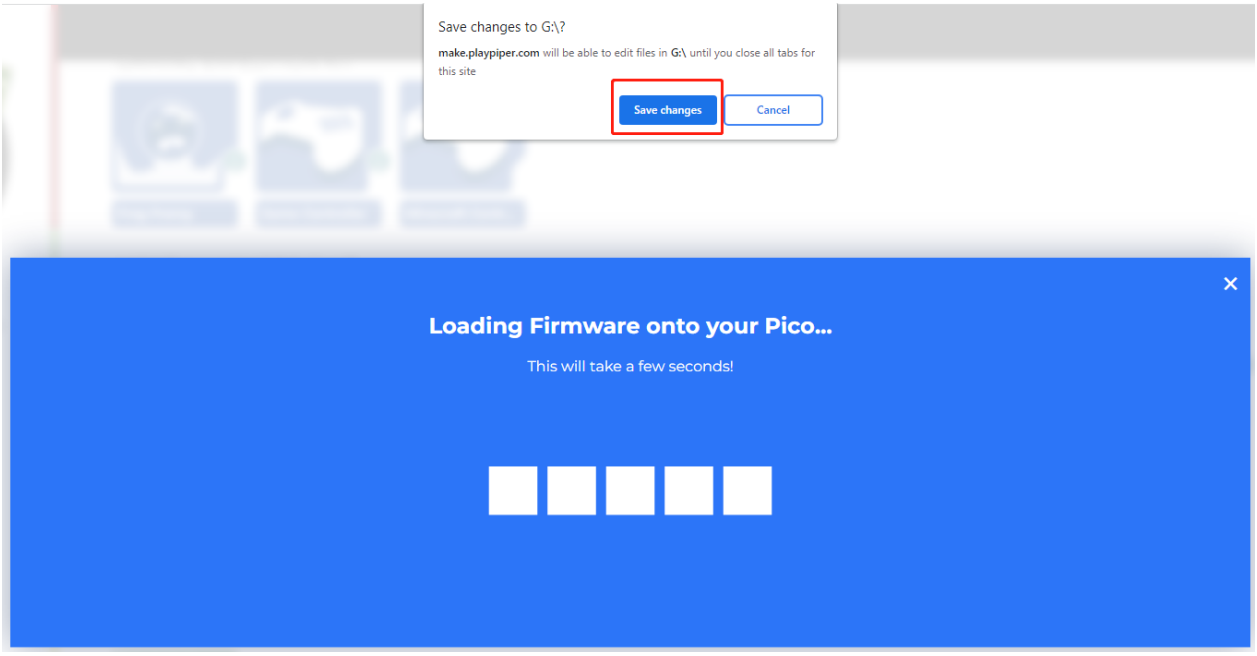


Pico W は USB ドライブとして表示されます。それが済んだら、 **Next** をクリックして、 **RPI-RP2** ドライブを選択します。

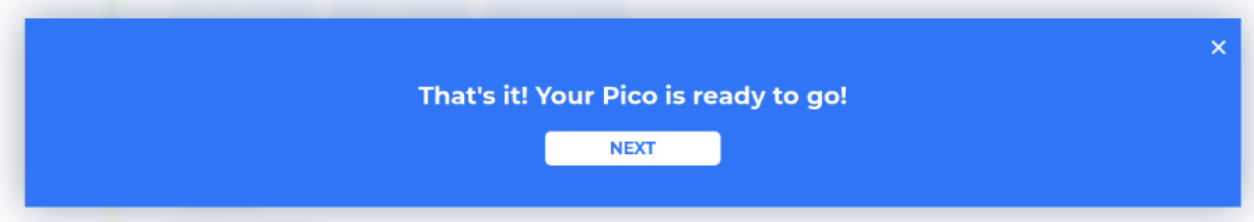
注釈: **RPI-RP2** ドライブを選択した後、Web ページがファイルを表示する許可をする必要があるポップアップウィンドウが上部に表示されます。



次に、Piper Make は Pico W にファームウェアをロードします。再度、Pico W が接続されているハードドライブに変更を保存する許可を与える必要があります。



このプロンプトが表示されたら、Pico W のセットアップが完了し、使用を開始できることを意味します。



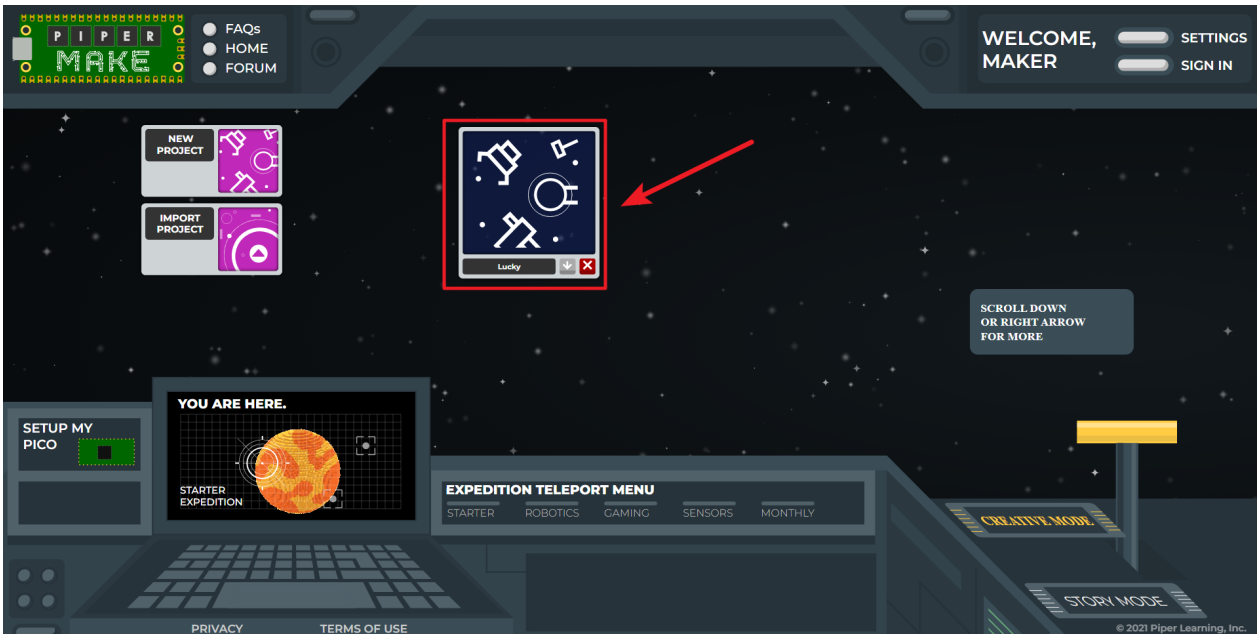
7.2 1.2 Piper Make のクイックガイド

7.2.1 1. 新しいプロジェクトを作成

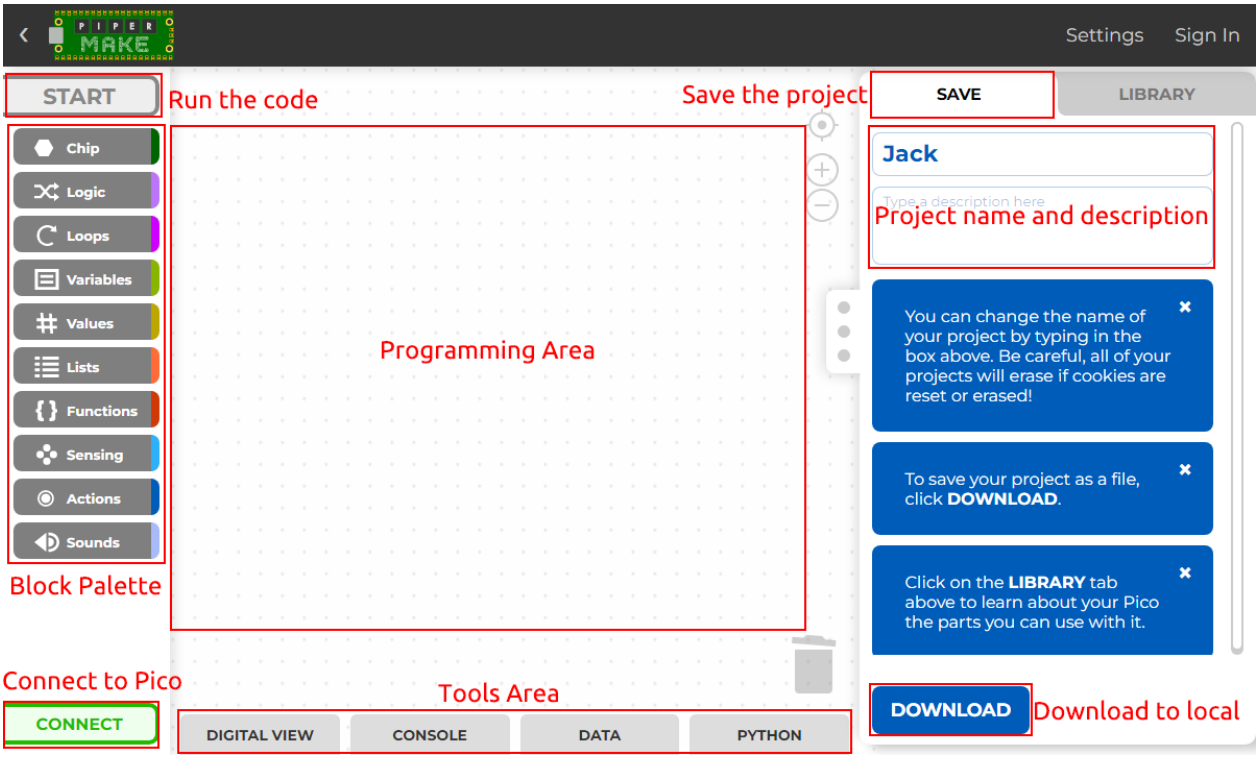
Pico W のセットアップが完了したら、プログラミングの方法を学ぶ時が来ました。それでは、オンボード LED を点灯させましょう。

`CREATIVE MODE` に切り替えて、 **新しいプロジェクト** ボタンをクリックします。すると、 **MY PROJECTS** セクションに新しいプロジェクトが表示され、プログラミングページから名前を変更できるランダムな名前が割り当てられます。

作成した新しいプロジェクトを開きます。



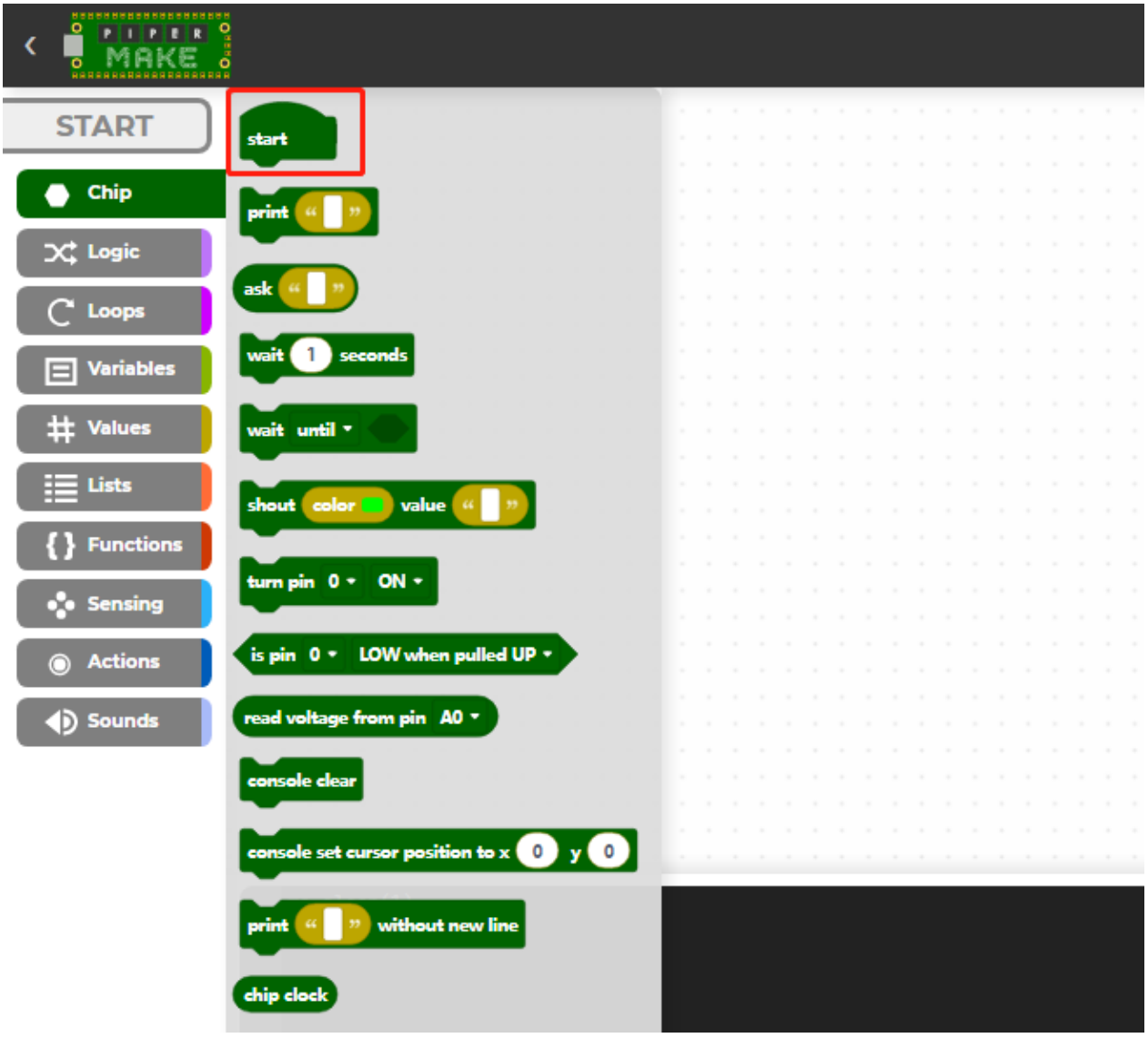
次に、Piper Make のプログラミングページに移動します。



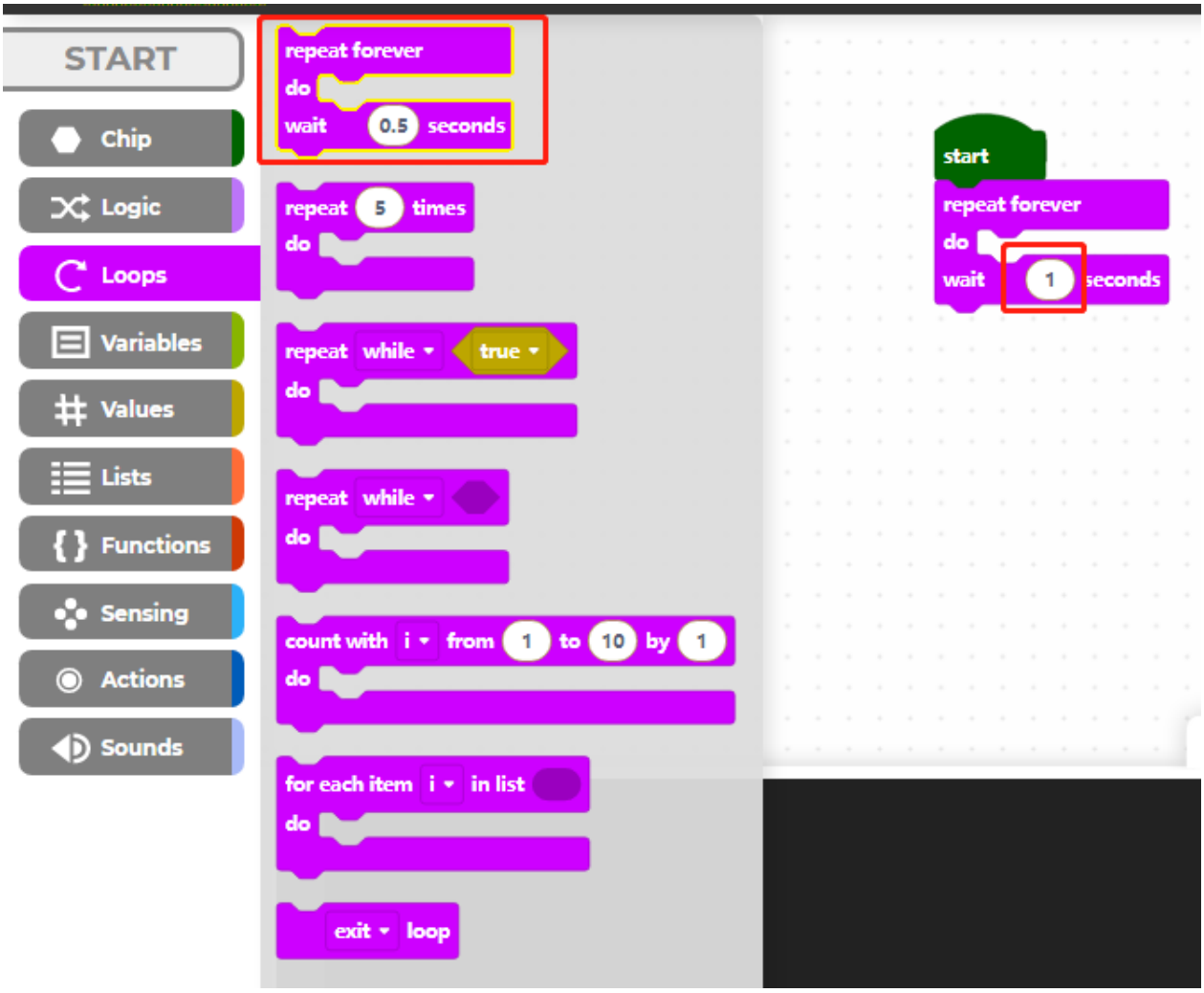
- **START**: コードを実行するために使用します。灰色の場合、現在 Pico W に接続されていません。
- **Block palette**: 様々な種類のブロックが含まれています。
- **CONNECT**: Pico W に接続するために使用します。Pico W に接続されていないときは緑色で、接続されると **DISCONNECT（赤）** になります。

- **Programming Area**: ここにブロックをドラッグしてプログラミングを完成させます。
- **Tools Area**: **DIGITAL VIEW**をクリックすると、Pico W のピン配置を見ることができます。 **CONSOLE** でプリント情報を表示でき、 **DATA** からデータを読み取り、 **Python** をクリックして Python ソースコードを表示できます。
- **プロジェクト名と説明**: プロジェクト名と説明を変更することができます。
- **DOWNLOAD**: **DOWNLOAD** ボタンをクリックしてローカルに保存できます。通常は **|** 形式です。次回は、ホームページの **Import Project** ボタンを使用してインポートできます。

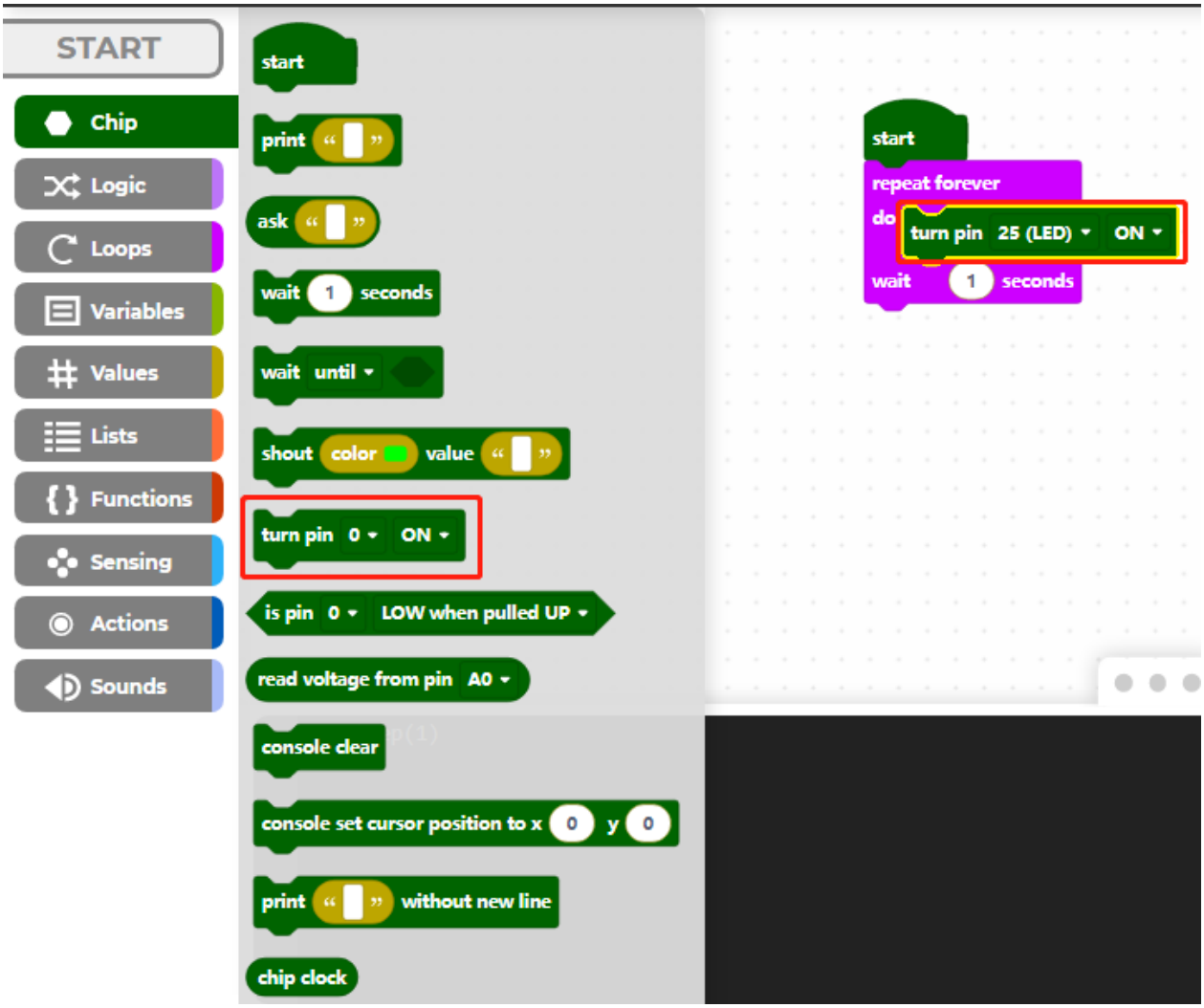
Chipパレットをクリックし、[start] ブロックを **Programming Area** にドラッグします。



次に、 **loops** パレットの [loop] ブロックを [start] ブロックの下にドラッグし、ループ間隔を 1 秒に設定します。

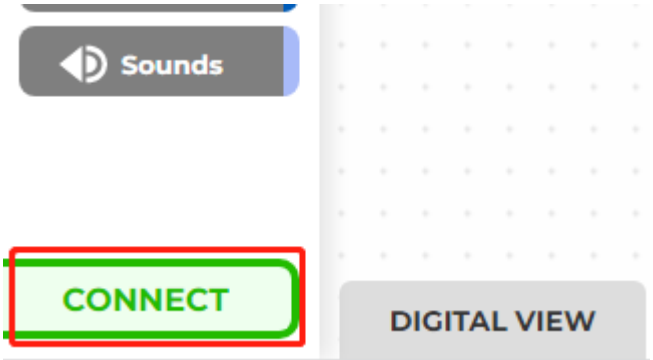


Raspberry Pi Pico のオンボード LED は pin25 にありますので、それを制御するために **Chip** パレットの [turn pin () ON/OFF] ブロックを使用します。

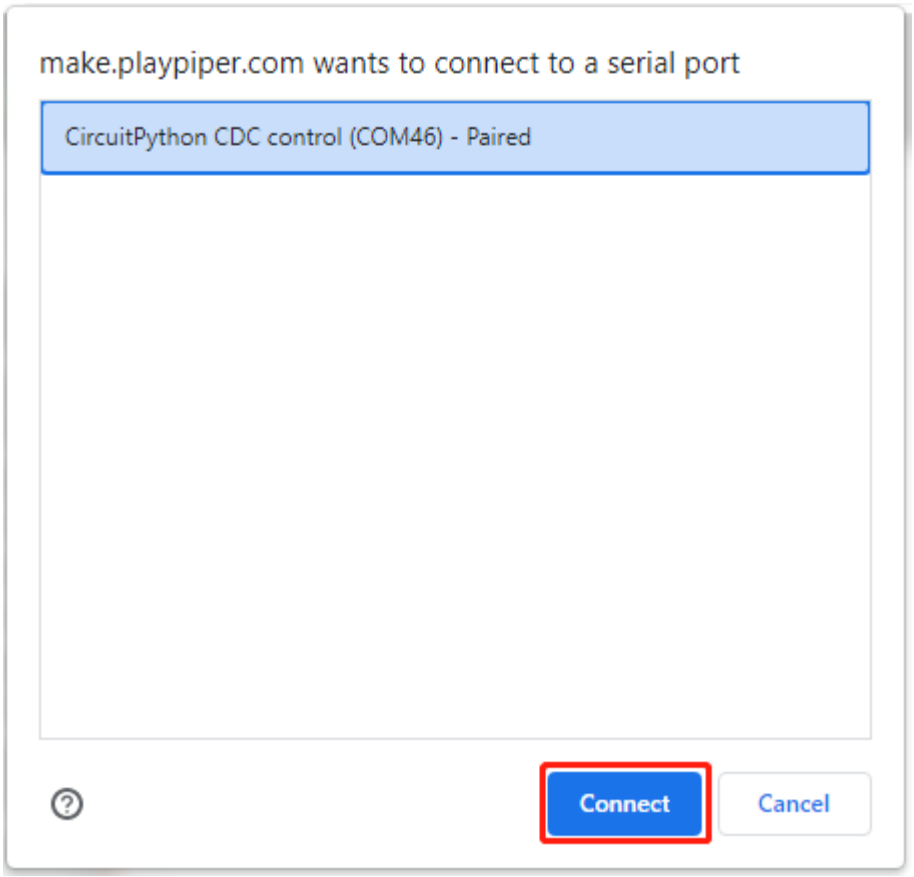


7.2.2 2. Pico W に接続

CONNECT ボタンをクリックして Pico W に接続します。クリックすると新しいポップアップが表示されます。



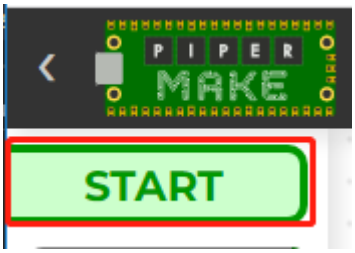
認識された **CircuitPython CDC control (COMXX)** ポートを選択し、 **Connect** をクリックします。



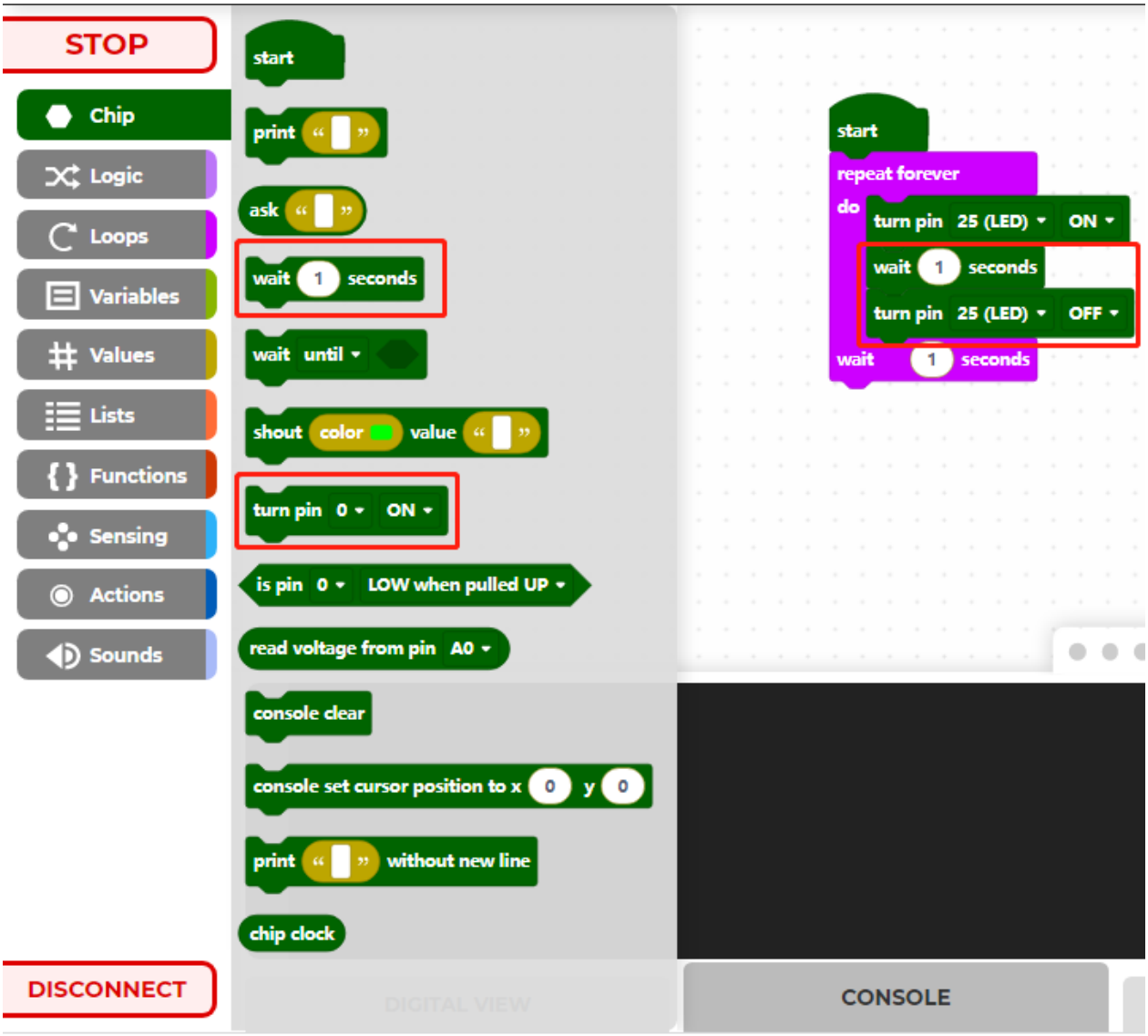
接続に成功すると、左下の緑色の **CONNECT** が赤い **DISCONNECT** に変わります。

7.2.3 3. コードを実行

START ボタンをクリックしてこのコードを実行します。Pico W の LED が点灯するはずです。もし灰色なら、Pico W が接続されていないので、再接続してください。



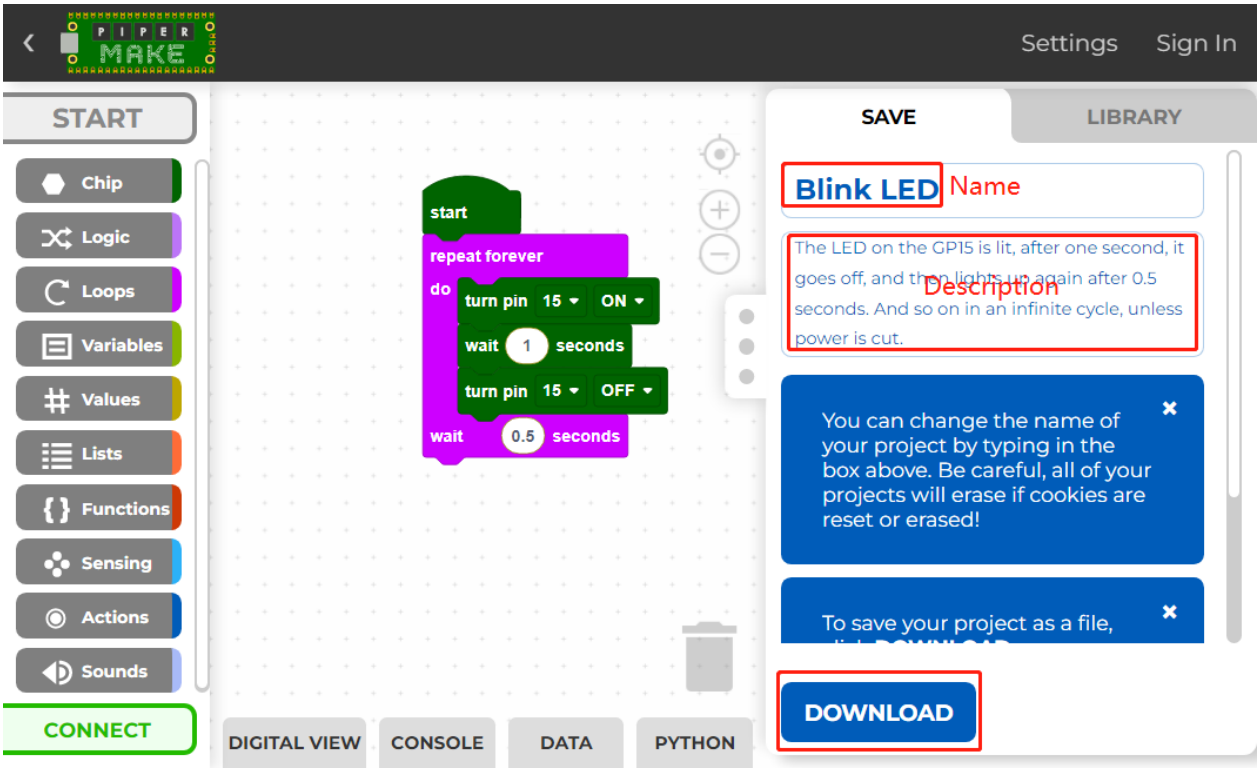
周期で pin25 を毎秒オフにし、左上の **START** を再度クリックすると、オンボード LED が点滅するのが確認できます。



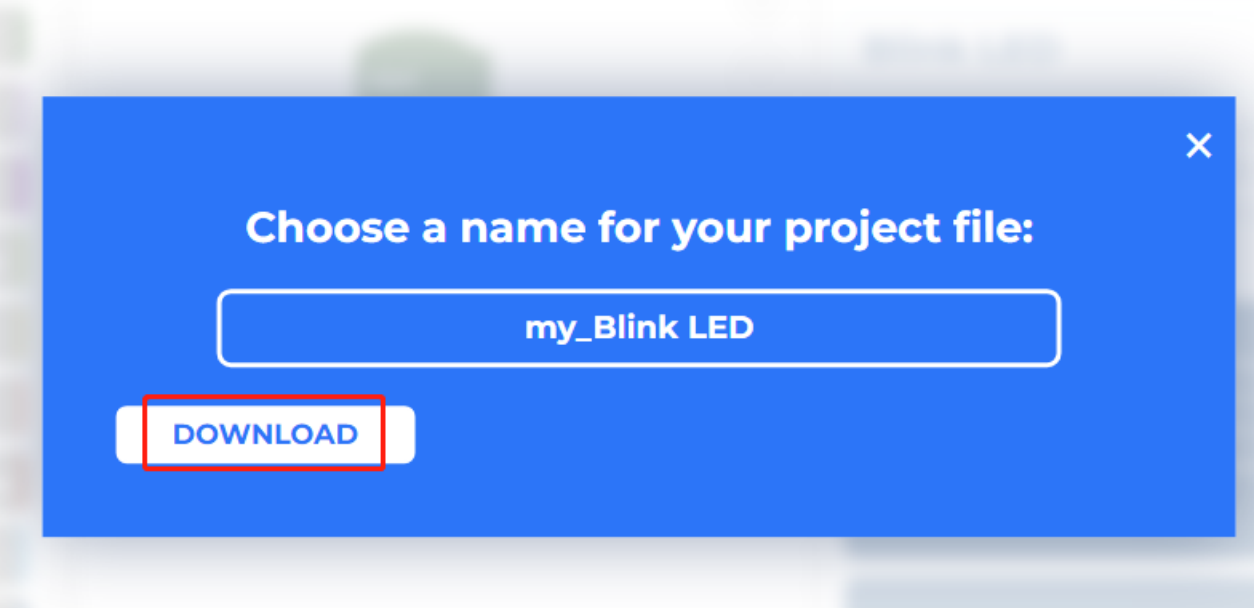
7.3 1.3 コードの保存やインポート方法は？

7.3.1 コードを保存

コードを書き終わった後、コードの名前と説明を変更できます。その後、 **Download** ボタンをクリックして、コードをローカルに保存するか他の人と共有することができます。

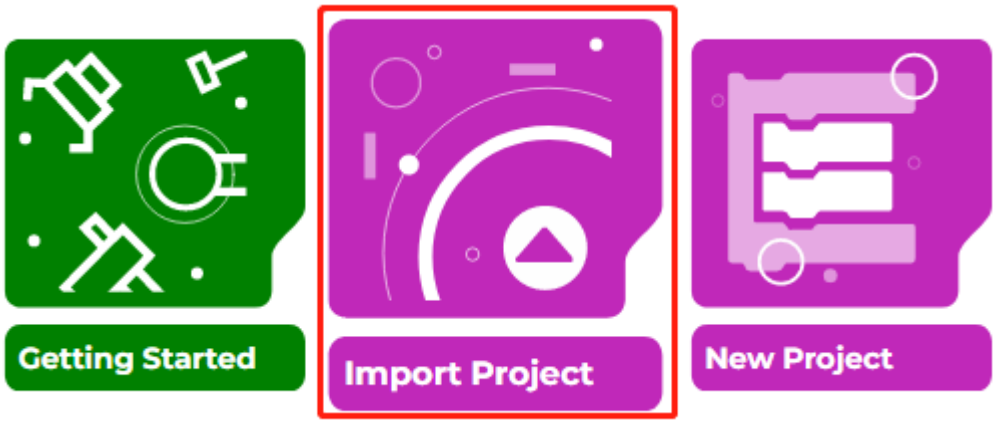


次に、ファイル名を入力し、 **Download** ボタンを再度クリックして、コードを .png ファイルとして保存します。

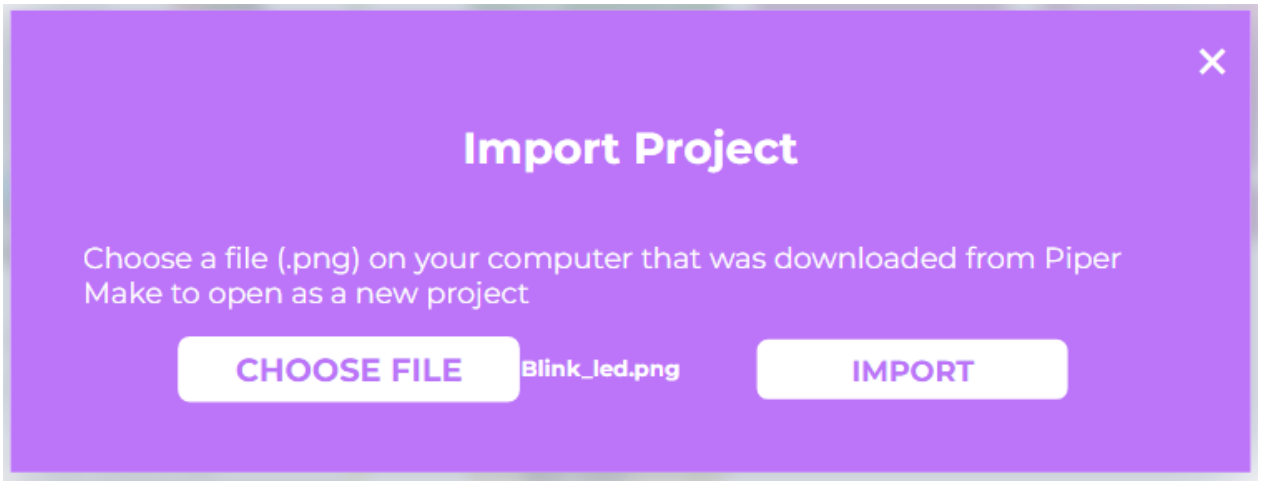


7.3.2 コードをインポート

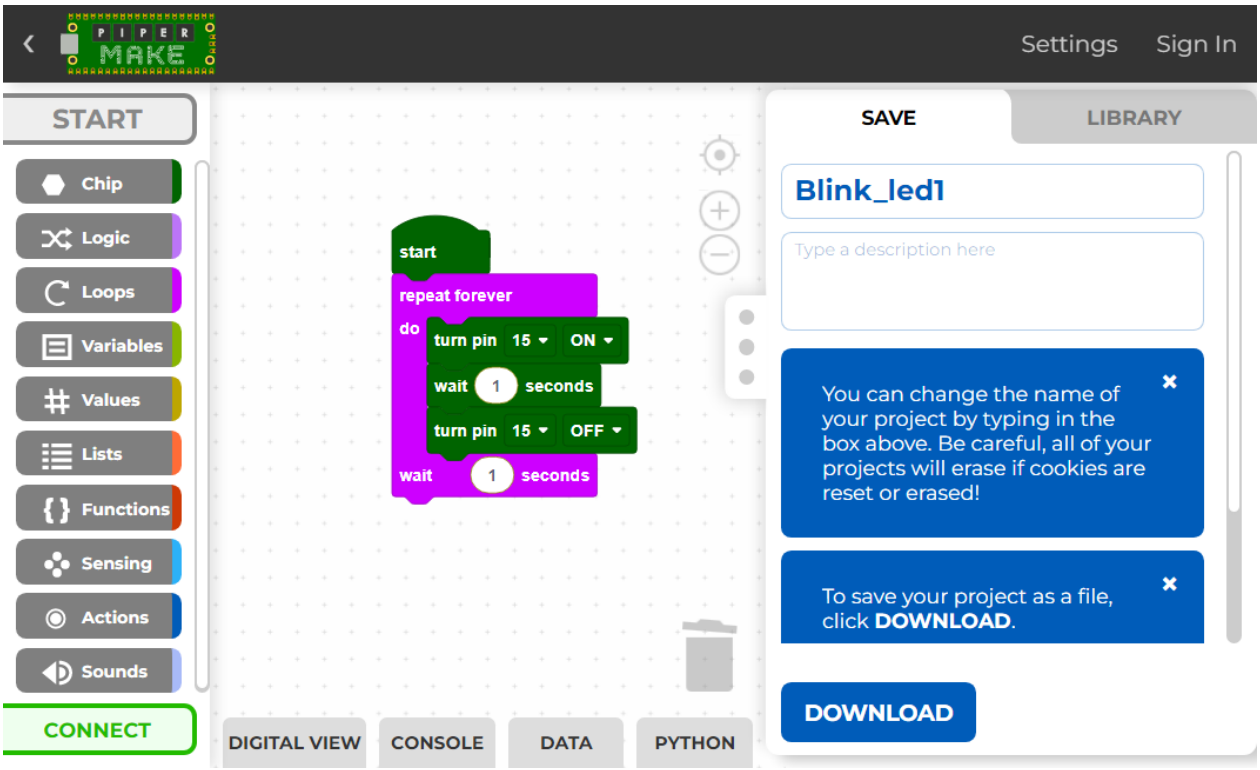
Piper Make の ホームページ で、 **Import Project** をクリックします。



`kepler-kit-main\piper` のパス内で `.png` ファイルを選択し、 **Import** をクリックします。注意：まず、 SunFounder Kepler Kit パッケージをダウンロードする必要があります。または、 Kepler Kit - GitHub でコードを確認できます。



これで、インポートしたファイルが表示されます。



2. プロジェクト

7.4 2.1 LED の点滅

このプロジェクトでは、拡張 LED ライトを点滅させることを目的としています。拡張電子部品を使用するには、はんだなしのブレッドボードが初心者にとって最も強力なパートナーとなるでしょう。

ブレッドボードは、多数の小さな穴が開いた長方形のプラスチック板です。これらの穴によって、簡単に電子部品を挿入し、電子回路を組むことができます。ブレッドボードは電子部品を永久に固定しないため、回路を修理したり、失敗した場合にやり直すことが容易です。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

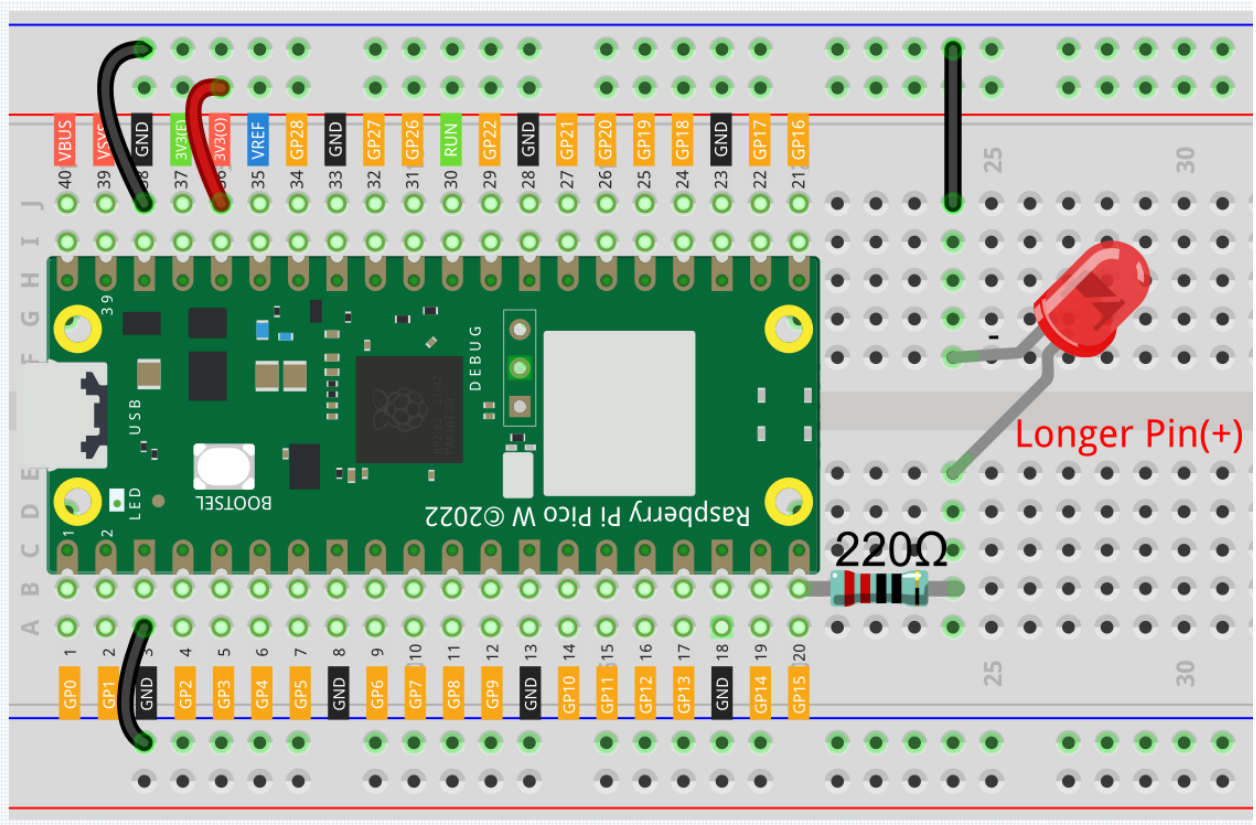
一式を購入する方が確実に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

SN	部品	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1（220 Ω）	
6	*LED*	1	

配線

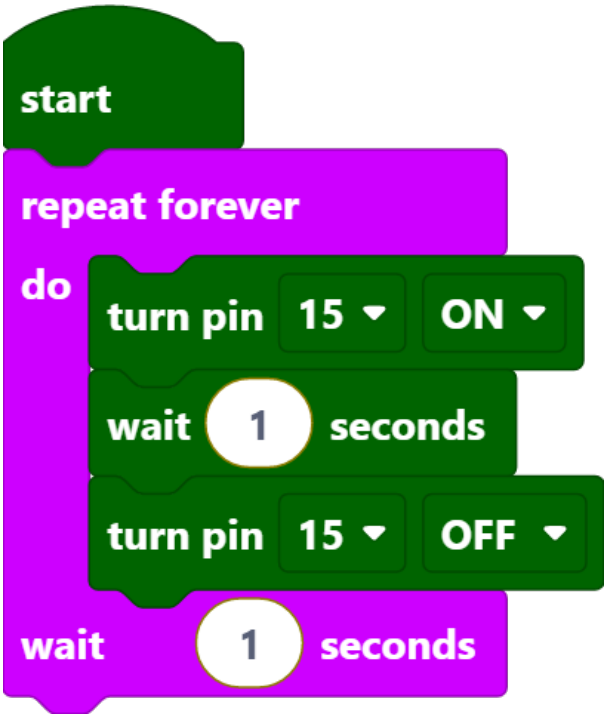


- 220 オームの抵抗のカラーリングは、赤、赤、黒、黒、茶です。
- LED の長いリードはアノード（+)、短いリードはカソード（-）として知られています。

コード

注釈:

- 下の画像を参考に、ドラッグアンドドロップでコードを書くことができます。
- 詳細なチュートリアルについては、 **コードをインポート** を参照してください。



Pico W に接続した後、 **スタート** ボタンをクリックすると LED が点滅します。詳細については、 *1.2 Piper Make のクイックガイド* を参照してください。

仕組み

これがループの本体です：ピン 15 をオンにして LED を点灯し、1 秒待ってから、ピン 15 をオフにして LED を消灯します。1 秒待ってから前のサイクルを再実行するので、LED は点灯と消灯を交互に繰り返している状態になります。

- [start]: このブロックはプログラムの基本的なフレームワークであり、プログラムの開始を表しています。
- [repeat forever do() wait()seconds]: これに含まれるブロックが繰り返し実行され、実行時間間隔は自分で定義します。
- [turn pin () ON/OFF]: 特定のピンを高レベル（ON）または低レベル（OFF）にすることを示しています。
- [wait () seconds]: ブロック間の実行間隔を設定します。

7.5 2.2 ボタン

このプロジェクトでは、ボタンを使用して LED をオンまたはオフにする方法を学びます。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

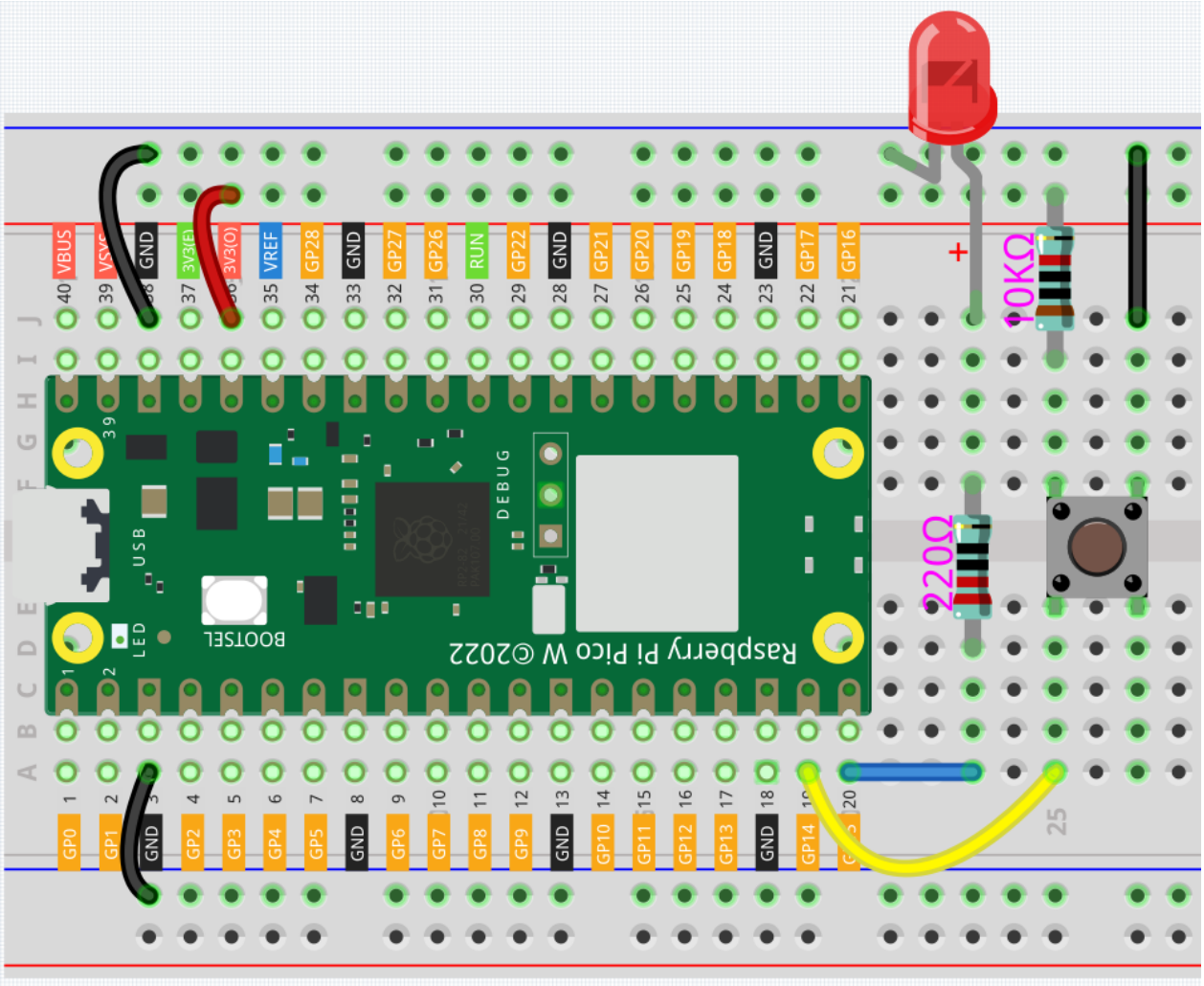
一式を購入する方が確実に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

SN	部品	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	2（220 Ω，10K Ω）	
6	*LED*	1	
7	**ボタン**	1	

配線

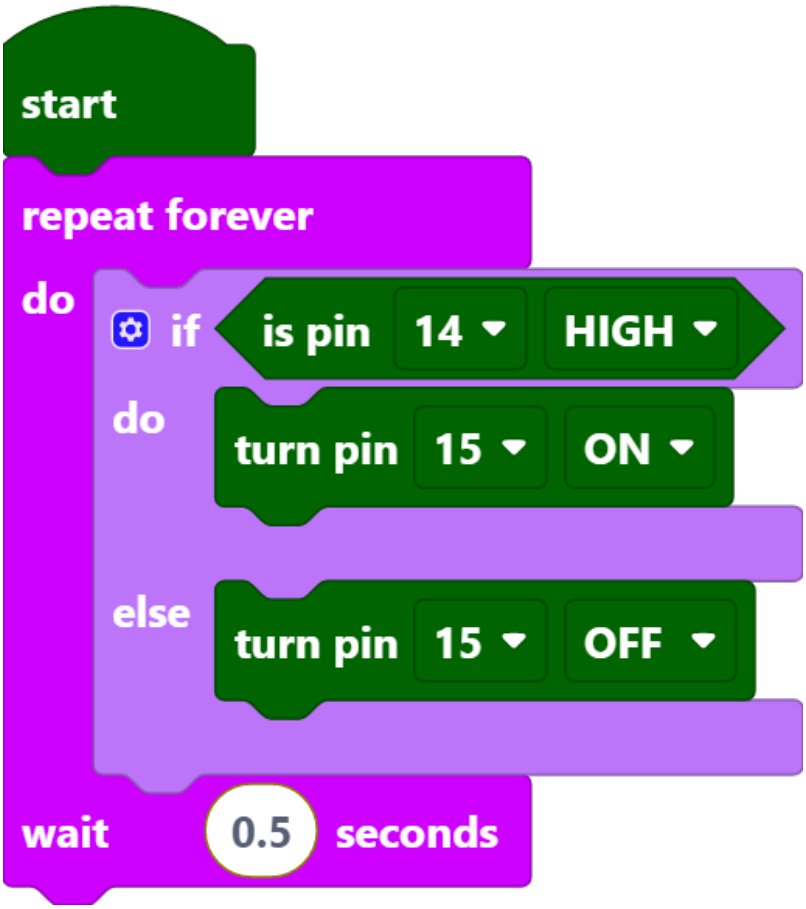


- ボタンのピンの一方は 3.3V に、もう一方は GP14 に接続されています。ボタンが押されると、GP14 は高レベルになります。
- しかし、ボタンが押されていないとき、GP14 はサスペンド状態にあり、高レベルでも低レベルでもありえます。ボタンが押されていないときに安定した低レベルを得るために、GP14 は 10K プルダウン抵抗を介して GND に再接続する必要があります。

コード

注釈:

- 下の画像を参考に、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.2_button.png` をインポートします。詳細なチュートリアルは、**コードをインポート** を参照してください。



Pico W に接続した後、 **スタート** ボタンをクリックするとコードが実行されます。ボタンが押されると、LED が点灯します。ボタンを離すと、LED が消灯します。

仕組み

ボタンが押されると、ピン 14 は高レベルになります。したがって、読み取ったピン 14 が高レベルであれば、ピン 15 をオンにして（LED を点灯）、それ以外の場合はピン 15 をオフにします（LED を消灯）。

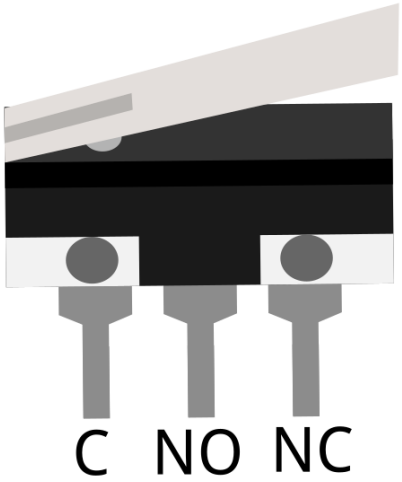
- [if () do () else ()]: これは判断ブロックであり、[if] ブロックの後の条件に応じて、[do] ブロック内または [else] ブロック内のブロックを実行するかどうかを決定します。
- [is pin () HIGH]: これは特定のピンのレベルを読むために使用されます。読み取ったレベルが設定した HIGH/LOW と同じ場合、[do] ブロック内のブロックを実行し、それ以外の場合は [else] ブロック内のブロックを実行します。

7.6 2.3 サービスベル

このプロジェクトでは、マイクロスイッチとアクティブブザーを使用してサービスベルを作成します。スイッチをタップすると、ブザーが音を出します。

マイクロスイッチも 3 ピンのデバイスであり、3 ピンの順番は C（共通ピン）、NO（通常開）および NC（通常閉）です。

マイクロスイッチが押されていない場合、1（C）と 3（NC）は接続されています。押されると、1（C）と 2（NO）が接続されます。



必要な部品

このプロジェクトで必要な部品は以下の通りです。

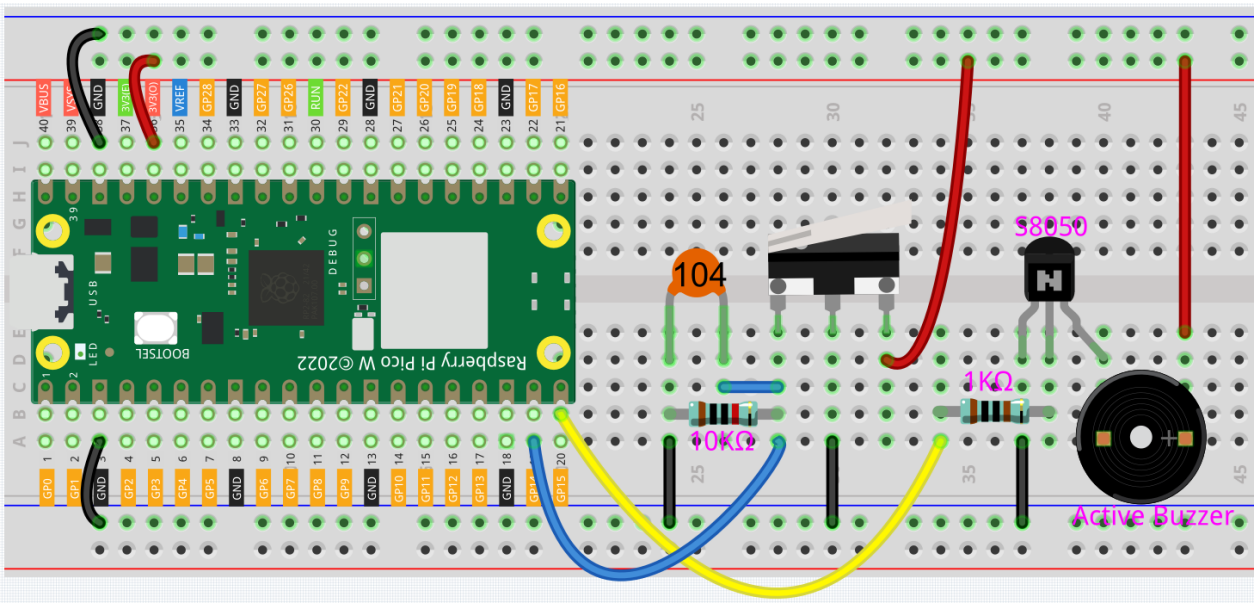
一式を購入する方が確実に便利です、リンクはこちらです：

名前	このキットに含まれるアイテム	リンク
ケプラーキット	450 以上	

以下のリンクから個別にも購入可能です。

SN	部品	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	2(1K Ω, 10K Ω)	
6	**コンデンサ**	1(104)	
7	**マイクロスイッチ**	1	
8	**トランジスタ**	1(S8050)	
9	アクティブ **ブザー**	1	

配線

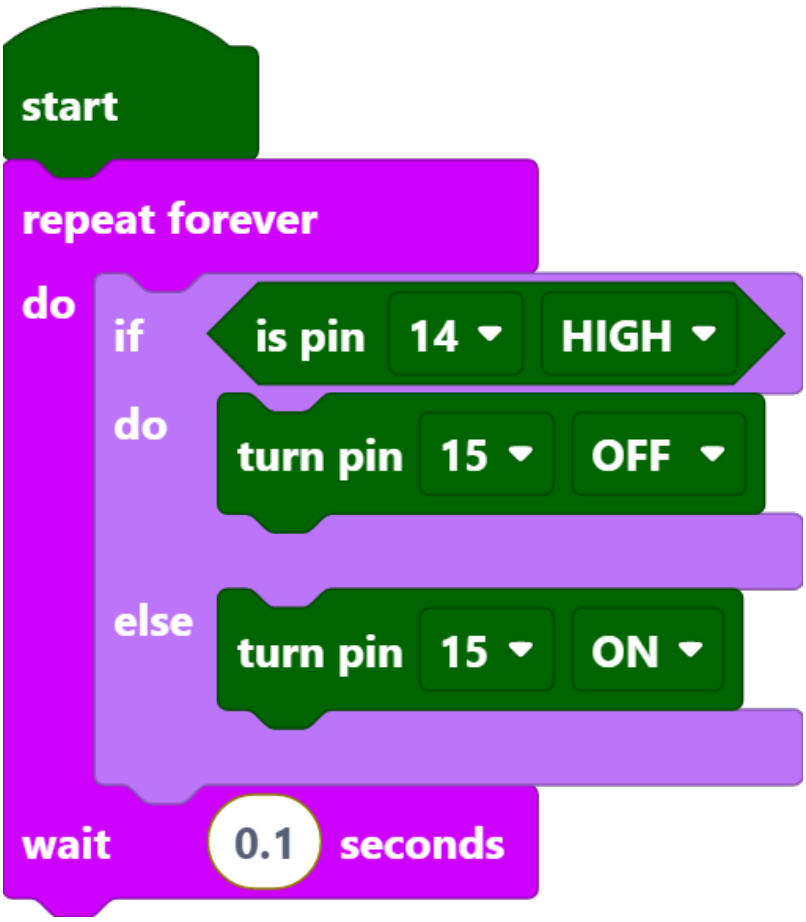


- デフォルトでは、マイクロスイッチのピン 1 と 3 は接続されており、GP14 は低い状態です。マイクロスイッチが押されると、GP14 は高い状態になります。
- GP15 は高い状態を出力して、ブザーが音を出します。

コード

注釈:

- 下の画像を参考にして、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.3_service_bell.png` をインポートしてください。詳細なチュートリアルについては、 **コードをインポート** を参照してください。



Pico W に接続した後、 **スタート** ボタンをクリックしてコードが実行されます。スイッチをタップすると、ブザーが音を出します。

注釈: このプロジェクトのコードは、前のプロジェクト *2.2* **ボタン** とまったく同じです。

7.7 2.4 レインボーライト

このプロジェクトでは、RGB LED を使って虹のような色彩を表示させます。

RGB LED は、赤・緑・青の LED を 1 つのランプキャップの下に封入するもので、3 つの LED は共通のカソードピンを共有しています。各アノードピンに電気信号が供給されるので、対応する色の光が表示されます。各アノードの電気信号の強度を変更することで、さまざまな色を生成することができます。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

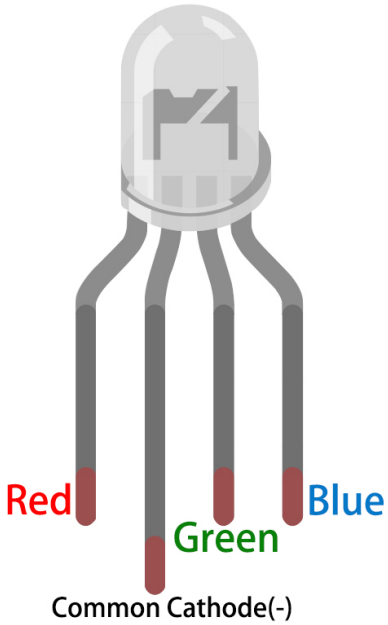
こちらのリンクでキット全体を購入すると便利です：

名前	キット内容	リンク
ケプラーキット	450+	

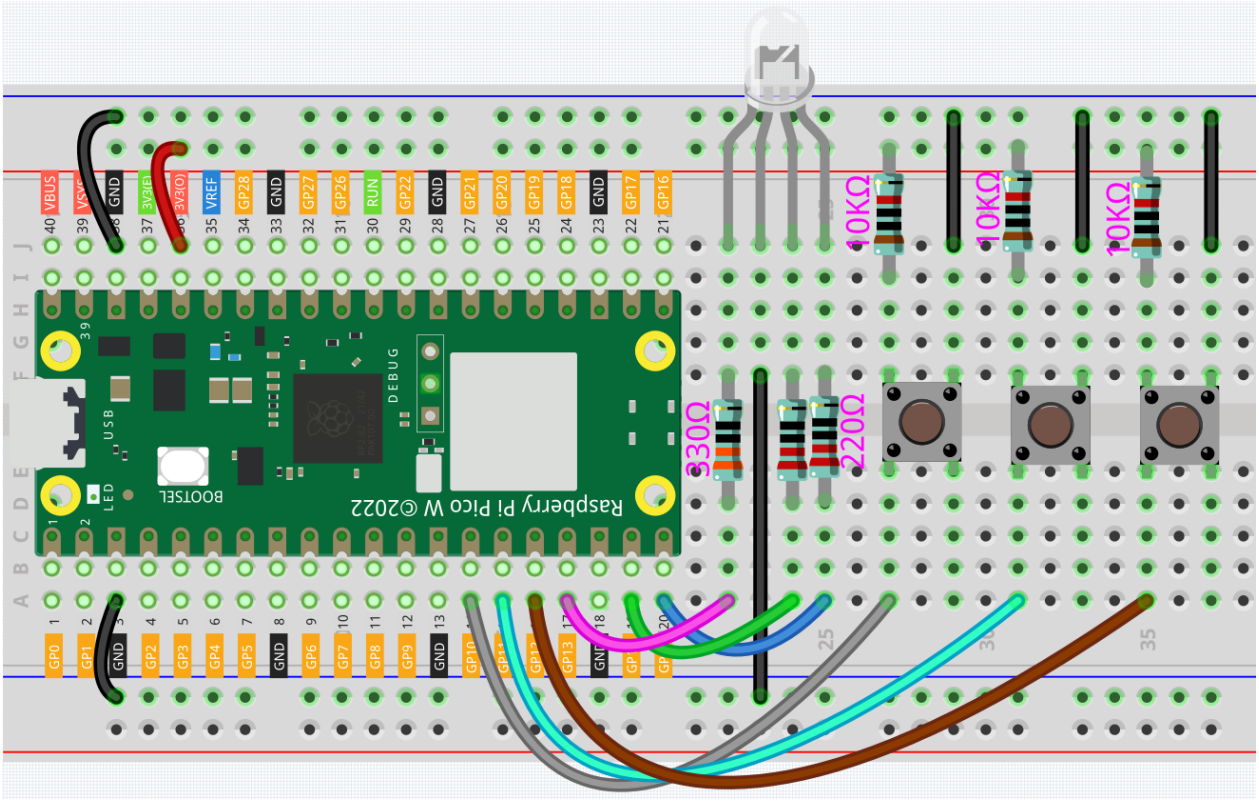
以下のリンクから個別に購入することもできます。

番号	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	6(1-330 Ω, 2-220 Ω, 3-10K Ω)	
6	**ボタン**	3	
7	*RGB LED*	1	

配線



RGB LED には 4 本のピンがあります：最も長いピンは共通のカソードピンで、通常 GND に接続されます。最も長いピンの隣の左側のピンは赤で、右側の 2 本のピンは緑と青です。

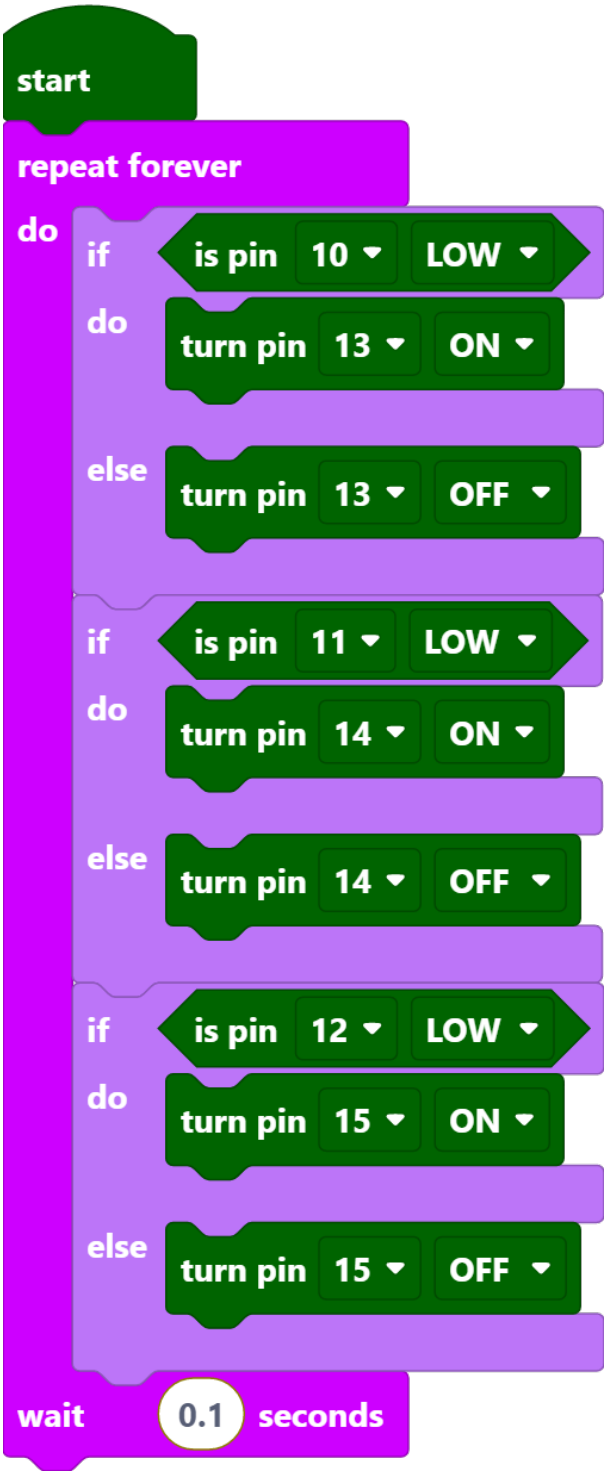


- 同じ電源強度を使用すると、赤い LED は他の 2 つよりも明るくなります。その明るさを減らすために、やや大きな抵抗器（330 Ω）を使用する必要があります。
- 3 つのボタンはそれぞれ赤、緑、青の LED の点灯を制御するために使用されます。

コード

注釈:

- 下の画像を参考にして、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.4_rainbow_light.png` をインポートしてください。詳細なチュートリアルについては、 **コードをインポート** を参照してください。



Pico W に接続した後、 **スタート** ボタンをクリックするとコードが実行されます。各ボタンを個々に押すと単色の光が放たれ、2 つのボタンが同時に押された場合、または 3 つのボタンが同時に押された場合、RGB LED は最大 7 色のさまざまな色を放ちます。

注釈: 実際には、RGB LED は最大 1600 万色を放つことができますが、Piper Make には PWM 信号を出力するた

めのブロックがないため、ここでは [turn pin() (ON/OFF)] ブロックを使用して RGB LED が 7 色を表示するようにしています。

仕組み

このプロジェクトは、三つのボタンで RGB LED を制御すると考えることができます。三つの if 判断条件を設定して、三つのボタンが押されているかどうかを判断します。ボタンが押されると、対応するピンのレベルが高くなり、RGB LED がさまざまな色を表示するようになります。

7.8 2.5 ドラムキット

このプロジェクトでは、3 つのボタンとスライドスイッチを使ってドラムキットを作りましょう。さあ、自分だけのドラムを演奏してみてください。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

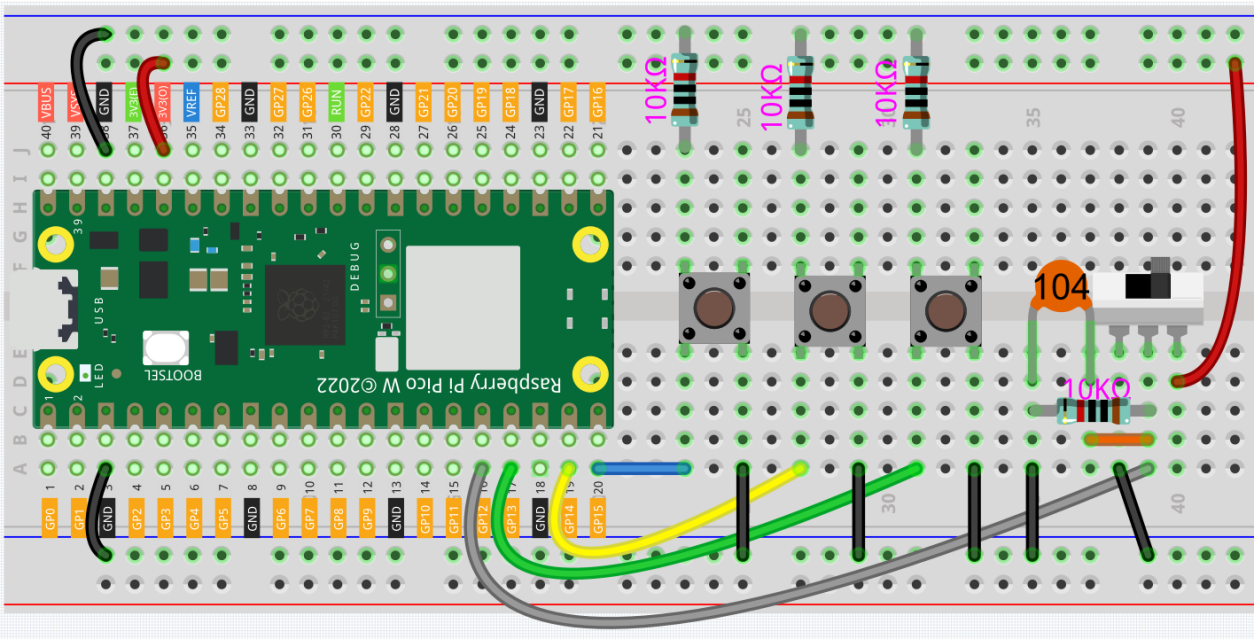
こちらのリンクでキット全体を購入すると便利です：

名前	キット内容	リンク
ケプラーキット	450+	

以下のリンクから個別に購入することもできます。

番号	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	4(10K Ω)	
6	**ボタン**	3	
7	**コンデンサ**	1(104)	
8	**マイクロスイッチ**	1	

配線



- スライドスイッチを右に切り替えると、GP12 は高い状態になります。左に切り替えると、GP12 は低い状態になります。
- 3 つのボタンはそれぞれプルダウン抵抗器に接続され、GP13～GP15 はデフォルトで低い状態です。ボタンが押されると、GP13～GP15 は高い状態になります。

コード

注釈:

- 下の画像を参考にして、ドラッグ＆ドロップでコードを作成できます。
- `kepler-kit-main\piper` のパスから `2.5_drum_kit.png` をインポートしてください。詳細なチュートリアルは、 **コードをインポート** を参照してください。

Pico W に接続した後、 **スタート** ボタンをクリックするとコードが実行されます。異なるボタンを押したり、スライドスイッチを切り替えたりすると、ドラムキットのようにさまざまなドラムの音が鳴ります。

注釈: コンピュータを使用している場合は、発生した音を聞くためにヘッドホンまたはオーディオをコンピュータに接続する必要があります。

7.9 2.6 スマートウォータータンク

このプロジェクトでは、水位センサーモジュールとサーボを使用してスマートウォータータンクをシミュレートします。水位センサーはタンク内に固定され、水位が閾値以下になった場合に、サーボによって制御されるバルブが開き、水が流れ込むようになります。

必要なコンポーネント

このプロジェクトには、以下のコンポーネントが必要です。

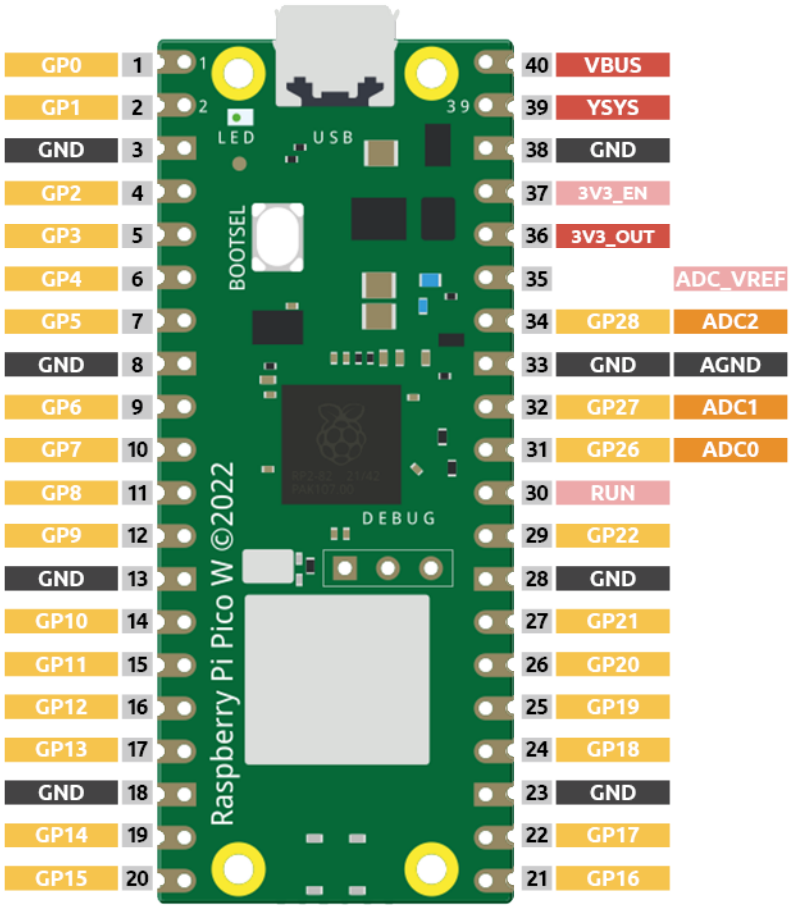
キット全体を購入すると便利です。リンクはこちら：

名前	キット内容	リンク
ケプラーキット	450+	

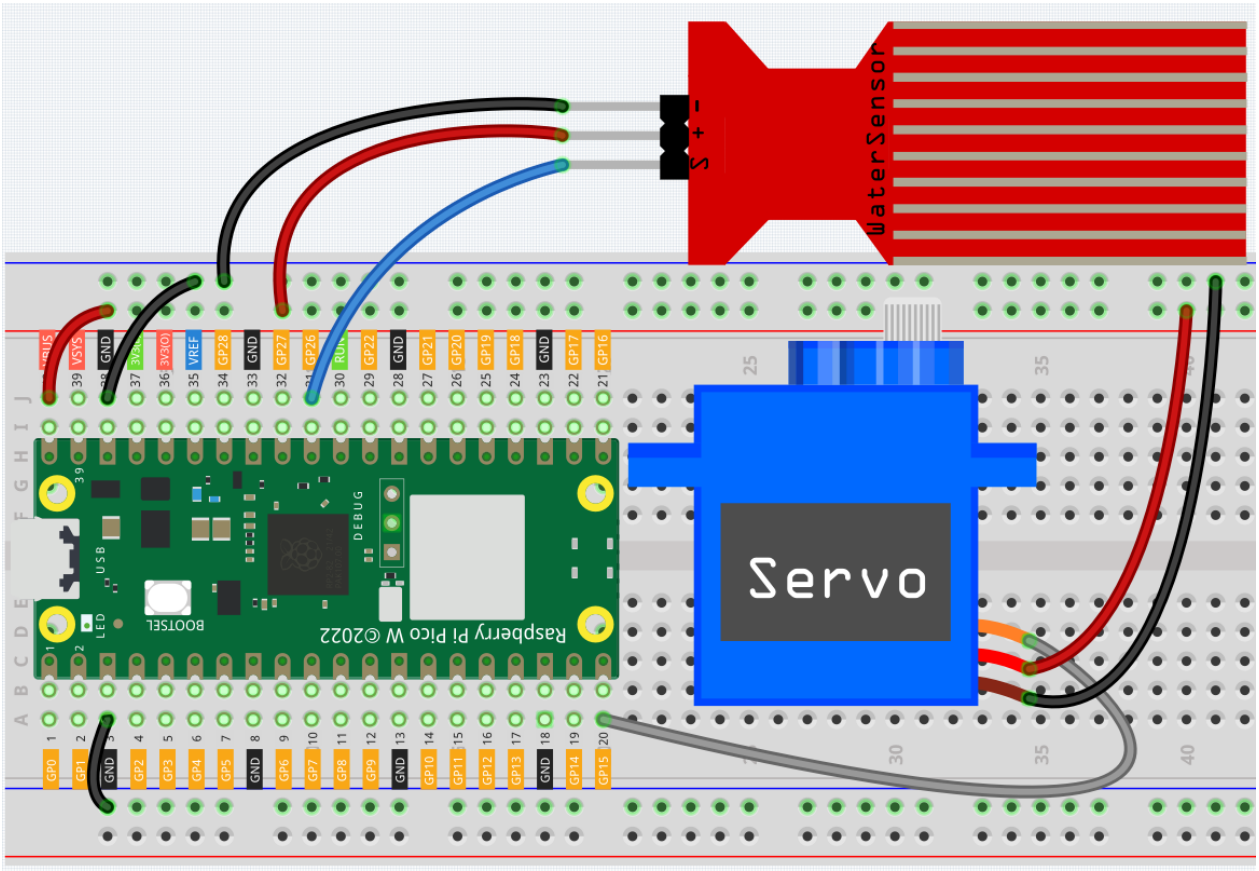
以下のリンクから個別に購入することもできます。

番号	コンポーネント	個数	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	
6	**水位センサーモジュール**	1	

配線



Pico W には、アナログ入力を使用できる GPIO ピンが 3 つあります。それは、アナログチャネル 0、1、および 2 の GP26、GP27、GP28 です。また、内蔵温度センサーに接続されている第 4 のアナログチャネルもありますが、ここでは紹介しません。

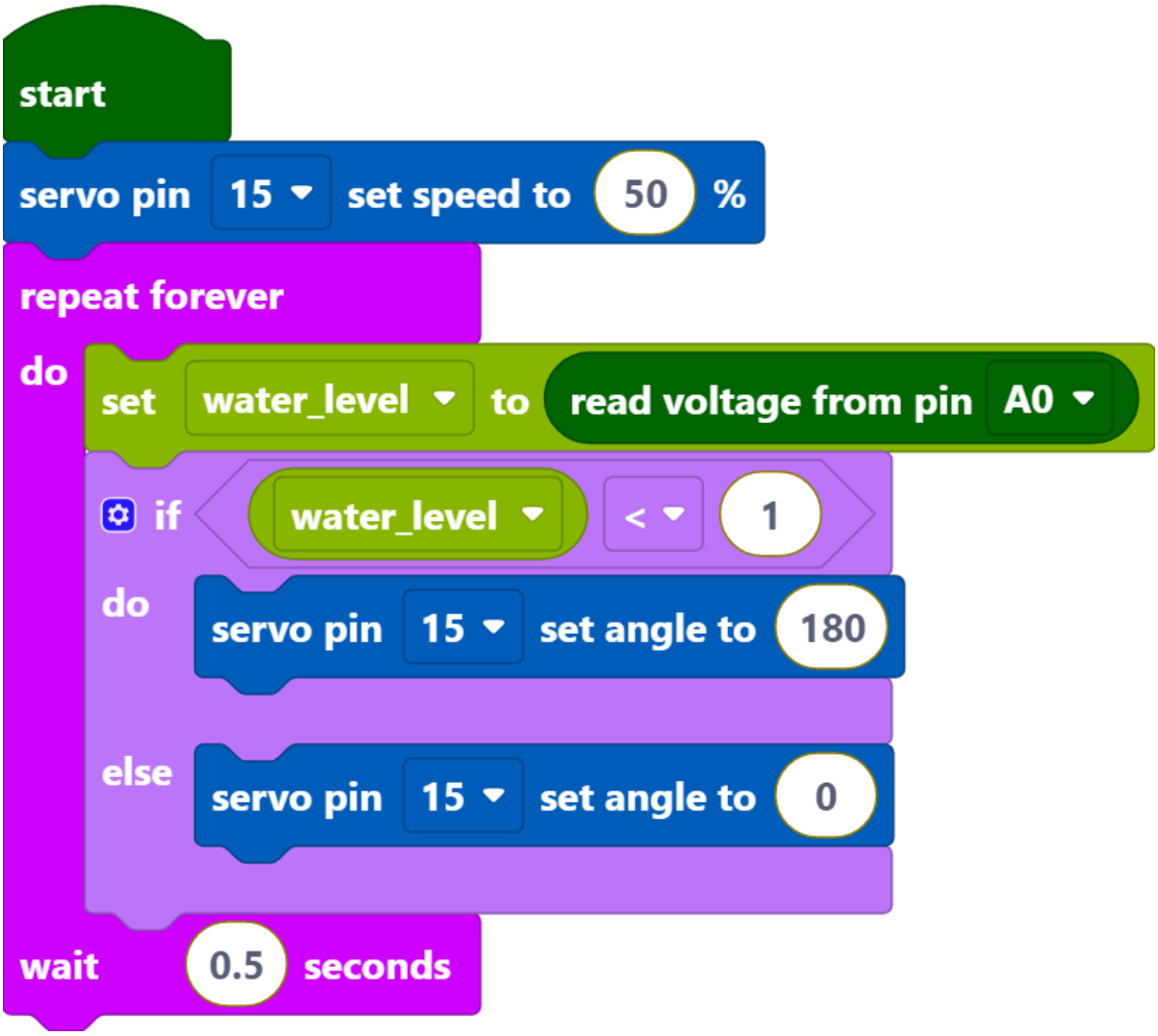


- 水位センサーの S は GP26（A0）に、+ は VBUS に、-ピンは GND に接続されています。
- サーボのオレンジ色のワイヤー（信号）は GP15 に、赤色のワイヤー（電源）は VBUS に、茶色のワイヤー（接地）は GND に接続されています。

コード

注釈:

- 下の画像を参照して、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.6_water_tank.png` をインポートしてください。詳細なチュートリアルについては、 **コードをインポート** を参照してください。



Pico W に接続した後、 **スタート** ボタンをクリックしてコードが実行されます。水位が水位センサーの 1/3 より低い場合、サーボは 180 度回転して入口が開くようにシミュレートされます。水位が水位センサーの 1/3 より高い場合、サーボは 0 度に回転して入口が閉じるように模倣されます。

動作原理

ピン 15（サーボ）の回転速度を 15% に設定します。

- [servo pin() set speed to ()%]：サーボピンの回転速度を設定するために使用されます。範囲は 0%～100% です。

ピン A0 の値を読み取り、変数 [water_level] に格納します。

- [set (water_level) to]：変数の値を設定するために使用されます。変数を **Variables** パレットから作成する必要があります。
- [read voltage from pin ()]：アナログピン（A0～A2）の電圧を読み取るために使用されます。範囲は 0～3.3V です。

電圧の閾値を 1 に設定します。水位センサーの電圧が 1 より小さい場合、サーボを 180 ° の位置に回転させます。それ以外の場合は、0 ° の位置に回転させます。

- [servo pin () set angle to ()]：サーボピンの角度を設定します。範囲は 0～180 ° です。

7.10 2.7 スイングサーボ

このプロジェクトでは、サーボとポテンショメーターを使って、操舵装置を模倣します。ポテンショメーターを回転させると、サーボも連動して回転します。

必要な部品

このプロジェクトに必要な部品は以下の通りです。

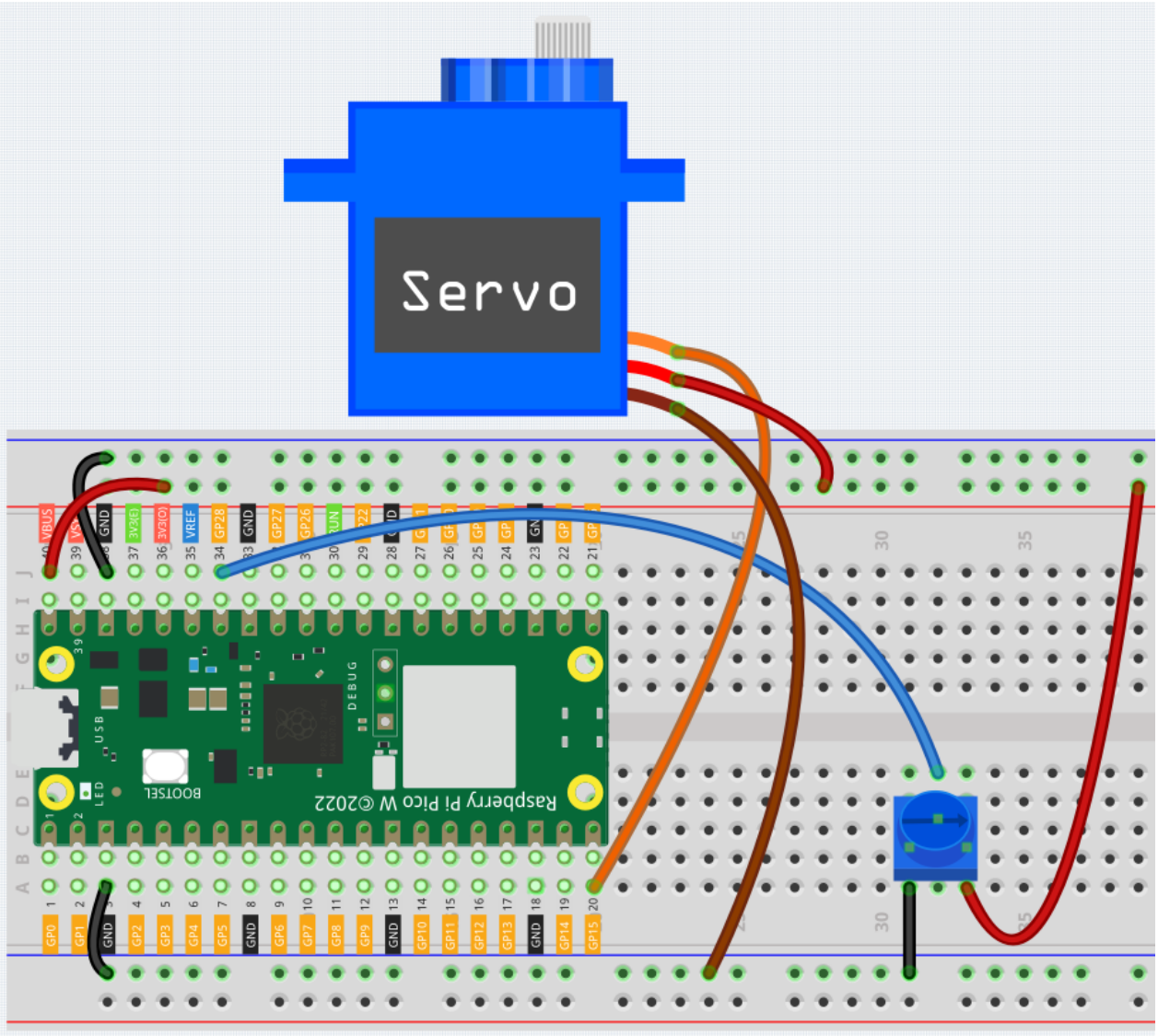
全ての部品が揃ったキットを購入すると便利です。リンクは以下です：

名称	キットの内容	リンク
ケプラーキット	450 以上	

下記のリンクから個別に購入することもできます。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	
6	**ポテンショメーター**	1	

配線方法

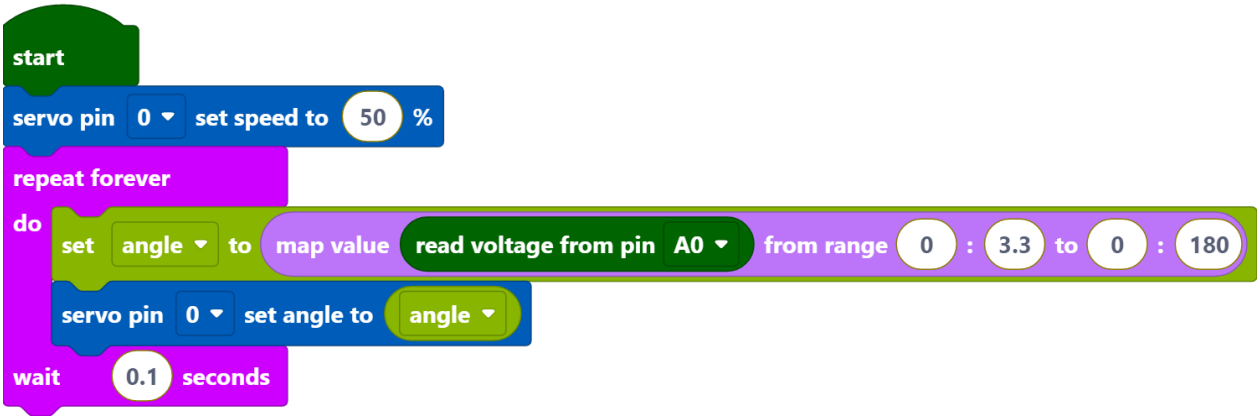


- サーボのオレンジ色の線（信号）は GP15 に、赤色の線（電源）は VBUS に、茶色の線（接地）は GND に接続します。
- ポテンショメーターは 3 端子の抵抗器で、両端のピンを 5V と GND に、中央のピンを GP26（A0）に接続します。

プログラム

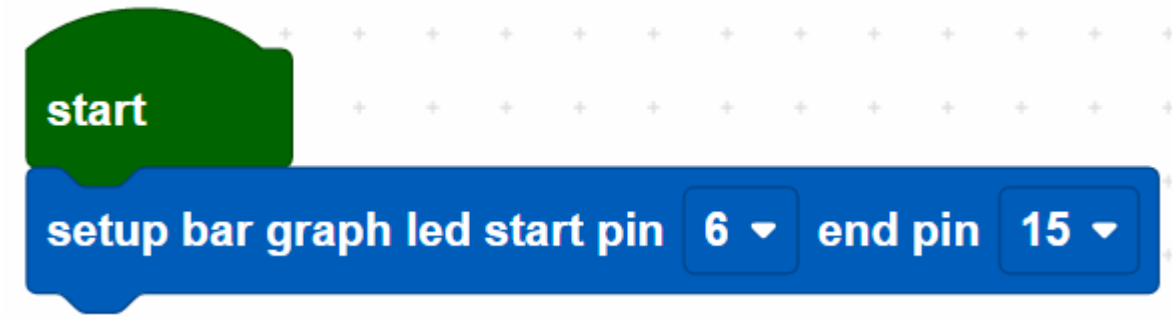
注釈:

- 下の画像を参考に、ドラッグ&ドロップでコードを作成できます。
- 詳しいチュートリアルは、 **コードをインポート** をご参照ください。



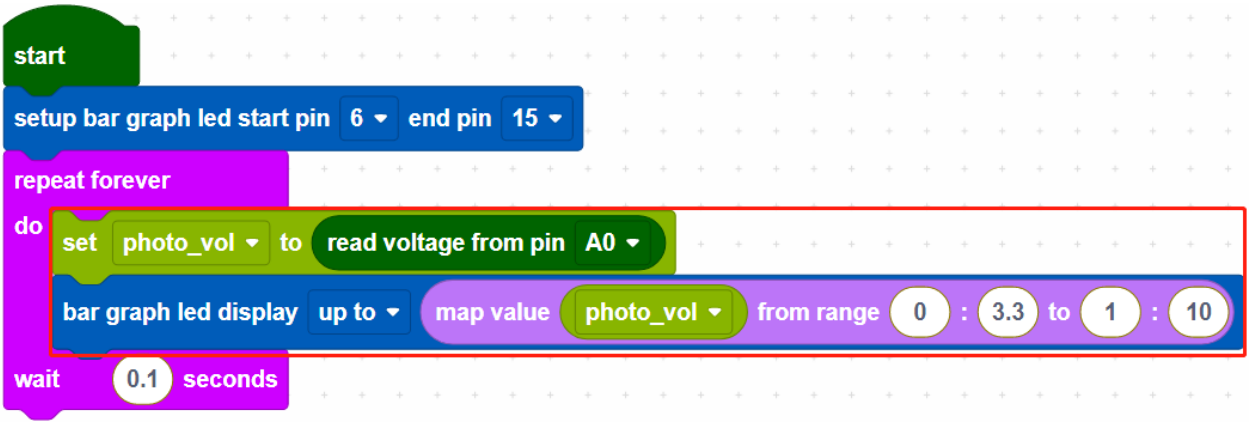
- Pico W を接続した後、 **スタート** ボタンを押してプログラムを実行します。
- ポテンショメーターを回転させると、サーボもそれに応じて動きます。明瞭に確認するためには、サーボの軸にロッカーアームを取り付けてください。

仕組み



ピン 15（サーボ）の回転速度を 15% に設定します。

- [servo pin() set speed to ()%]: サーボのピンの回転速度を設定するために使います。範囲は 0%~100% です。



変数 [angle] を作成し、A0 の電圧を読み取ります。[map value () from () to ()] ブロックを使って、A0 の電圧を 0 から 3.3V までの範囲にマッピングし、そのマッピングされた角度をサーボの回転角度として使用します。

- [map value () from () to ()]: 一つの範囲から別の範囲に値をマッピングします。

注釈: A0~A2 の電圧は、電源が VBUS（5V）に接続されていても、0~3.3V の範囲です。

7.11 2.8 光強度表示装置

このプロジェクトでは、フォトレジスタと LED バーグラフを使用して、光の強度に応じて LED バーグラフが点灯する光強度表示装置を作成します。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

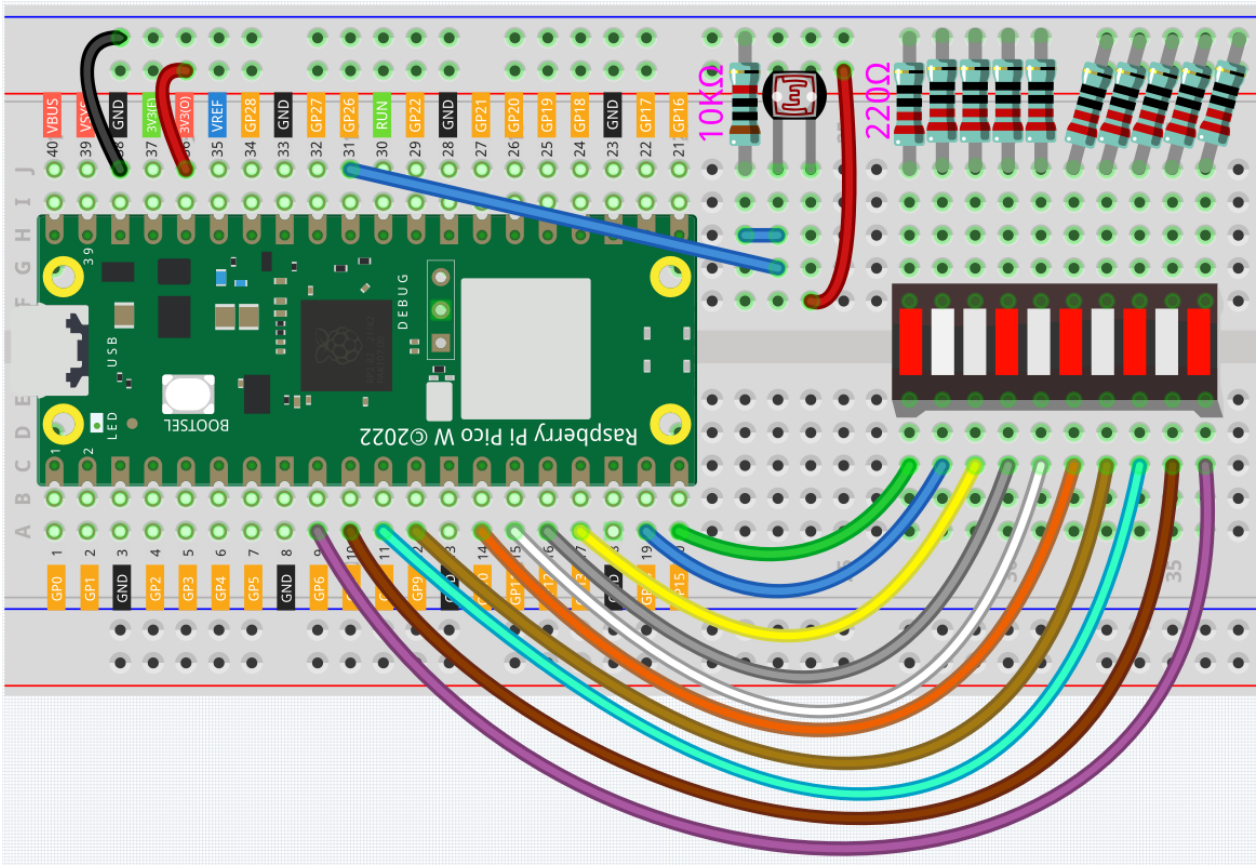
全てを一つのキットで購入すると便利です。リンクは以下です：

名称	キット内容	リンク
ケプラーキット	450 以上	

以下のリンクから個別に購入することもできます。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	11(10-220 Ω, 1-10K Ω)	
6	*LED* **バーグラフ**	1	
7	**フォトレジスタ**	1	

配線方法

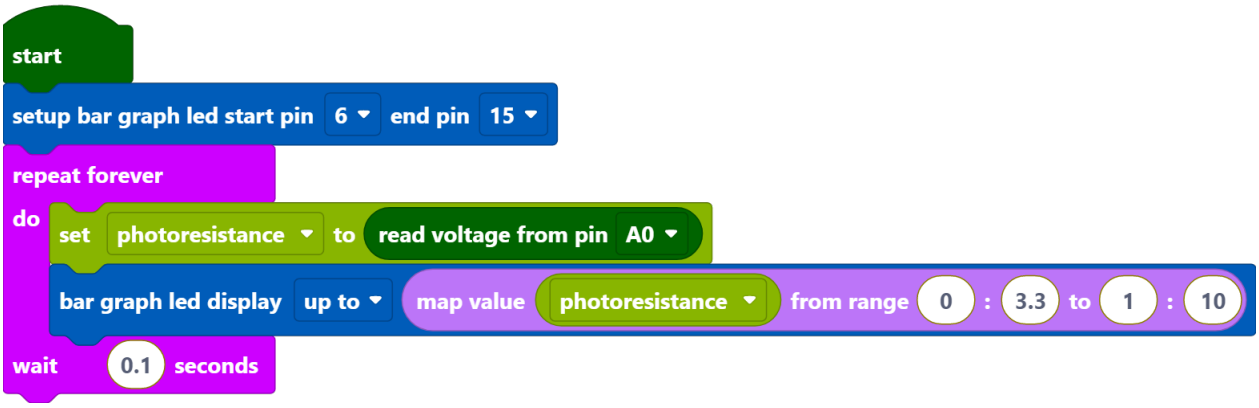


- LED バーグラフは 10 個の LED から構成され、ラベル側がアノード、反対側がカソードです。
- LED バーグラフのアノードは GP6~GP15 に接続され、カソードは 220 オームの抵抗器を介して GND に接続されます。
- フォトレジスタの一端を 3.3V に、もう一端を GP26（A0）に接続します。同時に、GP26 は別の 10K オームの抵抗器を介して GND に接続される必要があります。このようにすると、光が強くなるとフォトレジスタの抵抗が減少し、A0 の電圧が上昇します。

プログラム

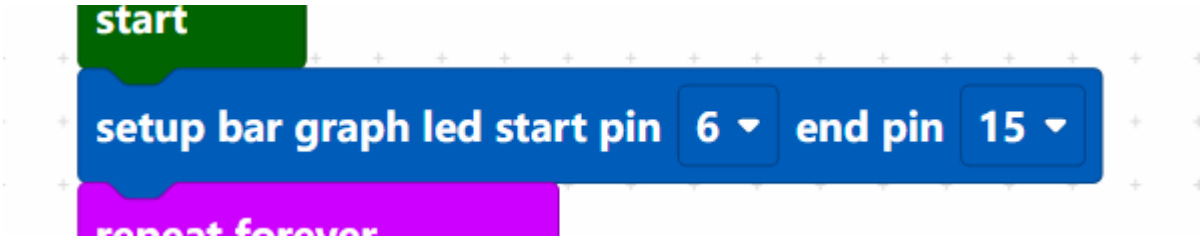
注釈:

- 下の画像を参考に、ドラッグ&ドロップでプログラムを作成できます。
- `kepler-kit-main\piper` のパスから `2.8_light_intensity_display.png` をインポートします。詳細なチュートリアルについては、 **コードをインポート** を参照してください。

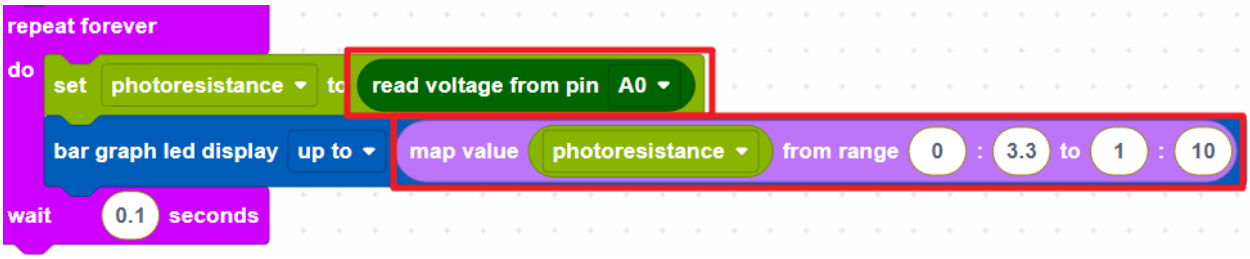


- Pico W を接続した後、 **スタート** ボタンを押してプログラムを実行します。
- 光が強いと、LED バーグラフの LED が多く点灯します。
- プログラムの実行がうまくいかない場合は、LED バーグラフを反転させてみてください。

仕組み



LED バーグラフのピンを GP6 から GP15 に接続します。



A0（GP26）の電圧値を変数 [photo_vol] に格納します。[map value () from () to ()] ブロックを用いて、変数 [photo_vol] を 0 から 3.3V の範囲で 0 から 10（LED バーグラフの LED 数）にマッピングします。

- [map value () from () to ()]: 一つの範囲から別の範囲へ値をマッピングします。

7.12 2.9 招き猫プロジェクト

このプロジェクトでは、PIR モジュールとサーボを使用して招き猫を作成します。PIR モジュールは来訪者を検出するために、サーボは招き猫の手を振る動作を模倣します。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

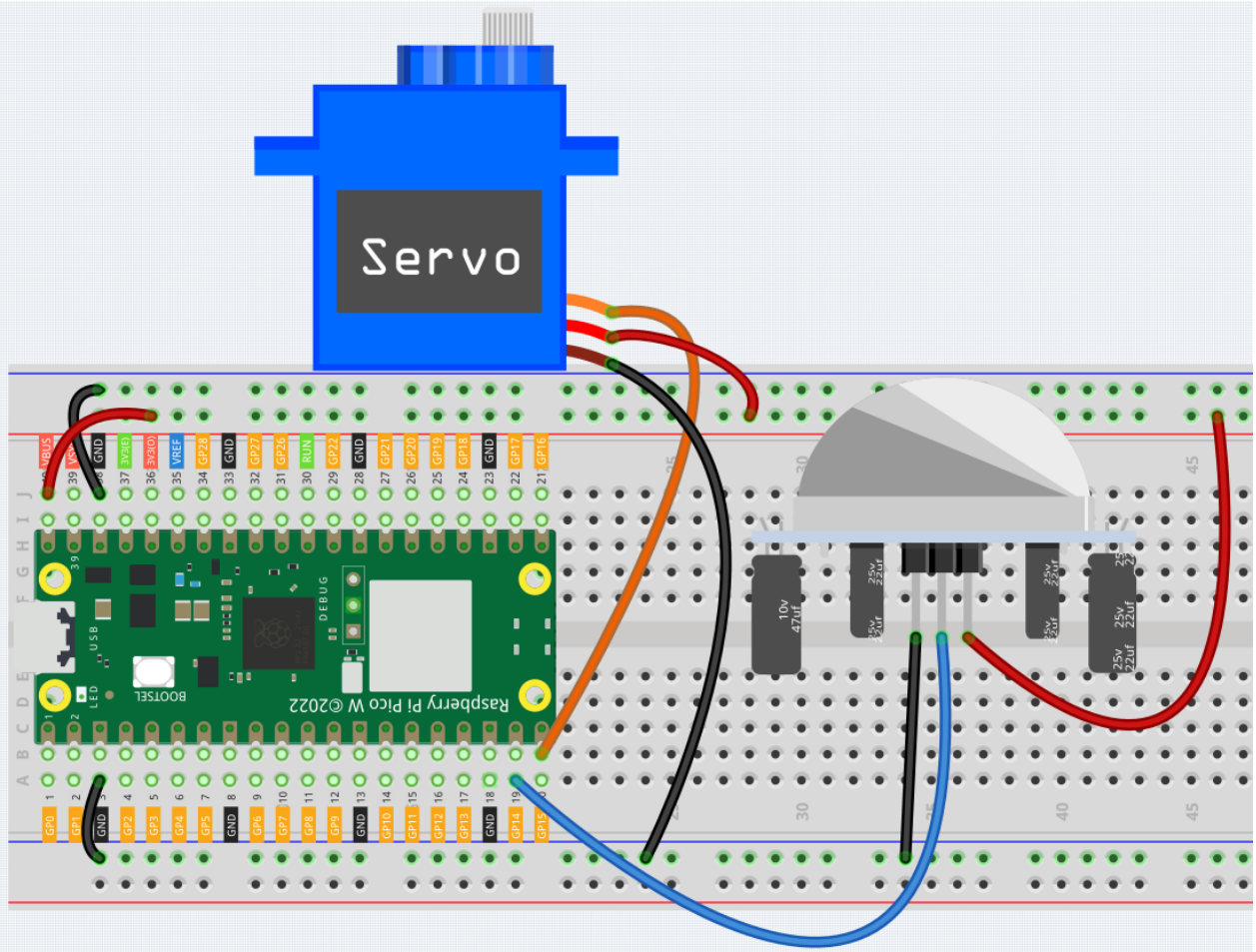
一式を購入すると便利です、リンクはこちら：

名称	キット内容	リンク
ケプラーキット	450 以上	

個別に購入することも可能です、以下のリンクを参照してください。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**サーボ**	1	
6	*PIR* **モーションセンサーモジュール**	1	

配線

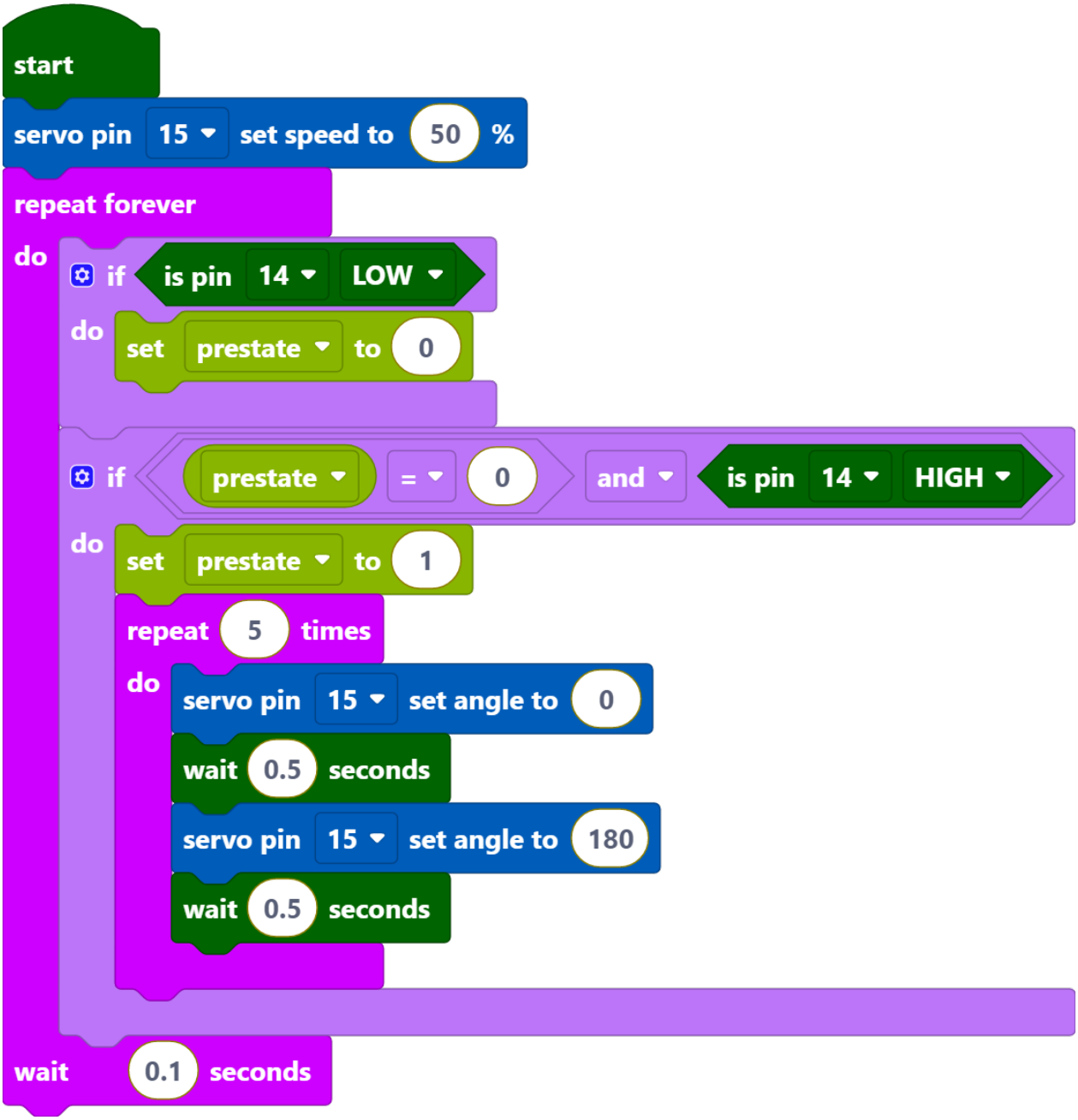


- サーボのオレンジ色の線（信号）は GP15 に接続し、赤い線（電源）は VBUS に、茶色の線（接地）は GND に接続します。
- PIR モジュールの中央のピンは GP3 に接続されます。

プログラム

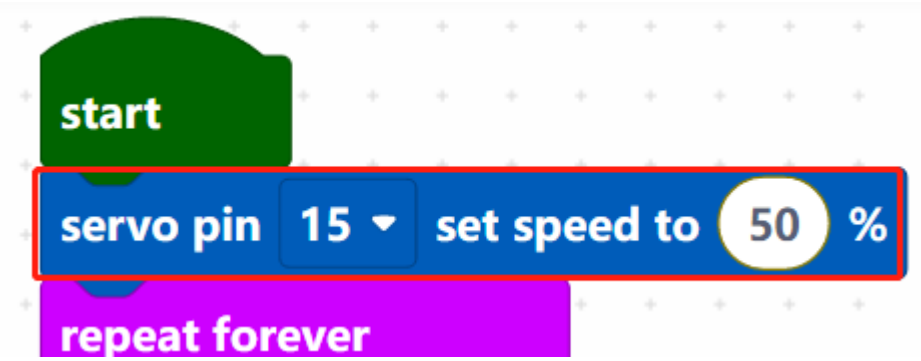
注釈:

- 下の画像を参考にして、ドラッグ&ドロップでプログラムを作成できます。
- 詳細なチュートリアルは、 **コードをインポート** をご参照ください。

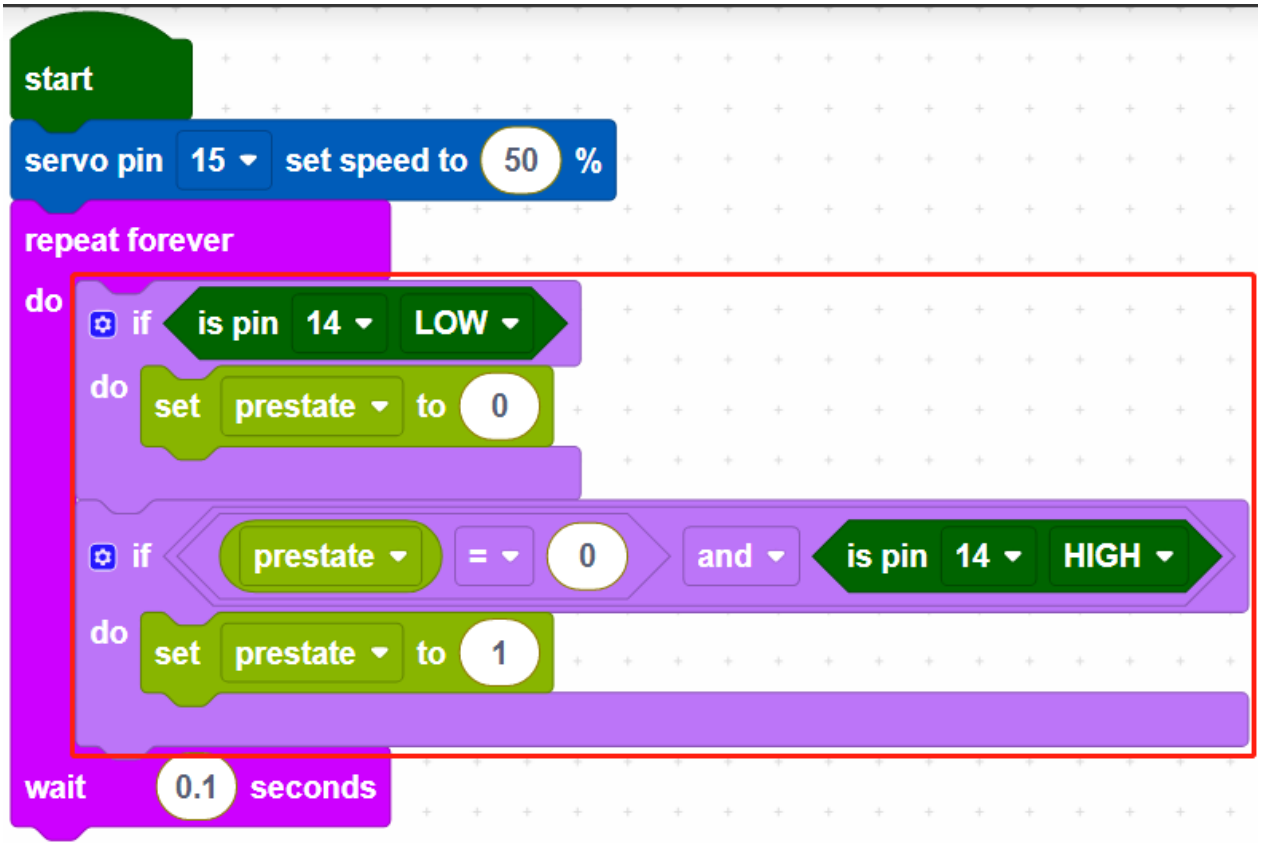


- Pico W を接続した後、 **スタート** ボタンをクリックしてプログラムを実行します。
- PIR モジュールが客の到来を検出すると、サーボは 5 回往復して停止します。

仕組み

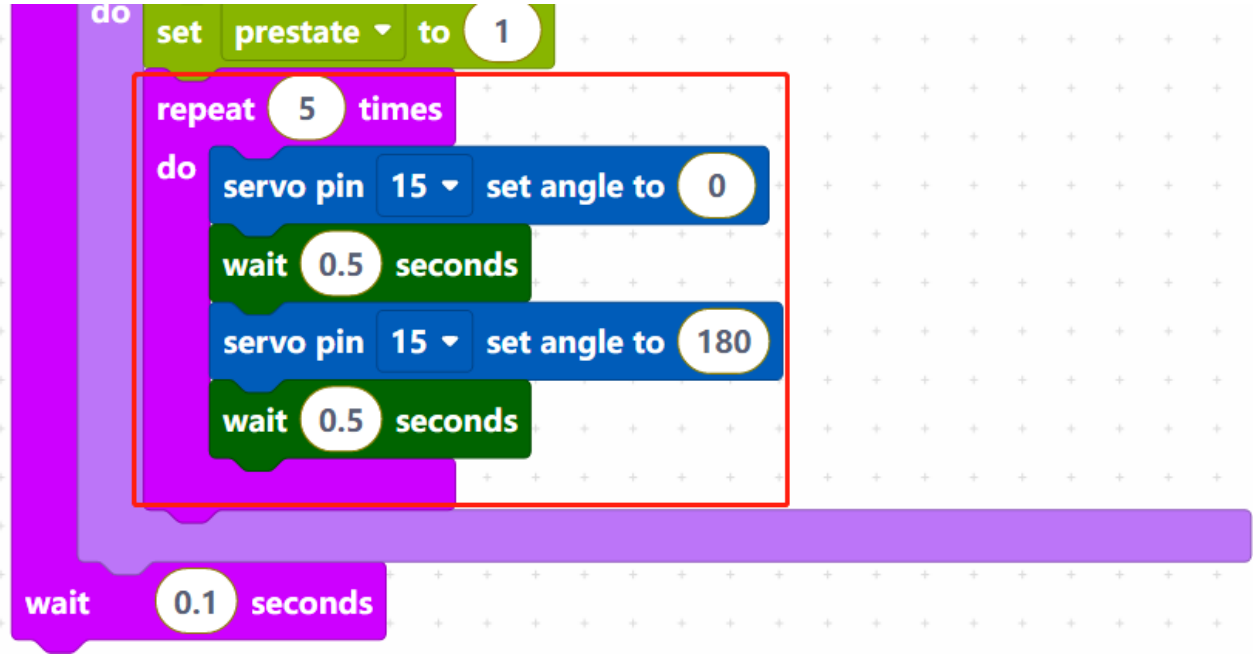


ピン 15（サーボ）の回転速度を 15% に設定します。



もし GP14 が低い場合、変数 [prestate] を 0 に設定します。変数 [prestate] が 0 で GP14 が高い（人が検出された）場合、変数 [prestate] を 1 に設定します。

これは、GP14 が低から高に変わった場合にのみメインのコードが動作するようにし、PIR モジュールが連続して人を検出しても一度しか反応しないようにするためです。



サーボを 0 度から 180 度まで 5 回循環させます。

- [repeat () times do]: do ブロック内のコードを指定した回数繰り返します。

7.13 2.10 流れる LED

このキットには WS2812 RGB LED ストリップが付属しており、各 LED は独立して制御できる多彩な色を表示できます。

このプロジェクトでは、傾斜スイッチを使用して、WS2812 RGB LED ストリップ上の LED の流れる方向を制御します。

必要な部品

このプロジェクトで必要な部品は以下の通りです。

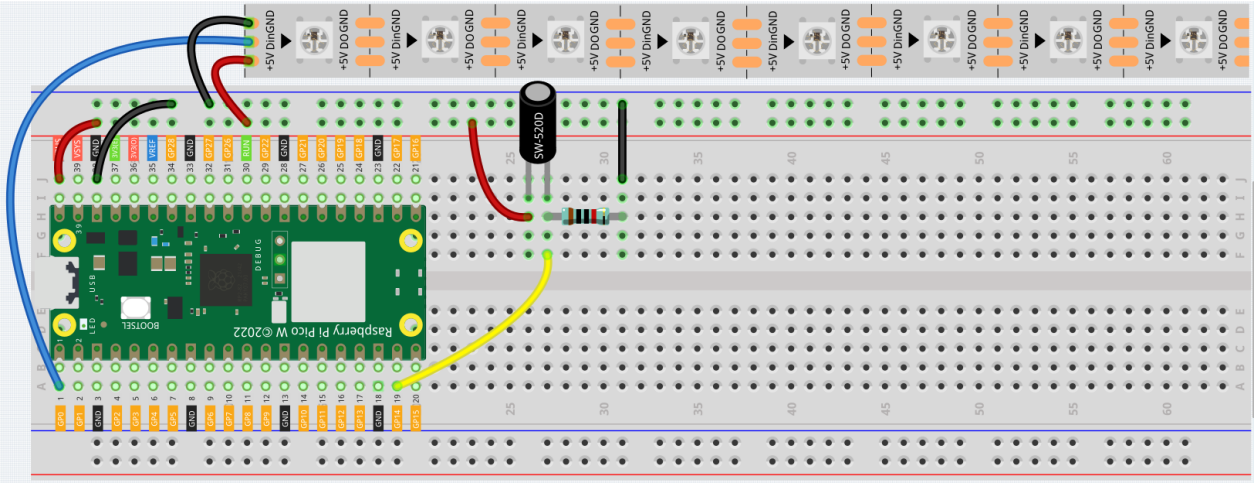
一式を購入すると便利です。リンクはこちら：

名称	キット内容	リンク
ケプラーキット	450 以上	

個別に購入することも可能です。以下のリンクを参照してください。

SN	部品	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	マイクロ USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**抵抗器**	1(10K Ω)	
6	**傾斜スイッチ**	1	
7	*WS2812 RGB 8 LED* **ストリップ**	1	

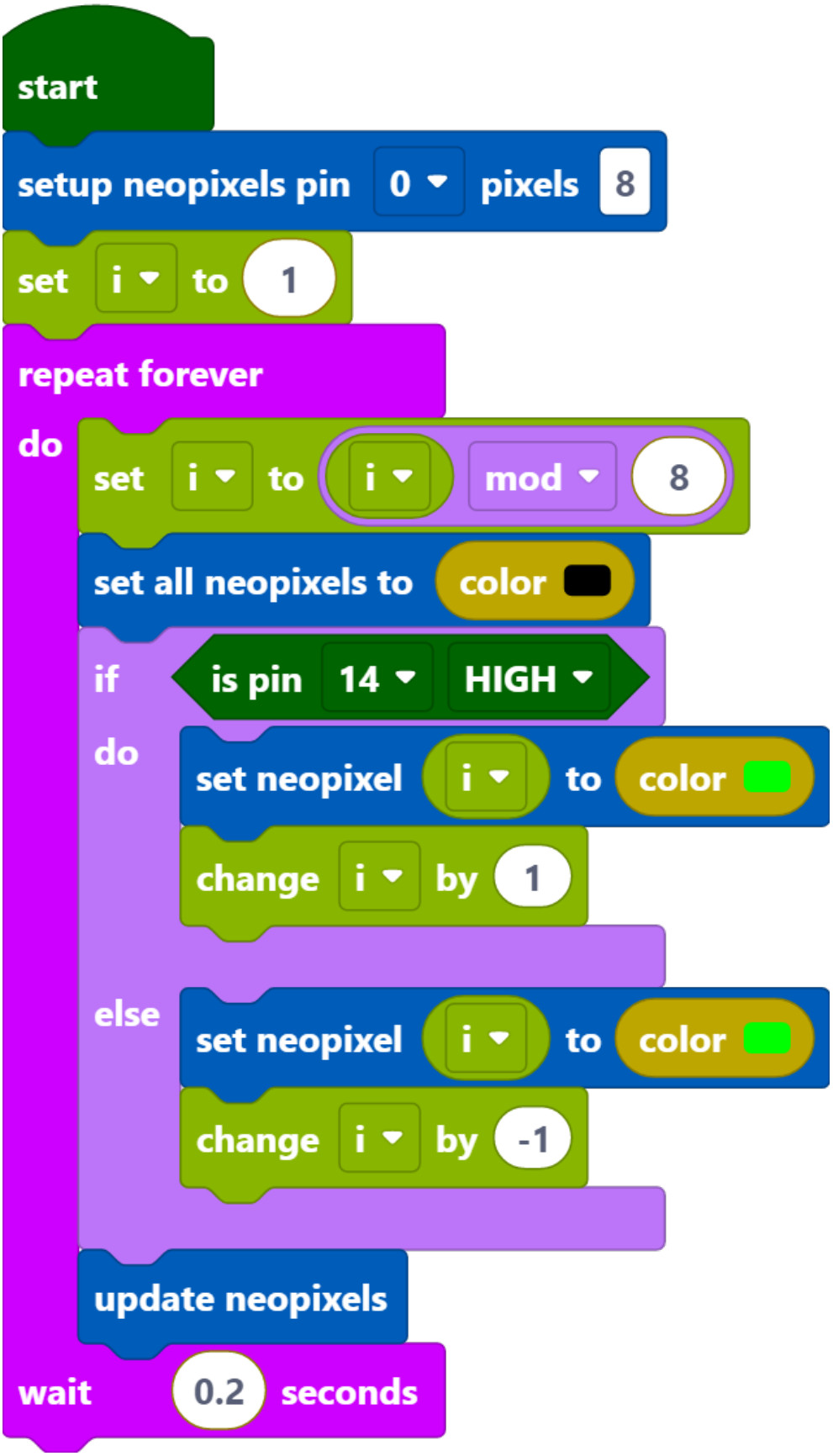
配線



コード

注釈:

- 下の画像を参考にして、ドラッグ&ドロップでコードを作成できます。
- `kepler-kit-main\piper` のパスから `2.10_flowing_led.png` をインポートしてください。詳細なチュートリアルは、 **コードをインポート** をご参照ください。

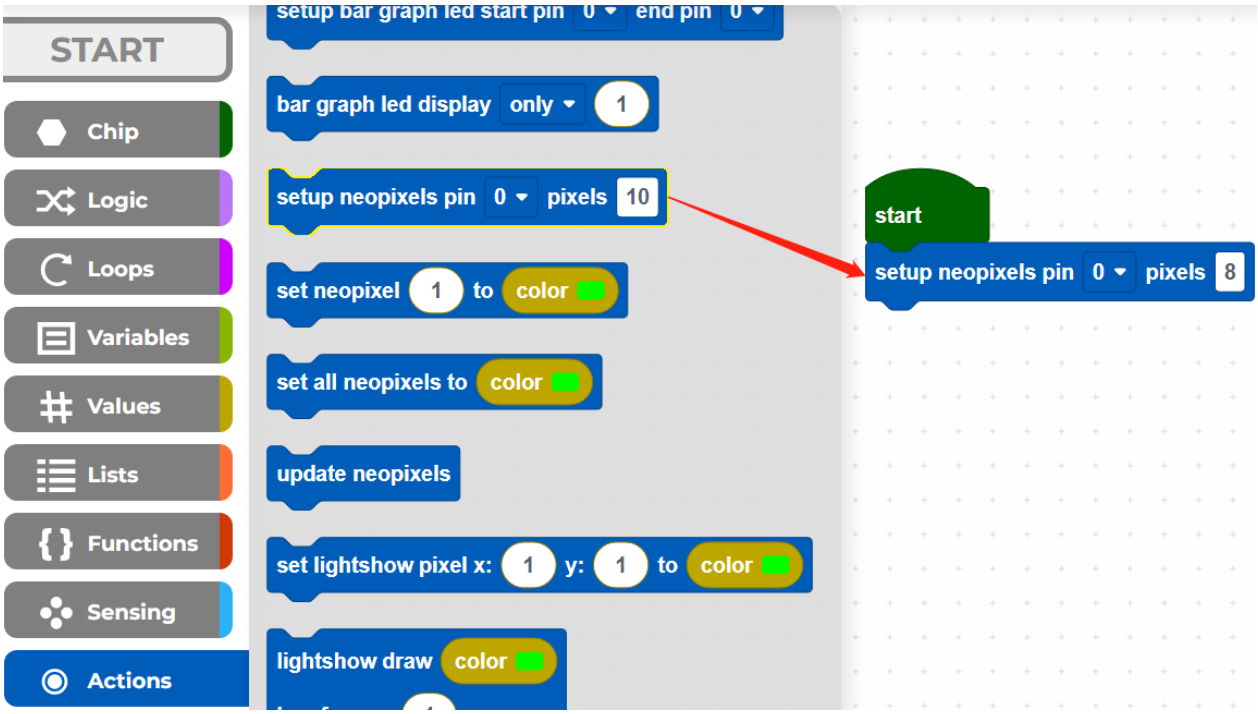

start
setup neopixels pin 0 pixels 8
set i to 1
repeat forever
do
set i to i mod 8
set all neopixels to color
if is pin 14 HIGH
do
set neopixel i to color
change i by 1
else
set neopixel i to color
change i by -1
update neopixels
wait 0.2 seconds

Pico W を接続した後、 **スタート** ボタンをクリックしてコードを実行します。

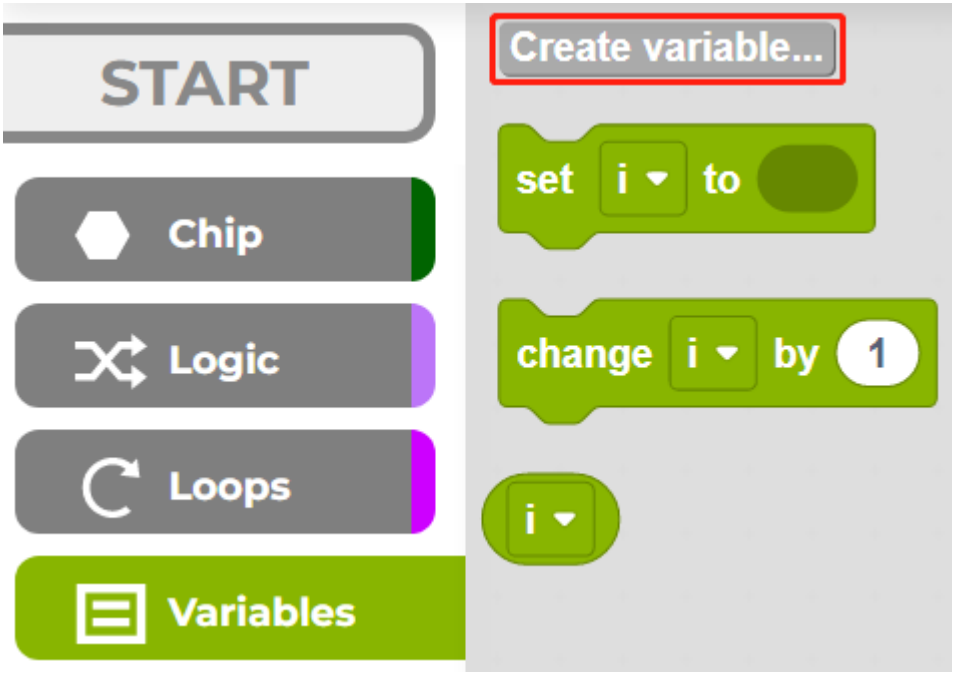
傾斜スイッチが垂直に置かれていると、WS2812 RGB LED ストリップ上の LED が順番に緑色で点灯します。水平に置かれている場合、緑色の LED が逆方向に一つずつ点灯します。

プログラミング

ステップ 1: **Actions** パレットの [setup neopixel pin() pixels()] ブロックを使用して、WS2812 RGB LED ストリップを初期化します。 **0** は接続されたピンが GP0 で、 **8** は WS2812 RGB LED ストリップに 8 つの RGB LED があることを意味します。

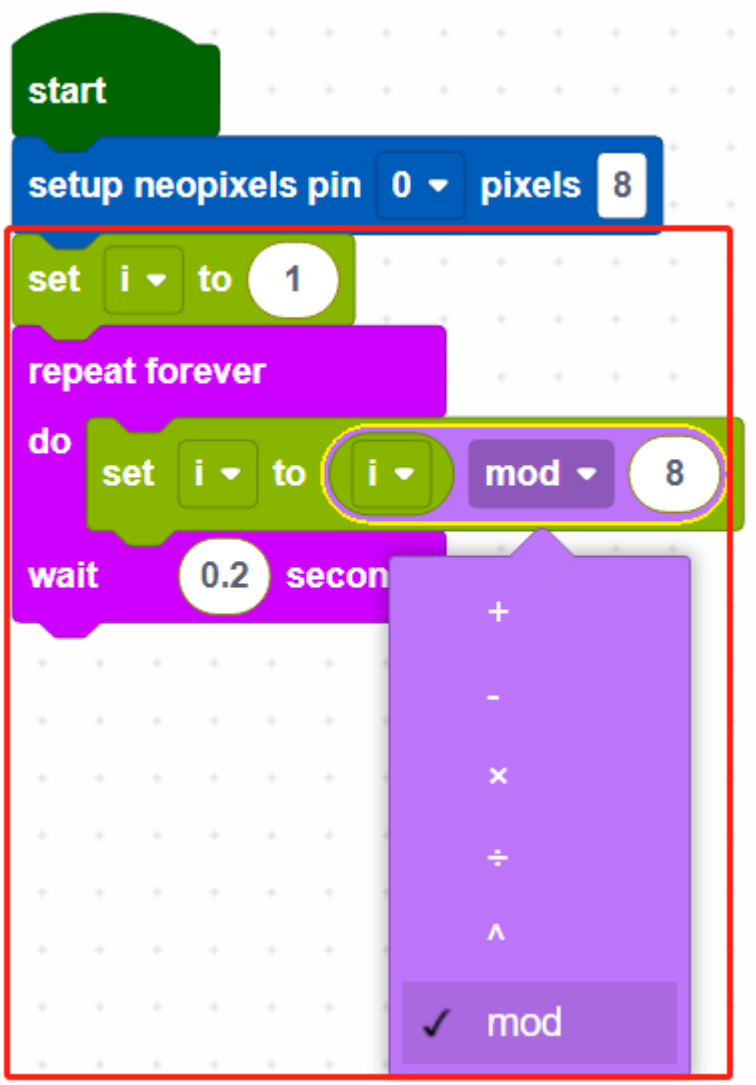


ステップ 2: **Variables** パレットで **Create variable** ボタンをクリックし、WS2812 RGB LED ストリップ上の LED を表す変数 **i** を作成します。

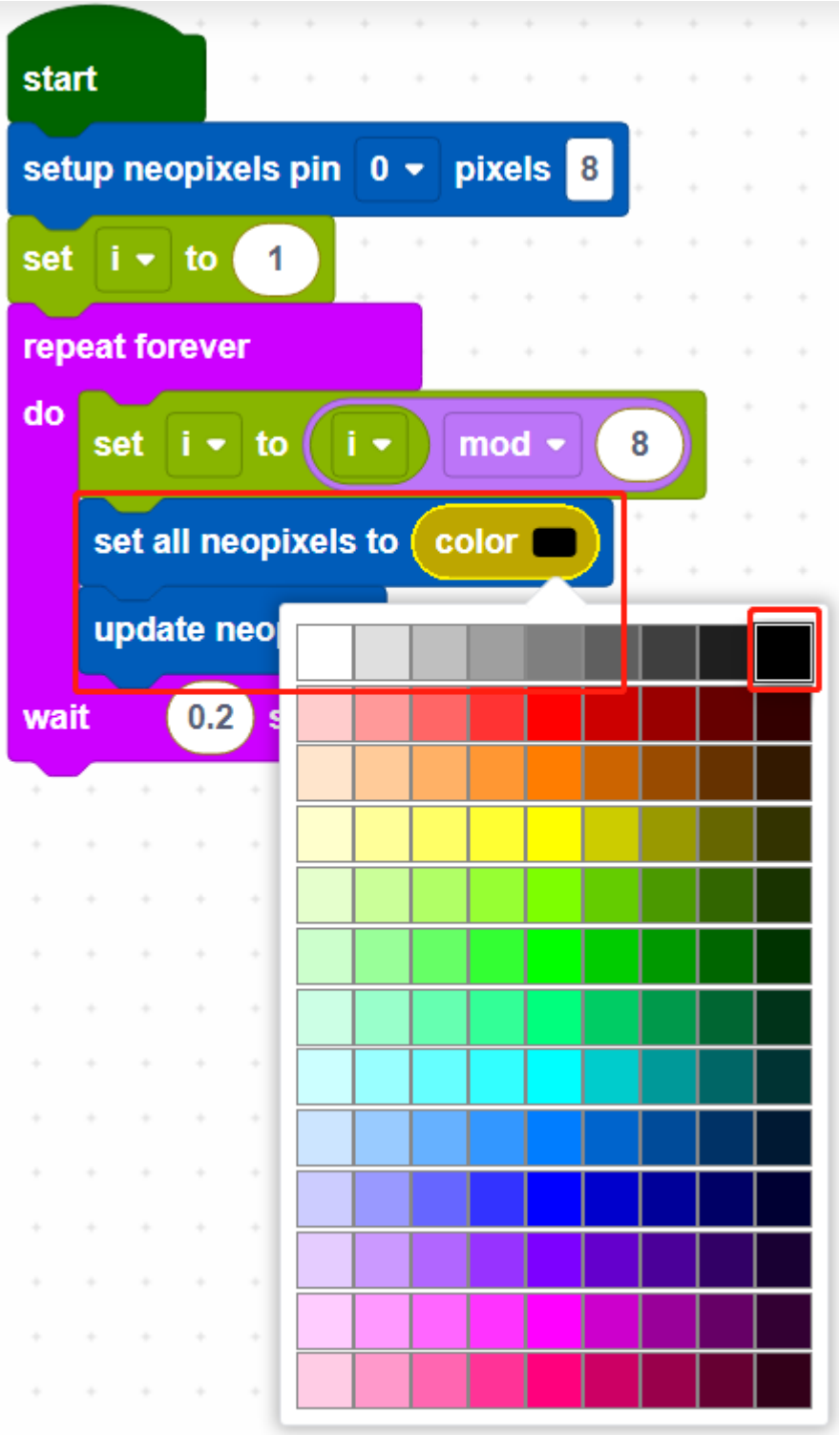


ステップ 3: 変数 **i** の初期値を 1（ワイヤーに近い LED）に設定し、[repeat forever] ブロック内で [() mod ()] を使用して i の値を 0 から 7 に設定します。例：1 mod 8 = 1、8 mod 8 = 0、9 mod 8 = 1 など。

- [() mod ()]: これは剰余演算子のブロックであり、 **Loops** パレットから [() = ()] をドロップダウンして **mod** を選択します。

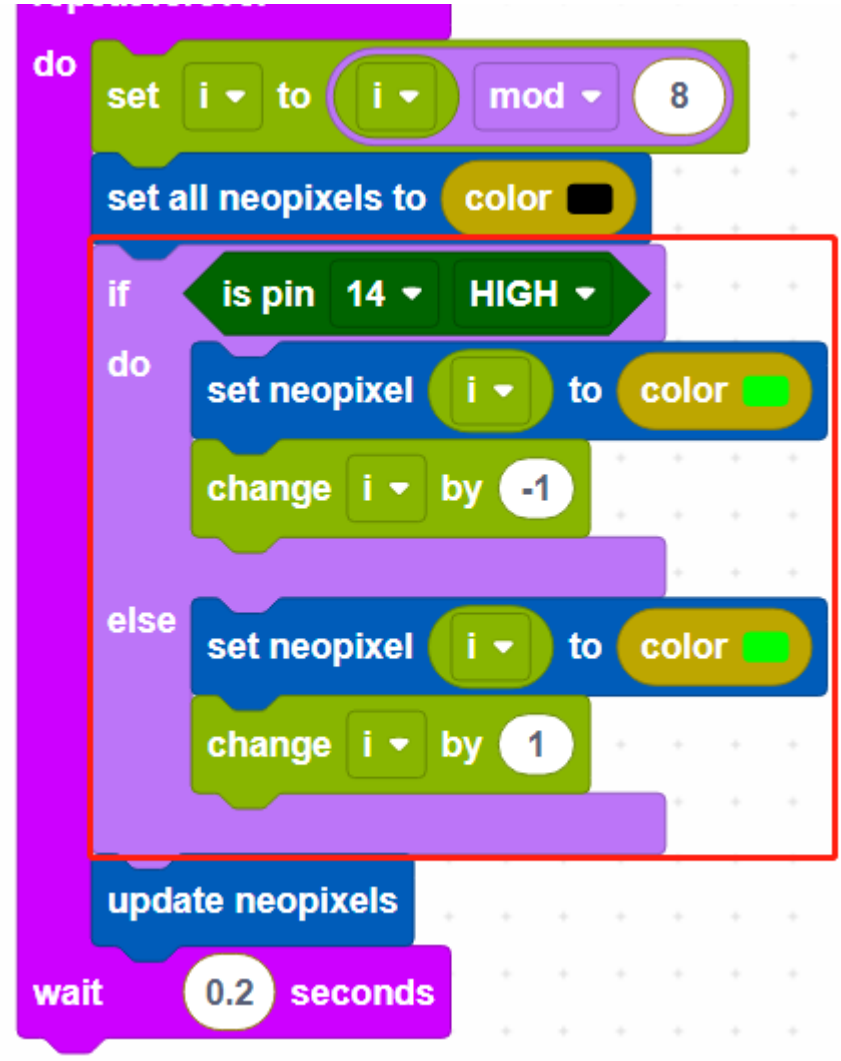


ステップ 4: 全てのネオピクセルを黒に設定して、全ての LED を消灯させます。その後、[updates neopixels] を使用してこの効果を WS2812 RGB LED ストリップに適用します。



- [set all neopixels to ()]: 全ての LED に色を設定するために使用します。13*9 色あり、右上の色は LED を消灯するための黒です。
- [updates neopixels]: この効果を WS2812 RGB LED ストリップに更新します。

ステップ 5: pin14 が高く読み取られた場合、WS2812 RGB LED ストリップ上の LED を緑色で一つずつ点灯させます。それ以外の場合は、逆方向で緑色で一つずつ点灯させます。



- [change () by ()]: 変数の値を特定のステップで増加（正）または減少（負）させるために使用されます。

7.14 2.11 逆転警報システム

このプロジェクトでは、超音波モジュールとアクティブブザーを使用して逆転警報システムを作成します。超音波モジュールは距離を検出し、ブザーは距離に応じて異なる周波数の警報音を発します。

必要なコンポーネント

このプロジェクトで必要なコンポーネントは以下の通りです。

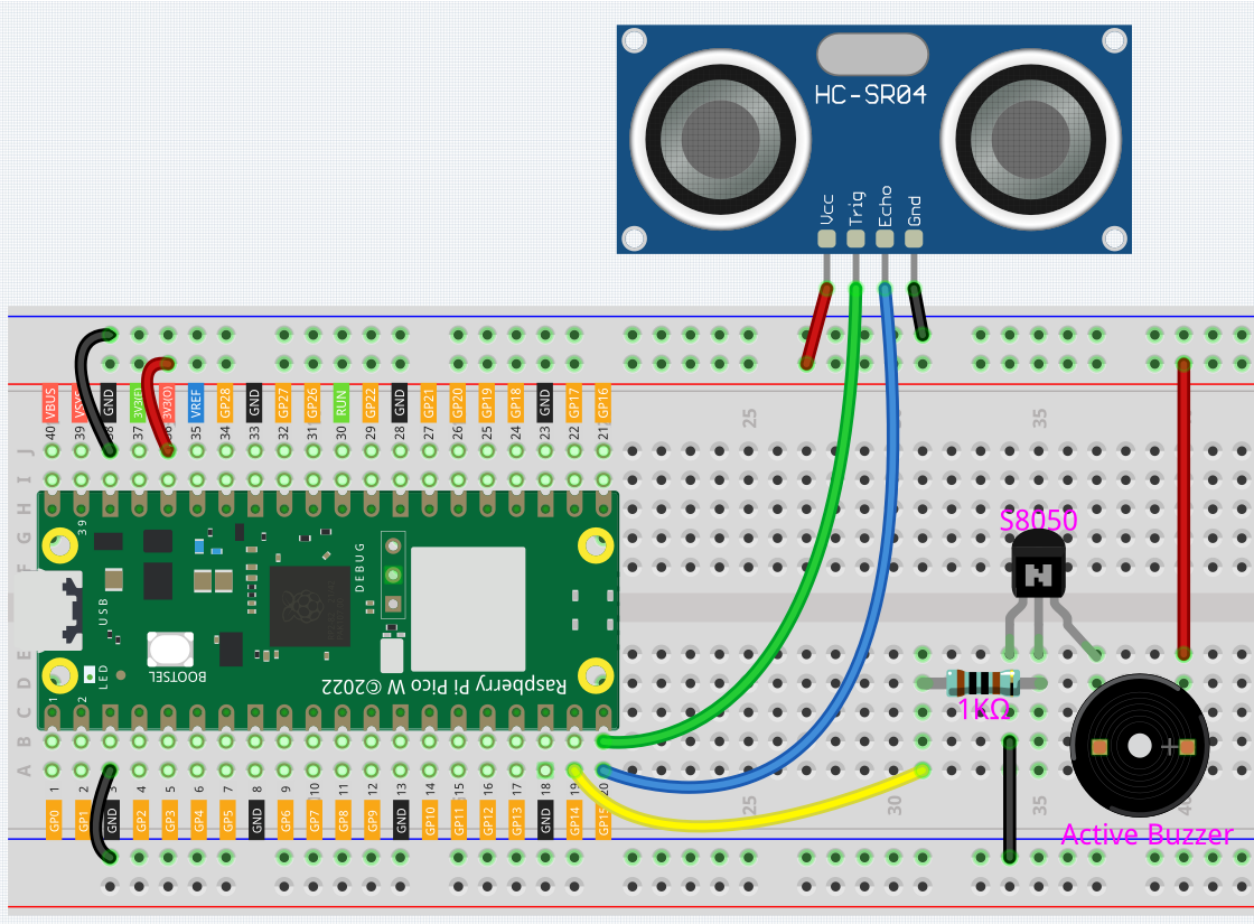
キット全体を購入するのが便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

以下のリンクから個別にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	1(1K Ω)	
7	アクティブ **ブザー**	1	
8	**超音波モジュール**	1	

配線

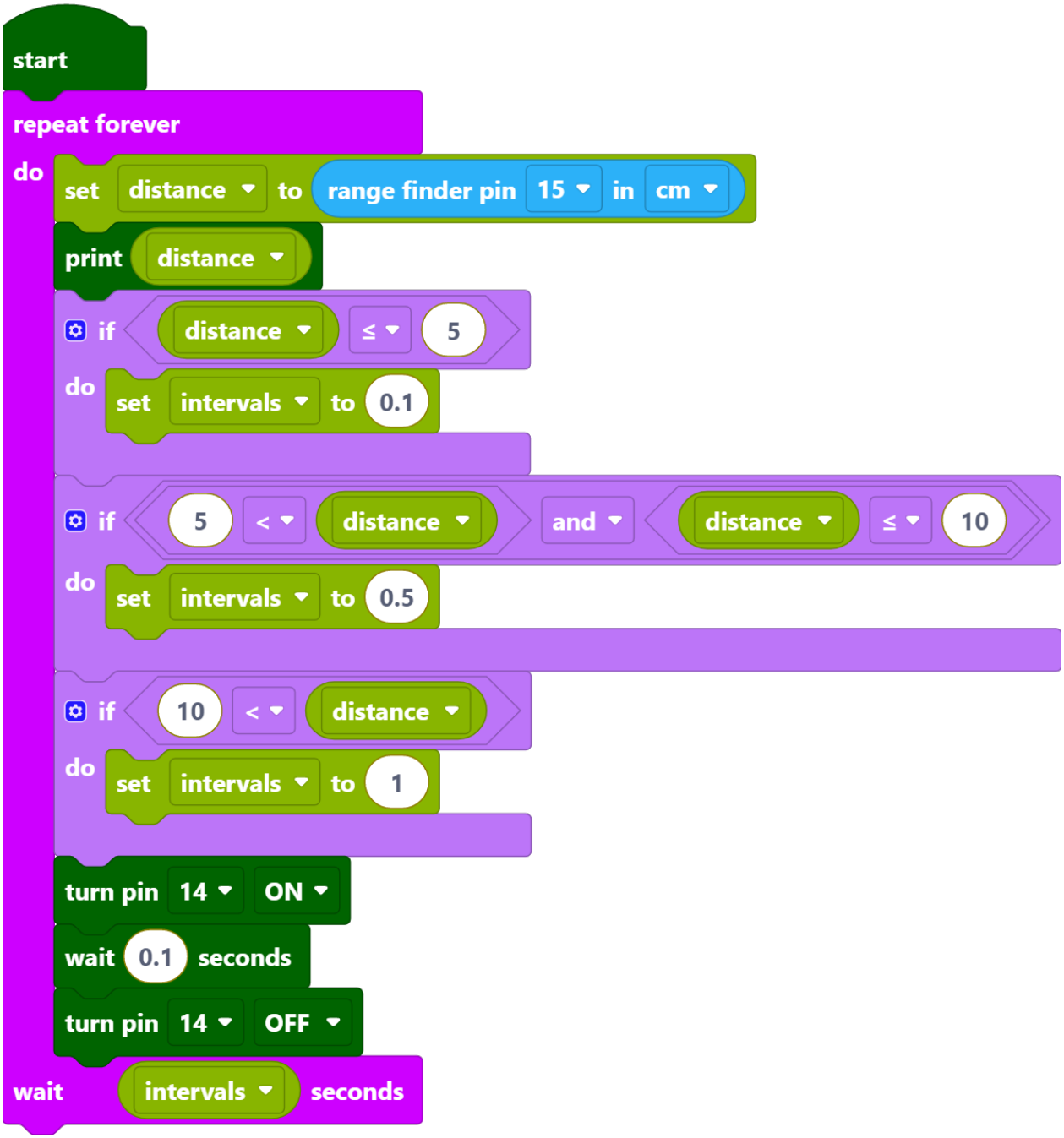


- 超音波モジュールの Echo と Trig ピンは、同時に GP15 に接続されています。これにより、超音波モジュールは GP15 から信号を送受信します。
- トランジスタの中央ピンは、1k Ωの抵抗器を介して GP14 に接続されています。

コード

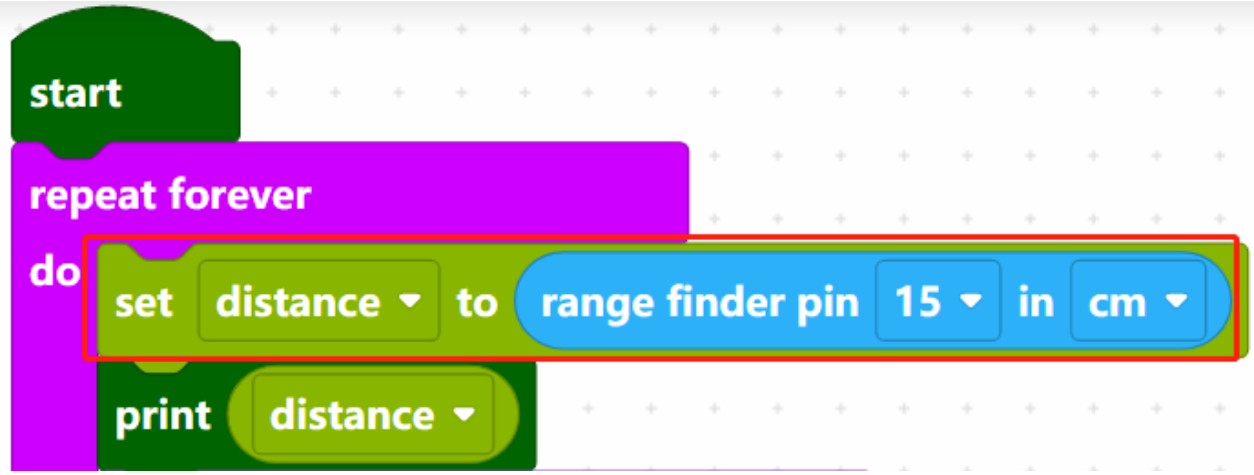
注釈:

- 下の画像を参照して、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.11_reversing_system.png` をインポートしてください。詳細なチュートリアルは、 **コードをインポート** を参照してください。

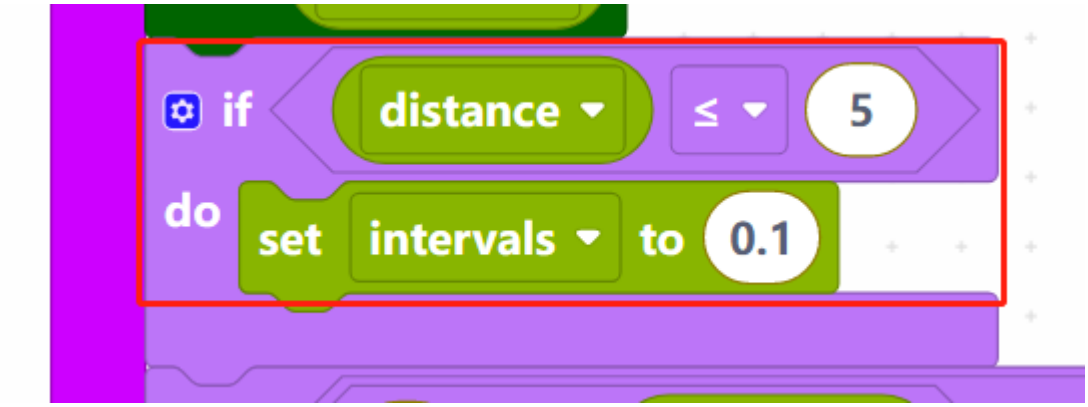


- Pico W を接続した後、 **スタート** ボタンをクリックしてコードを実行します。
- 超音波検出距離が 5cm 未満の場合、ブザーは鋭い音（0.1 秒）を発します。
- 検出距離が 5〜10cm の場合、ブザーはやや遅い音（0.5 秒）を発します。
- 検出距離が 10cm 以上の場合、1 秒ごとに音声プロンプトがあります。

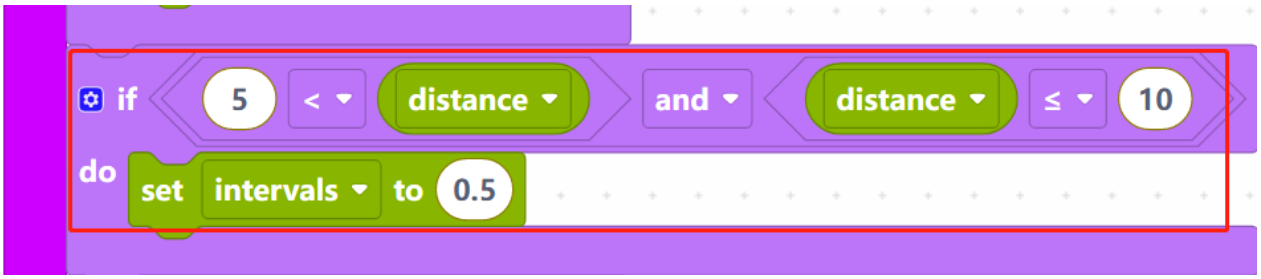
仕組み



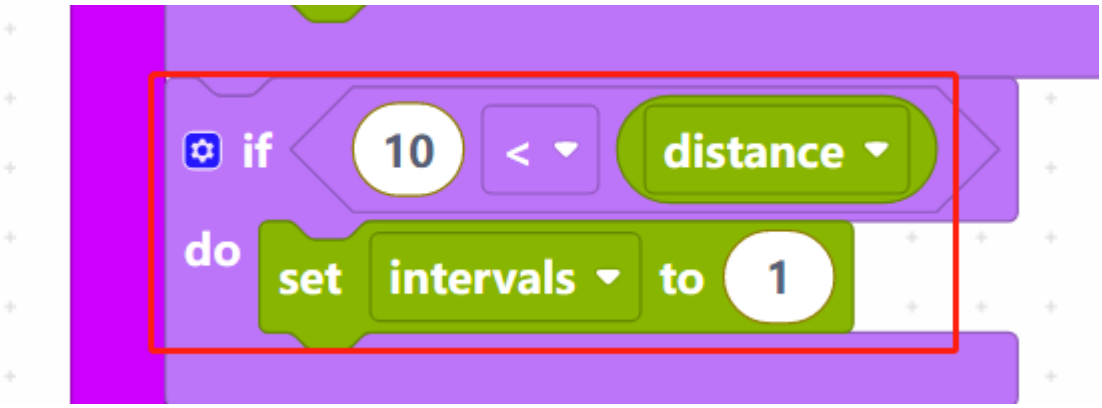
超音波検出の距離（cm）を読み取り、変数 [distance] に格納します。



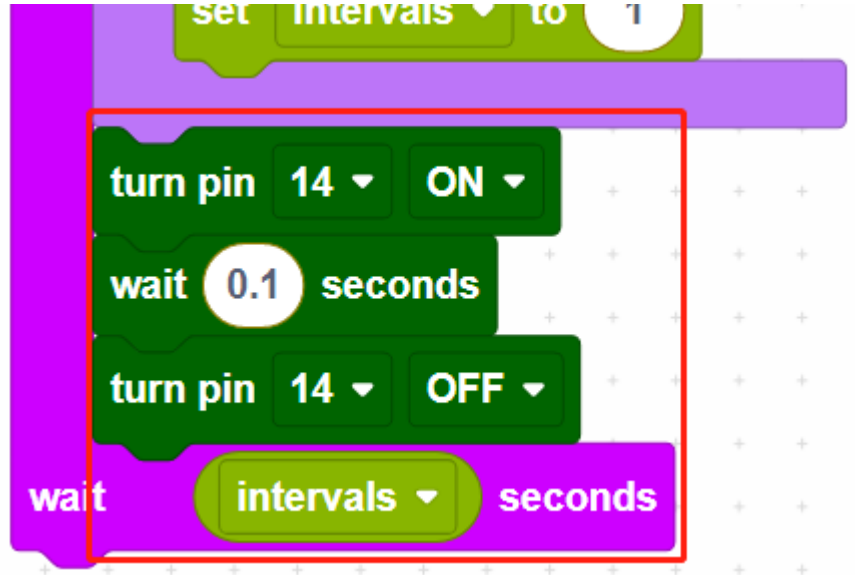
距離が 5 以下の場合、変数 [intervals] を 0.1 秒に設定します。変数 [intervals] はブザー音の間隔です。



距離が 5 より大きく、10 以下の場合、[intervals] を 0.5 秒に設定します。



距離が 10 より大きい場合、[intervals] の時間を 1 秒に設定します。



最後に、[intervals] 秒ごとにブザーが鳴るようにします。

7.15 2.12 スマートファン

このプロジェクトでは、サーミスター、TA6586、モーター、および電源モジュールを使用して、温度制御型のスマートファンを作成します。設定温度に達すると、ファンは自動的に回転します。

このプロジェクトで必要なコンポーネントは以下の通りです。

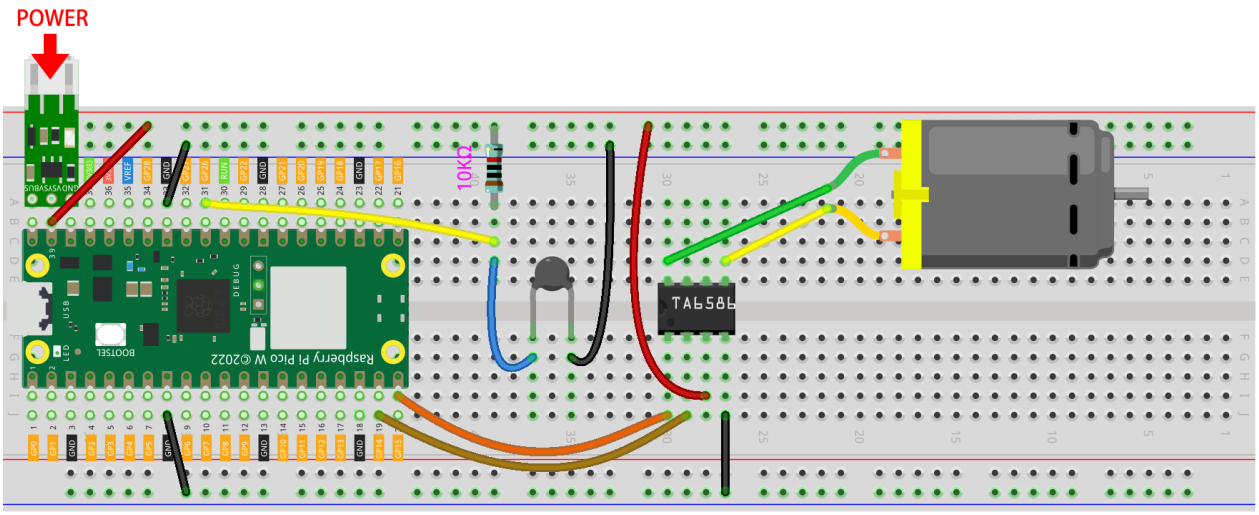
一式を購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

以下のリンクから個々にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	*TA6586* - **モータードライバーチップ**	1	
6	*DC* **モーター**	1	
7	*Li-po* **充電モジュール**	1	
8	18650 バッテリー	1	
9	バッテリーホルダー	1	
10	**抵抗器**	1(10K Ω)	
11	**フォトレジスタ**	1	

配線

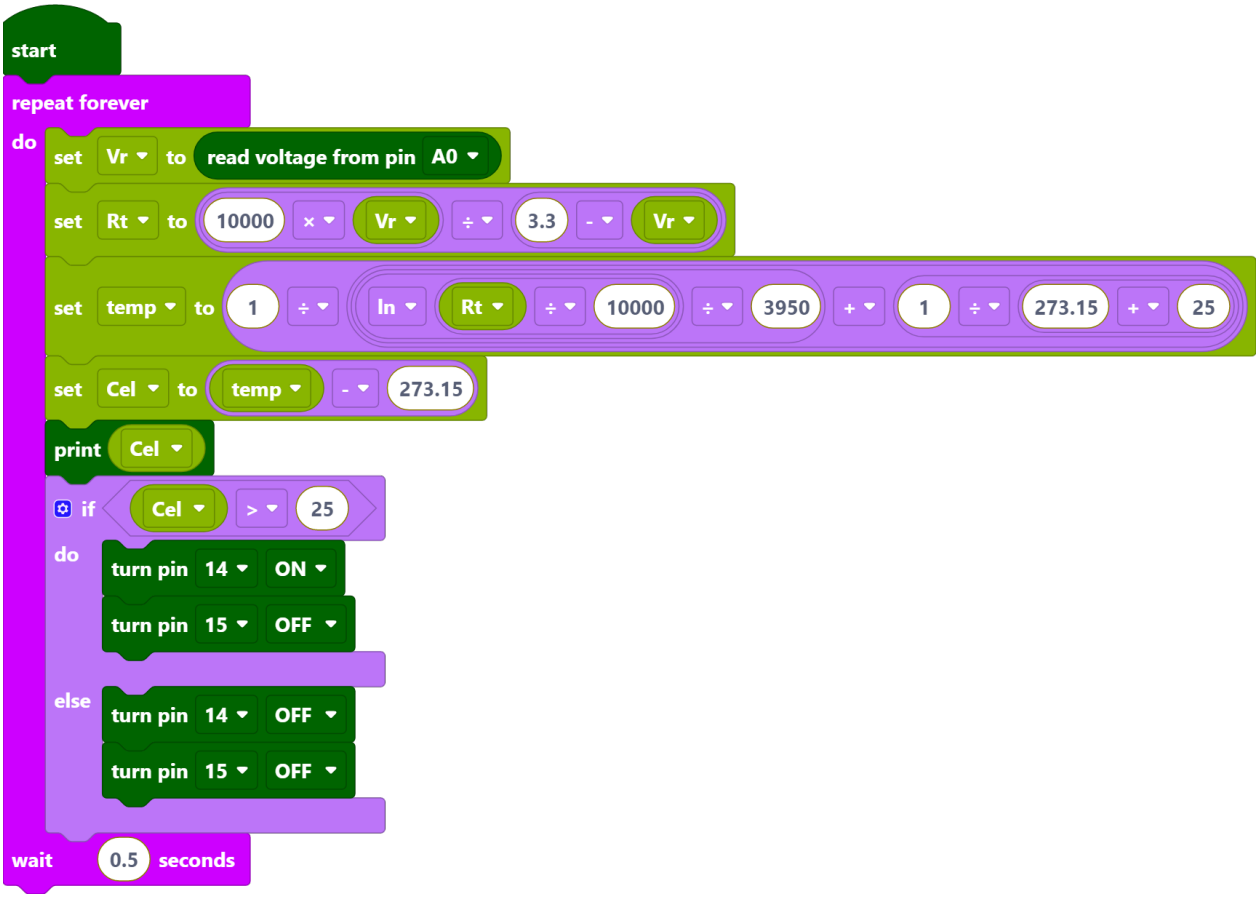


コード

注釈:

- 以下の画像を参照して、ドラッグアンドドロップでコードを作成できます。

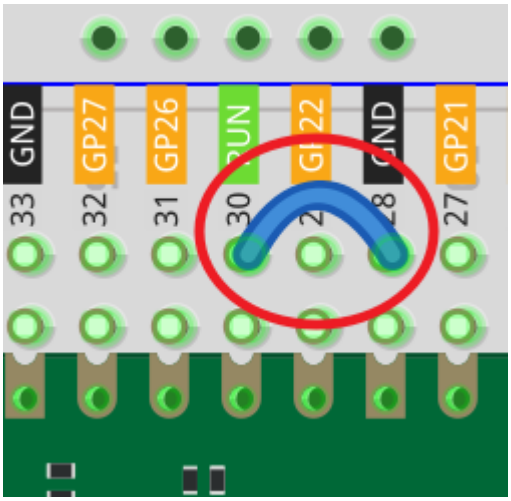
- `kepler-kit-main\piper` のパスから `2.12_smart_fan.png.png` をインポートしてください。詳細なチュートリアルは、 **コードをインポート** を参照してください。



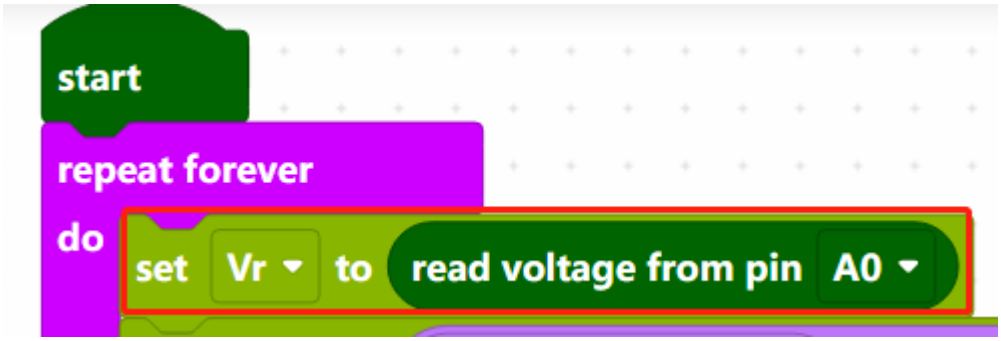
- Pico W に接続した後、 **スタート** ボタンをクリックしてコードが実行されます。
- CONSLE をクリックすると、現在の温度が摂氏で表示されます。
- 温度が 25 度以上の場合、ファンは回転を開始し、25 度未満の場合は停止します。

注釈:

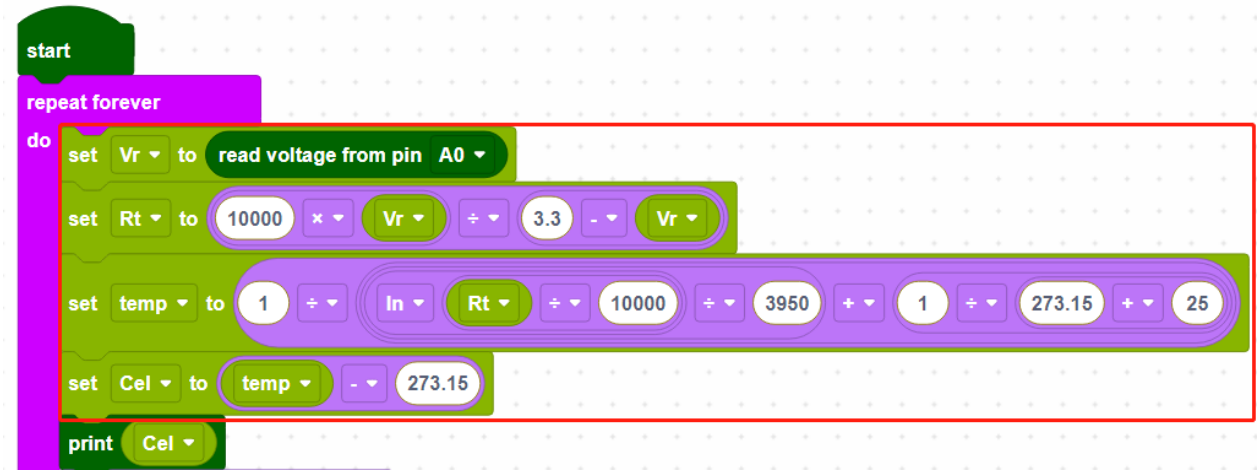
- モーターが停止ボタンをクリックした後も回転し続ける場合、この時点で Pico W の Run ピンを GND にワイヤーでリセットしてから、再度このワイヤーを抜いてコードを実行する必要があります。
- これは、モーターが多くの電流を使用しているため、Pico W がコンピュータから切断される可能性があるからです。



仕組み



A0（GP26）の電圧が読み取られ、変数 [Vr] に割り当てられます。



これらの計算は、サーミスターの値を摂氏度に変換します。

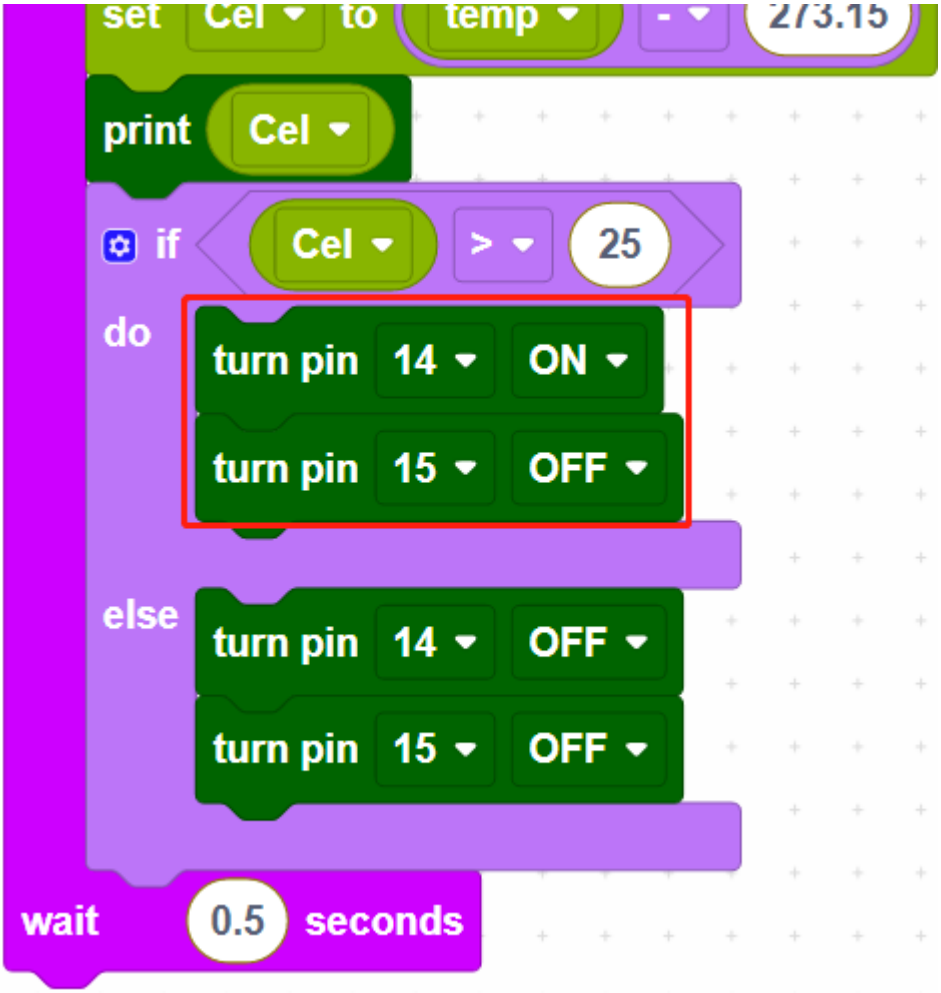
注釈: ここでは、抵抗と温度の関係があります：

RT =RN expB(1/TK – 1/TN)

- RT は、温度が TK の場合の NTC サーミスターの抵抗です。
- RN は、定格温度 TN での NTC サーミスターの抵抗です。ここでは、RN の数値は 10k です。
- TK はケルビン温度で、単位は K です。ここでは、TK の数値は 273.15 + 摂氏度です。
- TN は、定格ケルビン温度で、単位も K です。ここでは、TN の数値は 273.15+25 です。
- B（ベータ）は NTC サーミスターの材料定数であり、また熱感度指数とも呼ばれ、数値は 3950 です。
- exp は指数の略で、底数 e は自然数であり、約 2.7 です。

この式 **TK=1/(ln(RT/RN)/B+1/TN)** を変換して、ケルビン温度から 273.15 を引くと摂氏度になります。

この関係式は経験式です。温度と抵抗が有効範囲内である場合のみ正確です。



温度が 25 度摂氏を超える場合は、GP14 を ON にして GP15 を OFF に設定してモーターを回転させます。逆に、温度が 25 度摂氏以下の場合は、GP14 と GP15 の両方を低く設定してモーターを停止させます。

7.16 2.13 リアクションゲーム

このプロジェクトでは、複数のボタン、ブザー、および LED を使用してリアクションゲームを作成します。審判用のボタンを押すとゲームがスタートし、ブザーもゲームが続いていることを示すために鳴り続けます。2 人のプレイヤーボタンをそれぞれ素早く押します。再び審判ボタンが押されると、ゲームは終了し、ブザーが停止します。この時点で、Piper Make の CONSOLE を見て、誰の手の速さが速いかを確認します。

必要なコンポーネント

このプロジェクトで必要な部品は以下の通りです。

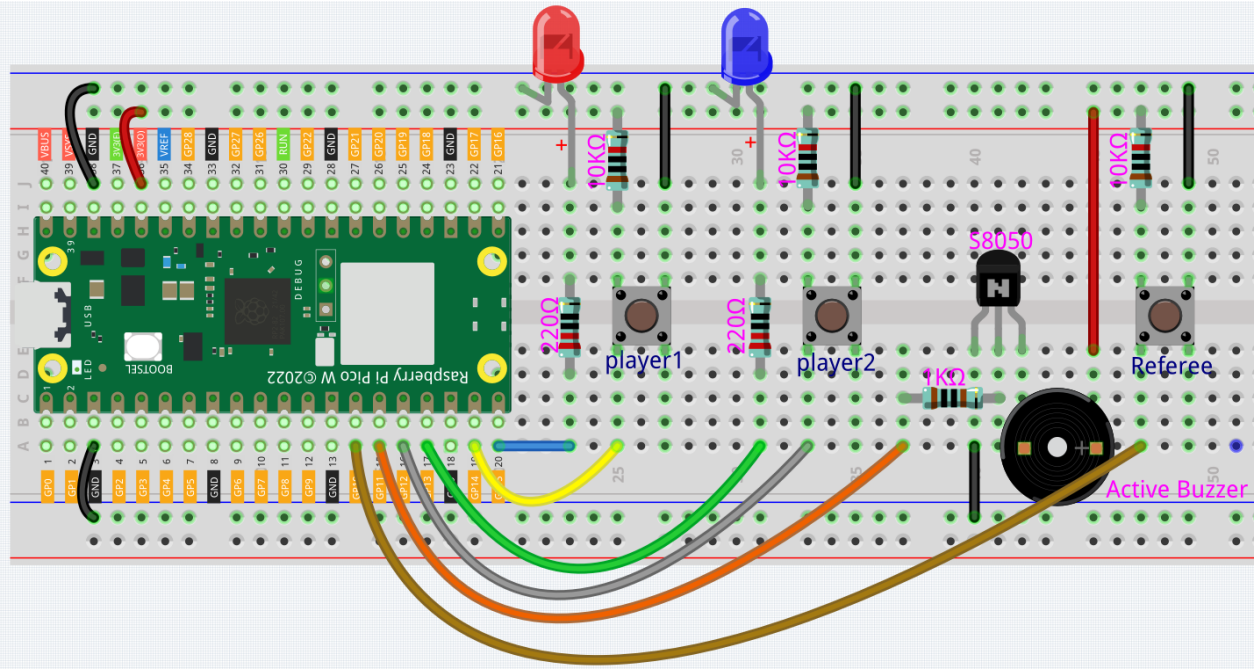
一式を購入する方が便利です、リンクはこちら：

名前	このキットに含まれるアイテム	リンク
Kepler Kit	450+	

以下のリンクから個々にも購入できます。

SN	コンポーネント	数量	リンク
1	*Raspberry Pi Pico W*	1	
2	Micro USB ケーブル	1	
3	**ブレッドボード**	1	
4	**ジャンパーワイヤー**	数本	
5	**トランジスタ**	1(S8050)	
6	**抵抗器**	6(2-220 Ω, 1-1K Ω, 3-10K Ω)	
7	Active **ブザー**	1	
8	**ボタン**	3	
9	*LED*	2	

配線

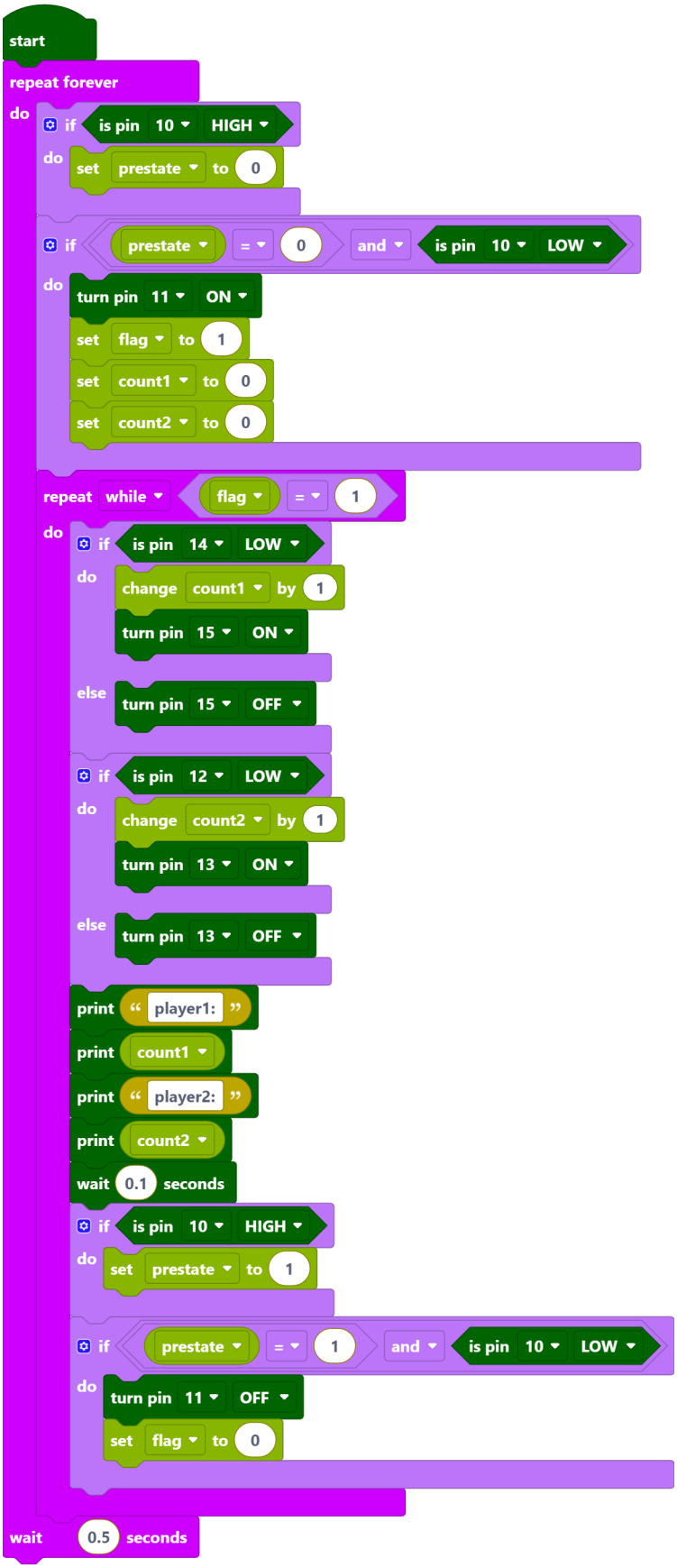


- プレイヤー 1（GP14）とプレイヤー 2（GP12）として 2 つのボタンを定義し、それぞれプルアップ抵抗が接続されています。ボタンが押されると、GP14 と GP12 はそれぞれ低くなります。
- それらのインジケーターは GP15 と GP13 に接続されており、高い出力で点灯します。
- GP10 に接続された審判用のボタンを定義します。ボタンが押されると、GP10 は低くなります。
- アクティブブザーは GP11 に接続されています。GP11 が高い出力の場合、ブザーが鳴ります。

コード

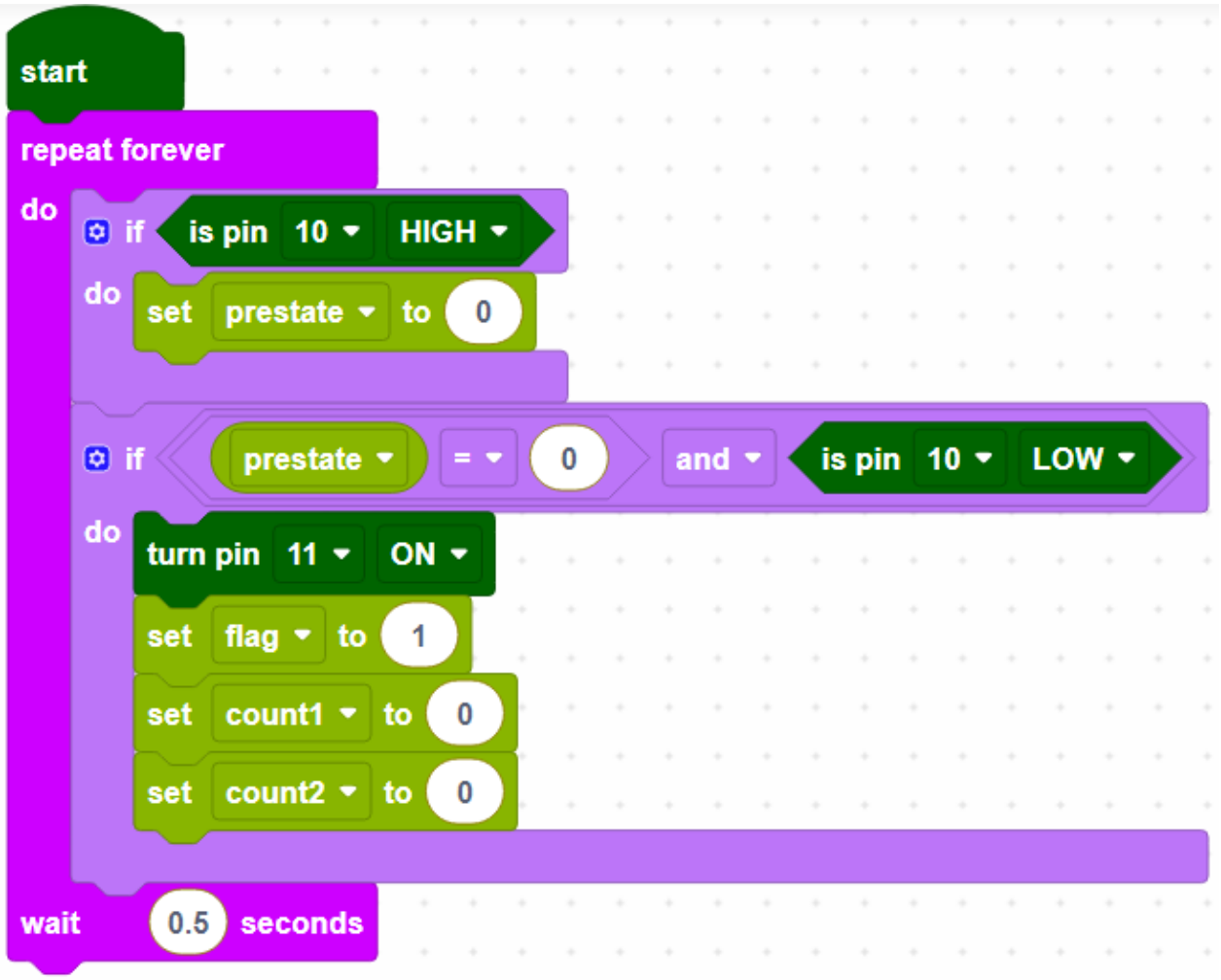
注釈:

- 下の画像を参照して、ドラッグ&ドロップでコードを書くことができます。
- `kepler-kit-main\piper` のパスから `2.13_reaction_game.png` をインポートしてください。詳細なチュートリアルは、 **コードをインポート** を参照してください。

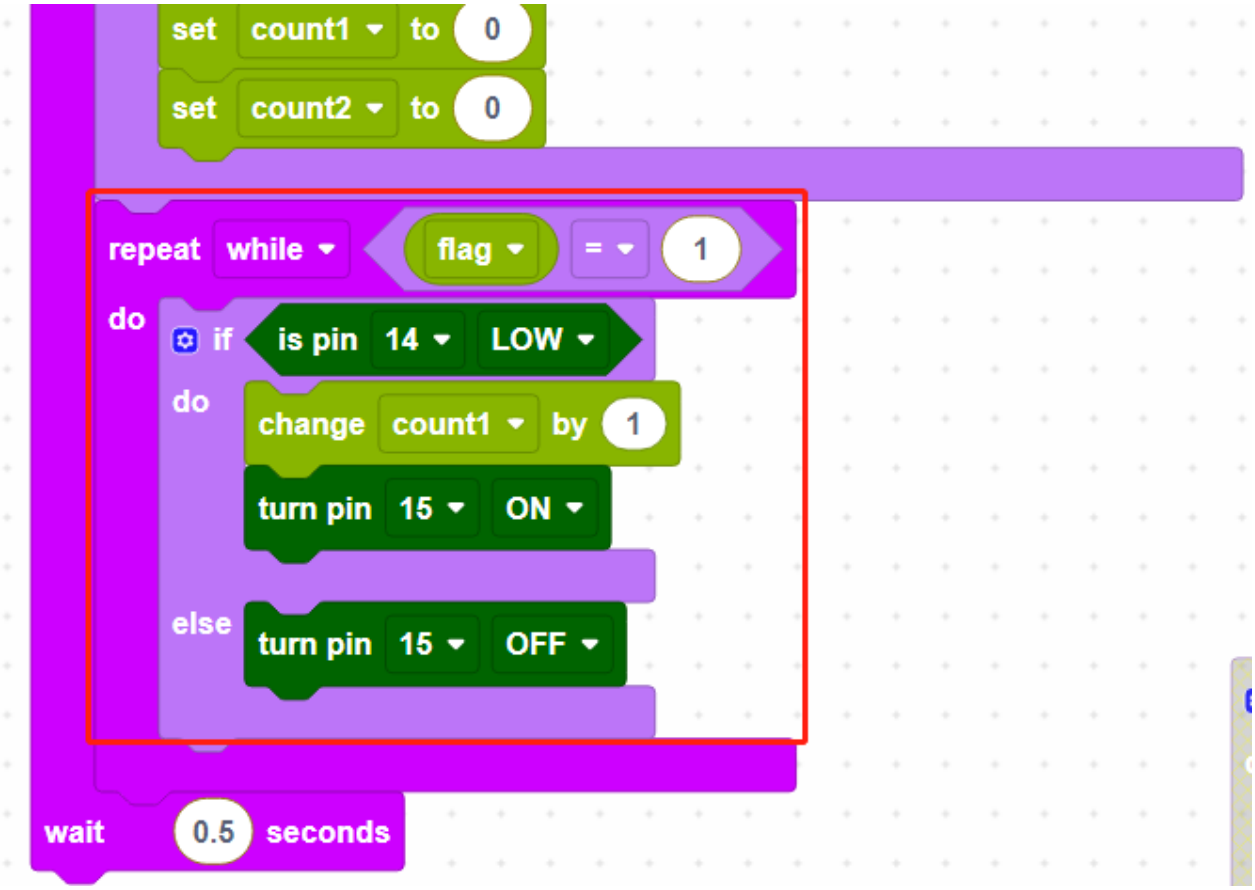

start
repeat forever
do
if is pin 10 HIGH
do
set prestate to 0
if prestate = 0 and is pin 10 LOW
do
turn pin 11 ON
set flag to 1
set count1 to 0
set count2 to 0
repeat while flag = 1
do
if is pin 14 LOW
do
change count1 by 1
turn pin 15 ON
else
turn pin 15 OFF
if is pin 12 LOW
do
change count2 by 1
turn pin 13 ON
else
turn pin 13 OFF
print " player1: "
print count1
print " player2: "
print count2
wait 0.1 seconds
if is pin 10 HIGH
do
set prestate to 1
if prestate = 1 and is pin 10 LOW
do
turn pin 11 OFF
set flag to 0
wait 0.5 seconds

- Pico W に接続した後、 **スタート** ボタンをクリックしてコードが実行されます。
- 審判ボタンを押すと、ブザーが連続して鳴り始め（ゲームの開始を示します）。
- この時点で、 **プレイヤー** ボタンを別々に素早く押し、対応する LED が点灯します。
- **審判** ボタンが再び押されると、ブザーが停止し、ゲームが終了します。
- この時点で CONSOLE をクリックして、誰がそれをより多く押したかを確認します。

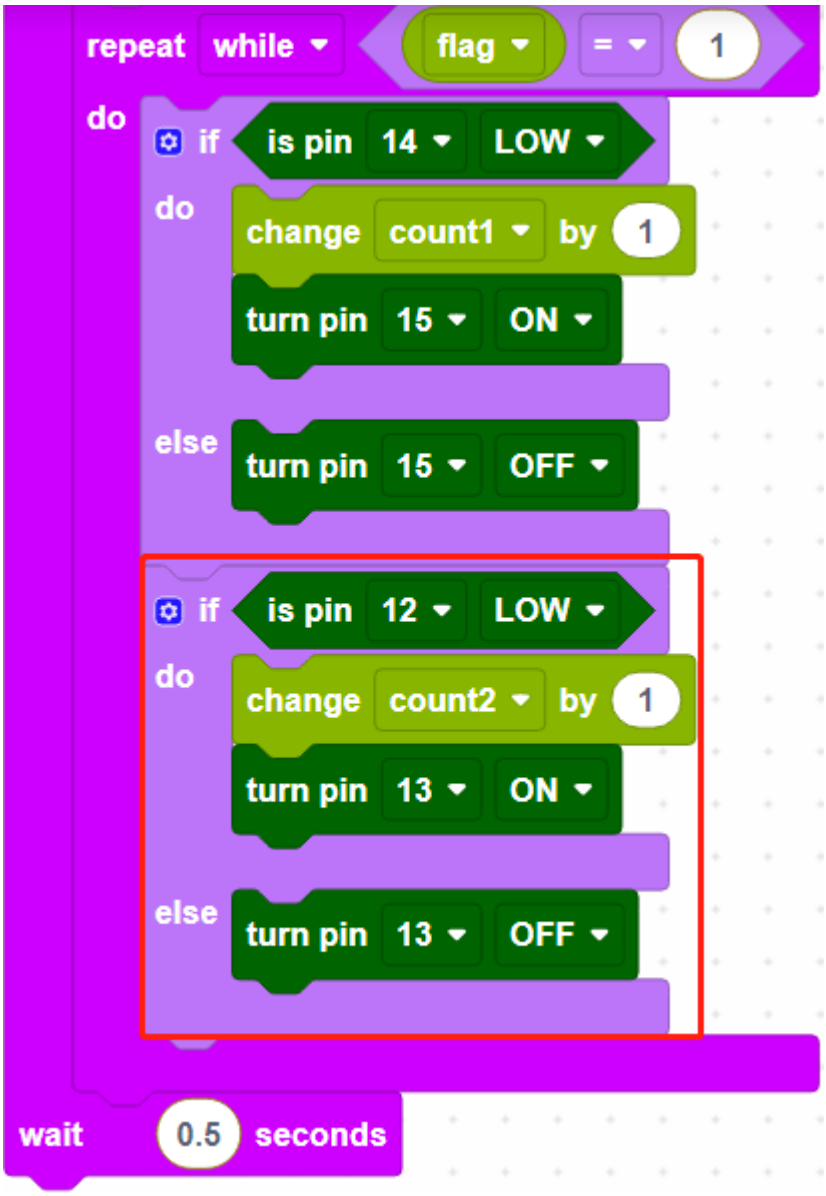
仕組み



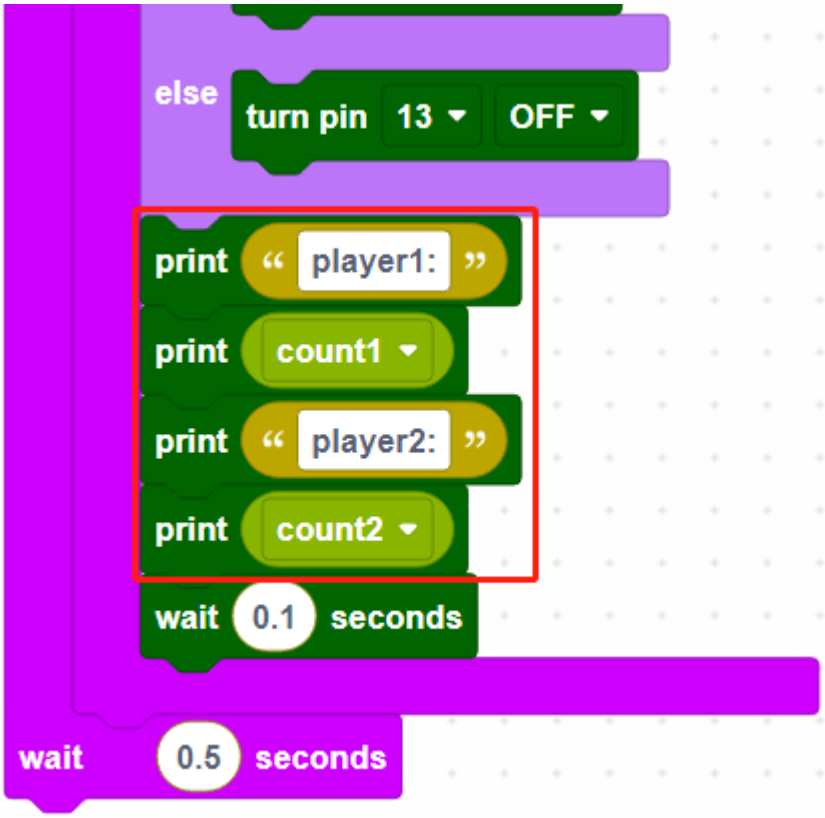
GP10 は審判ボタンで、審判ボタンが押されていない場合は高く、ゲームはまだ始まっていません。GP10 が低い（審判ボタンが押された場合）、ゲームがスタートします。GP11 を高く設定（ブザー）、変数を作成して初期値を設定します。



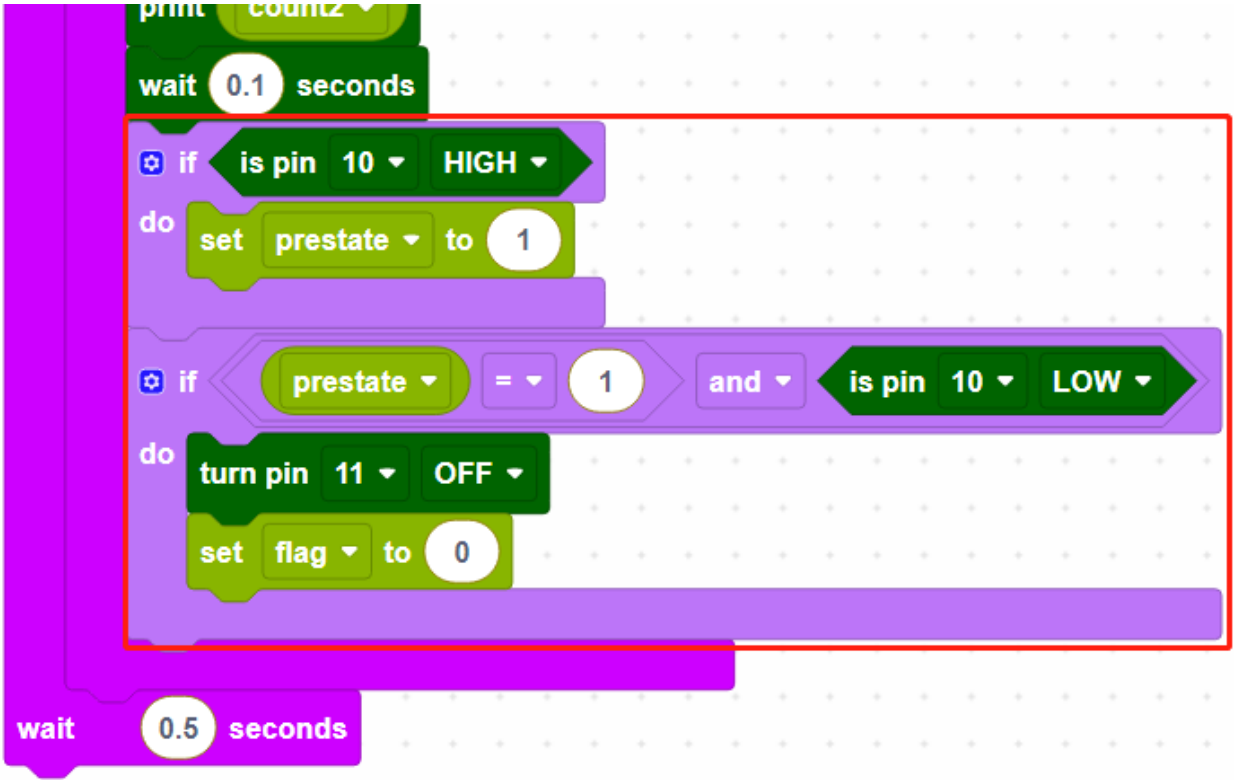
フラグが 1（ゲームスタート）の場合、次に GP14（プレイヤー 1）の値を読み取ります。プレイヤー 1 ボタンが押された場合、[count1] という変数に押す回数を保存し、GP15 のインジケーターを点灯させます。



同様にして、GP12（プレイヤー 2）の押し回数を読み取ります。



それぞれのプレイヤー 1 とプレイヤー 2 の押し回数を印刷します。

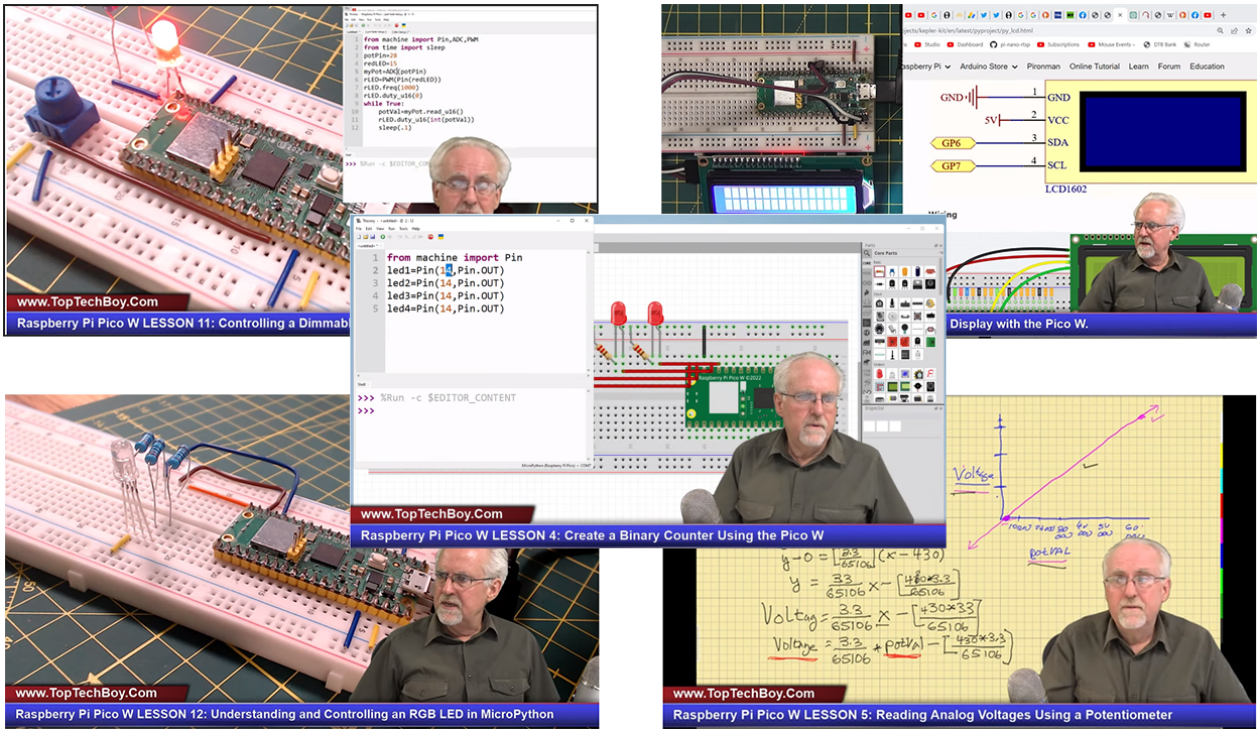


審判ボタンが再び押されると、ブザーが停止してゲームが終了します。

第 8 章

ビデオコース

オンラインドキュメントの内容が少し難解に感じる場合、心配無用です。ステップバイステップのビデオコースが、よりスムーズで楽しい学習経験を提供します。



ビデオコースには、以下のリンクでアクセスできます：。

このビデオシリーズは、Raspberry Pi Pico W に特化しており、MicroPython の基本から各コンポーネントの働き、さらには関連する電子回路の知識までを網羅しています。各ビデオは、視聴者が積極的に参加できるように、教育的かつエンターテイニングに設計されています。このビデオコースに没頭し、実践的なアクティビティに参加することで、Raspberry Pi Pico W の学習がより豊かで楽しいものに変わります。

第 9 章

FAQ

9.1 Arduino

1. Arduino IDE でコードのアップロードに失敗した場合は？

    - Arduino IDE が正しく Pico を認識しているか確認してください。ポートは COMXX（Raspberry Pi Pico）でなければなりません。詳細は *1.3 Raspberry Pi Pico W* **のセットアップ（重要）** を参照してください。
    - ボード（Raspberry Pi Pico）またはポート（COMXX（Raspberry Pi Pico））が正しく選択されているか確認してください。
    - コードに問題がなく、正しいボードとポートが選択されている場合でも、アップロードが成功しない場合は、 **アップロード** アイコンを再度クリックしてください。進行状況が「アップロード中...」と表示されたら、USB ケーブルを抜いて、 **BOOTSEL** ボタンを押しながら再度挿入します。その後、コードのアップロードが成功します。

9.2 MicroPython

1. コードを開いて実行する方法は？詳細なチュートリアルは **コードを直接開いて実行する** を参照してください。
2. Raspberry Pi Pico W にライブラリをアップロードする方法は？詳細なチュートリアルは *1.4 Pico* **にライブラリをアップロード** を参照してください。
3. Thonny IDE に MicroPython（Raspberry Pi Pico W）インタープリタオプションがない？

    - Pico W が USB ケーブルでコンピュータに接続されているか確認してください。
    - Pico W 用の MicroPython をインストールしたか確認してください（ *1.3 Raspberry Pi Pico* **に** *MicroPython* **をインストール** ）。

- Raspberry Pi Pico W インタープリタは、Thonny のバージョン 3.3.3 以降でのみ利用可能です。古いバージョンを使用している場合は、アップデートしてください（ *1.2 Thonny IDE* **をインストール** ）。
- この時点で Li-po チャージャーモジュールがブレッドボードに接続されている場合は、それを最初に抜いてから、Pico W をコンピュータに再接続してください。

4. Thonny IDE を経由して Pico W のコードを開いたり保存することができない？

   - Pico W が USB ケーブルでコンピュータに接続されているか確認してください。
   - インタープリタとして **MicroPython（Raspberry Pi Pico）** が選択されているか確認してください。

5. **Raspberry Pi Pico W は Thonny と Arduino で同時に使用できるか？** いいえ、いくつかの異なる操作が必要です。

   - 最初に Arduino で使用していて、今 Thonny IDE で使用したい場合は、その上で *1.3 Raspberry Pi Pico* **に** *MicroPython* **をインストール** が必要です。
   - 最初に Thonny で使用していて、今 Arduino IDE で使用したい場合は、 *1.3 Raspberry Pi Pico W* **のセットアップ（重要）** が必要です。

6. お使いのコンピュータが Win7 で Pico W が認識されない場合は？

   - USB CDC ドライバーを次の URL からダウンロードしてください： this link
   - `amtel_devices_cdc.inf` ファイルを `pico-serial` という名前のフォルダに解凍します。
   - `amtel_devices_cdc.inf` ファイルの名前を `pico-serial.inf` に変更します。
   - ノートパッドのような基本的なエディタで `pico-serial.inf` を開き/編集します。
   - 以下の見出しの下にある行を削除して置き換えます：

```
[DeviceList]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NTAMD64]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NTIA64]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NT]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[Strings]
Manufacturer = "ATMEL, Inc."
```

(次のページに続く)

(前のページからの続き)

```
PI_CDC_PICO = "Pi Pico Serial Port"
Serial.SvcDesc = "Pi Pico Serial Driver"
```

1. 閉じて保存し、名前が `pico-serial.inf` として保持されていることを確認してください。
2. PC のデバイスリストに移動し、CDC デバイスとして名前が付けられた pico をポートで見つけます。黄色い感嘆符がそれを示しています。
3. CDC デバイスを右クリックして、保存した場所から作成したファイルを選択してドライバーを更新またはインストールします。

9.3 Piper Make

1. Piper Make で Pico W をセットアップする方法は？

   詳細なチュートリアルは *1.1 Pico* **のセットアップ** を参照してください。

2. コードをダウンロードまたはインポートする方法は？

   詳細なチュートリアルは *1.3* **コードの保存やインポート方法は？** を参照してください。

3. Pico W に接続する方法は？

   詳細なチュートリアルは *2. Pico W* **に接続** を参照してください。

第 10 章

著作権について